

Lieferrex
Individualisierbare
Bestellplattform für Restaurants
mit Liefer-/Abholservice

Ersteller: Niklas Heim,
Michael Bogensberger,

Julian Meilinger,

Liuming Xia

Datum: 03.07.2022

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbst verfasst und keine anderen als die angeführten Behelfe verwendet habe. Alle Stellen, die wörtlich oder inhaltlich den angegebenen Quellen entnommen wurden, sind als solche kenntlich gemacht. Ich bin damit einverstanden, dass meine Arbeit öffentlich zugänglich gemacht wird.

Ort, Datum

Niklas Heim

Julian Meilinger

Michael Bogensberger

Liuming Xia

Vorwort

An dieser Stelle möchten wir uns bei allen bedanken, welche Unterstützung sowie Beratung am Projekt geleistet haben. Ein großer Dank geht an unseren Projektbetreuer DI Dr. Andreas Doblander, sowie an unseren Nebenbetreuer Mag. Dominik Moser, welche uns stets tatkräftig unterstützten und uns bei Fragen immer zur Seite standen. Ein weiterer großer Dank gilt unserem Auftraggeber Liu Hongmei, welcher uns einen guten Einblick in die alltäglichen Geschäftsfälle eines Restaurants gab. Des Weiteren möchten wir Diⁱⁿ Dr.ⁱⁿ Melani Osl für die ununterbrochene Unterstützung bei Fragen rund um die Dokumentation sowie den Ablauf der Diplomarbeit bedanken. Trotz dem Wegfall des Projektmitglieds Eraslan Burak wurde das Projekt in vollem Umfang realisiert.

Kurzfassung

Die Corona-Pandemie, sowie Faktoren wie beispielsweise die Digitalisierung, machen es Restaurants immer schwerer, ohne eine digitale Präsenz zu existieren. Immer mehr Menschen sehnen sich danach, bequem von der Couch aus Essen bestellen zu können. Das Problem ist, dass viele Besitzer von Restaurants oder Schnell-Imbissen meist keine Möglichkeit besitzen, Online-Bestellungen anzunehmen. Dies wird derzeit noch oft umständlich über Telefon abgewickelt. Durch unser entwickeltes System ist es einem Restaurant oder beispielsweise einem Schnell-Imbiss möglich, sich rasch eine Internetpräsenz mit Bestell- sowie Abholfunktionen aufzubauen. Sie können sich schnell und ohne Vorkenntnisse eine eigens konfigurierte Webseite nach ihren eigenen Designvorstellungen erstellen. Des Weiteren lassen sich Produkte beziehungsweise Gerichte zur Abholung oder zur Lieferung über die eigene Webseite anbieten. Zudem wird das Hosting der Webseite automatisch und ohne viel Aufwand erledigt.

Zur Umsetzung jenes Systems wird ein bewährter Technologie-Stack verwendet. Das Backend wird mittels Spring Boot umgesetzt. Ein beliebtes Java Framework für Enterprise Applikationen. Für das Frontend wird das auf dem Material Design Konzept basierende Framework Materialize verwendet. Als Verbindungsstück jener Technologien wird Thymeleaf verwendet.

Abstract

Inhaltsverzeichnis

1	Einleitung.....	1
2	Projektmanagement.....	1
2.1	Metainformationen	1
2.1.1	Projektteam.....	1
2.1.2	Projektbetreuer	2
2.1.3	Projektpartner	2
2.2	Vorerhebungen	2
2.2.1	Ist-Zustand.....	2
2.2.2	Soll-Zustand.....	3
2.2.3	Projektumfeldanalyse	3
2.2.4	Maßnahmen	5
2.2.5	Risikoanalyse	5
2.3	Pflichtenheft.....	7
2.3.1	Zielsetzung.....	8
2.3.2	Produkteinsatz und Umgebung	8
2.3.3	Funktionalitäten.....	8
2.4	Planung	9
2.4.1	Projektstrukturplan.....	9
2.4.2	Projektablaufplan.....	10
3	Vorstellung des Produkts.....	1
3.1	Hauptseite	2
3.2	Dashboard	5
4	Eingesetzte Technologien.....	10
4.1	Materialize	10
4.2	Materialize Stepper	10
4.3	js-cookie	10
4.4	Halfmoon	10
4.5	JQuery	10
4.6	Spring Boot.....	1
4.7	MySQL.....	1
4.8	Thymeleaf	2
4.9	Spring Security.....	2
4.10	Visual Paradigm	2
4.11	PayPal API.....	2

4.12	Google Maps API	2
4.13	GIT & GitHub	2
4.14	IntelliJ.....	2
5	Problemanaylse.....	2
5.1	Use-Case-Analyse	2
5.2	Domain-Class-Modelling	1
5.3	User-Interface-Design.....	1
6	Systementwurf.....	1
6.1	Architektur	1
6.1.1	C4-Diagramme	1
6.1.2	Design der Komponenten.....	3
6.1.3	Benutzerschnittstellen	3
6.1.4	Datenhaltungskonzept	3
6.1.5	Konzept für Ausnahmebehandlung	3
6.1.6	Sicherheitskonzept.....	3
6.1.7	Design der Testumgebung.....	5
6.1.8	Design der Ausführumgebung.....	5
6.2	Detailentwurf	5
6.3	Frontend	1
6.3.1	Einbindung der Frontend-Technologien	1
6.3.2	Struktureller Aufbau der Dateien	1
6.3.3	Verwendete Versionen	1
6.3.4	Interaktion zwischen den Seiten.....	2
6.3.5	Theming.....	3
6.4	Backend.....	2
6.4.1	Spring Boot	2
6.4.2	REST.....	3
1.1.1	Google Maps API.....	4
1.1.2	PayPal API.....	5
6.4.3	Baukastensystem	7
7	Implementierung	1
7.1	Frontend	1
7.1.1	Kundenseite.....	1
7.1.2	Kundenseite – Dark Mode	2
7.1.3	Dashboard	4
7.1.4	Dashboard – Dark Mode	4

7.1.5	Baukasten.....	6
7.1.6	JQuery REST request	1
7.1.7	Benachrichtigungen	1
7.2	Backend.....	2
1.1.3	Spring Security	2
1.1.4	Google Maps API.....	6
1.1.5	Registrierung	9
1.1.6	Benutzer Login	14
1.1.7	Kundenspezifische Daten anzeigen	15
1.1.8	Adresse ändern.....	16
1.1.9	Passwort ändern	17
1.1.10	Restaurantsuche	18
1.1.11	PayPal API.....	19
1.1.12	Checkoutseite	22
1.1.13	Bestellungen anzeigen	23
7.2.1	Overview Seite Angestellten / Mandant	25
7.2.2	Zubereitung	28
7.2.3	Zustellung	30
7.2.4	Gerichte.....	31
7.2.5	Bewertung	34
7.3	Benutzerrechte.....	36
7.3.1	Zahlungen.....	38
7.3.2	Öffnungszeit	41
7.3.3	Kategorien	43
7.4	Baukastensystem.....	1
7.4.1	Aufbau des Baukastens	1
7.4.2	Baukasten Frontend Funktionsweise	8
7.4.3	Baukasten Backend Funktionsweise	11
8	Deployment	1
8.1	SQL-Datenbank.....	3
9	GitHub & Scrum	4
9.1	GitGuardian.....	5
10	Codemenge	6
11	Tests.....	1
11.1	Systemtests	1
11.2	Akzeptanztests	5

12	Evaluation	5
12.1	Projektevaluation	5
12.1.1	Planungsabweichungen	5
12.1.2	Aufwand/Kosten	5
12.1.3	Zusammenarbeit	5
12.1.4	Zuordnung der Subthemen	6
12.2	Produktevaluation	6
12.3	Resümee	6
12.3.1	Michael Bogensberger	6
12.3.2	Niklas Heim	7
12.3.3	Liuming Xia	7
12.3.4	Julian Meilinger	7
12.4	Zeitaufstellung	7
13	Benutzerhandbuch	1
13.1	Verwendung des Baukastens	1
13.1.1	Allgemeine Einstellungen	1
13.1.2	Bearbeiten des Kopfbereiches	2
13.1.3	Modul hinzufügen	2
13.1.4	Löschen eines Moduls	4
13.1.5	Erstellen von Zusatz-Seiten	4
13.1.6	Löschen von Zusatz-Seiten	5
14	Abnahme	1
15	Einführung	1
16	Betriebsphase	2
17	Wartungsphase	2
18	Zusammenfassung	2
A.	Anhang	3

1 Einleitung

Im Rahmen der Suche nach einem geeigneten Thema für die Diplomarbeit stießen wir auf folgende Fragen. Wieso haben so viele Restaurants einen mangelhaften Webauftritt und wieso kann ich heutzutage immer noch nur über das Telefon mein Essen bestellen? Da die Eltern von Projektmitglied Liuming Xia ein Restaurant besitzen, fragten wir nach, ob Sie eine Lösung für jenes Problem bräuchten. Mit unserem Dienst könnten Sie sich ganz einfach einen Webauftritt erstellen und direkt über diesen ihre Gerichte zur Abholung oder zur Lieferung anbieten. Des Weiteren würde man keinen kostenintensiven Dienst wie beispielsweise Lieferando benötigen. So erklärte sich schlussendlich das Liu Hongmei Royal Asia Restaurant dazu ein Pilotbetrieb des Projektes zu werden.

Das Projektteam bestand ursprünglich aus fünf Mitgliedern. Eraslan Burak gibt das Projekt zum Haupttermin nicht ab. Es ist jedoch zu erwähnen, dass das Projekt in vollem Umfang realisiert wurde. Der Hosting- beziehungsweise Deployment-Teil wurde von Michael Bogensberger übernommen. Für das Hosting wurde der Cloudanbieter Heroku verwendet. Des Weiteren gab es im Projekt keinerlei Beitrag von Eraslan Burak. Es ist zudem unklar, ob Eraslan Burak das Projekt zu einem späteren Zeitpunkt abgeben möchte. Da das Projekt bereits vollständig realisiert wurde, würde sich vermutlich nicht genügend Stoff zu einer Abgabe von Eraslan Burak finden.

2 Projektmanagement

In folgendem Abschnitt sind Informationen zum Initiiieren, Planen, Steuern sowie Kontrollieren des Projektes zu finden.

2.1 Metainformationen

In den Metainformationen sind die Projektmitglieder und deren Aufgabenbereiche, die Projektbetreuer sowie der Projektpartner beschrieben.

2.1.1 Projektteam

Name: Niklas Heim

Funktion: Projektleiter

Zuständigkeit: Backend des Baukastensystems



Name: Michael Bogensberger

Funktion: Projektmitglied

Zuständigkeit: Frontend, Backend, Hosting & Deployment



Name: Julian Meilinger

Funktion: Projektmitglied

Zuständigkeit: Backend der Hauptseite, Spring Security, APIs



Name: Liuming Xia
Funktion: Projektmitglied
Zuständigkeit: Backend des Dashboards



2.1.2 Projektbetreuer

Name: DI Dr. Andreas Doblander
Funktion: Hauptbetreuer



Name: Mag. Dominik Moser
Funktion: Nebenbetreuer



2.1.3 Projektpartner

Name: Liu Hongmei Royal Asia Restaurant
Ansprechpartner: Liu Hongmei
Adresse: Kitzsteinhornstraße 27, 5700 Zell am See

2.2 Vorerhebungen

In folgendem Abschnitt ist eine vorausgehende Untersuchung des geplanten Projektes zu sehen. Sie umfasst die Erfassung des Ist- sowie Soll-Zustandes, die Projektumfeldanalyse, Maßnahmen sowie die Risikoanalyse.

2.2.1 Ist-Zustand

Seit der Corona Zeit gab es schon mehrere Lockdowns, Lokale wurden zugesperrt und konnten nicht vor Ort ihre Kunden bedienen. Die gesetzliche Lage verschärft sich für die Gastronomie in dieser Zeit, die Regelung besagt, dass die Lieferung von Waren erlaubt ist und auch empfohlen wird. Viele Besitzer von Restaurants oder einem Schnell-Imbissen besitzen meist keine Möglichkeit, Online-Bestellungen anzunehmen. Dies wird derzeit noch oft umständlich über Telefon abgewickelt. Durchs Telefonieren können häufig Missverständnisse entstehen und dadurch können die Bestellungen nicht eindeutig identifiziert oder wie gewünscht ausgeführt werden. Mithilfe des Internets und eines entwickelten Systems können Bestellung leicht und detailreich eingereicht werden

2.2.2 Soll-Zustand

In der Zeit, in dem das Internet keine Neuheit ist und die meisten Dinge über das World Wide Web abgewickelt wird, werden auch die meisten Lieferdienste über das Internet Angeboten. Viele Klein-Restaurants können mithilfe des neu entwickelten Systems schnell und ohne Vorkenntnisse eine eigens konfigurierte Webseite nach ihren eigenen Designvorstellungen erstellen. Das Hosting der Webseite wird automatisch und ohne viel Aufwand erledigt.

2.2.3 Projektumfeldanalyse

Im Rahmen des Projektes hat das Projektteam mit vielen verschiedenen Personen zu tun. Diese können und werden das Projekt positiv oder gar negativ beeinflussen. Es wird ermittelt, wer genau diese Stakeholder sind, welchen Einfluss sie haben, wie nahe sie zum Projekt stehen und welche Maßnahmen getroffen werden, um diese abzuholen. In *Tabelle 1: Stakeholder* werden alle Stakeholder mit ihrer Einstellung, ihrem Einfluss und ihrer Nähe (im Bereich von 1 bis 10) zum Projekt aufgelistet.

Stakeholder	Einstellung	Einfluss	Nähe
Projektmitglieder	Positiv	8	9
Auftraggeber	Positiv	3	6
Zukünftige Nutzer (z. B. Koch, Restaurantbesitzer)	Positiv	2	3
Betreuer	Neutral	7	8
Restaurantkunde	Positiv	2	2
Mitbewerber	Negativ	1	1

Tabelle 1: Stakeholder

Im Folgenden werden die einzelnen Stakeholder beschrieben:

Ein Stakeholder ist das Projektteam, es ist hauptsätzlich für das Projekt verantwortlich und trifft viele Entscheidungen, wie die Planung, Umsetzung und Einsatz von verschiedenen Technologien. Es ist wichtig, eine gute Kommunikation und Planung im Team zu halten, damit jedes Mitglied immer auf dem gleichen Stand ist und so gemeinsam, zielstrebig am Projekt arbeiten können.

Einen hohen Einfluss auf das Projekt hat der Auftraggeber. Er bestimmt die grundlegenden Rahmenbedingungen und Anforderungen des Projektes und wird anschließend durch das Ergebnis des Projektes vermarktet.

Der Projektbetreuer ist ein weiterer Stakeholder. Er betreut das Projekt und unterstützt das Team bei Entscheidungen und kontrolliert den Fortschritt. Der Betreuer wird regelmäßig durch Besprechungen über das Projekt informiert.

Zukünftige Nutzer werden leichten Einfluss auf das Projekt haben. Sie sind die Restaurantbesitzer und Köche, die das entwickelte System zur Vermarktung oder zum Planen ihrer Arbeitsabläufe verwenden werden.

Auch Restaurantkunden werden als Stakeholder definiert. Sie können das Ergebnis des Projektes verwenden, um bei den verschiedenen Restaurants, die das System verwenden, bestellen.

Das Projekt hat auch einen negativ gestimmten Stakeholder. Mitbewerber beziehungsweise Konkurrenten, die ähnliche Systeme anbieten und durch unser Projekt möglicherweise Kunden verlieren.

Hier in *Abbildung 1: Stakeholder grafisch* werden die Stakeholder grafisch dargestellt. Die dazugehörige Legende befindet sich unterhalb, in *Tabelle 2: Legende Stakeholder grafisch*.

Ihre Nähe zum Projekt wird als Abstand zum mittleren Kreis dargestellt. Deren Einstellung wird als Pfeil oder Waage interpretiert. Bei einer positiven Einstellung zeigt der Pfeil nach oben, bei einer negativen nach unten und bei einer Waage neutral. Der Einfluss auf das Projekt wird durch Farben veranschaulicht. Je höher der Einfluss ist, desto mehr ist der Stakeholder rot gefärbt. Es wird auch über Sprechblasen und Linien angezeigt, welche Stakeholder miteinander kommunizieren.

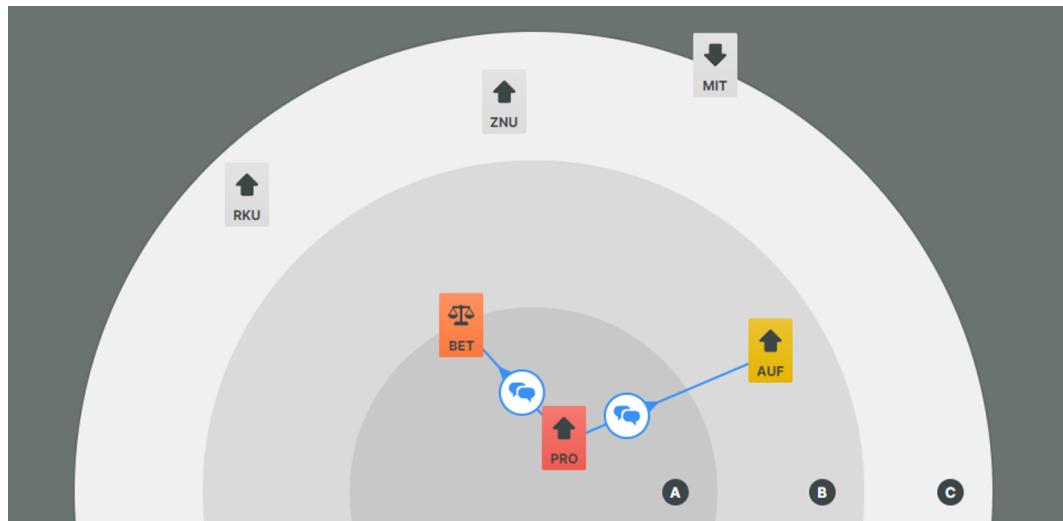


Abbildung 1: Stakeholder grafisch

Folgende Tabelle dient als Legende für die Stakeholderabbildung.

Symbol	Bedeutung
PRO	Projektmitglied
BET	Betreuer
AUF	Auftraggeber
ZNU	Zukünftige Nutzer
RKU	Restaurantkunde
MIT	Mitbewerber
Pfeil nach oben	Positive Einstellung
Pfeil nach unten	Negative Einstellung
Waage	Neutrale Einstellung
A	Hohe Nähe
B	Mittlere Nähe
C	Geringe Nähe
Nachrichtensymbol	Kommunizieren miteinander

Tabelle 2: Legende Stakeholder grafisch

2.2.4 Maßnahmen

Hier in Tabelle 3: Stakeholder Maßnahmen werden die Interessen der verschiedenen Stakeholder und einige Maßnahmen angeführt, um die verschiedenen Stakeholder besser zu stimmen und sie abzuholen.

Stakeholder	Beschreibung der Interessen	Maßnahmen
Projektmitglieder	Entwicklung des Endproduktes mit gewünschter Qualität	Gutes Klima im Projektteam sowie ein guter Informationsaustausch
Auftraggeber	Endprodukt mit gewünschten Funktionen und gewünschter Qualität	Regelmäßiger Informationsaustausch
Zukünftige Nutzer (z. B. Koch, Restaurantbesitzer)	Vereinfachte Prozesse (z. B. Leichtere Vertragsabwicklung, Leichtere Aufnahme und Verarbeitung von Bestellprozessen)	Möglichst gute Qualität des Endproduktes, sowie einfache Bedienbarkeit.
Betreuer	Bestmögliches Projekt sowie einen nicht zu großen Unterstützungsauflauf	Guter Informationsaustausch der Projektmitglieder sowie einen gelegentlichen Austausch mit dem Projektbetreuer
Restaurantkunde	Möchte schneller Restaurants in der Nähe finden sowie leicht und flexibel Essen bestellen.	Das Endprodukt möglichst intuitiv bedienbar gestalten.
Mitbewerber	Ist negativ gestimmt. Möchte nicht, dass das Projekt erfolgreich ist.	Geheimhaltung des Projektes

Tabelle 3: Stakeholder Maßnahmen

2.2.5 Risikoanalyse

Im Folgenden werden alle Risiken ermittelt, die beim Projekt auftreten können. Jedes Risiko erhält einen Titel, Status und eine Kategorie. Des Weiteren wird ermittelt, wie wahrscheinlich das Risiko eintreten kann und welche Folgen es mit sich bringt. Anschließend werden Maßnahmen definiert, um die Risiken zu überwachen oder eliminieren. Anhand eines Risikoportfolios werden ermittelten Risiken tabellarisch dargestellt. In den Tabellen (*Tabelle 4: Risikoportfolio Teil 1* und *Tabelle 5 Risikoportfolio Teil 2*) werden die einzelnen Risiken aufgelistet. Die Risiken in den jeweiligen Tabellen sind durch die Nummerierung (Nr.) erkennbar.

Die Risiken werden von 1 bis 11 durchnummieriert und erhalten jeweils einen Status der entweder mit *eliminiert* oder *überwacht* benannt wird. Zudem gibt es jeweils die Kategorien „menschlich/kulturell“, „technische/produktbezogen“, „wirtschaftlich“ und „politisch“. Dazu ist ein kurzer Titel zum Risiko, die Folgen des jeweiligen Risikos (wenn nichts dagegen unternommen wird) und dazu passende Gegensteuerungsmaßnahmen zu finden. Zudem finden sich Eintrittswahrscheinlichkeit, Auswirkungsgrad und Risikopotential wieder. Das Risikopotential ist die Multiplikation von Eintrittswahrscheinlichkeit und Auswirkungsgrad. Sie beschreibt wie schwerwiegend ein Risiko als Ganzes ist.

Nr.	Risikotitel	Status	Kategorie	Folgen des Risikos
1	Datenverlust	Eliminiert	menschlich/ kulturell	Projektfortschritt wird zurückgeworfen
2	Endprodukt entspricht nicht den Erwartungen des Auftraggebers	Überwacht	menschlich/ kulturell	Endprodukt kann vom Auftraggeber nicht verwendet werden
3	Anwendung spärlich zu bedienen	Überwacht	technisch/ produktbezogen	Anwender sind unzufrieden
4	Projektmitglied fällt aus	Überwacht	menschlich/ kulturell	Geplante Projektumsetzung im gleichen Ausmaß nicht mehr möglich
5	Projektmitglied führt Arbeitsaufträge nur mangelhaft aus	Überwacht	menschlich/ kulturell	Geplante Projektumsetzung im gleichen Ausmaß nicht mehr möglich
6	Fehleinschätzung des Aufwands	Überwacht	menschlich/ kulturell	Projektziele können nicht erreicht werden
7	Ausfall des Hosting Anbieters	Eliminiert	technisch/ produktbezogen	Anwendung kann nicht verwendet werden
8	Arbeitspakete der Projektanten ergänzen sich nicht	Überwacht	menschlich/ kulturell	Projektergebnis entspricht nicht den Erwartungen
9	Schnittstelle nicht kompatibel	Überwacht	technisch/ produktbezogen	Geplante Funktionalität nicht umsetzbar
10	Ähnliche Anwendung erscheint während der Projektentwicklung auf dem Markt	Überwacht	wirtschaftlich	Produkt des Mitbewerbers wird vorgezogen
11	Rechtliche Anforderungen werden nicht erfüllt	Überwacht	politisch	Anwendung darf nicht veröffentlicht werden

Tabelle 4: Risikoportfolio Teil 1

Nr.	Risikotitel	Eintritts- wahrschein- lichkeit	Auswirkung	Risiko- potential	Gegensteuerungs- maßnahme
1	Datenverlust	2	3	6	Projekt wird in GitHub gespeichert
2	Endprodukt entspricht nicht den Erwartungen des Auftraggebers	1	3	3	Regelmäßiger Informationsaustausch
3	Anwendung spärlich zu bedienen	1	2	2	Auf Designstandards achten
4	Projektmitglied fällt aus	1	3	3	Gutes Teambuilding
5	Projektmitglied führt Arbeitsaufträge nur mangelhaft aus	2	2	4	Regelmäßige Statusberichte
6	Fehleinschätzung des Aufwands	2	2	4	Arbeitsaufwand kontrollieren
7	Ausfall des Hosting Anbieters	1	3	3	Vertrauenswürdigen Hosting Anbieter auswählen
8	Arbeitspakete der Projektanten ergänzen sich nicht	1	3	3	Regelmäßiger Informationsaustausch

9	Schnittstelle nicht kompatibel	1	2	2	Im Vorfeld darüber informieren
10	Ähnliche Anwendung erscheint während der Projektentwicklung auf dem Markt	1	2	2	Marktentwicklung beobachten
11	Rechtliche Anforderungen werden nicht erfüllt	3	2	6	Über den BSI informieren

Tabelle 5 Risikoportfolio Teil 2

Folgende Tabelle dient als Legende für das Risikoportfolio.

Symbol	Bedeutung
Eintrittswahrscheinlichkeit	Gemessen von 1 bis 4, je höher desto öfter tritt das Risiko ein
Auswirkung	Gemessen von 1 bis 4, je höher, desto mehr Geld und Aufwand muss erbracht werden, um das Problem zu lösen
Risikopotential	Multiplikation der ermittelten Werte Eintrittswahrscheinlichkeit und Auswirkung, Werte 1 bis 16, entscheidet, wie schnell und sorgfältig mit dem Risiko umgegangen wird

Tabelle 6: Legende Risikoportfolio grafisch

In Abbildung 2: Risikomatrix werden die ermittelten Risiken in einer Matrix dargestellt. Die Risiken werden von der oberliegenden Tabelle 4: Risikoportfolio in die folgende Grafik überführt. Hier werden sie von R1 bis R11 durchnummert. Durch diese wird veranschaulicht, wo welche Risiken liegen und auf welche besonders geachtet werden sollten. Auf den beiden Achsen sind die Eintrittswahrscheinlichkeit und der Grad der Auswirkung aufgetragen. Risiken, die sich rechts oben in der Matrix befinden, sind besonders schwerwiegend. Dies wird auch durch einen immer roter werdenden Hintergrund hervorgehoben.

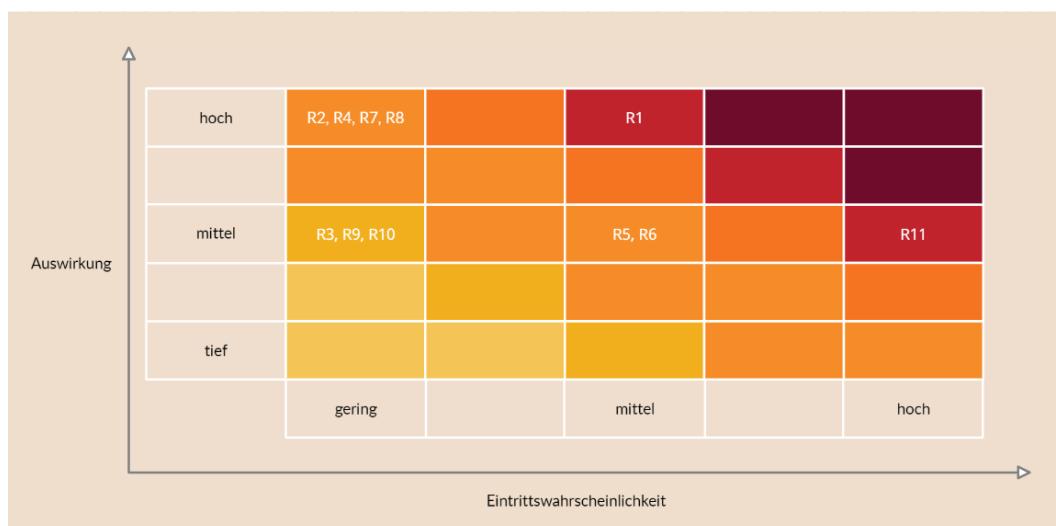


Abbildung 2: Risikomatrix

2.3 Pflichtenheft

Die folgenden Absätze konkretisieren die Pflichten des Projektteams. Dabei werden die Funktionalitäten, das Einsatzgebiet und die Ziele genau definiert.

2.3.1 Zielsetzung

Das Projektteam setzt sich die Entwicklung einer individualisierbaren Bestellplattform für die Gastronomie als Ziel. Damit sollen Restaurants die Möglichkeit haben, schnell und einfach einen Liefer-/Abholservice einzurichten. Das größte Hindernis ist dabei die Programmierung des individuell gestaltbaren Baukastensystems. Die Bestellplattform wird als Responsive Web-App veröffentlicht und folgt dem Material-Design als Formgebung. Als Schnittstelle zwischen dem Client und Server wird eine REST-API implementiert. Wunschziel ist es, dass unser Projektpartner und weitere mögliche Kandidaten die Bestellplattform verwenden und im täglichen Geschäft gebrauchen können. Die Absolvierung des Projekts hat für jeden Projektbeteiligten eine positive Auswirkung. Die Gastronomie bekommt eine weitere Option, ihre Lebensmittel Online zu vermarkten und können personalisierte Bestellplattformen erstellen, die ihrer Unternehmensphilosophie entsprechen. Die Gesellschaft hat eine weitere Möglichkeit Essen im Internet zu kaufen. Die Projektanten haben die Möglichkeit bei einem sehr erfolgreichen Abschluss des Projekts einen kleinen Nebenverdienst zu generieren und haben sich Wissen zu neuartigen Technologien in der Webentwicklung angeeignet. Das Erreichen der Mindestanforderungen ist realistisch. Das Projektteam hat die Kompetenz innerhalb des geforderten Zeitraums die Aufgaben zu bewältigen und das Projekt vorzustellen. Jeder Projektant ist motiviert, seine Aufgabenstellungen zu absolvieren und so gut wie möglich zu bearbeiten, um ein reibungsloses Zusammenspiel der Teilaufgaben zu ermöglichen. Die Diplomarbeit wird im Juni 2022 abgegeben. Ein erster Prototyp soll, bis Jänner 2022 fertiggestellt werden.

2.3.2 Produkteinsatz und Umgebung

Unsere Software soll die Arbeit in Gastronomiebetrieben vereinfachen. Deshalb versucht das Projektteam gezielt die Mensch-Computer-Interaktion so intuitiv wie möglich zu machen. Somit soll es den Mitarbeitern der Restaurants möglich sein, Onlinebestellungen schnell und einfach anzunehmen. Einen weiteren Einsatz findet unser Projekt bei allen hungrigen Personen, die sich bequem Essen bestellen möchten. Die Onlineplattform stellt dabei die besten Voraussetzungen, um eine Reibungslose Kommunikation zwischen Restaurant und Kunde zu ermöglichen.

2.3.3 Funktionalitäten

Es ist besonders wichtig, die Funktionalitäten des Projektes klar zu definieren. Diese müssen eingeteilt werden in Muss- und Kann-Funktionalitäten. Muss-Funktionalitäten müssen im Rahmen des Projektes erfüllt werden. Kann-Kriterien sind zusätzliche Funktionen, die erfüllt werden können, aber nicht müssen. Des Weiteren werden sie für eine klare Übersicht und Struktur in funktionale und nicht funktionale Anforderungen gegliedert. Funktionale Anforderungen beschreiben gewünschte Funktionalitäten. Sie beschreiben, was das entwickelte System kann. Nicht funktionale Anforderungen erhöhen die Qualität des Projektes. Die Anforderungen werden nach dem FURPS¹ System geplant. Sie beschreiben die Funktionalität, Benutzbarkeit, Zuverlässigkeit, Zuverlässigkeit und Wartbarkeit des Projektes.

Im folgenden Teil werden die Muss-Anforderungen aufgelistet. Sie beschreiben Anforderungen, die im Rahmen des Projektes erfüllt werden müssen, da sie wichtige Kernelemente dieses bilden. Sie werden in funktionale und nicht funktionale Anforderungen aufgeteilt.

- Funktional
 - Anmeldung für Benutzer der Webseite.
 - Dashboard für Restaurants für Verwaltung.
 - Erstellen von Bestellungen.

¹ Functionality, Usability, Reliability Perfomance, Supportability

- Anlegen neuer Gerichte.
- Gestalten von Webseiten mit dem Baukastensystem.
- Responsive Webseite auf allen Endgeräten.
- Nicht funktional
 - Das Design der Webseite ist ansprechend.
 - Die Webseite ist gut erweiterbar.
 - Das System ist gut dokumentiert.
 - Baukastensystem leicht bedienbar.

Im nächsten Teil werden die Kann-Funktionalitäten des Projektes aufgelistet. Sie können zusätzlich zu den Muss-Funktionalitäten erfüllt werden, müssen aber nicht. Auch die Kann-Funktionalitäten werden in funktionale und nicht funktionale Anforderungen aufgeteilt.

- Funktional
 - Weitere Anpassungsmöglichkeiten mit dem Baukastensystem.
 - Restaurant werden über Sub-Domains aufgerufen.
 - Online-Bezahlung über die Webseite.
- Nicht funktional
 - Die Daten sollen gut mit Backups gesichert werden.
 - Fehler der Webseite sollen leicht behoben werden können.
 - Die Webseite muss barrierefrei gestaltet sein.

2.4 Planung

Die folgenden Absätze schildern detailliert die Planungsansätze für unser Projekt. Der Projektstrukturplan beinhaltet die wichtigsten Arbeitspakete und kategorisiert sie in dessen Phasen ein. Der Projektablaufplan übernimmt diese Arbeitspakete und ordnet sie zeitlich ein.

2.4.1 Projektstrukturplan

In der folgenden Grafik (*Abbildung 3: Projektstrukturplan*) finden Sie den zum Projekt passenden Projektstrukturplan. Wir haben uns für einen Phasen-orientierten Projektstrukturplan entschieden, da sich dadurch die Ablaufplanung leichter gestalten lässt.



Abbildung 3: Projektstrukturplan

2.4.2 Projektlaufplan

Aus dem Projektstrukturplan wurde im nächsten Schritt ein Projektlaufplan (*Abbildung 4: Projektlaufplan*) erstellt. Dieser listet alle Teilaufgaben mit Datum, benötigter Arbeitszeit und verwandelter Ressourcen auf. Auch zu sehen sind Meilensteine, die zeigen, wann wichtige Kernelemente des Projekts abgeschlossen werden.

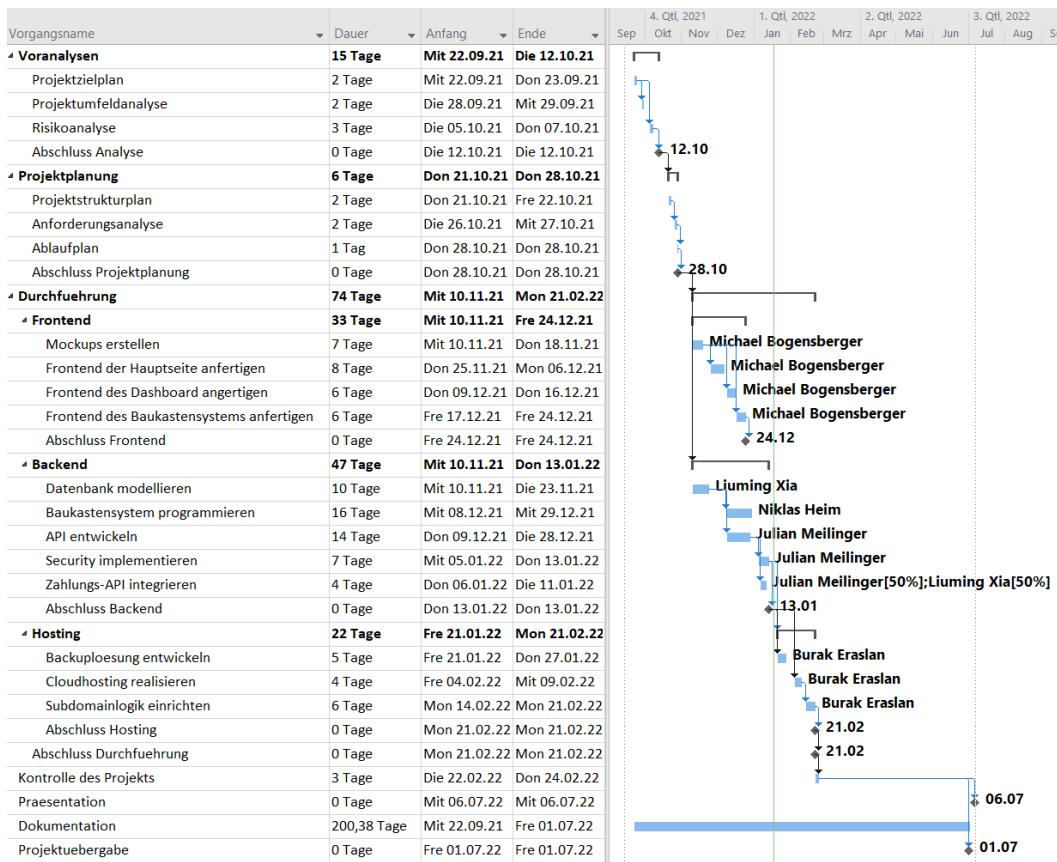


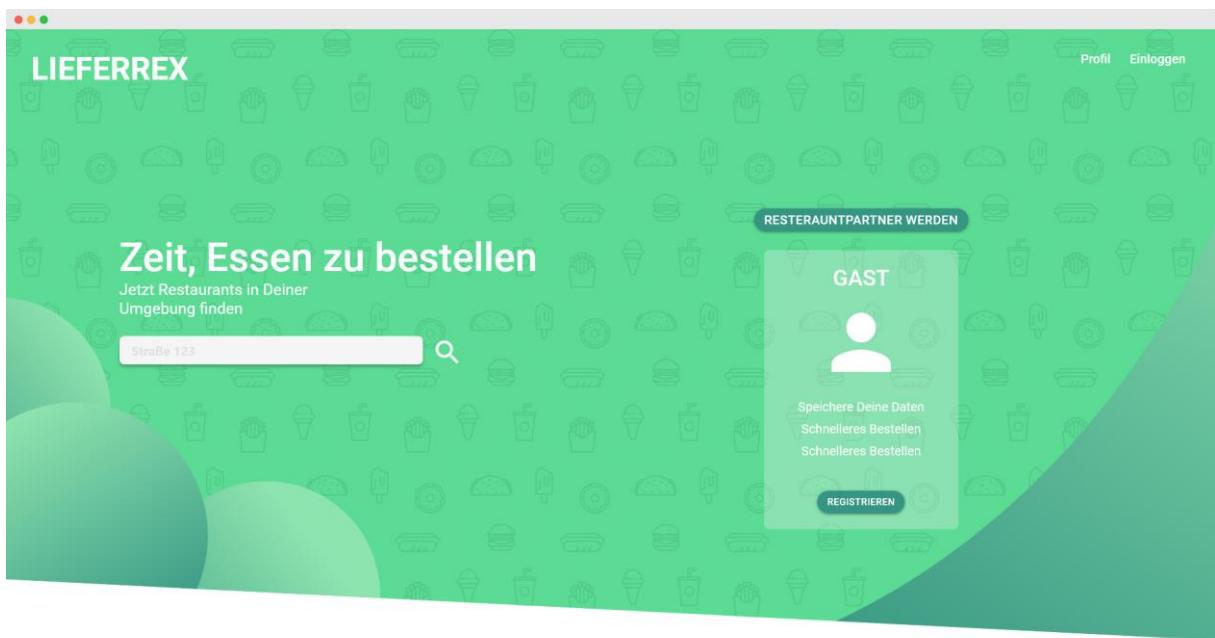
Abbildung 4: Projektlaufplan

3 Vorstellung des Produkts

Die Frontendstruktur unseres Projektes ist grundsätzlich in drei Teile (Kundenseite, Dashboard, Baukasten) unterteilt. Auf der Kundenseite kann sich ein Kunde registrieren und dort von Restaurants Essen bestellen und sich dieses liefern lassen. Auf jener Startseite kann man nun nach Restaurants suchen. Tut man dies, gelangt man auf eine Übersicht mit den gefundenen Restaurants. Von dort aus kann man nun einen Bestellprozess abschließen und über die Startseite seine Bestellungen überwachen.

Das Dashboard ist für das Restaurant gedacht. Dort kann man Bestellungen überwachen, Statistiken einsehen sowie Einstellungen ändern. Betritt man das Dashboard, so gelangt man zu einer Übersicht über das Restaurant. Dort sind einfache Statistiken, die letzten Bestellungen sowie eine Hilfe zu sehen. Von dort aus kann man nun auf die jeweiligen Seiten navigieren.

Auf der Baukastenseite kann das Restaurant sich nun eine eigene Webseite zusammenbauen. Auf jene Seite gelangt man über das Dashboard.



So einfach funktioniert!

Abbildung 5: Mockup der Hompage

Im Baukasten angelangt können vom Mandanten allgemeine Einstellungen beispielsweise zur Akzentfarbe getroffen werden. Als nächstes muss ein Layout aus vielen vordefinierten gewählt werden. Das Layout entscheidet, in welcher Form und Anordnung die kommenden Module dargestellt werden. Wurde ein Layout gewählt, wird dieses angezeigt. Über die einzelnen Kacheln mit Plussymbolen werden nun Module hinzugefügt. Es öffnet sich ein Pop-Up das alle verfügbaren Module auflistet. Verfügbare Module beinhalten beispielsweise: Text, Überschrift, Bild, Speisekarte, Karte, Öffnungszeiten und vieles mehr. Wurde ein Modul gewählt müssen entsprechende Daten eingegeben werden. Es können auch weitere Seiten wie Kontakt, Bildergalerie und mehr in der Navigationsleiste hinzugefügt werden. Des Weiteren kann ein Mandant zusätzliche Seiten wie „Über uns“, Bildergalerie und weiteres haben.

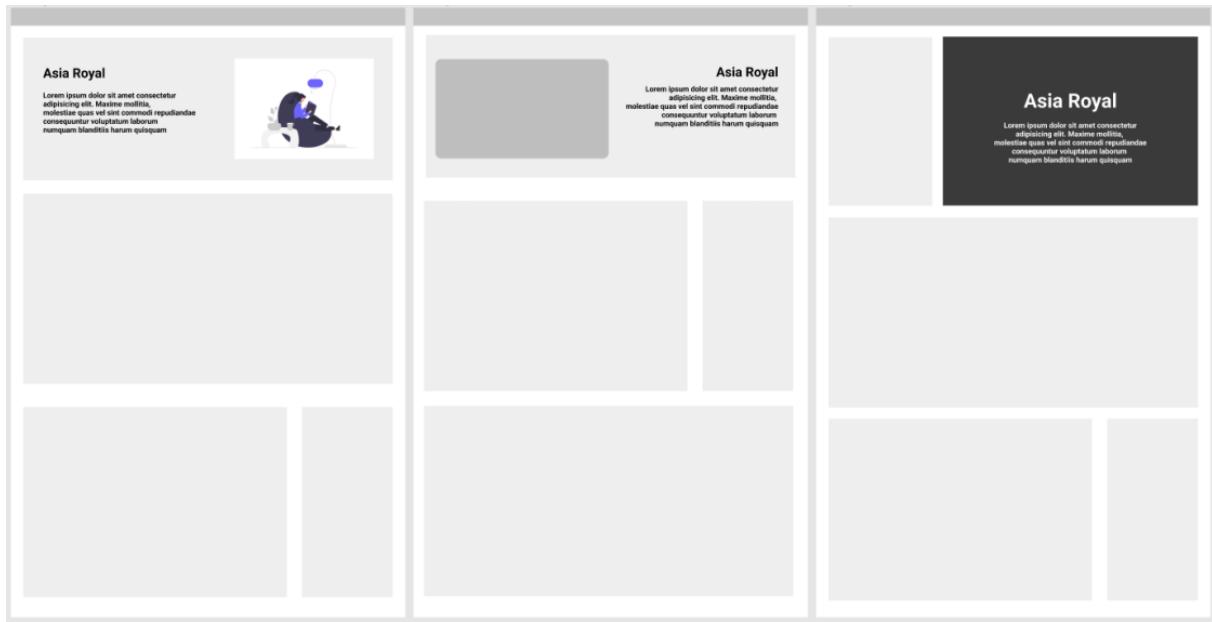


Abbildung 6 Mockup vom Baukastensystem

3.1 Hauptseite

Ist man eingeloggt, so findet man auf der Index-Seite oben rechts ein Link mit dem Namen „Profil“. Klickt man darauf, so öffnet sich ein Modal, welches auf weitere Seiten verlinkt. Von hier aus kann man seine Bestellungen einsehen, die Adresse ändern, das Passwort ändern oder sich ausloggen.

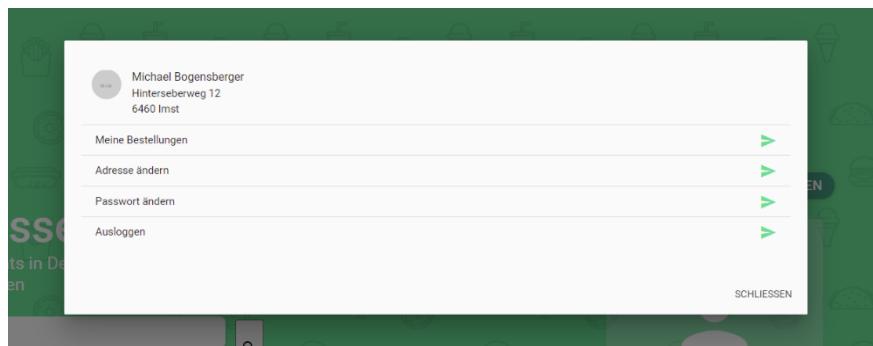


Abbildung 7: Kundenseite - Index Page - Modal

Klickt man auf „meine Bestellungen“, so gelangt man auf jene Seite, auf der alle bisherigen Bestellungen einzusehen sind. Hier wird eine Bestellung in jeweils einem ausklappbaren Tab dargestellt. Zudem hat der Kunde hier die Gelegenheit für seine Bestellung eine Bewertung abzugeben. Das bedeutet, dass der Kunde in unserem Projekt die Möglichkeit hat pro Bestellung eine Bewertung abzugeben. Bestellt er also mehrmals beim gleichen Restaurant, so kann er auch mehrmals eine Bewertung abgeben.

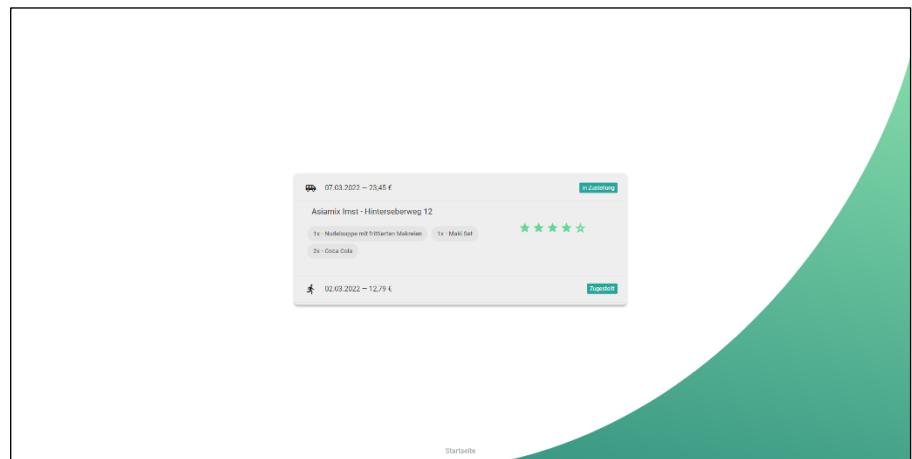


Abbildung 8: Kundenseite - Bestellungen Page

Gibt der Endnutzer auf der Startseite eine Adresse ein, so gelangt er auf die Search-Seite. Hier findet er alle Restaurants, die seinen Standort beliefern. Hier können zudem weitere Filter getroffen werden. Zum Beispiel kann man hier zwischen Lieferung und Abholung wählen, einen Mindestbestellwert festlegen oder den Radius genauer festlegen. Außerdem kann auch nach der Restaurantkategorie gesucht werden.

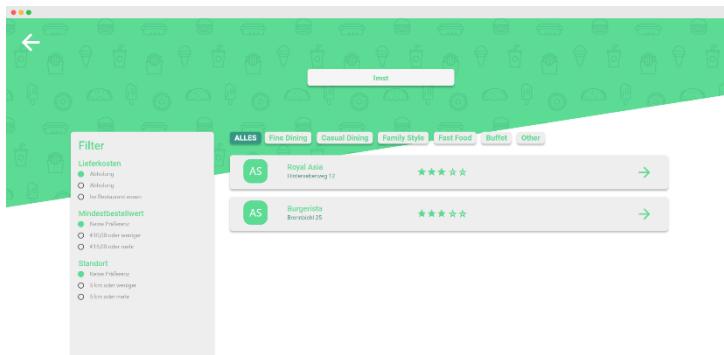


Abbildung 9: Kundenseite – Search-Seite - Desktop

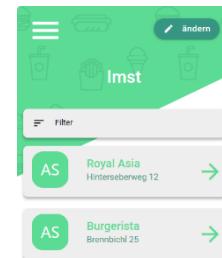


Abbildung 10:
Kundenseite – Search-
Seite- Desktop

Will man als Restaurant den Dienst Lieferrex verwenden, so muss man sich zunächst als Restaurantpartner registrieren. Dafür gibt es die Partner-werden-Seite. Hier findet sich ein Stepper der einen durch alle notwendigen Schritte führt. Jeder wichtige Schritt erhält hier einen eigenen Tab. Ist ein Schritt abgeschlossen, so gelangt man zum nächsten. Folgende Schritte sind für ein erfolgreiches Registrieren notwendig:

1. Beschreibung
2. Grunddaten eingeben
3. Admin Benutzer anlegen
4. Bestätigen und Registrieren

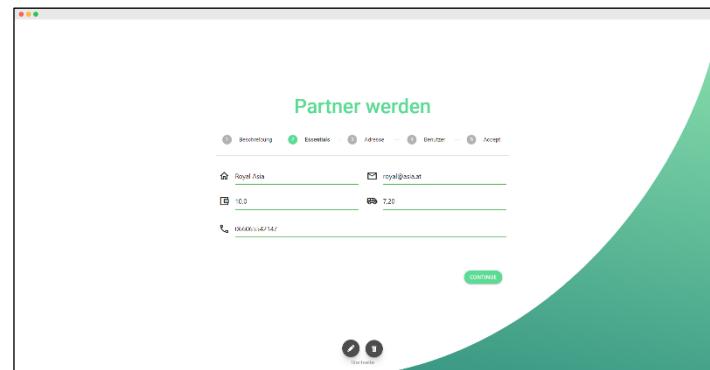


Abbildung 11: Kundenseite - Partner Page - Desktop

Will man den Dienst Lieferrex als Endnutzer verwenden, kann man ebenfalls einen Account erstellen. Dieser ist zwar optional, jedoch kann der Dienst von Nutzern ohne Account nicht ohne Einschränkungen verwendet werden. So können eingeloggte Nutzer zum Beispiel ihre Adresse speichern, Bewertungen abgeben sowie etwas auf den Seiten der Restaurants bestellen. Die Endnutzer-Registrierung wurde erneut mit einem Stepper umgesetzt. Zunächst muss man seine Grunddaten eingeben, danach seine Adresse und zum Schluss noch die AGBs akzeptieren.

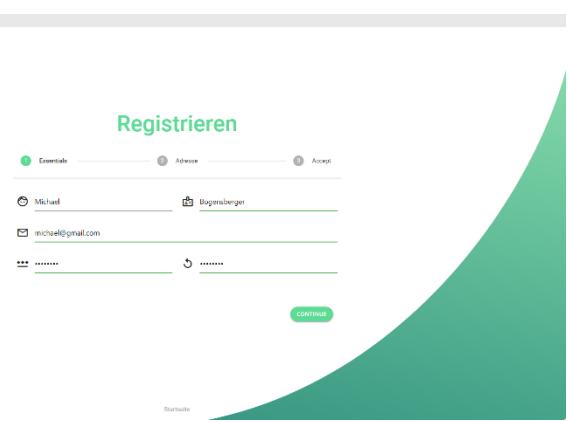


Abbildung 12: Kundenseite – User Register Page - Desktop

Sowohl Endnutzer als auch Restaurantpartner können sich einheitlich auf einer Seite einloggen. Loggt man sich als Endnutzer ein, so gelangt man als eingeloggter User auf die Startseite zurück. Ist man hingegen ein Restaurantbesitzer oder ein Angestellter eines Restaurants, so gelangt man auf das Dashboard.

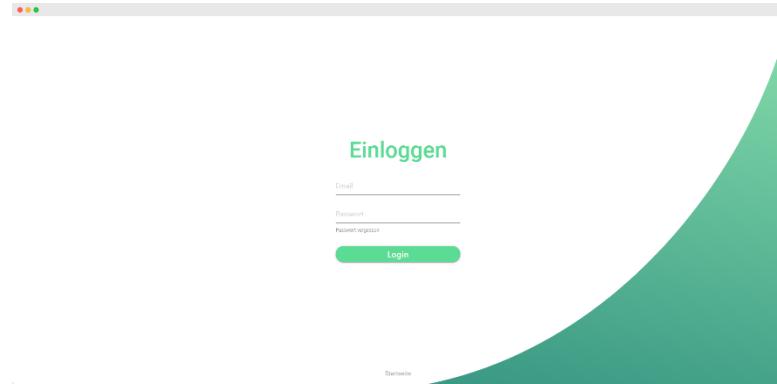


Abbildung 13: Kundenseite – Login Page - Desktop

Will der Endnutzer Daten in seinem Profil ändern, so gibt es eine Seite zum Ändern der Adresse sowie eine Seite zum Ändern des Passworts. Beide Seiten sind über die Startseite erreichbar. Hier muss dann nur noch das Formular ausgefüllt werden und auf speichern gedrückt werden.

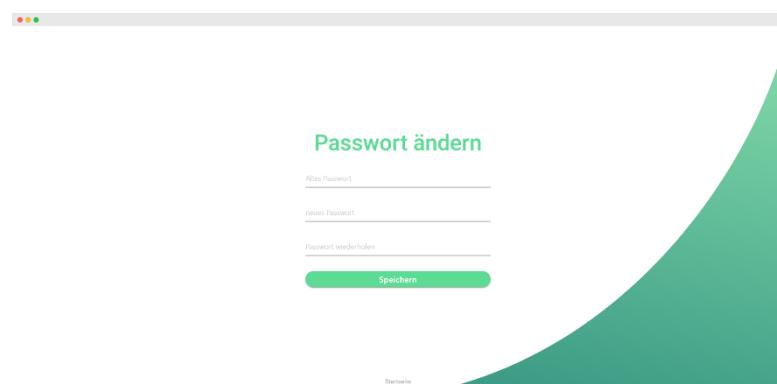


Abbildung 14: Kundenseite – Passwort ändern Page - Desktop

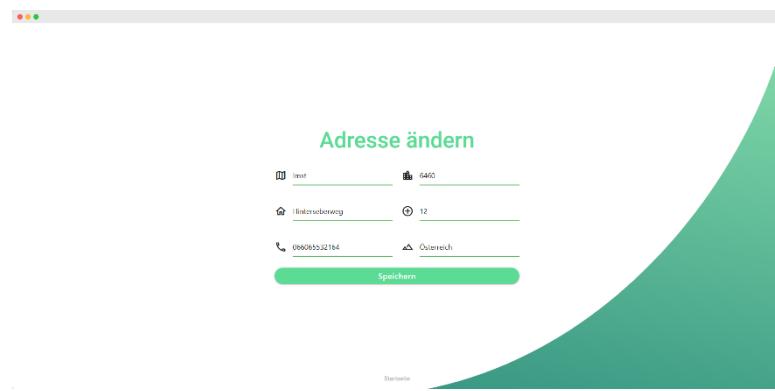


Abbildung 15: Kundenseite – Adresse ändern Page - Desktop

3.2 Dashboard

Wenn sich ein Angestellter oder ein Restaurantbesitzer einloggt, gelangt er zunächst auf die Overview-Seite oder auf die Zubereitungen-Seite auf dem Dashboard. Hier kann er aktuelle Informationen wie zum Beispiel die Seitenaufrufe seiner personalisierten Webseite oder den Umsatz in diesem Monat einsehen. Zudem ist rechts eine Hilfe mit Quicklinks zu finden. In den folgenden Abbildungen ist die Overview-Seite in der Desktop- sowie in der Smartphone Version zu sehen.

