

Projektarbeit der Vorlesung

“Mobile Technologien”

von

Alexander Ruppel,

Lukas Müller

und

Manuel Naumann

# 1 Einleitung

## 1.1 Produktbeschreibung

Peters Foodtruck ist eine mobile Anwendung für Android Geräte, welche sowohl für Betreiber, als auch Kunden ausgelegt ist. Sie ermöglicht dem Kunden oder dem Betreiber nach einer Registrierung Informationen über Foodtrucks abzurufen bzw. diese als Betreiber anzugeben.

Dem Betreiber soll die Anwendung ermöglichen eine interaktive Bindung zum Kunden aufzubauen, indem er seinen Standort und die Preise an die Bedürfnisse seiner Kunden anpassen kann. Zudem ist es ihm möglich durch die bessere Planung, anhand von Vorbestellungen mit hilfe der App Kosten und Lebensmittelverschwendung zu vermeiden.

Der Kunde kann mit der App den Verlauf und die geplante Route des Trucks verfolgen und somit seine Essenszeit an den für ihn besten Standort anpassen. Durch die Möglichkeit der Vorbestellung können längere Wartezeiten am Foodtruck vermieden werden und durch eine Onlinebezahlung ist sogar das Bargeldlose bezahlen möglich.

Ziel ist es also vor allem die Interaktion zwischen Kunden und Betreiber zu stärken, was zu einer Komfortverbesserung für beide Seiten führen soll. Auch die Umwelt soll langfristig von der App profitieren, da ein besseres Ressourcenmanagement ermöglicht wird.

## 1.2 Herangehensweise

Die Gruppe hat sich dazu entschieden, für die Koordinierung der anfallenden Aufgaben nach dem Kanban-Prinzip vorzugehen. Genutzt wurde dafür Trello.com. Trello ermöglicht die kostenlose Erstellung von Kanban Boards. Es können Listen und Karten erstellt werden mit Tasks und Subtasks. Diese können wiederum einzeln Mitgliedern eines Boards zugewiesen werden und so Stück für Stück abgearbeitet werden. Auch ein Fristen und Kommentare sind möglich.

Um das gemeinsame Programmieren innerhalb des Projektes so einfach wie möglich zu gestalten und den Austausch zwischen den Gruppenmitgliedern zu ermöglichen wurde GitHub eingesetzt. Mit GitHub gibt es ein externes Repository auf dem die Projektdateien gespeichert werden. Diese können lokal geklont, anschließend bearbeitet und dann wieder auf das Repository geschoben werden. Mit mehreren Branches ist so ein gleichzeitiges bearbeiten und eine Versionskontrolle möglich. In diesem Projekt wurde sich dazu entschieden, mit einem Masterbranch und jeweils einem Branch für jeden Teilnehmer zu arbeiten.

Die Aufgaben des Projekts wurden in mehrere Teilbereiche unterteilt. Nachdem zunächst die Anforderungen ermittelt wurden ergaben sich für die Umsetzung der Projektarbeit die Bereiche Datenbank und Datenbankanbindung, REST-Backend, Frontend und Dokumentation. Diese wurden nach bisherigen Erfahrungen auf die Gruppenmitglieder aufgeteilt. Im Frontendbereich wurde außerdem eine weitere Einteilung in verschiedene Bereiche des Frontends vorgenommen. Da der Plan war nur eine App für den Betreiber und die Kunden zu erstellen erschien es zum Beispiel Sinnvoll hier jeweils getrennt zu entwickeln. Allerdings wurde auch hier zunächst der visuelle Teil umgesetzt und abgestimmt, um eine einheitliche Gestaltung der App zu haben.

## 1.3 Anforderungen

Um die Anforderungen aus der gegebenen Beschreibung möglichst gut abzubilden wurden Verfahren aus dem Requirements Engineering verwendet.

Aus der Aufgabenstellung wurden die Anforderungen mithilfe von Satzschablonen extrahiert. Mithilfe von Satzschablonen ist eine einfache Erfassung von Anforderungen möglich. Außerdem ermöglichen diese mit einfachen sprachlichen Mitteln die Anforderung feingranularer, als in den vorher gegebenen Absätzen abzubilden. Auch bilden sie durch ihre strukturierte Herangehensweise Missverständnissen vorzubeugen.

Die so ermittelten Anforderungen wurden anschließend in die Kategorien “Muss” und “Soll” untergliedert und absteigend priorisiert. “Muss”-Anforderungen sind fest vorgenommene Zielanforderungen für dieses Projekt. “Soll”-Anforderungen sollen dagegen nur Umgesetzt werden, falls es die Zeit zulässt. Mit einer Aufteilung von 19 “Muss”-Anforderungen zu 8 “Soll”-Anforderungen zeigt sich, dass der Großteil der ursprünglichen Aufgabenstellung erfüllt werden soll. Die Gruppe hat sich zu diesem Vorgehen entschlossen, da eine Anwendung ohne viele der “Muss”-Anforderungen unfertig erschien und wichtige Funktionen fehlen würden.

Nachfolgend finden sich diese zwei Anforderungslisten.

**Muss**

1. Das System muss Kunden die Möglichkeit bieten, sich in der mobilen Anwendung zu registrieren.
2. Das System muss dem Betreiber die Möglichkeit bieten, Speisen aufzunehmen.
3. Das System muss dem Kunden die Möglichkeit bieten, bis zu einem gewissen Zeitpunkt Vorbestellungen anhand der Speisekarte aufzugeben.
4. Das System muss in der Lage sein, eine Bestellung einem Standort zuzuordnen.
5. Das System muss dem Betreiber die Möglichkeit bieten, die Lieferbarkeit von Speisen zu editieren.
6. Das System muss dem Betreiber die Möglichkeit bieten, eine Route mit Standorten anzulegen.
7. Das System muss dem Betreiber die Möglichkeit bieten, den Ankunftszeitpunkt und die Aufenthaltsdauer mit einem möglichen Zeitfenster zu versehen.
8. Das System muss dem Betreiber die Möglichkeit bieten, eine Einkaufsliste anhand der aktuellen Vorbestellungen zur Verfügung zu stellen.
9. Das System muss dem Betreiber die Möglichkeit bieten, sich alle Bestellungen anzusehen.
10. Das System muss dem Betreiber die Möglichkeit bieten, die Menge der Portionen auf der Einkaufsliste mit einem Zuschlag zu erhöhen.
11. Das System muss dem Kunden die Möglichkeit bieten, bis zum jeweiligen Bestellschluss am Standort aus den Zuschlag-Portionen eine Bestellung aufzugeben.
12. Das System muss dem Kunden die Möglichkeit bieten, die preisdynamische Speisekarte einzusehen.
13. Das System muss dem Betreiber die Möglichkeit bieten, die Preise der Speisen mit Rabatten oder Zuschlägen zu versehen.
14. Das System muss fähig sein, je nach Erfolg des Einkaufs im Großmarkt, die Bestellungen der Kunden auf “Bestätigt” oder “Nicht möglich” zu setzen.
15. Das System muss dem Kunden die Möglichkeit bieten, seine Bestellungen in der mobilen Anwendung zu verfolgen.
16. Das System muss in der Lage sein, allen Bestellungen des Tages eine fortlaufende Nummer zu geben.
17. Das System muss in der Lage sein, den Kunden über die Fertigstellung seiner Bestellung mit einer Bestellnummer zu informieren.
18. Das System muss fähig sein, nach Abschluss der Zubereitung die Bestellung mit “Zubereitung abgeschlossen” zu markieren.
19. Das System muss dem Betreiber die Möglichkeit bieten, Bestellungen auszuwählen und mit der Zubereitung zu beginnen.

**Soll**

1. Das System soll dem Kunden die Möglichkeit bieten, bei der Registrierung ihren Standort anzugeben.
2. Das System soll in der Lage sein, dem Kunden die nächsten drei Standorte des Food-Trucks mitzuteilen.
3. Das System soll dem Betreiber die Möglichkeit bieten, am aktuellen Standort einen Bestellschluss in x-Minuten auszulösen.
4. Das System soll verhindern, dass ein Kunde nach dem Bestellschluss eine Bestellung für den jeweiligen Standort auslösen kann.
5. Das System soll in der Lage sein, dem Kunden die Bewertung der Speisen zu ermöglichen.
6. Das System soll dem Betreiber die Möglichkeit bieten, die Bestellungen nach Standort zu filtern.
7. Das System soll dem Kunden die Möglichkeit bieten, den Food-Truck auf seiner Route zu verfolgen.
8. Das System soll in der Lage sein, den Kunden beim Eintreffen des Food-Truck am Standort seiner Bestellung zu informieren.

# 

# 2 Umsetzung

## 2.2 Architektur

### 2.2.1 Generelles

Im Rahmen dieser Projektaufgabe wurde sich für die Teilung der Anwendung in Frontend und Backend entschieden.

Diese Entscheidung wurde direkt zu Beginn des Entwicklungsprozesses mit einem zentralen Grund getroffen:

Teile der Projektgruppe hatten bereits vorherige Erfahrungen mit Backends und den in ihnen verwendeten Technologien gesammelt, während äquivalente Konzepte unter Android erst im Rahmen des Moduls “Mobile Technologien“ vorgestellt wurden.

So konnte durch die Trennung der Anwendung in zwei separate Teile bereits früh begonnen werden.

Eine vorgezogene Entwicklung des Backends mit Datenmodell, API-Layer, Persistenzschicht und Logik erschien so als attraktives Konzept - besonders unter Betrachtung des relativ knappen Zeitraums bis zur Projektabgabe.

Nichtsdestotrotz bringt die Trennung von Logik und Benutzeroberfläche auch einen gewissen Overhead mit sich. Im Gegensatz zu hybriden Anwendungen muss bei einer getrennten Anwendungsarchitektur die Kommunikation zwischen Frontend und Backend organisiert und implementiert werden. Dies resultiert natürlich in Mehraufwand, kann aber auch Ursache für schwer erkennbar Fehler sein.

Letztlich wurde sich dennoch für eine getrennte Anwendungsarchitektur entschieden, da die so gewonnene Zeit nach Ansicht der Projektgruppe die Nachteile des Ansatzes überwog.

