DHBW
Duale Hochschule
Baden-Württemberg
Mannheim

Duale Hochschule Baden-Württemberg
Mannheim

Zweite Projektarbeit

Prototypenentwicklung einer mobilen Applikation zum Self-Scanning in den Filialen

Studiengang Wirtschaftsinformatik

Studienrichtung Software Engineering

Sperrvermerk

Verfasser:	Mike Mülhaupt
Matrikelnummer:	1366418
Firma:	ALDI Einkauf GmbH & Co. oHG
Abteilung:	IIT Stores
Kurs:	WWI 11 SE B
Studiengangsleiter:	Prof. Dr.-Ing. Jörg Baumgart
Wissenschaftlicher Betreuer:	Prof. Dr.-Ing. Jörg Baumgart joerg.baumgart@dhbw-mannheim.de +49 (0)624 4105 - 1216
Firmenbetreuerin:	Ulrike Gasch ulrike.gasch@aldi-sued.com +49 (0)208 9927 - 1818
Bearbeitungszeitraum:	29. Juli 2013 bis 11. November 2013

Sperrvermerk

Diese Projektarbeit enthält vertrauliche Daten der *ALDI Einkauf GmbH & Co. oHG*. Eine Veröffentlichung oder Vervielfältigung dieser Arbeit, auch auszugsweise, ist ohne ausdrückliche Genehmigung der *ALDI Einkauf GmbH & Co. oHG* nicht zulässig. Diese Arbeit darf nur den Korrektoren, der Studiengangsleitung und dem Prüfungsausschuss zugänglich gemacht werden.

Kurzfassung

Verfasser: Mike Mülhaupt

Kurs: WWI 11 SE B

Firma: ALDI Einkauf GmbH & Co. oHG

Thema: Prototypenentwicklung einer mobilen Applikation zum Self-Scanning in den Filialen

Inhaltsverzeichnis

Verzeichnisse

Abbildungsverzeichnis

Tabellenverzeichnis

Listingverzeichnis

1 Einführung

Waren, die ein Kunde im Supermarkt einkaufen möchte, werden drei mal in die Hand genommen, bevor sie mit dem Kunden den Markt verlassen: zuerst um sie aus dem Regal in den Einkaufswagen zu legen, anschließend um sie aufs Kassenband zu legen und ein letztes Mal vom Kassentisch zurück in den Einkaufswagen. Die letzten beiden Arbeitsschritte (das Auflegen und erneute Einpacken an der Kasse) könnte der Kunde sich jedoch ersparen, wenn er den Inhalt seines Einkaufswagens selbstständig erfassen würde – beispielsweise mithilfe eines sog. „Self-Scanning“ Systems.

In einer ersten Projektarbeit wurden solche „Self-Scanning“ Systeme miteinander verglichen[2] und ein speziell für das Unternehmen ALDI SÜD geeignetes Konzept erarbeitet.[3] Das grundlegende Prinzip besteht darin, dass der Kunde die Artikel mithilfe seines Smartphones erfasst, während er sie dem Regal entnimmt. Die eingescannten Artikel werden zur Bezahlung dann an das Kassensystem übermittelt.

Die tatsächliche Erleichterung, die dem Kunden hierdurch entsteht, lässt sich anhand eines theoretischen Konzepts leider nur schwer nachvollziehen. Aus diesem Grund, soll das in der ersten Projektarbeit erarbeitete Konzept zum „mobilen Self-Scanning in den Filialen“ als mobile Applikation entworfen und implementiert werden.

Hierfür sollen zunächst nochmal die Grundlagen erläutert und anhand dessen die konkreten Anforderungen ermittelt werden. Kapitel 3 beschäftigt sich anschließend mit der Auswahl einer geeigneten Vorgehensweise zur Entwicklung mobiler Anwendungen, um daraufhin den Entwurf der Self-Scanning App im vierten Kapitel zu erarbeiten. Abschließend werden die Ergebnisse der Implementierung und die daraus gewonnenen Erkenntnisse vorgestellt.

[2] **?**, Vgl..
[3] **?**, Vgl..

2 Fachliches Konzept und Anforderungen an das System

2.1 Kapiteleinleitung

Zur Ermittlung der konkreten Anforderungen an eine mobile Self-Scanning Anwendung, soll zunächst der prinzipielle Ablauf, sowie die einzelnen Teilprozesse näher erläutert werden. Anschließend sollen fachliche und technische Anforderungen identifiziert und indiziert werden, sodass deren Erfüllung am Ende der Arbeit überprüft werden kann.

2.2 Konzept des mobilen Self-Scannings

2.2.1 Prinzipieller Ablauf und Abgrenzung zum stationären Self-Scanning

Abbildung 1 auf S. 3 zeigt einen vollständigen Einkaufsprozess mithilfe einer mobilen Anwendung aus Sicht des Kunden. Die Abbildung ist angelehnt an das in der ersten Projektarbeit erarbeitete Konzept, bei dem der Kunde seinen Einkauf mithilfe seines Smartphones selbstständig Erfassen und an einer herkömmlichen, bemannten Kasse bezahlen kann.[4]

Hiervon ausdrücklich abzugrenzen ist das sog. „stationäre Self-Scanning System“, bei dem der Kunde seine Artikel am Ende des Einkaufs an einer Selbstbedienungsstation einscannen und ggf. auch bezahlen kann.[5]

[4] **?**, Vgl..
[5] **?**, Vgl..

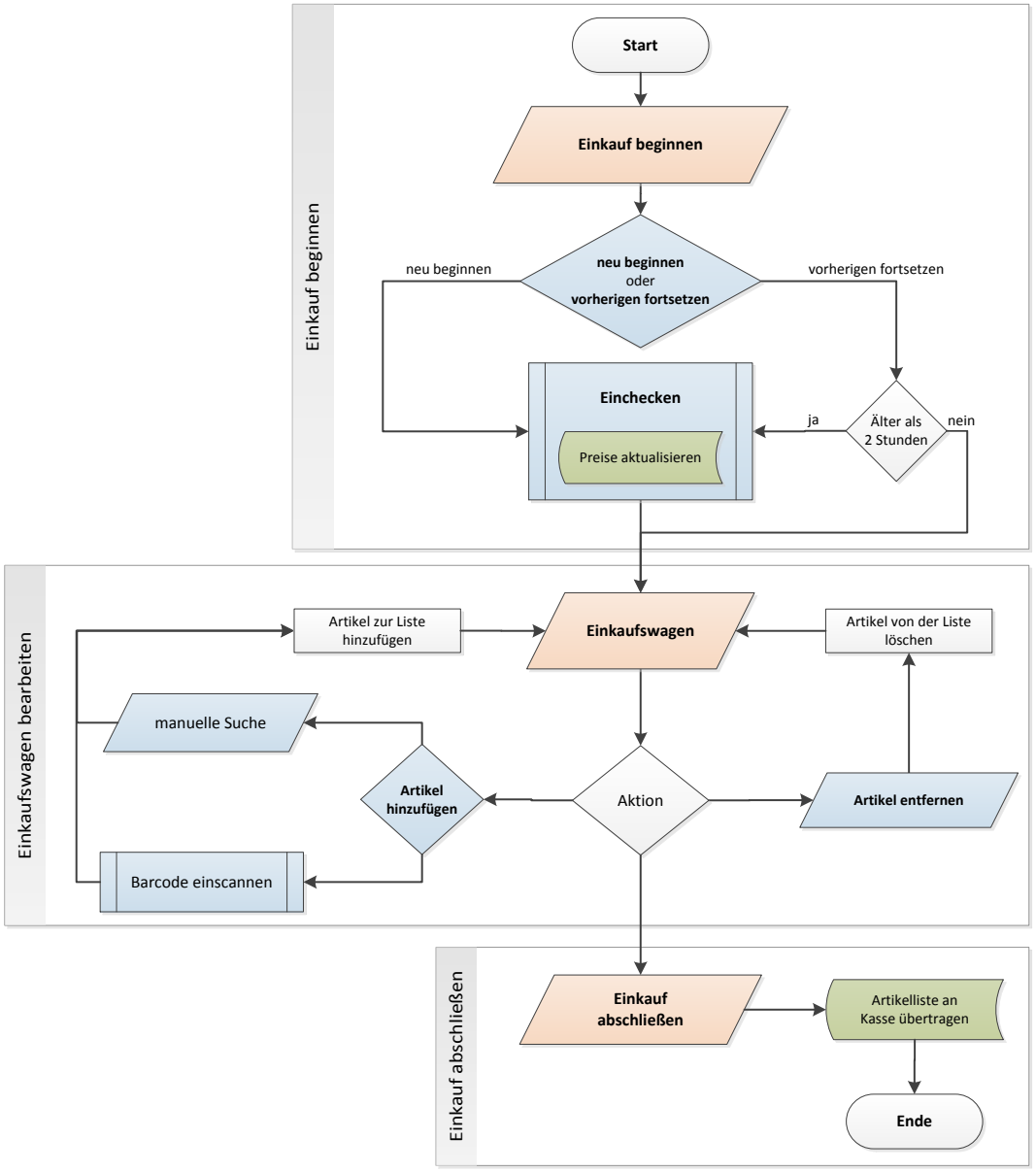

Abbildung 1: Der Einkaufsprozess mithilfe einer Self-Scanning Anwendung als Flussdiagramm

Der Einkaufsprozess gliedert sich in drei Teilprozesse, die in der Grafik durch ein Rechteck gruppiert werden: Einkauf beginnen, Einkaufswagen bearbeiten und Einkauf abschließen. Benutzerdialoge sind gekennzeichnet durch ein rot hinterlegtes Trapez. Ein blaues Element beschreibt eine Benutzeraktion, ein graues steht dage-

gen für eine Funktion der Anwendung. Letztlich sind noch die zwei Schnittstellen (grün) zu erwähnen: Preise aktualisieren und Artikelliste an Kasse übertragen.

