Prozessoptimierung mit mobilen Devices

Bachelorarbeit
FS 2015
Abteilung Informatik
Hochschule für Technik Rapperswil

Autoren: Marcel Loop, Philipp Koster
Betreuer: Hans Rudin
Projektpartner: Hilding Anders Switzerland AG

Erklärung der Eigenständigkeit

Ich erkläre hiermit,

- dass ich die vorliegende Arbeit selber und ohne fremde Hilfe durchgeführt habe, ausser derjenigen, welche explizit in der Aufgabenstellung erwähnt ist oder mit dem Betreuer schriftlich vereinbart wurde,
- dass ich sämtliche verwendeten Quellen erwähnt und gemäss gängigen wissenschaftlichen Zitierregeln korrekt angegeben habe.
- das ich keine durch Copyright geschützten Materialien (z.B. Bilder) in dieser Arbeit in unerlaubter Weise genutzt habe.

Ort, Datum:	Rapperswil, 11.06.2015
Name, Unterschrift	Marcel Loop

Philipp Koster

Inhaltsverzeichnis

Teil I.

Abstract

Abstract

Das KMU Hilding Anders Switzerland AG ist in der Schweiz unter dem Namen Bico bekannt und stellt seit über 150 Jahren Matratzen und Einlegerahmen her. Die Produktion arbeitet mit Losgrösse 1 und kann bis zu 600 Matratzen am Tag herstellen. Die Workcenter-Arbeitsaufträge werden täglich als Sequenzliste auf Papier an den Workcenter abgegeben und entsprechend abgearbeitet. Die Lösung ist nicht mehr zeitgemäss und unflexibel bezüglich kurzfristigen Annullierungen und Express Aufträgen.

Im Rahmen dieser Arbeit wurde einen Prototyp entwickelt, womit die Sequenzlisten durch Tablets ersetzt werden. Die Produktionsdaten des Tages werden zum Tagesbeginn, aus dem ERP, in das neu entwickelte System importiert. Jeder Mitarbeiter, der bisher mit einem Blatt Papier gearbeitet hat, bekommt ein Tablet, worauf er die Aufträge sieht, die er zu erledigen hat.

Nach Abschluss eines Auftrages markiert er diesen als erledigt. Der Produktionsleiter hat dadurch jederzeit einen Überblick über den aktuellen Produktionsstatus und auch die Mitarbeiter sehen untereinander, wer an was arbeitet. Zusätzlich haben die Mitarbeiter die Möglichkeit über das Tablet ein Problem zu melden, woraufhin der Produktionsleiter informiert wird.

Der entwickelte Prototyp besteht aus einem Server, einer Android Applikation für die Tablets, sowie einer Web-Applikation, in der die Produktionsdaten importiert und der Produktionsstatus überwacht werden kann.

Die Arbeit wurde soweit getrieben, dass neben einer umfangreichen Ist-Analyse, ein User Acceptance Test mit dem fertigen Prototyp durchgeführt wurde, bei dem dieser in der Live Produktion die Sequenzliste eines Mitarbeiters ersetze. Dieser war ein voller Erfolg und die Hilding Anders Switzerland AG wird das Produkt voraussichtlich komplettieren lassen, um den Betrieb in naher Zukunft auf diese moderne Lösung umzustellen.

Teil II.

Management Summary

1. Management Summary

1.1. Ausgangslage

Die Hilding Anders Switzerland AG ist in der Schweiz unter dem Namen Bico bekannt und stellt Matratzen und Einlegerahmen her. Die Produktion arbeitet mit Losgrösse 1 und kann bis zu 600 Matratzen am Tag herstellen.

Die Produktion ist in mehrere Workcenter aufgeteilt. Die Informationen für die Produktion entnehmen die Workcenter den täglich erstellten Sequenzlisten. Die Daten für die Produktion werden von einem Excel Makro aus dem ERP System Movex gelesen, woraus Sequenzlisten generiert werden.

Folgende Grafik zeigt einen Überblick des jetzigen Systems.

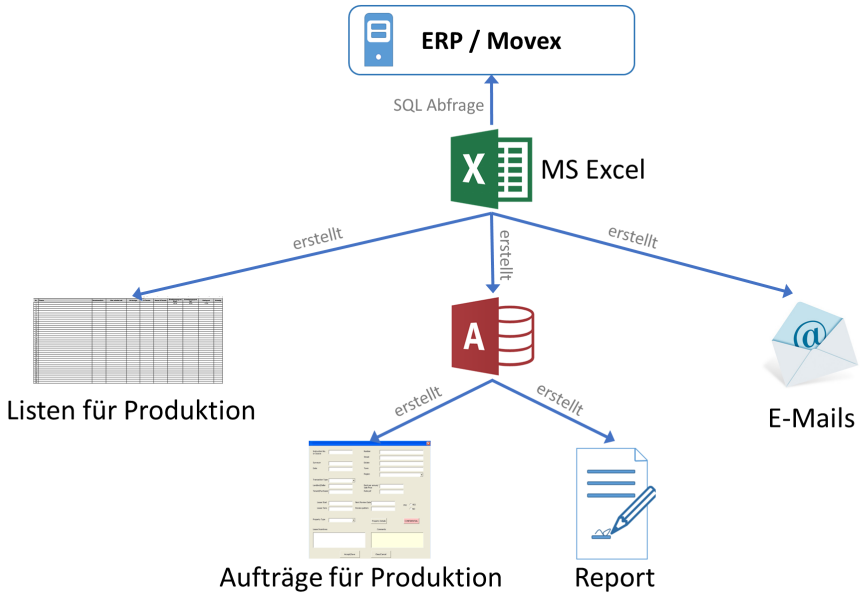


Abbildung 1.1.: Kontextdiagramm der Ausgangslage

Das abgebildete System ist nicht mehr zeitgemäss und bringt unter anderem folgende Nachteile mit sich:

- Der Produktionsleiter hat keine Übersicht über den aktuellen Arbeitsfortschritt.
- Die Workcenter-Mitarbeiter müssen Rückmeldungen, wie Probleme über das Telefon an den Produktionsleiter melden.
- Der Produktionsleiter kann keine Annullierungen vornehmen.
- Arbeitsaufträge können nicht priorisiert werden.

1.2. Ziele der Arbeit

Ziel dieser Arbeit ist ein Prototyp für die Ablösung des bestehenden nicht zeitgemässem Systems durch eine moderne Lösung, bei der alle Workcenter-Mitarbeiter mit Tablets ausgestattet werden. Der Produktionsleiter soll mit Hilfe einer Web-App Aufträge importieren und den Status der Produktion verfolgen können.

Die Daten sollen zu Tagesbeginn aus dem ERP System gelesen und ins neue System importiert werden. Die Workcenter-Mitarbeiter entnehmen ihren Tablets, welche Aufgaben zu erledigen sind. Daraufhin können sie diese bearbeiten und als erledigt markieren. Daraus ergeben sich Vorteile, wie die Möglichkeit den Produktionsstatus zu überwachen. Zudem ist eine Basis für zukünftige Erweiterungen gegeben, wie zum Beispiel das Erfassen von Problemen, Annullierungen oder Priorisieren von Aufträgen.

Im Rahmen dieser Arbeit wird eine Serverapplikation, ein Webclient und ein Android Client entwickelt, wobei über eine REST-Schnittstelle mit dem Server kommuniziert wird.

1.3. Vorgehen

Die Arbeit wurde mit einem iterativen Ansatz geplant und durchgeführt.

Zu Beginn musste eine umfangreiche Ist-Analyse des aktuellen Produktionsablaufs erstellt werden. In Zusammenarbeit mit der Hilding Anders AG wurden Sitzungen und Besichtigungen der Produktion durchgeführt. Dabei wurden wertvolle Informationen über den jetzigen Arbeitsprozess gewonnen, wie zum Beispiel die Verwendung der Sequenzliste. Bei Unklarheiten konnte direkt der Workcenter-Mitarbeiter befragt werden. Eine korrekte Ist-Analyse war die Grundvoraussetzung für das erfolgreiche Durchführen dieser Arbeit.

Das Resultat der Ist-Analyse ist folgendes Aktivitätsprozessdiagramm, welches das Zusammenspiel der Workcenter widerspiegelt.

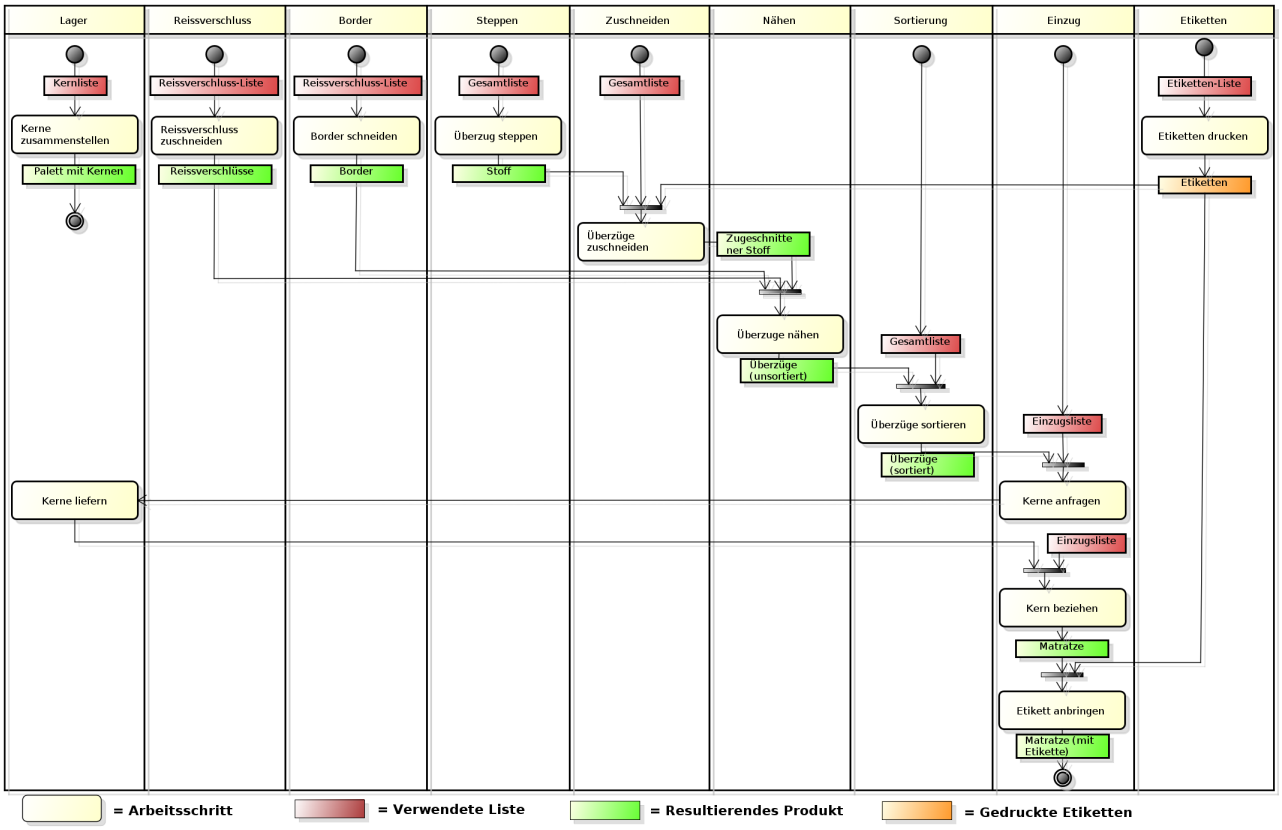


Abbildung 1.2.: Ablauf Produktionsprozess

Nachdem die Anforderungen definiert und abgesegnet waren, startete die Implementationsphase, wobei darauf geachtet wurde, regelmässig mit dem Kunden in Kontakt zu sein, um den aktuellsten Stand zu präsentieren. Im Verlauf der Implementationsphase wuchs die Anzahl der Sitzungsteilnehmer stetig an, sodass am Ende nebst dem IT-Controlling der Produktionsleiter sowie die Meister aus den Workcentern der Sitzung beiwohnten.

Um eine gute Qualität des Codes zu gewährleisten, wurde viel Wert auf das Unit-Testing gelegt. Zudem wurde jede Codeänderung von einem anderen Entwickler überprüft und erst dann in das Produkt eingespielt.

1.4. Technologien

Die Technologien waren frei wählbar. Die Serverapplikation wurde in Java implementiert und läuft in einem Servlet-Container. Der Client wurde ebenfalls in Java geschrieben, da dieser auf einem Android Tablet zum Einsatz kommt. Das Cockpit wurde als Webapplikation mit HTML, CSS und AngularJS realisiert.

1.5. Ergebnisse

Das Ergebnis ist ein funktionsfähiger Prototyp, der es dem Workcenter Mitarbeiter ermöglicht, Aufträge abzuarbeiten und Probleme zu melden, wobei der Produktionsleiter den Produktionsstatus mit Hilfe des Cockpits überprüfen kann.

Übersicht des entwickelten Systems.

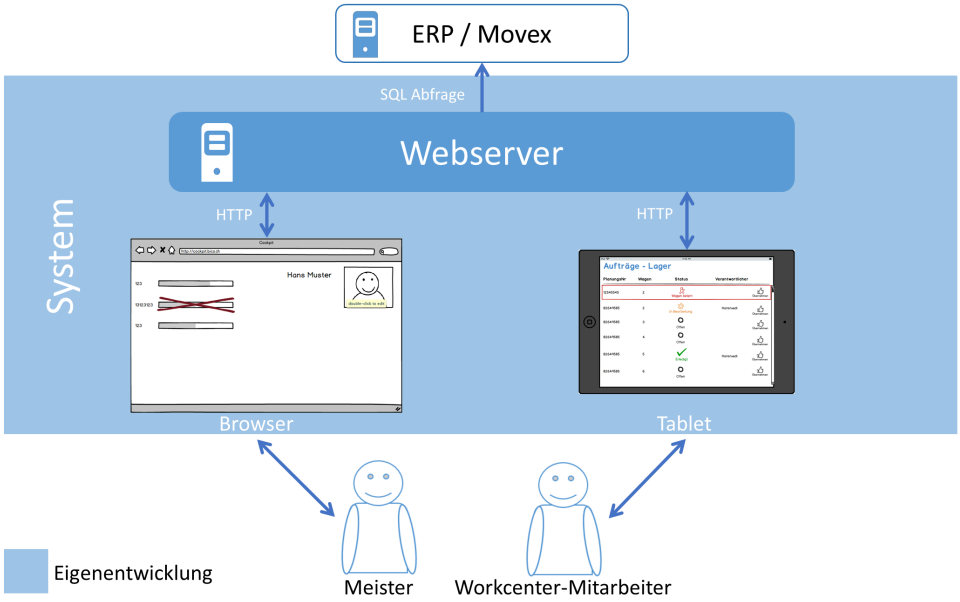


Abbildung 1.3.: Kontextdiagramm des entwickelnden Systems

Der Prototyp wurde gegen Projektende im Rahmen eines 'User Acceptance Tests' in der Produktion getestet. Dabei ersetzte ein Workcenter-Mitarbeiter seine Papierliste mit dem Tablet und versuchte, wie gewohnt seine Arbeit zu erledigen.

Folgendes Bild zeigt zeigt Teile des Clients, des Cockpits sowie eine Aufnahme des User Acceptance Test.

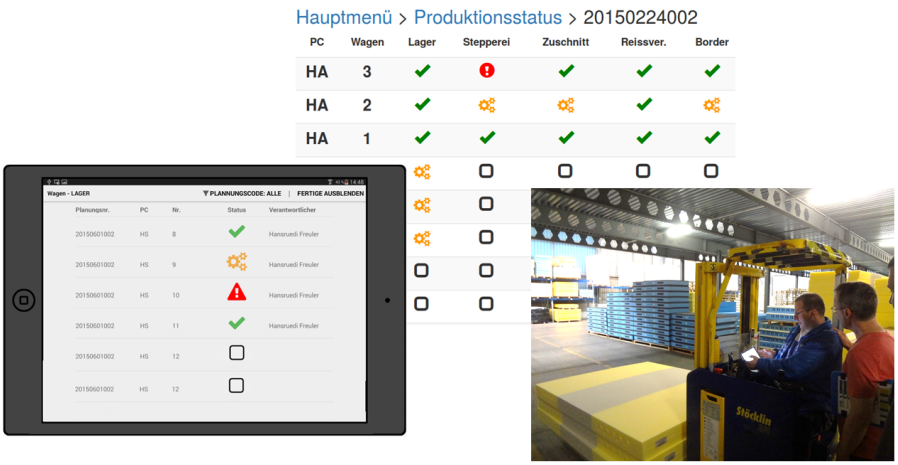


Abbildung 1.4.: Android Client, Cockpit und User Acceptance Test

1.6. Ausblick

Nicht zuletzt durch den erfolgreichen 'User Acceptance Test' hat die Hilding Anders AG sich entschieden, das Projekt weiter zu führen. Gegen Ende wurden im Rahmen dieser Arbeit die Arbeitspakete definiert und geschätzt, welche für den produktiven Einsatz der Software implementiert werden müssen. Die Schätzungen ergaben einen Aufwand von 90 Personenstunden. Wobei beachtet werden muss, dass die reine Entwicklungszeit geschätzt wurde. Die Einarbeitung in das Projekt sowie weitere User Acceptance Test inklusive deren Auswertung und Nachbearbeitung wurden bei der gemachten Schätzung nicht berücksichtigt.

Teil III.

Einführung und Aufagbenstellung

2. Einführung und Aufgabenstellung

Das Kapitel 'Einführung und Aufgabenstellung' gibt einen Überblick über den Aufbau der Projektdokumentation und zeigt die offizielle Aufgabenstellung, die zu Beginn der Bachelorarbeit verteilt wurde.

2.1. Einführung

Die Projektdokumentation beginnt mit der **Ist-Analyse**, in welcher der bestehende Produktionsprozess der Hilding Anders Switzerland AG analysiert wurde. Ein wichtiger Teil davon ist das erstellte Aktivitätsdiagramm, mit welchem die Bereiche, sowie deren Zusammenspiel visualisiert wird.

Auf die Ist-Analyse folgt die **Anforderungsspezifikation**, in welcher die funktionalen sowie nicht funktionalen Anforderungen an das zu erstellende Produkt spezifiziert werden.

In der **Evaluation** wird die Hard- und Software für die Umsetzung evaluiert. Hier stellte sich beispielsweise die Frage, ob der Client als Web- oder Native-Applikation implementiert wird.

In der darauf folgenden **Domainanalyse** geht es darum die Problemdomäne mit Hilfe eines Domain Model zu modellieren. Dieses bildet die Grundlage für die die Umsetzung der Komponenten.

Den grössten Teil der Dokumentation nimmt die **Architektur** ein. In dieser sind diverse UML-Elemente enthalten, sowie detaillierte Beschreibungen zur Umsetzung des Clients, Cockpits und Servers.

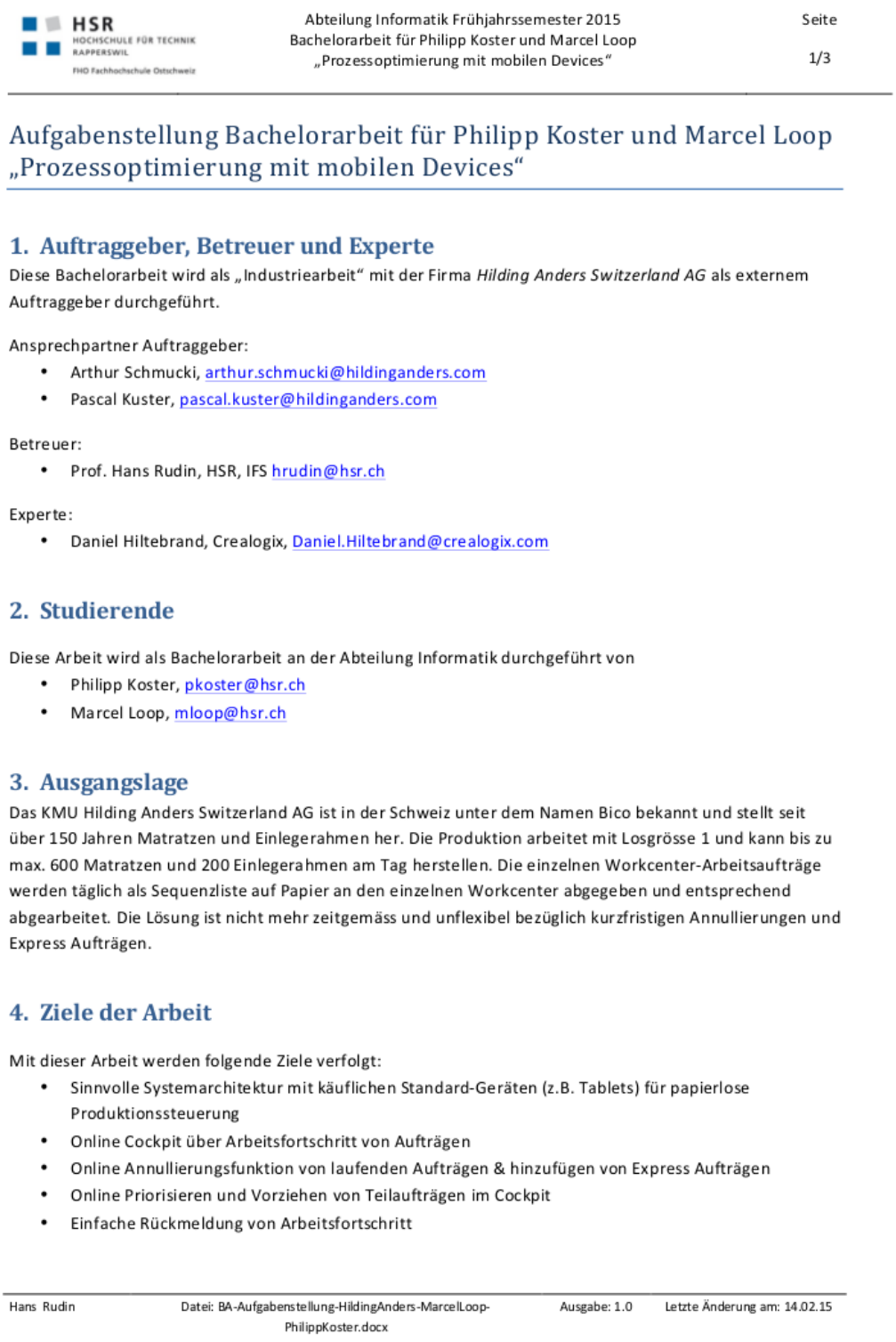
Beim **User Acceptance Test** wird beschrieben und bildlich dargestellt, wie der erstellte Prototyp in der produktiven Umgebung von einem Mitarbeiter der Hilding Anders AG eingesetzt wurde.

Das nächste Kapitel **Qualitätssicherung** beschreibt, wie entwickelt wurde, sowie die Sicherstellung einer guten Code Qualität.

Beim Kapitel **Erreichte Ziele / Offene Punkte** wird gegenübergestellt was erreicht wurde und was noch für den produktiven Einsatz der Software fehlt.

Die Dokumentation endet mit dem Kapitel **Inbetriebnahme**, in dem beschrieben wird, wie die Komponenten in Betrieb genommen werden können.

2.2. Aufgabenstellung



HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
FHO Fachhochschule Ostschweiz

Abteilung Informatik Frühjahrssemester 2015
Bachelorarbeit für Philipp Koster und Marcel Loop
„Prozessoptimierung mit mobilen Devices“

Seite 1/3

Aufgabenstellung Bachelorarbeit für Philipp Koster und Marcel Loop „Prozessoptimierung mit mobilen Devices“

1. Auftraggeber, Betreuer und Experte

Diese Bachelorarbeit wird als „Industriearbeit“ mit der Firma *Hilding Anders Switzerland AG* als externem Auftraggeber durchgeführt.

Ansprechpartner Auftraggeber:

- Arthur Schmucki, arthur.schmucki@hildinganders.com
- Pascal Kuster, pascal.kuster@hildinganders.com

Betreuer:

- Prof. Hans Rudin, HSR, IFS hrudin@hsr.ch

Experte:

- Daniel Hiltebrand, Crealogix, Daniel.Hiltebrand@crealogix.com

2. Studierende

Diese Arbeit wird als Bachelorarbeit an der Abteilung Informatik durchgeführt von

- Philipp Koster, pkoster@hsr.ch
- Marcel Loop, mloop@hsr.ch

3. Ausgangslage

Das KMU Hilding Anders Switzerland AG ist in der Schweiz unter dem Namen Bico bekannt und stellt seit über 150 Jahren Matratzen und Einlegerahmen her. Die Produktion arbeitet mit Losgrösse 1 und kann bis zu max. 600 Matratzen und 200 Einlegerahmen am Tag herstellen. Die einzelnen Workcenter-Arbeitsaufträge werden täglich als Sequenzliste auf Papier an den einzelnen Workcenter abgegeben und entsprechend abgearbeitet. Die Lösung ist nicht mehr zeitgemäss und unflexibel bezüglich kurzfristigen Annullierungen und Express Aufträgen.

4. Ziele der Arbeit

Mit dieser Arbeit werden folgende Ziele verfolgt:

- Sinnvolle Systemarchitektur mit käuflichen Standard-Geräten (z.B. Tablets) für papierlose Produktionssteuerung
- Online Cockpit über Arbeitsfortschritt von Aufträgen
- Online Annullierungsfunktion von laufenden Aufträgen & hinzufügen von Express Aufträgen
- Online Priorisieren und Vorziehen von Teilaufträgen im Cockpit
- Einfache Rückmeldung von Arbeitsfortschritt

Hans Rudin | Datei: BA-Aufgabenstellung-HildingAnders-MarcelLoop-PhilippKoster.docx | Ausgabe: 1.0 | Letzte Änderung am: 14.02.15

Abbildung 2.1.: Aufgabenstellung Seite 1

- Materialabruf (KANBAN) für Halbfabrikate & Rohmaterial via „Chat-Message" innerhalb Produktion evtl. mit integriertem Barcode-Scanner.

5. Aufgabenstellung

Die Arbeit beinhaltet folgende Teilaufgaben:

- IST-Analyse
- Systemarchitektur mit Kostenanalyse für Wirtschaftlichkeitsnachweis
- Realisierung System als Prototyp
- Dokumentation

Details sowie die Gewichtung der Teilaufgaben werden im Verlauf der Arbeit mit dem Auftraggeber vereinbart.

6. Zur Durchführung

Mit dem Betreuer finden wöchentliche Besprechungen statt. Zusätzliche Besprechungen sind nach Bedarf durch die Studierenden zu veranlassen. Besprechungen mit dem Auftraggeber sind von den Studierenden nach Bedarf zu initialisieren. Diese können auch mit den Besprechungen mit dem Betreuer zusammengelegt werden.

Alle Besprechungen sind von den Studierenden mit einer Traktandenliste vorzubereiten, die Besprechung ist durch die Studierenden zu leiten und die Ergebnisse sind in einem Protokoll festzuhalten, das dem Betreuer und dem Auftraggeber zur Verfügung gestellt wird.

Für die Durchführung der Arbeit ist ein Projektplan zu erstellen. Dabei ist auf einen kontinuierlichen und sichtbaren Arbeitsfortschritt zu achten. An Meilensteinen gemäss Projektplan sind einzelne Arbeitsresultate in vorläufigen Versionen abzugeben. Über die abgegebenen Arbeitsresultate erhalten die Studierenden ein vorläufiges Feedback. Eine definitive Beurteilung erfolgt auf Grund der am Abgabetermin abgelieferten Resultate.

7. Dokumentation

Über diese Arbeit ist eine Dokumentation gemäss den Richtlinien der Abteilung Informatik zu verfassen (siehe https://www.hsr.ch/Allgemeine-Infos-Diplom-Bach.4418.0.html). Die zu erstellenden Dokumente bzw. Berichtsteile sind im Projektplan festzuhalten. Alle Dokumente sind nachzuführen, d.h. sie sollten den Stand der Arbeit bei der Abgabe in konsistenter Form dokumentieren. Alle Resultate sind vollständig auf CD/DVD in 3 Exemplaren abzugeben.

8. Termine

Siehe auch Terminplan auf https://www.hsr.ch/Termine-Diplom-Bachelor-und.5142.0.html.

