BeePay Bezahlsystem

Bachelorarbeit

Abteilung Informatik
Hochschule für Technik Rapperswil

Herbstsemester 2015

Autoren:	Benjamin Kühnis, Martin Schaub
Betreuer:	Prof. Hansjörg Huser
Projektpartner:	Abrantix AG, Zürich
Experte:	Stefan Zettel
Gegenleser:	Farhad Mehta

Unser Dank gilt in erster Linie Prof. Hansjörg Huser für die Unterstützung während der Arbeit. Besonders bei der Beschaffung der Infrastruktur und bei Fragen zur Methodik und Strukturierung bekamen wir nützliche Hinweise.

Auch danken wir der Abrantix AG für die Unterstützung der Arbeit. Wir durften vom Fachwissen verschiedener Mitarbeiter profitieren, insbesondere von Herrn Daniel Eckstein und Herrn Christian Vetsch, welche uns mit ihrem langjährigen Wissen und ihren Kontakten zur Zahlungsbranche unterstützten.

Zum Schluss bedanken wir uns bei unseren Familien und Freunden für die mentale Unterstützung, sowie die Geduld während der Arbeit.

Abstract

BeePay ist ein Bezahlsystem, welches dem Kunden einen Mehrwert in Form von Hands-Free-Payment bietet. Nachdem der Kunde die Dienstleistung eines Händlers in Anspruch genommen hat (z.B. Essen im Restaurant), kann der Kunde den Standort einfach verlassen, ohne die Rechnung verlangen zu müssen. Der Kunde profitiert von einer Zeitersparnis, da er nicht auf die Rechnung warten muss. Diese erhält der Kunde nachträglich vom Händler via BeePay und wird mit einem zuvor hinterlegten Zahlungsmittel beglichen. Das Ziel der Arbeit ist es, die Identifikation des Kunden an einem Standort des Händlers anhand eines Prototyps für iOS und Android umzusetzen. Dabei spielt sowohl der Datenschutz als auch die Sicherheit eine wichtige Rolle.

Die Identifikation des Kunden findet in zwei Schritten statt. Zuerst muss der Kunde vor Ort erkannt werden. Dies wird Check-in genannt und soll möglichst automatisch mittels Technologien wie GPS oder Beacons geschehen. Damit der Kunde automatisch eingecheckt werden kann, muss er aus Datenschutzgründen zuvor mit dem Händler einen Vertrag abschliessen. Alternativ besteht für den Kunden die Möglichkeit des manuellen Check-ins. Nach dem Check-in ist ein Pairing erforderlich. Dabei werden allfällige Zusatzinfos wie die Tischnummer erfasst. Diese Zusatzinfo wird benötigt, um später die Rechnung zuweisen zu können. Ausserdem wird in diesem Schritt die Sicherheit des Pairings gewährleistet, indem je nach Sicherheitseinstellung des Kunden oder Händlers eine manuelle Bestätigung der Identifikation notwendig ist. Erst danach ist die Identifikation des Kunden abgeschlossen.

Als Ergebnis entstand ein funktionierender Prototyp für iOS und Android, welcher den gesamten Ablauf von der Identifikation bis zur Rechnung demonstriert. Dabei findet die Standorterkennung mittels GPS oder Beacons statt. Auf der Serverseite wurde ASP.NET Web API eingesetzt. Mittels Anbindung an einen Payment Service Provider wird eine Testzahlung ausgelöst.

Management Summary

Ausgangslage und Problembeschreibung

Im Rahmen der vorangegangenen Studienarbeit wurden bestehende Bezahlsysteme und Lösungen analysiert. Daraus entstand der Prototyp einer Plattform namens BeePay, welche dem Kunden ein neues Bezahlerlebnis in Form von Hands-Free-Payment bietet. Aufgrund der Erkenntnisse aus der Studienarbeit wird nun die Kernfrage der Identifikation genauer betrachtet.

Die Identifikation besteht aus zwei Schritten. Zuerst muss der Kunde am Standort erkannt werden, was Check-in genannt wird. Dies soll möglichst automatisch mithilfe geeigneter Technologien geschehen. Anschliessend muss ein Pairing zwischen Kunde und Händler stattfinden. Dabei werden beim Pairing je nach Anwendungsfall zusätzliche Informationen benötigt. Bei der Identifikation müssen sowohl Sicherheit als auch Datenschutzaspekte beachtet werden.

Vorgehen

Nach einer Analyse des Identifikationsprozesses und verschiedener technischer Möglichkeiten wurde ein Prototyp für Android und iOS umgesetzt. Zusätzlich wurde die BeePay Plattform entsprechend dem Lösungskonzept angepasst und erweitert.

Ergebnis

Die Kernfrage der Identifikation wurde mit einem mehrstufigen Prozess gelöst und in einem Prototyp für iOS und Android demonstriert. Die Mobilapplikationen bieten einen Überblick über aktuelle Identifikationen und deren Status. Je nach Status können verschiedene Aktionen ausgeführt werden. Ausserdem können Rechnungen angezeigt und bezahlt werden.

Das Check-in deckt dabei Datenschutzaspekte ab, indem ein automatischer Check-in nur bei Händlern geschieht, welche vom Kunden zuvor ausgewählt wurden. Durch einen manuellen Check-in hat der Kunde immer die volle Kontrolle.

Die Sicherheit für den Kunden und den Händler wird durch das Pairing gewährleistet. Kunde und Händler können jeweils festlegen, ob sie eine Identifikation manuell bestätigen wollen. Somit kann eine Identifikation nur durch eine zusätzliche Bestätigung abgeschlossen werden, wodurch die Sicherheit gewährleistet wird. Ebenfalls beim Pairing findet optional die Eingabe zusätzlicher Informationen statt. In einem Restaurant ist diese Zusatzinformation die Tischnummer, um eine Verknüpfung zum Kassensystem herzustellen.

Erst wenn alle erforderlichen Bestätigungen und Zusatzinfos vorhanden sind, ist die Identifikation abgeschlossen. Der Händler hat nun die Möglichkeit, dem Kunden eine Rechnung zu senden. Der Kunde erhält die Rechnung und kann diese mit dem zuvor hinterlegten Zahlungsmittel bezahlen.

Ausblick

Mit dem entwickelten Prototyp kann der gesamte Prozess von der Identifikation bis zur Rechnung demonstriert werden. Ein nächster Schritt ist der Einsatz des Prototyps in einer echten Umgebung. Dafür muss der Prototyp in die bestehenden Systeme und Prozesse des Händlers integriert werden, um einen effizienten Einsatz zu ermöglichen. Die Integration in das Kassensystem des Händlers ist dabei eine zentrale Aufgabe, damit eine automatisierte Rechnungsstellung möglich wird.

Inhaltsverzeichnis

1 Einleitung

Im Rahmen der vorangegangenen Studienarbeit [1] wurden bestehende Bezahlsysteme und Lösungen analysiert. Aus den Erkenntnissen entstand der Prototyp einer Plattform namens **BeePay**, welche dem Kunden ein neues Bezahlerlebnis bietet. Der Kunde profitiert von einem einfachen Bezahlprozess mittels **Hands-Free-Payment** sowie von Mehrwertleistungen. Beim Hands-Free-Payment wird der Kunde bei einem Händler automatisch erkannt und kann mit einem zuvor hinterlegten Zahlungsmittel bezahlen, ohne das Portemonnaie oder Smartphone hervorzunehmen. Mehrwertleistungen können in Form von Rabatten, standortspezifischen Angeboten, Vorbestellungen oder Schnellcheckouts angeboten werden [1, pp. 9-10]. Zentral beim Hands-Free-Payment und den Mehrwertleistungen ist die Frage, wie der Kunde vor Ort vom Händler identifiziert wird.

Abbildung 1 BeePay Logo in gelb (Studienarbeit) und blau (Bachelorarbeit)

1.1 Ausgangslage

Aufgrund der getätigten Marktanalyse von bestehenden Bezahllösungen wurde klar, dass sich die aktuellen Bezahlsysteme vor allem durch den Mehrwert in Form von Zusatzleistungen auszeichnen. Geht es jedoch um den Bezahlprozess selbst, mangelt es oft an der Benutzerfreundlichkeit und der Integration sowie der Nutzung der bestehen Infrastrukturen. [1, pp. 5-6]

Aus diesen Erkenntnissen entstand ein Lösungskonzept, welches den Prozess des Hands-Free-Payment umsetzt. Dabei setzt die Lösung auf bestehende Zahlungsmittel, welche vom Kunden zuvor hinterlegt werden. Der Bezahlprozess erfolgt in drei Phasen:

- Kundenidentifikation
- Rechnungsstellung
- Zahlungsausführung

Abbildung 2 Hands-Free-Payment Prozess [1, p. 7]

Für den Prototyp der BeePay Plattform wurden die einzelnen Phasen stark vereinfacht, um die Machbarkeit des Konzepts aufzuzeigen. Dabei wurden die einzelnen Schritte jeweils manuell via BeePay Webseite durchgeführt. Die Webseite greift auf das entwickelte BeePay API zu, welches einen flexiblen Zugriff durch weitere Clients (z.B. Smartphones) zulässt.

Die Identifikation des Kunden spielt beim Prozess die Kernrolle. Damit ein Kunde beim Händler identifiziert werden kann, muss der Kunde vorgängig eine Verknüpfung zum Händler erstellen. Dies ist ein Vertrag, der Sicherheitseinstellungen und Zahlungsmittel zwischen Kunde und Händler festlegt. Nachdem der Händler einen Kunden identifiziert hat, kann der Händler dem Kunden eine Rechnung senden. Die Rechnung wird anschliessend mit dem zuvor hinterlegten Zahlungsmittel beglichen. [1, pp. 10-11]

1.2 Ziele der Arbeit

Aufgrund der Erkenntnisse aus der Studienarbeit soll nun die Kernfrage der Identifikation genauer betrachtet werden. Dabei zeichnet sich ab, dass die Identifikation ein komplexer und mehrstufiger Prozess ist. Nebst der initialen Erkennung des Kunden muss auch ein Pairing zwischen Kunde und Händler stattfinden. Dabei gibt es hohe Anforderungen an Sicherheit und Datenschutz und zudem soll es möglich sein, die Identifikation sowohl in einem geschlossenen Raum als auch im Freien durchzuführen.

Nach einer Analyse des Identifikationsprozesses und verschiedener technischer Möglichkeiten soll ein Prototyp für Android und iOS umgesetzt werden, welcher den gesamten Zahlungsprozess umsetzt. Als Grundlage dient die bereits entwickelte Plattform aus der Studienarbeit, welche ein API zur Anbindung der Applikation anbietet. Die Umsetzung und Machbarkeit für Windows Phone wird nicht genauer betrachtet, da die Verbreitung zu gering ist [2].

2 Problembeschreibung

Während der Bachelorarbeit wird als zentrale Fragestellung und Aufgabe die Identifikation zwischen Kunde und Händler behandelt. Zusätzlich wird das Zusammenspiel zwischen Kunde, Händler und BeePay verfeinert. Dabei entstehen bei der Verwendung von BeePay verschiedene Herausforderungen.

Betätigte Transaktionen über die BeePay Plattform [1] werden als Distanzzahlung über das Vier-Parteien-System[2] abgewickelt. Indem der Kunde auf der BeePay Plattform eine Verknüpfung zu einem Händler erstellt, entsteht ein Vertrag, welcher es dem Händler ermöglicht, den Kunden an seinen Standorten zu identifizieren. Die Identifikation findet in zwei Schritten statt. Zuerst wird erkannt, dass der Kunde sich an einem Standort befindet, was als Check-in bezeichnet wird. Danach findet ein Pairing zwischen Händler und Kunde statt. Bei erfolgreichem Pairing hat der Händler die Möglichkeit dem Kunden eine Rechnung zu senden.

Dabei entsteht das Problem, wie genau der Kunde identifiziert wird, wobei das Check-in wie auch das Pairing genauer betrachtet werden muss. Anschliessend stellt sich die Frage, wie der Zahlungsprozess über die BeePay Plattform ausgelöst und abgewickelt wird.

Um die daraus entstehenden Probleme genauer zu beschreiben, wurden zwei fiktive Use Cases erstellt. Einerseits der Use Case eines Restaurantbesuchs und andererseits der Use Case eines Club Eintritts. Bei beiden Use Cases wird davon ausgegangen, dass der Händler (das Restaurant oder der Club) BeePay im Einsatz hat, sowie der Kunde ein Mobiltelefon besitzt, auf welchem die BeePay App installiert ist. Diese ermöglicht es ihm, sich gegenüber dem Händler zu identifizieren. Die anschliessende Zahlung wird über die BeePay Plattform abgeschlossen.

2.1 Use Case Restaurantbesuch

Es wird davon ausgegangen, dass der Kunde vorgängig einen Vertrag mit dem Händler auf der BeePay Plattform erstellt hat.

Der Kunde betritt das Restaurant und wird vom Restaurantpersonal an einen Tisch gewiesen. Im Hintergrund führt die BeePay App den Check-in für den Kunden durch. Dies kann nur geschehen, da der Kunde vorgängig auf der BeePay Plattform einen Vertrag mit dem Händler abgeschlossen, sowie ein Zahlungsmittel hinterlegt hat. Anschliessend wird das Pairing zwischen Kunde und Händler durchgeführt. Dabei wird dem eingecheckten Kunden seine Tischnummer zugewiesen. Mit dem erfolgreichen Pairing hat der Händler nun die Möglichkeit dem Kunden eine Rechnung zu stellen.

Der Kunde bestellt und verlässt nach dem Essen das Restaurant. Der Händler verrechnet dem Kunden die konsumierten Leistungen, indem er auf der Kasse den Tisch schliesst und dem Kunden die Rechnung ausstellt. Der Kunde erhält die Informationen über die verrechneten Leistungen, wobei die Rechnung automatisch via BeePay bezahlt wird.

[1] Siehe Kapitel 7 *Appendix II – BeePay Plattform Systemübersicht, Umsetzung der Studienarbeit*
[2] Siehe Kapitel 6 *Appendix I – Zahlungssystem*

2.2 Use Case Club Eintritt

Es wird davon ausgegangen, dass der Kunde keinen Vertrag mit dem Händler hat.

Der Kunde steht vor einem Club und sieht, dass dieser eine BeePay Fastline hat. Diese ermöglicht Clubbesuchern, welche mit BeePay bezahlen, schneller in den Club zu gelangen.

Der Kunde entscheidet sich, dass er die BeePay Fastline benutzen möchte. Er startet die BeePay App, sucht nach Händlern in der Umgebung und wählt den Club sowie ein Zahlungsmittel aus. Dadurch schliesst der Kunde einen einmal gültigen Vertrag mit dem Händler ab.

Da der Kunde nun einen Vertrag mit dem Club hat, kann die Identifikation gestartet werden. Im Hintergrund wird automatisch das Check-in durchgeführt. Der Kunde begibt sich zur BeePay Fastline, wo der Türsteher die üblichen Überprüfungen vor einem Club durchführt. Entscheidet sich der Türsteher, dass der Kunde in den Club darf, wählt er ihn aus den Personen aus, welche das Check-in durchgeführt haben. Die Auswahl startet den Pairing Prozess. Je nach Einstellung des Kunden, muss dieser noch mit der BeePay App das Pairing bestätigen, damit dieses erfolgreich abgeschlossen werden kann. Bei erfolgreichem Pairing wird der Kunde eingelassen und der Zahlungsprozess automatisch über die BeePay Plattform ausgelöst, womit dem Kunden der Eintritt in den Club verrechnet wird.

2.3 Identifikation

Die Identifikation besteht aus zwei Schritten. Erstens muss der Kunde am Standort erkannt werden, was Check-in genannt wird. Zweitens muss ein Pairing zwischen Kunde und Händler stattfinden. Dabei sollen sowohl Sicherheit als auch Datenschutzaspekte beachtet werden.

Für ein Check-in des Kunden an einem Standort muss zuvor ein Vertrag auf der BeePay Plattform zwischen Kunde und Händler abgeschlossen werden. Mit dem Vertrag stimmt der Kunde zu, dass er bei der Händlerlokalität den Check-in Prozess durchführen möchte. Die Schwierigkeit dabei ist es, wie der Kunde an einem Standort zuverlässig erkannt wird. Dies soll im Hintergrund automatisch und ohne Kundeninteraktion geschehen. Beim Use Case Restaurantbesuch wird das Check-in durchgeführt, sobald der Kunde die Lokalität betritt. Dabei befindet sich der Kunde in einem geschlossenen Raum. Ganz im Gegenteil zum Use Case Club Eintritt, wo sich der Kunde während dem Check-in vor dem Club im Freien befindet. Es muss also möglich sein, das Check-in sowohl im Freien wie auch in geschlossenen Räumen durchzuführen.

Wie im Use Case Restaurantbesuch beschrieben, besteht die Möglichkeit, dass während des Pairing Prozesses zusätzliche Informationen benötigt werden. Dabei stellt sich die Frage, wo diese Informationen herkommen und ob sie vom Kunden oder vom Händler erfasst werden. Am angenehmsten wäre es, wenn auf Kundenseite keine Interaktion nötig ist.

2.4 BeePay Händlersuche

Im Use Case Club Eintritt besteht noch kein Vertrag zwischen dem Kunden und dem Händler. Es ist aus Datenschutzgründen nicht möglich, den Check-in Prozess durchzuführen, da der Kunde mit dem Händler noch keinen Vertrag abgeschlossen hat und somit seine Einwilligung, dass der Händler seine Kundendaten erhält, noch nicht gegeben hat. Indem der Kunde nach Händlern in der Umgebung sucht, kann er schnell einen einmal gültigen Vertrag zwischen sich und dem Händler abschliessen. Dabei stellt sich die Frage, wie mit der BeePay Plattform kommuniziert werden muss, oder ob dies auch offline oder durch ein Ad Hoc Netzwerk zwischen Händler und Kunde geschehen kann.

2.5 Datenschutz

Während des Check-ins gelangen Kundendaten an den Händler. Dabei muss gewährleistet sein, dass dies im Einverständnis der Kunden geschieht. Mittels eines Vertrags, welcher auf der BeePay Plattform abgeschlossen wird, kann dies gewährleistet werden. Dabei muss zwischen einem einmaligen und einem dauerhaften Vertrag zu den einzelnen Händlern unterschieden werden.

Bei der Suche nach Händlern gelangen Händlerdaten an den Kunden, welche öffentlich sind. Dabei werden nur Händler angezeigt, welche BeePay im Einsatz haben. Diese sind sich bewusst, dass Kunden ihre Daten erhalten können.