Abbildung 17: Dashboard – Overview-Seite - Desktop

Abbildung 16: Dashboard – Overview-Seite - mobile

Auf der Zubereitungsseite kann man alle aktuellen Bestellungen sehen. Diese sind in Abholungen und Lieferungen unterteilt. Diese Seite dient dazu, dem Koch zu zeigen, was er zu kochen hat. Ist der Koch mit einem Gericht fertig, klickt er auf das Häkchen und die Bestellung verschwindet in die Zustellungsseite.

Abbildung 19: Dashboard - Zubereitungsseite - Desktop

Abbildung 18: Dashboard – Zubereitungsseite - mobile

Auf der Zustellungsseite findet man nun jene Gerichte die bereits gekocht und bereit für die Lieferung beziehungsweise Abholung sind. Hier gibt es erneut eine Aufteilung in Lieferung sowie Abholung. Ist ein Gericht geliefert oder abgeholt, so kann der jeweilige Benutzer dieser Seite diese Bestellung als abgehakt markieren. Danach ist die Bestellung voll abgeschlossen.

Abbildung 20: Dashboard - Zustellungsseite - Desktop

Abbildung 21: Dashboard - Zustellungsseite - mobile

Die angebotenen Gerichte eines Restaurants werden auf der Gerichte-Seite erstellt, bearbeitet sowie angezeigt. Hier kann der Restaurantbesitzer Gerichte hinzufügen, löschen, bearbeiten oder sie auf deaktiviert setzen. Diese Seite ist einer der wichtigsten für das Restaurant.

Abbildung 22: Dashboard - Gerichte Page - Desktop

Abbildung 23: Dashboard - Gerichte Page - mobile

Natürlich hat das Restaurant auch die Möglichkeit Bewertungen von Kunden einzusehen. Dazu gibt es die Bewertungen-Seite. Hier findet der Restaurantbesitzer die letzten Bewertungen sowie eine generelle Übersicht mit Kennzahlen wie beispielsweise der Durchschnittsbewertung oder der Anzahl der Bewertungen.

Abbildung 24: Dashboard – Bewertungen-Seite - Desktop

Abbildung 25: Dashboard – Bewertungen-Seite - mobile

Um Benutzer für das Dashboard zu verwalten, gibt es die Benutzerrechte-Seite. Hier kann der Restaurantbesitzer Benutzer anlegen, bearbeiten sowie löschen.

The screenshot shows the desktop version of the dashboard. On the left, there's a sidebar with various restaurant management options like Overview, Zubereitung, Zustellung, Verwaltung, Bewertungen, Benutzerrechte, Zahlungen, Mandant, Suchmaschine, Öffnungszeiten, and Restaurantkategorien. The main area has three cards: '2 Zugänge' (with edit and delete buttons), '1 Administrator' (with edit and delete buttons), and '1 Angestellte'. Below these is a table for managing users ('Zugänge'). It lists two users: 'angestellter@gmail.com' (Max Mustermann, Angestellter) and 'michael@gmail.com' (Michael Bögensberger, Administrator). There are edit and delete buttons for each. To the right is a 'Hinzufügen' (Add) section with a 'Zugang hinzufügen' button.

Abbildung 26: Dashboard - Benutzerrechte Page - Desktop

The screenshot shows the mobile version of the dashboard. It has a similar layout with cards for Zugänge, Administratoren, and Angestellte. The 'Hinzufügen' section is simplified, showing a placeholder for adding a new user. The overall design is more compact for mobile devices.

Abbildung 27: Dashboard - Benutzerrechte Page - mobile

Wenn man einen neuen Benutzer erstellen will, muss man zwischen einem Angestellten und einem Administrator wählen. In folgender Tabelle sind die jeweiligen Zugriffsrechte der Seiten für die Benutzerarten zu sehen. Links in der Tabelle sind die jeweiligen Seiten zu sehen. Rechts sehen sie wer darauf Zugriff hat. Das „R“ steht für den Restaurantbesitzer. Das „A“ steht für den Angestellten.

Seite	Benutzertyp
Overview	R
Zubereitung	R + A
Zustellung	R + A
Webseite	R
Bewertungen	R
Benutzerrechte	R
Zahlungen	R
Mandant	R
Öffnungszeiten	R
Restaurantkategorien	R

Abbildung 28: Dashboard - Benutzerrechte Page - Zugriffsrechte

Auf der Zahlungen-Seite kann das Restaurant alle möglichen Informationen über Zahlungseingänge sehen. Hier lassen sich unter anderem der Umsatz dieses Jahres, der Umsatz dieses Monats, die letzten Zahlungen und Diagramme zum Umsatz sowie Bestellungen finden.



Abbildung 29: Dashboard – Zahlungen-Seite - Desktop

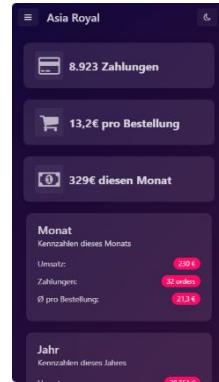


Abbildung 30: Dashboard – Zahlungen-Seite - mobile

Die Mandanten-Seite dient dazu die wichtigsten Informationen des Restaurants einzusehen sowie zu bearbeiten. Hier kann der Restaurantbesitzer Allgemeine Daten wie zum Beispiel der Firmenname oder die Telefonnummer ändern. Des Weiteren kann der Restaurantbesitzer auch Daten über den Standort ändern oder auch Einstellungen über die Lieferkosten sowie den Mindestbestellwert treffen.

The screenshot shows the desktop version of the dashboard under the "Mandant" section. It includes a header for "Royal Asia" and a user profile for "Michael B". The main area is divided into three sections: "Allgemein" (General), "Standort" (Location), and "Sonstiges" (Other). In the "Allgemein" section, fields are filled with "Firmenname: Royal Asia", "Email: asia@gmail.com", and "Telefon: 06501234567". In the "Standort" section, the "Land" is set to "Österreich" and the "Ort" is "Zell am See". In the "Sonstiges" section, "Lieferkosten" is set to "3,5 €" and "Mindestbestellwert" is set to "7,5 €". A "speichern" (save) button is at the bottom.

Abbildung 31: Dashboard – Mandant-Seite - Desktop

The screenshot shows the mobile version of the dashboard under the "Mandant" section. It has a similar structure to the desktop version, with a header for "Royal Asia" and a user profile for "Michael B". The "Allgemein", "Standort", and "Sonstiges" sections are displayed, showing the same data as the desktop version. The overall design is responsive and optimized for mobile devices.

Abbildung 32: Dashboard – Mandant-Seite - mobile

Auf der Öffnungszeiten-Seite kann der Restaurantbesitzer die Öffnungszeiten seines Restaurants anpassen. Diese werden dann dem Endnutzer auf der personalisierten Webseite angezeigt. Da die Benutzeroberfläche für mobile Endgeräte zu aufwändig zu bedienen wäre, ist die Öffnungszeiten-Seite nur auf Desktops verfügbar.

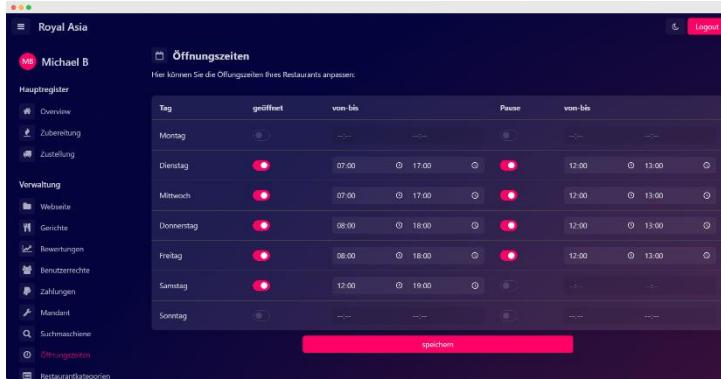


Abbildung 34: Dashboard – Öffnungszeiten-Seite - Desktop



Abbildung 33: Dashboard – Öffnungszeiten-Seite - mobile

Jedes Restaurant kann eine Restaurantkategorie haben. Nach jener Kategorie kann der Endnutzer dann auf der Hauptseite suchen beziehungsweise filtern. Diese Seite dient also dazu, eine der sechs vorgefertigten Restaurantkategorien zu wählen, damit Endnutzer ein besseres Sucherlebnis haben.

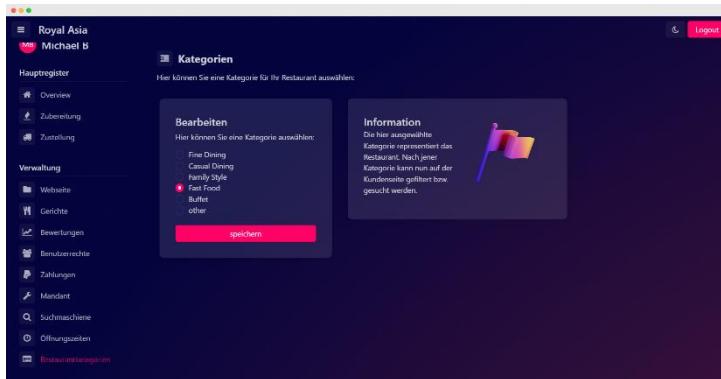


Abbildung 35: Dashboard - Restaurantkategorien Page - Desktop

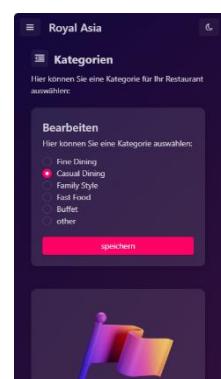


Abbildung 36: Dashboard - Restaurantkategorien Page - mobile

4 Eingesetzte Technologien

Im Zuge der Planung sowie der Entwicklung unseres Projektes kommen selbstverständlich auch Werkzeuge für zum Beispiel die Erstellung von Grafiken und Modell oder der Versionierung von Quellcode zum Einsatz. In folgendem Abschnitt sind die von uns verwendeten Werkzeuge aufgelistet.

4.1 Materialize

Materialize ist ein Frontend-Framework basierend auf der von Google entwickelten Designsprache Material Design. Es ist ein Framework, dass sich sehr gut für responsives Webdesign eignet. Das Projektteam hat sich für Materialize entschieden, da man relativ unkompliziert ansprechende User Interfaces nach dem Material Design Prinzip erstellen kann. Zudem haben uns die Komponenten für die mobile Darstellung überzeugt. Materialize wurde in unserem Projekt verwendet, um das Frontend der Hauptseite sowie des Baukastensystems zu realisieren. Eine sehr ähnliche Alternative wäre zum Beispiel das von Google entworfene Frontend-Framework Material Design. Dies richtet sich jedoch eher an JavaScript Frameworks wie Angular. (Materialize documentation, 2022)

4.2 Materialize Stepper

Der Materialize Stepper ist ein Plugin für das von uns verwendete Framework Materialize. Das Plugin erweitert das Framework um einen Stepper. Stepper sind Komponenten, die einen verbundenen Prozess in kleine Schritte beziehungsweise in kleine User Interfaces aufteilen. Dies macht einen längeren Prozess (zb. Die Restaurant-Registrierung) übersichtlicher. Jener Stepper wurde unter anderem bei der Registrierung der Kunden sowie der Restaurant-Partner verwendet. Als Alternative könnte man natürlich selbst ein User Interface für jenen Zweck entwerfen. Da ein Stepper jedoch nicht so einfach zu implementieren ist und es bereits für Materialize eine Stepper Erweiterung gibt hat sich das Projektteam für jene Erweiterung entschieden. (Materialize Stepper documentation, 2022)

4.3 js-cookie

Damit auf der Hauptseite mit Cookies gearbeitet werden kann wird eine Implementierung für das Handhaben von Cookies benötigt. Da sich Cookies mit normalen JavaScript sehr umständlich erstellen lassen hat sich das Projektteam für die Library js-cookie entschieden. Diese ist sehr leicht zu verwenden und hat eine sehr geringe Bundle-Size. Das kommt daher, da js-cookie nur die notwendigsten Features anbietet.

4.4 Halfmoon

Halfmoon ist ein Frontend-Framework, das sich an Bootstrap orientiert. Jedoch hat Halfmoon Features, die sich exzellent für unser Projekt eignen. So hat Halfmoon unter anderem einen eingebauten Dark Mode, der sich sehr leicht anpassen lässt und voll anpassbare CSS-Variablen. Halfmoon eignet sich also sehr für das Erstellen von Dashboards. Der Aufbau der Komponenten in Halfmoon ist ebenfalls sehr für die Entwicklung von Dashboards ausgerichtet. Jedoch hat Halfmoon nicht so viele Komponenten wie beispielweise das verwandte Bootstrap. Dennoch hat sich das Projektteam aus den oben genannten Gründen für Halfmoon entschieden. Mithilfe von Halfmoon wurde das Frontend des Dashboards realisiert. (Halfmoon dokumentation, 2022)

4.5 JQuery

JQuery ist eine beliebte JavaScript-Bibliothek, die unter anderem Funktionen zur DOM-Manipulation bereitstellt. Durch JQuery lässt sich unter anderem eine Menge Code sparen. Zudem besitzt JQuery ein breitgefächertes Repertoire an standardkonformen Funktionen zur Manipulation des „Document Object Models“. Da im Projekt hin und wieder für zum Beispiel die Validierung der Öffnungszeiten oder der asynchronen Änderung von Daten der DOM bearbeitet werden muss, kommt einem JQuery sehr gelegen. Zudem haben alle Mitglieder des Projektteams bereits Erfahrungen mit JQuery gesammelt. Deshalb fiel die Wahl für uns auf JQuery. Eine eher unpraktikable Alternative wäre es, jene

Funktionalitäten in Vanilla-JavaScript zu lösen. Man könnte aber auch JavaScript Frameworks wie zum Beispiel VueJs oder Angular verwenden. Jedoch würden diese Frameworks die Anforderungen übertreffen und einen zu hohen Zeitaufwand verursachen. (JQuery documentation, 2022)

4.6 Spring Boot

Spring Boot ist ein Java-basiertes Open-Source-Framework, dass die Java Entwicklung von Business-Applikationen stark vereinfacht. Es basiert auf dem Spring Framework und beinhaltet schon vorab einige Konfigurationen, dass die Entwicklung zusätzlich benutzerfreundlicher und einfacher macht. Es wird die Version 2.6.0 von Spring Boot verwendet, da es zum Zeitpunkt der Erstellung des Projekts die beste und aktuelle Version war. (Baeldung, 2022)

Durch die vereinfachte Maven-Konfiguration der „Starter“-POMs (Project Object Models) ist es ganz einfach Drittanbieter-Bibliotheken – auch Dependencies genannt – hinzuzufügen. Dadurch ist es möglich die einzelnen Services – die Notwendig für eine Business-Applikation sind – ohne größeren Aufwand zu implementieren.

Mithilfe der Starter-Web Bibliothek ist es möglich einen Apache Tomcat Webserver aufzusetzen. Dieser verwendet den Port 8080 und läuft sobald man die Applikation startet. Desweiterem inkludiert es die Möglichkeit REST-Schnittstellen zu erstellen. Durch diese Schnittstelle kann die Applikation nach außen kommunizieren.

Da im Frontend das Thymleaf verwendet wird auch die „spring-boot-starter-thymeleaf“ Dependency in die pom.xml verwendet. Somit kann man Thymleaf-Models über dem Webinterface mit senden.

Die „mysql-connector-java“ und die „spring-boot-starter-data-jpa“ Bibliothek ermöglichen es in einem vereinfachten Format mit einer MySql Datenbank zu kommunizieren.

Mit der Lombok-Bibliothek kann man die Länge des Quellcodes erheblich verkürzen und die Lesbarkeit erhöhen. Ziel ist somit die Vermeidung von repetitivem Boiler Plate-Code. Mit Annotationen kann man gewisse Parameter mitgeben, die dann zur Laufzeit in validen Java-Code umgewandelt wird.

Die „spring-boot-starter-security“ Bibliothek ermöglicht es Authentifizierungs-Verfahren ganz einfach in Spring Boot zu implementieren. Man kann somit gewisse Ressourcen auf dem Webserver schützen, und nur mit autorisiertem Zugang freigeben. Die „thymeleaf-extras-springsecurity5“ Dependency wird in diesem Zusammenhang ebenso benötigt, da es Formularbasiertes Login mit Thymleaf ermöglicht.

Um die Google Maps API in Java verwenden zu können, wird die „google-maps-services“ SDK eingebunden. Sie stellt Klassen und Methoden zur Verfügung, um Google API-Aufrufe so einfach wie möglich zu gestalten.

Für PayPal werden auch zwei Dependencies benötigt. Für die Bezahlung über den Browser, wird die „rest-api-sdk“ SDK verwendet. Um die Geldbeträge, dann an das Restaurant zu senden, wird die „payouts-sdk“ benötigt.

4.7 MySQL

MySQL ist ein relationales Datenbanksystem. Das Datenbanksystem ist für die Datenspeicherung von Webservices zuständig. Mithilfe der Structured Query Language können Daten einfach abgefragt werden. In der SQL-Datenbank werden die Schemen für die Daten Tabellarisch angelegt, das heißt es gibt in dieser Tabelle Spalten sowie Zeilen. Die Spalten werden mithilfe eines Namens und eines Typen erstellt. Eine Spalte beinhaltet den Primären Schlüssel, dieser Primär-Schlüssel ist eindeutig und kann nicht für dieselbe Tabelle verwendet werden. Der Primär-Schlüssel wird dafür verwendet, um

Datensätze eindeutig zu identifizieren. Das Datenbanksystem MYSQL wurde im Unterricht behandelt, dadurch wurde das Aneignen aus verschiedenen Quellen beschleunigt. Es gibt noch andere Datenbanksysteme, wie Key-Value oder Dokumentenbasierende Datenbanksysteme, die wurden jedoch zum Zeitpunkt des Projektstartes noch nicht behandelt. Die Entscheidung viel dadurch sehr schnell auf MySQL.

4.8 Thymeleaf

Thymeleaf ist eine Java XML/XHTML/HTML5 template engine. Das Hauptziel von Thymeleaf ist, dass HTML im Browser korrekt angezeigt werden kann und auch als statische Prototypen funktionieren. Im Template Mode können folgende Prozesse verwendet werden:

- HTML
- XML
- TEXT
- JAVASCRIPT
- CSS
- RAW

Das kleinste Detail kann mit Thymeleaf angepasst und realisiert werden. In der Kernbibliothek von Thymeleaf ist standardmäßig ein Dialekt namens Standard Dialect integriert, dieser Dialect sollte für die meisten Benutzer ausreichend sein. Thymeleaf ist für die Darstellung auf der Webseite zuständig, z.B. sind die Anzahl der Gerichte sowie den Namen des Gerichtes dynamisch dargestellt. Mithilfe von Maven kann die Library schnell aus dem Internet geholt werden. Spring Boot bietet schon von Anfang ein Webkit an, im Webkit ist Thymeleaf beinhaltet. Ein typisches Merkmal für Thymeleaf ist das im Quelltext th:action, th:field oder th:text oft vorkommen. Diese Merkmale sind Syntaxen, dieser Syntax ist speziell bei Thymeleaf dadurch können Texte oder Felder übergeben werden. (Thymeleaf documentation, 2022)

4.9 Spring Security

Spring Security Authentifizierungs- und Zugriffskontroll-Framework mit man den Zugriff auf Ressourcen des Webservers konfigurieren kann. (Baeldung, 2022)

Der Authentifizierungsprozess basiert auf verschiedenen Filtern und deren Interaktion untereinander. Wenn ein User seine Authentifizierung über einen Browser Request sendet, folgt entweder eine erfolgreiche Anmeldung oder in einem HTTP-403-Error. Falls die Autorisierung erfolgreich war, wird am Webserver eine Session mit einem bestimmten Token erstellt. Dieser Token wird als Response dem Client zurückgesendet und als Cookie gespeichert. Immer wenn der Client nun eine http-Anfrage macht, muss dieser Token mitgeschickt werden, um den User authentifizieren.

4.10 Visual Paradigm

Bei Visual Paradigm handelt es sich um ein sehr weit verbreitetes und bekanntes Werkzeug zum Erstellen von UML Grafiken. Visual Paradigm bietet eine frei verwendbare Version (Community Edition) an. Da öfters UML-Grafiken für zum Beispiel das ER-Modell, Use-Cases oder Klassendiagramme angefertigt werden müssen, ist man auf so ein Tool angewiesen. Visual Paradigm lässt sich intuitiv steuern und bietet zudem sehr verschiedene Vorlagen an. Deshalb viel für uns die Wahl eindeutig auf Visual Paradigm. Alternativen wären hier unter anderem Lucidchart, Draw.io oder Dia. Jedoch bieten die genannten Tools keine so intuitive Handhabung an. Zudem fehlt es einigen Alternativen an Features beziehungsweise Vorlagen.

4.11 PayPal API

Damit Kunden ihre Bestellungen bezahlen können verwenden wir die PayPal API. Somit können wir den Geldverkehr zwischen Kunde und Restaurant ohne großes Risiko realisieren.

Währenden der Entwicklungsphase wird der Sandbox-Modus von PayPal verwendet, damit man unbeschwert Geldbeträge versenden kann. Dabei muss in die entsprechende Dependency eingebunden werden. Über die REST API können die Geldbeträge gesendet werden.

PayPal wird verwendet, da es die einfachste Möglichkeit eine standardisierte Zahlungsmethode zu implementieren ist. (PayPal API documentation, 2022)

4.12 Google Maps API

Restaurant sollen ihren Standort auf einer Karte darstellen können. Um dies zu ermöglichen, wird die Google Maps API verwendet. Des Weiteren soll der Benutzer Restaurant in unmittelbarer Nähe finden können. Für die Registrierung ist die Google Maps API auch unabdinglich, da nur somit exakte Lieferadressen für die Lieferung, und valide Adressen für die Restaurantsuche gespeichert werden können.

Mithilfe der API sollen Karten dargestellt, Adressen in Geodaten übersetzt und Distanzen berechnet werden. (Google Maps Platform, 2022)

4.13 GIT & GitHub

GIT (Global Information Tracker) ist eine Software, die das Protokollieren von Projekten-Versionen ermöglicht. Mit GIT werden stets alle Änderungen am Projekt lokal gespeichert. Hiermit ist es möglich, in ältere Versionen einzusehen oder auf diese zurückzuspringen. Über GitHub kann das Projekt auch in der Cloud gespeichert werden. Dies ermöglicht eine gemeinsame Arbeit des Projektteams. GIT stellt hier die Änderungen der verschiedenen Mitglieder dar, um alle auf dem neuesten Stand zu halten. Heutzutage werden Projekte fast ausschließlich mit GIT verwaltet.

4.14 IntelliJ

IntelliJ IDEA ist eine IDE (Integrated Development Environment) von JetBrains für Java. Diese Software wurde bereits in Unterricht für verschiedenste Projekte verwendet. Es wird hierbei die Ultimate Edition benutzt, da diese eine höhere Funktionalität bietet. Diese kann über eine Schullizenz erworben werden. Im Rahmen der Diplomarbeit wird diese hauptsächlich verwendet, um das Backend zu implementieren.

Diese IDE bietet viele Funktionen, die das Schreiben von Code erleichtern. Das Verwalten von Maven-Projekten ist in IntelliJ IDEA integriert. Maven-Projekte können einfach erstellt und verwaltet werden. Eine automatische Code-Vervollständigung, nimmt dem Programmierer Arbeit beim Schreiben von Variablen oder großen Strukturen ab. IntelliJ IDEA bietet auch etwaige Keyboard-Shortcuts, mit denen das Schreiben von Code optimiert werden kann. Auch ist GIT in die IDE miteingebaut. Mithilfe von GIT kann in IntelliJ IDEA das Projekt schnell verwaltet werden. Änderungen können mithilfe weniger Knopfdrücke gespeichert werden.

Als Alternative zu IntelliJ IDEA gibt es Visual Studio Code von Microsoft. Visual Studio Code bietet vergleichbare Funktionen, muss aber vom Benutzer vorerst konfiguriert werden. Hier müssen beispielsweise Erweiterungen installiert werden, um die Unterstützung von Java, Spring Boot und weiterem zu ermöglichen.

5 Problemanaylse

5.1 Use-Case-Analyse

Ein Use Case ist eine Beschreibung, wie Benutzer mögliche Szenarien in einem System ausführen. Sie zeigen explizit Aktionen, aus der Sicht des Anwenders und erklären, wie das System darauf reagiert.

Der Akteur des Systems ist meist eine Person, eine Rolle, eine Organisation oder ein anderes System.

Dieser Akteur interagiert mit einem System, um ein bestimmtes Ziel in einer definierten Folge von Aktionen zu erreichen.

In diesem Abschnitt sind die für unser Projekt definierten User Stories als Aufzählung beschrieben. Diese sind zunächst in Kunde, Administrator, Mitarbeiter und Restaurant aufgeteilt. Der Kunde ist der normale Besucher unserer Webseite. Mit Restaurant ist der jeweilige Besitzer des Restaurants

gemeint. Der Mitarbeiter ist ein Angestellter des jeweiligen Restaurants. Administratoren sind die Verwalter des gesamten Systems.

An dieser Stelle sind die User Stories für den Kunden zu sehen:

- Als Kunde möchte ich Restaurants in meiner Nähe finden, um bei diesen Essen zu bestellen.
- Als Kunde möchte ich Restaurants in meiner Nähe finden, um bei diesen Essen abzuholen.
- Als Kunde möchte ich mein Profil auf der Webseite speichern können, um Aufwand beim Bestellen zu sparen.
- Als Kunde möchte ich über die Option eines Dark-Mode verfügen, um es meiner Präferenz anzupassen.
- Als Kunde möchte ich, dass die Webseite auf sämtlichen Endgeräten verfügbar ist, um auch von unterwegs aus Essen bestellen zu können.

An dieser Stelle sind die User Stories für den Administrator zu sehen:

- Als Administrator möchte ich Restaurant und auch Kunden verwalten können, um Probleme zu beheben.
- Als Administrator möchte ich Informationen (Anzahl Kunden, Verkäufe, Restaurants, usw.) von Restaurants und Kunden erhalten können, um zu sehen, wie rentabel unser Geschäft ist.
- Als Administrator möchte ich Server verwalten können, um Probleme zu beheben oder Funktionen anzupassen.

An dieser Stelle sind die User Stories für das Restaurant zu sehen:

- Als Restaurant möchte ich Statistiken von meiner Webseite bekommen (Umsatz, Anzahl Bestellungen, Lieblingsprodukt, usw.), um mein Angebot immer besser optimieren zu können.
- Als Restaurant möchte ich eine individuelle Webseite gestalten können, um diese für Kunden attraktiver zu machen.
- Als Restaurant möchte ich Informationen über aktuelle Bestellungen erhalten, um diese zu erfüllen.
- Als Restaurant möchte ich einstellen können, welche Produkte ich anbiete, um die Bedürfnisse meiner Kunden erfüllen zu können.
- Als Restaurant möchte ich Zugriffsrechte für mein Personal anpassen, um nicht alle Informationen meinem Personal zur Verfügung stellen zu müssen.

An dieser Stelle sind die User Stories für den Mitarbeiter zu sehen:

- Als Mitarbeiter möchte ich Informationen über aktuelle Bestellungen erhalten, um diese zu erfüllen.

Im Folgenden werden die drei wichtigsten Use-Cases genauer in Form von Tabellen beschrieben. Sie beinhalten den Namen des Use-Cases, eine Beschreibung, Vorbedingungen, Essenzielle Schritte und weiteres.

In *Tabelle 7: Use-Case "Essen bestellen"* wird der Bestellprozess eines Kunden dargestellt. Es wird der Ablauf einer Bestellung erklärt, unter welchen Bedingungen diese erfolgen kann und welche Ausnahmefälle vorhanden sind.

NR	UC-2021-001		
Name	Essen bestellen		
Akteur	Kunde		
Trigger	Essen bestellen		
Kurzbeschreibung	Der Kunde kann über die Webseite Essen bestellen. Er kann seine Wahl zwischen vielen verschiedenen Restaurants treffen. Bei der Bestellung müssen Kundendaten eingegeben werden und eine erfolgreiche Zahlung vorhanden sein.		
Vorbedingung	Kunde ist angemeldet		
Essenzielle Schritte	Intention des Benutzers	Reaktion des Systems	
	Kunde besucht die Webseite	Webseite zeigt Übersicht	
	Kunde meldet sich an	Webseite meldet den Kunden an und erstellt eine Session	
	Kunde sucht gewünschtes Restaurant	Webseite liefert eine Übersicht mit Restaurant	
	Kunde wählt Gerichte	Webseite liefert eine Übersicht mit Gerichten	
	Kunde gibt Lieferadresse und Zahlung ein	Webseite erstellt Bestellung	
Ausnahmefälle	Webseite nicht verfügbar		
Zeitverhalten	Notwendige Schritte werden vom System in wenigen Milli-Sekunden abgeschlossen.		
Verfügbarkeit	Sofern nicht vom System deaktiviert.		
Fragen / Kommentare			

Tabelle 7: Use-Case "Essen bestellen"

In Tabelle 8: Use-Case "Baukastensystem" wird die Verwendung des Baukastensystems dargestellt. Das Baukastensystem wird von den Restaurants verwendet, um eigene Webseiten zu gestalten.

NR	UC-2021-002		
Name	Gestalten der Webseite mit dem Baukastensystem		
Akteur	Restaurant		
Trigger	Baukastensystem		
Kurzbeschreibung	Restaurants können über ein Baukastensystem eine eigene und individuelle Webseite anlegen. So ist es ihnen möglich, einen ansprechenden Auftritt im Netz hinzulegen und mehr Aufträge zu erzielen.		
Vorbedingung	Restaurant meldet sich an.		
Essentielle Schritte	Intention des Restaurants Restaurant besucht die Webseite Restaurant meldet sich an Restaurant öffnet das Baukastensystem Restaurant bearbeitet die Webseite Restaurant speichert das gewünschte Ergebnis	Reaktion des Systems Webseite zeigt Übersicht Webseite meldet das Restaurant an und erstellt eine Session Webseite liefert eine Übersicht des Baukastensystems mit verschiedenen Einstellungen Webseite zeigt das Ergebnis mit den Änderungen Webseite speichert das Resultat und verwendet dieses anschließend für das Restaurant	
Ausnahmefälle	Webseite nicht verfügbar		
Zeitverhalten	Notwendige Schritte werden vom System in wenigen Milli-Sekunden abgeschlossen.		
Verfügbarkeit	Sofern nicht vom System deaktiviert.		
Fragen / Kommentare			

Tabelle 8: Use-Case "Baukastensystem"

In Tabelle 9: Use-Case "Anmelden" wird die Anmeldung eines Kunden genau beschrieben. Ein Kunde muss sich auf der Webseite anmelden, um Bestellungen aufgeben und personenbezogene Daten für weitere Bestellungen speichern zu können.

NR	UC-2021-003		
Name	Kunde meldet sich an		
Akteur	Kunde		
Trigger	Anmelden		
Kurzbeschreibung	Der Kunde kann sich auf der Webseite anmelden, um verschiedene Daten, wie Zahlungsinformationen, Lieferadresse und Bestellverlauf, zu speichern.		
Vorbedingung	Kunde ist registriert		
Essentielle Schritte	Intention des Benutzers	Reaktion des Systems	
	Kunde besucht die Webseite	Webseite zeigt Übersicht	
	Kunde wählt Login	Webseite liefert eine Maske zum Anmelden	
	Kunde gibt Benutzerdaten ein	Webseite validiert und meldet den Kunden an	
Ausnahmefälle	Webseite nicht verfügbar, Kunde nicht registriert		
Zeitverhalten	Notwendige Schritte werden vom System in wenigen Milli-Sekunden abgeschlossen.		
Verfügbarkeit	Sofern nicht vom System deaktiviert.		
Fragen / Kommentare			

Tabelle 9: Use-Case "Anmelden"

In folgendem Abschnitt sind die Use-Case-Diagramme zum Restaurant-Besitzer sowie zum Kunden dargestellt. Zuerst ist das Use-Case-Diagramm zum Restaurant-Besitzer zu sehen. Darunter sieht man das Use-Case-Diagramm zum Kunden.

Der Akteur in der (Abbildung 37) ist der Restaurant Besitzer. Im ersten Schritt muss sich der Kunde einloggen um auf das Baukastensystem zugreifen zu können. Danach kann der Besitzer die Webseite bearbeiten und zum Schluss muss der Besitzer das gewünschte Design abspeichern

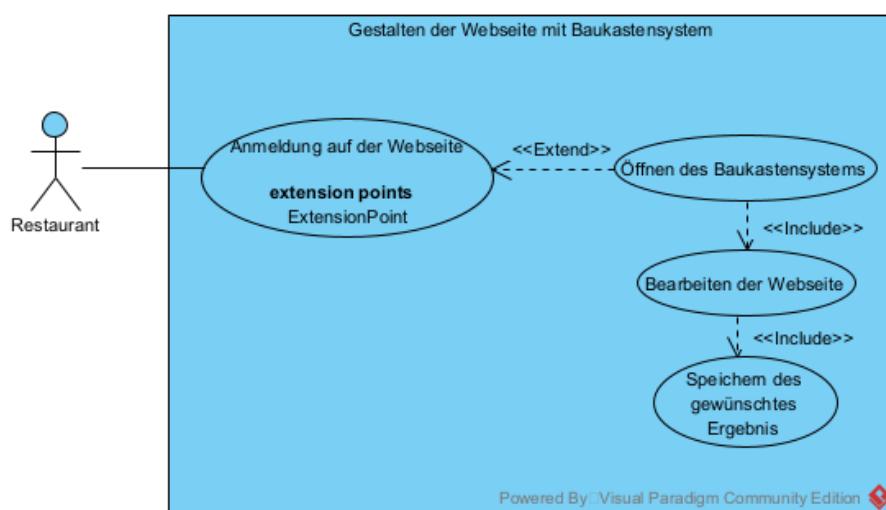


Abbildung 37: Use-Case-Diagramm Restaurant

In der Abbildung 38 ist der Akteur unser Kunde und der Kunde möchte sich Essen bestellen. Bis er zur Essensbestellung kommt muss der Kunde einzelne Schritte durchlaufen. Als erstes muss sich der Kunde Anmelden und sich das gewünschte Restaurant auswählen. Im nächsten Schritt kann sich er die Gerichte auswählen und in den Warenkorb legen, danach schickt der Kunde die Bestellung ab. Zum Schluss muss der Kunde noch die Lieferadresse und seine Zahlungsmethode angeben.

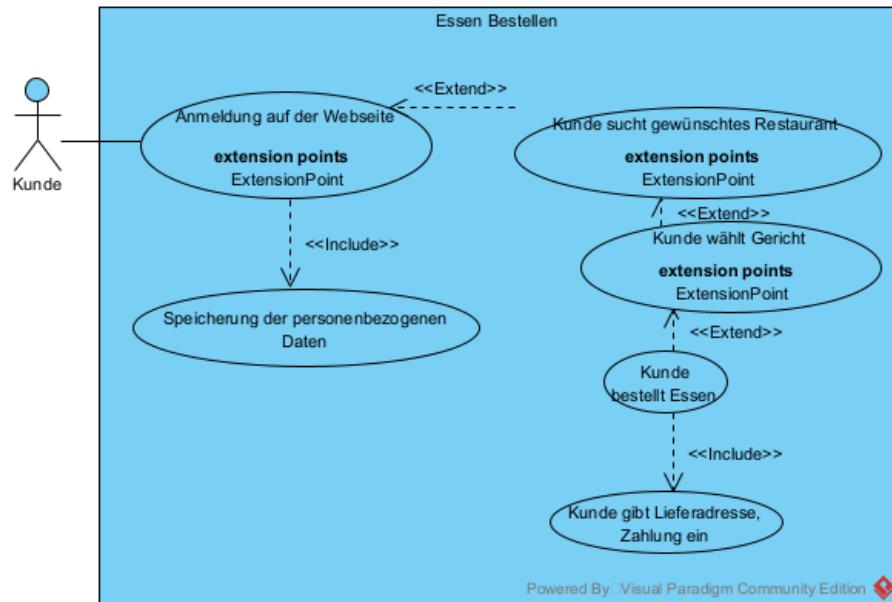


Abbildung 38: Use-Case-Diagramm Kunde

5.2 Domain-Class-Modelling

5.3 User-Interface-Design

Die in folgendem Abschnitt dargestellten Mockups wurden mit wem Web-Tool Figma erstellt. Figma ist ein webbasierter Vektorgrafik-Editor der sich perfekt für die Erstellung von Mockups für Anwendung eignet. Das folgende Mockup stellt die Startseite unseres Projekts dar. Auf jene Seite gelangt man, sobald man unsere Webadresse aufruft. Dies ist der Startpunkt für jeden Benutzer. Von hier aus soll man infolgedessen sich einloggen können, nach Restaurant suchen können und allgemeine Informationen bekommen. Links ist die Desktopdarstellung zu sehen. Rechts die für Smartphones. Die folgenden zwei Mockups (Abbildung 39, Abbildung 40) sind im hellen Farbmodus dargestellt. Zudem gibt es bei machen Mockups einen dunklen Farbmodus. Für den hellen Farbmodus wird das Synonym „Light Mode“ verwendet. Für den dunklen Farbmodus wird das Synonym „Dark Mode“ verwendet.



Abbildung 40: Mockup - Startseite für Desktops

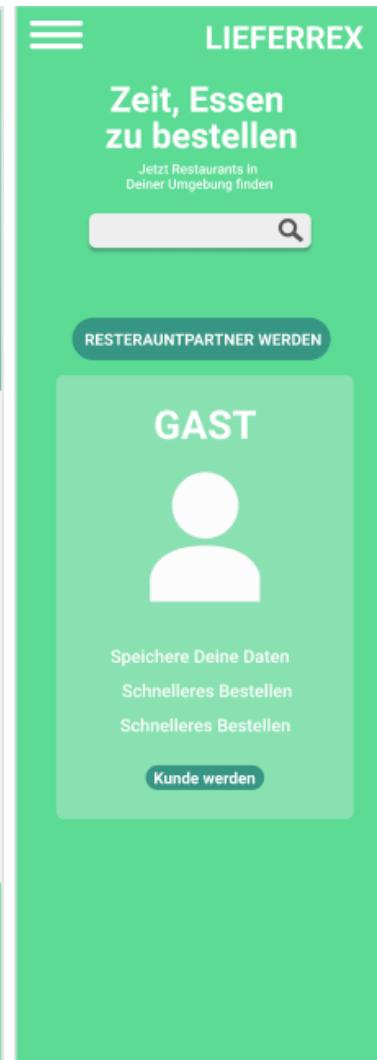


Abbildung 39: Mockup - Startseite für Mobilgeräte

Es soll ebenfalls die Möglichkeit bestehen einen Dark-Mode zu aktivieren. Deshalb ist in folgendem Mockup (*Abbildung 41*) der geplante Dark-Mode in der Desktopdarstellung zu sehen.

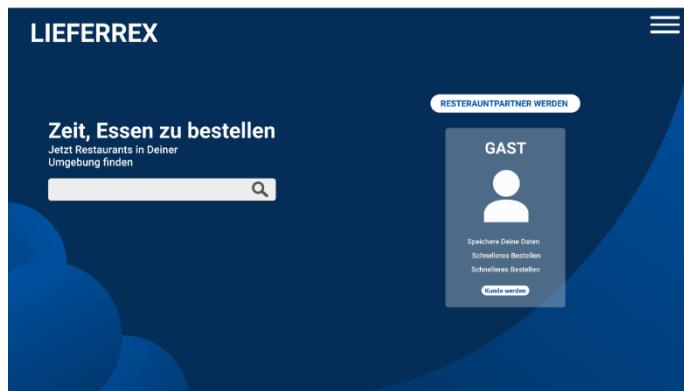


Abbildung 41: Mockup - Startseite für Desktops – Dark Mode

Die folgenden Wireframes (*Abbildung 42*, *Abbildung 43*) zeigen die Seite, die man erhält, wenn man von der Startseite aus nach einem Restaurant in der Nähe sucht. Links ist der normale Farbmodus zu sehen. Rechts ist der Dark-Mode zu sehen. Außerdem ist dies die Desktopdarstellung.

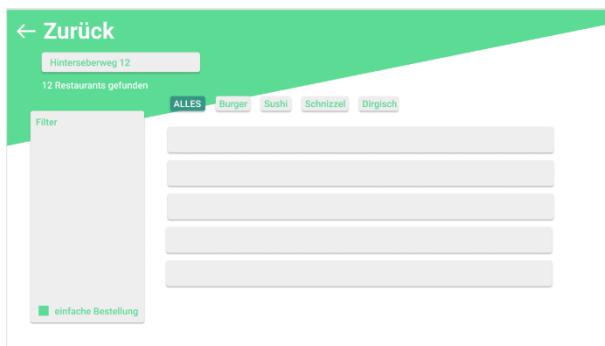


Abbildung 42: Mockup - Ergebnisse für Desktops – Light Mode

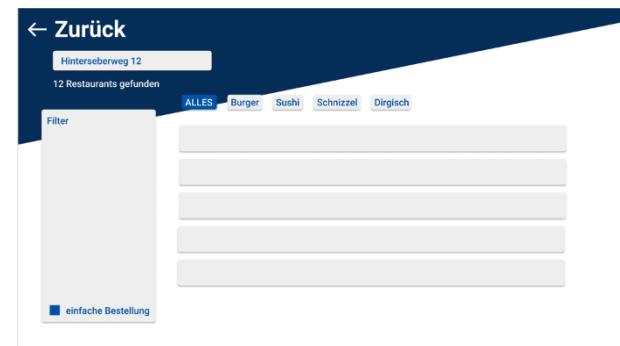


Abbildung 43: Mockup - Ergebnisse für Desktops – Dark Mode

Da das Restaurant natürlich ein Tool zur Verwaltung braucht, haben wir die folgenden zwei Mockups *Abbildung 45*, *Abbildung 44* entworfen. Die folgende Darstellung ist nur dem Restaurantbesitzer vorbehalten. Links ist der normale Farbmodus zu sehen. Rechts ist der Dark-Mode zu sehen.



Abbildung 44: Mockup - Dashboard für Desktops – Light Mode

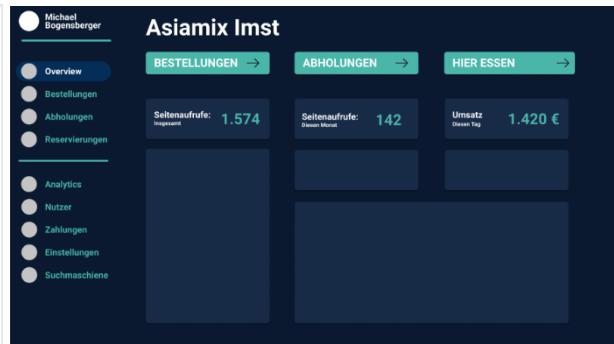


Abbildung 45: Mockup - Dashboard für Desktops – Dark Mode

Der Koch im Restaurant erhält folgende Darstellung. Hierfür gibt es das Mockup zurzeit nur im Dark-Mode. Links ist die Desktopdarstellung (*Abbildung 46*) zu sehen. Rechts (*Abbildung 47*) die für Smartphones.

Abbildung 46: Mockup - Dashboard für Desktops - Mitarbeiter - Dark Mode

Abbildung 47: Mockup - Dashboard für Mobilgeräte - Mitarbeiter - Dark Mode

Um sich registrieren zu können, gibt es ebenfalls ein Mockup (*Abbildung 48*). Die folgend dargestellte Registrierung ist den Kunden vorbehalten. Die Darstellung ist ebenfalls für Desktops und im hellen Farbmodus.

Abbildung 48: Mockup - Registrierung für Desktops - Light Mode

6 Systementwurf

Das entwickelte System besteht hauptsächlich aus dem Frontend und Backend. Um dies mit allen verschiedenen Komponenten möglichst übersichtlich zu strukturieren, wurde das MVC-Pattern angewendet. Dieses sogenannte Model-View-Controller-Pattern sorgt für eine Trennung von Darstellung und Verarbeitung von Daten. Objekte werden in Form von Models gespeichert. Diese Klassen werden mit allen Eigenschaften und Methoden definiert. Für die Ausgabe werden Views verwendet. Views, im Projekt HTML-Dateien, sorgen für eine saubere Präsentation der bereitgestellten Daten. Diese sind im Projekt unter den Ressourcen abgelegt. Ein Controller kümmert sich schließlich um Eingaben vom Benutzer (Aufruf der Webseite, Zugriff über REST-Schnittstelle) und liefert eine View mit entsprechenden Daten wieder.

Desweitern gibt es im Projekt Repositories und Services. Repositories ermöglichen die Kommunikation mit der Datenbank und stellen verschiedenste CRUD-Operationen bereit, um mit Daten aus dieser zu arbeiten. Die Services bauen auf die Repositories auf und erweitern diese um weitere Methoden.

Mandanten, oder auch Restaurant, können sich auf der Webseite über einen Baukasten ihre eigenen Webseiten zusammenbauen. Diese Webseiten bestehen aus verschiedensten Layouts und Modulen, die man nach Belieben wählen und bearbeiten kann. Ein User kann über die Webseite Restaurants durchsuchen und über die individuellen Seiten bestellen. Jedes Restaurant verfügt auch über ein Dashboard, das viele Informationen und Einstellungen beinhaltet. Über das Dashboard werden etwaige Informationen über den Mandaten eingegeben, so unter anderem die Speisekarte oder Öffnungszeiten.

6.1 Architektur

6.1.1 C4-Diagramme

Das C4-Model ist eine Methode zur Modellierung der Architekturen von Softwaresystemen. Es stützt sich dabei auf Modellierungstechniken wie zum Beispiel der Unified Modelling Language (UML). Mithilfe von C4-Diagrammen lassen sich Softwaresysteme gut und verständlich darstellen. In Abbildung 49 C4-Diagramm, Context View ist die Context-View des Projektes als C4-Diagramm zu sehen. Wie man sieht, greifen Nutzer auf die Spring Applikation zu. Die Applikation speichert und liest Daten aus der Datenbank. Des Weiteren kommuniziert die Applikation mit der Google Maps API, um zum Beispiel Adressen in Geodaten umzuwandeln, sowie mit der PayPal API, um Zahlungen abzuschließen.

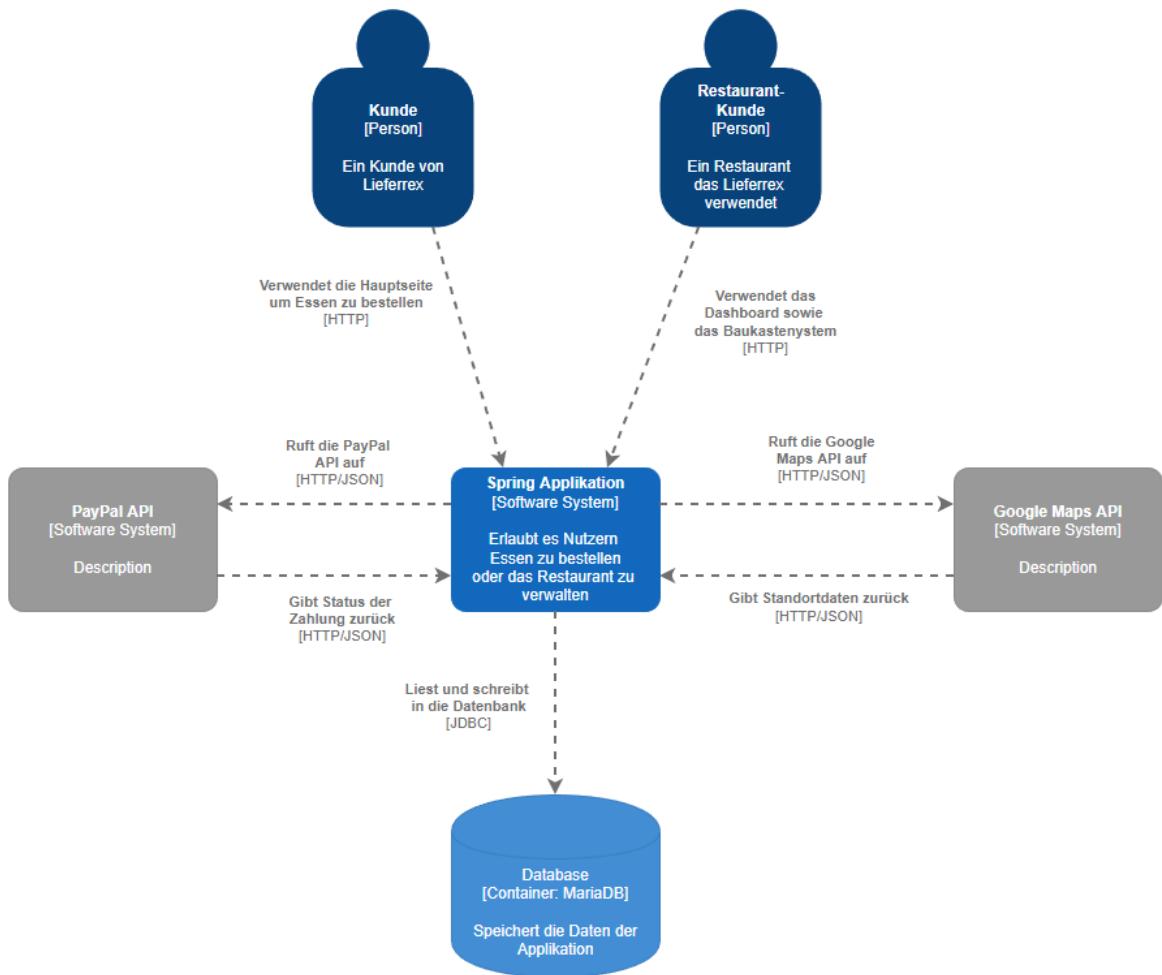


Abbildung 49 C4-Diagramm, Context View

In folgendem C4-Diagramm beziehungsweise in Abbildung 50: C4-Diagramm, Spring Applikation ist die Spring Applikation genauer zu sehen. Hier ist sehr gut die Kommunikation zwischen den einzelnen Komponenten (Controller, Service und Repository) erkennbar. Der User greift zunächst auf den Controller zu. Der Controller wiederum greift auf den Service zu. Innerhalb der Applikation greift der Service auf das Repository zu. Jedoch greift der Service auch auf die jeweiligen APIs zu. Das Repository

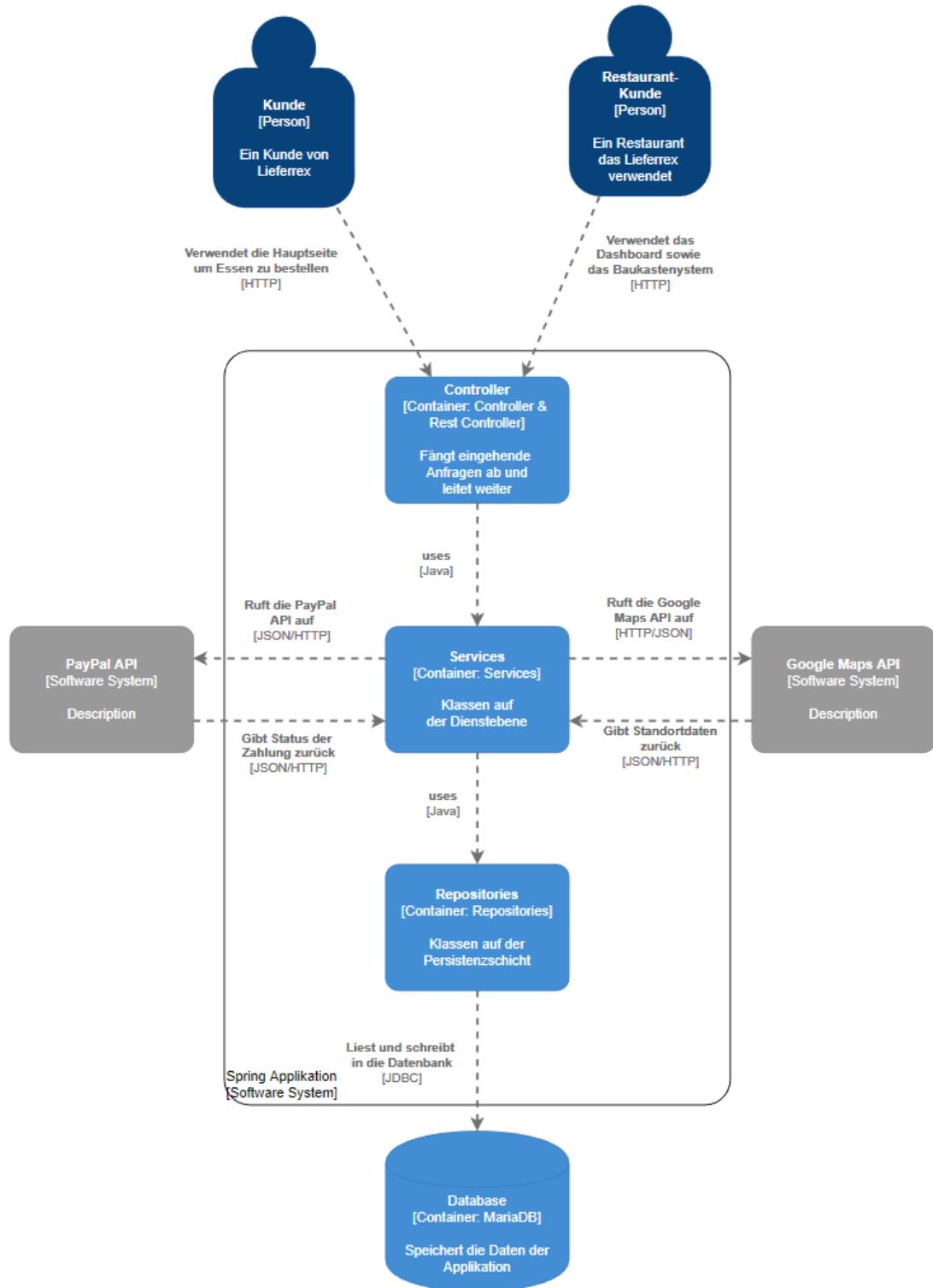


Abbildung 50: C4-Diagramm, Spring Applikation

greift nun auf die Datenbank zu.