### 2.2.2 Backend

Das Backend basiert auf den Technologien Spring Boot, Hibernate und Spring Web.

Es übernimmt die Definition des Datenmodells, die Persistierung der Daten sowie die Logik der einzelnen Komponenten. Weiterhin erlaubt es die Interaktion aus der App über eine REST-API.

Als Teil des von Pivotal angebotenen Spring-Frameworks agiert SpringBoot als Basis des Backends. Spring Boot vereinfacht dabei Entwicklung und Deployment von Java-Anwendungen, in dem es unter anderem viele Konfigurationsaufgaben für den Entwickler übernimmt. Zum Beispiel werden Versionen verschiedener Maven-Dependencies des Projekts automatisch auf untereinander kompatible Versionen festgelegt.

Hibernate ist das im Projekt verwendete ORM-Framework (**O**bject-**R**elational- **M**apping-Framework), welches klassische POJOs (**P**lain-**O**ld-**J**ava-**O**bjects) in relationale Tabellen abbildet. Hibernate ist ebenfalls in Spring-Boot integriert, sodass auch hier nur ein minimaler Konfigurationsaufwand nötig ist: Statt einer persistence.xml reicht es aus, in der SpringBoot eigenen application.properties-Datei Datenbank-URL, Benutzername und Passwort anzugeben. Zudem besitzt Hibernate neben Standard-JPA-Funktionen auch weitere, eigene Features wie zum Beispiel die Hibernate Query Language oder die Interface-basierte Datenbank-Interaktion.

Spring Web ermöglicht das komfortable Erstellen von REST-APIs und vervollständigt so das Backend. Mittels Spring Web können API-Endpunkte sowie Pfad- und Request-Parameter über Annotationen erstellt und ausgelesen werden. Spring Web bietet zudem die automatische Übersetzung von und nach JSON-Dateien (**J**ava**S**cript **O**bject **N**otation). Als fester Bestandteil des ursprünglichen Spring-Frameworks integriert sich auch Spring Web in Spring Boot.

Die Datenbank wird mittels Docker über einen Container bereitgestellt. Docker ist eine auf den allermeisten Betriebssystemen verfügbare Software zum Erstellen und Verwenden containerisierter Anwendungen. Aufgrund der Containerisierung ermöglicht Docker ein einfaches und schnelles Installieren der benötigten Anwendungen.

Als Datenbank wurde aufgrund vorheriger Erfahrungen im Bereich des Moduls Datenbanksysteme MySQL gewählt. Beim Starten des Backends wird - sofern noch kein Datenbank-Container verfügbar ist - automatisch ein neuer Docker- Container mit einer MySQL-Datenbank und dem benötigten Schema erstellt und gestartet. Weiterhin werden automatisiert einige Testdaten erstellt und in der Datenbank abgespeichert.

### 2.2.3 Frontend

Das Frontend der Anwendung stellt eine Android Application dar, welche in der Entwicklungsumgebung „Android Studio“, in der Sprache Java geschrieben wurde. Hierbei handelt es sich um eine IDE, welche ein hoch integriertes Gradle-Build-System nutzt und damit das Dependency-Management vereinfacht.

Die Anwendung wird von Kunden („Customer“) sowie von Betreibern („Operator“) eines Food-Trucks genutzt. Da diese Nutzergruppen unterschiedliche funktionale Anforderungen an das Frontend stellen, wurden sie auch in der Architektur voneinander getrennt. Es ergab sich folglich eine Struktur, welche sich in die Zweige Customer und Operator trennte.

Das entworfene Sitemap-Diagramm (siehe Anhang) visualisiert diesen Aufbau anschaulich und zeigt ebenso ein weiteres Kernkonzept der Frontend-Architektur auf:

Jede Activity des Programms geht im jeweiligen Nutzerzweig von einer zentralen Activity aus. Dies betrifft nicht nur die beiden Hauptmenüs mit ihren Menüpunkten, sondern auch die Menüpunkte selbst, mit allen Funktionen, die diese beinhalten. So gehen z.B. alle Operator-Funktionen, die die Speisekarte und ihren Inhalt betreffen, von der Activity „Speisekarte bearbeiten“ aus und kehren zu dieser nach Erledigung zurück.

Die Kommunikation mit dem Backend erfolgt mittels HTTP-Methoden. Hierzu werden innerhalb der Anwendung GSON-Requests erstellt. Um die Erstellung dieser Requests zu vereinfachen wurden die Klassen „GsonRequest“ und „DataService“ erstellt, welche Parameter und Methoden bereitstellen.

Die Daten innerhalb der Requests werden in Data Transfer Objects gekapselt und an das Backend gesendet. Hierzu wurden die relevanten Models aus dem Backend ins Frontend überführt, damit diese in den Activities auf einfache Weise für die Erstellung der GSON-Requests genutzt werden können. DTOs sind besonders praktisch, wenn Methoden mehrere Parameter benötigen. In diesem Fall kann dann einfach nur das genutzte Objekt mit allen Eigenschaften übergeben werden. Außerdem verringert sich die anfallende Datenmenge, welche zwischen Front- und Backend ausgetauscht werden muss.

## 2.3 Datenmodell

## 2.4 Sitemap

Um die in Kapitel 2.3 ermittelten Anforderungen innerhalb des Android-Frontends abzubilden wurden zwei Sitemaps für die Kundensicht und die Betreibersicht ohne grafische Details erstellt. Mit diesen wurden alle nötigen Activitys ermittelt. Auch die Übergänge zwischen den einzelnen Activitys ist dadurch ersichtlich. Diese beiden ersten Sitemaps ist im Anhang dieses Dokuments ersichtlich.

Aufbauend auf dieser ersten Sitemap wurden anschließend die Activitys noch ohne Funktionen erstellt. Diese waren also reine Oberflächen mit gedachten Übergängen. Dadurch haben sich zum Teil die vorher ermittelten Activitys nochmals geändert bzw. wurden zusätzliche hinzugefügt. Mit den in Android Studio erstellten Activitys wurde anschließend erneut eine Sitemap erstellt, welche nun auch das Aussehen der einzelnen Activitys festhält und die Übergänge zeigt. Eine vorher nicht geplanten Funktionen ist zum Beispiel der Header der Anwendung. Dieser dient gleichzeitig als eine Art Homebutton funktionieren soll und den Benutzer somit immer zurück zur Startseite führt. Diese Sitemap ist in Abbildung XXXX dargestellt.

Sitemap

# 3 Ergebnis

Dadurch, dass die Gruppe sich das Ziel gesetzt hat, möglichst alle Anforderungen umzusetzen und alle Funktionen abzubilden, konnte aufgrund des gegebenen zeitlichen Rahmens nur begrenzt ins Detail gegangen werden. So ist zum Beispiel das Oberflächendesign rein funktional umgesetzt und nicht mit Animationen oder sonstigen Details ausgestattet.

## 3.1 Backend

### 3.1.1 Datenmodell

Das Datenmodell basiert zum größten Teil auf den im Anforderungsobjekt definierten Archetypen. Insgesamt gibt es vier Untergruppen:

1. User: Die User-Klasse ist eine abstrakte Superklasse für den Operator und den Customer. Während in der User-Klasse generelle Attribute wie Benutzername oder Passwort gespeichert werden, enthalten die Spezialisierungen entsprechend eigene Attribute und Logik.
   1. Die Customer-Klasse hat keine weiteren eigenen Attribute.
   2. Im Gegensatz zur Customer-Klasse besitzt die Operator-Klasse eine Vielzahl von Attributen - zum Beispiel Listen für die Route des Foodtrucks oder dessen Menü. Der Operator bietet ebenfalls eigene Methoden zur Interaktion mit seinen Attributen. Beispielsweise müssen, wenn die Abfahrtszeit eines Standorts der Route geändert wird, auch die Ankunfts- und Abfahrtszeiten der darauffolgenden Standorte angepasst werden.
2. Bestellung: Ähnlich der User-Klasse ist auch die Bestellung-Klasse(Order) eine abstrakte Superklasse für Vorbestellungen(PreOrder) und Reservierungen (Reservation). Jede Order ist immer einem Kunden und einem Standort zugeordnet und beinhaltet eine Liste der bestellten Gerichte. Anders als beim User implementieren die Subklassen keine weiteren Funktionen, sondern dienen lediglich der Trennung von Vorbestellungen und Reservierungen.
3. Standort: Die Standort-Klasse(Location) ist ein zentraler Punkt des gesamten Datenmodells. Zum einen bildet sie über ein Koordinatenpaar eine simulierte geographische Position ab. Als Teil dieser Simulation bietet die Klasse auch Methoden, mit denen der Abstand zweier Standorte sowie, basierend auf einer vorher festgelegten Durchschnittsgeschwindigkeit, die Dauer der Fahrt zwischen zwei Standorten berechnet werden können. Diese Funktionalität wird beim Erzeugen neuer Standorte berücksichtigt: unter Angabe des vorherigen Standortes wird automatisch überprüft, ob die für den neuen Standort angegebene Ankunftszeit einhaltbar ist. Die Ankunftszeit wird weiterhin, wenn nötig, automatisch an die Fahrzeit zwischen beiden Standorten angepasst. Zum anderen speichert der Standort eine Liste aller für ihn aufgegebenen Bestellungen in zwei Listen - eine für Vorbestellungen und eine für Reservierungen. Diese Datenstruktur ermöglicht eine Detailansicht der jeweiligen Bestellungen eines Standorts, erlaubt gleichzeitig aber auch einen einfache Überblick über alle Bestellungen der festgelegten Route.
4. Gerichte und Zutaten: Die Gericht-Klasse(Dish) repräsentiert die jeweiligen Gerichte eines Foodtrucks und ist so immer einem Operator zugeordnet. Das Gericht selbst besteht aus einem Namen sowie einer Liste von Zutaten(Ingredients). Jedes Ingredient hat wiederum einen Namen und eine Anzahl, mit der es im Gericht verwendet wird.

### 3.1.2 Persistenz

Wie in Abschnitt 2.2.2 erwähnt, wird das Hibernate-ORM-Framework für das Persistieren der Daten in die Datenbank verwendet.

