Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Markus Stevens

Einsatz von webbasierten Anwendungen in klassischen Rich-Client Umgebung am Beispiel vom Blumengroßmarkt Hamburg.

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Markus Stevens

Einsatz von webbasierten Anwendungen in klassischen Rich-Client Umgebung am Beispiel vom Blumengroßmarkt Hamburg.

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Sarstedt
Zweitgutachter: Prof. Dr. Zukunft

Eingereicht am: 08. November 2012

Markus Stevens

Thema der Arbeit
Einsatz von webbasierten Anwendungen in klassischen Rich-Client Umgebung am Beispiel vom Blumengroßmarkt Hamburg.

Stichworte
Warenwirtschaftssystem, Blumengroßhandel, Webanwendung, RubyOnRails, Cucumber, ATDD, ProductBacklog, Userstory, Akzeptanzkriterien

Kurzzusammenfassung
Dieses Dokument befasst sich mit der Neuentwicklung eines Warenwirtschaftssystems auf Basis von RubyOnRails. Die Anwendung ist ausgerichtet für den Blumengroßhandel und soll bestehnde Rich-Client Anwendungen ablösen. Der Entwicklungsprozess wird mit agilen Methoden unterstützt, wobei Userstorys, ProductBacklog, ATDD und Sprints eingesetzt werden.

Markus Stevens

Title of the paper
Use of web-based applications in classic rich-client enviroment at the example of the wholesale flower market in Hamburg.

Keywords
Enterprise resource planning, Flowers wholesale, Webapplication, RubyOnRails, Cucumber, ATDD, ProductBacklog, Userstory, Acceptance criteria

Abstract
This document deals with a new development of an enterprise resource planning software based on RubyOnRails. The application is focused on the wholesale flower market and should replace rich-client applications. The development prozess is assisted by agile methods in usage of Userstorys, ProductBacklog, ATDD and Sprints

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einleitung

1.1. Problemstellung

Die Entwicklung von Anwendungen für das Internet unterliegt einem stetigen Wachstum. Das liegt zum einen an der anhaltend wachsenden Anzahl an Benutzer des Internets [BITKOM 2007] und zum anderen an der intensiveren Nutzung, die durch den technologischen Fortschritt im Bereich des mobilen Internets ermöglicht wird. [BITKOM 2012]. Die quantitative Zunahme hat zur Folge, das mehr qualitative Funktionalität gefragt ist. Dies führt unmittelbar weg von statischen Webseiten hin zu dynamischen Webanwendungen , die in betriebliche Vorgänge eingebunden werden können.
Was sich in der Gesellschaft und in großen Unternehmen bereits etabliert hat, findet in vielen mittelständischen und kleinen Unternehmen noch keine Anwendung. Das liegt einerseits an den finanziellen Möglichkeiten der Unternehmen und andererseits auch an der Angst vor Datenverlust bzw. Verlust der Akzeptanz der Anwender, speziell im Bereich der betrieblichen Abrechnung und Warenwirtschaft. Diese Probleme sind in den verschiedenen Branchen sehr unterschiedlich stark ausgeprägt. So ist die Branche der Blumenhändler, bedingt durch den schnellen Verfall ihrer Ware, dem starken Druck der Internetgeschäfte nicht so ausgeliefert, wie anderen Branchen. Diese Entwicklung wird sich jedoch, wie es sich in der Lebensmittelbranche [EDEKA 2012] zeigt , noch nachholen.
Auf dem Blumengroßmarkt Hamburg ist eine derartige Entwicklung bereits zu sehen. Die persönliche Bereitschaft der Kunden, auf den Großmarkt zu gehen, sinkt zunehmend. Für den Fall das, es akzeptable Internetlösungen für das schnelle Geschäft der verwelkenden Ware gibt, wird sich ein Teil der Kundschaft aus finanziellen als auch zeitlichen Gründen den Weg zum Großmarkt ersparen.

Der Großhandel von Blumen unterliegt, wie jede andere Branche, einer Dokumentationspflicht gegenüber der öffentlichen Hand. Alle Umsätze müssen, durch den Bundesfinanzhof geregelt, dokumentiert werden. Um den gegebenen Vorgaben des Rechnungswesens genüge zu tun, setzen die verschiedenen Blumengroßhändler Softwarelösungen ein, die von einer einfachen Abrechnungssoftware wie etwa “Databecker Rechnungsdruckerei“ [Databecker 2013]

bis hin zu branchenspezifischen Warenwirtschaft wie “Brückner WinAB“ [Brückner 2012] reichen.

Diese Softwarelösungen sind üblicherweise Desktopanwendungen für Rich-Clients, deren Funktionsweise in den meisten Fällen nicht den Anforderungen des Internets gewachsen sind. Sollte in Zukunft also das Internetgeschäft mit dem Geschäft auf dem Großmarkt in Verbindung gebracht werden können, so ist das Zusammenspiel eines Warenwirtschaftssystems für Internetbestellungen als auch für die Verarbeitung von persönlichen Verkäufen auf dem Blumengroßmarkt sinnvoll. Ein Anwendungsbereich könnte hier die Vorbestellung von Blumenware anhand tagesaktueller Bilder sein.

1.2. Zielsetzung der Arbeit

Um den Anforderungen der Internetgeschäfte genüge zu tun, soll in dieser Arbeit überprüft werden ob sich aktuelle Softwarelösungen mit einer Webanwendung ersetzen lassen. In Zusammenarbeit mit verschiedenen Großhändlern soll ein Prototyp entstehen, der die Einsetzbarkeit von Webanwendungen am Blumengroßmarkt Hamburg zeigt.
Die Händler setzen bereits sehr unterschiedliche Lösungen ein und genauso unterschiedlich ist der Umgang mit ihnen. Die Anforderungen sollen deswegen ganz neu analysiert werden, um eine Software zu schaffen, die den aktuellen Bedarf deckt und nicht wie ihm Rahmen der aktuellen Lösungen die Anforderungen verändert. Hierzu soll durch agile Praktiken, in regelmäßigen Abständen, eine voll integrierte Version des Prototypen gegen die Akzeptanzkriterien der umgesetzen Userstory 3.1 geprüft werden.
Ganz im Sinne des Spruches “Der Weg ist das Ziel“ wird dabei nicht nur die Realisierung der Anforderungen und damit die fertige Software betrachtet, sondern ebenso ein Blick auf agilen Entwicklungspraktiken geworfen. So soll detailliert der Werdegang von einer Userstory über die Akzeptanztests hin zur Implementierung der Funktionalität betrachtet werden.

1.3. Fachliche Ausrichtung

Diese Arbeit umfasst den fachlichen Teil des Rechnungswesens. Zu dem eine Stammdatenverwaltung für Kunden, Artikeln und geschäftsprozessoptimierte Erstellung, Ausgabe und Speicherung von Lieferschein und Rechnung zählen.

1.4. Technische Problemstellungen

Aus technischer Sicher ergeben sich zwei Bereiche deren Betrachtung im Rahmen dieser Arbeit wichtig erscheinen. Der erste Bereich stellt die Zustandslosigkeit des Http Protokolls da. Der zweite Bereich die fehlende Fähigkeit, einer Webanwendung, hardwarenahe Funktionalität auf dem Client auszuführen.
Die Arbeit wird sich hauptsächlich mit der praktischen Anwendbarkeit des HTTP Protokolls im Rahmen eines Warenwirtschaftssystems beschäftigen. Die hardwarenahen Funktionalitäten, wie Barcodescannen oder direktes ansteuern eines Druckers, werden vorerst außen vor gelassen.

1.5. Aufbau der Arbeit

Im Kapitel Grundlagen werden Probleme der Händler beschrieben und bewertet. Es werden Anforderungen festgehalten, die keinen direkten Bezug zu den Händlern haben. Losgelöst von den Anforderungen und Problemen werden grundlegende Praktiken der agilen Softwareentwicklung beschrieben und der Einsatz von RubyOnRails begründet.
Der Anforderungsworkshop ist das erste von vier Kapiteln, die den Entwicklungsprozess beschreiben. Dieses Kapitel wird bewusst nicht als Grundlagen behandelt, weil es einen Teil der praktischen Verwendung der agilen Praktiken darstellt.
Die aus dem Workshop heraus entstanden Anforderungen werden in den Kapiteln Sprint 1 - 3 realisiert. Im Sinne des agilen Vorgehens werden in jedem Kapitel vollständig lauffähige Versionen der Anwendung erstellt.
Das Kapitel Release betrachtet den praktischen Einsatz der Anwendung nach Abschluss der Sprints. Im Kapitel Zusammenfassung werden die Ergebnisse betrachtet und ein Ausblick auf eine Weiterentwicklung geboten.
Der Anhang enthält vor allem Screenshots der Anwendung und Akzeptanzkriterien der einzelnen Userstorys .

2. Grundlagen

2.1. Ist Zustand

Die Blumengroßhändler teilen sich bei der Frage nach Ihrer Softwarelösung zur betrieblichen Abrechnung und Steuerung in drei Lager auf. Die Einen nutzen umfangreiche branchenspezifische Warenwirtschaftssysteme, andere einfache allgemeine Abrechnungssysteme und zuletzt diejenigen, die keine Softwarelösung nutzen.

2.1.1. Warenwirtschaft

Die Warenwirtschaftssysteme sind Desktopanwendungen für das Betriebssystem Microsoft Windows. Diese Rich-Client Anwendungen arbeiten mit einer zentralen Datenbankanbindung und bilden somit eine Client-Server Architektur. Sie werden hauptsächlich auf den PC im Büro betrieben, um Preisaktualisierungen der Artikel durch zu führen. Die aktualisierten Datenbestände werden entweder manuell per USB Datenträger auf einen gleichwertig installierten PC am Großmarkt oder im Büro auf Handgeräte zur mobilen Datenerfassung (MOBIDA) übertragen. Diese Handgeräte arbeiten autark und halten eine eigene Kopie des Datenbestands, der jeden Abend synchronisiert wird. Sie werden per Tastatur und druckempfindlichen Touchsreen bedient und bieten die Möglichkeit eindimensionale Barcodes zu scannen. Sie dienen dazu unmittelbar Rechnungen oder Lieferscheine zu erstellen, sobald der Kunde sich die Artikel aussucht. siehe Abb. 2.1

2.1.2. Rechnungsprogramme

Die einfachen Rechnungsprogramme sind ebenso Rich-Client Anwendungen für das Betriebssystem Windows und laufen zumeist als alleinstehende Lösung ohne Datenbankanbindung. Sie werden in den meisten Fällen auf tragbaren Computern betrieben, sodass eine Datenpflege vom Büro aus möglich ist. Sie werden ausschließlich zum Drucken von Rechnungen und Lieferscheinen verwendet und speichern die Adressen von Kunden und einfache Artikelinformationen.

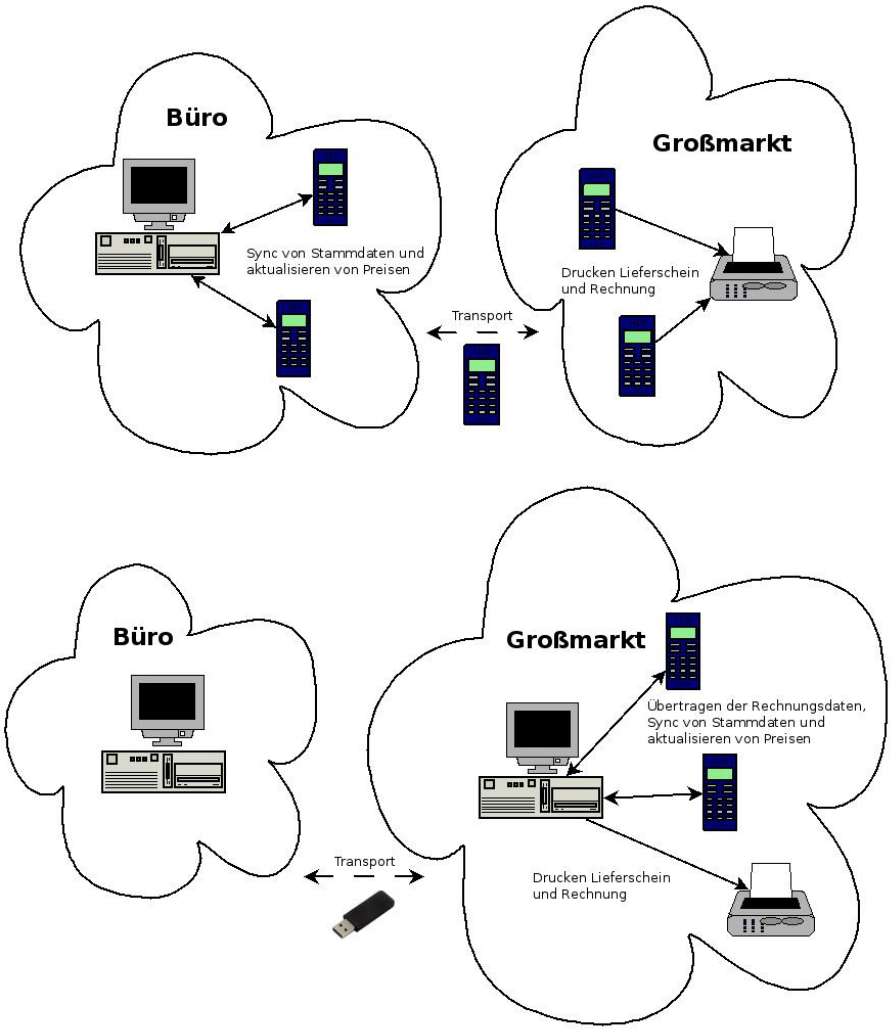


Abbildung 2.1.: Netzwerkdiagramme vom Einsatz von Warenwirtschaftssystemen

2.1.3. Keine Softwarelösung

Diejenigen ohne Softwarelösung nutzen sogenannte Rechnungsvordrucke mit einer laufenden Rechnungsnummer. Diese werden für jeden Händler individuell gedruckt. Um den steuerrechtlichen Bestimmungen genüge zu tun, müssen Einkäufe über 150€ mit dem Namen und Adresse des Käufers versehen werden. Dies wird zumeist aus dem Gedächtnis oder manuell aus Listen entnommen. Die Rechnungsdaten, wie die enthaltene Umsatzsteuer, werden per Hand ausgerechnet. Da eine zweifache Ausfertigung der Rechnung vorhanden sein muss, werden diese entweder zweimal geschrieben oder mittels Durchschlagpapier vervielfältigt.

2.1.4. Bestellungen

Bestellungen werden per Telefon, Fax oder Email aufgegeben und in den meisten Fällen auf Papier festgehalten. Die nötigen Artikeleinkäufe werden dann manuell aus ihnen errechnet.

2.2. Bewertung

Es stellt sich nun die Frage wie es zu diesen so unterschiedlichen Lösungen kommt. Ein Grund hierfür ist die sehr unterschiedlichen Größen der Händler, die Anzahl der Beschäftigten liegt zwischen einem und 25 Personen. So wird im kleinsten Fall nur eine Person mit der Softwarelösung arbeiten und in den größeren Betrieben mehrere Personen gleichzeitig einen Zugriff benötigen.
Ein weiterer Grund sind die doch erheblich abweichenden Anschaffungskosten. Die Warenwirtschaftssysteme fangen in einer Einzelplatzversion bei ungefähr 1.500 € an. Für die mobile Datenerfassung sind Preise pro Gerät bis zu 3.000 € keine Seltenheit. Dagegen stehen einfache Rechnungsprogramme bereits für 50 € zur Verfügung.
Viele Händler sehen zudem nicht die Notwendigkeit in einer komplexen Lösungen und wollen ausschließlich die Mindestanforderungen der steuerrechtlichen und handelsrechtlichen Vorgaben erfüllen. So werden Rechnungspositionen oftmals ungenügend mit der Angabe “An Blumen“ gebucht und für die Erfüllung der Vorgaben ein handschriftlicher Lieferschein mit den genaueren Angaben zum Artikel angeheftet.
Eine Befragung von fünf großen Händlern ergab diesbezüglich, das sie den Umfang ihrer Warenwirtschaftssysteme nicht ausnutzen. So werden von jedem die Funktionalitäten zur Erstellung von Rechnung und Lieferschein eingesetzt, jedoch von den wenigsten die Artikelbestände gepflegt und somit kaum Auswertungen über die Stammdaten vorgenommen. Der Grund für den Einsatz dieser komplexeren Systeme liegt hier wohl in der Fähigkeit mit mehre-

ren Benutzern gleichzeitig arbeiten zu können. Zudem bieten nur die Warenwirtschaftssysteme die Möglichkeit an mit mobilen Geräten die Verkäufe unmittelbar an der Ware aufzunehmen. Bei der Begutachtung von drei eingesetzten Lösungen fällt die benutzerunfreundliche Bedienung und hohe Komplexität auf. Stark überladene Benutzeroberflächen versuchen alle Funktionen unterzubringen. Hier fehlt es an Lösungen, die sich mit wenigen Handgriffen an die Bedürfnisse der Anwender anpassen lassen.
Diese Komplexität, die hohen Anschaffungskosten und die fehlende Notwendigkeit von Multiuserfähigkeit führt dazu, das einfache Rechnungsprogramme eingesetzt werden. Diese branchenfremden Lösungen bieten in der Regel genug Funktionalität für das Schreiben einfacher Rechnungen und Lieferscheine. Sie bieten dagegen aber nicht die ausreichenden Möglichkeiten, die sich schnell ändernden Artikeleigenschaften, wie Preise und Verfügbarkeit, zu pflegen. Anforderungen wie Pfandartikel, Preiskalkulation, Abrechnung von Kommissionsware werden außerhalb dieser Lösungen auf Papier erledigt. Ebenso sind aus diesen Systemen keine betriebswirtschaftlichen Auswertungen möglich. Hier kommt es zu Missständen gegenüber den Anforderungen der Steuergesetzgebung und dem eigenen wirtschaftlichen Überblick. Ausgeglichen wird dies mit erhöhtem Aufwand in der Buchhaltung und Steuerberatung, das automatisch höhere Kosten verursacht.

Ein großes Problem stellen zudem die verschiedenen Standorte dar. Jeder Händler hat mindestens ein Büro, von wo aus er Vorbestellungen entgegen nimmt, seine Ware bestellt und gegebenenfalls Preise kalkuliert. Sowie den Blumengroßmarkt, an dem er seine Ware verkauft, Rechnungen und Lieferscheine erstellt. Viele Händler sind gleichzeitig Erzeuger und somit ebenfalls in Ihren Gartenbaubetrieben tätig, von wo aus sie gleichermaßen Bestellungen bearbeiten müssen. Die Desktopanwendungen bieten hier nicht die nötige Flexibilität, um von unterschiedlichen Standorten aus arbeiten zu können. Hieraus ergibt sich die Folge der Zettelwirtschaft und Datentransporten im Koffer.

Die stetig steigenden Anforderungen der Finanzbehörden, wie die E-Bilanz [BMF 2011], aber auch die immer knapper kalkulierteren Handelspreise werden den Einsatz von umfangreicheren Softwarelösungen notwendig machen. In diesem Zusammenhang müssen die nötigen Informationen direkt beim Verkauf festgehalten werden, um sie später für Rentabilitätsbetrachtungen und Steuererklärung vorliegen zu haben. Um die Anwendbarkeit solcher Lösungen zu verbessern, müssen sie jedoch an die speziellen Anforderungen der Blumenhändler angepasst werden.

2.3. Einsatz agiler Praktiken

Die verschiedenen Sichten auf eine Software, führen zu unterschiedlichen Anforderungen. Die Sicht des Anwenders ist dabei wohl die Wichtigste. Der Anwender muss schließlich einen Mehrwert aus der Software ziehen können, woraus sich eine Akzeptanz aufbaut, die zum Begleichung der Rechnung führt. Um diesen Mehrwert aus Sicht des Entwicklers in den Vordergrund zu stellen, sind agile Praktiken gut geeignet.

Im Rahmen dieser Arbeit wird auf ausgewählte Praktiken der agilen Softwareentwicklung zurückgegriffen. So wird versucht durch den Einsatz von Userstorys, die Verständlichkeit der Anforderungen für den Kunden zu verbessern und so eine Priorisierung von seitens des Kunden zu ermöglichen. Im Gegensatz zu nicht agilen Methoden ist das Ziel in kurzen Abständen diejenigen Anforderungen vollständig zu integrieren, die der Kunde als die am höchsten priorisierten Anforderungen ansieht [Agile Manifest 2012] ohne dabei auf eine auf Vollständigkeit ausgelegte Spezifikation wert zu legen. Da dem Kunde viele Anforderungen nicht klar sind, soll frühzeitig mit lauffähigen Versionen die Software durch den Kunden geprüft werden. So kann frühzeitig erkannt werden, ob die Akzeptanz des Kunden vorhanden ist oder Anforderungen vergessen oder falsch priorisiert wurden. Diese Arbeit verfolgt jedoch keinen durchgängiges Ansatz, wie Scrum oder XP. Die Abweichungen von diesen Vorgangsmodellen, im Rahmen dieser Arbeit, wären zu Groß, als das dies Sinnvoll angewendet werden könnten.

Agile Praktiken in dieser Arbeit

- ProductVision
- UserStory
- Priorisierung anhand des Geschäftswertes bzw. MuSCoW
- Agiles Schätzen
- Product Backlog
- Akzeptanztest getriebene Entwicklung
- Sprints für vollständig Integrierte Teile der Software

Productvision

Ein Productvision soll in kürzester Weise die wesentlichsten Funktionalitäten eines Projekts beschreiben. Sie dient zur Motivation der Beteiligten und gibt gibt ein Ziel vor. Im späteren Verlauf eines Projekt bietet sie die Möglichkeit zurück zu blicken und zu prüfen, ob der erzielte Sinn und Zweck des Projekts noch immer verfolgt wird.

2.3.1. Userstory

Der Einsatz von Userstory wurde hier aus folgenden Gründen gewählt. Sie lassen sich vom Kunden lesen und schreiben. Sie enthalten genügend Informationen, um sie für eine grobe Priorisierung verwenden zu können, sind aber klein genug, um sich nicht in den Details zu verlieren, was oft genug zum Verlust des Überblicks führt. Um die Userstorys später auch verwenden zu können, müssen sie bestimmte Eigenschaften haben, die sich durch das Acronym INVEST gut beschreiben lassen: [WIRD 2011, S.64].

- I - Independent - Unabhängigkeit voneinander
- N - Negotiable - Verhandelbar
- V - Valueable - sollen einen Wert für den Kunden haben
- E - Estimatable - sollen schätzbar sein
- S - Small - sollen klein sein
- T - Testable - sollen testbar sein

Das Einhalten dieser Eigenschaften ist oft nicht einfach und so wird in dieser Arbeit mit der Anwendung des folgendem Musters versucht Abweichungen entgegen wirken.