Die drei Teilprozesse Einkauf beginnen, Einkaufswagen bearbeiten und Einkauf abschließen werden in den folgenden Unterkapiteln näher erläutert.

2.2.2 Einzelne Teilprozesse im Detail

Einkauf beginnen

Im ersten Schritt soll der Kunde zunächst entscheiden, ob er einen neuen Einkauf beginnen oder einen vorherigen fortsetzen möchte. Da sich einzelne Artikelpreise von Filiale zu Filiale unterscheiden können, ist die Information über den aktuellen Standort des Kunden eine zentrale Voraussetzung für den Einkauf mit dem Smartphone. Hierfür wird vor jedem neuen Einkauf ein Zwischenschritt eingeführt, der im Folgenden „Einchecken“ genannt wird.

Beim Einchecken greift die Anwendung auf eine Schnittstelle der Filiale zu, um alle für den Einkauf notwendigen Informationen auf das Smartphone zu übertragen. Die konkreten Anforderungen, die sich an diese Schnittstelle ergeben, werden in Kapitel 2.3.2 auf S. 7 beschrieben.

Für den Fall, dass der Nutzer während eines Einkaufs unterbrochen wird (z. B. durch einen eingehenden Anruf), hat er die Möglichkeit, einen bereits angefangenen Einkauf fortzuführen. Daran knüpft sich die Bedingung, dass das Erstellungsdatum des Einkaufs nicht länger als 2 Stunden in der Vergangenheit liegt. Ansonsten ist eine Aktualität der Preise nicht mehr gewährleistet. Der Nutzer müsste in diesem Fall erneut einchecken.

Einkaufswagen bearbeiten

Nachdem ein neuer Einkauf eröffnet bzw. ein unterbrochener fortgesetzt wurde, befindet sich der Kunde im „Einkaufswagen“-Dialog. Hier werden alle bisher erfassten Artikel in einer Liste aufgeführt. Neue Artikel können der Liste hinzugefügt oder bestehende von der Liste entfernt werden.

Das Hinzufügen von Artikeln kann auf zwei Weisen erfolgen: entweder per Einscan-

nen des Barcodes auf der Artikelverpackung oder per manueller Suche, für den Fall, dass Verpackung oder Barcode nicht vorhanden sind. Die Suche soll dann anhand Artikelbezeichnung oder -nummer ermöglicht werden.

Einkauf abschließen

Sobald sich alle Artikel im Einkaufswagen befinden, kann der Kunde seinen Einkauf an einer der herkömmlichen Kassen bezahlen. Hier greift die Anwendung auf eine zweite Schnittstelle zu, um die vom Kunden eingescannten Artikel an das Kassensystem zu übertragen. Die konkreten Anforderungen, die sich an diese Schnittstelle ergeben, werden in Kapitel 2.3.2 auf S. 7 beschrieben.

2.3 Anforderungsermittlung

2.3.1 Fachliche Anforderungen

Anhand des in der ersten Projektarbeit erarbeiteten und im vorherigen Kapitel zusammengefassten Lösungskonzepts sollen nun fachliche und technische Anforderungen definiert und nummeriert werden.

F01 Der Kunde soll seinen Einkauf mithilfe einer mobilen Anwendung durchführen können. Der gesamte Prozess lässt sich in die drei folgenden Bestandteile gliedern, deren detaillierte Beschreibung in Kapitel 2.2.2 zu finden ist:

- a *Einkauf beginnen*
- b *Einkaufswagen bearbeiten*
- c *Einkauf abschließen*

F02 An die Stammdaten, d. h. die Artikel- und Preisinformationen, die der Anwendung zur Verfügung stehen, ergeben sich folgenden Unteranforderungen:

- a *Verfügbarkeit der Daten*
  Während des Einkaufs soll weder eine Verbindung zum Internet noch eine Verbindung zum Kassensystem oder dem Filialnetzwerk erforderlich sein.
- b *Vollständigkeit der Artikel*
  Jeder Artikel der in der Filiale erworben werden kann, soll mithilfe des Smartphones erfasst werden können. Dazu gehören ausdrücklich auch

unverpackte bzw. lose Waren (z. B. aus der O&G-Abteilung oder dem Backautomaten). Darüber hinaus sollen auch sog. Gewichtsartikel (Artikel deren Preis abhängig vom Gewicht ist) sowie Pfandbons dem virtuellen Einkaufswagen hinzugefügt werden können.

c *Abweichungen einzelner Filialen*
Auch „lokale Preisveränderungen“ sollen berücksichtigt werden. Darunter werden Preisabweichungen verstanden, die lediglich innerhalb einer Filiale für einen begrenzten Zeitraum gültig sind (z. B. beim Abverkauf von O&G kurz vor Feierabend), .

d *Aktualität der Preise*
Die Preisinformationen sollen stets mit den aktuell gültigen Preisen der Filiale übereinstimmen. Toleriert werden lediglich Preisänderungen, die nach dem Einchecken vorgenommen werden.

F03 Die im Dialog Einkaufswagen anzuzeigenden Informationen werden in der Tabelle 1 in der Übersicht dargestellt:

Aktuelle Filiale des Kunden				
Menge	Artikelbezeichnung			Gesamtpreis
	EAN	Einzelpreis	MwSt.-Satz	
weitere Artikelpositionen				
Zwischensumme der Steuersätze				Summe

Tabelle 1: Tabellarische Darstellung der Inhalte im Dialog: Einkaufswagen

F04 In einigen Fällen führt das Erfassen von Artikeln zu Ausnahmen. Diese werden im Folgenden erläutert:

a *Preisanzeige von Gewichtsartikeln*
In der Filiale befinden sich keine für den Kunden zugänglichen Waagen. Der Preis eines Gewichtsartikels soll deshalb lediglich in Bezug auf eine Grundeinheit angezeigt werden (z. B. 1,99 €/kg). Zusätzlich soll ein Hinweis erscheinen, dass der Artikel an der Kasse noch gewogen werden muss.

b *Altersbeschränkung bei Artikeln*
Beim Erfassen von Artikeln mit Altersbeschränkung (z. B. Spirituosen) soll ein Hinweis erscheinen, dass die Volljährigkeit des Kunden an der Kasse zunächst kontrolliert werden muss.

c *Abweichung vom tatsächlich zu zahlenden Betrag*
Die Bonsumme, die der Kunde auf dem Smartphone angezeigt bekommt, könnte von der Summe abweichen, die er tatsächlich zu zahlen hat (z. B. bei einer Preisveränderung während des Einkaufs). Dies sollte vom Kassensystem überprüft und den Kunden ggf. darauf hinweisen.

2.3.2 Technische Anforderungen

T01 Beim Einchecken greift die Anwendung auf eine Schnittstelle zu, mit dessen Hilfe die folgenden Informationen abgerufen werden sollen:

a *Ermittlung des Standorts*
Aufgrund der fachlichen Anforderung *F02c Abweichung einzelner Filialen* muss vor jedem Einkauf die aktuelle Filiale des Kunden ermittelt werden. Die entsprechenden Standortdaten sollen beim Einchecken an das Smartphone übermittelt werden.

b *Aktualisierung der lokalen Preise*
Darüber hinaus fordert die Anforderung *F02d Aktualität der Preise*, dass die in der Anwendung anzuzeigenden Preise mit denen der Filiale stets übereinstimmen sollen. Um dies sicherzustellen sollen auch aktuelle Preisveränderungen der Filiale von der Schnittstelle bereitgestellt werden.