Hibernate stellt für die Interaktion mit der Datenbank sogenannte Repositories zur Verfügung. Repositories sind normale Java-Interfaces, welche von der Klasse JpaRepository erben. Jede Klasse des Datenmodells bekommt ein Repository zugeordnet. Über diese können im Anschluss Daten der Klasse gespeichert, aktualisiert oder gelöscht werden. Außerdem können im Interface verschiedene Methoden zur weiterführenden Datenbank-Interaktion definiert werden.

Damit Hibernate aber versteht, wie genau die Objekte der Java-Welt auf relationale Tabellen abzubilden sind, muss das Datenmodell mit Annotationen ergänzt werden. Diese Annotation definieren zum Beispiel die ID einer Klasse oder wie genau Beziehungen zu anderen Objekten abzubilden sind.

In einem ersten Entwurf des Backends wurde Anstatt der jetzigen Objekt-Kapselung Java-Maps verwendet. Während standardmäßige Beziehungen zu anderen Objekten meist über Listen oder Sets abgebildet werden und einfach zu annotieren sind, war für die Persistenz einer Map eine komplizierte Kombination von Annotationen nötig, weswegen diese Architektur letztlich verworfen wurde.

### 3.1.3 DTO- und REST-Schicht

DTO ist ein Akronym für **D**ata-**T**ransfer-**O**bject. DTOs sind vereinfachte Repräsentationen der Klassen des Datenmodells. Die Verwendung von DTOs bietet zwei zentrale Vorteile:

Zum einen wird durch die Verwendung einer DTO-Schicht einer SQL-Injection-Attacke vorgebeugt. Während bei einer REST-Api ohne DTO-Schicht zum Beispiel schädlicher Code anstelle eines zu persistierenden Objekts versendet werden kann, bietet eine Kapselung durch DTOs die Möglichkeit einer weitergehenden Kontrolle.

Zum anderen sind DTOs durch ihre Vereinfachung perfekt für den Datenaustausch über HTTP, da sie nur die wirklich benötigten Objekteigenschaften(zum Beispiel keine Objekt-Methoden) übertragen und so die ausgetauschten Daten auf ein Minimum reduzieren.

REST ist ein Akronym für **R**epresentational **S**tate **T**ransfer. Mit dem Ziel die Anforderungen des modernen Webs besser zu erfüllen, ist REST ein Programmierparadigma, an dem sich vor allem Client-Server-Architekturen orientieren. Anders als vergleichbare Verfahren(wie zum Beispiel SOAP) bietet REST eine Reihe verschiedener Standardmethoden an, die auf dem Server vom Client aufgerufen werden können. Diese Standardmethoden werden in der Praxis über die Verwendung diverser HTTP-Methoden(zum Beispiel GET oder POST) abgebildet.

Im Projekt wird REST zur Kommunikation zwischen App und Backend verwendet.

Damit die App die von REST definierten Methoden benutzen kann, müssen diese als Endpunkt im Backend bereitgestellt werden. Zuerst werden hierfür sogenannte REST-Controller angelegt, welche verschiedene Endpunkte logisch gruppieren (beispielsweise nach Klassen). In den jeweiligen Controllern werden anschließend die eigentlichen Endpunkte definiert. Jeder Endpunkt bekommt hier eine HTTP-Methode und einen bestimmten Pfad zugeordnet. Während REST-Controller eine auf eigene Klasse abgebildet werden, sind Endpunkte Methoden, die bei Erhalt einer entsprechenden HTTP-Methode zum definierten Pfad automatisch vom Spring-Framework aufgerufen werden.

Beispielsweise wird im REST-Controller CustomerController unter anderem ein Endpunkt mit dem Pfad /api/customer und der Methode PUT definiert, über welchen neue Kunden angelegt werden können. Die Logik zum Übersetzen der DTO-Objekte und zum Anlegen der Kunden befindet sich im Methodenkörper.

Standardmäßig wird in REST JSON für den Datenaustausch verwendet. Generell gilt die Best-Practice, dass alle Datentypen - Primitive und Komplexe - in JSON-Objekte umgewandelt werden. Im Backend wird das Serialisieren und Deserialisieren automatisch von Spring Web übernommen.

Im Frontend wurde für die Kommunikation über HTTP die von Google entwickelte Bibliothek Volley verwendet. Volley ermöglicht hierbei eine enge Verknüpfung zu GSON, einer weiteren von Google entwickelten Technologie. Während Volley das eigentliche Senden und Empfangen der HTTP-Befehle übernimmt, ist es GSONs Aufgabe, die erhaltenen und zu sendenden Nachrichten zu Serialisieren beziehungsweise zu Deserialisieren.

Genau hier kam es auch zu diversen Problemen bei der Kommunikation zwischen Frontend und Backend: Da SpringWeb und Volley/GSON unterschiedliche Bibliotheken von unterschiedlichen Entwicklern sind, gab es besonders bei exotischen oder komplexen Datentypen Unterschiede in deren Serialisierung und Deserialisierung. Zum Beispiel sendete das Backend Objekte der Klasse java.time.Duration als einen String, während GSON in der App ein Array mit Integern erwartet hätte. So musste für diverse Ausnahmefälle eine eigene Serialisierungs- und Deserialisierungs-Logik implementiert werden musste.

## 3.2 Frontend

Bei dem Design des Frontends wurde sich für eine einfache und klare Ausrichtung entschieden. Die verwendeten Farben beschränken sich auf eine kleine Palette von weißen, grauen und schwarzen Farben. Lediglich für das Logo, welches auffallen soll, wurde in Orange-Töne ausgesucht.

Auch das Layout aller Sites wurde einheitlich programmiert. Zu beginn jeder Seite findet sich im „Header“ das Logo der Food-Truck-App als ImageButton wieder. Es dient dem Benutzer gleichzeitig als bequeme Möglichkeit in das jeweilige Hauptmenü zurückzukehren – angemeldete Kunden werden in die „CustomerMenuActivity“ und Betreiber in die „OwnerMenuActivity“ navigiert. Dies wird erreicht, indem jede Activity im Programm eine Funktion „getToHome“ besitzt, welche die aktuelle Activity zur Hauptmenü-Activity wechseln lässt.

Im Operator-Zweig der Anwendung befindet sich im Header zusätzlich ein Zurück-Button, welcher die Benutzererfahrung weiterhin verbessert. Der Header wird vom eigentlichen Inhalt jeder Site durch einen Divider, einer horizontalen Linie in grauer Farbe, getrennt.



Die nachstehende Abbildung 3.2.1 zeigt das beschriebene Grundlayout einer jeden Site der Food-Truck-App.

Es handelt sich also insgesamt um ein funktionales Design, welches dem Benutzer bei seiner Navigation durch das Programm unterstützen soll und die Anwendung einfach zu verstehen macht.

Abb. 3.2.01 Blankes Grundlayout

Wie bereits in der Architekturbeschreibung des Frontends unter „2.2.3 Frontend“ beschrieben, spaltet sich die Anwendung in die Zweige Customer und Operator. Die Grundfunktionen für Login und Registrierung teilen sich die Zweige jedoch, da diese nicht speziell auf eine Benutzergruppe zugeschnitten sein müssen. Um beim Login/Registrierung zwischen einem Kunden und einem Betreiber unterscheiden zu können, wird der Nutzer in einer vorgestellten Ansicht aufgefordert anzugeben, welcher Benutzergruppe er angehört.

Bei einem erfolgreichen Login wird der Kunde bzw. Betreiber in sein jeweiliges Hauptmenü navigiert (Siehe Abbildung „3.2.2 Hauptmenü Operator“ und „3.2.3 Hauptmenü Customer“).