Bei der Zahlung dürfen Kreditkartendaten nicht auf der BeePay Plattform gespeichert werden. Dies wird durch den PCI Security Standard [3] untersagt. Damit beim Bezahlprozess nicht jedes Mal die Zahlungsdaten eingegeben werden müssen, wird der Alias Mechanismus[3] von Datatrans verwendet.

2.6 Sicherheit

Bei der Sicherheit muss einerseits garantiert werden, dass die Kommunikation zwischen Händler, Kunde und der BeePay Plattform verschlüsselt ist, andererseits muss sichergestellt werden, dass die Identifizierung mit dem richtigen Händler abgeschlossen wird. Da es verschiedene Technologien zur Identifikation gibt, muss die Sicherheit für jede Technologie einzeln betrachtet werden.

Hinzu kommt, dass der Kunde jederzeit die Kontrolle haben soll, wann eine Zahlung ausgelöst wird. Der Kunde muss entscheiden können, ob eine Rechnung automatisch bezahlt wird, oder ob er die Rechnung manuell bestätigen will.

[3] Siehe Kapitel 7.1.1 *Anbindung an Datatrans*

3 Lösungskonzept

Der Prozess von der Registrierung bis zur Zahlungsauslösung wurde bereits in der Studienarbeit realisiert. Dabei wurde die Identifikation vereinfacht implementiert. Der Händler konnte über die Weboberfläche der BeePay Plattform irgendeinen Kunden jederzeit identifizieren und ihm anschliessend eine Rechnung stellen. Der Vorgang der Identifikation wird überarbeitet, sodass der Kunde automatisch oder mit wenig Interaktion identifiziert werden kann.

Anhand der zwei fiktiven Use Cases Restaurantbesuch und Club Eintritt wird das Lösungskonzept erarbeitet. Daraus soll der Prototyp mit der Identifikation des Kunden sowie die Anbindung an die bestehende BeePay Plattform aus der Studienarbeit [1] entstehen.

Beide Use Cases verwenden den gleichen Ablauf. Sowohl der Händler, wie auch der Kunde, haben sich auf der BeePay Plattform registriert. Der Kunde schliesst mit dem Händler einen Vertrag ab. Dies kann wie im Use Case Restaurantbesuch ein dauerhafter oder wie im Use Case Club Eintritt ein einmaliger Vertrag sein. Mit dem Vertrag kann die Identifikation des Kunden am Standort des Händlers durchgeführt werden.

Die Identifikation besteht aus dem Check-in und dem Pairing. Nach erfolgreichem Pairing kann der Händler den Zahlungsvorgang auf der BeePay Plattform auslösen. Der Zahlungsvorgang wird über den **Payment Service Provider** Datatrans abgewickelt. Es wäre auch möglich die Transaktion über einen anderen Weg abzuwickeln, solange der Händler und der Kunde die gleiche Zahlungsmethode verwenden. Ein Beispiel dazu könnte Bitcoin[4] oder PayPal[5] sein.

[4] https://bitcoin.org
[5] https://www.paypal.com

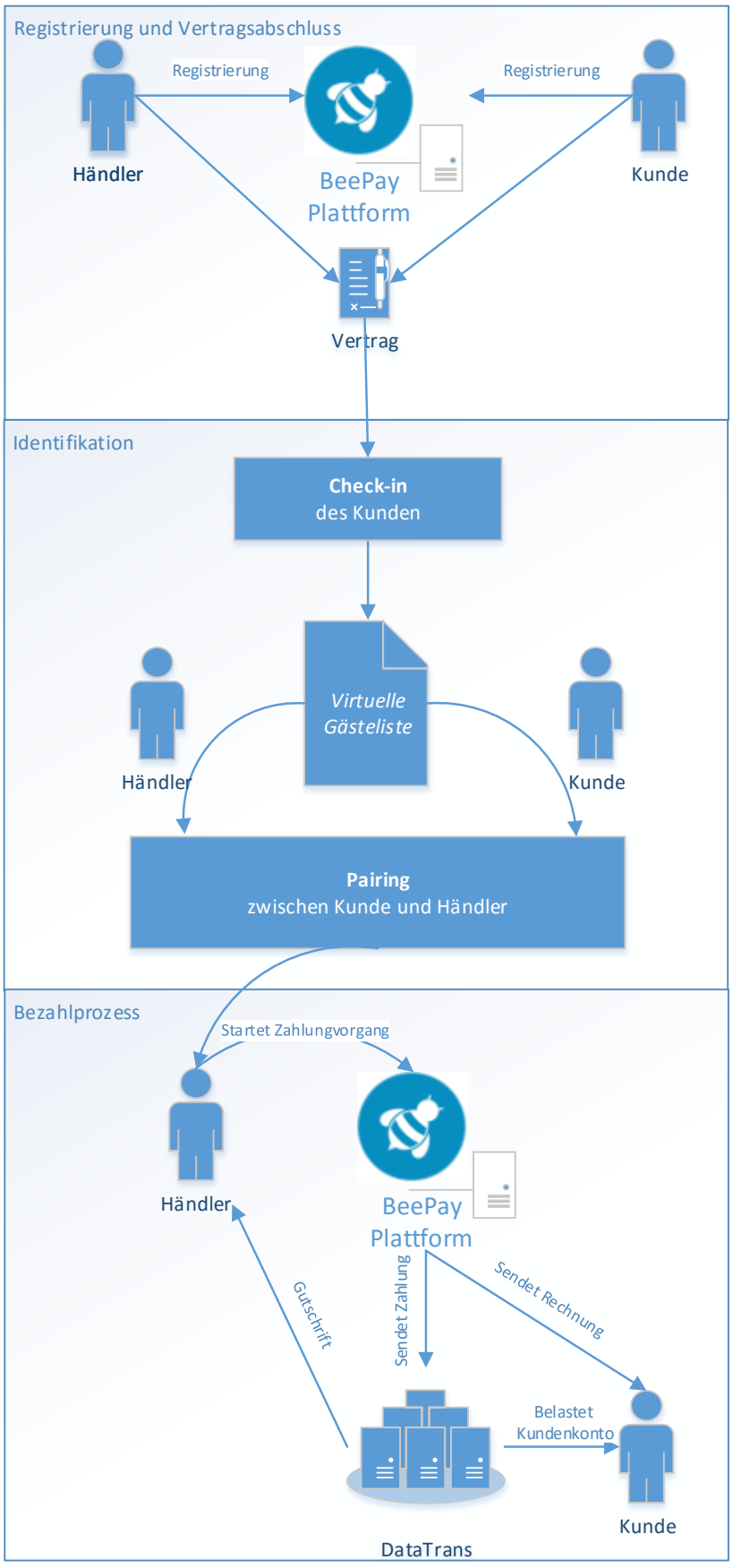

Abbildung 3 BeePay Systemübersicht

3.1 Identifikation

Die Identifikation besteht aus zwei Schritten. Erstens muss erkannt werden, dass sich der Kunde am Standort des Händlers befindet, was als Check-in bezeichnet wird. Zweitens muss das Pairing durchgeführt werden.

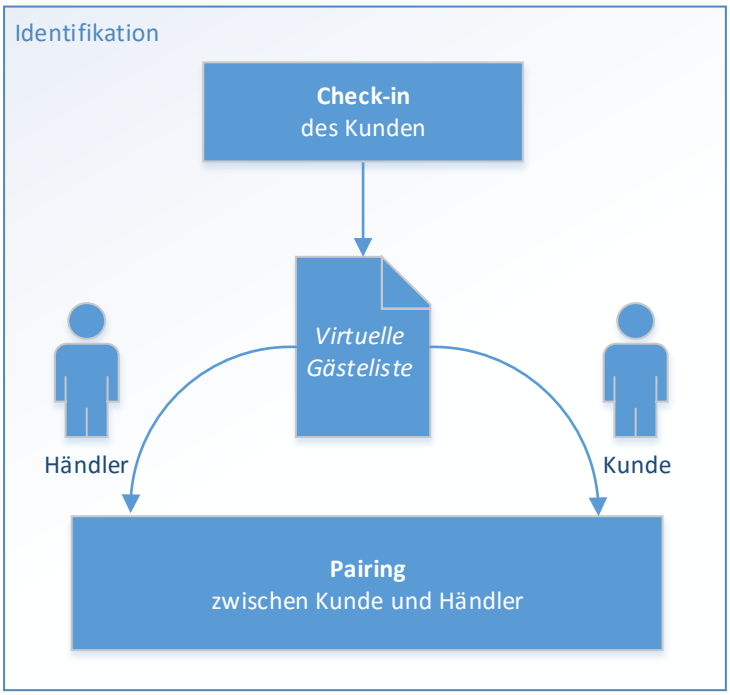

Abbildung 4 Identifikation im Überblick

3.1.1 Check-in

Der erste Schritt zur Identifizierung des Kunden ist der Check-in. Durch den Check-in wird der Kunde auf eine virtuelle Gästeliste gesetzt, welche vom Händler einsehbar ist. Dadurch wird dem Händler signalisiert, dass der Kunde vor Ort anwesend ist.

Der Check-in soll für den Kunden möglichst einfach und wenn möglich automatisch geschehen. Voraussetzung dafür ist, dass der Kunde zuvor einen Vertrag mit dem Händler auf der BeePay Plattform abgeschlossen hat. Dadurch wird sichergestellt, dass der Kunde nur bei denjenigen Händlern automatisch eingecheckt wird, denen er vertraut. Kunden gelangen also ohne Vertrag mit dem Händler nie auf die virtuelle Gästeliste.

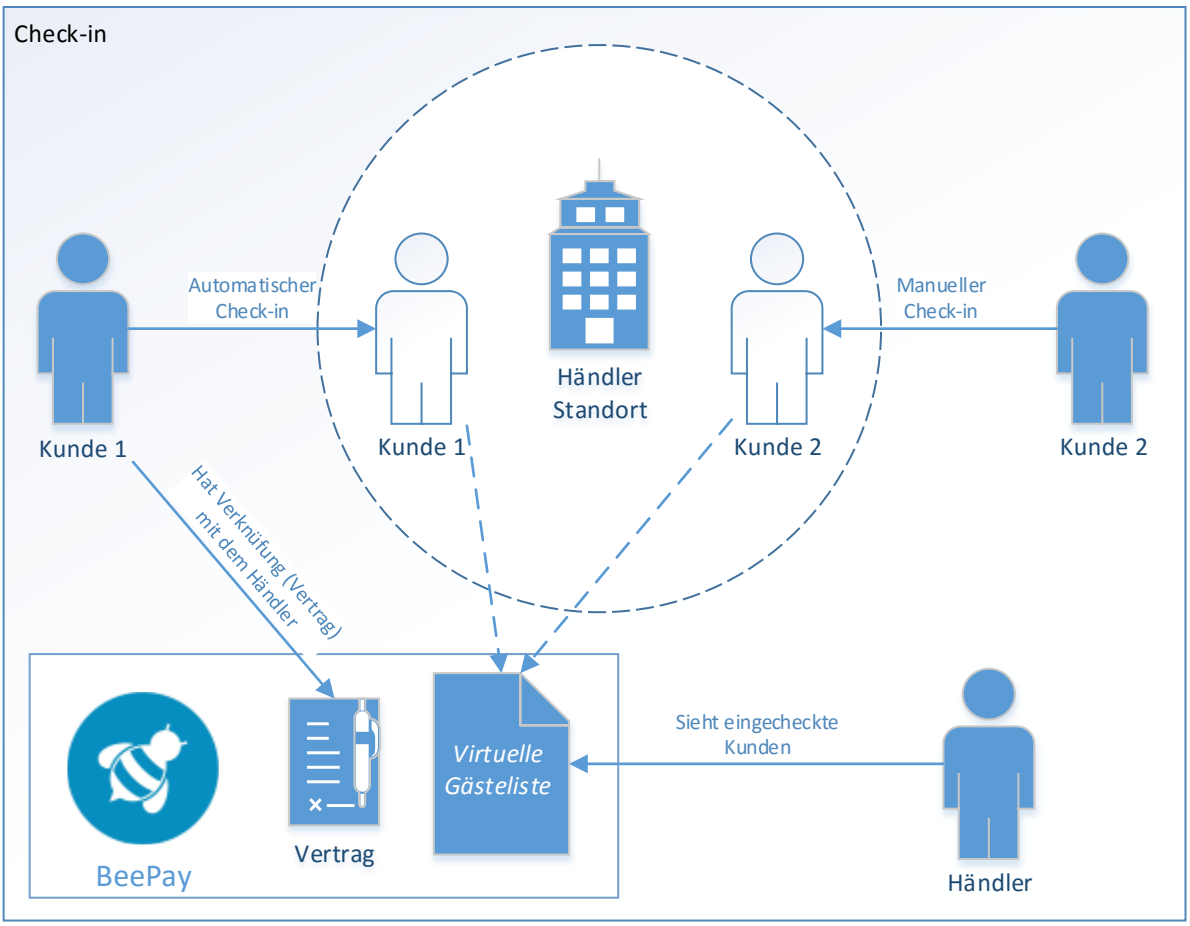

Abbildung 5 Check-in im Detail

Um den automatischen Check-in zu ermöglichen, stehen verschiedene Technologien zur Verfügung. Dabei beschränkt sich die Auflistung auf diejenigen Technologien, welche heutzutage vorwiegend in Smartphones verfügbar sind.

Während der Umsetzung werden die verschiedenen Technologien für Android und iOS analysiert. Es ist absehbar, dass bei beiden Plattformen eine Kombination von Technologien eingesetzt wird, um die Positionierung und somit das Check-in zu realisieren. Nachfolgend werden die verschiedenen Technologien kurz vorgestellt, sowie auf ihre Stärken und Schwächen hingewiesen.

3.1.1.1 WLAN

Mittels der umliegenden WLAN-Netzwerke in Reichweite kann festgestellt werden, wo sich der Kunde aktuell befindet. Je mehr Netzwerke in Reichweite sind, desto genauer wird die Positionsbestimmung. Anhand von Datenbanken mit den Standorten der WLAN-Netzwerke, kann der Standort bestimmt werden. Dabei gibt es kostenlose und kostenpflichtige Anbieter solcher Datenbanken.

Der Händler kann beim Einrichten von BeePay an seinem Standort sein eigenes WLAN angeben. Somit könnten Kunden zuverlässig erkannt werden [4] [5] ohne auf Datenbanken mit Listen von Netzwerken zugreifen zu müssen.

Problematisch ist, dass die 802.11 Beacon Frames zwar den Namen (SSID) des WLAN mitliefern [6], dieser sich jedoch ändern kann und es nicht garantiert ist, dass es sich um das richtige WLAN handelt, da der Name nicht eindeutig ist [7].

3.1.1.2 Beacons

Beacons ermöglichen eine genaue Positionierung mit Bluetooth[6]. Falls ein Kunde einen Standort betritt, kann die installierte Mobilapplikation im Hintergrund automatisch aktiviert werden, um das Check-in durchzuführen. Dabei ist diese Variante zwar zuverlässig, da der Standort mittels Beacons sehr genau bekannt ist, jedoch muss der Kunde Bluetooth aktiviert haben [8].

Für den Händler ist dies eine einfache und kostengünstige Lösung, da er lediglich die Identität des Beacons (UUID, Major und Minor) bekannt geben muss [9]. Dabei können Beacons sowohl in Form von Hardware Beacons als auch mit Hilfe eines Mobiltelefons simuliert verwendet werden [10].

3.1.1.3 GPS

GPS ermöglicht im Freien eine sehr genaue Positionsbestimmung. Innerhalb von Gebäuden funktioniert die Positionierung jedoch nur noch ungenau [11] [4].

Falls sich ein Kunde in unmittelbarer Nähe von mehreren Händlern befindet, kann nicht zwischen den einzelnen Händlern unterschieden werden. Dies ist jedoch nicht problematisch, da ein Kunde an mehreren Standorten gleichzeitig eingecheckt sein kann.

3.1.1.4 NFC

Mittels NFC (Near Field Communication) kann eine exakte Positionierung erfolgen [12]. Nachteil für den Kunden ist, dass er sein Gerät direkt an den NFC-Tag halten muss, damit ein automatischer Check-in möglich ist. Der Vorteil dieser Methode ist jedoch, dass durch die genaue Positionierung im Raum zusätzliche Informationen erfasst werden können. Dies könnte im Fall eines Restaurants die Tischnummer sein.

3.1.1.5 QR-Code

Der QR-Code verwendet ein ähnliches Prinzip wie NFC. Der Kunde muss einen QR-Code manuell mit seiner Smartphone Kamera scannen. Dadurch ist eine genaue Positionierung möglich. Auch hier können durch den QR-Code zusätzliche Informationen erfasst werden, indem beispielsweise an jedem Tisch im Restaurant ein eindeutiger QR-Code platziert wird.

3.1.2 Pairing Prozess

Nachdem der Kunde eingecheckt ist und sich auf der virtuellen Gästeliste befindet, muss der Kunde zusätzlich gepairt werden, bevor ihm der Händler eine Rechnung stellen kann. Dieser Schritt ist notwendig, um allfällige Zusatzinfos (z.B. Tischnummer im Restaurant) einzuholen, sowie die Sicherheit zu erhöhen, um ungewollte Transaktionen zu verhindern. Das Pairing kann je nach Szenario entweder durch den Kunden oder den Händler gestartet werden.

Erst nach dem erfolgreichen Pairing ist der Kunde komplett identifiziert und kann via BeePay bezahlen.

3.1.2.1 Pairing durch den Händler

Der Händler hat jederzeit Zugriff auf die virtuelle Gästeliste, welche alle eingecheckten Kunden enthält. Dadurch kann er den Kunden aus der Gästeliste auswählen und den Pairing Prozess starten. Nun sind je nach Use Case noch zusätzliche Infos vom Händler oder Kunden notwendig. Sind keine zusätzlichen Informationen und keine Sicherheitsbestätigung vom Kunden notwendig, merkt dieser im einfachsten Fall nichts vom Pairing Prozess.

In *Abbildung 6 Pairing durch Händler (Use Case Restaurantbesuch)* wird das Pairing durch den Händler anhand des Use Case Restaurant dargestellt. Die zentrale Rolle übernimmt der Händler

[6] http://www.bluetooth.com/

(Restaurantbedienung), indem er den eingecheckten Kunden nach dem Namen fragt und diesen aus der virtuellen Gästeliste auswählt. Dabei wird der Händler aufgefordert, eine Zusatzinformation einzugeben. In diesem Fall ist das die Tischnummer. Mit der Eingabe ist das Pairing abgeschlossen und der Händler kann ab sofort eine Rechnung stellen. Der Kunde sieht ebenfalls den Status, dass die Identifikation abgeschlossen ist. Damit weiss der Kunde genau, dass er das Restaurant nach dem Essen einfach verlassen kann und die Rechnung via BeePay erhält.