T02 Bei der Bezahlung erfolgt ein weiterer Informationsaustausch mit dem Kassensystem: die erfassten Artikel müssen an die Kasse übertragen werden, um den Einkauf anschließend bezahlen zu können. Von der Kasse werden folgende Informationen benötigt:

a *Standortdaten*
Zur erneuten Prüfung, ob der Standort des Kunden mit der tatsächlichen Filiale übereinstimmt, sollen die beim Einchecken empfangenen Standortdaten auch der Kasse bereitgestellt werden.

b *Details zu Artikelpositionen*
Für jeden Artikel im Einkaufswagen soll lediglich Artikelnummer und die zugehörige Menge zur Kasse übertragen werden.

c *Gesamtsumme des Einkaufswagens*
Um zu prüfen, ob die in der Anwendung berechnete Summe mit dem tatsächlich zu zahlenden Betrag übereinstimmt (siehe *F04c Abweichung vom tatsächlich zu zahlenden Betrag*), soll auch die berechnete Summe an die Kasse übertragen werden.

T03 An die Vorgehensweise zur Entwicklung mobiler Applikationen ergeben sich

folgende technische Anforderungen:

a *Kamerazugriff* zum Erfassen von Barcodes

b *Datenspeicher* zur permanenten Ablage von Artikel- und Preisdaten.

c *Zugriff auf Internetverbindung*, zur Aktualisierung der Daten.

d *Kurzfristige Entwicklungsergebnisse*, aufgrund der zeitlichen Begrenzung des Projekts, sowie des prototypischen Charakters. Das Hauptaugenmerk soll weniger auf Performance oder Usability liegen, sondern vielmehr auf einer kurzen Entwicklungszeit und raschem Erkenntnisgewinn (steile Lernkurve).

2.4 Kapitelzusammenfassung

Dem Kunden soll die Möglichkeit geboten werden, alle Artikel bereits während des Einkaufs in einer Filiale mithilfe seines Smartphones zu erfassen und in einer Liste zu speichern. Das Erfassen wird durch das Einscannen eines Barcodes auf der Artikelverpackung oder alternativ durch manuelles Suchen in einer Artikeldatenbank ermöglicht. Die Liste aller eingescannten Artikel wird bei der Bezahlung an das Kassensystem übertragen, sodass der Einkauf an einer herkömmlichen Kasse bezahlt werden kann. Das Auflegen der Artikel aufs Kassenband, anschließende Einscannen und erneute Einpacken an der Kasse entfällt somit.

3 Vorgehensweisen zur Entwicklung mobiler Anwendungen

3.1 Kapiteleinleitung

Ausgehend von den in Kapitel 2.3 definierten Anforderungen, kann die mobile Applikation entworfen und implementiert werden. Zunächst sollen jedoch grundsätzliche Vorgehensweisen zur Entwicklung mobiler Applikationen vorgestellt und anhand der Anforderung an die Entwicklungsmethode (T03) bewertet werden, um abschließend eine für das Projekt passende Methode auszuwählen.

3.2 Beschreibung der einzelnen Vorgehensweisen

3.2.1 Entwicklung einer nativen App

Als native Apps werden eigenständige Anwendungen bezeichnet, die speziell für ein konkretes Betriebssystem implementiert werden und letztendlich nur auf diesem System installations- und lauffähig sind. Auf diese Weise können die Systemressourcen optimal ausgereizt und mithilfe entsprechender Grundkonzepte des jeweiligen Betriebssystems (Multithreading, Speicherzugriff, Grafikrendering) eine hohe Performance erzielt werden. Dieser Aspekt macht sich besonders bei komplexen Anwendungen bemerkbar. Darüber hinaus ermöglichen native Apps den Zugriff auf Gerätekomponenten, wie z.B. Kamera, GPS-Sensor oder NFC-Chip.[6]

Neben den eben genannten Vorteilen auf technischer Seite, gibt es noch einen weiteren, marketingtechnischen Aspekt: native Apps werden über den offiziellen App-Store des jeweiligen Herstellers vertrieben. Potenziellen Nutzer können die App in

[6] **?**, vgl..

gewohnter Art und Weise finden, Bewertungen lesen und die App gegebenenfalls direkt auf dem Smartphone installieren. Nach der Installation befindet sich eine Verknüpfung auf dem Homescreen des Nutzers, sodass die App jederzeit zur Verfügung steht.

Dieser Aspekt gilt jedoch gleichzeitig auch als Nachteil zu erwähnen: vor der Veröffentlichung im App-Store von Apple zunächst ein aufwendiger Freigabeprozess durchlaufen werden. Die App kann jederzeit vom Store-Betreiber abgelehnt oder nachträglich aus dem Store entfernt werden. Außerdem ist für die offizielle Entwicklung einer nativen App im Falle von Apple eine kostenpflichtige Lizenz sowie ein Mac Voraussetzung.

Hinzu kommt die aufwendige Einarbeitung für unerfahrene Entwickler. Selbst bei guter Kenntnis der zu implementierenden Sprache ist eine sorgfältige Einarbeitung in das jeweilige SDK unverzichtbar.[7]

Tabelle 2 zeigt die eben genannten Vor- und Nachteile in der Übersicht.

	Vorteile	**Nachteile**	
Hohe Performance, auch bei komplexen Anwendungen	(+)	(−)	Aufwendiger Freigabeprozess
Einfacher und umfangreicher Zugriff auf Gerätekomponenten	(+)	(−)	Ggf. kostenpflichtige Entwicklerlizenzen erforderlich
Effiziente Vermarktung über den App-Store	(+)	(−)	Äußerst zeitintensive Einarbeitung notwendig
Schneller Zugriff durch Verknüpfung auf dem Homescreen	(+)		

Tabelle 2: Vor- und Nachteile bei der nativen App-Entwicklung

3.2.2 Entwicklung einer Web-App

Eine Webanwendung wird im Gegensatz zur nativen App nicht eigenständig, sondern mithilfe eines Browsers auf dem Zielsystem ausgeführt, eine Installation ist nicht notwendig. Sie kann direkt durch manuelles Eingeben der URL oder Aufrufen eines Lesezeichens geöffnet werden. Im Regelfall wird keine Verknüpfung auf

[7] **?**, vgl..

dem Homescreen erstellt. Eine Webanwendung unterscheidet sich nur geringfügig von einer herkömmlichen Webseite. Die größten Unterschiede liegen in der für mobile Geräte optimierten Bedienoberfläche und dem reduzierten Funktions- und/oder Informationsumfang.[8]

Der limitierende Faktor einer Web-App ist der jeweilige Browser, in dem die App ausgeführt wird. Funktionen, die innerhalb des Browsers nicht zur Verfügung stehen, können von der App nicht verwendet werden. So gibt es beispielsweise keine Möglichkeit, den Nutzer einer Web-App per Push-Notification zu kontaktieren.

Dank moderner Webtechnologien wie HTML5 stehen allerdings zahlreiche Möglichkeiten für den Zugriff auf Systemkomponenten zur Verfügung. Der Anwender muss diesen Zugriff leider jedes Mal explizit gewähren. Darüber hinaus ermöglicht der sog. DOM-Storage eine permanente lokale Datenspeicherung bis zu 5 bzw. 10MB – je nach verwendetem Browser. Die zusätzliche Integration zahlreich vorhandener JavaScript-Frameworks ermöglicht dem Entwickler auch komplexe Applikationen schnell und einfach zu entwickeln.

Eine Webanwendung muss lediglich einmal implementiert werden und ist dank der einheitlichen Webstandards auf allen Betriebssystemen lauffähig. Zur Veröffentlichung ist weder ein Freigabeprozess noch eine Lizenz notwendig. Es genügt ein Webserver auf dem die Anwendung läuft.

Tabelle 3 auf S. 12 zeigt die eben genannten Vor- und Nachteile in der Übersicht.

[8] **?**, vgl..

	Vorteile		Nachteile
Keine Installation notwendig	(+)	(–)	Aufwendiger Aufruf über den Browser, kein direkter Zugriff über Verknüpfung auf dem Homescreen
Auf verschiedenen Betriebssystemen lauffähig	(+)	(–)	Limitierter Funktionsumfang, je nach verwendetem Browser
Zugriff auf Gerätekomponenten dank HTML5 möglich	(+)	(–)	Zugriff auf Gerätekomponenten muss jedes Mal gewährt werden
Komplexe Anwendungen durch Verwendung zusätzlicher Frameworks möglich	(+)		
Kein Freigabeprozess oder Lizenz notwendig	(+)		

Tabelle 3: Vor- und Nachteile bei der Entwicklung einer WebApp

3.2.3 Entwicklung einer hybriden App

Hybride Apps vereinen die Vorteile beider Entwicklungsalternativen: sie werden zunächst als WebApp implementiert und anschließend in eine native App eingebettet. Die native App besitzt einen sog. InApp-Browser, in dem die ursprünglich entwickelte Webanwendung gerendert wird.