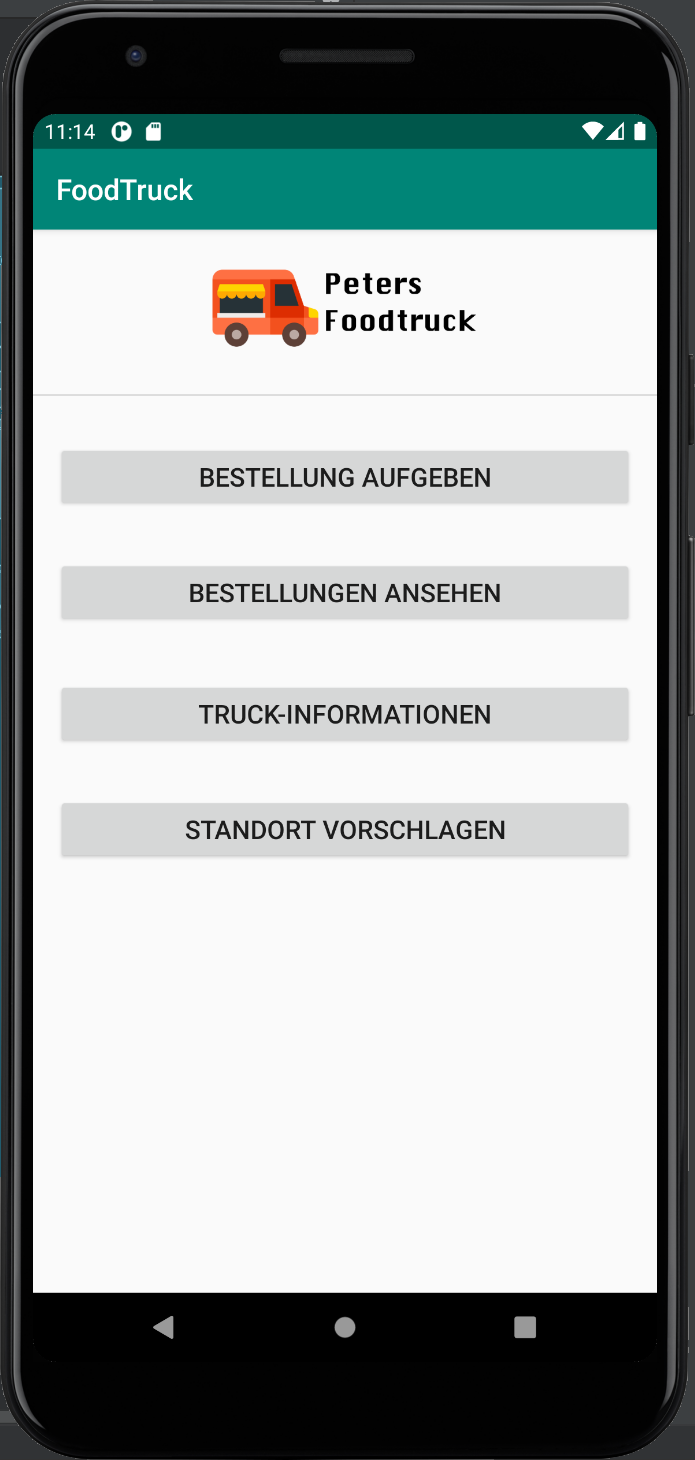
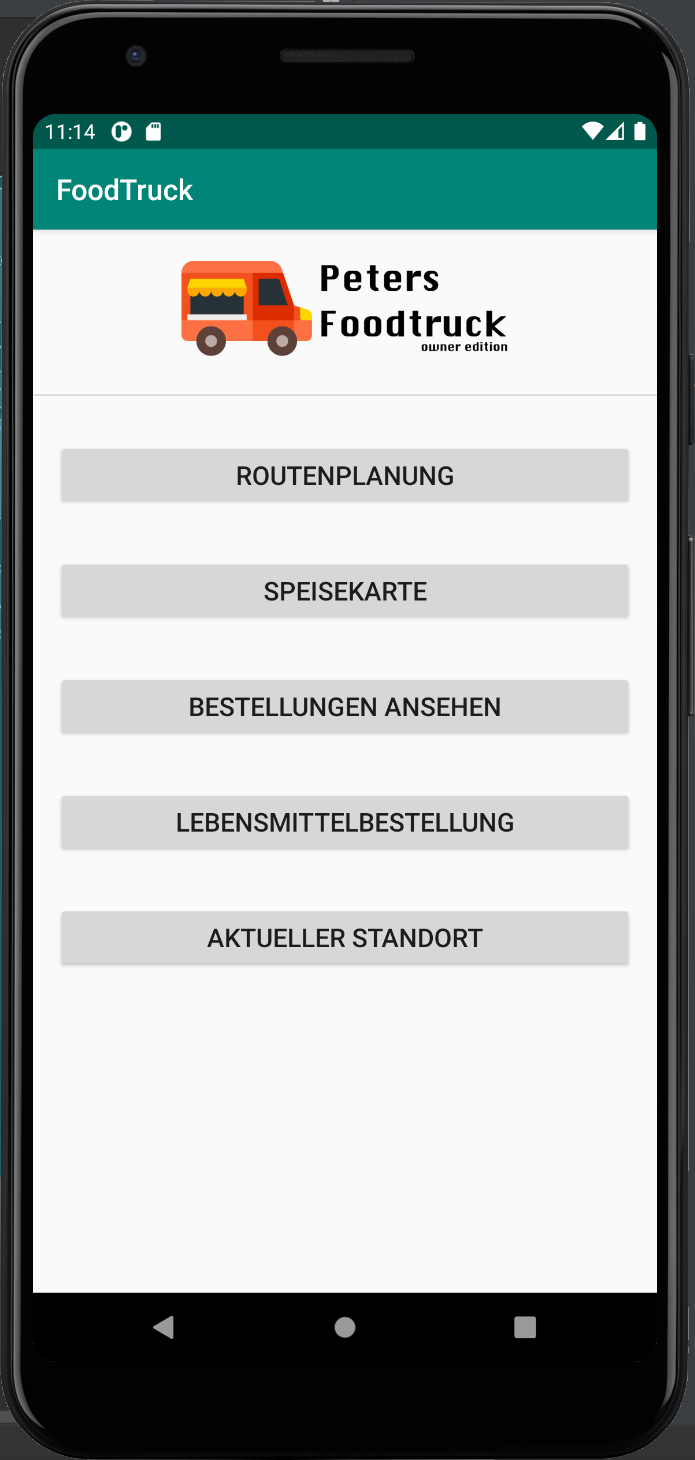


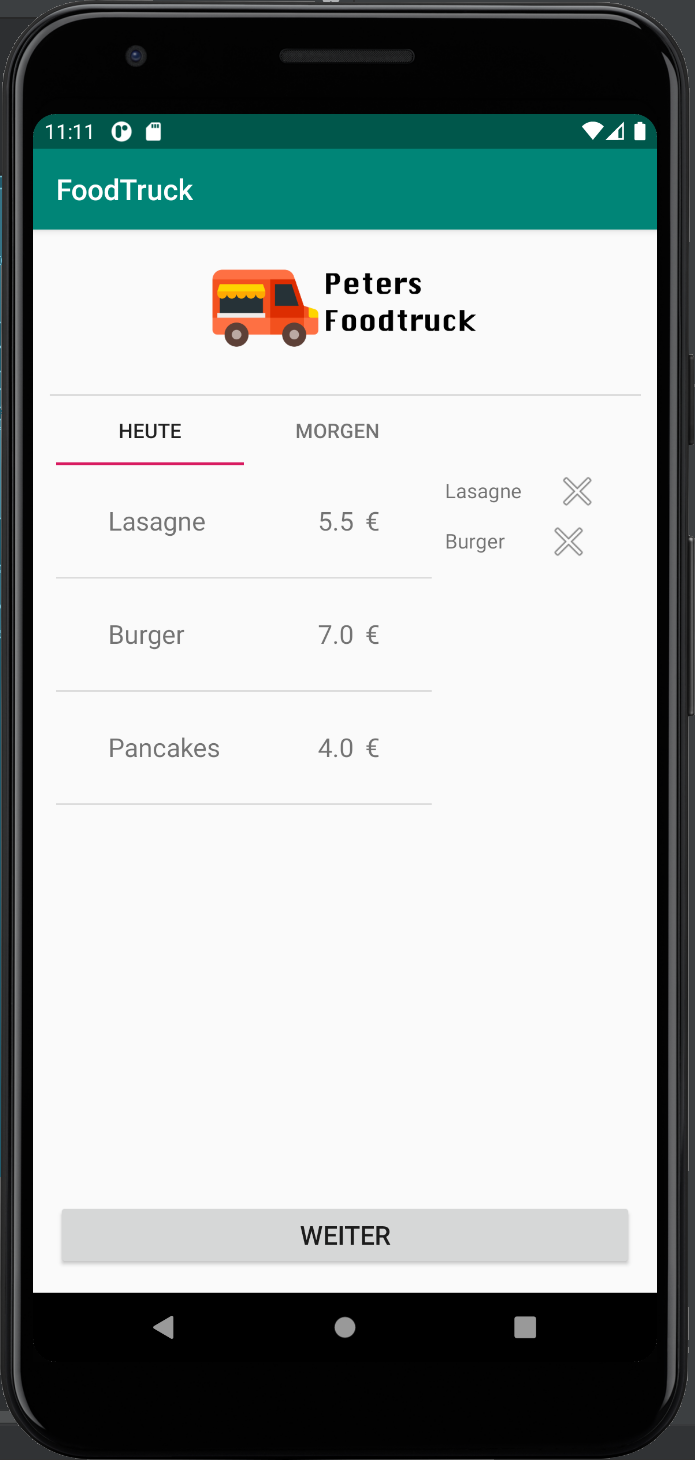
Abb. 3.2.02 Hauptmenü Operator Abb. 3.2.03 Hauptmenü Customer

**CustomerMenuActivity**

Ausgehend von der CustomerMenuActivity (Hauptmenü eines Kunden) erreicht man Activities für das Aufgeben einer Bestellung, das Nachverfolgen der eigenen Bestellungen und das Anzeigen der Truck-Infos des Betreibers sowie eine Activity, mit welcher man sich einen Standort des Food-Trucks vorschlagen lassen kann, indem man seinen eigenen angibt.

