CARL VON OSSIETZKY universität OLDENBURG

Fakultät II - Informatik, Wirtschafts- und Rechtswissenschaften
Department für Informatik

Masterstudiengang Wirtschaftsinformatik

Masterarbeit

Entwurf und Implementierung eines prototypischen Web-Shops zur Erfassung und Abwicklung von Steckperlenbilderbestellungen

vorgelegt von
Simon Jakubowski

Betreuender Gutachter
Prof. Dr.-Ing. Oliver Theel

Zweiter Gutachter
Dipl. Inform. Eike Moehlmann

Oldenburg, 27. Juli 2015

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

Im Zuge einer vorangegangenen Abschlussarbeit wurde eine Maschine entwickelt, die es ermöglicht, automatisiert Steckperlenbilder anzufertigen. Diese Maschine wird zur Zeit durch ein kleines Touch-Display gesteuert. Der Benutzer erfasst auf dem Display sein Steckperlenbild und gibt es nach Fertigstellung in den "Druck". Die Maschine steckt ohne menschliche Interaktion das Steckperlenbild zusammen und gibt es daraufhin aus. Das Bild wird anschließend von einem Mitarbeiter gebügelt, um die Steckperlen miteinander zu verbinden.

Das Erfassen eines solchen Steckperlenbildes kann aber mitunter sehr zeitaufwändig sein. Aus diesem Grund soll innerhalb dieser Abschlussarbeit ein Web-Shop konzipiert und implementiert werden, der die Bedienung der "Steckperlenmaschine" über einen Web- Browser ermöglicht. Dabei soll der gesamte Prozess von der Anfertigung des Steckperlenbildes über die Bestellung bis hin zum Versand des Bildes modelliert, umgesetzt und weitgehend automatisiert werden. Der Web-Shop in Verbindung mit der Steckperlenmaschine soll darüber hinaus auf verschiedenen Veranstaltungen, wie z.B. dem Tag der offenen Tür an der Universität Oldenburg verdeutlichen, welchen Mehrwert die Informatik in unserem täglichen Leben generiert. Dies ist besonders für Jugendliche interessant, da sie einen direkten Nutzen, durch die Unterstützung der Informatik, an einem einfachen und konkreten Beispiel vorgeführt bekommen.

1.1 Ziel der Arbeit

Ziel dieser Arbeit ist es, einen Web-Shop zu konzipieren und prototypisch zu implementieren, der die Steuerung der Steckperlenmaschine über einen Web-Browser ermöglicht. Darüber hinaus muss der Steckperlen-Web-Shop über grundlegende Funktionen eines Web-Shops verfügen, wie beispielsweise eine Kontoverwaltung oder eine Übersicht von verschiedenen Bildern, die den Kunden zu einem schnellen Kauf verleiten könnten (z.B. beliebte Bilder, neue Bilder). Die grundlegenden Anforderungen aus Sicht des Benutzers an den Shop sind dabei:

- Erstellung von Steckperlenbildern im Web-Browser
  - manuelles Setzen von Steckperlen
  - Generierung eines Steckperlenbildes aus Vorlage eines beliebigen Bildes, das ein Benutzer hochlädt
- Bestellung von Steckperlenbildern
- Überwachung der eigenen, getätigten Bestellungen (Bearbeitung und Versand)
- Suche nach Bildern mithilfe verschiedener Parameter (wie z.B. Name, Kategorie)

Über diese Anforderungen hinaus ist es notwendig, dass Bestellungen von Mitarbeitern bearbeitet und abgeschlossen werden können. Die Rolle "Mitarbeiter" stellt demzufolge folgende Anforderungen an den Web-Shop:

- Ansicht der eingegangen Bestellungen
- Druck von einzelnen Bestellungen
- Statusveränderungen an Bestellvorgängen vornehmen (z.B. von "in Bearbeitung" zu "wird versandt")
- Benutzerverwaltung

1.2 Aufbau der Arbeit

Zu Beginn der Arbeit werden zunächst einige Grundlagen zu den zentralen Technologien erläutert. Dabei werden einige serverseitige und clientseitige

Technologien betrachtet.

Nachdem die Grundlagen zu den verschiedenen Technologien beschrieben wurden, werden im nachfolgenden Kapitel die Anforderungen an das System erfasst. Diese teilen sich dabei in funktionale und nicht funktionale Anforderungen. Auf Grundlage dieser Anforderungen werden dann die bereits beschriebenen Technologien verglichen.

Anknüpfend an die Analyse folgt das Kapitel der "Konzeptionierung". Dabei werden zunächst die geeigneten Technologien für die weitere Umsetzung ausgewählt. Anschließend wird das Datenbankschema hergeleitet sowie eine geeignete Kodierung für die Steckperlenmotive beschrieben. Darauf folgt die Konzeptionierung der einzelnen Web-Shop-Komponenten. Dazu gehören unter anderem der Designer, der Bestellvorgang sowie diverse Suchmöglichkeiten und Ansichten. Nachdem die Komponenten beschrieben wurden, wird der Druckprozess erläutert. Dieser umfasst das Zusammenspiel zwischen Web-Shop und Steckperlendrucker. Weiterhin wird innerhalb dieses Kapitels das Design-Pattern beschrieben, welches der folgenden Implementierung zugrunde liegen wird.

Nach der Konzeptionierung folgt nun das Kapitel "Implementierung", welches die Umsetzung der zuvor entworfenen Komponenten beschreibt. Dabei werden zunächst einige Richtlinien beschrieben, die während der Implementierung eingehalten werden müssen. Dazu zählt das "Test-Driven-Development" wie auch die "Clean Code"-Vorgaben.

Das anschließende Kapitel "Evaluation" beschäftigt sich mit gängigen Use-Cases, die aus Sicht der Benutzer am System analysiert werden. Hierbei wird ein Hauptaugenmerk auf die jeweiligen Anforderungen gelegt, die in den verschiedenen Use-Cases aufgehen.

Abschließend werden die Ergebnisse, die im Laufe dieser Arbeit gewonnen wurden im letzten Kapitel zusammengefasst. Dazu gehört neben einer Zusammenfassung der Arbeit auch die Beobachtung der Applikation in einem Testbetrieb mit einigen Test-Nutzern. Nachfolgend wird ein Ausblick, mögliche offene Punkte bzw. sinnvolle weitere Features des Shops aufgezeigt.

Kapitel 2

Grundlagen

In dem folgenden Kapitel werden die Grundlagen zu den wichtigsten Konzepten und Technologien näher betrachtet, die innerhalb dieser Arbeit im weiteren Verlauf genauer betrachtet werden. Dazu zählen als Back-End-Technologien EJB (Enterprise Java Beans), CDI (Contexts and Dependency Injection) und Spring. Als Font-End-Technologien werden, JSF als klassischer JEE (Java Enterprise Edition) Vertreter sowie AngularJS als ein, HTML5 und JavaScript Vertreter untersucht.

2.1 Aufbau einer Web-Applikation

Eine Web-Applikation besteht in der Regel aus einem client- und einer serverseitigen Implementierung. Der Client ist in den meisten Fällen ein handelsüblicher Web-Browser (wie z.B. Mozilla Firefox, Google Chrome oder Microsofts Internet Explorer). In einigen, eher seltenen Fällen, sind dies andere Applikationen die über Web-Schnittstellen die Applikation ansteuern. Spricht man in einer solchen Web-Applikation von der clientseitigen Umsetzung, so wird im weiteren Verlauf dieser Arbeit die Rede vom Front-Endsein. Die Serverseite wird als Back-Endbezeichnet. Die nachfolgenden Kapitel beschreiben Technologien für die jeweiligen Bereiche Front- und Back-End. Alternative Mischformen aus server- und clientseitiger Implementierung wie beispielsweise Vaadin oder GWT (Google Web Toolkit) sind eher selten am Markt vertreten und oftmals nur in wenigen Szenarien sinnvoll. Aus diesem Grund wird auf diese innerhalb der Arbeit nicht weiter eingegangen.

2.2 Back-End-Technologien

Zur serverseitigen Umsetzung der Web-Applikation stehen diverse Frameworks im Java-Umfeld zur Auswahl. Innerhalb dieser Arbeit werden die Frameworks EJB 3, CDI sowie Spring betrachtet, da diese seit geraumer Zeit marktführend sind.

2.2.1 EJB 3

Enterprise Java Beans (EJB) ist ein Framework bzw. eine Technologie zur leichteren Entwicklung von Java Enterprise Applications. Dabei soll das Framework dem Entwickler diverse Aufgaben abnehmen, wie z.B. die Steuerung der Datenbank Transaktionen, die Lebensdauer von Java Beans sowie die Kontrolle der Nebenläufigkeit. EJB ist spezifiziert in der JSR 220: Enterprise JavaBeans 3. In den vorherigen Versionen von EJB (Versionen 1.x und 2.x) war das gesamte Framework sehr umständlich aufgebaut und für die meisten Entwickler nur sehr verständlich. Ein Hauptziel von EJB in der Version 3.x war es, das Framework zu vereinfachen und die Komplexität zu reduzieren, um so die Entwicklung zu beschleunigen. [KW06][DES13]

EJB 3 teilt sich grundsätzlich in zwei große Bereiche auf. Das EJB Component Model und das EJB-Framework, als solches. Das Component Model von EJB sieht 3 verschiedene Arten von Beans vor. Session Beans, Message-Driven Beans sowie Entities.

- **Entity Beans** sind Objekte mit eindeutigen Identitäten. Diese Entities spiegeln i.d.R. persistierte Daten wieder. Entity Beans sind ein veraltetes Konstrukt aus der J2EE (EJB 1.0 und EJB 2.0) Welt. Diese werden heutzutage nicht mehr verwendet und sind innerhalb der API (Application Programming Interface) als "deprecated" markiert. Sie werden nur deshalb weiter angeboten, um eine Kompatibilität zu älteren Programmen zu gewährleisten. Entity Beans werden vom EJB-Container bei der Konfiguration Container-Managed-Persistence ständig mit der Datenbank synchronisiert. Diese ist i.d.R. sehr teuer, da jeder Methodenaufruf auf der Bean zu einem Update auf der Datenbank führt bzw. führen kann. Wesentlich performanter ist da die Bean-Managed-Persistence, die heutzutage in EJB 3.0 Standard ist.

- **Session Beans** bilden für gewöhnlich Prozesse ab und bieten dem Entwickler eine Schnittstelle, wie beispielsweise gegen eine Datenbank. Sie werden unterschieden in *stateful* und *stateless*. *Stateful* bedeutet, dass prozessspezifische Informationen innerhalb der Bean gehalten werden. Das könnte z.B. die Zwischenspeicherung eines vorher berechneten Wertes oder eine Benutzereingabe sein. *Stateless* hingegen bedeutet, dass die Bean keinen eigenen Zustand hält. Die *stateless* Beans werden also wie eine Art Singleton verwendet. Grundsätzlich sollte man vermeiden, *stateful* Beans zu verwenden, da diese innerhalb der Session eines Benutzers gehalten wird. Das heißt, solange eine Benutzer-Session aufrecht gehalten wird, müssen auch die Informationen aus den stateful Beans gehalten werden. Folglich muss der EJB-Container, insofern er nicht genügend Hauptspeicher zur Verfügung hat, die *stateful* Beans auf die Festplatte serialisieren, da er den Zustand der Beans nicht verlieren darf. Schreiboperationen auf eine Festplatte sind zeitintensive Operationen und sollten in jedem Fall auf ein Minimum reduziert werden. Stateless Beans hingegen werden nicht an eine Session gebunden und werden je nach Bedarf erzeugt. Nach Beendigung der Operation werden sie in einem Pool gespeichert. Benötigt später eine andere Session ebenfalls eine Bean des vorherigen Typs, so wird auf die bereits erzeugte Instanz zurückgegriffen und keine neue erzeugt.

- **Message-Driven Beans**, die asynchron Events von außerhalb fangen und diese in die eigene Applikationslogik einreihen. Die Message Driven Beans sind nur im Bereich von JMS (Jave Messaging Service) notwendig. Mithilfe von JMS lassen sich Schnittstellen nach außen realisieren. Heutzutage werden Schnittstellen aber eher auf Web- oder REST-Service-Ebene nach außen zur Verfügung gestellt.

Der EJB-Container (s. Abbildung 2.1) bietet eine Umgebung für die verschiedenen Beans sowie viele weitere Funktionen wie z.B. Transaktionen, Pooling, Caching von Resourcen, Komponentenlebenszyklen. Im Folgenden werden kurz die wichtigsten Funktionen des EJB-Containers vorgestellt. [KW06] [Gup13]

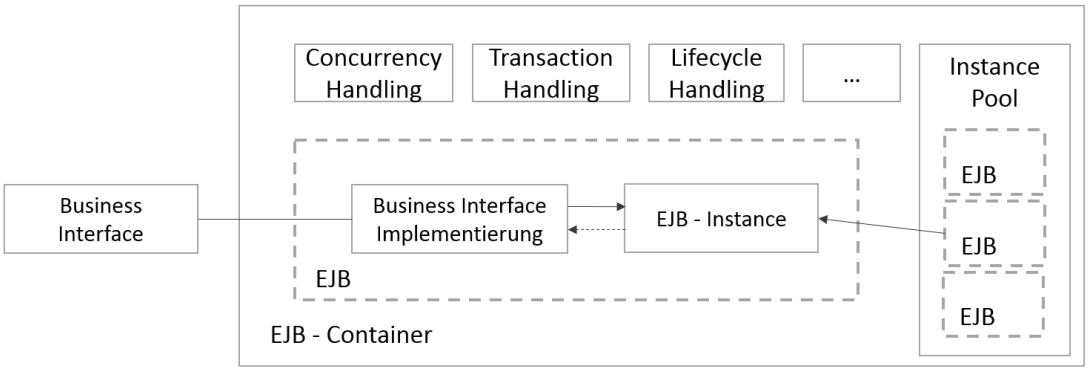

Abbildung 2.1: Der EJB-Container [DES13]

2.2.1.1 Persistenzmanagement

Im Gegensatz zu den vorherigen Versionen von EJB wurde das Persistenzmanagement aus dem EJB-Container entfernt. Für diesen Zweck wird in EJB 3.x eine Implementierung von JPA (Java Persistence API) verwendet, wie z.B. TopLink, EclipseLink oder Hibernate. [DES13]

2.2.1.2 Transaktionsmanagement

Der EJB-Container stellt ein Transaktionsmanagement bereit. Der Entwickler hat die Möglichkeit, das Transaktionsverhalten bis hinunter zur einzelnen Methode zu deklarieren. Hierbei definiert er, welche Methode einen transaktionalen Rahmen benötigt. Sobald eine solche markierte Methode betreten wird, startet der Container eine Transaktion und beendet diese bei verlassen wieder (commit, bzw. rollback bei auftreten eines Fehlers). [DES13]

2.2.1.3 Nebenläufigkeit

Grundsätzlich muss sich der Entwickler innerhalb des EJB-Kontextes nicht um konkurrierende Zugriffe kümmern. Dies übernimmt der EJB-Container, der sicherstellt, dass keine gleichzeitigen Aufrufe auf *stateless* Beans stattfinden. *Stateful* Beans werden direkt einem Client bzw. einer Session zugewiesen und diesem exklusiv zur Verfügung gestellt. Dabei wendet der Container keinerlei Prüfungen auf Nebenläufigkeit an. [KW06]

2.2.2 CDI

Da EJB einige Funktionen zur leichten Entwicklung von Web-Anwendungen fehlen, wurde innerhalb der JSR 299, CDI spezifiziert. Dabei ist CDI als Erweiterung für EJB konzipiert worden, um diverse Funktionen von EJB in einem Web-Kontext zu nutzen. Dazu zählen u.a. Scopes, die die Lebensdauer von Beans einschränken, wie auch leichtgewichtige Beans zur Darstellung und Verarbeitung von Benutzerinteraktionen.

Darüber hinaus bietet CDI diverse Möglichkeiten, fachliche Injections zu generieren. So können nicht nur Beans, sondern auch Ergebnisse über sogenannte *Qualifier* referenziert und ausgewertet werden. Dieses Verhalten wird über *Producer* realisiert. Sie generieren ein Objekt, welches an anderer Stelle genutzt (also *injected*) werden kann. Das Producen von fachlichen Ergebnissen ermöglicht eine sehr lose Kopplung zwischen Modulen und bestärkt das *Seperation of Concerns*-Prinzip. So kann eine Bean sich direkt ein fachliches Ergebnis injecten, ohne Kenntnis über den Producer zu haben. Demzufolge hat die Bean keine direkte Abhängigkeit zu dem Producer. CDI benötigt seinen eigenen Container, der EJBs über JNDI lookups (Java Naming and Directory Interface) lädt und diese im Context eines einfachen Web-Servers bereitstellen kann. Das erleichtert die Modularisierung bzw. fördert eine gute Software-Architektur in dessen die Trennung der verschiedenen Schichten möglichst strikt und lose gekoppelt ist. So lässt sich die gesamte Business Logik in EJBs umsetzen und mithilfe von CDI an eine geeignete Präsentationstechnologie weiterleiten, wie z.B. JSF (Java Server Faces) (siehe dazu Abbildung 2.2). [DES13][Gup13]

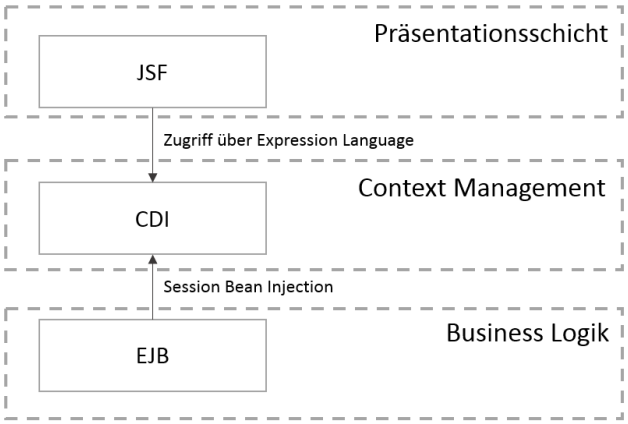

Abbildung 2.2: Zusammenspiel von JSF, CDI und EJB

Der CDI-Container bietet diverse Scopes an, um die Sichtbarkeit und Lebensdauer von Beans zu steuern. CDI definiert vier Scopes, die nach Bedarf mit eigenen Scopes erweitert werden können. In der Tabelle 2.1 werden die vordefinierten Scopes beschrieben. [DES13][Gup13][KW06]

Scope Name	Lebensdauer
Request Scope	Für die Dauer eines HTTP-Requests
Conversation Scope	Die Länge einer Conversation wird vom Entwickler festgelegt. Eine Conversation umspannt i.d.R. einen Prozess (z.B. die Bestellung eines Produktes).
Session Scope	Entspricht der Dauer einer benutzerspezifischen HTTP-Session
Application Scope	Die Lebensdauer entspricht der Laufzeit der Applikation

Tabelle 2.1: CDI-Scopes

2.2.3 Spring Framework

Das Spring Framework ist ein Java Framework mit dem Ziel, die Entwicklung (Java und Java EE) zu vereinfachen und zu vereinheitlichen. Das gesamte Framework besteht aus vielen unterschiedlichen Modulen, die je nach Bedarf importiert oder ausgeschlossen werden können. Insgesamt gibt es ca. 20 verschiedene Module. Die in Abbildung 2.3 zusammengefassten Kernmodule werden im Folgenden nun kurz erläutert. [JHD14]

2.2.3.1 Core-Container

Der Spring-Core-Container stellt die grundlegenden Funktionen des Spring-Frameworks bereit (*spring-core* und *spring-beans*). Diese umfassen u.a. die Dependency Injection Funktionen als auch die Funktionen des IoC (Inversion of Control). IoC steht für die Abgabe der Steuerungsfunktion von dem eigentlich entwickelten Programmteil, an das Framework. Das sorgt für die spezifizierte Abarbeitung der Programmteile wie z.B. Listenern. Der *spring-context* beinhaltet eine Art *registry*, die sich sowohl um die Erzeugung, als auch um die Verwaltung von Beans kümmert. Das *spring-expression*-Modul bietet alle nötigen Funktionen, um aus der Präsentationsschicht auf Beans zuzugreifen und diese auszuwerten. [JHD14]

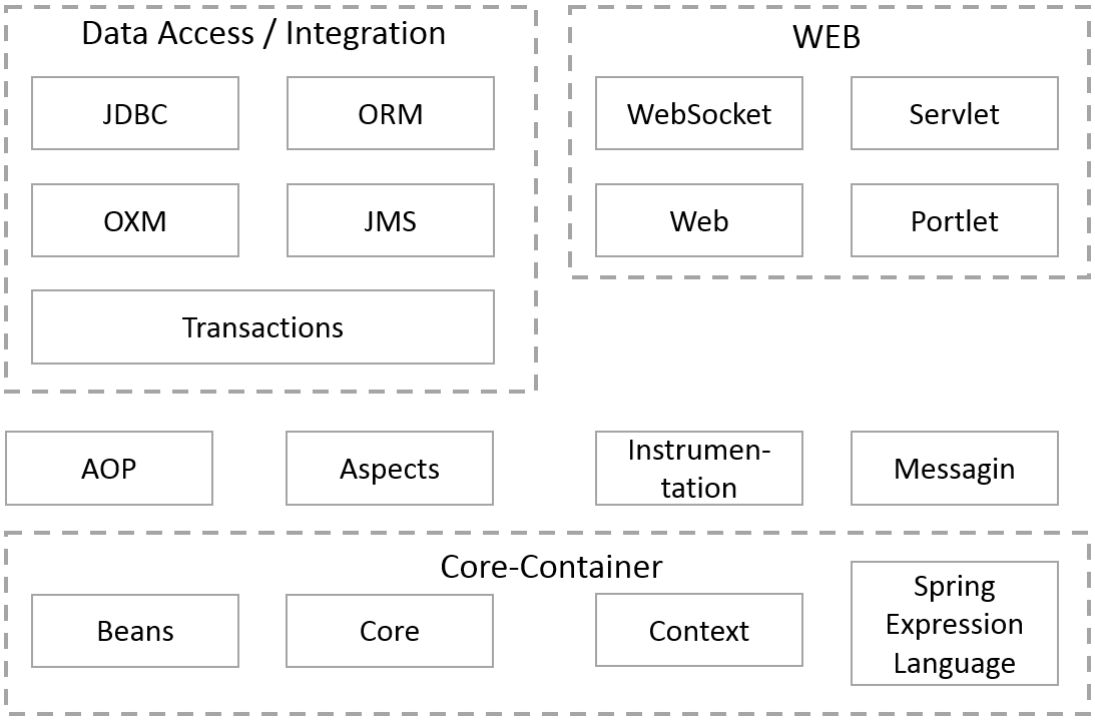

Abbildung 2.3: Spring Moduleübersicht

2.2.3.2 Data Access / Integration

Das Data Access / Integration Modul beinhaltet die Submodule JDBC (Java Database Connectivity), ORM (Object Relational Mapping), OXM (Object/XML Mapping), JMS (Java Message Service) und Transaction. Diese Module bieten vereinfachte Abstraktionsebenen über den von Java bereitgestellten Resourcen. Sie erleichtern die Ansteuerung wie auch das Resourcen-Handling. Des Weiteren werden die Resourcen um diverse Funktionalitäten erweitert, um sie im Core-Container zu integrieren. [JHD14]

2.2.3.3 Web

Das Web-Modul besteht aus den jeweils optionalen Submodulen WebSocket, Servlet und Portlet sowie dem Pflicht-Modul Web. Das Web-Modul bietet Basisfunktionalitäten, die benötigt werden, um das Übermodul in den Spring Core-Container zu integrieren. Die optionalen Module bieten darüber hinaus Zusatzfunktionalitäten, die das Entwickeln von Web-Applikationen unterstützen und erleichtern können.