Abbildung 2.2.: Aufgabenstellung Seite 2

- Materialabruf (KANBAN) für Halbfabrikate & Rohmaterial via „Chat-Message" innerhalb Produktion evtl. mit integriertem Barcode-Scanner.

5. Aufgabenstellung

Die Arbeit beinhaltet folgende Teilaufgaben:

- IST-Analyse
- Systemarchitektur mit Kostenanalyse für Wirtschaftlichkeitsnachweis
- Realisierung System als Prototyp
- Dokumentation

Details sowie die Gewichtung der Teilaufgaben werden im Verlauf der Arbeit mit dem Auftraggeber vereinbart.

6. Zur Durchführung

Mit dem Betreuer finden wöchentliche Besprechungen statt. Zusätzliche Besprechungen sind nach Bedarf durch die Studierenden zu veranlassen. Besprechungen mit dem Auftraggeber sind von den Studierenden nach Bedarf zu initialisieren. Diese können auch mit den Besprechungen mit dem Betreuer zusammengelegt werden.

Alle Besprechungen sind von den Studierenden mit einer Traktandenliste vorzubereiten, die Besprechung ist durch die Studierenden zu leiten und die Ergebnisse sind in einem Protokoll festzuhalten, das dem Betreuer und dem Auftraggeber zur Verfügung gestellt wird.

Für die Durchführung der Arbeit ist ein Projektplan zu erstellen. Dabei ist auf einen kontinuierlichen und sichtbaren Arbeitsfortschritt zu achten. An Meilensteinen gemäss Projektplan sind einzelne Arbeitsresultate in vorläufigen Versionen abzugeben. Über die abgegebenen Arbeitsresultate erhalten die Studierenden ein vorläufiges Feedback. Eine definitive Beurteilung erfolgt auf Grund der am Abgabetermin abgelieferten Resultate.

7. Dokumentation

Über diese Arbeit ist eine Dokumentation gemäss den Richtlinien der Abteilung Informatik zu verfassen (siehe https://www.hsr.ch/Allgemeine-Infos-Diplom-Bach.4418.0.html). Die zu erstellenden Dokumente bzw. Berichtsteile sind im Projektplan festzuhalten. Alle Dokumente sind nachzuführen, d.h. sie sollten den Stand der Arbeit bei der Abgabe in konsistenter Form dokumentieren. Alle Resultate sind vollständig auf CD/DVD in 3 Exemplaren abzugeben.

8. Termine

Siehe auch Terminplan auf https://www.hsr.ch/Termine-Diplom-Bachelor-und.5142.0.html.

Abbildung 2.3.: Aufgabenstellung Seite 3

Teil IV.

Ist-Analyse

3. Ist-Analyse

Bis zur fertigen Matratze durchleben die einzelnen Komponenten wie die Kerne, die Stoffe, und die Reissverschlüsse mehrere Arbeitsschritte. Die Ist-Analyse soll den jetzigen Arbeitsprozess beschreiben, welche die Hilding Anders AG für die Produktion benötigt. Um das Ziel dieser Arbeit zu erreichen ist es essentiell diese Prozesse genau zu verstehen. In den folgenden Abschnitten werden die Prozesse beschrieben, wie die Hilding Anders AG ihre tägliche Produktion plant und durchführt.

3.1. Übersicht

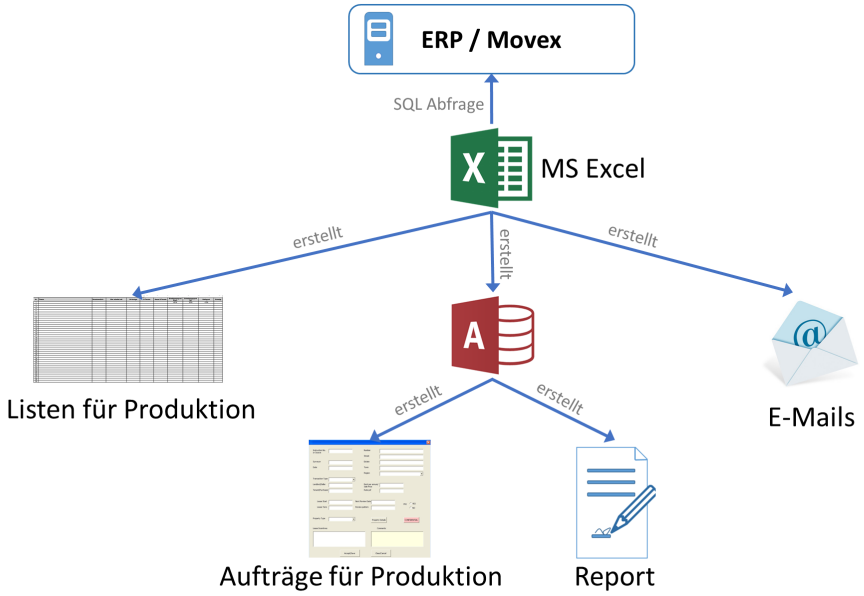


Abbildung 3.1.: Kontextdiagramm

Das Diagramm 3.1 zeigt einen Überblick, woher die Produktionsinformationen stammen und wo sie verarbeitet werden. Makros, welche in VBA (Visual Basic for Application) geschrieben sind, lesen die jeweiligen Produktionsdaten aus dem ERP System Movex. Dabei werden folgende Listen in Excel generiert.

- Kernlisten
- Reissverschlussliste
- Einzugliste
- Etikettenliste
- Textliste

Die generierten Listen werden von den Workcenter verwendet.

Weiter befüllt das Makro eine Access Datenbank, aus welcher wiederum ein Report und ein Formular erstellt wird, welches für die Auftragssteuerung benutzt werden kann.

Nachdem Erzeugen der Listen werden ausserdem Mails generiert, die einen Link auf die generierten Listen enthalten. Dies wurde aus ökologischen sowie ökonomischen Gründen gemacht, da so Papier gespart werden kann.

Nachfolgend wird der Produktionsprozess (Kapitel 3.2) und die Workcenter (Kapitel 3.3) beschrieben. Daraufhin folgt eine Spezifikation der generierten Listen und deren Verwendung (Kapitel 3.4).

3.2. Produktionsprozess

Nachfolgendes Aktivitätsdiagramm zeigt den kompletten Produktionsprozess, welcher alle Workcenter durchläuft. Das Diagramm dient unter anderem als Übersicht der Abhängigkeiten, zwischen den verschiedenen Workcenter. Die jeweils verwendeten Listen sind rot, die resultierenden Produkte grün gefärbt.

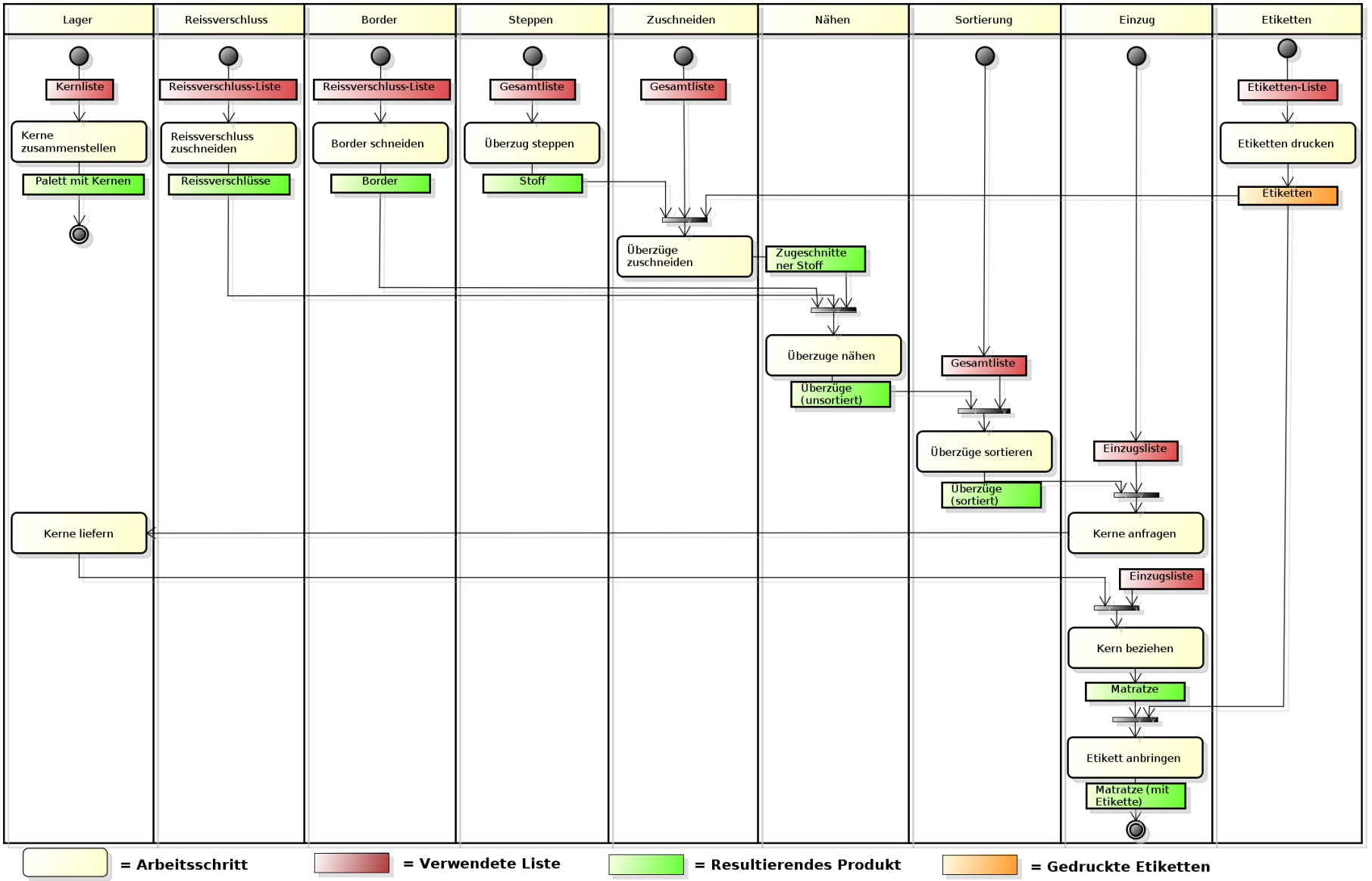


Abbildung 3.2.: Ablauf Produktionsprozess

3.3. Workcenter

Die Bereiche des Produktionsprozess werden bei der Hilding Anders AG Workcenter genannt. Wenn ein Produkt am Tag x fertig sein muss, werden alle Einzelkomponenten am Tag x - 1 hergestellt. Das Workcenter Einzug kann dann am Tag x die Matratze fertigstellen. Nachfolgend werden die einzelnen Workcenter und deren Zusammenarbeit genau beschrieben.

3.3.1. Lager

Im Lager werden mit Hilfe der Kernliste die Kerne für den nächsten Produktionstag bereit gestellt. Die Kerne können aus Platz Gründen nicht direkt zum Überzug gebracht werden, weswegen die gerade benötigten Kerne auf Anfrage des Überzugs Workcenter in die Produktionshalle gebracht werden. Die Arbeiter organisieren sich untereinander, wer welchen Auftrag übernimmt.

Abbildung 3.3.: Workcenter - Lager

3.3.2. Reissverschluss

Im Reissverschluss Workcenter werden die Reissverschlüsse mit Hilfe der Reissverschlussliste für den nächsten Produktionstag zusammengestellt. Dafür werden sie an ein Brett gehängt, an welchem sich nummerierte Hacken befinden.

Abbildung 3.4.: Workcenter - Reissverschluss

3.3.3. Border

Das Workcenter Border bereitet die Border mit Hilfe der Reissverschlussliste zu. Border sind die Seitenränder eines Matratzenüberzugs.

Abbildung 3.5.: Workcenter - Border

3.3.4. Steppen

In der Stepperei wird der Stoff für den Überzug produziert.

Abbildung 3.6.: Workcenter - Steppen

Die Stepperei arbeitet mit der Gesamtliste.

3.3.5. Zuschneiden

Im Workcenter Zuschnitt werden die Stoffe für die Näherei zugeschnitten. Dafür werden Rollen mit einer entsprechenden Vorrichtung geholt. Von der Rolle werden dann, entsprechend des jeweiligen Auftrags, Zuschnitte gemacht. Da es aufwendig ist, diese Rollen zu wechseln, werden die benötigten Stoffe von den Listen nach Bauchgefühl des Mitarbeiters zusammengefasst und zugeschnitten. Dadurch wird nicht in der Reihenfolge der Listen gearbeitet, weswegen ab diesem Zeitpunkt die Wagen mit Stoffen unsortiert sind.

Im Zuschnitt wird mit der Gesamtliste gearbeitet.

Abbildung 3.7.: Workcenter - Zuschneiden

3.3.6. Nähen

Im Workcenter Nähen wird der Reissverschluss, der Border sowie der Stoff zum fertigen Überzug zusammen genäht. Die Näherinnen und Näher arbeiten mit keiner Liste, sondern nur mit Hilfe der Etiketten, die auf dem Stoff angebracht sind.

Abbildung 3.8.: Workcenter - Nähen

3.3.7. Sortierung

Die fertigen Überzüge aus der Näherei müssen sortiert werden damit die Reihenfolge der Überzüge mit den Kernen auf den Paletten übereinstimmen. Diese Aufgabe ist nicht fix einem Mitarbeiter zugeteilt.

3.3.8. Etiketten

Die Etiketten werden mit dem Pago System erstellt. Weiter müssen die Etiketten an die Matratzen geklebt werden, damit das Einzug Workcenter die Fertigstellung der Matratze ins Movex melden kann. Zudem entnimmt die Spedition den Etiketten, wohin die Matratze geliefert werden soll.

3.3.9. Einzug

Das Workcenter Einzug bezieht den Kern mit dem Überzug aus der Näherei. Aufgrund von Platzmangel können nicht alle Kerne beim Workcenter Einzug deponiert werden. So muss jeweils das Lager informiert werden, sobald weitere Kerne benötigt werden. Die Beauftragung geschieht über ein Access Formular, indem der Arbeiter die Palettnummer einträgt. Die Matratze wird mit Hilfe eines Barcodescanners in das ERP System zurückgemeldet.
Das Workcenter Einzug arbeitet mit der Einzugsliste.

Abbildung 3.9.: Workcenter - Einzug

3.4. Listen für die Produktion

Aus den, aus dem ERP System Movex importierten Aufträgen, werden die Listen für die Workcenter generiert. Dabei sind alle Listen gleich aufgebaut, enthalten aber je nach Workcenter unterschiedliche Informationen.

Alle Listen beinhalten folgende Informationen in der Kopfzeile:

Bezeichnung	Beispielwert	Beschreibung
Name	2 RV HR	Der Name der Liste. Dies ist wichtig, da sich eine Liste auf mehrere Seiten verteilen kann
Pl. Nr.	20150224002, 3V0	Planungsnummer für eine Produktion. Standardmässig gibt es eine pro Tag, bei einem Lagerauftrag kann es aber auch mehrere geben.

Tabelle 3.1.: Informationen in der Kopfzeile

Auf jeder Liste sind die Produktinformationen zu den Matratzen, sowie die Zuordnung zu den Paletten vorhanden. Diese Werte sind wie folgt aufgebaut:

Bezeichnung	Beispielwert	Beschreibung
Pal	1	Palettnummer, welche jeden Tag neu vergeben wird.
Nr	10	Position des Kerns im Wagen. Beim Einzug ist diese jedoch gespiegelt.
Art. Nr	H1121.12.6524	Die Artikelnummer des Auftrages
Name	ClimaLuxe med. 90x190 BC ClimaLuxe	Name des fertigen Produkts inklusive der Masse
Auftr. Nr	1700566125	Die Auftragsnummer
Enddat.	24.02.15	Das Enddatum, wann der Auftrag abgeschlossen sein muss
Linie	HA	Die Linie zu welcher die Matratze gehört

Tabelle 3.2.: Gemeinsame Informationen der Listen

Aufträge mit Spezialmassen werden auf der Liste jeweils farbig gekennzeichnet. Der Mitarbeiter aus dem Workcenter Lager weiss somit, dass der Kern zusätzlich zugeschnitten werden muss.

Position

Jede Position der Liste gehört zu einem Auftrag. Dabei wird zwischen Auftrag und Lagerauftrag unterschieden. Ein Auftrag kann aus mehreren Positionen bestehen. Lageraufträge kommen ausschliesslich von Grossabnehmern. Dabei werden Artikelnummer und Auftragsnummer zusammengefasst, sodass beispielsweise bei der Schneiderei möglichst viele Überzüge auf einmal zugeschnitten werden können. Zudem gibt es beim Lagerauftrag einen längeren Vorlauf das heisst es wird nicht nur für den Folgetag gerüstet.

Palett-Zuordnung

Jede Zeile der Liste ist einem Palett zugeordnet. Die Zuordnung geschieht anhand der Kerne. Es wird auf folgende Faktoren geachtet:

- Gleiche Kerne sollen möglichst zusammen auf einem Palett sein.
- Grössere Kerne müssen weiter unten liegen.
- Härtegrad des Kerns berücksichtigen (härtere weiter unten).
- Die Höhe des Wagens darf festgelegte Höhe nicht überschreiten (Standard 1850mm, kann aber individuell für Kernfamilie festgelegt werden.

3.4.1. Kernliste

Die Kernlisten werden im Workcenter Lager und Einzug benötigt.

Kernliste		Hüllen Kerne Baby HV		Pl.Nr	20150302002		3V0	Gabi
Pal	Nr	Art. Nr	Name	Auftr. Nr	Enddat.	PANR	Kern Nr.	Kern
1	10	HA142657	SJH Trevira hellblau 80x200	1700556080	02.03.15	3197227	HS20515.21	Kern Dreizone 35/40/35 Eco Top
1	11	HA142657	SJH Trevira hellblau 80x200	1700556080	02.03.15	3197227	HS20515.21	Kern Dreizone 35/40/35 Eco Top
1	12	HA142657	SJH Trevira hellblau 80x200	1700556080	02.03.15	3197227	HS20515.21	Kern Dreizone 35/40/35 Eco Top
1	13	HA142657	SJH Trevira hellblau 80x200	1700556080	02.03.15	3197227	HS20515.21	Kern Dreizone 35/40/35 Eco Top
1	14	HA142657	SJH Trevira hellblau 80x200	1700556080	02.03.15	3197227	HS20515.21	Kern Dreizone 35/40/35 Eco Top
1	15	HA142657	SJH Trevira hellblau 80x200	1700556080	02.03.15	3197227	HS20515.21	Kern Dreizone 35/40/35 Eco Top
1	16	HA142657	SJH Trevira hellblau 80x200	1700556080	02.03.15	3197227	HS20515.21	Kern Dreizone 35/40/35 Eco Top
1	17	HA142657	SJH Trevira hellblau 80x200	1700556080	02.03.15	3197227	HS20515.21	Kern Dreizone 35/40/35 Eco Top
1	18	HA142657	SJH Trevira hellblau 80x200	1700556080	02.03.15	3197227	HS20515.21	Kern Dreizone 35/40/35 Eco Top
1	19	HA142657	SJH Trevira hellblau 80x200	1700556080	02.03.15	3197227	HS20515.21	Kern Dreizone 35/40/35 Eco Top
2	20	HA142657	SJH Trevira hellblau 80x200	1700556080	02.03.15	3197227	HS20515.21	Kern Dreizone 35/40/35 Eco Top
2	21	HA142657	SJH Trevira hellblau 80x200	1700556080	02.03.15	3197227	HS20515.21	Kern Dreizone 35/40/35 Eco Top
2	22	HA142657	SJH Trevira hellblau 80x200	1700556080	02.03.15	3197227	HS20515.21	Kern Dreizone 35/40/35 Eco Top
2	23	HA142657	SJH Trevira hellblau 80x200	1700556080	02.03.15	3197227	HS20515.21	Kern Dreizone 35/40/35 Eco Top
2	24	HA142657	SJH Trevira hellblau 80x200	1700556080	02.03.15	3197227	HS20515.21	Kern Dreizone 35/40/35 Eco Top
2	25	HC1100.27.6036	Curem Heaven 19 160x200	1700565834	02.03.15	3197237	HS20212.27	Kern Curem Heaven 19
2	26	HA149430	SJH Trevira hellblau 140x200	1700556080	02.03.15	3197233	HS20515.26	Kern Dreizone 35/40/35 Eco Top
2	27	HA149430	SJH Trevira hellblau 140x200	1700556080	02.03.15	3197233	HS20515.26	Kern Dreizone 35/40/35 Eco Top
2	28	HA149430	SJH Trevira hellblau 140x200	1700556080	02.03.15	3197233	HS20515.26	Kern Dreizone 35/40/35 Eco Top
2	29	HA149430	SJH Trevira hellblau 140x200	1700556080	02.03.15	3197233	HS20515.26	Kern Dreizone 35/40/35 Eco Top
3	30	HC1103.12.6036	>> Curem Heaven Forte 22 90x190	1700567604	02.03.15	3197238	HS20215.22	Kern Curem Heaven Forte 22

Abbildung 3.10.: Beispiel einer Kernliste

Folgende Informationen werden nur auf den Kernlisten dargestellt:

Bezeichnung	Beispielwert	Beschreibung
PANR	3179884	Produktionsauftragsnummer welche für die Rückmeldung ins Movex verwendet wird um einen Auftrag abzuschliessen.
Kern Nr.	HS20027.12	Die Nummer des Matratzenkern, der zum Auftrag gehört
Kern	Kern ClimaLuxe med. 89x190x18	Name des Matratzenkern inklusive der Masse

Tabelle 3.3.: Informationen der Kernlisten

3.4.2. Einzugliste

Die Einzugliste wird im Einzug benötigt. Sie ist identisch mit der Kernliste (siehe 3.4.1), nur dass die Spalte PANR nicht aufgelistet ist.

3.4.3. Reissverschluss-Liste

Die Reissverschluss-Listen werden im Workcenter Nähen benötigt. Nach ihr werden die Reissverschlüsse für die Überzüge zusammengestellt.

Folgende Informationen werden nur auf den Reissverschluss-Listen dargestellt:

Bezeichnung	**Beispielwert**	**Beschreibung**
RV Nr.	HS31053	Die Artikelnummer des Reissverschlusses
RV-Bezeichnung	RV S60 tb.m. 90x200cm r.weiss	Die genaue Bezeichnung des Reissverschlusses

Tabelle 3.4.: Informationen der Reissverschluss-Listen

3.4.4. Etikettenliste

Die Etikettenliste wird vom Workcenter Etiketten verwendet um die Etiketten zu drucken.

3.4.5. Textliste

Die Textliste wird nur an bestimmte Personen im Workcenter Nähen verteilt. Sie beinhaltet zusätzlich die Spalte Sondertext, welche die Näherin über bestimmte Anpassungen informiert.
Für welche Person die Liste ausgedruckt wird ist im VBA-Makro hinterlegt, welches die Listen generiert.

Teil V.

Anforderungsspezifikation

4. Anforderungsspezifikation

In der Anforderungsanalyse werden die Anforderungen an das neue System beschrieben.

4.1. Kontextdiagramm

Folgendes Diagramm zeigt einen abstrakten Aufbau der Architektur:

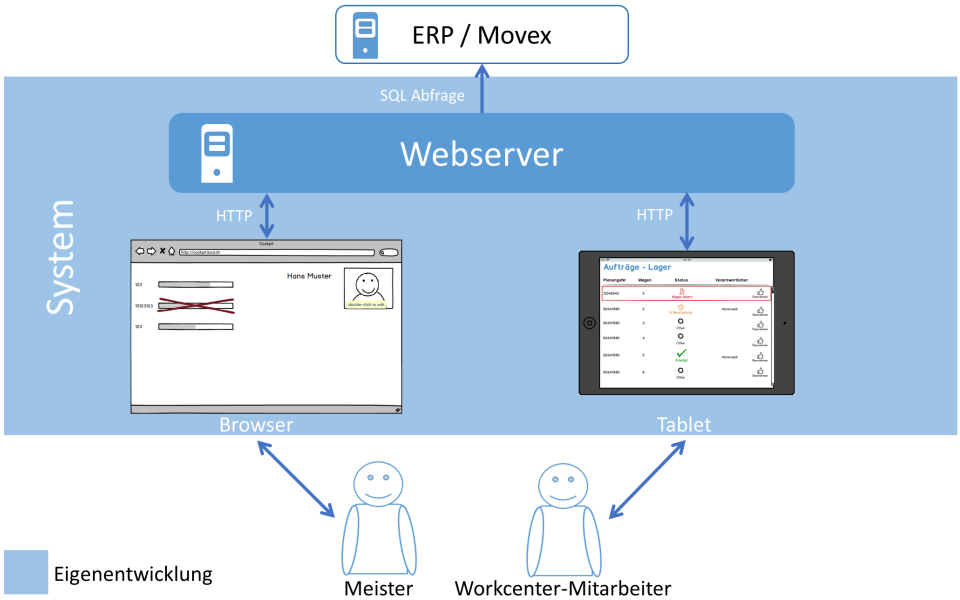


Abbildung 4.1.: Einfache Architektur

Auf dem Webserver läuft eine Server-Applikation, mit der man Produktionsdaten aus dem ERP System Movex importieren kann. Vorerst werden von der Applikation keine Daten zurück in Movex geschrieben.

Angesprochen wird die Server-Applikation mit dem Cockpit, welches eine Web-Applikation ist, welche in einem Browser läuft, oder den Clients, bei denen es sich um Android Applikation handelt, die auf einem Tablet installiert sind.

4.2. Use Case Diagramm

Folgende Use Cases wurden mit dem Auftraggeber ausgearbeitet:

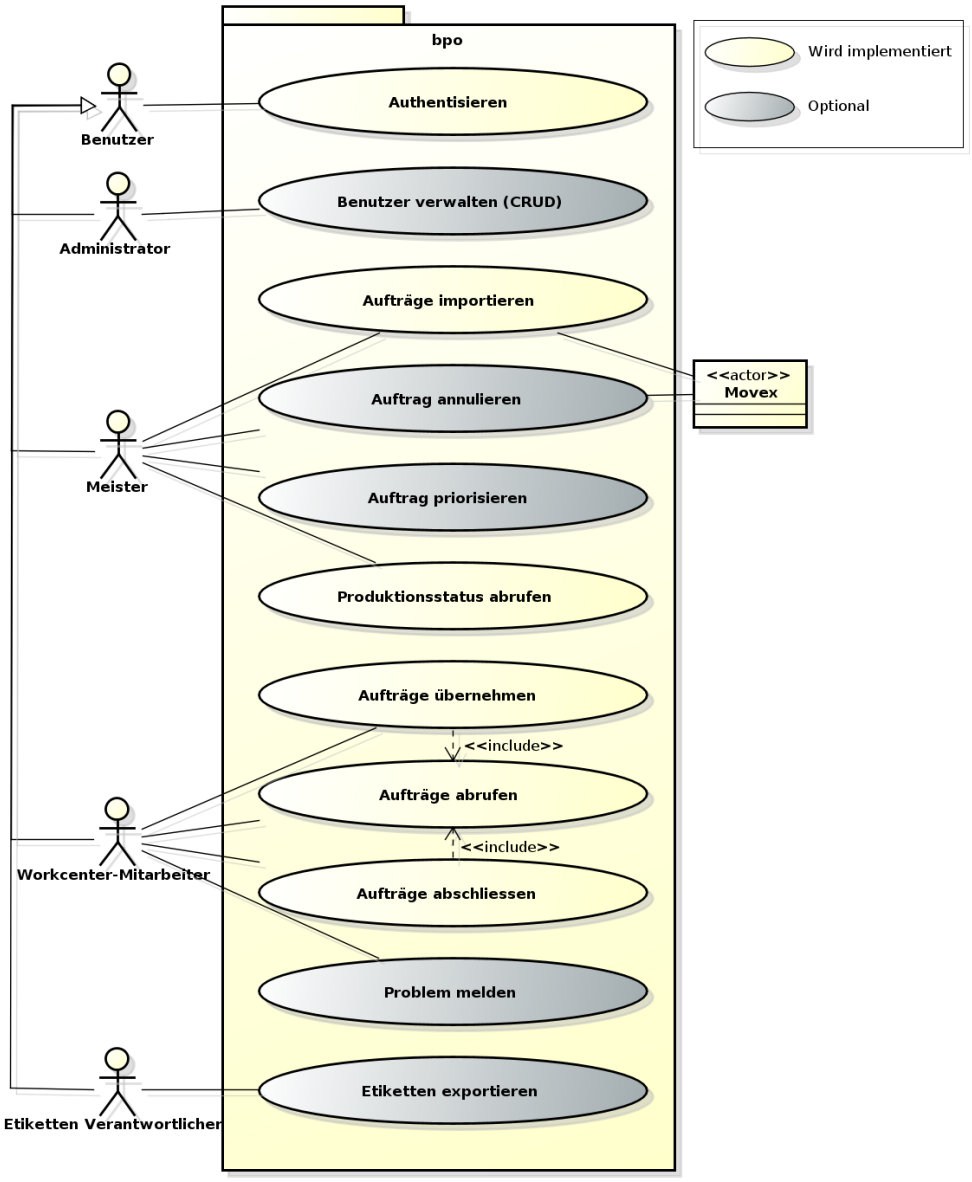


Abbildung 4.2.: Use Case Diagramm

Grau gefärbte Use Cases werden optional implementiert.