6.1.2 Design der Komponenten

6.1.3 Benutzerschnittstellen

6.1.4 Datenhaltungskonzept

Im Projekt wird MySQL benutzt um die Daten, die vom dem User kommen, in die Datenbank zu speichern. Durch das Speichern dieser Daten auf einen auf Server, können jederzeit die Daten abgerufen werden. Ein Anwendungsfall ist z.B. die Speicherung von Gerichten sowie die Erstellung der Angestellten-Accounts. Die MYSQL Konfiguration kann unter resources und application-dev.properties entnommen werden. In dieser Konfiguration ist die URL des MySQL Server sowie Username und Passwort enthalten, auch ist der Treiber für die MYSQL Verbindung enthalten. Für die Darstellung einer Tabelle werden oftmals ein ER-Diagramm vorgefertigt. Für das Projekt ist das ER-Diagramm in der Abbildung 51: Vollständiges ER-Diagramm verwendet worden.

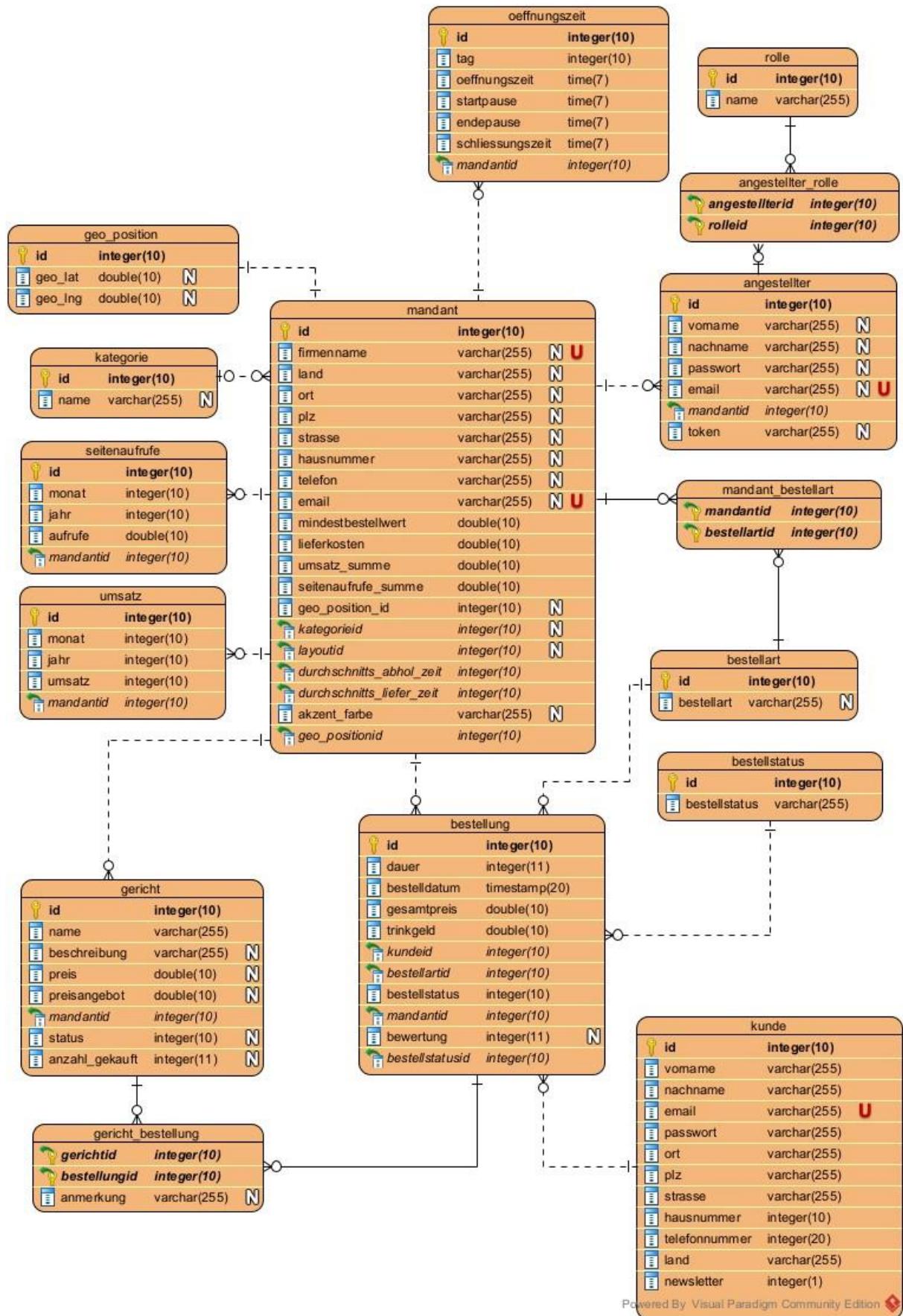


Abbildung 51: Vollständiges ER-Diagramm

In Abbildung XX ist das ER-Modell der Applikation exklusive der Baukastenlogik.

Zentraler Dreh und Angelpunkt ist dabei die Mandanten-Tabelle. Ein Datensatz repräsentiert dabei die Entität eines Restaurants. Die Tabelle enthält Kontaktdaten, wie E-Mail-Adresse oder Telefonnummer und Adressdaten, um den Ort des Restaurants genau zu bestimmen.

Um die genaue Geoposition zu bestimmen, existiert die Geoposition-Tabelle, die den Längengrad und Breitengrad eines Restaurants beinhaltet.

Die Kategorie-Tabelle beinhaltet alle Kategorien, mit der sich ein Restaurant identifizieren kann.

In der Seitenaufruf-Tabelle ist die Logik hinterlegt, um die Anzahl an Seitenaufrufe pro Monat eines Mandanten zu zählen.

Die Gericht-Tabelle beinhaltet alle Attribute, die ein Restaurant benötigt, um persistieren.

Immer dann, wenn eine Bestellung erstellt wird, wird ein Eintrag in der Bestellungstabelle gemacht. Diese Bestellung enthält einen Mandanten, einen Kunden und ein oder mehrere Gerichte.

Die Kunden-Tabelle enthält Persönliche Informationen eines Kunden, der Essen bei einem Restaurant bestellen möchte.

In der Bestellstatus-Tabelle steht der aktuelle Status der Bestellung.

Die Bestellart-Tabelle entscheidet, welche Art von Bestellung bestellt worden ist.

Die Angestellter-Tabelle beinhaltet alle Informationen, die ein Angestellter benötigt. Um Berechtigungen zu verteilen steht diese Tabelle mit der Rollen-Tabelle in einer „1:n“ Beziehung.

In der Öffnungszeiten-Tabelle steht für jeden Tag die Öffnungszeit des Restaurants.

6.1.5 Konzept für Ausnahmebehandlung

6.1.6 Sicherheitskonzept

Da wir verschiedene Benutzer haben, die verschiedene Berechtigungen besitzen, hat jeder Benutzer auch eine Rolle. Es gibt die Rollen User, Mandant und Angestellter. Als User soll man in der Lage sein Essen zu bestellen. Als Mandant muss man alle eingehenden Bestellungen und Zahlungen sehen, und seine Restaurant-Seite bearbeiten können. Als Angestellter soll man wissen, welche Gerichte man kochen muss und sie, als erledigt markieren, sobald die Bestellung zubereitet ist.

Spring Security ist ein Framework, das Authentifizierung und Autorisierung in Spring Boot ermöglicht.

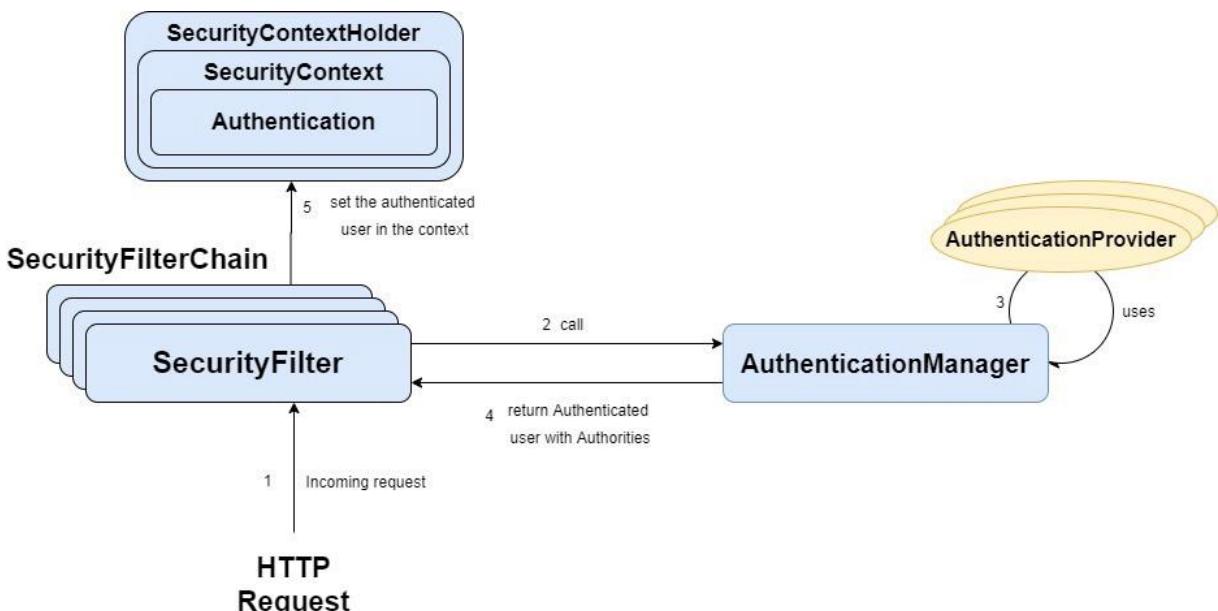
Authentifizierung ist der Prozess wo Anmeldeinformationen – wie Benutzername und Passwort – geprüft werden, um festzustellen, ob die Person ist, für die Sie sich ausgibt. Oder einfach ausgedrückt: Bei der Authentifizierung geht es darum, zu wissen, wer Sie sind.

Autorisierung ist der Prozess, um festzustellen, ob der authentifizierte Benutzer Zugriff auf eine bestimmte Ressource hat. Einfach ausgedrückt, bei der Autorisierung geht es darum zu wissen, ob Sie das Recht haben, auf das zuzugreifen, was Sie wollen oder nicht.

Um Spring Security zu verstehen, müssen folgende Begrifflichkeiten erklärt werden:

- Principal: Aktuell angemeldeter Benutzer
- GrantedAuthority: Erteilte Erlaubnis an den Benutzer
- AuthenticationManager: Controller im Authentifizierungsprozess. Authentifizierter Benutzer, wird über authenticate() in Memory gespeichert.

- AuthenticationProvider: Interface, die zum Datenspeicher zugeordnet ist, um Logindaten zu speichern
- UserDetails: Datenobjekt, das die Anmeldeinformationen des Benutzers, aber auch die Rolle dieses Benutzers enthält
- UserDetailsService: Sammelt die Benutzeranmeldeinformationen, Berechtigungen (Rollen) und erstellt ein UserDetails-Objekt.



Quelle

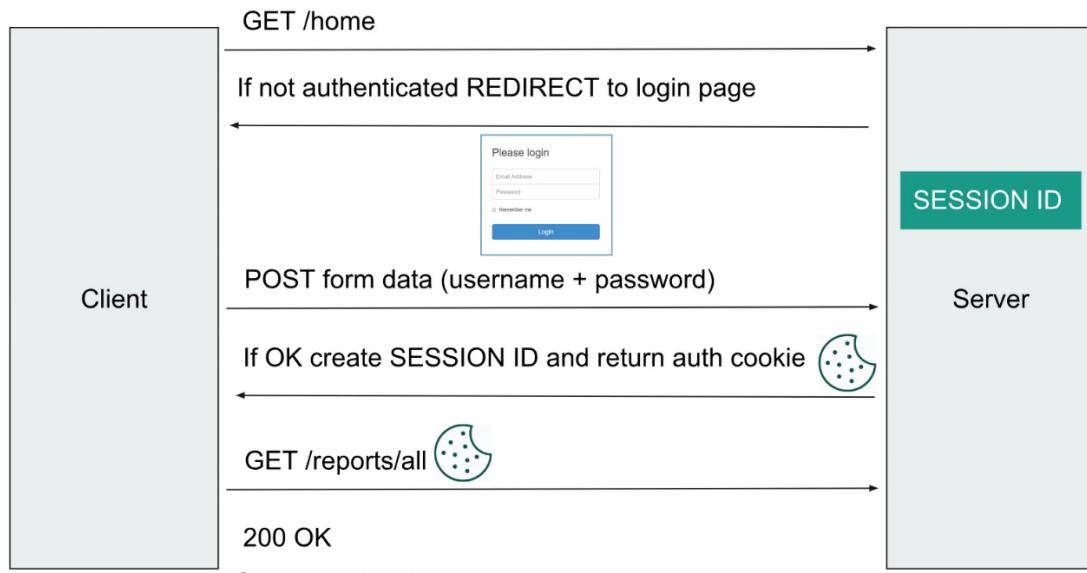
Spring Security verwaltet intern eine Filterkette, in der jeder der Filter in einer bestimmten Reihenfolge aufgerufen wird. Jeder Filter versucht, die Anforderung zu verarbeiten und Authentifizierungsinformationen daraus abzurufen.

Sobald die Anfrage den Authentifizierungsfilter passiert hat, werden die Anmeldeinformationen des Benutzers in einem Objekt gespeichert. Was nun tatsächlich für die Authentifizierung verantwortlich ist, ist der AuthenticationProvider (Interface mit der Methode `authenticate()`).

Deshalb findet der Authentifizierungsmanager den geeigneten AuthenticationProvider, indem er die Methode `supports()` von jedem Authentifizierungsanbieters aufruft. Die Methode `supports()` gibt einen booleschen Wert zurück. Wenn „true“ zurückgegeben wird, ruft der Authentifizierungsmanager seine Methode „`authenticate()`“ auf.

Nachdem die Anmeldeinformationen an den Authentifizierungsanbieter übergeben wurden, sucht dieser nach dem vorhandenen Benutzer im System von UserDetailsService. Es gibt eine UserDetails-Instanz zurück, die der Authentifizierungsanbieter überprüft und authentifiziert. Bei Erfolg wird das Authentifizierungsobjekt mit dem Prinzipal und den Autoritäten zurückgegeben, andernfalls wird eine AuthenticationException ausgelöst.

Um zu prüfen, ob ein User autorisiert ist, auf eine Ressource zugreifen zu können, wird beim Login Prozess die Session ID der Sitzung als Cookie gespeichert. Diese ID muss der User immer senden, um zu beweisen, dass er die Berechtigungen dazu hat.

**Quelle**

Spring Security wird verwendet, da es die einfachste Methode ist Benutzer in Spring Boot zu Authentifizieren.

- 6.1.7 Design der Testumgebung
- 6.1.8 Design der Ausführumgebung
- 6.2 Detailentwurf

6.3 Frontend

Unser Frontend ist prinzipiell in drei Teile aufgeteilt. Die Hauptseite für die Kunden, das Baukastensystem und das Dashboard für das Restaurant. Dabei teilen sich erstere zwei die gleichen Technologien, während für das Dashboard teilweise unterschiedliche Technologien verwendet werden. Die zugehörigen Mockups für das Frontend sind im Abschnitt *User-Interface-Design* zu finden. In folgendem Abschnitt sind die verwendeten Technologien sowie deren Einsatzbereiche zu sehen.

Die verwendeten Technologien im Frontend werden in folgender Tabelle (*Tabelle 10: Verwendete Frontend-Technologien*) dargestellt. Dabei ist der Bereich und die für den jeweiligen Bereich benützten Technologien zu sehen.

Bereich	Materialize	Materialize Stepper	Halfmoon	JQuery	js-cookie
Hauptseite	✓	✓		✓	✓
Baukastensystem	✓			✓	
Dashboard			✓	✓	

Tabelle 10: Verwendete Frontend-Technologien

6.3.1 Einbindung der Frontend-Technologien

Alle im Frontend verwendeten Technologien sind über CDN (Content Delivery Network) eingebunden. Dies ermöglicht einen leichten Versionswechsel jener Technologien. Zudem hält es die Struktur im Projekt schlanker.

6.3.2 Struktureller Aufbau der Dateien

Grundsätzlich befindet sich das Frontend in einer Spring Boot Applikation immer unter der „resources“ Ordner. Dort finden sich nun wieder zwei Ordner. Einmal „static“ und einmal ein „templates“ Ordner. Im „static“ Ordner sind die CSS- und JavaScript-Dateien zu finden. Im „templates“ Ordner sind die HTML Dateien zu finden. In den zwei zuvor genannten Ordner gibt es jeweils eine Unterteilung zwischen Hauptseite, Dashboard und Baukastensystem.

6.3.3 Verwendete Versionen

In folgender Tabelle (*Tabelle 11: Frontend-Technologien Versionen*) werden die jeweiligen Versionen der Frontend-Technologien dargestellt. Dabei sind die jeweiligen Bereiche und die jeweiligen Versionen der Technologien zu sehen.

Bereich	Materialize	Materialize Stepper	Halfmoon	JQuery	js-cookie
Hauptseite	V 1.0.0	V 3.1.0		V 3.5.1	V3.0.1
Baukastensystem	V 1.0.0			V 3.5.1	
Dashboard			V 1.1.1	V 3.5.1	

Tabelle 11: Frontend-Technologien Versionen

6.3.4 Interaktion zwischen den Seiten

Dem Projektteam war es ein Anliegen, die Interaktionen zwischen den Seiten möglichst strukturiert und intuitiv zu gestalten. In folgender Grafik ist zu sehen, wie man sich zwischen den Seiten bewegen kann. Dabei ist die Home-Page der zentrale Ausgangspunkt. Von hier aus startet jede Interaktion mit unserem Projekt. Eine Ausnahme bildet das Dashboard. Ist ein Restaurantbesitzer oder ein Koch in jenem Restaurant bereits eingeloggt so gelangt er auf das Dashboard. Ist er es nicht, gelangt er ebenfalls auf die Home-Page.

In jedem Kästchen befindet sich eine Seite, zu der man gelangen kann. Die Seiten, die ein Schloss-Symbol auf der Seite haben, sind nur für eingeloggte Nutzer sichtbar. Seiten mit einem Kochhut-Symbol sind für Angestellte beziehungsweise Köche zugänglich. Seiten mit dem Nutzer-Zahnrad-Symbol sind für Restaurantbesitzer zugänglich. Die blau markierten Seiten sind jene Seiten, die nur für Nutzer verfügbar sind. Sprich für Nutzer, die bei einem Restaurant essen bestellen wollen. Die gelb markierten Seiten sind jene Seiten die ausschließlich für das Restaurant zugänglich sind. Der blaue Bereich ist also der Geschäftsbereich der normalen Endnutzer unter der gelb markierte Bereich ist der Geschäftsbereich der Restaurants. Da der Login von beiden Geschäftsbereichen genutzt wird ist er hier mit einem Farbverlauf dargestellt.

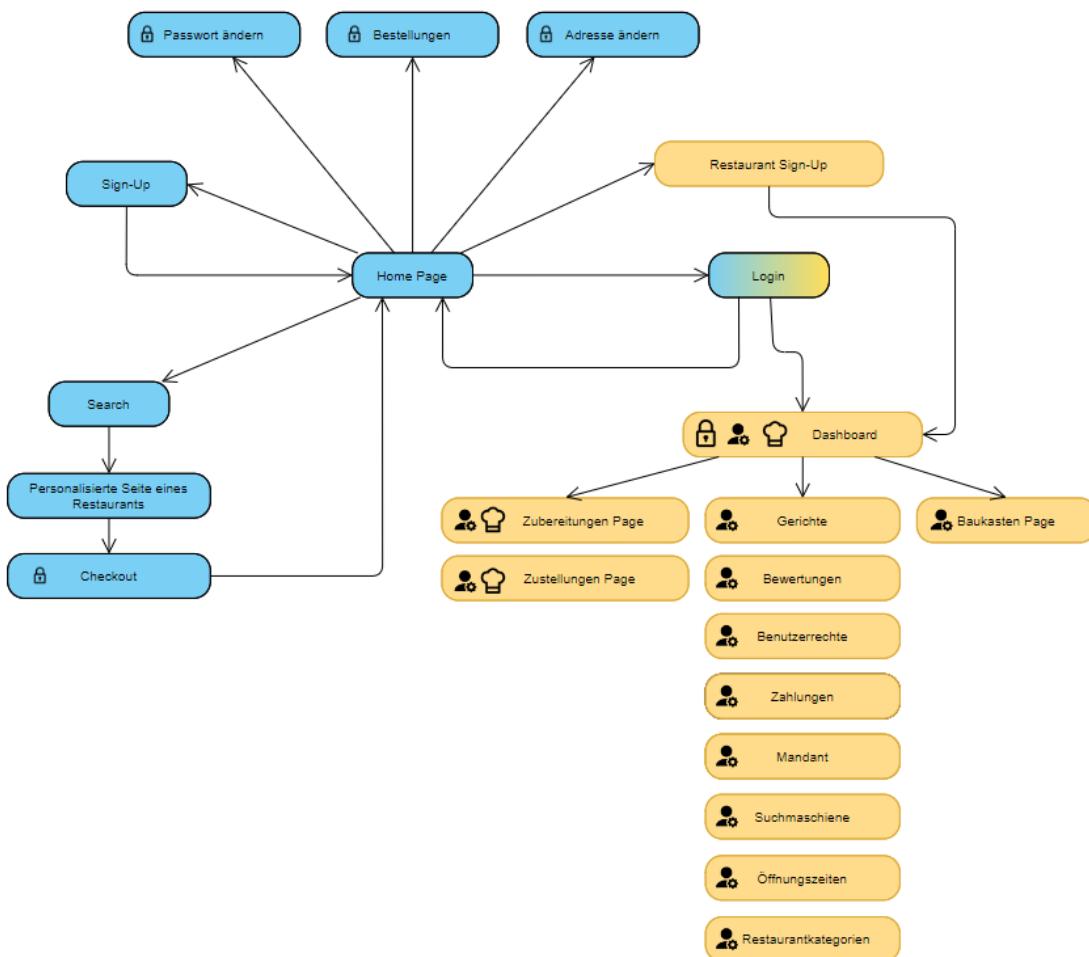


Abbildung 52: User-Interaktion zwischen den Seiten

Symbol	Bedeutung
Schloss	Diese Seite ist nur für eingeloggte Nutzer zugänglich
Kochhut	Diese Seite ist für Angestellte eines Restaurants zugänglich
Nutzer mit Zahnrad	Diese Seite ist für den Restaurantbesitzer zugänglich
blauer Bereich	Ist der Geschäftsbereich der Endkunden
gelber Bereich	Ist der Geschäftsbereich der Restaurants
gelb-blauer Bereich	Dieser Bereich wird von Restaurants und von Endkunden geteilt.

Abbildung 53 Legende zur User-Interaktionen Grafik

6.3.5 Theming

Damit eine Webseite heutzutage noch als modern gilt, benötigt sie ein helles sowie ein dunkles Theme. Die User können sich dann, ganz nach ihrer Präferenz eines der beiden Themes auswählen. Es stehen jeweils auf der Kundenseite als auch auf dem Dashboard zwei Themes zur Auswahl.

Auf der Kundenseite entschied man sich beim hellen Theme für eine Kombination zwischen modernen Grüntönen, einem dezenten Grauton sowie weiß. In folgender Grafik ist die Farbauswahl des hellen Themes zu sehen.

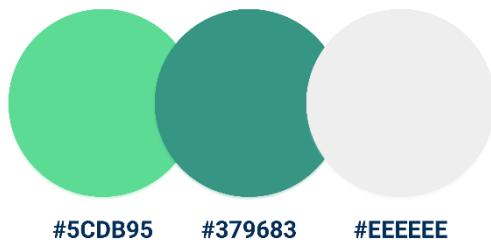


Abbildung 54: Farbpalette - Kundenseite - helles Theme



Abbildung 55: Kundenseite - Index Page - helles Theme

Beim dunklen Theme entschied sich das Projektteam für moderne Blautöne, da normale Schwarz- beziehungsweise Grautöne zu klassisch für dieses Projekt wären.

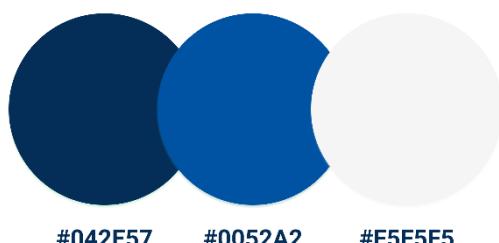


Abbildung 56: Farbpalette - Kundenseite - dunkles Theme

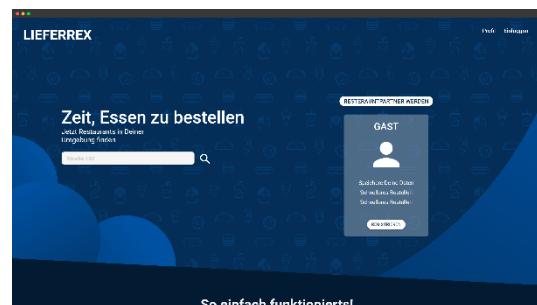


Abbildung 57: Kundenseite - Index Page - dunkles Theme

Eine wichtige Design-Regel ist die 60-30-10 Regel. Sie besagt das, wenn man drei Farben für eine Webseite als Palette zur Verfügung hat, man jene genau in dieser Proportion verwenden soll. Dies kann auch in den obigen Abbildungen gut sehen. Sowie im Dark als auch im Light Mode wurde diese Regel angewandt.

Beim Dashboard hat man sich für ein eigenes Theme entschieden. Auf dem Dashboard steht das dunkle Theme im Vordergrund. Jenes dreht sich ganz um dunkle sowie pinke Farben. In den folgenden zwei Abbildungen ist das dunkle Theme zu sehen.

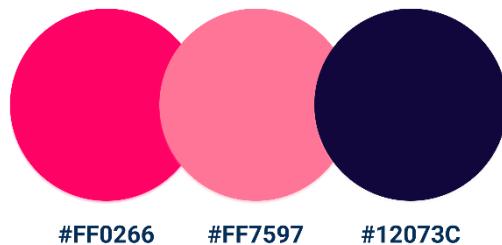


Abbildung 58: Farbpalette - Dashboard - dunkles Theme

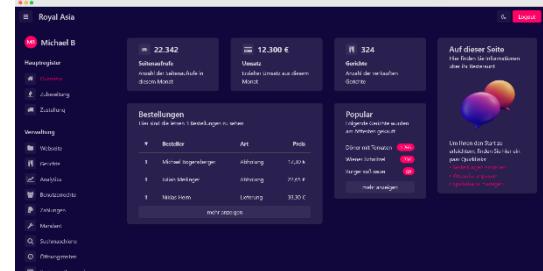


Abbildung 59: Dashboard - dunkles Theme

Hier gibt es zwischen dem dunklen und dem hellen Theme keinen allzu großen Unterschied. Der einzige Unterschied zwischen den Themes ist, dass im dunklen Theme als Hintergrundfarbe ein dunkles Lila verwendet wird und im hellen Theme ein leicht graues Weiß. In den folgenden zwei Abbildungen ist das helle Theme zu sehen.

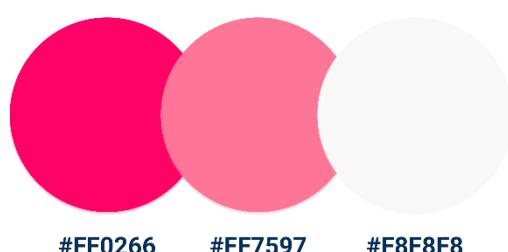


Abbildung 60: Farbpalette - Dashboard - helles Theme

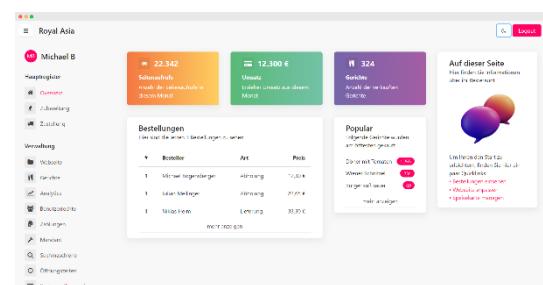


Abbildung 61: Dashboard - helles Theme

Im hellen Theme werden zudem auf der Overview Page die Infokarten mit jeweils drei verschiedenen Farbverläufen dargestellt. In der folgenden Abbildung sieht man die drei verschiedenen Farbverläufe mit den jeweiligen Hex-Codes.



Abbildung 62: Farbverläufe - Dashboard - Infokarten

6.4 Backend

Das Backend ist die Datenzugriffsebene für die http-Anfragen des Frontend. Es stellt somit das Rückgrat unserer Applikation dar. Über die Benutzeroberfläche des Frontend soll es somit möglich sein Daten zu bearbeiten, erstellen und zu löschen. Diese Daten sind in einer Datenbank gespeichert. Bei einem Aufruf der Webseite wird eine Anfrage an das Webinterface² des Webserver gesendet. Eine Spring-Boot Business-Anwendung nimmt diese Anfrage an und verarbeitet sie weiter. Je nach Art des Request werden unterschiedliche Aktionen in der Applikation ausgeführt. Die Software beinhaltet die Logik, um die Daten der Datenbank zu manipulieren. Somit kann eine Spezifische Antwort dem Client zurückgesendet werden. Ein persistenter Datenfluss zwischen dem Frontend und der Datenbank ist dadurch gewährleistet.

Wie schon zuvor erklärt wird das MVC-Pattern verwendet. In Spring Boot werden dabei die Model und Controller Komponenten implementiert. Zusätzlich wird aber noch ein Service-Layer eingebaut, der sich zwischen Model und Controller befindet und die Businesslogik beinhaltet. In Spring Boot spricht man daher von folgenden Layer.

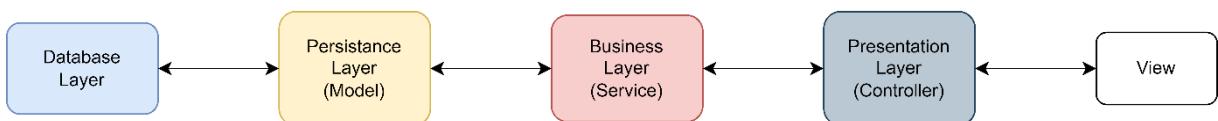


Abbildung 63: MVC-Pattern Spirng Boot

Der Database Layer beinhaltet die konkrete Datenbank.

Im Persistance Layer befinden sich alle Model-Klassen, die die Entitäten der Datenbank widerspiegeln. Diese Klassen stellen ein POJO³ dar und können wie gewohnt verwendet werden. Sie beinhalten auch Validierungs Annotationen, damit nur Valide Daten gespeichert werden. Diese Klassen werden für CRUD-Operationen verwendet.

Der Business Layer implementiert die Service Klassen welche für die Validierung, Business Logik und die Autorisierung zuständig ist.

Die Controller-Klassen befinden sich im Presentation Layer. Dieser Stellt die Schnittstelle zu den Views dar und konvertiert die JSON-Requests zu Java verständlichen Code.

Die Views stellt die Client Seite dar.

6.4.1 Spring Boot

Folgende Grafik Abbildung 64: Konkrete Spring Boot Architektur stellt nun die Konkrete Implementierung dar.

² http Schnittstelle

³ Plain Old Java Object

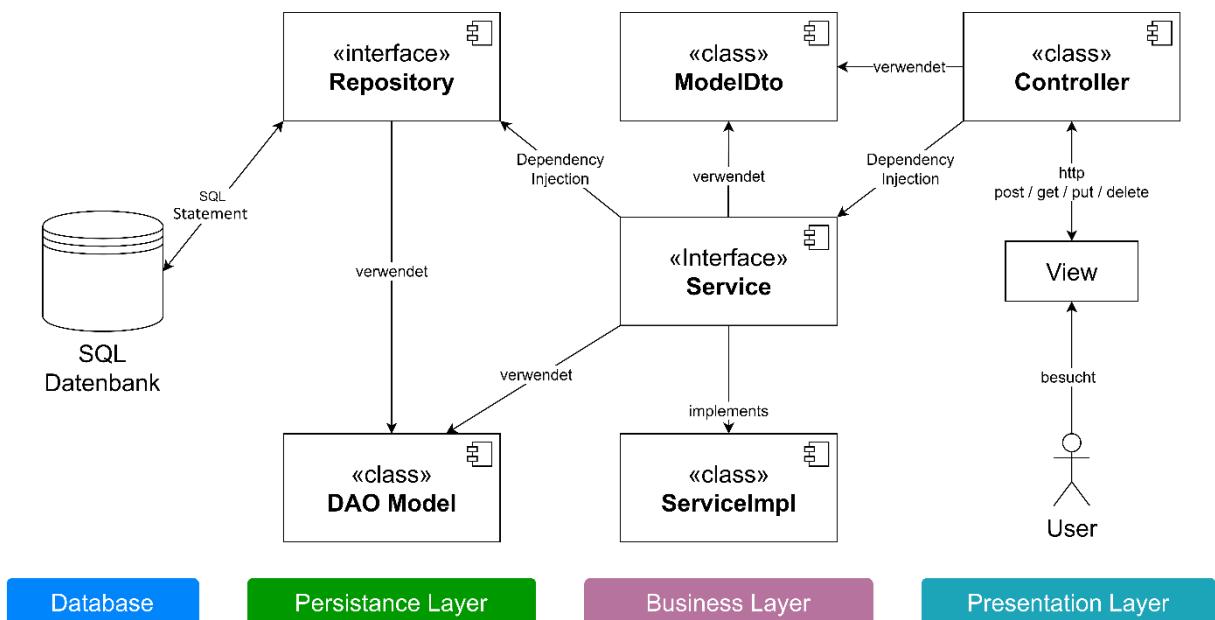


Abbildung 64: Konkrete Spring Boot Architektur

In Abbildung 64: Konkrete Spring Boot Architektur sind wieder die vier Layer von Spring Boot zu sehen. Es Zeigt, welche Komponenten für eine Anfrage. Um das Modell zu vereinfachen, wird nur jede Komponente einmal gezeigt. In der Realität gibt es eine Vielzahl an Klassen, die die Logik widerspiegeln.

Herzstück unserer Anwendung ist das Service Interface. Dieses Interface beinhaltet alle Methoden, die notwendig sind. Damit der Controller Zugriff auf diese Klasse hat wird mittels der `@Autowired` Annotation von Spring Boot eine Dependency Injection durchgeführt. Somit entsteht eine geringe Koppelung zwischen den Klassen. Zwischen dem Controller und dem Service werden i.d.R. DTO's⁴ Objekte übergeben. Dies vereinfacht die Programmierung und gewissen stellen und bietet der Schnittstelle zusätzliche Sicherheit.

Das Repository Interface erweitert sich zum Spring Boot proprietären JPA-Repository. Dieses JPA-Repository beinhaltet die wichtigsten CRUD-Operationen, die man für ein DAO-Model verwenden kann. Im Repository werden Spezifische Methoden angegeben, die nicht Standardmäßig im JPA-Repository vorhanden sind. Spring Boot leistet dann im Hintergrund die ganze Arbeit löst die Methode in ein valides SQL-Statement auf. Mittels Depandency Injection wird im Service layer die passende Funktion aus dem Repository aufgerufen, und auf die Datenbank zugegriffen werden. Dabei werden die DAO-Model-Klassen verwendet.

Spring Boot wird verwendet, da es eine sehr umfangreiche und komplexe Backend-Applikation realisieren kann. Desweiterem haben alle Projektanten gute Kenntnisse in der Java Entwicklung. Während dem Programmierunterricht haben wurden ebenfalls schon einfache Projekte umgesetzt, was die Auswahl des Backendsystems nur noch einfacher gemacht hat.

6.4.2 REST

REST ist eine Schnittstelle, womit der Datentausch im Internet zwischen zwei Geräte ermöglicht wird. REST verwendet hauptsächlich die Protokolle HTTP und HTTPS. Rest ist auch so gut mit jeder Firewall kompatibel. Mit REST kommen auch die DNS-Versionierungen sowie die URL-Versionierungen dazu ein Beispiel ist die DNS <http://v1.api.foo.com/customer/1234>. Mit REST können viele Methoden wie

⁴ Data Transfer Object

GET, POST, PUT, DELETE, HEAD genutzt werden. In der folgenden Tabelle sind die Methoden dargestellt.

Methode	Beschreibung
GET	Bei einer GET Anfrage können die Daten vom Server mittels GET Methode angefordert werden.
PUT	Mit PUT können die Ressourcen, die schon vorhanden sind abgeändert werden, falls keine Ressource vorhanden ist wird eine neue Ressource angelegt.
DELETE	Mittels DELETE kann die angegebene Ressource gelöscht werden.
HEAD	Mithilfe HEAD können die Metadaten von der Ressource angefordert werden.

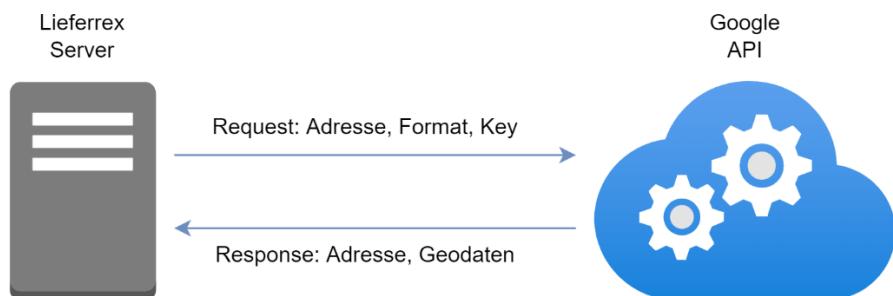
Tabelle 12: REST Methoden

Der Vorteil bei REST ist, dass REST zustandlos ist, das heißt, dass jede REST-Nachricht alle Information beinhaltet die für den Server sowie den Client notwendig sind. In diesem Verfahren werden keine Zustandsinformationen zwischen den zwei Nachrichten gespeichert. Dies wird auch als zustandloses Protokoll bezeichnet. Durch dieses zustand können Webservices besser skaliert werden.

1.1.1 Google Maps API

Die Google Maps API bei der Abfrage von Wohnadressen eine zentrale Rolle. Bei der Restaurantwahl und der Registrierung wird die Schnittstelle verwendet, um zu überprüfen, ob die Adresse valide ist. Aus diesem Grund ist es auch notwendig bei der Registrierung die Wohnadressen im richtigen Format abzuspeichern.

Um http Anfragen, an die API senden zu können, wird ein API-Key benötigt. Dieser Key muss bei jeder Anfrage mitgesendet werden, um den Request zu autorisieren. Das Responseformat muss beim Request mitgesendet werden. Pro 1000 Abfragen, ist ein fixer Geldbetrag zu bezahlen.



1.1.1.1 Geocoding API

Mittels der Geocoding API kann man Adressen in deren Geodaten – Längengrad und Breitengrad – umwandeln. Somit ist es möglich auf dem Frontend die genaue Position des Restaurants auf einer Karte anzeigen zu lassen.

Ein Geocoding Request benötigt dabei einen Teil der Adresse, das Rückgabeformat – JSON oder XML – und den API-Key. Optional kann man auch die Sprache, Region, etc. mitsenden. Als Response liefert die API die vollständige Adresse samt Geodaten zurück. Diese Funktionalität wird bei der Speicherung der Geodaten, des Restaurants verwendet.

Als Gegenstück existiert dabei der Reverse Geocoding Request, der aus Länge- und Breitengraden eine Adresse zurückliefert. Diese wird auch verwendet, um die PLZ einer Adresse zu holen, wenn diese nicht vorhanden vorhanden ist.

1.1.1.2 Places API

Die Places API nimmt als Request wider eine Adresse, das Format und den Key entgegen. Als Response liefert sie dieses Mal alle ähnlichen Adressen passend zur Eingabe. Die API versucht somit zu erraten, nach welcher Adresse gesucht wird. Speziell bei der Registrierung, wird diese Schnittstelle verwendet, um nur Adressen zu speichern, die Google Maps kennt, und um die Eingabe zu erleichtern. Dies wird realisiert, indem die Places Autocomplete Webservice angesprochen wird. Im Frontend kann somit eine intuitive Eingabemaske erstellt werden.

1.1.2 PayPal API

Die PayPal API ist für alle Geld Transaktionen der Applikation verantwortlich. Sie ermöglicht den sicheren und zuverlässigen Geldverkehr zwischen zwei Parteien.

Um auf die PayPal API zugreifen zu können, muss man sich bei PayPal Developer Registrieren und eine REST API-Applikation erstellen. Momentan ist diese nur eine Sandboxanwendung, d.h., dass die Transaktionen nicht mit Realen Geld stattfindet und nur an Testaccounts gesendet wird. Im Zuge dessen, wurden vier Kundenaccounts und fünf Businessaccounts für die Restaurants erstellt.

<input type="checkbox"/> Account name	Type	Country	Date created
<input type="checkbox"/> royal@business.example.com	Business	AT	20 Jun 2022
<input type="checkbox"/> kunde04@example.com	Personal	AT	19 Jun 2022
<input type="checkbox"/> kunde03@example.com	Personal	AT	19 Jun 2022
<input type="checkbox"/> kunde02@example.com	Personal	AT	19 Jun 2022
<input type="checkbox"/> kunde01@example.com	Personal	AT	19 Jun 2022
<input type="checkbox"/> mandant4@business.example.com	Business	AT	19 Jun 2022
<input type="checkbox"/> mandant3@business.example.com	Business	AT	19 Jun 2022
<input type="checkbox"/> mandant2@business.example.com	Business	AT	19 Jun 2022
<input type="checkbox"/> mandant1@business.example.com	Business	AT	19 Jun 2022

Diese Testaccounts besitzen alle das Passwort „Passwort1!“ und haben einen Kontostand von 100€. Der Account der Applikation besitzt folgende E-Mail und Client ID.

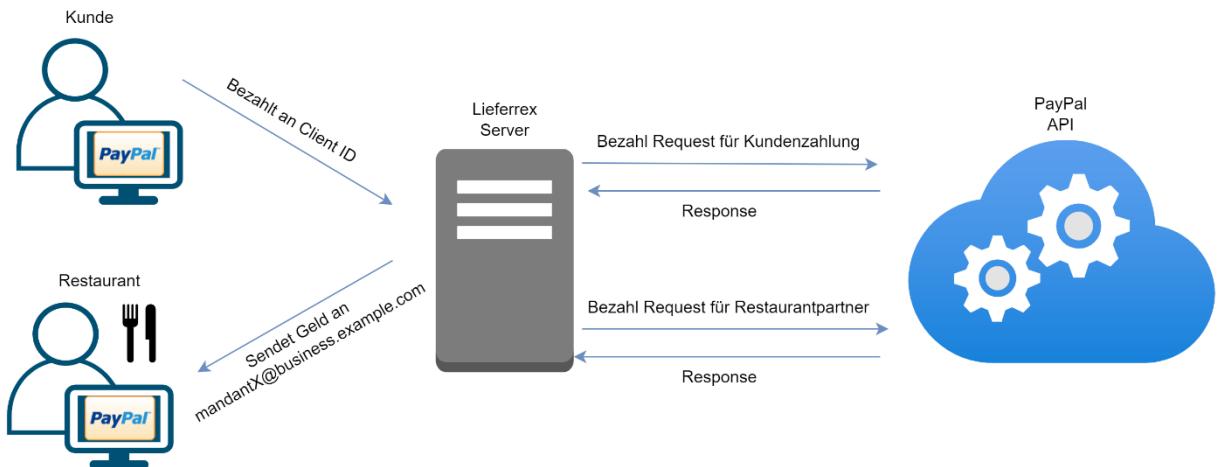
Sandbox account	lieferrex@business.example.com
Client ID	AYppyNGKPw2EOEhUsmclq4jbwcGlvt3j6XFB0fbqX4bQDS8DD52VKemiG77l8DqDl5tSKFYZaZbGsRND
Secret	Show

Mittels der Client ID und dem Secret, ist es möglich API Requests gegen die API zu senden.

Für unsere Plattform wäre die Multiparty Payments Schnittstelle der PayPal API am besten geeignet gewesen. Sie ermöglicht es eine E-Commerce Plattform zu erstellen, indem die Betreiber der Applikation nur als Zwischenhändler fungiert. Das Projektteam hat sich jedoch gegen diese

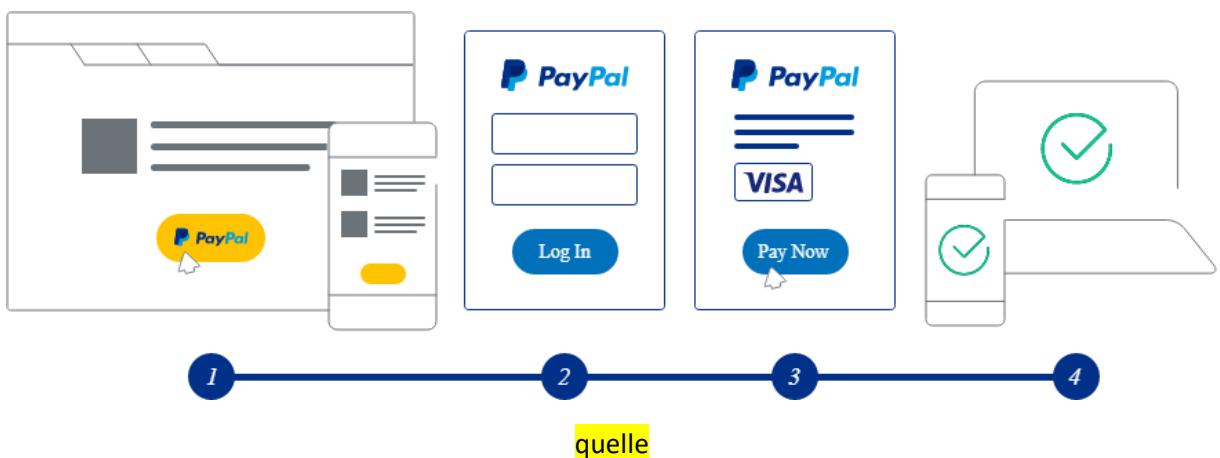
Implementierung entschieden, da es die Zeitlichen Rahmenbedingungen überschritten hätte. Gründe dafür war die fast nicht vorhandene Dokumentation, und die Ungewissheit, wie viel Mehraufwand diese Funktionalität bringen würde.

Aus diesem Grund besteht der Transaktionsvorgang aus zwei Phasen. Als erstes, bezahlt der Kunde wie gewohnt, mit seinem PayPal Konto auf der Bezahlungsseite. Dieser Geldbetrag, wird an das PayPal Konto von Lieferrex gesendet. In der zweiten Phase, sendet wird derselbe Geldbetrag vom Lieferrex PayPal Account an die E-Mailadresse des Restaurants geschickt, wo der Kunde bezahlt hat.



1.1.2.1 Kundenbezahlung

Damit der Kunde einen sicheren und einfachen Bezahlungsvorgang haben, wird der Zahlungsdienst „PayPal Checkout“ verwendet. Sobald der Kunde auf der Liferrex Bezahlungsseite den Bezahlbutton geklickt hat, wird er auf Checkout Seite von PayPal weitergeleitet. Dort muss er sich mit seinem eigenen PayPal Account anmelden – in diesem Fall mit einem der Kunden Sandbox Accounts. Danach kann er eine beliebige Zahlungsmethode auswählen und die Bestellung abschließen. Falls die Bestellung geklappt hat, wird der Kunde samt Bestelldaten auf eine Erfolgsseite weitergeleitet. Erst danach wird die Transaktion ausgeführt und die Geldbeträge an das PayPal Konto von Lieferrex gesendet.



1.1.2.2 Mandantenbezahlung

Sobald der Kunde die Rechnung der Bestellung an uns überwiesen hat, wird der Geldbetrag an den Mandanten weitergeleitet. Dies wurde mit der PayPal Payout API realisiert. Diese Schnittstelle ermöglicht es einen oder mehrere Geldbeträge, an eine E-Mail-Adresse zu senden, vorausgesetzt, der Account ist mit dieser E-Mail-Adresse bei PayPal registriert. In diesem Fall, muss somit die E-Mail-Adresse des Registrierten Restaurantpartners dieselbe sein, wie bei seinem PayPal Account.

Für den API-Aufruf, wird nur die E-Mail-Adresse und der Geldbetrag benötigt. Problematisch bei dieser Methode ist, dass bei jeder Transaktion eine Gebühr von 2% anfällig ist. Speziell bei vielen eingehenden Bestellungen, müsste man dieses Konzept also überdenken. Die Multiparty Payments Schnittstelle von PayPal könnte dabei die bessere Option sein, die konnte jedoch leider nicht wegen den oben genannten Gründen implementiert werden.

6.4.3 Baukastensystem

Der Baukasten stellt eine zentrale Komponente des Projektes dar. Über den Baukasten können Restaurant ihre eigenen Seiten erstellen und individuell anpassen. Hierfür müssen viele verschiedene Daten gespeichert werden (Abbildung 65: ER-Modell Baukasten).

Als erstes wird ein Layout benötigt. Das Layout gibt an, welche Vorlage verwendet wird. Diese Vorlage bestimmt die Struktur der Seite (Anzahl der verschiedenen Module, Positionierung dieser Module, Design des Kopfbereiches). Jedes solches Layout besteht aus vielen Positionen. Jede Position enthält ein Modul.

Diese Module gibt es in vielen verschiedenen Formen. Jedes Fragment ist einem Fragment-Typ zugeordnet. Der Fragment-Typ entscheidet, in welcher Form das Modul gespeichert wird und welche Informationen darin enthalten sind.

Im Rahmen dieses Projektes müssen speziell drei Fragment-Typen in der Datenbank gespeichert werden: Fragment-Text, Fragment-Image und Fragment-Header. Fragment-Text ist eines der einfachsten Module. In diesem kann das Restaurant einen beliebigen Titel und einen Text definieren. Das Fragment-Image Modul tauscht den Text mit einem Bild aus. Jedes Layout besitzt ein Fragment-Header Modul, das immer die gleiche Datenstruktur hat, diese aber anders darstellt. Der Header beinhaltet einen Titel, Text und ein Bild.

Des Weiteren kann ein Mandant bis zu zwei zusätzliche Seiten im Baukasten erstellen, eine Bildergalerie- und eine „Über uns“-Seite. In der Bildergalerie sind ein Titel und fünf Bilder enthalten. Die „Über uns“-Seite kann vom Restaurant mit Titel, zwei Texten und zwei Bildern befüllt werden.

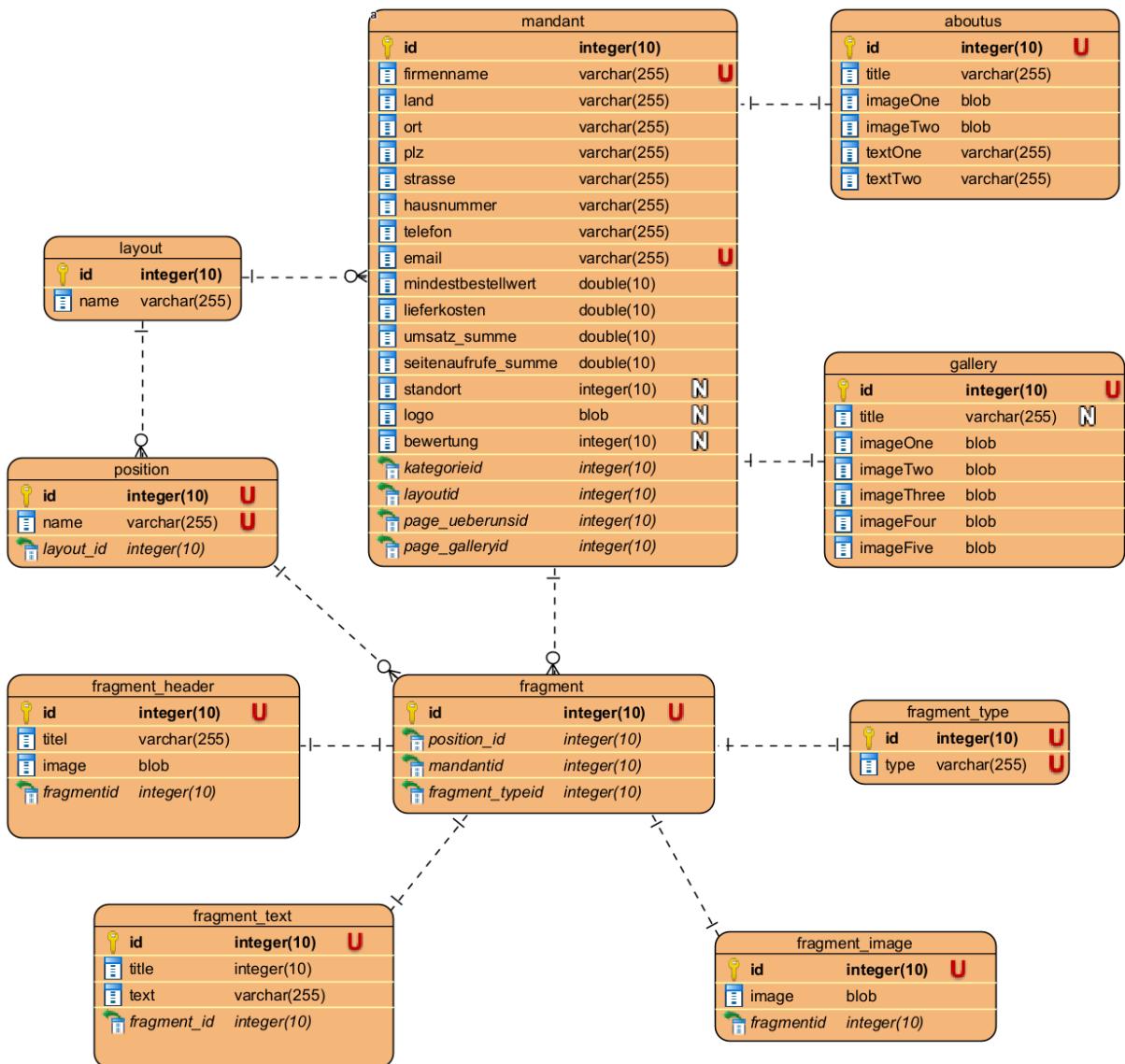


Abbildung 65: ER-Modell Baukasten

7 Implementierung

In folgendem Abschnitt ist die jeweilige Implementierung zu sehen.

7.1 Frontend

In folgendem Abschnitt ist die Implementierung des Frontends beschrieben. Dies ist hier zwischen Dashboard und Kundenseite unterteilt. Da nicht nur Materialize sondern auch Halfmoon verwenden gibt es zwei verschiedene Herangehensweisen für den Dark Mode. Deshalb wird in beiden Abschnitten der Dark Mode thematisiert.