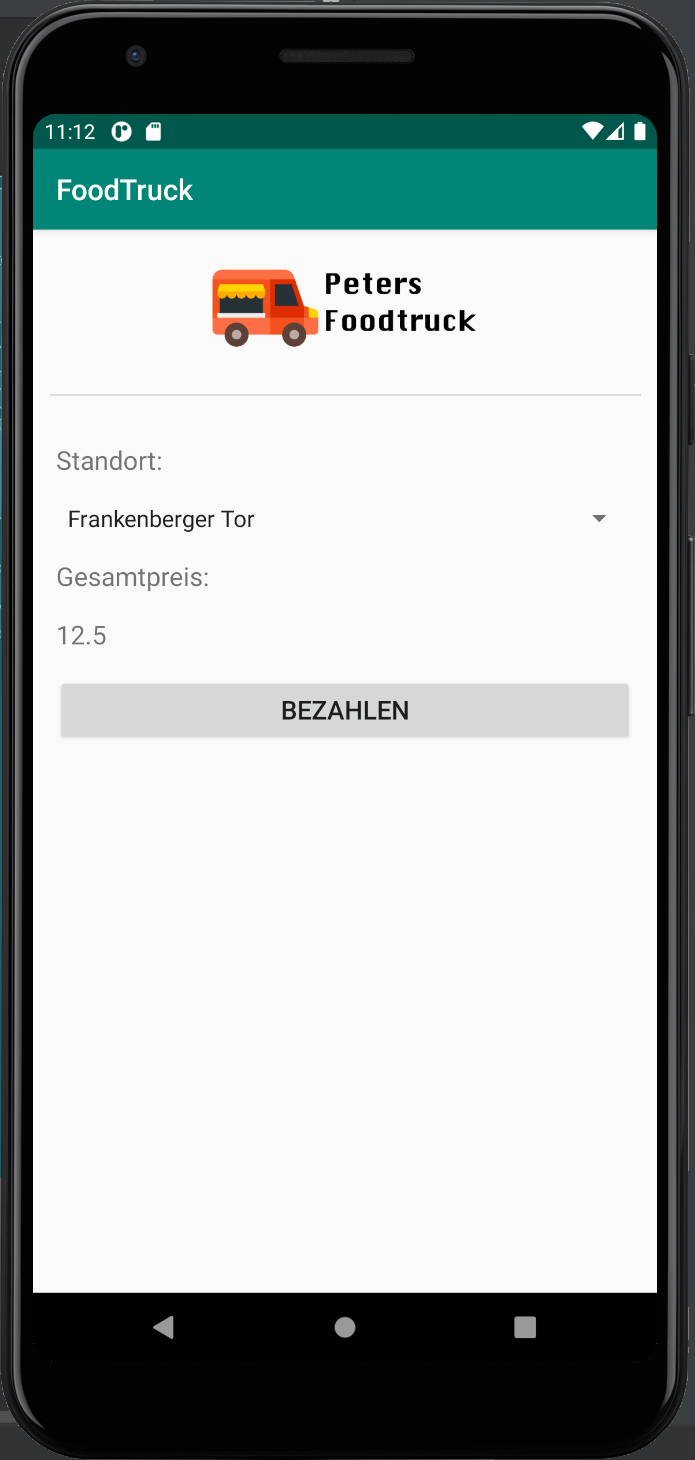
**CustomerNewOrderActivityOne**

= Haupt-Activity zur Bestellungsaufgabe eines Kunden (Auswahl der Speisen). Hier werden die Tages- und Vorbestellungsspeisekarte (Speisekarte für den folgenden Tag) in zwei ListViews angezeigt, welche die Namen und Preise der Speisen enthalten. Die ListViews befinden sich jeweils in einem eigenen Tab eines TabLayouts. Durch klicken eines Tabs „Heute“ oder „Morgen“ gerät die jeweilige Speisekarte in den Vordergrund. Durch Klicken eines Elements der Liste wird die gewünschte Speise zur Bestellung hinzugefügt. Möchte man eine Speise mehrfach bestellen, so muss das Element in der gewünschten Anzahl angeklickt werden. Die ausgewählten Speisen werden in einer Liste zwischengespeichert und erscheinen rechts von der Speisekarte in einer separaten ListView, welche die Namen und einen „X“-Button zum wieder-abwählen der Speisen enthält. Mittels eines Buttons „weiter“ erreicht man eine weitere Activity „CustomerNewOrderActivityTwo“, welche den Bestellprozess fortführt. Gleichzeitig übergibt man die Liste der selektierten Speisen.

Abb. 3.2.04 CustomerNewOrderActivityOne

**CustomerNewOrderActivityTwo**

= Unter- bzw. Folge-Activity zur Bestellungsaufgabe eines Kunden (Standortwahl und Bezahlung). Hier findet sich eine TextView, welche den Gesamtpreis der Bestellung anzeigt sowie ein Dropdown-Menü zur Auswahl eines Standorts, wo die Bestellung entgegengenommen werden soll. Mittels eines Buttons „bezahlen“ wird eine Funktion ausgelöst, die die Bestellung anlegt. Hierbei wird zwischen einer Reservierung und einer Vorbestellung unterschieden, je nachdem ob von der heutigen oder morgigen Speisekarte bestellt wurde. Anschließend wird der Kunde zur Activity „CustomerThankYouActivity“ geleitet.

Abb. 3.2.05 CustomerNewOrderActivityTwo

**CustomerThankYouActivity**

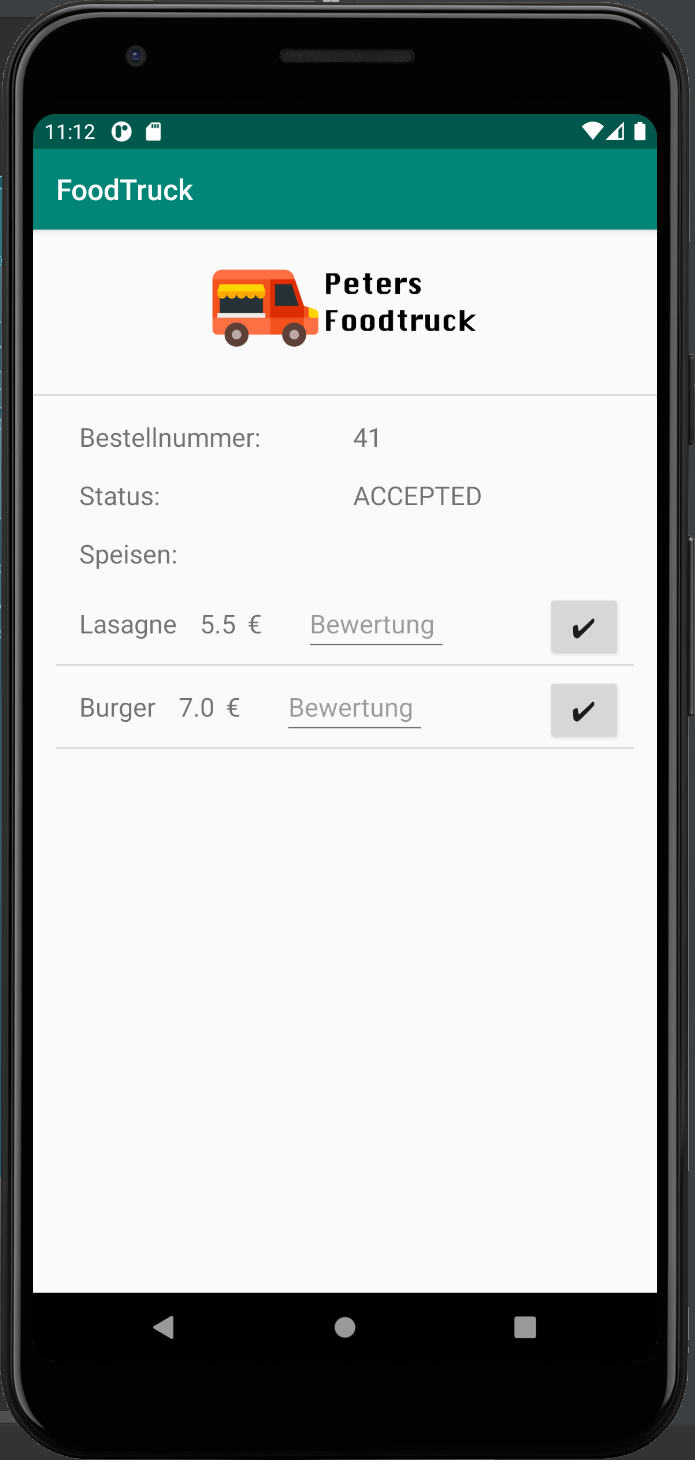
= Unter- bzw. Folge-Activity als End-Screen eines Bestellprozesses. Hier wird eine TextView angezeigt, welche dem Kunden für die Bestellung dankt und ihm diese bestätigt. Mittels eines Buttons „Bestellungen ansehen“ kann die Activity „CustomerShowOrderActivity“ aufgerufen werden. Der Bestellprozess ist hiermit abgeschlossen.

**CustomerShowOrderActivity**

= Haupt-Activity zur Ansicht der aufgegebenen Bestellungen. Hier werden die Bestellungen eines Kunden in einer ListView angezeigt, welche die Bestellnummern und Status der Bestellungen enthält. Durch Klicken eines Elements der Liste erreicht man eine Activity „CustomerShowOrderDetailsActivity“, gleichzeitig übergibt man das Element der Liste.

**CustomerShowOrderDetailsActivity**

= Unter-Activity zur Ansicht einer bestimmten aufgegebenen Bestellung. Hier wird das übergebene Element in die TextViews „Bestellnummer“ und „Status“ gefüllt. Ein ListView zeigt die zugehörigen Speisen mit „Namen“ und „Preis“. Hier kann ebenfalls mittels Eingabefeld und Button eine Bewertung mit 0 bis 5 Sternen vergeben werden.

Abb. 3.2.06 CustomerShowOrderDetailsActivity

**CustomerTruckInfoActivity**

= Haupt-Activity zur Ansicht der Truck-Route und -Speisekarte. Hier wird der aktuelle Standort des Food-Trucks in einer TextView angezeigt. Mittels eines Buttons „Route einsehen“ erreicht man eine Activity „CustomerShowRouteActivity“. Mittels eines weiteren Buttons „Speisekarte einsehen“ erreicht man eine weitere Activity „CustomerShowMenuActivity“.

**CustomerShowRouteActivity**

= Unter-Activity zur Ansicht der aktuellen Route des Food-Trucks. Hier wird die aktuelle Route in einer ListView angezeigt, welche die Namen sowie die Ankunfts- und Abfahrtszeiten der Standorte enthält.

**CustomerShowMenuActivity**

= Unter-Activity zur Ansicht der Tages- und Vorbestellungsspeisekarte (Speisekarte für den folgenden Tag). Hier werden die zuvor genannten Speisekarten in zwei ListViews angezeigt, welche die Namen, Preise und Bewertungen der Speisen enthalten. Die ListViews befinden sich jeweils in einem eigenen Tab eines TabLayouts. Durch klicken eines Tabs „Heute“ oder „Vorbestellung“ gerät die jeweilige Speisekarte in den Vordergrund.