Alle Aktoren sind Benutzer des Systems und erben deshalb vom **Aktor Benutzer**. Der **Aktor Meister** ist der Produktionsverantwortliche und verwaltet deswegen die Aufträge. Der **Aktor Workcenter-Mitarbeiter** fasst die Mitarbeiter aus den unterschiedlichen Produktionsbereichen zusammen, da diese das System alle in gleicher Weise verwenden. Der **Aktor Etiketten-Verantwortlicher** ist lediglich für das Erstellen der Etiketten verantwortlich. Beim Systemaktor **Movex** handelt es sich um ein ERP System, welches unter anderem die Daten für die Produktion enthält.

4.3. Use Cases Priorisierung

Folgende Prioritäten der Use Cases wurden mit dem Auftraggeber ausgearbeitet:

Use Case	**Umsetzung**	**Priorität**
Authentisieren	Geplant	1
Aufträge importieren	Geplant	1
Produktionsstatus abrufen	Geplant	1
Aufträge abrufen	Geplant	1
Aufträge übernehmen	Geplant	1
Aufträge abschliessen	Geplant	1
Auftrag priorisieren	Optional	2
Problem melden	Optional	2
Auftrag annullieren	Optional	2
Benutzer verwalten (CRUD)	Optional	3
Etiketten exportieren	Optional	3

Legende:

1 – Muss im Rahmen dieser Arbeit umgesetzt werden.

2 – Optionale Use Cases, die nach allen Prio 1 Use Cases umgesetzt, werden.

3 – Optionale Use Cases, die nach allen Prio 2 Use Cases umgesetzt, werden.

Tabelle 4.1.: Priorisierung der Use Cases

4.4. Use Cases Brief

Nachfolgend eine kurze Beschreibung zu jedem Use Case.

4.4.1. Authentisieren

Jeder Benutzer des Systems kann sich mit seinen Benutzerdaten authentisieren.

4.4.2. Aufträge importieren

Der Meister kann Aufträge aus dem ERP System importieren.

4.4.3. Produktionsstatus abrufen

Der Meister kann den aktuellen Produktionsstatus abrufen, worin er erkennt ob die Tagesproduktion planmässig verläuft oder im Verzug ist.

4.4.4. Aufträge abrufen

Jeder Workcenter-Mitarbeiter kann die Aufträge auflisten, die er abarbeiten soll. Dabei werden je nach Workcenter andere Werte dargestellt.

4.4.5. Aufträge übernehmen

Jeder Mitarbeiter kann einen Auftrag oder eine Selektion von Aufträgen übernehmen. Damit ist ersichtlich woran der Mitarbeiter momentan arbeitet und ein Auftrag wird nicht versehentlich doppelt ausgeführt.

4.4.6. Aufträge abschliessen

Jeder Mitarbeiter kann einen übernommenen Auftrag oder eine Selektion von Aufträgen abschliessen.

4.5. Use Cases Fully Dressed

Nachfolgend werden die Use Cases mit Priorität 1 und 2 genauer beschrieben, wobei auch auf Spezialfälle eingegangen wird. Use Cases mit der Priorität 3 befinden sich im Anhang.

4.5.1. Authentisieren

Primary Actor: Benutzer

Preconditions:

- Benutzer im System vorhanden
- Benutzer nicht authentisiert

Main Success Scenario:

1. (Benutzer öffnet System)
2. System leitet Benutzer an Login-Formular weiter.
3. Benutzer gibt Benutzername und Passwort in System ein.
4. System authentisiert Benutzer anhand von Benutzername und Passwort.
5. System leitet Benutzer an Startseite weiter.
6. (Authentisieren abgeschlossen)

Extensions (or Alternative Flows)

2a. Benutzer bereits authentisiert.

1. Weiter bei 5.

5a. Benutzer gibt falsches Passwort oder falschen Benutzernamen ein

1. System meldet Benutzer Authentisierungsfehler.
2. Weiter bei 2.

5b. Benutzer möchte sich vom System abmelden.

1. Benutzer drückt auf Knopf 'Abmelden'.
2. System beendet Benutzer Session.
3. System leitet Benutzer an Login-Formular weiter
4. Weiter bei 6.

5c. Automatisches Logout nach Timeout.

1. System beendet Benutzer Session.
2. System leitet Benutzer an Login-Formular weiter.
3. Weiter bei 6.

4.5.2. Aufträge importieren

Primary Actor: Meister

Main Success Scenario:

1. (Meister öffnet Import-Dialog).
2. Meister gibt Planungsnummer in System ein, für welche die Daten importiert werden sollen.
3. Meister führt 'Auftrag laden' aus.
4. System lädt Daten aus Movex.
5. System ordnet die Auftragspositionen an Wagen zu.
6. System stellt folgende Informationen dar:
    - Anzahl Wagen, Aufträge und Position
    - Wagen mit den jeweiligen Auftragspositionen
    - Auftragspositionen mit Artikelnr. Artikelname, Auftragsnr und Enddatum
7. Meister führt 'Aufträge importieren' aus.
8. System speichert Aufträge im System.
9. (Aufträge importieren abgeschlossen)

4.5.2.1. Extensions (or Alternative Flows)

2a Planungsnummer bereits importiert

1. System meldet Meister, dass Daten schon importiert wurden.
2. Weiter bei 9.

2b Planungsnummer existiert nicht

1. System meldet Meister, dass die Planungsnummer nicht existiert.
2. Weiter bei 2.

4.5.3. Produktionsstatus abrufen

Primary Actor: Meister

Main Success Scenario:

1. (Meister öffnet Produktionsstatus)
2. System zeigt Meister folgende Informationen an:
    a) Planungsnummer
    b) Enddatum
    c) Wagen
    d) Status
        i. Abgeschlossen
        ii. In Bearbeitung
        iii. Problem aufgetreten
3. (Produktionsstatus abrufen abgeschlossen)

4.5.3.1. Extensions (or Alternative Flows)

2a Meister öffnet spezifische Planungsnummer

1. System stellt folgende Details für die gewählte Planungsnummer dar:
    a) Wagennummer
    b) Status der einzelnen Workcenter:
        i. Abgeschlossen
        ii. In Bearbeitung
        iii. Problem aufgetreten
2. Weiter bei 3

4.5.4. Aufträge abrufen

Primary Actor: Workcenter-Mitarbeiter

Main Success Scenario:

1. (Workcenter-Mitarbeiter öffnet Dialog um Aufträge einzusehen)
2. System stellt alle Wagen mit folgenden Informationen dar:
    a) Planungsnummer
    b) Wagennummer
    c) Status
        i. Abgeschlossen
        ii. In Bearbeitung
        iii. Problem aufgetreten
    d) Verantwortlicher
3. (Aufträge abrufen abgeschlossen)

4.5.5. Aufträge übernehmen

Primary Actor: Workcenter-Mitarbeiter

Preconditions: Use Case 'Aufträge abrufen' abgeschlossen

Main Success Scenario:

1. (Workcenter-Mitarbeiter öffnet Dialog um Aufträge einzusehen)
2. Workcenter-Mitarbeiter übernimmt Auftrag.
3. System aktualisiert Status des Auftrags.
4. System stellt Detailinformationen des Auftrags dar.
5. (Aufträge übernehmen abgeschlossen)

4.5.5.1. Extensions (or Alternative Flows)

3a Auftrag wurde gleichzeitig von anderem Mitarbeiter übernommen

1. System informiert Mitarbeiter, dass Wagen bereits übernommen wurde.
2. Weiter bei 1.

4.5.6. Aufträge abschliessen

Primary Actor: Workcenter-Mitarbeiter

Preconditions: Use Case 'Aufträge abschliessen' abgeschlossen

Main Success Scenario:

1. (Workcenter-Mitarbeiter hat Auftrag übernommen)
2. Workcenter-Mitarbeiter schliesst Positionen des Auftrags ab.
3. Wenn die letzte Position abgeschlossen wurde, aktualisiert das System den Status des Auftrags.
4. (Aufträge abschliessen abgeschlossen)

4.5.7. Auftrag annullieren

Primary Actor: Meister

Main Success Scenario:

1. (Meister öffnet Ansicht für Annullierung).
2. Meister teilt System mit, für welche Auftragsnummer Annullierungen vorgenommen werden soll.
3. System zeigt folgende Informationen von Aufträgen an:
    a) Anzahl Positionen
    b) Positionen des Auftrags
4. Meister annulliert den Auftrag.
5. System aktualisiert Informationen für Workcenter-Mitarbeiter.
6. System zeigt Erfolgsmeldung an.
7. (Aufträge annullieren abgeschlossen)

4.5.7.1. Extensions (or Alternative Flows)

4a Einzelne Positionen annullieren.

1. Meister wählt einzelne Positionen aus.
2. Meister annulliert die Positionen.
3. Weiter bei 5.

4b Auftrag/Position wird gerade von Workcenter-Mitarbeiter bearbeitet.

1. System gibt eine Benachrichtigung an den Workcenter-Mitarbeiter
2. Weiter bei 5.

4.5.8. Aufträge priorisieren

Primary Actor: Meister

Main Success Scenario:

1. (Meister öffnet Ansicht für den Auftragsstatus)
2. Meister selektiert einen oder mehrere Aufträge.
3. Meister betätigt 'Auftrag priorisieren'.
4. System aktualisiert die Priorität für den Auftrag.
5. (Aufträge priorisieren abgeschlossen)

4.5.9. Problem melden

Primary Actor: Workcenter-Mitarbeiter

Main Success Scenario:

1. (Workcenter-Mitarbeiter öffnet Dialog um Aufträge einzusehen)
2. Workcenter-Mitarbeiter betätigt Button für Problemmeldung.
3. System meldet an Meister Problem mit folgenden Informationen:
    a) Name des Workcenter
    b) Vor- und Nachname des Mitarbeiters
    c) Art des Problems
4. (Problem melden abgeschlossen)

4.6. Optionale Use Cases

4.6.1. Auftrag annullieren

Der Meister kann einen Auftrag oder Position annullieren, woraufhin dieser nicht mehr in den Produktionslisten angezeigt wird.

4.6.2. Auftrag priorisieren

Der Meister kann Aufträge priorisieren, damit diese schneller abgearbeitet werden.

4.6.3. Problem melden

Hat ein Mitarbeiter ein Problem bei seinem aktuellen Arbeitsschritt (beispielsweise Material nicht vorhanden, Maschine defekt) kann er dies über das System melden, woraufhin der Meister benachrichtigt wird.

4.6.4. Benutzer verwalten (CRUD)

Der Administrator kann Benutzer erstellen, auflisten, bearbeiten und löschen. Dabei wird der Benutzern einem Workcenter zugeordnet.

4.6.5. Etiketten exportieren

Der Etiketten-Verantwortliche kann die Etiketten-Informationen in einem für den Etiketten Drucker passendes Format (beispielsweise csv) erstellen.

4.7. Zukünftige Use Cases

Bei den folgenden Use Cases handelt es sich um sinnvolle Erweiterungen, die für eine zukünftige Implementation in Betracht gezogen werden sollten.

4.7.1. Rückmeldung in ERP

Wenn ein Auftrag komplett abgeschlossen wurde, soll dies im System erfasst werden können, woraufhin das System dies zurück in das ERP System schreibt. Ausgenommen sind dabei Problemfälle.

4.7.2. Wegoptimierung im Lager

Um Zeit zu sparen beim Palett vorbereiten bzw. liefern, soll der Weg der die Lager-Mitarbeiter fahren optimiert werden.

4.7.3. Auswertung der Mitarbeiter-Aktivitäten

Da die Mitarbeiter erfassen welche Aufträge sie übernehmen und wann sie diese abschliessen, können Auswertungen erstellt werden, was sie am Tag leisten.

4.7.4. Authentisierung (via LDAP)

Jeder Benutzer des System kann sich mit seinen Benutzerdaten identifizieren. Die Authentisierung wird dann von LDAP bzw. Active Directory übernommen.

4.7.5. Report erstellen

Am Ende eines Produktionstag wird ein Report generiert, welcher die im System gemachten Annullierungen und Probleme auflistet.

4.7.6. Annullierungen aus ERP

Wird eine Annullierung im ERP System vorgenommen, so wird das System automatisch darüber informiert.

4.8. Mockups

Zu jedem Use Case werden Mockups gezeichnet, bei denen es sich um einen Papier-Prototypen des User Interfaces handelt. Nachfolgend werden die Mockups aufgelistet und wenn notwendig genauer beschrieben.

4.8.1. Symbol Legende

Folgende Tabelle beschreibt die Symbole, welche in den Mockups verwendet werden.

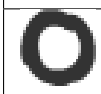
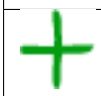
Symbol	Beschreibung
	Auftrag ist abgeschlossen.
	Auftrag ist in Bearbeitung.
	Beim Auftrag ist ein Problem aufgetreten.
	Der Auftrag ist noch zu erledigen.
	Der Auftrag wurde annulliert
	Mit einem Klick auf das Symbol, kann man den Auftrag übernehmen.
	Das gemeldete Palett muss vom Workcenter-Lager zum Workcenter-Einzug gebracht werden.
	Der Administrator kann durch Klick auf das Symbol einen neuen Benutzer erstellen.
	Der Administrator kann durch Klick auf das Symbol einen Benutzer bearbeiten.
	Der Administrator kann durch Klick auf das Symbol einen Benutzer löschen.
	Alle Workcenter haben Ihre Aufträge erledigt. Das Material ist bereit für den Einzug.

Tabelle 4.2.: Beschreibung der Symbole

4.8.2. Authentisieren

Im Cockpit authentisiert man sich mit folgendem Formular:

Abbildung 4.3.: Mockup - Authentisierung Cockpit

Nach erfolgreicher Authentisierung wird man auf das Hauptmenü weitergeleitet.

Abbildung 4.4.: Mockup - Cockpit Hauptmenü

Im Hauptmenü werden je nach Rolle verschiedene Menüpunkte angezeigt. So muss der Administrator zum Beispiel keine Aufträge importieren können.

4.8.3. Aufträge importieren

Zum Importieren der Aufträge muss der Meister eine Planungsnummer eingeben. Mit einem Klick auf den Button 'Aufträge laden' werden die Aufträge geladen und aufgelistet. Dabei sind sie bereits den jeweiligen Wagen zugeordnet. Mit einem Klick auf den Button 'Aufträge importieren' werden die Aufträge im System gespeichert und sind bereit für die Produktion.

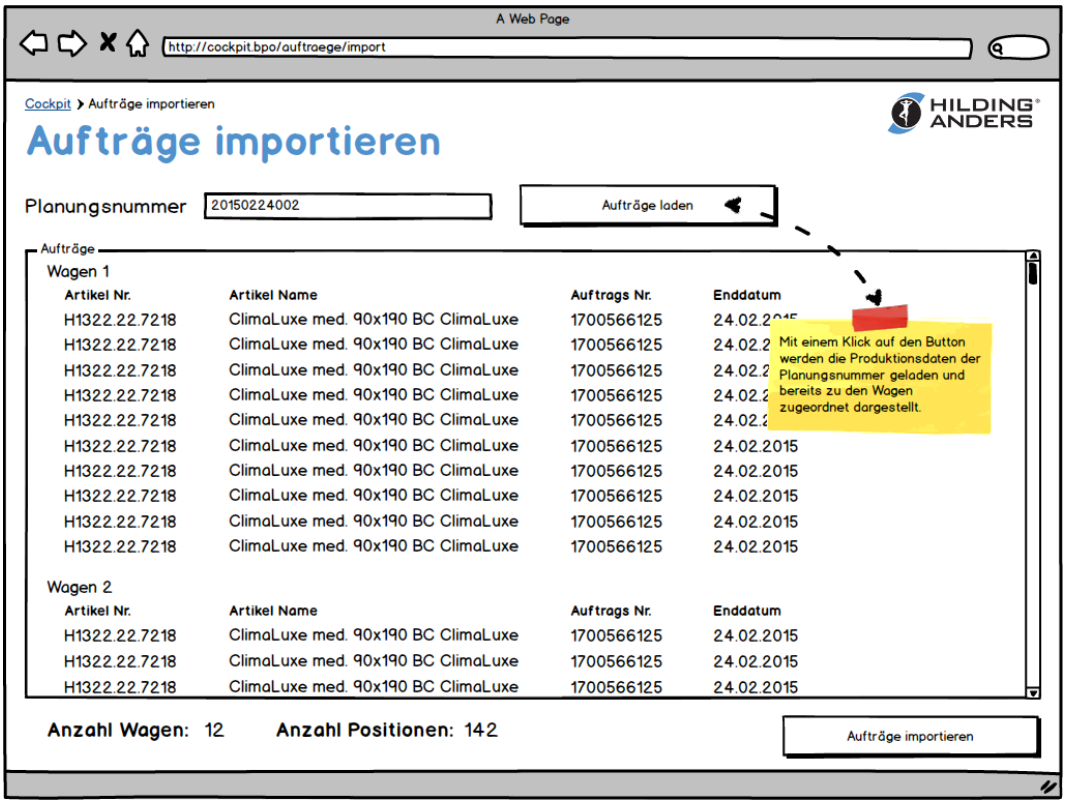


Abbildung 4.5.: Mockup - Aufträge importieren

4.8.4. Auftrag annullieren

Um einen Auftrag zu annullieren gibt man die Auftragsnummer ein, woraufhin die Auftragsinformationen geladen werden. Mit einem Klick auf den Button 'Auftrag annullieren' wird dieser Auftrag annulliert.

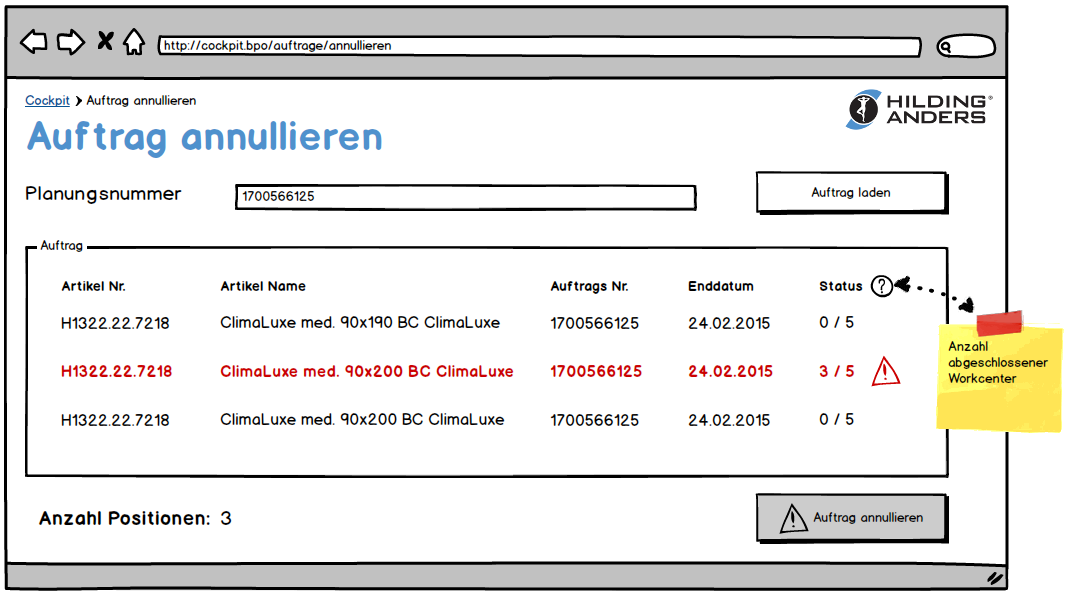


Abbildung 4.6.: Mockup - Auftrag annullieren

In der Spalte 'Status' ist ersichtlich wie viele Workcenter ihren Produktionsschritt bereits abgeschlossen haben. Wenn der Status nicht '0 / 5' entspricht, wird die Zeile rot markiert, da für den Kunden Kosten anfallen werden.

4.8.5. Produktionsstatus abrufen

Der Produktionsstatus wird in zwei Ebenen dargestellt. Auf der ersten Ebene sieht man die aktuellen Produktionen mit ihren Planungsnummern. Es ist erkennbar, wie viele Wagen bereits abgeschlossen worden sind. Der aktuelle Status (In Bearbeitung, Problem gemeldet oder Abgeschlossen) wird durch ein passendes Symbol dargestellt. Klickt man bei einer der aufgelisteten Produktionen auf den Details-Button kommt man in die zweite Ebene.

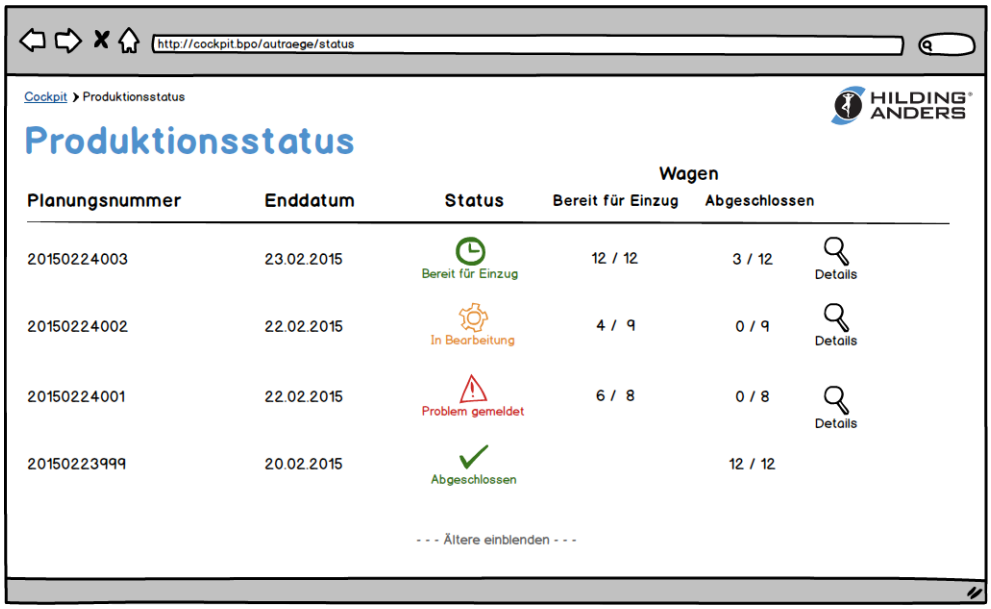


Abbildung 4.7.: Mockup - Produktionsstatus Ebene 1

In der zweiten Ebene sieht man detaillierte Informationen zur ausgewählten Produktion. Jede Reihe entspricht einem Wagen und jede Spalte einem Workcenter, wobei der Wert in der Liste den aktuellen Status (Erledigt, In Bearbeitung, Problem gemeldet oder wartend) in Form eines Icons darstellt.

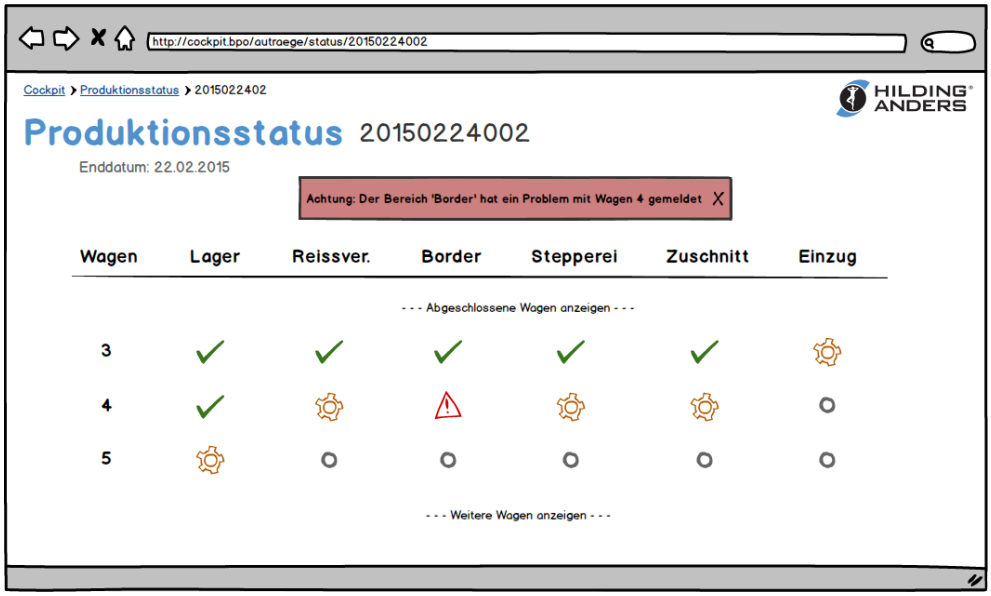


Abbildung 4.8.: Mockup - Produktionsstatus Ebene 2

4.8.6. Aufträge abrufen

Das Abrufen der Aufträge wird in zwei Ebenen dargestellt. In der ersten Ebene sind alle Wagen aufgelistet und man sieht den jeweiligen Status (Offen, In Bearbeitung, Abgeschlossen). Mit dem Button 'Details' kommt man in die zweite Ebene 'Aufträge übernehmen' oder 'Aufträge abschliessen'.

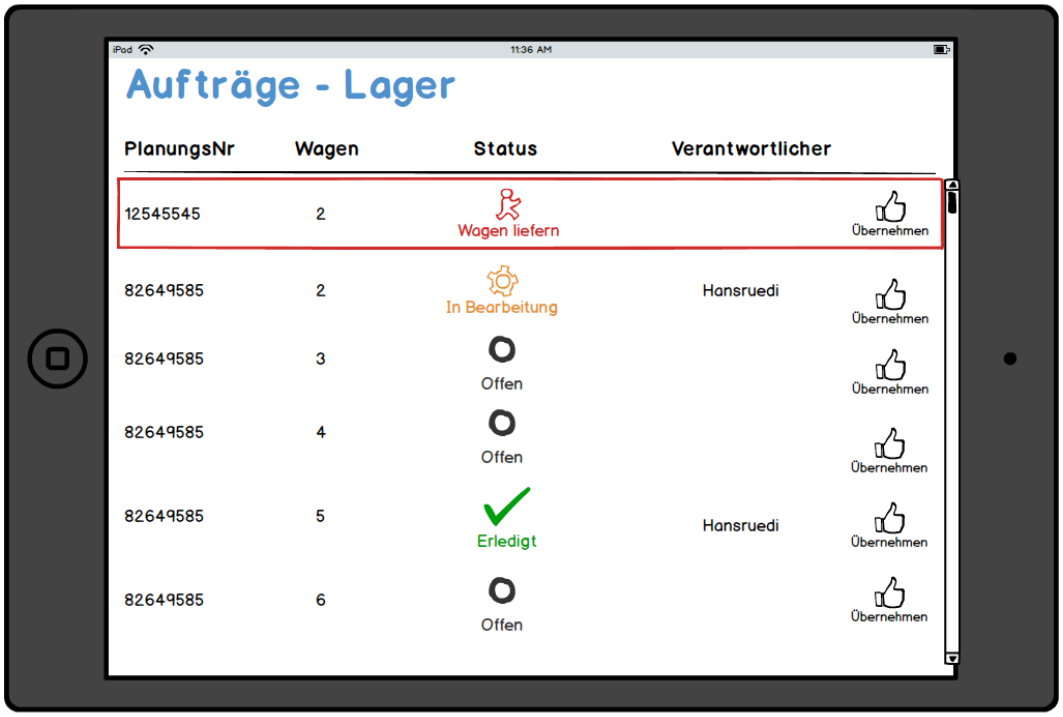


Abbildung 4.9.: Mockup - Auftrage abrufen Ebene 1

4.8.7. Aufträge übernehmen

In der zweiten Ebene sieht man alle Aufträge, die dem Wagen zugeordnet sind. Der Wagen kann in dieser Ansicht übernommen werden.