- Das **WebSocket**-Modul bietet Funktionen, um eine Art "Standleitung" zwischen dem Client (i.d.R. ein handelsüblicher Web-Browser) und dem Server aufzubauen. Folglich muss der HTTP-Header nicht bei jeder Nachricht zwischen Client und Server mitgesendet werden muss. Dadurch wird die Nachricht auf ein Minimum reduziert und führt demnach zu einer schneller Übertragung.
- Das **Servlet**-Modul erleichtert die Erstellung und Registration von Servlets. Servlets nehmen direkt Client-Requests entgegen und beantworten diese mit den entsprechenden generierten Responses.

2.3 Front-End-Technologien

Das folgende Kapitel soll verschiedene Grundlagen zu den Front-End-Technologien geben, um diese im späteren Verlauf der Arbeit gegenüberzustellen. Das Kapitel beschäftigt sich mit den Technologien JSF und AngularJS.

2.3.1 JSF

JavaServer Faces (JSF) wurde 2004 durch den JSR 127 in der Version 1.0 veröffentlicht, um den damaligen Standard, Servlets / JSPs (Java Server Pages) mit klaren APIs zu versehen. Darüber hinaus musste sich der damalige Entwickler um die gesamte Kommunikationsstruktur zwischen dem Web-Client (JSP) und dem Server-Servlet kümmern. Mittlerweile ist JSF in der Version 2.2 verfügbar und Bestandteil der Java EE Spezifikation. [JCNE+14] "JSF ist ein Serverseitiges Komponentenframework zur Erstellung von Java technologiebasierten Web Applikation" [JCNE+14]. Die Kernaufgaben von JSF sind:

- Bereitstellung von APIs für
  - UI-Komponenten
  - Zustandskontrolle und -manipulation von UI-Komponenten
  - Eventmechanismen
  - Serverseitige Validation
  - Navigation [BM01]
- Bereitstellung von Tag-Libraries

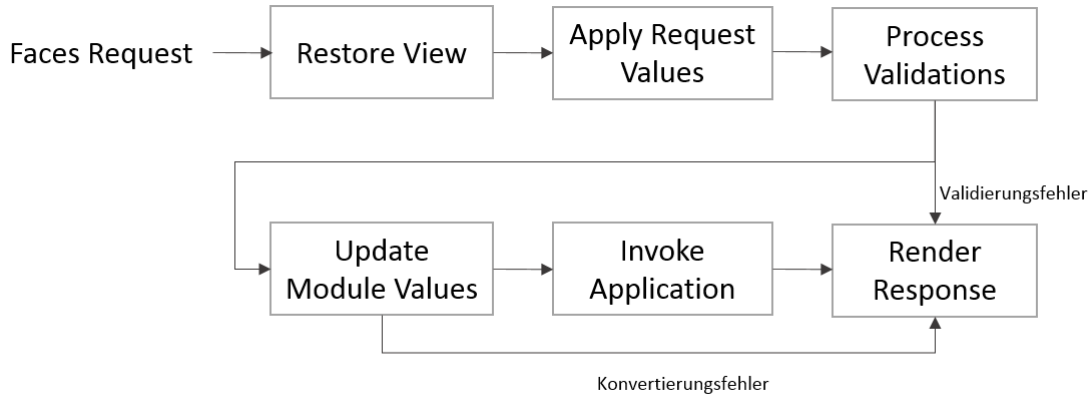

Abbildung 2.4: Der JSF-Lifecylce

Ein wichtiger Bestandteil des JSF-Frameworks ist der sogenannte Application-Lifecycle. Dieser besteht aus sechs Phasen (siehe Abbildung 2.4). Die verschiedenen Phasen lassen sich grundsätzlich in zwei Hauptphasen unterteilen - der *Execute-* und *Render-Phase*. JSF durchläuft bei jedem Request sequentiell diese Phasen. Nachfolgend werden kurz die jeweiligen Phasen und deren Aufgabe beschrieben. [JCNE+14] [BM01] [MKM10]

2.3.1.1 Faces Request

Der Faces Request unterschiedet sich in den *Initial HTTP-Request* und dem *Postback HTTP-Request*. Der *Initial Request* ist der erste HTTP-Request der auf einer JSF-View ausgelöst wird. Jeder weitere Request auf der JSF-View ist ein *Postback Request*. Bei einem Initial Request wird von JSF der *View-State* aufgebaut. Dieser beschreibt die View in einem JSF eigenen Modell. Der View-State wird daraufhin bei jeglicher Art von Kommunikation auf der View an den Server mit übertragen. Löst der Benutzer einen weiteren Request (Postback Request) auf der gleichen View aus, so wird der View-State nicht vollständig neu aufgebaut, sondern lediglich aktualisiert. [MKM10]

2.3.1.2 Restore View Phase

Die Restore View Phase baut die View auf. Sie bindet Event-Handler und Validatoren an die jeweiligen Komponenten und speichert die View im *FacesContext*. Der *FacesContext* enthält alle nötigen Informationen um einen einzelnen Request zu bearbeiten. Alle Komponenten (Validatoren, Event-Handler, Con-

verter, etc.) haben Zugriff auf den *FacesContext*. Er bildet also eine zentrale Verwaltungskomponente.

Handelt es sich um einen *Initial Request*, wird eine leere View im *FacesContext* angelegt. Bei einem *Postback Request* wird die View mit dem Zustand aus dem *FacesContext* oder dem Zustand aus dem Client wiederhergestellt. [MKM10]

2.3.1.3 Apply Request Values Phase

Innerhalb dieser Phase werden die Request Parameter decodiert und an die jeweiligen Komponenten (Validatoren, Converter, etc.) übertragen. [JCNE+14]

2.3.1.4 Process Validations Phase

JSF durchläuft alle Validatoren, die innerhalb der JSF-Konfiguration als Komponenten angemeldet wurden. Bei fehlgeschlagenen Validierungen werden die Ergebnisse oder auch Fehlermeldungen im *FacesContext* gespeichert. Enthält der *FacesContext* nach Vollendung dieser Phase Validierungsfehler, werden die folgenden Phasen übersprungen und direkt zur Phase "Render Response" (Kapitel 2.3.1.7) fortgeschritten. [JCNE+14]

2.3.1.5 Update Module Values Phase

Sollten die vorherige Phase ergeben, dass die Werte korrekt sind, werden diese ins Model übertragen (Model umfasst hier die verschiedenen Komponenten, die an die View geknüpft sind). Sollte es bei der Konvertierung der Werte Fehler geben, wird direkt die Phase "Render Response" eingeleitet, welche dann die jeweiligen Fehler darstellt. [JCNE+14][MKM10]

2.3.1.6 Invoke Application Phase

Während dieser Phase werden alle "Application-level Events" verarbeitet. Diese schließen z.B. den Form-Submit oder ein Redirect zu einer anderen View mit ein. Ein weiterer Bestandteil dieser Phase ist die Ausführung aller Java-Action-Methoden, die an JSF-Komponenten verknüpft sind. [MKM10]

2.3.1.7 Render Response Phase

Bei einem Initial Request wird innerhalb dieser Phase der sogenannte *component-tree* im View-State aufgebaut. Dieser enthält die Komponenten, die innerhalb des Views dargestellt werden sollen. Bei einem *Postback Request* existiert der *component-tree* bereits.

Darüber hinaus wird bei einem *Postback Request* geprüft, ob die View im *FacesContext* Fehler (z.B. Validierungsfehler oder Konvertierungsfehler) und Meldungen enthält. Im Falle eines Fehlers wird der PageFlow nicht fortgesetzt, sondern die gleiche View mit einem veränderten View-State wiederhergestellt. Dieser beinhaltet die Validierungsergebnisse, die mit den jeweiligen Web-Komponenten verknüpft wurden. [JCNE+14][MKM10]

2.3.2 AngularJS

AngularJS ist ein JavaScript Framework zur Entwicklung von dynamischen Web-Applikationen. Entwickelt wurde es von Google und 2009 als Open-Source-Projekt veröffentlicht. Angular ist inzwischen in der Version 1.3.0 verfügbar und wurde unter anderem von Google für einen Teil der Entwicklung von YouTube genutzt. [GS13] Die Hauptziele des Frameworks sind:

- Reduktion des Codes
- Modularisierbarkeit (vor allem durch Delegates) von JavaScript-Funktionaliät
- Verwendung von Datentypen [GS13]

Im Allgemeinen soll AnularJS den Einsatz von Javascript in einer Web-Applikation vereinfachen und übersichtlicher gestalten. Das Hauptkonzept dahinter ist der Fokus auf Modularisierbarkeit von Funktionaltiäten. Dies fördert nicht nur die Vermeidung von Code-Duplikaten, sondern auch die Wartbarkeit durch gute Strukturierung der vorhandenen Funktionen. AngularJS setzt darüber hinaus sehr stark auf REST-Services (Representational State Transfer). Ein Großteil der Server Kommunikation wird hier über REST-Services abgebildet, die mithilfe von AngularJS sehr leicht und mit wenigen Zeilen Code implementiert werden können. [BBT14][Bay02]

Seitens der Entwickler lassen sich neben den vielen positiven Bemerkungen zu AngularJS auch ebenso viele negative Kommentare über das Framework finden.

Grundlegend wird bemängelt, dass sehr viel in Angular implizit passiert. Das hat eine Intransparenz zur Folge, die für Laien nur schwer nachvollziehbar ist. Das beste Beispiel hierfür ist das sogenannte "Two-way data-binding". Angular erlaubt es für alle Arten von Events oder Veränderungen von Variablen *$watch*-Punkte zu definieren. Diese können über den gesamten Code verstreut sein und werden immer dann aufgerufen, wenn eine Veränderung stattfand bzw. ein Event geworfen wurde.[Kos15]

Ein weiterer Nachteil bei Angular ist die fundamentale Weiterentwicklung des Framworks. Aktuell ist Angular in der Version 1.3.15 verfügbar. Die Entwickler hinter dem Framework haben allerdings bereits angekündigt, dass sich sehr viel der grundlegenden Funktionalität des Frameworks mit der Version 2 ändern wird. Dies gibt natürlich Anlass zu hinterfragen, ob es aktuell sinnvoll ist, AngularJS für neue Projekte zu nutzen. Bislang ist noch unklar, wann die Version 2 veröffentlicht werden soll. Sollten zukünftige Browser-Versionen mit Angular untersützt werden wollen, muss zwangsläufig die Migration auf die Version 2 vorgenommen werden. Dies führt bislang noch unbekannte Migrationsaufwände mit sich und stellt ein aktuelles Risiko dar. [Sot14][Kos15]

Kapitel 3

Analyse

Dieses Kapitel dient dazu, die Anforderungen an die zu erstellende Software festzulegen sowie eine Evaluation der zu verwendenden Technologien, unter Betrachtung der vorher definierten Anforderungen.

3.1 Anforderungen

Innerhalb dieses Unterkapitels werden die Anforderungen an die Anwendung, die innerhalb dieser Arbeit erstellt wird, erfasst. Diese unterteilen sich in funktionale und nicht funktionale Anforderungen.

3.1.1 Funktionale Anforderungen

Die Anforderungen werden im Folgenden als kurze "User-Stories" formuliert. User Stories beschreiben Funktionen oder ähnliches, welche einen Mehrwert der zu erstellenden Software beiträgt. Häufig werden User-Stories in einem agilen Softwareentwicklungsprozess eingesetzt, um anstehende Aufgaben oder Funktionen aus Sicht des Benutzers zu beschreiben. Um im späteren Verlauf der Arbeit einen besseren Bezug auf die User-Stories nehmen zu können, werden diese mit laufenden Nummern versehen. [Coh04]

BI01 - Kunde - Ich möchte mich registrieren können

Um den Kunden nicht bei jeder Bestellung nach seinen Daten zu fragen, sollen sich die Kunden am Shop registrieren können. Dies bietet weiterhin die Möglichkeit, seine Bestellungen zu verfolgen.

BI02 - Kunde - Ich möchte ein Steckperlenbild erstellen

Der Kunde soll sein eigenes Bild innerhalb des Browsers gestalten können. Dazu muss ihm der Shop innerhalb des Steckperlenkonfigurator (Designer) verschiedenfarbige Perlen anbieten, die er in einem beliebigen Motiv kombinieren kann.

BI03 - Kunde - Ich möchte ein Bild hochladen, welches in ein Steckperlenbild umgerechnet wird

Der Steckperlenkonfigurator (Designer) muss eine Funktion bieten, ein beliebiges Bild hochzuladen. Dieses wird dann in ein Steckperlenmotiv umgerechnet und dem Kunden angezeigt. Anschließend soll er noch die Möglichkeit erhalten, das Bild nachzubearbeiten.

BI04 - Kunde - Ich möchte ein oder mehrere Steckperlenbilder bestellen

Der Kunde benötigt einen Warenkorb, in dem er verschiedene Bilder platzieren kann. Die Bilder im Warenkorb können dann auf Wunsch des Kunden bestellt werden. Hierzu werden Informationen wie z.B. die Lieferadresse benötigt.

BI05 - Kunde - Bestelle ich ein Bild, soll dieses auf mögliche Schwachstellen analysiert werden

Sobald ein Benutzer eine Perle setzt, muss die Software diese Aktion auswerten. Bevor der Kunde die Bestellung abschickt, muss er auf potentielle Schwachstellen hingewiesen werden. Zu den Schwachstellen gehören unter anderem Verbindungen mit lediglich einer Perle und unverbundene Teilstücke. Verbindungen mit nur einer Perle sind im Bügelprozess nur schwer durchzuführen und können mitunter sehr schnell brechen. Solch ein Bild sollte vom Kunden also nicht bestellbar sein.

BI06	- Kunde	-	Ich möchte Steckperlenbilder anderer Benutzer betrachten und bestellen können

Der Kunde kann entscheiden ob, er ein Steckperlenbild, welches er erstellt hat, veröffentlichen möchte. Diese veröffentlichten Steckperlenbilder sollen von anderen Kunden, wie in einer Art Katalog, betrachtet und bestellt werden können.

BI07	- Kunde	-	Ich möchte meine Bestellungen verfolgen können

Der Shop muss dem Benutzer die Möglichkeit bieten, seine getätigten Bestellungen zu verfolgen. Dazu muss eine Funktion geschaffen werden, die seine Bestellungen auflistet und die jeweiligen Status zu den Bestellungen anzeigt.

BI08	- Kunde	-	Ich möchte die Möglichkeit haben, verschiedene Zahlungsarten für meine Bestellung auszuwählen

Der Kunde soll unter verschiedenen Zahlungsarten wählen können. Dazu zählen:

- per Nachname,
- per Vorkasse,
- per Guthaben.

Die Zahlungsart "Guthaben" bedeutet, dass ein Kunde, persönliches Guthaben von einem Mitarbeiter erwirbt (wie z.B. ein Gutschein). Dieser Mitarbeiter schreibt dann über einen geeigneten Mechanismus das Guthaben dem Kunden innerhalb der Plattform zu. Der Kunde kann daraufhin sein Guthaben zum bezahlen von Bildern nutzen.

BI09	- Mitarbeiter	-	Ich möchte einem Kunden, Guthaben zuschreiben können

Der Mitarbeiter benötigt eine Möglichkeit, persönlich entgegengenommenes Geld, einem Benutzer des Shops gutschreiben zu können. Das Guthaben kann in Folge der Kunde benutzen, um Bilder zu bezahlen.

BI10	-	Mitarbeiter	- Das Gutschreiben von Guthaben zu einem Kunden soll historisiert werden

Schreibt ein Mitarbeiter einem Benutzer Guthaben zu, so muss dieses in geeigneter Form protokolliert werden, sodass eine spätere Betrachtung der Guthaben-Historie zu einem Benutzer möglich ist. Die Guthaben-Historie eines Kunden darf nicht editierbar bzw. löschbar sein.

BI11	-	Mitarbeiter	- Ich möchte die bisher angefallenen Bestellungen einsehen können

Der Mitarbeiter benötigt einen Bereich zur Betrachtung der bisher getätigten Bestellungen der Benutzer. Die Bestellungen sollen nach Status gefiltert werden können.

BI12	-	Mitarbeiter	- Ich möchte eine Bestellung in den Druck geben

Der Shop muss Bestellungen separat in den Druck geben können. Dabei setzt der Mitarbeiter den Status einer Bestellung um, woraufhin diese Bestellung an den Steckperlendrucker übermittelt wird. Nachdem die Bestellung gedruckt wurde, muss die Funktion für weitere Bestellungen wieder freigegeben werden.

BI13	-	Mitarbeiter	- Nachdem eine Bestellung gedruckt wurde, soll der Shop eigenständig den Status der Bestellung aktualisieren

Sobald ein Bild gedruckt wurde, soll der Shop die Bestellung von dem Status "In Bearbeitung" in den Status "Bearbeitet" versetzen.

BI14	-	Mitarbeiter	- Ich möchte den Status einer Bestellung ändern können.

Der Mitarbeiter benötigt die Möglichkeit, folgende Statusübergänge für Bestellungen vollziehen zu können

- "Unbearbeitet" → "In Bearbeitung"
  Der Mitarbeiter bereitet den Druck vor
- "Bearbeitet" → "wird versandt"
  Das Bild ist gedruckt und wird für den Versand vorbereitet

- "wird versandt" → "Versendet"
  Das Bild befindet sich auf dem Postweg

BI15	-	Mitarbeiter	-	Ich möchte andere Benutzer als Mitarbeiter kennzeichnen

Initial wird ein Benutzer als Mitarbeiter gekennzeichnet. Nachfolgend kann dieser, über eine dedizierte Benutzersuche, andere Benutzer des Shops die Rolle "Mitarbeiter" zuweisen. Folglich erhalten ebenfalls auch diese Benutzer den Status des Mitarbeiters.

BI16	-	Mitarbeiter	-	Ein Bild, welches für andere Kunden freigegeben wurde, soll zunächst von mir freigegeben werden

Mitunter kann es dazu kommen, dass Kunden Bilder erstellen, die nicht auf der Plattform dargestellt werden sollen. Dazu zählen z.B. geschmacklose, gesetzlich verbotene (Hackenkreuze, o.ä.) oder politisch inkorrekte Bilder. Um die Freigabe solcher Bilder zu verhindern, muss zunächst ein Mitarbeiter das Bild überprüfen, bevor es veröffentlicht wird.

BI17	-	Mitarbeiter	-	Der Preis eines Steckperlenbildes soll abhängig von der Anzahl verschiedener Farben berechnet werden

Die Preisberechnung eines selbst erstellten Steckperlenbildes soll nicht nur von der Anzahl der Perlen abhängig sein. Der Drucker muss, je nach verwendeter Farbe, anders gerüstet werden. Hier muss ein Mitarbeiter manuell die jeweils erforderlichen Farben in die jeweiligen Magazine des Druckers einfüllen. Diese Tätigkeit soll mit in die Preisberechnung einbezogen werden. Eine nähere Erläuterung zu der Preiszusammensetzung wird im Folgenden Unterkapitel 3.1.1 stattfinden.

Preisgestaltung

Der Preis eines selbst zusammengestellten Steckperlenbildes setzt sich aus der Anzahl von Steckperlen sowie der Anzahl von unterschiedlich-farbigen Steckperlen zusammen. Dies geht aus der Anforderung BI17 hervor. Dabei besitzt eine Perle einen fixen Preis von 0.01 €. Der Druck eines Bildes erfordert es,

dass der Steckperlendrucker mit den jeweiligen Farben gerüstet ist. Dies führt zu Rüstkosten, die im Preis enthalten sein müssen. Die Rüstkosten wurden innerhalb dieser Arbeit auf 10% Aufschlag geschätzt. Um die Preisberechnung deutlich zu machen, wird diese anschließend an einem vereinfachten Beispiel verdeutlicht.

Das Beispielbild 3.1 besteht aus ca. 1.000 Perlen (1.008 genau, der einfachheitshalber wird in Folge von 1.000 Perlen ausgegangen). Die Kosten für die Perlen belaufen sich also bislang auf

$$1.000 \text{ Perlen} \times 0,01 \text{ €} \qquad = 10,00 \text{ €}$$

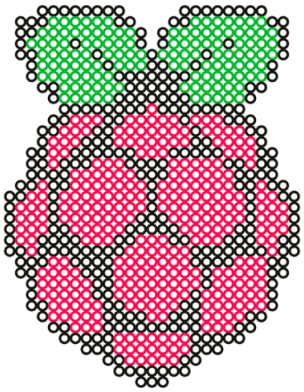

Abbildung 3.1: Beispielbild für die Preisberechnung

Zusätzlich zu den reinen Stückkosten, wird ein Rüstkosten-Multiplikator berechnet. Dieser erhöht den Preis je unterschiedlicher Farbe, um 10%. In unserem Beispiel werden 3 unterschiedliche Farben genutzt: hellgrün, hellrot und schwarz.

$$(3 \div 10) + 1 = 1,3$$

Der Preis von bislang 10,00 €wird also um Faktor 1,3 erhöht. Somit ergibt sich ein Endpreis des Bildes von 13,00 €.

3.1.2 Nicht funktionale Anforderungen

Die nichtfunktionalen Anforderungen stellen Qualitätskriterien der Software dar. Dabei leiten sich diese teilweise aus den funktionalen Anforderungen ab. Die erhobenen, nichtfunktionalen Anforderungen werden im Folgenden beschrieben.

- **Benutzerschnittstelle**: Das User Interface (UI) sollte ansprechend und intuitiv gestaltet werden. Dabei sollen gängige Standards der UI-Gestaltung eingehalten und berücksichtigt werden.