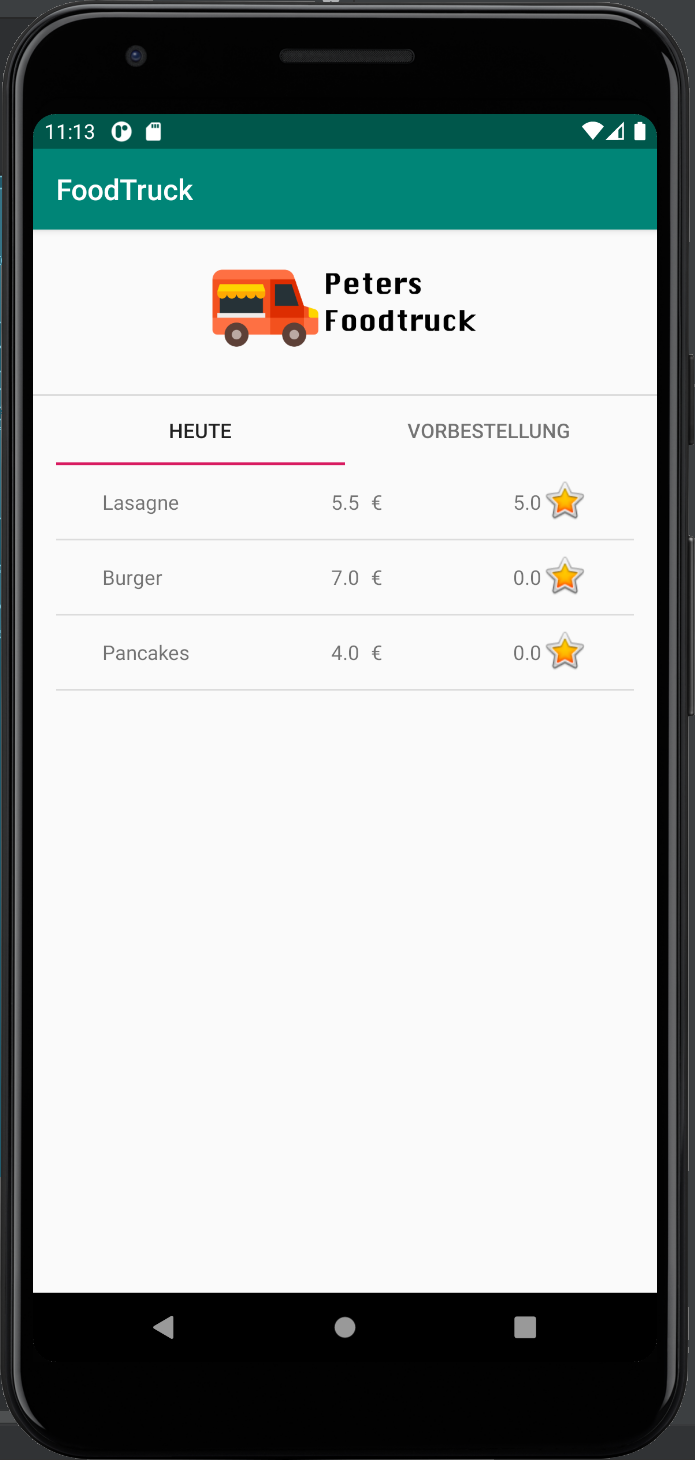


Abb. 3.2.07 CustomerShowMenuActivity

**CustomerLocationActivity**

= Haupt-Activity zum Erhalten eines Standortvorschlags. Hier finden sich Eingabefelder für die „x“- und „y“-Koordinaten des eigenen Standorts. Mittels eines Buttons „Food Truck Standorte vorschlagen“ wird eine Funktion ausgelöst, die den nächstgelegenen Standort auf der Route des Food-Trucks vorschlägt. Mittels eines weiteren Buttons „ohne Standortwahl fortfahren“ wird die Activity ohne Standortvorschlag geschlossen.

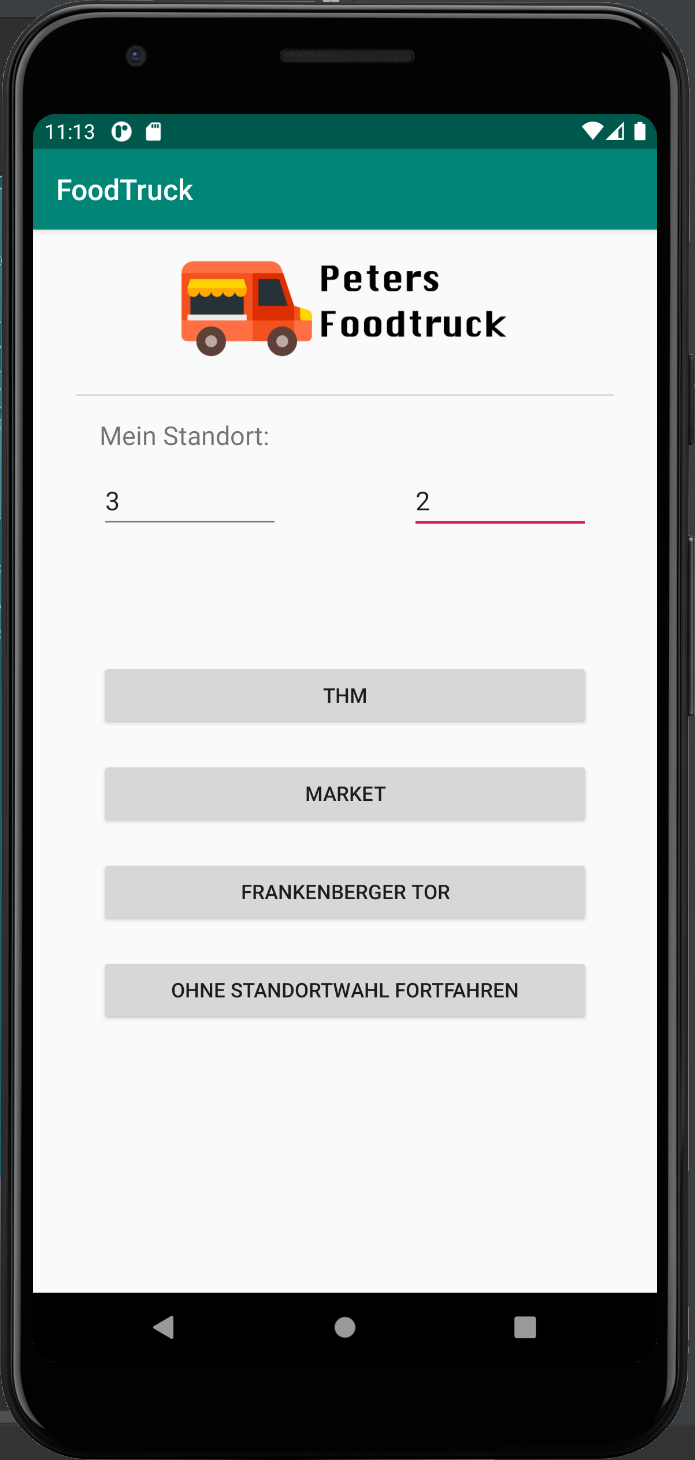


Abb. 3.2.08 CustomerLocationActivity

**OwnerMenuActivity**

Ausgehend von der OwnerMenuActivity (Hauptmenü eines Betreibers) erreicht man Activities für die Routenplanung, das Anpassen der Speisekarte, das Einsehen der Bestellungen, die Lebensmittelbestellung, als auch für Funktionen die den aktuellen Standort des Betreibers betreffen.

**OwnerRoutebearbeitenActivity**

= Haupt-Activity zur Routenplanung. Hier wird die aktuelle Route in einer ListView angezeigt, welche die Namen sowie die Ankunfts- und Abfahrtszeiten der Standorte enthält. Mittels eines Buttons „Standort hinzufügen“ erreicht man eine weitere Activity „OwnerAddstandortActivity“. Durch Klicken eines Elements der Liste erreicht man eine Activity „OwnerStandortbearbeitenActivity“, gleichzeitig übergibt man das Element der Liste.



Abb 3.2.09 OwnerRoutebearbeitenActivity

**OwnerAddstandortActivity**

= Unter-Activity zum Hinzufügen eines Standorts. Hier finden sich Eingabefelder für den Namen, den Breitengrad(X), den Längengrad(Y), die Ankunftszeit und die Zeit in Minuten, welche man an dem Standort verbringen möchte. Mittels eines Buttons „Standort hinzufügen“ wird eine Funktion ausgelöst, die den Standort anlegt.

**OwnerStandortbearbeitenActivity**

= Unter-Activity zum Bearbeiten oder Löschen eines ausgewählten Standorts. Hier wird das übergebene Element in die Eingabefelder analog der „OwnerAddstandortActivity“ gefüllt und kann bearbeitet werden. Mittels eines Buttons „Standort speichern“ wird eine Funktion ausgelöst, die die Änderungen speichert. Mittels eines weiteren Buttons „Standort entfernen“ wird eine Funktion ausgelöst, die den Standort löscht.

**OwnerSpeisekarteActivity**

= Haupt-Activity zur Ansicht und Bearbeitung der Speisekarte. Hier wird die aktuelle Speisekarte in einer ListView angezeigt, welche die Namen und Preise der Speisen enthält. Mittels eines Buttons „Speise hinzufügen“ erreicht man eine weitere Activity „OwnerSpeiseNeuActivity“. Durch Klicken eines Elements der Liste erreicht man eine Activity „OwnerSpeiseBearbeitenActivity“, gleichzeitig übergibt man das Element der Liste.