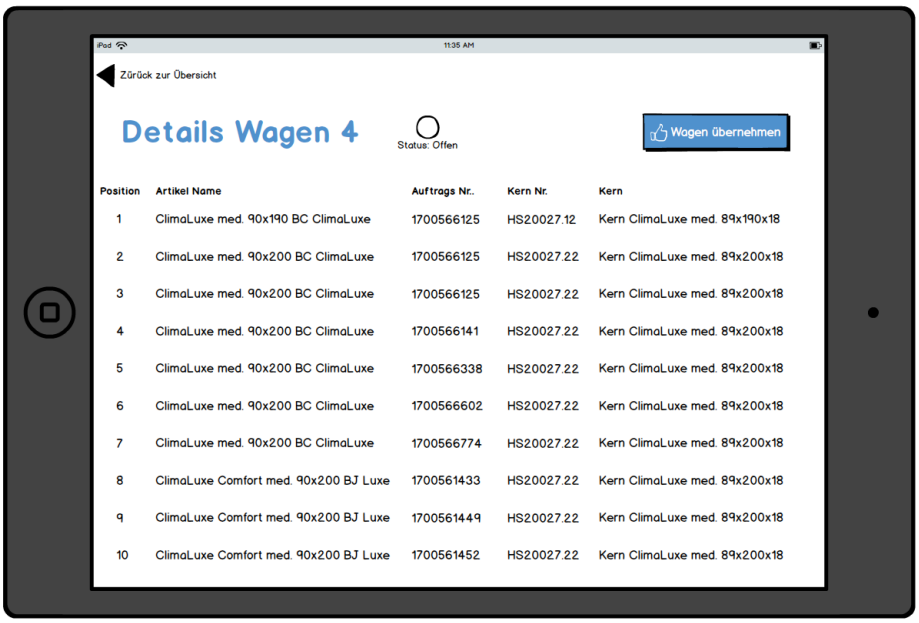


Abbildung 4.10.: Mockup - Auftrage abrufen Ebene 2 - Übernehmen

4.8.8. Aufträge abschliessen

Nachdem der Wagen übernommen wurde sieht der Workcenter-Mitarbeiter folgendes Formular, indem er die Positionen abschliessen kann.

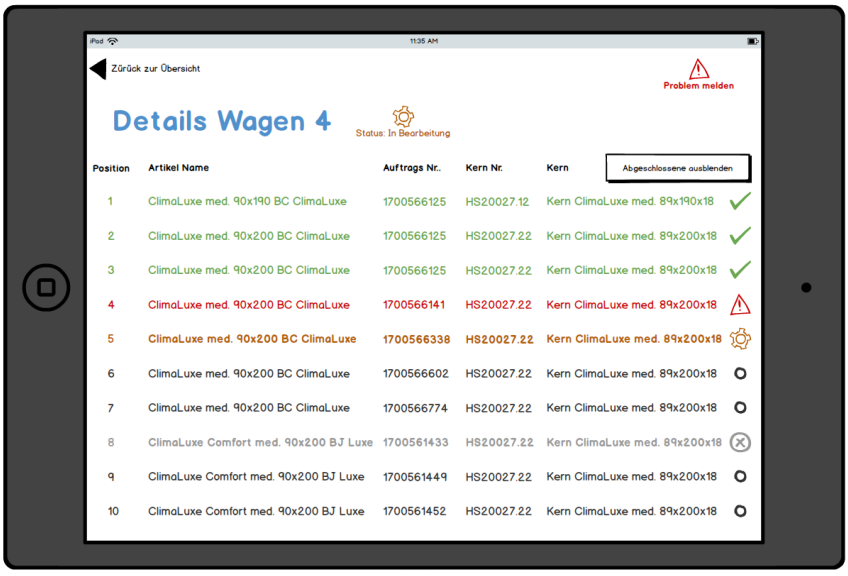


Abbildung 4.11.: Mockup - Auftrage abrufen Ebene 2 - Abschliessen

4.9. Workcenter Informationen

Nachfolgend wird spezifiziert welche Informationen von welchem Workcenter benötigt werden um ihre Arbeitsschritte auszuführen. Die Informationen, die besonders wichtig sind, sind fett markiert.

Workcenter	**Informationen**
Alle Workcenter	• Position auf Wagen • Auftrags Nummer • Matratzen Bezeichnung • Planungscode
Lager	• **Kern Nummer** • Kern Bezeichnung • Zeichen ob Schneiden / Kleben notwendig
Reissverschluss / Border	• **Reissverschluss Nummer** • Reissverschluss Bezeichnung
Steppen	• **Artikel Nummer** • Kern Bezeichnung Die Informationen müssen sich konsolidieren lassen.
Zuschneiden	• Artikel Nummer • Kern Bezeichnung Die Informationen müssen sich konsolidieren lassen.
Sortierung	• **Matratzen Bezeichnung** • Kern Nummer • Matratzen Artikel Keine Sortierung der Einträge notwendig
Einzug	• **Matratzen Bezeichnung** • Kern Nummer • Kern Bezeichnung • Matratzen Artikel

Tabelle 4.3.: Informationen je Workcenter

5. Nichtfunktionale Anforderungen

Folgende Anforderungen stützen sich auf ISO 9126.

5.1. Funktionalität

5.1.1. Richtigkeit

Es wird gewährleistet, dass immer die aktuellen Daten angezeigt werden. Ein Workcenter-Mitarbeiter oder auch der Meister muss innerhalb maximal drei Sekunden sehen, wenn eine Änderung vorgenommen wurde. So kann beispielsweise eine doppelte Übernahme eines Wagens verhindert werden. Wird trotzdem zur selben Zeit die gleiche Aktion von zwei Mitarbeiter ausgeführt, muss dies abgefangen werden und der eine Mitarbeiter darüber informiert werden.

5.1.2. Interoperabilität

Die Daten für die Produktion werden aus dem ERP System Movex geholt. Die Software muss also Daten aus Movex lesen können. Das Cockpit muss mit einem Windows Rechner bedient werden können. Die Software für die Workcenter-Mitarbeiter mit einem Tablet.

5.1.3. Sicherheit

Der Zugriff auf das System erfordert eine erfolgreiche Identifikation und Authentifikation. Der Aktor ist aufgrund seiner Funktion authorisiert seine Aufgaben zu erledigen. Das heisst, ein Workcenter-Mitarbeiter ist beispielsweise nicht berechtigt einen Benutzer im System anzulegen.

5.2. Zuverlässigkeit

5.2.1. Fehlertoleranz

Das System stürzt nicht aufgrund eines Fehlers ab. Fehler werden abgefangen und der Benutzer entsprechend benachrichtigt. Tritt ein Fehler in der Business Logik auf, so wird dieser geloggt. Exceptions werden vom Server mit einem passenden HTML Status Code an den Client zurückgemeldet.

5.3. Benutzbarkeit

5.3.1. Erlernbarkeit

Der Benutzer versteht das Konzept und die Bedienung nach einer Mitarbeiter-Schulung.

5.3.2. Bedienbarkeit

Der Workcenter-Mitarbeiter kommt innert maximal drei Schritten zum Ziel. Das User Interface soll nur die nötigen Informationen anzeigen.

5.4. Übertragbarkeit

5.4.1. Anpassbarkeit

Der Serverteil soll auf jedem Java Applikationsserver laufen können. Das Cockpit ist mit einem Browser aufrufbar. Die Software für den Workcenter-Mitarbeiter setzt ein Tablet mit einem Android-Betriebssystem voraus.

Teil VI.

Evaluation

6. Evaluation

6.1. Native- und Webapplikation

In der Entwicklung von Applikationen für mobile Geräte stellt sich oft die Frage, ob eine Webapp oder eine Native Applikation erstellt werden soll. In diesem Kapitel soll diese Frage, für die gegebene Umgebung und die erfassten Anforderungen, geklärt werden.

6.1.1. Allgemeine Gegenüberstellung

Die Erstellung einer Native- oder Webapplikation bringt jeweils seine Vor- und Nachteile. In der folgende Liste werden diese gegenübergestellt.

Technologie	**Vorteile**	**Nachteile**
Webapplikation	• Plattformunabhängig • Keine Installation nötig • Updates schnell eingespielt	• Offline Betrieb wird erschwert • Nur Teil der Geräte-Funktionalitäten verfügbar
Native Applikation	• Hardware Unterstützung – Bessere Performance – Einfacher Zugriff auf Sensoren, GPS etc. • Offline Verfügbarkeit • Bessere Benutzbarkeit	• Aufwendigere Pflege (Updates etc.) • Geräte-Betriebssystem gebunden

Tabelle 6.1.: Vor- und Nachteile von Native- und Webapplikationen

6.1.2. Anforderung an Applikationen

Nachfolgend werden die Anforderungen an die Applikation festgehalten.

6.1.2.1. Cockpit

Das Cockpit liefert dem Meister einen Überblick über die laufende Produktion. Die Daten dienen zur Information, das heisst es gibt kaum Interaktion zwischen dem Meister und der Applikation. Momentan ist lediglich der Import von Aufträgen, Aufträge annullieren sowie Aufträge priorisieren vorgesehen.

Das Cockpit muss:

- mit einem Desktop-Computer aufrufbar sein
- übersichtlich sein
- responsive sein

6.1.2.2. Applikation für Workcenter-Mitarbeiter

Der Client wird mit einem Tablet bedient. Es gibt häufige Interaktion zwischen den Workcenter-Mitarbeiter und dem System. Es müssen Aufträge angezeigt, übernommen sowie abgeschlossen werden. Zudem muss der Workcenter-Mitarbeiter Probleme melden können. Es ist es notwendig, dass die Applikation performant und effizient ist. Im Workcenter-Lager kann es zu kleineren Verbindungsunterbrüchen kommen, da der Arbeiter stets mit dem Stapler in Bewegung ist. Da zukünftig eine Wegoptimierung des Workcenter-Lager Mitarbeiter angedacht ist, ist es notwendig, dass das GPS angesteuert werden kann. Auch eine Einbindung der Kamera wäre denkbar.

- Sehr gute Performance und Antwortzeiten
- Keine Probleme bei Verbindungsunterbrüchen
- Zugriff auf hardwarenahe Funktionalitäten des Tablet
- Sehr gute Benutzbarkeit

6.1.3. Auswahl

Die folgenden Tabellen zeigen die Entscheidung mit einer Begründung dazu.

6.1.4. Cockpit

WebApp	NativeApp	Begründung
✓	✗	Anhand der Anforderungen fiel die Wahl klar auf eine Web-App. Folgende Faktoren waren dabei entscheidend: • Der Meister kann bequem, mit jedem PC, über den Browser auf die Applikation zugreifen. • Es braucht keine zusätzliche Software für den PC und somit gibt es keinen Mehraufwand für die Administration. • Die Applikation muss Responsive sein, damit mit jedem Bildschirm eine optimale Anzeige erzielt werden kann.

Tabelle 6.2.: Auswahl Native- oder Webapp für Cockpit

6.1.4.1. Applikation für Workcenter-Mitarbeiter

WebApp	NativeApp	Begründung
✗	✓	Anhand der aufgenommenen Anforderungen fiel die Wahl auf eine Native App. Folgende Faktoren waren dabei entscheidend. • Damit die Workcenter-Mitarbeiter die Applikation akzeptieren, sind eine gute Benutzbarkeit, sowie schnelle Antwortzeiten unabdingbar. • Daten können problemlos offline gehalten werden, damit es keine Probleme bei Verbindungsunterbrüchen gibt. • Die ganze Hardware steht zur Verfügung und somit können zukünftige Anforderungen wie die Wegoptimierung erfüllt werden.

Tabelle 6.3.: Auswahl Native- oder Webapp für Client

6.2. Evaluation der Hardware

In diesem Kapitel soll evaluiert werden, was für ein Tablet für die Workcenter-Mitarbeiter zum Einsatz kommen soll. Nachfolgend werden die Anforderungen spezifiziert, verschiedene Tablets evaluiert und das geeignetste Tablet festgelegt.

6.2.1. Anforderungen

6.2.1.1. Betriebssystem

Wird die Applikation als Web-Applikation umgesetzt, ist diese plattformunabhängig und es kommen ein Android oder Apple Tablet in Frage. Da allerdings eine Native Applikation umgesetzt wird, ist die Auswahl auf Android Geräte beschränkt, da die Applikation in Java geschrieben sein soll.

6.2.1.2. Leistung

Da auf dem Tablet hauptsächlich Informationen dargestellt und nur wenig Interaktion stattfindet, sind kaum Anforderungen an die Leistung vorhanden. Die minimalen Leistungen auf dem Markt sind absolut ausreichend.

6.2.1.3. Display

Die Workcenter-Mitarbeiter brauchen eine gute Übersicht über eine grosse Anzahl Informationen. Da die Informationen gut lesbar sein müssen, ist eine minimale Auflösung von 1280x800 Pixeln notwendig.

6.2.1.4. Preis

Da mehrere Workcenter-Mitarbeiter mit einem Tablet ausgerüstet werden müssen, sollte der Preis möglichst günstig sein.

6.2.1.5. Akku

Die Akkulaufzeit ist nicht relevant, da die Tablets am Strom hängen.

6.2.1.6. Zubehör

Für das Tablet sollte ein widerstandsfähiges Cover, sowie eine Halterung für die Stapler zur Verfügung stehen. Optional könnte auch ein externes Numpad gebraucht werden.

6.2.2. Tablets

Nachfolgend wird eine Auswahl von Tablets miteinander verglichen und die geeignetsten zwei ausgewählt.

	Prestigio MultiPad	**Samsung Galaxy Tab S 8.4**	**Samsung Galaxy Tab 4 10.1**	**ASUS Transformer TF103C**
				
Betriebssystem	Android 4.2	Android 4.4	Android 4.4	Android 4.4
Auflösung	1280 x 800	2560 x 1600	1280 x 800	1280 x 800
Grösse	10.1	8.4	10.1	10.1
Prozessor	Quad-core 1.6 GHz	8-core 1.90 GHz	Quad-core 1.2 GHz	Quad-core 1.33 GHz
Arbeitsspeicher	1 GB	3 GB	1.5 GB	1 GB
Zubehör	Authohalterung verfügbar und gute Cover Auswahl			
Sonstiges	Unbekannter Hersteller	Namhafter Hersteller	Namhafter Hersteller	Schlechte Bewertungen
Preis	189.-	349.-	239.-	179.-
Auswahl	✗	✓	✓	✗

Tabelle 6.4.: Evaluation der Tablets

[**?**]

6.2.2.1. Auswertung

Preislich ist das Prestigio, sowie das Transformer-Tablet sehr günstig. Allerdings empfehlen sich diese nicht aufgrund des veralteten Android Betriebssystems, sowie den schlechten Bewertungen in Online-Shops.
Es macht Sinn auf einen namhaften Hersteller wie Samsung zu setzen, da für diese Tablets auch mehr Zubehör und Software-Updates zur Verfügung stehen.
Die Wahl fällt somit auf die zwei Samsung Tablets, da damit unter anderem zwei verschiedene Displaygrössen (8.4 und 10.1) abgedeckt sind, womit diese für den Gebrauch evaluiert werden können.

Teil VII.

Domainanalyse

Domainanalyse

Aus der Anforderungsspezifikation ergibt sich das folgende Domain Model:

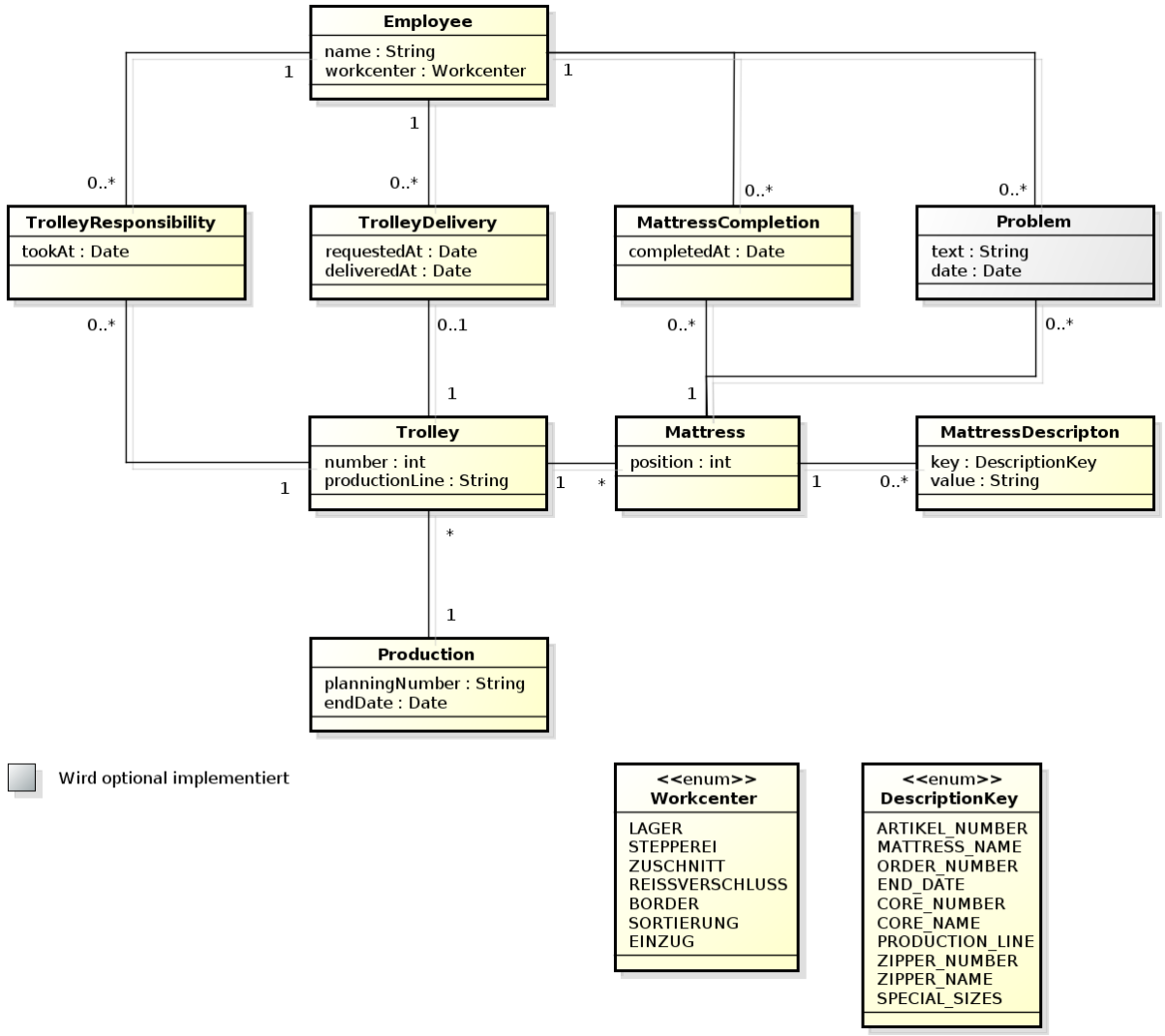


Abbildung 6.1.: Domain Model

Nachfolgend werden alle Klassen des Domain Models beschrieben. Es werden nur Attribute und Beziehungen aufgeführt, die einer Beschreibung bedürfen. Selbstverständliche werden nicht behandelt. Ausserdem wird jeweils nur ein Ende einer Beziehung beschrieben.

6.2.3. Klasse Production

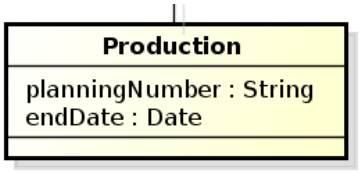


Abbildung 6.2.: Klasse Production

Die Klasse Production entspricht einer Produktion mit einer Planungsnummer. Im Normalfall gibt es pro Tag eine neue Instanz der Klasse. Diese wird beim Import der Aufträge erstellt.

Attribut	Type	Beschreibung
planningNumber	integer	Eindeutige Identifier für die Production. Die planningNumber entspricht der Planungsnummer aus Movex.
endDate	Date	Datum bis wann die Produktion abgeschlossen sein muss.

Tabelle 6.5.: Attribute der Klasse Production

6.2.4. Klasse Employee

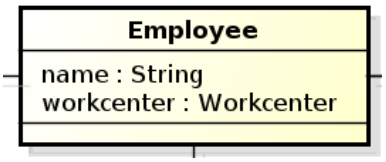


Abbildung 6.3.: Klasse Employee

Die Klasse entspricht einem Workcenter-Mitarbeiter

Assoziation	Beschreibung
TrolleyResponsibility	Wird gesetzt wenn eine Verantwortung für ein Trolley übernommen wird.
TrolleyDelivery	Wird beim Employee gesetzt wenn ein Mitarbeiter aus dem Einzug einen Trolley bestellt und wenn einer ankommt.
MattressCompletion	Instanz wird erstellt, wenn der Mitarbeiter eine Matratze abschliesst.

Tabelle 6.6.: Assoziationen der Klasse Employee

6.2.5. Klasse Trolley

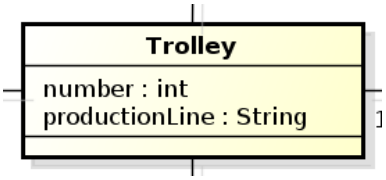


Abbildung 6.4.: Klasse Trolley

Die Klasse Trolley entspricht einem Wagen.

Attribut	Type	Beschreibung
number	integer	Eine pro Produktionslinie eindeutige Nummer des Wagens.
productionLine	String	Die Produktionslinie in der der Wagen verarbeitet wird.

Tabelle 6.7.: Attribute der Klasse Trolley

Assoziation	Beschreibung
Production	Jeder Trolley muss einer Production zugeordnet sein.

Tabelle 6.8.: Assoziationen der Klasse Trolley

6.2.6. Klasse Mattress

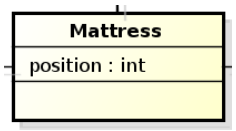


Abbildung 6.5.: Klasse Mattress

Die Klasse Mattress entspricht einer Matratze, welche produziert wird.

Attribut	Type	Beschreibung
position	integer	Position der Matratze auf dem Wagen.

Tabelle 6.9.: Attribute der Klasse Mattress

Assoziation	Beschreibung
MattressDescription	Die Werte der Matratze werden mit der MattressDescription Klasse abgebildet. Eine Matratze kann mehrere davon haben.

Tabelle 6.10.: Assoziationen der Klasse Mattress

6.2.7. Klasse TrolleyResponsibility

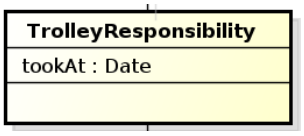


Abbildung 6.6.: Klasse TrolleyResponsibility

Wenn die Verantwortlichkeit eines Wagens übernommen wird, wird eine Instanz dieser Klasse erstellt.

Assoziation	Beschreibung
Employee	Der Mitarbeiter, der die Verantwortlichkeit übernommen hat.
Trolley	Wagen, für den die Verantwortlichkeit übernommen wurde.

Tabelle 6.11.: Assoziationen der Klasse TrolleyResponsibility

6.2.8. Klasse TrolleyDelivery

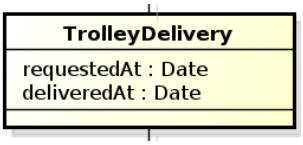


Abbildung 6.7.: Klasse TrolleyDelivery

Es wird eine Instanz der Klasse TrolleyDelivery erstellt, sobald ein Wagen bereit für den Einzug ist.

Assoziation	Beschreibung
Employee	Mitarbeiter der die Verantwortlichkeit für die Auslieferung des Wagens übernommen hat.
Trolley	Wagen, welcher bereit für den Einzug ist.

Tabelle 6.12.: Assoziationen der Klasse TrolleyDelivery

6.2.9. Klasse MattressCompletion

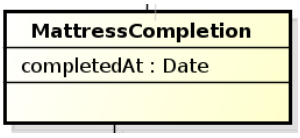


Abbildung 6.8.: Klasse MattressCompletion

Hat ein Workcenter seinen Prozess für eine Matratze abgeschlossen, wird eine Instanz der Klasse MattressCompletion erstellt.

Teil VIII.

Architektur

7. Architektur

7.1. Systemübersicht

Folgendes Diagramm zeigt eine Übersicht und den Kontext des Systems.

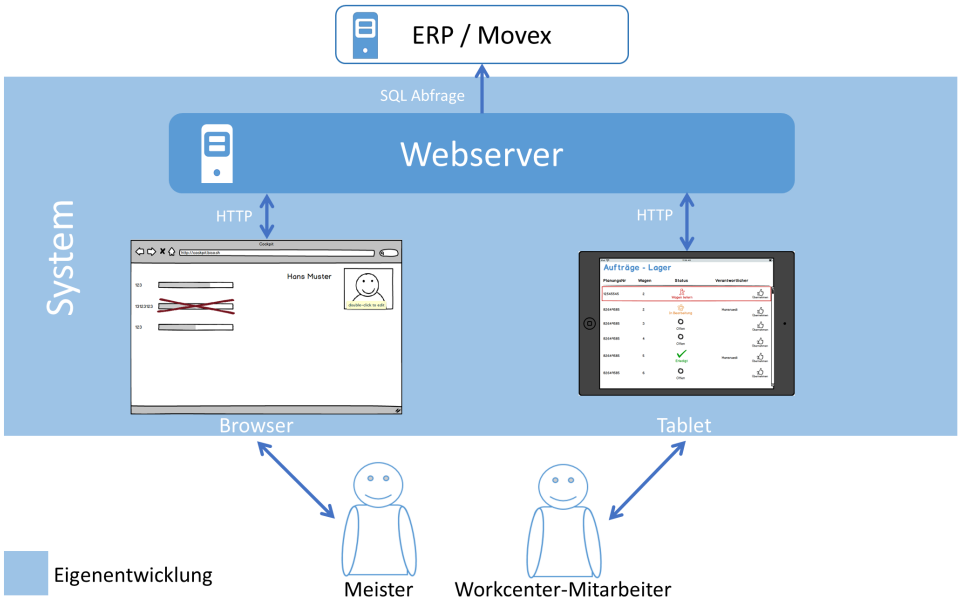


Abbildung 7.1.: Kontext und Systemübersicht

Der Meister kann über den Browser die Webapplikation verwenden, über die die Aufträge aus dem ERP System Movex importiert. Die importieren Aufträge kann der Workcenter-Mitarbeiter über ein Tablet abrufen und abarbeiten.