- **Passwortsicherheit**: Um die Sicherheit des Benutzerkontos zu verstärken, muss ein Benutzerpasswort aus mindestens 6 Zeichen bestehen. Mindestens eines dieser Zeichen muss ein Großbuchstabe sein.

- **Plattformunabhängigkeit**: Der Shop wird als Webapplikation realisiert, weshalb hier eine generelle Plattformunabhängigkeit gegeben ist. Der Shop sollte allerdings, für diverse Browser optimiert werden. Aus diesem Grund ist es notwendig mindestens die gängigsten Browser in ihren aktuellen zu unterstützen. Dazu zählen:

  - Internet Explorer 11
  - Firefox, Version 33
  - Chrome, Version 38

  Darüber hinaus wird bei der Entwicklung eine Mindestauflösung von 1024 x 768 Pixel zu Grunde gelegt. Wie innerhalb des Artikels [And15] beschrieben, ist diese Auflösung noch immer weit verbreitet und wird nur sehr langsam durch höhere Auflösungen ersetzt.

- **Stabilität**: Fehler, die zur Laufzeit der Applikation auftauchen, müssen über geeignete Fehlermeldungen dem Benutzer präsentiert werden und mit Anweisungen versehen werden, wie der Benutzer weiter verfahren soll.

3.2 Technologieentscheidung Backend

Die aktuellen Versionen der verschiedenen Technologien bieten inzwischen diverse Möglichkeiten, um die Komplexität der Entwicklung von Applikationen weit möglichst zu reduzieren. Die Grundidee dabei ist es, viele Infrastrukturelle arbeiten auf ein Minimum zu reduzieren, damit sich der Entwickler auf die Implementierung der Fachlichkeit (Business-Logik) konzentrieren kann. In der Vergangenheit, fiel die Wahl der Backingtechnologie häufig auf JavaEE-Standards (EJB und CDI) oder auf Spring. Ein weiteres, bekanntes Framework ist JBoss Seam. Da dieses allerdings seit einiger Zeit nicht mehr aktiv weiterentwickelt wird, werden diese Technologie, nicht weiter innerhalb dieser Arbeit beschreiben.

3.2.1 JEE7 versus Spring

Im Folgenden Kapitel werden die Frameworks Spring und einige Technolgoien aus dem JEE7-Universum gegenübergestellt.

3.2.1.1 JEE7

JEE ist ein Standard zum Aufbau und zur Entwicklung von Enterprise Applikationen. Dieser Standard wird von verschiedenen Herstellern implementiert. Dadurch schafft man keine direkte Abhängigkeit zu einem konkreten Anbieter und kann frei zwischen den Herstellern wechseln. JEE in der Version 7 umfasst die Technologien EJB 3 und CDI. Diese Technologien ermöglichen einen schnellen Einstieg in die fachliche Entwicklung mit einem minimalen Konfigurationsaufwand. Ein weiterer Vorteil, der mit den beiden Technologien einhergeht, ist die Typsicherheit bei Injections. Injections werden über Annotation gesteuert, die es dem Entwickler sogar ermöglichen, fachliche Objekte in Beans zu injecten. Gemeint ist hier das *Producer*-Konzept von CDI. Ein *Producer* erzeugt ein Objekt bzw. eine Menge von Objekten die über weitere Annotationen qualifiziert werden (z.B. @AllCustomer, @CurrentUser, ...). [Del12][Gup12][Rah14]

3.2.1.2 Spring

Spring wurde damals mit dem Fokus entwickelt, die Arbeit eines Entwicklers auf das zu beschränken, was seine Software ausmacht - die Fachlichkeit. Dieses war mit den damaligen Möglichkeiten (J2EE) sehr komplex und schwierig, was Spring vereinfachen wollte, um die Effizienz der Softwareentwicklung zu maximieren. Dabei erfordert Spring an vielen Stellen expliziete Konfigurationen (z.B. der ApplicationContext, Beans, ...). Das ermöglicht Spring eine gewisse Art von Flexibilität, da die Konfigurationen zur Deploymentzeit ausgetauscht werden können, ohne den jeweiligen Code neu zu kompilieren. Die Konfiguration über XML-Konfigurationsdateien steigert aber im Gegenzug die Komplexität und verringert die Effektivität. 2009 wurde SpringSource (der damalige Hersteller von Spring) von VMWare aufgekauft. Somit verbirgt sich hinter dem Framework ein kommerzielles Unternehmen, dem es obliegt, ob und in welcher Form Spring weiterentwickelt werden soll. Im Gegensatz hierzu liegt JEE eine große Community zugrunde, welche JSRs (Java Specification Requests) her-

vorbringt. Die JSRs werden dann eine Prozess zugeführt, bei dem viele weitere Experten die Änderungswünsche beurteilen und bewerten. Erst nach längege Prüfung wird dann ein solcher JSR umgesetzt und die jeweilige Spezifikation angepasst. Wird nun Spring zur Entwicklung verwendet, können zwischen zwei verschiedenen Version möglicherweise große Unterschiede entstehen. Darüber hinaus ist man zwingend an den einen Hersteller und dessen Unterstützung gebunden.

Ein weiteren Unterschied zwischen Spring und JEE Applikationen ist der, dass JEE Applikationen nur auf Application-Servern betrieben werden können. Die Application-Server halten die Implementierung der Spezifikation, gegen welche die entwickelt wird. Demzufolge liegt die Verantwortung der Kompatibilität und Aktualität der Bibliotheken (Implementierung der Spezifikationen) bei den Herstellern der Application-Servern. Das macht es für den Fachentwickler leichter, seine Applikation zu aktualisieren. Bei Spring werden alle Bibliotheken in dem ausgelieferten Artefakt verpackt und liegen somit in der Verantwortung des Fachentwicklers. Application-Server waren zur Zeiten von J2EE große und unflexible Programme, die nur schwer wartbar waren. Dahingegen bot Spring einen großen Vorteil, dass die ausgebrachten Artefakte leicht auf z.B. einem Apache Tomcat deployed werden konnten. Dieser ist ein leichtgewichtiger Servlet Container der nur einen minimalen Konfigurationsaufwand benötigt. In der heutigen Zeit sind Application-Server aber einen Schritt weiter und bieten gute und einfache Mittel, um Applikationen leichter wartbar zu machen. Darüber hinaus kapseln die Application-Server die Datenbankanbindung. Sollte also aus Sicherheitsgründen die Entwicklung keinen Zugriff auf Produktivdaten haben, dann lässt sich dies leicht über die Konfiguration des Servers durch den Betrieb ermöglichen.

Aufgrund der Tatsache, dass Spring relativ kurze Entscheidungswege hat, hinsichtlich neuer Funktionen bzw. Verbesserungen, ist das Framework meist Vorreiter bei Anforderungen des Marktes. Dies war z.B. der Fall bei der Entwicklung einer Schnittstelle für Soziale Netzwerke oder der Anbindung einer Cloud. Diese Features waren bereits lange in Spring verfügbar, bevor sie den zeitaufwändigen Prozess der JSRs hinter sich hatten und Einzug in die JEE-Spezifikation hielten (siehe z.b. JSR 357).

3.2.1.3 Ergebnis

Spring ist in einer Vielzahl von Entwicklungsprojekten überaus sinnvoll. Allerdings benötigt man in den meisten Kundenprojekten nicht die Flexibilität und Anbieterabhängigkeit. Die Entwicklung einer Applikation für einen Kunden ist i.d.R. ein kostspieliges und zeitaufwändiges Unterfangen. Dabei sollte der Fokus auf Entwicklungsgeschwindigkeit, Nachhaltigkeit und Wartbarkeit gelegt werden. Dies wird durch die JEE-Standards garantiert. Der Lebenszyklus einer solchen Applikation ist mitunter sehr lang, weshalb Software-Architekten und Entwickler auf zukunftssichere Technologien bzw. leicht zu migrierende Technologien setzen sollten. Bei Spring liegt die Verantwortung der abhängigen Bibliotheken bei den Entwicklern der Applikation. Aus diesem Grund sind Upgrades auf höhere Versionen bei Spring Applikation weitaus teurer als Applikationen, die auf JEE setzen. Hier werden die abhängigen Bibliotheken von dem jeweiligen Application Server verwaltet. Natürlich fallen dabei auch Migrationskosten an. Die Höhe der Kosten steht dabei aber in keinem Verhältnis zu der Migration einer Spring Applikation.

Spring entstand aus der Not der Entwickler, dass J2EE über die Maßen komplex war. Dies ist heutzutage allerdings nicht mehr der Fall. EJB 3.0 und CDI bieten dem Entwickler einfache und schnelle Möglichkeiten, effizient Business-Logik umzusetzen.

3.3 Technologieentscheidung Frontend

Wie bereits in den Kapiteln 2.3.1 und 2.3.2 beschrieben eignen sich JSF und AngularJS hervorragend zur Umsetzung von UI-Funktionalität. Enger verknüpft mit dem Java Universum ist dabei JSF, da es Teil der JEE-Spezifikation ist und nahtlos mit EJB und CDI integriert werden kann. AngularJS kommuniziert mit dem Server hingegen über ausschließlich REST-Services. Hierbei ist zu beachten, dass Angular oftmals noch als "unausgereift" gilt. Darüber hinaus erhält Angular in naher Zukunft ein grundlegendes Update. Dies stellt den Entwickler aktuell natürlich vor die Frage, ob er AngularJS für Enterprise Applications einsetzt, vor dem Hintergrund, der unabsehbaren Migrationsaufwände auf die nächste Version.

JSF und AngularJS sind beides aktuelle Frameworks zur Umsetzung von UI-Funktionalität. JSF hat bereits Einzug in unzählige "Business"-Applications

gefunden und ist aufgrund seines Lifecycles attraktiv für stark, formular-basierte Anwendungen. Ebenfalls bietet JSF 2.2 einfache und saubere Möglichkeiten an, Funktionalitäten zu kapseln (Seperation of concerns) mithilfe der sogenannten "Composite-Components". Ein Nachteil des Frameworks ist jedoch die Speicherung des UI-States auf der Serverseite. In der heutigen Zeit, haben die Clients (Browser) viel Leistung zur Verfügung, weshalb es unnötig erscheint, einen Großteil der Last auf den Server zu verteilen.

AngularJS ist ein leichtes und mächtiges Framework zur Entwicklung der UI-Schicht. Das Framework bietet dabei sehr gute Möglichkeiten, Funktionalitäten zu kapseln und damit den Code sauber zu strukturieren. Ein Vorteil von AngularJS gegenüber reinem JavaScript ist die mögliche Typisierung von Variablen. Der größte Nachteil von Angular ist die fundamentale Weiterentwicklung des Frameworks. Hierbei werden aktuell grundlegende Funktionalitäten und Annahmen überarbeitet, was eine Migration auf höhere Version mitunter überaus teuer machen kann.

Im Bereich des Front-End ist es nahezu essentiell, auf aktuelle Technologien zu setzen, da diese z.B. die Kompatibilität zu den aktuellen Browser Versionen herstellt.

Veraltet ein eingesetztes UI-Framework (keine Weiterentwicklung seitens des Herstellers) so werden häufig aktuelle Trends und Weiterentwicklungen der Browser zu einem hohen Wartungsaufwand. Unter diesem Gesichtspunkt müssen sich die Entwickler fragen, welche Vorteile bietet Angular und an welchen Stellen sind diese durch keine anderen Maßnahmen zu erreichen.

Java-Script wird voraussichtlich im Laufe dieser Arbeit nur für die Umsetzung des Designers genutzt (also das tatsächliche Setzen von Perlen). Darüber hinaus besteht keine Notwendigkeit, Java-Script einzusetzen. Neben den Java-Script Anteilen, innerhalb des Designers, ist der Rest der Anwendung auf eine formularbasierte Applikation fokusiert, in dessen JSF seine großen stärken besitzt. So bietet JSF einfache Methoden um beispielsweise Validierungslogik, Konvertierungsmechaniken wie auch Modularisierung von Komponenten (Composite Components) umzusetzen.

Zusammenfassend ist JSF die Technologie, die bei der Umsetzung der, im vorherigen Kapitel beschriebenen, Anforderungen den größten Vorteil verspricht. Die geringen Anteile von JavaScript innerhalb der Anwendungen bedingen kein eigenes Framework und können in reinem JavaScript implementiert wer-

den. Aus diesem Grund wird AngularJS zur Umsetzung des Web-Shops nicht verwendet.

3.4 Zusammenfassung

Innerhalb dieses Kapitels wurden zunächst die Anforderungen genauer definiert. Hierbei wurden die funktionalen Anforderungen aus der Sicht der jeweiligen Benutzerrolle in Form von User-Stories formuliert. Nachfolgend wurde genauer die Preisberechnung der zukünftig zu erstellenden Steckperlenbilder erläutert. Im Anschluss wurden die nicht funktionalen Anforderungen beschrieben. Diese erfassen Bereiche wie beispielsweise die Stabilität des Web-Shops, die Benutzerschnittstelle (User Interface) sowie die Plattformunabhängigkeit.

Neben den Anforderungen, wurden die Technologieentscheidungen für die Bereiche Front- und Back-End getroffen. Hierbei standen im Back-End die Technologien Spring, EJB3 und CDI zur Auswahl. Im Bereich des Front-Ends wurden JSF und AngularJS verglichen.

Im Back-End viel die Wahl auf EJB 3 und CDI. Diese Technologien sind perspektivisch stabiler und daher leichter zu warten, da sie Java Enterprise Standard Technologien sind. Spring und EJB in Verbindung mit CDI sind überaus mächtige Frameworks bzw. Technologien. Grundsätzlich lassen sich Anforderungen ähnlich schnell in den beiden Welten umsetzen, weshalb es schwer fällt, Vor- bzw. Nachteile dem anderen gegenüber zu finden. Oftmals werden Technologien nach persönlicher Präferenz ausgewählt, da keine Technologie der beiden große Unterschiede aufweist.

Im Front-EndBereich wurde JSF gewählt. Unter Berücksichtigung der bereits analysierten Anforderungen, stellte sich diese Technologie als die geeignetere heraus. Ebenfalls ein wichtiger Faktor war ein absehbares, großes Update von AngularJS, welches in absehbarer Zeit hohe Aufwände herbeiführen würde.

Kapitel 4

Konzeptionierung

Unter Berücksichtigung der im vorangegangenen Kapitel erarbeiteten Ergebnisse, wird innerhalb dieses Kapitels die Applikationen konzipiert. Zunächst wird die Auswahl der Technologien beschreiben bevor dann das Datenbankschema hergeleitet wird. Im weiteren Verlauf des Kapitels werden dann die unterschiedlichen Komponenten des Web-Shops untersucht und entworfen. Abschließend wird das Design-Pattern beschrieben, welches der anschließenden Implementierung zu Grunde liegt.

4.1 Technologien

Aus dem vorangegangen Kapitel gehen verschiedene Technologien als "gesetzt" in den Web-Shop mit ein (siehe auch Abbildung 4.1). Als User-Interface-Framework wird JSF 2.2 (Implementierung: Mojarra) mit Bootstrap genutzt. Bootstrap ist dabei ein CSS und JavaScript Framework, welches einfache und gute Vorlagen bietet, um ein portables-Design zu erschaffen. Man spricht in diesem Zusammenhang auch häufig von einem "responsive"-Design. Das bedeutet, dass sich das Design an den Vorgaben des Besuchers richtet. Wird eine

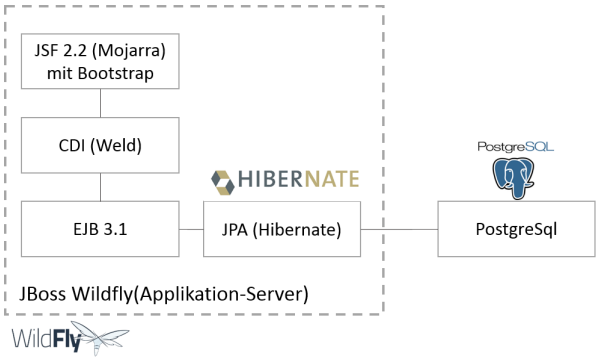

Abbildung 4.1: Technologieauswahl

Webseite beispielsweise von einem Smartphone besucht, hat dieses ganz andere Anforderungen als ein normaler Desktop-PC. Somit soll sich das Design je nach Auflösung des Clients verändern und auf seine Ansprüche anpassen. Die Kommunikation zwischen JSF und EJB 3.1 findet über CDI (Implementierung: Weld) statt. CDI bietet mithilfe seiner Producer-Mechaniken gute und leichte Möglichkeiten, um "fachliche" Injections zu vollziehen (z.B. der aktuelle Benutzer, die beliebtesten Bilder, etc.). Als Bindeglied des User-Interfaces zu der Datenbank wird EJB 3.1 eingesetzt. Hier wird die gesamte Business-Logik abgelegt und der Zugriff auf die Datenbank gekapselt.

4.2 Datenbank

Innerhalb dieses Unterkapitels wird zu Beginn das Datenbankschema in Form eines ER-Models (Entity-Relationship-Modell) hergeleitet. Anschließend folgt eine detaillierte Beschreibung der Persistierung von Steckperlenbildern.

4.2.1 Schema

Der Web-Shop wird von insgesamt drei unterschiedlichen Benutzergruppen verwendet - Anwender, Kunden und Mitarbeiter. Ein Anwender stellt dabei einen noch nicht angemeldeten oder registrierten Benutzer dar. Registriert sich ein Benutzer, wird er zunächst als Kunde klassifiziert und ein Eintrag in der Tabelle *User* (siehe Abbildung 4.2) wird angelegt. Sollte es sich hierbei um einen Mitarbeiter handeln, muss die Rolle des Benutzers vom Administrator oder eines anderen Mitarbeiters angepasst werden. Des weiteren kann der Benutzer während der Bestellung eine Adresse in seinem Benutzerkonto hinterlegen. Neben der Adresse kann ein Kunde oder Mitarbeiter Guthaben besitzen. Dieses Guthaben wird von einem Mitarbeiter zugewiesen und als Summe am *User* gespeichert (*User.userCredit*). Ebenfalls wird bei der Zuweisung von Guthaben, ein Eintrag in der *UserCreditHistory* erzeugt. Dieser umfasst die Menge des zugewiesenen Guthabens, den Zeitpunkt an dem es zugewiesen wurde, eine Beschreibung (wie z.B. "Hinzugefügt" oder "Bestellung", wenn Guthaben durch eine Bestellung abgebucht wurde) und eine Referenz auf den Mitarbeiter, der das Guthaben modifiziert hat.

Ein Kunde kann, nach seiner Anmeldung am System, Bilder erstellen. Diese werden mit einer Referenz auf den Kunden in der Tabelle *Picture* gespeichert.

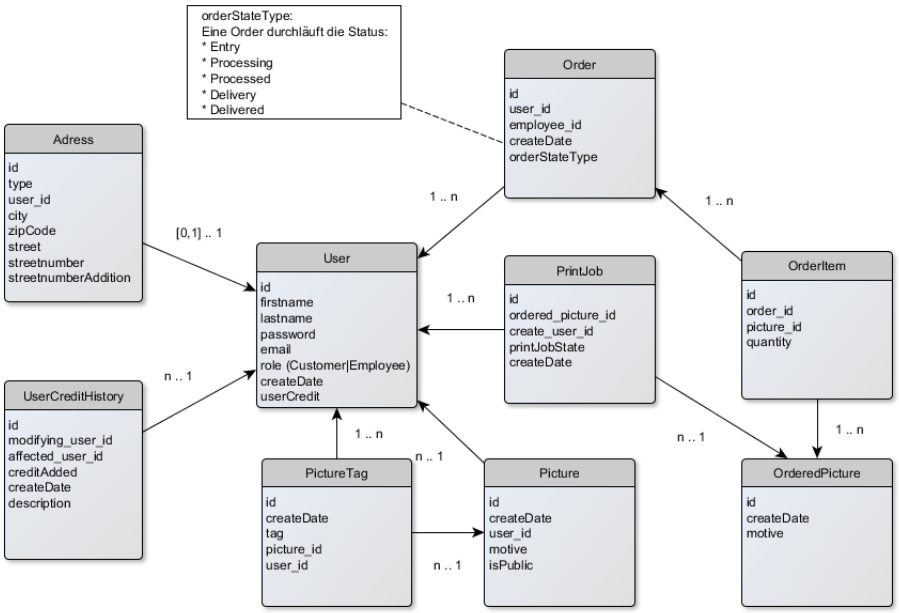

Abbildung 4.2: ER-Digramm

Zudem hat er die Möglichkeit, das Bild für die Öffentlichkeit freizugeben (*Picture.isPublic*), um es so anderen Kunden zu ermöglichen, das Bild zu bestellen. Zusätzlich kann der Kunde sogenannte *PictureTags* vergeben. Diese repräsentieren Stichwörter. Erstellt ein Kunde ein Bild von einem Fußball, so vergibt er z.B. die Stichwörter "Sport" und "Ball". Innerhalb der später beschriebenen Bildersuchen, können dann andere Benutzer nach diesen Stichwörtern (also *PictureTags*) suchen. Sie können also eine weitere Klammer um verschiedene Arten von Bildern bilden. Die *PictureTags* referenzieren ein Bild wie, auch den Benutzer, der diese Stichwörter angelegt hat.

Ein angemeldeter Kunde kann eine Bestellung aufgeben, welche dann in der Relation *Order* persistiert wird. Diese enthält *OrderItems*, welche Bestellpositionen repräsentieren. Ein *OrderItem* referenziert dabei ein *OrderedPicture* und definiert die Menge, in der das Bild bestellt wurde. Sobald eine Bestellung abgeschickt wurde, wird eine Kopie des *Picture* in *OrderedPicture* erzeugt. Dies soll verhindern, dass sich das ursprüngliche *Picture* während des Bestellvorgangs möglicherweise verändern könnte. Für jedes bestellte Bild wird also eine Kopie erzeugt, damit der Kunde genau das Bild erhält, welches er zum Zeitpunkt der Bestellung, beauftragt hatte.

Nachdem eine Bestellung persistiert wurde, durchläuft diese verschiedene Status (*Order.orderStateType*). Diese werden von verschiedenen Mitarbeitern innerhalb des Web-Shops bearbeitet. Befindet sich eine *Order* in dem Status

Farbwert	Farbcodierung
Leer	00
Blau	01
Rot	02

Tabelle 4.1: Beispielhafte Farbcodierung

"Processing" (also "in Bearbeitung"), so müssen die Bilder der Bestellung zunächst gedruckt werden. Dazu erzeugt ein Mitarbeiter einen *PrintJob*, der das *OrderedPictre* und den Mitarbeiter, der den Druck in Auftrag gegeben hat, referenziert. Der *PrintJob* besitzt, genau wie die *Order*, ein eigenes Status-Modell (*PrintJob.printJobState*). Dieser definiert den Statuts des *PrintJobs* und kann folgende Ausprägung annehmen: "*WAITING_FOR_PRINT*", "*PRINTED*" und "*FAILED*". Innerhalb der Applikation wird ein Scheduler diese Zustandsübergänge steuern.