Als <BenutzerRolle> will ich <das Ziel > [, so dass <Grund für das Ziel] [WIRD 2011, S.59]

Diese Schreibweise hilft dabei End-zu-End-Funktionalität zu formulieren. Für den Entwickler bedeutet dies, das er bei der Realisierung automatisch dazu gezwungen wird, einen vertikalen Schnitt durch seine Architektur zu machen und die Funktionsfähigkeit zu überprüfen. Für den Anwender wird damit die Testbarkeit gewährleisten.[WIRD 2011, S.113]

2.3.2. Akzeptanzkrierien

Die Akzeptanzkriterien beschreiben die Anforderungen an diese Anwendung und werden aus Userstorys heraus gewonnen. Sie konkretisieren die Userstory und beschreiben, wann sie als Fertig gilt[1]. Sie helfen eine Anwendung zu schreiben, die den Bedürfnissen des Kunden entspricht. Sie ersetzen das Pflichtenheft. [MORS 2012][s. 170]

2.3.3. ATDD mit Cucumber

Die Aktzeptanztest getriebene Entwicklung sieht vor, dass bevor eine Funktionalität implementiert wird mit dem Schreiben der Tests begonnen wird. Die Tests basieren auf Akzeptanzkriterien, die für jede Userstoy festgehalten werden. Sie beschreiben dabei die Funktionalität und das Verhalten der zukünftigen Anwendung. Es ist ein Ziel dem Auftraggeber eine verständliche Testumgebung, siehe Userstory U34, bereit zu stellen. Die Tests werden deshalb in einer für den Anwender verständlichen Sprache, einer sogenannten DSL[2], in Deutsch verfasst. Um zu verstehen wie Cucumber arbeitet und wie die Tests formuliert werden müssen, muss die Übersetzung der Schlüsselwörter bekannt sein:

```
cucumber --i18n de
      | feature          | "Funktionalitat"                    |
      | background       | "Grundlage"                         |
      | scenario         | "Szenario"                          |
      | scenario_outline | "Szenariogrundriss"                 |
      | examples         | "Beispiele"                         |
      | given            | "* ", "Angenommen ", "Gegeben sei " |
      | when             | "* ", "Wenn "                       |
      | then             | "* ", "Dann "                       |
      | and              | "* ", "Und "                        |
      | but              | "* ", "Aber "                       |
      | given (code)     | "Angenommen", "Gegebensei"          |
      | when (code)      | "Wenn"                              |
      | then (code)      | "Dann"                              |
      | and (code)       | "Und"                               |
      | but (code)       | "Aber"                              |
```

Im Rahmen dieser Arbeit wird das ATDD-Framework Cucumber eingesetzt. Die Test teilen sich in Funktionen auf, die wiederum einzelne Szenarios beinhalten. Jedes Szenario ist für sich geschlossen und beeinflusst die Anderen nicht. Die Akzeptanzkriterien werden damit

[1] DoD - Definition of Done
[2] Domain Specific Language = Fachsprache

auf Scenarios abgebildet und die übergeordneten Usertorys bilden die Funktionalität. In der Datei article.feature werden damit alle Akzeptanztest des Artikels der Testumgebung bekannt gegeben. Hier also ein Beispiel

```
# language: de
Funktionalität: U21 Als Einkäufer will ich einen Artikel neu
anlegen um zukünftig unter seinem Namen Rechnungen zu speichern.

Szenario:Eingabefelder dürfen nach fehlerhafter Formulareingabe
nicht verloren gehen.
Angenommen der Einkäufer geht zur Seite neuen Artikel anlegen
Wenn ich die Artikelname "Avalache+" eingebe
Und ich gebe keinen USt_Satz ein
Und die Gewinnspanne "0,25" eingebe
Und auf Speichern klicke
Dann wurde der Artikel "Avalache+"  nicht erfolgreich gespeichert
Und ich bekomme den Fehler "Umsatzsteuer muss ausgefüllt werden" angezeigt
Und das Feld "Name" enthält den Wert "Avalache+"
Und das Feld "Margin" enthält den  Wert "0,25"
```

Die in einer DSL verfassten Akzeptanztests werden in Ruby implementiert und in der Datei article_steps.rb gespeichert. Um Funktionalität auf der Seite des Benutzers prüfen zu können, wird die Integrationstestumgebung Capybara [CAPYBARA 2012] eingesetzt. Sie ermöglicht den Zugriff auf das Userinterface der Anwendung und simuliert damit den Anwender.

```
Angenommen /^der Einkäufer geht zur Seite neuen Artikel anlegen$/
do
  visit new_article_path
  page.should have_content("New Article")
end

Wenn /^ich die Artikelname "(.*?)" eingebe$/ do |arg1|
  fill_in "Name", :with => arg1
end

Wenn /^ich gebe keinen Umsatzsteuer_Satz ein$/ do
end

Wenn /^die Gewinnspanne "(.*?)" eingebe$/ do |arg1|
  fill_in "Margin",  :with => arg1
end
```

```
17
18 Wenn /^auf Speichern klicke$/ do
19   click_button "Speichern"
20 end
21
22 Dann /^wurde der Artikel "(.*?)"  nicht erfolgreich gespeichert$/
23 do |arg1|
24   if Article.exists?(:name => arg1)
25   raise("Artikel wurde falsch gespeichert")
26   end
27 end
28
29 Dann /^ich bekomme den Fehler "(.*?)" angezeigt$/ do |arg1|
30   page.should have_content(arg1)
31 end
32
33 Dann /^das Feld "(.*?)" enthält den Wert "(.*?)"$/
34 do |field, value|
35   field_labeled(field).value.should =~ /#{value}/
36 end
37
38 Dann /^das Feld "(.*?)" enthält den  Wert "(.*?)"$/
39 do |field, value|
40   field_labeled(field).value.should =~ /#{value}/
41 end
```

Cucumber sieht das schrittweise Erfüllen der Tests vor. Dabei ergeben sich die vier Schritte: Test formulieren, Test implementieren, Funktion implementieren, Test erfüllen. Diese Schritte werden solange wiederholt bis alle Tests fehlerfrei durchlaufen. siehe Abb. 2.2

Die anfängliche Mehrarbeit für die Tests wird bei der Implementation der Funktionalität wieder eingeholt, da nur relevante und zugleich richtigen Funktionalitäten implementiert werden.
Im Rahmen dieser Arbeit wurde festgestellt, das es wichtig ist sich über den Grad der Test-Abdeckung Gedanken zu machen. Die Priorisierung der Anforderungen helfen dabei, an den Stellen Akzeptanztestgetrieben vor zu gehen, die wichtig sind und unwichtigere außen vor zu lassen. Eine hundertprozentige Abdeckung steht in keinem Verhältnis zu der Relevanz vieler einzelner Funktionalitäten und wurden damit nicht über Cucumber implementiert.

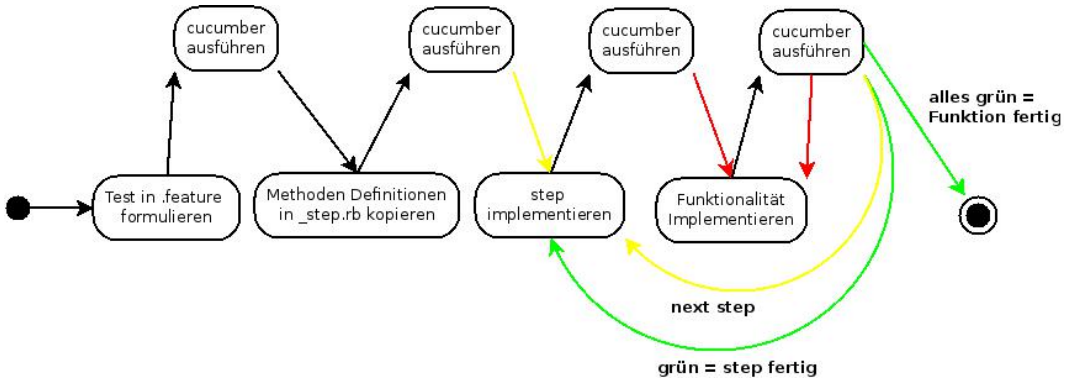


Abbildung 2.2.: ATDD, Schrittweises Vorgehen, “GRÜN“ machen

2.3.4. ProductBacklog

Das ProductBackog ist ein zentrales Dokument zur Verwaltung aller Anforderungen. Es enthält alle Userstorys und sonstigen Anforderungen, die für die Sprintplanung wichtig sind. Es wird nach Priorität der Anforderungen sortiert, wobei die wichtigsten oben und die weniger wichtigen unten stehen. Die obersten Userstorys müssen dabei am Detailliertesten sein. Zwischen jedem Sprint werden die erledigten Userstorys markiert und von den am höchsten priorisierte, unerledigte für den nächsten Sprint ausgewählt.

2.4. Einsatz von RubyOnRails

Der Kern dieser Arbeit ist das Erstellen einer Warenwirtschaft für das betriebliche Rechnungswesen auf Basis von Webtechnologien. Es wird damit eine Umgebung gesucht, die das Erstellen von umfangreichen Webanwendungen unterstützt.

2.4.1. Softwareanforderungen

Für das Erstellen einer betrieblichen Anwendung werden damit folgende Anforderungen gestellt.

- hohe Entwicklungsgeschwindigkeit
- Objektorientierte Programmierung
- Gewährleistung der Datenintegrität
- Trennung der Anwendungslogik und Darstellung

- gute Erweiterbarkeit für agiles Vorgehen
- Unterstützung von AJAX für die Optimierung des Verhalten der Benutzeroberflächen.
- Unterstützung von Akzeptanztest getriebener Entwicklung
- Lizenz Opensource

2.4.2. Entwicklungsgeschwindkeit

Eine hohe Entwicklungsgeschwindkeit ist einer der Hauptkriterien für dieses Projekt, da sie es ermöglicht im Rahmen der agilen Softwareentwicklung schnell auf Änderungen reagieren zu können. RubyOnRails bietet hier mehrere Konzepte, die eine hohe Geschwindigkeit ermöglichen, zu nennen wäre hier das DRY[3] - Prinzip . Es sagt aus: "Schreibe keinen Code zweimal, wenn der logische Zusammenhang an anderer Stelle bereits festgelegt ist". Ein Beispiel hierfür ist die objektrelationale Abbildung ActiveRecord. Die Definition der Attribute einer Klasse ergeben sich aus den Spalten der zugehörigen Datenbanktabellen. Alle Getter und Setter sind automatisch vorhanden und müssen nicht definiert werden. Ein weiteres Prinzip ist das CoC[4] - Prinzip, das eine Konvention einer Konfigurations vorzieht. Diese durchgängige Namens und Strukturkonventionen, siehe Abb. 2.3, erlauben es weitaus weniger Konfiguration schreiben zu müssen. Zeiteinsparung sowohl bei der Ersteinrichtung, als auch bei der späteren Pflege und Restrukturierung ist die Folge. Nachweislich sind die Lines of Code in in einer RubyOnRails Anwendung erheblich weniger, als in einer vergleichbaren Struts oder Spring-Anwendung. [JavaSpektrum 2006][S.22 Line Of Code].

Für eine schnelle Entwicklungszeit spricht PHP als Alternative. Leider ist die Objektorientierung bei PHP merklich aufgesetzt und führt bei komplexen Projekten schnell zu schlechter Wartbarkeit. Der Einsatz von Frameworks, wie beispielsweise Zent, ist natürlich denkbar aber begründet den Einsatz von PHP nicht.

2.4.3. Erweiterbarkeit

Für eine gute Erweiterbarkeit in RubyOnRails sprechen, neben den bereits oben genannten Prinzipien, auch die Art und Weise wie relationale Abbildungen mit ActiveRecord dargestellt werden. Die Relationen zwischen den Objekten werden ausschließlich in den Klassendefinition festgelegt. Für eine 1:n Relation zwischen Klassen "customer" und "invoice" sieht eine Definition wie folgt aus.

[3] Don'Repeat Yourself vgl. Once and Only Once
[4] Convention over Configuration

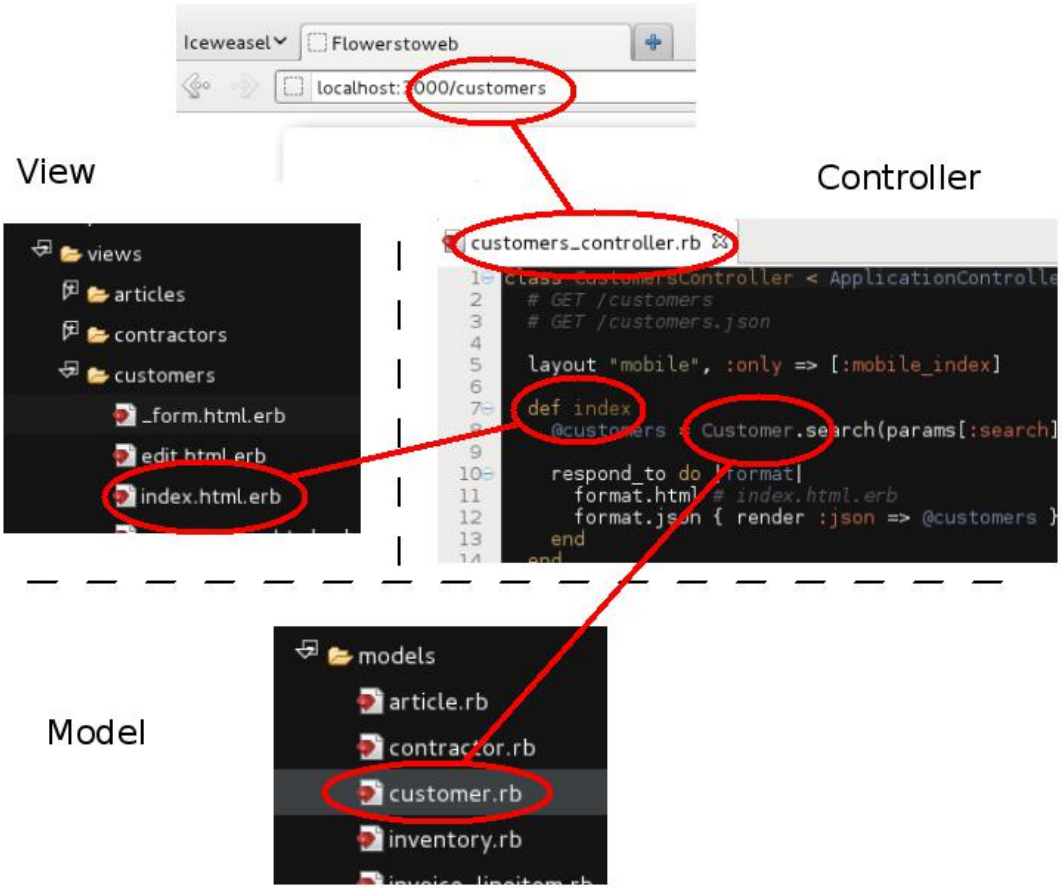


Abbildung 2.3.: Convention over Configuration

```
class Customer < ActiveRecord::Base
  has_many :orders, :dependent => :restrict
end
...
class Invoice < ActiveRecord::Base
  belongs_to :customer
end
```

Das bewusste Verzichten auf Fremdschlüsseln und Bedingungen in der Datenbank erscheint zuerst als strukturtechnische Schwachstelle. Es erlaubt jedoch einen wesentlich einfacheren Umgang bei Test, Pflege sowie Restrukturierung der Anwendung.

ActiveRecord bildet nach dem CoC-Prinzip eine Klasse “customer“ auf die Tabelle “customers“ ab. Die Spalten der Tabelle stellen dabei die Attribute der Klasse da. Für die meisten Datenzugriffe sind fertige Funktionen vorhanden. So ist es in den seltensten Fällen notwendig SQL Code zu schreiben. Ein gutes Beispiel hierfür sind die dynamischen “find“ Methoden. Wie z.B.

```
Customer.find_by_first_name("Uwe")
```

Auch komplexere bedingte Abfragen werden auf das Wesentliche reduziert.

```
Customer.where("first_name like ? or last_name like ?",
%:search_string%)
```

Ebenso wie Abfragen über mehrere Tabellen.

```
OrderLineitem.joins(:order).where("customer_id =
#{localization.id} and product_id = #{self.id}")
```

2.4.4. Datenintegrität

Für Datenintegrität sorgt ActiveRecord mit einer integrierten Validierungsumgebung und der Unterstützt für optimistisches oder wahlweise pessimistisches Sperren von Datenbankeinträgen. Die Validierung findet auf Modelebene statt und macht es damit unmöglich sie zu umgehen. Eine Validierung auf Ebene der Controller wäre zwar denkbar, ließe sich aber bei der Nutzung der Klassen leicht aushebeln. Eine browserseitige Variante würde hier keine ausreichende Sicherheit mit sich bringen, da dies zumeist mit Javascript implementiert wird, was durch Abschalten oder Manipulieren zu Fehlern führen würde. Natürlich gelten diese Angaben nur, solange der Zugriff über ActiveRecord des hier erzielten Systems stattfindet. Sollte später mit einem fremden System auf die Datenbank zugegriffen werden, so macht das

Verwenden von Fremdschlüsseln und Bedingungen in der Datenbank sicherlich Sinn. Die Validierung von ActiveRecord wird dann helfen genau diese Bedingungen zu erfüllen.

```
class Article < ActiveRecord::Base
  has_many :inventories
  validates :number, :name, :tax, :margin, :presence => true
  validates :number, :uniqueness => { :case_sensitive=> false }
end
```

2.4.5. Trennung der Anwendungslogik und Darstellung

Die Trennung der Anwendungslogik und ihrer Darstellung hat das Ziel die Flexibilität, Modularität und die Wiederverwendbarkeit des Codes zu erhöhen. [MORS 2012][S. 24] RubyOnRails setzt dies mit dem Model-View-Controller-Paradigma um.

Der Architekturentwurf Model-View-Controller sieht dabei vor, dass die Verantwortlichkeiten für die Objekte, der Geschäftslogik und der Darstellung entkoppelt werden, (siehe Abb. 2.4).

2.4.6. Scaffolding

Das Scaffolding dient ebenso der Verbesserung der Entwicklungsgeschwindigkeit, wie auch den Prinzipien der agilen Softwareentwicklung. Unter dem Begriff versteht man das Erstellen von Grundgerüsten, um den Beginn der Entwicklung zu beschleunigen. Durch die voll integrierten, also lauffähigen, Gerüste ist eine schnelle Konzentration auf individuelle Funktionalität möglich. Im späteren Verlauf werden die nicht verwendeten Bestandteile der Gerüste wieder entfernt. RubyOnRails kann also mittels eines Befehls ein komplettes Gerüst für Create, Read, Update und Delete erstellen. Dem Framework ist das zum einen durch das CoC-Prinzip und zum anderen durch das verfolgte Programmparadigma Restful[5] möglich.

```
rails generate scaffold customer first_name:string
last_name:string
```

2.5. Softwareumgebung

Zu vielen, der schon beschriebenen aber ebenso noch kommenden Anforderungen an eine Softwareumgebung, gibt es eine Vielzahl von Alternativen. Ohne diese Alternativen im Detail zu vergleichen, werden im Rahmen dieser Arbeit folgende Versionen, Erweiterungen und Tools

[5] Representational State Transfer

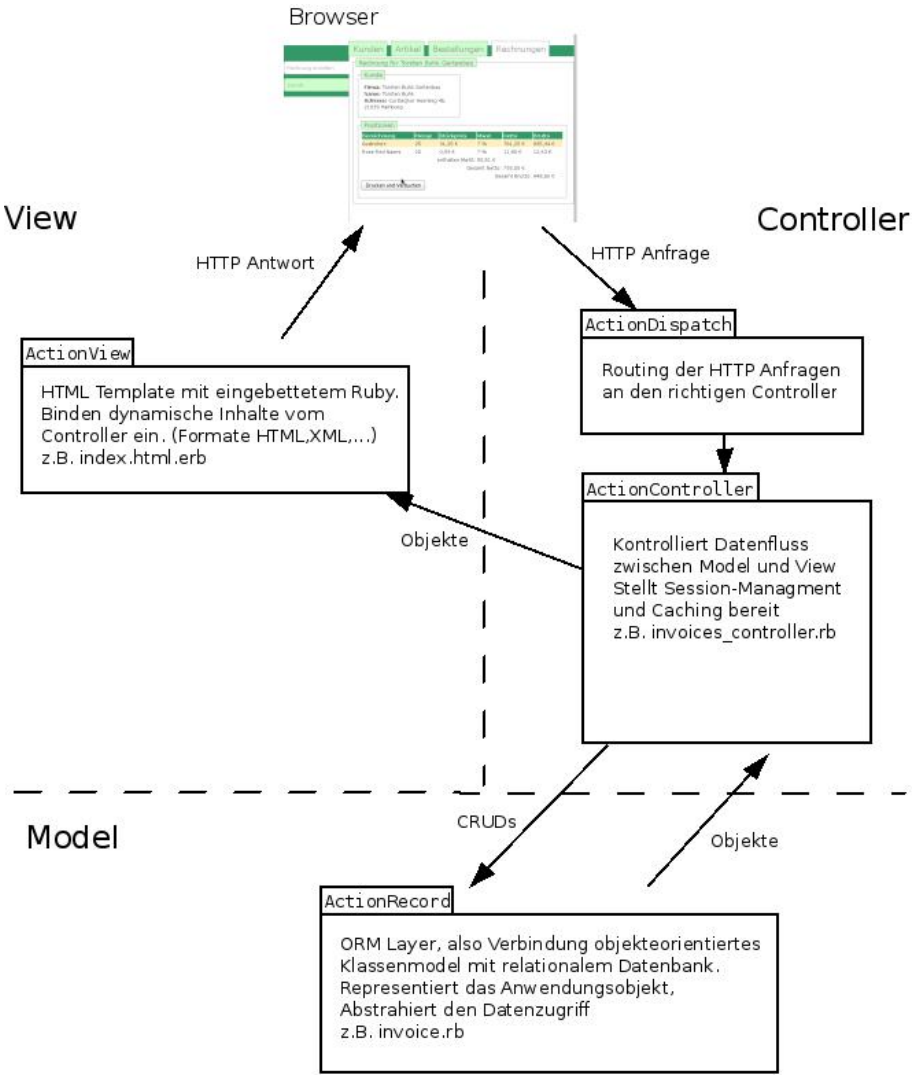


Abbildung 2.4.: ModelViewController Paradigma

eingesetzt. Kriterien für diese Auswahl sind gute Dokumentation, hoher Verbreitungsgrad, ausgiebig geprüfte Stabilität und Unterstützung heterogener Umgebung.