7.1.1 Kundenseite

Da die Kundenseite entscheidend für den Erfolg von Lieferrex ist, hat das Projektteam Wert darauf gelegt, dass die Index-Seite der Kundenseite besonders ansprechend aussieht. Die Index-Seite ist die am aufwändigsten gestaltete Seite des ganzen Projektes. Sie ist so gestaltet, dass wenn man auf die Seite navigiert, einem ein großer Bereich präsentiert wird. Hier finden sich Links zum Einloggen, registrieren aber auch zum Suchen nach den Restaurants. Die übereinander dargestellten Vector-Illustrationen sowie der schiefe Schnitt in den nächsten Bereich, machten die Gestaltung der Seite so aufwändig. In der linken Abbildung sieht man die jeweiligen Bereiche der Index Seite. In der rechten Abbildung sieht man die fertige Seite zum Vergleich. In der mit eins markierten Box befindet sich der wichtigste Content. Darin befinden sich auch die Boxen zwei und drei. Die Seite ist so gestaltet, dass die Box eins als Hintergrund dient. In Box zwei und drei befinden sich die Vector-Illustrationen. Diese zwei Boxen befinden sich über Box eins. Die Box vier dient als Übergang in den nächsten Bereich. Hier ist auffällig, dass es einen schiefen Schnitt gibt. Dieser wurde mithilfe clip-path Funktion von CSS erzielt.

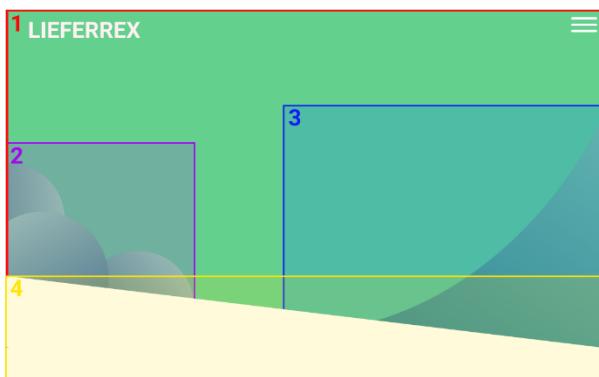


Abbildung 67: Kundenseite - Index Page - Aufbau

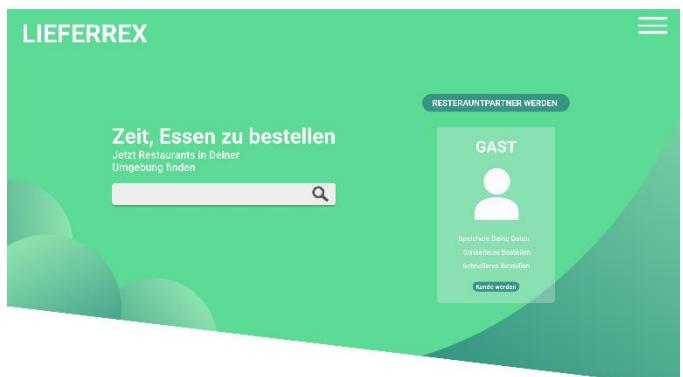


Abbildung 66: Kundenseite - Index Page

Die HTML Struktur zu der Startseite ist in folgender Abbildung zu sehen. Der „main_wrapper“ ist die rot gekennzeichnete Box. Darin befinden sich nun weitere vier weitere Komponenten. Zunächst gibt es im Main-Wrapper noch eine div für die Navbar. Danach folgen die Box zwei (Lila) sowie die Box drei (Blau). Darin befinden sich die Hintergrundgrafiken. Unterhalb des Main-Wrappers in Box 4 (Gelb) befindet sich der weitere Content.

```

<div class="main_wrapper">
  <div class="row main-row"></div>
  <div class="wrapper-bg">
    <div class="main_right"></div>
    <div class="main_left"></div>
  </div>
</div>
<div class="main-content"></div>

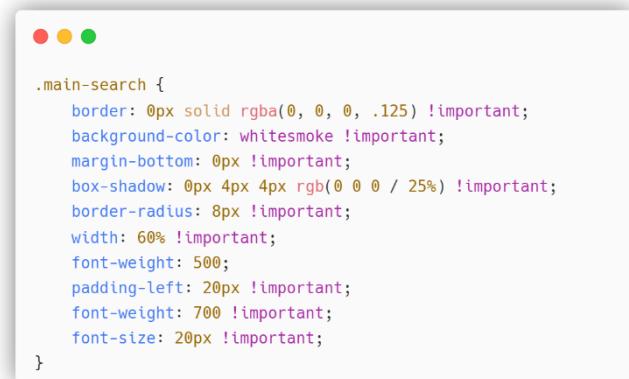
```

Abbildung 68: Hauptseite HTML-Struktur

7.1.2 Kundenseite – Dark Mode

Auf der Hauptseite wurde der Dark Mode von Grund auf implementiert. Es gibt auf der Hauptseite eine CSS-Datei, die den grundlegenden Style der Seite definiert. Je nachdem ob der Dark Mode oder der Light Mode aktiviert ist, wird zusätzlich eine weitere CSS-Datei geladen. Hat der User also den Dark Mode aktiviert wird die style.css Datei und zusätzlich die dark.css Datei geladen. Hat der User den Light Mode aktiviert wird die style.css Datei sowie die light.css Datei geladen. In den Dateien für den jeweiligen Modus finden sich die Anpassungen für das aktuell ausgewählte Theme wieder.

Ein gutes Beispiel dafür bildet die Klasse main-search. Diese styled das Inputfeld auf der Hauptseite mit dem man nach einem Standort suchen kann. In folgender Abbildung ist nun der Style der style.css Datei zu sehen. Sie legt die wichtigsten Attribute wie zum Beispiel die Breite oder die Schriftgröße fest.



```

.main-search {
    border: 0px solid rgba(0, 0, 0, .125) !important;
    background-color: whitesmoke !important;
    margin-bottom: 0px !important;
    box-shadow: 0px 4px 4px rgb(0 0 0 / 25%) !important;
    border-radius: 8px !important;
    width: 60% !important;
    font-weight: 500;
    padding-left: 20px !important;
    font-weight: 700 !important;
    font-size: 20px !important;
}

```

Abbildung 69: main-search Klasse in style.css Datei

In der dark.css Datei findet sich nun die gleiche Klasse wieder. Darin ist nun die Schriftfarbe definiert. Diese ist im Dark Mode ein dunkles Blau. Ist der Dark Mode also aktiviert so wird folgender Style übernommen.



```

.main-search {
    color: var(--dark-main);
}

```

Abbildung 71: main-search Klasse in dark.css Datei



Abbildung 70: main-search style auf der Hauptseite (Dark Mode)

Ist der Light Mode aktiviert so wird die light.css Datei geladen und folgender Style übernommen. Hier wird für die Schriftfarbe ein helles Grün verwendet.



```

.main-search {
    color: var(--light-main);
}

```

Abbildung 72: main-search Klasse in light.css Datei



Abbildung 73: main-search style auf der Hauptseite (Light Mode)

```
● ○ ●
if (Cookies.get("dark-mode") == null) {
    toDark();
} else if (Cookies.get("dark-mode") == "on") {
    $("#switch-dm-js").attr("Checked", "Checked");
    toDark();
} else if (Cookies.get("dark-mode") == "off") {
    $("#switch-dm-js").removeAttr("Checked");
    toLight();
}
```

Abbildung 74: toggleSwitch Funktion / JavaScript

Ruft man nun eine Seite der Hauptseite auf so wird folgende Funktion ausgeführt. Zunächst wird gecheckt, ob der Cookie existiert. Tut er das nicht so wird automatisch der Dark Mode verwendet und der Switch wird gecheckt. Ist der Cookie auf den Dark Mode gesetzt so wird der Dark Mode aktiviert und der Switch ebenfalls gecheckt. Ist der Cookie auf „off“ gesetzt so wird der Light Mode aktiviert und der Switch wird auf „off“ beziehungsweise uncheckt gesetzt.

Wie in der obigen Abbildung zu sehen ist wird entweder die Funktion `toDark()` oder die Funktion `toLight()` verwendet. Beide Funktionen erstellen zunächst eine Variable, die einen String enthält. Darin wird ein Link-Tag mit dem Pfad zur CSS-Datei (`dark.css` oder `light.css`) erstellt. Danach wird der link an den Body angehängt. Da der Body ausgeblendet wird so lange nicht ein Modus gewählt wurde, wird nun der Body noch auf sichtbar gestellt. Ansonsten würde man bei jedem Wechsel ein unschönes Flackern sehen.

```
● ○ ●
function toDark() {
    var stylesheet = $("<link>", {
        rel: "stylesheet",
        type: "text/css",
        href: "/css/main/dark.css",
    });
    stylesheet.appendTo("body");
    $(document).ready(function () {
        $("body").css({ display: "block" });
    });
}
```

Abbildung 76: ToDark Funktion

Um den Modus zu wechseln, gibt es einen Switch am Ende der Hauptseite. Dort kann man den Dark Mode entweder aktivieren oder deaktivieren. Wird der Switch angeklickt so wird folgende Funktion ausgeführt. Ist der Switch gecheckt so wird der Cookie „dark-mode“ auf „on“ gesetzt. Andernfalls wird der Cookie auf „off“ gesetzt. In beiden Fällen wird danach die Seite neu geladen. Dies ist jedoch kaum merkbar, da JavaScript dies sehr gut überdeckt.

```
● ○ ●
if (Cookies.get("dark-mode") == null) {
    toDark();
} else if (Cookies.get("dark-mode") == "on") {
    $("#switch-dm-js").attr("Checked", "Checked");
    toDark();
} else if (Cookies.get("dark-mode") == "off") {
    $("#switch-dm-js").removeAttr("Checked");
    toLight();
}
```

Abbildung 75: Cookie check für Dark Mode

```
● ○ ●
function toLight() {
    var stylesheet = $("<link>", {
        rel: "stylesheet",
        type: "text/css",
        href: "/css/main/light.css",
    });
    stylesheet.appendTo("body");
    $(document).ready(function () {
        $("body").css({ display: "block" });
    });
}
```

Abbildung 77: toLight Funktion

7.1.3 Dashboard

Das Dashboard folgt grundsätzlich einem bestimmten Aufbau. Es sei denn, die Seiten eignen nicht dazu. Dazu gehören unter anderem die Öffnungszeiten-Seite die sowie die Mandanten-Seite. Abgesehen davon hat jede Seite im Dashboard wie man in folgendem Bild sehen kann ein bestimmtes Muster. Links auf der Seite ist natürlich die Navigationsleiste zu sehen. Diese findet man ausnahmslos auf jeder Seite. Des Weiteren ist in der Navigationsleiste immer der aktuell eingeloggte User zu sehen. Ganz oben auf der Seite (mit einer eins markiert) finden sie meist Statistiken zu der entsprechenden Seite. Auf der Benutzerrechte Seite finden sie zum Beispiel Informationen darüber wie viele Angestellte und wie viele Administratoren das Restaurant hat. In der Mitte (mit einer zwei markiert) findet man den Hauptinhalt. In diesem Fall sind hier die Zugänge in einer Tabelle zu sehen. Auf der rechten Seite (mit einer drei markiert) findet man Aktionen, die ausgeführt werden können. In diesem Fall kann man hier einen neuen Benutzer anlegen.

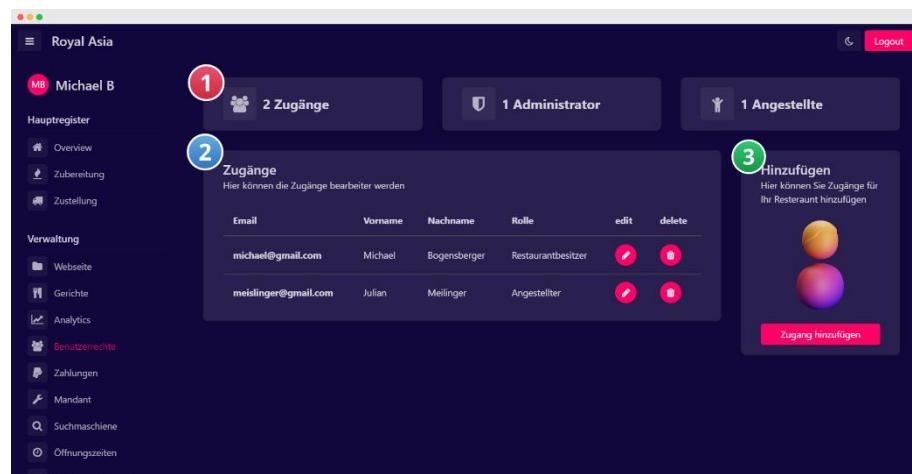


Abbildung 78: Dashboard - Benutzerrechte - Aufbau

7.1.4 Dashboard – Dark Mode

Da Halfmoon bereits einen eingebauten Dark Mode hat wurde hier nur das Theme für die jeweiligen Modi angepasst. Dazu wurden zunächst die Farben der jeweiligen Modi als CSS-Variablen deklariert und gesetzt (*Abbildung 79: Dashboard*). Der Dark Mode bei Halfmoon wird gesetzt, indem vorgefertigte Variablen auf den gewünschten Wert gesetzt werden. So wurde zum Beispiel die Primäre Farbe auf die zuvor definierte Variable gesetzt (*Abbildung 80: Dashboard Halfmoon Variablen*). So werden zum Beispiel alle Buttons nun mit der ausgewählten Farbe dargestellt.

```
--pink-color-main: #ff0266;
--pink-color-light: #ff0256;
--pink-color-sec: #ff7597;
--blue-color-dm-main: #021b32;
--blue-color-dm-sec: #fffff0a;
```

Abbildung 79: Dashboard CSS-Variablen

```
--primary-color: var(--pink-color-main);
--primary-color-light: var(--pink-color-light);
--primary-box-shadow-color: var(--pink-color-sec);
```

Abbildung 80: Dashboard Halfmoon Variablen

Will man jedoch Styles nur für einen speziellen Modus definieren, so kann man in CSS vor der eigentlichen Klasse die Klasse „dark-mode“ oder „light-mode“ schreiben. Dadurch wird dann der definierte Style nur für den ausgewählten Modus angewandt.



```
.dark-mode .card {
    background: rgb(255 255 255 / 10%) !important;
    backdrop-filter: blur( 16.5px );
    -webkit-backdrop-filter: blur( 16.5px );
    border-radius: 10px !important;
}
```

Abbildung 81: Halfmoon eigener Style

Halfmoon gibt praktischerweise direkt Komponenten beziehungsweise die Grundstruktur für das Dashboard vor. Dies ist unter anderem einer der Gründe, weshalb Halfmoon für das Dashboard gewählt wurde. In Abbildung 82: Halfmoon Grundstruktur ist jene Grundstruktur zu sehen. Im Body werden mithilfe von Data-Tags ein paar Grundeinstellungen getroffen. Es wird zum Beispiel beim Laden der Seite ein präferierter Farbmodus (in unserem Fall der Dark Mode) gewählt. Im Body befindet sich nun der Page-Wrapper. Darin findet sich zunächst eine div für die Alerts. Jene div bleibt immer leer. Danach findet sich die div für die Navbar und die Sidebar. Zu guter Letzt befindet sich im Wrapper der Content-Wrapper. Hier kommt der eigentliche Content hinein. Dieser Aufbau ist für alle Seiten im Dashboard gleich.



```
<body class="with-custom-webkit-scrollbars with-custom-css-scrollbars" data-dm-shortcut-
enabled="true" data-sidebar-shortcut-enabled="true" data-set-preferred-mode-onload="true">

<!-- Page wrapper start -->
<div class="page-wrapper with-navbar with-sidebar">

    <!-- Sticky alerts (toasts), empty container -->
    <div class="sticky-alerts"></div>

    <!-- Navbar start -->
    <nav class="navbar">
    </nav>
    <!-- Navbar end -->

    <!-- Sidebar start -->
    <div class="sidebar">
    </div>
    <!-- Sidebar end -->

    <!-- Content wrapper start -->
    <div class="content-wrapper">
        <!-- Add your page's main content here -->
    </div>
    <!-- Content wrapper end -->

</div>
<!-- Page wrapper end -->

</body>
```

Abbildung 82: Halfmoon Grundstruktur

7.1.5 Baukasten

Da das Baukastensystem was das Frontend betrifft zur Hauptseite gehört wurde das Baukastensystem mit Materialize erstellt. Dabei gibt es im Baukastensystem zwei verschiedene Ansichten. Einmal jene für den Restaurantbesitzer und einmal jene für den Kunden. Obwohl es zwei unterschiedliche Ansichten sind, handelt es sich hier dennoch um eine Datei. Dem Restaurantbesitzer wird die Ansicht in Abbildung 83: Baukastensystem - Ansicht Restaurantbesitzer angezeigt und dem Kunden wird die Ansicht in Abbildung 84: Baukastensystem - Ansicht Kunde angezeigt. Es werden also je nachdem welche Art von Nutzer eingeloggt ist verschiedene CSS-Attribute geändert. So wird beispielsweise, wenn der Kunde eingeloggt ist die Sidebar ausgeblendet und der Content rechts davon auf die gesamte Breite der zur Verfügung stehenden Fläche ausgeweitet.

Die folgende Abbildung zeigt die Ansicht des Restaurantbesitzers. Hier kann er seine eigene Webseite anpassen. In der Sidebar links ist der jeweilige eingeloggte Restaurantbesitzer zu sehen sowie Einstellungen wie zum Beispiel die Einstellung für das Ändern des Namens in der Navbar oder das Ändern der Farbe der Webseite.

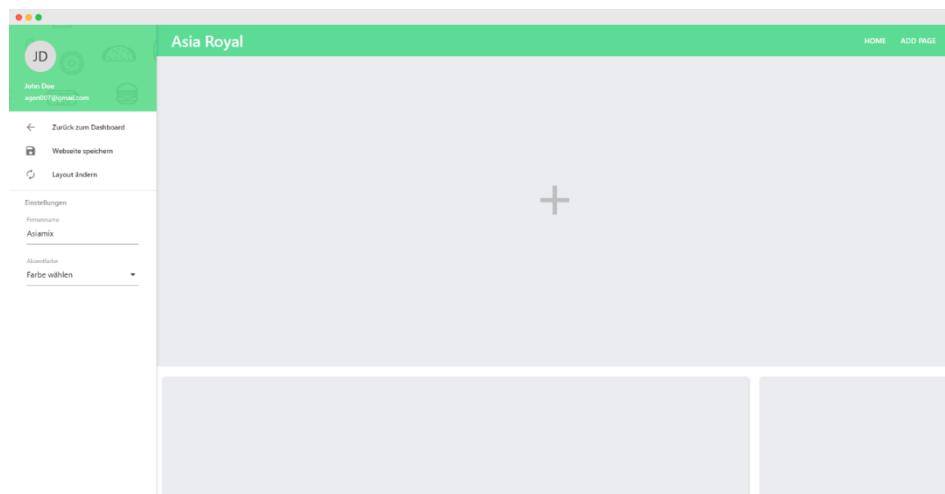


Abbildung 83: Baukastensystem - Ansicht Restaurantbesitzer

In folgender Abbildung ist die Ansicht des Kunden zu sehen. Die grauen Felder dienen hier als Platzhalter. Darin befinden sich normalerweise die vom Restaurantbesitzer eingestellten Module. Hier kann der Kunde Informationen zum Restaurant abfragen und über ein Bestellformular Gerichte zur Abholung oder zur Lieferung bestellen.

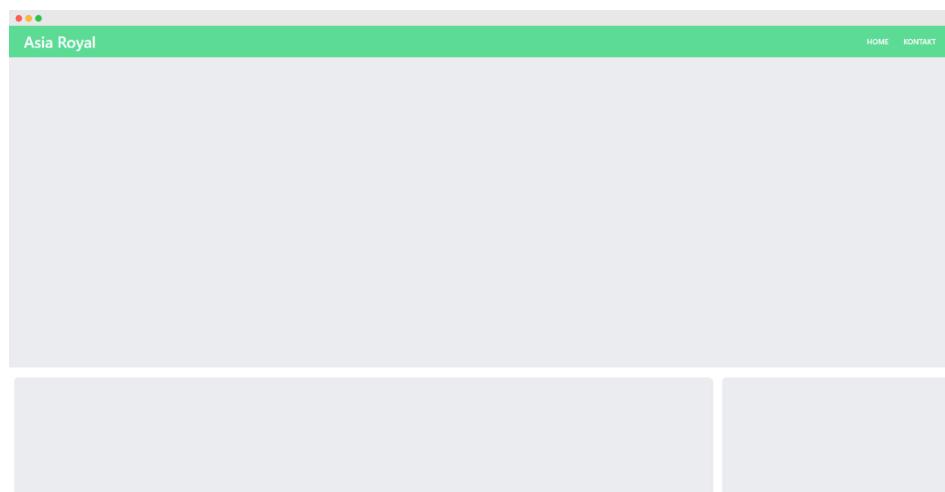


Abbildung 84: Baukastensystem - Ansicht Kunde

7.1.6 JQuery REST request

Im Projekt wird REST hauptsächlich für die Datenübertragung an JQuery verwendet. JQuery stellt dann die einzelnen Daten im Modal dar. REST vereinfacht die Darstellung im Modal dadurch muss keine weitere Webpage erstellt werden. Falls die Methode mit einer neuen Webpage gewählt wird, ist REST dann überflüssig, da die Daten direkt vom Objekt an Thymeleaf übergeben werden kann. Die einzige Lösung asynchron Daten in ein Modal zu laden, ist es JQuery sowie REST zu verwenden. Mithilfe der GET Methode von JQuery können einfach die Daten von REST ausgelesen und dargestellt werden. In der folgenden Abbildung 85: Codebeispiel für die Darstellung im Modal ist ein Codebeispiel zu diesem Problem.



```
$.get("http://localhost:8080/api/gericht/" + id, function(data, status){  
    console.log(data);  
    $('#gericht-info-name').val(data.name);  
    $('#gericht-info-beschreibung').val(data.beschreibung);  
    $('#gericht-info-preis').val(data.preis);  
    $('#gericht-info-aktionspreis').val(data.preisangebot);  
});
```

Abbildung 85: Codebeispiel für die Darstellung im Modal

7.1.7 Benachrichtigungen

Um Fehler- oder Erfolgsmeldungen auszugeben, bietet Materializecss die Möglichkeit Informationen in Form von Toasts auszugeben. Das sind Pop-up-Benachrichtigungen, die kurze Mitteilungen beinhalten. Die Funktionalität wird clientseitig in JavaScript implementiert. Da die Benachrichtigungen meistens beim Aufruf der Webseite erscheinen soll, wird die „ready“ Funktion von jQuery verwendet.



```
$(document).ready(function () {  
    let searchParams = new URLSearchParams(window.location.search);  
  
    if (searchParams.get('login') == 'success') {  
        M.toast({html: 'Erfolgs Meldung', displayLength: '4000'});  
    }  
});
```

Abbildung XX veranschaulicht, wie der Wert eines URL-Parameters als Toast ausgegeben wird.

Falls man Informationen des Backend über Thymeleaf als Toast anzeigen möchte, zeigt Abbildung XX, wie dies zu implementieren ist.

```
● ● ●  

```

Dieser Code wird bei der Ausgabe von Eingabefehlern, bei der Registrierung verwendet. Dabei erhält es eine Array Liste, vom Backend und iteriert über jede einzelne Fehlermeldung. Die „postToast“ Funktion zeigt den Toast mit passender Benachrichtigung an.

7.2 Backend

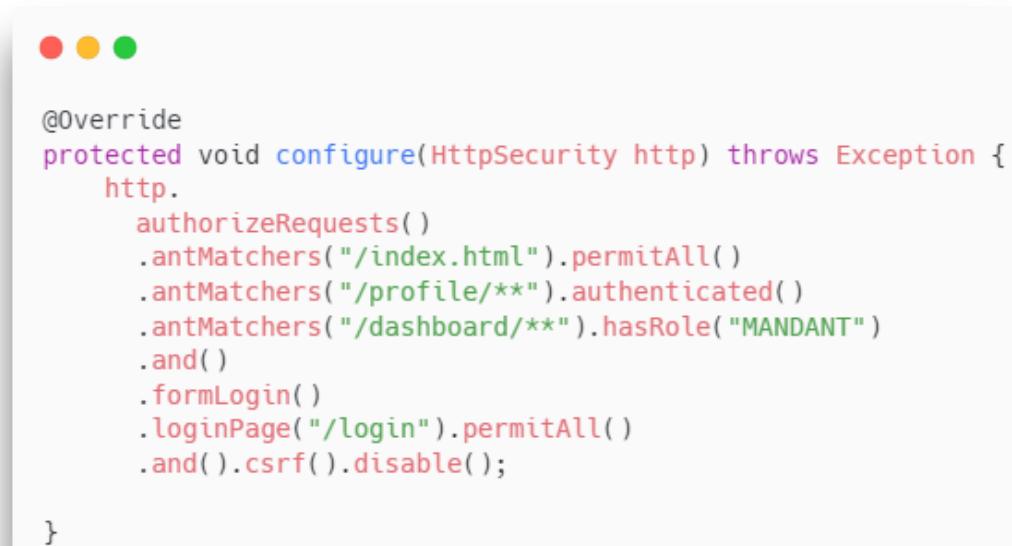
Die folgenden Abschnitte dokumentieren, wie das Backend implementiert wurde. Es wird beschrieben, wie Daten korrekt aus der Datenbank oder anderen Schnittstellen geladen werden und im Frontend angezeigt werden. Dabei wird konkret auf den Login- und Registrierungsprozess, die Restaurantsuche, den Zahlungsprozess und die Dashboard Implementierung eingegangen.

1.1.3 Spring Security

Da verschiedene Benutzertypen existieren, die verschiedene Berechtigungen besitzen, hat jeder Benutzer auch eine Rolle. Es gibt die Rollen Kunde, Mandant und Angestellter. Als Kunde soll man in der Lage sein Essen zu bestellen. Als Mandant muss man alle eingehenden Bestellungen und Zahlungen sehen, und seine Restaurant-Seite bearbeiten können. Als Angestellter soll man wissen, welche Gerichte man kochen muss und sie, als erledigt markieren, sobald die Bestellung zubereitet ist.

In Spring Security kann man nun konfigurieren, welche Ressourcen welche Rollen aufrufen dürfen, oder welche sie nicht sehen dürfen. Dafür ist die diese Methode aus der SecurityConfiguration Klasse zuständig.

NEUES BILD



```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http.
        authorizeRequests()
        .antMatchers("/index.html").permitAll()
        .antMatchers("/profile/**").authenticated()
        .antMatchers("/dashboard/**").hasRole("MANDANT")
        .and()
        .formLogin()
        .loginPage("/login").permitAll()
        .and().csrf().disable();

}
```

Abbildung 86: Spring Boot Security Konfiguration

Desweiterem wird hier konfiguriert, welche Art von Login implementiert wird und welcher Pfad beim Einloggen bzw. Ausloggen aufgerufen werden muss.

Da Spring Security einige Voreinstellungen hat, muss in der Main-Klasse diese Einstellung deaktiviert werden.

```
@SpringBootApplication(exclude = { SecurityAutoConfiguration.class })
```

1.1.3.1 Passwort Hashen mit Bcrypt

Um die Passwörter verschlüsseln zu können, benötigt man eine Hashfunktion, der den String zu einem Hash umwandelt. Der BCryptPasswordEncoder ist dabei die gängige Variante in Spring Boot.

```
@Bean
public BCryptPasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder();
}
```

Diese kann man Global im Projekt nutzen, da eine Bean in der Spring Konfiguration Klasse implementiert wurde. Somit kann man diese Bean in jeder beliebigen Klasse injizieren und verwenden.

1.1.3.2 Authentifizierungsprozess

Dieser Abschnitt erklärt, wie Spring Security implementiert wurde, um einen User zu authentifiziert.

Wie schon unter **Fehler! Verweisquelle konnte nicht gefunden werden. Fehler! Verweisquelle konnte nicht gefunden werden.** beschrieben, ruft der AuthenticationManager den passenden

AuthenticationProvider auf, um den Benutzer anzumelden. Dabei sind folgende zwei Methoden zu implementieren.

```
● ● ●  
 @Override  
 protected void configure(AuthenticationManagerBuilder auth){  
     auth.authenticationProvider(authenticationProvider());  
 }  
  
 @Bean  
 DaoAuthenticationProvider authenticationProvider(){  
     DaoAuthenticationProvider daoAuthenticationProvider = new DaoAuthenticationProvider();  
     daoAuthenticationProvider.setPasswordEncoder(passwordEncoder());  
     daoAuthenticationProvider.setUserDetailsService(this.userPrincipalDetailsService);  
  
     return daoAuthenticationProvider;  
 }
```

Dem AuthenticationProvider wird desweiterem auch der Passwort Encoder und der PrincipalDetailsService übergeben. Der PrincipalDetailsService ist dazu dar einen Datensatz aus der Kunden- oder Angestelltendatenbank zu finden. Dazu existiert die loadUserByUsername Methode, die vom UserDetailsService implementiert wird.

```
● ● ●  
 @Override  
 public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {  
     UserPojo user = findUserByEmail(username);  
     if (user ==null) {  
         throw new UsernameNotFoundException("User not found");  
     }  
     UserPrincipal userPrincipal = new UserPrincipal(user);  
     return userPrincipal;  
 }
```

Diese Methode ruft die Hilfs Methode findUserByEmail auf. Falls kein Objekt zurückliefert wird, wird die UsernameNotFoundException ausgegeben, und der Login war erfolglos. Falls ein User gefunden wurde, wird es zu einem UserPrincipal Objekt umgewandelt und dem Authentifizierung Provider zurückgegeben.

```

public UserPojo findUserByEmail(String email){
    UserPojo userPojo = null;
    Optional<Kunde> kunde = Optional.ofNullable(this.kundeRepository.findByEmail(email));
    Optional<Angestellter> angestellter = Optional.ofNullable(this.angestellterRepository.findByEmail(email));

    if (kunde.isPresent()){
        userPojo = UserPojo.builder().
            benutzername(kunde.get().getEmail()).
            passwort(kunde.get().getPasswort()).
            rollen(Arrays.asList(new Rolle("ROLE_KUNDE"))).
            build();
    } else if (angestellter.isPresent()){
        userPojo = UserPojo.builder().
            benutzername(angestellter.get().getEmail()).
            passwort(angestellter.get().getPasswort()).
            rollen(angestellter.get().getRolle()).
            build();
    }

    return userPojo;
}

```

Die `findUserByEmail` Methode wird nicht nur von der `loadUserByUsername` Methode aufgerufen, da sie beim Registrierungsprozess auch verwendet wird. Diese Methode sucht in der Kunde und Angestellter Tabelle, ob ein Datensatz mit der übergebenen E-Mail vorhanden ist. Falls ja, dann wandelt es das Objekt in ein `UserPojo` um, um ein einheitliches Objekt zu erstellen, das unabhängig von der Tabelle die essenziellen Einträge besitzt.

Das `UserPrincipal`-Objekt besitzt nicht nur einen Benutzernamen –in diesem Fall die E-Mail– und ein Passwort, sondern auch Autoritäten bzw. Rollen, die als Berechtigungen fungieren, um Ressourcen aufrufen zu dürfen. Die `getAuthorities` Methode liefert eine Liste an `GrantedAuthorities` zurück, die dem Benutzer zugesprochen wurden.

```

@Override
public Collection<? extends GrantedAuthority> getAuthorities() {
    List<GrantedAuthority> authorities = new ArrayList<>();

    // Extract list of roles (ROLE_name)
    this.user.getRollen().forEach(r -> {
        System.out.println(r.getRolle());
        GrantedAuthority authority = new SimpleGrantedAuthority(r.getRolle());
        authorities.add(authority);
    });

    return authorities;
}

```

Somit ist der ganze Implementierungsteil des Logins beschrieben. Den restlichen Teil erledigt der Authentifizierung Provider, falls ein Benutzer versucht sich anzumelden.

1.1.3.3 Angemeldeten Benutzer im Controller auslesen

Falls man im Controller Layer den angemeldeten Benutzer auslesen möchte, kann man das folgenderweise machen.

```
● ● ●  
@GetMapping("/currentusername")  
public String currentUserName(@AuthenticationPrincipal UserPrincipal principal) {  
    return principal.getUsername();  
}
```

Mittels der Annotation `@AuthenticationPrincipal`, kann man sich den im Moment authentifizierten Benutzer als `UserPrincipal` speichern.

1.1.4 Google Maps API

Um die Google API nutzen zu können, muss der API Key in der `application.properties` Datei hinterlegt sein. Mit der `@Value` Annotation, kann man diesen Wert auslesen.

```
● ● ●  
@Value("${google.api.key}")  
private String apiKey;
```

In der Applikation sind folgende drei verschiedene Schnittstelltypen, der Google Maps API implementiert:

- Geocoding API
- Reverse Geocoding API
- Places Autocomplete API

Jeder API Aufruf läuft dabei nach folgendem Prinzip ab:

- Rückgabeobjekt erzeugen
- GeoApiClient erstellen
- API-Aufruf ausführen
- GeoApiClient-Verbindung schließen

1.1.4.1 Geocoding API

Die Geocoding API, wird für zwei Use-Cases benötigt. Zum einen, wird die aufgerufen, immer dann, wenn ein neues Restaurant sich als Mandant registriert und zum anderen, wenn man nach einem Restaurant in der Umgebung sucht.

```
● ● ●  
@Service  
public interface GeocodingApi {  
    GeoPositionDto getGeodaten(String placeId) throws AdresseNotFoundException ;  
    String findPlzByAdresse(String adresse) throws AdresseNotFoundException;  
}
```

Die `getGeodaten` Methode fordert eine `placeId` als Parameter an, und gibt dann ein `GeoDatenDto` zurück. Dieses DTO beinhaltet den Längengrad, Breitengrad und die Postleitzahl. Falls die Adresse nicht gefunden werden konnte, wird die `AdresseNotFoundException` geworfen. Diese wird bei der Registrierung aufgerufen.

Damit man von einer Adresse die Postleitzahl bekommt, wird die `findPlzByAdresse` Methode verwendet. Zu beachten ist dabei, dass die Google Maps API leider inkonsistent ist, da bei ungenauen Aufrufen, keine eindeutige Postleitzahl zurückgeliefert wird. Ein gutes Beispiel ist dabei, wenn man als Adresse nur nach der Stadt Wien sucht. Die API liefert keine genaue Response zurück, da Wien mehrere Postleitzahlen hat. Dieses Problem wird gelöst, indem man nochmals einen umgedrehten Geocoding Request macht. Man sendet die Geodaten, an die Reverse Geocoding API, und sucht speziell nach der Postleitzahl.

```
//Bei z.B. Wien gibt es mehrere PLZ, deshalb liefert keine PLZ
//if "locationType": "APPROXIMATE"
//then reverse lookup latlng mit postalcode als response
if (!gson.toJson(results[0].geometry.locationType).equals("\u201cAPPROXIMATE\u201d")){
    int length = Integer.parseInt(gson.toJson(results[0].addressComponents.length));
    String plz = gson.toJson(results[0].addressComponents[length-1].longName);
    plz = plz.substring(1);
    plz = plz.substring(0,plz.length()-1);
    return plz;
} else {
    double lat = Double.parseDouble(gson.toJson(results[0].geometry.location.lat));
    double lng = Double.parseDouble(gson.toJson(results[0].geometry.location.lng));
    LatLng latLng = new LatLng( lat, lng );
    String plz = reverseGeocodingApi.getPlz(latLng);
    return plz;
}
```

Um Code Duplikate zu vermeiden, wurde die `geocode` Hilfs-Methode erstellt, was den API Request ausführt.

```
public GeocodingResult[] geocode(String adresse) throws AdresseNotFoundException {
    GeocodingResult[] results;
    GeoApiClientContext context = new GeoApiClientContext.Builder()
        .apiKey(apiKey)
        .build();
    try {
        results = com.google.maps.GeocodingApi.geocode(context,adresse)
            .components(ComponentFilter.country("Österreich")).language("de").await();
    } catch (ApiException e) {
        e.printStackTrace();
        throw new AdresseNotFoundException();
    } catch (InterruptedException e) {
        e.printStackTrace();
        throw new AdresseNotFoundException();
    } catch (IOException e) {
        e.printStackTrace();
        throw new AdresseNotFoundException();
    }
    context.shutdown();
    return results;
}
```

1.1.4.2 Reverse Geocoding API

Mittels der Reverse Geocoding API, wird, wie vorher schon erklärt, die Postleitzahl, aus den Geodaten herausgelesen.

```
● ● ●

@Override
public String getPlz(LatLng latLng){

    GeocodingResult[] results = new GeocodingResult[0];
    GeoApiClientContext context = new GeoApiClientContext.Builder()
        .apiKey(apiKey)
        .build();

    try {
        results = GeocodingApi.reverseGeocode(context, latLng)
            .language("de").resultType(AddressType.POSTAL_CODE).await();
    } catch ( ApiException e) {
        e.printStackTrace();
    } catch ( InterruptedException e) {
        e.printStackTrace();
    } catch ( IOException e) {
        e.printStackTrace();
    }
    context.shutdown();

    Gson gson = new GsonBuilder().setPrettyPrinting().create();
    String plz = gson.toJson(results[0].addressComponents[0].longName);
    plz = plz.substring(1);
    plz = plz.substring(0,plz.length()-1);

    return plz;
}
```

Zu beachten ist dabei, dass resultType(AddressType.POSTAL_CODE), der API mitteilt, dass die Postleitzahl mitgesendet werden soll.

1.1.4.3 Places Autocomplete API

Wann immer im Frontend eine Adresse eingegeben wird, sendet die Applikation einen Request an die Places Autocomplete API. Diese API liefert alle möglichen Örtlichkeiten zurück, dessen Name Ähnlichkeiten, mit der eingegebenen Adresse hat. Implementiert wurden aus diesem Grund zwei Methoden, in der PlacesApImpl Klasse, welche das PlacesAPI Interface implementiert.

```
● ● ●

@Service
public interface PlacesApi {
    List<String> getPlaces(String adresse) throws AdresseNotFoundException;
    List<AddressDto> getAdressen(String adresse) throws AdresseNotFoundException;
}
```

Die getPlaces Methode, liefert eine Liste an allen möglichen Adressen zurück.

Die getAdressen Methode filtert die Örtlichkeiten nur nach validen Wohnadressen. Somit ist die Genauigkeit bei der Registrierung genauer und verursacht weniger Fehler.

```

● ● ●

try {
    predictions = com.google.maps.PlacesApi
        .placeAutocomplete(context, adresse, new PlaceAutocompleteRequest.SessionToken("Lieferrex"))
        .language("de")
        .components(ComponentFilter.country("at"))
        .types(PlaceAutocompleteType.ADDRESS)
        .await();
} catch (ApiException e) {
    e.printStackTrace();
}

```

Desweiterem werden nur Adressen zurückgegeben, die einen Ort, eine Straße, eine Hausnummer und eine Placeld zurückliefern.

1.1.5 Registrierung

Folgende Abschnitte erklären, wie sich Kunden, und Restaurants bei unserer Webseite einen Account erstellen können.

1.1.5.1 Kunden Registrierung

Die Kundenregistrierung ist ein essenzieller Teil, um die Webseite als Kunde benützen zu können. Der Benutzer muss dabei folgende Persönliche Daten, Vorname, Nachname, E-Mail-Adresse, Telefonnummer, Passwort und Adresse in die Registrierungsmaske eingeben. Desweiterem muss er die Allgemeinen Geschäftsbedingungen akzeptieren.

```

● ● ●

<form class="col s12" id="reg-form" th:action="@{/register}" th:object="${user}" th:method="POST">
    <input id="vorname" type="text" class="validate login-inp" placeholder="Vorname"
    th:field="*{vorname}">

    <input id="nachname" type="text" class="validate login-inp" placeholder="Nachname"
    th:field="*{nachname}">

    <input id="email" type="email" class="validate login-inp" placeholder="Email"
    th:field="*{email}">

    <input id="telefonnummer" type="tel" class="validate login-inp" placeholder="Telefonnummer"
    th:field="*{telefonnummer}">

    <input id="passwort" type="password" class="validate login-inp" placeholder="Passwort"
    th:field="*{passwort}">

    <input id="confirmPasswort" type="password" class="validate login-inp"
    placeholder="Passwort wiederholen" th:field="*{confirmPasswort}">

    <input id="adresse" type="text" class="validate login-inp" placeholder="Adresse"
    th:field="*{adresse}">

    <input th:name="ort" id="ort" type="hidden">
    <input th:name="strasse" id="strasse" type="hidden">
    <input th:name="land" id="land" type="hidden">
    <input th:name="hausnummer" id="hausnummer" type="hidden">
    <input th:name="placeId" id="placeId" type="hidden">

    <input type="checkbox" name="agb" th:value="on"
    th:checked="${user.agb eq 'on'}">

    <input type="checkbox" name="newsletter" th:value="on"
    th:checked="${user.newsletter eq 'on'}">

    <button type="submit" class="btn btn-success">Registrieren</button>
</form>

```

Abbildung XX zeigt ausschließlich nur den Thymleaf Code, wie ein POST Request, auf „/register“ gesendet wird. Jedes einzelne Thymleaf-Feld ist dabei eine Eigenschaft im KundeRegistrationDto-Objekt.

Sobald das Formular abgesendet wurde, wird im Controller, die Eingabe Daten validiert. In Spring Boot kann man dies implementieren, indem man das eingehende Objekt mit der @Valid Annotation markiert. Für die Fehlermeldungsausgabe ist es notwendig ein BindingResult Objekt direkt hinter dem eingehenden Objekt als Parameter in der Methodesignatur zu stellen.

```
● ● ●

@PostMapping("/register")
public String registerUserAccount(@ModelAttribute("user") @Valid KundeRegistrationDto registrationDto,
                                    BindingResult result, Model model) {
    List<String> returnVal = new ArrayList<>();

    UserPojo existingEmail = userPrincipalDetailsService.findUserByEmail(registrationDto.getEmail());
    if (existingEmail != null) {
        returnVal.add("Benutzer mit dieser E-Mail Adresse existiert bereits");
        model.addAttribute("returnVal", returnVal);
        return "main/register";
    }

    //Pseudocode
    for each error in result do
        model.add(error)

    if (!kundeService.save(registrationDto)) {
        returnVal.add("Adresse konnte nicht gefunden werden");
        model.addAttribute("returnVal", returnVal);
        return "main/register";
    }

    returnVal.add("Benutzer erfolgreich erstellt");
    model.addAttribute("returnVal", returnVal);
    return "main/register";
}
```

In Abbildung XX, wird gezeigt, wie die Methode implementiert wurde, die den POST-Request vom Formular übernimmt. Sie überprüft zuerst, ob bereits eine E-Mail in der Kunde- oder Angestelltentabelle vorhanden ist. Danach überprüft es, ob im result Error Meldungen eingetragen wurden. Fall keine Fehler aufgetreten sind wird die „save“ Methode aufgerufen, welche die Kundendaten in die Kunden-Tabelle speichert.

Damit die Validierung funktioniert, muss das KundenRegistrationDto mit Annotationen korrekt markiert werden. Dabei hilft die „java.validation“ Bibliothek. Sie beinhaltet Annotationstypen, um die variablen in einem bestimmten Wert einzuschränken zu können.

```

import javax.validation.constraints.*;

@FieldMatch.List({
    @FieldMatch(first = "password", second = "confirmPassword",
               message = "Passwörter müssen übereinstimmen"),
})
public class KundeRegistrationDto {
    @NotEmpty(message = "Vorname darf nicht leer sein")
    private String vorname;

    @NotEmpty(message = "Passwort darf nicht leer sein")
    @Size(min = 8)
    private String password;

    @NotEmpty(message = "Prüf Passwort darf nicht leer sein")
    @Size(min = 8)
    private String confirmPassword;

    @ValidPhoneNumber
    private String telefonnummer;

    @AssertTrue(message = "Agb's müssen angekreuzt sein")
    private boolean agb;
}

```

Abbildung XX ist nur ein Auszug der KundenRegistrationDto-Klasse.

Die Annotationen „@ValidPhoneNumber“ und „@FieldMatch“, sind eigens geschriebene Annotationen. Die „ValidPhoneNumber“ Annotation die überprüfen mit Reguläre Ausdrücke, ob die eingegebene Telefonnummer valide ist.

```

@Override
public boolean isValid(String s, ConstraintValidatorContext constraintValidatorContext) {
    Pattern pattern = Pattern.compile("^(+)(\\d{1,3})\\s)?((\\(\\d{3,5}\\))|\\d{3,5})(\\s)?\\d{3,8}$");
    Matcher matcher = pattern.matcher(s);
    return matcher.matches();
}

```

Die „FieldMatch“ Annotation, überprüft, ob zwei Strings vom Inhalt identisch sind.

```

@Override
public boolean isValid(final Object value, final ConstraintValidatorContext context) {
    try {
        final Object firstObj = BeanUtils.getProperty(value, firstFieldName);
        final Object secondObj = BeanUtils.getProperty(value, secondFieldName);
        return firstObj == null & secondObj == null || firstObj != null & firstObj.equals(secondObj);
    } catch (final Exception ignore) {}
    return true;
}

```

Falls die Überprüfung keine Fehlerhaften Daten diagnostiziert hat, wird „true“ zurückgegeben. Andernfalls ist die Eingabe nicht valide.

1.1.5.2 Mandanten Registrierung

Damit man Bestellungen, bei einem Mandant aufgeben kann, muss das Restaurant sich zuvor registriert haben. Dabei werden Restaurantspezifische Daten, wie Firmenname, Restaurant-E-Mail-Adresse, Mindestbestellwert, Lieferkosten, Telefonnummer und Adresse benötigt. Um einen Dashboard Benutzer zu erstellen, werden auch daten für einen Dashboard Administrator benötigt. Hierbei sind der Vorname, Nachname, E-Mail-Adresse und Passwort einzugeben. Zum Schluss ist noch das Akzeptieren der AGBs von Nöten.



```

<form class="col s12" id="reg-form" th:action="@{/register}" th:object="${user}" th:method="POST">
    <input id="firmenname" type="text" class="validate login-inp" placeholder="Firmenname"
        th:field="*{firmenname}">
    <input id="firmenemail" type="email" class="validate login-inp" placeholder="Restaurant-Email"
        th:field="*{firmenemail}">
    <input id="mindestbestellwert" type="number" step="any" class="validate login-inp"
        placeholder="Mindestbestellwert €" th:field="*{mindestbestellwert}">
    <input id="lieferkosten" type="number" step="any" class="validate login-inp"
        placeholder="Lieferkosten €" th:field="*{lieferkosten}">
    <input id="telefonnummer" type="tel" class="validate login-inp" placeholder="Telefonnummer"
        th:field="*{telefonnummer}">

    <input id="adresse" type="text" class="validate login-inp" placeholder="Adresse" th:field="*{adresse}">

    <input th:name="ort" id="ort" type="hidden">
    <input th:name="strasse" id="strasse" type="hidden">
    <input th:name="land" id="land" type="hidden">
    <input th:name="hausnummer" id="hausnummer" type="hidden">
    <input th:name="placeId" id="placeId" type="hidden">

    <input id="vorname" type="text" class="validate login-inp" placeholder="Vorname"
        th:field="*{vorname}">
    <input id="nachname" type="text" class="validate login-inp" placeholder="Nachname"
        th:field="*{nachname}">
    <input id="email" type="email" class="validate login-inp" placeholder="Email"
        th:field="*{email}">
    <input id="password" type="password" class="validate login-inp" placeholder="Passwort"
        th:field="*{password}">
    <input id="confirmPassword" type="password" class="validate login-inp"
        placeholder="Passwort wiederholen" th:field="*{confirmPassword}">

    <input type="checkbox" name="agb" th:value="on" th:checked="${user.agb eq 'on'}">
    <input type="checkbox" name="newsletter" th:value="on" th:checked="${user.newsletter eq 'on'}">

    <button type="submit" class="btn btn-success">Registrieren</button>
</form>

```

Abbildung XX zeigt den Thymleaf Code, um einen POST-Request auf „/register“ zu machen. Dort werden die Eingabefelder wieder mittels der @Validate Annotation validiert. Für mehr Informationen wurde dieselbe Art Eingabedaten zu validieren in 1.1.5.1 Kunden Registrierung durchgeführt.

Im Controller Layer, wird der POST-Request entgegengenommen, und überprüft, ob bereits ein User mit derselben E-Mail-Adresse vorhanden ist. Falls fehlerhafte Daten eingegeben worden sind, werden diese auch zurückgegeben.

Mit der „saveRegistrationDto“ Methode, aus dem Mandanten Service, wird die Restaurantinformationen, in die Datenbanktabelle geschrieben.

```

@Override
public boolean saveRegistrationDto(MandantRegistrationDto mandantRegistrationDto) {
    try {
        geoPositionDto = geocodingApi.getGeodaten(mandantRegistrationDto.getPlaceId());
    } catch (AdresseNotFoundException e) {
        return false;
    }

    geoPositionRepository.save(geoPosition);

    Mandant mandant = new Mandant().builder().
        ...
        .build();
    mandantRepository.save(mandant);

    Angestellter angestellter = new Angestellter().builder().
        ...
        .build();
    angestellterRepository.save(angestellter);

    return true;
}

```

Die saveRegistrationDto Methode, ruft zu aller erst die „getGeodaten“ Methode des geocodingAPI Service auf. Diese liefert eine geoPosition zurück, welche mittels der „save“ Methode des Geopositions Repository in die Datenbank gespeichert wird.

Desweiterem, werden die Restaurantspezifische Daten, aus dem DTO geladen und mittels des Builder-Patterns in ein Mandant-Objekt gespeichert. Dieses wird danach auch in die Datenbank gespeichert. Ein Angestellter-Objekt mit Mandanten rechte, wird danach auch erstellt und in die Datenbank gespeichert. Zum Schluss wird noch ein boolescher wert zurückgegeben, je nach dem, ob die Registrierung erfolgreich war.

1.1.5.3 Interaktive Adresseingabe mit der Google Maps API

Bei der Registrierung, wird besonders wert daraufgelegt, dass eine Valide Adresse gespeichert wird. Deshalb wird mittels JavaScript die Autocomplete Funktion der Google Maps Places API verwendet. Dabei wird 500ms, nachdem man etwas Eingegeben hat der „/seach/adresse/{inputAdresse}“ End Point im LoginRegistrationRestController aufgerufen. Dieser liefert alle Adressen zurück, die die Google API als Response sendet.

```

const input = document.querySelector('#adresse');
input.addEventListener('keyup', (e) => {
    const text = e.currentTarget.value;
    // Clear timer
    clearTimeout(timer);
    // Wait for X ms and then process the request
    timer = setTimeout(() => {
        search(text);
    }, waitTime);
});

```

Mittels JavaScript wird jede Änderung des Adress-Inputfeld zu erfassen. Sobald man für 500ms keinen Tastenanschlag gemacht wurde, wird die Funktion „search“ aufgerufen.

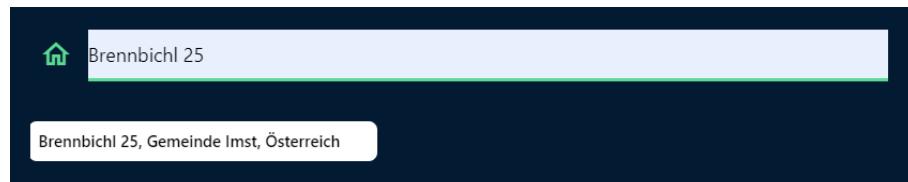
```
● ● ●
async function search(adresse) {
    // Storing response
    const response = await fetch(api_url + adresse);

    // Storing data in form of JSON
    data = await response.json();

    if (await data.length != 0){
        show(data);
    }else {
        clear();
    }
}
```

Diese Asynchrone Funktion holt die Daten von der REST Schnittstelle und lässt sie anzeigen mit der „show“ Funktion.

Im Frontend werden danach alle Adressen angezeigt, die für die Eingabe zutreffen.



1.1.6 Benutzer Login

Wie Spring Boot mit Spring Security, einen Benutzer authentifiziert wurde, bereits in [1.1.3.2 Authentifizierungsprozess](#) genauer beschrieben. Dieser Abschnitt fokussiert sich darauf, wie die Logindaten, ans Backend gelangen und wohin der User weitergeleitet wird, nach einer erfolgreichen Anmeldung.

Dabei wird ein HTML-Formular für den POST Request benötigt. Dabei ist zu beachten, dass die Felder „name“ und „id“ jeweils „username“ oder „password“ heißen. Spring Security kann somit die Inputfelder identifizieren, und den Benutzer anmelden.

```
● ● ●
<form class="col s12" id="login-form" th:action="@{/login}" method="post">
    <input id="username" name="username" type="email" class="validate login-inp"
           placeholder="Email">
    <input id="password" name="password" type="password" class="validate login-inp"
           placeholder="Passwort">
    <button type="submit" name="login-submit" id="login-submit"
           class=" btn waves-light btn-raised btn-round waves-effect btn-login">login</button>
</form>
```

In der Spring Security Konfiguration, ist konfiguriert, dass nach einer erfolgreichen Anmeldung, der Benutzer an "/successLogin" weitergeleitet wird. Dort wird mittels der „@AuthenticationPrincipal“ Annotation der angemeldete User identifiziert.

```

● ● ●

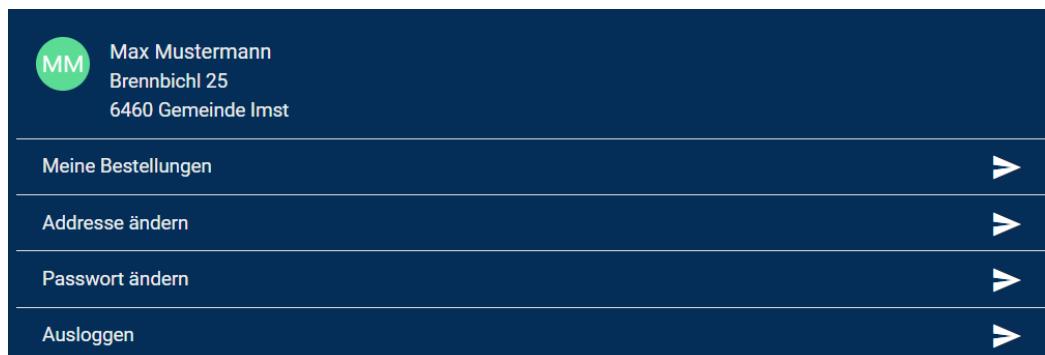
@GetMapping("/successLogin")
public String redirectSuccessLogin(RedirectAttributes redirectAttrs, @AuthenticationPrincipal UserPrincipal principal) {
    if (principal.getAuthorities().contains(new SimpleGrantedAuthority("ROLE_KUNDE"))) {
        redirectAttrs.addAttribute("login", "success");
        return "redirect:/";
    } else if (principal.getAuthorities().contains(new SimpleGrantedAuthority("ROLE_MANDANT"))) {
        return "redirect:/dashboard";
    } else if (principal.getAuthorities().contains(new SimpleGrantedAuthority("ROLE_ANGESTELLTER"))) {
        return "redirect:/dashboard/bestellungen";
    } else {
        return "redirect:/";
    }
}

```

An dieser Stelle, wird entschieden, wohin der Benutzer weitergeleitet wird. Ein Kunde wird auf die Homepage weitergeleitet. Ein Benutzer mit der Rolle Mandant oder Angestellter wird zum Dashboard weitergeleitet.

1.1.7 Kundenspezifische Daten anzeigen

Sobald sich ein Kunde angemeldet hat, sollen Persönliche Daten auf der Homepage von ihm angezeigt werden. Somit wird die Usability verbessert, und der Kunde weiß direkt Bescheid, dass dies sein Account ist. Beispiel dafür ist die Profilanzeige, auf der Homepage.