Auf diese Weise können mobile Anwendungen mithilfe der bekannten Webtechnologien entwickelt werden, sind auf verschiedenen Betriebssystemen lauffähig und bieten dank der umgebenden nativen App trotzdem den vollen Funktionsumfang (d. h. hybride Apps ermöglichen bspw. auch die Verwendung von Push-Notifications).

Für den Nutzer sind hybride Apps von nativen kaum zu unterscheiden, da sie wie gewohnt über den App-Store heruntergeladen, installiert und über eine Verknüpfung auf dem Homescreen aufgerufen werden. Lediglich in der Benutzeroberfläche lässt sich ein Unterschied erkennen, da für das Rendering nicht die nativen Bedienelemente, sondern lediglich HTML5 und CSS3 Komponenten verwendet werden.

Tabelle 4 auf S. 13 zeigt die eben genannten Vor- und Nachteile in der Übersicht.

	Vorteile	Nachteile
Einfache Implementierung als Webanwendung	(+)	(-) Keine nativen Bedienelemente
Lauffähig auf verschiedenen Betriebssystemen	(+)	
Funktionsumfang einer nativen App	(+)	
Vermarktung, Installation und Aufruf wie bei einer nativen App	(+)	

Tabelle 4: Vor- und Nachteile bei der hybriden App-Entwicklung

3.3 Bewertung und Auswahl einer Vorgehensweise

Der Vergleich der drei Herangehensweisen verdeutlichte, dass jede Entwicklungsmethode äußerst unterschiedliche Vor- und Nachteile mit sich bringt. Inwiefern die technischen Anforderungen an die Entwicklungsmethode aus Kapitel 2.3.2 erfüllt werden, soll Tabelle 5 zeigen.

ID	Bezeichnung	nativ	web	hybrid
T03a	Kamerazugriff	(+)	(-)	(+)
T03b	Datenspeicher	(+)	(-)	(+)
T03c	Internetzugriff	(+)	(+)	(+)
T03d	Entwicklungszeit	(-)	(+)	(+)

Tabelle 5: Erfüllung der an die Entwicklungsmethode gestellten Anforderungen

Aufgrund der Tatsache, dass der Zugriff auf Kamera und Datenspeicher bei Verwendung einer WebApp jedes Mal gewährt werden muss, wird diese Vorgehensweise nicht weiter betrachtet.

Die Implementierung als native App wäre aufgrund der hohen Menge an Artikel- und Preisdaten und der damit einhergehenden Rechenlast sicherlich die richtige Vorgehensweise. Im Rahmen dieses Projektes liegt der Fokus jedoch nicht auf Perfor-

mance, sondern auf einer kurzen Entwicklungszeit. Vor diesem Hintergrund soll die Self-Scanning Anwendung als hybride App entworfen und implementiert werden.

3.4 Kapitelzusammenfassung

Die drei unterschiedlichen Vorgehensweisen zur App-Entwicklung wurden zunächst vorgestellt und deren Vor- und Nachteile herausgearbeitet. Die tabellarische Übersicht am Ende jedes Kapitels verdeutlichte die wesentlichen Punkte:

- native Apps erzielen selbst bei komplexen Anwendungen eine hohe Performance, der nötige Entwicklungsaufwand ist jedoch für unerfahrene Entwickler besonders hoch.

- Webanwendung lassen sich dagegen mithilfe bekannter Webtechnologien entwickeln und ohne weitere Umstände veröffentlichen. Der vom Nutzer verwendete Browser limitiert den Funktionsumfang jedoch.

- hybride Apps vereinen die Vorteile beider Alternativen: äußerst einfache Entwicklung dank bekannter Webtechnologien und gleichzeitig den vollen Funktionsumfang dank der umgebenden nativen App.

Die Entwicklung einer reinen Web-App wurde aufgrund des eingeschränkten Zugriffs auf Gerätekomponenten ausgeschlossen. Der entscheidende Vorteil einer hybriden App gegenüber einer nativen App liegt in der sehr viel kürzeren Entwicklungszeit, weshalb die Self-Scanning Anwendung im folgenden Kapitel als hybride Applikation entworfen werden soll.

4 Entwurf der Self-Scanning App

4.1 Kapiteleinleitung

4.2 Die Architektur der Anwendung

4.2.1 Grundlegende Struktur einer PhoneGap-Anwendung

PhoneGap ermöglicht dem Entwickler, webbasierte Anwendungen in eine native App zu packen. Gleichzeitig wird der Webanwendung eine Schnittstelle zur nativen App bereitgestellt, sodass die Anwendung im Inneren auf die nativen Geräte- und Systemkomponenten zugreifen kann. Abbildung 2 veranschaulicht diesen sog. Packaging-Prozess und den daraus resultierenden Aufbau einer PhoneGap-Anwendung.

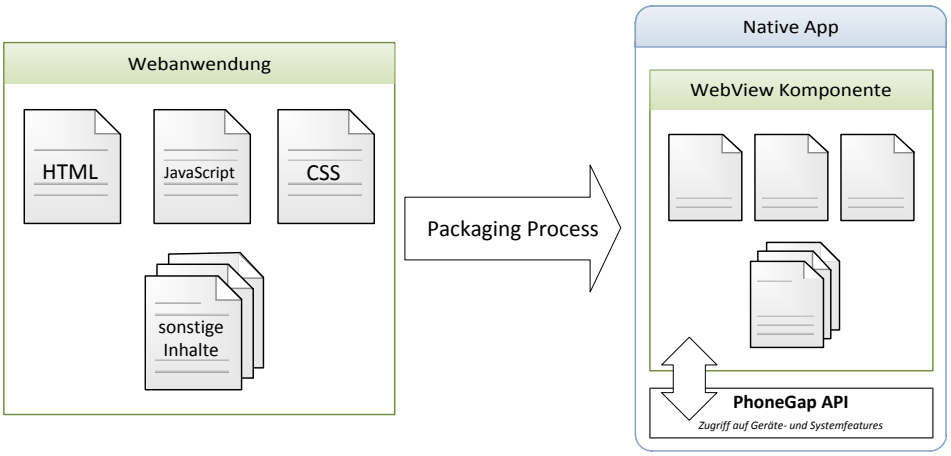

Abbildung 2: Packaging-Prozess von PhoneGap und Aufbau einer hybriden App[9]

[9] in Anlehnung an **?**,

Die konkrete Gestaltung von Datenhaltung, Programmlogik und Benutzeroberfläche, wird von PhoneGap jedoch nicht vorgegeben. Hierfür soll Sencha Touch zum Einsatz kommen – ein JavaScript Framework, welches einerseits eine möglichst nativ wirkende Benutzeroberfläche schafft und andererseits ein Grundgerüst für die Architektur der Webanwendung schafft. Ein wichtiger Bestandteil dieses Frameworks ist das MVC-Konzept, welches im folgenden Unterkapitel näher vorgestellt werden soll.

4.2.2 Prinzipien und Bestandteile des MVC-Konzepts

Das MVC-Konzept entspringt der Idee des sog. Beobachter-Musters. Diesem Muster zufolge wird zwischen Objekten unterschieden, welche die Rolle eines Beobachters einnehmen und solchen, die beobachtet werden. Beobachtete Objekte kennen ihre Beobachter, sodass letztere benachrichtigt werden können, sobald sich der Zustand eines beobachteten Objekts ändert. Die Beobachter-Objekte können außerdem den aktuellen Zustand der beobachteten Objekte jederzeit erfragen. Abbildung verdeutlicht dieses Konzept.

Darüber hinaus wird die strikte Trennung von Datenhaltung, Darstellung und Steuerung verfolgt. Diese drei Bestandteile werden als Model, View und Controller bezeichnet. Durch diese Aufteilung wird das System sehr leicht skalierbar und bleibt dennoch flexibel. Es können beispielsweise weitere Views unabhängig von Model oder Controller hinzugefügt, bestehende Views ausgetauscht oder entfernt werden.

Die drei zentralen Begriffe Model, View und Controller sollen im Folgenden etwas näher erläutert werden, um daraufhin den Bezug zum Beobachter-Muster herzustellen.

Das Model enthält fachlich strukturierte Informationen [Effekt. Software Arch., S. 247] und verwaltet in seinen Instanzen jeweils einen konkreten Zustand. Außerdem kann eine Model-Instanz Informationen zu seinem Zustand liefern oder diesen bei Bedarf ändern. [OO Programmierung, S. 516]

Der View hingegen ist ausschließlich für die Darstellung (Ausgabe) der Daten zuständig. Der entsprechende Input hierfür wird von den Models geliefert. Außerdem hat ein View die Zusatzaufgabe Benutzereingaben entgegenzunehmen und diese an den Controller weiterzuleiten.