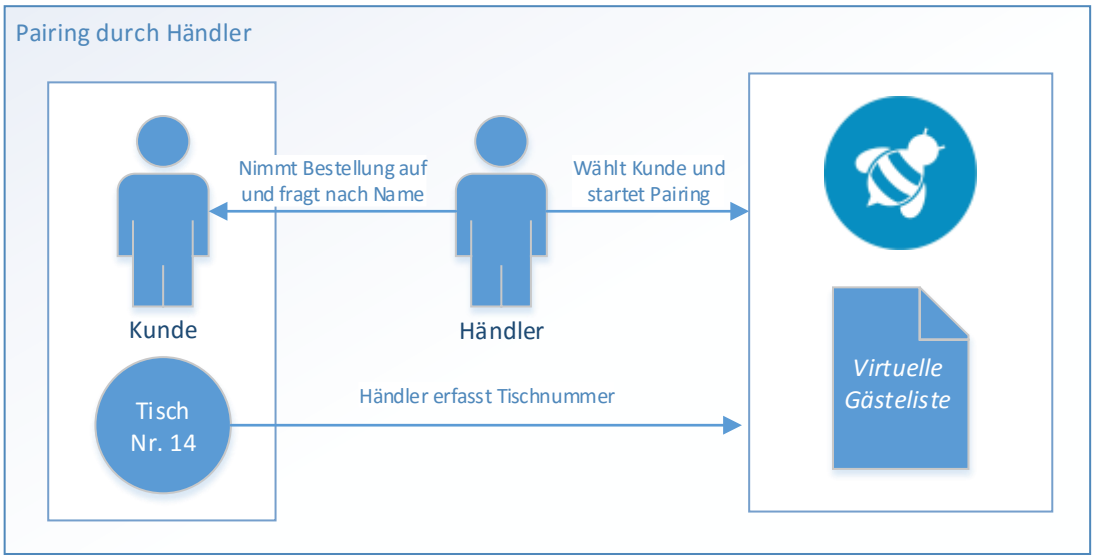

Abbildung 6 Pairing durch Händler (Use Case Restaurantbesuch)

3.1.2.2 Pairing durch den Kunden

Je nach Anwendungsszenario kann es sinnvoll sein, dass der Kunde den Pairing Prozess selbständig startet. Ein Vorteil dabei ist, dass der Prozess beschleunigt werden kann. Der Kunde muss nicht warten bis der Händler das Pairing startet. Ausserdem kann es sein, dass der Kunde Zusatzinformationen angeben muss, welche sowieso eine Interaktion mit dem Kunden erfordert.

In der folgenden Grafik wird dasselbe Szenario wie beim vorherigen Pairing durch den Händler nochmals dargestellt. Diesmal wird jedoch das Pairing durch den Kunden selbst gestartet. Der Händler wird dazu nicht benötigt. Der Kunde wird dabei aufgefordert, seine Tischnummer einzugeben. Durch die Eingabe der Tischnummer ist das Pairing und somit die Identifikation abgeschlossen. Der Händler kann dem Kunden nun eine Rechnung senden.

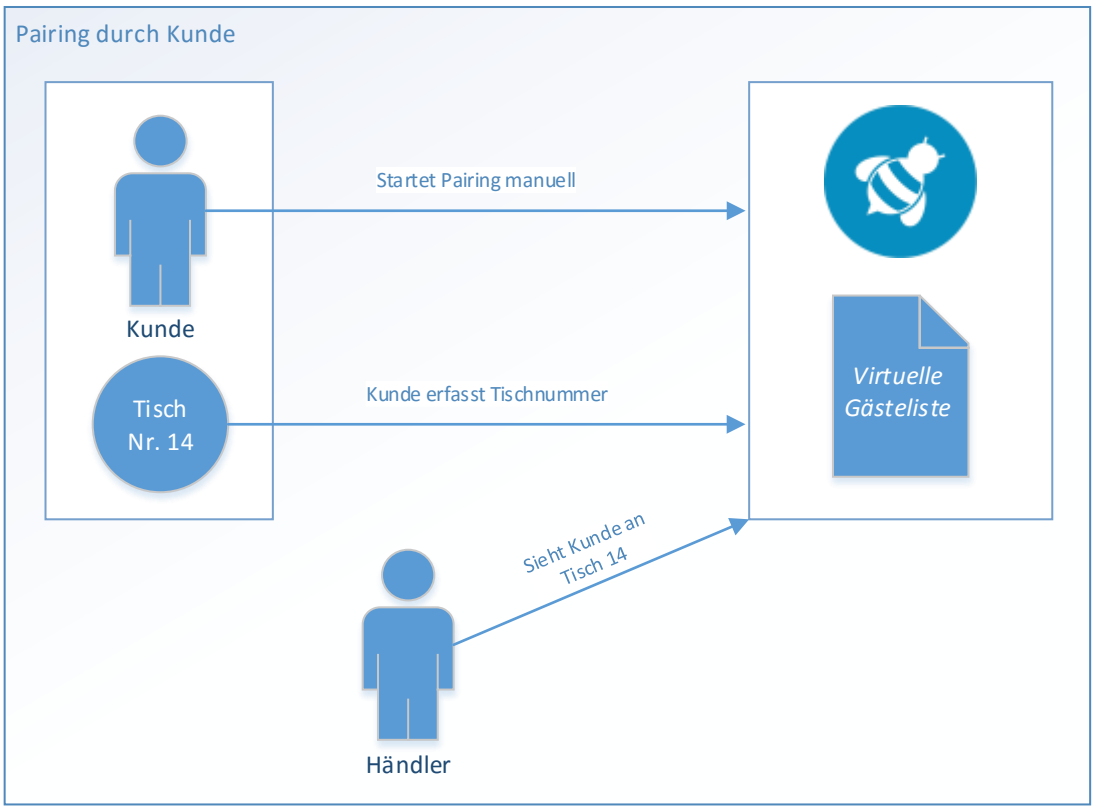

Abbildung 7 Pairing durch Kunde (Use Case Restaurantbesuch)

3.1.2.3 Optionale Zusatzinfos

Je nach Anwendungsfall werden zusätzliche Informationen benötigt, um das Pairing abzuschliessen. Dies kann in einem Restaurant die Tischnummer sein, damit später die Rechnung zugewiesen werden kann. Allgemein gibt es folgende Kommunikationsmöglichkeiten, um Zusatzinfos einzuholen:

- Eingabe der Infos durch Händler
  - Händler kennt die benötigte Information (z.B. Tischnummer)
  - Händler fragt beim Kunden nach (z.B. gewünschte Preiskategorie)
- Eingabe durch Kunde
  - Kunde startet Pairing und wird automatisch zur Eingabe von Zusatzinfos aufgefordert.
  - Händler fordert Zusatzinfo vom Kunden via Mobilapplikation an.

Es gibt verschiedene Wege wie die Informationen erfasst werden können. Neben der manuellen Erfassung können folgende Technologien eingesetzt werden.

QR-Code	Informationszuordnung: QR-Code beinhaltet Information. Der Händler muss den QR-Code generieren und ausdrucken. Sonstiges: Der QR-Code muss gut ersichtlich sein, somit braucht er Platz im Lokal, was bei gewissen Händler nicht erwünscht ist.
NFC	Informationszuordnung: Dem NFC-Tag müssen die Zusatzinformationen hinterlegt werden. Diese Aufgabe muss der Händler übernehmen. Sonstiges: NFC kann momentan nur von Android Benutzern verwendet werden, da iOS keinen Zugriff auf die NFC-Komponente gewährt.
Beacons	Informationszuordnung: Der Beacon enthält nur numerische Major und Minor Werte. Sonstiges: Keine exakte Zuordnung der Information möglich, da der Beacon von allen Geräten im Umkreis erkannt wird und die Positionierung zu ungenau ist.

3.1.2.4 Sicherheitseinstellungen

Wenn das Pairing durch den Händler ausgelöst wird, muss der Kunde sich darauf verlassen können, dass keine falschen Buchungen getätigt werden können. Falls der Kunde automatisch eingecheckt und vom Händler gepairt wird, ist die Identifikation ohne explizite Kenntnisnahme des Kunden abgeschlossen und es kann theoretisch eine unrechtmässige Buchung durch den Händler stattfinden. Die gleiche Situation kann auftreten, wenn der Kunde das Pairing ohne aktive Beteiligung des Händlers startet. Falls der Kunde eine falsche Information angibt, ist die Identifikation zwar abgeschlossen, jedoch kann der Händler durch die falschen Informationen die Rechnung nicht korrekt stellen.

Um dies zu verhindern, sind diverse Sicherheitsstufen vorgesehen, welche je nach Wunsch auf der Seite des Kunden und auch des Händlers aktiviert werden können. Der Kunde kann dabei für verschiedene Händler jeweils eigene Sicherheitseinstellungen definieren. Als einfachste Massnahme bietet sich auf beiden Seiten die manuelle Bestätigung des Pairings an. Somit kann das Pairing nicht abgeschlossen werden, ohne dass beide Parteien die Aktion manuell bestätigt haben.

3.2 Bezahlung

Nachdem die Identifikation des Kunden abgeschlossen ist, kann der Händler dem Kunden eine Rechnung senden. Zur Vereinfachung des Bezahlprozesses wird für den Prototyp lediglich eine Rechnung pro identifizierten Kunden zugelassen. Denkbar wäre jedoch auch, dass pro Identifikation mehrere Belastungen stattfinden können (z.B. einzelne Runden bezahlen in einer Bar).

3.2.1 Rechnungstellung

Die Rechnung wird durch den Händler ausgelöst. Dabei wird eine hohe Automation angestrebt. Dies wäre beim Use Case Restaurantbesuch durch die Anbindung an die bestehenden Kassensysteme möglich. Der Händler müsste nur noch den Tisch schliessen. Falls für den Kunden eine Identifikation besteht, kann die Rechnung automatisch ausgelöst werden.

Falls keine Kassenanbindung besteht, kann die Zahlung manuell über die BeePay Mobilapplikation oder Plattform ausgelöst werden.

3.2.2 Zahlungsausführung

Die Zahlung wird über die BeePay Plattform abgewickelt. Dabei wird die Anbindung an den Payment Service Provider Datatrans verwendet.[7]

3.2.2.1 Ablehnung der Rechnung

Wie bei jeder normalen Rechnung kann es auch bei BeePay vorkommen, dass der Händler dem Kunden eine falsche Rechnung zustellt. In diesem Fall soll der Kunde die Möglichkeit haben, die Rechnung abzulehnen oder falls die Zahlung automatisch ausgelöst wurde, eine Rückerstattung zu verlangen. Da dies ein Spezialfall ist, wird dieser Prozess im Prototyp nicht umgesetzt.

[7] Siehe Kapitel 7.1.1 *Anbindung an Datatrans*

4 Umsetzung

Die Umsetzung umfasst die Entwicklung der Prototypen für iOS und Android sowie die Anpassung des bestehenden API. Dabei wurden die Applikationen für iOS und Android unabhängig voneinander entwickelt. Lediglich das Backend API wurde gemeinsam erweitert und benutzt. Dadurch können iOS oder Android spezifische Technologien unabhängig voneinander genutzt werden, wodurch eine optimale Lösung für beide Plattformen entsteht. Ausserdem entstehen durch die unabhängige Entwicklung zwei leicht unterschiedliche Prototypen mit Vor- und Nachteilen. Die Erkenntnisse beider Prototypen können somit für eine spätere Weiterentwicklung genutzt werden.

Nachfolgend wird zuerst eine Übersicht der BeePay Plattform gezeigt. Anschliessend wird die Umsetzung der Identifikation anhand der einzelnen Prototypen für iOS und Android beschrieben.

4.1 BeePay Plattform

4.1.1 Systemübersicht

Aufgrund der bestehenden Architektur aus der Studienarbeit konnten die neuen Clients für iOS und Android ohne Probleme an den Core Tier angebunden werden. Die Clients greifen via REST-Schnittstelle auf das WebAPI zu. [1, p. 13]

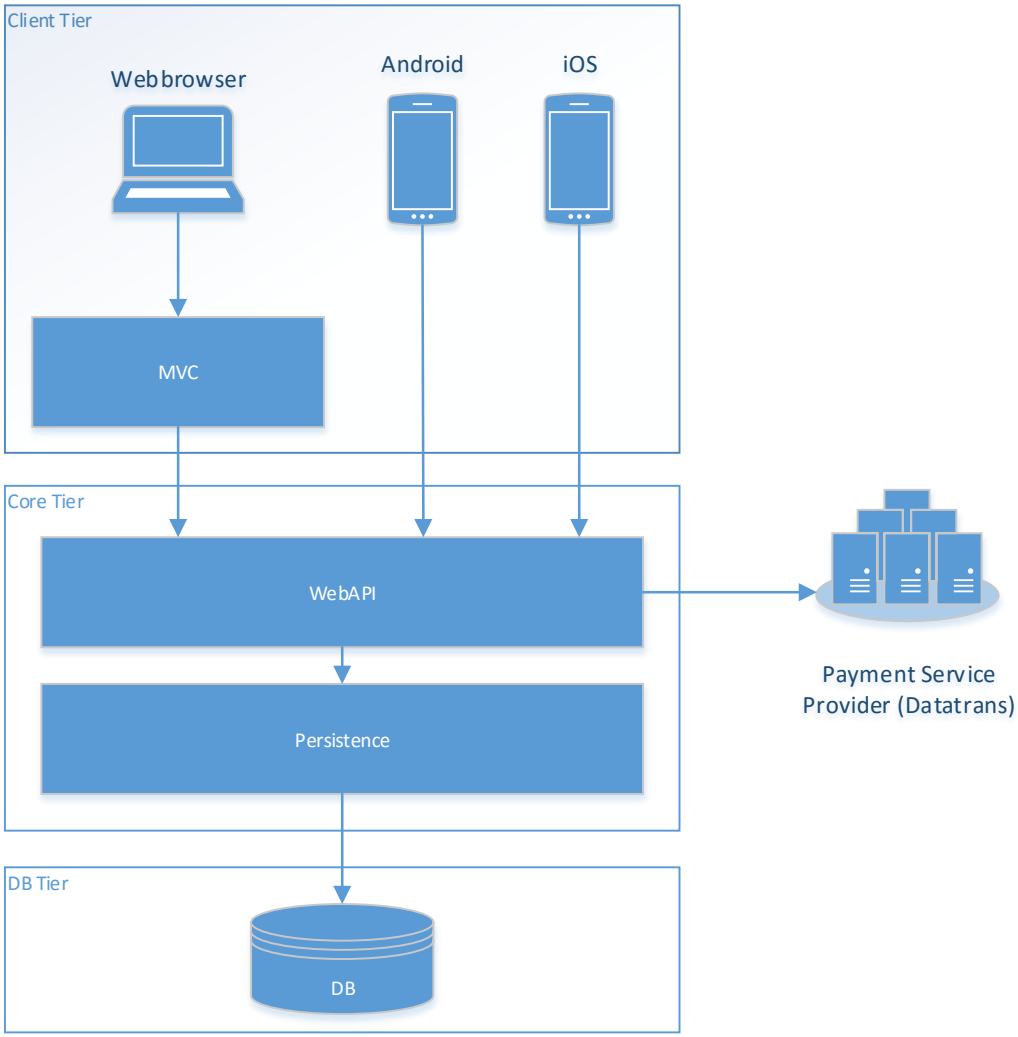

Abbildung 8 Systemübersicht BeePay Plattform

4.1.2 Authentifizierung und Autorisierung

Die Authentifizierung und Autorisierung wurde bereits während der Studienarbeit gelöst und basiert auf ASP.Net Identity unter Verwendung von JSON Web Token (JWT). [1, p. 14]

Die Clients werden gegenüber dem WebAPI mittels JWT **Bearer Token** authentifiziert. Durch den Login mit E-Mail-Adresse und Passwort wird der Benutzer authentifiziert und erhält einen Token. Zusätzlich bestimmt der Role Claim, ob es sich beim Benutzer um einen **Customer** oder **Merchant** handelt. Bei jedem API Zugriff wird der Token zur Authentifizierung mitgesendet. Mittels Role Claims wird der Zugriff autorisiert.

4.1.3 Deployment

Das Projekt wird bei jedem Commit automatisch auf Azure deployed. Dabei erfolgt das Deployment in eine Testumgebung, welche während der Entwicklung genutzt wird. In der Testumgebung wird die Datenbank nach jedem Start der Azure Web App neu generiert. Da für die Testumgebung die kostenfreie Version von Azure Web App verwendet wird, wird die Umgebung nach einer gewissen Zeit von Azure gestoppt, falls kein Zugriff erfolgt.

Die Stable Umgebung wird nur manuell deployed und ist mit der kostenpflichtigen **Always On** Option konfiguriert. Somit steht die Umgebung jederzeit sofort zur Verfügung [13]. Ausserdem wird in der Stable Umgebung die Datenbank nicht nach jedem Deployment neu generiert.

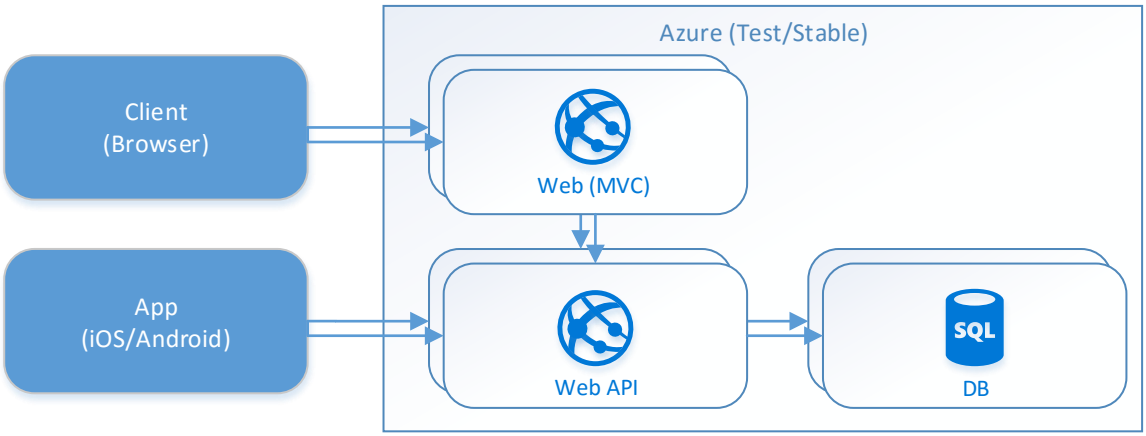

Abbildung 9 Azure Deployment Übersicht

4.2 Domainmodell

Das Domainmodell wurde grundsätzlich aus der Studienarbeit übernommen. Da die Frage der Identifikation jedoch erst im Lösungskonzept genauer betrachtet wurde, stellte sich heraus, dass das Domainmodell teilweise vereinfacht werden kann. Nachfolgend werden dazu die einzelnen Klassen genauer erklärt.