- Business Logik: Ruby
- Webframework: RubyOnRails 3.2 (RoR)
- UserInterface: HTML5,CSS3, Javascript mittels dem Framework JQuery
- Http Server: Apache2
- Application Server: Fusion Passenger
- Datenbank: Mysql
- Versionskontrolle: Git
- Deployment: Capistrano
- Testen: Cucumber (ATDD)

2.6. Anforderungen

Wenn Anforderungen durch agile Methoden erörtert werden, so haben diese im Regelfall einen starken Bezug zu den persönlichen Anforderungen des Auftraggebers. Dieser Bezug ist gewollt und führt zu einer akzeptierten Software. Jedoch sind Einflüsse von außen ebenso zu beachten. So wird in diesem kurzen Abschnitt die Anforderungen vom Gesetzgeber und die seitens der Entwickler betrachtet. Die fachlichen Anforderungen werden in dem Kapitel “Anforderungsworkshop“ analysiert, denn sie gehören genauso zu einem agilen Entwicklungprozess wie die Implementierung und sind somit in einem eigenen Kapitel.

2.6.1. Anforderungen aus Sicht des Entwicklers

Die Anforderungen werden, wie auch später im Anforderungsworkshop, als Userstorys festgehalten, um eine möglichst konsistente Schreibweise zu gewährleisten. Hierfür wird eine initiale Rolle des Entwicklers eingeführt

- U26 Der Entwickler will eine Versionskontrolle um mehr Sicherheit vor versehentliches Überschreiben von Quellcode

- U27 Der Entwickler will einen automatisiertes Deployment um schneller veröffentlichen zu können.
- U28 Der Entwickler will ein integrierte Entwicklungsumgebung nutzen um nicht lange nach Werkzeugen zu suchen und damit Zeit zu sparen.
- U29 Der Entwickler will einen Produktivserver um frühzeitig Releases zu veröffentlichen.
- U30 Der Entwickler will ein Webframework das ihn bei der schnellen Umsetzung der Anforderungen hilft um in wöchentlichen Schritten lauffähige vollintegrierte Versionen der Software zu veröffentlichen.
- U34 Der Entwickler will die akzeptanztestgetriebene Entwicklung einsetzten um die Testbarkeit durch die Stakeholder zu erhöhen.

2.6.2. Anforderungen vom Gesetzgeber

Neben den Anforderungen, die der Blumengroßhandel im Speziellen an ein Rechnungswesen hat, gibt es verpflichtende Anforderungen, die durch die Vorgaben des Umsatzsteuergesetzes und des Handelgesetzbuches bestimmt werden. Sie lassen sich zumeist in offene Anforderungen überführen. Dies soll in diesem Abschnitt genauer analysiert werden, um sie aus den fachlichen Anforderungen des Blumengroßhandels zu separieren. Sie sind von pauschal hohem Geschäftswert, da sie uns die Rechtssicherheit gegenüber Finanzamt und Handelspartnern geben sollen.

Rechnungen müssen nach geltendem Recht gem. § 14 Abs. 4 i.V.m. § 14a Abs. 5 UStG folgende Angaben enthalten:[IHK 2011]

- A2 Vollständiger Name und Anschrift des leistenden Unternehmers und des Leistungsempfängers
- A3 Steuernummer oder Umsatzsteueridentifikationsnummer
- A4 Ausstellungsdatum der Rechnung
- A5 Fortlaufende Rechnungsnummer
- A6 Menge und handelsübliche Bezeichnung der gelieferten Gegenstände oder die Art und den Umfang der sonstigen Leistung
- A7 Zeitpunkt der Lieferung bzw. Leistung

- A8 Nach Steuersätzen und -befreiungen aufgeschlüsseltes Entgelt
- A9 Im Voraus vereinbarte Minderungen des Entgelts
- A10 Entgelt und hierauf entfallender Steuerbetrag sowie Hinweis auf Steuerbefreiung
- A11 Ggf. Hinweis auf Steuerschuld des Leistungsempfängers
- A12 Als Rechnung gilt auch eine Gutschrift, die vom Leistungsempfänger ausgestellt wird.

Des Weiteren gibt es verpflichtende Angaben für einen Geschäftsbrief nach dem HGB, zu dem auch eine Rechnung zählt.[IHK 2011] Diese Angaben unterscheiden sich zwischen den Gesellschaftformen stark und müssen von Kunde zu Kunde angepasst werden. Sie werden hier nicht weiter beachtet, da es sich um Formularanpassungen handelt, die die Software in ihrer Funktionsfähigkeit nicht beeinflussen.

2.6.3. Nicht funktionale Anforderungen

Die nicht funktionale Anforderungen sind zu Beginn nicht vollständig betrachtet, können aber im Verlauf des agilen Vorgehens Entscheidungen beeinflussen und mit bedacht werden. Zwei Anforderungen dieser Art lassen sich aber bereits festhalten und sollen von Anbeginn an mit einfließen.

- A1 Mit Hilfe der Anwendung muss es möglich sein innerhalb von 2 Minuten dem Kunden eine ausgedruckte Rechnung in die Hand zu geben.
- A13 Ein unautorisierter Zugriff auf die Applikation muss verhindert werden.

3. Anforderungsworkshop

Der Anforderungsworkshop dient zur Analyse der Anforderung und der Motivation der Beteiligten. Er wird in Zusammenarbeit mit potenziellen Anwendern der Software durchgeführt. Geladen wurden dafür zwei Händler, verschiedener Größe die bereits unterschiedliche Lösungen für ihre betriebliche Abrechnung einsetzen.

3.1. Productvision und Vorstellung

Eine Productvision sollte mit konkreten Angaben verfasst werden. Somit wird der Name “flowerstoweb“ für das Projekt bestimmt.

FlowersToWeb ist eine Software für das Rechnungswesen mit integriertem Artikelstamm. Sie lässt sich vom Büro, von Zuhause und auf dem Blumengroßmarkt gleichzeitig nutzen. Die Abrechnung ist so vereinfacht, das sie ein nicht fachkundiger Anwender nach einem Tag benutzen kann. Eine Abrechnung am Blumengroßmarkt dauert weniger als 2 Minuten. Es ist möglich 3 Kunden gleichzeitig zu bedienen. Vorbestellungen können vom Kunden über das Internet abgegeben werden und lassen sich am Großmarkt direkt in eine Abrechnung überführen. FlowersToWeb bringt den Großmarkt und Handelsgeschäfte im Internet zusammen.

3.2. Rollen

Im Rahmen des ersten Anforderungsworkshops wurden folgenden Rollen erarbeitet, mit deren Hilfe sich die Teilnehmer in die verschiedenen Anwendungssituationen hinein versetzten, um im zweiten Schritt Userstory schreiben zu können.

- Kunde
- Einkäufer
- Verkäufer
- Buchhalter

3.3. Userstorys

Um ein initiales Product Backlog zu erstellen, wurde im Rahmen eines Workshops unter Beteiligung ausgewählter Großhändler die ersten Anforderungen gesammelt und einer einfachen Priorisierung unterzogen. Anforderungen werden durch Brainstorming erörtert und mit Hilfe des ProductOwners durch die Beteiligten in Form von Userstorys auf Karteikarten verfasst, (siehe Abb. 3.2)

- Kunde
  - U1 Als Kunde will ich eine Rechnung mit meiner Adresse und einer Rechnungsnummer, um den Einkauf gegenüber dem Finanzamt belegen zu können.
  - U2 Als Kunde will ich eine Rechnung mit ausgewiesener Umsatzsteuer, um sie als Beleg für meine Vorsteuer nutzen zu können.
  - U12 Als Kunde will ich einen Lieferschein ohne Preise, da der Abholer die Preise nicht kennen soll.
  - U14 Als Kunde will ich eine Sammelrechnung bekommen, um nicht für jeden Einkauf einen Zahlungsvorgang starten zu müssen.
- Verkäufer
  - U3 Als Verkäufer will ich Bestellungen anlegen können, um sie später in eine Rechnung umzuwandeln.
  - U6 Als Verkäufer will ich auf eine Rechnung Skonti geben können, um gute Kunden zu binden.
  - U5 Als Einkäufer will ich Umsätze von Kommissionsware angezeigt bekommen, um eine Abrechnung mit dem Lieferanten machen zu können.
  - U7 Als Verkäufer will ich den Stückpreis und die verkaufte Menge für jede Rechnungsposition angeben können, um die Positionen den realen Vorgängen an zu passen.
  - U10 Als Verkäufer will ich dem Kunden eine Gutschrift geben können, um zurück gegebenes Pfandgut verrechnen zu können.
  - U13 Als Verkäufer will ich einen Lieferschein drucken können, um die Ware zusammenzustellen.
  - U17 Als Verkäufer will ich eine Rechnung ausdrucken, um sie meinem Kunden mit zu geben.

  - U18 Als Verkäufer will ich einen Kunden neu anlegen, um zukünftig unter seinem Namen Rechnungen zu speichern.
  - U19 Als Verkäufer will ich nach Kunden suchen, um eine neue Bestellung an zu legen.
  - U20 Als Verkäufer will ich eine Rechnung nach Barzahlung als bezahlt markieren können.
  - U23 Als Verkäufer möchte ich nach Artikeln suchen, um Anfragen nach Preis oder Verfügbarkeit beantworten zu können.
  - U25 Als Verkäufer möchte ich bei der Suche nach Artikeln simultan zur Eingabe Suchtreffer angezeigt bekommen, um schneller Artikel zu finden.
  - U31 Als Verkäufer will ich einen Artikel auswählen, um ihn in eine Bestellung aufzunehmen.
  - U32 Als Verkäufer will ich einen Rechnung schreiben, um meine verkaufte Ware mit meinem Kunden abzurechnen.
  - U35 Als Verkäufer will ich wissen welcher Kunde meine Pfandartikel hat, um sie abrechnen zu können.

- Einkäufer
  - U4 Als Einkäufer möchte ich Bestellungen einsehen können, um eine Übersicht über zu bestellende Ware zu bekommen.
  - U8 Als Einkäufer will ich an verschiedenen Geräten an unterschiedlichen Standorten meine Artikelbestände pflegen können, um Zeit zu sparen.
  - U16 Als Einkäufer will ich sehen können wie mein aktueller Artikelbestand ist, um ggf. neue Artikel zu ordern.
  - U21 Als Einkäufer will ich neue Artikel anlegen können, um ihren Verbleib nachvollziehen zu können.
  - U33 Als Einkäufer will ich einzelnen Einkäufe eines Artikels unterscheiden können, um mir Preisentwicklung und Einkaufsmengen aus der Vergangenheit an zu sehen.

- Buchhalter
  - U9 Als Buchhalter will ich den Gewinn einzelner Einkäufe gegenüber den Verkäufen sehen, um die Rentabilität der Unternehmung zu ermitteln.

- U15 Als Buchhalter möchte ich die Rechnungen in einer medienbruchfreien Art in meine Buchhaltung übernehmen.

3.4. Prioriserung nach MuSCoW

Eine erste Priorisierung in Zusammenarbeit mit den Großhändlern wurde nach MusCow im Rahmen des Workshops durchgeführt. Diese Art der Priorisierung kommt ohne konkrete Angaben von Geschäftswert, Kosten, Risiko und Kundenzufriedenheit aus und unterstützt in dieser ersten Phase so eine intuitiv Herangehensweise [WIRD 2011, S.110], (siehe Abb. 3.1)

3.5. Priorisierung nach Abhängikeiten

Diese Priorisierung ist zum größten Teil bestimmt durch die Entwicklung. Zwar sollten Userstorys unabhängig voneinander sein, doch lässt sich das aus technischer Sicht nicht immer verhindern. So ist Beispielsweise die Userstory U32 vor U7 zu implementieren, denn bevor ich Rechnungspositionen anlegen kann, brauche ich eine Rechnung. So macht es Sinn bestimmte Userstorys mit ihrer Abhängigkeit zu anderen in eine Rangfolge zu bringen.

3.6. Priorisierung nach Geschäftswert

Um eine weitere Art der Priorisierung zu überprüfen, wurden die gefunden Userstory nach ihrem Geschäftswert priorisiert. Hierfür werden zwei Faktoren berücksichtigt. Zum einen die Möglichkeit direkt mit dieser umgesetzten Funktion Geld zu verdienen und zum anderen mit ihr Geld zu sparen.

3.7. Aufwandsschätzung

Zur Schätzung der Storys wird eine agile Methode angewandt, die nicht wie üblich die Entwicklungsdauer betrachtet, sondern die Größenverhältnisse zueinander. So ist die Userstory U31 als Mittelgroße Userstory bestimmt worden und wird mit drei Punkten geschätzt. Alle anderen werden zu ihr in Relation gesetzt und bekommen die Wertigkeiten “1“ Klein, “3“ Mittel, “5“ Groß.[WIRD 2011, S.13] Diese Größenangaben werden verwendend, um anhand der gegebenen Geschwindigkeit [1] des Entwicklungsteams, den Umfang der im Sprint[2] enthalten Userstorys zu bestimmen.

[1] auch Velocity genannt
[2] kurze Iteration mit dem Ziel vollständig integrierter Software zu schaffen

3.8. ProductBacklog

Das ProductBacklog ist das Resultat des Workshops und enthält die priorisierten und geschätzten Userstorys. In den folgenden Sprints wird sie stetig aktualisiert und zur Organisation und Planung verwendet.

User Story	Größe
Must Have	
U18	3
U21	3
U32	3
U1	1
U2	3
U3	5
U31	3
U7	3
U5	5
U6	3
U35	5
U10	5
Should Have	
U4	1
U8	3
U33	5
U9	3
U17	5
U19	1
U20	1
Could Have	
U12	3
U13	3
U14	5
U15	5

U16	5
U23	1
U25	5

Tabelle 3.1.: Product Backlog mit priorisierten und geschätzten Userstorys

Abbildung 3.1.: Erstellte Userstorys aus dem 1. Anforderungsworkshop bereits Prioriserung nach MuSCoW

Abbildung 3.2.: Geladene Großhändler bei der Erstellung von Userstorys

4. Sprint Umgebung aufsetzen

4.1. Sprint-Planung

Bevor mit den eigentlichen Entwicklungsphasen begonnen wird, betrachten wir die Userstorys, also Anforderungen, der Entwickler. So wählen wir die Userstorys aus und bringen sie in folgende Reihenfolge:

- U30 Der Entwickler will ein Webframework, dass ihn bei der schnellen Umsetzung der Anforderungen hilft, um in wöchentlichen Schritten lauffähige vollintegrierte Versionen der Software zu veröffentlichen.
- U34 Der Entwickler will die akzeptanztestgetriebene Entwicklung einsetzten, um die Testbarkeit durch die Auftraggeber zu erhöhen.
- U28 Der Entwickler will ein integrierte Entwicklungsumgebung nutzen, um nicht lange nach Werkzeugen zu suchen und damit Zeit zu sparen.
- U26 Der Entwickler will eine Versionskontrolle, um mehr Sicherheit vor versehentliches Überschreiben von Quellcode zu erhalten.
- U27 Der Entwickler will einen automatisiertes Deployment, um schneller veröffentlichen zu können.
- U29 Der Entwickler will einen Produktivserver, um frühzeitig Releases zu veröffentlichen.

4.2. Installation und Einrichtung

Grundlegende Installation für die Umgebung

```
1 /$ sudo apt-get install build-essential libopenssl-ruby
2 libfcgi-dev
```

Install Ruby

```
/$ sudo apt-get install ruby1.8-dev ruby1.8 ri1.8 rdoc1.8 irb1.8
rubygems
/$ apt-get install libreadline-ruby1.8 libruby1.8 libopenssl-ruby
```

Install Rails

```
/$ gem install rails
```

Datenbank installieren

```
/$ apt-get install mysql-server mysql-client mysql-common
libmysql-ruby libmysqlclient-dev
```

Anlegen eines Datenbankbenutzers und der Datenbank

```
/$ mysql -u root -p GRANT ALL PRIVILEGES ON *.* TO
'flowertoweb'@'localhost' IDENTIFIED BY 'password' WITH
GRANT OPTION;
```

Install JavaScript Framework

```
/$ echo deb http://ftp.us.debian.org/debian/ sid main >
/etc/apt/sources.list.d/sid.list
/$ apt-get update
/$ apt-get install nodejs
```

Aufsetzen des Rails Projekts “flowerstoweb“. Anlegen des gesamten Projektstruktur und Installation der Abhängigkeiten wie Rails, mysql2, jquery und weitere . Datenbankuser RubyOnRails bekannt machen. Hierzu “config/database.yml“ bearbeiten.

```
/$ rails new flowerstoweb --skip-test-unit --database mysql
/$ rake db:create
```

Install nokogiri Umgebung(Voraussetzung für Cucumber)

```
/$ sudo apt-get install libxslt-dev libxml2-dev
```

Um Abhängigkeiten von Erweiterungen in eine RubyOnRails Anwendung zu erfüllen wird seit der Version 3.0 das Abhängigkeitsmanagment Bundle [BUNDLER 2012] verwendet. Angegeben werden die Abhängigkeiten ganz im Sinne des DRY Prinzips an einer zentralen Stelle, dem Gemfile. So wird für die Testumgebung cucumber [CUCUMBER 2012] folgende Gems in das gemfile eingetragen und die Abhängikeiten installiert. Im weiteren Verlauf dieser Arbeit wird nicht weiter auf die Erfüllung von Abhängikeiten eingegangen.

```
group :test do
  gem 'nokogiri'
  gem 'cucumber-rails'
```

```
  gem 'rspec-rails'
  gem 'database_cleaner'
end
```

Abhängigkeiten installieren

```
bundle install
```

Cucumber Projektdaten generieren

```
/$ rails generate cucumber:install
```

Installation und Einrichtung der Versionskontrolle für das Projekt.
Install Version Control:

```
/$ apt-get install git
```

Wechsel in das Projektverzeichnis. Inizialisieren des Git-Repositorys

```
/$ git init
```

Füge alle Dateien und Verzeichnisse dem Git Repository zu

```
/$ git add .
```

Erster initialier Commit in das lokale Repository:

```
/$ git commit -m "Initial commit"
```

Da Git ein verteiltes Versionsverwaltungssystem ist und im Gegensatz zu SVN keinen zentralen Server braucht, macht trotzdem der Einsatz eines zentralen Repositorys[1] Sinn. Es bietet eine zentrale Zugriffsmöglichkeit an, von der aus sich jeder Versionen holen und speichern kann. Das entfernte Repository wird auf einem Fileserver mit ssh Anbindung abgelegt, deren Zugriff in der ".git/config" Datei unter dem Namen "origin" gespeichert wird. Das Übertragen der lokalen Versionsstände auf das entfernte Repository kann dann über den folgenden Befehl ausgeführt werden.

```
/$ git push origin
```

Das Deployment mit Capistrano ist vergleichbar einfach und wird folgendermaßen eingerichtet: Installation von Capistrano im Projekt

```
/$ capify .
```

Unter anderem wurde hiermit die "deploy.rb" angelegt, die nun an die Umgebung des Servers auf den ausgeliefert wird, angepasst werden muss.

Einrichtung des Servers.

[1] Lager, Archiv

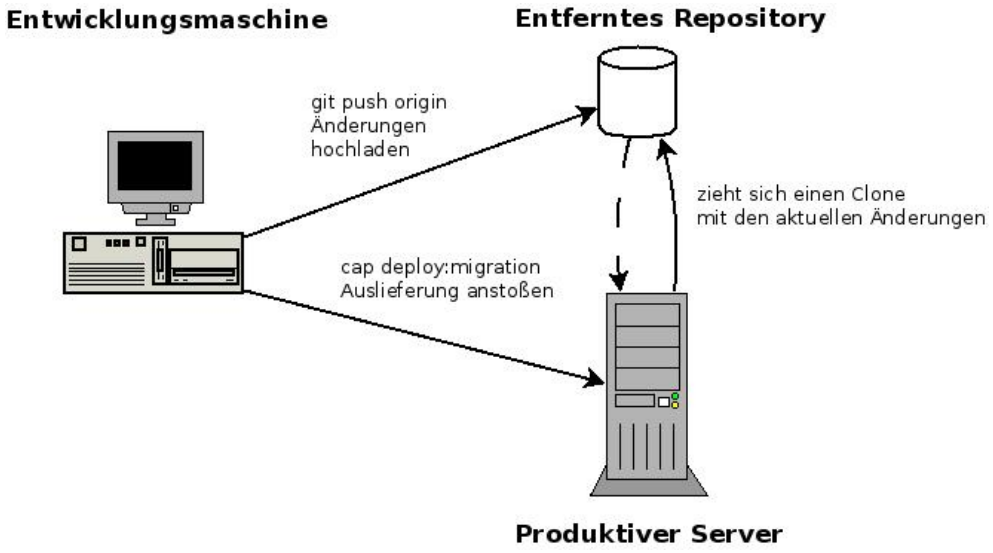


Abbildung 4.1.: Capistrano Deployment

```
/$ cap deploy:setup
```

Deployment anstoßen

```
/$ cap deploy:migrations
```

Siehe hierzu Abbildung 4.1

4.3. Review

Die Installation und Einrichtung der Entwicklungsumgebung lässt sich vollständig auf der Konsole durchführen. Es ist damit möglich diese Einrichtung leicht mittels eines Scripts zu automatisieren.

Capistrano kann weitausmehr als nur eine Kopie eines Versionsstands zu ziehen. Aufgaben können in Tasks abgelegt werden und je nach Kontext führt Capistrano sie aus. So wird für den produktiven Einsatz das Vorkompilieren der CSS und JS Dateien angestoßen.

```
task :precompile_assets, :roles => :web, :except => {
:no_release => true } do
    run "cd #{current_path}; rm -rf public/assets/*"
    run "cd #{current_path}; RAILS_ENV=production bundle
    exec rake assets:precompile"
end
```

5. Sprint 1 Artikel, Kunde,Bestellung, Rechnung

5.1. Sprint-Planung

Im Rahmen der Sprintplanung wird von einer Sprintdauer von 14 Tagen ausgegangen, die wiederum einer Velocity von 30 Punkten hat. Da es sich zum ersten Sprint um eine grobe Schätzung der Velocity handelt, wird im späteren Verlauf des Projekts entsprechende Korrekturen anhand der real bearbeitete Menge an Punkten vorgenommen.

Die ausgewählten Userstorys werden nun detailiert bearbeitet und offene Fragen auf der Storykarte dokumentiert. Es werden im Sinne des "Definition of Done"[1] Akzeptanzkriterien festgelegt, die als direkte Vorlage für die Akzeptanztest dienen. Die Userstory wird auf diese Weise in einem iterativen Prozess detailierter. Zu beachten ist hier, dass dies ein stetiger Prozess ist, der eben nicht wie bei UseCases, eine Abgeschlossenheit darstellen muss, bevor mit der Implementierung begonnen wird. Möglicherweise werden offene Fragen gar nicht erst beantwortet, sondern gleich in die Praxis mit umgesetzt und dem Kunden vorgeführt. Die Userstory ist somit erst abgeschlossen, sobald die vollständige Akzeptanz des Kunden vorhanden ist.

Es werden die ersten Userstorys aus dem Produktbacklog 8.1 genommen, dabei werden sinnvolle Implementierungsreihenfolgen mit berücksichtigt . Das führt dazu, dass die Userstory U33 und U14 mit in die erste Sprint einbezogen wird, obwohl sie geringere Prioritäten als andere haben.