4.2.2 Perlenkodierung

Ein Perlenbild besteht aus einer Matrix verschiedenfarbiger Perlen. Dabei ist die maximale Größe abhängig von dem größtmöglichen Motiv des Steckperlendruckers. Eine Herausforderung hierbei ist es, nicht jede Perle einzeln mit ihrer Koordinate zu speichern. Um eine Konkrete Vorstellung davon zu bekommen, um wie viele Werte es sich dabei handelt, soll zunächst ein vereinfachtes Beispiel mit einer 5 x 5 Matrix (siehe Abbildung 4.4) dienen. Dabei sind die Positionen 7, 8 und 9 mit roten und die Positionen 13 und 19 mit blauen Perlen besetzt. Der Kodierung liegt die Farbkodierung aus der Tabelle 4.1 zugrunde.

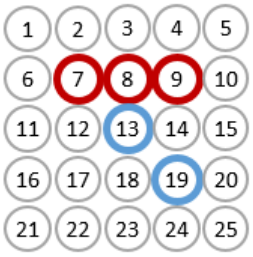

Abbildung 4.3: Perlenkodierung

4.2.2.1 Speichern aller Werte

Insgesamt gibt es 25 Positionen, die jeweils mit einem zweistelligen Zeichen beschrieben werden. Dies führt dazu, dass in der Datenbank, 25 x 2 Zeichen → **50 Zeichen** gespeichert werden müssen. Es ist davon auszugehen, dass der Steckperlendrucker eine Matrix von mindestens 50 x 50 Positionen drucken kann. Das würde bedeuten, wenn jede Farbe mit durch 2 Zeichen repräsentiert wird - 50 x 50 x 2 → 5.000 Zeichen in der Datenbank, pro Bild. Da alle Werte gespeichert werden, werden auch nicht gesetzte Perlen persistiert. Ein Zeichen in der Datenbank nimmt ca. 1 Byte ein. Daraus folgt, dass jedes Bild **2,5 KB** groß ist.

4.2.2.2 Value-Point-Encoding

Eine gängige Methode zur Reduktion des Speicherbedarfs in der Geoinformationsdatenerfassung ist das Value-Point-Econding. Dieses sieht vor, dass Daten nicht einzeln, sondern im Kontext gespeichert werden.

Dabei wird das Bild zeilenweise durchlaufen. Ändert sich der Zustand (also die Farbe), wird die letzte Zeilenendnummer für diesen Wert gespeichert. In dem vereinfachten Beispiel wird das Bild von links nach rechts, Zeile für Zeile durchlaufen und es ergibt sich die Wertetabelle aus Abbildung 4.4.

Farbwert	Endpunkt
00	6
02	9
00	12
01	13
00	18
01	19
00	25

Abbildung 4.4: Value-Point-Encoding Wertepaare

Um die Werte eindeutig von einander zu trennen, lassen sich sogenannte *Espace-Zeichen* definieren. Dieses ist notwendig, um die Wertepaare voneinander zu trennen. Aus der Farbkodierung geht hervor, dass eine Farbe immer durch 2 Zeichen (maximal 100 verschiedene Farben) gekennzeichnet ist. Der Endpunkt kann bei einer 50 x 50 Matrix, maximal den Wert 2500 einnehmen. Somit ist es sinnvoll, die Wertepaare mit z.B. einer Raute (#) zu trennen. Für das Beispiel ergibt sich also folgende Zeichenkette:

```
006#029#0012#0113#0018#0119#0025#
```

Insgesamt also **33 Zeichen**. Je nachdem, ob ein Bild aus vielen verschiedenen

Farben besteht, steigt die Anzahl der Zeichen (da höhere Anzahl von Wertepaaren) und überschreitet sogar schnell die Zeichenanzahl die benötigt würden, wenn alle Zeichen hintereinander gespeichert würden.

4.2.2.3 Speichern der Werte ohne Leerräume

Bei Steckperlenbildern ist es eher unwahrscheinlich, dass große Flächen in der selben Farbe, zeilenübergreifend gespeichert werden. Dies würde bedeuten, dass die Speichergröße bei dem Value-Point-Encoding die Speicherung aller Werte übersteigt.

Es ist davon auszugehen, dass viele Felder nicht mit Perlen besetzt werden. Dabei ist es also vollkommen unnötig, diese nicht existenten Perlen zu speichern. Die genauen Maße der Zeichnung sind bekannt, weshalb eine Speicherung der Werte plus die Anzahl der folgenden Leerräume ausreichen würde, um die Zeichnung, aus der kodierten Form, wiederherzustellen. Wichtig ist jedoch, dass die Zeichenanzahl der Leerräume, eindeutig von denen der Farbwerte abgegrenzt werden können. Theoretisch kann die Anzahl der aufeinanderfolgenden Leerräumen ein-, zwei- oder dreistellig sein. Um dies eindeutig abzugrenzen, folgt vor und nach der Anzahl der Leerräume eine Raute (#). Das Beispiel würde in dieser Art und Weise, folgende Zeichenkette ergeben:

```
#6#020202#3#01#5#01#6#
```

Es ergibt sich eine Zeichenkette mit **22 Zeichen**.

4.2.2.4 Ergebnis

Da es sehr wahrscheinlich ist, dass immer wieder viele Leerräume und unterschiedliche Perlenfarben aufeinanderfolgen, ist das Speichern mithilfe der Value-Point-Encoding-Methode nicht sinnvoll. Das Speichern aller Werte inklusive Leeräume, erzeugt sehr viele gleiche Werte hintereinander, da der Leerraum vermutlich der häufigste Wert sein wird. Das Speichern der Werte ohne Leeräume hat an dem Beispiel gezeigt, dass es sich hierbei um eine sinnvolle Art und Weise der Kodierung handelt, um die Speichergröße von Steckperlenbildern zu minimieren.

4.3 Komponenten des Shops

Die Grundfunktionen des Shops werden sein:

- Erstellung von eigenen Bildern (Designer)
- Erstellung eines Steckperlenbildes aus der Vorlage eines hochgeladenen Bildes (Perlenkonverter)
- Schwachstellenanalyse eines Bildes
- Bestellvorgang
- verschiedene Arten von Galerien
- Bildersuche
- Bestellübersicht der Mitarbeiter
- Benutzersuche bzw. -ansicht
- Bildfreigabe

Diese Grundfunktionen werden in eigenen Komponenten implementiert, welche die Funktionalitäten größtmöglich kapseln sollen. Innerhalb dieses Unterkapitels werden die verschiedenen Komponenten konzipiert und ggf. anhand von Mockups verdeutlicht werden.

4.3.1 Designer

Eine Hauptkomponente des Shops wird es sein, Bilder aus verschieden farbigen Perlen zu erstellen. Diese Komponente gesaltet sich wie eine Art "Mal-Programm" (siehe auch 4.5), in dem der Benutzer eine Farbpalette angeboten bekommt, die möglichst, die Farben der tatsächlichen Perlen widergespiegelt. Zunächst wählt der Benutzer eine Farbe aus der Farbpalette aus. Daraufhin kann er in einem definierten Bereich diese in beliebigen Motiven kombinieren. Darüber hinaus muss der Benutzer die Möglichkeit haben, Perlen wieder zu entfernen.

Das setzen des Bildes muss auf der Client-Seite realisiert werden. Daher muss der Designer zu großen Teilen mithilfe von JavaScript umgesetzt werden. Wählt der Benutzer eine Farbe aus der Farbpalette aus, wird diese Farbe als aktiv

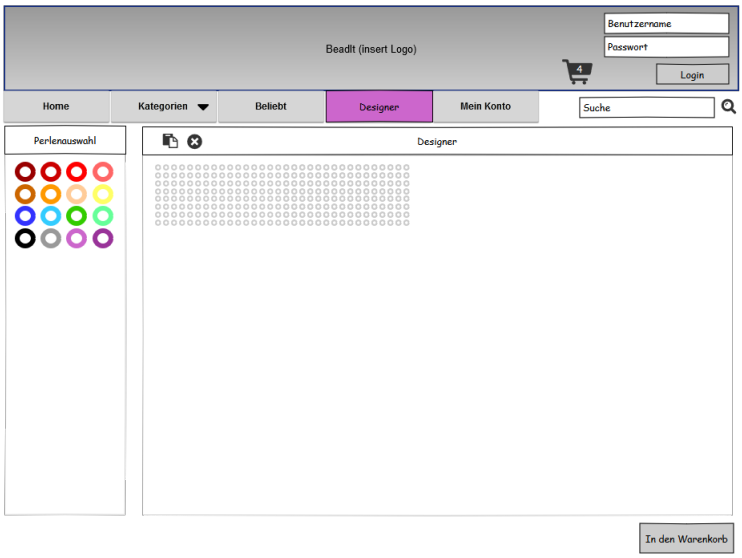

Abbildung 4.5: Mockup des Steckperlendesigners

markiert. Das setzen der Perlen soll in einem Canvas-Objekt passieren. Canvas-Objekte sind eine Neuerung aus HTML5, welche es ermöglichen, direkt im Browser zu zeichnen. Es wird im Grunde wie eine Leinwand benutzt, welche mithilfe von JavaScript bemalt wird. Nachdem also der Benutzer seine Farbauswahl getroffen hat, klickt er nun in den Canvas Bereich. Dieser muss initial mit einem Listener versehen werden, der auf Mausklicks reagiert. Registriert der Listener nun einen solchen Klick auf das Canvas-Objekt, wird die Position der Perle bestimmt, die gerade gesetzt werden soll. Hierzu muss die Klick-Position in einen Index umgerechnet werden. Das Canvas-Objekt ist dabei in Zellen unterteilt, wobei eine Zelle genau so groß ist wie eine Perle. Dies verhindert es, dass Perlen sich überlappen. Der Listener zeichnet daraufhin einen Kreis auf dem Canvas an den entsprechenden X und Y Koordinaten mit der gerade gewählten Farbe. Um zu verhindern, dass der Zustand des Bildes nur im Browser des Benutzers gehalten wird, wird nach dem Zeichnen ein REST-Call an den Server geschickt, der die Informationen über Position und Farbe an den Server überträgt. Dies ermöglicht es, den Zustand des Bildes, auch nach versehentlichen verlassen des Designers wiederherzustellen.

Nachdem der Benutzer dann mit dem Erstellen seines Bildes fertig ist, wird der aktuelle Zustand des Bildes vom Server aus, in der Datenbank, persistiert. Wie die Bilder in der Datenbank abgelegt werden, wird innerhalb des Kapitels 4.2.2 genauer erläutert.

4.3.2 Perlenkonverter

Neben dem Designer soll es ebenfalls möglich sein, Perlenbilder aus bestehenden Bildern zu generieren. Das Vorgehen dabei ist, dass der Benutzer ein beliebiges Bild hochlädt, welches daraufhin auf dem Server in ein Perlenbild umgewandelt wird. Das umgewandelte Bild soll dann dem Benutzer angezeigt werden, sodass es ihm möglich ist, Korrekturen am Bild vorzunehmen.

Das hochladen eines Bildes ist mithilfe von JSF 2.2 leich zu handhaben, da eine neue Komponente hinzugekommen ist, die das hochladen von Dateien übernimmt. Auf dem Server muss dazu eine Funktion erstellt werden, die die hochgeladene Datei entgegennimmt.

Über das hochgeladene Bild wird dann auf der Server-Seite ein Raster gelegt. Eine Zelle des Rasters entspricht dabei der Größe einer Steckperle. Innerhalb der sich ergebenden Zellen wird dann die vorherrschende Farbe ermittelt. Daraufhin sucht der Konverter nach der Steckperlenfarbe, welche dieser am ähnlichsten ist. Dieses Vorgehen wird für alle Zellen wiederholt, bis jede Zelle durch eine bestimmte Steckperle repräsentiert wird. Zuletzt wird der Steckperlendesigner mit den jeweilige Steckperlen initialisiert und dargestellt. Nun hat der Benutzer die Gelegenheit, manuelle Änderungen an dem Steckperlenbild vorzunehmen.

4.3.3 Schwachstellenanalyse

Wie in der Anforderung BI05 beschrieben, möchte ein Kunde, Feedback zu dem Bild erhalten, welches er gerade erstellt hat. Aus der gesammelten Druckerfahrung haben sich einige potentielle Schwachstellen ergeben, die bereits zu Erstellungszeit eines Bildes identifiziert werden können. Aus diesem Grund soll der Kunde bereits bei Erstellung eines Bildes darauf hingewiesen werden, dass sein Bild womöglich nicht gedruckt bzw. schnell kaputt gehen würde.

Zur Zeit der Erstellung dieser Masterarbeit war lediglich eine potentielle Schwachstelle bekannt, dass eine Perlenverbindung nur dann stabil ist, wenn mindestens zwei benachbarte Perlen an die neu gesetzte Perle angrenzen.

Um diese Schwachstellen zeitnah dem Benutzer nahezubringen, soll jede Platzierung einer Perle validiert und in geeigneter Form dem Benutzer mitgeteilt werden. Die Platzierung einer Perle wurde im Kapitel 4.3.1 bereits konzipiert.

So wird jede einzelne Positionierung per REST-Schnittstelle an den Server übertragen. Die Schwachstellenanalyse umfasst mitunter eine komplexe Validierungslogik, weshalb diese zwingend auf dem Server liegen muss, um sie gut zu vertesten. Die Perlenpositionierung erfolgt, wie bereits erwähnt, mithilfe eines asynchronen REST-Calls an den Server. Hier wird die Perle im serverseitigen Modell hinterlegt und die Schwachstellenvalidierungslogik wird durchlaufen. Ergeben sich dabei Schwachstellen, werden diese an den Client (Browser) übertragen, welche dieser dann farblich im Canvas-Objekt markiert. Der Benutzer erhält also ein unmittelbares Feedback über seine gerade platzierte Perle und kann sein Bild dahingehend korrigieren.

Bevor der Benutzer sein Bild bestellen kann, muss erneut diese Schwachstellenanalyse durchgeführt werden und eine Bestellung so lange blockiert werden, bis alle Schwachstellen aus dem Bild eliminiert wurden. Eine Bestellung eines Bildes kann erst dann stattfinden, wenn der Benutzer alle Schwachstellen aus seinem Bild eliminiert hat. Sollte das Bild weiterhin Schwachstellen aufweisen, muss eine geeignete Fehlermeldung ausgegeben werden.

Sicherlich ist es sinnvoll, die Schwachstellenanalyse in Zukunft zu erweitern, sobald mehr Erfahrungen mit dem Druck der Steckperlenbilder gewonnen wurde. Identifizieren die Mitarbeiter im späteren Verlauf des Betriebes weitere potentielle Schwachstellen, so sollte die Analyse um die weitere Prüfungen erweitert werden.

Die Perlenverbindung ist also nur dann stabil, wenn eine Perle an mindestens zwei anderen, zusammenhängenden Perlen angrenzt. Lediglich dann kann im Druck eine feste Bügelverbindung zwischen den verschiedenen Perlen gewährleistet werden. Wird also eine Perle vom Benutzer platziert, wird diese an den Server übertragen. Der Server betrachtet daraufhin die Umgebung der gerade gesetzten Perle. In Abbildung 4.6 wird dies an einem einfachen Beispiel verdeutlicht. Die Validierung betrachtet die Umgebung im Uhrzeigersinn und sucht nach zwei benachbarten Perlen. Findet die Validierungslogik diese nicht, weist sie die gerade betrachtete Position (mittlere, blau markierte) als Schwachstelle aus und übermittelt diese an den Client (Browser). Der Client nimmt diese Ausweisung auf und verzeichnet sie im Canvas Objekt des Browesers.

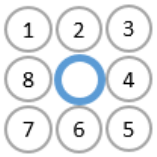

Abbildung 4.6: Perlenverbindungsanalyse

4.3.4 Bestellvorgang

Der Benutzer kann eigene Bilder, wie auch freigegebene Bilder anderer Benutzer in seinem Warenkorb hinterlegen. Der Zustand seines Warenkorbs soll dabei über alle Seiten des Shops hinweg transparent in der Menüleiste angezeigt werden. Innerhalb des Warenkorbs wird dem Benutzer angezeigt, welche Bilder er aktuell in diesen gelegt hat. Sollte der Benutzer mit seiner Auswahl zufrieden sein, kann er nach Angabe seiner Adresse die Bilder bestellen. Es erscheint eine weitere Zusammenfassung seiner Bestellung mit den jeweiligen Bildern. Bestätigt er diese Zusammenfassung, wird eine Bestellung persistiert.

Die Bestellung enthält dabei nicht die Referenzen auf die tatsächlichen Bildern, es werden stattdessen Kopien der Bilder angelegt um zu verhindern, dass diese im Nachhinein noch verändert werden können.

Die Abarbeitung der Bestellung geschieht in mehreren Schritten. Diese Schritte werden über verschiedene Status repräsentiert. Der Status kann dabei folgende Werte annehmen:

- **Unbearbeitet** - Die Bestellung ist eingegangen, jedoch noch von keinem Mitarbeiter bearbeitet (*Entry*)
- **in Bearbeitung** - Die Bestellung wird gerade von einem Mitarbeiter ausgedruckt (*Processing*)
- **Bearbeitet** - Die Bestellung wurde ausgedruckt und wartet auf den Versand (*Processed*)
- **wird versandt** - Die Bestellung befindet sich gerade im Warenausgang (*Delivery*)
- **Versendet** - Die Bestellung hat das Haus verlassen und befindet sich auf dem Postweg (*Delivered*)

4.3.5 Galerien

Um Benutzer zum Kauf von Bildern zu animieren, soll früh die Aufmerksamkeit des Benutzers auf populäre und neue Bilder gelenkt werden. Dazu sollen auf der Hauptseite des Shops zwei Galerien platziert werden, die eine Übersicht der populärsten und neuesten Bilder anzeigt (siehe auch Abbildung 4.7). Die Popularität eines Bildes wird in Form von Views (Bilderansichten) gemessen.

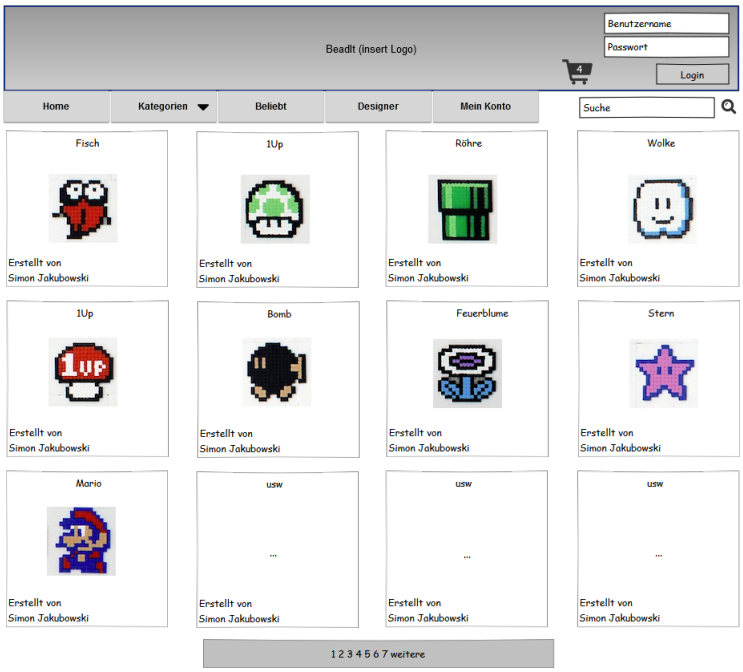

Abbildung 4.7: Mockup der Startseite

Schaut sich ein Benutzer ein Bild in der Detailansicht an (vergrößerte Darstellung des Bildes), wird in der Datenbank ein Zähler inkrementiert. Die Galerie der populärsten Bilder ist eine Ansicht der Bilder in absteigender Sortierung der Views. Neueste Bilder werden absteigend nach ihrem Erstellungsdatum sortiert.

4.3.6 Bildersuche

Der Benutzer benötigt eine Funktion zum Suchen von Bildern. Dabei soll er nach der Kategorie, dem Bildernamen oder einem Stichwort suchen können. Zusätzlich soll er die Ergebnismenge nach Popularität und Erstellungsdatum sortieren können.

4.3.7 Bestellübersicht der Mitarbeiter

Eine zentrale Komponente für die Mitarbeiter ist die Bestellübersicht. Diese bietet einen Überblick über alle Bestellungen sowie Filtermöglichkeiten. So soll der Mitarbeiter die Bestellungen nach ihrem Status filtern und eine Art "Freitextsuche" über diese erhalten. Neben der reinen Ansicht der Bestellungen, kann der Mitarbeiter hier die einzelnen Bilder einer Bestellung in den Druck

geben können. Gedruckte Bilder sollen besonders gekennzeichnet sein, wie z.B. mithilfe einer Beschriftung. Die Ansicht einer Bestellung soll des weiteren Aufschluss auf die Lieferadresse geben. So kann der Mitarbeiter, nur durch die Ansicht der Bestellung, alle notwendigen Informationen zu dieser erhalten, um sie abzuarbeiten.

4.3.8 Benutzersuche bzw. -ansicht

Der Mitarbeiter benötigt eine Ansicht aller registrierten Benutzer. Die Benutzersuche bzw. Benutzeransicht soll folgende Anforderungen erfüllen:

- Der Mitarbeiter möchte einem Kunden Guthaben zuschreiben können (BI09),
- das Gutschreiben von Guthaben zu einem Kunden soll historisiert werden (BI10),
- der Mitarbeiter möchte andere Benutzer als Mitarbeiter kennzeichnen (BI15).

Die Ansicht bzw. Suche soll also eine Auflistung aller registrierten Benutzer sein. Der Mitarbeiter benötigt darüber hinaus Funktionen, um die Rolle eines Benutzers ändern zu können. In diesem Zusammenhang soll der Mitarbeiter auswählen können, ob es sich bei dem Benutzer um einen "Mitarbeiter" oder "Kunden" handelt. Neben dieser beschriebenen Rollenverwaltung, soll der Mitarbeiter einem Benutzer ein beliebiges Guthaben zuschreiben können. Das Guthaben eines Benutzer muss in einem seperaten Bereich ausgezeichnet werden. Über eine weitere Funktion, soll der Mitarbeiter die Historie der Guthabenzuweisung eines Benutzers betrachten können. Diese darf nicht editierbar sein. Die Benutzersuche bzw. -ansicht ist der Benutzerrolle "Mitarbeiter" vorbehalten. "Einfache" Kunden dürfen keinen Zugriff auf diesen Bereich erhalten, dieses muss über eine Art Recht o.ä. abgesichert werden.