Die Teile der Rechnung, Zahlungsmittel und Zahlungsabwicklung wurden aus der Studienarbeit übernommen und nicht angepasst. Daher wird dieses Konzept hier nicht nochmals genauer erläutert.

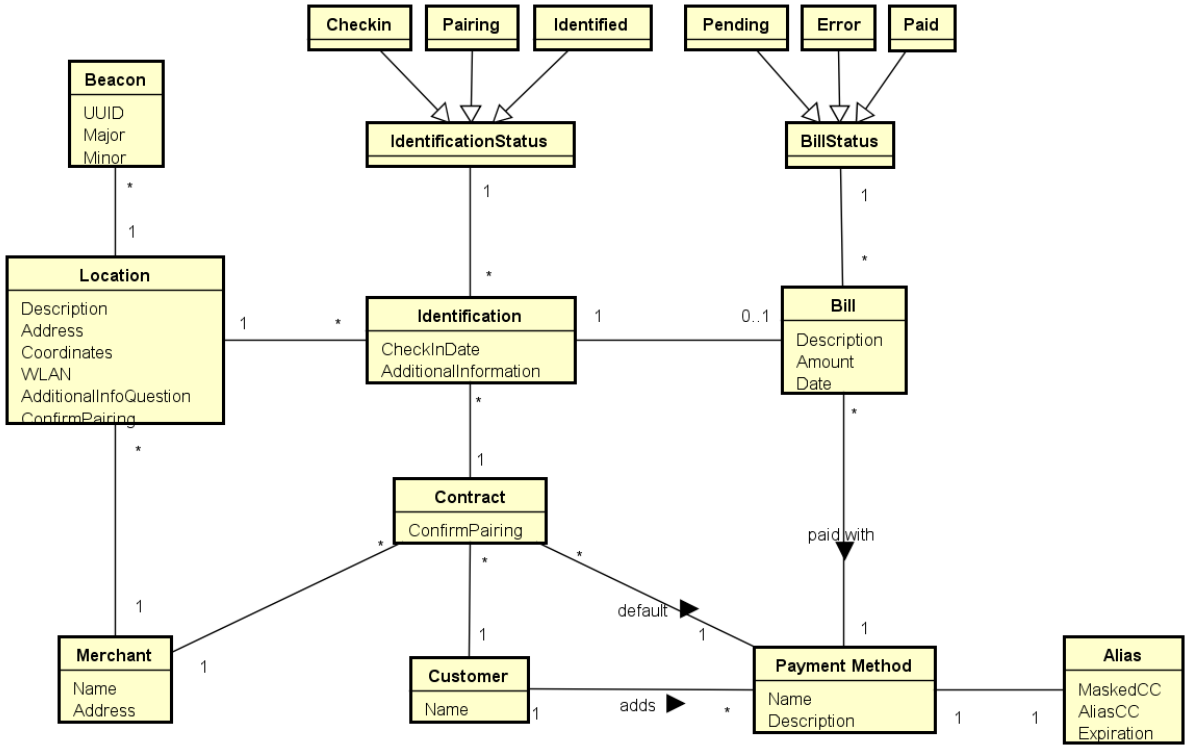

Abbildung 10 BeePay Domainmodell

4.2.1 Location und Beacon

Ein Merchant kann beliebig viele Locations haben. Dies können beispielsweise einzelne Filialen einer Restaurantkette sein. Jede Location wird mittels GPS-Koordinaten exakt positioniert. Zudem können zusätzliche Methoden zur Positionierung hinzugefügt werden. Einerseits ist dies der WLAN–Name. Andererseits kann eine Location beliebig viele Beacons beinhalten, welche ebenfalls zur Ortung verwendet werden.

Zusätzlich wird bei der Location optional eine Zusatzinfo-Frage hinterlegt, welche für eine Identifikation notwendig ist.

4.2.2 Contract

Zwischen Kunde und Händler muss ein Vertrag abgeschlossen werden, bevor BeePay zur Bezahlung verwendet werden kann. Der Kunde legt mit diesem Vertrag fest, mit welchem Zahlungsmittel er Rechnungen bezahlen möchte und welche Sicherheitseinstellungen beim Pairing angewendet werden.

Ausserdem bestätigt der Kunde durch diesen Vertrag explizit, dass er beim Händler eingecheckt werden möchte und der Händler ihm Rechnungen stellen darf.

4.2.3 Identification und Status

Die Identifikation ist der Kern des Domainmodells und wird im nächsten Kapitel genauer erläutert. Grundsätzlich befindet sich eine Identifikation in einem von drei verschiedenen Status (Check-in, Pairing oder Identified).

4.2.3.1 Check-in

Bei diesem Status befindet sich der Kunde auf der virtuellen Gästeliste eines Standorts. Entweder wurde der Kunde automatisch oder manuell eingecheckt. Voraussetzung ist ein bestehender Contract mit dem Händler.

4.2.3.2 Pairing

Sobald der Kunde oder Händler den Pairing Vorgang startet, hat die Identifikation den Status „Pairing". Dieser Status behält sie so lange, bis allfällige Zusatzinfos erfasst, sowie Securitychecks durchgeführt wurden.

4.2.3.3 Identified

Mit diesem Status ist die Identifikation komplett abgeschlossen. Der Händler kann nun dem Kunden eine Rechnung senden. Für den Kunden bedeutet dies, dass er den Standort verlassen kann und die Rechnung via BeePay erhält und bezahlen kann.

4.3 Identifikation

Die Identifikation besteht aus zwei Schritten, dem Check-in und dem Pairing. Die Aufteilung kann auch als Teilung zwischen Datenschutz (Check-in) und Sicherheit (Pairing) betrachtet werden. Mittels Check-in wird sichergestellt, dass die Datenschutzaspekte gewährleistet sind. Mit dem Pairing werden die Sicherheitsaspekte gewährleistet. Dabei gibt es verschiedene Sicherheitsmassnahmen, welche sowohl Kunde als auch Händler einstellen können.

4.3.1 Check-in

Damit das Check-in gestartet werden kann, muss der Kunde auf der BeePay Plattform einen Vertrag mit dem Händler abschliessen. Anschliessend kann er am Standort des Händlers erkannt und eingecheckt werden. Hat der Kunde im Voraus einen Vertrag auf der BeePay Plattform mit dem Händler abgeschlossen wird das automatische Check-in[8] durchgeführt, sobald er sich in der Nähe des Händlers befindet. Andernfalls muss der Kunde manuell einchecken[9].

4.3.1.1 Automatisches Check-in

Beim automatischen Check-in werden die Contracts des Kunden von der BeePay Plattform geladen. Diese beinhalten die Informationen über die Locations des Händlers. Die Locations und Beacons werden in der BeePay Mobilapplikation registriert und informieren diese, sobald ein Standort des Händlers in der Nähe ist. Dadurch wird das Check-in für den Kunden für den entsprechenden Standort gestartet.

Das Check-in bleibt während zwei Stunden gültig. Bei nochmaligem Check-in wird die Zeit aktualisiert, damit das Check-in wieder zwei Stunden gültig bleibt. Dieser Prozess ist zentral bei der Identifikation, da das Check-in gleichzeitig von mehreren Technologien wie GPS, WLAN oder Beacons gestartet werden kann. Somit wird verhindert, dass ein Kunde mehrmals an einem Standort eingecheckt ist.

[8] Siehe Kapitel 4.3.1.1 *Automatisches Check-in*
[9] Siehe Kapitel 4.3.1.2 *Manuelles Check-in*

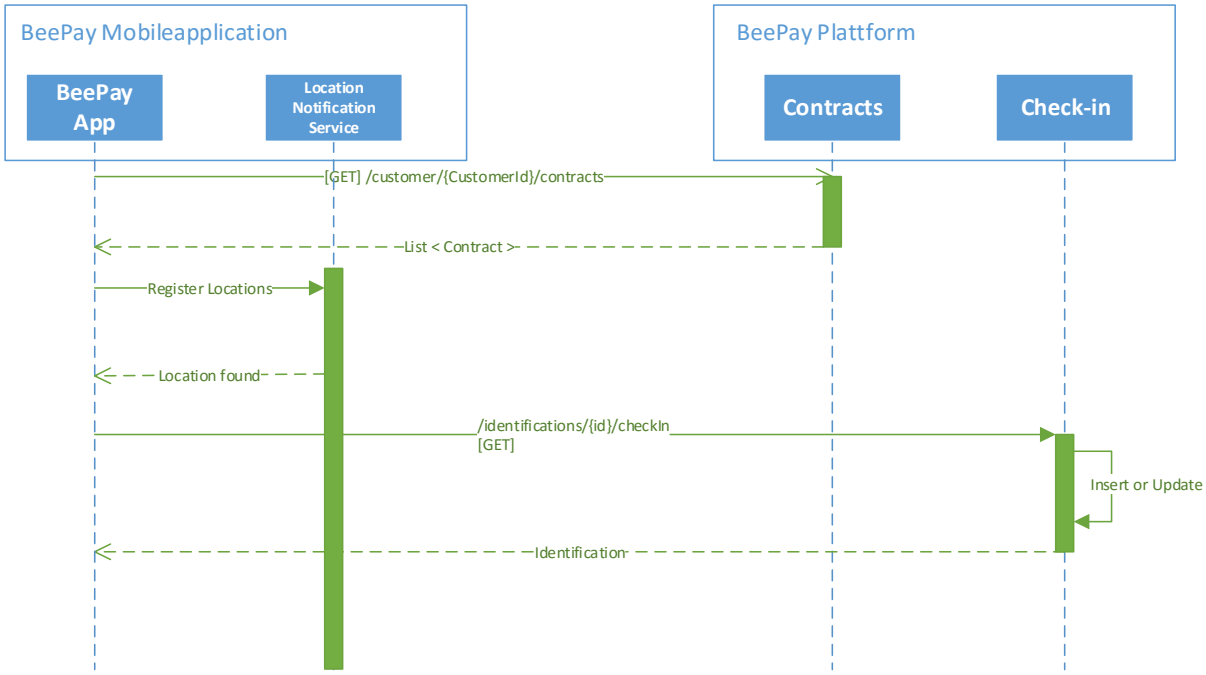

Abbildung 11 Sequenzdiagramm automatisches Check-in

4.3.1.2 Manuelles Check-in

Beim manuellen Check-in wird eine Liste aller Merchants und deren Locations dargestellt. Der Customer wählt eine Location davon aus. Dadurch wird das Check-in für die ausgewählte Location durchgeführt, welches sich gleich wie das automatische Check-in verhält.

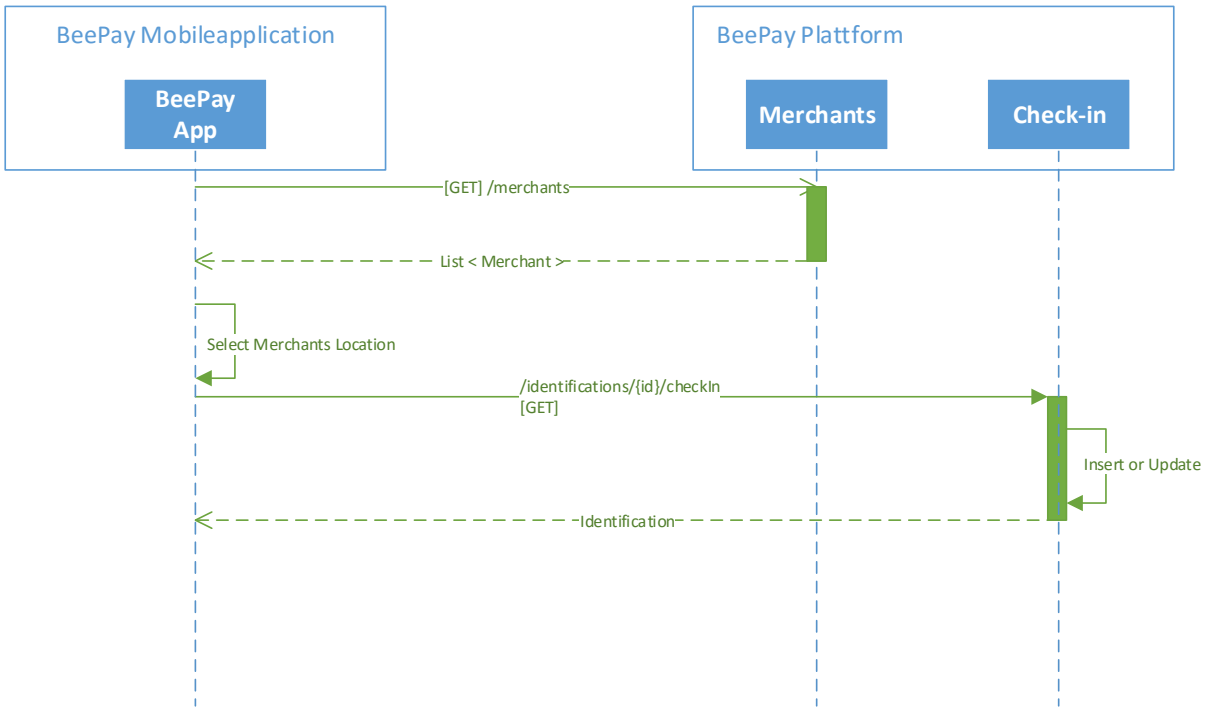

Abbildung 12 Sequenzdiagramm manuelles Check-in

4.3.2 Pairing

Beim Pairing wird vorausgesetzt, dass ein Check-in durchgeführt wurde. Das Pairing kann entweder vom Kunden oder vom Händler gestartet werden. Während dem Pairing werden die Sicherheitskriterien des Kunden sowie die des Händlers überprüft. Diese werden anhand des *Pairing Security Check Handlers* realisiert. Wurde das Pairing erfolgreich durchgeführt, kann der Händler dem Kunden eine Rechnung der konsumierten Ware senden.

4.3.2.1 Kunde startet Pairing

Der Kunde startet das Pairing manuell über die BeePay Mobilapplikation. Dabei wird das Pairing auf dem Server gestartet. Falls die Identifikation zusätzliche Daten benötigt, wird der Kunde aufgefordert diese einzugeben. Mittels dem *Pairing Security Check Handler* werden die Sicherheitseinstellungen abgefragt. Diese sind optional und je nach Einstellung des Kunden oder des Händlers unterschiedlich. Der *Pairing Security Check Handler* gibt an, ob die Security Checks schon abgeschlossen oder noch ausstehend sind. Wenn diese abgeschlossen sind, wird der Identifikationsstatus auf „Done“ gesetzt. Von nun an kann der Händler dem Kunden die konsumierte Ware in Rechnung stellen.

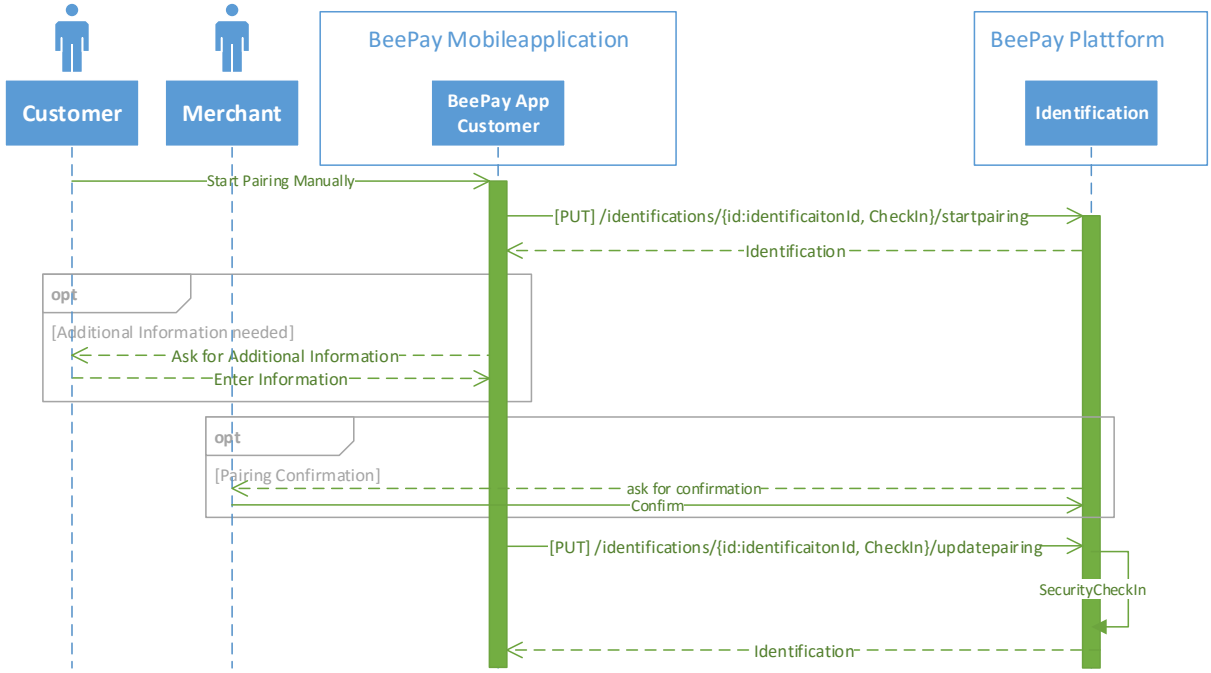

Abbildung 13 Sequenzdiagramm Kunde startet Pairing

4.3.2.2 Händler startet Pairing

Informiert der Kunde den Händler, dass er gerne mit BeePay bezahlen möchte, startet der Händler das Pairing, indem er den Kunden aus der virtuellen Gästeliste auswählt. Dabei muss der richtige Kunde ausgewählt werden. Damit dieser Prozess möglichst einfach ist, werden Kunden mit Name und optional mit Profilbild dargestellt. Dies vereinfacht die Auswahl des Kunden für den Händler.