7.2. Deployment View

Folgendes Deployment Diagramm zeigt eine Grobübersicht über die Komponenten:

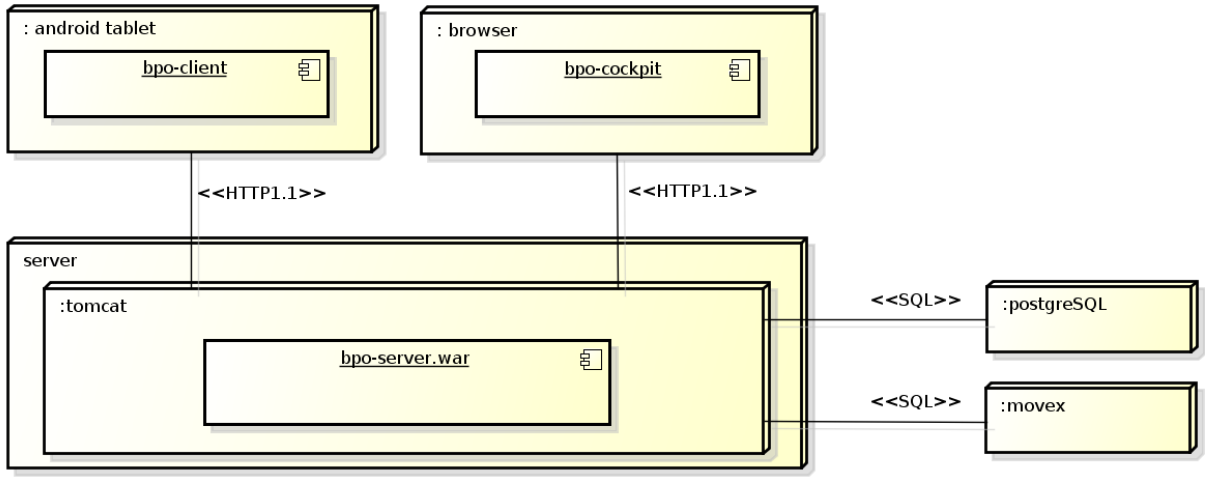


Abbildung 7.2.: Deployment Diagramm

Die Komponente **bpo-client** läuft auf einem Android Gerät, die Komponenten **bpo-cockpit** in einem beliebigen Browser. Beide Komponenten greifen auf die REST Schnittstelle der Komponente **bpo-server** zu, welche auf einem Web-Server läuft. Der **bpo-server** kommuniziert über SQL mit dem ERP System Movex und dem Datenbanksystem PostgreSQL. Im Rahmen dieser Arbeit werden jedoch die Produktionsaufträge aus einer CSV-Datei gelesen.

Nachfolgend wird auf den Aufbau der einzelnen Komponenten eingegangen.

7.3. Komponenten

Das Komponentendiagramm zeigt eine Übersicht über die einzelnen Komponenten. Jede der grün eingefärbten Komponenten wird im Rahmen der Bachelorarbeit implementiert. Die Restlichen sind verwendete Komponenten.

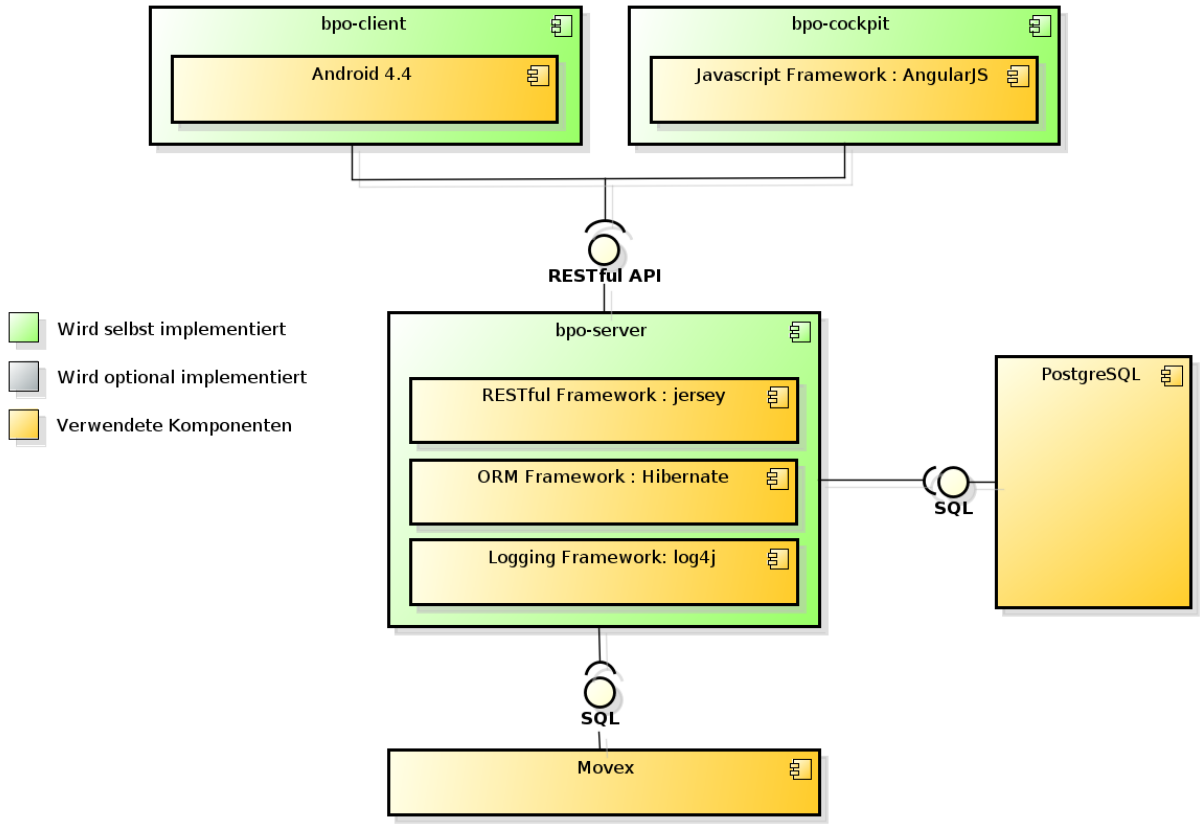


Abbildung 7.3.: Komponenten Diagramm

7.4. Komponente: bpo-server

In diesem Kapitel wird der Server beschrieben.

7.4.1. Verwendete Technologien

Nachfolgend wird eine Übersicht über die verwendeten Technologien gegeben.

Apache Tomcat

Apache-Tomcat wird als Servlet-Container eingesetzt. Servlets sind Java-Klassen, die vom Container instanziiert werden, Anfragen entgegennehmen und beantworten. Dabei können auch Filter konfiguriert werden, durch die die Anfragen geleitet und manipuliert werden können (Beispielsweise für Session-Management). [**?**]

Jersey

Bei Jersey handelt es sich um eine Referenzimplementierung des Java API für RESTful Web Services, kurz JAX-RS. Die Methoden von Resourcen werden mit Annotationen versehen, womit Endpunkte der REST-Schnittstelle definiert werden. Alle Anfragen werden an das Jersey-Framework übergeben, welches diese dann an die entsprechenden Resourcen weiterleitet.

Folgender Code-Ausschnitt zeigt die Annotation einer Resource-Klasse:

```
@Path("/productions")
public class ProductionsResource extends BaseResource {
  //...

  @GET
  @Path("/{planning_number}")
  @Produces(MediaType.APPLICATION_JSON)
  public Production getProduction(@PathParam("planning_number") final String ↙
      planningNumber) {
    return getSession().get(Production.class, planningNumber);
  }

  //...
}
```

Listing 7.1: Beispiel für die Jersey-Annotationen einer Resource

Dieser Endpunkt kann dann beispielsweise mit der URL 'http://base_url:8080/productions/123' aufgerufen werden, wobei die Production mit der übergebenen ID, im JSON-Format zurückgegeben wird. [**?**]

genson

Die Serialisierung von Java Objekten zu JSON wird mit genson erledigt. Genson integriert sich automatisch in Jersey, wenn die Bibliothek eingebunden wird. [**?**]

PostgreSql

PostgreSQL ist ein ein weit verbreitetes Open-Source Datenbankmanagementsystem. Es zeichnet sich durch seine hohe Stabilität und Skalierbarkeit aus. [**?**]

Hibernate

Hibernate ist ein ORM-Framework, womit Instanzen von Java-Klassen in eine relationale Datenbank gespeichert werden können. Jeglicher Datenbank Zugriff findet über das Hibernate-Framework statt.

Global wird Hibernate über die Datei 'hibernate.cfg.xml' konfiguriert. Darin werden die verwendete Datenbank, sowie die entsprechenden Entitäten festgelegt.

Die Attribute der Java-Klassen können dafür mit Annotationen versehen werden, woraus auch das Datenbank-Schema generiert werden kann. Beispielsweise wird aus folgenden Annotationen eine Datenbank-Tabelle 'employee' mit den Attributen id und name generiert. Zusätzlich wird eine Assoziation zum Problem erstellt.

```
@Entity
public class Employee {
  @Id
  @GeneratedValue
  private int id;

  @Column(nullable = false)
  private String name;

  @OneToMany
  private List<Problem> problems = new ArrayList<>();

  //..
}
```

Listing 7.2: Beispiel für die Hibernate-Annotationen einer Model-Klasse

In der Datei 'hibernate.cfg.xml' muss zusätzlich angegeben werden, welche Klassen Hibernate kennen muss.

```
<mapping class="ch.bpo.server.model.TaskCompletion" />
<mapping class="ch.bpo.server.model.Production" />
<mapping class="ch.bpo.server.model.Trolley" />
<mapping class="ch.bpo.server.model.TrolleyResponsibility" />
<mapping class="ch.bpo.server.model.TrolleyDelivery" />
<mapping class="ch.bpo.server.model.Task" />
<mapping class="ch.bpo.server.model.Employee" />
<mapping class="ch.bpo.server.model.Problem"></mapping>
```

Listing 7.3: Konfiguration der Klassen in der Datei 'hibernate.cfg.xml'

Wurde die Datenbank generiert, kann eine Instanz der Klasse Employee, einfach über die Hibernate-Session gespeichert werden:

```
hibernateSession.persist(employee);
```

Listing 7.4: Beispiel für die Verwendung der Hibernate-Session

Zusätzlich bietet Hibernate ein Transaction-Management. Kann eine Transaction nicht in die Datenbank geschrieben werden, kann ein Rollback durchgeführt werden. [**?**]

log4j over slf4j

Log4j ist ein weit verbreitetes Logging-Framework für Java. Als Abstractionslayer wird slf4j verwendet. Damit können Log-Nachrichten von eingesetzten Frameworks ebenfalls an log4j weitergeleitet werden.

Log4j implementiert das Singleton-Pattern. Benötigt eine Klasse einen Logger, so intanziiert sie diesen über die LoggerFactory von slf4j. Mit dieser Singleton-Instanz des Loggers können Log-Nachrichten mit der entsprechenden Wichtigkeit (z.B Info, Warn, Error) generiert werden. [**?**]

```
public class UncaughtExceptionMapper implements ExceptionMapper<Throwable> {

  private static Logger logger = LoggerFactory.getLogger(↙
      UncaughtExceptionMapper.class);

  @Override
  public Response toResponse(Throwable throwable) {
    logger.error("an uncaught exception happened", throwable);

    //...
  }
}
```

Listing 7.5: Beispiel für die Verwendung von slf4j over log4j

7.4.2. Build Management Tool

Apache Maven

Apache Maven ist ein weit verbreitetes Buildmanagement-Tool für Java. Damit können beispielsweise Abhängigkeiten konfiguriert werden, wobei diese dann aus einem zentralen Verzeichnis heruntergeladen und eingebunden werden.

Konfiguriert werden diese Abhängigkeiten in der Datei pom.xml. Folgende Ausschnitt bindet beispielsweise die das Jersey-Framework ein: [**?**]

```
<dependencies>
  <dependency>
    <groupId>org.glassfish.jersey.containers</groupId>
    <artifactId>jersey-container-servlet-core</artifactId>
    <version>2.17</version>
  </dependency>
</dependencies>
```

Listing 7.6: Ausschnitt aus Mavens pom.xml

7.4.3. Request Ablauf

Für ein besseres Verständnis wird nachfolgend der Ablauf eines typischen Requests aufgeführt.

Apache Tomcat nimmt ankommende HTTP-Requests entgegen. Für jeden Request wird eine Datenbank-Session erstellt, sowie eine Authentisierung des Benutzers durchgeführt. Danach wird der Request an Jersey weitergeleitet. Mithilfe von Dependency Injection injected Jersey der HTTP-Request, worin nun Informationen über den Benutzer sowie die Datenbank-Session sind. Anhand der aufgerufenen URL weiss Jersey welche Methode, in welcher Resource aufgerufen werden muss.

Das folgende Sequenzdiagramm visualisiert diesen Ablauf:

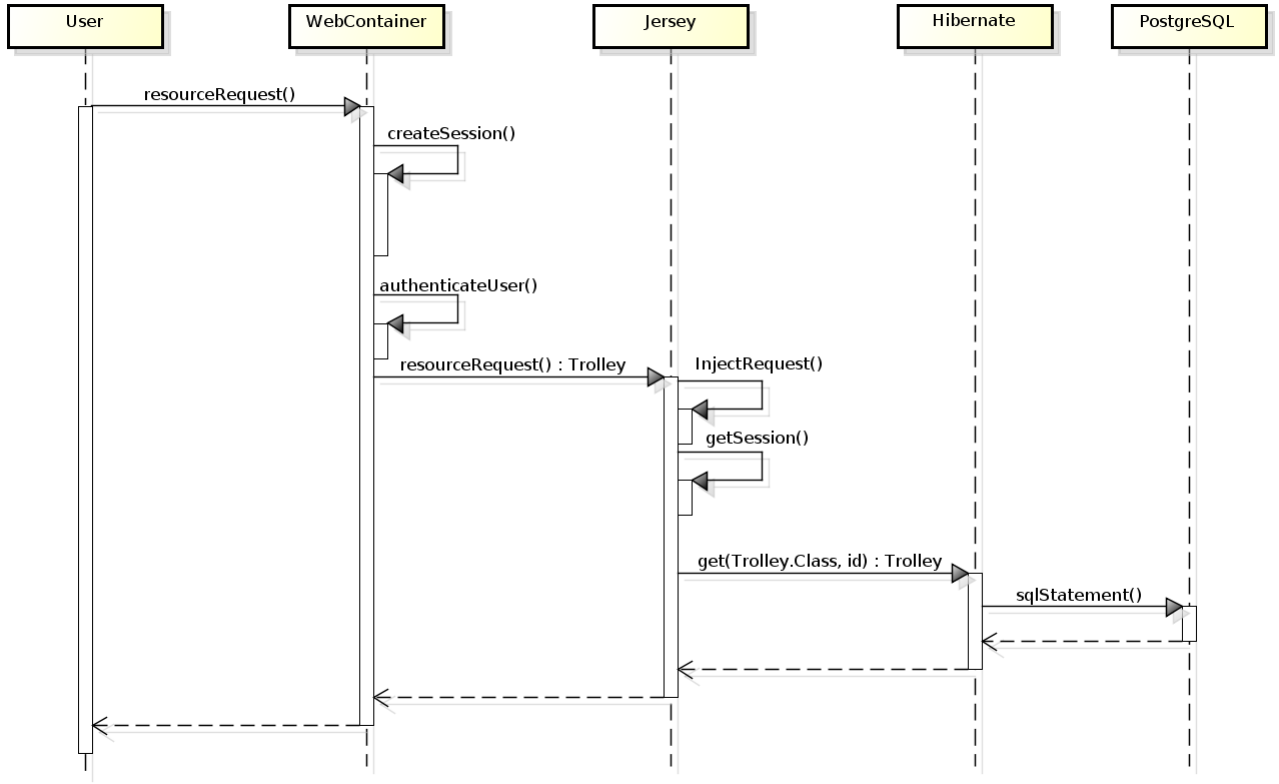


Abbildung 7.4.: Ablauf eines Trolley Requests

7.4.4. Package Übersicht

Nachfolgend eine Übersicht über die Packages des Servers.

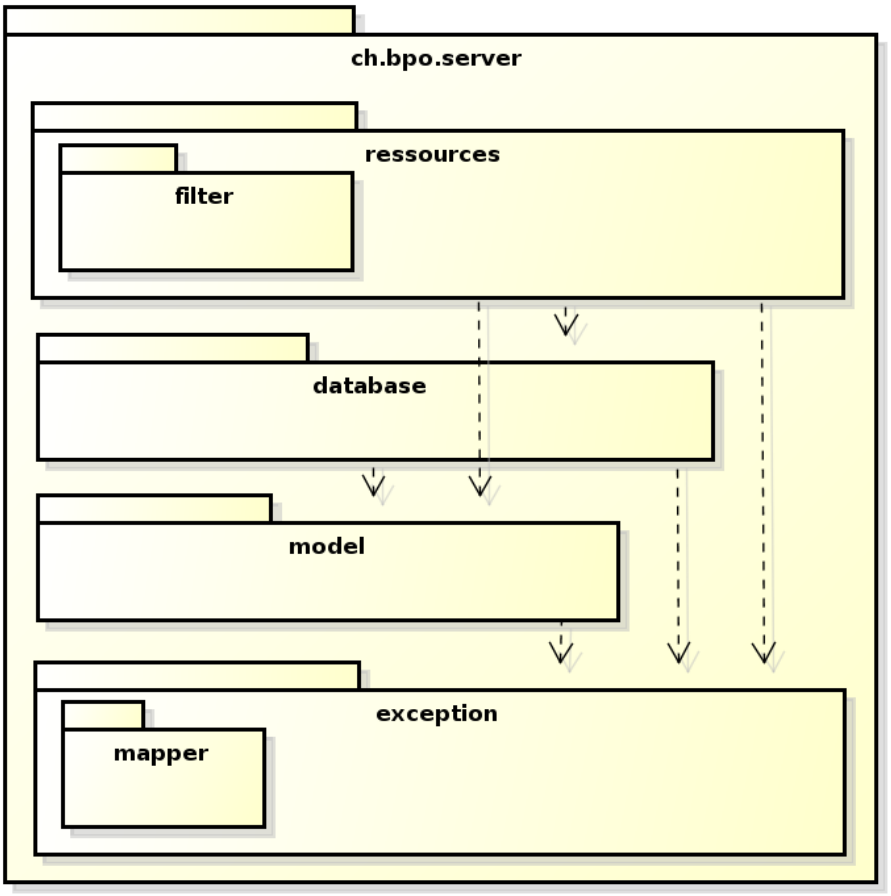


Abbildung 7.5.: Package Übersicht

Es folgt eine detaillierte Beschreibung der Packages und Klassen. Dabei werden wichtige Abläufe mit Sequenzdiagrammen visualisiert.

7.4.5. Package: Resources

Der Server bietet eine REST-Schnittstelle, über die er angesprochen werden kann. Nach der Jersey-Konvention befinden sich die aufrufbaren Methoden in Klassen im Package 'resources'. Für jede Resource gibt es eine eigene Klasse (z.B. TrolleysResource für die Operationen betreffend der Wagen). Damit ergeben sich folgende Resource-Klassen:

- AuthenticationResource
- ImportResource
- ProblemsResource
- ProductionsResource
- ProductionStatesResource
- TasksResource
- TrolleyDeliveriesResource
- TrolleysResource
- TrolleyStatesResource

Neben den Resource-Klassen beinhaltet das Package noch folgende Klassen:

- BaseResource
- RequestResourceAttributeProvider

Zudem beinhaltet das Sub-Package 'filter' die folgenden Servlet-Filter:

- DatabaseSessionFilter
- AuthenticationFilter

Nachfolgend werden die zusätzlichen Klassen, sowie die Filter beschrieben. Dies geschieht in einer aufbauenden Reihenfolge, damit ihr Zusammenspiel verständlich wird. Für die folgenden Ausführungen wird ein Verständnis für die Funktionsweise von Servlets und Servlet-Filtern vorausgesetzt.

DatabaseSessionFilter

Der DatabaseSessionFilter öffnet für jeden Request eine neue Datenbank-Session und setzt diese auf dem Request. Ist der Request abgearbeitet, wird ein Commit oder im Fehlerfall ein Rollback ausgeführt.

Das folgende Sequenzdiagramm zeigt diesen Ablauf ohne Fehler:

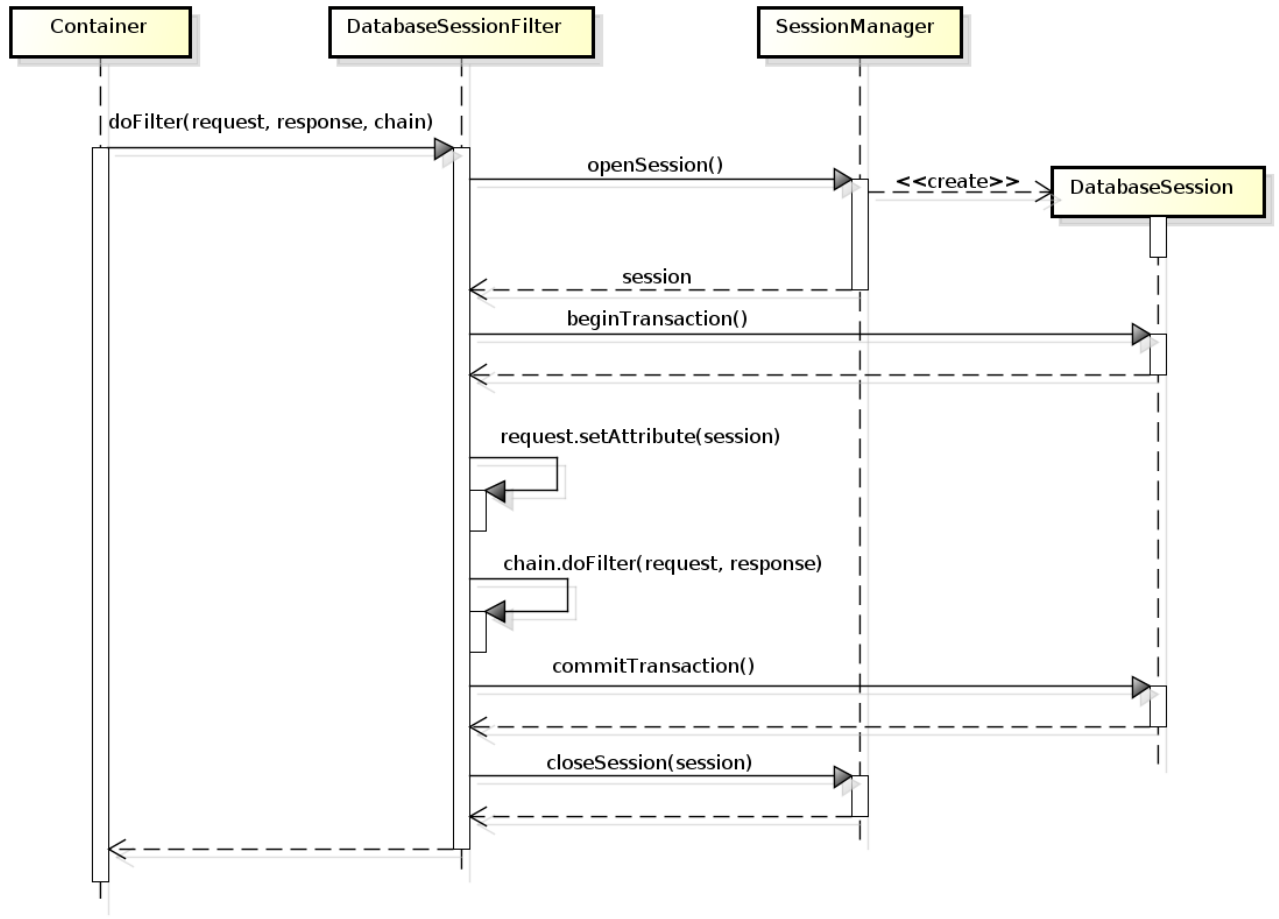


Abbildung 7.6.: Sequenzdiagramm des DatabaseSessionFilter

AuthenticationFilter

Der AuthenticationFilter liest aus dem Authentication-Header eines Requests den Benutzername und das Passwort. Diese werden mit der Datenbank auf ihre Korrektheit überprüft. Sollten diese nicht korrekt sein, wird eine NotAuthorizedException geworfen. Ansonsten wird eine Instanz des gefunden Mitarbeiters auf dem Request gesetzt und der Request weitergeleitet.

Das folgende Sequenzdiagramm zeigt diesen Ablauf ohne Fehler. Die Operationsaufrufe auf der Instanz des Requests und der Chain werden, in einer für Sequenzdiagramme unüblichen Syntax abgebildet, um dieses zu vereinfachen.

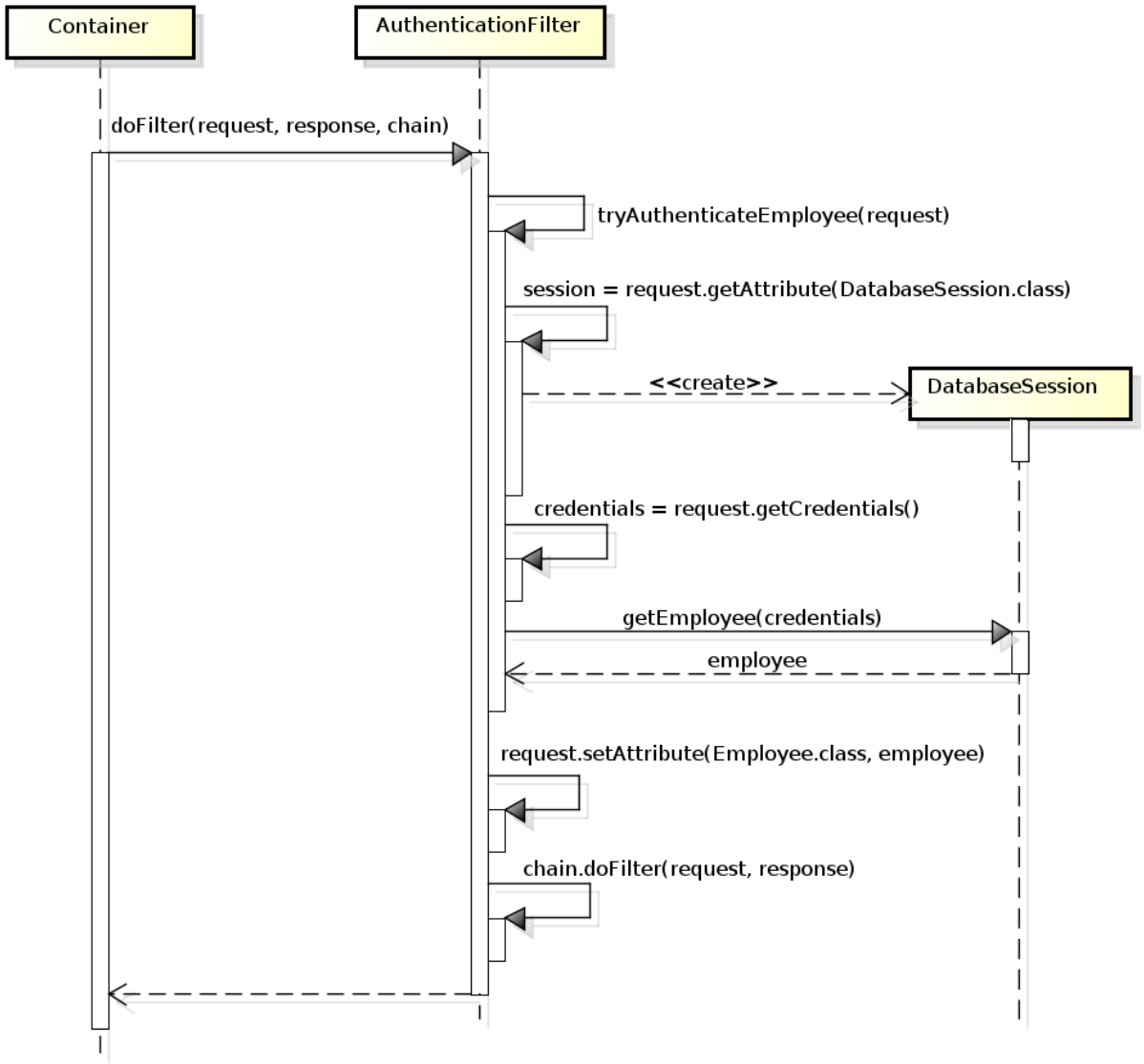


Abbildung 7.7.: Sequenzdiagramm des AuthenticationFilter

RequestResourceAttributeProvider

In die RequestResourceAttributeProvider-Klasse wird der aktuelle Request injected. Von diesem liest diese dann die Attribute, die von den Filtern gesetzt wurden.

```
public class RequestResourceAttributeProvider implements ↙
    ResourceAttributesProvider {

  @Context
  public HttpServletRequest request;

  @Override
  public DatabaseSession getDatabaseSession() {
    return (DatabaseSession) request.getAttribute(DatabaseSession.class.↙
        toString());
  }

  @Override
  public Employee getAuthenticatedEmployee() {
    return (Employee) request.getAttribute(Employee.class.toString());
  }
}
```

Listing 7.7: RequestResourceAttributeProvider-Klasse

Eine Instanz der Klasse wird im Konstruktor der BaseResource-Klasse erwartet.