[1] Fertigstellungskriterien

User Story	in Arbeit	Fertig
U18 Als Verkäufer will ich einen Kunden neu anlegen, um zukünftig für unter seinem Namen Rechnungen zu speichern.		
U21 Als Einkäufer will ich neue Artikel anlegen können, um ihren Verbleib nach-vollziehen zu können. U32 Als Verkäufer will ich eine Rechnung schreiben, um meine verkaufte Ware mit meinem Kunden abzurechnen.		
U3 Als Verkäufer will ich Bestellungen anlegen können, um sie später in eine Rechnung umzuwandeln.		
U7 Als Verkäufer will ich den Stückpreis und die verkaufte Menge für jede Rechnungsposition angeben können, um die Positionen den realen Vorgängen an zu passen.		
U31 Als Verkäufer will ich einen Artikel auswählen, um ihn in eine Bestellung auf zu nehmen.		
U33 Als Einkäufer will ich einzelnen Einkäufe eines Artikels unterscheiden können, um mir Preisentwicklung und Einkaufsmengen aus der Vergangenheit an zu sehen.		
U14 Als Kunde will ich eine Sammelrechnung bekommen, um nicht für jeden Einkauf einen Zahlungsvorgang starten zu müssen.		
U19 Als Verkäufer will ich nach Kunden suchen, um eine neue Bestellung an zu legen.		

Tabelle 5.1.: Sprint Backlog 1

5.2. ausgewählte Akzeptanzkriteren mit Akzeptanztest

Die Akzeptanzkriterien und deren Akzeptanztests spielen ein zentrale Rolle für die Entwicklung, deshalb werden hier einige exemplarisch dargestellt. Die Weiteren sind im Anhang zu finden.

Zu beachten ist hier, dass es sich bei den Tests um einen ausführbaren Programmcode von Cucumber handelt.

5.2.1. U21

- Ein Artikel hat nur einen aktuellen Artikelbestand. Dieser ist der jüngste in der Vergangenheit oder dem aktuellem Datum entsprechende Artikelbestand.

```
Szenario: Ein Artikel hat nur einen aktuellen Artikelbestand.
Dieser ist der jüngste in der Vergangenheit oder dem
aktuellem Datum entsprechende Artikelbestand.
Angenommen man ist Angemeldet als "admin"
Und es gibt den Artikel "Avalanche+"
Und der Lieferant "powerflowers" ist gespeichert
Wenn für Artikel: "Avalanche+" ein Artikelbestand
Lieferant: "flowerpower", Abverkauf heute vor "3" Tagen
angelegt
Und für Artikel: "Avalanche+" ein Artikelbestand
Lieferant: "flowerpower", Abverkauf heute in "0" Tagen
angelegt
Dann bekomme ich für den Artikel "Avalanche+" den
Artikelbestand mit dem Abverkaufsdatum heute in "0" Tage
```

5.2.2. U18

- Der Kunde soll automatisch vom System eine eindeutige 5 Stellige Kundennr bekommen.

```
Szenario: Der Kunde soll automatisch vom
System eine eindeutige 5 Stellige Kundennr bekommen.
Angenommen man ist Angemeldet als "admin"
Und der Verkäufer geht zur Seite neuen Kunde anlegen
Wenn er den "Firmenname" "Blumen am
Schlossteich GbR" eingebe
Und er den "Vorname" "Ursel" eingebe
Und er den "Nachname" "Meier" eingebe
Und er die "Straße" "Am Schlossteich 43" eingebe
Und er die "Plz" "22546" eingebe
Und er die "Ort" "Reinbeck" eingebe
Und auf Speichern klicke
Dann wurde der Kunde "Blumen am Schlossteich
GbR" erfolgreich gespeichert
Und bekommt er den Kunden "Blumen am Schlossteich GbR"
mit einer "5" Stelligen eindeutigen "K.Nr" angezeigt.
```

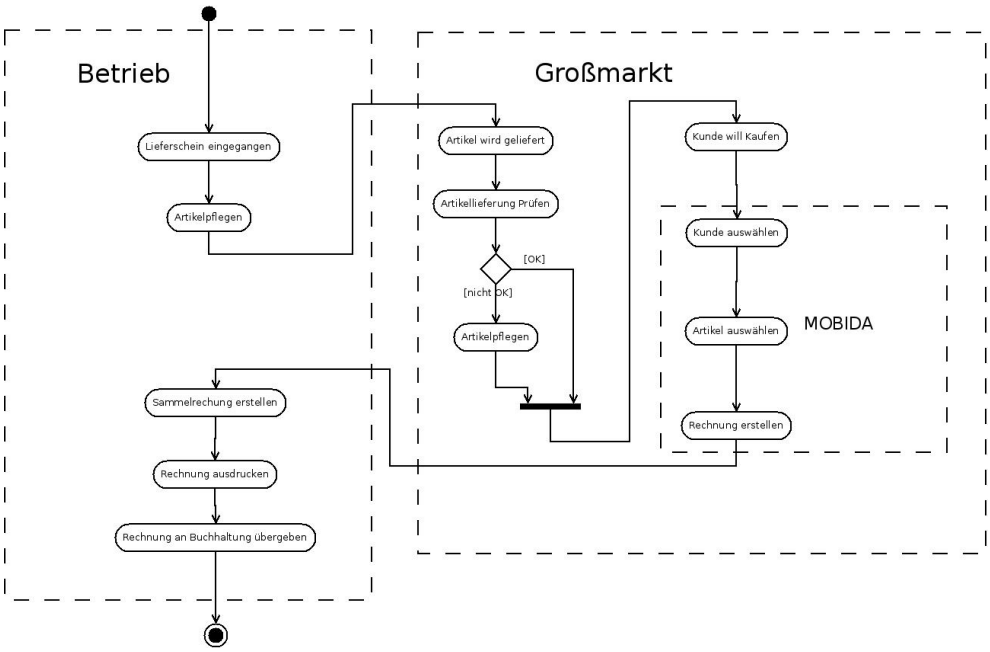


Abbildung 5.1.: Aktivitätsdiagramm Artikel und Rechung

5.3. Design und Implementierung

Die Integrität und Konsistenz der enthalten Informationen ist eine wichtige Eigenschaft, die eine betrieblichen Software, die zur Abrechnung genutzt wird, erfüllen muss. Um diese Eigenschaft zu erreichen, wird ein hoher Grad der Normalisierung gewählt. Die Berechnung von Informationen über die Assoziationen zwischen den Klassen wird gegenüber einer redundanten Datenhaltung klar bevorzugt. Abweichungen von diesem Konzept kommen nur an Stellen vor an denen eine Redundanz gewünscht ist, wie zum Beispiel einer Rechnungsadresse. Die Datenstruktur wird in RubyOnRails über ein ObjectRelationalesMapping, ActionRecord realisiert und hier verwendet. (siehe Abb. 5.6)

5.3.1. Artikel

Artikel werden im System aus den Klassen Artikel(article) und Artikelbestand(inventory) bestehen. Der Artikel beinhaltet die Informationen, die für alle Einkäufe und Verkäufe dieses Artikels gelten. Der Artikelbestand beinhaltet dagegen die Informationen, die für einen konkreten Einkauf des Artikels gelten. Ein Artikel hat damit mehrere Artikelbestände, die sich in Menge und Einkaufspreis unterscheiden können. Der Verkauf dieses Artikels wirkt sich jedoch

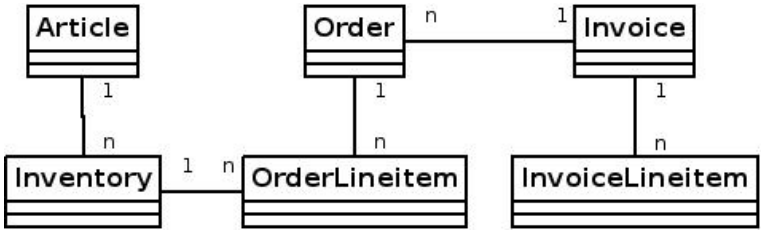


Abbildung 5.2.: Assoziation Artikel,Bestellung und Rechnung

nur auf den Artikelbestand mit dem Neusten in der Vergangenheit liegenden Abverkaufsdatum aus. Dieser wird in Zukunft als aktueller Artikelbestand bezeichnet.

```
def current_inventory
     Inventory.where("article_id = #{self.id}").order(
     "sales_date DESC").each do |inventory|
      if inventory.sales_date <= Time.now
        inv = inventory
        return inv
      end
    end
  end
```

Eine mit dem Kunden abgesprochene Besonderheit ist, dass ein Artikelbestand, der durch einen Neuen ersetzt wird, als verdorben gilt.

5.3.2. Bestellungen

Bestellungen stellen im System die Verbindung zwischen Kunden und Artikeln da. Sie setzt sich aus den Klassen Bestellung(order) und Position(lineitem) zusammen. Die Bestellung stellt alle Informationen bereit, die für die gesamt Bestellung relevant sind. Die Positionen stellen dagegen die Informationen zu einem bestellten Artikel bereit. Sobald ein Artikel als Position aufgenommen ist, hat dies eine Auswirkung auf den aktuellen Bestand des Artikels. Wird eine Position gelöscht, fällt diese Auswirkung automatisch weg. (siehe Abb. 5.2)

Eine Bestellung ohne Referenz auf eine Rechnung gilt als offen und kann jederzeit verändert oder gelöscht werden. Wird eine Rechnung erstellt, kann eine Bestellung nicht mehr geändert oder gelöscht, werden und gilt als abgerechnet. Die Existenz der Referenz gilt im System also als hinreichende Bedingung für ihr bestehen und gewährleistet automatisch das eine Bestellung nur einmal abgerechnet werden kann.

5.3.3. Rechnungen

Eine Rechnung wird im System durch die Klassen Rechnung(Invoice) und Rechnungspositionen(InvoiceLineitem) dargestellt. Sie stellt in erster Linie alle durch den Gesetzgeber geforderten Informationen aus dem Kapitel2.6.2 bereit. Eine Rechnung kann eine aber auch mehrere Bestellungen gleichzeitig abrechnen. Hierfür wird für einen Kunden das Erstellen von einer Rechnung gestartet und das System übernimmt automatisch alle nicht abgerechneten Bestellungen des Kunden in die Rechnung auf.(siehe Abb. 5.3)

Sobald eine Rechnung durch das Ausdrucken oder dem Emailversand einem Kunden zugestellt wird, gilt sie als verbucht. Die Existenz eines Rechnungsdatums wird dabei als eine hinreichende Bedingung angesehen, das eine Rechnung verbucht ist. Somit ist eine Rechnung ohne Rechnungsdatum nicht verbucht. Eine verbuchte Rechnung kann nicht mehr geändert werden.

```
1  class InvoicesController < ApplicationController
2  ...
3  def update
4      @invoice = Invoice.find(params[:id])
5      respond_to do |format|
6        if @invoice.entered?
7          format.html { redirect_to @invoice, :notice => Rechnung
8          ist bereits verbucht. }
9        else
10         if @invoice.update_attributes(params[:invoice])
11         ...
12 end
13 ...
14 end
```

Die Inhalte einer Rechnung müssen persistent im System gespeichert werden. Wenn sich die Stammdaten, aus denen sie erstellt wurden, geändert werden, darf dies keinen Einfluss auf sie haben. An dieser Stelle wird also mit einer bewussten Redundanz gearbeitet. Folgende Informationen werden damit in einer Rechnung gespeichert: Firmenname, Vor- und Nachname, Straße, Plz, Ort, Rechnungsnummer, Rechnungsdatum, Menge, Artikelbezeichnung, Umsatzsteuer-Satz, Netto Stückpreis. Die fortlaufende Rechnungnummer wird über die Datenbankeinträge generiert. So wird anhand des größten Rechnungsnummer die nächst größere Gewählt.

```
1  class Invoice < ActiveRecord::Base
2  ...
```

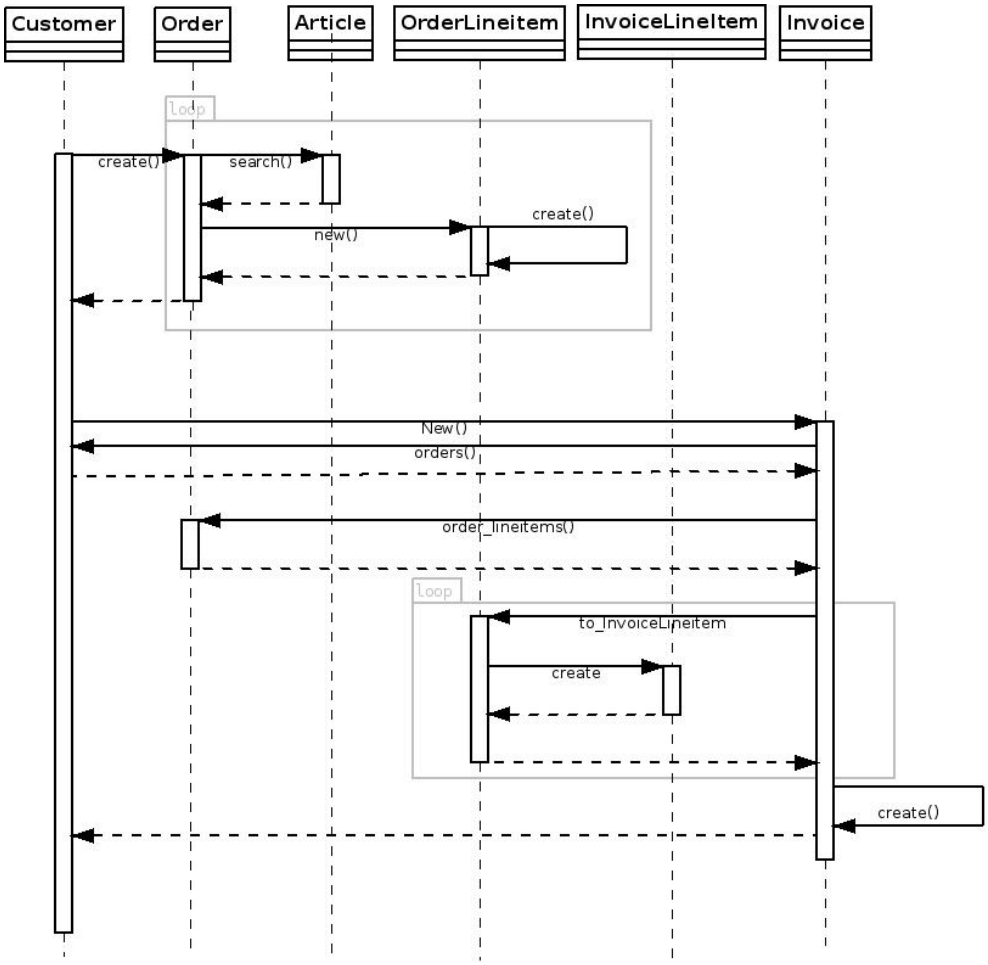


Abbildung 5.3.: Sequenzdiagramm Bestellung und Rechnung erstellen

```
def self.next_number
   current_number=Invoice.maximum(:number)
   if current_number
                 current_number+= 1
   else
        10000
   end
end
...
end
```

Rechnungen und Bestellungenpositionen werden über eine gemeinsame Oberklasse Positionen implementiert, da sie sowohl identische Attribute als auch identische Berechnungen haben. Eine Bestellung beinhaltet damit die inhaltlich gleichen Positionen, wie die dazugehörige Rechnung. In dem Fall das eine Rechnung storniert wird, kann die dazugehörige Bestellung aber wieder verändert werden. Die Positionen der Rechnung müssen dabei aber bestehen bleiben. So ergibt sich die Notwendigkeit einer Redundanz der Positionen.

5.3.4. Stornierung

Muss eine verbuchte Rechnung geändert werden, steht nur die Möglichkeit der Stornierung zur Verfügung. Wird die Rechnung also storniert, so behält sie ihre Rechnungsnummer und alle anderen Inhalte bei. Die Assoziationen zu den Bestellungen, die durch diese Rechnung abgerechnet wurden, werden gelöst und die Bestellungen sind automatisch wieder offen und können bearbeitet und neu abgerechnet werden.

```
class Order < ActiveRecord::Base
...
def cleared?
    if invoice
      true
    else
      false
    end
  end
...
end
```

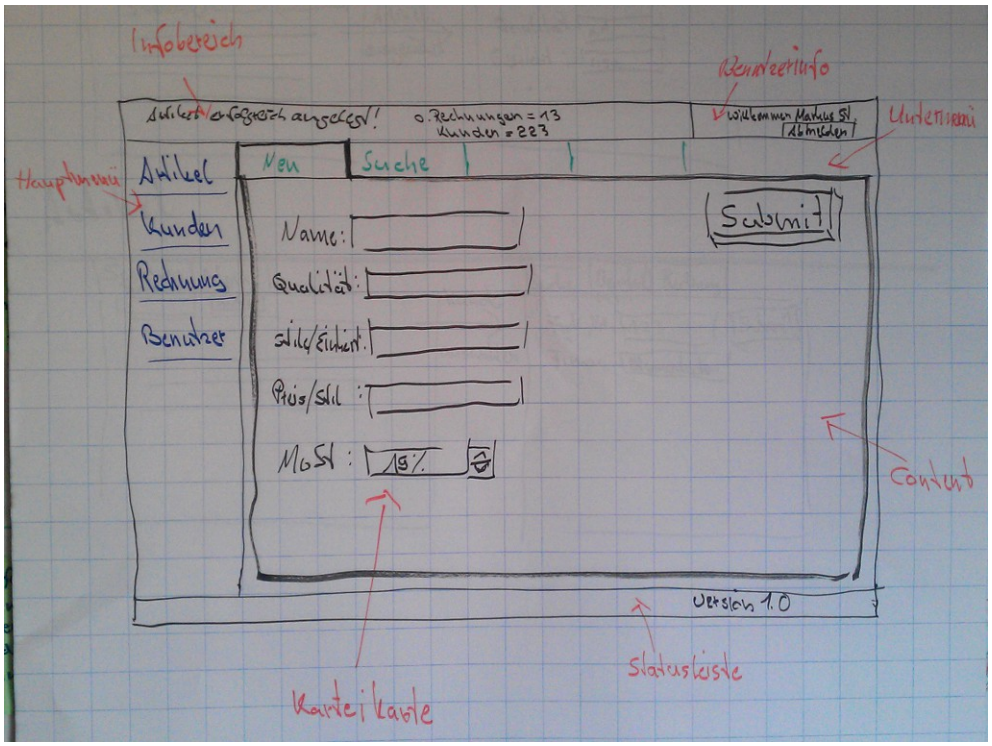


Abbildung 5.4.: Entwurf CSS Template

5.3.5. FrontEnd

Um dem Kunden zum Abschluss dieser Sprints nicht nur die erfolgreich durchlaufenden Akzeptanztests zeigen zu können, wird das Bedienoberfläche, in Form eines CSS Layouts eingeführt. Das Design soll die Bedienbarkeit und Struktur der Anwendung unterstützen und wurde bei einem kurzen Meeting mit den Händlern an der Tafel erörtert. (siehe Abb. 5.4 und 5.5)

5.4. Review

Nach erfolgreicher Abnahme über die Akzeptanztests und dem ausführlichen Testen der ersten Anwendungsfunktionen über das Userinterface ergeben sich einige Anpassungswünsche. Dem Kunden ist aufgefallen, dass die Bedienung der Software leichter fällt, wenn nur die Informationen anzeigt werden, die an entsprechender Stelle sinnvoll sind. Eine konkrete Änderung ist dabei die Funktion zum Hinzufügen eines Artikels zur Bestellung in den Vordergrund zu

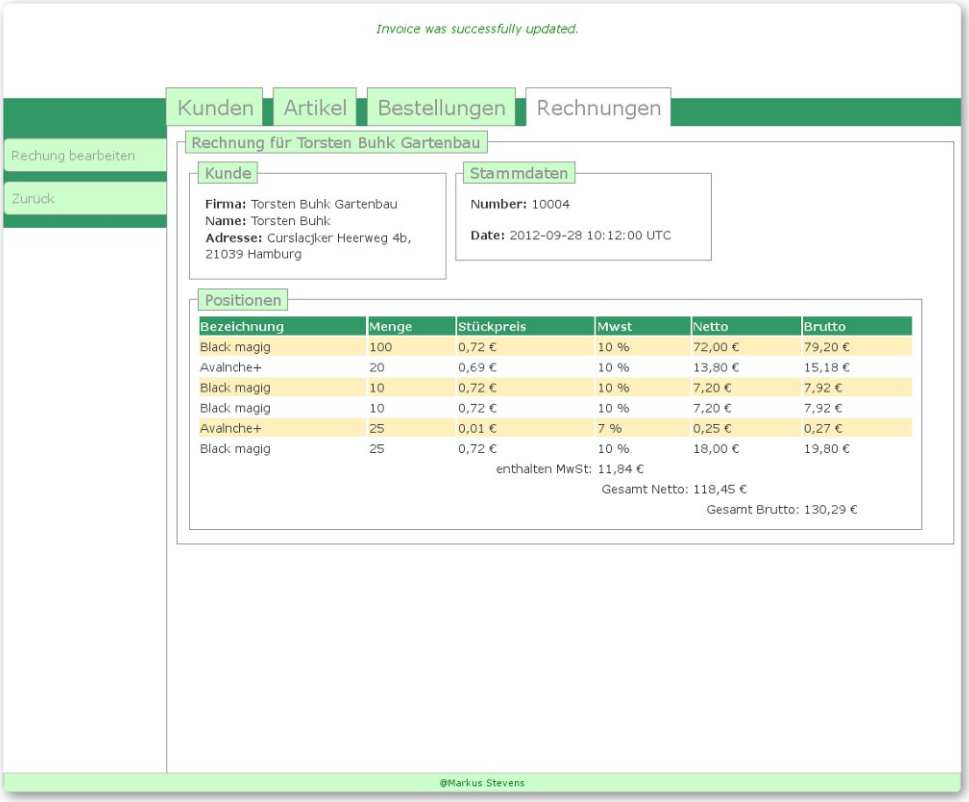


Abbildung 5.5.: CSS Grundlayout

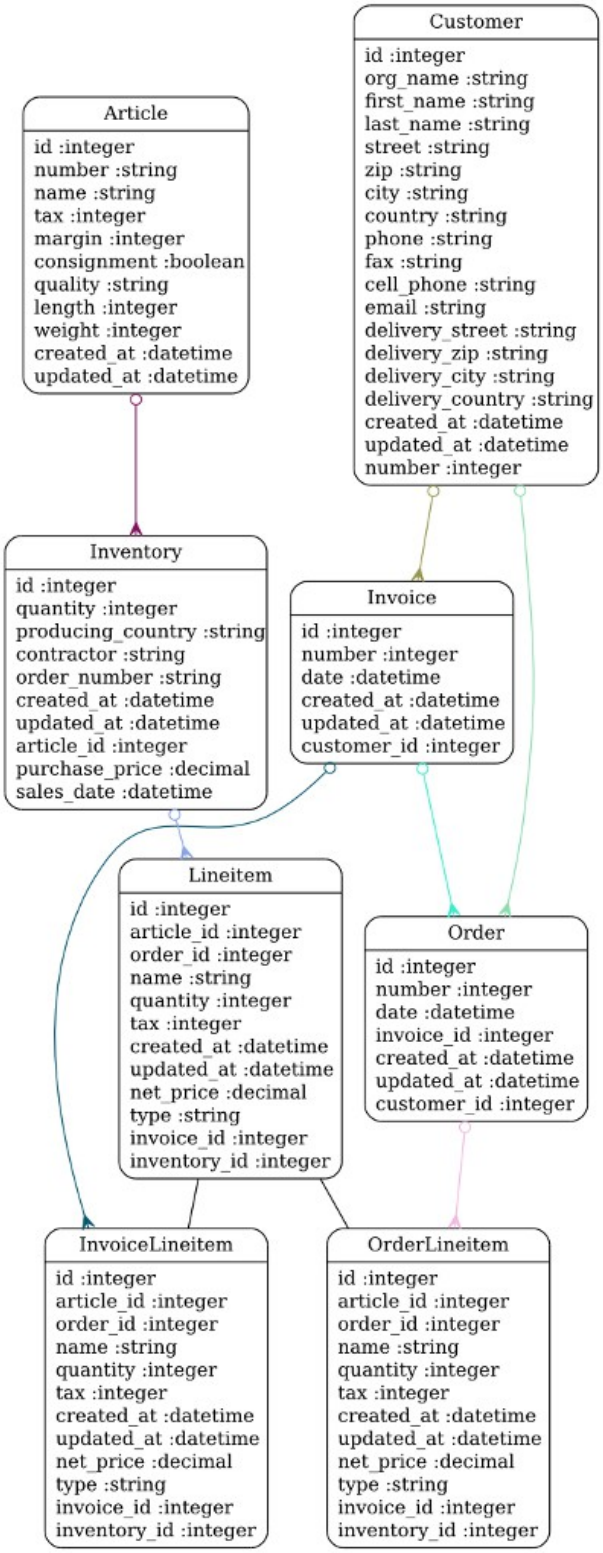


Abbildung 5.6.: Domain Model 1

stellen und damit Fehlbedienung entgegen zu wirken. Standardwerte für Umsatzsteuer und Gewinnspanne erhöhen zudem die Geschwindigkeit bei der Eingabe. Für die Planung des nächsten Sprints wurde die Priorität von Gutschriften und Pfandartikeln vom Kunden erhöht.