Wenn der Händler den Kunden ausgewählt hat, muss dieser falls erforderlich noch Zusatzinformationen erfassen. Anschliessend wird der Security Check durchgeführt. Falls dieser erfolgreich war, ist die Identifikation abgeschlossen und der Händler kann dem Kunden die konsumierte Ware verrechnen.

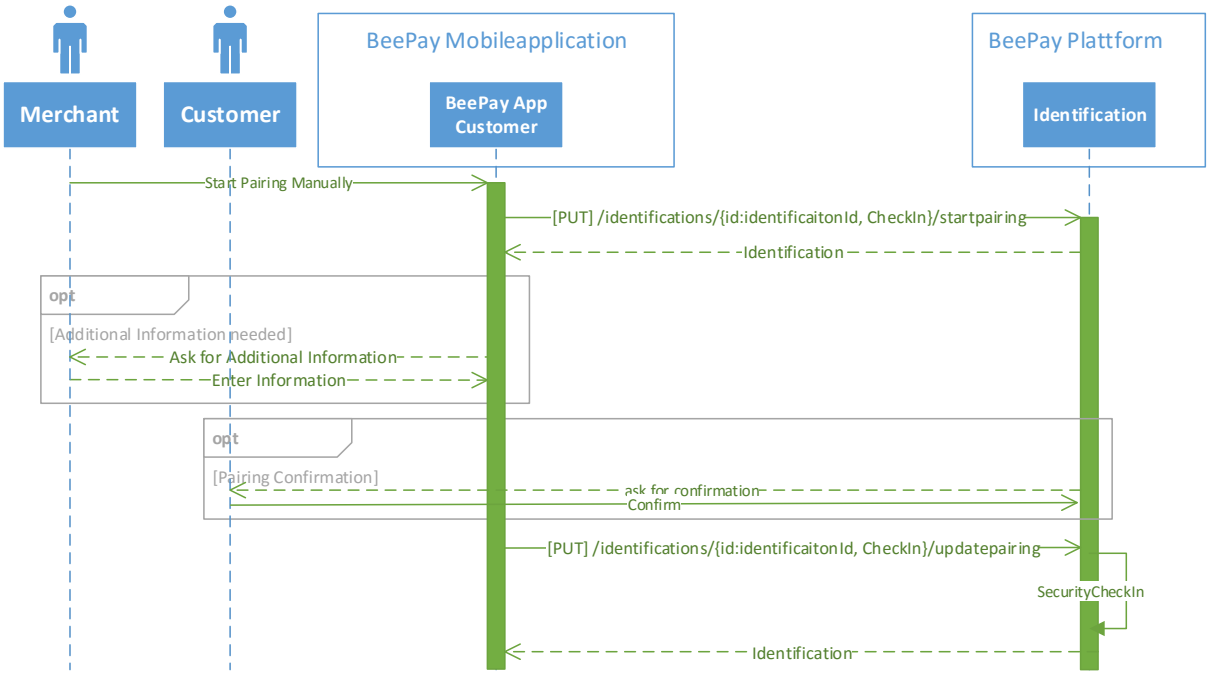

Abbildung 14 Sequenzdiagramm Händler startet Pairing

4.3.2.3 Pairing Zusatzinformationen

Während des Pairings können Zusatzinformationen erfasst werden. Diese Informationen können dem Händler helfen, dem Kunden die korrekte Rechnung zu stellen. Als Beispiel kann in einem Restaurant die Tischnummer als Zusatzinformation erfasst werden, sodass der Händler anschliessend weiss, welchen Tisch er über BeePay verrechnen muss.

Bei der Erhebung der Informationen wurde nur die manuelle Erfassung der Informationen umgesetzt. Die anderen Technologien wie Beacon, NFC oder QR-Code wurden geprüft und sind technisch betrachtet einfach umsetzbar. Das Problem bei diesen Technologien ist, dass der Händler die gewünschten Informationen der Technologie zuordnen muss und eine Interaktion des Kunden erforderlich ist.

4.3.2.4 Pairing Security Check Handler

Der SecurityCheckHandler[10] ist das zentrale Element beim Pairing, um die Sicherheitseinstellungen des Kunden und Händlers zu überprüfen. Die Idee dahinter ist, dass alle Sicherheitseinstellungen ISecurityCheck implementieren. Wobei in der FactoryMethode vom SecurityCheckHandler überprüft wird, ob der SecurityCheck für die ISecurtiyCheck Implementation aktiv ist oder nicht. Der Händler kann dann überprüfen, ob die SecurityChecksFinished sind. Dabei ruft er alle ISecurityCheck auf und führt SecurityCheckFinished durch.

[10] Siehe *Abbildung 15 UML SecurityCheckHandler und ISecurityCheck*

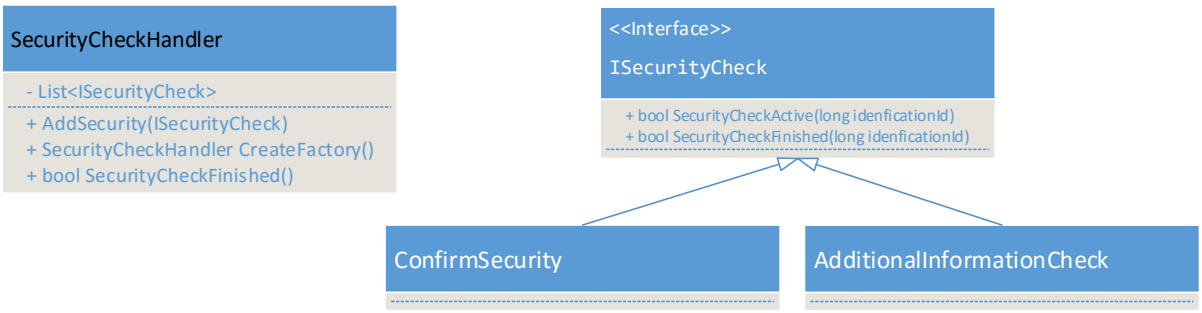

Abbildung 15 UML SecurityCheckHandler und ISecurityCheck

Bei jeder Änderung an einem Identifikationsobjekt kann der Security Handler initialisiert werden. Anschliessend wird überprüft, ob die Security Checks fertig sind oder nicht. Falls diese abgeschlossen sind, wird der Status der Identifikation auf „Done“ gesetzt und der Händler kann dem Kunden die konsumierte Ware in Rechnung stellen.

```
var securityHandler = SecurityCheckHandler.CreateFactory(id, _identificationRepository);

if (securityHandler.SecurityCheckFinished())
{
        identification.IdentificationStatus = IdentificationStatus.Done;
        updatedIdentifications = _identificationRepository.Update(identification);
}
```

Abbildung 16 Initialisierung SecurityCheckHandler

4.4 iOS

Für die Entwicklung der iOS App wurde Swift 2[11] verwendet. Die Entwicklung erfolgt nativ mithilfe der Xcode Entwicklungsumgebung für iOS 9 und höher. Da sich der Prototyp auf die Identifikation fokussiert, wurden die verschiedenen Technologien aus dem Lösungskonzept auf ihre Praxistauglichkeit für iOS geprüft. Die iOS Plattform ist teilweise sehr restriktiv, daher können nicht alle Technologien beliebig genutzt werden. Daraus entstand ein Identifikationskonzept, welches unter *Identifikationskonzept* genauer beschrieben wird.

4.4.1 Architektur

Die iOS App baut auf einer Model-View-Controller Architektur auf. Das MVC Pattern ist zentraler Bestandteil jeder iOS Applikation. [14, pp. 22-24]

4.4.1.1 Externe Libraries

Für die Einbindung von externen Libraries wird CocoaPods[12] verwendet. CocoaPods ist ein weit verbreiteter Dependency Manager für Swift und Objective-C Projekte. Mittels Podfile werden die entsprechenden Libraries angegeben.

```
platform :ios, '9.0'
use_frameworks!

target 'BeePay' do
    pod 'Alamofire', '~> 3.0'
    pod 'ObjectMapper', '~> 1.0'
end
```

Abbildung 17 Podfile

- **Alamofire**[13]
  Wird für den Zugriff auf die REST-Schnittstelle verwendet.
- **ObjectMapper**[14]
  Mapping von JSON auf Model und umgekehrt.

Zusätzlich wird für Remote Push Notifications das Windows Azure Messaging [15] Framework verwendet. Dieses ist Bestandteil des Azure Mobile Services SDK. Da dafür kein CocoaPod vorhanden ist, wird dies manuell eingebunden. Windows Azure Messaging bietet momentan nur eine Objective-C Version. Um diese zu verwenden, wird zusätzlich ein Bridging-Header eingesetzt [15]. Dieser ist notwendig, um Objective-C Code in Swift verwenden zu können.

4.4.1.2 API-Zugriff

Die Applikation greift via Alamofire auf das WebAPI der BeePay Plattform zu. Die Aufrufe werden abstrahiert, um unabhängig vom dahinterliegenden API zu sein. Dafür wird das API-Parameter-Abstraction [16] Beispiel verwendet. Zusätzlich wird der Authorization Header [17] automatisch hinzugefügt, falls zuvor ein Token gespeichert wurde.

[11] https://developer.apple.com/swift
[12] https://cocoapods.org
[13] https://github.com/Alamofire/Alamofire
[14] https://github.com/Hearst-DD/ObjectMapper
[15] https://azure.microsoft.com/en-us/documentation/articles/notification-hubs-ios-get-started
[16] https://github.com/Alamofire/Alamofire#api-parameter-abstraction
[17] https://github.com/Alamofire/Alamofire#crud--authorization

```
Alamofire.request(BeePayApi.Customer.GetIdentifications)
    .validate()
    .responseJSON {
        response in
        BeePayRepository.identifications.removeAll()
        switch response.result {
        case .Success(let data):
            print(data)
            if let identifications = Mapper<Identification>().mapArray(data) {
                BeePayRepository.identifications = identifications
            }
        case .Failure(let error):
            print(error)
        }
        refreshControl?.endRefreshing()
        self.updateUI()
}
```

Abbildung 18 Alamofire API-Request

4.4.2 Kunden und Händler Applikation

Der Fokus des iOS Prototyps liegt auf der Kunden Mobilapplikation. Jedoch soll als Resultat ein fertiger Showcase entstehen. Dafür wird auch eine einfache Händler Mobilapplikation benötigt. Der Bereich für den Händler umfasst dabei die wichtigsten Funktionen, um den gesamten Prozess von der Identifikation bis zur Rechnung aufzuzeigen.

Händler und Kunde verwenden die gleiche Mobilapplikation. Nach dem erfolgreichen Login wird je nach Rolle des Benutzers entweder der Kunden- oder Händlerbereich angezeigt.

4.4.3 Identifikationskonzept

Das Identifikationskonzept beschreibt die Nutzung der vorgeschlagenen Technologien aus dem Lösungskonzept in Bezug auf die iOS Plattform. Dabei fällt auf, dass die Nutzung von möglichen Technologien stark davon abhängt, welche iOS Einstellungen der Benutzer konfiguriert und welche Zugriffsrechte der App vom Benutzer gewährt werden.

4.4.3.1 GPS und Beacons

Grundsätzlich ist der Zugriff auf den aktuellen Standort nur möglich, wenn dies der Benutzer für die Mobilapplikation erlaubt. Dabei bietet iOS dem Benutzer verschiedene Einstellungen. Es wird unterschieden zwischen dem Zugriff auf den Standort, wenn die App benutzt wird (Zugriffsrecht „Bei Benutzung“) oder wenn die App beziehungsweise das Telefon nicht benutzt wird (Zugriffsrecht „Immer“). Der Benutzer muss den Zugriff auf den Standort immer erlauben, damit ein automatischer Check-in möglich ist.

Beacons werden von iOS gleich wie ein GPS-Standort behandelt. Dies bedeutet, dass für die Erkennung von Beacons ebenfalls der Zugriff auf den Standort vom Benutzer gewährt werden muss. Zusätzlich muss für die Verwendung von Beacons Bluetooth aktiviert sein.

4.4.3.2 Ausgeschlossene Technologien

Die Technologien *NFC*, *QR-Code* und *WLAN* aus dem Lösungskonzept werden während der Umsetzung nicht weiterverfolgt. Neuere iPhone Generationen sind zwar auf der Hardwareseite mit NFC ausgerüstet, jedoch haben App-Entwickler keinen Zugriff auf die NFC-Schnittstelle, da diese exklusiv für Apple Pay genutzt wird [16]. Bei WLAN können zwar Informationen zum aktuell verbundenen Netzwerk ausgelesen werden, jedoch hilft dies nicht weiter, da keine passive Überwachung von Netzwerken in Reichweite möglich ist. Die Verwendung von QR-Codes wurde nicht weiterverfolgt, da dieser manuell vom Kunden gescannt werden müsste und somit zu umständlich ist.

4.4.3.3 Manuelle Identifikation

Falls für die automatische Identifikation keine Technologie verfügbar ist, weil der Benutzer die entsprechenden Rechte nicht erteilt hat, bleibt nur noch eine manuelle Identifikation. Dabei wird der Check-in manuell via App durchgeführt, indem der Kunde innerhalb der App den gewünschten Standort des Händlers auswählt.

4.4.4 Core Location Framework

Für den Zugriff auf Standort und Beacons wird das offizielle Core Location Framework[18] von iOS verwendet. Der Vorteil dabei ist, dass GPS-Koordinaten und Beacons fast gleich behandelt werden. Beide werden als Region definiert und können mittels Region Monitoring[19] überwacht werden. Bei GPS-Koordinaten wird ein Radius in Metern um den exakten Punkt herum definiert. Bei Beacons setzt sich die Region aus UUID, Major und Minor Wert zusammen.

Sobald sich das iOS Gerät innerhalb einer Region befindet, wird die *DidEnterRegion* Methode innerhalb der BeePay Applikation aufgerufen und somit ein Check-in durchgeführt. Dabei kann es vorkommen, dass diese Methode mehrfach aufgerufen wird. Dies ist der Fall, wenn die GPS-Koordinaten und zusätzlich Beacons verwendet werden. Ein mehrfacher Check-in ist jedoch nicht weiter tragisch, da dies von der BeePay Plattform entsprechend gehandhabt wird.

4.4.4.1 Region Limitierung durch iOS

Das Monitoring von GPS-Regionen wird durch iOS auf maximal 20 pro Applikation limitiert. Dies ist ein Problem, falls ein Benutzer mehre Verträge mit Händlern hat und die Summe aller Standorte dieses Limit überschreitet. Ein möglicher Lösungsansatz für die Umgehung dieses Problems ist, lediglich die Regionen in unmittelbarer Nähe des Kunden zu überwachen. Nun können je nach Bewegung des Kunden weiter entfernte Standorte wieder aus der Überwachung gelöscht und nähere neu hinzugefügt werden. Da dies eine aufwändige Implementation zur Folge hätte, wurde darauf im Prototyp verzichtet.

4.5 Android

Die Android Applikation wurde mit Java entwickelt. Dabei wird bewusst auf Xamarin[20] oder andere Crossplattform-Technologien verzichtet, aus der Befürchtung, dass die zusätzlichen Technologien nicht alle Funktionalitäten von Android unterstützen. Zusätzlich waren die bisherigen Erfahrungen mit Xamarin oder anderen Crossplattform-Technologien eher negativ.

4.5.1 Architektur

Die Applikation verwendet die vom Android Framework vorgegebenen Konzepte. Dabei wird für das Frontend XML verwendet, welches von Activities[21] bzw. Fragments[22] aufgerufen wird.

[18] https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/LocationAwarenessPG/CoreLocation/CoreLocation.html
[19] https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/LocationAwarenessPG/RegionMonitoring/RegionMonitoring.html
[20] http://xamarin.com/
[21] http://developer.android.com/reference/android/app/Activity.html
[22] http://developer.android.com/guide/components/fragments.html

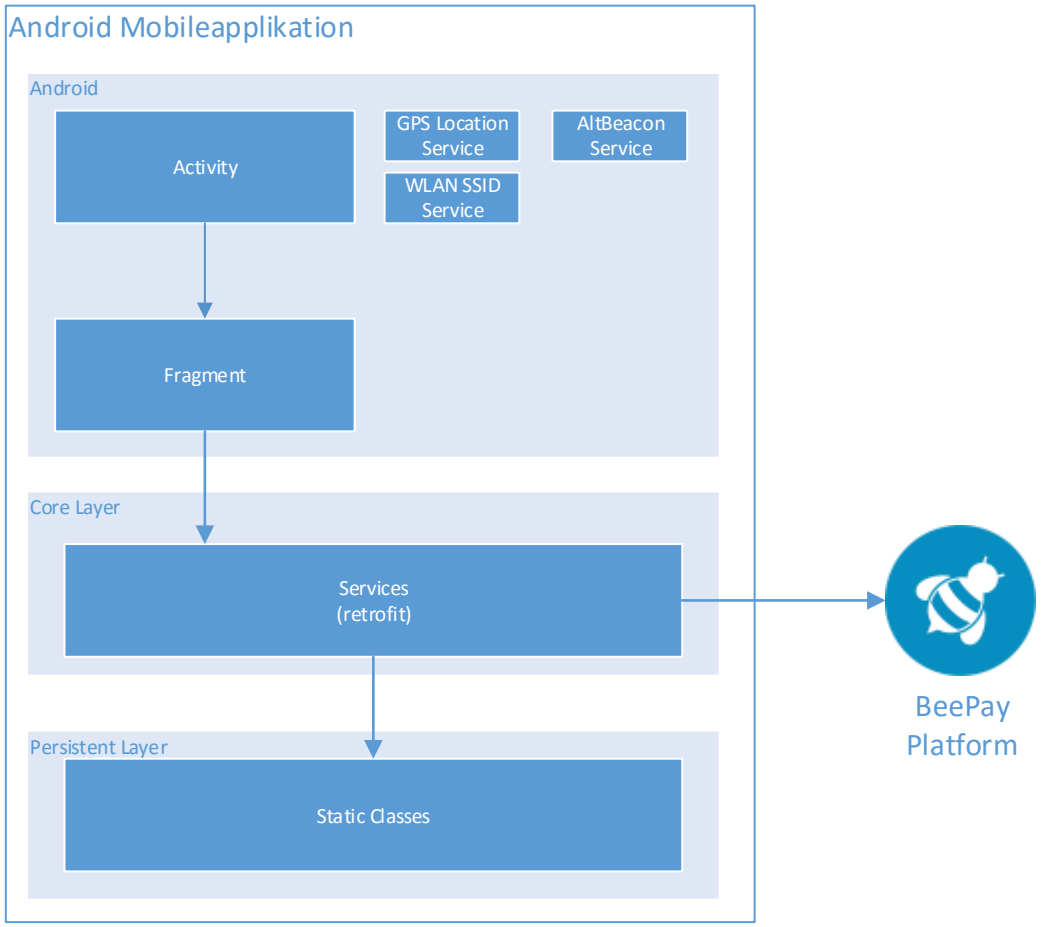

Abbildung 19 Android Mobilapplikation Übersicht

Die Businesslogik der Applikation wird in der BeePay Plattform behandelt. Diese wird mittels dem retrofit[23] Framework über die REST-Schnittstelle der BeePay Plattform aufgerufen. Die Daten werden bei erfolgreichem Aufruf in einer statischen Klasse gespeichert, welche den Persistent Layer darstellt. Dies wird bewusst so gemacht, da die Applikation nur online verfügbar sein muss. Bei der Umsetzung war klar, dass dies nicht die idealste Lösung für den Persistent Layer darstellt, jedoch genügt dies für die Ansprüche an den Prototyp.