Das „GetMapping“ der Homepage sieht dabei folgendermaßen aus.

```

● ● ●

@GetMapping("")
public String showIndexPage(@AuthenticationPrincipal UserPrincipal principal, Model model) {
    if (principal == null) {
        System.out.println("Index page loaded");
        return "main/index";
    } else {
        model.addAttribute("login", true);
        Kunde kunde = kundeRepository.findByEmail(principal.getUsername());
        KundenIndexDto kundenIndexDto = new KundenIndexDto().builder()
            ...
            .build();
        model.addAttribute("kunde", kundenIndexDto);
        return "main/index";
    }
}

```

Falls kein Benutzer angemeldet ist, wird die Homepage wie gewohnt geladen. Ist jedoch ein User authentifiziert, dann wird ein „kundenIndexDto“ als Attribut dem Model mitgegeben. Desweiterem, wird der boolesche Wert „true“ mitgeliefert, falls jemand angemeldet ist.

1.1.8 Adresse ändern

Als Kunde seine Adresse ändern zu können, ist eine der wichtigsten Use-Cases der ganzen Applikation, da es gleichzeitig auch die Lieferadresse ist. Dafür muss man mit seinem Kundenkonto, angemeldet sein, und „/changeAddress“ Seite aufrufen. Dort befindet sich eine Eingabemaske für die Adresse. Diese holt sich wie bei 1.1.5.3 Interaktive Adresseingabe mit der Google Maps API alle ähnlichen Adressen der Eingabe über die Google Places API und gibt diese aus.

Das „PostMapping“ des Controllers sieht folgendermaßen aus.

```
● ○ ●

@PostMapping("/changeAddress")
public String changeAddress(@ModelAttribute("adresse") AdressDto adressDto,
                            @AuthenticationPrincipal UserPrincipal principal) {
    if (adressDto.getPlaceId().isEmpty()){
        return "redirect:/changeAddress?NA";
    }
    if (!kundeService.adresseAendern(adressDto, principal)) {
        return "redirect:/changeAddress?error";
    }

    return "redirect:/changeAddress?success";
}
```

In Abbildung XX wird zuerst entschieden, ob die Eingabe eine PlaceId beinhaltet. Danach wird vom Kunden Service die „adresseAendern“ Methode aufgerufen. Falls die Änderung erfolgreich war, wird eine Erfolgsmeldung ausgegeben. Wenn keine Adresse mitgesendet wurde, oder die Adressänderung nicht durchgeführt werden konnte, wird eine Fehlermeldung ausgegeben.

```
● ○ ●

@Override
public boolean adresseAendern(AdressDto adressDto, UserPrincipal principal) {
    GeoPositionDto geoPositionDto;

    try {
        geoPositionDto = geocodingApi.getGeodaten(adressDto.getPlaceId());
    } catch (AdresseNotFoundException e) {
        System.out.println("Geodaten nicht gefunden");
        return false;
    }

    Kunde kunde = kundeRepository.findByEmail(principal.getUsername());
    kunde.setPlz(geoPositionDto.getPlz());
    //Pseudocode
    kunde.setAddress(adressDto);
    kundeRepository.save(kunde);

    return true;
}
```

Im Kunden Service, wird zuerst die Postleitzahl mittels der geocodingApi ermittelt. Danach wird der passende Datensatz der Kundentabelle mit der E-Mail-Adresse geladen. Bei diesem Kunden-Objekt wird die Adresse laut dem adressDto-Objekt geändert. Danach wird das Kunden-Objekt mit der

„save“ Methode in die Datenbank gespeichert. Falls die Änderung erfolgreich war, wird „true“ als boolescher Wert zurückgegeben, ansonsten „false“.

1.1.9 Passwort ändern

Falls der Kunde aus Sicherheitsgründen sein Passwort ändern möchte, hat er die Möglichkeit, dies über den "/changePassword" Link zu erledigen. Dazu muss er angemeldet sein und sein altes Passwort, wie ein neues Passwort samt Bestätigungspswrd als POST-Request an das Backend senden. Mit der @Valid Annotation, wird das DTO-Objekt auf Validierungsfehler überprüft. Das DTO-Objekt sieht dabei folgendermaßen aus.

```
● ● ●

@FieldMatch.List({
    @FieldMatch(first = "password", second = "confirmPassword",
               message = "Passwörter müssen übereinstimmen"),
})
public class PasswortAendernDto {

    @NotEmpty
    private String altesPasswort;

    @NotEmpty
    @Size(min = 8)
    private String password;

    @NotEmpty
    @Size(min = 8)
    private String confirmPassword;
}
```

Falls die Eingabewerte valide sind, ruft der „LoginRegistrationController“ im Kunden Service die „passwortAendern“ Methode auf. Diese wird das „PasswortAendernDto“, und der angemeldeten Benutzer als Parameter weitergegeben. Die „passwortAendern“ Methode sucht danach das Passende Kunden Objekt mittels der „findByEmail“ Methode des Kunden Repositorys. Danach wird überprüft, ob das eingegebene alte Passwort, mit dem Passwort übereinstimmt, welches in der Datenbank hinterlegt ist. Dazu wird die „matches“ Methode des Passwort Encoders aufgerufen. Diese liefert „true“ zurück, falls die Passwörter gleich sind und „false“ falls sie ungleich sind. Im letzterem Fall, wird „false“ an den Controller zurückgesendet. Falls die Passwörter übereinstimmen, wird das neue Passwort in die Datenbank geschrieben, und „true“ an den Controller zurückgegeben.

```
● ● ●

@Override
public boolean passwortAendern(PasswortAendernDto passwortAendernDto, UserPrincipal principal) {

    Kunde kunde = kundeRepository.findByEmail(principal.getUsername());
    String password = kunde.getPassword();

    if (!passwordEncoder.matches(passwortAendernDto.getAltesPasswort(), password)){
        return false;
    }

    kunde.setPassword(bCryptPasswordEncoder.encode(passwortAendernDto.getPassword()));
    kundeRepository.save(kunde);

    return true;
}
```

1.1.10 Restaurantsuche

Damit Kunden bequem, Restaurants in ihrer Umgebung finden können, wurde eine Interaktive Eingabemaske erstellt, die abhängig von der Eingabe alle ähnlichen Adressen anzeigt. Dabei wurde die Google Maps Places API verwendet. Sobald der Kunde seine Wunschadresse eingegeben hat, wird die Postleitzahl seiner Adresse berechnet. Danach wird der Kunde auf eine Seite weitergeleitet, die alle Restaurants anzeigt, die dieselbe Postleitzahl haben, wie die der Adresseingabe. Falls der Kunde nur eine Postleitzahl eingibt, dann wird in der Datenbank gesucht, ob ein Restaurant mit dieser Postleitzahl existiert. Wenn ja, dann gibt es direkt alle Restaurants mit dieser Postleitzahl aus. Wenn nicht, wird die bereits genannte Variante gewählt.

Desweiteren gibt es mehrere Filtermöglichkeiten, um die Auswahl einzuschränken. Es kann nach der Kategorie, den Lieferkosten, den Mindestbestellwert, der Lieferkosten und den Öffnungszeiten gefiltert werden. Die Restaurants, die den Filter nicht passieren, werden nicht angezeigt. Die Filterwerte, werden als GET-URL-Parameter mitgesendet. Folgender URL String ist ein Beispiel für eine Restaurantsuche.



```
/restaurants/6460?geoeffnet=true&lieferkosten=2.5&mindestbestellwert=10&kategorie=FINE_DINING
```

Die folgende Abbildung, zeigt, wie das „GetMapping“ des Controller-Layer für diesen Use-Case aussieht.



```
@GetMapping("/restaurants/{plz}")
public String showRestaurants(@PathVariable String plz, @RequestParam(required = false) String geoeffnet,
    @RequestParam(required = false) String lieferkosten,
    @RequestParam(required = false) String mindestbestellwert, @RequestParam(required = false) String kategorie,
    @RequestParam(required = false) String adresse, Model model) {

    double dLieferKosten = lieferkosten == null ? 0.0 : Double.parseDouble(lieferkosten);
    double dMindestbestellwert = mindestbestellwert == null ? 0.0 : Double.parseDouble(mindestbestellwert);

    try {
        List<MandantSuchDto> mandanten = mandantService.findMandantByPlz(plz, Boolean.parseBoolean(goeffnet),
            dLieferKosten, dMindestbestellwert, kategorie);
        model.addAttribute("plz", plz);
        model.addAttribute("adresse", adresse);
        model.addAttribute("mandanten", mandanten);
        return "main/search";
    } catch (MandantNotFoundException mandantNotFoundException) {
        model.addAttribute("error", "Es wurden keine Restaurants in der Umgebung gefunden!");
        return "main/search";
    }
}
```

Um alle Restaurants zu finden, wird die „findMandantByPlz“ des Mandanten Service aufgerufen. Dort befindet sich die Businesslogik, die entscheidet, ob das Restaurant durch den Filter angezeigt wird oder nicht.

```

    ● ● ●
@Override
public List<MandantSuchDto> findMandantByPlz(String Adresse, boolean isGeöffnet, double lieferKosten,
                                              double mindestbestellwert, String kategorie) throws MandantNotFoundException {

    List<Mandant> mandantenList = mandantRepository.findMandantByPlz(Adresse);
    if (mandantenList.isEmpty()) {throw new MandantNotFoundException();}

    LocalDate currentDate = LocalDate.now();
    WochentagEnum currentDay = WochentagEnum.valueOf(String.valueOf(currentDate.getDayOfWeek()));
    LocalTime currentTime = LocalTime.now();

    List<MandantSuchDto> mandantSuchDtoList = new ArrayList<>();
    for (Mandant mandant : mandantenList) {
        int bewertung;
        boolean open = isGeoeffnet(mandant, currentDay, currentTime);
        if (mandant.getBestellungen().size()==0){bewertung = 0;}
        else {bewertung = getBewertung(mandant);}

        if (lieferKosten != 0.0 && lieferKosten < mandant.getLieferkosten()) {continue;}
        if (mindestbestellwert != 0.0 && mindestbestellwert < mandant.getMindestbestellwert()) {continue;}
        if (kategorie != null && !kategorie.equals(mandant.getKategorie().getName().toString())) {continue;}
        if (isGeöffnet && !open) {continue;}

        mandantSuchDtoList.add(new MandantSuchDto().builder()
            ...
            .build());
    }
    return mandantSuchDtoList;
}

```

Aus dem Mandanten Repository, werden alle Mandanten, per Postleitzahl aus der Datenbank geladen. Danach wird mit einer For-Each-Schleife über diese Liste iteriert. Falls ein wert den Filter nicht passiert, wird mit „continuie“ dieser Listeneintrag übersprungen. Die Mandant-Objekte, die den Filter passiert haben, werden in ein DTO gespeichert, und dem Controller retourniert.

Die „isOpen“ Methode untersucht, ob das Restaurant gerade geöffnet ist. Dabei sucht sie die aktuelle Öffnungszeit eines Mandanten, und überprüft, ob sich die aktuelle Uhrzeit zwischen der Öffnungszeit befindet.

```

    ● ● ●
@Override
public boolean isGeoeffnet(Mandant mandant, WochentagEnum currentDay, LocalTime currentTime) {

    boolean open = true;
    Optional<Oeffnungszeit> optHeutigeOeffnungszeit = null;

    optHeutigeOeffnungszeit = oeffnungszeitRepository.findOeffnungszeitsByMandantAndTag(mandant, currentDay);
    if (optHeutigeOeffnungszeit.isPresent()) {
        Oeffnungszeit heutigeOeffnungszeit = optHeutigeOeffnungszeit.get();
        if (heutigeOeffnungszeit.getOeffnungszeit() == null || heutigeOeffnungszeit.getSchliessungszeit() == null){
            open = false;
            return open;
        }
        open = open && currentTime.isAfter(heutigeOeffnungszeit.getOeffnungszeit().toLocalTime());
        open = open && currentTime.isBefore(heutigeOeffnungszeit.getSchliessungszeit().toLocalTime());

        if (heutigeOeffnungszeit.getStartpause() != null && heutigeOeffnungszeit.getStartpause() != null) {
            open = open && (currentTime.isBefore(heutigeOeffnungszeit.getStartpause().toLocalTime()) ||
                currentTime.isAfter(heutigeOeffnungszeit.getEndepause().toLocalTime()));
        }
    } else {
        open = false;
    }
    return open;
}

```

1.1.11 PayPal API

Um die PayPal API nutzen zu können, muss der paypal.mode, die paypal.client.id und das paypal.client.secret aus der „application.properties“ Datei gelesen werden. Diese wird in der „PayPalConfig“ Klasse verwendet, um Requests and die API senden zu können. Für die Checkout API, wurden folgende Beans implementiert.

```
● ● ●  
@Bean  
public OAuthTokenCredential oAuthTokenCredential() {  
    return new OAuthTokenCredential(clientId, clientSecret, paypalSdkConfig());  
}  
  
@Bean  
public APIContext apiContext() throws PayPalRESTException {  
    APIContext context = new APIContext(oAuthTokenCredential().getAccessToken());  
    context.setConfigurationMap(paypalSdkConfig());  
    return context;  
}
```

Die „apiContext“ Methode, wird immer dann aufgerufen, wenn ein Request an die API gesendet werden soll.

Um einen Request auf die Payout API durchführen zu können, ist die Bean in Abbildung XX implementiert worden.

```
● ● ●  
@Bean  
public PayPalHttpClient apiClient() {  
    PayPalEnvironment environment = new PayPalEnvironment.Sandbox(clientId, clientSecret);  
    PayPalHttpClient client = new PayPalHttpClient(environment);  
    return client;  
}
```

1.1.11.1 Checkout API Implementierung

Der Bezahlungsprozess mittels der Checkout API, wird in zwei Phasen aufgeteilt. In ersterer, wird ein Kunde auf eine externe Bezahlseite weitergeleitet, wo er sich mit seinem PayPal Account anmelden muss, und die Rechnung bezahlen kann. Dafür ist die „createPayment“ Methode zuständig.

```

    @Override
    public Payment createPayment(Double total, String description, String cancelUrl, String successUrl)
        throws PayPalRESTException {

        Amount amount = new Amount();
        amount.setCurrency("EUR");
        total = new BigDecimal(total).setScale(2, RoundingMode.HALF_UP).doubleValue();
        System.out.println(String.valueOf(total));
        amount.setTotal(String.valueOf(total));

        Transaction transaction = new Transaction();
        transaction.setDescription(description);
        transaction.setAmount(amount);

        List transactions = new ArrayList();
        transactions.add(transaction);

        Payer payer = new Payer();
        payer.setPaymentMethod("paypal");

        Payment payment = new Payment();
        payment.setIntent("ORDER");
        payment.setPayer(payer);
        payment.setTransactions(transactions);
        RedirectUrls redirectUrls = new RedirectUrls();
        redirectUrls.setCancelUrl(cancelUrl);
        redirectUrls.setReturnUrl(successUrl);
        payment.setRedirectUrls(redirectUrls);

        return payment.create(apiContext);
    }
}

```

Diese Methode, erstellt ein Payment-Objekt, mit allen notwendigen Daten, wie z.B. die Währung, den Geldbetrag und die Links für die Weiterleitung, falls die Bezahlung auf der PayPal-Seite erfolgreich bzw. abgebrochen wurde.

In Phase zwei, wird die Bezahlung ausgeführt und bezahlt. Dafür wird die „executePayment“ Methode ausgeführt. Um einen bezahl API Request durchzuführen, ruft diese Funktion die „execute“ Methode, des Payments-Objekt auf. Als Parameter, muss das apiContext-Objekt –was den API Request sendet– und das paymentExecute-Objekt – was die Zahlungsinformationen beinhaltet – mitgegeben werden.

```

    @Override
    public Payment executePayment(String paymentId, String payerId) throws PayPalRESTException {
        Payment payment = new Payment();
        payment.setId(paymentId);
        PaymentExecution paymentExecute = new PaymentExecution();
        paymentExecute.setPayerId(payerId);
        return payment.execute(apiContext, paymentExecute);
    }
}

```

1.1.11.2 Payout API Implementierung

Nachdem der Kunde das Geld an das Lieferrex PayPal Konto überwiesen hat, wird der geldbetrag, an den PayPal Account des Restaurants gesendet. Dafür ist die „payMandant“ Methode zuständig. In dieser Methode, müssen zuerst Metadaten, wie die Währung den Preis und die E-Mail-Adresse des Mandanten gesetzt werden. Diese werden in ein PayoutItem-Objekt gespeichert und der „item“ Liste hinzugefügt. Allgemeine Daten, wie den Empfängertyp oder den E-Mail-Betreff wird in einem CreatePayoutRequest-Objekt übergeben.

```

● ● ●

@Override
public void payMandant(BezahlDto bezahlDto) throws ClientsideMandantPaymentException, ServersideMandantPaymentException {
    double preis = new BigDecimal(bezahlDto.getPreis()).setScale(2, RoundingMode.HALF_UP).doubleValue();
    List<compaypal.payouts.PayoutItem> items = new ArrayList<compaypal.payouts.PayoutItem>();
    items.add(new compaypal.payouts.PayoutItem().senderItemId("")
        .note("").receiver(bezahlDto.getMandantEmail())
        .amount(new Currency().currency("EUR").value(String.valueOf(preis))));

    CreatePayoutRequest request = new CreatePayoutRequest()
        .senderBatchHeader(new SenderBatchHeader().senderBatchId(bezahlDto.getBestellNr()))
        .emailMessage(bezahlDto.getMandantNachricht()).emailSubject("Eingehende Lieferrex bestellung")
        .note("Beste Grüße dein Lieferrex-Team").recipientType("EMAIL").items(items);

    try {
        PayoutsPostRequest httpRequest = new PayoutsPostRequest().requestBody(request);
        HttpResponseMessage<CreatePayoutResponse> response = apiClient.execute(httpRequest);

        CreatePayoutResponse payouts = response.result();
        payouts.links().forEach(link -> System.out.println(link.rel() + " => " + link.method() + ":" + link.href()));
    } catch (IOException ioe) {
        if (ioe instanceof HttpException) {
            // Something went wrong server-side
            HttpException he = (HttpException) ioe;
            System.out.println(he.getMessage());
            throw new ServersideMandantPaymentException();
        } else {
            // Something went wrong client-side
            throw new ClientsideMandantPaymentException();
        }
    }
}

```

Im „try-catch-Block“ der Abbildung XX, wird der API Request, gesendet. Falls dabei eine „IOException“ auftreten sollte, wird zwischen Serverseitigen oder Clientseitigen Fehler unterschieden, und eine Passende Exception geworfen.

1.1.12 Checkoutseite

Sobald der Kunde alle Gerichte, in seinem Warenkorb gepackt hat und bezahlen möchte, wird er auf die Checkoutseite weitergeleitet. Dort kann er seine Gerichte nochmals überprüfen. Diese Gerichte werden vom Cookie des Warenkorbs geladen und angezeigt.

Im Controller wird dieses Cookie gelesen und die „createPayment“ Methode des PayPal Service ausgeführt. Dort wird man zur PayPal Bezahlungsseite weitergeleitet.

```

● ● ●

try {
    Payment payment = paypalService.createPayment(bezahlDto.getPreis(), bezahlDto.getKundenNachricht(),
        "/restaurant/" + id + "/checkout/cancel",
        "/restaurant/" + id + "/checkout/success");

    for(Links link:payment.getLinks()) {
        if(link.getRel().equals("approval_url")) {
            return "redirect:" + link.getHref();
        }
    }
} catch (PayPalRESTException e) {
    e.printStackTrace();
}

```

Falls alles funktioniert hat, wird man zur „restaurant/id/checkout/success“ Seite weitergeleitet.

Der „BezahlController“ nimmt diese GET-Anfrage an, und liest die Parameter, paymentId und payerId aus. Diese beinhalten eine lange Zeichenkette, um die Bezahlung, und den Bezahler zu identifizieren.

Die „executePayment“ Methode führt diese Bezahlung danach aus und liefert „approved“ als Status zurück, wenn die Bestellung vom Kunden, an Lieferrex geglückt ist. Danach wird der Mandant mit der

„payMandant“ Methode bezahlt, und die Bestellung mit der „bestellungAufgeben“ Methode in die Datenbank geschrieben.

```
● ● ●

try {
    Payment payment = paypalService.executePayment(paymentId, payerId);

    if (payment.getState().equals("approved")) {
        BezahlDto bezahlInfos = bestellungService.getBezahlDto(einkaufswagenDto);
        paypalService.payMandant(bezahlInfos);
        bestellungService.bestellungAufgeben(einkaufswagenDto, principal, bezahlInfos);
    }
} catch (PayPalRESTException e) {
    System.out.println(e.getMessage());
    return "redirect:/";
} catch (ServersideMandantPaymentException e) {
    System.out.println(e.getMessage());
    return "redirect:/";
} catch (ClientsideMandantPaymentException e) {
    System.out.println(e.getMessage());
    return "redirect:/";
}
```

1.1.12.1 Bestellung anlegen

Sobald alle Bezahlungen erfolgreich durchgeführt wurden, wird die Bestellung in die Datenbank geschrieben, damit das Restaurant die Gerichte zubereiten kann.

```
● ● ●

@Override
public void bestellungAufgeben(EinkaufswagenDto einkaufswagenDto, UserPrincipal userPrincipal, BezahlDto bezahlDto) {
    Mandant mandant = mandantRepository.getById(einkaufswagenDto.getMandantId());
    BestellartEnum bestellartEnum = BestellartEnum.valueOf(einkaufswagenDto.getBestellArt());
    int dauer = bestellartEnum == BestellartEnum.LIEFERUNG ? mandant.getDurchschnittsLieferZeit()
        : mandant.getDurchschnittsAbholZeit();
    mandant = setUmsatz(mandant, bezahlDto.getPreis());
    Bestellung bestellung = new Bestellung().builder()
        ...
        .build();
    bestellungRepository.save(bestellung);

    for (EinkaufswagenDetailDto einkaufswagenDetailDto : einkaufswagenDto.getEinkaufswagenDetails()) {
        for (int i=0; i < einkaufswagenDetailDto.getAnzahl(); i++) {
            GerichtBestellung gerichtBestellung = new GerichtBestellung(null, einkaufswagenDetailDto.getAnmerkung(),
                gerichtRepository.getById(einkaufswagenDetailDto.getGerichtID()), bestellung);

            gerichtBestellungRepository.save(gerichtBestellung);
        }
    }
}
```

Dabei muss Zwischen Abholung und Lieferung unterschieden werden, da die Lieferung länger benötigt als die Abholung. Des Weiteren, muss beachtet werden, dass die Umsatzsumme und der Umsatz in für den aktuellen Monat angepasst wird. Um den Bestellungen die Gerichte hinzuzufügen, wird in der Gericht-Bestellung Tabelle ein Eintrag des Gerichts-ID, und der Bestellungs-ID als Fremdschlüssel eingefügt.

1.1.13 Bestellungen anzeigen

Als Kunde möchte man einen guten Überblick, über seine Bestellungen haben. Dazu kann man unter „/orders“ alle seine Bestellungen, als angemeldeter Kunde einsehen. In der „BestellController“ Klasse, ist dazu ein passendes „GetMapping“ vorhanden.

```

    @GetMapping("/orders")
    public String showOrdersPage(@AuthenticationPrincipal UserPrincipal principal, Model model) {
        List<BestellDto> bestellDtos = bestellungService.getBestellDto(principal.getUsername());
        model.addAttribute("bestellungen", bestellDtos);
        return "main/orders";
    }

```

Die „showOrders“ Methode, ruft im Bestellung Service die Methode auf, um vom angemeldeten Benutzer alle Bestellinformationen zu erhalten.

```

@Override
public List<BestellDto> getBestellDto(String kundenEmail) {
    Kunde kunde = kundeRepository.findByEmail(kundenEmail);
    List<Bestellung> bestellungList = bestellungRepository.getBestellungByKunde(kunde);
    List<BestellDto> bestellDtoList = new ArrayList<>();
    for (Bestellung bestellung: bestellungList) {
        HashMap<String, Integer> gerichtNameAnzahl = new HashMap<>();

        for (GerichtBestellung gericht : bestellung.getGerichteBestellungen()) {
            if(!gerichtNameAnzahl.containsKey(gericht.getGericht().getName())){
                gerichtNameAnzahl.put(gericht.getGericht().getName(),1);
            } else{
                gerichtNameAnzahl.put(gericht.getGericht().getName(),gerichtNameAnzahl
                    .get(gericht.getGericht().getName())+1);
            }
        }

        BestellDto bestellDto = new BestellDto().builder()
            ...
            .build();
        bestellDtoList.add(bestellDto);
    }
    return bestellDtoList;
}

```

In der „getBestellDto“ Methode, werden zunächst alle Bestellungen geladen, die der angemeldet User getätigt hat. Da jedoch in der Datenbank, die Anzahl an gleichen Gerichten gespeichert wird, muss mit über alle Bestellungen durchiteriert werden, um zu überprüfen, welche Gerichte pro Bestellung häufiger gekauft wurden. Dies wird in einer HashMap gespeichert, wo der Key der Gerichtsname und der Value die Anzahl ist. Zum Schluss wird die Gerichte-HashMap und die Bestelldetails in dem BestellDto gespeichert.

1.1.13.1 Bestellung bewerten

Damit Restaurants Feedback von den Kunden einholen können und Kunden, die Bewertung eines Restaurants feststellen können, muss es möglich sein die Bestellungen, auf der Bestellhistorie zu bewerten.

```

<form th:action="@{/rate/{bestellId}(bestellId=${bestellung.bestellId})}" method="get" class="d-inline">
    <a onclick="$(this).closest('form').submit();">
        <span class="material-icons-outlined sr-rest-star-span c-pointer star-1">star_rate </span>
        <input type="hidden" th:name="rating" th:value="1">
    </a>
</form>

```

In Abbildung XX, ist ein Auszug des Thymleaf Code zu sehen, um ein GET-Request, an den Bestellung Controller zu senden. Das Input-Feld beinhaltet dabei den Value der Bewertung. Im Controller, wird

der Wert der Bewertung, der momentan authentifizierte Benutzer und die zu bewertende Bestellung an den Bestell Service geleitet.

```

@Override
public boolean makeRating(UserPrincipal principal, long bestellId, int rating) {
    if (!(rating == 1 || rating == 2 || rating == 3 || rating == 4 || rating == 5)){
        return false;
    }

    Optional<Bestellung> optionalBestellung= bestellungRepository.findBestellungById(bestellId);
    if (!optionalBestellung.isPresent()){return false;}

    Kunde kunde = kundeRepository.findByEmail(principal.getUsername());
    if (optionalBestellung.get().getKunde().getId() != kunde.getId()){return false;}

    optionalBestellung.get().setBewertung(rating);
    bestellungRepository.save(optionalBestellung.get());

    return true;
}

```

Die „makeRating“ Methode des Bestell Service, überprüft, ob die Bewertung, die Bestellung und der angemeldete Benutzer valide ist. Falls ja, wird der Wert der Bewertung in die Datenbank eingefügt.

7.2.1 Overview Seite Angestellten / Mandant

In diesem Abschnitt werden die Darstellungen auf der Webseite beschrieben sowie einzelne Bedienelemente und Programmabschnitte erläutert. Der Aufbau des eingeloggten Users wird unterschieden in Mandant und Angestellter, auch sind zwei verschiedene Einstiegspunkte zu sehen. Der Angestellte besitzt den Einstiegspunkt Zubereitung, hingegen zum Mandanten der in Overview startet. Die Sicht des Angestellten beschränkt sich auf zwei Hauptregister, eines davon ist die Zubereitung und die zweite Seite ist Zustellung. In den jeweiligen Seiten können die Bestellungen von den Kunden abgerufen werden. Die Ansicht wird in der folgenden Abbildung 87 dargestellt.

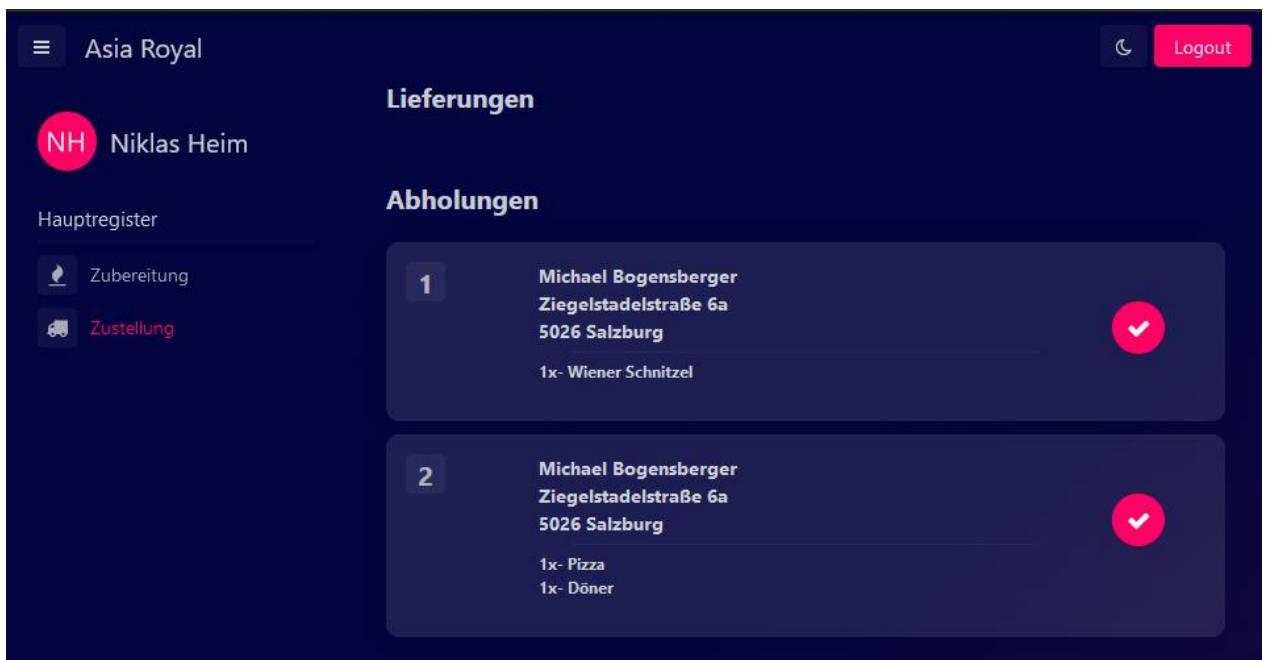


Abbildung 87 Ansicht des Angestellten

Hingegen zum Angestellten hat der Restaurantbesitzer bzw. der Mandant eine komplette Ansicht über sein eigenes Unternehmen sowie die aktuellen Zustellungen und Zubereitungen. Die Ansicht

des Mandanten wird Hauptsächlich in zwei Register geteilt, eines davon ist das Hauptregister in welchem sich eine grobe Zusammenfassung sowie die Bestellungen befinden.

Im zweiten Register befinden sich Hauptsächlich Restaurant Spezifische Daten. In der folgenden Abbildung 88 sind die zwei Register sowie eine grobe Übersicht des Restaurants zu sehen.

#	Besteller	Art	Preis
1	Michael Bogensberger	LIEFERUNG	12.0 €
1	Michael Bogensberger	ABHOLUNG	123.0 €
1	Michael Bogensberger	ABHOLUNG	15.5 €

Abbildung 88 Mandanten Ansicht

In der Overview Seite sind die Seitenaufrufe in diesem Monat zu sehen, dass bedeutet wie oft die Kunden in diesem Monat die Seite besucht haben. In der nächsten Box wird der Umsatz in diesem Monat dargestellt. Darauffolgend kommen die verkauften Gerichte im gesamten Zeitraum. In der nächsten Reihe sind die letzten 3 Bestellungen zu sehen und die Popularität der Gerichte, das heißt welches Gericht am meisten gekauft wurde.

In der Kachel auf der rechten Seite sind Verlinkungen zu den anderen Seiten. Die Darstellung der einzelnen Infos sind aus der Datenbank zu holen, dies erfolgt durch den OverviewController. In der folgenden Abbildung 89 ist ein Codeausschnitt mit den Daten des eingeloggten Users zu sehen, sowie die Daten von dazugehörigen Mandanten.

```
● ● ●

public String seitenAufruf(@AuthenticationPrincipal UserPrincipal principal, Model model){
    Angestellter foundAngestellter = angestellterService.findByEmail(principal.getUsername());
    Mandant foundMandant = foundAngestellter.getMandant();
    return "dashboard/dashboard.html";
}
```

Abbildung 89 Daten des eingeloggten User

Die Daten des eingeloggten Users sind Essentiell, sie sind dafür da um weiteren Daten des dazugehörigen Mandanten aus der Datenbank zu holen. In der Abbildung 90 erfolgt der Aufruf an den Service welche eine Schnittstelle zum Repository besitzt. Das Repository holt sich die Daten aus der Datenbank und übergibt dies dem Service. Der Service übergibt die Daten dem Controller. Bis man zum letzten Endpunkt Thymeleaf gelangt. Thymeleaf stellt dann die Daten mithilfe der Thymeleaf Template Engine dar. Durch den Aufruf werden in derselben Abbildung die Seitenaufrufe sowie die verkauften Gerichte aus der Datenbank geholt.

```
● ● ●

public String seitenAufruf(@AuthenticationPrincipal UserPrincipal principal, Model model){
    Angestellter foundAngestellter = angestellterService.findByEmail(principal.getUsername());
    Mandant foundMandant = foundAngestellter.getMandant();

    Seitenaufrufe seitenaufrufe = overviewService.seitenaufrufe(foundMandant);
    long verkaufteGerichte = overviewService.getVerkaufteGerichte(foundMandant.getId());
    return "dashboard/dashboard.html";
}
```

Abbildung 90 Daten auslesen aus der Datenbank

Zum Schluss werden die Daten an einem Model gebunden. Dieser Erfolgt durch die Methode model.addAttribute(), dort werden als Parameter den Attribut Namen sowie die Methode bzw. die Datenwerte übergeben. In der Abbildung 91 sind die Models für die Overview Seite zu sehen.

```
● ● ●

public String seitenAufruf(@AuthenticationPrincipal UserPrincipal principal, Model model){
    model.addAttribute("seitenaufrufe", seitenaufrufe.getAufrufe());
    model.addAttribute("umsatz", umsatzSumme);
    model.addAttribute("gerichteVerkauft", verkaufteGerichte);
    model.addAttribute("user", foundAngestellter.getVorname() + ' ' + foundAngestellter.getNachname());
    model.addAttribute("vname", foundAngestellter.getVorname());
    model.addAttribute("nname", foundAngestellter.getNachname());
    model.addAttribute("firmenname", foundMandant.getFirmenname());
    model.addAttribute("role", foundAngestellter.getRolle().iterator().next().getRolle());
    return "dashboard/dashboard.html";
}
```

Abbildung 91 OverviewController Attribute des Model hinzufügen

7.2.2 Zubereitung

In der Zubereitung Seite werden die aktuelle Bestellungen angezeigt. In den Kacheln werden die Gerichte mit der Anzahl angezeigt, dadurch weiß der Koch welche Gerichte zubereitet werden müssen. Ist die Bestellung fertig, so kann das Gericht abgehakt werden. Die Bestellung landet automatisch in die Zustellungsseite. Die Darstellung der einzelnen Elemente erfolgen mithilfe einer HashMap. Die HashMap bietet den Vorteil eines Key Value Prinzip, dies ist genau passend für die Darstellung von z.B „2x – Nudeln“. Der Key des HashMaps sind die Gerichte und die Values sind unsere Stückzahlen. Um auch noch die anderen Werte in einem Objekt darzustellen werden die Daten in ein Data Transfer Object gepackt kurz DTO. Mithilfe Thymeleaf werden die Objekte dargestellt, um die Objekte bzw. die HashMap darzustellen werden die Methoden th:each für das iterieren der Map verwendet und th:text für die Darstellung der einzelnen Datenfelder bzw. die Eigenschaften der Objekte.

The screenshot shows a dark-themed web application interface. On the left is a sidebar with navigation links: 'Asia Royal', 'Hauptregister' (selected), 'Overview', 'Zubereitung' (selected), 'Zustellung', 'Verwaltung' (selected), 'Webseite', 'Gerichte', 'Bewertungen', 'Benutzerrechte', 'Zahlungen', 'Mandant', 'Suchmaschine', and 'Offnungszeiten'. The main content area is divided into 'Lieferungen' and 'Abholungen'. The 'Abholungen' section displays two orders: '1x- Wiener Schnitzel' and '2x- Döner'. Each order card includes the quantity (1 or 2), the item name, the entry time ('Eingang: 17:54' or '20:11'), additional info ('Zusatzinfo: Extra Preiselbeeren' or 'viel Soße, viel Fleisch'), and a checked checkbox icon.

Abbildung 92 Zubereitung Seite

Jede Bestellung die in die Zubereitung Seite kommt besitzt den Status „IN_ZUBEREITUNG“. Durch das Abhaken der jeweiligen Elemente ändert sich der Status von „IN_ZUBEREITUNG“ zu den jeweiligen Status „FERTIG_ZUM_ABHOLEN“ oder „IN_AUSLIEFERUNG“. In der folgenden Abbildung 93 sind die vorgefertigten Status im PHPMYADMIN zu sehen.

+ Optionen			
		bestellstatus_id	bestellstatus
<input type="checkbox"/>	Bearbeiten Kopieren Löschen	1	IN_ZUBEREITUNG
<input type="checkbox"/>	Bearbeiten Kopieren Löschen	2	FERTIG_ZUM_ABHOLEN
<input type="checkbox"/>	Bearbeiten Kopieren Löschen	3	IN_AUSLIEFERUNG
<input type="checkbox"/>	Bearbeiten Kopieren Löschen	4	ABGESCHLOSSEN

Abbildung 93 PHPMYADMIN Status von den Bestellungen

In der Abbildung ist ein Codeausschnitt mit der Template Engine Thymeleaf. Dort werden die einzelnen Methoden verwendet.

```

● ● ●

<div class="content card pt-15 pb-15 pl-15 pr-15 mb-10 mt-0 ml-0 mr-0" th:each="bestellung: ${gerichteBestellungsDetail}">
    <div class="row">
        <div class="col-2 ">
            <span class="sidebar-icon">
                <h2 class="card-title font-weight-bold mb-0" th:text="${bestellung.counter}">1</h2>
            </span>
        </div>
        <div class="col-8">
            <ul style="list-style-type: none">
                <li class="font-weight-bold mb-0 font-size-14" th:each="gericht: ${bestellung.gerichtBestellungModelList}" th:text="${gericht.value} + 'x- ' + ${gericht.key}">1x - Dürüm</li>
            </ul>
        </div>
    </div>
</div>

```

Abbildung 94 Codeausschnitt der Darstellung von den Daten mithilfe Thymeleaf

Anknüpfend sind Codeausschnitte von der HashMap, sowie die Befüllung der HashMap. Am Anfang bevor die HashMap gefüllt wird, wird abgefragt ob der Key bereits vorhanden ist. Ist der Key nicht vorhanden wird ein Eintrag erstellt mit dem Wert 1. Im Anschluss wird geprüft ob der Key vorhanden ist, falls ja erhöht man den Wert um 1.

```

● ● ●

public String seitenAufruf(@AuthenticationPrincipal UserPrincipal principal, Model model){
    for (Bestellung bestellung: alleBestellungen) {
        SimpleDateFormat sdf = new SimpleDateFormat("HH:mm");
        String timestamp = sdf.format(new Date(bestellung.getBestelldatum().getTime()));
        String zusatzinfo = "";
        String timestampAbholung = sdf.format(new Date(bestellung.getBestelldatum().getTime()));
        String zusatzinfoAbholung = "";

        if(bestellung.getBestellart().getBestellart() == BestellartEnum.LIEFERUNG){
            HashMap<String, Integer> gerichtBestellungModelList = new HashMap<>();
            for(GerichtBestellung gerichtBestellung : bestellung.getGerichteBestellungen()){
                if (gerichtBestellung.getAnmerkung() != null){
                    zusatzinfo += gerichtBestellung.getAnmerkung() + ", ";
                }

                if(!gerichtBestellungModelList.containsKey(gerichtBestellung.getGericht().getName())){
                    gerichtBestellungModelList.put(gerichtBestellung.getGericht().getName(),1);
                } else {
                    gerichtBestellungModelList.put(gerichtBestellung.getGericht().getName(),gerichtBestellungModelList.get(gerichtBestellung.getGericht().getName())+1);
                }
            }

            if(!zusatzinfo.isEmpty()){
                zusatzinfo = zusatzinfo.substring(0, zusatzinfo.length()-2);
            }
            abholung += 1;
            modeldata.add(new BestellungModel(gerichtBestellungModelList,timestamp,zusatzinfo,abholung,bestellung.getId()));
        }
    }
}

```

Abbildung 95 HashMap befüllen

Hakt man die Bestellungen ab wird eine HTTP Request gesendet. In diesem Fall wird eine POST Methode angewendet.

Um die richtige Bestellung zu erwischen wird eine ID als Value vom Button hinzugefügt. Dies sieht wie folgt aus. Mit der Methode th:value von Thymeleaf können die Objekteigenschaften des Objektes Bestellung als Value von dem Button gespeichert werden.

```
● ● ●
<button type="submit" th:value="${bestellung.id}" name="lieferungChangeToFertigZumAbholen"
        class="btn btn-lg btn-square btn-primary rounded-circle mt-0"
        mb-0 "
        role="button"><i class="fa fa-check" aria-hidden="true">
</i></button>
```

Abbildung 96 ID Zuweisung des Buttons

Im Controller wird mithilfe einer Annotation @PostMapping ein PostMapping erstellt, dort werden die Daten mithilfe @RequestParam(name="") an den Controller übergeben. Dazu muss das HTML Element mit der Eigenschaft name="" den gleichen Namen des @RequestParam(name="") entsprechen.

```
● ● ●
public String checkingLieferungAndAbholung(@RequestParam Optional<Long> lieferungChangeToFertigZumAbholen,...){
    if(abholungChangeToFertigZumAbholen.isPresent()){
        Bestellung bestellung =
        bestellungService.bestellungByIdAnzeigen(abholungChangeToFertigZumAbholen.get());
        bestellung.setBestellstatus(bestellungService.bestellstatusAnzeigen(BestellstatusEnum.FERTIG_ZUM_ABHOLEN));
        bestellungService.save(bestellung);
    }
}
```

Abbildung 97 PostMapping der Zubereitung

7.2.3 Zustellung

Auf der Zustellung Seite werden die Daten vom Kunden sowie die Bestellung vom Kunden dargestellt. Hakt man die Bestellung ab wird der Status ABGESCHLOSSEN gesetzt. Die Bestellung verschwindet dann von der Seite und bleibt nur noch in der Datenbank bzw. auf anderen Seiten die die letzten Bestellungen beinhalten.

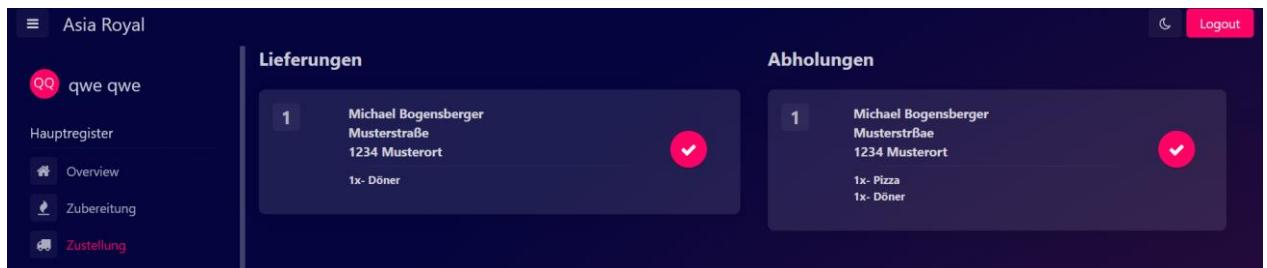


Abbildung 98 Zustellung Seite

Dort wird dieselbe Logik mit der HashMap angewendet um die Gerichte der Bestellung darzustellen. Das DTO umfasst zwei Objekte einmal das Bestellung Objekt und einmal das Kunden Objekt.

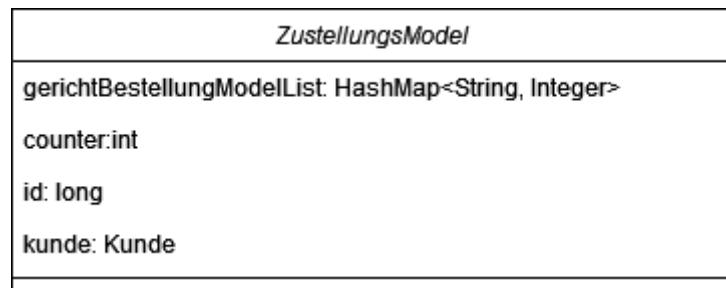


Abbildung 99 Klassendiagramm des DTO ZustellungsModel

Die Speicherung der Daten erfolgt mithilfe eines Posts. Dort wird wie am Anfang schon beschrieben ein Status Wechsel durchgeführt von IN_AUSLIEFERUNG oder FERTIG_ZUM_ABHOLEN zu ABGESCHLOSSEN. In der Abbildung 100 werden die Bestellungen mit der Bestellart Abholung geholt, danach wird der Status auf Abgeschlossen gesetzt.

```

public String checkingLieferungAndAbholung(...){
    if(abholungChangeToFertigZumAbholen.isPresent()){
        Bestellung bestellung =
        zustellungService.bestellungByIdAnzeigen(abholungChangeToFertigZumAbholen.get());
        bestellung.setBestellstatus(zustellungService.bestellstatusAnzeigen(BestellstatusEnum.ABGESCHLOSSEN));
        zustellungService.save(bestellung);
    }
}
  
```

The screenshot shows a Java code editor with a method named 'checkingLieferungAndAbholung'. The code checks if a specific field is present, retrieves a 'Bestellung' object, sets its status to 'ABGESCHLOSSEN', and saves it.

Abbildung 100 ZustellungController Wechsel von IN_AUSLIEFERUNG zu ABGESCHLOSSEN.

7.2.4 Gerichte

Auf der Gerichte Seite kann der Mandant Gerichte bearbeiten, deaktivieren oder neue Gerichte anlegen. Die folgende Abbildung 101 zeigt die Gerichte Seite.

The screenshot shows the 'Gerichte' page in the 'Asia Royal' application. On the left, there is a sidebar with user information ('qwe qwe') and navigation links for 'Overview', 'Zubereitung', 'Zustellung', 'Webseite', 'Gerichte' (which is currently selected), 'Bewertungen', 'Benutzerrechte', 'Zahlungen', 'Mandant', 'Suchmaschine', and 'Öffnungszeiten'. The main area displays three cards: '3 Gerichte', '2 aktiviert', and '1 deaktiviert'. Below these, a table lists two dishes: 'Wiener Schnitzel' and 'Döner'. Each dish has a 'detail' and 'edit' button. To the right, there are two sections: 'Hinzufügen' (with an icon of a flame) and 'Deaktiviertes' (with an icon of a crossed-out dish). A large image of a coffee cup is also visible.

Abbildung 101 Gerichte Seite

In den Oberen 3 Kacheln werden die Status angezeigt. Die Erste Kachel gibt die Anzahl an Gerichte an dort ist es egal welchen Status das Gericht hat, egal ob aktiviert oder deaktiviert. In der zweiten Kachel werden nur aktive Gerichte angezeigt und in der letzten Kachel die deaktivierten Gerichte.

Die Details vom Gericht können abgerufen werden, sowie eine Bearbeitung des Gerichtes. Dies wird mithilfe eines Modals dargestellt.

Informationen zum Gericht

Name:

Wiener Schnitzel

Beschreibung:

Paniertes Kalbsschnitzel mit Kartoffelsalat

Preis:

10.5

Aktionspreis:

0

aktiviert

Aktionspreis verwenden

Abbildung 102 Detailmodal des Gerichtes

Die Modelle werden jeweils mithilfe eines Rest Controller und JavaScript bzw. JQuery befüllt. Der Rest Controller holt die Daten aus der Datenbank, mithilfe JQuery werden die Daten als Variablen in JavaScript gespeichert. Die Methode \$.get("../api/gericht/") holt sich die Daten vom Controller. Um die Felder zu befüllen wird in JQuery die Methode val() verwendet. Dadurch das die Input Felder einen Namen besitzen kann JQuery auf die Felder Namen zugreifen und die Werte setzen. Diesbezüglich hat der User eine saubere und detaillierte Ansicht des Gerichtes.

```
$.get("../api/gericht/" +id, function(data, status){
    $('#gericht-edit-name').val(data.name);
    $('#gericht-edit-desc').val(data.beschreibung);
    $('#gericht-edit-preis').val(data.preis);
    $('#gericht-edit-aktionspreis').val(data.preisangebot);
    $('#gericht-edit-id').val(data.id);
    $('#gericht-edit-status').val(data.status);
    $('#gericht-edit-anzahl-gekauft').val(data.anzahl_gekauft);
```

Abbildung 103 JQuery funktion um die Felder zu befüllen

Nicht nur das Detailmodal verwendet JQuery und Rest auch das Bearbeitungsmodal verwendet JQuery sowie Rest.

Gericht bearbeiten

Name *

Wiener Schnitzel

Beschreibung *

Paniertes Kalbsschnitzel mit Kartoffelsalat

Preis *

10,5

Aktionspreis *

0

aktiviert

Aktionspreis verwenden

speichern

Abbildung 104 Gericht bearbeiten Modal

In Gericht bearbeiten werden ebenso die Daten in die Inputfelder mithilfe JQuery geladen. Damit der User die Daten nicht neu eingeben muss. In Thymeleaf werden weitere Methode verwendet um die Objekte an den Controller zu schicken, diese Methode lautet th:object. Bei th:object werden die Felder in HTML in ein Objekt abgespeichert und an den jeweiligen Controller geschickt. Ein kleiner Ausschnitt des HTML Codes kann in der folgenden Abbildung 105 betrachtet werden.

```
<form th:action="@{/dashboard/gerichte/save}" method="post" th:object="${gericht}">
    <input type="hidden" id="gericht-edit-id" th:field="*{id}">
    <input type="hidden" id="gericht-edit-anzahl-gekauft" th:field="*{anzahl_gekauft}">
    <div class="form-group">
        <label for="gericht-edit-name" class="required">Name</label>
        <input type="text" id="gericht-edit-name" class="form-control" value="" required="required" th:field="*{name}">
    </div>
```

Abbildung 105 HTML mit Thymeleaf th:object

Bei dem Speichern von dem Objekt wird noch vorher überprüft ob der User das Gericht aktivieren will oder deaktivieren möchte. Auch kann der User Entscheiden ob das Gericht einen Aktionspreis besitzt. Das Setzen des Status wird in der folgenden Abbildung 106 dargestellt. Die Zahl eins bedeutet, dass das Gericht aktiviert ist. Die Zahl zwei bedeutet, dass das Gericht aktiviert ist und einen Aktionspreis beinhaltet. Die Zahl 0 hat die Bedeutung, dass das Gericht deaktiviert ist.

```

public String saveGericht(Gericht gericht, @RequestParam Optional<String> aktiviert, @RequestParam
Optional<String> aktion, @AuthenticationPrincipal UserPrincipal principal){
    Angestellter foundAngestellter = angestellterService.findByEmail(principal.getUsername());
    Mandant foundMandant = foundAngestellter.getMandant();
    gericht.setMandant(foundMandant);
    if(!aktiviert.isPresent()){
        gericht.setStatus(0);
    } else if(!aktion.isPresent()){
        gericht.setStatus(1);
    } else {
        gericht.setStatus(2);
    }
}

```

Abbildung 106 Gericht Status setzen

7.2.5 Bewertung

Auf der Bewertung Seite können die Bewertungen zu den einzelnen Bestellungen betrachtet werden. Dort wird der Durschnitt, Median und die Anzahl der Bewertungen berechnet. Sowie die letzten Bestellungen dargestellt. Jede Bestellung wird von 1-5 bewertet bzw. null falls der Kunde keine Bewertung abgeben möchte. Bei den Letzten Bewertungen wird das Datum und die Bestellzeit sowie die Bestellart dargestellt. Der Durschnitt wird mit der Formel d berechnet.

$$d = \frac{\text{Summe der Bewertungen}}{\text{Anzahl der Bewertungen}}$$

In der folgenden Abbildung 107 wird der Durschnitt mit der Summe von 12 und die Anzahl der Bewertungen 4 berechnet, dass bedeutet das $d=12/4$ ist. Somit beträgt der Durschnitt 3.0.