BaseResource

Alle Resource-Klassen erben von der BaseResource-Klasse. Diese erwartet eine Instanz der RequestResourceAttributeProvider-Klasse, welche in die abgeleiteten Resourcen injected und an die Basis-Klasse weitergegeben wird.

```
public class AuthenticationResource extends BaseResource {

  @Inject
  public AuthenticationResource(ResourceAttributesProvider attributeProvider) {
    super(attributeProvider);
  }
}
```

Listing 7.8: Beispiel für die Injection des ResourceAttributeProviders in eine Resource

Damit können die Resource-Klassen auf die Attribute zugreifen, die von den Filtern gesetzt werden. Also beispielsweise den authorisierten Mitarbeiter abrufen:

```
public class BaseResource {
  private ResourceAttributesProvider attributeProvider;

  protected ResourceAttributesProvider getAttributeProvider() {
    return attributeProvider;
  }

  protected Employee getAuthenticatedEmployee() {
    return getAttributeProvider().getAuthenticatedEmployee();
  }
}
```

Listing 7.9: Funktionsweise der ResourceAttributeProvider-Klasse

Der Umweg über die RequestResourceAttributeProvider-Klasse ist notwendig, da es nicht möglich ist, den Request direkt auf der Resource zu injecten.

Dependency Injection

Die Dependency Injection wird über das mit Jersey mitgelieferte Framework HK2 konfiguriert. Dafür ist lediglich eine Klasse notwendig, die von der abstrakten Klasse AbstractBinder erbt. In der configure-Methode können daraufhin die Bindings definiert werden.

Beispielsweise wird in folgendem Statement das Interface ResourceAttributesProvider auf die Klasse RequestResourceAttributeProvider gemappt. Daraus ergibt sich der Vorteil, dass in den Unit Tests ein anders Mapping für die Dependency Injection verwendet werden kann um eine andere Instanz der Klasse zu mappen.

```
public class ServerBinding extends AbstractBinder {

  @Override
  protected void configure() {
    \\...
    bind(RequestResourceAttributeProvider.class).to(ResourceAttributesProvider.↙
        class);
    \\...
  }
}
```

Listing 7.10: Ausschnitt aus ServerBinding-Klasse

7.4.6. Package: Database

Im Database-Package befinden sich die folgenden Klassen:

- SessionManager
- ProductionQueries
- TrolleyQueries
- DatabaseSession

Nachfolgend eine detaillierte Beschreibung dieser Klassen.

SessionManager

Der SessionManager ist eine abstrakte Klasse, welche statische Methoden zum vereinfachten verwalten der Sessions zur Verfügung stellt. Beispielsweise kann eine neue Datenbank-Session geöffnet und diese wieder geschlossen werden.

ProductionQueries und TrolleyQueries

Die Queries-Klassen beinhalten lediglich Hibernate-Queries, die von gewissen Resourcen verwendet werden um effizienter Daten aus der Datenbank abzufragen. Dies ist bei gewissen Abfragen notwendig, da über mehrere Tabellen Daten abgefragt werden müssen und diese bis zu mehrere Sekunden dauern könnten.

Die genannten Queries sind in der Hibernate eigenen Sprache HQL verfasst.

DatabaseSession

Die Klasse 'DatabaseSession' ist ein Wrapper um die Hibernate-Session. Dabei werden Methoden, die auch direkt auf der Hibernate-Session aufgerufen werden könnten, gekapselt und komfortabler zur Verfügung gestellt. Nachfolgender Code-Ausschnitt zeigt einige Beispiele für die vereinfachte Verwendung der Hibernate-Session:

```
package ch.bpo.server.database;

import java.io.Serializable;

public class DatabaseSession {
  private Session hibernateSession;

  public DatabaseSession(Session session) {
    this.hibernateSession = session;
  }

  //...

  public void persistAll(List<?> entities) {
    for (Object object : entities) {
      getHibernateSession().persist(object);
    }
  }

  //...

  @SuppressWarnings("unchecked")
  public <T> T get(Class<T> classz, Serializable id) {
    T entity = (T) getHibernateSession().get(classz, id);

    if (entity == null) {
      throw new EntityNotFoundException();
    }

    return entity;
  }

  //...

  public <T> T getByAttribute(Class<T> classz, String attribute, Object value) ↙
      {
    List<T> entities = getAllByAttribute(classz, attribute, value);

    if (entities.size() == 0) {
      throw new EntityNotFoundException();
    } else if (entities.size() > 1) {
      throw new MultipleEntitiesFoundException();
    }

    return entities.get(0);
  }

  //...
}
```

Listing 7.11: Ausschnitt aus der DatabaseSesion-Klasse

7.4.7. Package: Model

Im Model-Package befinden sich die benötigten Models des Servers. Diese werden im nachfolgenden Klassendiagramm des Servers genauer beschrieben.

Klassendiagramm

Das Klassendiagramm des Servers beinhaltet alle Klassen aus dem Domain Model der Domainanalyse. Folgende Unterschiede hat dieses:

- Die Klasse Mattress heisst Task
- Die Klasse MattressCompletion heisst TaskCompletion
- Der Enum DescriptionKey heisst TaskValue
- Die MattressDescriptions der Matress werden nicht als Assoziation, sondern als Hashtable abgebildet

Zusätzlich gibt es folgende Klassen und Enums:

- Klasse ProductionState
- Klasse TrolleyState
- Enum State

Daraus resultiert das folgende Diagramm:

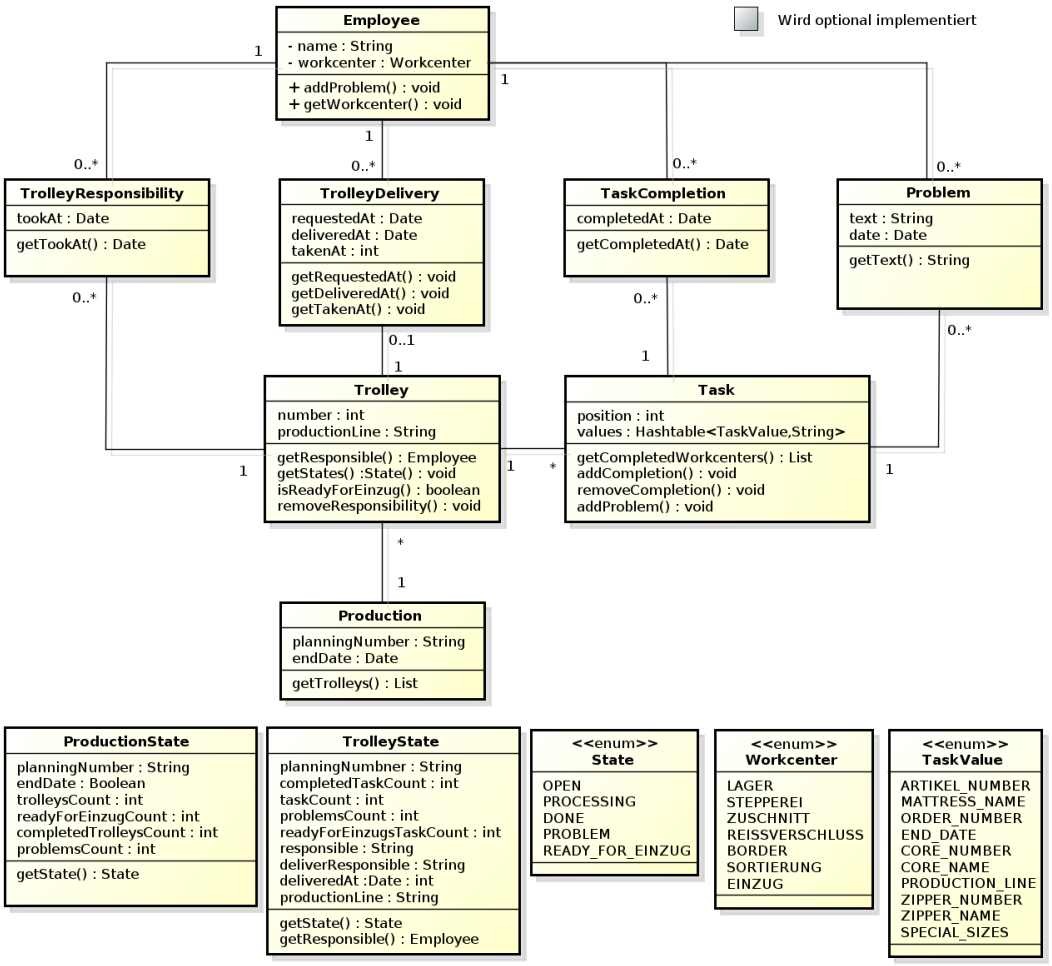


Abbildung 7.8.: Klassendiagramm des Servers

Die Klassen Production- und TrolleyState geben den Status der jeweiligen Ressource zurück. Die beiden Klassen wurden aus Performancegründen eingeführt, da beim direkten konstruieren des Status aus den Objekten, also beispielsweise aus dem Production Objekt, ebenfalls alle Trolleys und Tasks aus der Datenbank geladen werden mussten. Diese Abfragen erwiesen sich allerdings als sehr unperformant.

Das Performanceproblem ist mit Hilfe von HQL (Hibernate Query Language) gelöst. Damit werden lediglich die, für die jeweilige Berechnung eines Status benötigten Werte aus der Datenbank gelesen und in die Status-Klasse geschrieben.

Beispiel der Abfrage der abgeschlossenen Tasks einer Produktion:

```
public static String taskCompletionsCount() {
  return "(select count(completions) from TaskCompletion as completions"
  + "          inner join completions.task as task"
  + "          inner join task.trolley as trolley"
  + "          where trolley.production.planningNumber = production.↙
    planningNumber"
  + "  ) as taskCompletionsCount";
}
```

Listing 7.12: Abfrage die die abgeschlossenen Tasks einer Produktion zählt

Aus diesem Wert und den Anzahl Tasks kann dann die ProductionState-Klasse den Status der Produktion erschliessen:

```
public State getState() {
  if (problemsCount > 0) {
    return State.PROBLEM;
  } else if (completedTrolleysCount == trolleysCount) {
    return State.DONE;
  } else if (inProgress || (completedTrolleysCount > 0 && ↙
     completedTrolleysCount < trolleysCount)) {
    return State.PROCESSING;
  } else if (completedTrolleysCount == 0) {
    return State.OPEN;
  } else {
    throw new InvalidStateException();
  }
}
```

Listing 7.13: Methode die den Status der Produktion zurückgibt

Datenbankschema

Aus den Hibernate-Annotationen im Model ergibt sich folgendes Datenbankschema:

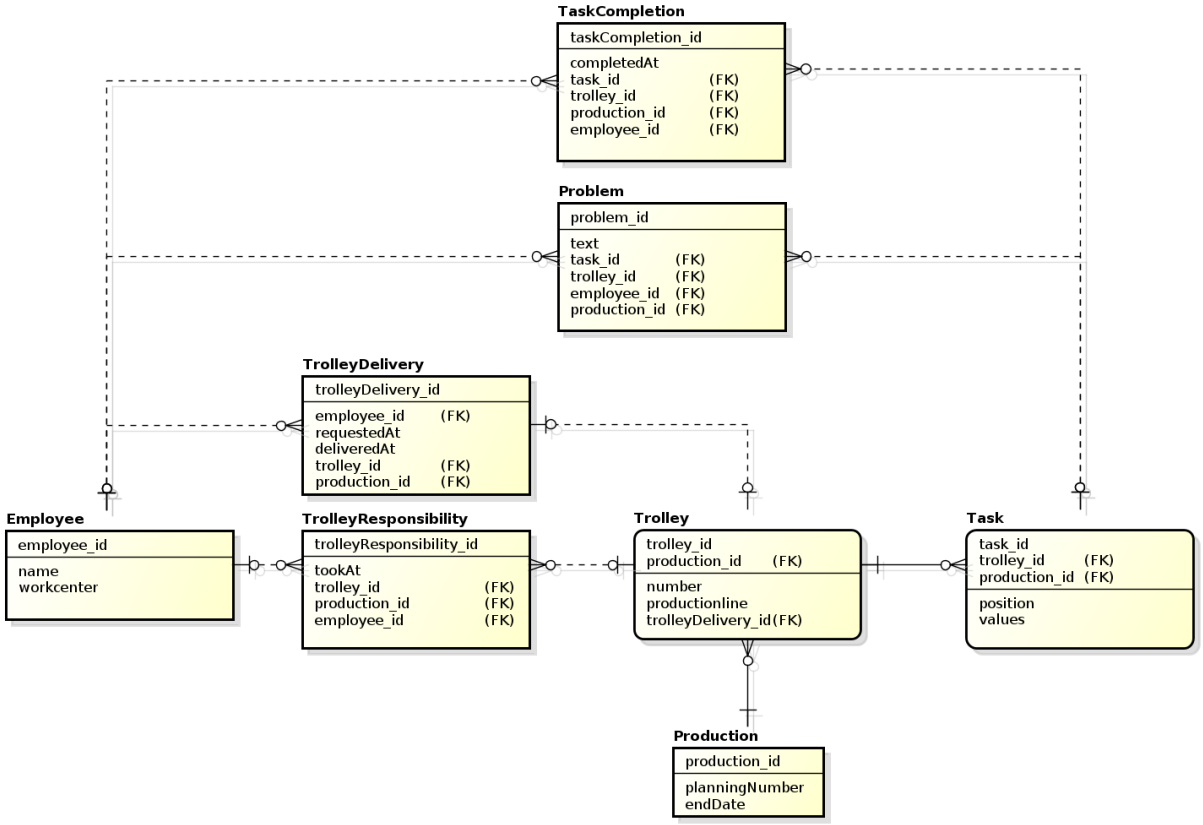


Abbildung 7.9.: Entity Relation Diagramm

Die durchzogenen Verbindungen stellen eine Identifying Relation dar. Das heisst die Verbindungen ist zwingend (beispielsweise ein Trolley braucht zwingend eine Production). Eine Non-Identifying Relation wird gestrichelt dargestellt und bedeutet, dass die Relation optional ist.

7.4.8. Package: Exception

Im Exception-Package befinden sich, neben den selbstdefinierten Exceptions auch die Exception-Mapper im Sub-Package 'mapper'. Diese werden vom Jersey-Framework verwendet um Exceptions auf die jeweiligen HTML-Status-Codes zu mappen.

Nachfolgend ein Beispiel eines Mapper, der die EntityNotFoundException auf den Status-Code 404 mappt:

```
@Provider
public class EntityNotFoundExceptionMapper implements ExceptionMapper<↙
    EntityNotFoundException> {
  public Response toResponse(final EntityNotFoundException ex) {
    return Response.status(Status.NOT\_FOUND).entity(ex.getMessage()).build();
  }
}
```

Listing 7.14: ExcpetionMapper für die EntityNotFoundException

Ist kein Mapper für eine Exception registriert, kommt ein UncaughtExceptionHandler zum Einsatz, der sich ebenfalls im Sub-Package Mapper befindet. Dieser loggt den Fehler und mappt diesen auf den Status-Code 500.

7.5. Komponente: bpo-client

In diesem Kapitel wird der Client beschrieben.

7.5.1. Verwendete Technologien

Nachfolgend eine Übersicht über die wichtigsten, verwendeten Technologien des Clients.

Android

Android ist ein von Google entwickeltes Betriebssystem, welches primär für mobile Geräte wie Smartphones, Tablets, Smart Watches etc. eingesetzt wird. Das Betriebssystem baut auf einem Linux Kernel auf und ist unter der GPLv2 Lizenz veröffentlicht. Die Entwicklungsumgebung kann unter https://developer.android.com/sdk/index.html heruntergeladen werden. [**?**]

Gson

Der Client kommuniziert über eine REST Schnittstelle mit dem Server. Die Schnittstelle erwartet und produziert Daten im JSON-Format. Gson ist eine Java Libary welche das de- sowie serialisieren von Java Objekten in JSON und umgekehrt vornimmt. [**?**]

Butterknife

Butterknife ist eine Injection Libary für Android. Butterknife wird im Client vor allem für die vereinfachte Instanziierung von User Interface Elementen wie Views, Knöpfen, Textfeldern etc. verwendet.

Ohne Butterknife werden User Interface Elemente oft in der `onCreate()` Methode der jeweiligen Activity instanziiert, womit diese unübersichtlich wird.

```
View view = (View) findViewById(R.id.login_form);
EditText mUsername = (EditText) findViewById(R.id.user_name);
EditText mPasswordView = (EditText) findViewById(R.id.password);
View mProgressView = (View) findViewById(R.id.login_progress);
```

Listing 7.15: Standardmässiges Instanziieren von UI-Elementen in Android

Mit Butterknife genügt hingegen folgender Aufruf in der `OnCreate()` Methode.

```
ButterKnife.inject(this);
```

Listing 7.16: Verwendung von Butterknife Teil 1

Anschliessend können die Membervariabeln der Elemente mit Annotationen versehen werden, womit diese von Butterknife korrekt injected werden

Folgendes Beispiel injected die Login View. [**?**]

```
@InjectView(R.id.login_form)
View mLoginFormView;
```

Listing 7.17: Verwendung von Butterknife Teil 2

OkHttp

OkHttp vereinfach das Erstellen von HTTP Requests. Im Client wird OkHttp für das Ansprechen der REST Schnittstelle des Servers verwendet.

Folgendes Beispiel zeigt einen GET Request, welcher vorgängig aufgebaut wurde. Dem `httpClient` Objekt muss der Request, sowie eine Callback-Methode übergeben werden. [**?**]

```
httpClient.newCall(request).enqueue(new DefaultCallback(callback){
  @Override
  public void onResponse(Response response) throws IOException {
    if (!response.isSuccessful()) {
      callback.failure(response);
    }
    else {
      callback.result(gson.fromJson(response.body().charStream(), classOfT));
    }
  }
});
```

Listing 7.18: Beispiel eines GET Requests mit OkHttp

7.5.2. Build Management Tool

Gradle

Gradle ist ein auf Java basierendes Build Managment Tool. Das Konfigurations-File 'build.gradle' ist standardmässig im Root Verzeichnis der Android Applikation abgelegt.

Beispielsweise können die die Dependencies zu Gson und OkHttp wie folgt eingebunden werden: [**?**]

```
dependencies {
  compile 'com.google.code.gson:gson:2.3.1'
  compile 'com.squareup.okhttp:okhttp:2.4.0'
}
```

Listing 7.19: Beispiel für das Managed von Dependencies mit Gradle

7.5.3. Android Grundlagen

Android Manifest

Das `AndroidManifest.xml` ist eine XML-Datei welche die Applikationsumgebung beschreibt. Im AndroidManifest.xml sind unter anderem folgende Informationen enthalten:

- Applikationsname
- Verwendetes Theme
- Verwendete Services
- Activitiy die beim Starten der Applikation ausgeführt werden soll
- Alle weiteren Activities
- API Level
- Berechtigungen (beispielsweise Zugriff auf Netzwerkinformationen)

Activity

Öffnet der Benutzer die Applikation, wird eine Activity aufgerufen, welche dann wiederum die View aufruft. Jedes Fenster braucht eine eigene Activity, welche die Interaktion mit dem Benutzer steuert. Einzelne Activities sind unabhängig voneinander.

Jede erstellte Activity erbt direkt oder indirekt von der `Activity` Klasse. Dabei können Methoden wie `OnCreate()`, `OnStart()`, `OnResume()`, `OnPause()`, `OnDestroy()`, `OnStop()`, `OnRestart()` und weitere überschrieben werden. Einer der wichtigsten ist dabei die `OnCreate()` Methode. Innerhalb dieser Methode wird üblicherweise die View, der Adapter, die Listener und der ErrorHandler instanziiert.

Nachfolgend ein Beispiel der TrolleyStatesActivity.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState, R.layout.activity_trolley_states);
  ButterKnife.inject(this);

  setTitle(getString(R.string.title_activity_trolley, getWorkcenter().toString↙
      ()));

  adapter = new TrolleyStatesAdapter(this, listView, R.layout.↙
      trolley_state_list_layout);

  initRecurringReloadTrolleyStates();
}
```

Listing 7.20: Beispiel der OnCreate Methode der TrolleyStatesActivity

Intent

Ein Intent wird dazu verwendet eine nächste Activity aufzurufen. Dem Intent können Informationen mitgeben werden, welche an die aufzurufende Activity weitergegeben werden. Dabei wird dem Intent die aktuelle und die nächste Activity übergeben.

```
private void startTrolleyStatesActivity() {
  Intent tasksIntent = new Intent(LoginActivity.this, TrolleyStatesActivity.↙
      class);
  startActivity(tasksIntent);
}
```

Listing 7.21: Beispiel Aufruf der TrolleyStatesActivity aus der LoginActivity

View

Die Views werden im XML-Format beschrieben. Innerhalb des Clients wird primär das LinearLayout verwendet, womit die Elemente horizontal oder vertikal angeordnet werden können. Innerhalb eines LinearLayout können weitere Layouts erstellt werden. Nachfolgend ein Ausschnitt aus dem XML des UncaughtActivityView.

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:bootstrapbutton="http://schemas.android.com/apk/res-auto"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <LinearLayout
        android:orientation="vertical"
        ...
        <TextView
            android:id="@+id/error_icon_wrench"
            ...
            android:gravity="center" />

        <TextView
            android:layout_width="fill_parent"
            ...
            android:text="@string/unexpected_error_occurred" />

        <TextView
            android:id="@+id/error_message"
            ...
            android:padding="@dimen/padding_large" />

        <com.beardedhen.androidbootstrap.BootstrapButton
            android:id="@+id/btnRestart"
            android:onClick="reStart"
            ...
            android:text="Neustart"
            bootstrapbutton:bb_type="success" />
    </LinearLayout>
</ScrollView>
```

Listing 7.22: Ausschnitt aus UncaughtActivityView XML

Daraus resultiert die folgende View:

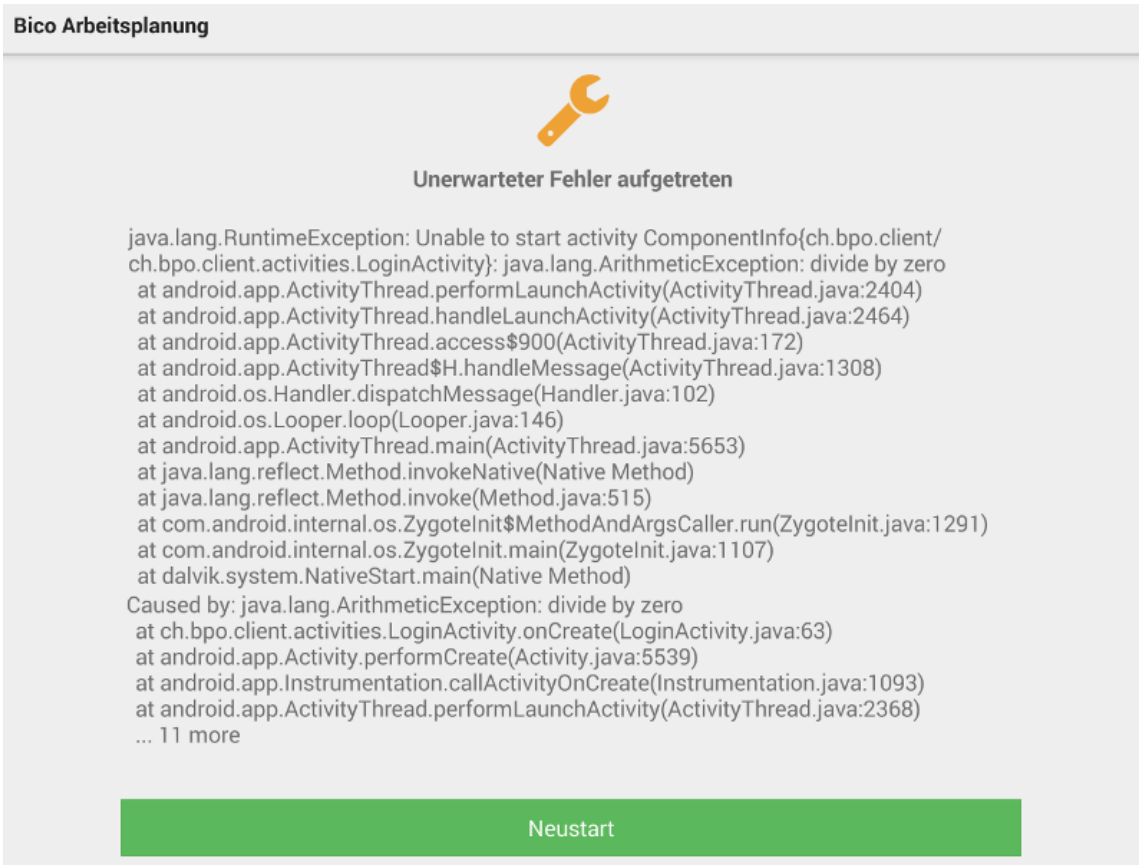


Abbildung 7.10.: Uncaught Exception View

View Holder Pattern

Das View Holder Pattern wurde beim Client aus Performancegründen eingesetzt. Der Adapter hält einen ViewHolder, der die Daten für die View 'cached'. Wird eine View gezeichnet, so müssen nicht mehr alle Elemente neu erstellt werden, sondern die bestehenden werden modifiziert.

Adapter

Adapter werden für die Verwendung von ListViews benötigt. Eine ListView wird mit einem Adapter instanziiert, womit die Activity nur noch mit dem Adapter kommuniziert und nicht direkt über die View. Im Adapter sind die Daten vorhanden, welche von der View dargestellt werden sollen.

7.5.4. Package Übersicht

Nachfolgend eine Übersicht über die Packages des Clients.

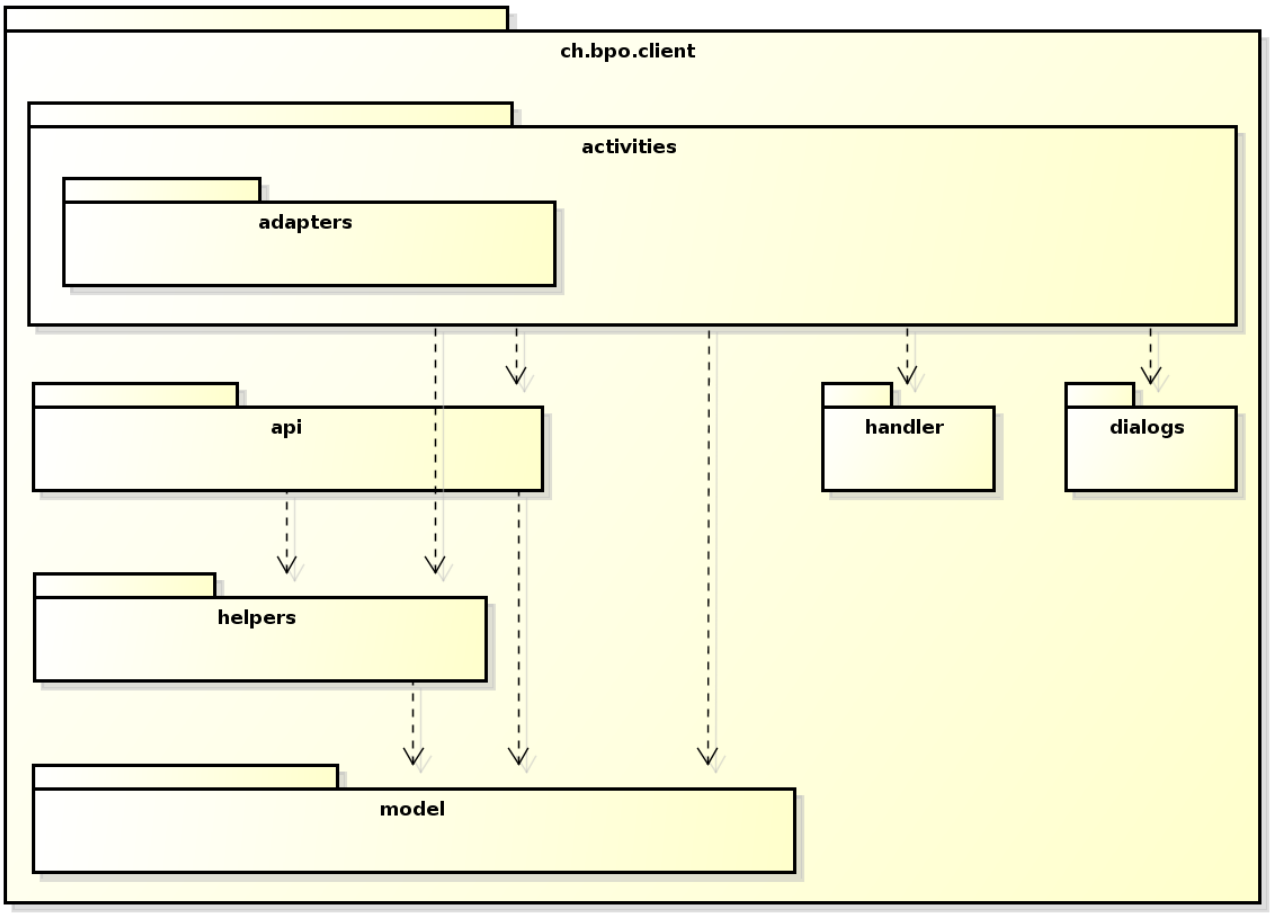


Abbildung 7.11.: Package Übersicht des Clients

Es folgt eine detaillierte Beschreibung der Packages und Klassen.