Der Controller hatte ursprünglich die Aufgabe, Benutzerinteraktionen zu verarbeiten (bspw. um einen Mausklick einem konkreten Button zuzuordnen). Diese Aufgaben werden mittlerweile jedoch von Modulen übernommen, die in Betriebssystem, Browser oder Basisbibliotheken integriert sind [OO Programmierung, S.515].
Ein Controller im heutigen Sinne ist für die Ablaufsteuerung einer Anwendung und die Ausführung entsprechender Operationen zuständig. Typisches Beispiel hierfür ist die Änderung der im Model enthaltenen Daten, nachdem der Benutzer ein Eingabeformular ausgefüllt und abgeschickt hat.

Das Prinzip des oben erwähnten Beobachter-Musters findet im Rahmen von MVC nun folgendermaßen Anwendung:

a) **View beobachtet Model**
Jeder View kann sich in die Beobachterliste eines Models eintragen. So kann das Model alle beobachtenden Views bei Aktualisierung der Daten benachrichtigen. Die Views wiederum aktualisieren dann bei Bedarf ihre Darstellung.

b) **Controller beobachtet Views**
Ein Controller kann Views beobachten, um im Falle von Benutzereingaben benachrichtigt zu werden. Der Controller holt sich die Eingaben bei Bedarf und leitet weitere Schritte ein.

4.2.3 Sencha Touch als formgebendes Framework

Ergänzung zum MVC-Konzept

Sencha Touch bietet neben den drei Standardkomponenten von MVC diverse zusätzliche Bausteine an. Im Folgenden sollen die für das Projekt relevanten Bestandteile von Sencha Touch beschrieben werden, um anschließend die Funktionsweise des Frameworks näher zu erläutern.

Der Store beinhaltet Model-Instanzen eines bestimmten Typs. Er stellt Funktionen bereit, um die Menge seiner Model-Objekte zu sortieren, zu gruppieren oder zu filtern. Außerdem können einzelne Model-Instanzen hinzugefügt, verändert oder entfernt werden.

Proxies werden von Stores benutzt, um die Daten eines Models aus dem Speicher zu laden bzw. diese wieder in den Speicher zu schreiben. Bei dem Speichermedium muss es sich allerdings nicht immer um die lokale Festplatte handeln. Auch die Verbindung zu einer externen Datenbank oder einem Webservice wären denkbare Möglichkeiten.

Model-Assoziationen Zusätzlich zu den bekannten Eigenschaften eines Models bietet Sencha Touch die Möglichkeit, einzelne Model-Typen miteinander in Beziehung zu versetzen. Dabei werden die drei Beziehungstypen „belongsTo“, „hasOne“ und „hasMany“ unterschieden. [Sencha Docs - Ext.data.association.Association] Anhand eines Beispiels lassen sich die Beziehung wie folgt erklären

a) belongsTo: jedes Auto gehört zu einem Besitzer

b) hasMany: ein Besitzer kann mehrere Autos besitzen

c) hasOne: jedes Auto hat genau einen Motor

Eine Model-Instanz kennt seine in Beziehung stehenden Model-Instanzen. Sencha Touch stellt auf Basis dieser Tatsache zusätzliche Funktionen bereit, um Model-Assoziationen zu traversieren. Im Falle des obigen Beispiels lässt sich also mithilfe einer Besitzer-Instanz (aufgrund der hasMany-Beziehung) ein Store erzeugen, der alle Autos dieses Besitzers enthält – durch einen einzigen Funktionsaufruf.

Funktionsweise von Sencha Touch

4.2.4 Der Bezug zur Self-Scanning App

Die drei Views der Anwendung

Aus Abbildung 1 auf S. 3 lassen sich die folgenden drei Benutzerdialoge erkennen: Einkauf beginnen, Einkaufswagen bearbeiten und Einkauf abschließen. Diese drei Dialoge entsprechen jeweils einem separaten View. Zusätzlich ist ein weiterer View für die manuelle Suche von Artikeln notwendig. Die folgende Abbildung veranschaulicht die Reihenfolge und Bestandteile der einzelnen Views:

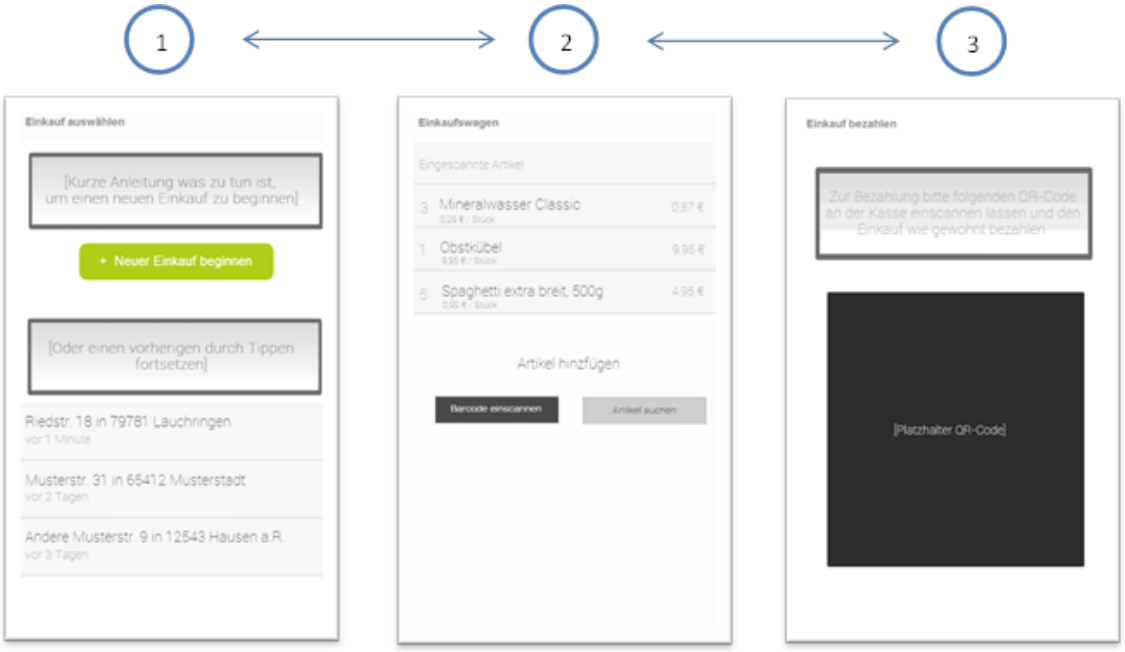

Abbildung 3: Abfolge der drei Benutzerdialoge

Models und die zugehörigen Assoziationen

Anhand der fachlichen Anforderungen in Kapitel 2.3 und des prinzipiellen Ablaufs in Kapitel 2.2.1 wurden die in Abbildung 4 dargestellten Model-Typen identifiziert.