4.5.1.1 Kommunikation zwischen Activities, Fragments und Services

In diesem Kapitel werden kurz die wichtigsten Komponenten erläutert, welche nichts mit der Identifikation zu tun haben. Die Umsetzung der Identifikation wird im Kapitel *Identifikationskonzept* beschrieben.

BeePay besteht aus zwei Activities. Der Loginscreen wird beim Start der Applikation dargestellt. Diese Activity stellt das Login dar und ist für die Rechteanfrage an den Benutzer zuständig. Seit Android 6.0 (API-Level 23) wird die Rechtefreigabe während der Laufzeit der Applikation vergeben, damit der Benutzer mehr Kontrolle über die vergebenen Rechte besitzt[24].

Nach dem Login hat der Benutzer Zugriff auf die Funktionalitäten der Applikation. Jeder Screen wurde als Fragment umgesetzt. Die MainActivity ist dafür zuständig, das richtige Fragment je nach Status der

[23] http://square.github.io/retrofit/
[24] http://developer.android.com/training/permissions/requesting.html

Mobilapplikation zu laden. Das Hauptlayout wie Menu oder ActionBar [25] wird ebenfalls in der MainActivity behandelt.

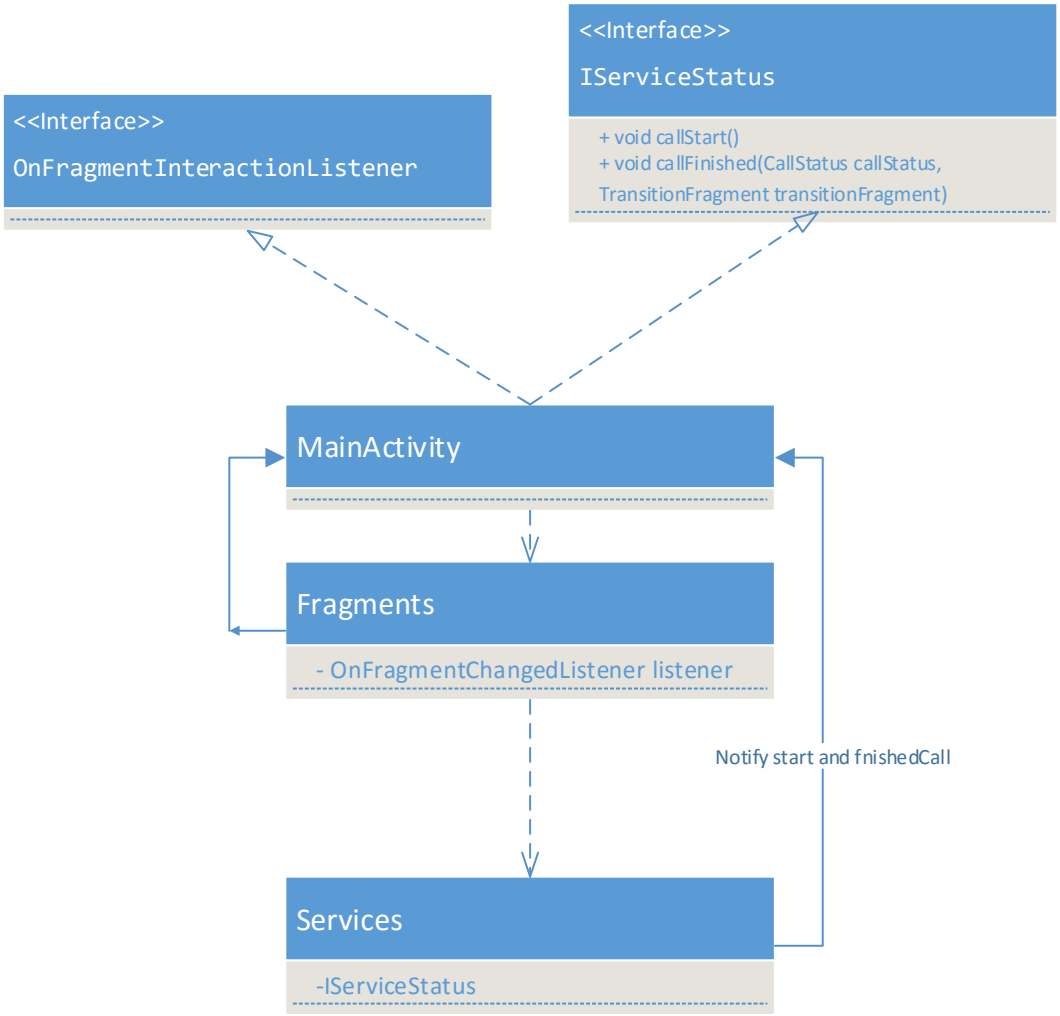

Abbildung 20 Kommunikation zwischen Activities, Fragments und Services

Die MainActivity ruft die einzelnen Fragments auf, wobei diese mit dem OnFragmentChangedListener angeben, ob sich das Fragment geändert hat. Die MainActivity implementiert den Listener und lädt je nach Mobilapplikationsstatus das nächste Fragment.

Die Services rufen mit retrofit die REST-Schnittstelle der BeePay Plattform auf. Der Aufruf erfolgt asynchron und kann je nach Internetverbindung länger oder kürzer dauern. Während des Aufrufs wird ein Loadingscreen dargestellt. Die MainActivity implementiert das IServiceStatus Interface, welches beim Start und Ende der Services aufgerufen wird. Somit kann die MainActivity während dieser Zeit den Loadingscreen darstellen. Endet ein Service wird der CallStatus, erfolgreich oder nicht, sowie das TransitionFragment mitgegeben. Anhand dieses TransitionFragments und dem CallStatus stellt die MainActivity den nächsten Screen dar.

[25] http://developer.android.com/training/appbar/index.html

4.5.1.2 API-Zugriff

Mittels retrofit greift die Applikation auf das WebAPI der BeePay Plattform zu. Dafür wird jeweils im Interface eine Methode pro angesprochener Schnittstelle erstellt. Mittels Annotation werden URL und Parameter für den Aufruf definiert.

```
/**
 * Gets the bills of a customer
 * @param authorisation string to authorize the customer
 * @param customerId Id of customer
 * @return the bills of a given customer
 */
@Headers({
        "Content-Type: application/json"
})
@GET("customers/{customerId}/bills")
Call<List<Bill>> getBill(@Header("Authorization") String authorisation,
@Path("customerId") int customerId);
```

Abbildung 21 Interfacemethode für den WebAPI Call

Beim Aufruf erstellt retrofit mittels reflection die Methode getBill. Indem noch der CallBack implementiert wird, kann das Handling bei erfolgreichem oder nicht erfolgreichem Call umgesetzt werden.

```
/**
 * Gets the bill data
 */
public void getBillData() {
    serviceStatus.callStart();
    Retrofit retrofit = getRetrofit(getOkHttpClient(), getGson());

    //Creates a new instance of the IBillService
    IBillService billService = retrofit.create(IBillService.class);

    //Calls getBill from customer on the WebApi BeePay plattform
    Call<List<Bill>> billCall = billService.getBill("Bearer " + Utils.User.accessToken,
Utils.User.id);
    billCall.enqueue(new Callback<List<Bill>>() {

        @Override
        public void onResponse(Response<List<Bill>> response, Retrofit retrofit) {
            Log.i("Response", response.toString());
            if (response.body() != null) {
                Utils.BILLS = response.body();
                serviceStatus.callFinished(CallStatus.Successfull,
                                           TransitionFragment.BillList);
            } else {
                Log.i(Utils.TAG, "bills error");
                serviceStatus.callFinished(CallStatus.Error, TransitionFragment.None);
            }
        }

        @Override
        public void onFailure(Throwable t) {
            serviceStatus.callFinished(CallStatus.Error, TransitionFragment.None);
            Log.i(Utils.TAG, "bills error: " + t.toString());
        }
    });
}
```

Abbildung 22 WebAPI Aufruf mittels retrofit

4.5.2 Identifikationskonzept

Für die Identifikation wurden die vorgeschlagenen Technologien aus dem Lösungskonzept darauf untersucht, ob diese in Android zur Verfügung stehen. Für jede Technologie gibt es einen eigenen

Service, bei dem sich die Applikation mit den Daten wie GPS-Koordinaten oder WLAN-SSID registrieren kann. Der Service meldet, wenn sich der Kunde an einem Standort befindet, damit das Check-in durchgeführt werden kann.

Falls zwei Services gleichzeitig das Check-in starten, werden zwei Check-in Requests an die BeePay Plattform gesendet. Die Plattform stellt sicher, dass ein Kunde nur einmal pro Location eingecheckt wird.

4.5.2.1 WLAN-Service

Der WLAN-Service ist eine eigene Implementation. Der Service wird nach dem Login gestartet und überwacht die WLAN-SSIDs in der Umgebung. Falls eine WLAN-SSID eines Händlers erkannt wird, wird der Kunde automatisch eingecheckt.

4.5.2.2 Beacon-Service

Der Beacon-Service wurde mit AltBeacon[26] umgesetzt. Der Händler fügt beim Einrichten des Standorts seine Beacons hinzu oder er startet virtuelle Beacons via seiner Mobilapplikation. Wenn ein Kunde den Beacon sieht, wird er beim Händler eingecheckt.

4.5.2.3 GPS-Service

Der GPS-Service wurde implementiert, indem der Location Service[27] von Android verwendet wird. Nach dem Login wird der Service gestartet und es können GPS-Koordinaten hinterlegt werden. Kommt ein Kunde in die Nähe der hinterlegten Koordinaten, wird das Check-in gestartet.

Diese Koordinaten muss der Händler vorher pro Location hinterlegen. Dabei kann er die aktuellen GPS-Koordinaten seines Mobiltelefons oder die ungefähren GPS-Koordinaten einer beliebigen Adresse abfragen[28].

4.5.2.4 NFC-Service

Der NFC-Service wurde zwar bei der Umsetzung ausprobiert, jedoch nicht vollständig umgesetzt. Der Grund dafür ist, dass iOS kein NFC unterstützt und somit die Funktion nur für Android zur Verfügung steht.

Bei der Umsetzung wurde auf den P2P-Mode[29] gesetzt, wobei der Händler seine NFC-Tags auf der BeePay Plattform hinterlegen muss. Wenn ein Kunde das Lokal betritt, muss er sein Mobiltelefon auf den NFC-Tag legen, damit das Check-in durchgeführt wird.

4.5.2.5 QR-Code Service

Um QR-Codes zu lesen wird die Library zxing[30] verwendet, da diese sowohl auf Android als auch iOS läuft. Nach ersten Versuchen wurde die Implementation von QR-Codes verworfen, da es zu umständlich für den Kunden ist. Er muss jedes Mal sein Mobiltelefon hervornehmen, die Applikation starten und dann den QR-Code lesen, damit er an einem Händlerstandort eingecheckt wird.

Vorstellbar ist, dass die Zusatzinformationen während des Pairings via QR-Code eingelesen werden. Dieses Feature wurde jedoch aus Zeitgründen nicht weiterverfolgt.

---

[26] https://github.com/AltBeacon/android-beacon-library
[27] http://developer.android.com/guide/topics/location/index.html
[28] http://developer.android.com/reference/android/location/Geocoder.html
[29] http://developer.android.com/guide/topics/connectivity/nfc/index.html
[30] https://github.com/zxing/zxing

5 Ergebnis

Aus dem Lösungskonzept wurde der Identifikationsprozess in einem Prototyp für Android und iOS umgesetzt.

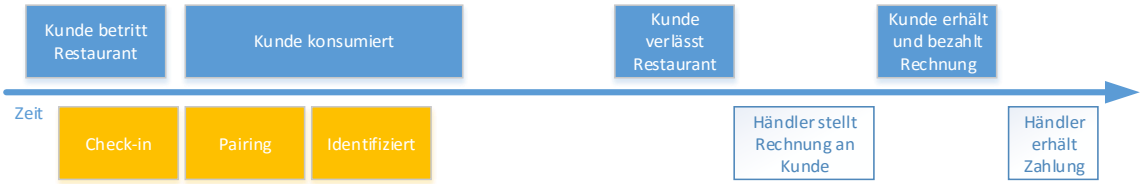

Abbildung 23 Identifikation anhand des Use Case Restaurantbesuch

5.1 iOS Applikation

Für iOS entstand eine kombinierte Applikation, welche sowohl vom Kunden als auch vom Händler benutzt werden kann. Nachfolgend wird die Applikation anhand einer Identifikation mittels Beacons präsentiert. Um besser zwischen Kunden und Händleraktionen unterscheiden zu können, wurden die Screenshots jeweils oben links als Kunde oder Händler gekennzeichnet.

5.1.1 Identifikation via Beacon

Ein Kunde wird in einer Pizzeria automatisch via Beacon eingecheckt und anschliessend identifiziert. Für die Identifikation in der Pizzeria ist nebst der Bestätigung von Kunde und Händler auch die Tischnummer als Zusatzinfo erforderlich.

5.1.1.1 Check-in

Der Kunde betritt das Restaurant. Innerhalb des Restaurants befindet sich ein Beacon, welcher nun in Reichweite zum Kunden ist. Die Applikation erkennt den Beacon und sendet im Hintergrund den Check-in an die BeePay Plattform. Der Benutzer wird mittels Push Benachrichtigung informiert (*Abbildung 24 Kunde erhält Push Benachrichtigung*). Der Händler sieht in seiner Gästeliste den eingecheckten Kunden (*Abbildung 25 Händler sieht Gästeliste mit Kunden*).

Abbildung 24 Kunde erhält Push Benachrichtigung *Abbildung 25 Händler sieht Gästeliste mit Kunden*

5.1.1.2 Pairing

Der Händler nimmt nun wie gewohnt die Bestellung des Kunden entgegen. Der Kunde sagt während dem Bestellen, dass er gerne mit BeePay bezahlen möchte. Daraufhin wählt der Händler den Kunden anhand seines Namens aus der Gästeliste aus und startet das Pairing.

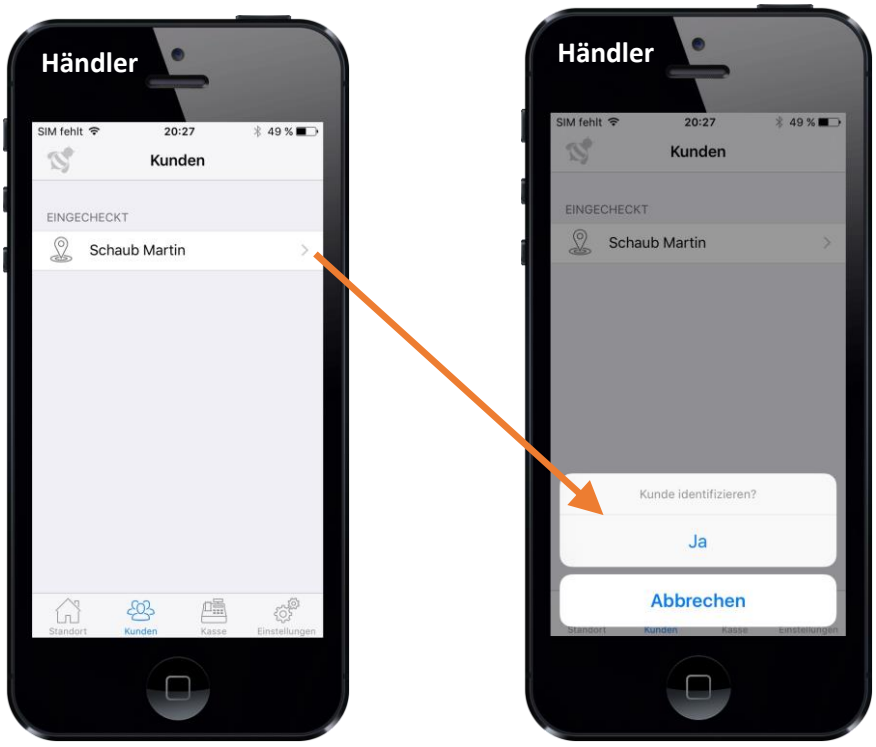

Abbildung 26 Händler startet Pairing

Um das Pairing abzuschliessen sind Zusatzinfos notwendig. In diesem Beispiel ist das die Tischnummer des Kunden, um später die entsprechende Rechnung zu stellen.

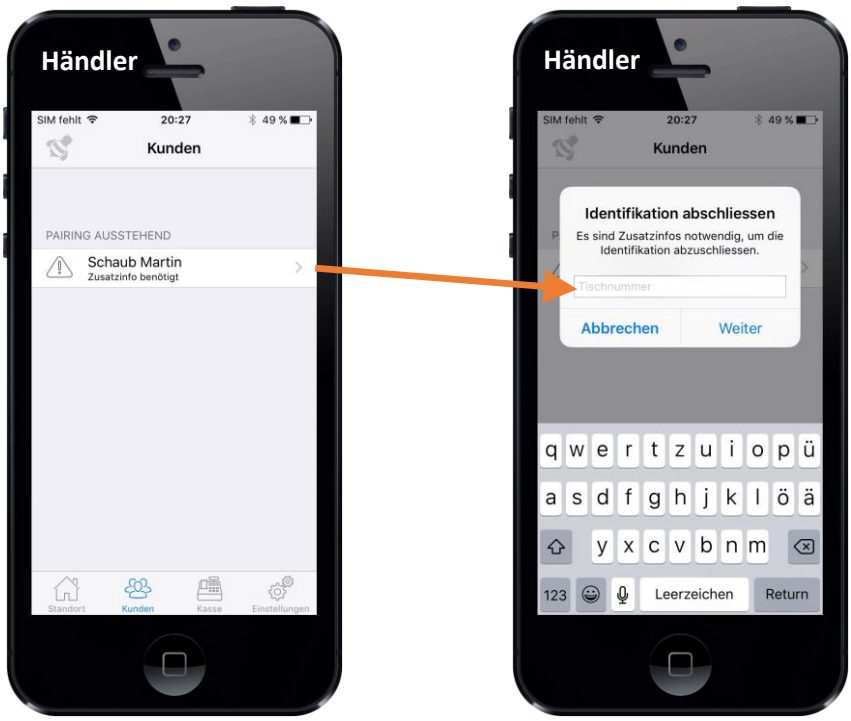

Abbildung 27 Händler erfasst Zusatzinfo

5.1.1.3 Identifiziert

Nun ist der Kunde identifiziert. Falls der Kunde mehr Sicherheit wünscht, kann er die manuelle Bestätigung aktivieren. Dadurch muss der Kunde beim Pairing zusätzlich die Identifikation bestätigen.