4.3.9 Bildfreigabe

Die Anforderung BI16 beschreibt die Restriktion, dass Bilder zunächst einem Mitarbeiter-Review unterzogen werden sollen, bevor es veröffentlicht wird. Es gibt beliebige Beispiele für unangebrachte, beleidigende oder politisch inkorrekte Bilder. Dies macht es zwingend notwendig, Bilder erst durch Mitarbeiter

freizugeben, bevor sie im Web-Shop veröffentlicht werden.

Jedes zu veröffentliche Bild trägt zunächst ein Flag, das dieses als "nicht freigeben" markiert. Im Mitarbeiterbereich des Shops soll es dann einen Bereich geben, der all diese Bilder auflistet. Diese können dann individuell für die Öffentlichkeit von einem Mitarbeiter freigegeben werden. Während dieser Zeit kann das Bild weder in der Bildersuche (siehe Kapitel 4.3.6), noch in den verschiedenen Galerien (siehe Kapitel 4.3.5) gefunden werden. Der Benutzer hat allerdings die Möglichkeit, das Bild im Bereich "Meine Bilder" mit dem Hinweis "Wartet auf Freigabe" zu finden.

4.4 Druck

Wie innerhalb der Anforderung BI12 beschrieben, soll der Mitarbeiter innerhalb der Bestellübersicht (siehe auch Kapitel 4.3.7) Bilder in den Druck geben können. Dazu wird der bereits häufig genannte Steckperlendrucker an den Web-Shop angebunden. Der Steckperlendrucker wird auf einem Raspberry Pi ausgeführt, der über das lokale Netzwerk des Web-Servers erreichbar ist. Die Architektur des Steckperlendruckers sieht als externe Schnittstelle ein Programm namens *bbconsole* vor, welches über die Shell angesteuert wird. Das Programm bietet derzeit folgende Paramter:

Parametername	Bedeutung
-s (–size)	Mit diesem Parameter kann die Größe des Bügelperlenbildes ausgewählt werden. Mögliche Werte sind "small" (25x25 Perlen), "medium" (40x40 Perlen) und "large"(54x54 Perlen). Der Standardwert dieses Parameters ist "small".
-p (–plotter)	Über diesen Parameter kann das Druckverfahren ausgewählt werden. Es gibt zwei alternative Werte: "line", wenn der Druck zeilenweise ausgeführt werden soll und "color", wenn der Druck farbweise ausgeführt werden soll. Letzterer Wert ist der Standard.
-d (–dither)	Mithilfe dieses Parameters wird Dithering zur Farbanpassung verwendet.
–show-statistics	Das Programm zeigt nach erfolgreichem Druckvorgang Druckstatistiken an.

Tabelle 4.2: bbconsole Shell Parameter [Zie13]

Ein Beispielaufruf (beschrieben innerhalb [Zie13]) für die bbconsole ist:

```
./bbconsole --size small --plotter color --show-statistics
./testimage.png
```

Um diese Konsolenanwendung korrekt zu verwenden, muss der Web-Shop zunächst das zu druckende Bild in das lokale Dateiverzeichnis des Steckperlendruckers per **scp** (Secure Copy) kopieren und dann per **scp** (Secure Shell) den Aufruf der bbconsole absetzen.

Innerhalb des Web-Shops werden die Steckperlenbilder nicht in Form von tatsächlichen Bilddateien gespeichert, sondern in einer kodierten Form innerhalb der Datenbank gehalten (siehe auch Kapitel 4.2.2). Somit muss zunächst die kodierte Form des Bildes in ein tatsächliches Bild umberechnet werden, bevor dieses dann auf das System des Steckperlendruckers kopiert wird. Darüber hinaus muss die Ansteuerung des Druckers an einer zentralen Stelle (Applikationsweit) implementiert werden. Diese Ansteuerung muss dafür Sorge tragen, dass der Steckperlendrucker nicht mit Druckaufrufen "überlaufen" wird. Die Abarbeitung des Druckers erfolgt sequenziell. Zu viele, gleichzeitige Druckaufträge könnten zu einem Verlust von Druckaufträgen führen oder den Drucker überfordern.

Der Druck sollte also von der eigentlichen Benutzer-Thread-Ausführung entkoppelt werden und applikationsweit, zentral gelöst sein. Möchte also ein Mitarbeiter ein Bild drucken, wählt er dieses aus der Bestellübersicht (siehe auch Kapitel 4.3.7) aus und klickt auf den "Drucken"-Button. Dieser erzeugt daraufhin einen Eintrag in der Print-Job Tabelle der Datenbank mit den nötigen Informationen, die den Druck betreffen (wie z.B. die ID des Bildes, die ID des Mitarbeiters, etc.). Diese Print-Jobs werden in regelmäßigen Abständen von einem Scheduler (Logik, die in einem fest definierten zeitlichen Abstand ausgeführt wird (z.B. jede 15 Minuten)) abgefragt. Dieser prüft ob neue oder offene Print-Aufträge eingetroffen sind. Findet er einen neuen Print-Job, so erzeugt er das Bild aus dem Auftrag und überträgt dieses an den Steckperlendrucker. Nachfolgend gibt er den Befehl zum eigentlichen Druck des Bildes. Nachdem der Druck abgeschlossen ist, aktualisiert der Scheduler den Status den Print-Jobs innerhalb der Datenbank.

4.5 Design-Pattern

Der Web-Shop soll nach den Prinzipien des **MVC-Patterns** (Model View Controller) umgesetzt werden (siehe Abbildung 4.8). Das Pattern besagt, dass es verschiedene Zuständigkeiten von Funktionalitäten gibt. Dabei gilt z.B. eine Hibernate Entität (Darstellung eines persistierten Objekts) als Modell der Software. Die View beschreibt die Logik zur spezifischen Darstellung und der Controller zur Steuerung und Umsetzung der Business-Logik. Diese drei Bereiche sollten möglichst voneinander getrennt werden. Ein Beispiel dafür ist z.B. eine Applikation mit einem Java-Swing User-Interface. Liegt die gesamte Business Logik vermischt mit der Darstellungslogik, so ist es so gut wie unmöglich die Darstellungsform auszutauschen, z.B. in eine Web-Anwendung. Sind die Bereiche jedoch sauber getrennt, so kann eine beliebige Darstellungstechnologie eingesetzt werden, die sich an die Controller wendet. [Sal09]

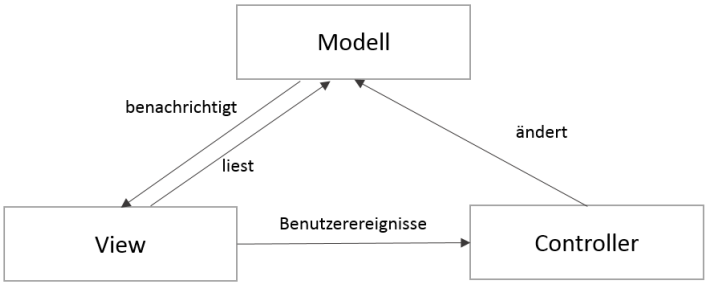

Abbildung 4.8: MVC-Pattern

4.6 Zusammenfassung

Dieses Kapitel beinhaltet zunächst eine Technologieauswahl der zuvor beschriebenen und verglichenen Technologien. Dabei wurden für den Front-EndBereich JSF sowie EJB 3.0 und CDI für den Back-EndBereich ausgewählt.

Ferner wurde das Datenbankschema hergeleitet und darüber hinaus einige Verfahren zur Kodierung der Steckperlenmotive beleuchtet. Hierbei hat sich das Verfahren am effektivsten erwiesen bei dem die Farbwerte (repräsentiert durch Farbindizes) ohne Leerräume gespeichert werden. Dabei wird das Bild zeilenweise kodiert und Leerräume ergeben sich aus der Anzahl, fehlender Perlen zwischen der vorherigen und der nachfolgenden Perle.

Nachfolgend wurden die einzelnen Komponenten konzipiert, die zur Erfüllung der vorher analysierten Anforderungen nötig sind. Zu den Komponenten zählen mitunter der Designer, der Perlenkonverter, die Schwachstellenanalyse sowie viele weitere.

Kapitel 5

Implementierung

Das nachfolgende Kapitel beschreibt das Vorgehen zur konkreten Implementierung der im vorherigen Kapitel beschriebenen Konzepte. Dazu werden zunächst unterstützende Methoden und Maßnahmen zur Entwicklung beschrieben, bevor dann auf die Implementierung der Komponenten eingegangen wird.

5.1 Entwicklungsumgebung und -vorgehen

Die Entwicklung des Steckperlen-Shops wird, wie bereits in den vorangegangenen Kapiteln, in der Programmiersprache Java verfasst. Zur Entwicklung wird dabei eine IDE (**I**ntegrated **D**evelopment **E**nvironment) verwendet, konkret die IDE "Intellij Idea" von der Firma Jetbrains. Der zu erstellende Code wird mithilfe von GIT versioniert. Dieses bietet u.a. die Sicherheit, dass der Code nicht auf einem lokalen Rechner verloren geht und um eine geräteunabhängige Entwicklung zu ermöglichen. Darüber hinaus lassen sich verschiedene Versionen von Daten miteinander vergleichen, die es erleichtern, gewisse Änderungen nachzuvollziehen.

Eine Java Web-Anwendung wird in der Regel in einem WAR- oder EAR-Archiv ausgeliefert. Um diese zu Erstellen, wird Apache Maven verwendet. Apache Maven ist ein Java-Build-Management Tool, welches neben der Einbindung von Plugins, wie z.B. zum Bau von WAR-Archiven, auch eine zentrale Stelle zur Definitionen von Fremdbibliotheken bietet. Diese werden zentral in einer Datei (das sogenannte POM-File) festgehalten und bei dem ersten Bauen des Projekts heruntergeladen, insofern sie sich noch nicht im Benutzerverzeichnis

befinden. Das erleichtert die Entwicklung einer Software mit mehreren Personen deutlichst, da nicht jede neu eingebundene Bibliothek unter den Entwicklern ausgetauscht werden muss.

5.1.1 Test-driven-development

Die Umsetzung der Komponenten soll bei der Entwicklung weitestgehend testgetrieben entwickelt werden. Das sogenannte **T**est-**d**riven-**d**evelopment (TDD) sieht es vor, das zunächst Unit-Tests für die zu entwickelnde Komponente erstellt werden. Dies sorgt dafür, dass der Entwickler sich zunächst über die Schnittstelle und die Funktion der Methode Gedanken macht, bevor er sich mit technischen Details beschäftigt. Das Vorgehen sorgt darüber hinaus für eine gute Testabdeckung, das wiederum die Funktionalität des Codes auch in der Zukunft sicherstellt. TDD soll innerhalb dieses Projektes bei der Entwicklung der gesamten Business-Logik eingesetzt werden. Die Präsentationslogik wiederum ist nur sehr schwer abzutesten. Hierbei müsste jegliche Logik, die JSF bereitstellt, nachgebaut oder mithilfe von Frameworks wie "Arquillian" nachempfunden werden. Projekte aus der Vergangenheit haben jedoch gezeigt, dass diese sogenannten UI-Tests überaus wartungsaufwändig sind und nur einen geringen Mehrwert schaffen. Aus Gründen den eben genannten Gründen wird innerhalb dieses Projekts auf Tests von reiner UI-Logik (z.B. die Auswahl einer Combobox, werden alle UI-Komponenten korrekt dargestellt, ...) verzichtet. Alles was jedoch über die reine UI-Logik hinausgeht muss mit geeigneten Unit-Tests, vertestet werden (z.B. Transformation von User-Eingaben, Suchen, ...).[Bec02][Mar08]

5.1.2 Clean Code

Neben dem beschriebenen TDD soll des weiteren auf **Clean Code** geachtet werden. Clean Code bzw. Sauberer-Code soll verhindern, dass die Wartbarkeit des Codes im laufe der Entwicklung im Vordergrund steht. Das Vorgehen soll dabei verhindern, dass die Produktivität der Entwicklung aufgrund von unschöner Programmierung abnimmt. Das sorgt zwar dafür, dass häufig Features schnell umgesetzt werden, diese aber nur sehr schwer bzw. kaum weiterentwickelt werden können. Robert C. Martin beschreibt in seinem Buch *Clean Code* [Mar08] diverse "Design-Flaws" (Unschönheiten im Code) und wie diese verhindert werden können. Dabei geht er auch darauf ein, wie grundsätz-

lich Code sauber und nachvollziehbar strukturiert werden sollte. Einige der Hauptprobleme die Robert C. Martin in seinem Werk identifiziert, sind folgende:

Probleme	Beschreibung
Unsaubere Interfaces	Es muss auf den ersten Blick ersichtlich sein, was eine Methode eines Interfaces ausmacht. Welche Funktionalität soll die Methode bieten und welches die Übergabeparameter sind. Die Übergabeparameter sollten dabei ausschließlich zur Erfüllung der Funktionalität beitragen.
Vermischung von Zuständigkeiten	Ein Objekt benötigt eine klare Art der Zuständigkeit. So sollte z.B. eine Bean, welche für die Darstellung eines Objekts zuständig ist, nicht noch zusätzlich deren Persistierung verwalten. Hier sollten die Objekte klar getrennt und strukturiert werden, damit deren Zuständigkeit auf den ersten Blick ersichtlich wird.
Nebeneffekte von Funktionen	In der Praxis findet man häufig Methoden, die viele verschiedene Funktionen erfüllen. Ein beliebtes Beispiel hierfür ist z.B. eine Methode `void createMessages(List messageList)`. Ein Außenstehender würde sich sofort wundern, weshalb die Methode einen **void**-Rückgabeparameter hat. In Wirklichkeit werden innerhalb der Methode Nachrichten erzeugt und der übergebenen Liste hinzugefügt. Dieses wird auch als Nebeneffekt einer Methode beschrieben, der von außen nicht ersichtlich ist. Die einzige Aufgabe der Methode sollte es sein, Nachrichten zu erzeugen. Sauberer wäre es, wenn die Methode wie folgt aussehen würde: `List createMessages()`. Diese Methode macht sofort deutlich, dass hier Nachrichten erzeugt und diese zurückgegeben werden. Außerhalb der Methode kann nun der Entwickler entscheiden, was er mit den erzeugten Nachrichten macht.

DRY-Probleme (Don't repeat yourself)	In der Entwicklung von Business-Anwendungen ist es außerordentlich wichtig, dass die Business-Logik lediglich an einer Stelle implementiert wird. Oftmals kommt es vor, dass gleiche Logik an verschiedenen Stellen verwendet werden muss (z.B. die Berechnung der Summe des aktuellen Warenkorbs eines Benutzers). Das duplizieren von Code ist sehr gefährlich, da unter Umständen Fehler nur in einer der duplizierten Stellen behoben werden. Die anderen duplizierten Code-Stellen bleiben dann weiterhin fehlerhaft.

Tabelle 5.1: Clean-Code Probleme

Im Verlauf der Umsetzung müssen diese Probleme berücksichtigt und verhindert werden, sodass die Weiterentwicklung und Wartung des Web-Shops unter geringst möglichem Aufwand und Kosten erfolgen kann.

5.2 Umsetzung des Entwurfs

Im folgenden Unterkapitel wird die konkrete Umsetzung der im Kapitel 4.3 konzipierten Komponenten beschrieben.

Grundsätzlich erfolgt die Implementierung auf Basis des in Kapitel 4.5 beschriebenen Model-View-Controller Prinzips. Dieses lässt sich auf die verschiedenen Schichten der Applikation nahezu 1 zu 1 abbilden. Die Schichten der Applikation teilen sich auf vier verschiedene Technologien auf. *JSF* und *CDI* als View bzw. Web-Schnittstelle, *EJB* als Controller und *JPA* als Modell. Die Entwicklung der verschiedenen Komponenten muss dabei diese Prinzipien berücksichtigen, wobei es z.B. keinerlei Business-Logik im Bereich View bzw. JSF geben darf. Die JSF Templates beeinhalten reine User Interface Logik, welche über CDI mit dem Server kommuniziert, um hier beispielsweise User-Eingaben zu halten. Nachdem eine Action auslöst und eine Art von Business-Logik notwendig ist, muss die CDI-Bean an einen EJB-Service delegieren, die die notwendige Logik implementiert. Abbildung 5.1 zeigt an einigen Beispielen die Aufteilung in diese Schichten.

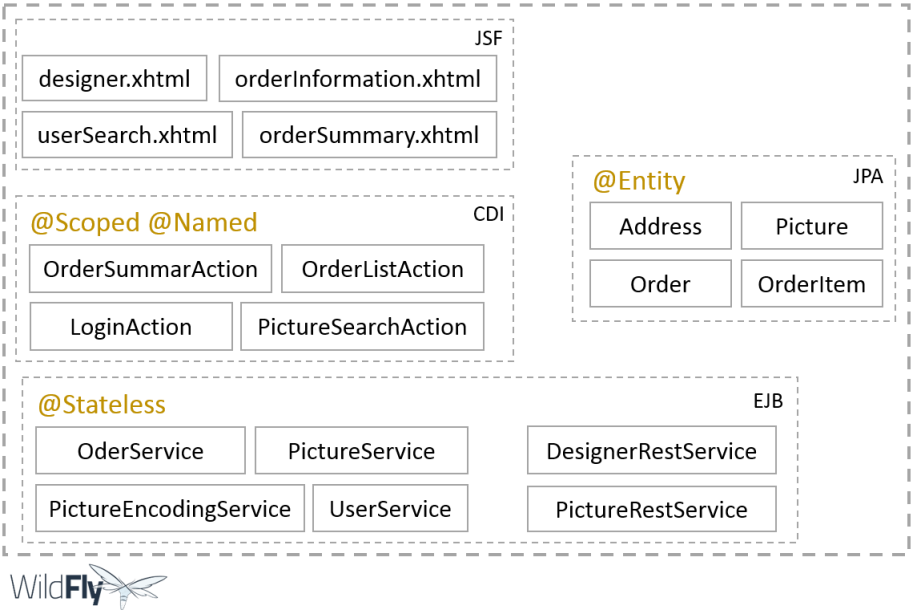

Abbildung 5.1: Konkrete Gesamtarchitektur

5.2.1 Designer

Der Designer beschreibt die Komponente zum manuellen Erstellen von Steckperlenbildern, welche innerhalb des Kapitels 4.3.1 konzipiert wurde. Der Designer besteht aus fünf verschiedenen Komponenten (siehe auch Abbildung 5.2).

- designer.xhtml
- designer.js
- DesignerAction.java
- DesignerRestService.java
- PictureEncodingService.java

Die *designer.xhtml* beschreibt den HTML-Aufbau des Designers mithilfe der JSF-Markup-Language. Hier werden die UI-Komponenten sowie deren Anordnung in Form von XHTML und CSS beschrieben. Darüber hinaus bindet die *designer.xhtml* die JavaScript Bibliothek *designer.js* ein. Diese beinhaltet die gesamte JavaScript Logik zum Setzen von Steckperlen, Auswahl der Farbe und das Laden von bereits gesetzten Perlen. Neben der erweiterten UI-Logik, die in der JavaScript Bibliothek beschrieben ist, werden hier auch Aufrufe an den *DeignerRestService* gekapselt. *DesignerRestService* bietet zwei Methoden nach außen an, `void putPearl(PlacedPearlInformation`

`placedPearlInformation)` und `String[][] loadAlreadyPlacedPearls()`. Die Methode `putPearl` wird immer dann aufgerufen, wenn der Benutzer eine Steckperle gesetzt hat. Das übergebene *PlacedPearlInformation*-Objekt beinhaltet dabei die Informationen über den X- und Y-Index sowie die Farbe der gerade gesetzten Perle. Der *DesignerRestService* übermittelt daraufhin die Werte an die *DesignerAction*, welche den Zustand des Bildes hält. Verlässt nun der Benutzer den Designer, ohne das Bild abzuschließen, bleibt der Zustand in der *SessionScoped DesignerAction*.

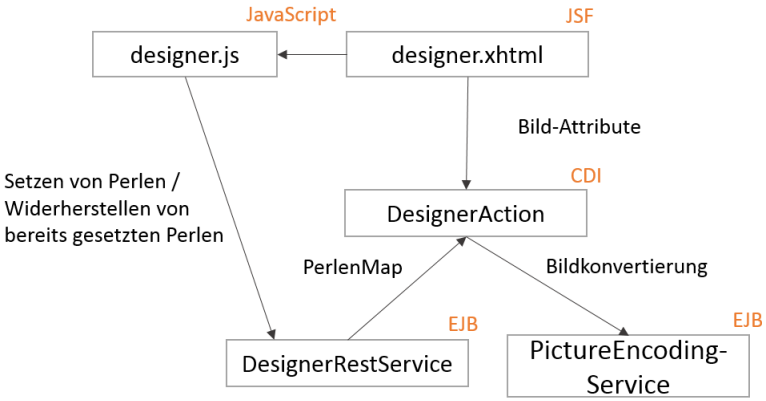

Abbildung 5.2: Umsetzung des Designers

Betritt er später erneut den Designer, so wird sein vorheriges Bild mithilfe der loadAlready-PlacedPearls-Methode wiederhergestellt. Der *DersignerRestService* liest den aktuellen Zustand aus der *DesignerAction* und übermittelt diesen an den JavaScript-Code. Dieser analysiert daraufhin das zurückgelieferte String Array und setzt die Perlen in der UI.

Der Lesende Zugriff von der *designer.xhtml* auf die *DesignerAction* begründet sich auf Attribute des Bildes. So werden Bild-Attribute wie z.B. der Name des Bildes über direktes JSF-Property-Binding an eine TextInput Komponente in der UI gebunden.

Wenn ein Benutzer ein Bild fertiggestellt hat, wird es über die *DesignerAction* in der Datenbank persistiert. Dazu wird zunächst die aktuelle PerlenMap, wie in Kapitel 4.2.2 von dem *PictureEncodingService* kodiert, um möglichst wenig Speicherplatz in der Datenbank nutzen zu müssen.