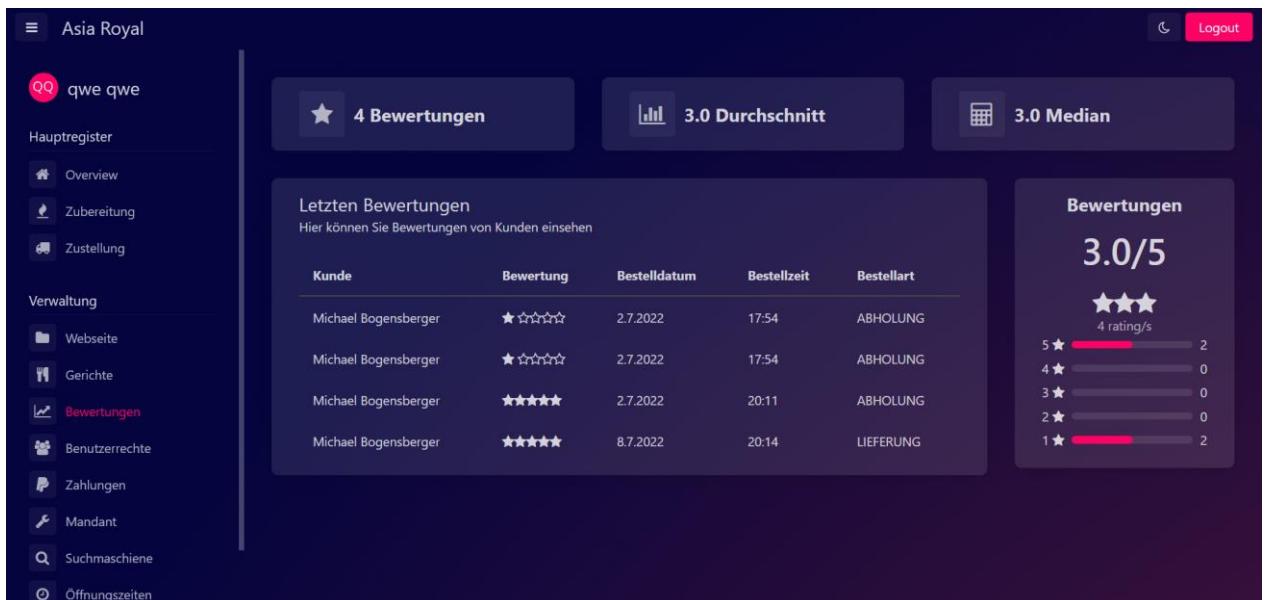


Abbildung 107 Bewertung Seite

Der Median wird durch eine Formel x_{median} berechnet. Bei dem Median wird unterschied ob die Anzahl der Bewertungen gerade oder ungerade ist. x_n ist die Länge des Arrays.

$$\text{Ungerade: } x_{\text{med}} = \frac{x_n}{2}$$

$$\text{Gerade: } x_{\text{med}} = \frac{\frac{x_n}{2} + (\frac{x_n}{2} - 1)}{2}$$

In der Abbildung 108 liegt der Median bei 3.0 das bedeutet bei einem sortierten Array als Beispiel (1, 1, 5, 5) ist es die Stelle 2 (von 0 anfangen zu zählen). Der Wert beträgt 5, und die Stelle 1 beträgt den Wert 1 das heißt der Rechenweg ist $(5+1)/2 = 3$. Ein Codeausschnitt des Median wird in der folgenden Abbildung 108 dargestellt.

```
● ● ●
double median = 0.0;
if (numArray.length != 0 && numArray.length % 2 == 0){
    median = ((double)numArray[numArray.length/2] + (double)numArray[numArray.length/2 - 1])/2;
    System.out.println(Arrays.toString(numArray));
} else if (numArray.length != 0 && numArray.length % 2 == 1){
    median = (double) numArray[numArray.length/2];
}
```

Abbildung 108 Berechnung des Medians

In der Abbildung 109 wird ein Codeausschnitt gezeigt welche den Durchschnitt berechnet und auf zwei Nachkommastellen runden. Als erstes wird überprüft ob die Summe an Bewertungen 0 ist. Ist es nicht der Fall so wird die Summe der Bewertung / die Anzahl an Bewertungen berechnet. Danach runden man auf 2 Nachkommastellen, dies wird mit der Methode setScale(2, RoundingMode.HALF_UP) durchgeführt.

```
● ● ●
if(durchschnittBewertung != 0.0){
    BigDecimal proBestellungBD = new BigDecimal(durchschnittBewertung /
alleBestellungen.size()).setScale(2, RoundingMode.HALF_UP);
    durchschnittBewertung = proBestellungBD.doubleValue();
}
```

Abbildung 109 Berechnung des Durchschnittes anhand der Bewertungen

Für den Bewertungsbalken wurden die Prozente anhand der Anzahl der Bestellung berechnet. Dies wird anhand des Codeausschnitt in der nächsten Abbildung beschrieben.

```
● ● ●
for (Map.Entry<Integer, Double> entry : valueNow.entrySet()){
    if(entry.getValue() != 0){
        BigDecimal proBestellungBD = new BigDecimal((entry.getValue()/sum) *100 ).setScale(2,
RoundingMode.HALF_UP);
        entry.setValue(proBestellungBD.doubleValue());
    }
}
```

Abbildung 110 Berechnung Teilanzahl in % von der Gesamtanzahl

Im Code wird die Teilanzahl z.B 2 von 4 Bestellungen genommen und durch 4 gerechnet um auf die Teilanzahl in Prozent zu kommen. Dies wird benötigt um eine richtige Anzeige des Balkens zu generieren. Mithilfe einer HashMap können durch die Keys 1,2,3,4,5 eine richtige Summe an Sternen generiert werden. Das bedeutet 1 Stern ist als Beispiel der Key und die Value ist 5, d.h 5 Bestellungen wurden mit 1 Stern bewertet. In der Abbildung 111 ist die HTML Darstellung der Balken.



```
<div class="progress-bar" role="progressbar" th:style="'width:' + ${valueNow.get(5)} + '%;'" aria-valuenow="75" aria-valuemin="0" aria-valuemax="100"></div>
```

Abbildung 111 Bewertungsbalken

In diesem Fall ist valueNow.get(5) die 5 Sterne Bewertung in Prozent zur Gesamtanzahl, dies wird dann in Breite eingetragen. Die Sternebewertung in den letzten Bewertungen wird mit einer Schleife in Thymeleaf erzeugt. Dort werden die Bewertung als Zahl in die Funktion numbers.sequence(1, Bewertung) eingegeben. Die Schleife zählt so oft hoch bis die numbers.sequence die bei 1 startet bei der Bewertung landet, während dem Schleifendurchgang werden die einzelnen Sterne erzeugt.



```
<i th:if="${entry.bestellung.bewertung <= 5 and entry.bestellung.bewertung != null}" th:each="i : ${#numbers.sequence(1, entry.bestellung.bewertung)}" class="fa fa-star font-size-30" aria-hidden="true"></i>
```

Abbildung 112 Volle Sterne erzeugen

Die leeren Sterne werden genau anders rum erzeugt, dort wird 5 minus der Bewertung gerechnet.



```
<i th:if="${entry.bestellung.bewertung < 5 and entry.bestellung.bewertung != null}" th:each="i : ${#numbers.sequence(1, 5-entry.bestellung.bewertung)}" class="fa fa-star-o font-size-30" aria-hidden="true"></i>
```

Abbildung 113 Leere Sterne erzeugen

7.3 Benutzerrechte

Auf der Benutzerechte Seite können Angestellte sowie die Mandanten Accounts angelegt werden. Die Accounts die der Koch als Beispiel benutzt sind Angestellte Accounts und können einfach vom Mandanten hinzugefügt werden.

Email	Vorname	Nachname	Rolle	edit	delete
angestellt@gmail.com	Niklas	Heim	Angestellter		
qwe@gmail.com	qwe	qwe	Mandant		
hernandesad@gmail.com	Hernandes	Adonis	Mandant		

Abbildung 114 Benutzerrechte Seite

Auf dieser Seite werden ebenso Modale verwendet, um die Zugänge zu bearbeiten oder zu löschen. Löscht man den Zugang so ist dieser auch in der Datenbank nicht mehr vorhanden. Die Administratoren in dem Fall die Mandanten, haben einen Vollzugriff auf die Webseite, hingegen der normale Angestellter nur Zugriff auf zwei Seiten besitzt.

The screenshot shows a modal window with a dark background and light-colored input fields. The title is "Zugang bearbeiten". It contains the following fields:

- Vorname *: qwe
- Nachname *: qwe
- Email *: qwe@gmail.com
- Rolle *:
 - Angestellter
 - Administrator

A pink "speichern" (store) button is at the bottom.

Abbildung 115 Zugang bearbeiten Modal

Wie auch bei Gericht werden die Daten in die Felder geladen. Bei der Bearbeitung von den Zugängen können die Angestellten Zugänge zu Administratoren bzw. zur Mandanten Rolle gewechselt werden und auch anders rum. Die Befüllung der Daten erfolgt mithilfe eines DTO. In der Datenbank werden die Administratoren Rollen als ROLE_MANDANT abgespeichert.

```
for (Angestellter angestellter : angestellterList){
    Iterator itr = angestellter.getRolle().iterator();

    if(itr.hasNext()){
        Rolle rolle = (Rolle) itr.next();
        if(rolle.getRolle().trim().equals("ROLE_ANGESTELLTER")){
            benutzerRolleModellList.add(new BenutzerRolleModel(angestellter, "Angestellter"));
        } else if (rolle.getRolle().trim().equals("ROLE_MANDANT")){
            benutzerRolleModellList.add(new BenutzerRolleModel(angestellter, "Mandant"));
        }
    }
}
```

Abbildung 116 Rollen als String ins DTO laden

In der Abbildung 117 werden die Daten der Zugänge geholt und überprüft ob es ein Angestellter oder Mandant ist, danach werden die Zeichenketten „Angestellter“ oder „Mandant“ an das DTO übergeben. Die Darstellung in HTML wird in der folgenden Abbildung 117 dargestellt.

```
<th:block th:each="list: ${angestellterList}">
    <tr>
        <th class="text-truncate max-width-on-md-140 max-width-on-sm-100">[[$list.angestellter.email]]</th>
        <td class="d-none d-md-table-cell">[[$list.angestellter.vorname]]</td>
        <td class="d-none d-md-table-cell">[[$list.angestellter.nachname]]</td>
        <td class="d-none d-md-table-cell">[[$list.rolle]]</td>
```

Abbildung 117 Darstellung der Zugänge

Das Modal zum Bearbeiten der Zugänge wird mittels Post an den Controller gesendet und in die Datenbank gespeichert.

```
public String saveAngestellter(Angestellter angestellter, @RequestParam String rollen,...){
    Angestellter foundAngestellter = angestellterService.findByEmail(principal.getUsername());
    Mandant foundMandant = foundAngestellter.getMandant();
    angestellterService.saveAngestellter(angestellter, rollen, foundMandant);
    return "redirect:/dashboard/benutzer";
}
```

Abbildung 118 Angestellter in die Datenbank Speichern

Mithilfe des @RequestParam wird die Rolle von Thymeleaf übertragen und somit kann die Rolle überschrieben werden. Danach wird wie gewohnt der dazugehörige Angestellte, der zum Mandanten gehört in die Datenbank gespeichert.

7.3.1 Zahlungen

Auf der Zahlungen Seite kann der Mandant die spezifischen Daten zum Restaurant abrufen und einsehen. Auf der Seite sind zu sehen die Zahlungen, Preis pro Bestellung, Umsatz gesamt und den Umsatz des aktuellen Monates sowie den Umsatz dieses Jahres.

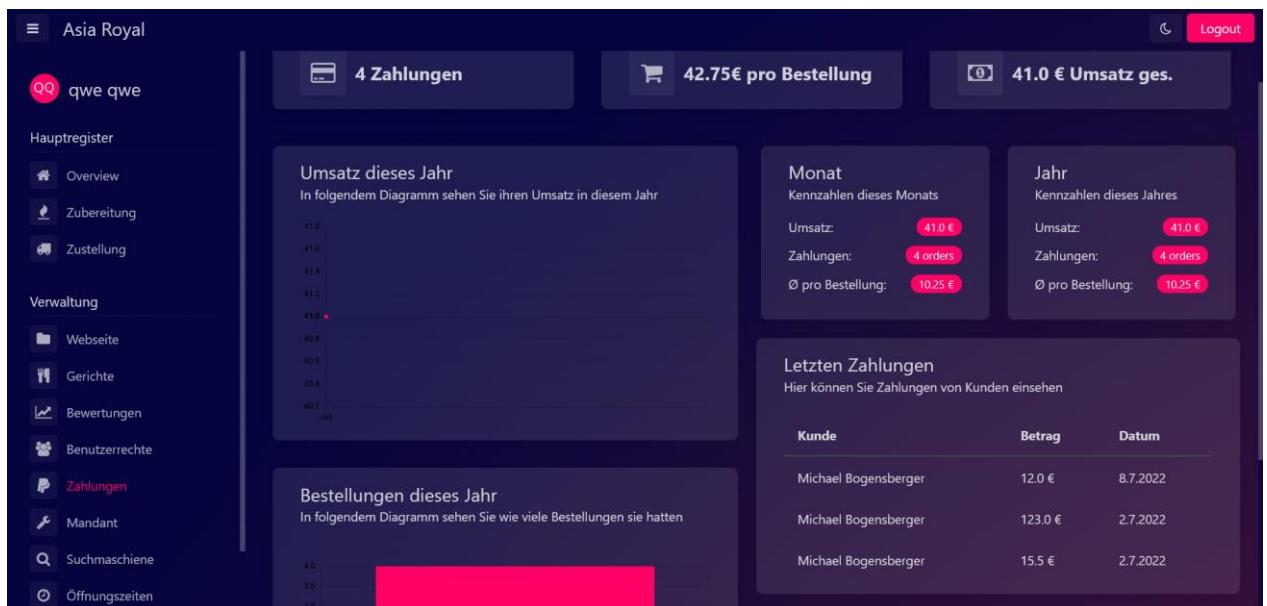


Abbildung 119 Zahlungen Seite

Da die Bestellungen mit PayPal direkt bezahlt wird sind es auch Zahlungen. Mithilfe der Funktion zahlungService.alleBestellungen(foundMandant.getId()) können alle Bestellungen vom Mandanten geladen werden. Die Listet bietet eine vorgefertigte Methode size() an. Mit der Größe der Liste können die Elemente gezählt werden die in der Liste sind. Dadurch bekommt man die Anzahl an Zahlungen.



```
List<Bestellung> alleBestellungenList = zahlungService.alleBestellungen( foundMandant.getId() );
int zahlungen = alleBestellungenList.size();
```

Abbildung 120 Zahlungen Seite Zahlungen

Den Preis pro Bestellung wird berechnet aus der Summe der Preise aller Bestellungen, durch alle Bestellungen.



```
for (Bestellung bestellung : alleBestellungenList){
    proBestellung += bestellung.getGesamtpreis();
}
proBestellung = proBestellung/(alleBestellungenList.size());
```

Abbildung 121 Preis pro Bestellung

Der Umsatz gesamt wird durch die Umsatztabelle berechnet. Dort wird jeder Umsatz vom Mandanten geholt und zusammengezählt.



```
for (Umsatz umsatz : alleUmsaetze){
    umsatzGesamt += umsatz.getUmsatz();
}
```

Abbildung 122 Gesamt Umsatz berechnen

Die Monats sowie die Jahresberechnung erfolgt durch den Codeausschnitt. Bei dem Jahresumsatz sowie Monatsumsatz werden die Umsätze im Monat bzw. im Jahr zusammengerechnet

```

if(diesenMonat != 0.0 && jahresUmsatz != 0.0){
    proBestellung = proBestellung/alleBestellungenList.size();
    durchschnittProBestellungImMonat = diesenMonat / anzahlAnOrdersImMonat;
    durchschnittProBestellungImJahr = jahresUmsatz / anzahlAnOrdersImJahr;

    if(anzahlAnOrdersImMonat != 0 && anzahlAnOrdersImJahr != 0){
        BigDecimal proBestellungBD = new BigDecimal(proBestellung).setScale(2, RoundingMode.HALF_UP);
        BigDecimal durchschnittProBestellungImMonatBD = new BigDecimal(durchschnittProBestellungImMonat).setScale(2, RoundingMode.HALF_UP);
        BigDecimal durchschnittProBestellungImJahrBD = new BigDecimal(durchschnittProBestellungImJahr).setScale(2, RoundingMode.HALF_UP);

        proBestellung = proBestellungBD.doubleValue();
        durchschnittProBestellungImMonat = durchschnittProBestellungImMonatBD.doubleValue();
        durchschnittProBestellungImJahr = durchschnittProBestellungImJahrBD.doubleValue();
    } else {
        durchschnittProBestellungImJahr = 0.0;
        durchschnittProBestellungImMonat = 0.0;
    }
}

```

Abbildung 123 Berechnung des Monat sowie Jahresumsatz

Am Anfang werden die Daten geholt und überprüft ob der Jahresumsatz sowie der Monatsumsatz nicht 0 ist. Danach rechnet man den Monatsumsatz durch die Anzahl an Bestellungen im Monat, so erhältet man die durchschnittliche Bestellung im Monat. Dies gilt ebenso für den Jahresumsatz, Jahresumsatz durch die Anzahl an Bestellungen im Jahr. Im Anschluss werden die Werte aufgerundet. Um die letzten drei Zahlungen anzuzeigen werden die letzten Einträge aus der Datenbank geholt und ausgegeben.

```

List<Bestellung> letztenDreiBestellungen = zahlungService.letztenDreiBestellunge(foundMandant.getId());

```

Abbildung 124 Die Letzten drei Bestellungen

Um die Grafiken darzustellen, werden die Daten vom Controller an das JavaScript übergeben, mithilfe der inline Funktion von Thymeleaf.

```

<script th:inline="javascript">
/*<![CDATA[*/
var datumUmsatz = /*[$datumUmsatz]*/ null;
var valueUmsatz = /*[$valueUmsatz]*/ null;
var bestellungenImJahr = /*[$bestellungenImJahr]*/ null;
/*]]&gt;*/
</pre>

```

Abbildung 125 Übergabe der Werte an JavaScript

Um das Chart darzustellen wird in JavaScript ein neues Chart Objekt erstellt. Dort muss man die X-Achsenwerte sowie die Y-Achsenwerte als Array angeben. Da im Controller ein Assoziatives Array übergeben wird und das Chart ein Normales Array ohne Key Value paare benötigt pusht man einfach die Werte in ein normales Array.

```
● ● ●
var datumBestellung = [];
var valueBestellung = [];

Object.entries(bestellungenImJahr).forEach(([key, value]) => {
    datumBestellung.push(key);
    valueBestellung.push(value);
})
```

Abbildung 126 Werte in ein normales Array pushen

Dieses normale Array kann man benutzen für die X sowie Y Achsenwerte. Bevor man die Arrays dem Chart zuweist muss man das Array einmal umdrehen d.h vom letzten Wert anfangen, damit die Datumswerte sowie Umsatzwerte korrekt dargestellt werden. Dies kann mithilfe einer Funktion gelöst werden.

```
● ● ●
datumUmsatz = datumUmsatz.reverse();
valueUmsatz = valueUmsatz.reverse();
```

Abbildung 127 Array reverse Funktion

Nachdem müssen nur noch Minimum sowie Maximum berechnet werden. Dies ist mithilfe der Math Klasse möglich.

```
● ● ●
var umsatzMax = Math.trunc(Math.max(...valueUmsatz));
var umsatzMin = Math.trunc(Math.min(...valueUmsatz));
var bestellungMax = Math.trunc(Math.max(...valueBestellung));
```

Abbildung 128 Minimum Maximum Berechnung

In der folgenden Abbildung 129 wird das Chart erstellt, mit den eingesetzten Werten.

```
● ● ●
new Chart("umsatzChart", {
    ...
    ...
    data: {
        labels: xValues,
        ...
        datasets: [
            ...
            data: yValues
            yAxes: [
                {ticks: {min: umsatzMin, max: umsatzMax,
```

Abbildung 129 Erstellung des Charts

7.3.2 Öffnungszeit

Auf der Öffnungszeit Seite sind von Montag bis Sonntag Inputfelder, womit man seine Öffnungszeiten auf der Webseite darstellen kann.

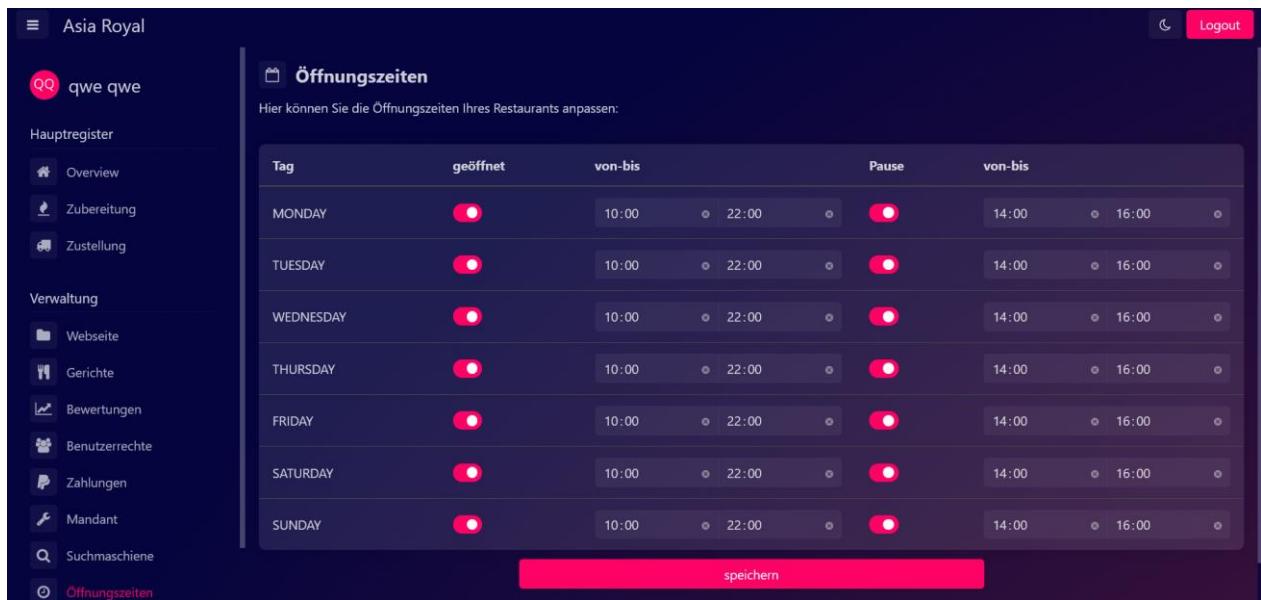


Abbildung 130 Öffnungszeiten Seite

Der Knackpunkt auf der Seite ist, dass dort null Werte übergeben werden. Auch werden null Werte in die Datenbank gespeichert. Die Ausgabe der Öffnungszeiten erfolgt durch ein DTO, welche folgende Eigenschaften beinhalten, dies wird mithilfe eines Klassendiagramm dargestellt.

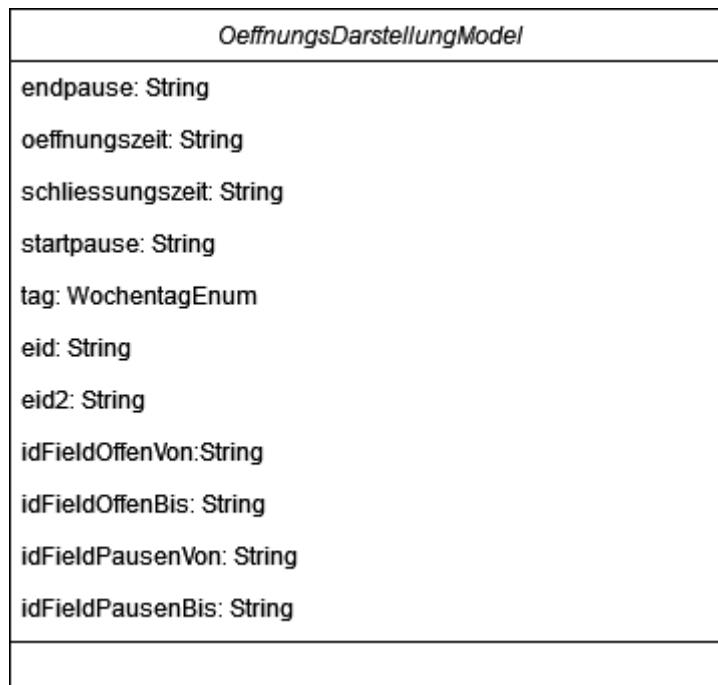


Abbildung 131 Klassendiagramm OeffnungsDarstellungModel

Es gibt in diesem Fall nicht nur ein DTO, es gibt ein zweites DTO namens OeffnungListModel. In dieser Klasse befindet sich eine List aus OeffnungsDarstellungModel. Dies wird benötigt um einzelne Zeilen dynamisch erstellen zu können, sowie das dynamische befüllen der Felder. Auch gibt es eine Umstellung, Thymleaf verwendet dort eine neue Methode. In der th:each wird ein Zähler verwendet. Der Zähler ist für die Liste zuständig, um genauer zu sein für den Listenindex.

```

    <form th:action="@{/dashboard/oeffnungszeiten/save}" method="post" th:object="${ozl}">
        <!-- Montag -->
        <tr th:each="oz,iter : ${ozl.list}">
            <td th:text="${oz.tag}">Montag</td>
            <td>
                <div class="custom-switch">
                    <input th:if="${oz.oeffnungszeit != null and oz.schliessungszeit != null}" th:name="btn1" type="checkbox"
                           th:id="${oz.eid2}" value="true" checked>
                    <input type="checkbox" th:id="${oz.eid2}" value="" >
                    <label th:for="${oz.eid2}"></label>
                </div>
            </td>
            <div class="input-group">
                <input th:id="${oz.idFieldOffenVon}" th:name="fn1" th:field="*"
                       list="${list[_${iter.index}_].oeffnungszeit}" type="time" class="form-control" placeholder="Von" th:value="${oz.oeffnungszeit}">
                <input th:id="${oz.idFieldOffenBis}" th:name="fn2" type="time" th:field="*"
                       list="${list[_${iter.index}_].schliessungszeit}" class="form-control" placeholder="Bis" th:value="${oz.schliessungszeit}">
            </div>
        </tr>
    </table>

```

Abbildung 132 Thymeleaf Darstellung der Oeffnungszeiten

Die einzelnen Felder werden durch das th:each erstellt. Das th:object ist zuständig für das zusammenhängen vom Objekt OeffnungListModel. th:field= setzt die Felder für das Objekt OeffnungListModel. Durch die Übergabe einer Liste kann ganz einfach die Liste vom Controller übernommen werden und wie gewohnt mit foreach die Liste durchhiteriert werden.

Bei der Speicherung von Öffnungszeiten müssen die Werte auf null überprüft werden bzw. auf die Länge der Zeichenkette. Ist die Zeichenkette 0 lang so kann ganz einfach die Öffnungszeit auf null gesetzt und in die Datenbank gespeichert werden. Sonst wird das normale Enum in die Datenbank gespeichert.

```

for (Oeffnungszeit oeffnungszeit : oeffnungszeitList){

    String oeffnungszeit = oeffnungListModel.getList().get(counter).getOeffnungszeit();
    String schliessungszeit = oeffnungListModel.getList().get(counter).getSchliessungszeit();
    String startpause = oeffnungListModel.getList().get(counter).getStartpause();
    String endpause = oeffnungListModel.getList().get(counter).getEndpause();

    if(oeffnungszeit == null || oeffnungszeit.length() == 0){
        oeffnungszeit.setOeffnungszeit(null);
    } else {
        oeffnungszeit.setOeffnungszeit(new
Time(formatter.parse(oeffnungListModel.getList().get(counter).getOeffnungszeit()).getTime())));
    }
}

```

Abbildung 133 Speicherung von Oeffnungszeiten

7.3.3 Kategorien

Bei der Kategorie kann der Mandant die passende Kategorie für das Restaurant auswählen.

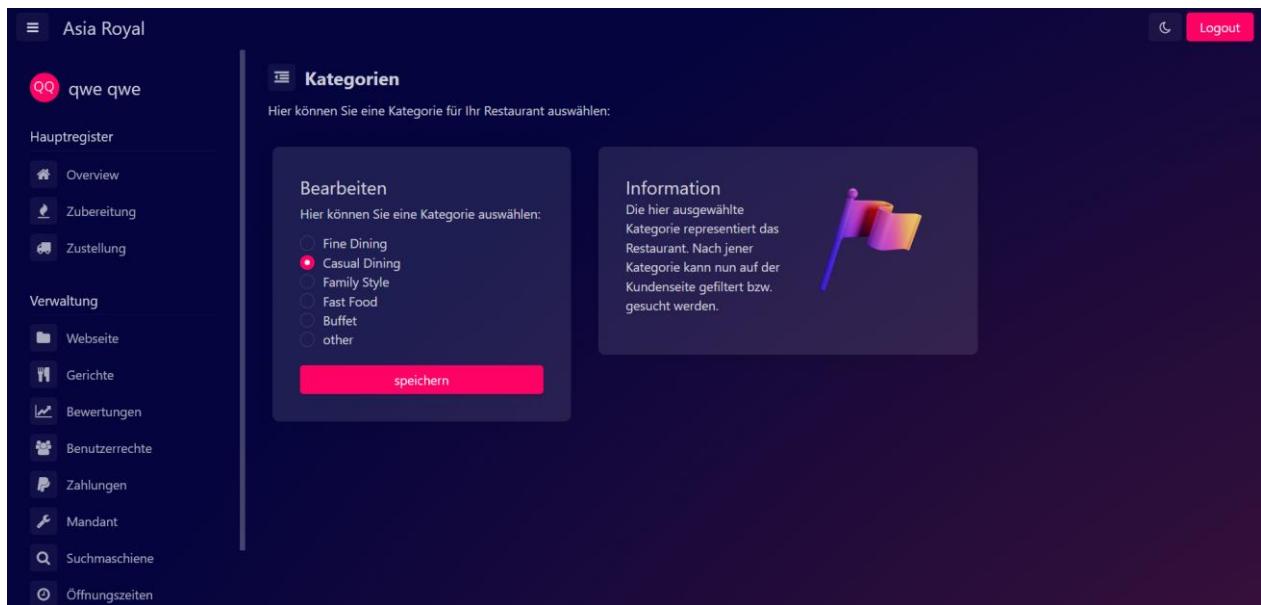


Abbildung 134 Kategorie Seite

Die Darstellung der Kategorie Seite wird mithilfe der DTO Klasse dargestellt. Die DTO Klasse besitzt nur ein Datenfeld namens B1 und ist ein KategorieEnum. Kommen null Werte aus der Datenbank, so wird automatisch die Kategorie auf other gesetzt. Sonst werden die Daten normal gesetzt.

```
KategorieModel kategorieModel = new KategorieModel();

if (foundMandant.getKategorie() == null){
    kategorieModel.setB1(KategorieEnum.OTHER);
} else {
    kategorieModel.setB1(foundMandant.getKategorie().getName());
}
```

Abbildung 135 Setzen der Kategorie

Bei der Darstellung der Button werden von der DTO Klasse die Kategorie geholt und überprüft welche der Buttons dem dazugehörigen Wert entspricht, der Button mit dem dazugehörigen Wert wird dann angekreuzt. Dies wird in Thymeleaf mit th:if geprüft.

```
<input type="radio" name="radio-set-1" id="fine-dining" checked="checked" value="FINE_DINING" th:if="${kategorie == 'FINE_DINING'}">
<input type="radio" name="radio-set-1" id="fine-dining" value="FINE_DINING">
```

Abbildung 136 Button anhaken zu dem dazugehörigen Wert

Das Speichern erfolgt durch die POST Methode, die Daten die am Controller ankommen werden mithilfe der Annotation @RequestParam ausgelesen. Vor dem Speichern wird überprüft ob die Kategorie keinen null Wert besitzt oder die Anfragen vorhanden ist, sind keine null wert vorhanden wird die Kategorie wie gewohnt in die Datenbank gespeichert.

```
● ● ●  
if(kategorie.isPresent() && kategorie != null){  
    foundMandant.getKategorie().setName(kategorie.get());  
    mandantService.save(foundMandant);  
}
```

Abbildung 137 Speichern der Kategorie

7.4 Baukastensystem

7.4.1 Aufbau des Baukastens

Da der Baukasten die Ausgabe von vielen verschiedenen Konfigurationen bewerkstelligen muss, muss dieser sehr modular implementiert werden. In Abbildung 138: Baukasten Aufbau wird der Baukasten wie sich der Baukasten aus den verschiedenen Elementen zusammenbaut näher, dargestellt.

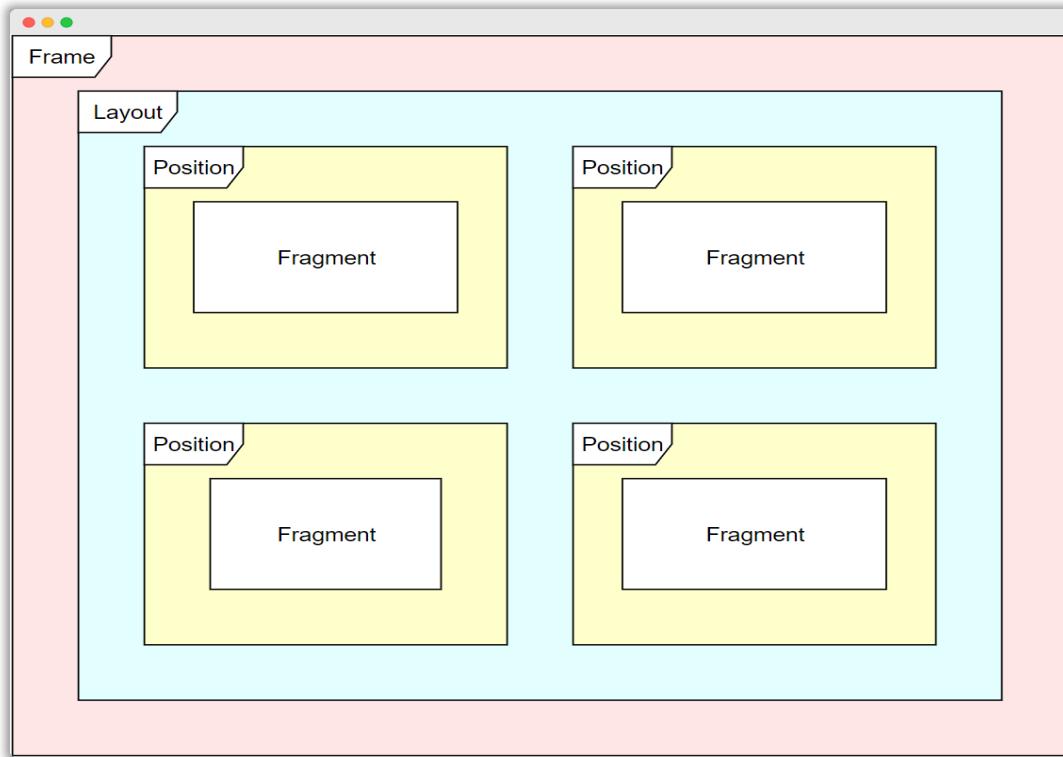


Abbildung 138: Baukasten Aufbau

Abbildung 139: Baukasten Bearbeitungsmodus stellt den Baukasten aus Sicht des Benutzers dar. Dieser ist noch leer und nicht konfiguriert.

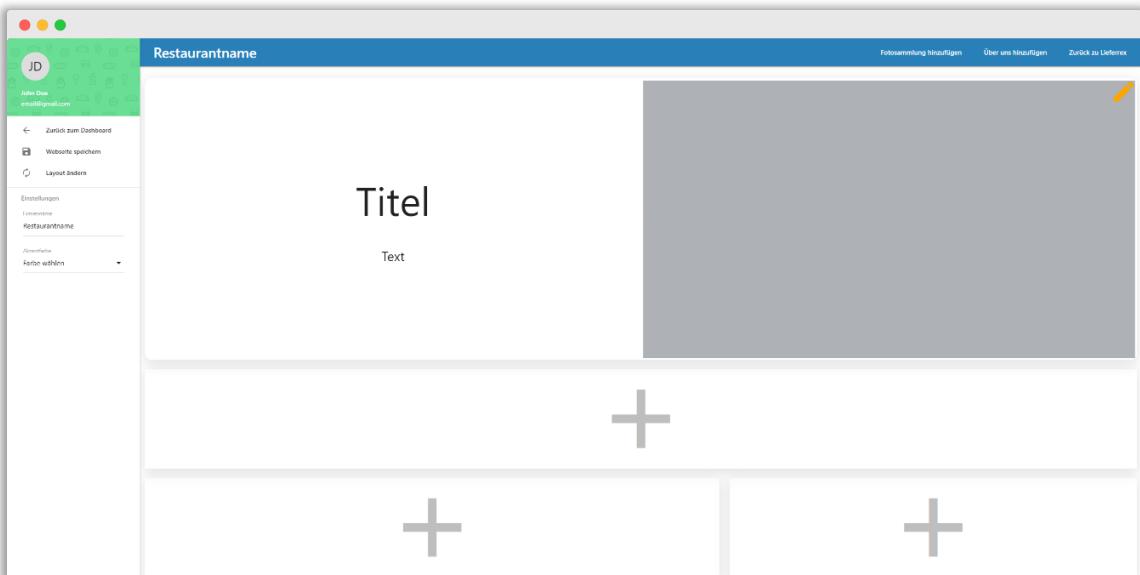


Abbildung 139: Baukasten Bearbeitungsmodus

7.4.1.1 Frame

Als Basis des Baukastens dient ein Frame. Dieser Frame ist bis auf ein paar wenige allgemeine Abschnitte (Styles, Navigationsleiste und weiteres) leer und enthält nur ein div, welches durch Thymeleaf mit einem Layout gefüllt wird.



```
<div th:replace="'baukasten/layouts/' + ${layout}"></div>
```

Abbildung 140: Baukasten Frame

Thymeleaf entscheidet mithilfe einiger Informationen des Kontrollers, in welcher der beiden Ansichten: Bearbeitungsmodus und Betrachtungsmodus sich der Benutzer befindet. Je nach Modus müssen verschiedene Dinge ausgegeben werden.

In der Ansicht eines Kunden enthält die Navigationsleiste Links, um auf die Start-, „Über uns“, oder Galerie Seite zu gelangen. Des Weiteren enthält sie eine Weiterleitung auf die Startseite von Lieferrex und einen Button, um den Warenkorb zu öffnen. In der Ansicht des Restaurants dient die Navigationsleiste einem anderen Zweck. Über diese Leiste kann ein Mandant die „Über uns“, oder Galerie Seite erstellen oder entfernen.

Um all diese verschiedenen Varianten zu ermöglichen, wird Thymeleaf verwendet. Anhand von Variablen, die Thymeleaf vom Kontroller erhält, kann entschieden werden, wo welcher Link ausgegeben werden muss. Des Weiteren sind diese Links alle relativ zur aktuellen Position auf der Webseite. Deswegen müssen auch diese von Thymeleaf ausgewertet werden.

In der Abbildung 141: Baukasten Navigationsleiste Bearbeitungsmodus wird gezeigt wie Thymeleaf entscheidet, welche Links in der Ansicht des Mandanten ausgegeben werden. Die Variablen „AboutUs“ und „Gallery“ enthalten Informationen, ob die zusätzlichen Seiten „Über uns“ und Galerie existieren und ob Links zur Erstellung oder zum Löschen dieser angezeigt wird.



```
<ul th:if="${AboutUs == null and edit == true}" class="right hide-on-med-and-down">
    <li><a href="#" name="addPageAboutUs" class="site-add">Über uns hinzufügen</a></li>
</ul>
<ul th:if="${Gallery == null and edit == true}" class="right hide-on-med-and-down">
    <li><a href="#" name="addPageGallery" class="site-add">Fotosammlung hinzufügen</a></li>
</ul>

<ul th:if="${AboutUs == true and edit == true}" class="right hide-on-med-and-down">
    <li><a href="#" name="deletePageAboutUs" class="site-remove">Über uns entfernen</a></li>
</ul>
<ul th:if="${Gallery != null and edit == true}" class="right hide-on-med-and-down">
    <li><a href="#" name="deletePageGallery" class="site-remove">Fotosammlung entfernen</a></li>
</ul>
```

Abbildung 141: Baukasten Navigationsleiste Bearbeitungsmodus

Für den Bearbeitungsmodus werden außerdem weitere Inhalte eingebunden, um alle Funktionen des Baukastens zu ermöglichen (ABBILDUNG). Diese Inhalte beinhalten Modals (Pop-Ups), JavaScript-Code und ein Menü mit allgemeinen Einstellungsmöglichkeiten.

```
<div th:replace="${edit} ? ~{baukasten/fragments/sidenav} : ~{}></div>
<div th:if="${edit}"><div th:replace="baukasten/fragments/modals"></div></div>
<script th:if="${edit}" type="text/javascript" th:src="@{/js/baukasten/script.js}"></script>
```

Abbildung 142: Baukasten Addons

Befindet man sich nicht im Bearbeitungsmodus, sondern dem Betrachtungsmodus, wird die Ansicht für den Kunden ausgegeben (Abbildung 143: Baukasten Navigationsleiste Kundenansicht). Diese Ansicht enthält Links zu den Unterseiten des Restaurants, falls diese existieren sollten, und einen Button zum Öffnen des Warenkorbs.

```
<ul th:if="${AboutUs != null and edit == null}" class="right hide-on-med-and-down">
    <li><a th:href="${AboutUsLink != null ? AboutUsLink : './AboutUs/'}">Über uns</a></li>
</ul>
<ul th:if="${Gallery != null and edit == null}" class="right hide-on-med-and-down">
    <li><a th:href="${GalleryLink != null ? GalleryLink : './Gallery/'}">Fotosammlung</a></li>
</ul>
<ul th:if="${edit == null}" class="right hide-on-med-and-down">
    <li><a href="#" onclick="openWarenkorb()">Warenkorb</a></li>
</ul>
```

Abbildung 143: Baukasten Navigationsleiste Kundenansicht

Die mobile Ansicht der Navigationsleiste sieht anders aus, enthält aber die gleiche Logik.

7.4.1.2 Layouts

Zur Auswahl für das Layout gibt es vier Varianten, aus denen das Restaurant wählen kann. Jedes Layout hat eine andere Struktur und Anzahl von Modulen. Ein Layout enthält die verschiedenen Positionen, die mit Modulen gefüllt sind. Die erste Position jedes Layouts ist fest definiert als Header. Jedes Layout hat einen eigenen Header, der immer gegeben sein muss. Die Positionen werden in Zeilen und Spalten aufgeteilt. So ist die Position „r1c1“ in der ersten Zeile und Spalte des Layouts.

Bei einer Ausgabe einer solchen Position gibt es drei Möglichkeiten:

1. Fragment existiert und wird ausgegeben
2. Fragment existiert nicht und der Bearbeitungsmodus ist aktiv
3. Fragment existiert nicht und der Betrachtungsmodus ist aktiv

Im ersten Fall existiert bereits ein Modul und dieses wird mit Zusatzinformationen (Inhalt, Position und Bearbeitungsmodus) ausgegeben. Im zweiten Fall existiert noch kein Modul an dieser Position und es muss überprüft werden, ob sich der Benutzer aktuell im Bearbeitungs- oder Betrachtungsmodus befindet. Im zweiten Fall, dem Bearbeitungsmodus wird ein Platzhalterfragment angezeigt, über welches das Restaurant ein neues Modul anlegen kann. Im letzten Falle, dass der Betrachtungsmodus aktiv ist, wird nichts ausgegeben.

In Abbildung 144: Baukasten Ausgabe eines Fragments wird die Ausgabe eines dynamischen Moduls veranschaulicht. Hier wird je nach Fragmenttyp das richtige Modul geladen.

```


Abbildung 144: Baukasten Ausgabe eines Fragments



Bei der Ausgabe des festgelegten Header-Moduls wird auf den entsprechenden Header des Layouts verwiesen und dieser mit den Informationen befüllt (ABBILDUNG).



```


Abbildung 145: Baukasten Ausgabe eines Headers

7.4.1.3 Fragmente

Fragmente gibt es in vielen Formen. Jedes Fragment ist für die Aus- oder Eingabe verschiedener Daten zuständig. In Tabelle 13: Baukasten Fragment-Typen werden die verschiedenen Typen mit Funktion aufgelistet.

Fragmenttyp	Funktion
Text	Ausgabe eines Titels und einem Text.
Image	Ausgabe eines Titels und einem Bild.
Header	Kopfbereich der Webseite mit Titel, Text und Bild.
Speisekarte	Bestellformular/Menükarte. Listet alle Gerichte des Restaurants. Über dieses Fragment kann ein Kunde bestellen.
Google-Maps	Ausgabe eines Titels und einer Google-Map des Restaurant-Standorts.
Kontaktinformationen	Stellt die Kontaktinformationen (Telefonnummer, E-Mail und Adresse) des Mandanten dar.
Öffnungszeiten	Ausgabe einer Liste der Öffnungszeiten des Restaurants.
Add	Fragment zur Bedienung des Baukastens. Wird verwendet, um ein neues Modul einzufügen.

Tabelle 13: Baukasten Fragment-Typen

Jedes Fragment hat den gleichen Aufbau, nur der Inhalt ändert sich je nach Typ. In Abbildung 146: Baukasten Text-Fragment wird als Beispiel das einfache Text-Fragment gezeigt. Alle Fragmente, mit Ausnahme des Header-Fragments hat außerdem einen Knopf, der nur im Bearbeitungsmodus angezeigt wird (wieder über die „edit“ Variable ausgewertet), um ein bestehendes Modul zu löschen und mit einem anderen auszutauschen. Das Header-Fragment kann nicht gelöscht, aber angepasst werden.

Heim

4


```


```



```
<div th:fragment="text(content, edit, position)" class="fragment-text br-16">
    <i th:if="${edit}" th:name="${position + '-' + content.fragmenttype.type}"
       class="fragment-delete material-icons">delete_forever</i>
    <div>
        <h2 th:text="${content.fragmenttext.titel}"></h2>
        <p th:text="${content.fragmenttext.text}"></p>
    </div>
</div>
```

Abbildung 146: Baukasten Text-Fragment

Das Image-Fragment sieht ähnlich aus, der Text wurde aber mit einem Bild ausgetauscht. Hierbei muss darauf geachtet werden, dass vom Kontroller kein Pfad zum Bild, sondern ein Base64-String übergeben wird. Um diesen String als Bild darzustellen, muss in der „src“ des Bildes darauf geachtet werden (Abbildung 147: Baukasten Image-Fragment).



```
<div th:fragment="image(content, edit, position)" class="fragment-image br-16">
    <i th:if="${edit}" th:name="${position + '-' + content.fragmenttype.type}"
       class="fragment-delete material-icons">delete_forever</i>
    <div>
        <div class="row" style="text-align: center;">
            <h1 class="mob-text-mod" th:text="${content.fragmentimage.titel}"></h1>
        </div>
        
    </div>
</div>
```

Abbildung 147: Baukasten Image-Fragment

Das Fragment für die Speisekarte fügt nicht nur eine einfache Oberfläche wie die anderen Fragmente hinzu, sondern aus weiter Funktionalität, die für den Bestellvorgang benötigt wird. Das Speisekarte-Fragment liefern JavaScript-Code zur Verwaltung des Warenkorbs und einer Oberfläche für den Warenkorb mit.

In Abbildung 148: Baukasten - Gericht hinzufügen wird der Ablauf dargestellt, wie ein Gericht dem Warenkorb hinzugefügt wird. Zuerst wird der Warenkorb Cookie ausgelesen, ist dieser leer, wird er neu angelegt. Danach wird aus den verschiedenen Datenfeldern des Gerichtes der Input des Kunden ausgelesen. Es wird geprüft, ob ein Gericht mit gleicher ID und Namen bereits im Warenkorb ist. Wenn ja wird die Anzahl dieses Gerichtes erhöht. Wenn nein wird es neu erstellt. Das neue Cookie wird gespeichert und die Darstellung auf der Webseite neu geladen.

```

$( ".buyButton" ).click( function () {

    var added = false;
    var mandantID = /*[${restaurantID}]*/ '0';
    var id = $(this).attr("id").split("-")[1];

    if (Cookies.get("warenkorb") == null) {
        var warenkorb = [];
    } else {
        var warenkorb = window.JSON.parse(Cookies.get("warenkorb"));
    }

    gericht = {
        id: id,
        anmerkung: $("#anmerkung-" + id).val(),
        anzahl: $("#anzahl-" + id).val(),
        name: $("#name-" + id).text(),
        madant: mandantID
    };

    warenkorb.forEach(element => {
        if (element["id"] == gericht["id"] && element["anmerkung"] == gericht["anmerkung"]) {
            element["anzahl"] = Number(element["anzahl"]) + Number(gericht["anzahl"]);
            added = true;
        }
    });

    if (!added) { warenkorb.push(gericht); }

    M.toast({ html: 'Gericht hinzugefügt!' })
    Cookies.set("warenkorb", window.JSON.stringify(warenkorb));
    updateWarenkorb(warenkorb);
});

});

```

Abbildung 148: Baukasten - Gericht hinzufügen

Die „updateWarenkorb“ Funktion (Abbildung 149: Baukasten - Warenkorb Ausgabe) liest das Cookie aus und erstellt eine Tabelle daraus. Diese Tabelle wird im Warenkorb angezeigt. Jedes Gericht im Warenkorb wird als Tabellenzeile ausgegeben.

```

function updateWarenkorb(warenkorb) {
    var data = "";
    warenkorb.forEach(element => {

        data += '\
        <tr>\
            <td>' + element["name"] + '</td>\
            <td>' + element["anmerkung"] + '</td>\
            <td>' + element["anzahl"] + '</td>\
            <td onclick="removeGericht(' + element["id"] + ', \'' + element["anmerkung"] + \
                '\')"><a href="#">Entfernen</a></td>\
        </tr>\
    ';
    });

    $("#"waren").html(data)
}

```

Abbildung 149: Baukasten - Warenkorb Ausgabe

Um ein Gericht wieder aus dem Warenkorb zu löschen, wird die Funktion „removeGericht“ ausgeführt (Abbildung 150: Baukasten - Gericht entfernen). Das Gericht mit gleicher ID und Anmerkung wird aus der Liste gelöscht, das Cookie gespeichert und der Warenkorb aktualisiert.

```
● ● ●

function removeGericht(id, anmerkung) {
    var warenkorb = window.JSON.parse(Cookies.get("warenkorb"));
    var gerichtToRemove = "";

    warenkorb.forEach(element => {
        if (element["id"] == id && element["anmerkung"] == anmerkung) {
            gerichtToRemove = element;
        };
    });
    warenkorb.splice(warenkorb.indexOf(gerichtToRemove), 1)

    M.toast({ html: 'Gericht entfernt!' })

    Cookies.set("warenkorb", window.JSON.stringify(warenkorb));
    updateWarenkorb(warenkorb);
}
```

Abbildung 150: Baukasten - Gericht entfernen

7.4.1.4 Modals

Modals von Materialize sind kleine Pop-Up-Fenster, die sich über der Webseite öffnen. Diese werden auch beim Baukasten verwendet. Sie sind im Rahmen des Baukastens super geeignet, um vom Restaurant Eingaben für das Hinzufügen von Modulen entgegenzunehmen. Beim Erstellen eines neuen Modules öffnet sich zuerst ein Modal, das alle verfügbaren Fragmente auflistet. Aus diesen kann sich der Mandant eines aussuchen. Jeder Fragmenttyp hat ein eigenes Modal, über das Informationen für das Modul eingegeben werden können. In Abbildung 151: Baukasten - Text-Fragment Modal wird der Aufbau eines solchen Modals veranschaulicht. Dieses Modal wird verwendet, um ein Text-Fragment zu erstellen.

```
● ● ●

<div id="addText" class="modal">
    <div class="modal-content">
        <h4>Titel mit Text Modul hinzufügen</h4>

        <div class="row mb-0">
            <div class="input-field col s12">
                <i class="material-icons prefix">title</i>
                <input id="addTextTitle" type="text" data-length="100">
                <label for="addTextTitle">Überschrift</label>
            </div>
        </div>

        <div class="row mb-0">
            <div class="input-field col s12">
                <i class="material-icons prefix">title</i>
                <input id="addTextText" type="text" data-length="100">
                <label for="addTextText">Text</label>
            </div>
        </div>

    </div>
    <div class="modal-footer">
        <a href="#" class="modal-close waves-effect waves-red btn-flat">Abbrechen</a>
        <a href="#" id="saveText" class="saveFragment modal-close waves-effect waves-green btn-flat">Speichern</a>
    </div>
</div>
```

Abbildung 151: Baukasten - Text-Fragment Modal

Abbildung 152: Baukasten - Text-Fragment Modal Bearbeitungsansicht zeigt, wie das fertige Modal auf der Webseite aussieht. Es gibt bei diesem Typ zwei Texteingabefelder, für Titel und Text. Auch gibt es einen Button, um die Eingabe zu speichern oder abzubrechen.

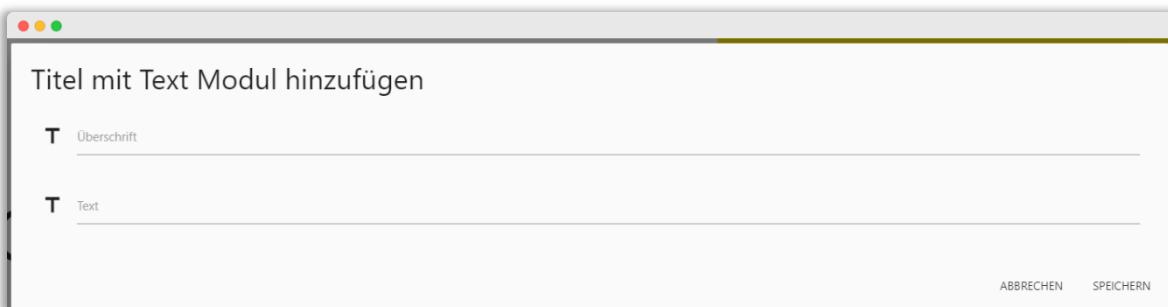


Abbildung 152: Baukasten - Text-Fragment Modal Bearbeitungsansicht

7.4.2 Baukasten Frontend Funktionsweise

Alle Funktionen des Baukastens im Frontend wurden durch JavaScript und JQuery umgesetzt. Alle Eingaben des Benutzers werden von JavaScript ausgewertet und verarbeitet. Änderungen werden dem Kontroller über POST-Anfragen mitgeteilt. Bei einer Änderung wird das Ergebnis vom Kontroller erhalten und angezeigt.

Der Baukasten teilt die Eingabe vom Benutzer in sieben verschiedene Aktionen ein:

1. Erstellen eines Fragmentes
2. Löschen eines Fragmentes
3. Bearbeiten des Kopfbereiches
4. Wechseln des Layouts
5. Speichern der allgemeinen Einstellungen
6. Hinzufügen von Zusatz-Seiten („Über uns“ und Galerie)
7. Entfernen der Zusatz-Seiten

Je nach Aktion müssen bestimmte Eingaben vom Restaurant verarbeitet werden. In Abbildung 153: Baukasten - Frontend Aktionen wird gezeigt, wie JavaScript die Art der Aktion auswertet. Es wird überprüft, auf welches Element der Webseite der Benutzer klickt.

```

$(document).off("click").on("click", ".fragment-add, .fragment-delete, .fragment-edit,
.layout-change, .saveSettings, .site-add, .site-remove", function() {

    if($(this).attr("class").indexOf("fragment-add") >= 0){
        ...
    } else if ($(this).attr("class").indexOf("fragment-delete") >= 0) {
        ...
    } else if ($(this).attr("class").indexOf("layout-change") >= 0) {
        ...
    } else if ($(this).attr("class").indexOf("saveSettings") >= 0) {
        ...
    } else if ($(this).attr("class").indexOf("site-add") >= 0) {
        ...
    } else if ($(this).attr("class").indexOf("site-remove") >= 0) {
        ...
    } else {
        ...
    }
})

```

Abbildung 153: Baukasten - Frontend Aktionen

7.4.2.1 Erstellen eines Fragmentes

Im ersten Fall wird ein neues Modul vom Restaurant angelegt. Es wird das „selector“-Modal geöffnet, in dem sich alle verfügbaren Fragment-Typen befinden. Es wird außerdem ausgewertet, an welcher Position sich das neue Modul befindet und welchen Typ dieses hat. Es wird das entsprechende Modal für den Typen geöffnet (Abbildung 154: Baukasten - Modal öffnen).

```

var pos = $(this).attr('name');
var name;
$('#selector').modal('open');

// Open specific modal for fragment
$(".AddFragment").off("click").click(function() {
    $('#selector').modal('close');
    name = '#' + $(this).attr('name')
    $(name).modal('open');
});

```

Abbildung 154: Baukasten - Modal öffnen

Es können nun die entsprechenden Daten vom Benutzer eingegeben werden (Text, Bilder etc.). Beim Speichern wird wieder nach Typ unterschieden. In Abbildung 155: Baukasten - Eingabe Auswertung wird dargestellt, wie der Ablauf beim Speichern aussieht. Hier werden als Beispiel die die Speicherung eines Text-Fragments gezeigt. Die ermittelten Daten werden in einem „formData“-Objekt gespeichert.

```
$( '.saveFragment' ).off("click").click(function() {  
  
    var formData = new FormData();  
    switch ($(this).attr('id')) {  
        case "saveText":  
            formData.append( 'data', JSON.stringify({  
                "title": $('#addTextTitle').val(),  
                "text": $('#addTextText').val(),  
                "position": pos,  
                "type": "text"  
            }));  
            break;  
        ...  
    }  
});
```

Abbildung 155: Baukasten - Eingabe Auswertung

Die Daten können nun an das Backend übergeben werden. Die geschieht über eine POST-Request (Abbildung 156: Baukasten - Post-Request Speicherung). Ist die Anfrage erfolgreich, wird als Antwort des Kontrollers das fertige Modul zurückgeliefert. Dieses kann von JavaScript in der aktuellen Position eingefügt werden.

```
$.ajax({  
    type: "POST",  
    url: "./module/save",  
    async: false,  
    enctype: 'multipart/form-data',  
    processData: false,  
    contentType: false,  
    data: formData,  
    success: function ( data ) {  
        setTimeout( function () { $('#' + pos).html(data); }, 1000 )  
    }  
});
```

Abbildung 156: Baukasten - Post-Request Speicherung

7.4.2.2 Speichern der allgemeinen Einstellungen

Beim Speichern der allgemeinen Einstellungen werden die Eingaben des Users ermittelt und wieder per POST-Request dem Kontroller übergeben (Abbildung 157: Baukasten - Speicherung der allgemeinen Einstellungen). Um die Änderungen anzuzeigen, wird bei erfolgreicher Request die Seite neu geladen.

```

    formData.append('data', JSON.stringify({
      "restaurantName": $("#firmenname").val(),
      "color": ($("#color").val() || ""),
      "layout": (layout || "")
    }));

    $.ajax({
      type: "POST",
      url: "./update",
      async: false,
      processData: false,
      contentType: false,
      data: formData,
      success: function ( data ) {
        setTimeout( function () { location.reload();}, 1000 )
      }
    });
  
```

Abbildung 157: Baukasten - Speicherung der allgemeinen Einstellungen

7.4.3 Baukasten Backend Funktionsweise

Das Backend des Baukastens nimmt alle Anfragen des Frontends entgegen und verarbeitet diese. Es übernimmt alle Aufgaben, die vom Baukasten kommen und liefert außerdem die fertige Webseite für Kunden aus. Das Backend für den Baukasten hat verschiedene Routen, die verschiedene Aktionen des Benutzers übernehmen.