6. Sprint 2 Drucken, Pfand, Gutschrift, Skonto, Kommission

6.1. Sprint-Planung

Nachdem die ersten Userstorys realisiert und mittels Akzeptanztests durch die Kunden abgenommen wurden, werden im zweiten Sprint die nächsten Userstory aus dem Product-Backlog 8.1 genommen. Dabei werden die Erfahrungen aus dem ersten Sprint genutzt, um die Velocity zu optimieren. Die Priorität der Userstorys hat sich für den Kunden nochmals geändert und die ersten Erfahrungen beim Schreiben von Test und der Implementierung helfen dabei die richtigen Userstorys auszuwählen. Es hat sich eine neue Userstory ergeben U36, die direkt umgesetzt werden soll, da sie eine Voraussetzung für hier ausgewählte Userstorys darstellt.

Die Velocity wird jetzt mit 25 angenommen und die Dauer des Sprints liegt wiederum bei 14 Tagen.

User Story	in Arbeit	Fertig
U1Als Kunde will ich eine Rechnung mit meiner Adresse und einer Rechnungsnummer, um den Einkauf gegenüber dem Finanzamt belegen zu können.		
U36 Als Einkäufer will ich Lieferanten anlegen können, um meine Einkaufszahlen gegenüber dem Lieferanten sehen zu können.		
U5Als Einkäufer will ich Umsätze von Kommissionsware angezeigt bekommen, um eine Abrechnung mit dem Lieferanten machen zu können.		
U6Als Verkäufer will ich auf eine Rechnung Skonti geben können, um gute Kunden zu binden.		
U9Als Buchhalter will ich den Gewinn einzelner Einkäufe gegenüber den Verkäufen sehen, um die Rentabilität der Unternehmung zu sehen.		

U10	Als Verkäufer will ich dem Kunden eine Gutschrift geben können, um zurück gegebenes Pfandgut verrechnen zu können.
U35	Als Verkäufer will ich wissen welcher Kunde meine Pfandartikel hat, um sie abrechnen zu können.

Tabelle 6.1.: Sprint Backlog 2

6.2. ausgewählte Akzeptanzkriterien mit Akzeptanztest

weitere Akzeptanzkriterien sind im Anhang angegeben.

6.2.1. U1

- Der Kunde erwartet seine Anschrift, eine Liste aller gekauften Artikel, einen Endbetrag und die enthaltene Umsatzsteuer auf seiner Rechnung.

```
Szenario: Der Kunde erwartet seine Anschrift, eine Liste
aller gekauften
Artikel, einen Endbetrag und die enthaltene MwSt auf seiner
Rechnung.
Angenommen man ist Angemeldet als "admin"
Und es gibt den Kunden Firma "Blumenmeyer" "Ursel" "Meyer"
"Meyerstraße 123" "21039" "Hamburg"
Und der Artikel "Sexy Red" hat eine Gewinnspanne "30" ,
eine MwSt. "7" und einen akt. Artikelbestand mit einem EK
von "0.50"
Wenn der Verkäufer auf die Liste der Kunden auf "Blumenmeyer"
klickt
Und klickt auf den Link "Neue Bestellung"
Und kommt er auf die Seite "Artikel für Blumenmeyer"
Und klickt auf den Link "Sexy Red"
Und gibt die Menge "120" und den Preis "0.22" ein
Und klickt auf den Button "Speichern und fertig"
Und kommt er auf die Seite "Bestellung für Blumenmeyer"
Und klickt auf den Link "Rechnung erstellen"
Und kommt er auf die Seite "Rechnung für Blumenmeyer"
Und klickt auf den Button "Drucken und Verbuchen"
Und klickt auf "Rechnung drucken"
Dann gibt das System eine PDF Datei aus mit der Adresse
"Blumenmeyer", "Meyerstraße 123", "21039", "Hamburg"
```

```
Und dem Artikel "Sexy Red", "120" ,"0,22"
Und den Betraegen enthaltene MwSt: "1,85" Gesamt Brutto:
"28,25"
```

6.2.2. U5

- Die Kosten berechnen sich dabei Einkaufspreis * umgesetzte Menge.

```
Szenario: Im Lieferanten werden die VK-Menge und die Kosten
eines Einkaufs angezeigt.
Angenommen man ist Angemeldet als "admin"
Und der Artikel "Avalache+" ist gespeichert.
Und der Lieferant "powerflowers" ist gespeichert
Und es gibt den Kunden "Blumenmeyer"
Und ein Artikelbestand in Kommission ist mit Menge "200",
EK-Preis "0.45" ist gespeichert
Wenn ein Kunde "100" Artikel "Avalanche+" kauft
Und der Einkäufer die Seite des Lieferanten
"powerflowers" geht
Dann wird im eine VK-Menge "100" und Kosten von "45,00"
angezeigt
```

6.3. Design und Implementierung

6.3.1. Drucken

Das System soll das Erstellen von Dokumenten ermöglichen, die sich sowohl ausdrucken aber auch langfristig speichern lassen sollen. Eine ausgedruckte oder als digitales Dokument zugestellte Rechnung entspricht einem rechtsgültigen Beleg. Um die Kompatibilität und eine langfristige Aufbewahrung von mindestens 10 Jahren zu gewährleisten, wird unter anderem das PDF/A-Format nach ISO 19005-1:2005 empfohlen [PDF/A Sachsen 2009][Seite 7]. Um grundsätzlich diesen Empfehlungen nachzukommen, wird das Drucken von Rechnung und Lieferschein über das Erstellen von PDF Dateien realisiert. Ob ein PDF Dokument dem ISO Standard entspricht, kann mit verschiedene Validierungstools überprüft werden, [Online PDF/A Validierung].

Fachlich hat das Erstellen von rechtsgültigen Belegen zur Folge, dass die Objekte aus denen sie erstellt wurden, unveränderlich im System gespeichert werden müssen. Sie dürfen also nicht verändert oder gelöscht werden. Dieser Vorgang wird hier als verbuchen bezeichnet.

Für das Erstellen von PDF Dateien wird die Bibliothek Prawn [PRAWN PDF] verwendet. Prawn ist eine Implementierung in Ruby und wird als Erweiterung für RubyOnRails angeboten. Prawn arbeitet mit einer eigenen Auszeichnungssyntax für ein flexibles aber dennoch einfaches formatieren. Die Prawnklasse eines PDF Dokuments beinhaltet damit sowohl syntaktische Bestandteile zur Formatierung als auch reines Ruby für die Logik und den dynamischen Inhalten.

Ein Druckvorgang auf dem Großmarkt muss sehr schnell gehen und würde über einen PDF-Download zu viel Zeit kosten. Um dieses Problem zu umgehen, soll es möglich sein, dass der Anwendungsserver einen Druckvorgang selber startet, anstatt dem Client die PDF zu schicken. Ein weiterer Vorteil dieser Methode ist, dass auf den Clients kein Drucker eingerichtet werden müssen.

```
class InvoicesController < ApplicationController
...
def print
    @invoice = Invoice.find(params[:invoice_id])
    temp_pdf = Tempfile.new('pdf')
    temp_pdf << InvoicePdf.new(@invoice, view_context).render
    temp_pdf.close
    `lpr -P kyocera -o media=A4 #{temp_pdf.path}`
    redirect_to invoice_path(@invoice), :notice =>
    'Rechnung wird gedruckt'
end
...
end
```

6.3.2. Kommissionsware

Der Einkauf von Kommissionsware muss im System gesondert behandelt werden. Zum einen unterscheidet sich die Berechnung des Gewinns, zum anderen wird die verkaufte Menge für die Abrechnung mit dem Lieferanten verwendet. Um einen Einkauf einem Lieferanten zuordnen zu können, wird dem Domain-Model die Klasse Lieferant(contractor) hinzugefügt.

Im ersten Sprint wurde die Information, ob es sich um eine Kommissionsware handelt, im Artikel gespeichert. Diese Entscheidung wird ab hier revidiert und dem Artikelbestand zugeordnet. Der Grund hierfür liegt in dem Ziel einem Artikel sowohl Kommissionsware als auch normale Wareneinkäufe zuordnen zu können. Die Alternative hierzu, wäre das Anlegen verschiedener Artikelbestände für das gleiche Produkt. Diese Variante würde eine

Bestandskontrolle und Auswertung von Artikeln jedoch erschweren und den selbst zugrunde gelegten Prinzipien der Redundanzfreiheit widersprechen.

```
class Inventory < ActiveRecord::Base
...
def cost_amount
   if self.sale_or_return
     Order.sales_volume(self)*purchase_price
   else
     quantity*purchase_price
   end
end
...
end
```

6.3.3. Skonto

Der Skonto ist ein Abzug von den Beträgen einer Rechnung anhand eines Prozentsatzes. Er hat das Ziel, die Kunden zu belohnen, die einer Forderung unmittelbar oder binnen eines festgelegten Zeitraumes nachkommen.

Im System wird ausschließlich vom Nettobeträgen abgezogen, so spricht man im allgemeinen vom Netto-Skonto. Um die nötigen Funktionalitäten zu realisieren, wird der Klasse Rechnung(Invoice) ein Skontosatz hinzugefügt. Die Analyse der Anwendungsdomäne hat ergeben, dass sich der Skontosatz zwischen den Kunden unterscheidet aber für ein und den selben Kunden oft gleich bleibt. So wird der Klasse Kunde(Customer) ebenso ein Skontosatz hinzugefügt, der als Vorlage für die Rechnung verwendet wird. Existiert kein Skontosatz, so wird er auf 0 gesetzt.

Die Berechnung des Skontoabzugs wird über die einzelnen Positionen einer Rechnung durchgeführt. So ist es möglich bestimmte Artikel vom Skontoabzug auszuschließen. Die Skontofähigkeit der Artikel wird in der Klasse Artikel(article) festgelegt. Auf einer Rechnung werden die Skontoabzüge sowohl in den einzelnen Positionen als auch als Summe für die gesamte Rechnung angezeigt.

6.3.4. Pfandgut

Das Pfandgut unterliegt einem Kreislauf, der beim Lieferanten beginnt und endet. Für die Übergabe von Pfandgut an einen Käufer wird in den meisten Fällen ein Pfandgeld entgegen gerechnet. Die Userstorys U10 und U35 bilden die Grundlage für die Anforderungen. Um die

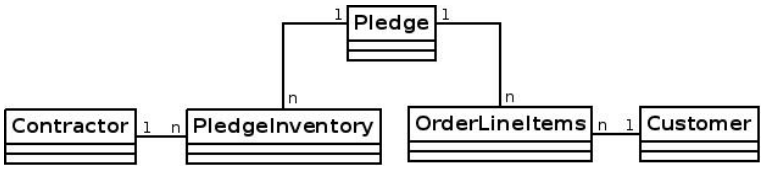


Abbildung 6.1.: Assoziationen Pfandgut, Lieferant und Kunde

neuen Funktionalitäten zu realisieren, wird die Klasse Pfandgut(pledge) eingeführt. Da ein Pfandgut ebenso wie ein Artikel einer Bestellung hinzugefügt werden soll und eine vergleichbare Struktur aufweist, wird eine Oberklasse Produkt(Product) eingeführt. Sie beinhaltet die gemeinsamen Eigenschaften und vererbt sie an Artikel und Pfandgut. Die Vererbung erlaubt es, das Pfandgut gesondert zu behandelt und trotzdem in einem gemeinsamen Bestellprozess einzubinden. Um das Verbleiben von Pfandgut beim Kunden als auch beim Lieferanten nachzuvollziehen, wird der Begriff Pfandkonto eingeführt. Die schon vorhandene Datenstruktur bietet uns bereits alle nötigen Informationen dazu. Die Buchungen eines Pfandkontos ergeben sich beim Lieferanten aus den bei ihm eingekauften Beständen und bei dem Kunden aus seinen Bestellpositionen,(siehe Abb. 6.1).

```
class Pledge < Product
...
def current_account_balance(localization)
    if localization.class.name=="Contractor"
        pledge_inventories.where(
        "#{localization.class.name.downcase}_id =
        #{localization.id}").sum("quantity")
    elsif localization.class.name=="Customer"
        OrderLineitem.joins(:order).where(
        "customer_id = #{localization.id} and product_id =
        #{self.id}").sum("quantity")
    else
        pledge_inventories.sum("quantity")
    end
end
...
end
```

6.3.5. Gutschrift

Eine Gutschrift wird durch das Angeben von negativen Mengen in einer Bestellposition ermöglicht. Da die Berechnung der Beträge eine Multiplikation zwischen Menge und Preis ist, wird bei einer negativen Menge der Betrag automatisch negativ und stellt damit eine Gutschrift auf der Bestellung da. Um eine Validierung von sowohl positiven als auch negativen Mengen zu ermöglichen, wird eine redundante Information gespeichert, die angibt ob es sich um eine Gutschrift handelt.

6.3.6. Gewinn

Der Gewinn ist ein Betrag, der sich aus dem Umsatz abzüglich der Kosten für den Einkauf ergibt. Diese Berechnung ist sehr einfach, eine praktische Umsetzung jedoch nicht. Zu beachten sei hier, dass jeder Einkauf und jeder Verkauf eines Artikels zu unterschiedlichen Preisen stattfinden kann. Die Datenstruktur ist genau für diesen Fall gewählt worden. Ein Gewinn wird über die in der Datenbank enthaltenen Einträge errechnet. Der Rechenaufwand wird hier, zugunsten der Datenkonsistents, einer redundanten Datenhaltung bevorzugt.

```
1  class Order < ActiveRecord::Base
2  ...
3    def self.turnover(inventory)
4      amount= 0
5      OrderLineitem.where("inventory_id = #{inventory.id}").each do
6      |lineitem|
7        amount+= lineitem.net_amount
8      end
9      amount
10   end
11 ...
12 end
13
14 class Inventory < ActiveRecord::Base
15 ...
16   def profit
17       Order.turnover(self)-cost_amount
18   end
19 ...
20 end
```

6.3.7. Model

Das Domainmodel hat sich damit wie in Abbildung 6.2 geändert.

6.4. Review

Im Zusammenhang mit der Einführung von Kommissionsware hat sich nach Rücksprache mit dem Kunden herausgestellt, dass sich der Umsatzsteuersatz zwischen dem Einkauf und dem Verkauf unterscheiden kann. Dieser Sachverhalt wurde vor Beginn des Sprints nicht beachtet. Das agile Vorgehen ermöglicht es nun diese Änderungen direkt in Akzeptanzkriterien umzusetzen und sie im nächsten Sprint mit einfließen zu lassen.

In einem kurzen Praxistest hat sich gezeigt, dass das Verbuchen von Einkäufen zu zeitaufwändig ist. Nun wurde in Zusammenarbeit mit den Kunden herausgefunden, dass die Informationen zum Verbuchen immer aus der Rechnung des Lieferanten entnommen werden. Das bedeutet das System sollte diesbezüglich optimiert werden. Um die neue Anforderung im nächsten Sprint mit zu berücksichtigen, wird eine neue Userstory erstellt.

- U37 Der Einkäufer will die Rechnung des Lieferanten in einem Zug eingeben, um nur einmal den Lieferanten auswählen zu müssen.

Ein weitere Verbesserung könnte sein, den Fokus auf die relevanten Formularfelder zu legen, sobald eine Seite geöffnet wird.

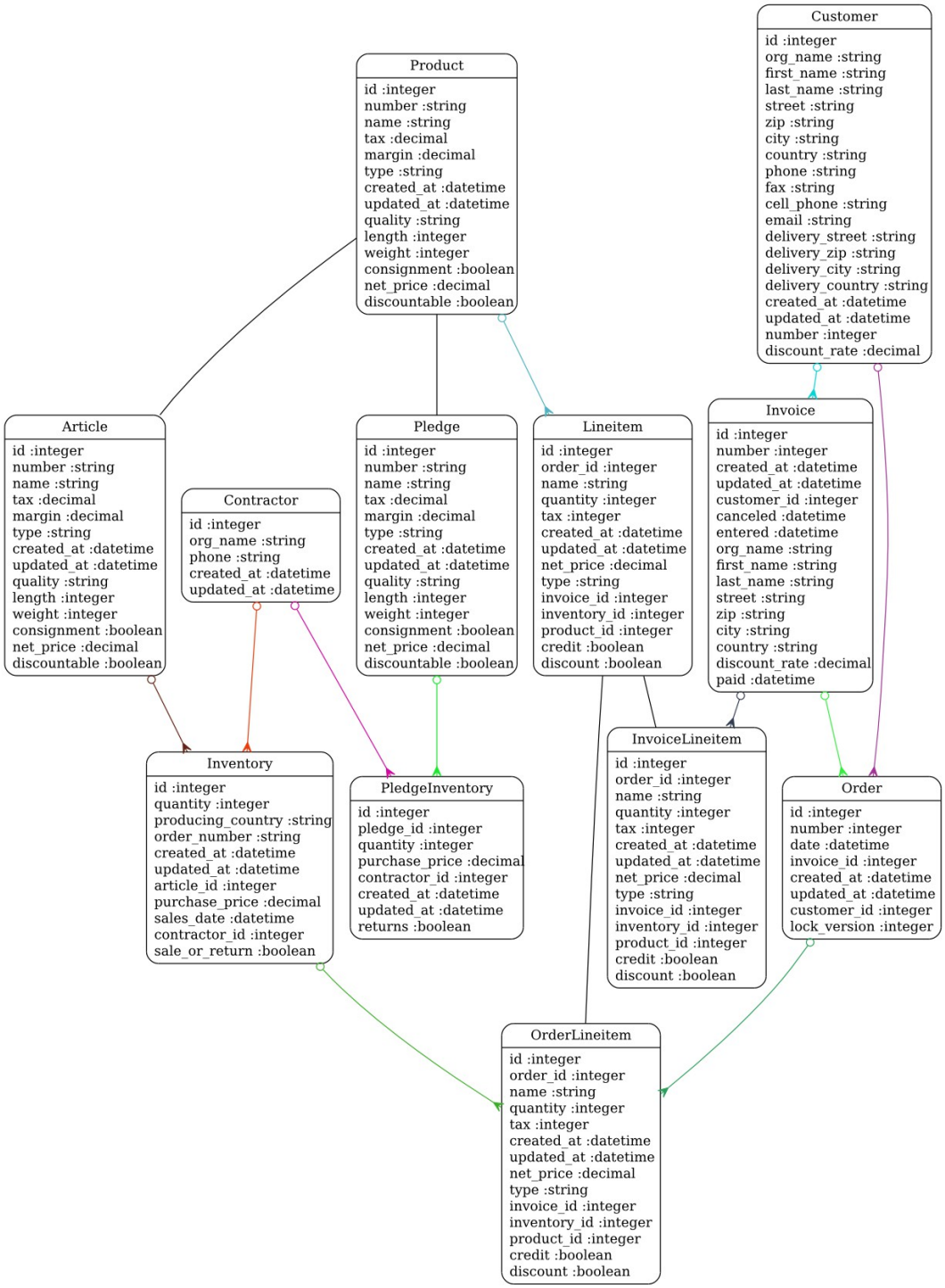


Abbildung 6.2.: Domain Model 2

7. Sprint 3 Suche, Lieferschein, Artikelbestandspflege, Autofocus, Authentifizierung

7.1. Sprint-Planung

User Story	in Arbeit	Fertig
U12 Als Kunde will ich einen Lieferschein ohne Preise, da der Abholer die Preise nicht kennen soll.		
U13 Als Verkäufer will ich einen Lieferschein drucken können, um die Ware zusammen zu stellen.		
U37 Der Einkäufer will die Rechnung des Lieferanten in einem Zug eingeben, um nur einmal den Lieferanten auswählen zu müssen.		
U25 Als Verkäufer möchte ich bei der Suche nach Artikeln, simultan zur Eingabe, Suchtreffer angezeigt bekommen, um schneller Artikel zu finden.		
U23Als Verkäufer möchte ich nach Artikeln suchen, um Anfragen nach Preis oder Verfügbarkeit beantworten zu können.		
A13 Ein unautorisierter Zugriff auf die Applikation muss verhindert werden.		

Tabelle 7.1.: Sprint Backlog 3

7.2. ausgewählte Akzeptanzkriterien mit Akzeptanztest

7.3. Design und Implementierung

7.3.1. Lieferschein

Ein Lieferschein stellt keine neuen Informationen bereit. Er ist ein weiteres druckfähiges Dokument, dass sich aus den bereits bestehende Klassen erstellen lässt. Hierfür wird wiederum Prawn [PRAWN PDF] zum erstellen von PDF Dateien verwendet.