Der Aufbau der UI (siehe Abbildung 5.3) gestaltet sich nach folgendem Muster. Der Benutzer hat auf der linken Seite ein Panel, in welchem er eine Farbe einer Steckperle auswählen kann. Initial ist die erste Farbe als ausgewählt markiert. Unter den Farben befindet sich darüber hinaus eine Farbe zum löschen von bereits gesetzten Perlen (Transparent). Diese entfernt bei einem Mausklick die jeweilige Perle an dem gewählten Index. Mittig auf der Seite befindet

sich ein Canvas-Objekt, welches als "Leinwand" für das Steckperlenbild dient. Dieses ist durch ein besonderes Hintergrundbild als solches markiert. Oberhalb des Bildes befinden sich diverse Aktions-Buttons wie z.B. "Bild zurücksetzen" und "Bild hochladen". Der Aktion-Button Bild zurücksetzen entfernt alle bereits gesetzten Perlen nach einer Sicherheitsabfrage ("Hiermit werden alle bereits gesetzten Perlen entfernt."). Der Button "Bild hochladen" wird genauer in dem Kapitel 5.2.2 beschrieben. Er leitet das Hochladen eines Bildes ein. Unterhalb des Bildes befindet sich eine CheckBox mit der man das Steckperlenbild für andere Benutzer freigeben kann. Bei einem Klick auf die Checkbox werden drei weitere Eingabefelder aufgedeckt. Hier kann der Benutzer den Namen des Bildes, die Kategorie sowie beliebig viele Stichwörter zu dem Bild festlegen. An der unteren, rechten Seite des Designer befindet sich ein Button zum abschicken des Bildes. Hiermit kann der Benutzer das Bild in seinem Warenkorb platzieren.

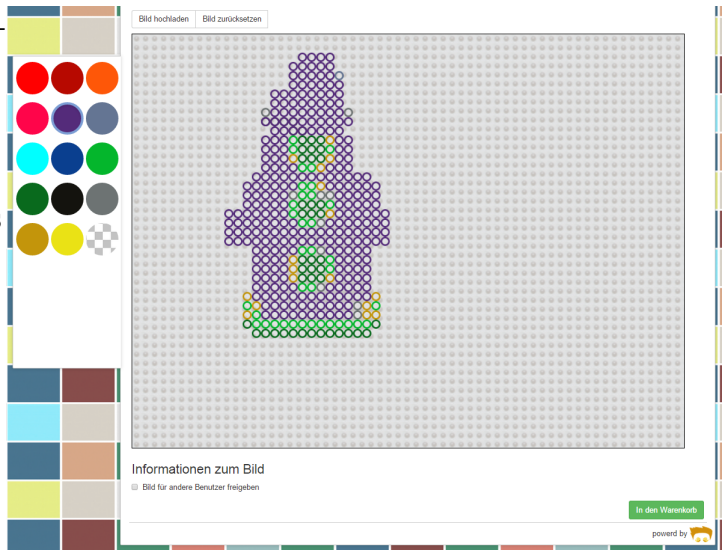

Abbildung 5.3: Screenshot des Designers

5.2.2 Perlenkonverter

Der Perlenkonverter beschreibt die Komponente, die ein herkömmliches Bild in ein Steckperlen-Bild umrechnet. Die Komponente wurde innerhalb des Kapitels 4.3.2 konzipiert.

Der Upload befindet sich innerhalb der *DesignerAction*. Die *DesignerAction* enthält eine Property, welche an eine JSF `h:inputFile` Komponete aus der *designer.xhtml* gebunden ist. Lädt ein Benutzer nun ein Bild hoch (siehe auch Abbildung 5.5), so wird dieses in ein *Part*-Objekt in der *DesignerAction* gesetzt. Nachdem der Upload erfolgreich abgeschlossen wurde, delegiert die *DesignerAction* die Umrechnung des tatsächlichen Bildes an die Konvertierungslogik

innerhalb der *ImageUtils*. Diese Utility-Klasse bietet nach außen hin eine Methode `PearlColor[][] pixelateImage(...)` an. Innerhalb dieser Methode wird zunächst das Bild anhand der maximal möglichen Breite (possibleWidth) und Höhe (possibleHeight) prozentual heruntergskaliert (`resizeImage()`), sollte es zu groß sein. Daraufhin unterteilt ein Algorithmus das Bild in Zellen, die die Größe einer Steckperle haben. Innerhalb der sich ergebenden Zellen wird nachfolgend die vorherrschende Farbe ermittelt (`PearlColor findMostMatchingPearlColor(Color)`) und diese in einer temporären PerlenMap gehalten. Sobald alle Zellen des Bildes analysiert und auf vorhanden Perlenfarben gemapped wurden, wird die PerlenMap an die *DesignerAction* zurückgegeben. Die PerlenMap wird danach in dem Canvas-Objekt unter Zuhilfenahme der *designer.js* neu gezeichnet.

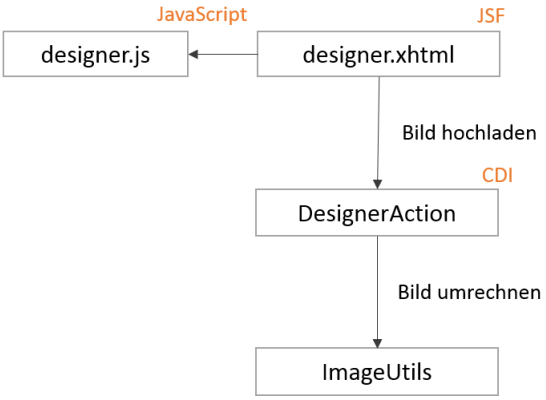

Abbildung 5.4: Umsetzung des Perlenkonverters

5.2.3 Schwachstellenanalyse

Im Kapitel 4.3.3 wurde bereits der fachliche Hintergrund der Schwachstellenanalyse betrachtet. So soll diese nach einer Platzierung einer Perle, mögliche, neu entstandene Schwachstellen am Bild aufdecken und dem Benutzer präsent gemacht werden. Die Schwachstellenanalyse befindet sich innerhalb der Klasse *PictureDiagnoseService*. Die Methode `findPossiblePictureWeakSpots()` erwartet das aktuelle, serverseitige Modell des Steckperlenbildes (`PearlColor[][] pearlMap`) und gibt eine Liste von *PearlIndex* zurück. Diese beschreiben X-

Abbildung 5.5: Screenshot des Perlenkonverters

und Y-Koordinaten von Perlen, die als Schwachstellen identifiziert werden konnten.
Aufgerufen wird diese Analyse über den `DesignerRestService`. Dieser ist direkt in der UI eingebunden und wird über einen Validierungsbutton direkt genutzt. Zur Anzeige der Schwachstellen wurde ein weiteres Canvas, über den bereits bestehenden Canvas gelegt. Dieser dient einzig und allein der Anzeige von Validierungsfehlern bzw. Schwachstellen.

Ein eigenes Canvas für Validierungsfehler hat den Vorteil, dass diese sehr leicht entfernt werden können (man löscht einfach alle Inhalte des Canvas). Würden die Schwachstellen direkt innerhalb des normalen Canvas-Objektes gezeichnet werden, müsste man diese explizit identifizieren und sehr fein ausschneiden, um die herumliegende Perle nicht zu beeinflussen. Liefer der Server also eine bzw. mehrere Schwachstellen, werden diese in dem überliegenden Canvas gezeichnet. Sobald der Benutzer daraufhin eine Perle setzt, werden die Validierungsfehler wieder entfernt, sodass diese nicht den Eindruck des Bildes verfälscht. Bei erneuter Betätigung des Validierungsbuttons, werden erneut die Schwachstellen analysiert und dem Benutzer erneut gezeichnet.

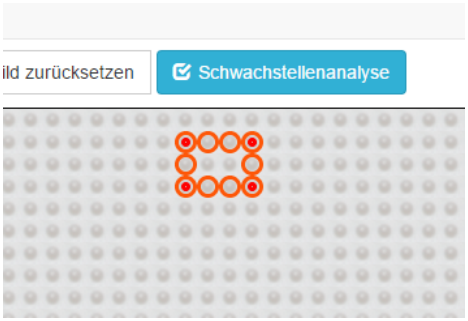

Abbildung 5.6: Screenshot der Schwachstellenanalyse

Neben der direkt Analyse des Bildes über den Validierungsbutton, wird ebenfalls die Schwachstellenanalyse beim Betätigen des "Weiter"-Buttons ausgelöst. Enthält das Bild Schwachstellen, kann der Benutzer es nicht bestellen oder speichern. Der Benutzer wird durch eine Fehlermeldung darüber informiert, dass das Bild noch Schwachstellen enthält. Drüber hinaus werden die Schwachstellen, wie über den Validierungsbutton, farblich im Bild hervorgehoben, damit der Benutzer diese auch erkennt. Erst nach erfolgreicher Schwachstellenanalyse, ist der Benutzer in der Lage, das Bild zu bestellen.

5.2.4 Galerien

Die Bildergalerien bilden einen zentralen Baustein des Shops. Aktuell sind zwei unterschiedliche Suchen implementiert. Die "Beliebten Bildern"- und "Neueste

Bilder"-Galerie. Die beiden Galerien unterscheiden sich nur in der zugrundeliegenden Query. Innerhalb der UI basieren die beiden Galerien auf den selben XHTML-Templates, welche mit unterschiedlichen Implementationen des *PictureDataProvider*-Interfaces initialisiert werden. Konkret wird das Interface von dem *PopularPictureProvider* (Beliebte Bilder) und dem *NewestPictureDataProvider* (Neueste Bilder) implementiert. Diese definieren die konkreten Queries um die Bilder in ihrer entsprechenden Reihenfolge zu laden. Das Interfaces *PictureDataProvider* definiert 2 unterschiedliche Methoden die aus dem Kontext des Web-Templates für verschiedene Funktionen angesprochen werden. Die Methode `findInitialPictures(int pictureCount)` der Klasse *PictureDataProvider* liefert die Grundmenge an Bildern, welche innerhalb der Galerie dargestellt werden. Der übergebene *pictureCount* begrenzt die Ergebnismenge auf n-Bilder. Über die weitere Methode `getResults()` können diese zunächst geladenen Bilder dann abgerufen werden. Unterhalb der Galerien befindet sich jeweils ein Button "Weitere" mit dessen Hilfe man in die Bildersuche gelangt. Hier ist nun die jeweilige Sortierung ausgewählt ("Nach Aufrufen" oder "Nach Erstellungsdatum").

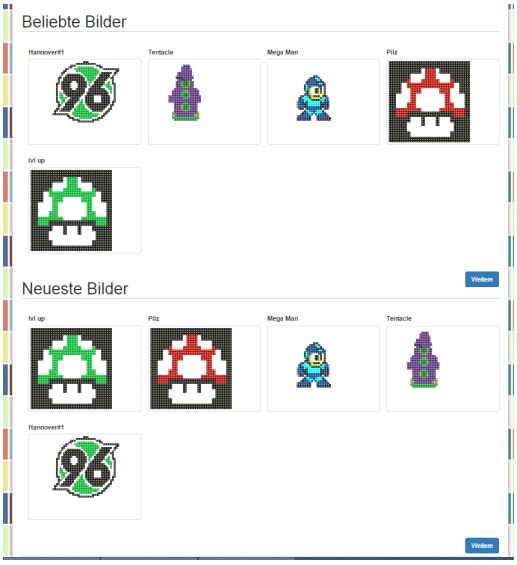

Abbildung 5.7: Screenshot der Galerien

5.2.5 Bildersuche

Die Bildersuche basiert ebenfalls auf dem im Kapitel 5.2.4 beschriebenen Interface *PictureDataProvider*. Sie enthält bietet neben den beiden Methoden *findInitialPictures* und *getResults* eine weitere Methode `findNextPictures(int pictureCount)` an. Mit dessen Hilfe lassen sich weitere Blöcke von Bildern laden.

Die Bildersuche bietet, wie der Name vermuten lässt, diverse Suchfilter, mit dessen Hilfe sich die Ergebnismenge an Bildern reduzieren lässt. Sie definiert 3 Suchfelder und eine Sortierfunktion. Der Anwender kann die Bilder nach ihrem Namen, nach einer Bildkategorie oder nach Stichwörtern filtern. Wählt bzw. füllt der Anwender eines oder mehrere der Suchkriterien aus, so wird auf der Serverseite dynamisch eine Query generiert. Diese Funktionalität ist innerhalb der Klasse *QueryUtils* gekapselt. Sie implementiert dabei zwei Methode, `createQueryString()` und `applyQueryParameter()`. Die Methode `createQueryString` erwartet eine Liste von *queryRestrictions*, welche innerhalb der *PictureSearchAction* auf Grundlage der Benutzereingaben generiert werden. *applyQueryParameter* füllt daraufhin die Query mit den jeweiligen Benutzereingaben. Die vom Anwender gewählte Sortierungsfunktion legt innerhalb der Query das "sort by" fest.

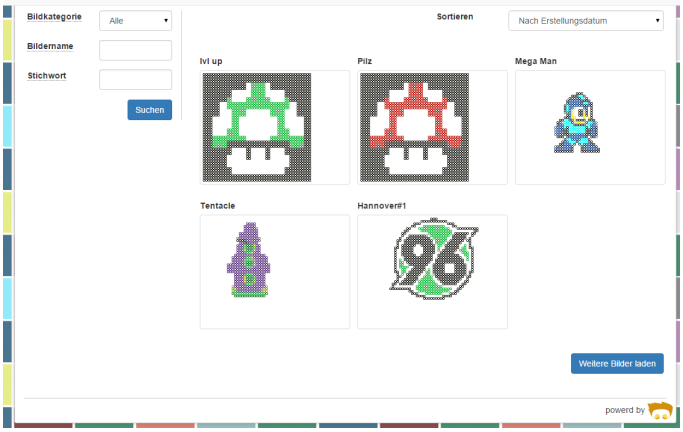

Abbildung 5.8: Screenshot der Bildersuche

5.2.6 Bestellübersicht

Die Bestellübersicht listet alle Bestellungen (Orders) aus der Datenbank auf. Diese können in dem rechten Bereich (siehe auch Abbildung 5.9) aufgeklappt werden. Klappt der Mitarbeiter eine Bestellung auf, werden im Detailinformationen zu dieser präsentiert. Dazu gehören neben der Lieferanschrift, die bestellten Bilder. Zu jeder Bestellung hat der Mitarbeiter die Möglichkeit, diese in

einen nachfolgenden Status zu versetzen. Die Status sind dabei wie im Kapitel 4.3.4 beschrieben, umgesetzt.

In der Bearbeitungsphase, kann der Mitarbeiter jedes einzelne Bild ausdrucken. Bei einem Klick auf den "Druck"-Button unterhalb eines Bildes, wird ein Eintrag in die Datenbank geschrieben, dass dieses Bild nun gedruckt werden soll. Auf der linken Seite stehen Filterfunktionen über die jeweiligen Bestellungstatus zur Verfügung. Oftmals interessiert sich ein Mitarbeiter nur für Bestellungen in einem bestimmten Status, weshalb er hier, alle nicht relevanten Bestellungen ausblenden kann.

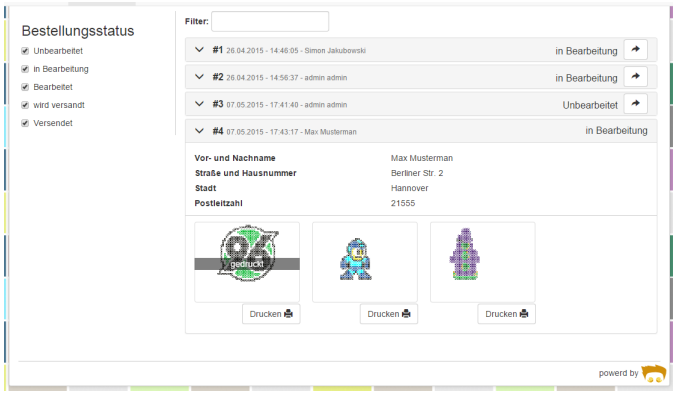

Abbildung 5.9: Screenshot der Bestellübersicht

5.2.7 Druck

Wie innerhalb des Kapitels 4.4 beschrieben, erfolgt die Anbindung des Steckperlendruckers in Form von *SCP*- und *SSH*-Befehlen. Dieses setzt natürlich voraus, dass der Web-Server des Web-Shops über ein Netzwerk den Steckperlendrucker erreichen kann. Zur Kommunikation mit dem Steckperlendrucker wird innerhalb des Web-Shops das Java-Framework *Jsch* von der Firma *JCraft* verwendet. Dieses erleichtert die Kommunikation über *SSH* und *SCP*, da es die Kommunikation kapselt und komfortablere Schnittstellen bietet. Die Ansteue-

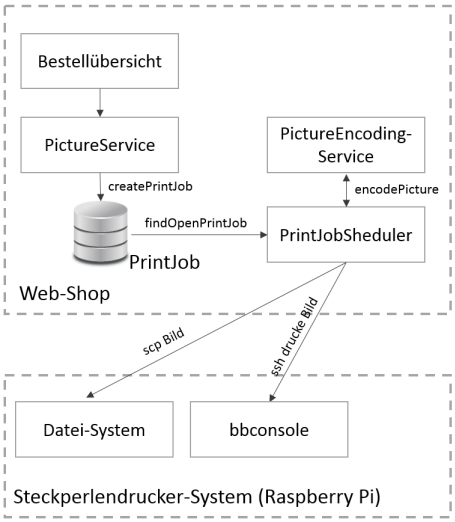

Abbildung 5.10: Anbindung des Steckperlendruckers

rung der Steckperlendrucker-Schnittstelle wird innerhalb des *PrintJobScheduler.java* implementiert (siehe dazu auch Abbildung 5.10). Der Scheduler wird einzeln als Thread bei dem Serverstart instantiiert. Dieser läuft Benutzerunabhängig und existiert lediglich einmal. Die Kommunikation in Richtung des Druckers wird also zentral von einer Instanz des Schedulers gesteuert. Anderenfalls wäre es wahrscheinlich, dass die Schnittstelle des Druckers durch verschiedene Threads überlaufen wird. Dies hätte zur Folge, dass ein Druck nicht gewährleistet werden könnte, da möglicherweise Aufrufe verloren gehen, die zu schnell hintereinander erfolgen.

Der Scheduler führt in regelmäßigen Abständen (aktuell jede halbe Minute) eine Methode *processOpenPrintJobs* aus.

5.2.7.1 Ablauf des PrintJobScheduler

Der Ablauf des *PrintJobScheduler.java* wird innerhalb dieses Unterkapitels näher erläutert.

5.2.7.1.1 Laden eines offenen PrintJobs Diese fragt zunächst den ältesten, offenen *PrintJob* aus der Datenbank ab. Die PrintJobs werden von Mitarbeitern aus der 5.2.6 angelegt und asynchron durch den *PrintJobScheduler.java* abgearbeitet. Findet der *PrintJobScheduler.java* einen offenen PrintJob (PrintJob.printJobState = "WAITING_FOR_PRINT"), so lädt er diesen aus der Datenbank.

5.2.7.1.2 Encodieren des kodierten Bildes Da wir, wie bereits in Kapitel 4.2.2 beschrieben, das Bild lediglich als encodierten String in der Datenbank halten, muss hieraus zunächst wieder ein tatsächliches Bild erzeugt werden. Zu diesem Zweck wird das kodierte Bild unter Zuhilfenahme des *PictureEncodingService.java* in ein zweidimensionales Array von Farben umgewandelt.

5.2.7.1.3 Umwandeln des zweidimensionalen Farben-Arrays in ein tatsächliches Bild Das Array wird dann durch die Methode *drawPixelatedPicture* der *ImageUtils.java* wieder in ein Bild umgewandelt. Die Methode *drawPixelatedPicture* nimmt das zweidimensionale Array aus Farben entgegen

und liefert ein *BufferedImage* zurück. Das Bild ist eine verpixelte Version des eigentlichen Steckperlenbildes.

5.2.7.1.4 SCP des Bildes auf den Steckperlendrucker Der *PrintJobScheduler.java* baut nun eine Session zwischen dem Web-Shop und dem Steckperlendrucker auf. Darauffolgend wird das Bild per SCP-Befehl an den Steckperlendrucker übertragen. Der Name des Bildes setzt sich dabei aus der Datenbank-ID des Bildes und der aktuellen Serverzeit zusammen.

5.2.7.1.5 SSH-Anweisung an den Drucker zu eigentlich Druck des Bildes Wurde das Bild korrekt an den Steckperlendrucker übertragen, so erfolgt ein *SSH*-Aufruf des Schedulers an den Steckperlendrucker. Ein beispielhafter Aufruf könnte sein:

```
./bbconsole --size large --plotter color --show-statistics
    /usr/webshopimages/<Datenbank-Id>-<serverzeit>.png
```

Der Aufruf wird bewusst nicht geforked, damit der Aufrufende Thread des *PrintJobScheduler.java* blockiert. Würde der Aufruf asynchron ausgeführt werden, müsste ein weitere Scheduler den die tatsächlich gedruckten Bilder wieder den Datenbank-PrintJobs zuordnen und diese innerhalb der Datenbank aktualisieren. Ein Blockieren des Threads ist aufgrund der Ausführung eines Scheduler auf einem Applikation-Server kein Risiko, weshalb der Aufruf bewusst synchron stattfindet. Hat der Drucker den Druck abgeschlossen, so liefert dieser eine Statistik des Drucks (aufgrund des `--show-statistics` Parameters).

5.2.7.1.6 Aktualisierung des PrintJob-Status in der Datenbank Die Ergebnisse bzw. die Statistik des Druckers wird nach Beendigung des Drucks ausgewertet. Zeigt sich kein Fehler innerhalb der Statistik, so wird der PrintJob-Statuts in der Datenbank auf "PRINTED" aktualisiert. Gab es jedoch einen Fehler während des Drucks, so wird der PrintJob-Statuts auf "FAILED" aktualisiert. Ein Mitarbeiter ist daraufhin angewiesen, den Druck zu kontrollieren und ggf. den Druck erneut über die Bestellübersicht zu starten.

5.2.7.2 Übersicht der PrintJobs

Aus administrativer Sicht ist es sinnvoll und erforderlich, dass die Mitarbeiter eine Übersicht der aktuellen PrintJobs erhalten. Diese zeigt eine rudimentäre Auflistung der aktuellen PrintJobs (wie in Abbildung 5.11 zu sehen ist). Angezeigt werden Informationen über den Mitarbeiter, der den Druckauftrag erzeugt hat, das Erstellungsdatum des Drucks sowie der Status des Drucks. Die Liste bietet den Mitarbeitern die Möglichkeit, die Status der Jobs zu überwachen und ggf. auf fehlerhafte PrintJobs zu reagieren.