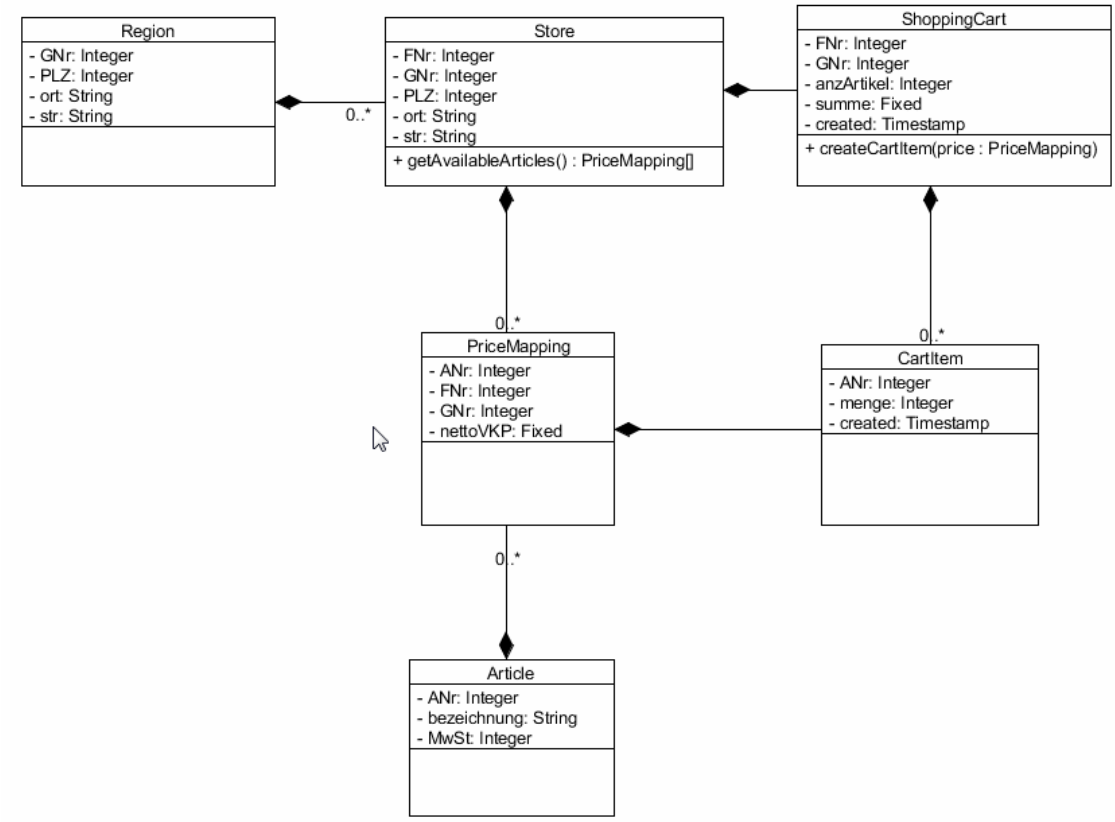

Abbildung 4: Zu implementierende Model-Typen und zugehörige Assoziationen

An erster Stelle stehen die beiden Models „Region“ und „Store“, um Gesellschaften und Filialen abzubilden. Diese besitzen eine landes- bwz. regionalweit eindeutige Nummer („GNr“- bzw. „FNr“-Attribut). Da jede Filiale Teil einer Gesellschaft ist (Komposition), wird sie anhand ihrer Filialnummer in Kombination mit der zugehörigen Gesellschaftsnummer identifiziert.

Außerdem gibt es ein „Article“-Model, das eine eindeutige Artikelnummer („ANr“), sowie weitere Artikelinformationen enthält. Zu einem Article gehören beliebig viele „PriceMapping“-Models. Mit deren Hilfe kann ein Artikel einer Filiale zugeordnet werden, mit der Zusatzinformation zu welchem Verkaufspreis der Artikel in dieser Filiale angeboten wird. Andersrum formuliert besitzt ein Store beliebig viele PriceMappings, wodurch sich eine Liste aller in dieser Filiale verfügbaren Artikel inkl. der zugehörigen Verkaufspreise ergibt.

Letztlich bleibt noch das „ShoppingCart“-Model, das alle Artikel („CartItems“) enthält, die im Einkaufswagen liegen. Jeder Einkaufswagen ist genau einer Filiale zu-

geordnet. Jedes CartItem wiederum genau einem PriceMapping.

Stores und Proxies

Zu jedem Model wurde ein zugehöriger Store angelegt, der die von Sencha Touch bereitgestellten Standardfunktionen (siehe [Kapitel: Sencha Touch – Stores]) anbietet.

Außerdem besitzt jedes Model einen eigenen Proxy, um die Daten in einer lokalen Datenbank abzuspeichern. Die Models Region, Store, PriceMapping und Article besitzen zusätzlich einen sog. „RemoteProxy“, der sich mit einer zentralen Datenbank im Internet verbinden und von dort aktuelle Informationen beziehen kann.

Der Controller

Die Anwendung soll lediglich einen einzigen Controller enthalten, der alle drei Views gleichzeitig beobachtet und auf entsprechende Ereignisse reagiert.

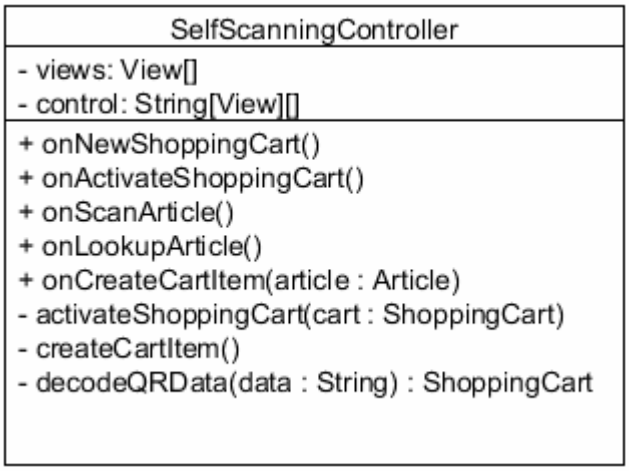

Abbildung 5: Der zentrale Controller der Anwendung

4.3 Der Entwurf einzelner Teilprozesse

4.3.1 Einkauf eröffnen

Abbildung 6 zeigt den Prozess „Einkauf eröffnen“. Dieser wird angestoßen, wenn sich ein Nutzer im „Einkauf auswählen“-Dialog dafür entscheidet, einen neuen Einkauf zu beginnen. Hierfür ruft der Controller zunächst den Scanner auf, um den QR-Code im Eingangsbereich für den Check-In Prozess zu erfassen. Die darin enthaltenen Daten werden anschließend dekodiert und die entsprechenden Models (PriceMapping-Models) aktualisiert bzw. angelegt (ShoppingCart-Model). Das neu erstellte Model wird dann dem Einkaufswagen-View zugewiesen und angezeigt.

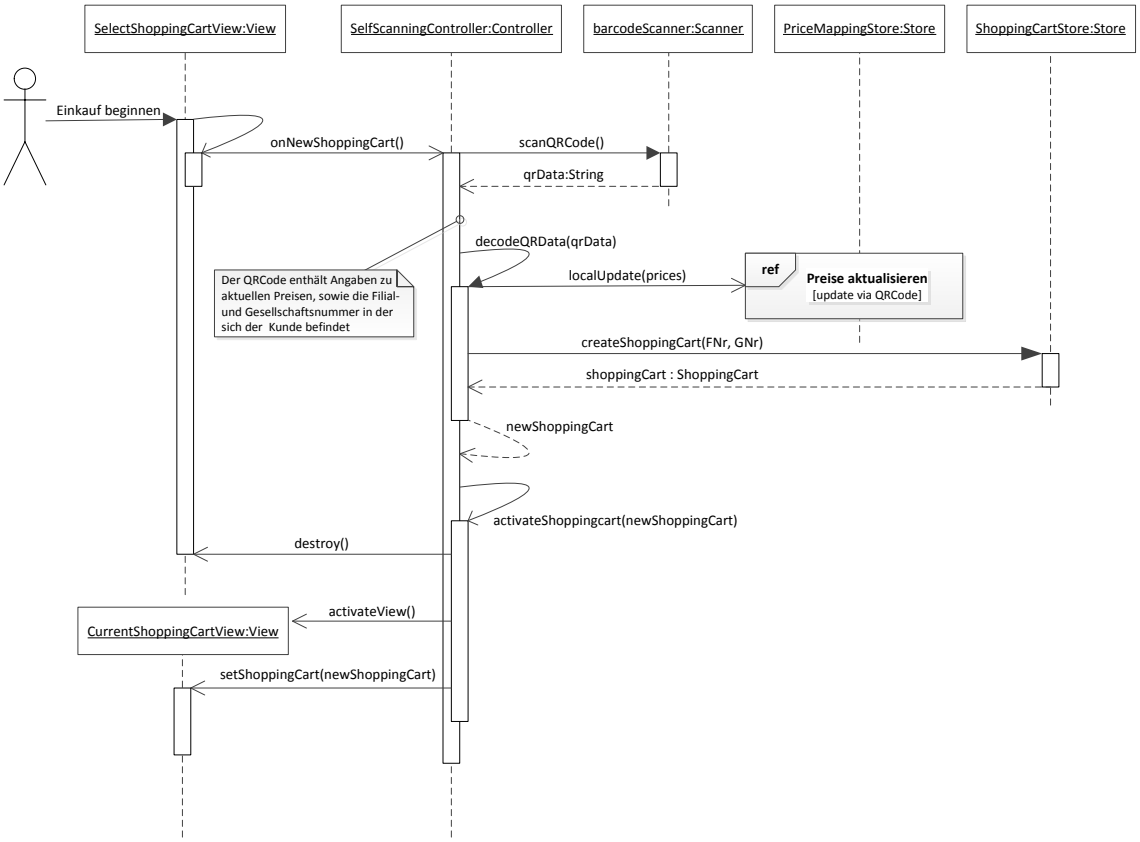

Abbildung 6: Der Prozess „Einkauf eröffnen“ als Sequenzdiagramm

4.3.2 Artikel hinzufügen

Um einen Artikel zu einem Einkaufswagen hinzuzufügen, bieten sich dem Nutzer zwei Möglichkeiten an: er scannt entweder den Barcode auf der Artikelverpackung oder sucht den passenden Artikel in einer Datenbank anhand Bezeichnung oder PLU-Nummer. Je nach Entscheidung werden die Methoden onScanArticle() bzw. onLookupArticle() ausgeführt, welche beide das passende Article-Model zurückliefern. Mit dessen Hilfe kann der Artikel dem aktuellen Einkaufswagen hinzugefügt und angezeigt werden.