Die einzige fachliche Konsequenz aus der Existenz eines Lieferscheins ist, dass die Bestellung zu dem der Lieferschein gehört als verbucht gilt. Eine Bestellung kann also nur dann bearbeitet oder gelöscht werden, wenn sie weder eine Rechnung noch einen Lieferschein besitzt. Um diese Anforderung zu erfüllen, wird ein weiteres Datum (DeliveryNote) erstellt, dass eine hinreichende Bedingung für die Existenz eines bereits gedruckten Lieferschein darstellt.

7.3.2. optimierung Artikelbestandspflege

Um die Artikelbestände schneller pflegen zu können, wird das System den Arbeitsabläufen besser angepasst. Der Händler wählt zuerst seinen Lieferanten aus und verbucht solange für einen Lieferanten bis die Eingangsrechnung vollständig verbucht ist,(siehe Abb. 7.1). Im Sprint 1 wurde für jede Buchung der Lieferant ausgewählt.

7.3.3. optimierung der Suche mit AJAX

Um dies Suche zu optimieren soll das System, noch während der Eingabe von Suchbegriffen, Ergebnisse liefern. Siehe dazu Userstory U25. Um dies zu ermöglichen, muss ein Tastendruck eine Anfrage an den Server auslösen. Da das vollständige Laden der gesamten Seite eine zu große Verzögerung zu Folge hätte, wird hier AJAX[1] eingesetzt. Diese Technologie ermöglicht das Laden von einzelnen Elementen auf eine Seite. So wird bei jedem lösen einer Taste eine XMLHttpRequest ausgelöst und mittels Javascript die neuen Suchergebnisse in die Seite integriert. siehe Abb. 7.2. Verwendung findet diese sogenannte “Livesearch“ im Bereich der Artikelsuche, der Kundensuche sowie bei der Erstellung von Bestellungen.

```
1 <%= form_tag articles_path, :method => 'get', :remote=> true do %>
2 <p>
3   <%= text_field_tag :search, params[:search], :autofocus=>:true,
4   :autocomplete=> 'off', :onkeyup=>'livesearch()' %>
```

[1] Asynchronous JavaScript and XML

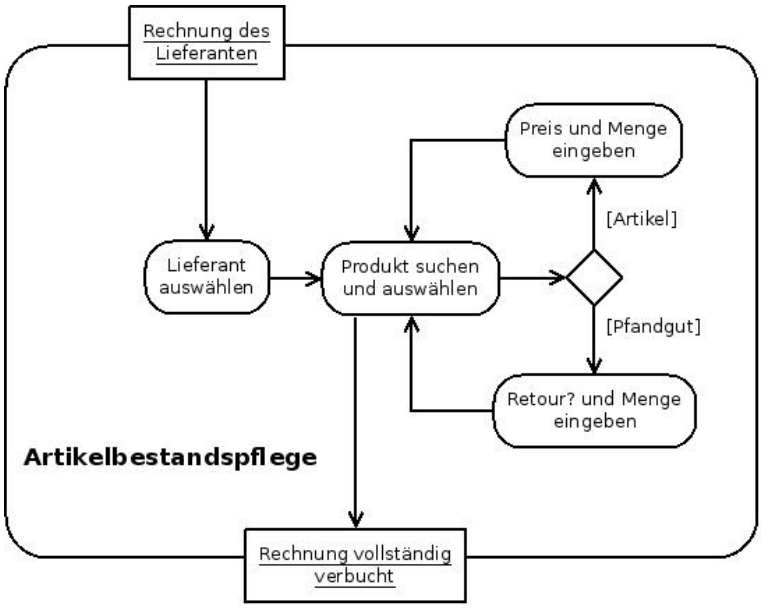


Abbildung 7.1.: Aktivitätsdiagramm der optimierten Bestandspflege

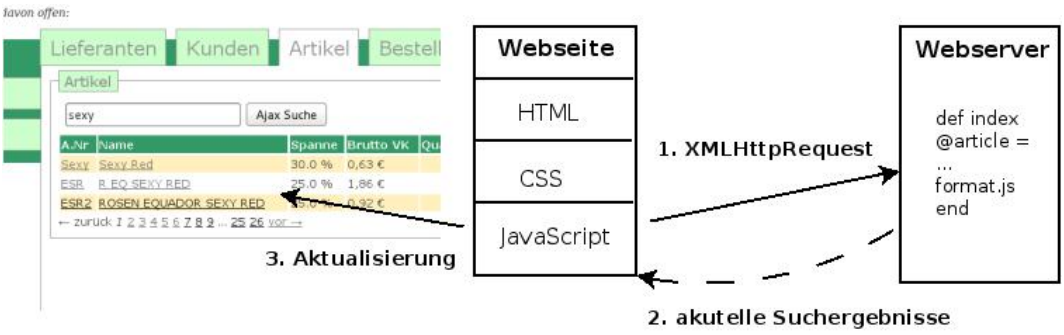


Abbildung 7.2.: Kommunikation Browser, Server mit Ajax

```
  <%= submit_tag "Ajax Suche", :name => nil %>
  </p>
<% end %>
```

```
function livesearch()
{
  $('form').delay(200).submit();
};
```

```
$('table tr:not(:first)').remove();
$('table').append('<%= j render @articles%>');
```

7.3.4. Autofocus

In der Vergangenheit war das Setzen des Focus auf ein bestimmtes Formularfeld nur mittels Javascript möglich. Seit HTML5 ist dies ein Bestandteil von HTML, dass bereits von allen gängigen Browsern unterstützt wird, außer dem Internet Explorer(IE).

```
<input autofocus="autofocus"/>
```

An dieser Stelle wird die fehlende HTML5 kompatibilität vom Internet Explorer ignoriert. Der Anwenderkreis der diesen Komfort benötigt ist so begrenzt, das die Nutzung eines kompatiblen Browser empfohlen werden kann.

7.3.5. Authentifizierung

Für die Benutzerauthentifizierung wird eine Klasse Benutzer(User) erstellt. Die Objekte der Klasse werden in der Datenbank gespeichert und umfassen den Benutzernamen sowie das Passwort als Hash. Für die asymmetrische Verschlüsselung des Passworts, wird der OpenBSD Algorithmus "bcrypt()"verwendet. Die Sessionverwaltung übernimmt vollständig RubyOnRails und dient zur Überprüfung ob ein Benutzer angemeldet ist. Um den Zugriff auf die Anwendung zu verwehren, ist ein Filter zur Authentifizierung vorgeschaltet, der an zentraler Stelle die gesamte Anwendung schützt.

```
class ApplicationController < ActionController::Base
  before_filter :authenticate
  protected
  def authenticate
    if User.find_by_id(session[:user_id]).nil?
    redirect_to :controller => "sessions", :action => "new"
    end
```

```
8     end
9     ...
10  end
```

7.4. Review

Die Optimierung der Suche mittels AJAX hat gezeigt, dass sich das dynamische Verhalten der Anwendung noch verbessern lässt. Die Interaktion mit der Anwendung kommt einer Rich-Client Anwendung damit sehr viel näher. Es sollte überlegt werden an welchen weiteren Stellen der Einsatz von AJAX Sinn macht. Wiederum wurde mittels einiger Akzeptanztests die Vollständigkeit der Anwendung durch die Händler überprüft und abgenommen. Die Bedienung der Benutzeroberflächen wurde gelobt und deren weitere Optimierung auf einen Zeitpunkt nach dem ersten praktischen Einsatz gelegt.

8. Release 0.1

8.1. Finales Productbacklog

User Story	Größe	Erledigt
Must Have		
U18	3	OK
U21	3	OK
U32	3	OK
U1	1	OK
U2	3	OK
U3	5	OK
U31	3	OK
U7	3	OK
U5	5	OK
U6	3	OK
U35	5	OK
U10	5	OK
Should Have		
U4	1	OK
U8	3	
U33	5	OK
U9	3	OK
U17	5	OK
U19	1	OK
U20	1	
U36	1	OK

U37	3	OK
Could Have		
U12	3	OK
U13	3	OK
U14	5	OK
U15	5	
U16	5	OK
U23	1	OK
U25	5	OK

Tabelle 8.1.: Finales Product Backlog

8.2. Integration in eine produktive Umgebung

Um "flowerstoweb" einsetzen zu können, muss die Anwendung auf einem Anwendungserver bereit gestellt werden. Hier gibt es drei Varianten.

1. Integration auf einem einzelnen Notebook, siehe Feldtest. Der Nachteil, die Anwendung ist nur ansprechbar, wenn der Notebook zur Verfügung steht. Also nur für sehr kleine Betriebe sinnvoll.

2. Integration bei einem Internetprovider. Die Nachteile: Kosten, Datenintegrität und direktes Drucken über die Anwendung nicht möglich .

3. Integration auf einem dedizierten Applikationserver auf dem Großmarkt (Empfohlen), (siehe App 8.1).

8.3. Online Demo

Im Rahmen dieser Arbeit wurde die Anwendung auf einem Internetserver veröffentlicht. Das Deployment wird wie bereits auf dem Entwicklungsserver mittels Capistrano durchgeführt. Anpassung in der deploy.rb, der database.yml sind dafür ausreichend.

Sie ist damit unter der URL `http://www.flowerstoweb.de` erreichbar. Für einen Demozugang bitte ich den Leser, sich bei mir unter mstevens@cbs-stevens.com zu melden.

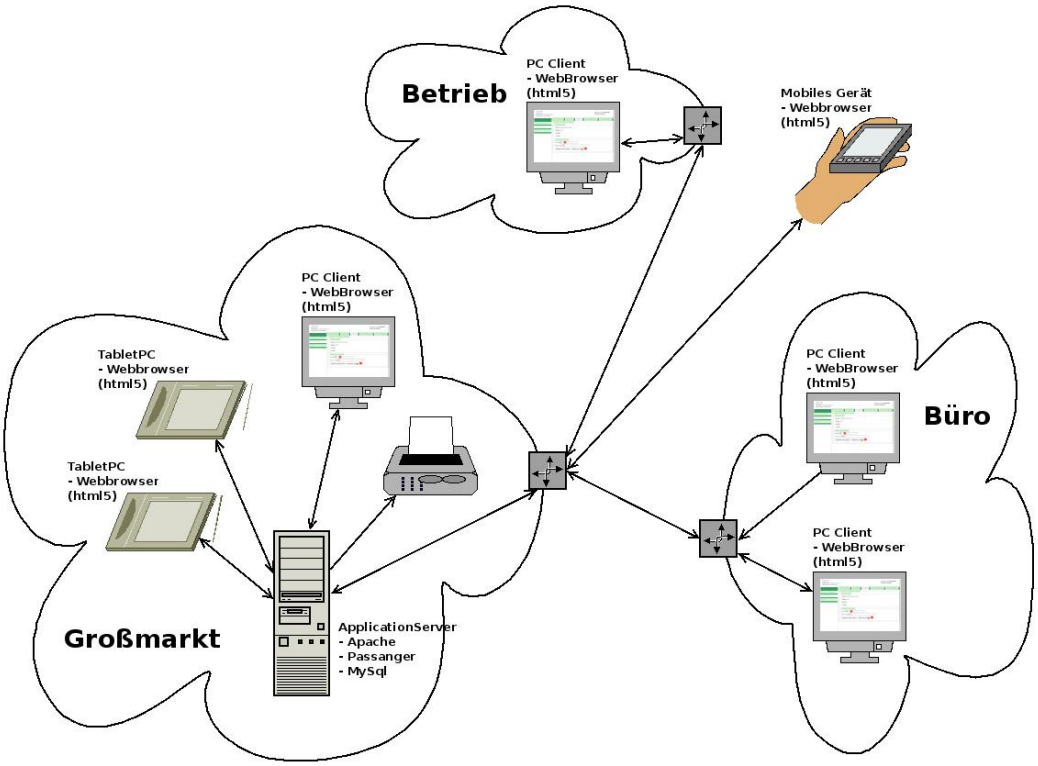


Abbildung 8.1.: Netzwerkintegration flowerstoweb

8.4. Feldtest

Um die praktische Anwendbarkeit zu überprüfen, wurden zwei Feldtests durchgeführt. Als Feldtest bezeichnet man einen empirischen Versuch die Software unter Realbedingungen zu überprüfen.

8.4.1. PC

Im ersten Test wurde die Anwendung auf einem Desktop PC geprüft. Hierzu gehörten die Funktionalitäten zum Anlegen eines Kunden, Anlegen von Artikeln und verbuchen von Einkäufen im Büro. Auf dem Großmarkt wurde das Anlegen und Drucken von Bestellungen sowie von Rechnungen getestet. Die Testumgebung war dabei folgendermaßen aufgebaut:

- Anwendungsserver Apache/Passenger auf PC mini-ATX Intel Atom 2x1,6 Ghz 2GB-Ram auf dem Großmarkt
- Drucker Canon PIXMA iP100 direkt am Anwendungsserver
- Lenovo Thinkpad T61 Dualcore 2,0Ghz 2GB Ram, Browser Firefox 10.0.6 auf dem Großmarkt mit 100MBit/s Lan
- Intel Pentium 512MB-RAM Browser Firefox 15.0.0.1 im Büro mit Wlan.

Die Bedienung der Anwendung war kein Problem, da sie dem Händler aus dem Abnahmetest bereits bekannt war. Die Eingabe der Buchungen wurde in Verbindung mit den Optimierungen aus dem Sprint 3 für schnell genug angesehen. Die Möglichkeit eingehende Bestellungen per Fax und Anrufbeantworter von praktisch überall aus direkt in die Anwendung einzugeben, wurde als hilfreiche Funktionalität bewertet. Auf dem Großmarkt ist das Erstellen von Bestellungen nur in zwei Schritten möglich. Zuerst werden Notizen auf Papier erstellt und nachträglich in die Anwendung eingegeben. Hier entsteht die Notwendigkeit einer praktikablen mobilen Variante des Bestellprozesses.

8.4.2. Mobida mit Tabletpc

Im zweiten Test ging es um die Anwendbarkeit einer mobilen Variante des Bestellprozesses. Hierfür wurde extra eine auf JQuery mobile basierende Oberfläche entwickelt und direkt in diesem praktischen Test überprüft. Getestet wurde mit folgender Umgebung

- Lenovo Thinkpad T61 Dualcore 2,0Ghz 2GB Ram

  - Anwendungsserver Apache/Passenger auf KVM mit SingleCore 2Ghz und 1GB RAM

- Lenovo A1 7“ Touchscreen Tablet 1Ghz Single Core und 512MB RAM
  - Webbrowser Android, Dolphin, Firefox.

- Netzwerk D-Link Wifi mit 54Mbit/s Access Point

Der Test dieses mobilen Prototypen hat verschiedene Probleme gezeigt. Das Tablet muss trotz der kleinen Größe von nur 7 Zoll mit zwei Händen bedient werden. Da oftmals eine Hand für die Ware gebraucht wird, muss das Tablet ständig aus der Hand gelegt werden. Die Eingabe der Formulardaten per Softwaretastatur über den Touchscreen ist, zwischen Kunde und Ware stehend, sehr fehleranfällig. Die Eingabe besteht hauptsächlich aus Kürzeln, Eigennamen und Zahlen. Die Autovervollständigung ist damit nicht hilfreich und wurde nach kurzem Testen deaktiviert. Das Auswählen von Listenelementen und das Abschicken der Formulardaten dauert zu lange und verhindert ein flüssiges Arbeiten. Eine Verzögerung von ca. 500ms ist die Regel, von denen alleine der Browser 300ms abwartet, um gewollte von ungewollten Eingaben zu unterscheiden.

In dem Bereich der mobilen Datenerfassung mit einer mobilen Webanwendung muss noch viel getan werden. Von eindeutigen Ergebnisse darf nicht ausgegangen werden, da hierfür weitere Tests mit anderen Geräten und alternativen zu JQuery mobil durchgeführt werden müssten. Jedoch lassen die Ergebnisse den Schluss zu, das beim Einsatz von Webanwendungen auf einem Tablet mit Problemen im Bereich Handhabung, Geschwindigkeit und Fehleingaben zu rechnen ist.

9. Zusammenfassung und Ausblick

9.1. Zusammenfassung

Der Einsatz eines Warenwirtschaftssystem findet seine Begründung in den hohen Anforderungen des Umsatzsteuergesetzes und weiteren gesetzlichen Bestimmungen, sowie in den immer knapper kalkulierteren Gewinnspannen, die eine Rentabilitätsbetrachtung notwendig machen. Um diese Anforderungen zu erfüllen, müssen alle Handelsaktivitäten in einer Softwarelösung festgehalten werden. Nach Befragung und Beobachtungen der betrieblichen Prozesse auf dem Blumengroßmarkt zeigt sich, dass zwei Gründe einen solchen Einsatz eines Warenwirtschaftssystems für die Händler erschweren. Zum einen ist es der Aufwand für die Artikelbestands- und Preis-pflege auf Grund von sich täglich ändernden Preisen, zum anderen die Notwendigkeit die Verkäufe direkt an der Ware registrieren zu müssen, da ein Verkauf in Zusammenarbeit mit dem Kunden binnen weniger Minuten stattfindet. Die bereits eingesetzten Lösungen scalieren von keiner Softwarelösung bis hin zu umfangreichen branchenspezifischen Warenwirtschaftssystemen. Neben diesen Lösungen findet jedoch eine umfangreiche Zettelwirtschaft für Bestellungen, Kalkulation, Lieferscheine sowie Einkauf statt. Die Gründe hierfür sind unter anderem die schlechte Verfügbarkeit an verschiedene Standorten und die Bedienerunfreundlich der Lösungen.

Der Einsatz von Webtechnologien zeigt hier seine Vorteil im Bereich der flexiblen Anpassung der Benutzeroberflächen aber auch die hohe Verfügbarkeit an verschiedenen Standorten und auf verschiedenen Gerätetypen.

Diese Arbeit beschäftigt sich mit der Einsetzbarkeit von Webanwendungen als Ersatz für bestehende Rich-Client Lösungen. Für dieses Projekt mit dem Namen “flowerstoweb“ sind die Anforderungen Grundlegend neu analysiert worden.

Eine enge Zusammenarbeit mit den Händlern und ein starker Fokus auf deren Anforderungen, waren ein Kriterium fürs Vorgehen. Die Verwendung von agilen Praktiken stellt sich dabei als funktionsfähiges Fundament für die Erstellung dieses Projektes heraus. Die enge Zusammenarbeit mit den Händlern während des Entwicklungsphase hatte zur Folge, dass Fehler und Lücken, die bei der Analyse entstehen können, frühzeitig auffallen und in einem

iterativen Prozess eine wertvolle Anwendung entsteht. Damit diese enge Zusammenarbeit möglich war, wurde eine Umgebung geschaffen in der sich der Händler beteiligen kann. Im Rahmen dieser Arbeit wurden dafür Userstory eingesetzt, die aus Sicht des Händlers geschrieben werden. Es wurde die Testumgebung Cucumber eingesetzt in der sich die Testfälle in einer für den Händler lesbaren Sprache verfassen lassen. Und zuletzt wurde akzeptanztestgetrieben Entwickelt, um in kurzen Abständen voll integrierte, also lauffähige Versionen der Software dem Kunden zu präsentieren. Dabei wurde ein ProductBacklog verwendet, dass die Bearbeitungsreihenfolge der Userstory festlegt, die zuvor in einem Anforderungsworkshop unter Beteiligung der Händler erarbeitet wurde.

Die Neuentwicklung im Rahmen dieser Arbeit hat gezeigt, das sich die Anforderungen, die Ihren Ursprung in den rechtlichen Vorgaben finden, leicht in einer Webanwendung realisieren lassen. Eine optimierte Artikelbestands- und Preis-pflege sowie fachliche Anforderungen in den Bereichen Artikelpfand, Kommissionsware sind ebenso erfolgreich realisierbar. Die mobile Datenerfassung im Verkauf, stellt dagegen ein größeres Problem da. Bei dem Versuch auf Basis von JQuery mobile eine mobile Datenerfassung über ein handelsüblichen Tablet zu realisieren, haben sich Probleme in den Bereichen Geschwindigkeit, Eingabefehlern und Handhabung ergeben.

Das eingesetzte Webframework RubyOnRails hat im Rahmen des agilen Vorgehens seine Stärken in den Bereichen Entwicklungsgeschwindigkeit, Erweiterbarkeit und Datenintegrität gezeigt. Die konsequenten Konzepte dieses Frameworks wie DRY und COC schränken die Wahlfreiheit des Entwicklers zwar ein, doch überwiegt der daraus entstehende Vorteil sich schnell zurechtzufinden und einen konsequenten Stiel beizubehalten.

Die akzeptanztestgetriebene Entwicklung führt die anwendungsorientierten Konzepte des agilen Vorgehens in der Entwicklungsphase fort. Sie bildet eine Schnittstelle zwischen Anwender und Entwickler und fördern automatisch ein gemeinsames Verständnis über die Anwendungsdomäne.

9.2. Ausblick

Zum Abschluss dieser Arbeit gibt es Anforderungen, die entweder bewusst außen vor gelassen wurden oder aus zeitlichen Gründen nicht mehr realisiert wurden.

- Um den praktischen Einsatz zu ermöglichen, muss es eine mobile Datenerfassung für den Verkauf geben. Ein Lösungsansatz dafür könnte phonegap in Verbindung mit einem industriellen Handgerät zur Datenerfassung sein.

- Die Übergabe von Rechnungsdaten wurde aus zeitlichen Gründen nicht mehr realisiert. Sie sollte aber wegen der Kosteneinsparungen gegenüber der Buchhaltung implementiert werden, (siehe Userstory U15).
- Die Markierung von Rechnung als Bezahlt wurde aus zeitlichen Gründen nicht mehr realisiert, (siehe Userstory U15).
- Die mobile Implementierung der Anwendung wurde nur für den Feldtest erstellt und kann noch nicht Produktiv eingesetzt werden, (siehe Userstory U8).
- Eine Anforderung, die vollständig weggelassen wurde, ist das Bilden von Zwischenergebnissen für die Inventur. So sollte es möglich sein, für ein abgeschlossenes Quartal einen Schnitt einzuführen, um zu verhindern, dass sich fehlerhafte Buchungen dauerhaft auswirken.

9.3. Kritische Würdigung

Der Einsatz einer untypisierten Programmsprache wie Ruby hinterlässt an vielen Stellen den Eindruck ohne Netz und doppelten Boden vorzugehen. Die korrekte Verwendung öffentlicher Methoden obliegt spürbar dem Entwickler, was an wichtigen Stellen eine Unsicherheit darstellen kann. Ein weiterer Punkt ist das Fehlen von Interfaces in Ruby. Für die Konstruktion von Komponenten und Schnitten innerhalb des Projekts wäre die Verwendung solcher Strukturelemente sinnvoll.