7.4.3.1 Baukasten Ausgabe

Ist ein Mandant oder ein Angestellter eines Mandanten angemeldet, kann dieser den Baukasten über die URL „/baukasten/“ öffnen. Der Kontroller bestimmt den Mandanten abhängig vom angemeldeten Benutzer. Es wird ein Model erstellt, das mit allen relevanten Informationen befüllt wird, die Thymeleaf braucht. In Abbildung 158: Baukasten - Model Attribute werden die allgemeinen Informationen dem Model hinzugefügt. Anhand der Variable „layout“ kann Thymeleaf das richtige Layout ausgeben. „edit“ legt fest, dass alle Elemente zur Bearbeitung der Seite geladen werden.

```

model.addAttribute("layout", mandant.getLayout().getName());
model.addAttribute("edit", true);
model.addAttribute("restaurantName", mandant.getFirmenname());
model.addAttribute("color", mandant.getAkzentFarbe());
  
```

Abbildung 158: Baukasten - Model Attribute

Im nächsten Schritt wird überprüft, ob bereits ein Header-Fragment existiert. Wenn nein, muss dieses angelegt werden. Danach wird das Model mit allen bereits bestehenden Fragmenten befüllt. Hier wird der Fragment-Typ bestimmt und die entsprechenden Informationen für diesen Typ dem Model übergeben. Für das Kontakt-Fragment müssen beispielsweise alle Kontaktinformationen (Telefonnummer, E-Mail, Adresse etc.) ausgelesen werden. Hierfür wird die Hilfsfunktion „allFragments“ verwendet. Die Abbildung 159: Baukasten - Model mit Fragmenten befüllen zeigt einen Ausschnitt dieser Funktion.

```

public void allFragments(List<Fragment> fragments, Model model, Mandant mandant) {
    for (Fragment fragment : fragments) {
        model.addAttribute(fragment.getPosition().getName(), fragment);

        if (fragment.getFragmenttype().getType().equals("karte")) {
            model.addAttribute("gerichte", mandant.getGerichte());
        } else if (fragment.getFragmenttype().getType().equals("kontakt")) {
            model.addAttribute("kontaktstrasse", mandant.getStrasse());
            model.addAttribute("kontaktnummer", mandant.getHausnummer());
            model.addAttribute("kontakttort", mandant.getOrt());
            model.addAttribute("kontaktplz", mandant.getPlz());
            model.addAttribute("kontaktmail", mandant.getEmail());
            model.addAttribute("kontakttelefon", mandant.getTelefonnummer());
        } else if (fragment.getFragmenttype().getType().equals("image")) {
            if (fragment.getFragmentimage().getImageBlob() != null) {
                model.addAttribute(fragment.getPosition().getName() + "image",
                    new String((fragment.getFragmentimage().getImageBlob())));
            }
        } else if (fragment.getFragmenttype().getType().contains("header")) {
            ...
        }
        ...
    }
}

```

Abbildung 159: Baukasten - Model mit Fragmenten befüllen

Ist das Model mit allen Informationen befüllt, liefert der Kontroller dem Frontend den Frame aus. Dieser wird von Thymeleaf mit Layout und Fragmenten befüllt.

7.4.3.2 Restaurant Ausgabe

Die Ausgabe des Restaurants aus der Sicht des Kunden ist sehr ähnlich zur Ausgabe des Baukastens. Auf die Variable „edit“ kann hier verzichtet werden, da ein Kunde die Seite nicht bearbeiten kann. Diese Ansicht kann über die URL „/restaurant/{ID}“ aufgerufen werden (Restaurantseite wird meist über die Suche geöffnet).

Eine zusätzliche Funktionalität, die dieses Mapping hat, ist die Verwaltung der Seitenaufrufe. Diese wird in Abbildung 160: Baukasten – Seitenaufrufe veranschaulicht. Immer wenn ein Kunde die Seite betritt, wird ein Zähler erhöht. Dieser Zähler gilt immer für das aktuelle Monat und kann vom Mandanten im Dashboard eingesehen werden. Hiermit kann der Mandant feststellen, wie viele Kunden seine Seite betreten. Es wird überprüft, ob bereits ein aktueller Eintrag für diesen Mandanten für dieses Monat existiert und erstellt diesen, im Falle, dass dieser noch nicht vorhanden ist.

```

int year = Calendar.getInstance().get(Calendar.YEAR);
int month = Calendar.getInstance().get(Calendar.MONTH);
Optional<Seitenaufrufe> seitenaufrufe = seitenaufrufeRepository
    .getSeitenaufrufeByMandantAndJahrAndMonat(mandant, year, month);

if (seitenaufrufe.isPresent()) {
    seitenaufrufe.get().setAufrufe(seitenaufrufe.get().getAufrufe() + 1);
    seitenaufrufeRepository.save(seitenaufrufe.get());
} else {
    seitenaufrufeRepository.save(new Seitenaufrufe(null, month, year, 1, mandant));
}

```

Abbildung 160: Baukasten – Seitenaufrufe

7.4.3.3 Fragment speichern

Die Speicherung eines neuen Fragmentes ist die komplexeste Aufgabe des Kontrollers. Der Kontroller erhält die Informationen des neuen Moduls in Form von JSON vom Frontend. Es muss bestimmt werden, welchen Typ das neue Fragment hat. Je nach Fragment-Typ werden andere Informationen in der Datenbank gespeichert. Der Ablauf des Speicherns wird in folgender Abbildung 161: Baukasten - Fragment Speicherung Ablauf beschrieben.

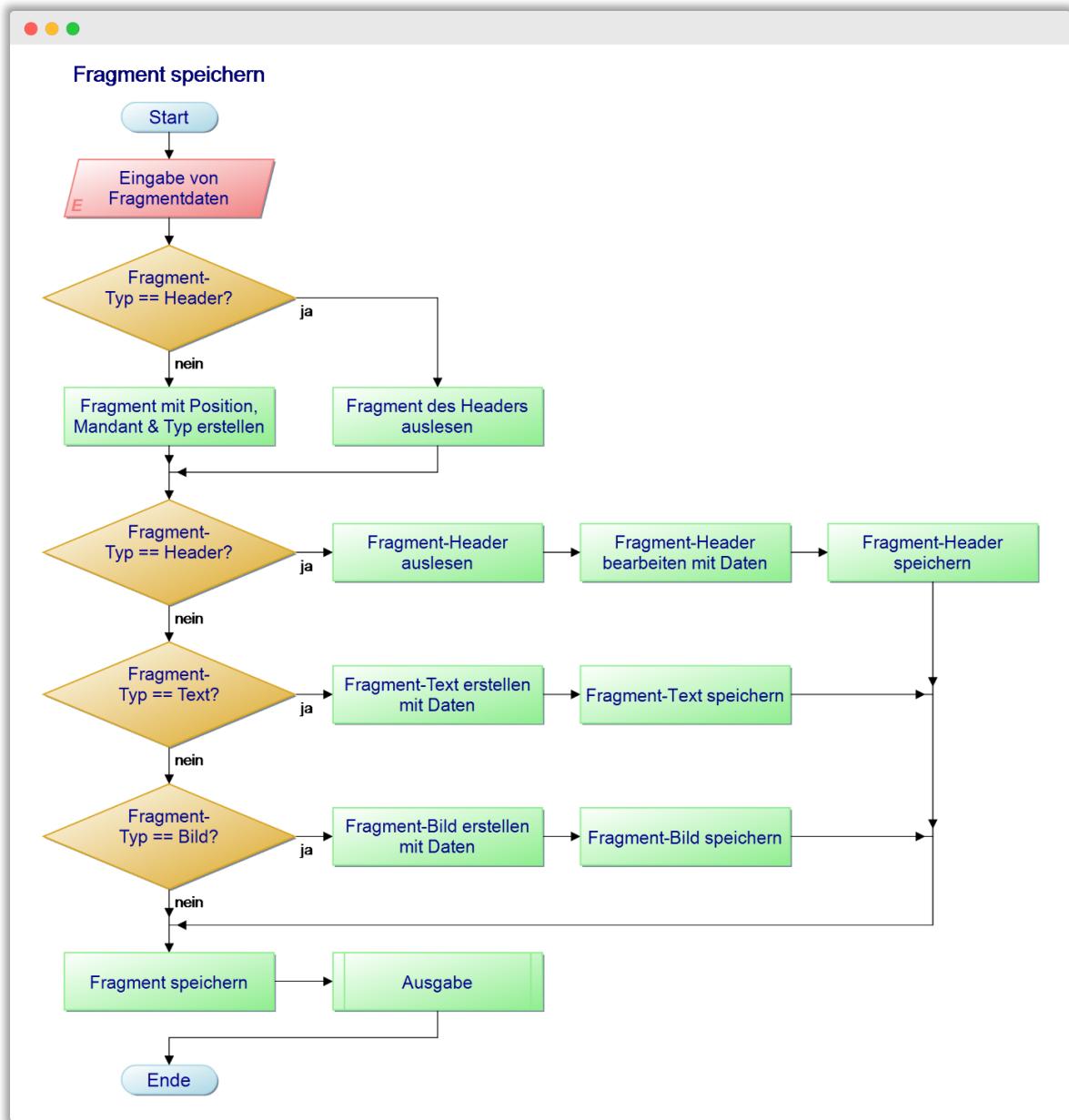


Abbildung 161: Baukasten - Fragment Speicherung Ablauf

Im Falle eines neuen Fragmentes, wird dieses komplett neu angelegt. Für jedes Modul gibt es ein Fragment-Element. Dieses enthält allgemeine Informationen, wie Position, Typ und zu welchem Mandanten es gehört (Abbildung 162: Baukasten - Ausnahme Header-Fragment).

```
● ○ ●
if (!fragmenttype.get().getType().contains("header")) {
    fragment = fragmentServiceImpl.save(
        new Fragment(null, position.get(), mandant.get(), fragmenttype.get(), null, null, null));
}
```

Abbildung 162: Baukasten - Ausnahme Header-Fragment

Für die Fragment-Typen Text, Bild und Header werden auch entsprechende Elemente angelegt, die spezifische Daten enthalten (für Text-Fragment Titel und Text, Bild-Fragment hat Titel und Bild usw.). In Abbildung 163: Baukasten - Text-Fragment Speicherung wird ein Text-Fragment erstellt. Es wird ein neues „FragmentText“-Element erstellt und gespeichert. Das zuvor erstellte Fragment wird um das „FragmentText“-Element ergänzt und ebenfalls gespeichert.

```
● ○ ●
case "text":
    FragmentText fragmentText = fragmentTextServiceImpl.save(
        new FragmentText(null, result.get("title"), result.get("text"), fragment));

    fragment.setFragmenttext(fragmentText);
    fragmentServiceImpl.save(fragment);
    break;
```

Abbildung 163: Baukasten - Text-Fragment Speicherung

Bei der Speicherung eines Bild- oder Header-Fragments werden Bilder gespeichert. Diese sind in der Datenbank als BLOB (Binary Large Object) Datentyp definiert. Um dies zu ermöglichen, muss das Bild, das der Kontroller vom Benutzer erhält in einen Base64-String umgewandelt werden. Ein Base64-Encoded wandelt die Bytes des Bildes in diesen String um. Dieser Schritt wird in Abbildung 164: Baukasten - Bild BLOB gezeigt.

```
● ○ ●
Base64.getEncoder().encode(image.get().getBytes())
```

Abbildung 164: Baukasten - Bild BLOB

Bei der Ausgabe dieses BLOBs muss dieses in einen normalen String umgewandelt werden, damit HTML dieses anzeigen kann. Aus dem Base64 String wird ein normaler Java-String erstellt (Abbildung 165: Baukasten - Image String).

```
● ○ ●
new String((fragment.getFragmentimage().getImageBlob()))
```

Abbildung 165: Baukasten - Image String

Ist der Typ des Moduls aber Header, muss dieses Modul aus der Datenbank zuerst ausgelesen werden, da dieses bereits existieren muss. Danach wird dieses mit den neuen Informationen überschrieben und wieder gespeichert.

Die restlichen Fragment-Typen (Kontaktinformationen, Öffnungszeiten, Google-Maps etc.) haben keinen spezifischen Eintrag, da alle Informationen vom Mandanten ausgelesen werden können und nicht vom Benutzer im Baukasten eingegeben werden. Der Kontroller befüllt für diese Module wieder ein Model mit allen Informationen.

Nach der erfolgreichen Speicherung eines Fragments gibt der Kontroller dieses Fragment wieder zurück, damit dieses direkt, ohne neu laden zu müssen, angezeigt werden kann. In Abbildung 166: Baukasten - Modul Rückgabe wird veranschaulicht, wie der Kontroller das Model mit den Informationen des neuen Fragments befüllt. Hier wird nicht der gesamte Frame zurückgegeben, sondern nur das Modul. Deswegen muss auch die Position übergeben werden, damit das Frontend weiß, wo das neue Modul eingesetzt werden muss.

```
● ● ●
model.addAttribute("content", fragment);
model.addAttribute("edit", true);
model.addAttribute("position", position.getName());
return "baukasten/fragments/modules/" + fragment.getFragmenttype().getType();
```

Abbildung 166: Baukasten - Modul Rückgabe

7.4.3.4 Fragment löschen

Fragmente können auch wieder vom Mandanten gelöscht werden. Gelöscht werden können alle Fragmente, mit Ausnahme des Header-Fragmentes. Im Falle eines Text- oder Bild-Fragments müssen zuerst die spezifischen Einträge entfernt werden. Danach kann der allgemeine Fragment-Eintrag gelöscht werden. Der Kontroller gibt anschließend das Modul zum Hinzufügen eines neuen Elements an entsprechender Position zurück. Ein Teil dieses Ablaufs wird in Abbildung 167: Baukasten - Fragment löschem gezeigt.

```
● ● ●
switch (fragmenttype.getType()) {
    case "text":
        Optional<FragmentText> fragmentText = fragmentTextServiceImpl
            .findFragmenttextByFragment_id(fragment.getId());
        fragmentTextServiceImpl.delete(fragmentText.get());
        break;

    case "image":
        ...
        break;
}

fragmentServiceImpl.delete(fragment);
model.addAttribute("position", position.getName());
return "baukasten/fragments/modules/add";
```

Abbildung 167: Baukasten - Fragment löschem

7.4.3.5 Allgemeine Einstellungen speichern

Vom Benutzer können auch allgemeine Einstellungen zur Seite bearbeitet werden. Diese Einstellungen beinhalten eine Akzentfarbe (Farbe von Buttons und der Navigationsleiste), den Restaurantnamen und das verwendete Layout. Die Einstellungen für Farbe und Namen können einfach am Mandanten angepasst werden. Es wird vorerst überprüft, ob der Benutzer Werte für diese Einstellungen

eingegeben hat. Wenn ja werden diese gespeichert (Abbildung 168: Baukasten - Allgemeine Einstellungen).

```
● ● ●

if (!result.get("color").isEmpty()) {
    mandant.get().setAkzentFarbe(result.get("color"));
}

if (!result.get("restaurantName").isEmpty()) {
    mandant.get().setFirmenname(result.get("restaurantName"));
}
```

Abbildung 168: Baukasten - Allgemeine Einstellungen

Beim Ändern des Layouts werden alle bereits bestehenden Fragmente, mit Ausnahme des Headers, gelöscht (Abbildung 169: Baukasten - Layout ändern). Dies geschieht, da Positionen und Anzahl der Positionen zwischen den verschiedenen Layouts unterschiedlich sind und es ansonsten zu Problemen kommen kann.

```
● ● ●

if (layout.isPresent()) {
    mandant.get().getFragmente().stream().forEach(item -> {
        if (item.getFragmenttype().getType().equals("text")) {
            fragmentTextServiceImpl.delete(item.getFragmenttext());
        }
        if (item.getFragmenttype().getType().equals("image")) {
            fragmentImageServiceImpl.delete(item.getFragmentimage());
        }

        if (!item.getFragmenttype().getType().contains("header")) {
            fragmentServiceImpl.delete(item);
        }
    });
}
```

Abbildung 169: Baukasten - Layout ändern

Nach erfolgreicher Änderung aller Einstellungen wird der Baukasten neu ausgegeben, um alle Änderungen zu zeigen.

7.4.3.6 Zusatz-Seite speichern

Der Benutzer kann zwei Zusatz-Seiten erstellen. Beim Kontroller wird nach dem Typ dieser Seite ein anderer Eintrag in der Datenbank erstellt (Abbildung 170: Baukasten - Zusatz-Seite speichern). Die „Über uns“-Seite enthält einen Titel, zwei Texte und zwei Bilder. Die Galerie-Seite hat einen Titel und fünf Bilder.

```

switch (result.get("type")) {
    case "AboutUs":
        aboutUsServiceImpl.save(
            new AboutUs(null, result.get("title"), result.get("textOne"),
            result.get("textTwo"), Base64.getEncoder().encode(image.get(0).getBytes()),
            Base64.getEncoder().encode(image.get(1).getBytes()), mandant.get()));
        break;
    case "Gallery":
        galleryServiceImpl.save(
            new Gallery(null, result.get("title"),
            Base64.getEncoder().encode(image.get(0).getBytes()),
            Base64.getEncoder().encode(image.get(1).getBytes()),
            Base64.getEncoder().encode(image.get(2).getBytes()),
            Base64.getEncoder().encode(image.get(3).getBytes()),
            Base64.getEncoder().encode(image.get(4).getBytes()),
            mandant.get()));
        break;
}

```

Abbildung 170: Baukasten - Zusatz-Seite speichern

7.4.3.7 Zusatz-Seite löschen

Die Seiten können auch wieder vom Benutzer entfernt werden. Dies wird in Abbildung 171: Baukasten - Zusatz-Seite entfernen gezeigt.

```

switch (result.get("type")) {
    case "AboutUs":
        aboutUsServiceImpl.delete(aboutUsServiceImpl.findAboutusByMandant(mandant.get()));
        break;

    case "Gallery":
        galleryServiceImpl.delete(galleryServiceImpl.findGalleryByMandant(mandant.get()));
        break;

    default:
        break;
}

```

Abbildung 171: Baukasten - Zusatz-Seite entfernen

7.4.3.8 Zusatz-Seite Ausgabe

Wurden diese Seiten erstellt, kann ein Kunde diese über die URL „/restaurant/{restaurant}/{type}“ erreichen. Der Typ sagt aus, welche er beiden Seiten betrachtet wird. Neben den allgemeinen Werten setzt der Kontroller unabhängig vom Typ weitere Werte im Model fest. Die Variable „backLink“ wird auf diesen beiden Seiten verwendet, um wieder auf die Startseite des Restaurants gelangen zu können.

Der Kontroller befüllt das Model je nach Typ mit den nötigen Werten. In der Abbildung 172: Baukasten - Zusatz-Seite Ausgabe wird der Teil des Kontrollers dargestellt, welcher die „Über uns“-Seite ausgibt. Alle Informationen dieser Seite werden aus der Datenbank ausgelesen und dem Model übergeben. Des Weiteren wird geprüft, ob die andere jeweils Zusatz-Seite existiert. Wenn ja, werden dem Model weitere Attribute gegeben. Diese Attribute zeigen, dass die jeweils andere Seite vorhanden ist, und über welchen Link diese aufrufbar ist.

Der Kontroller gibt am Ende auch einen leeren Frame zurück. Dieser wird aber nicht von einem Layout gefüllt, sondern von der entsprechenden Seite. Diese Seite ist wie ein Layout aufgebaut, jedoch ohne dynamische Positionen.

```
switch (type) {  
    case "AboutUs":  
        AboutUs aboutUs = mandant.get().getAboutus();  
        model.addAttribute("layout", "AboutUs");  
        model.addAttribute("title", aboutUs.getTitle());  
        model.addAttribute("textOne", aboutUs.getTextOne());  
        model.addAttribute("textTwo", aboutUs.getTextTwo());  
        model.addAttribute("imageOne", new String(aboutUs.getImageBlobOne()));  
        model.addAttribute("imageTwo", new String(aboutUs.getImageBlobTwo()));  
  
        if (galleryServiceImpl.findGalleryByMandant(mandant.get()).isPresent()) {  
            model.addAttribute("Gallery", true);  
            model.addAttribute("GalleryLink", "../Gallery/");  
        }  
  
    break;  
}
```

Abbildung 172: Baukasten - Zusatz-Seite Ausgabe

8 Deployment

Das Projekt wird klassisch in der Cloud gehostet. Dafür wurde der Dienst Heroku verwendet. Bei jedem Push auf das GitHub Repository wird durch Continuous Integration beziehungsweise mithilfe von GitHub Actions und Heroku ein neues Build erstellt und dieses direkt in der Cloud gehostet.

Dazu wurde zunächst ein Workflow mithilfe von GitHub Actions erstellt, welcher das Projekt baut. Der Workflow dient dazu zuerst sicher zu gehen, dass das Projekt auch ohne Probleme gebaut werden kann.

Wie in der Abbildung zu sehen ist wird der Workflow bei jedem push durchgeführt. GitHub baut im Falle eines Pushes das Projekt und geht so sicher, dass das Projekt auch einwandfrei funktioniert.

Des Weiteren bedarf es einer Procfile. Darin findet man jenen Befehl, den Heroku beim Empfangen eines Pushes ausführt. Heroku führt nun also das Projekt aus. Dabei verwendet Heroku das produktivbetrieb erstellt. Da verschiedene APIs verwendet werden und eine Verbindung zu einer Datenbank hergestellt werden muss und die API-Keys sowie die Passwörter nicht öffentlich zugänglich sein sollen werden hier Platzhalter beziehungsweise Variablen eingesetzt die Heroku dann automatisch befüllt.

```
name: Java CI with Maven

on:
  push:
    branches: [ "main" ]

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v3
      - name: Set up JDK 11
        uses: actions/setup-java@v3
        with:
          java-version: '11'
          distribution: 'temurin'
          cache: maven
      - name: Build with Maven
        run: mvn --batch-mode --update-snapshots verify
```

Abbildung 173: maven.yml

```
web: java -Dserver.port=$PORT -Dpaypal.client.id=$PAYPALCID -
Dpaypal.client.secret=$PAYPALCS -Dgoogle.api.key=$GOOGLE -
Dspring.datasource.password=$PASS $JAVA_TOOL_OPTIONS -jar target/lieferrex-0.0.1-
SNAPSHOT.jar --spring.profiles.active=prod
```

Abbildung 174: Procfile

Das Heroku Konto muss zudem mit dem GitHub Repository verbunden sein. Zudem müssen im Heroku Dashboard die Variablen für die Procfile sowie eine Variable mit dem Namen „MAVEN_CUSTOM_GOALS“ angelegt werden. Letztere dient dazu das Projekt zu bauen. Heroku baut also zunächst das Projekt und führt danach den Befehl in der Procfile aus. In folgender Abbildung sind die Einstellungen der Variablen in Heroku zu sehen.

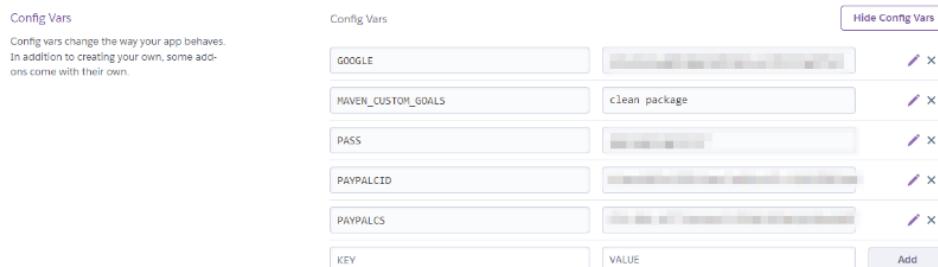
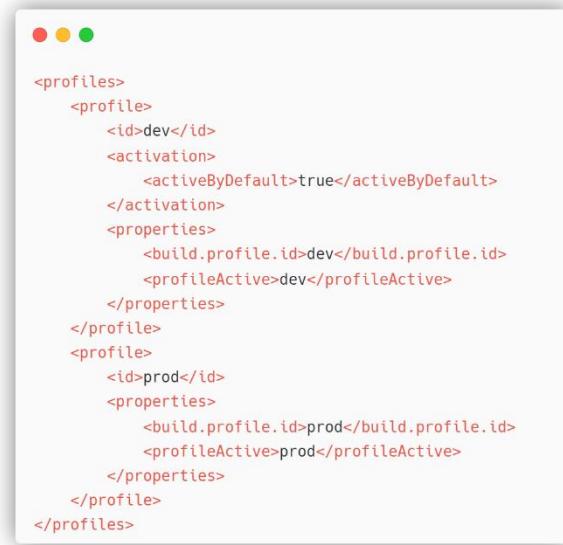


Abbildung 175: Heroku Variablen Einstellungen

Nun sind natürlich auch Spring Boot Profile für das Funktionieren des Deployments notwendig. Dazu wurden zwei Profile erstellt. Einmal gibt es das Profil „dev“ für die Entwicklung und einmal das Profil „prod“ für die Produktivumgebung. Dazu muss man zunächst in der pom.xml die Profile erstellen. Die Konfiguration der Profile ist in folgender Abbildung zu sehen.



```

<profiles>
    <profile>
        <id>dev</id>
        <activation>
            <activeByDefault>true</activeByDefault>
        </activation>
        <properties>
            <build.profile.id>dev</build.profile.id>
            <profileActive>dev</profileActive>
        </properties>
    </profile>
    <profile>
        <id>prod</id>
        <properties>
            <build.profile.id>prod</build.profile.id>
            <profileActive>prod</profileActive>
        </properties>
    </profile>
</profiles>

```

Abbildung 176: pom.xml Spring Profiles

Nun müssen zwei zusätzliche Dateien angelegt werden. Einmal die application-dev.properties Datei und einmal die application-prod.properties Datei. In der application.properties Datei muss nun folgende Zeile enthalten sein: `spring.profiles.active=@activatedProperties@`.

Das Default-Profil ist jenes für die Entwicklung. So kann man ohne Probleme am Projekt arbeiten und sobald es in das Repo gepusht wird, verwendet Heroku durch den Befehl in der Procfile das Produktiv-Profil.

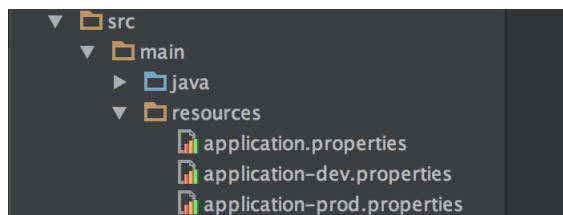


Abbildung 177: application.properties Dateien

Wie in folgender Abbildung zu sehen ist, sind einige Felder der application-prod.properties Datei leer. Jene leeren Felder werden von Heroku mit den definierten Variablen befüllt.



```

spring.jpa.hibernate.ddl-auto=update
spring.datasource.url=jdbc:mysql://${MYSQL_HOST:lieferrex.ddns.net}:3307/lieferrex
spring.datasource.username=conn
spring.datasource.password=
spring.datasource.driver-class-name =com.mysql.jdbc.Driver
spring.web.resources.add-mappings=true
paypal.mode=sandbox
google.api.key=
paypal.client.id=
paypal.client.secret=

```

Abbildung 178: application-prod.properties

8.1 SQL-Datenbank

Da Heroku kein SQL-Datenbank-Hosting anbietet wird die SQL-Datenbank separat gehostet. Dazu wird das Synology-NAS von Michael Bogensberger verwendet. Dazu wurde auf dem NAS eine DDNS- Domain sowie eine MariaDB Instanz erstellt. Da es sich bei dem NAS um ein Produkt von Synology handelt, kann die MariaDB Instanz ganz einfach über den Store installiert werden. Des Weiteren wurde noch PhpMyAdmin installiert.



Abbildung 179: NAS MariaDB Insatz

The screenshot shows the 'User' section of the PhpMyAdmin interface. The top navigation bar includes 'Global', 'Datenbank', 'Passwort ändern', and 'Anmeldeinformation'. Below this, it says 'Rechte ändern: Benutzerkonto 'conn'@'%''. The main area is titled 'Datenbankspezifische Rechte' and shows a table with columns: Datenbank, Rechte, GRANT, Tabellenspezifische Rechte, and Aktion. There is one row for 'lieferrex' with 'ALL PRIVILEGES' under 'Rechte' and 'Nein' under both 'GRANT' and 'Tabellenspezifische Rechte'. Buttons for 'Rechte ändern' and 'Entfernen' are available. A dropdown menu below the table lists databases: 'cloudsys', 'gymtracker', 'ironman_2021', and 'mariokart'. A note at the bottom says 'Rechte zu folgender Datenbank(en) hinzufügen:' followed by a text input field and a help icon.

Abbildung 180: PhpMyAdmin User

In PhpMyAdmin wurde darauf ein User für das Projekt mit passenden Rechten erstellt. Dessen Anmeldedaten werden in den Heroku Variablen gespeichert.

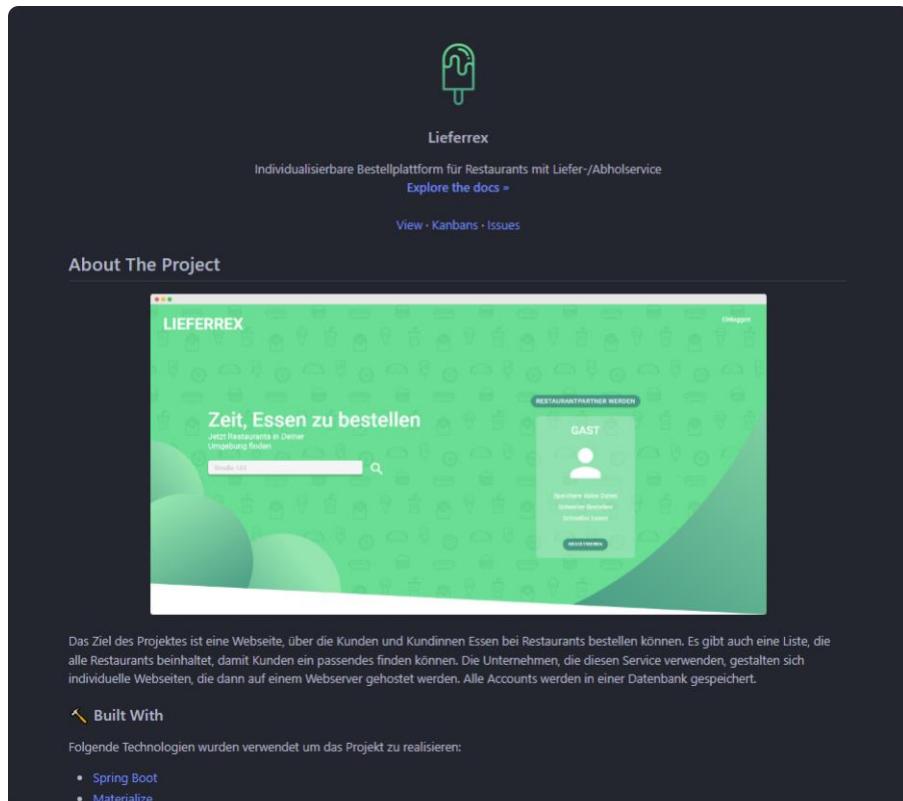
The screenshot shows the 'DDNS' tab of the Synology Systemsteuerung. The top navigation bar includes 'DDNS', 'Routerkonfiguration', and 'Erweitert'. Below this, there are buttons for 'Hinzufügen', 'Bearbeiten', 'Löschen', 'Jetzt aktualisieren', and 'Anpassen'. A table lists service providers and their corresponding hostnames and external addresses. Two entries are shown: 'Synology' with hostname 'mbogensberger.synology.me' and external address '62.116.62.103', and 'No-IP.com' with hostname 'lieferrex.ddns.net' and external address '62.116.62.103'. Both entries have a status of 'Normal'.

Abbildung 181: NAS DDNS-Einstellungen

Durch das Deployment ist die aktuellste Version des Projektes unter folgender URL erreichbar:
lieferrex.herokuapp.com

9 GitHub & Scrum

Um das Projekt gut versionieren und verwalten zu können wird GitHub verwendet. Des Weiteren läuft beispielsweise das Kanban für Scrum im GitHub Repository. Das GitHub Repository ist unter folgendem Link zu finden: github.com/MichaelBogensberger/Lieferrex. In Abbildung 182: GitHub Repository, Readme File ist die Readme Datei des Projektes zu sehen. Dabei wurde erneut viel Wert auf ein



entsprechendes Design der Readme Datei gelegt.

Abbildung 182: GitHub Repository, Readme File

Da es uns wichtig war, das Projekt möglichst nahe an der Realität zu entwickeln, haben wir uns für eine Entwicklung mithilfe des Scrum-Modells entschieden. Dazu gab es regelmäßige Meetings, definierte Sprints und ein klares Ziel vor den Augen. Für die Verwendung von Scrum wurde das in GitHub integrierte Feature der „Projects“ verwendet. Damit können ganz einfach Kanban Boards erstellt werden. In Abbildung 183: GitHub - Kanban Board ist ein solches Kanban Board zu sehen.

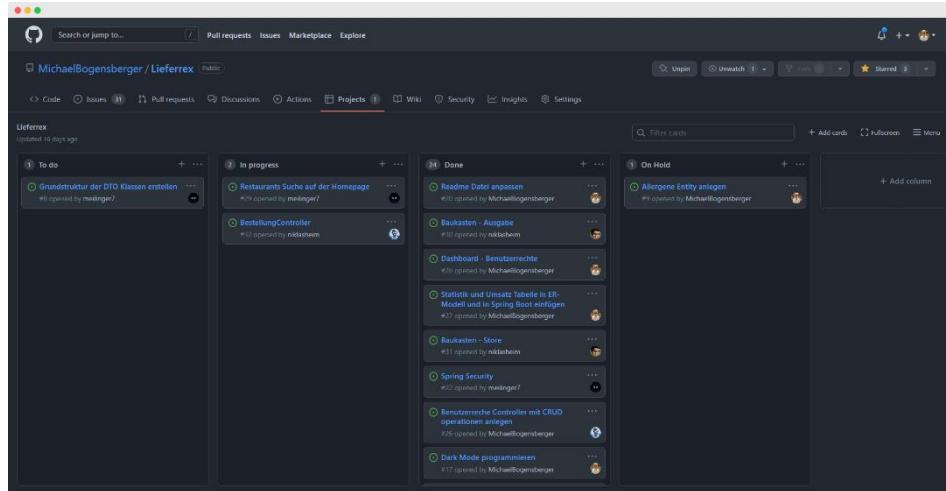


Abbildung 183: GitHub - Kanban Board

9.1 GitGuardian

GitGuardian ist eine kostenlose Extension für GitHub die einen zusätzlichen Security Layer bietet. Wird aus Versehen bei einem Push ein API-Key oder sonstige sensible Daten ins öffentliche Repository geladen, so schlägt GitGuardian Alarm. GitGuardian scannt also bei jedem Push, ob sich ein Secret im jeweiligen Push befindet und schickt gegebenenfalls eine E-Mail mit einer Nachricht, dass sich ein Secret im Push befindet. Zudem hat man auf dem GitGuardian-Dashboard weitere Möglichkeiten zur Einsicht. Des Weiteren bietet GitGuardian eine sehr gute CI/CD Integration an.

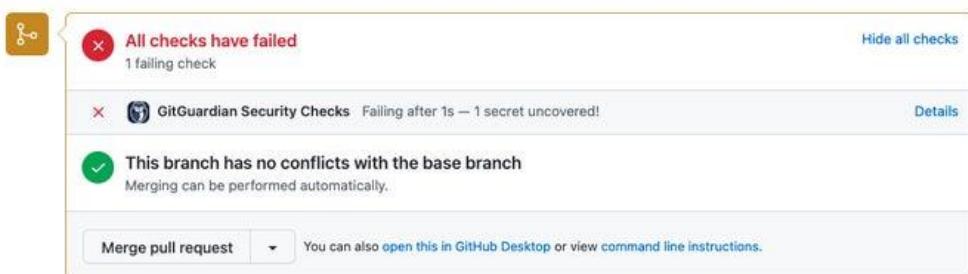


Abbildung 184: GitGuardian, Merge Pull Request

10 Codemenge

Insgesamt entstanden sechs CSS-Dateien mit über 1.800 Zeilen, 46 HTML-Dateien mit über 6.000 Zeilen, 158 Java Dateien mit über 5.800 Zeilen und 5 JavaScript Dateien mit über 400 Zeilen Code. In Abbildung 185 Lieferrex, Code-Verteilung ist die Verteilung des Codes noch einmal in einem Kreisdiagramm zu sehen.

Code-Verteilung

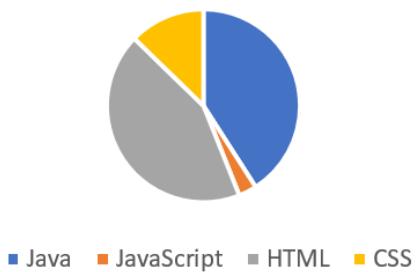


Abbildung 185 Lieferrex, Code-Verteilung

11 Tests

Tests werden in der Systementwicklung häufig genutzt um Fehler, oder sogenannte Bugs, zu finden und diese zu beseitigen. Dies wird gemacht, um die Funktion aller Komponenten zu garantieren und am Ende ein vollständiges Produkt abliefern zu können.

11.1 Systemtests

Im Bereich Systemtest werden die einzelnen Komponenten einer Anwendung auf ihr zusammenwirken getestet. Die Systemtests konzentrieren sich auf die Funktionalität der Anwendung. Mithilfe von Black-Box-Tests, eine Methode für Softwaretests bei denen der Tester nicht die genaue Implementierung von den verwendeten Methoden kennen muss, werden die einzelnen Komponenten anhand der Spezifikationen bzw. Anforderungen getestet und weiterentwickelt.

Im Rahmen dieses Projektes, fallen viele Tests an. Im Folgendem werden einige Testfälle aufgelistet. Ein Testfall ist der Mandanten Kontroller, dort wird getestet ob die Stammdaten des Mandanten veränderbar sind.

Stammdaten des Mandanten ändern		
Nummer:	1	Erfolgreich getestet:
Beschreibung:	Es soll getestet werden, ob eine Änderung der Stammdaten des Mandanten auf dem Dashboard korrekt gespeichert und wieder angezeigt werden.	
Betroffener Programmteil:	MandantController	
Vorbedingung:	Damit dieser Testfall getestet werden kann muss man sich zuvor einloggen. Andernfalls gelangt man nicht aufs Dashboard.	
Tester:	Michael Bogensberger	Datum: 17.05.2022
Testschritte		
Schritt	Aktion	Erwartetes Ergebnis
1	Einloggen mit E-Mail und Passwort auf der Kundenseite.	Session wird erstellt und man wird aufs Dashboard weitergeleitet.
2	In der Navigationsleiste auf die "Mandant" Seite klicken.	Man gelangt auf die Mandanten-Seite.
3	Adresse, Lieferkosten sowie Mindestbestellwert ändern und auf Speichern klicken.	Die Seite wird neu geladen und eine Meldung mit "Daten gespeichert" erscheint.
4	Checken, ob die eingegeben Daten auf der Seite angezeigt werden.	Die zuvor eingegeben Daten werden korrekt angezeigt.

Tabelle 14 Stammdaten des Mandanten ändern

Ein weiterer Testfall ist der Bestell Kontroller, dort wird getestet ob ein Kunde eine Bestellung tätigen kann.

Bestellung tätigen			
Nummer:	1	Erfolgreich getestet:	✓
Beschreibung:	Es soll getestet werden, ob eine Bestellung über die Webseite erfolgreich angelegt werden kann. Dazu wird übers Frontend eine Bestellung mit verschiedenen Gerichten angelegt und diese Abgeschickt. Danach wird zu jener Seite gewechselt, bei der man die Bestellungen einsehen kann. Nun wird getestet, ob die Bestellung hier auch angezeigt wird.		
Betroffener Programmteil:	BestellController		
Vorbedingung:	Damit dieser Testfall getestet werden kann muss man sich zuvor einloggen. Ansonsten kann man keine Bestellung tätigen.		
Tester:	Michael Bogensberger	Datum:	17.05.2022
Testschritte:			
Schritt	Aktion	Erwartetes Ergebnis	
1	Einloggen mit E-Mail und Passwort auf der Kundenseite.	Session wird erstellt und man wird auf die Startseite weitergeleitet.	
2	Auf der Hauptseite eine Adresse eingeben und nach einem Restaurant suchen.	Die "Suchen" Seite mit Restaurants wird angezeigt.	
3	Restaurant anklicken.	Man gelangt auf die Seite des Restaurants.	
4	Im Menü-Formular Gerichte auswählen und auf hinzufügen klicken.	Gerichte werden dem Warenkorb hinzugefügt.	
5	Auf Bestellen klicken.	Man gelangt auf die "Bestellung abschließen" Seite.	
6	Mit PayPal bezahlen. Dazu drückt man auf Bezahlen.	Nun öffnet sich ein Fenster der PayPal API.	
7	Mit PayPal Daten anmelden und auf Bezahlen klicken.	Das Fenster schließt sich und man gelangt zurück auf die Startseite.	
8	Auf die Bestellung Seite wechseln. Dazu klickt man in der Navigationsleiste auf Profil und dann auf Bestellungen.	Man gelangt zur "Bestellungen" Seite.	
9	Checken, ob die Bestellung sichtbar ist.	Die zuvor getätigte Bestellung wird nun angezeigt.	

Tabelle 15 Bestellung tätigen

Im nächsten Fall wird getestet, ob der Mandant seine Öffnungszeiten anpassen kann.

Öffnungszeiten des Restaurants ändern			
Nummer:	1	Erfolgreich getestet:	✓
Beschreibung:	Es soll getestet werden, ob eine Änderung der Öffnungszeiten des Mandanten auf dem Dashboard korrekt gespeichert und wieder angezeigt werden.		
Betroffener Programmteil:	OeffnungszeitenController		
Vorbedingung:	Damit dieser Testfall getestet werden kann muss man sich zuvor einloggen. Andernfalls gelangt man nicht aufs Dashboard.		
Tester:	Michael Bogensberger	Datum:	17.05.2022
Testschritte:			
Schritt	Aktion	Erwartetes Ergebnis	
1	Einloggen mit E-Mail und Passwort auf der Kundenseite.	Session wird erstellt und man wird aufs Dashboard weitergeleitet.	
2	In der Navigationsleiste auf die "Öffnungszeiten" Seite klicken.	Man gelangt auf die Öffnungszeiten-Seite.	
3	Öffnungszeiten an verschiedenen Tagen ändern und auf Speichern klicken. Dazu sollte man bei zwei Tagen die Öffnungszeiten ändern und an einem Tag das Restaurant als geschlossen markieren.	Die Seite wird neu geladen und eine Meldung mit "Daten gespeichert" erscheint.	
4	Checken, ob die eingegeben Daten auf der Seite angezeigt werden.	Die zuvor eingegebenen Daten werden korrekt angezeigt.	

Tabelle 16 Öffnungszeiten des Restaurants ändern

Auch getestet wird die Erstellung einer Webseite mit dem Baukastensystem. Hier muss überprüft werden, ob die erstellte Seite korrekt gespeichert und anschließend angezeigt wird.

Webseite im Baukasten anlegen			
Nummer:	4	Erfolgreich getestet:	X
Beschreibung:	Es soll getestet werden ob, ob Mandanten mithilfe des Baukastensystems eine Webseite erstellen können. Diese muss korrekt in der Datenbank gespeichert werden damit sie anschließend aus der Sicht des Kunden ausgegeben werden kann.		
Betroffener Programmteil:	BaukastenController, RestaurantController		
Vorbedingung:	Damit dieser Testfall getestet werden kann, muss sich der Mandant vorher anmelden.		
Tester:	Niklas Heim	Datum:	23.06.2022
Testschritte:			
Schritt	Aktion	Erwartetes Ergebnis	

1	Einloggen mit E-Mail und Passwort als Mandant.	Session wird erstellt und man wird auf die Startseite weitergeleitet.
2	Über die Einstellungen den Baukasten öffnen.	Leere Baukastenseite öffnet sich.
3	Allgemeine Einstellungen treffen.	Allgemeine Einstellungen werden übernommen.
4	Hinzufügen der Module mit entsprechenden Werten.	Ausgabe der erstellten Module.
5	Webseite vollständig konfiguriert.	Erstellte Webseite wird in der Datenbank gespeichert.
6	Suchen des Restaurants über das Such-Menü.	Entsprechendes Restaurant wird ausgegeben.
7	Zuvor erstellte Webseite des Restaurants wird angezeigt.	Webseite des Restaurants wird ohne Fehler ausgegeben.
8	Auf die Bestellung Seite wechseln. Dazu klickt man in der Navigationsleiste auf Profil und dann auf Bestellungen.	Man gelangt zur "Bestellungen" Seite.

Tabelle 17 Webseite im Baukasten anlegen

In Tabelle 18 wird der Registrierung und Login Use-Case für Kunden getestet. Dabei wird der gesamte Ablauf vom Erstellen eines Benutzers bis zum Anmeldevorgang beschrieben. Das erwartete Ergebnis ist ein Valider User, der in korrekt in der Datenbank gespeichert ist. Desweiterem soll eine sichere Session erzeugt werden, die den angemeldeten User verifiziert.

User Registrierung und Login		
Nummer:	1	Erfolgreich getestet: ✓
Beschreibung:	Es soll überprüft werden, ob sich ein Kunde oder ein Restaurant Mitarbeiter erfolgreich ins System einloggen kann.	
Betroffener Programmteil:	SecurityController, UserPrincipalDetailsService	
Vorbedingung:	Es darf kein Benutzer auf der Webseite bereits angemeldet sein. Ein User mit derselben E-Mail darf nicht vorhanden sein.	
Tester:	Julian Meilinger	Datum: 23.05.2022
Testschritte		
Schritt	Aktion	Erwartetes Ergebnis
1	Auf der Homepage klickt man auf den "Registrieren" Button	Man wird zur Registrierungs-Seite weitergeleitet.
2	Man gibt gültige Anmeldeinformationen ein und akzeptiert die ABG's. Danach klickt man auf dem Button "Registrieren".	Es erscheint die Meldung "Benutzer erfolgreich angemeldet".
3	Man geht zurück zur Homepage und klickt rechts oben auf den "Einloggen" Button.	Man wird zur Login-Seite weitergeleitet.

4	Hier gibt man die zuvor gewählte E-Mail und das dazu passende Passwort ein. Danach klickt man auf dem Button "Login".	Danach wird man zur Homepage weitergeleitet und man ist angemeldet. Desweiterem wird ein neues Cookie gespeichert, der die Session-ID zur Verifizierung beinhaltet
---	---	--

Tabelle 18 User Registrierung und Login

11.2 Akzeptanztests

Im Gegensatz zu den Systemtests sind die Akzeptanztests, Tests, die aus der Sicht des Benutzers funktionieren sollen, das heißt die Überprüfung wird vom Benutzer getestet, sogenannte Beta-Tests. Die Akzeptanztest werden meist in der letzten Phase eines Projektes durchgeführt, im besten Fall bevor der Kunde die Software benutzt.

12 Evaluation

In folgendem Abschnitt ist die Evaluation des Projektes zu sehen. Diese dient dazu das Projekt sach- und fachgerecht zu untersuchen und zu bewerten.

12.1 Projektevaluation

Hier ist die Evaluation des Projektes zu finden. In dieser Passage wird auf Themen wie zum Beispiel die Planungsabweichungen oder die Zusammenarbeit der jeweiligen Projektmitglieder untereinander, aber auch die Zuordnung der Subthemen auf die Projektmitglieder eingegangen.

12.1.1 Planungsabweichungen

Grundsätzlich gab es im Projekt keine allzu großen Planungsabweichungen. Jedoch lässt sich sagen, dass es vor allem gegen Ende des Projektes eine suboptimale Zeitplanung gab. Da unter anderem der Stress der letzten Wochen im Sommersemester der letzten Klasse sowie der Mehraufwand für das Beheben von Fehlern oder das Optimieren des Codes nicht ausreichend berücksichtigt wurden, gab es im Vergleich zum Projektstart am Ende des Projektes eine eher stressigere Phase. Ansonsten entstanden durch die Verwendung des Scrum Modells keine weiteren oder nennenswerten Planungsabweichungen.

12.1.2 Aufwand/Kosten

Während der Entwicklung sind keine Kosten in das Projekt eingeflossen. Eine kostenpflichtige Lizenz für IntelliJ, ist von der Schule bereitgestellt worden. Des Weiteren wird derzeit die Webseite über das NAS von Michael Bogensberger gehostet, was keine Zusatzkosten verursacht. Die Nutzung der Google API verursacht je nach Typ und Anzahl der Abfrage Kosten. Deshalb wurde das 90 Tage und 284,30 € gratis Guthaben verwendet, was von Google bereitgestellt wird. Die API verursacht jedoch während der Produktivphase bestimmte variable Kosten. Des Weiteren muss die Applikation in einem Echtbetrieb bestimmt auf einen performanteren und zuverlässigeren Webserver laufen, was auch Kosten verursachen wird.

12.1.3 Zusammenarbeit

Die Zusammenarbeit während der Projektzeit ist sehr reibungslos verlaufen. Problemstellungen, die jeden Projektant betroffen haben, wurden als Team besprochen und ausgearbeitet. Während der Projektplanung in Unterrichtsfach SYP wurde immer eine sehr faire und effiziente Arbeitsaufteilung gemacht. Bei Problemen in der Implementierungsphase hatte jedes Mitglied ein offenes Ohr für

andere. Besonders hilfreich war dabei, dass alle Projektmitglieder im selben Wohngebäude wohnen und Kommunikationswege sehr kurz waren.

Die einzige Problematik, war das frühe Ausscheiden von Burak Eraslan, ein ehemaliges Projektmitglied. Dies hatte jedoch keine grobe Auswirkung, auf das Projekt, da er keine kritische Teilaufgaben zugeteilt bekommen hat bzw. sein Themenbereich sehr abgetrennt vom Rest zu bearbeiten ist.

12.1.4 Zuordnung der Subthemen

Michael Bogensberger war für das Frontend zuständig. Dies beinhaltet das Designen der Webseite, als auch die Planung, welche Ansichten benötigt werden, wie man sich auf der Webseite bewegen kann und welche Informationen dem Benutzer oder Restaurant ausgegeben werden. Auch war Michael Bogensberger unterstützend am Backend beteiligt.

Das Backend wurde von Julian Meilinger entwickelt. Hier gab es viele Aufgaben, mit denen er sich beschäftigen musste. Dazu zählen das Entwerfen der verschiedenen Entitäten, die Datenverarbeitung, Registrierung und Anmeldung von Benutzern und Restaurants, sowohl das Einbinden von APIs fürs Bezahlen.

Xia Liuming war bei diesem Projekt verantwortlich für die Datenbank. Dies umfasst die Planung und Umsetzung, um Daten von Restaurant und Kunden speichern zu können. Des Weiteren unterstützte er beim Eingeben und Ausgeben von Daten am Dashboard für Restaurants.

Der Baukasten, der von den Restaurants verwendet wird, wurde von Niklas Heim entworfen. Dazu gehört das Designen der verschiedenen Bausteine, entwerfen der Datenbank, um diese zu speichern und eine Oberfläche mit dem Restaurants ihre Webseiten bauen können.

12.2 Produktevaluation

Das fertige Produkt ist sehr gut gelungen. Es konnten alle Anforderungen des Projektpartners weitgehend erfüllt werden. Durch gute Planung und saubere Arbeit gab es nichts Großartiges, das wir im Laufe des Projekts anpassen mussten.

12.3 Resümee

In folgendem Abschnitt sind die Resümeees der jeweiligen Teammitglieder zu finden. Hier wird jeweils pro Teammitglied darauf eingegangen, wie die Erfahrungen mit dem Projekt waren und was man aus jenen gelernt hat. Die gesetzten Muss-Anforderungen konnten alle gut erfüllt werden. Auch wurden einige Kann-Anforderungen, wie beispielsweise einer Online-Zahlung oder weiterer Anpassungsmöglichkeiten im Baukastensystem und weiteres erfüllt werden.

12.3.1 Michael Bogensberger

Bei so einem Projekt merkt man erst, wie wichtig es ist, sich einen guten Plan für das Projekt zu erstellen. Legt man einfach los, so geht es anfangs flott voran, jedoch merkt man, dass man später mit einem unnötigen Mehraufwand draufzahlt. Deshalb war es auch wichtig, ein möglichst strukturiertes Vorgehen durch zum Beispiel den Einsatz von Scrum zu haben. Dennoch lief nicht alles reibungslos ab. Ich habe unter anderem gemerkt, wie wichtig es ist, sich mit den Projektmitgliedern im ständigen Austausch zu befinden. So besteht nicht die Gefahr, dass man etwas nicht so implementiert, wie es das Projektmitglied benötigt. Die richtige Kommunikation der Teammitglieder untereinander ist diesbezüglich das A und O. Aber auch die eigene Organisationsfähigkeit ist sehr wichtig. Es ist notwendig, sich selbst die Zeit gut einzuteilen. Ich hatte das Gefühl, dass wir allesamt eher gegen Ende des Projektes mehr Zeit hineinsteckten als am Anfang. Jedoch weiß man sowas erst, wenn es zu spät ist. Ich werde mir also mitnehmen, dass man sich einen gewissen Spielraum für die Abschlussphase von Projekten lassen sollte. Alles in allem bin ich sehr zufrieden mit unserer Leistung. Ich bin der Meinung, dass sich unser Projekt sehr gut sehen lassen kann.