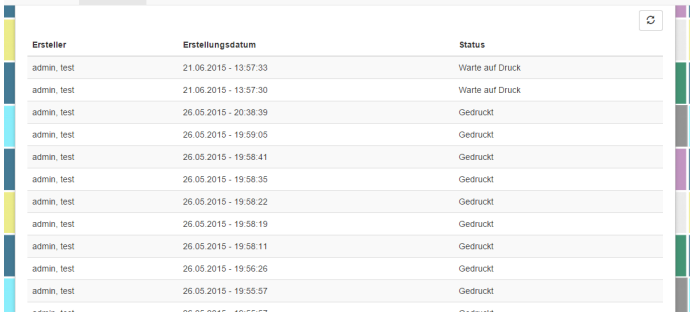

Abbildung 5.11: Screenshot der Druckauftragsübersicht

5.2.8 Benutzersuche bzw. -ansicht

Die Benutzersuche soll den Mitarbeitern eine administrative Oberfläche zur Benutzerverwaltung bieten, wie bereits in Kapitel 4.3.8 beschrieben. Zu diesem Zweck sollen zunächst einfache Suchfilter vorhanden sein (siehe Abbildung 5.12), die die Ergebnismenge an Benutzern einschränken soll (Vorname, Nachname und E-Mail). Diese Suchfilter sind als Wildcard-Suchfilter zu betrachten. Sucht der Mitarbeiter also nach z.B. "Vorname: Seb" dann erhält er ebenfalls Ergebnisse mit dem Vornamen "Sebastian" oder "Hansebille". Markiert der Mitarbeiter einen Benutzer aus der Ergebnismenge, so aktivieren die Buttons sich oberhalb der Ergebnistabelle. Diese bieten dem Mitarbeiter folgende Funktionen:

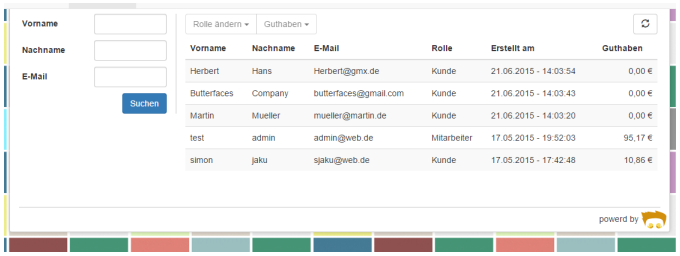

Abbildung 5.12: Screenshot der Benutzersuche

- Ändern der Benutzerrolle (Kunde, Mitarbeiter),
- Hinzufügen bzw. Abbuche von Guthaben,
- Anzeigen der Guthaben-Historie für den ausgewählten Benutzer.

Abbildung 5.13: Screenshot der Guthaben hinzufügen Funktion

Initial soll ein Administrations-Benutzer eingerichtet werden, der die Rolle "Mitarbeiter" besitzt. Dieser kann dann, andere Benutzer als "Mitarbeiter" kennzeichnen, sodass diese wiederum andere Benutzer als "Mitarbeiter" kennzeichnen können. Dieses erleichtert deutlich die Mitarbeiterverwaltung, da sie bequem aus der Plattform bedient und nicht über Datenbank-Updates getätigt werden muss. Neben den rein administrativen Aufgaben wie die Rollenverwaltung, kann aus der Benutzersuche heraus auch das Guthaben der Benutzer verwaltet werden. Markiert ein Mitarbeiter einen Benutzer, so kann er diesem Guthaben zuschreiben und abbuchen (siehe Abbildung 5.13). Trägt der Mitarbeiter einen negativen Wert in das Popup "Guthaben gutschreiben" ein, so wird diesem vom aktuellen Guthaben des selektierten Benutzers abgezogen. In jedem Fall wird ein Vermerk in der Datenbank hinterlegt, dass der aktuell eingeloggte Mitarbeiter eine Änderung an dem Guthaben des Benutzers vollzogen hat. Dieser Vermerk kann in der Guthaben-Historie nachvollzogen werden (siehe Abbildung 5.14).

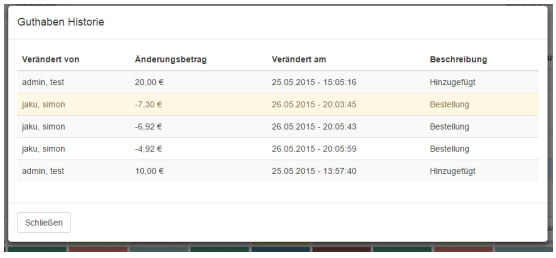

Abbildung 5.14: Screenshot der Guthaben-Historie

Diese enthält eine Auflistung aller Änderungen, an dem Guthaben eines Benutzers. Diese Änderungen können Hinzubuchungen von Mitarbeitern oder aber Abzüge aufgrund von Bestellungen sein. Bestellt ein Kunde ein Bild mit dem Zahlunsmittel "Guthaben", so wird dieses von seinem Guthaben abgebucht und ebenfalls in der Guthaben-Historie vermerkt.

5.2.9 Bildfreigabe

Die Funktion der Bildfreigabe ermöglicht es, wie bereits in Kapitel 4.3.9 erläutert, den Mitarbeitern nicht angebrachte Bilder abzulehnen. Dies verhindert, dass solche Bilder innerhalb des Web-Shops angezeigt werden.

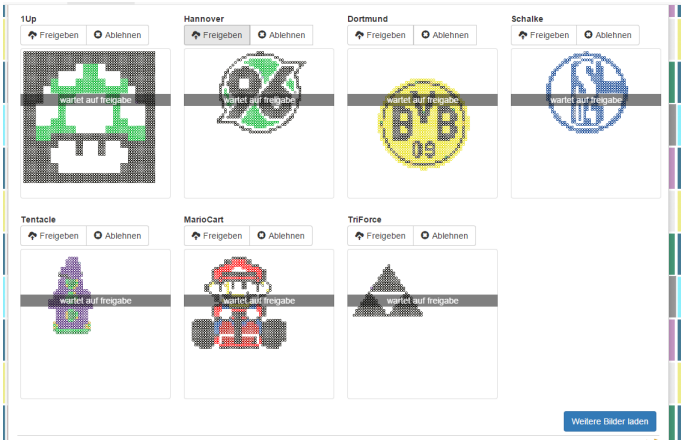

Abbildung 5.15: Screenshot der Bildfreigabe

Die Bildfreigabe listet dem Mitarbeiter alle Bilder auf (siehe Abbildung 5.15), deren Freigabe noch aussteht. Der Mitarbeiter sieht pro Bild den Namen, eine Vorschau des Bildes sowie die zwei Aktionsbuttons "Freigeben" und "Ablehnen". Mit einem klick auf das Vorschau-Bild wird dieses in einem Popup, vergrößert angezeigt. Wie man es bereits aus den Suchen und Galerien kennt.

Hier erhält er zusätzliche Informationen über das Bild, wie z.B. Stichwörter, Bildkategorie, und weitere. Betätigt der Mitarbeiter den "Freigeben"-Button, so wird das Bild in der Datenbank als "approved" markiert und kann fortan in allen Suchen und Galerien von anderen Benutzern gefunden und angeschaut werden (siehe). Betätigt der Mitarbeiter den "Ablehnen"-Button, so wird das Bild mit seinen Stichwörtern aus der Datenbank entfernt. Alternativ hätte man das Bild mit einer Markierung "gelöscht" versehen können, worauf allerdings bewusst verzichtet wurde. Hier könnten sich über den Verlauf der Zeit sehr viele abgelehnte Bilder anhäufen, die unnötigerweise Speicherplatz verbrauchen würden und tatsächlich keinen Mehrwert für den Web-Shop erzeugen. Die Bildfreigabe besitzt, genau wie die Bildersuche (siehe Kapitel 5.2.5) eine Funktion um weitere Bilder zu laden. Initial ist die Ergebnismenge auf 24 Bilder beschränkt (6 Zeilen mit 4 Bildern pro Zeile). Lädt der Mitarbeiter weitere Bilder, so werden diese unterhalb der ersten 24 Bilder angehängt.

5.3 Zusammenfassung

Das vorangegangene Kapitel beschreibt zunächst das Entwicklungsvorgehen bei der Umsetzung des Entwurfs aus dem Kapitel 4.3. Dabei wurden die Vorgehen Test-driven-development sowie Clean Code vorgestellt, die der eigentlichen Entwicklung der Komponenten zu Grunde liegen. Dabei wird ein großer Wert auf die Lesbarkeit, Wartbarkeit sowie Wiederverwendtbarkeit von Code gelegt. Dieses besitzt einen hohen Stellenwert, da wie z.B. innerhalb des Artikels [Aga05] von Rahul Agarwal beschrieben, Wartung oftmals einen sehr hohen Kostenfaktor besitzt. In manchen Fällen übersteigen sogar die Wartungskosten, die Kosten der eigentlichen Entwicklung. Aus diesem Grund ist es von außerordentlicher Wichtigkeit, dass eine Applikation gut Wartbar und auch für andere Entwickler nachvollziehbar strukturiert und entwickelt wurde. Dies soll die Wartungskosten in einem angemessenen Rahmen halten. Darauffolgend wurde dann die Implementierung der bereits konzipierten Komponenten beschrieben und an geeigneten Stellen mit Screenshots dokumentiert.

Kapitel 6

Evaluation

Inhalt dieses Kapitels ist es, typische Use-Cases an dem zuvor entworfenen System vorzustellen. Die Anwendungsfälle werden innerhalb dieses Kapitels aus den bereits im vorherigen Verlauf verdeutlichten Rollen beschrieben. Dazu gehört die Rolle Anwender (nicht registriert), Kunde (registrierter Anwender) und Mitarbeiter (registrierter Anwender mit der Rolle Mitarbeiter).

6.1 Use-Cases

Innerhalb dieses Kapitels werden die Anwendungsfälle im Detail beschrieben. Eine Übersicht der betrachteten Anwendungsfälle können Sie der Tabelle 6.1 finden.

Suche eines bestimmten Bildes

Der Anwender ruft den Perlenshop unter einer registrierten URL auf und befindet sich zunächst auf der Startseite. Er navigiert über die Navigationsleiste in den Bereich "Bilder Suchen". Hier trägt er z.B. einen belieben Bildernamen oder ein Stichwort ein um so die Ergebnismenge an Bildern einzugrenzen. Sollte das Bild nicht unter den ersten 21 Bildern sein, kann er den Button "Weitere Bilder laden" am unteren Ende der Suchergebnisse drücken, welches weitere 21 Bilder nachlädt. Hat der Benutzer schlussendlich das gesuchte Bild gefunden, kann er es mit einem Mausklick vergrößern und ggf. in seinem Warenkorb

Rolle	Anwendungsfall
Anwender	Suche eines bestimmten Bildes
Anwender	Bestellung eines gerade erstellten Bildes mit der Zahlungsart "per Nachname"
Kunde	Bestellung eines bereits vorhanden Bildes mit Zahlungsart "Vorkasse"
Kunde	Bestellung eines bereits vorhanden Bildes mit Zahlungsart "Guthaben"
Kunde	Bearbeiten eines bereits vorhandenen Bildes mit anschließender Speicherung
Mitarbeiter	Freigabe / Ablehnung eines Bildes für die Öffentlichkeit
Mitarbeiter	Ablehnung eines Bildes
Mitarbeiter	Bearbeitung einer eingegangenen Bestellung (inkl. Druck)
Mitarbeiter	Änderung der Rolle eines bestimmten Anwenders
Mitarbeiter	Zuweisung von Guthaben zu einem Anwender

Tabelle 6.1: Anwendungsfälle

platzieren.

Bestellung eines gerade erstellten Bildes mit der Zahlungsart "per Nachname"

Der nicht angemeldete Benutzer befindet sich auf der Startseite. Er klickt in der Navigationsleiste auf "Designer" und landet daraufhin in dem Steckperlenkonfigurator. Hier zeichnet er nun sein gewünschtes Bild. Rechts, oberhalb der Zeichenfläche befindet sich ein "Preiszähler" der direktes Feedback über den aktuellen Preis des Bildes gibt. Platziert der Anwender eine neue Perle, so wird der Preis des Bildes neu berechnet und dort angezeigt. Nachdem der Benutzer mit dem Bild zufrieden ist, klickt er auf "In den Warenkorb". Es erfolgt im Hintergrund eine Prüfung auf mögliche Schwachstellen. Weist das Bild Schwachstellen auf, so werden diese in der Zeichenfläche durch rote Punkte innerhalb der gefährdeten Perlen hervorgehoben. Das Bild kann erst bestellt werden, wenn die Schwachstellenanalyse erfolgreich durchlaufen wurde. Der Anwender landet wieder auf der Startseite des Shops. Auf der rechten Seite der Navigationsleiste ist nun eine rote '1' an dem Warenkorbsymbol zu

erkennen. Mit einem klick auf den Warenkorb, landet der Anwender in der Warenkorbansicht. Hier befindet sich das soeben im Warenkorb platzierte Bild mit auszeichnen des entsprechenden Bildpreis sowohl dem Gesamtpreis des Warenkorbs, welche in unserem Fall identisch sind. Der Anwender bestätigt seinen Warenkorb mit einem Klick auf "Weiter". Da der Anwender bislang noch nicht angemeldet ist, muss er dieses nun zunächst tun. Er gibt seine E-Mail-Adresse sowie sein Passwort ein und befindet sich nachfolgend auf der Zahlungs- und Versandinformationsseite. Er wählt als Zahlungsart "Nachname" und füllt die Versandinformationen aus. Mit einem Klick auf "Weiter" wird dem Kunden eine Zusammenfassung seiner Bestellung angezeigt. Die Zusammenfassung enthält seinen Warenkorb, die gewählte Zahlungsart sowie die Versandinformationen. Er bestätigt diese Zusammenfassung mit einem Klick auf "Bestellen" woraufhin die Bestellung gespeichert und für die Mitarbeiter Sichtbar wird. Der Eingang der Bestellung wird dem Kunden per E-Mail quittiert.

Bestellung eines bereits vorhanden Bildes mit Zahlungsart "Vorkasse"

Der Kunde ist eingeloggt und befindet sich auf der Startseite des Shops. Er klickt auf eines der Bilder aus einer der Galerien, die sich auf der Startseite befindet. Es erscheint ein PopUp welches das Bild in einer vergrößerten Ansicht darstellt sowie Zusatzinformationen über das Bild enthält (z.B. Name des Bildes, Bildkategorie, Preis, ...). Der Benutzer klickt "In den Warenkorb". Der Kunde navigiert sich, durch den Warenkorb und wählt als Zahlungsart "Vorkasse". Er bestätigt seine Bestellung mit dem Klick auf "Bestellung". Der Kunde erhält daraufhin eine E-Mail mit den Informationen seiner Bestellung sowie eine weitere E-Mail mit den nötigen Bank-Kontoinformationen. Diese Mail enthält darüber hinaus eine eindeutige Rechnungsnummer, auf die er sich bei der Überweisung des Geldes beziehen muss.

Bestellung eines bereits vorhanden Bildes mit Zahlungsart "Guthaben"

Der Kunde bestellt ein bereits vorhandenes Bild und wählt auf der Seite Zahlungsinformationen als Zahlungsart "Guthaben" aus. Bei dem Klick auf "Wei-

ter" wird überprüft, ob der Kunde über genügend Guthaben an seinem Benutzer verfügt. Sollte dies nicht der Fall sein, wird ein Fehler auf der Seite angezeigt: "Sie haben nicht genügend Guthaben um die Bestellung abzuschließen. Es sind x.yy €notwendig". Der Kunde ist somit also gezwungen, eine andere Zahlungsart auszuwählen. Verfügt er über genügen Guthaben so kann er die Bestellung regulär abschicken. Bei dem Klick auf "Bestellen", werden die Kosten für die Bestellung von seinem Guthaben abgebucht. Darüber hinaus wird ein Eintrag in seiner Guthaben-Historie mit der Beschreibung "Bestellung" erzeugt. Somit kann lückenlos der Verlauf seines Guthabens von einem Mitarbeiter nachvollzogen werden. Wie auch bei den anderen Bestellung, erhält auch hier der Kunde mit der Bestätigung seiner Bestellung.

Bearbeiten eines bereits vorhandenen Bildes mit anschließender Speicherung

Der Kunde befindet sich auf der Startseite und klickt auf ein beliebiges Bild. Das Bild wird in einem PopUp, vergrößert dargestellt. Unterhalb des Bildes befindet sich neben dem Button "In den Warenkorb" der Button "Bearbeiten". Dieser ermöglicht es dem Benutzer, eine eigene Kopie des Bildes anzulegen und diese nach seinen Wünschen zu verändern. Nach einem Klickt auf den besagten Button, wird der Kunde in den Designer umgeleitet und Steckperlen wurden entsprechend der Vorlage des vorherigen Bildes gezeichnet. Der Benutzer kann nun das Bild weiterentwickeln und klickt daraufhin auf den Button "Bild speichern" welcher immer dann im Designer angezeigt wird, wenn der Benutzer angemeldet ist. Der Benutzer wird wieder auf die Startseite geleitet. Sein Bild befindet sich nun unter "Mein Konto" → "Meine Bilder".

Freigabe / Ablehnung eines Bildes für die Öffentlichkeit

Der Mitarbeiter befindet sich auf der Startseite und klickt im Navigationsbereich auf "Mitarbeiter" → "Bildfreigabe". Hier werden ihm alle Bilder aufgelistet, die gerade auf eine Freigabe warten. Oberhalb der einzelnen Bilder befinden sich zwei Buttons, "Freigeben" und "Ablehnen". Gibt der Mitarbeiter ein Bild frei, so wird es umgehend für alle Benutzer des Web-Shops sichtbar. D.h. das Bild kann in den Suchen gefunden und in der Galerie angezeigt

werden. Lehnt der Mitarbeiter ein Bild ab, so wird dieses aus dem System entfernt.

Bearbeitung einer eingegangenen Bestellung (inkl. Druck)

Der Mitarbeiter ist eingeloggt und befindet sich auf der Startseite. Unterhalb des Bereichs "Mitarbeiter" gibt es für die Mitarbeiter einen Menüpunkt "Bestellungen". Hier werden alle Bestellungen gelistet. Initial sind bereits versandte Bestellungen ausgeblendet, sodass der Mitarbeiter nur die Bestellungen sieht, die noch nicht abgeschlossen sind. Der Mitarbeiter klickt auf die Bestellung die er bearbeiten will. Es blenden sich daraufhin die Details der Bestellung auf. Diese enthalten die Liefer- und Rechnungsadresse sowie die Bilder, die bestellt wurden. Zunächst wechselt der Mitarbeiter den Status der Bestellung von "Unbearbeitet" in "In Bearbeitung". Nun hat er die Möglichkeit, die Bilder einzeln zu drucken. Der drückt dazu unterhalb der jeweiligen Bilder den Button "Drucken". Der Steckperlendrucker fängt nun an das Bild auszudrucken. Sobald dieser Vorgang abgeschlossen ist, wird der Status des Bildes auf "Gedruckt" gesetzt und ihm dementsprechend innerhalb der Bestellung angezeigt. Sollte bei dem Druck etwas schief gegangen sein, kann er jederzeit einen weiteren Druckvorgang für ein Bild auslösen. Sind alle Bilder der Bestellung gedruckt, kann der Mitarbeiter die Bestellung in den Status "Bearbeitet" versetzten. Daraufhin lässt sich der Status der Bestellung von "Bearbeitet" in "wird versandt" und letztendlich in "Versendet" versetzen. Ist eine Bestellung "Versendet" so gilt sie für den Web-Shop als abgeschlossen.

Änderung der Rolle eines bestimmten Anwenders

Jeder Mitarbeiter kann andere angemeldete Anwender als Mitarbeiter kennzeichnen. Der Mitarbeiter navigiert dazu in den Bereich "Mitarbeiter" → "Benutzersuche". Eine Auflistung aller registrierte Benutzer wird angezeigt mit Informationen über Vor- und Nachnamen sowie E-Mail Adresse, Rolle, Erstellt am und Guthaben. Wählt der Mitarbeiter einen registrierten Benutzer aus, aktivieren sich die Buttons "Rolle ändern" und "Guthaben" oberhalb der Suchergebnistabelle. Klickt der Benutzer auf "Rolle ändern" so kann er zwischen der Rolle "Kunde" und "Mitarbeiter" auswählen. Ändert der Mitarbeiter die Rolle eines Benutzers z.B. von "Kunde" auf "Mitarbeiter" so wird diese Änderung sofort gespeichert. Sollte währenddessen der besagte Benutzer noch

am Web-Shop angemeldet sein, so erfordert es zunächst eine Wiederanmeldung am Shop, damit die Rollen-Änderungen für ihn wirksam werden.

Zuweisung von Guthaben zu einem Anwender

Der Mitarbeiter wählt innerhalb der Benutzersuche einen Benutzer aus, welchem er Guthaben zuweisen will. Der Button "Guthaben" oberhalb der Tabelle wird aktiviert. Mit einem Klick auf "hinzufügen" öffnet sich ein PopUp. Hier gibt es ein Eingabefeld, in welches der Mitarbeiter das hinzuzufügende Guthaben einträgt (z.B. "4.25"). Das PopUp schließt sich und meldet ggf. Konvertierungsfehler, so ein Wert eingegeben wurde, der nicht als Dezimalzahl erkannt werden konnte. Ist der Wert valide, erscheint kein weitere Hinweis. Der Mitarbeiter kann die Eingabe des Guthaben in dem "Guthaben" → "anzeigen" PopUp validieren. In diesem erscheint nach erfolgreichem hinzubuchen von Guthaben ein neuer Eintrag. Dieser zeigt an, wer Guthaben, wann und in welcher Höhe hinzugefügt hat.

6.2 Zusammenfassung

Innerhalb dieses Kapitels wurden zunächst definierten Anwendungsfälle an den konkreten Rollen vorgestellt. Die Anwendungsfälle wurden dabei unter Berücksichtigung der in Kapitel 3.1 erfassten Anforderungen gewählt.

Kapitel 7

Ergebnisse und Ausblick

Dieses Kapitel fasst abschließend die Ergebnisse der vorliegenden Arbeit zusammen. Neben der Ergebnisbetrachtung erfolgt auch eine kritische Bewertung der Ergebnisse. Des Weiteren wird ein Ausblick auf nützliche Funktionalitäten aufgezeigt, die dem Web-Shop noch hinzugefügt werden können.

7.1 Ergebnisse

Die Ergebnisse dieser Arbeit werden in die Unterkapitel "Zusammenfassung der Arbeit" sowie der "Web-Shop im Testbetrieb" unterteilt.

7.1.1 Zusammenfassung der Arbeit

Innerhalb dieser Masterarbeit wurde ein Web-Shop zur Erfassung und Abwicklung von Steckperlenbilderbestellungen entworfen und prototypisch implementiert. Der Web-Shop ermöglicht eine dezentrale Steuerung des bereits vorhandenen Steckperlendruckers und kapselt seine Steuerung über das Web nach außen.

Zu diesem Zweck wurden zunächst die Anforderungen an die verschiedenen Komponenten erhoben. Folgend wurden Technologien verglichen, die zur Umsetzung einer solchen Web-Lösung möglich wären. Hierbei haben sich EJB 3.0 und CDI im Back-End Bereich sowie JSF im Bereich des Front-Ends als geeignete Technologien zur Umsetzung der Anforderungen ergeben. Nachfolgend wurden die verschiedenen Aspekte der Applikation konzipiert. Dazu gehörten

neben des Datenbank-Schemas verschiedene Komponenten zur Lösung von Teilproblemen wie beispielsweise der Designer (Erstellung von Steckperlenbildern), die Bildersuche, diverse Galerien, die Bestellübersicht der Mitarbeiter und viele weitere.