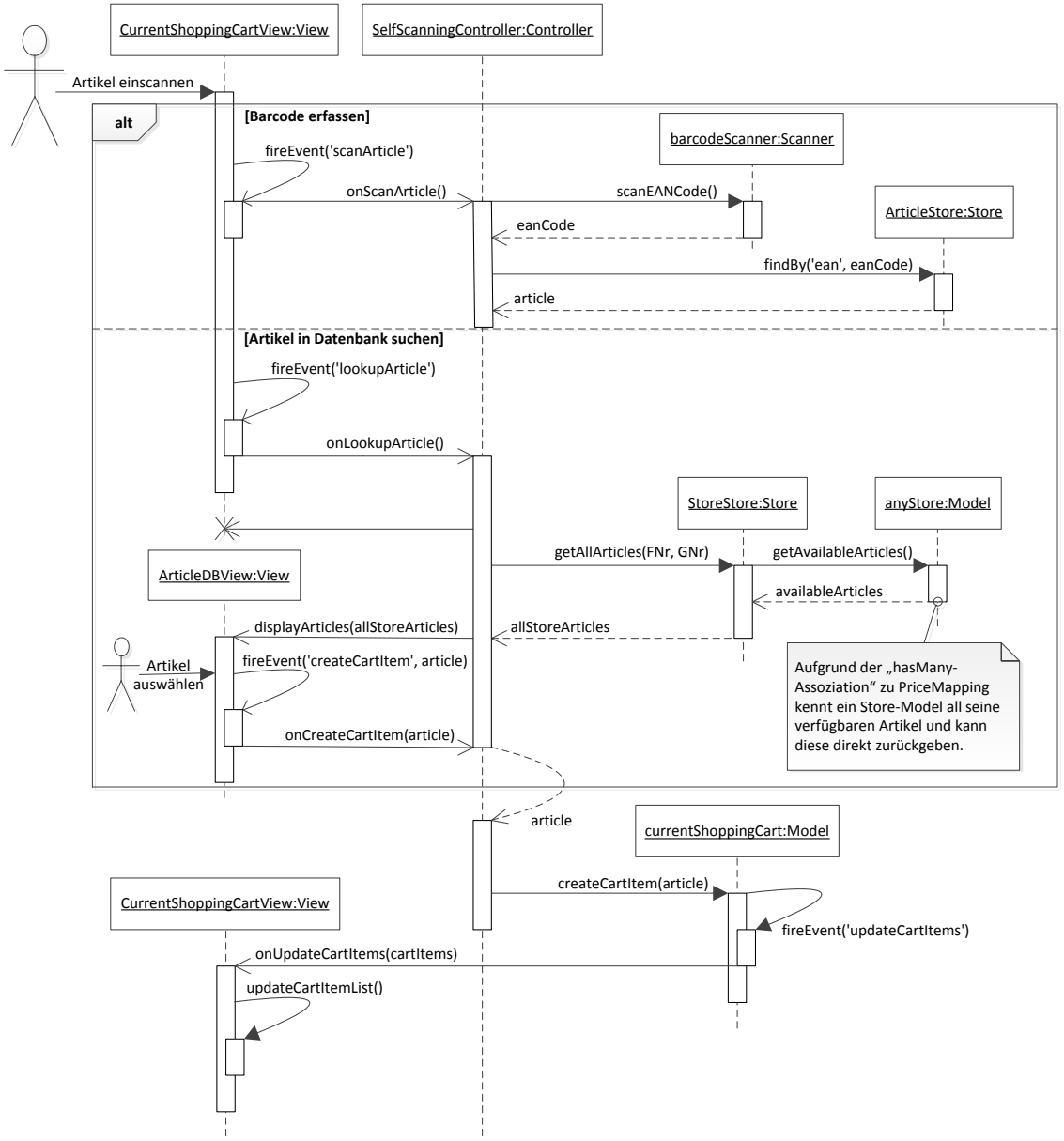

Abbildung 7: Der Prozess „Artikel hinzufügen“ als Sequenzdiagramm

4.3.3 Preise updaten

Preise lassen sich entweder anhand des QR-Codes beim Check-In aktualisieren oder nach Anstoßen der remoteUpdate()-Funktion, um ein Update über das Internet durchzuführen. Letzteres erfolgt über den PriceMappingProxy, der von einer zentralen Datenbank seine Daten bezieht.

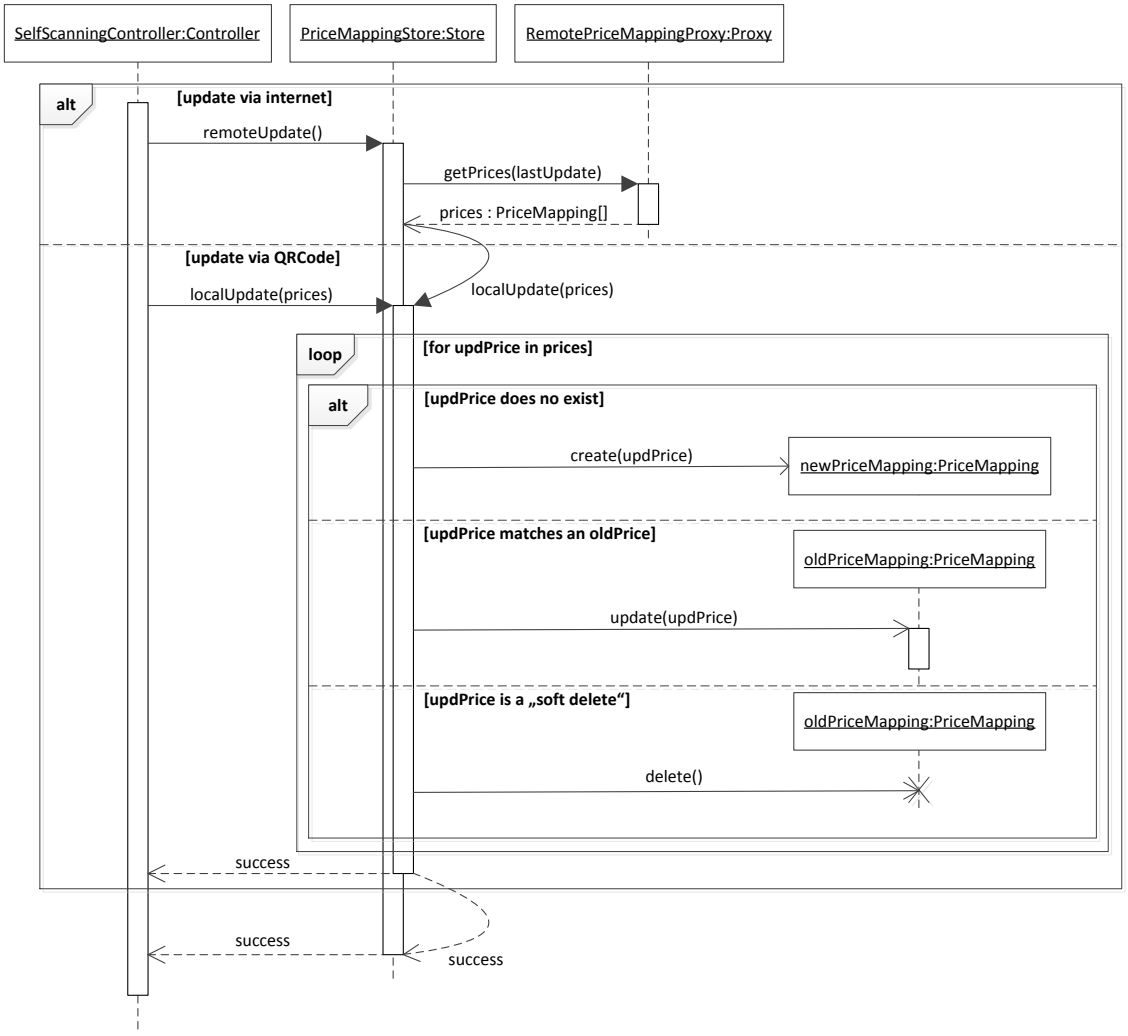

Abbildung 8: Der Prozess „Preise updaten“ als Sequenzdiagramm

4.4 Spezifikation der Schnittstellen

4.4.1 Kasse → Smartphone: Einchecken

Um dem Kunden auch dann aktuelle Preisinformationen zu bieten, wenn sein Smartphone nicht mit dem mobilen Internet verbunden ist, werden bei jedem Check-In die lokalen Preisveränderungen (LPVs) der Filiale an das Smartphone übertragen.