12.3.2 Niklas Heim

Es war ein großartiges Projekt. Die Diplomarbeit war eine sehr gute Erfahrung und ich habe vieles gelernt. Dies beinhaltet sowohl fachliches Wissen als auch die Arbeit in einem Team. Kommunikation untereinander war stets gut und wir konnten und regelmäßig miteinander absprechen. Am schwierigsten, aber auch am wichtigsten war die saubere Planung des Projektes. Hier mussten wir besonders sorgfältig sein, damit das Projekt ohne Probleme ablaufen konnte. Zeitlich bin ich sehr gut mit meinen Aufgaben ausgekommen. An manchen Stellen bin ich hin und wieder etwas hängen geblieben, das konnte aber durch sorgfältiges Planen schnell behoben werden. Als Projektleiter war ich sehr zufrieden mit meinem Team. Jeder hat stets sauber an seinen Teilgebieten gearbeitet. Es gab nur wenig Dinge, die ich als Projektleiter erledigen musste. Auf meine Leistung bei dieser Diplomarbeit bin ich stolz. Ich bin froh, diese Arbeit mit meiner Gruppe machen zu können und ich würde jederzeit wieder mit diesem Team ein Projekt starten.

12.3.3 Liuming Xia

In der Anfangsphase des Projektes gab es einige Probleme, die durch gute Zusammenarbeit und Planung ausgeglättet werden konnten. Dort ist mir auch aufgefallen, wie wichtig es ist eine richtige Struktur und einen akkuraten Terminplan zu gestalten. Wird der Plan nicht sorgfältig gestaltet kommt es innerhalb des Projektes zu Komplikationen bzw. das Projekt kommt zum Stocken, dadurch entstehen im Team hitzige Angelegenheiten. Die Angelegenheiten konnte durch den Projektleiter geschlichtet werden und die Arbeit konnte wie gewohnt weiter ausgeführt werden. Die Hauptangelegenheit ist, dass ein Projektmitglied vom Projekt ausgeschieden ist. Dies ist sehr bedauernswert. Im Projektverlauf sind ebenso einige Probleme aufgetreten, die durch regelmäßige Audits gelöst wurden. Im Großen und Ganzen ist das Projekt ein voller Erfolg, ich konnte vieles dazu lernen. Ein Beispiel ist der Zusammenhalt innerhalb des Teams, jeder hilft jedem, wenn wer Probleme hat.

12.3.4 Julian Meilinger

Rückblickend kann ich behaupten, dass diese Diplomarbeit mir einen guten Einblick in die Projektarbeit gegeben hat. Besonders zum Schluss habe ich gemerkt, wie wichtig es ist immer am Ball zu bleiben, da sich sonst die Arbeit am Ende zusammenhäuft. Die Tests und Schularbeiten während der Schulzeit haben mich dabei öfters aus der Bahn geworfen, da für mich damals die Diplomarbeit nicht an erster Stelle war und ich öfter aus dem Rhythmus gefallen bin. Dennoch denke ich, dass ich meine Zeit sehr effektiv genutzt habe und sehr gute Fortschritte gemacht habe. Besonders stolz macht mich, dass ich mehrere neue Technologien erlernt habe. Für die richtige Wahl des Technologie Stacks werde ich, in zukünftigen Projekten auch wieder beachten, da ich gelernt habe, dass es die Arbeitskomplexität sehr stark vereinfachen kann. Im Großen und Ganzen bin ich zufrieden mit der Leistung des Projektteams. Es ist schön zu sehen, wie eine einfache Idee, im Kollektiv in die Realität umgesetzt wird, auch wenn das Resultat noch nicht ganz mit den Marktführern mithalten kann.

12.4 Zeitaufstellung

In folgender Tabelle sind die geloggten Stunden der jeweiligen Projektmitglieder zu sehen. Links in der Tabelle ist das jeweilige Projektmitglied zu sehen. Links ist der Aufwand des Projektes in Stunden zu sehen.

Projektmitglied	Stunden
Michael Bogensberger	157h
Niklas Heim	155h
Liuming Xia	152h
Julian Meilinger	154h

Tabelle 19 Zeitaufstellung

13 Benutzerhandbuch

Das Benutzerhandbuch beschreibt, wie die Funktionen des fertigen Produktes verwendet werden.

13.1 Verwendung des Baukastens

Im folgenden Abschnitt werden alle Funktionen des Baukastens erklärt.

13.1.1 Allgemeine Einstellungen

Im linken Bereich des Baukastens befindet sich ein Menü, über das der Name des Restaurants, die Akzentfarbe und das Layout angepasst werden können. Die Akzentfarbe ändert die Farbe aller Buttons und der Navigationsleiste.

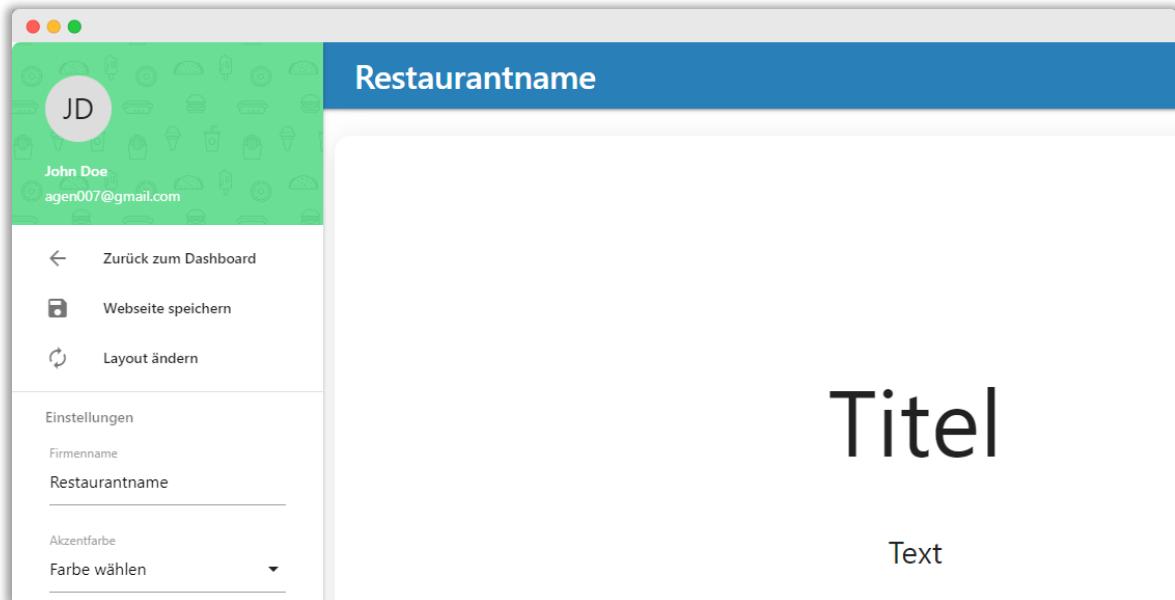


Abbildung 186: Benutzerhandbuch Baukasten - Allgemeine Einstellungen

Beim Ändern des Layouts, öffnet sich ein kleines Fenster, das alle verfügbaren Layouts anzeigt. Hier kann auf das gewünschte geklickt werden.

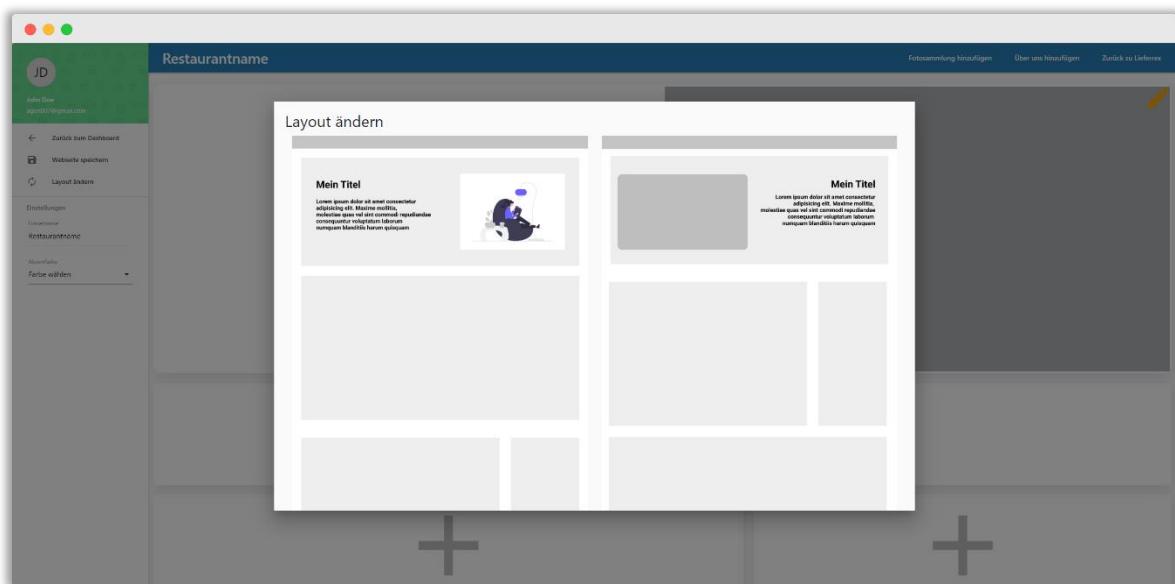


Abbildung 187: Benutzerhandbuch Baukasten - Layout Wahl

Diese Einstellungen werden durch einen Klick auf „Webseite speichern“ übernommen. Nach dem Klick wird die Seite neu geladen und die Änderungen sind aktiv. Achtung: Beim Wechseln des Layouts werden alle Fragmente gelöscht!

13.1.2 Bearbeiten des Kopfbereiches

Nachdem das Layout gewählt wurde, kann der Kopfbereich bearbeitet werden. Dies kann durch einen Klick auf den gelben Stift im rechten oberen Eck gemacht werden. Es öffnet sich ein Fenster, in welchen man den Titel, einen Text und das Bild bearbeiten kann. Sollte das Bild nicht geändert werden, muss der Schalter „Neues Bild hinzufügen“ deaktiviert sein.

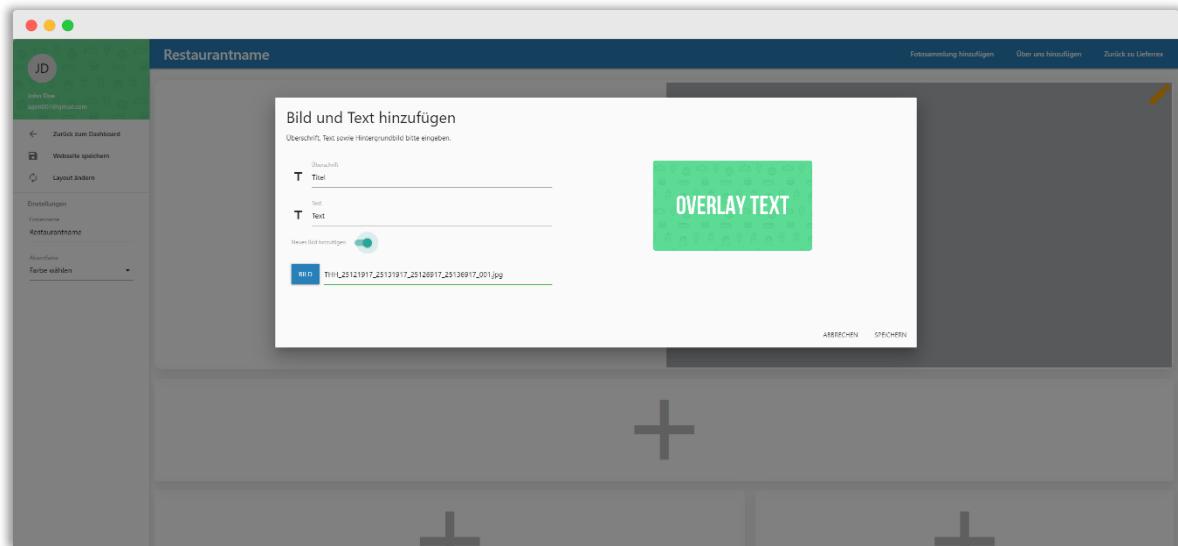


Abbildung 188: Benutzerhandbuch Baukasten - Kopfbereich bearbeiten

Durch einen Klick auf Speichern werden die Einstellungen verarbeitet und die Seite neu geladen.

13.1.3 Modul hinzufügen

Wurde das gewünschte Layout ausgewählt, können neue Module eingefügt werden. Durch einen Klick auf einer der Plus-Zeichen öffnet sich ein Fenster, das alle verwendbaren Module auflistet.

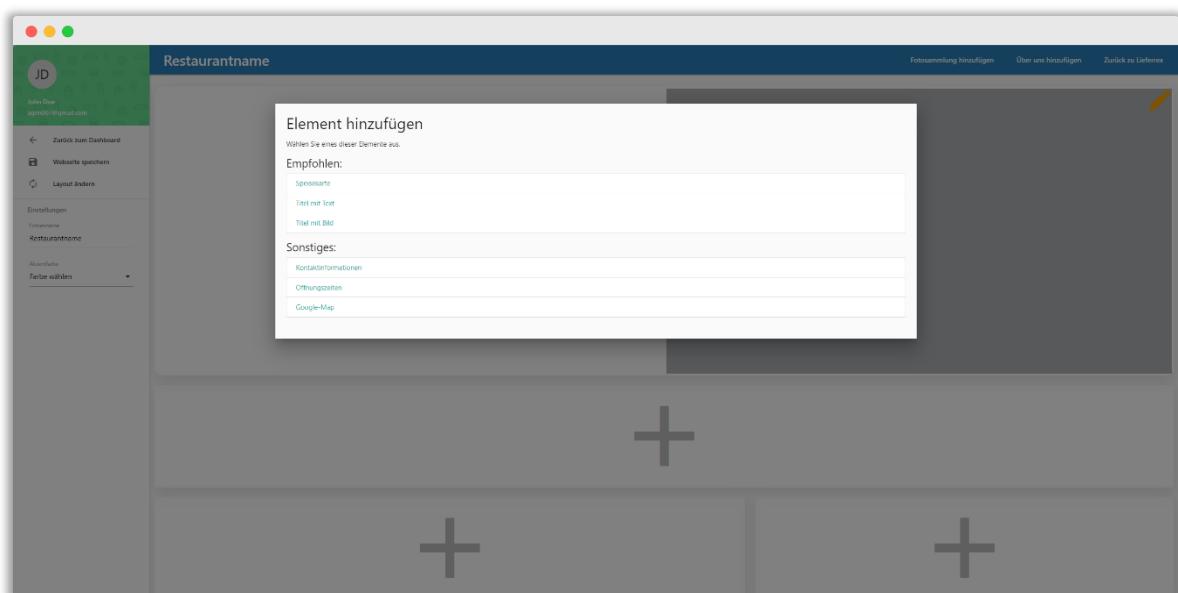


Abbildung 189: Benutzerhandbuch Baukasten - Wahl eines neuen Fragments

Aus diesen Optionen kann das gewünschte Modul gewählt werden. Die Module „Titel mit Text“ und „Text mit Bild“ benötigen zusätzliche Eingaben, die in einem neuen Fenster stattfinden.

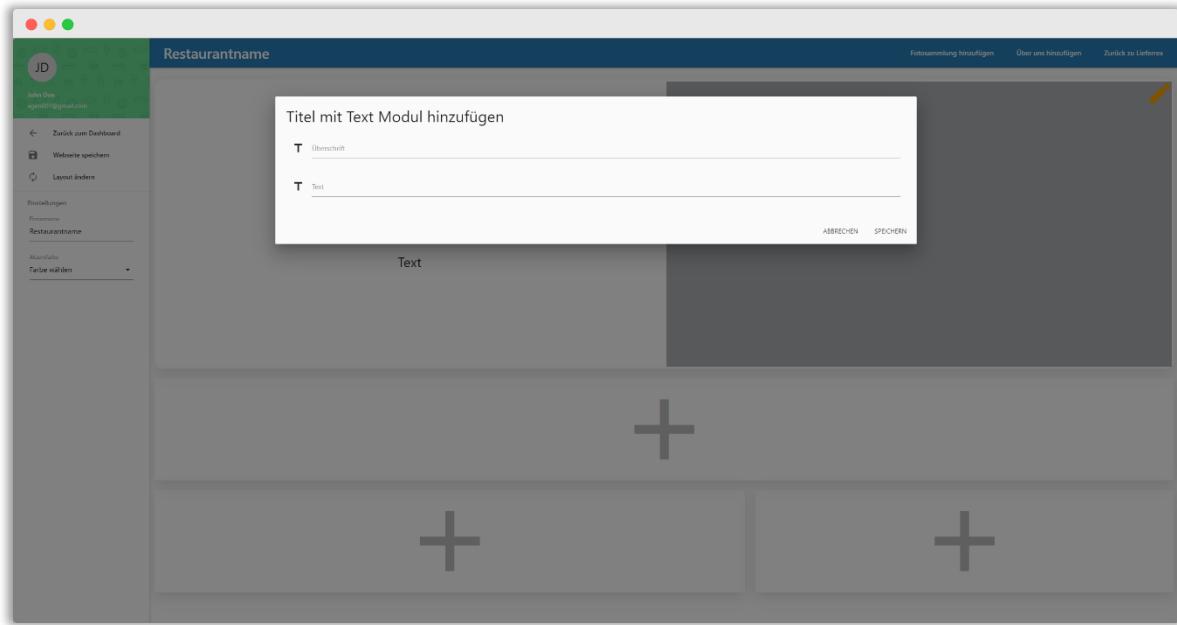


Abbildung 190: Benutzerhandbuch Baukasten - Text-Fragment erstellen

Die anderen Module benötigen keine weiteren Informationen und es öffnet sich ein Fenster, das um eine Bestätigung bittet.

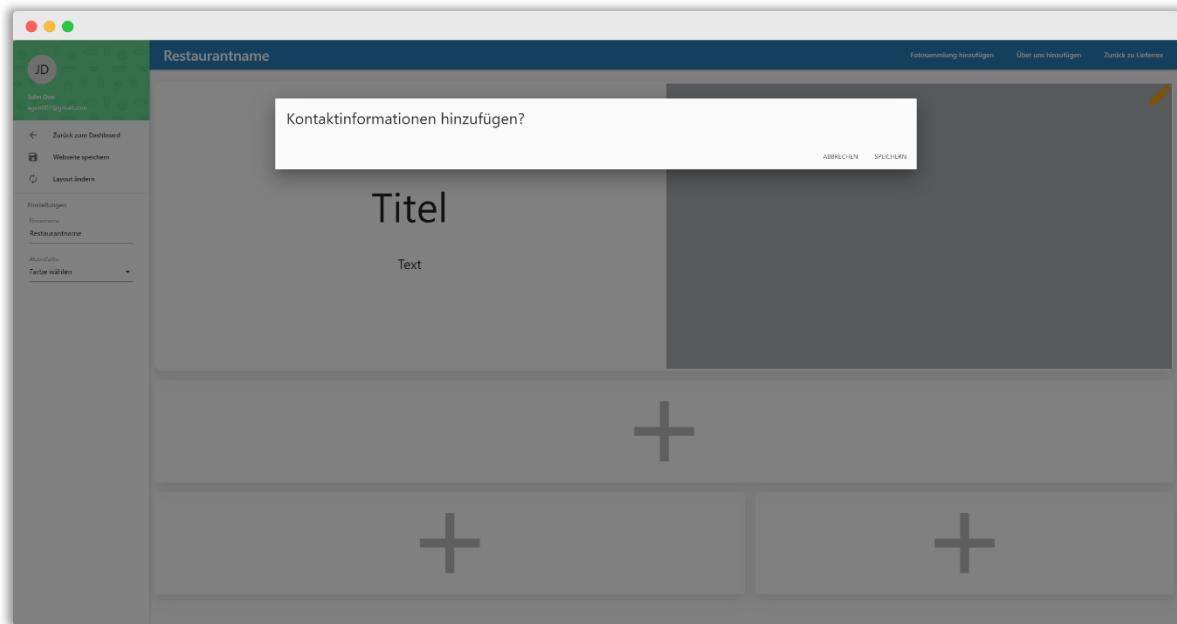


Abbildung 191: Benutzerhandbuch Baukasten - Speicherung bestätigen

Nach dem Speichern wird das neue Modul auf der Webseite angezeigt.

13.1.4 Löschen eines Moduls

Module können auch wieder gelöscht werden. Jedes Modul hat im rechten oberen Eck eine kleine Rote Mülltonne. Ein Klick auf diese öffnet ein Fenster, um den Vorgang zu bestätigen.

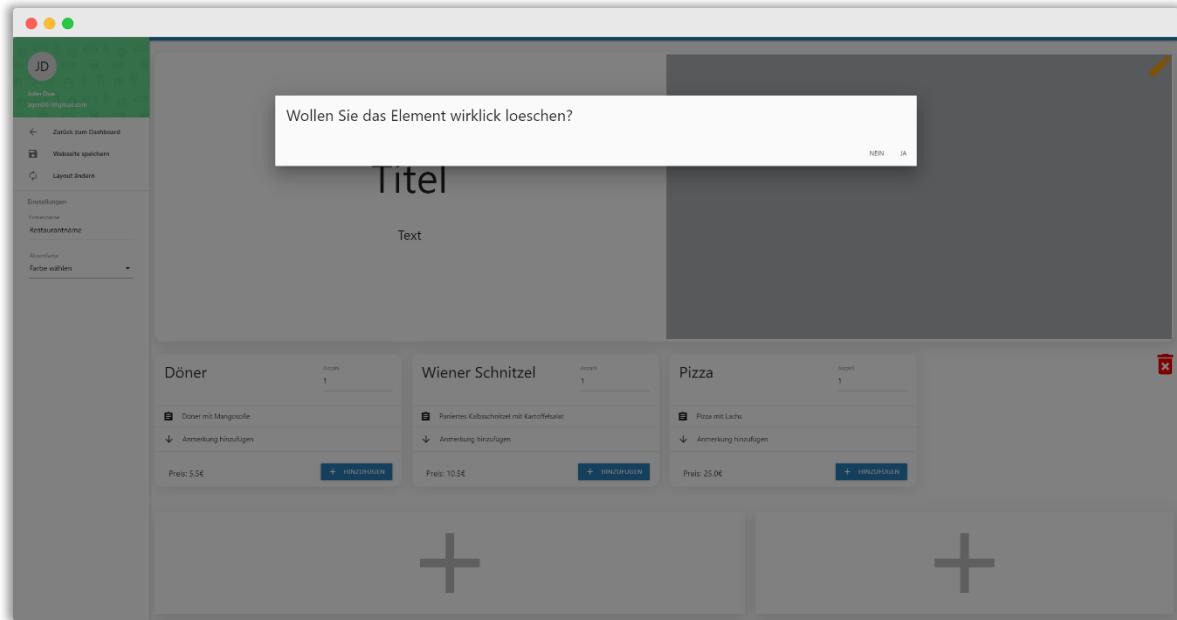


Abbildung 192: Benutzerhandbuch Baukasten - Fragment entfernen

13.1.5 Erstellen von Zusatz-Seiten

Über die Navigationsleiste können die zusätzlichen Seiten „Über uns“ und „Galerie“ angelegt werden. Je nach gewählter Seite öffnet sich ein Fenster, das um die entsprechenden Eingaben bittet.

Die Galerie benötigt einen Titel und genau fünf Bilder.

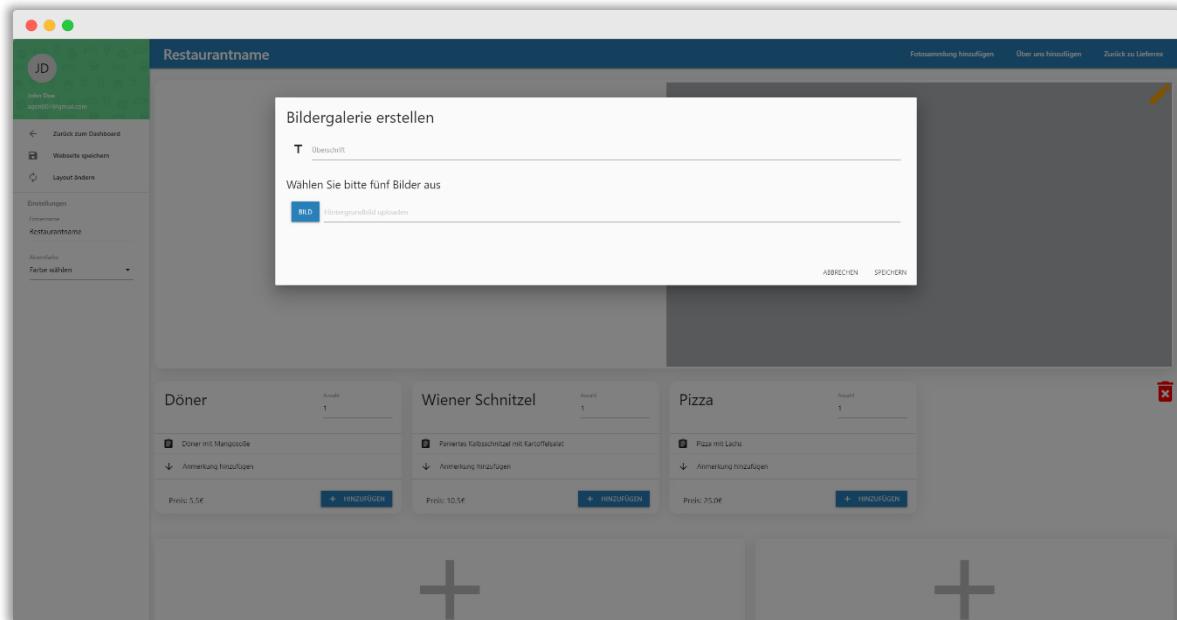


Abbildung 193: Benutzerhandbuch Baukasten - Erstellung der Zusatz-Seite Galerie

Für die „Über uns“-Seite werden ein Titel, zwei Texte und zwei Bilder eingegeben.

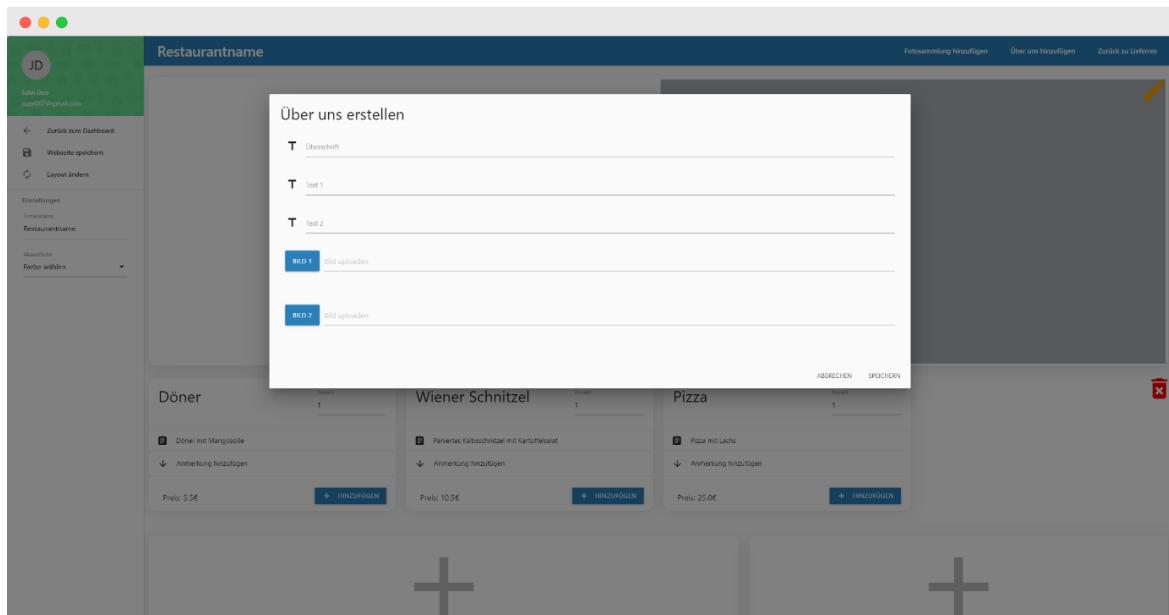


Abbildung 194: Benutzerhandbuch Baukasten - Erstellung der Zusatz-Seite "Über uns"

Nach dem Speichern sind diese für den Kunden zur Verfügung.

13.1.6 Löschen von Zusatz-Seiten

Wurden eine oder zwei Zusatz-Seiten erstellt, können diese nach belieben auch wieder gelöscht werden. Die Buttons für das Löschen ersetzen die für das Hinzufügen. Nach einem Klick auf diese, wird ein Fenster geöffnet, dass um Bestätigung des Benutzers fragt.

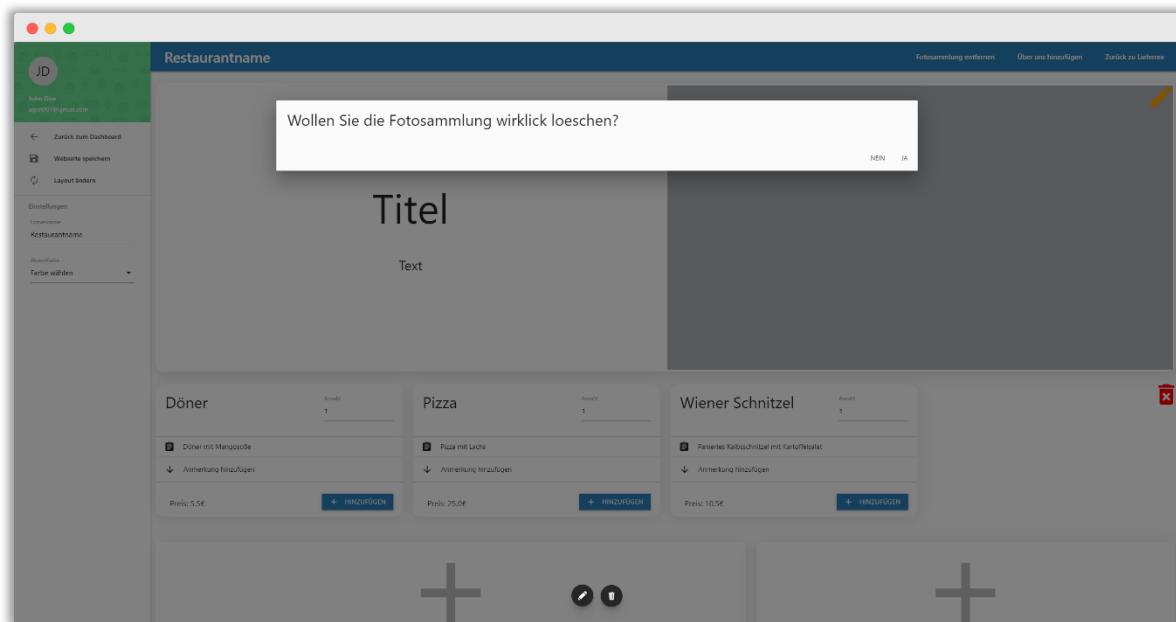


Abbildung 195: Benutzerhandbuch Baukasten - Zusatz-Seiten löschen

14 Abnahme

Die Abnahme erfolgt am 09.07.2022 um 8 Uhr beim Projektpartner vor Ort. Dieser Samstag wurde gewählt, um eine möglichst leichte und schnelle Übergabe zu ermöglichen. An diesem Tag werden vom Partner nur wenige Kunden erwartet. Der Projektpartner erhält das fertige Produkt inklusive Dokumentation. Vor der Abnahme werden auch viele Abnahmetest durchgeführt, um sicherzustellen, dass alle Anforderungen erfüllt wurden und das Produkt stabil ohne Fehler läuft. Getestet wird, ob das Produkt alle nötigen Funktionen aufweist und diese ohne Probleme verwendbar sind. Dies geschieht über Black-Box Tests. So werden die verschiedenen Funktionen des Systems, ohne Kenntnisse über dieses, getestet und das Ergebnis mit den Anforderungen des Pflichtenhefts verglichen. Vor allem getestet werden:

- Verwendbarkeit der Oberfläche
- Registrierung/Anmeldung von Benutzern und Restaurants
- Funktionalität des Baukastensystems
- Ablauf des Bestellprozesses

Das Ergebnis dieser Test wird dokumentiert und als Abnahmeprotokoll dem Auftraggeber übergeben. Da das Produkt zentral vom Projektteam gehostet wird, kann dies sofort über das Internet vom Projektpartner verwendet werden. Nach der Übergabe steht das Projektteam weiterhin bereit, Fehler auszubessern, die während des Echtbetriebs auftreten. Des Weiteren müssen noch die Urheber-, Nutzungs- oder Verwertungsrechte vertraglich geklärt werden, da das Produkt auch von anderen Restaurants verwendet werden wird und das Projektteam unter Umständen weiter Funktionen entwickeln wird. Das fertige System kann sofort vom Auftraggeber verwendet werden. Eine direkte Umstellung ist möglich, da der Partner noch kein vergleichbares System verwendet und das Produkt als Erweiterung für das Unternehmen entwickelt wurde.

15 Einführung

Das Projektteam hat sich zusammen mit dem Auftraggeber darauf geeinigt eine direkte Umstellung einzusetzen. Dies hat vor allem für den Auftraggeber mehrere Vorteile. Zunächst verwendet beziehungsweise bietet der Auftraggeber zurzeit noch keine Lieferungen an. Einen Konflikt gibt es hier nur bei den Bestellungen auf Abholung. Um deutlich zu machen, weshalb man sich hier für eine direkte Umstellung entschieden hat, muss man zunächst die Nachteile des Parallelbetriebs verstehen. Ein Parallelbetrieb würde gleich mehrere Probleme verursachen. Zunächst hat der Auftraggeber nicht die Kapazitäten, zwei Systeme gleichzeitig zu betreiben. Weder aus personeller noch aus monetärer Sicht wäre dies machbar. Der direkte Umstieg macht so einen Mehraufwand nicht nötig. Ganz im Gegenteil, unser Projekt macht die Handhabung der internen Restaurantprozesse um einiges effizienter. Natürlich bekommt das Personal von uns eine Einschaltung in das neue System. Zudem wird auch die Dokumentation mit übergeben. Des Weiteren ermöglicht es dem Restaurant sich direkt auf das neue Modell der Lieferungen sowie der Bestellungen über das Internet zu konzentrieren. Die direkte Umstellung erfordert zudem eine sehr genaue Vorbereitung. Dazu stehen wir im ständigen Kontakt mit dem Auftraggeber. Mehr Informationen zur Abnahme (wie, wann, unter welchen Umständen, usw.) finden sie weiter oben bei der Abnahme. Da das alte System ausschließlich auf Bestellungen über das Telefon beruht, erfordert die Abstellung des alten Systems keinen allzu großen Aufwand. Zudem müssen auch keinerlei Daten in das neue System eingespielt werden. Aus jenen Gründen haben wir uns für einen sauberen und direkten Umstieg entschieden.

16 Betriebsphase

Sobald unser Produkt erfolgreich in den Tagesbetrieb des Unternehmens integriert wurde, beginnt die Betriebsphase. An erster Priorität steht dabei die Arbeit der Mitarbeiter zu erleichtern und für die Restaurants so viele online Bestellungen wie möglich zu generieren. Die Software soll deshalb auch während einem hektischen Arbeitsalltag schnell und einfach zu bedienen sein. Wie die Restaurants die eingehenden Bestellungen verwalten, wird nicht vorgeschrieben. Einfache und realistische Möglichkeiten wären dabei, das Dashboard über ein Smartphone oder einem Tablet zu bedienen. Empfohlen wird jedoch ein Terminal in einem abgesicherten Modus. Dieses Terminal öffnet unsere Internetseite, sobald der Rechner eingeschaltet wird. Aus Sicherheitsgründen sollte der Bildschirm nicht vom Kundenbereich lesbar einsehbar sein. Des Weiteren sollte man den Nutzerprofil des Computers nur eingeschränkte Rechte vergeben, damit niemand unbeaufsichtigt den Rechner manipulieren kann. Um einen Störungsfreien Betrieb zu gewährleisten, muss eine zuverlässige Internetverbindung bestehen. Somit wird eine Langfristige und Standfeste Verwendung des Produkts garantiert.

Dem Projektteam ist bewusst, dass Lieferrex ein Prototyp ist, und stand jetzt noch mehrere Bugs vorhanden sein könnten. Trotzdem soll es während der Betriebsphase möglich sein Fehler in der Software so schnell wie möglich mitteilen zu können. Falls vorhanden, sollte per E-Mail oder Telefon schnellstmöglich das Entwicklerteam helfen können.

17 Wartungsphase

Nach der erfolgreichen Übergabe und Einführung des Produktes startet die Wartungs- und Pflegephase. Hier ist das Projektteam zuständig, aufgetretene Fehler zu beheben und das Produkt zu verbessern. Dafür steht das Projektteam stets zur Verfügung. Im Rahmen der Wartungsarbeiten können Fehler ausgebessert, Funktionen verändert oder neu entwickelt werden. Im Falle einer Wartungsarbeit werden Wartungsanforderungen definiert. Daraus wird ein Lösungskonzept entworfen und entwickelt. Je nach Größe des Fehlers kann das System vorübergehend nicht verwendbar sein. Hier muss die Kommunikation zwischen Auftraggeber und Projektteam reibungslos funktionieren, um schnell reagieren zu können und die Funktionalität des Systems wieder herzustellen. Nach erfolgreichem Testen können die Korrekturen eingeführt werden. Anpasst wird auch die Dokumentation an die neuen Änderungen.

18 Zusammenfassung

Restaurants können durch unser entwickeltes Produkt schnell und bequem einen Auftritt im Internet erstellen. Sie bauen sich über einen Baukasten selber eine individuelle Webseite nach ihren Wünschen. Über diese erstellte Webseite können nun Kunden bei den Restaurants auf Lieferung oder zur Abholung bestellen. Dem Restaurant werden über ein Dashboard relevante Informationen und Statistiken wie beispielsweise Seitenaufrufe oder Umsatz angezeigt. Es werden auch aktuelle Bestellungen aufgelistet.

Als Backend wird Spring Boot, ein beliebtes und robustes Java Framework, verwendet. Mithilfe von Spring Boot werden Anfragen und Daten für alle Funktionen des Systems verarbeitet. Im Projekt wird das MVC-Pattern angewendet. Dieses Pattern sorgt für eine übersichtliche Struktur, indem Darstellung, Verarbeitung und Speicherung von Daten getrennt werden. Alle Informationen, die das System benötigt, werden in Form von Models oder auch Klassen dargestellt. So werden beispielsweise Kunden mit Namen, Adresse und weiterem angelegt. Diese werden in der MySQL Datenbank gespeichert. Ein Kontroller ist verantwortlich für die Verarbeitung von Informationen. Je nach Art der Anfrage, die der Kontroller erhält, liest, verarbeitet, löscht oder erstellt dieser Daten. Auf eine Anfrage an den Webserver antwortet ein Kontroller mit einer View. Dies ist eine Vorlage einer Webseite, die mit

entsprechenden Daten aus der Datenbank befüllt wird, um dynamisch Informationen darstellen zu können. Um die erhaltenen Daten auf der Seite anzuzeigen, wird Thymeleaf verwendet. Die Vorlagen für die Webseiten werden mit Platzhaltern ausgestattet, die später Thymeleaf mit den Daten des Kontrollers befüllt. Thymeleaf kann des Weiteren durch verschiedenste Funktionen auf verschiedenste Daten reagieren und die Vorlage dementsprechen anpassen. Auch gibt es eigene Kontroller, die als REST-Schnittstelle in Form einer API verwendet werden können. Mithilfe dieser API können zum Beispiel Daten auf der Webseite geladen werden, ohne diese neu laden zu müssen. Das fertige System verwendet auch externe API, wie Google Analytics für Statistiken, Google Maps für die Darstellung von Karten und Paypal für das Abwickeln von Zahlungen.

A. Anhang

Abbildungsverzeichnis

Abbildung 1: Stakeholder grafisch	4
Abbildung 2: Risikomatrix	7
Abbildung 3: Projektstrukturplan	10
Abbildung 4: Projektablaufplan	11
Abbildung 5: Mockup der Hompage	1
Abbildung 6 Mockup vom Baukastensystem	2
Abbildung 7: Kundenseite - Index Page - Modal	2
Abbildung 8: Kundenseite - Bestellungen Page	2
Abbildung 9: Kundenseite – Search-Seite - Desktop	3
Abbildung 10: Kundenseite – Search-Seite- Desktop	3
Abbildung 11: Kundenseite - Partner Page - Desktop	3
Abbildung 12: Kundenseite – User Register Page - Desktop	4
Abbildung 13: Kundenseite – Login Page - Desktop	4
Abbildung 14: Kundenseite – Passwort ändern Page - Desktop	4
Abbildung 15: Kundenseite – Adresse ändern Page - Desktop	5
Abbildung 16: Dashboard – Overview-Seite - mobile	5
Abbildung 17: Dashboard – Overview-Seite - Desktop	5
Abbildung 18: Dashboard – Zubereitungsseite - mobile	5
Abbildung 19: Dashboard - Zubereitungsseite - Desktop	5
Abbildung 20: Dashboard - Zustellungsseite - Desktop	6
Abbildung 21: Dashboard - Zustellungsseite - mobile	6
Abbildung 22: Dashboard - Gerichte Page - Desktop	6
Abbildung 23: Dashboard - Gerichte Page - mobile	6
Abbildung 24: Dashboard – Bewertungen-Seite - Desktop	6
Abbildung 25: Dashboard – Bewertungen-Seite - mobile	6
Abbildung 26: Dashboard - Benutzerrechte Page - Desktop	7
Abbildung 27: Dashboard - Benutzerrechte Page - mobile	7
Abbildung 28: Dashboard - Benutzerrechte Page - Zugriffsrechte	7
Abbildung 29: Dashboard – Zahlungen-Seite - Desktop	8
Abbildung 30: Dashboard – Zahlungen-Seite - mobile	8
Abbildung 31: Dashboard – Mandant-Seite - Desktop	8
Abbildung 32: Dashboard – Mandant-Seite - mobile	8
Abbildung 33: Dashboard – Öffnungszeiten-Seite - mobile	9
Abbildung 34: Dashboard – Öffnungszeiten-Seite - Desktop	9
Abbildung 35: Dashboard - Restaurantkategorien Page - Desktop	9
Abbildung 36: Dashboard - Restaurantkategorien Page - mobile	9
Abbildung 37: Use-Case-Diagramm Restaurant	3
Abbildung 38: Use-Case-Diagramm Kunde	1
Abbildung 39: Mockup - Startseite für Mobilgeräte	2
Abbildung 40: Mockup - Startseite für Desktops	2
Abbildung 41: Mockup - Startseite für Desktops – Dark Mode	1
Abbildung 42: Mockup - Ergebnisse für Desktops – Light Mode	1
Abbildung 43: Mockup - Ergebnisse für Desktops – Dark Mode	1
Abbildung 44: Mockup - Dashboard für Desktops – Light Mode	1
Abbildung 45: Mockup - Dashboard für Desktops – Dark Mode	1
Abbildung 46: Mockup - Dashboard für Desktops - Mitarbeiter - Dark Mode	2

Abbildung 47: Mockup - Dashboard für Mobilgeräte - Mitarbeiter - Dark Mode	2
Abbildung 48: Mockup - Registrierung für Desktops - Light Mode	2
Abbildung 49 C4-Diagramm, Context View.....	1
Abbildung 50: C4-Diagramm, Spring Applikation	2
Abbildung 51: Vollständiges ER-Diagramm	2
Abbildung 52: Spring Boot Security Konfiguration	Fehler! Textmarke nicht definiert.
Abbildung 53: User-Interaktion zwischen den Seiten.....	2
Abbildung 54 Legende zur User-Interaktionen Grafik	3
Abbildung 55: Farbpalette - Kundenseite - helles Theme	3
Abbildung 56: Kundenseite - Index Page - helles Theme.....	3
Abbildung 57: Farbpalette - Kundenseite - dunkles Theme.....	3
Abbildung 58: Kundenseite - Index Page - dunkles Theme	3
Abbildung 59: Farbpalette - Dashboard - dunkles Theme	4
Abbildung 60: Dashboard - dunkles Theme	4
Abbildung 61: Farbpalette - Dashboard - helles Theme	4
Abbildung 62: Dashboard - helles Theme	4
Abbildung 63: Farbverläufe - Dashboard - Infokarten	4
Abbildung 64: MVC-Pattern Spirng Boot.....	Fehler! Textmarke nicht definiert.
Abbildung 65: Konkrete Spring Boot Architektur	3
Abbildung 66: ER-Modell Baukasten.....	8
Abbildung 67: Kundenseite - Index Page	1
Abbildung 68: Kundenseite - Index Page - Aufbau	1
Abbildung 69: Hauptseite HTML-Struktur.....	1
Abbildung 70: main-search Klasse in style.css Datei.....	2
Abbildung 71: main-search style auf der Hauptseite (Dark Mode)	2
Abbildung 72: main-search Klasse in dark.css Datei	2
Abbildung 73: main-search Klasse in light.css Datei	2
Abbildung 74: main-search style auf der Hauptseite (Light Mode)	2
Abbildung 75: toggleSwitch Funktion / JavaScript	3
Abbildung 76: Cookie check für Dark Mode	3
Abbildung 77: ToDark Funktion	3
Abbildung 78: toLight Funktion	3
Abbildung 79: Dashboard - Benutzerrechte - Aufbau.....	4
Abbildung 80: Dashboard CSS-Variablen	4
Abbildung 81: Dashboard Halfmoon Variablen.....	4
Abbildung 82: Halfmoon eigener Style	5
Abbildung 83: Halfmoon Grundstruktur	5
Abbildung 84: Baukastensystem - Ansicht Restaurantbesitzer.....	6
Abbildung 85: Baukastensystem - Ansicht Kunde	6
Abbildung 86: Codebeispiel für die Darstellung im Modal.....	1
Abbildung 87 Ansicht des Angestellten.....	25
Abbildung 88 Mandanten Ansicht	26
Abbildung 89 Daten des eingeloggten User	27
Abbildung 90 Daten auslesen aus der Datenbank	27
Abbildung 91 OverviewController Attribute des Model hinzufügen	27
Abbildung 92 Zubereitung Seite	28
Abbildung 93 PHPMYADMIN Status von den Bestellungen	28
Abbildung 94 Codeausschnitt der Darstellung von den Daten mithilfe Thymeleaf	29
Abbildung 95 HashMap befüllen	29

Abbildung 96 ID Zuweisung des Buttons.....	30
Abbildung 97 PostMapping der Zubereitung	30
Abbildung 98 Zustellung Seite	30
Abbildung 99 Klassendiagramm des DTO ZustellungsModel	31
Abbildung 100 ZustellungController Wechsel von IN_AUSLIEFERUNG zu ABGESCHLOSSEN.	31
Abbildung 101 Gerichte Seite.....	31
Abbildung 102 Detailmodal des Gerichtes	32
Abbildung 103 JQuery funktion um die Felder zu befüllen.....	32
Abbildung 104 Gericht bearbeiten Modal.....	33
Abbildung 105 HTML mit Thymeleaf th:object	33
Abbildung 106 Gericht Status setzen.....	34
Abbildung 107 Bewertung Seite	34
Abbildung 108 Berechnung des Medians.....	35
Abbildung 109 Berechnung des Durschnittes anhand der Bewertungen.....	35
Abbildung 110 Berechnung Teilanzahl in % von der Gesamtanzahl	35
Abbildung 111 Bewertungsbalken.....	36
Abbildung 112 Volle Sterne erzeugen.....	36
Abbildung 113 Leere Sterne erzeugen.....	36
Abbildung 114 Benutzerrechte Seite	36
Abbildung 115 Zugang bearbeiten Modal.....	37
Abbildung 116 Rollen als String ins DTO laden.....	37
Abbildung 117 Darstellung der Zugänge	38
Abbildung 118 Angestellter in die Datenbank Speichern	38
Abbildung 119 Zahlungen Seite	38
Abbildung 120 Zahlungen Seite Zahlungen.....	39
Abbildung 121 Preis pro Bestellung.....	39
Abbildung 122 Gesamt Umsatz berechnen.....	39
Abbildung 123 Berechnung des Monat sowie Jahresumsatz	40
Abbildung 124 Die Letzten drei Bestellungen	40
Abbildung 125 Übergabe der Werte an JavaScript.....	40
Abbildung 126 Werte in ein normales Array pushen.....	41
Abbildung 127 Array reverse Funktion	41
Abbildung 128 Minimum Maximum Berechnung	41
Abbildung 129 Erstellung des Charts	41
Abbildung 130 Öffnungszeiten Seite.....	42
Abbildung 131 Klassendiagramm OeffnungsDarstellungModel.....	42
Abbildung 132 Thymeleaf Darstellung der Oeffnungszeiten	43
Abbildung 133 Speicherung von Oeffnungszeiten.....	43
Abbildung 134 Kategorie Seite	44
Abbildung 135 Setzen der Kategorie	44
Abbildung 136 Button anhaken zu dem dazugehörigen Wert	44
Abbildung 137 Speichern der Kategorie	45
Abbildung 138: Baukasten Aufbau	1
Abbildung 139: Baukasten Bearbeitungsmodus.....	1
Abbildung 140: Baukasten Frame	2
Abbildung 141: Baukasten Navigationsleiste Bearbeitungsmodus	2
Abbildung 142: Baukasten Addons	3
Abbildung 143: Baukasten Navigationsleiste Kundenansicht	3
Abbildung 144: Baukasten Ausgabe eines Fragments	4

Abbildung 145: Baukasten Ausgabe eines Headers.....	4
Abbildung 146: Baukasten Text-Fragment.....	5
Abbildung 147: Baukasten Image-Fragment.....	5
Abbildung 148: Baukasten - Gericht hinzufügen	6
Abbildung 149: Baukasten - Warenkorb Ausgabe	6
Abbildung 150: Baukasten - Gericht entfernen	7
Abbildung 151: Baukasten - Text-Fragment Modal	7
Abbildung 152: Baukasten - Text-Fragment Modal Bearbeitungsansicht	8
Abbildung 153: Baukasten - Frontend Aktionen	9
Abbildung 154: Baukasten - Modal öffnen.....	9
Abbildung 155: Baukasten - Eingabe Auswertung.....	10
Abbildung 156: Baukasten - Post-Request Speicherung	10
Abbildung 157: Baukasten - Speicherung der allgemeinen Einstellungen.....	11
Abbildung 158: Baukasten - Model Attribute.....	11
Abbildung 159: Baukasten - Model mit Fragmenten befüllen	12
Abbildung 160: Baukasten – Seitenaufrufe.....	12
Abbildung 161: Baukasten - Fragment Speicherung Ablauf.....	13
Abbildung 162: Baukasten - Ausnahme Header-Fragment.....	14
Abbildung 163: Baukasten - Text-Fragment Speicherung.....	14
Abbildung 164: Baukasten - Bild BLOB.....	14
Abbildung 165: Baukasten - Image String	14
Abbildung 166: Baukasten - Modul Rückgabe.....	15
Abbildung 167: Baukasten - Fragment löschen	15
Abbildung 168: Baukasten - Allgemeine Einstellungen	16
Abbildung 169: Baukasten - Layout ändern	16
Abbildung 170: Baukasten - Zusatz-Seite speichern.....	17
Abbildung 171: Baukasten - Zusatz-Seite entfernen	17
Abbildung 172: Baukasten - Zusatz-Seite Ausgabe.....	18
Abbildung 173: maven.yml.....	1
Abbildung 174: Procfile.....	1
Abbildung 175: Heroku Variablen Einstellungen	1
Abbildung 176: pom.xml Spring Profiles	2
Abbildung 177: application.properties Dateien	2
Abbildung 178: application-prod.properties	2
Abbildung 179: NAS MariaDB Einsatz	3
Abbildung 180: PhpMyAdmin User.....	3
Abbildung 181: NAS DDNS-Einstellungen	3
Abbildung 182: GitHub Repository, Readme File	5
Abbildung 183: GitHub - Kanban Board	5
Abbildung 184: GitGuardian, Merge Pull Request.....	5
Abbildung 185 Lieferrex, Code-Verteilung	6
Abbildung 186: Benutzerhandbuch Baukasten - Allgemeine Einstellungen	1
Abbildung 187: Benutzerhandbuch Baukasten - Layout Wahl.....	1
Abbildung 188: Benutzerhandbuch Baukasten - Kopfbereich bearbeiten.....	2
Abbildung 189: Benutzerhandbuch Baukasten - Wahl eines neuen Fragments	2
Abbildung 190: Benutzerhandbuch Baukasten - Text-Fragment erstellen	3
Abbildung 191: Benutzerhandbuch Baukasten - Speicherung bestätigen.....	3
Abbildung 192: Benutzerhandbuch Baukasten - Fragment entfernen	4
Abbildung 193: Benutzerhandbuch Baukasten - Erstellung der Zusatz-Seite Galerie	4

Abbildung 194: Benutzerhandbuch Baukasten - Erstellung der Zusatz-Seite "Über uns"	5
Abbildung 195: Benutzerhandbuch Baukasten - Zusatz-Seiten löschen.....	5

Tabellenverzeichnis

Tabelle 1: Stakeholder	3
Tabelle 2: Legende Stakeholder grafisch.....	4
Tabelle 3: Stakeholder Maßnahmen.....	5
Tabelle 4: Risikoportfolio Teil 1	6
Tabelle 5 Risikoportfolio Teil 2	7
Tabelle 6: Legende Risikoportfolio grafisch.....	7
Tabelle 7: Use-Case "Essen bestellen"	1
Tabelle 8: Use-Case "Baukastensystem"	2
Tabelle 9: Use-Case "Anmelden"	3
Tabelle 10: Verwendete Frontend-Technologien.....	1
Tabelle 11: Frontend-Technologien Versionen	1
Tabelle 12: REST Methoden	4
Tabelle 13: Baukasten Fragment-Typen.....	4
Tabelle 14 Stammdaten des Mandanten ändern	1
Tabelle 15 Bestellung tätigen	2
Tabelle 16 Öffnungszeiten des Restaurants ändern	3
Tabelle 17 Webseite im Baukasten anlegen	4
Tabelle 18 User Registrierung und Login	5
Tabelle 19 Zeitaufstellung	7

Quelltexte

Baeldung. (März 2022). Von <https://www.baeldung.com> abgerufen

Computer Weekly. (Mai 2022). Von <https://www.computerweekly.com/de/definition/Systemtest> abgerufen

Google Maps Platform. (März 2022). Von <https://developers.google.com> abgerufen

Halfmoon dokumentation. (April 2022). Von <https://www.gethalfmoon.com/docs> abgerufen

JQuery documentation. (März 2022). Von <https://api.jquery.com/> abgerufen

Materialize documentation. (April 2022). Von <https://materializecss.github.io/materialize/> abgerufen

Materialize Stepper documentation. (März 2022). Von shorturl.at/pPT89 abgerufen

PayPal API documentation. (März 2022). Von <https://developer.paypal.com/api/rest/> abgerufen

Thymeleaf documentation. (März 2022). Von <https://www.thymeleaf.org/> abgerufen

Wikipedia. (Mai 2022). Von [https://de.wikipedia.org/wiki/Akzeptanztest_\(Softwaretechnik\)](https://de.wikipedia.org/wiki/Akzeptanztest_(Softwaretechnik)) abgerufen