Diese vorangegangenen Arbeiten ermöglichten es, dass von der Erstellung des Bildes über die Bestellung des Kunden bis hin zu der Bearbeitung der Bestellungen durch einen Mitarbeiter der gesamte Prozess in dem Web-Shop realisiert wurde.

Bei der Umsetzung des Entwurfs traten lediglich Herausforderungen bei der Ansteuerung des Druckers auf. Hieraus hat sich ergeben, dass eine Steuerung des Druckers mithilfe eines Schedulers nicht praktikabel ist. Ist ein Druck abgeschlossen, muss zunächst ein Mitarbeiter die Steckperlen aus der Vorlage entnehmen, bevor ein weiterer Druck gestartet werden kann. Dies kann nicht automatisiert erfasst und gemeldet werden, weshalb der Druck manuell stattfinden sollte. Hierzu wurden nachträglich die nötigen Änderungen an der Implementierung vorgenommen.

7.1.2 Web-Shop im Testbetrieb

Der Testbetrieb umfasste ein ca. 2-Stunden-Zeitfenster in dem fünf unterschiedliche Benutzer an dem System gearbeitet haben. Dabei traten vier der fünf Benutzer als Kunden und der verbleibende als Mitarbeiter auf.

Der Web-Shop verhielt sich während dieser Testphase sehr stabil und es traten nur wenige Fehler auf. Diese wurden umgehend erfasst und nachfolgend behoben. Aus den Aussagen der Testbenutzer haben sich einige, wenige Verbesserungen für den Shop ergeben. Dazu gehören:

- **Preiszähler im Designer:** Bislang hat der Kunde erst dann gesehen, wie viel ein Bild kostet, als er es in seinem Warenkorb platziert hat. Die Benutzer wünschen sich natürlich eine direkte Preistransparenz. Das heißt, sobald ein Benutzer eine Perle im Designer platziert, möchte er wissen, wie diese Perle den Preis des Bildes beeinflusst.
- **Vorschau der Bilder innerhalb der Druckaufträge:** Der Mitarbeiter hat angemerkt, dass er gerne eine Vorschau des Bildes bei den derzeit eingestellten Druckaufträgen sehen möchte. Aktuell enthielt die Ansicht der Druckaufträge lediglich Meta-Informationen zu dem Druck (Name

des Mitarbeiters, Datum der Erstellung).

- **Ausblenden von beendeten Druckaufträgen:** Bisher gab es keine Möglichkeit, beendete Druckaufträge auszublenden, was dazu führte, dass die Liste der Druckaufträge sehr unübersichtlich wurde. Hierbei wünscht sich der Testmitarbeiter eine Möglichkeit, erledigte Druckaufträge auszublenden.
- **Bereits erledigte Bestellungen sollen in der Bestellübersicht ausgeblendet sein:** Der Mitarbeiter möchte in der Bestellübersicht nur die Bestellungen sehen, die für ihn relevant sind. Deshalb sollen alle bereits erledigten Bestellungen zunächst ausgeblendet werden. Aus Historisierungsgründen soll er aber weiterhin die Möglichkeit haben, diese erneut anzuzeigen um ggf. Rückfragen von Kunden beantworten zu können.

Die soeben beschriebenen Hinweise aus dem Testbetrieb der jeweiligen Testbenutzer wurden direkt als Verbesserungen an dem System ergänzt.

7.2 Ausblick

Neben den Hinweisen aus dem Testbetrieb, haben sich einige längerfristige Funktionen bzw. Ziele herausgestellt, die bei der schlussendlichen Umsetzung des Web-Shops berücksichtigt werden könnten. Diese sind als mögliche Verbesserungsvorschläge zu sehen und sollten zunächst mit den Mitarbeitern und deren längerfristigen Zielen abgeglichen werden. Nachfolgend werden diese näher betrachtet.

- **Konfigurierbare Farben:** Der Steckperlendrucker besitzt eine seperate Konfigurationsdatei, in welcher die unterschiedlichen Farben definiert wurden. Bei der Umsetzung dieses Web-Shops wurden die Farben zunächst fest im Code hinterlegt. Möglicherweise ist es sinnvoll, diese auf der Web-Oberfläche konfigurierbar zu machen.
- **Erweiterung der Zahlungsmethoden:** In der aktuellen Umsetzung unterstützt das System drei unterschiedliche Zahlungsformen. "Per Guthaben", "per Nachnahme" sowie "per Vorkasse". Sinnvoll wären sicherlich weitere Zahlungsformen wie beispielsweise "per Paypal" oder aber "per Kreditkarte".

- **Geeignete AGBs:** Ziel dieser Arbeit war es, eine prototypische Umsetzung eines Web-Shops zu beschrieben. Für den tatsächlichen Betrieb des Shops ist es zwingend notwendig geeignete AGBs (Allgemeine Geschäftsbedingungen) zu formulieren und diese den Benutzern zugänglich zu machen.
- **Unterstützung von mobilen Endgeräten:** Der Shop wurde unter Zuhilfenahme des CSS-Frameworks "Bootstrap" entwickelt. Dieses ermöglicht eine flexible Gestaltung des User Interfaces. So skaliert die Anwendung je nach Auflösung des Benutzers. Eine vollständige Unterstützung mobiler Endgeräte erfordert jedoch noch einige Feinarbeiten sowie einen geeigneten Testbetrieb. In der aktuellen Umsetzung wurde der Shop nicht auf solchen Geräten getestet bzw. für solche optimiert. Hierbei können aktuell noch nicht abschätzbare Aufwände auftreten.

Anhang A

Betriebshandbuch

Zu der Inbetriebnahme des Web-Shops werden folgende Vorrausetzungen benötigt:

- Eine laufende PostgreSQL-Datenbank (Version 9.2)
- Einen Applikation-Server (JBoss Wildfly Version 8.2.0.Final)
- Eine Netzwerkanbindung an den Steckperlendrucker

Nachfolgend wird die Installation der jeweiligen Komponenten beschrieben

A.1 Installation PostgreSQL Datenbank

Der Web-Shop wurde auf der Basis einer PostgreSQL-Datenbank in der Version 9.2 entwickelt. Diese sollte in der genannten Version installiert sowie eingerichtet sein. Dazu muss zunächst die korrekte Distribution von der Internetseite `http://www.postgresql.org/download/` heruntergeladen und die beschriebenen Schritte zum starten befolgt werden.

Ist der PostgreSQL-Prozess gestartet muss man zunächst eine Datenbank anlegen (siehe dazu auch http://www.postgresql.org/createdb.html/).

Befehl	Beschreibung
`createdb -U postgres beaditdb`	Erzeugen einer Datenbank mit dem Namen "beaditdb" mit dem Benutzer "postgres" (Standardbenutzer)
`psql -U postgres beaditdb`	Verbinden mit der eben erzeugten Datenbank
`> CREATE USER beadit WITH PASSWORD '<DB-PASSWORT>';`	Benutzer "beadit" anlegen mit einem eigenen Passwort
`> GRANT ALL PRIVILEGES ON DATABASE 'beaditdb' to beadit;`	Gewähren aller Rechte auf der "beaditdb" dem Benutzer "beadit" zuweisen

Nun haben wir eine Datenbank ("beaditdb") und einen Benutzer ("beadit") mit vollen Rechten auf der Applikationseigenen Datenbank.

A.2 Installation Wildfly Applikationsserver

Der Web-Shop wird auf einem JEE Applikation Server betrieben. Dazu benötigen wir den JBoss Wildfly in der Version 8.2.0.Final (Java EE7 Full & Web Distribution) erhältlich bei JBoss: `http://wildfly.org/downloads/`.

Bevor der Applikationsserver gestartet werden kann, muss zunächst die Verbindung zur Datenbank konfiguriert werden (Datasource). Diese wird in der Konfiguration des Applikationsservers hinterlegt, welche er dann Applikationen bereitstellt, die in ihm deployed wurden. Somit hält der Server die Datenbankverbindung und stellt diese lediglich den Applikationen zur Verfügung. Das hat den Charm, dass die Applikation keinerlei Kenntnis über Passwörter für Datenbanken sowie Verbindungsinformationen (URLs, Ports, SIDs, ...) halten muss.

A.2.1 Installation Datenbanktreiber

Zunächst müssen wir den Datenbanktreiber installieren. In unserem Fall benötigen wir den Treiber für die Version 9.2 (postgresql-9.2-1004.jdbc41.jar). Der Datenbanktreiber muss daraufhin in folgendes Verzeichnis kopiert werden:

`$JBOSS_HOME\modules\org\postgresql\main\`

Des weiteren muss in diesem Verzeichnis eine Datei **module.xml** mit folgendem Inhalten angelegt werden:

Listing A.1: PostgreSQL module.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<module xmlns="urn:jboss:domain:datasources:2.0"
  name="org.postgresql">
  <resources>
    <resource-root path="postgresql-9.2-1004.jdbc41.jar"/>
  </resources>
  <dependencies>
     <module name="javax.api"/>
     <module name="javax.transaction.api"/>
   </dependencies>
</module>
```

A.2.2 Installation JSch

Neben dem Datenbanktreiber nutzt der Webshop eine Bibliothek namens JSch. Diese ist zur Ansteuerung des Drucker notwendig und muss, wie der Datenbanktreiber, zunächst im Applikationsserver installiert werden. Wir benötigen dazu die Bibliothek in der Version 0.152 (`http://sourceforge.net/projects/jsch/files/jsch.jar/0.1.53/jsch-0.1.53.jar/download`). Die Jar muss daraufhin in das Verzeichnis:

`$JBOSS_HOME\modules\com\jcraft\jsch\main\`

kopiert werden. Darüber hinaus benötigen wir auch hier eine beschreibende **module.xml** mit folgendem Inhalt:

Listing A.2: Jsch module.xml

```
<module xmlns="urn:jboss:module:1.0"
  name="com.jcraft.jsch">
  <resources>
    <resource-root path="jsch-0.1.53.jar"/>
  </resources>

  <dependencies>
    <module name="javax.api"/>
  </dependencies>
</module>
```

Der Inhalt der Wildfly modules sollte abschließend wie folgt aussehen:

```
JBOSS_HOME
└── modules
    ├── com
    │   └── postgresql
    │       └── main
    │           ├── module.xml
    │           └── postgresql-9.2-1004.jdbc41.jar
    └── org
        └── jcraft
            └── jsch
                └── main
                    ├── module.xml
                    └── jsch-0.1.53.jar
```

A.2.3 Konfiguration Wildfly

Da wir nun die nötigen Bibliotheken installiert haben, müssen wir die Datasource im Wildfly konfigurieren. Hierzu müssen wir die folgende Konfigurationsdatei anpassen:

`$JBOSS_HOME\standalone\configuration\standalone.xml`

Hierzu müssen wir zunächst den Bereich **Datasources** in der Konfiguration anpassen. Wir fügen eine Datasource mit dem JNDI-Namen **java:jboss/datasources/beaditdb** hinzu, legen die Connection-URL fest sowie den Benutzernamen und das Passwort. Darüber hinaus melden wir den vorher installierten Treiber an der Datasource an. In der Standardinstallation des Wildfly ist zunächst eine **ExampleDS** vorkonfiguriert. Diese sollte nicht verändert

werden. Im Detail sieht die Konfiguration folgendermaßen aus:

Listing A.3: Wildfly standalone.xml Konfiguration

```
<datasources>
  <!-- die vorkonfigurierte DataSource sollte
  weiterhin beibehalten werden -->
  <datasource> ... ExampleDS </datasource>

  <datasource jndi-name="java:jboss/datasources/beaditdb"
        pool-name="beaditdbpool"
        enabled="true"
        use-java-context="true">
    <connection-url>
      jdbc:postgresql://localhost:5432/beaditdb
    </connection-url>
    <driver>postgresql</driver>
    <security>
      <user-name>beadit</user-name>
      <password><!--DB-PASSWORT--></password>
    </security>
  </datasource>
  <drivers>
    <driver name="postgresql" module="org.postgresql">
      <xa-datasource-class>
        org.postgresql.xa.PGXADataSource
      </xa-datasource-class>
    </driver>
  </drivers>
</datasources>
```

Die Werte *connection-url*, *user-name* und das *password* müssen ggf. angepasst werden, je nachdem wo die Datenbank läuft und wie die Zugangsdaten zu dieser sind.

A.2.4 Einrichtung der Email-SMTP Verbindung

Der Web-Shop benötigt eine konfigurierte SMTP Verbindung mit dem JNDI-Namen:

`java:jboss/mail/beadIt`

Die wird z.B. dafür benötigt, Bestellbestätigungen sowie Kontodaten zu verschicken.

Eine hervorragende Anleitung ist zu finden unter `http://www.opensiam.com/2014/06/wildfly-8-sending-email-using-gmail.html`.

A.3 Deployen des Artefakts

Nachdem der Applikationsserver nun soweit eingerichtet ist, kann das Artefakt (zu finden auf der abgegebenen CD) auf ihm deployed werden.

Dazu muss lediglich das Artefakt (**beadIt.war**) in folgendes Verzeichnis kopiert werden.

`$JBOSS_HOME\standalone\deployments`

Dies kann auch im laufenden Betrieb des Servers passiert. Er scannt in regelmäßigen Abständen das Verzeichnis auf neue, deploybare Artefakte. Findet der Wildfly ein Artefakt, so legt er eine Datei **<artefaktname>.isdeploying** in dem Verzeichnis ab. Nach Abschluss des Deployments legt er entweder eine Datei **<artefaktname>.deployed** oder **<artefaktname>.failed** an, welche den finalen Status des Deployments beschreibt. Insofern das Deployment erfolgreich verlaufen ist, kann die Applikation nun unter folgender Adresse von dem lokalen Rechner erreicht werden:

`http://localhost:8080/beadIt/`

Literaturverzeichnis

[Aga05] Agarwal, R.: *Software Development vs Software Maintenance.* Oktober 2005. – Online erhältlich unter `http://www.irahul.com/2005/10/software-development-vs-software.html`; abgerufen am 05. Juli 2015

[And15] Anderson, S: *Best Screen Resolution to Design Websites.* Juli 2015. – Online erhältlich unter `http://www.hobo-web.co.uk/best-screen-size/`; abgerufen am 24. Juli 2015

[Bay02] Bayer, T: *REST Web Services.* November 2002. – Online erhältlich unter `http://www.oio.de/public/xml/rest-webservices.htm`; abgerufen am 22. März 2015

[BBT14] Bring, S ; Böhn, R ; Tarasiewicz, P: *AngularJS Online-Buch.* November 2014. – Online erhältlich unter `http://angularjs.de/buch`; abgerufen am 06. März 2015

[Bec02] Beck, K: *Clean Code: A Handbook of Agile Software Craftsmanship.* Boston : Addison-Wesley, 2002 (1. Auflage)

[BM01] Burns, E ; McClanahan, C: *JSR 127: JavaServer Faces.* Mai 2001. – Online erhältlich unter `https://www.jcp.org/en/jsr/detail?id=127`; abgerufen am 02. Mai 2015

[Coh04] Cohn, M: *User Stories Applied - For Agile Software Development.* Boston : Addison-Wesley, 2004 (1. Auflage)

[Del12] Delor, T: *Java EE 6 VS Spring 3 : Java EE has killed Spring? No way!* Juli 2012. – Online erhältlich unter `http://invalidcodeexception.com/java-ee-6-vs-spring/`; abgerufen am 02. März 2015

[DES13] Düvel, C ; Erck, N ; Steinhöfer, D: *EJB 3.1 professional.*

Paderborn : dpunkt.verlag, 2013 (2. Auflage)

[GS13] Green, B ; Seshadri, S: *AngularJS*. Gravenstein : O Reilly Media Inc, 2013 (3. Auflage)

[Gup12] Gupta, A: *Why is Java EE 6 better than Spring ?* März 2012. – Online erhältlich unter `https://blogs.oracle.com/arungupta/entry/why_java_ee_6_is`; abgerufen am 12. Mai 2015

[Gup13] Gupta, A: *Java EE 7 - Essentials*. O Reilly Media Inc, 2013 (1. Auflage)

[JCNE+14] Jendrock, E ; Cervera-Navarro, R ; Evans, I ; Haase, K ; Markito, W: *Oracle - The Java EE 7 Tutorial*. September 2014. – Online erhältlich unter `http://docs.oracle.com/javaee/7/tutorial/doc/home.htm`; abgerufen am 20. Mai 2015

[JHD14] Johnson, R ; Hoeller, J ; Donald, K: *Spring Framework Reference Documentation - 4.1.1. Release*. April 2014. – Online erhältlich unter `http://docs.spring.io/spring/docs/4.1.x/spring-framework-reference/pdf/spring-framework-reference.pdf`; abgerufen am 10. Juni 2015

[Kos15] Koshelko, E: *Why you should not use AngularJS*. Januar 2015. – Online erhältlich unter `https://medium.com/@mnemon1ck/why-you-should-not-use-angularjs-1df5ddf6fc99`; abgerufen am 22. März 2015

[KW06] Kodali, R ; Wetherbee, J: *EJB 3 - Application Development, From Novice to Professional*. Berkeley : Apress, 2006 (Auflage 2006)

[Mar08] Martin, R: *Test Driven Development. By Example*. New Jersey : Prentice Hall, 2008 (1. Auflage)

[MKM10] Marinschek, M ; Kurz, M ; Müllan, G: *JavaServer Faces 2.0 - Grundlagen und erweitert Konzepte*. Paderborn : dpunkt.verlag, 2010 (2. Auflage)

[Rah14] Rahman, R: *Spring and Java EE Side by Side*. Mai 2014. – Online erhältlich unter `http://de.slideshare.net/reza_rahman/java-ee-and-spring-sidebyside-34320697`; abgerufen am 15. April 2015

[Sal09] Salatian, A: *Model View Controller*. Juli 2009. – Online erhältlich unter `http://computerservicesforyou.com/ecommerce/lecture2.pdf`; abgerufen am 10. April 2015

[Sot14] Sotelo, C: *Migrating to AngularJS 2.0*. November 2014. – Online erhältlich unter `http://paislee.io/migrating-to-angularjs-2-0/`; abgerufen am 22. März 2015

[Zie13] Ziegler, H: *BeadBox: Entwicklung eines informatischen Systems zur automatisierten Erstellung von Bügelperlenbildern*. Universität Oldenburg, 2013

Tabellenverzeichnis

Glossar

Back-End Das Back-End besteht aus der serverseitigen Implementierung. Diese setzt sich aus dem Model und der eigentlichen Geschäftslogik zusammen und sollte, strikt von der eigentlichen Präsentationsschicht (Front-End) getrennt werden.. 5, 28, 45, 71

CDI **C**ontexts and **D**ependency **I**njection. CDI ist ein Framework zur einfachen Verwaltung von Resourcen. Resourcen können dabei diverse Arten von Beans wie auch fachliche Ergebnisse sein. So kann man durch einfache Injects an verschiedenen Stellen im Java-Code auf diverse Resourcen zugreifen, ohne sich um deren Herkunft zu kümmern. CDI ist Bestandteil der JEE-Spezifikation.. 6, 9, 24, 26, 28, 30, 45, 50, 71

EJB **E**nterprise **J**ava **B**eans. EJB ist Framework zu erleichterten Entwicklung von mehrschichtigen, verteilten Systemen. Dabei steuert der EJB-Container eine Vielzahl von unterschiedlichen Verhalten, wie beispielsweise das Transaktionsverhalten.. 6, 8, 9, 24, 26, 28, 30, 45, 71

Front-End Das Front-End oder auch der Client steht für die Präsentationsschicht einer Web-Applikation. Sie umfasst die Komponenten die dem Benutzer dargestellt werden (z.B. ein Button, ein Textfeld, ...) sowie die Verarbeitungslogik zur Auswertung der Eingaben (Konvertierungen, Validierungen, ...).. 5, 27, 28, 45

JEE **J**ava Platform, **E**nterprise **E**dition. JEE ist ein Überbegriff für eine Sammlung von Spezifikationen. Dazu gehören unter anderem JSF, CDI, EJB und viele weitere. Die JEE-Spezifikation dient im Prinzip wie eine Technologie- und Architektur-Vorlage zur Erstellung, großer Software-Systeme.. 24–26, 76

JNDI **J**ava **N**aming and **D**irectory **I**nterface. JNDI steht im Grunde für eine Art Objekt- oder Resourcenverzeichnis. Diese Resourcen können über eindeutige Namen adressiert und manipuliert werden. Häufig wird dieses bei der Bereitstellung von Resourcen zwischen verschiedenen Applikationen genutzt, wie beispielsweise eine Datasource oder eine E-Mail-Resource.. 9, 78

JSF **J**ava **S**erver **Faces**. "JSF ist ein Serverseitiges Komponentenframework zur Erstellung von Java technologiebasierten Web Applikationen" [MKM10]. 9, 12, 13, 26–28, 30, 45, 48, 50, 71

REST **RE**presentational **S**tate **T**ransfer ist ein Protokoll zur Kommunikation vor allem zwischen Maschinen. REST baut dabei auf dem HTTP auf und unterstützt dabei HTTP-Methoden wie beispielsweise *GET*, *POST*, und viele weitere.. 36, 37

TDD **T**est **D**riven **D**evelopment. TDD beschreibt ein Vorgehensmodell bei der Erstellung von Softwarekomponenten. Dabei wird zunächst der Test erstellt, bevor die eigentliche Geschäftslogik implementiert wird. Dies soll vor allem dem Entwickler helfen, sich zunächst über die Anforderungen an die Logik Gedanken zu machen. Das sorgt dafür, dass zuallererst die Fachlichkeit im Vordergrund steht und nicht die eigentliche, technische Realisierung.. 48

User-Stories Eine User-Story ist eine kurze Beschreibung einer Anforderung aus Sicht der fordernden Rolle (z.B. Kunde, Mitarbeiter, ...). Dabei ist es entscheidend, dass eine User-Story möglichst wenig Abhängigkeiten zu anderen Anforderungen haben soll.. 17, 28

Index

Versicherung

Hiermit versichere ich, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegeben Quellen und Hilfsmittel benutzt habe. Außerdem versichere ich, dass ich die allgemeinen Prinzipien wissenschaftlicher Arbeit und Veröffentlichung, wie sie in den Leitlinien guter wissenschaftlicher Praxis der Carl von Ossietzky Universität Oldenburg festgelegt sind, befolgt habe.

Simon Jakubowski