A. Anhang

A.1. Screenshots der Bestellung

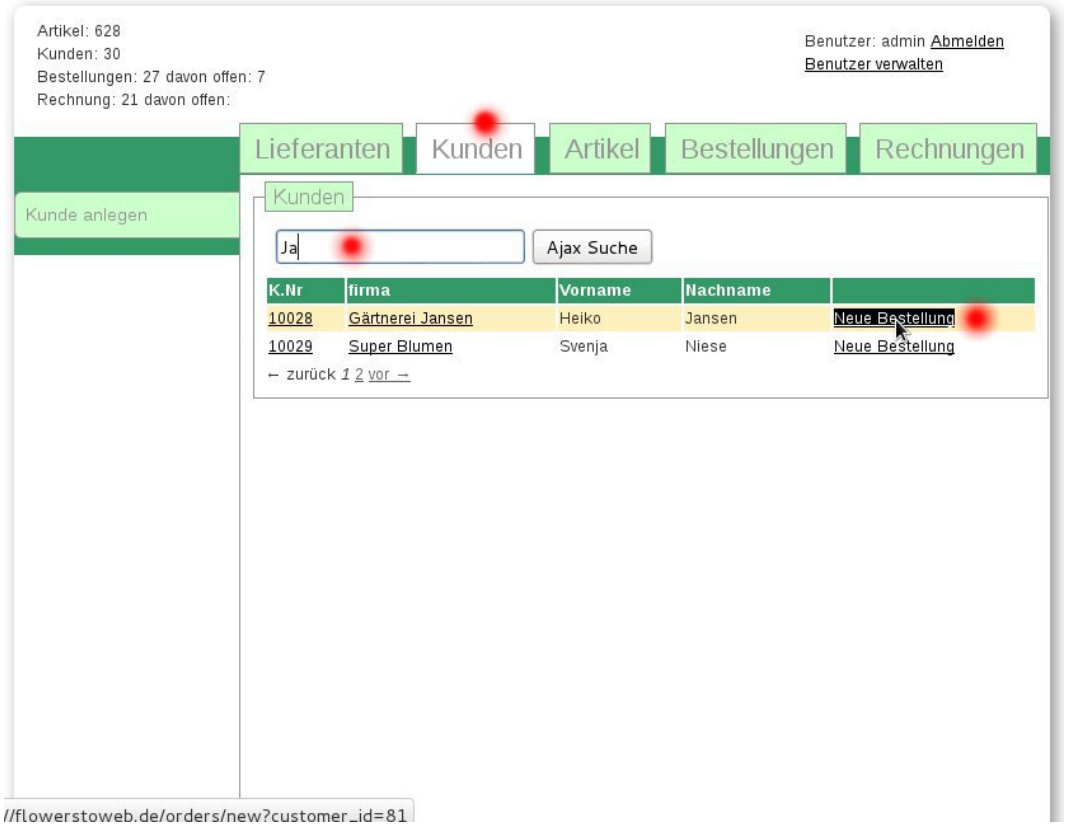


Abbildung A.1.: Kunde auswählen

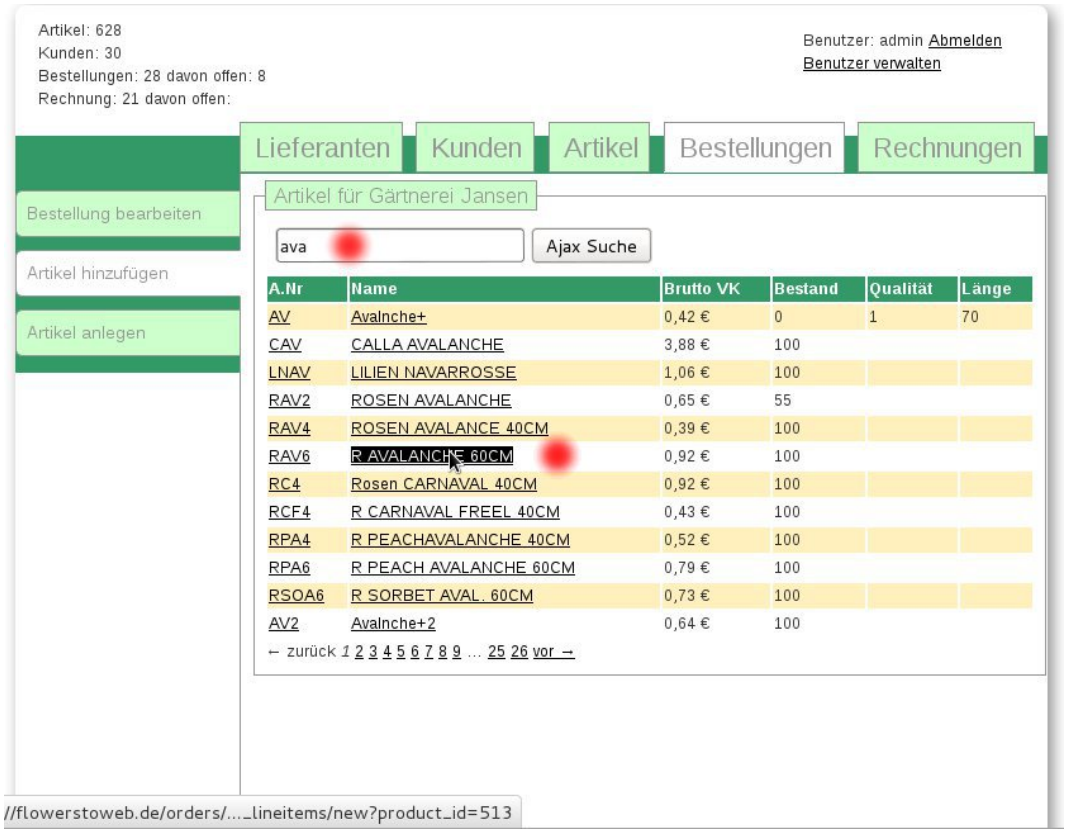


Abbildung A.2.: Artikel auswählen

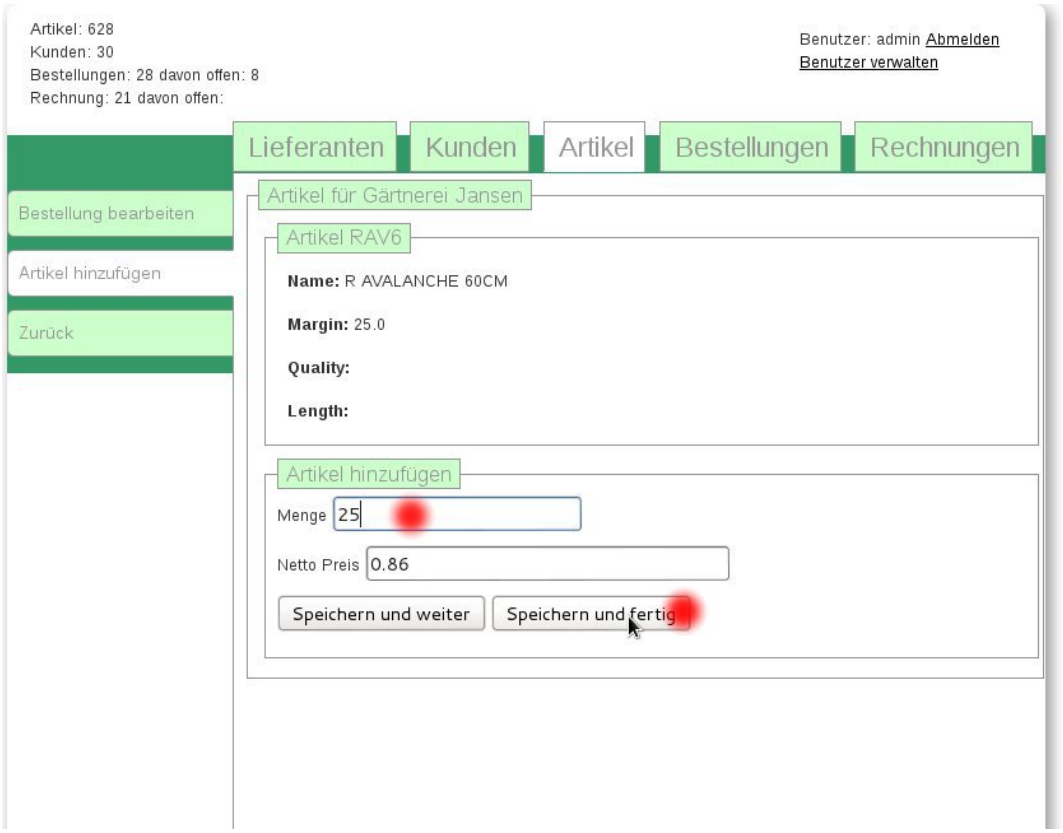


Abbildung A.3.: Artikel hinzufügen

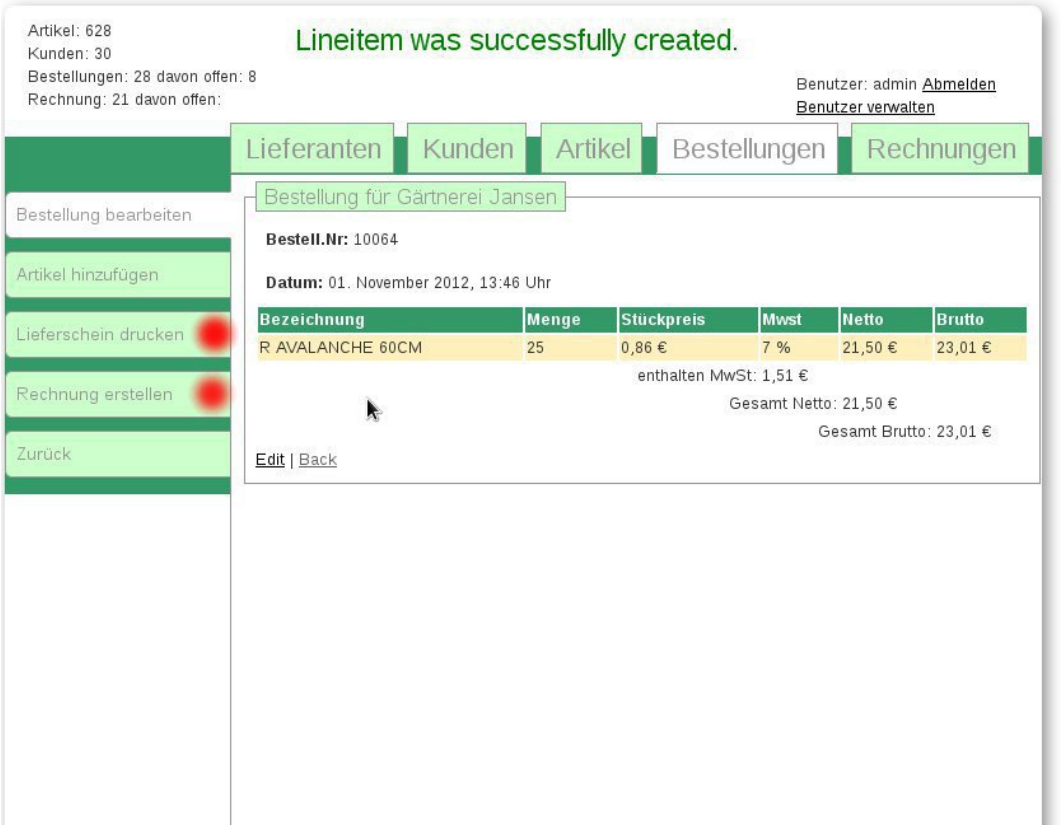
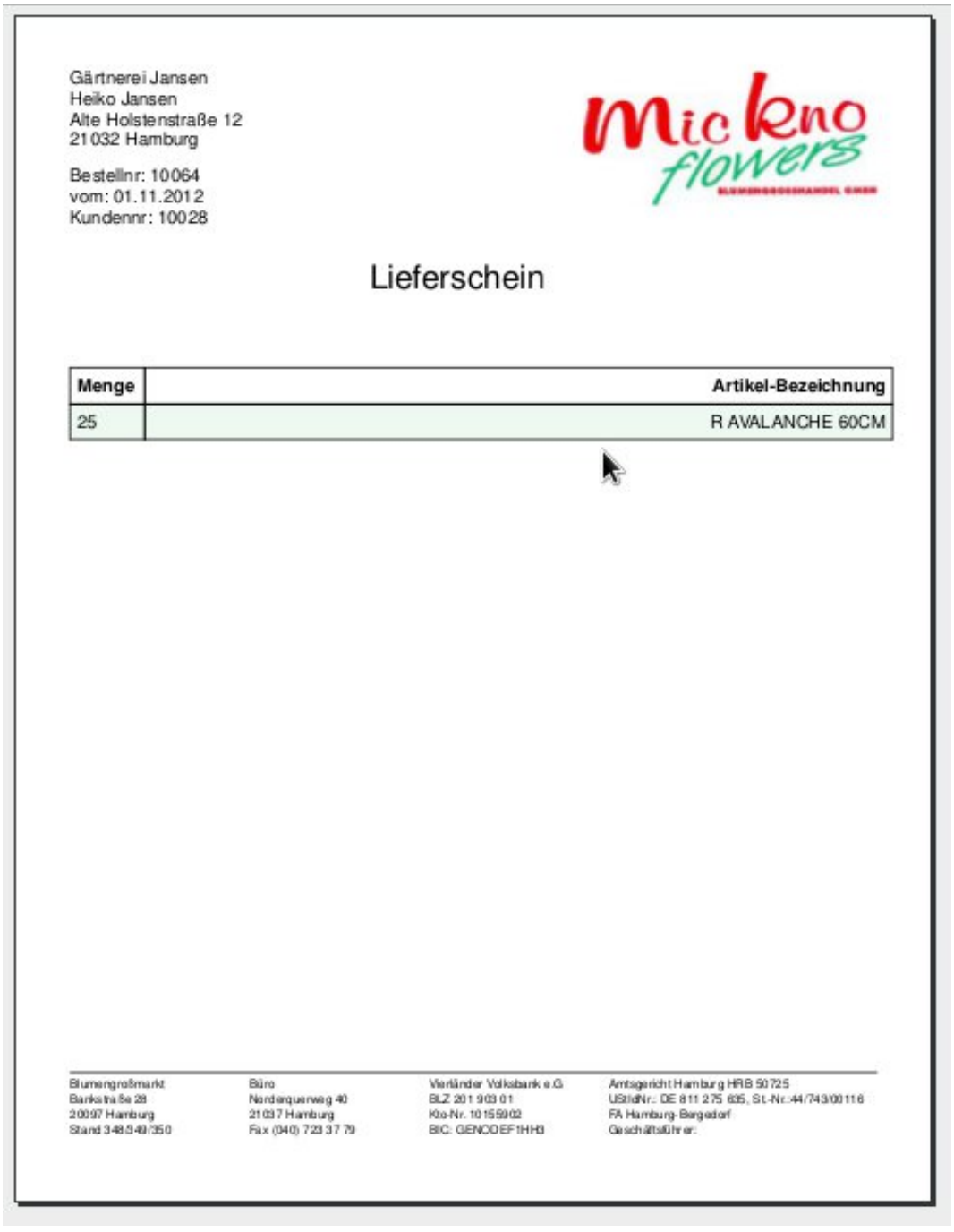


Abbildung A.4.: Lieferschein Drucken



Gärtnerei Jansen
Heiko Jansen
Alte Holstenstraße 12
21032 Hamburg

Bestellnr: 10064
vom: 01.11.2012
Kundennr: 10028

Lieferschein

Menge	Artikel-Bezeichnung
25	R AVALANCHE 60CM

Blumengroßmarkt
Bankstraße 28
20097 Hamburg
Stand 348/349/350

Büro
Norderquerweg 40
21037 Hamburg
Fax (040) 723 37 79

Vierländer Volksbank e.G.
BLZ 201 903 01
Kto-Nr. 1015 5902
BIC: GENODEF1HH3

Amtsgericht Hamburg HRB 50725
USt/dNr.: DE 811 275 635, St.-Nr.:44/743/00116
FA Hamburg-Bergedorf
Geschäftsführer:

Abbildung A.5.: Lieferschein

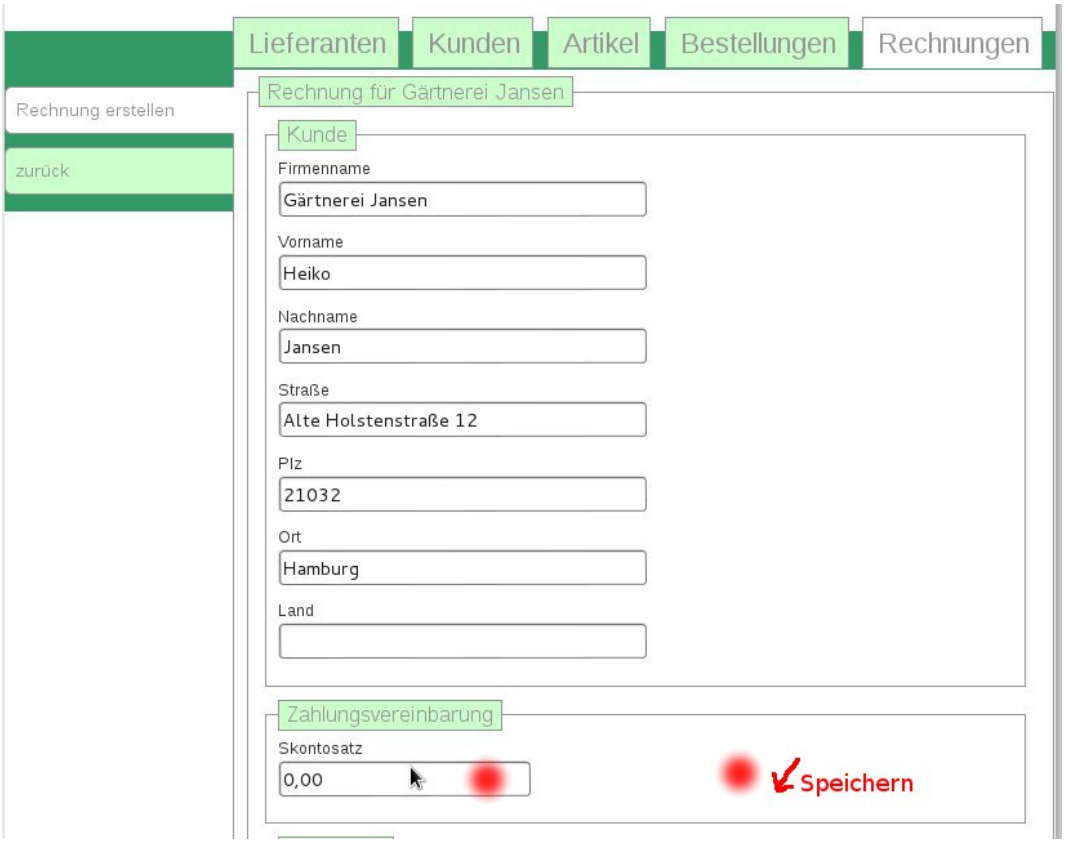
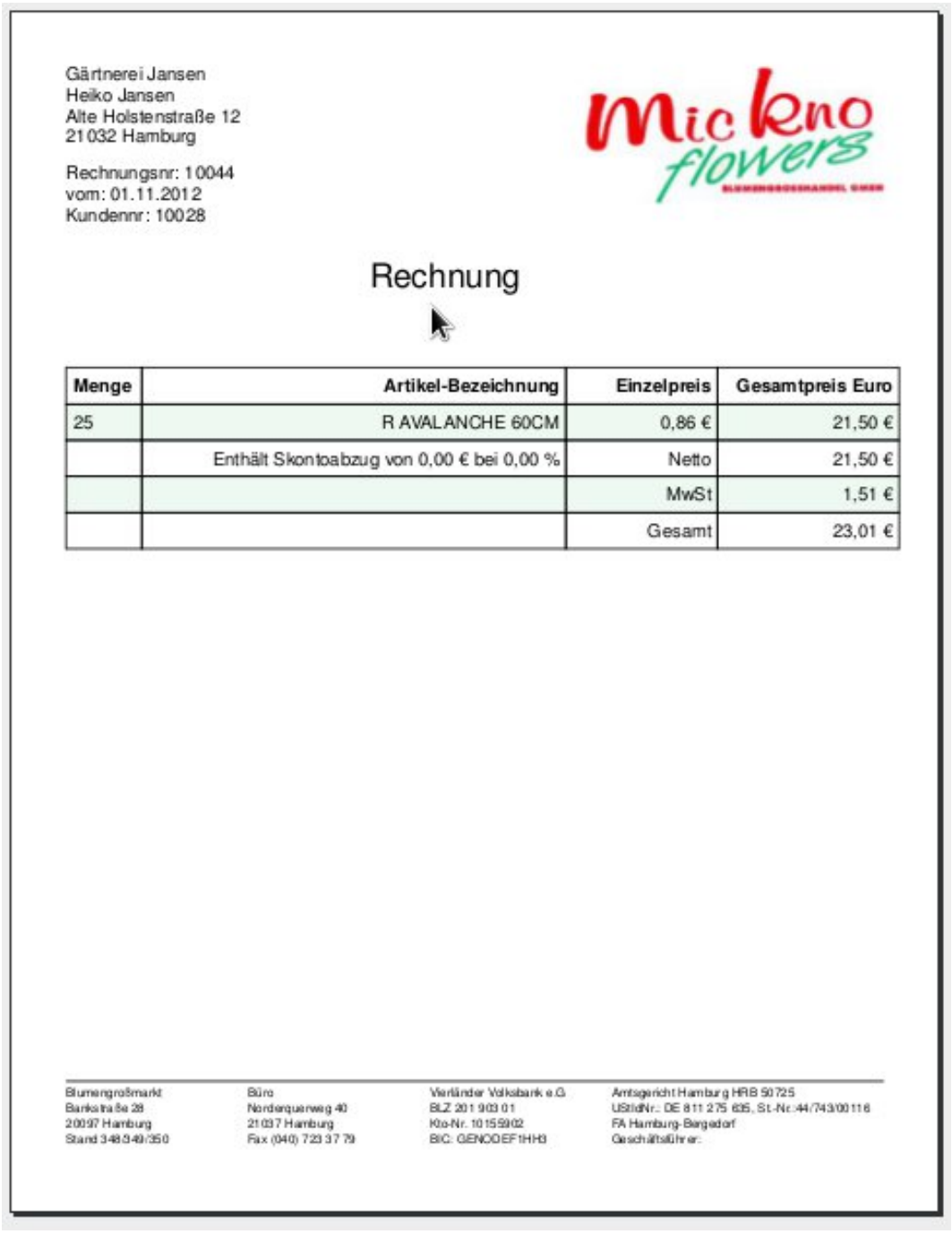


Abbildung A.6.: Rechnung erstellen



Gärtnerei Jansen
Heiko Jansen
Alte Holstenstraße 12
21032 Hamburg

Rechnungsnr: 10044
vom: 01.11.2012
Kundennr: 10028

Rechnung

Menge	Artikel-Bezeichnung	Einzelpreis	Gesamtpreis Euro
25	R AVALANCHE 60CM	0,86 €	21,50 €
	Enthält Skontoabzug von 0,00 € bei 0,00 %	Netto	21,50 €
		MwSt	1,51 €
		Gesamt	23,01 €