7.5.5. Package: Activities

Das Activities-Package beinhaltet die folgenden Klassen:

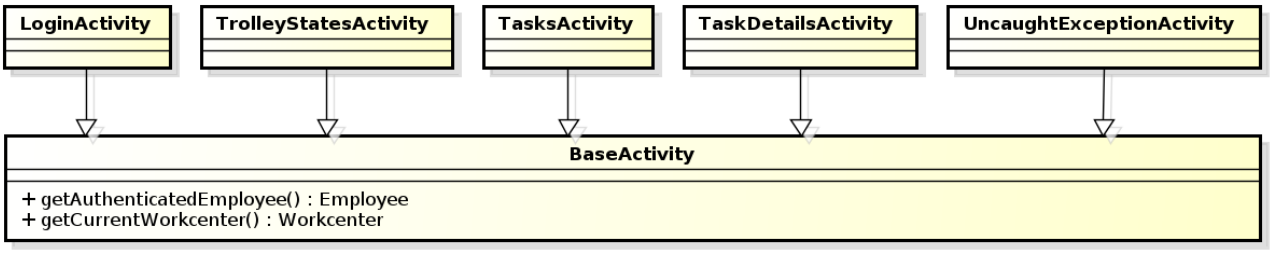


Abbildung 7.12.: Activities des Clients

Um das DRY-Prinzip einzuhalten wurde die abstrakte Klasse `BaseActivity` eingeführt, welche Methoden beinhaltet, die von allen Activities verwendet werden.

Im Sub-Package Adapters sind die Adapter abgelegt, welche von Android für die Verwendung der ListViews benötigt werden.

7.5.6. Package: API

Das API Package wird ausschliesslich vom Activities Package verwendet und bildet die Schnittstelle zum Server.

Übersicht der Klassen im Package

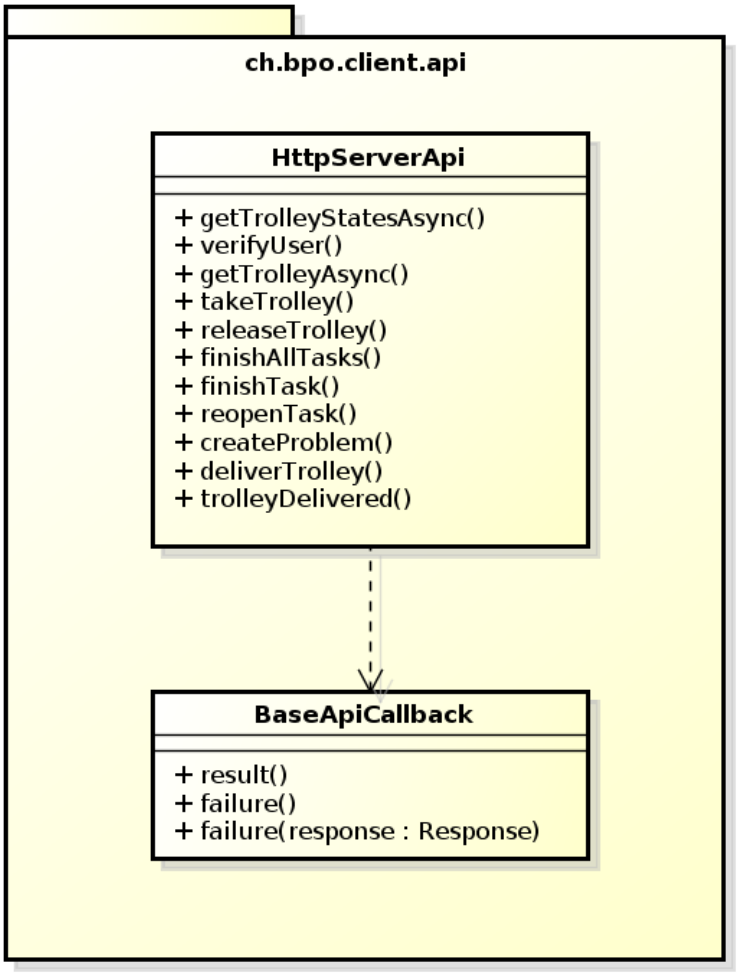


Abbildung 7.13.: Übersicht der Klassen im Package ch.bpo.client.api

Jeder Methode wird ein Callback übergeben, welcher vom API im Erfolgs- sowie im Fehlerfall aufgerufen wird. War der Request erfolgreich, wird die Antwort bereits zu der entsprechenden Java-Klasse deserialisiert.

Das folgende Sequenzdiagramm zeigt den Aufruf der `getTrolleyAsync` Methode. Dafür wird ein Callback übergeben, der aufgerufen wird, sobald der Server antwortet.

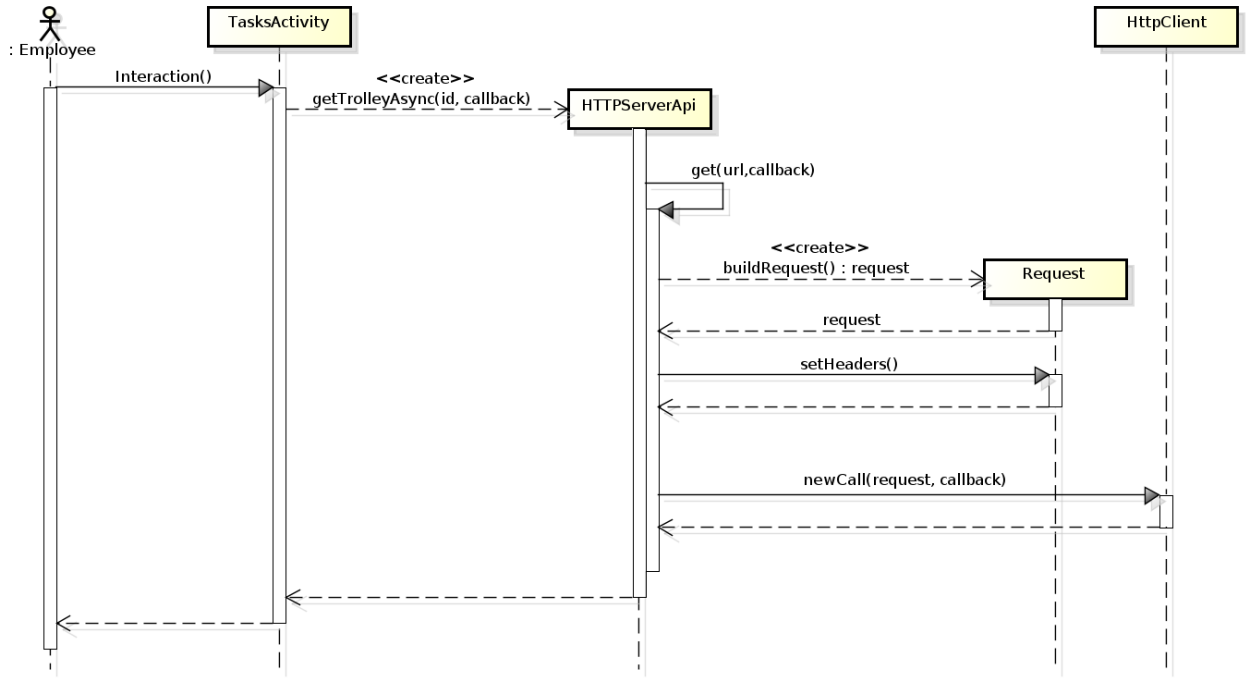


Abbildung 7.14.: Ablauf getTrolleyAsync Aufruf

Der Aufruf des Requests aus der Activity sieht wie folgt aus:

```
new HttpServerApi().getTrolleyAsync(getTrolleyId(), new BaseApiCallback<Trolley
    >(this) {
  @Override
  public void result(final Trolley trolley) {
    //...
  }
});
```

Listing 7.23: Aufruf der getTrolleyAsync-Methode

Mit dem Trolley-Objekt kann dann weitergearbeitet werden.

7.5.7. Package: Helpers

Im Helpers-Package sind mehrere Helfer-Klassen untergebracht. Ein Beispiel dafür ist die `ToastHelper` Klasse, welche das Ausgeben von ToastMeldungen vereinfacht:

```
public abstract class ToastHelper {
  public static void makeToastMessageInCenter(Context context, String message) ↙
      {
    final Handler handler = new Handler(Looper.getMainLooper());
    final Toast toast = Toast.makeText(context.getApplicationContext(), message↙
        , Toast.LENGTH_LONG);
    toast.setGravity(Gravity.CENTER, 0, 0);
    handler.post(new Runnable() {
      @Override
      public void run() {
        toast.show();
      }
    });
  }
}
```

7.5.8. Package: model

Das Model-Package beinhaltet alle Model-Klassen. Der Client implementiert die selben Klassen, wie der Server. Auf den Klassen gibt es allerdings nur Getter, da alle Daten lediglich vom Server abgerufen werden und keine Werte auf dem Client gesetzt werden müssen. Erstellt werden die Instanzen von der Library genson, welche dafür keine Setter benötigt.

7.5.9. Package: handler

Im Handler-Package befindet sich zur Zeit nur die `ErrorHandler` Klasse. Der Handler wird in der Klasse BaseActivity gesetzt, von welcher alle Activities erben.

```
Thread.setDefaultUncaughtExceptionHandler(new ExceptionHandler(this));
```

Listing 7.24: Setzen des ErrorHandlers

Tritt eine UncaughtException auf, führt dies nicht zu einem Absturz der Applikation sondern es wird eine Activity aufgerufen, die den Stack Trace des Fehlers aufzeigt.

7.5.10. Package: dialogs

In diesem Package werden Dialoge abgelegt. Zur Zeit beinhaltet das Package lediglich die `ReportProblemDialog` Klasse. Diese wird dazu verwendet um ein Problem auf einem Task zu melden.

7.6. User Interface

Der Client hat eine überschaubare Anzahl Ansichten. Folgende Abbildung zeigt diese, inklusive der Navigationsmöglichkeiten zwischen den Ansichten auf.

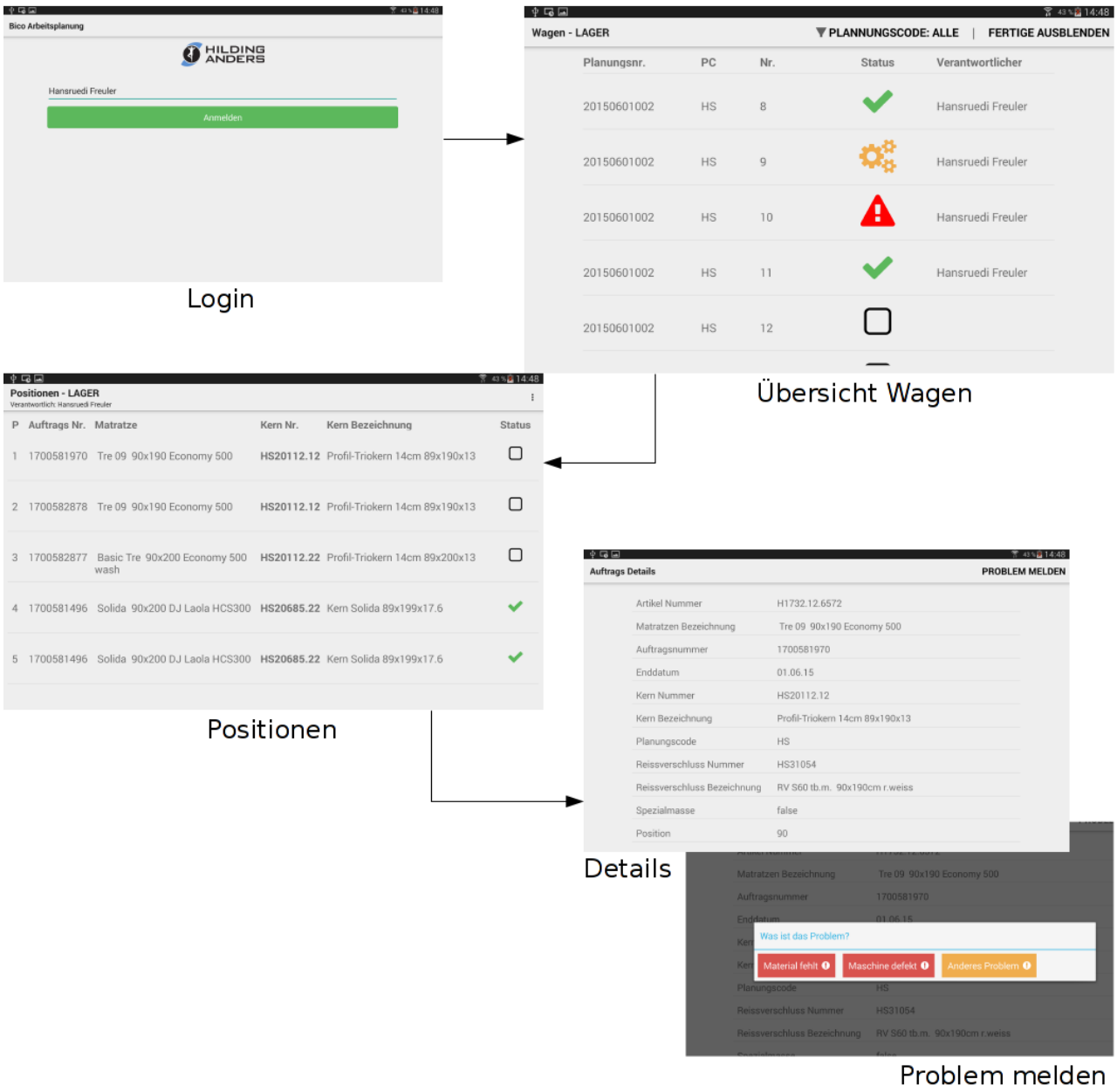


Abbildung 7.15.: Ansichten und deren Zusammenspiel des Clients

7.7. Komponente: bpo-cockpit

Das Cockpit ist eine Web-Applikation, über die der Meister Produktionsdaten importieren und den aktuellen Produktionsstatus überwachen kann. Das Cockpit ist mit HTML und Javascript implementiert und kommuniziert über die REST-Schnittstelle des Servers mit diesem.

7.7.1. Verwendete Technologien

Nachfolgend eine Übersicht über die wichtigsten, verwendeten Technologien des Cockpits.

AngularJS

Angular JS ist ein Open Source Javascript-Framework von Google, welches die Java Script Programmierung deutlich vereinfacht. [**?**]

Abbildung 7.16.: Logo von AngularJS

AngularJS arbeitet nach dem 'configuration over convention' Prinzip. Das heisst, dass die Projektstruktur nicht vorgegeben ist. Von Google empfohlen wird jeweils alle Dateien, wie Controller und View, eines Use-Cases zusammen abzulegen.

```
app
├── app.css
├── app.js
├── import_load.json
├── index.html
├── main_menu
│   ├── main_menu.html
│   └── main_menu.js
├── production_import
│   ├── production_import.html
│   ├── production_import.js
│   └── production_import_resource.js
└── production_state
    ├── level1.html
    ├── level2.html
    ├── production_resource.js
    └── production_state.js
```

Abbildung 7.17.: Datei Struktur der AngularJS Applikation

Bootstrap

Bootstrap ist ein weit verbreitetes CSS-Framework, womit ein ansprechendes Design umgesetzt werden kann. [**?**]

Bower

Bower ist ein Open Source Paketverwaltungstool zum komfortablen verwalten von Abhängigkeiten. Beispielsweise kann damit AngularJS und Bootstrap eingebunden werden. Dies geschieht über die Konfigurationsdatei 'bower.json', worin die Abhängigkeiten definiert werden:

```
{
  "name": "bpo-cockpit",
    "description": "bpo-cockpit",
    "version": "1.0.0",
    "dependencies": {
      "angular": "~1.3.15",
      "angular-resource": "~1.3.15",
      "bootstrap": "~3.3.4",
      "angular-route": "~1.3.15",
      "font-awesome": "~4.3.0",
      "angular-notify": "~2.5.0",
      "angular-busy": "~4.1.3"
    }
}
```

Listing 7.25: Ausschnitt aus bower.json

Die Abhängigkeiten werden mit dem Befehl 'bower install' in das Verzeichniss 'app/bower_components/' heruntergeladen. Daraufhin können diese, beispielsweise in der Datei 'index.html' eingebunden werden. [**?**]

```
  <!-- ... -->
  <script src="bower_components/angular/angular.js"></script>
  <!-- ... -->
```

Listing 7.26: Beispiel für das Einbinden einer von bower installierten Abhängigkeit

7.7.2. User Interface

Der Cockpit hat eine überschaubare Anzahl Ansichten. Folgende Abbildung zeigt diese, inklusive der Navigationsmöglichkeiten zwischen den Ansichten auf.

Abbildung 7.18.: Ansichten und deren Zusammenspiel des Cockpits

Teil IX.

User Acceptance Test

8. User Acceptance Test

Nach Abschluss der Implementationsphase wird der erstellte Prototyp des Clients im Produktionsumfeld getestet. Der Test wird mit einem Arbeiter aus dem Lager-Workcenter durchgeführt. Dieser benutzt dabei eines der evaluierten Android-Tablets um seine alltägliche Arbeit zu verrichten. Die Zeitbeschränkung liegt bei einer Stunde. Am Ende soll der Mitarbeiter ein Feedback und weitere Inputs geben, welche für Ergänzungen oder Anpassungen zum produktiven Einsatz noch nötig sind.

8.1. Ziele

- Feedback Usability Client.
- Feedback Usability des Tablet.
- Feedback zur Tauglichkeit für den produktiven Einsatz.
  Sind alle benötigten Informationen vorhanden?
  Ist der Workflow praxistauglich?
  Ist die W-Lan Verbindung stabil?

8.2. Umgebung

Nachfolgend eine paar Worte zur Umgebung, in der Test stattfinden wird.

8.2.1. Ort

Der Test wird in der Lagerhalle des Industriepartners Bico durchgeführt. Dies entspricht der realen Produktionsumgebung und gibt Aufschluss über die Stabilität der Verbindung vom Android Client zum Server, da mobiles Roaming oder anderen Störvariablen zu Verbindungsunterbrüchen führen könnten.

8.2.2. Testperson

Die Testperson ist mit der Handhabung eines Android Systems vertraut und technisch versiert. Dadurch ist die Einführung in den Client vereinfacht.

8.3. Vorbereitung

Zum Arbeitsbeginn startet der Mitarbeiter die Produktion mit den üblichen Produktionslisten. In dieser Zeit werden die aktuellen Produktionsdaten in das entwickelte System importiert und auf ihre Korrektheit überprüft. Anschliessend wird der Arbeitsstand mit dem des Mitarbeiters abgeglichen. Daraufhin bekommt die Testperson eine kurze Einführung in den Client und es findet eine fliegender Wechsel auf den Gebrauch des Tablets statt. Hierfür müssen dem Gerät die W-LAN Verbindungen im Lager bekannt sein.

8.4. Ablauf

Die Testperson arbeitet wie üblich in der Lagerhalle. Anstatt der Produktionslisten benutzt er aber den Android-Client um seine Arbeit zu verrichten. Somit ergibt sich die Möglichkeit den Mitarbeiter zu begleiten und das Produkt unter realen Bedingungen zu beobachten. Geplant ist, dass der Mitarbeiter in einer ersten Phase folgende Use Cases durchspielen wird:

- Auftrag abrufen
- Auftrag übernehmen
- Auftrag abschliessen

In einer zweiten Phase wird er in folgende Use Cases eingeführt, die er ausprobieren soll:

- Verantwortlichkeit abgeben
- Problem melden

8.5. Durchführung

Auf folgenden Seiten werden die Ergebnisse des Tests festgehalten und ausgewertet.

8.5.1. Beobachtungen

Abbildung 8.1.: Ein Ausschnitt aus dem Lager.

Beginn

Nachdem die aktuellen Produktionsdaten in das System importiert wurden, erhielt die Testperson das Tablet, ohne weitere Erklärung. Intuitiv verwendete der Mitarbeiter die Applikation richtig und rief die benötigten Informationen ab.

Verantwortlichkeit übernehmen

Es war nicht intuitiv, dass der Mitarbeiter die Verantwortung für einen Wagen übernehmen musste, um einzelne Matratzen als abgeschlossen zu markieren. Der Mitarbeiter erhielt eine Fehlermeldung, dass er die Verantwortlichkeit für den Wagen übernehmen muss, um eine Position abzuschliessen. Daraufhin suchte er nach der entsprechenden Option, fand diese allerdings erst nach einer kurzen Hilfestellung.

Positionen abschliessen

Der Mitarbeiter schloss die Positionen jeweils ab, nachdem er die Kerne auf das Palett auflud. Da der gleiche Kern oft mehrmals hintereinander vorkommt und er diese zusammen auf das Palett laden kann, musste der Mitarbeiter auf dem Client jeweils mehrere Positionen einzeln abschliessen, was er monierte.

Abbildung 8.2.: Der Mitarbeiter schliesst die Positionen ab, nachdem er sie aufgeladen hat.

Wagen abschliessen

Werden im Client alle Positionen eines Wagens abgeschlossen, kommt der Mitarbeiter wieder zurück auf die Wagen-Übersicht. Dies führte zu Verwirrung, da abgeschlossene Wagen standardmässig ausgeblendet werden. Denn der Mitarbeiter wollte am Ende nochmals überprüfen, ob alle Matratzen auf dem Wagen sind, wenn er diesen zur Endstation gebracht hat.

Fotos von Kernen

Als Hilfe für die Lager-Mitarbeiter haben diese das System eingeführt, dass sie spezielle Kerne fotografieren und diese Fotos unter dem Kernnamen abspeichern. Wissen sie dann nicht genau um welchen Kern es sich handelt, können sie diese Fotos anschauen.

Dies geschah beim Beladen des ersten Kerns. Der Mitarbeiter war bereits ans andere Ende des Lagers gefahren und musste dann nochmal zurück zum Computer um das Foto zu betrachten. Dies geschehe anscheinend des öfteren.

Lagerbestand

Da dieselben Kerne aus Platzgründen nicht immer beieinander eingeräumt werden können, kann es vorkommen, dass sich diese im Lager verteilen. Sind dann keine Kerne mehr vorhanden, wo sie eigentlich sein sollten, muss der Mitarbeiter zurück zum Computer und das ERP-System Movex abfragen, ob irgendwo im Lager noch solche Kerne sein sollten. Wenn dies der Fall ist, muss sich der Mitarbeiter auf die Suche den Kernen machen, da ihr Standort nicht erfasst ist.

Bedienung Tablet

Zu Beginn des Tests schaltete sich sehr schnell der Bildschirmschoner des Tablets ein. Deswegen wurde diese Sperrzeit auf eine grössere Zeitspanne von 15 Minuten festgelegt.

Zudem gab es einige Probleme, da sich die Applikation regelmässig rotierte. Die Rotation konnte allerdings auch in den Tablet-Einstellungen unterbunden werden.

Transport

Der Mitarbeiter stellte das Tablet vor sich auf sein Fahrzeug. Dabei wurde klar, dass eine Halterung zwingend notwendig sein wird.

Abbildung 8.3.: Lager Mitarbeit mit dem Tablet auf seinem Fahrzeug.

Spezialmasse

Die Matratzen mit Spezialmassen werden wie bisher farbig markiert. Der Mitarbeiter begrüsste dies und bereitete die Matratze entsprechend zu.

Abbildung 8.4.: Der Mitarbeiter klebt einen Matratzenkern, damit sie den Spezialmassen entspricht.

Listen der vergangenen Tage

Der Mitarbeiter erklärte, dass er jeweils zwei bis drei Produktionslisten der vergangenen Tage behält und mit sich führt. Dies tut er, da regelmässig Anfragen zu fehlenden Kernen aus anderen Bereichen kommen und er diese damit schneller beantworten kann.

W-Lan Verbindung

Das Lager hat eine sehr gute W-Lan Abdeckung und der Client verlor deswegen zu keinem Zeitpunkt die Verbindung.

8.5.2. Feedback der Testperson

Während und in einer Fragerunde nach dem Test wurde von der Testperson folgendes Feedback entgegengenommen.

Visualisierung Spezialmasse

Spezialmasse werden bereits farbig hinterlegt. Es wäre aber von Vorteil, wenn der Zuschnitt und das Kleben sich auch farblich unterscheiden würden.

Bedienbarkeit

Der Mitarbeiter war durchweg zufrieden mit der Bedienung des Clients. Allerdings äusserte er auch gewisse Skepsis gegenüber der Ablösung der Papierlisten durch Tablets.

Tablet

Dem Mitarbeiter gefiel das kleinere Tablet besser, da er es handlicher empfand und er durch die bessere Auflösung, die dargestellten Informationen besser lesen konnte.

8.6. Auswertung

Der User Acceptance Test war ein voller Erfolg. Der Mitarbeiter konnte den Client ohne eine Schulung bedienen und seine Arbeit damit verrichten.

Es zeigte sich allerdings, dass eine Konfrontation des Clients mit dem Endbenutzer schon eher hätte stattfinden müssen, damit sein Feedback früher einfliessen kann.

Die folgenden Punkte ergaben sich aus dem User Acceptance Test und müssen bei der weiteren Planung des Produkts diskutiert und in Betracht gezogen werden:

- Fotos von Kernen bei den Matratzen Informationen hinterlegen.
- Lagerbestand eines Kerns darstellen, damit der Mitarbeiter weiss, ob er danach suchen soll.
- Aufträge der vergangen Tage sollen abrufbar bleiben.
- Kleben und Zuschneiden bei Spezialmassen farblich differenzieren.
- Halterung für Wagen evaluieren.
- Abgeschlossene Wagen standardmässig ein- und nicht ausblenden.
- Gleiche Kerne eventuell zusammenführen und gemeinsam abschliessbar machen.
- Darstellen ob ein anderer Mitarbeiter Spezialmasse zubereiten muss, damit sich diese nicht gegenseitig blockieren.
- Es kann vorkommen, dass der Mitarbeiter anstatt des Kerns auf der Liste einen anderen Kern verwendet. Dies muss auf dem Client hinterlegt werden können.

Teil X.

Qualitätssicherung

9. Qualitätssicherung

9.1. Unit Tests

Beim Entwickeln des Servers wurde grossen Wert auf Unit Tests gelegt. Daraus resultieren 93 Tests welche insgesamt eine Laufzeit von 30 Sekunden haben.

Alle Tests die eine Datenbankverbindung benötigen erben von der `BaseDataBaseTest` Klasse, worin die Handhabung vereinfacht wird. Zudem wird vor jedem Test die Test-Datenbank neu initialisiert. Damit ist gewährleistet, dass die Tests unabhängig voneinander ausgeführt werden.

9.2. Development Workflow

Nachfolgend wird der festgelegte Entwicklungsablauf beschrieben, welcher eine hohe Code- Qualität gewährleistet.

Beispiel für die Entwicklung eines neuen Features:

1. Redmine Ticket wählen / erstellen
2. Aktuellen Master pullen
3. Branch mit Redmine Ticket Id + frei wählbaren Namen erstellen
4. Entwickeln
5. Änderungen pushen
6. Pull Request auf Github erstellen
7. Pull Request für Code Review an anderen Entwickler geben
8. Allfälliges Feedback verbessern
9. Branch mergen und löschen

9.3. Code Review und Refactoring

Bei Github hat man mit einem Pull Request eine gute Übersicht der Änderungen. Damit kann ein anderer Entwickler perfekt nachvollziehen was angepasst wurde. Zudem kann er die Änderungen kommentieren und damit Feedback geben und allfällige Änderungen vorschlagen.