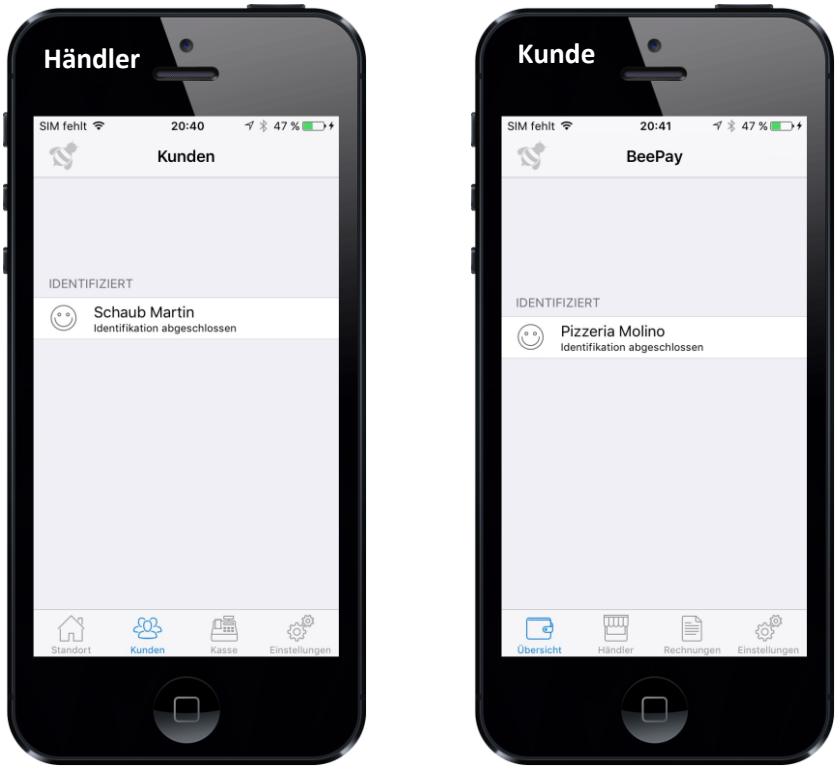

Abbildung 28 Identifikation abgeschlossen

Durch die abgeschlossene Identifikation kann der Händler dem Kunden nun eine Rechnung senden. Der Kunde erhält die Rechnung und kann diese überprüfen und bezahlen.

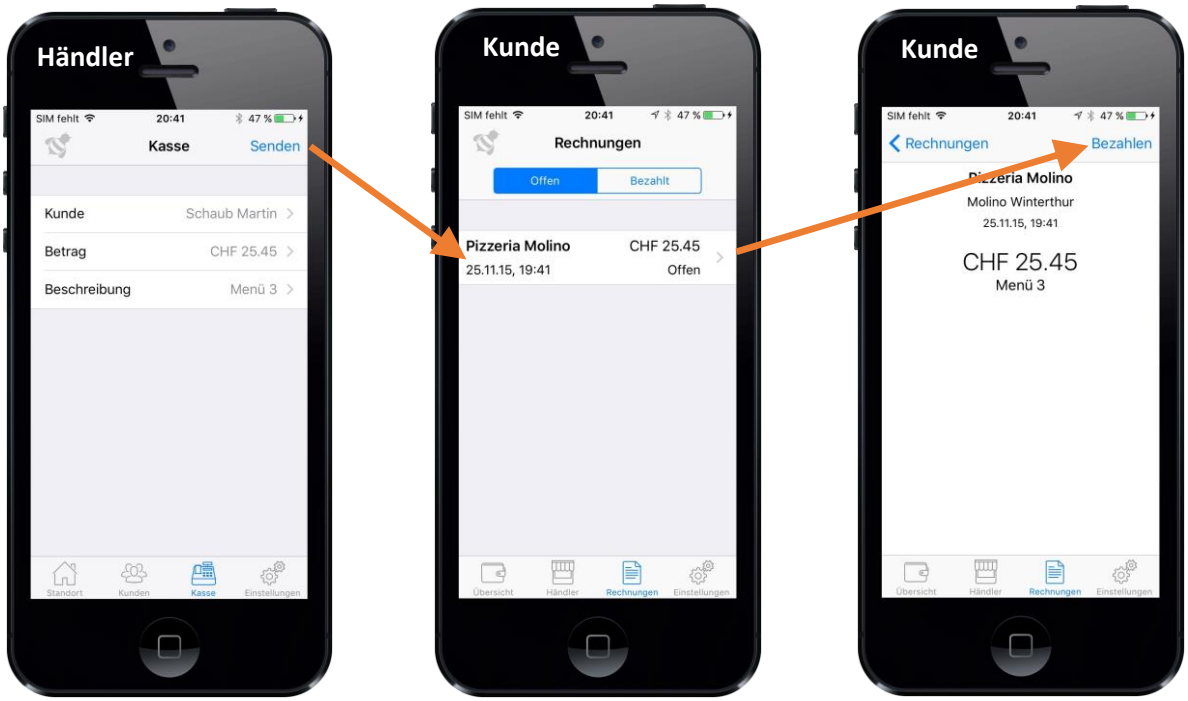

Abbildung 29 Rechnung senden und bezahlen

5.2 Android Applikation

5.2.1 Identifikation via GPS

In diesem Beispiel wird der Kunde per GPS eingecheckt. Er besitzt ein Android Mobiltelefon und betritt den Club Abrantix[31] der Abrantix AG. Auf seinem Mobiltelefon wird die Benachrichtigung angezeigt, dass er eingecheckt wurde.

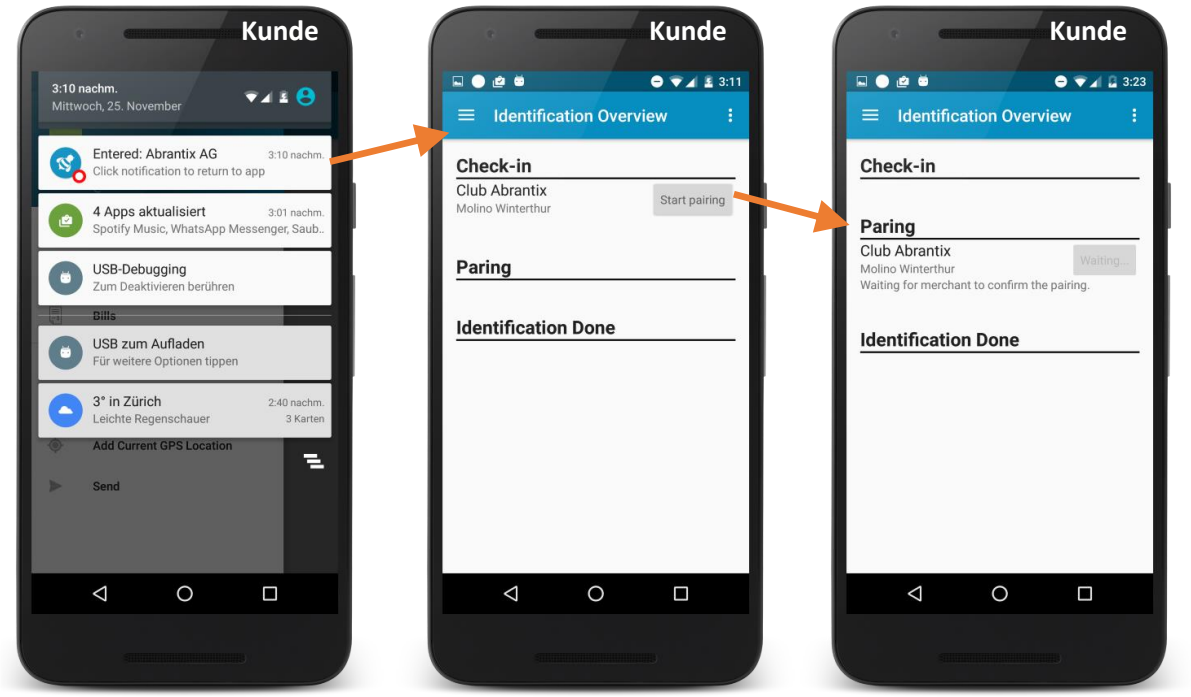

Abbildung 30 Android Check-in und Pairing

Er drückt auf die Benachrichtigung und wird auf den Screen „Identification Overview“ geleitet. Dort sieht er, dass er im Club Abrantix eingecheckt ist und kann das Pairing starten. Es ist auch möglich, dass der Händler das Pairing startet. Die Identifikation hat nun den Status „Pairing“. Weil der Händler die Sicherheitseinstellung *Pairing bestätigen* gewählt hat, muss er das Pairing noch manuell bestätigen, bevor die Identifikation auf „Identification Done“ gesetzt wird.

Der Händler bestätigt das Pairing und kann nun dem Kunden eine Rechnung für die konsumierte Ware stellen. Der Kunde erhält die Rechnung und bezahlt diese sogleich. Die Zahlung wird über die BeePay Plattform an Datatrans gesendet und dort abgewickelt.

[31] Fiktiver Club

5.3 BeePay Plattform

Die BeePay Plattform wurde um die Identifikation gemäss Lösungskonzept erweitert. Zusätzlich wurde eine Übersicht aller Kunden, welche den Identifizierungsprozess durchführen, erstellt. Dieser dient zum Testen der BeePay Applikation. Viel wichtiger ist jedoch das Aufzeigen der Übersicht der Identifikationen für den Händler. Wenn der Händler einen Kunden identifizieren muss, hilft ein Bild als Erkennungsmittel, um den Identifikationsprozess möglichst sicher zu gestalten. Dank dem Bild, welches der Benutzer hochladen kann, ist der Identifikationsprozess für den Händler einfacher und sicherer zugleich. Leider genügte die Zeit nicht, diese Funktionalität auch in die Mobilapplikationen einzubauen.

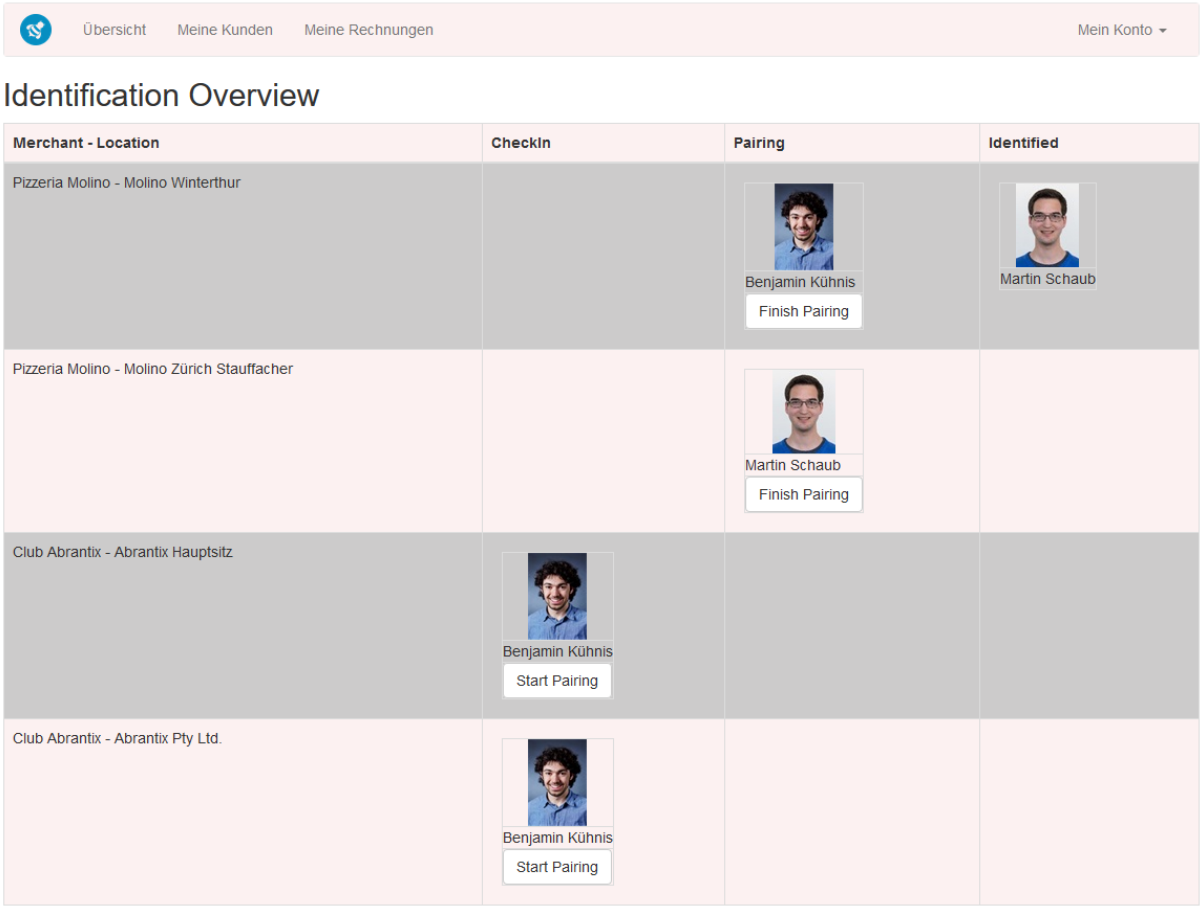

Abbildung 31 BeePay Plattform Identifikationsübersicht

5.4 Offene Punkte

Während der Umsetzung ergaben sich viele Details und Ideen, welche aus zeitlichen Gründen nicht weiterverfolgt werden konnten. Die nachfolgende Auflistung ist nicht abschliessend.

- **Usability**
  Dem Kunden soll ein möglichst einfacher und wenn möglich immer gleicher Prozess präsentiert werden. Ausserdem soll der Kunde jederzeit die Kontrolle über den Prozess haben.
- **Risiko bei Rechnungen**
  Es kommt überall vor, dass eine Rechnung nicht bezahlt wird oder die Rechnung falsch ist. Daher muss genau definiert werden, wer dieses Risiko trägt und was beim Eintritt eines solchen Falls geschieht.
- **Trinkgeld**
- **Einmaliger Vertrag**
- **Smartwatch Unterstützung**
- **Rechnungspositionen**
- **Rechnung mit Freunden teilen**

5.5 Ausblick

Die Firma Abrantix AG hat bereits während der Umsetzung dieser Arbeit einigen Partnern einen ersten Prototyp präsentiert. Die Partner aus der Finanzbranche zeigten sich sehr interessiert. Die Abrantix AG zieht daher eine Weiterführung des Projekts in Betracht.

Falls sich ein passender Partner findet, ist der nächste Meilenstein, das System in einer echten Umgebung zu testen. Dafür ist jedoch eine Weiterentwicklung des Prototyps notwendig.

Für einen Einsatz bei einem Händler muss der Prototyp zwingend in die bestehenden Systeme und Prozesse integriert werden, damit ein effizienter Einsatz möglich ist. Am wichtigsten ist die Integration ins Kassensystem, um die Rechnungsstellung automatisieren zu können. In einem weiteren Schritt ist sogar der Live Zugriff auf das Kassensystem denkbar. Dadurch kann der Kunde jederzeit auf die aktuellen Rechnungspositionen via BeePay zugreifen.

6 Appendix I – Zahlungssystem, Kapitel aus der Studienarbeit

Im aktuellen Zahlungssystem gibt es verschiedene Parteien. Je nach Art, wie bezahlt wird, kommen andere Parteien ins Spiel. Wird mit Debit- / Kreditkarte bezahlt, kommt das Vier-Parteien-System zum Tragen. Dabei kommt es noch darauf an, ob es sich um eine Distanz- oder Präsenzzahlung handelt. Bei einer Distanzzahlung kann es sein, dass noch ein **Payment Service Provider** seine Dienste anbietet.

6.1 Vier-Parteien-System

Der Kunde, auch Karteninhaber genannt, bezahlt normalerweise eine Jahresgebühr für die Karte an den **Issuer**. Bezahlt der Kunde mit einer Karte, wird ihm lediglich der Preis der Ware oder Dienstleistung verrechnet[32]. Der Händler muss hingegen dem Acquirer für die Transaktion eine **Merchant Service Charge (MSC)** Gebühr bezahlen. Der Acquirer zahlt seinerseits bei jeder Transaktion eine **Interchange Fee** an den Issuer. Der gleiche Prozess wird auch verwendet, wenn die Debit- / Kreditkarte virtualisiert wird, wie zum Beispiel bei einer Onlinetransaktion durch einen Payment Service Provider (PSP).

Abbildung 32 Vier-Parteien-System [17]

Bei einer Zahlung muss noch zwischen Distanz- und Präsenzzahlung unterschieden werden. Jedoch kommt bei beiden Zahlungsvarianten das Vier-Parteien-System zum Einsatz.

Distanzzahlung bedeutet, dass der Karteninhaber nicht persönlich mit seiner Karte vor Ort bezahlen muss. Diese Methode wird meistens bei Internetbezahlungen in Onlineshops verwendet. Die Anbindung an einen Onlineshop geschieht oft durch den Payment Service Provider (PSP). Dieser bietet dem Händler ein virtuelles Terminal an, welches die einfache Integration des Vier-Parteien-Systems in einen Onlineshop ermöglicht.

Bei Präsenzzahlungen hingegen ist der Karteninhaber vor Ort und tätigt die Zahlung via ein EFT/POS Gerät.

6.1.1 Issuer

Der Issuer ist für die Herausgabe der Debit- / Kreditkarte verantwortlich. Typischerweise ist der Issuer eine Bank oder gehört einer Bankengruppe.