Hierfür muss der Kunde vor jedem Einkauf einen QR-Code im Eingangsbereich der Filiale einscannen. Dieser enthält Angaben zu seinem Standort (die Filiale in der er sich aufhält) und alle aktuell gültigen Preisveränderungen dieser Filiale. Die kodierten Informationen lassen sich als konkatenierten String darstellen, dessen einzelnen Bestandteile in der folgenden Tabelle zunächst aufgezählt und anschließend näher erläutert werden.

Pos.	Länge	Beschreibung
1	3	Gesellschaftsnummer
2	3	Filialnummer
3	var	Filialpreise
4	5	Separator
5	var	Gesellschaftspreise
6	5	Separator
7	var	Landespreise

Tabelle 6: Bestandteile des QR-Codes im Filialeingang

1. **Gesellschaftsnummer** (3 stellig)
   Nummer der Regionalgesellschaft
   Beispiel: 012 für Gesellschaft 12 (Donaueschingen)
2. **Filialnummer** (3 stellig)
   Nummer der Filiale
   Beispiel: 053 für Filiale 53 (Riedstr. 18 in Lauchringen)
3. **Filialpreise** (variable Länge)
   Das Filialpreisfeld ist wiederum ein zusammenhängender String aus einzelnen Artikelnummern und Preisen. Die ersten 5 Stellen enthalten die Artikelnummer (im Falle von kürzeren Artikelnummern werden die Stellen von links mit Nullen

aufgefüllt). Die folgenden und auch letzten 5 Stellen enthalten den Preis ohne Dezimaltrennzeichen, wobei die letzten beiden Ziffern als 1/10 und 1/100 Stellen interpretiert werden.

Beispiel: 0123400595313379950 für

- den Artikel 1234 zu einem Preis von 5,95 €
- den Artikel 31337 zu einem Preis von 99,50 €

Wichtige Hinweise

- Die Artikelnummer 00000 darf nicht vergeben werden! Die Zeichenfolge ist für den Separator (siehe unten) reserviert.
- Ein Preis der als 00000 kodiert ist, ist ein sogenannter „soft delete". D.h. diese Preisveränderung ist nicht länger gültig und muss (falls vorhanden) in der Datenbank des Smartphones entfernt werden.

4. **Separator** (5 stellig)
   Das Separatorfeld ist eine reservierte Zeichenkette aus fünf Nullen („00000"). Es dient dazu, die Filialpreise von den Gesellschaftspreisen zu trennen.

   Wichtiger Hinweis

   - Dieses Feld wird nur dann angehängt, wenn Gesellschaftspreise enthalten sind.

5. **Gesellschaftspreise** (variable Länge, optional)
   Pendant zu den Filialpreisen. Siehe *3. Filialpreise* für weitere Details.

6. **Separator** (5 stellig)
   Siehe *4. Separator*.

   Wichtiger Hinweis

   - Dieses Feld wird nur dann angehängt, wenn Landespreise enthalten sind.

7. **Landespreise** (variable Länge, optional)
   Pendant zu Filial- und Gesellschaftspreisen. Siehe *3. Filialpreise* für weitere Details.

Durch den nachfolgenden QR-Code wird ein Beispiel gegeben, das alle oben beschriebenen Informationen enthält. Sein Inhalt kann mit einer gewöhnlichen QRCode-Reader App nachgeprüft werden:

Abbildung 9: Beispielhafter QR-Code mit lokalen Preisveränderungen

```
012053011110000001235000500555500000066660000009999000000000
055550050006666000190999900000000000111100995012340190001235
000290555501250066660060009999000001234500556
```

4.4.2 Smartphone → Kasse: Bezahlung

Nachdem der Kunde alle Waren eingescannt hat, muss der virtuelle Einkaufswagen an das Kassensystem übertragen werden. Hierfür wird auf dem Display des Smartphones ein QR-Code angezeigt, der die Artikelnummern und zugehörigen Mengen aller eingescannten Waren enthält.

Der Inhalt des QR-Codes lässt sich als konkatenierten String darstellen. Seine einzelnen Bestandteile werden im Folgenden näher beschrieben.

Pos.	Länge	Beschreibung
1	3	Gesellschaftsnummer
2	3	Filialnummer
3	3	Anzahl Artikelpositionen
4	var	Artikeldetails
5	2	Anzahl Pfandbons
6	?	Pfandbons
7	6	Summe

Tabelle 7: Bestandteile des QR-Codes auf dem Smartphone

1. **Gesellschaftsnummer** (3 stellig)
   Analog zu *1. Gesellschaftsnummer* in Kapitel 4.4.1 auf S. 26.
2. **Filialnummer** (3 stellig)
   Analog zu *2. Filialnummer* in Kapitel 4.4.1 auf S. 26.
3. **Anzahl Artikelpositionen** (3 stellig)
   Anzahl aller Artikelpositionen (nicht die Gesamtmenge aller Artikel) im Einkaufswagen, zur Berechnung des hierauf folgenden Feldes (Artikelinformationen).

   Wichtiger Hinweis

   - verknüpfte Artikel (z.B. Pfand) sollen hier nicht enthalten sein, da sie auch nicht Bestandteil des nächsten Feldes sind.
4. **Artikeldetails** (variabel)
   Dieses Feld beinhaltet jeweils die Artikelnummer (5 stellig) mit zugehöriger Menge (2 stellig) jedes eingescannten Produkts. Artikelnr. und Menge werden gegebenenfalls von links mit Nullen aufgefüllt.

   Beispiel: 01234033133701 für

   - 3 Stück von Artikel 1234
   - 1 Stück von Artikel 31337

   Wichtiger Hinweis

   - Verknüpfte Artikel (Pfand) sollen hier nicht enthalten sein. Diese werden von der Kasse erneut dem Kassenbon hinzugefügt.
5. **Menge der Pfandbons** (2 stellig)
   Menge der im Einkaufswagen enthaltenen Pfandbons. Wird ggf. von links mit

Nullen aufgefüllt.

6. **Pfandbons** (variabel)
   TODO

7. **Gesamtsumme** (6 stellig)
   An der Kasse zu zahlender Betrag. Die Summe besteht aus 6 Zeichen, ohne Dezimaltrennzeichen. Die letzten beiden Ziffern werden als $\frac{1}{10}$ bzw. $\frac{1}{100}$ Stelle interpretiert.

4.5 Kapitelzusammenfassung

5 Ergebnisse der Implementierung und Ausblick

6 OLD - Entwurf der SelfScanning App

Zur Implementierung einer hybriden App bieten sich verschiedene sog. „mobile Application Frameworks“ an. Nach kurzer Durchsicht der gängigen Frameworks besteht der wesentliche Unterschied hauptsächlich in deren Funktionsumfang. Während einige Frameworks sich lediglich auf eine bestimmte Fähigkeit spezialisieren (bspw. „PhoneGap“ auf die Bereitstellung nativer Gerätefunktionen oder „jQuery mobile“ auf die Darstellung möglichst nativ-wirkender Benutzeroberflächen) gibt es andere, die sich als Multitalente profilieren.

Besonders die zwei Frameworks „Sencha Touch“ und „Kendo UI“ zeichnen sich neben der Bereitstellung von Schnittstellen zur Hardware und einer nativ-wirkenden Darstellung des UI durch ein zusätzliches, durchdachtes Architekturkonzept aus, das für den Entwurf komplexer Anwendungen besonders interessant ist. Da zur Entwicklung mit Kendo UI allerdings kostenpflichtige Lizenzen notwendig sind, soll das Projekt mithilfe von Sencha Touch realisiert werden.

Im folgenden Unterkapitel soll zunächst ein vollständiger Einkaufsprozess anhand eines Programmablaufplans dargestellt und erläutert werden. Das anschließende Kapitel geht etwas näher auf die grundsätzliche Funktionsweise von Sencha Touch und das damit in Verbindung stehende Architekturkonzept ein, um dann anschließend diese Prinzipien auf den Entwurf der Applikation anzuwenden. Anhand von Sequenzdiagrammen sollen einzelne Teilprozesse näher veranschaulicht und beschrieben werden. Abschließend folgt die formale Definition der für die Anwendung benötigten Schnittstellen.

Ehrenwörtliche Erklärung

„Ich erkläre hiermit ehrenwörtlich:

1. dass ich meine Projektarbeit mit dem Thema

   Prototypenentwicklung einer mobilen Applikation zum Self-Scanning in den Filialen

   ohne fremde Hilfe angefertigt habe;

2. dass ich die Übernahme wörtlicher Zitate aus der Literatur sowie die Verwendung der Gedanken anderer Autoren an den entsprechenden Stellen innerhalb der Projektarbeit gekennzeichnet habe;

3. dass ich meine Projektarbeit bei keiner anderen Prüfung vorgelegt habe;

4. dass die eingereichte elektronische Fassung exakt mit der eingereichten schriftlichen Fassung übereinstimmt.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.“

Ort, Datum Unterschrift