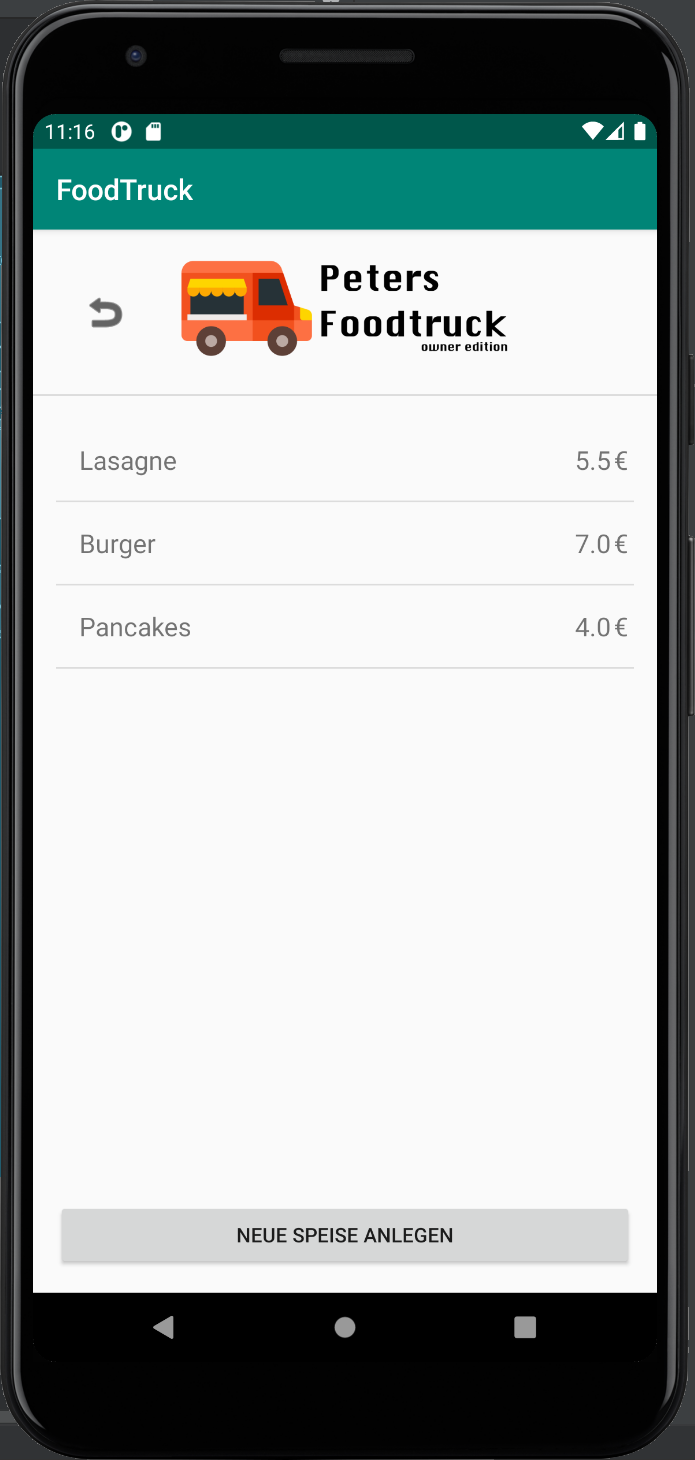
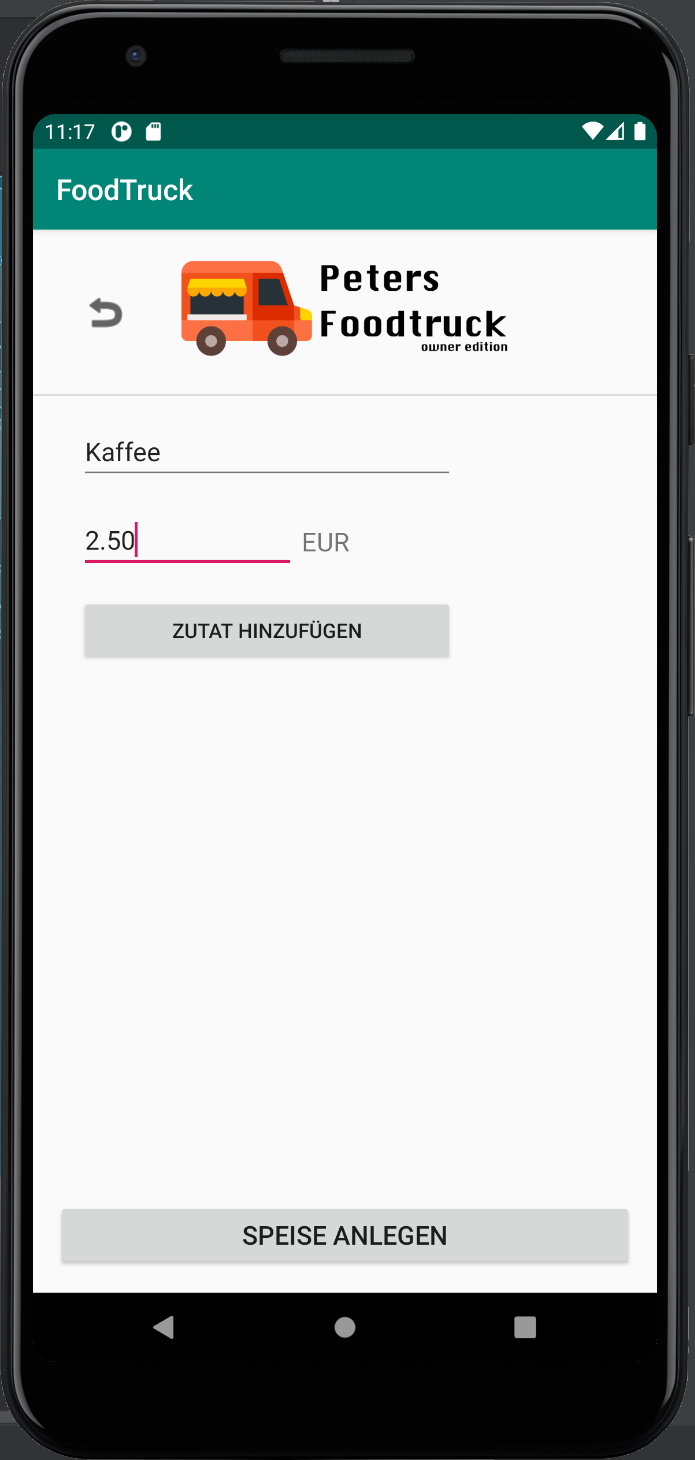


Abb. 3.2.10 OwnerSpeisekarteActivity

**OwnerSpeiseNeuActivity**

= Unter-Activity zum Hinzufügen einer Speise. Hier finden sich Eingabefelder für den Namenund den Preis einer Speise. Mittels eines Buttons „Zutat hinzufügen“ erreicht man eine separate Activity zum Erfassen einer Zutat und der geforderten Menge. In einer ListView werden die angelegten Zutaten angezeigt. Mittels eines weiteren Buttons „Speise anlegen“ wird eine Funktion ausgelöst, die die Speise anlegt.



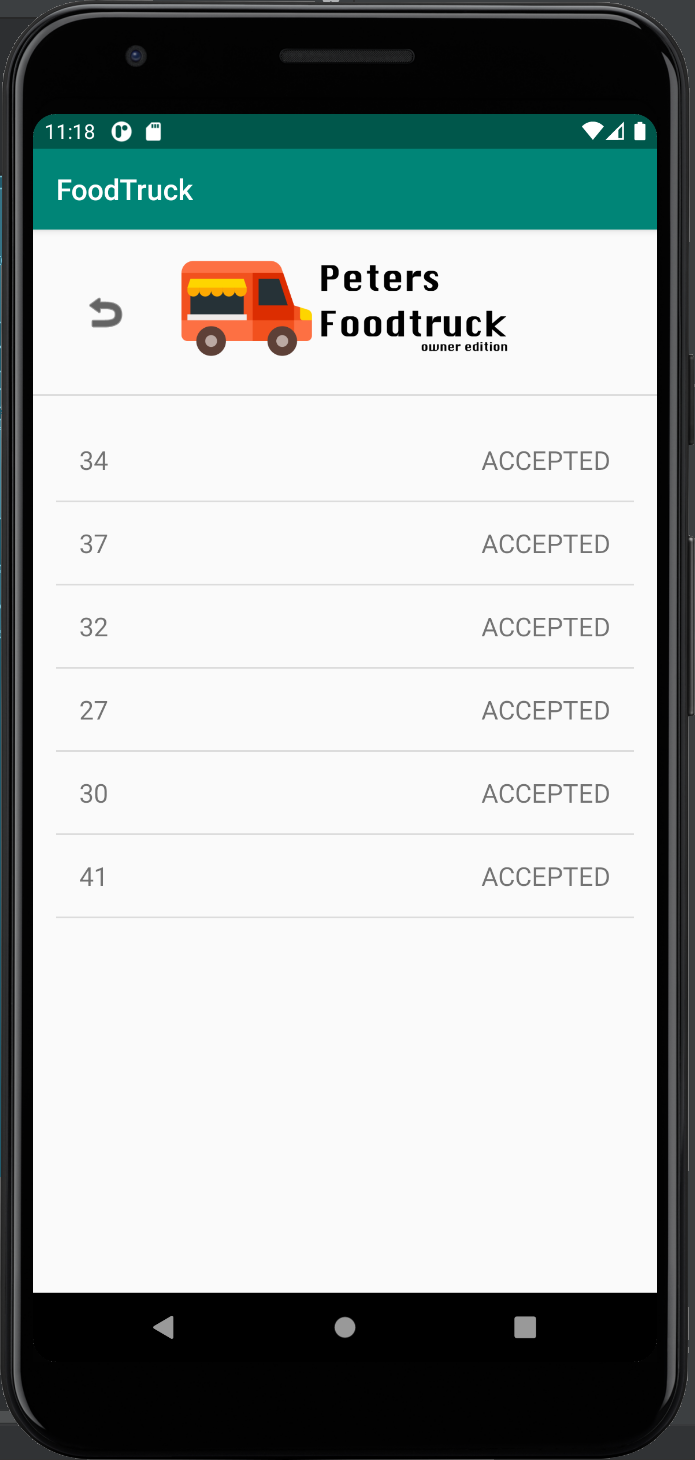
3.2.11 OwnerSpeiseNeuActivity

**OwnerSpeiseBearbeitenActivity**

= Unter-Activity zum Bearbeiten einer ausgewählten Speise. Hier wird das übergebene Element in die Eingabefelder analog der „OwnerSpeiseNeuActivity“ gefüllt und können bearbeitet werden. Dies gilt auch für die ListView der zum Gericht angelegten Zutaten. Mittels eines Buttons „Bearbeitung abschliessen“ wird eine Funktion ausgelöst, die die Änderungen speichert.

**OwnerBestellungenActivity**

= Haupt-Activity zur Ansicht der Bestellungen. Hier werden von Kunden aufgegebene Bestellungen in einer ListView angezeigt, welche die IDs und Status der Bestellungen enthält. Durch Klicken eines Elements der Liste erreicht man eine Activity „OwnerBestellungActivity“, gleichzeitig übergibt man das Element der Liste.



3.2.12 OwnerBestellungenActivity

**OwnerBestellungActivity**

= Unter-Activity zur Detailansicht einer Bestellung. Hier wird das übergebene Element in die TextViews für Uhrzeit, ID, Standort, Person und Status gefüllt. Diese können nicht bearbeitetwerden. Ebenfalls wird eine ListView geladen, welche die bestellten Speisen inkl. Preis anzeigt. Mittels eines Buttons „Zubereitung beginnen“ wird eine Funktion ausgelost, die den Status der Bestellung auf „in arbeit“ setzt. Mittels eines weiteren Buttons „Zubereitung abschliessen“ wird eine Funktion ausgelost, die den Status der Bestellung auf „erledigt“ setzt.