[32] Es gibt Händler, welche die Gebühr dem Kunden weiter verrechnen, jedoch sind dies nur wenige.

6.1.2 Karteninhaber

Als Karteninhaber wird die Person bezeichnet, welche eine Debit- / Kreditkarte vom Issuer bezieht. Er wird auch KI (Karteninhaber) oder Endkunde genannt.

6.1.3 Acquirer

Der Acquirer verarbeitet Debit- / Kreditkarten Transaktionen. Möchte ein Händler Debit- / Kreditkarten Transaktionen verarbeiten muss er einen Vertrag mit dem Acquirer abschliessen.

6.1.4 Händler

Der Händler, auch Merchant genannt, ist die Instanz, welche die Ware/Dienstleistung dem Kunden verkauft. Bei einer Zahlung mit der Debit- / Kreditkarte zahlt der Händler dem Acquirer eine Transaktionsgebühr.

6.2 Bezahlen gegen Ware

Beim Bezahlen gegen Ware handelt es sich um ein Tauschgeschäft. Meistens wird mit Geld gehandelt. Es kann jedoch auch mit anderen Waren, wie Bitcoins oder Geschenkkarten ein Tausch stattfinden.

7 Appendix II – BeePay Plattform Systemübersicht, Umsetzung der Studienarbeit

7.1 Systemübersicht

Aufgrund des Lösungskonzepts wurde der Prototyp der Plattform entworfen. Es ist eine flexible Architektur entstanden, welche offen für zukünftige Erweiterungen ist. BeePay ist eine Drei-Tier-Applikation. Zentral ist dabei der Zugriff auf die Daten via universelles API. Dadurch lassen sich auch zukünftige Clients (z.B. Mobilapplikation) einfach anbinden.

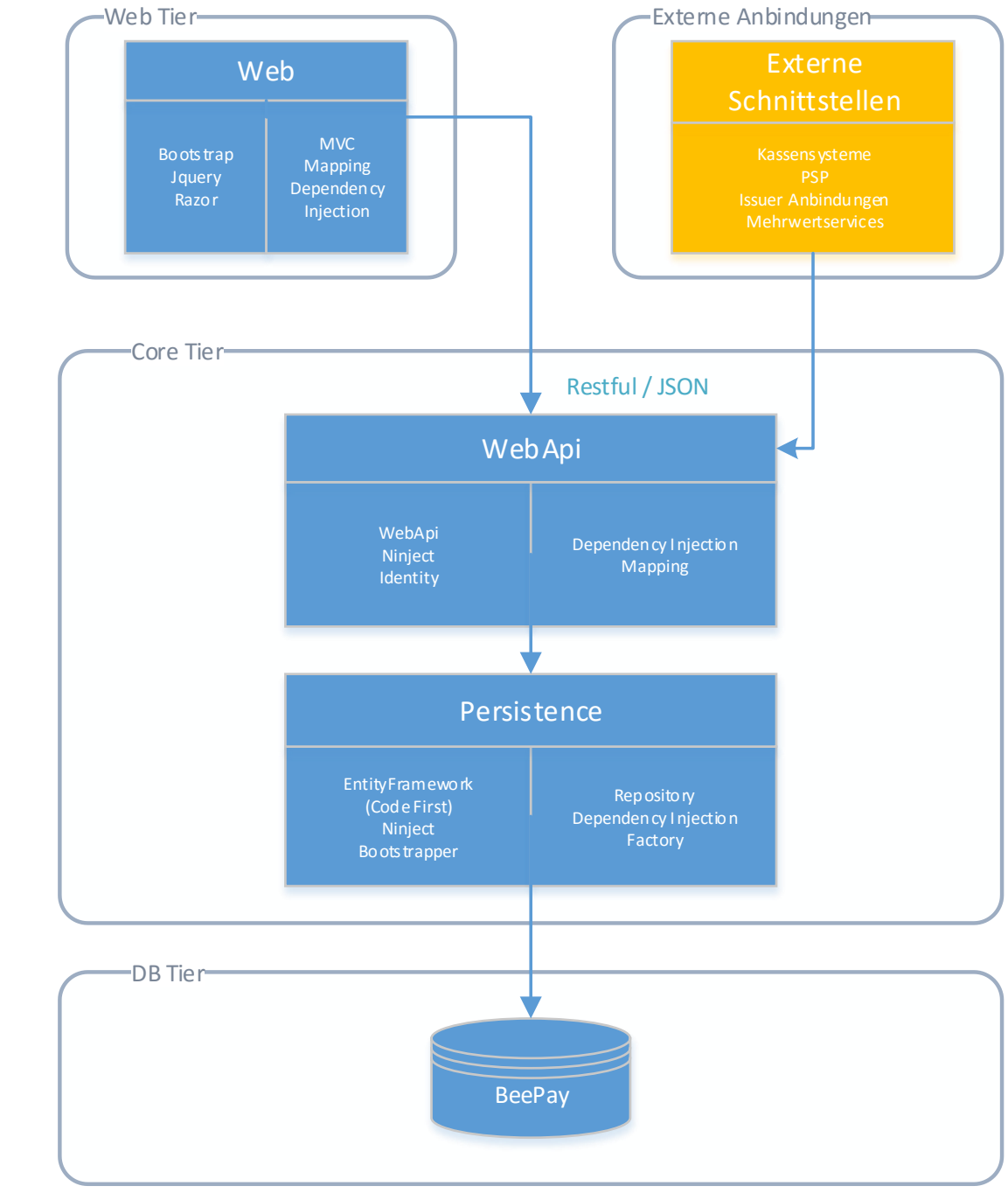

Abbildung 33 Systemübersicht

7.1.1 Anbindung an Datatrans

Datatrans ist ein Schweizer Unternehmen, welches sich auf E-Payment spezialisiert hat. BeePay verwendet die E-Payment Schnittstelle von Datatrans um Zahlungsmittel des Kunden zu erfassen sowie Zahlungen auszulösen.

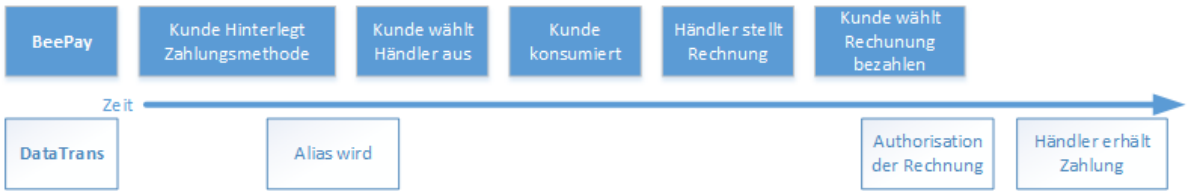

Abbildung 34 Datatrans Schnittstelle Verwendung

Beim Erstellen des **Alias** kommt der Hidden Mode[33] zum Tragen. Dieser erlaubt das Speichern eines Alias (Token) anstelle der Kreditkartennummer. Das Speichern der Kreditkartennummer ist durch den PCI Security Standard [3] untersagt. Der Alias darf jedoch im System gespeichert werden. Dabei wird der Alias bei einer Zahlung an Datatrans gesendet, welches den Alias dem Zahlungsmittel zuordnet und somit die Zahlung ausführt.

Wenn der Kunde eine Rechnung bezahlt, wird die XML Authorisations Schnittstelle[34] verwendet. Dabei wird über eine HTTPS Verbindung ein Autorisierungs-XML gesendet, mit welchem Datatrans die Zahlung auslöst.

7.1.2 User Management

Für das User Management wird ASP.Net Identity[35] als solide Grundlage verwendet. Wichtig ist dabei die Unterscheidung zwischen Kunde (Customer) und Händler (Merchant). Diese leiten sich vom allgemeinen BeePayUser ab. BeePayUser wird von ASP.Net Identity für die Autorisierung und Authentifizierung verwendet. Um auf einfache Weise zwischen Customer und Merchant zu unterscheiden, werden entsprechende Rollen zugewiesen.

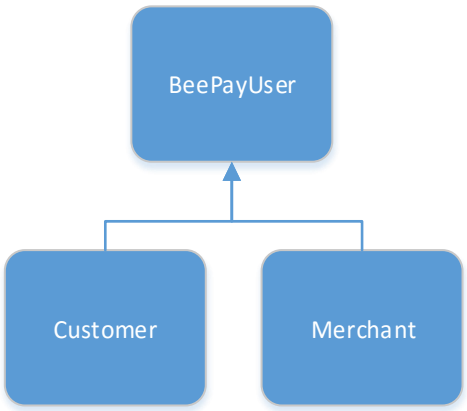

Abbildung 35 User Management Vererbung

[33] https://www.datatrans.ch/showcase/documentations/technical-documentation
[34] https://www.datatrans.ch/showcase/authorisation/xml-authorisation
[35] https://www.asp.net/identity

Das bestehende Authorize Attribut wurde durch eigene Custom Attribute erweitert, um den Zugriff weiter auf Customer oder Merchant einzuschränken. Somit kann jeder Controller (MVC + Web API), der geschützt werden soll, mit einem entsprechenden Attribut ergänzt werden.

7.1.3 API

Für das API wird ASP.Net Web API[36] verwendet, um eine einheitliche REST-Schnittstelle zu definieren. Der Zugriff erfolgt ausschliesslich via HTTP und dient als Single Point of Access für den Core Layer und die darunterliegende Datenbank. Alle Daten werden als JSON übermittelt.

7.1.4 Error Handling

Tritt beim Zugriff auf das WebAPI ein Fehler auf, wird dieser mittels eines HTTP Status Codes an den Aufrufer propagiert. Im Webprojekt werden Fehler global abgefangen und es wird aus Sicherheitsgründen eine allgemeine Fehlerseite ohne Details angezeigt.

7.1.5 Entity Framework

Das Entity Framework[37] wird als **Object-relational mapping (ORM)** für den Datenzugriff verwendet. Für die Erstellung der Datenbank wird der Code First [38] Ansatz verwendet. Dabei werden die Beziehungen zwischen den Tabellen via Annotationen[39] und dem Fluent API[40] definiert.

Das Entity Framework wird ohne den Funktionen des Lazy Loadings[41] und des Auto Detect Change[42] verwendet, da die Kommunikation zwischen dem Aufrufer und dem Entity Framework asynchron verläuft und die zwei Funktionen nur bei einer synchronen Kommunikation Sinn machen.

7.1.6 Repository für Abfragen zum Entity Framework

Beim Entity Framework wird ein Repository verwendet. Dabei wird bewusst auf ein generisches Create, Read, Update, Delete (CRUD) aller Tabellen verzichtet. Dies aus Sicherheitsgründen, da nicht jeder Benutzer Zugriff auf alle Tabellen haben soll und anderseits weil nur selten eine Tabelle alle CRUD Funktionen benötigt.

[36] https://www.asp.net/web-api
[37] https://msdn.microsoft.com/en-us/data/ef.aspx
[38] https://msdn.microsoft.com/en-us/data/jj193542.aspx
[39] https://msdn.microsoft.com/en-us/data/jj591583.aspx
[40] https://msdn.microsoft.com/en-us/data/jj591620.aspx
[41] https://msdn.microsoft.com/en-us/data/jj574232.aspx
[42] https://msdn.microsoft.com/en-us/data/jj556205.aspx

8 Glossar

Activity, Android Framework Klasse, die eine Aktivität des Benutzer behandelt, siehe Kapitel 4.5.1.1.

Acquirer, siehe Kapitel 6.1.3

Alias, siehe Kapitel 7.1.1

Always On, Standardmässig sind Azure Server, welche nicht benötigt werde, im Leerlauf. Die „Always On“ Funktion bewirkt, dass der Server immer online ist und nicht erst nach einem Aufruf gestartet wird.

Bearer Token, wird genutzt um auf eine geschützte Ressource eines Servers zuzugreifen.

BeePay, Name des entwickelten Prototyps. Wobei zwischen BeePay Plattform, BeePay Web und BeePay Mobilapplikation unterschieden wird.

BeePay Fastline, ist ein Fiktives Produkt, welches es Kunden mit BeePay ermöglicht, schneller einen Standort zu betreten indem der BeePay Identifikationsprozess verwendet wird.

BeePay Mobilapplikation, Mobilapplikation, iOS oder Android, welche während der Bachelorarbeit erstellt wurde.

BeePay Plattform, siehe Kapitel 7.1

BeePay Web, siehe Kapitel 7.1

Check-in, siehe Kapitel 4.2.3.1

Customer, siehe Kunde.

EFT/POS, manchmal auch EFTPOS geschrieben, steht für Electronic Funds Transfer at Point Of Sale und beschreibt somit das Terminal-Gerät, an dem ein Consumer mit seiner Bezahlkarte oder seinem Smartphone mit Mobile Payment Fähigkeit bezahlt [18].

Fragment, Android Framework Klasse, ein Fragment ist ein Teil einer Activity, siehe Kapitel 4.5.1.1.

Gästeliste, siehe Check-in

Hands-Free-Payment, Zahlungsmethode bei welcher der Kunde keine Kredit-/Debitkarte direkt benutzen muss, siehe 1.1.

Händler, auch Merchant genannt, bietet Waren/Dienstleistungen an, siehe Kapitel 6.1.4.

Identifizierung, siehe Kapitel 3.1 und 4.3

Interchange Fee, die Gebühr, die der Acquirer dem Issuer bezahlt, wenn der Karteninhaber eine Transaktion mit der Karte des Issuers durchführt.

Issuer, Herausgeber der Kredit-/Debitkarte, siehe Kapitel 6.1.1

Karteninhaber, Besitzer einer Karte, siehe Kapitel 6.1.1.

Kunde, auch Customer genannt, nutzt die Dienstleistung eines Händlers

Merchant, siehe Händler.

Merchant Service Charge, ist eine Gebühr die der Händler dem Acquirer bezahlt.

Millennials, auch Generation Y sind demographisch die Personen, welche zwischen 1980 und 2000 geboren wurden [19].

Object-relational mapping (ORM), Technik der Softwareentwicklung um die Relation zwischen Objekten und Tabellen herzustellen.

Pairing, siehe Kapitel 5.1.1.2

Prepaidkarte, Karte mit einem Wert. Dieser Wert muss im vorherein auf die Karte geladen werden, zum Beispiel Geschenkkarten.

Payment Service Provider (PSP), Anbieter eines virtuellen Terminals.

REST, Representational State Transfer, bezeichnet ein Programmierparadigma für verteilte Systeme

Wireless local area network (WLAN), drahtloses Netzwerk nach Standard IEEE-802.11.

WLAN-SSID, Name des Netzwerkes [7].

9 Abbildungsverzeichnis

10 Literaturverzeichnis

[1] B. Kühnis und M. Schaub, „Marktanalyse und Entwicklung eines Bezahlsystems mit Mehrwertleistungen,“ 2015.

[2] J. Müller, „Apple bleibt Platzhirsch,“ Nr. http://www.nzz.ch/wirtschaft/unternehmen/apple-bleibt-platzhirsch-1.18563593, 2015.

[3] pcisecuritystandards, „PCI Security Standard,“ [Online]. Available: https://de.pcisecuritystandards.org/minisite/en/. [Zugriff am 23 Mai 2015].

[4] G. G. P. K. A. P. A. T. Kostas Fouskas, „On the Potential Use of Mobile Positioning Technologies in Indoor Environments,“ in *AIS Electronic Library (AISeL)*, 2002, pp. 417-420.

[5] A. L. S. M. I. a. I. N. Yanying Gu, „A Survey of Indoor Positioning Systems for Wireless Personal Networks,“ 2009.

[6] „mathwork wlan,“ [Online]. Available: http://ch.mathworks.com/help/comm/examples/ieee-802-11-wlan-beacon-frame.html#commwlan80211Beacon-10. [Zugriff am 05 Oktober 2015].

[7] „wlan rfc 5416,“ [Online]. Available: https://tools.ietf.org/html/rfc5416. [Zugriff am 04 Oktober 2015].

[8] „Beacon,“ Apple, [Online]. Available: https://developer.apple.com/ibeacon/Getting-Started-with-iBeacon.pdf. [Zugriff am 04 Oktober 2015].

[9] „https://support.kontakt.io/hc/en-gb/articles/201620741-iBeacon-Parameters-UUID-Major-and-Minor,“ [Online]. [Zugriff am 04 Oktober 2015].

[10] „Beacon Transition,“ 2015. [Online]. Available: https://github.com/AltBeacon/android-beacon-library/blob/master/src/main/java/org/altbeacon/beacon/BeaconTransmitter.java.

[11] S. H. Y. Hakan Koyuncu, „A Survey of Indoor Positioning and Object Locating Systems,“ 2010.

[12] S. N. C. a. Y. L. L. Jing Hang Choo, Design and Development of NFC Smartphone Indoor Interactive Navigation System, 2014.

[13] „Azure Web App Config,“ [Online]. Available: https://azure.microsoft.com/en-us/documentation/articles/web-sites-configure. [Zugriff am 23 November 2015].

[14] Apple, „App Programming Guide for iOS,“ 2015.

[15] „Swift and Objective-C in the same Project,“ [Online]. Available: https://developer.apple.com/library/ios/documentation/Swift/Conceptual/BuildingCocoaApps/MixandMatch.html. [Zugriff am 23 November 2015].

[16] [Online]. Available: http://appleinsider.com/articles/14/09/16/apple-to-limit-iphone-6-nfc-to-apple-pay-restricts-developer-access. [Zugriff am 24 November 2015].

[17] vez-epay, „Verband Elektronischer Zahlungsverkehr (VEZ) Vier-Parteien-System,“ [Online]. Available: http://www.vez-epay.ch/de/bargeldloses-zahlen.html. [Zugriff am 12 März 2015].

[18] S. V. Christian Mäder, „HSR ePrints,“ Machbarkeitsstudie einer Smartphone-App für EMV-kompatible Zahlungen via NFC, 2013. [Online]. Available: http://eprints.hsr.ch/309/. [Zugriff am 03 März 2015].

[19] H. Perennial, „Generations: The History of America's Future, 1584 to 2069,“ 1991, p. 335.

[20] S. Graf und B. Gehring, „Hat Bargeld als Zahlungsmittel bald ausgedient?,“ *Das Magazin für Innovation,* p. http://zhaw.inx.ch/images/SML_Newsletter/02_2013/ZHAW_SML_Competence_Nr_5_2013_Bargeld.pdf, Dezember 2013.

[21] R. H. J. V. Erich Gamma, „Design Patterns:Elements of Reusable Object-Oriented Software,“ 1995.