Blumengroßmarkt
Bankstraße 28
20097 Hamburg
Stand 348/349/350

Büro
Norderquerweg 40
21037 Hamburg
Fax (040) 723 37 79

Vierländer Volksbank e.G.
BLZ 201 903 01
Kto-Nr. 10155902
BIC: GENODEF1HH3

Amtsgericht Hamburg HRB 50725
USt/dNr.: DE 811 275 635, St.-Nr.:44/743/00116
FA Hamburg-Bergedorf
Geschäftsführer:

Abbildung A.7.: Rechnung

A.2. weitere Akzeptanzkriterien mit Akzeptanztest

A.2.1. U21

1. Pflichtfelder des Artikels sind: Artikelnr. Artikelbezeichnung, Umsatzsteuer-Satz, kalkulierte Gewinnspanne,

```
Szenario:Pflichtfelder des Artikels sind: Artikelnr.
Artikelbezeichnung, MwSt Satz, kalkulierte Gewinnspanne,
Angenommen man ist Angemeldet als "admin"
Und der Einkäufer geht zur Seite neuen Artikel anlegen
Wenn ich die Artikelname "Avalache+" eingebe
Und die Artikelnummer "1000" eingebe
Und den MwSt_Satz "0,07" eingebe
Und die Gewinnspanne "0,25" eingebe
Und auf Speichern klicke
Dann wurde der Artikel "Avalache+" erfolgreich gespeichert
```

```
Szenario:Nicht erfolgreich einen Artikel anlegen
Angenommen man ist Angemeldet als "admin"
Und der Einkäufer geht zur Seite neuen Artikel anlegen
Wenn ich die Artikelname "Avalache+" eingebe
Und ich gebe keinen MwSt_Satz ein
Und die Gewinnspanne "0,25" eingebe
Und auf Speichern klicke
Dann wurde der Artikel "Avalache+"  nicht erfolgreich
gespeichert
Und ich bekomme den Fehler "MwSt muss ausgefüllt werden"
angezeigt
```

2. Optionale Felder des Artikels sind: Kommissionsware, Qualität, Länge, Gewicht

3. Eingabefelder dürfen nach fehlerhafter Formulareingabe nicht verloren gehen.

```
Szenario:Eingabefelder dürfen nach fehlerhafter
Formulareingabe nicht verloren gehen.
Angenommen man ist Angemeldet als "admin"
Und der Einkäufer geht zur Seite neuen Artikel anlegen
Wenn ich die Artikelname "Avalache+" eingebe
Und ich gebe keinen MwSt_Satz ein
Und die Gewinnspanne "0,25" eingebe
```

```
Und auf Speichern klicke
Dann wurde der Artikel "Avalache+"  nicht erfolgreich
gespeichert
Und ich bekomme den Fehler "MwSt muss ausgefüllt werden"
angezeigt
Und das Feld "Artikelbezeichnung" enthält den Wert
"Avalache+"
Und das Feld "Gewinnspanne" enthält den  Wert "0,25"
```

4. Ist ein Artikel angelegt kann jeder Einkauf dieses Artikel als Artikelbestand registriert werden.

```
Szenario: Ist ein Artikel angelegt  kann jeder Einkauf dieses
Artikel als Artikelbestand registriert werden.
Angenommen man ist Angemeldet als "admin"
Und der Artikel "Avalache+" ist gespeichert.
Und der Lieferant "powerflowers" ist gespeichert
Wenn der Einkäufer auf die Seite der Lieferanten geht
Und den Lieferanten "powerflowers" auswähle
Und er "Einkauf verbuchen" klickt
Und er den Artikels "Avalache+" auswählt
Und die Menge "100" und EK-Preis "0.45" eingebe
Und auf "Speichern" klicke.
Dann hat der Artikel "Avalache+" einen aktuellen
Artikelbestand
von "100" mit einem EK Preis von "0.45"
```

5. Ein Artikelbestand hat die Pflichtfelder: Menge, EK-Preis

6. Ein Artikelbestand hat die optionalen Felder, Abverkaufsdatum, Herkunftsland, Lieferant, Artikelnummer des Lieferanten.

7. Der aktuelle Preis eines Artikels ergibt sich aus dem EK des aktuellen Artikelbestandes sowie der im Artikelstamm angegebene Gewinnspanne und Umsatzsteuer.

```
Szenario: Der aktuelle Preis eines Artikels ergibt sich aus
dem EK des aktuellen Artikelbestandes sowie der im
Artikelstamm angegebene Gewinnspanne und MwSt.
Angenommen man ist Angemeldet als "admin"
Und der Artikel "SexY Red" hat eine Gewinnspanne "30" ,
eine MwSt. "7" und einen akt. Artikelbestand mit einem
```

```
EK von "0.50"
Wenn der Verkäufer auf die Artikelseite von "Sexy Red" geht
Dann muß er den Preis von "0,70" dargestellt bekommen
```

A.2.2. U18

1. Pflichtfelder des Kunden sind Firmentitel, Vorname, Nachname, Straße, PLZ, Ort, Land

```
Szenario:Pflichtfelder des Kunden sind Firmentitel,
Vorname, Nachname, Straße, PLZ, Ort
Angenommen man ist Angemeldet als "admin"
Und der Verkäufer geht zur Seite neuen Kunde anlegen
Wenn er den "Firmenname" "Blumen am Schlossteich GbR"
eingebe
Und er den "Vorname" "Ursel" eingebe
Und er den "Nachname" "Meier" eingebe
Und er die "Straße" "Am Schlossteich 43" eingebe
Und er die "Plz" "22546" eingebe
Und er die "Ort" "Reinbeck" eingebe
Und auf Speichern klicke
Dann wurde der Kunde "Blumen am Schlossteich GbR"
erfolgreich gespeichert
```

2. Eingabefelder dürfen nach fehlerhafter Formulareingabe nicht verloren gehen

```
Szenario:Eingabefelder dürfen nach fehlerhafter
Formulareingabe nicht verloren gehen
Angenommen man ist Angemeldet als "admin"
Und der Verkäufer geht zur Seite neuen Kunde anlegen
Wenn er den "Firmenname" "Blumen am Schlossteich
GbR" eingebe
Und er den "Vorname" "Ursel" eingebe
Und er den "Nachname" "Meier" eingebe
Und er die "Straße" "" eingebe
Und er die "Plz" "22546" eingebe
Und er die "Ort" "Reinbeck" eingebe
Und auf Speichern klicke
Dann wurde der Kunde "Blumen am Schlossteich GbR"
nicht erfolgreich gespeichert
Und ich bekomme den Fehler "Straße muss ausgefüllt
werden" angezeigt
```

```
Und das Feld "Firmenname" enthält den Wert "Blumen
am Schlossteich GbR"
Und das Feld "Vorname" enthält den Wert "Ursel"
Und das Feld "Nachname" enthält den Wert "Meier"
Und das Feld "Plz" enthält den Wert "22546"
Und das Feld "Ort" enthält den  Wert "Reinbeck"
```

3. Ein Kunde hat optionale Felder einer Lieferadresse mit Straße, Plz, Ort haben
4. Ein Kunde darf sich nicht löschen lassen sobald er eine Bestellung oder Rechnung im System hat.

A.2.3. U19

1. Eine suche soll über die Eingabe von Kundennummer Firmenname, Nachname und Vorname möglich sein.

```
Szenario: Eine suche soll über die Eingabe von Kundennummer
Firmenname, Nachname und Vorname möglich sein.
Angenommen man ist Angemeldet als "admin"
Und es gibt den Kunden Firma "Blumenmeyer" "Ursel" "Meyer"
"Meyerstraße 123" "21039" "Hamburg" "10001"
Und der Verkäufer ist auf der Seite der Kunden
Wenn er im Suchfeld "10001" eingibt
Dann bekommt er den Kunden "10001" angezeigt
Wenn er im Suchfeld "Blumenmeyer" eingibt
Dann bekommt er den Kunden "10001" angezeigt
Wenn er im Suchfeld "Meyer" eingibt
Dann bekommt er den Kunden "10001" angezeigt
Wenn er im Suchfeld "Ursel" eingibt
Dann bekommt er den Kunden "10001" angezeigt
```

2. Auf der Liste von Kunden soll das direkte Erstellen von Bestellung zur Verfügung stehen.

A.2.4. U3

1. Eine Bestellung muss einem Kunden zugeordnet werden können
2. Eine Bestellung hat ein Datum das den Zeitpunkt der Erstellung festhält
3. Eine Bestellung beinhaltet die Informationen zur Menge, Preis und Bezeichnung der enthaltenen Artikel

4. Sobald eine Bestellung abgerechnet wird darf sie nicht mehr geändert oder gelöscht werden können.

```
Szenario: Sobald eine Bestellung abgerechnet oder verbucht
wird darf sie nicht mehr geändert oder gelöscht werden können.
Angenommen man ist Angemeldet als "admin"
Und es gibt ein verbuchte Bestellung mit der Nummer "1000"
Wenn der Benutzer auf die Liste der Bestellungen geht
Dann ist dort keine Möglichkeit zum löschen oder bearbeiten
vorhanden
Wenn er für die Bestellung "1000" eine gespeicherte Link zum
bearbeiten ausführt
Und auf "Order aktualisieren" klickt
Dann bekommt er die Meldung "Bestellung kann nicht geändert
werden. Sie ist bereits abgerechnet"
```

A.2.5. U31

1. Um einen Artikel oder Pfandgut aus zu wählen muss es möglich sein ihn anhand von Artikelnummer und Name zu suchen.

2. Auf der Liste angezeigter Artikel darf es nur Links zum Auswählen des Listeneintrags geben.

3. Artikel und Pfandgut soll über die gleichen Seite ausgewählt werden können.

```
Szenario: Artikel und Pfandgut soll über die gleiche Seite
ausgewählt werden können.
Angenommen man ist Angemeldet als "admin"
Und es gibt ein Artikel "Avalanche+"
Und es gibt ein Pfandgut "Eimer"
Und es gibt den Kunden "Blumenmeyer"
Wenn der Verkäufer eine neue Bestellung für "Blumenmeyer"
anlegt
Dann kann er sowohl "Avalanche+" als auch den "Eimer"
auswählen
```

A.2.6. U7

1. Wenn ein Artikel ausgewählt wird muss eine Eingabe von Menge, Stückpreis und Umsatzsteuer-Satz erzwungen werden.

2. Sind Angaben zu Preis und Umsatzsteuer im Artikel vorhanden so müssen sie als Vorlage verwendet werden.

A.2.7. U33

1. Die Verbuchung eines Einkaufs muss die Eingabe von Menge und Einkaufspreis erzwingen.
2. Der Einkauf eines Artikels muss sich einem Lieferanten zuordnen lassen.
3. Ein Einkauf muss sich ändern und löschen lassen können.
4. Wird der Bestand eines Einkaufs durch einen Verkauf erfasst so darf er nicht mehr gelöscht werden können.
5. die Kosten für einen Einkauf berechnen sich durch die Menge und dem Stückpreis in Netto.

A.2.8. U2

1. Die Umsatzsteuer muss als Summe, aller enthaltener Positionen mit Umsatzsteuer , auf der Rechnung ausgewiesen sein.
2. Es muss ersichtlich sein mit welchem Steuersatz eine Rechnungsposition versteuert wird.

A.2.9. U32

1. Es kann eine neue Rechnung für einen Kunden angelegt werden.

```
Szenario: Es kann eine neue Rechnung für einen Kunden
angelegt werden.
Angenommen man ist Angemeldet als "admin"
Und es gibt den Kunden "Blumenmeyer"
Und der Artikel "Sexy Red" hat eine Gewinnspanne "30" ,
eine MwSt. "7" und einen akt. Artikelbestand mit einem EK
von "0.50"
Wenn der Verkäufer auf die Liste der Kunden auf
"Blumenmeyer" klickt
Und klickt auf den Link "Neue Bestellung"
Und kommt er auf die Seite "Artikel für Blumenmeyer"
Und klickt auf den Link "Sexy Red"
```

```
Und gibt die Menge "120" und den Preis "0.22" ein
Und klickt auf den Button "Speichern und fertig"
Und kommt er auf die Seite "Bestellung für Blumenmeyer"
Und klickt auf den Link "Rechnung erstellen"
Und kommt er auf die Seite "Rechnung für Blumenmeyer"
Und klickt auf den Button "Drucken und Verbuchen"
Dann gibt es im System eine "Rechnung für Blumenmeyer" mit
dem gesamt Netto: "26,40"
```

2. Eine Rechnung hat eine eindeutige fortlaufende, eindeutige Rechnungsnummer. siehe A5
3. Eine Rechnung muss den vollständigen Namen und Anschrift des Verkäufers und Kunden haben. siehe A2
4. Eine Rechnung muss ein Ausstellungsdatum haben. siehe A4
5. Eine Rechnung muss ein den Zeitpunkt der Lieferung und Leistung wiedergeben. siehe A7
6. Eine Rechnung muss die Menge und handelsübliche Bezeichnung der gelieferten Gegenstände wiedergeben. siehe A6
7. Eine Rechnung muss das Entgelt und den hierauf entfallenden Steuerbetrag ausweisen . siehe A10
8. Der Artikelbestand wird bei jedem Verkauf des Artikel gemindert.
9. Deckungsbeitrag wird bei jedem Verkauf eines Artikels vom aktuellen Artikelbestand neu berechnet.

A.2.10. U1

1. Die Angaben von Name und Adresse des Kunden müssen links oben auf einer Rechnung zu sehen sein .

A.2.11. U5

1. Es muss möglich sein einen Einkauf als Kommission zu kennzeichnen.
2. Im Lieferanten werden die umgesetzte Menge und die damit folgenden Kosten eines Einkaufs angezeigt.

A.2.12. U6

1. Skonto wird in Prozent angegeben.
2. Skonto ist ein Netto-Skonto und wird vom Nettobtrag abgezogen.
3. Der Skontobetrag errechnet sich durch (Nettobetrag*Skontosatz)/100
4. Skonto mindert den Nettobetrag einer Position eines skontofähigen Artikels um den Skontobetrag.
5. Bei Skonto wird der Abzug in einer Rechnungsposition angegeben.
6. Der Skonto Gesamtbetrag muss auf der Rechnung ausgewiesen werden.

A.2.13. U17

- Eine Rechnung die gedruckt wird gilt als verbucht und darf nicht mehr verändert oder gelöscht werden.

```
Szenario: Eine Rechnung die gedruckt wird gilt als
verbucht und darf nicht mehr verändert oder gelöscht werden.
Angenommen man ist Angemeldet als "admin"
Und es gibt ein unverbuchte Rechnung mit der Nummer "1000"
Wenn der Verkäufer die Rechnung "1000" öffnet
Dann wird der Status "offen" angezeigt
Und es gibt die Funktionen "Rechnung bearbeiten" und "löschen"
Wenn der Verkäufer auf "Rechnung drucken" klickt
Und der Verkäufer die Rechnung "1000" öffnet
Dann wird der Status "verbucht" angezeigt
Und es gibt kein Funktionen "Rechnung bearbeiten" und "löschen"
Wenn er für die Rechnung "1000" einen gespeicherten Link zum
bearbeiten ausführt
Und auf "Drucken und Verbuchen" klickt
Dann bekommt er die Meldung "Rechnung ist bereits verbucht."
```

- Der Rechnung wird direkt auf einem Drucker ausgedruckt oder wahlweise als PDF Download angeboten.
- Der Ausdruck muss die Steuernummer oder Umsatzsteueridentifikationsnummer aufweisen. siehe A3

A.2.14. U9

1. Der Gewinn wird durch die Summe der Nettobeträge des Verkaufs abzüglich der Kosten des Einkaufs eines Artikels berechnet.

A.2.15. U10

1. Das Pfandgut muss auf der Rechnung mit aufgeführt werden und in den Gesamtbetrag der Rechnung berücksichtigt werden.
2. Gutschriften sollen in gleicher Weise, wie Verkäufe, einer Bestellung zugewiesen werden.

A.2.16. U35

1. Der aktuelle Bestand eines Pfandguts soll im Kunden angezeigt werden.
2. Ist der aktuelle Bestand gleich null soll dieser nicht mit angezeigt werden
3. Der Bestand ergibt sich aus der Summe aller Verkäufen und aller Gutschriften eines Pfandguts.

A.2.17. U12

1. Ein Lieferschein muss Menge, Artikelname der Artikel einer Bestellung beinhalten.
2. Ein Lieferschein muss das Datum der Bestellung, den Namen des Kunden und deren Lieferanschrift beinhalten.

A.2.18. U13

1. Der Lieferschein wird direkt auf einem Drucker ausgedruckt oder wahlweise als PDF Download angeboten.

A.2.19. U14

1. Eine Sammelrechnung soll automatisch alle noch nicht abgerechneten Bestellungen abrechnen.

A.2.20. U23

1. Es muss nach Artikelnummer und Artikelname gesucht werden können.
2. Die Liste von Artikeln muss der Preis und die verfügbare Menge angezeigt werden.
3. Die verfügbare Menge berechnet sich aus Menge des aktuellen Einkaufs abzüglich der bereits verkauften Menge.

A.2.21. U25

1. Für jede neue Zeichen eines Suchkriteriums muss automatisch,ohne explizitem Abschicken des Formulars, eine passende Ergebnisliste ausgegeben werden.

A.2.22. U4

1. Die Liste der Bestellungen muss wiedergeben welche Bestellung noch nicht abgerechnet wurden.
2. Die neuste Bestellung soll in der Liste oben angezeigt werden und absteigend nach dem Bestelldatum sortiert sein.

A.2.23. U8

1. Es muss möglich sein, von unterschiedlichen PC aus ,auf den aktuellen Datenbestand zu zu greifen.

Literaturverzeichnis

[WIRD 2011] WIRDEMANN, Ralf (2011): Scrum mit User Stories. 2. überarbeitete Auflage. Carl Hansen Verlag.

[MORS 2012] MORSY, Hussein/OTTO, Tanja(2012): Ruby on Rails 3.1. 2. aktualisierte Auflage. Galileo Press.

[RAILSCASTS 2007] BATES,Ryan(2007):AJAX with RJS. http://railscasts.com/episodes/43-ajax-with-rjs?view=asciicast, Abrufdatum: 02.09.2012

[BITKOM 2007] BITKOM (2007):Fast jeder fünfte Mensch ist online http://www.bitkom.org/de/presse/49919_46069.aspx Abrufdatum: 03.09.2012

[BITKOM 2012] Hochwertige Smartphones werden Standard http://www.bitkom.org/de/presse/30739_73193.aspx Abrufdatum:03.09.2012

[Databecker 2013] :Rechnungsdruckerei 2013 http://www.databecker.de/shop/software/goldene-serie/druckereien/rechnungsdruckerei.php Abrufdatum:03.09.2012

[Brückner 2012] :WinAB http://www.brueckner-gmbh.de/winab/blumen.html Abrufdatum: 03.09.2012

[EDEKA 2012] :Lebensmittel online http://www.edeka24.de/Lebensmittel/ Abrufdatum: 03.09.2012

[IHK 2011] :Pflichtangaben in der Rechnung http://www.frankfurt-main.ihk.de/recht/steuerrecht/umsatzsteuer_national/rechnungsstellung_pflichtangaben/ Abrufdatum:03.09.2012

[IHK 2011] :Pflichtangaben in der Rechnung http://www.frankfurt-main.ihk.de/recht/themen/handelsrecht/brief/index.html Abrufdatum:08.09.2012

[BUNDLER 2012] :Bundler Home http://gembundler.com/ Abrufdatum:13.09.2012

[CUCUMBER 2012] :Cucumber Home http://cukes.info/ Abrufdatum:14.09.12

[CAPYBARA 2012] :Capybara Home https://github.com/jnicklas/capybara Abrufdatum:20.09.2012

[JavaSpektrum 2006] :Ruby on Rails -Alternative zur Webentwicklung mit Java http://www.sigs-datacom.de/fileadmin/user_upload/zeitschriften/js/2006/04/baustert_wirdemann_JS_04_06.pdf Abrufdatum:07.10.2012

[Agile Manifest 2012] :Prinzipien hinter dem Agilen Manifest http://agilemanifesto.org/iso/de/principles.html Abrufdatum: 19.10.2012

[BMF 2011] :Elektronische Übermittlung von Bilanzen sowie Gewinn- und Verlustrechnungen; Anwendungsschreiben zur Veröffentlichung der Taxonomie http://www.bundesfinanzministerium.de/Content/DE/Downloads/BMF_Schreiben/Steuerarten/Einkommensteuer/051.html Abrufdatum: 20.10.2012

[PRAWN PDF] :Fast, Nimble PDF Generation For Ruby https://github.com/prawnpdf/prawn Abrufdatum: 25.10.2012

[PDF/A Sachsen 2009] :Handreichung zur rechtsicheren Aufbewahrung von elektronischen Dokumenten http://www.moderneverwaltung.sachsen.de/download/Handreichung_rechtssichere_Aufbewahrung_V1_2009.11.16.pdf Abrufdatum: 25.10.2012

[Online PDF/A Validierung] :PDF/A Check von intarsys http://www.intarsys.de/pdfa-check Abrufdatum: 25.10.2012

Glossar

AJAX	Asynchronous JavaScript and XML
Artikel	Artikel stellen im System alle Handelsgüter da. Synonym wird das Wort Produkt verwendet um zwischen Pfandgut und Artikeln unterscheiden zu können.
ATDD	Acceptance Test Driven Development, Überführung von Akzeptanzkriterien in Tests, die der Implementierung als Vorlage dienen
CoC	Convention over Configuration, Prinzip zur Verkleinerung von Konfigurationen
DoD	Definition Of Done , Fertigstellungskriterien, beschrieben durch Akzeptanzkriterien
DRY	Don't Repeat Yoursel, Prinzip zur Vermeidung von Redundanzen
DSL	Domain Specific Language, Fachsprache des Anwendungsbereichs
Händler	Blumengroßhändler und potenzielle Anwender der Software
Kunde	Käufer der Artikel vom Blumenhändler
MOBIDA	Mobile Datenerfassung
Pfandgut	Artikel die einem Pfandkreislauf unterliegen. Hier zumeist Behälter für den Transport
Rich Client	Arbeitsplatzrechner auf dem eine Desktopanwendung verwendet wird, deren Anwendungslogik und Datenbankzugriffe lokal ausgeführt werden
RoR	RubyOnRails, Webframework auf Ruby
U12	Eine konkrete Userstory, hier mit der Nummer 12

Umsatzsteuer	Stellt die gesetzliche Umsatzsteuer da. Wird umgangssprachlich auch Mehrwertsteuer genannt
Userstory	Ist eine Art ausformulierter Anwendungsfall mit den Angaben: Wer ,will was , und warum will er es
Velocity	Geschwindigkeit oder Leistungsfähigkeit eines Entwicklungsteams als fiktive Zahl
W3C	Internationales Gremium zur Standardisierung des Internets

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 08. November 2012 Markus Stevens