**OwnerLebensmittelbestellungActivity**

= Haupt-Activity zur Lebensmittelbestellung. Hier wird die aktuelle Anzahl der Bestellungen pro Gericht in einer ListView angezeigt, welche die Namen und Bestellzahlen der Gerichte enthält. Zusätzlich zu der Anzahl der Bestellungen lässt sich ein Zuschlag pro Gericht eingeben. Mittels eines Buttons „Lebensmittel bestellen“ wird eine Funktion ausgelöst, die ermittelt, welche Zutaten pro Gericht gefordert sind und multipliziert diese mit der geforderten Menge an Gerichten.

Mittels eines weiteren Buttons „zeige alle Zutaten“ wird diese Anzahl der Zutaten in einer separaten Activity angezeigt, ohne die Lebensmittelbestellung auszulösen.

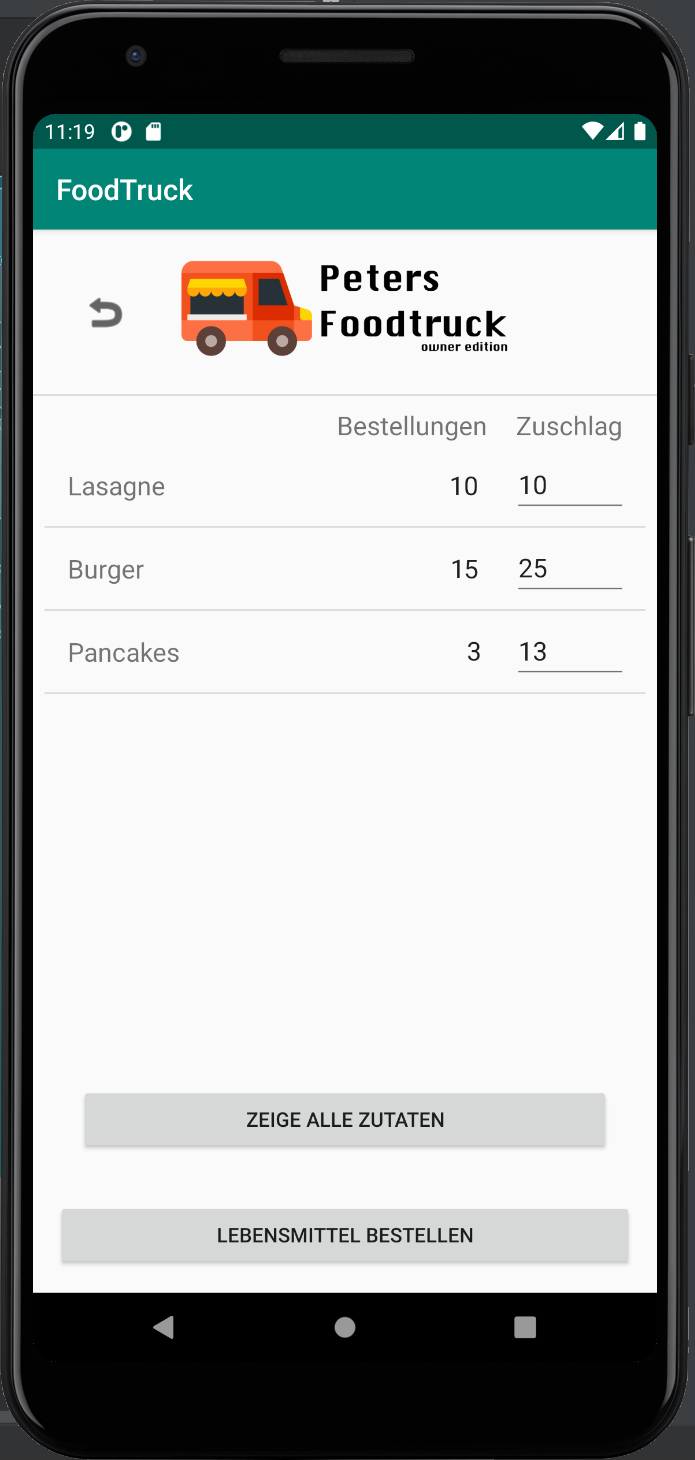


Abb. 3.2.13 OwnerLebensmittelbestellungActivity

**OwnerAktuellerstandortActivity**

= Haupt-Activity für Funktionen, die den aktuellen Standort betreffen. Hier wird mittels Eingabefeldern und Buttons das Auslösen der „Reservierungsannahme“ und des „Reservierungsschluss“ in „X“ Minuten ermöglicht.

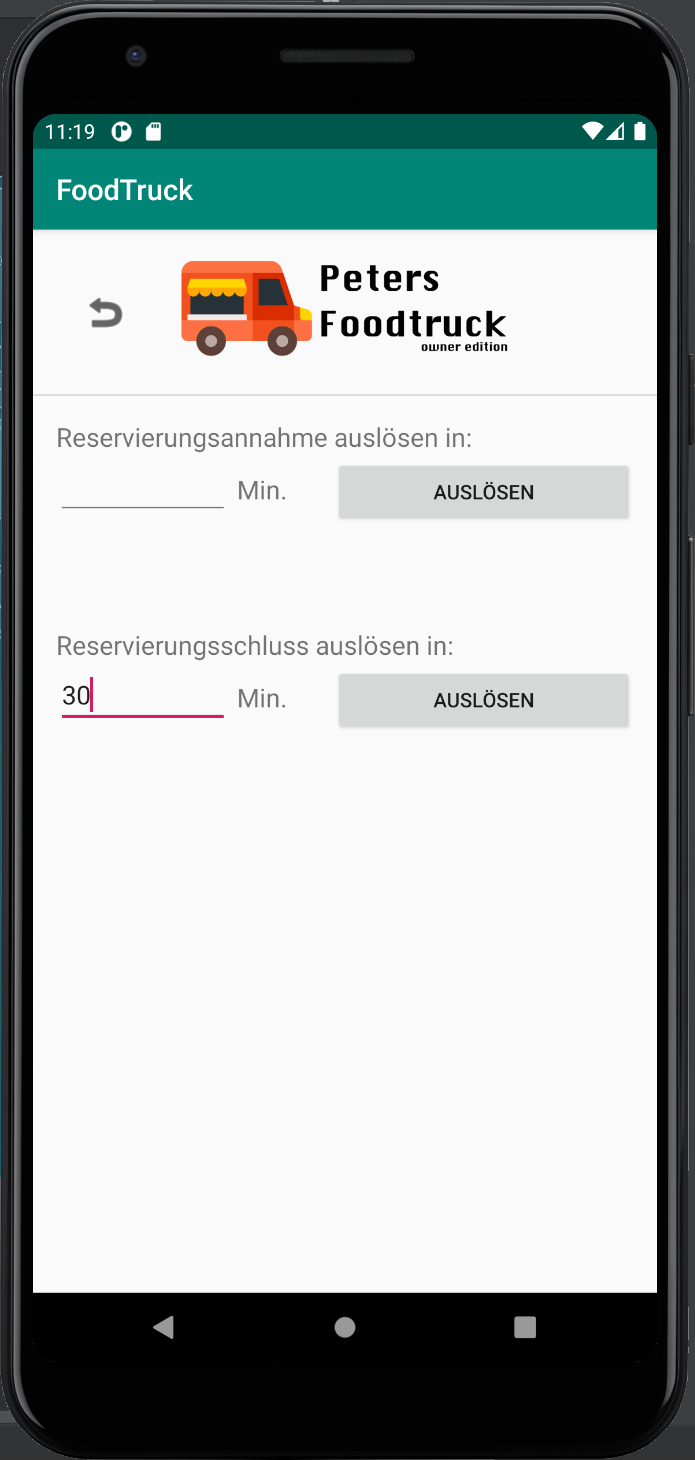
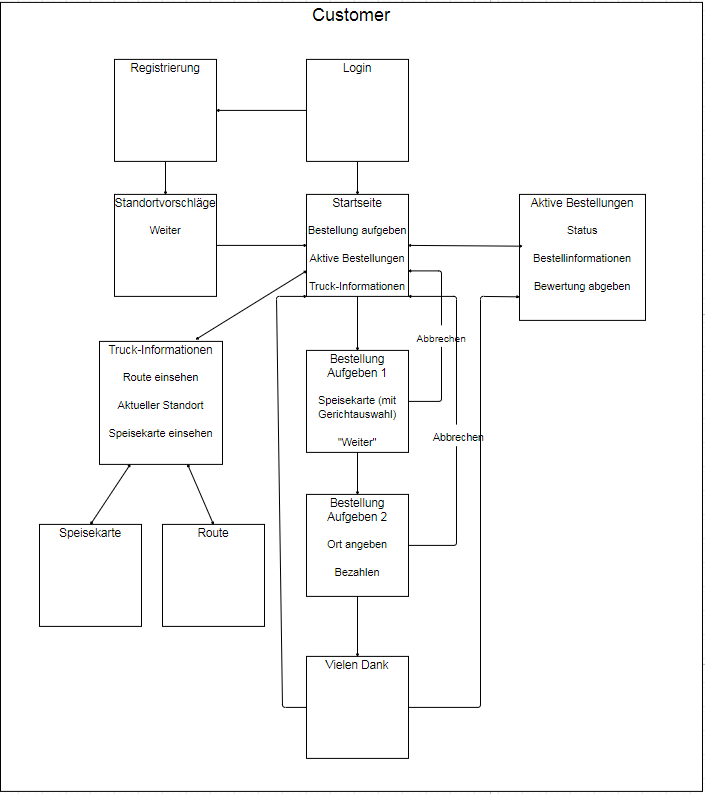


Abb. 3.2.14 OwnerAktuellerstandortActivity

# 4 Anhang

Sitemap der Kundensicht  


Sitemap der Betreibersicht

## 