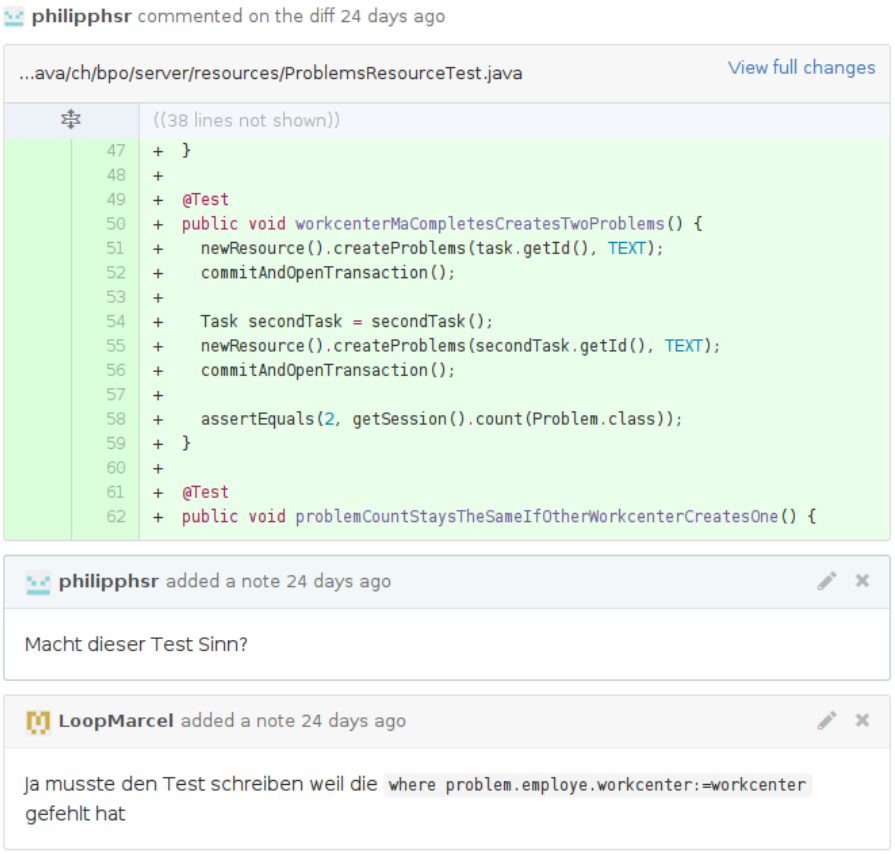


Abbildung 9.1.: Beispiel eines Code Reviews

9.4. DevOps

Da diese Arbeit drei Komponenten umfasst, wurde entschieden sich an der Entwickler-Methode DevOps zu orientieren. Dafür wurde für die Entwicklungsumgebung des Servers eine virtuelle Maschine mit Vagrant und Chef konfiguriert.

Bei Vagrant handelt es sich um ein Tool zum verwalten von virtuellen Maschinen. Dabei wird VirtualBox als Virtualisierungsebene eingesetzt. Zusätzlich kommt Chef zum einrichten der virtuellen Maschine zum Einsatz.

Im Rahmen dieses Projekts wird Vagrant und Chef mit der Datei 'Vagrantfile' konfiguriert. Darin werden die zu installierende Software, die Konfiguration der Software, sowie zwischen Host und virtueller Maschine geteilte Ports und Ordner, konfiguriert.

Daraus ergibt sich der Vorteil, dass jeder Entwickler in der selben Entwicklungsumgebung arbeitet. Der Entwickler holt sich einfach die aktuelle Version aus der Versionsverwaltung und startet die Vagrant-Maschine mit dem Befehl 'vagrant up'. Dadurch wird eine komplett neue virtuelle Maschine eingerichtet

und der Server darauf eingerichtet. Der Server ist dann über die IP-Adresse der virtuellen Maschine erreichbar.

Möchte nun der Entwickler beispielsweise nur am Client entwickeln, muss er nicht die komplette Entwicklungsumgebung für den Server starten, um diesen laufen zu lassen, sondern kann diesen einfach mit dem Befehl 'vagrant up' starten.

9.5. Continous Integration

Zur Continuous Integration wurde jenkins aufgesetzt. Da lediglich die Server-Komponente getestet ist, ist nur diese auf jenkins eingerichtet. Dies wurde so mit Github integriert, dass bei jedem neuen Pull Request, sowie Änderungen daran ein Build ausgeführt wird. Dabei gibt es eine Rückkopplung in Github, dass dort ersichtlich ist, ob bei den aktuellen Commits des Pull Requests der Build fehlerfrei durchläuft oder gewisse Tests nicht erfolgreich waren. [**?**]

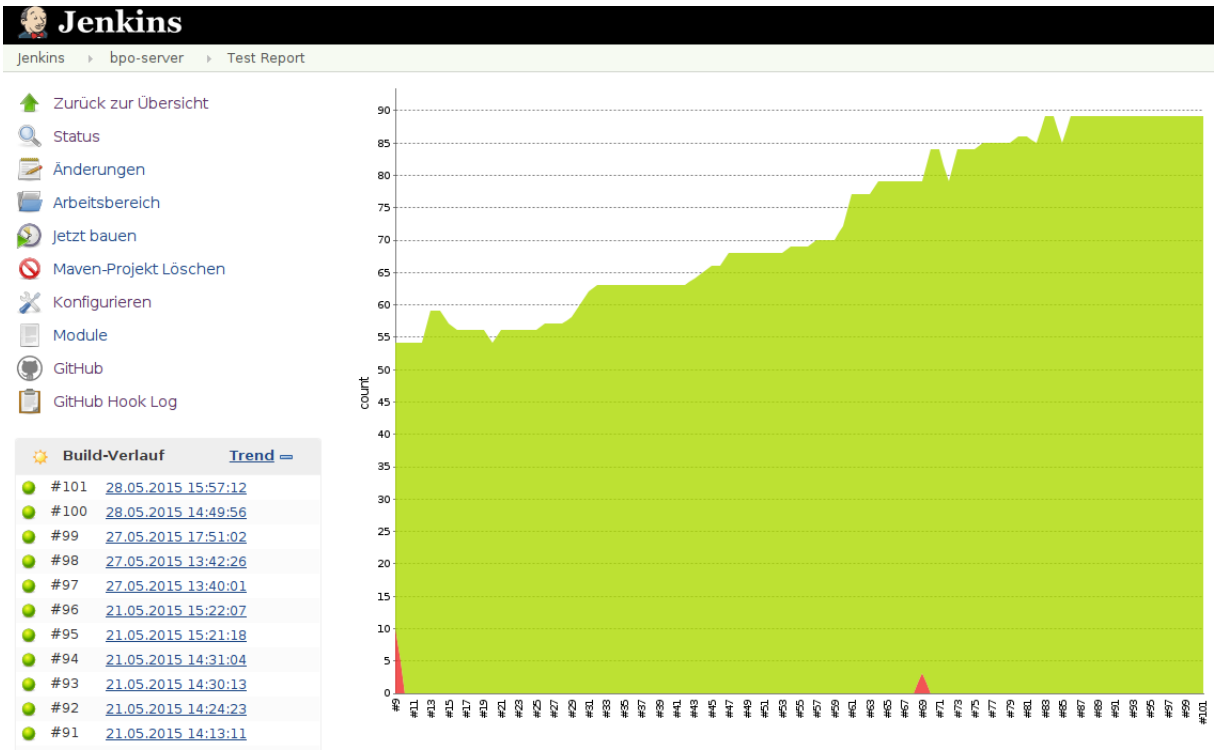


Abbildung 9.2.: Jenkins Build-Verlauf des Servers

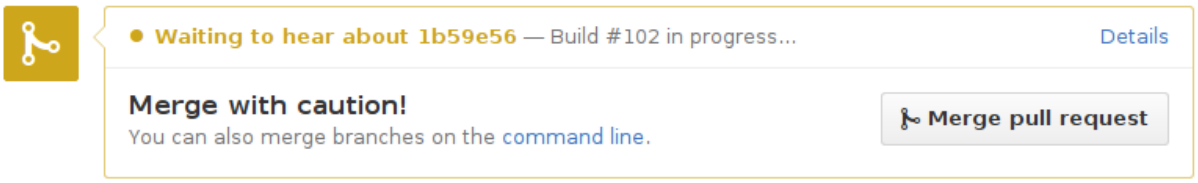


Abbildung 9.3.: Beispiel für Integration von jenkins in Github

9.6. STAN

Für das Visualisieren der Abhängigkeiten wird das Analyse-Tool STAN für Eclipse verwendet. Analysiert man damit die Abhängigkeiten der Server-Komponente, erhält man folgendes Resultat: [**?**]

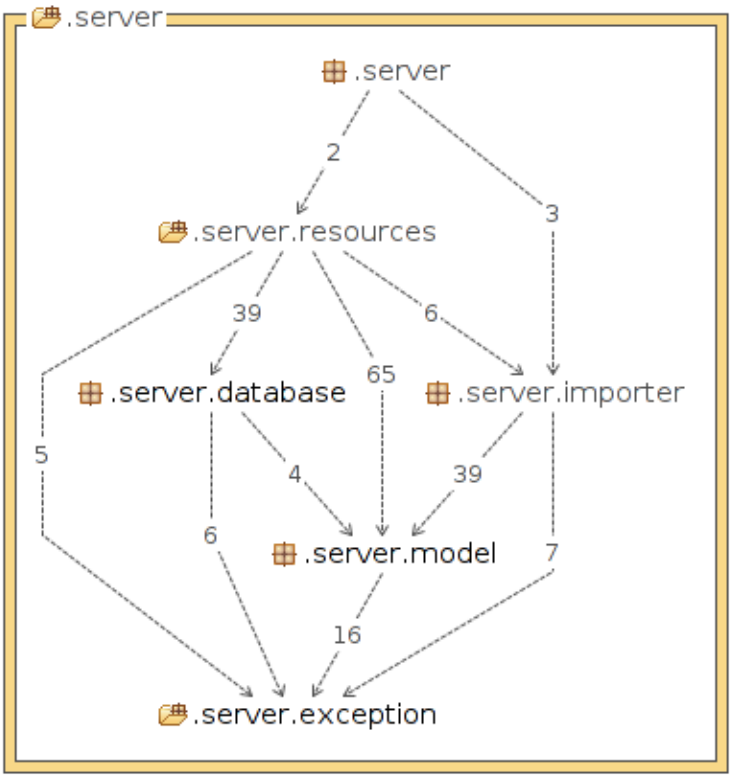


Abbildung 9.4.: Package Abhängigkeiten des Servers

Wie man sieht, weisst die Auswertung von STAN keine Mängel auf.

9.7. Metriken

Die Auswertungen der Metriken ergeben folgende Resultate:

Wert / Komponente	**Server**	**Client**	**Cockpit**
Anzahl Klassen	79	65	-
Anzahl Methoden	359	-	-
Anzahl Packages	15	7	-
Anzahl Zeilen Code	3021	2261	532

Tabelle 9.1.: Metriken der Komponenten

Die Auswertungen umfassen ebenfalls den Test-Code.

Wie man sehen kann hat der Client fast gleich viele Codezeilen wie der Server, obwohl dieser weniger Logik beinhaltet. Dies liegt daran, dass der GUI-Code von Android sehr umfangreich ist.

9.8. Styleguide

Um einen einheitlichen Code-Style zu gewährleisten, wurde das Eclipse Plugin Checkstyle verwendet. Es überprüft den geschrieben Code und generiert Warnungen, wenn dieser nicht dem konfigurierten Styleguide entspricht. Im Rahmen dieses Projekts wurde der Java-Styleguide verwendet. [**?**]

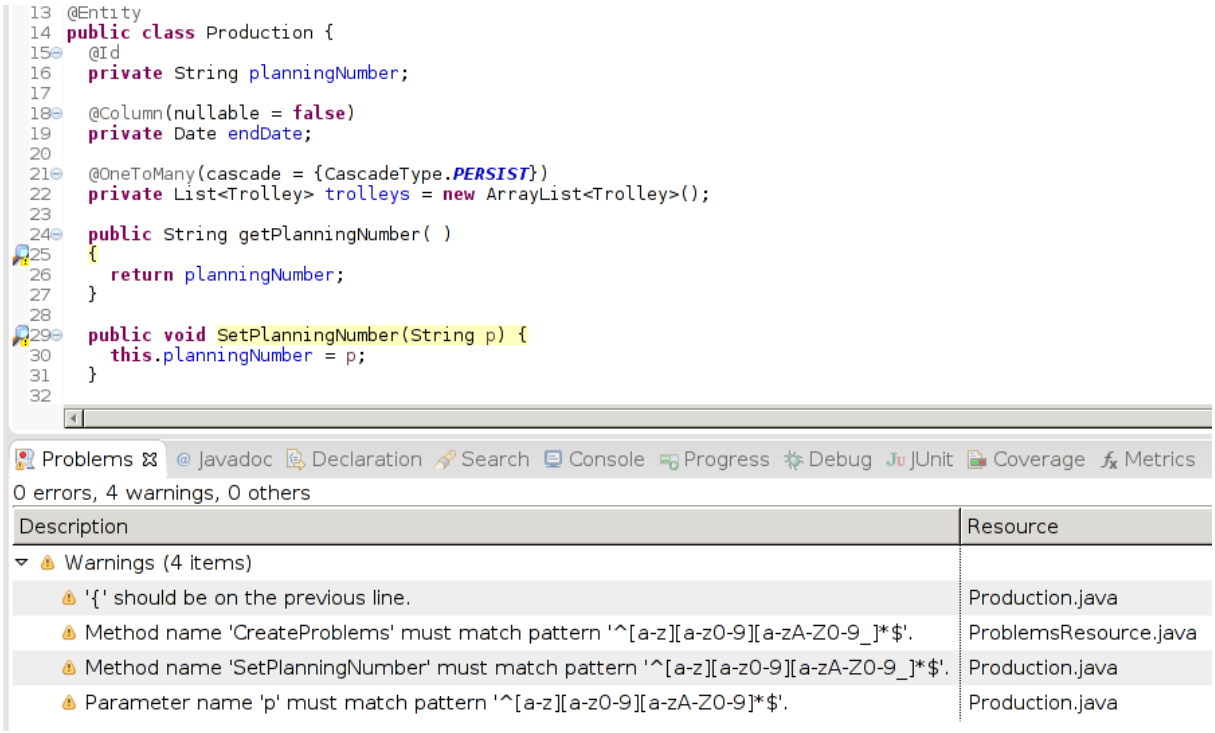


Abbildung 9.5.: Beispiel für Warnungen von Checkstyle

Teil XI.

Erreichte Ziele / Offene Punkte

10. Erreichte Ziele / Offene Punkte

In diesem Kapitel wird festgehalten in welchem Umfang die spezifizierten Anforderungen erfüllt wurden und welche Punkte noch offen sind.

10.1. Erreichte Ziele

10.1.1. Use Cases

Die folgenden Use Cases waren geplant und wurden erfolgreich implementiert:

- Authentisieren
- Aufträge importieren
- Produktionsstatus abrufen
- Aufträge abrufen
- Aufträge übernehmen
- Aufträge abschliessen

Zusätzlich wurde der folgende optionale Use Case implementiert:

- Problem melden

10.1.2. Nichtfunktionale Anforderungen

10.1.2.1. Richtigkeit

Anforderung: Es wird gewährleistet, dass immer die aktuellen Daten angezeigt werden. Ein Workcenter-Mitarbeiter oder auch der Meister muss innerhalb maximal drei Sekunden sehen, wenn eine Änderung vorgenommen wurde. So kann beispielsweise eine doppelte Übernahme eines Wagens verhindert werden. Wird trotzdem zur selben Zeit die gleiche Aktion von zwei Mitarbeiter ausgeführt, muss dies abgefangen werden und der eine Mitarbeiter darüber informiert werden.

Umsetzung: Jeder Mitarbeiter sieht innert maximal drei Sekunden den aktuellsten Stand. Auch im Cockpit werden Änderungen entsprechend aktualisiert. Versuchen zwei Mitarbeiter gleichzeitig einen Wagen zu übernehmen wird das entsprechend behandelt.

10.1.2.2. Interoperabilität

Anforderung: Die Daten für die Produktion werden aus dem ERP System Movex geholt. Die Software muss also Daten aus Movex lesen können. Das Cockpit muss mit einem Windows Rechner bedient werden können. Die Software für die Workcenter-Mitarbeiter mit einem Tablet.

Umsetzung: Das Cockpit wurde als Web-Applikation entwickelt, welche mit jedem verbreiteten Browser aufgerufen werden kann. Der Client ist als Android Applikation realisiert.

Im Rahmen dieser Arbeit wurde keine direkte Verbindung zum Movex System implementiert, da die Daten aus einer Csv-Datei importiert werden. Die Architektur ist allerdings so entworfen, dass eine Wechsel auf den direkten Movex-Zugriff kein Problem darstellt.

10.1.2.3. Sicherheit

Anforderung: Der Zugriff auf das System erfordert eine erfolgreiche Identifikation und Authentifikation. Der Aktor ist aufgrund seiner Funktion authorisiert seine Aufgaben zu erledigen. Das heisst, ein Workcenter- Mitarbeiter ist beispielsweise nicht berechtigt einen Benutzer im System anzulegen.

Umsetzung: Jeder Request auf den Server muss authentisiert sein. Ist dies nicht der Fall, wird ein entsprechender HTML Status-Code zurückgegeben. Dasselbe gilt für die Aktionen eines Benutzers.

10.1.2.4. Fehlertoleranz

Anforderung: Das System stürzt nicht aufgrund eines Fehlers ab. Fehler werden abgefangen und der Benutzer entsprechend benachrichtigt. Tritt ein Fehler in der Business Logik auf, so wird dieser geloggt. Exceptions werden mit einem passenden HTML Status Code zurückgemeldet.

Umsetzung: Fehler auf dem Server werden geloggt und auf die jeweiligen Status Codes gemappt. Der Client weiss ebenfalls mit Fehlern umzugehen und informiert entsprechend den Benutzer.

10.1.2.5. Erlernbarkeit

Anforderung: Der Benutzer versteht das Konzept und die Bedienung nach einer Mitarbeiter-Schulung.

Umsetzung: Im Rahmen des User Acceptance Test wurde bestätigt, dass der Client grossteils selbsterklärend ist und intuitiv bedient werden kann. Spezialfunktionen können in einer kurzen Schulung beigebracht werden.

10.1.2.6. Bedienbarkeit

Anforderung: Der Workcenter-Mitarbeiter kommt innert maximal drei Schritten zum Ziel. Das User Interface soll nur die nötigen Informationen anzeigen.

Umsetzung: Die dargestellten Informationen wurden, in Absprache mit dem Endbenutzer, auf ein Minimum reduziert. Die Hauptaufgaben kann der Benutzer innert drei Schritten erledigen.

10.1.2.7. Anpassbarkeit

Anforderung: Der Serverteil soll auf jedem Java Applikationsserver laufen können. Das Cockpit ist mit einem Browser aufrufbar. Die Software für den Workcenter-Mitarbeiter setzt ein Tablet voraus mit einem Android- Betriebssystem.

Umsetzung: Diese Anforderungen sind erfüllt.

10.2. Offene Punkte GoLive

Bei der Umsetzung im Rahmen dieser Arbeit handelt es sich um einen Prototyp. Nachfolgend werden die offenen Punkte aufgelistet, die noch notwendig sind um das Produkt im produktiven Alltag einzusetzen. Zudem wird eine Schätzung abgegeben, wie viel Aufwand die Umsetzung zur Folge hat. Es ist zu ergänzen, dass die nachfolgenden Schätzungen keine Durchführung von weitere User Acceptance Tests und Umsetzung der gewonnen Erkenntnisse umfassen. Dies wäre allerdings zur Vollendung des Systems, für den produktiven Einsatz unabdingbar.

10.2.1. Anbindung an Movex

Beschreibung: Die Daten wurden bisher aus einer CSV-Datei importiert. Diese müssen in Zukunft direkt aus dem ERP System Movex abgerufen werden. Dabei kann die benötigte SQL-Abfrage aus der bestehenden Excel-Datei ausgelesen werden.

Geschätzter Aufwand: 2 Tag

10.2.2. Zuordnungsalgorithmus der Matratzen

Beschreibung: Da die Matratzen bisher bereits sortiert aus einer CSV-Datei importiert werden, ist der Zuordnungsalgorithmus, der die Matratzen den Wagen zuordnet, noch nicht umgesetzt.

Der Algorithmus muss modular aufgebaut werden, dass dieser jederzeit erweitert werden kann. Dabei kann er stark an den aktuellen Algorithmus angelehnt werden, der sich aus der Excel-Datei extrahieren lässt.

Geschätzter Aufwand: 5 Tage

10.2.3. Etiketten exportieren

Beschreibung: Aus den aus Movex importierten Daten müssen die Etiketten exportiert werden können. Diese werden im CSV-Format an den Etiketten-Drucker weitergegeben. Das Format ist definiert.

Geschätzter Aufwand: 2 Tage

10.2.4. Benutzer verwalten

Beschreibung: Die Benutzer wurden bisher fest in der Datenbank abgelegt. Da diese flexibel angelegt werden müssen, soll im Cockpit eine einfache Benutzerverwaltung umgesetzt werden. Die Benutzer können dann einem Workcenter zugeordnet werden.

Auch eine Anbindung an das bestehende LDAP-System wäre denkbar.

Geschätzter Aufwand: 3 Tage

10.2.5. Offline Betrieb

Beschreibung: Aktuell ist der Client auf eine dauerhafte W-Lan Verbindung angewiesen. Dieser sollte auch offline bedienbar sein und die Anfragen an den Server schicken, sobald die W-Lan Verbindung wieder verfügbar ist.

Geschätzter Aufwand: 3 Tage

10.3. Zukünftige Use Cases

Die folgenden Use Cases sind für eine zeitnahe Umsetzung geeignet und würden einen Mehrwert der Lösung gegenüber der bisherigen Papier-Lösung bieten.

10.3.1. Fotos auf Matratzen-Kernen hinterlegen

Damit die Lager-Mitarbeiter die Matratzen-Kerne besser finden, sollten Fotos hinterlegt werden können.

10.3.2. Auftrag annullieren

Der Meister kann einen Auftrag oder Position annullieren, woraufhin dieser nicht mehr in den Produktionslisten angezeigt wird.

10.3.3. Auftrag priorisieren

Der Meister kann Aufträge priorisieren, damit diese schneller abgearbeitet werden.

10.3.4. Problem melden

Hat ein Mitarbeiter ein Problem bei seinem aktuellen Arbeitsschritt (beispielsweise Material nicht vorhanden, Maschine defekt, usw.) kann er dies über das System melden, woraufhin der Meister benachrichtigt wird.

10.3.5. Rückmeldung in ERP

Wenn ein Auftrag komplett abgeschlossen wurde, soll dies im System erfasst werden können, woraufhin das System die ins ERP schreibt. Ausgenommen sind dabei Problemfälle.

10.3.6. Wegoptimierung im Lager

Um Zeit zu sparen beim Palett vorbereiten bzw. liefern, soll der Weg der die Lager-Mitarbeiter fahren optimiert werden.

10.3.7. Auswertung der Mitarbeiter-Aktivitäten

Da die Mitarbeiter erfassen welche Aufträge sie übernehmen und wann sie diese abschliessen, können Auswertungen erstellt werden, was sie am Tag leisten.

10.3.8. Authentisierung (via LDAP)

Jeder Benutzer des System kann sich mit seinen Benutzerdaten identifizieren. Die Authentisierung wird dann von LDAP bzw. Active Directory übernommen.

10.3.9. Report erstellen

Am Ende eines Produktionstag wird ein Report generiert, welcher die im System gemachten Annullierungen und Probleme auflistet.

10.3.10. Annullierungen aus ERP

Wird eine Annullierung im ERP System vorgenommen, so wird das System automatisch auf darüber informiert.

10.4. Verbesserungsvorschläge

Um den Prototyp besser für den produktiven Alltag einsetzbar zu machen, wäre es sinnvoll folgende Verbesserungen zu implementieren.

10.4.1. Einsatz von Sockets

Der Client, wie auch das Cockpit fragen bisher alle drei Sekunden den Server nach neuen Daten ab. Es wäre performance-technisch sinnvoller Sockets einzuführen, damit der Server den Client bei Änderungen informieren kann.

10.4.2. Tests für den Client

Die Server Komponente ist sehr gut getestet. Der Client hat allerdings keine Tests. Da der Client weiter anwachsen wird, wäre es sinnvoll so bald wie möglich Tests für den Client einzuführen.

10.4.3. Migrationskonzept für die Datenbank

Es wird ein Konzept nötig sein, wie Migrationen des Datenbankschema bei Änderungen am Model durchgeführt werden.

Teil XII.

Inbetriebnahme

11. Inbetriebnahme

11.1. Server

11.1.1. Voraussetzungen

Um die Server Komponente zu deployen wird folgende Software vorausgesetzt:

- Java 7, https://java.com/en/
- Tomcat 7, http://tomcat.apache.org/
- PostgreSQL, http://www.postgresql.org/

11.1.2. Konfiguration

Nachfolgend werden die Dateien beschrieben, mit welchen der Server konfiguriert werden kann. Diese befinden sich im Verzeichnis 'src/main/resources'. Für die Test-Umgebung gibt es jeweils ein Pendant im Verzeichnis 'src/test/resources'.

hibernate.cfg.xml

Hier wird die Datenbank, sowie der Benutzer für den Datenbankzugriff konfiguriert.

Folgende Zeilen sind die wichtigsten:

```
<property name="hibernate.connection.driver_class">org.postgresql.Driver</property>
<property name="hibernate.connection.username">bpo-server</property>
<property name="hibernate.connection.password">password</property>
<property name="hibernate.connection.url">jdbc:postgresql://localhost:5432/bpo</property>
```

Listing 11.1: Ausschnitt aus hibernate.cfg.xml

Wobei grundsätzlich nur der Username und das Passwort angepasst werden sollten.

log4j.xml

Hier wird konfiguriert, wohin Log-Meldungen geschrieben werden sollen.

In der Standardkonfiguration wird in das Verzeichnis '/var/log/bpo-server/' geloggt.

```
<param name="File" value="/var/log/bpo-server/full.debug.log" />
```

Listing 11.2: Ausschnitt aus log4j.xml

Dies kann bei Bedarf angepasst werden. Ansonsten muss darauf geachtet werden, dass das Verzeichnis existiert.

11.1.3. Vagrant

Vagrant wird verwendet, damit eine einheitliche Entwicklungsumgebung zur Verfügung steht. Ausserdem können die Tests in die, mit Vagrant erstellte, Datenbank schreiben.

Vagrant erstellt eine virtuelle Maschine und richtet diese mit dem Provisioning Tool Chef ein (siehe Qualitätssicherung für eine detaillierte Beschreibung). Vagrant wird in der Datei 'Vagrantfile' konfiguriert.

Um Vagrant startet zu können müssen folgende Schritte ausgeführt werden (Auszug aus Readme von Server-Komponente):

```
Install Virtualbox:
sudo apt-get install virtualbox

Install Vagrant:
sudo apt-get install vagrant

Install chef as provisioner:
http://stackful-dev.com/three-ways-to-get-chef-chef-solo-installed-on-your-↙
    server.html

Install the chef-dk for more chef:
wget https://opscode-omnibus-packages.s3.amazonaws.com/ubuntu/12.04/x86_64/↙
    chefdk_0.4.0-1_amd64.deb
sudo dpkg -i chefdk_0.4.0-1_amd64.deb

Install following Vagrant plugins:
vagrant plugin install vagrant-berkshelf
vagrant plugin install vagrant-triggers
vagrant plugin install vagrant-hostmanager

And start the machine:
vagrant up
```

Listing 11.3: Schritte für die Installation von Vagrant und Chef

Damit wird eine virtuelle Maschine mit Java, Tomcat und PostgreSQL eingerichtet. Zudem werden die jeweiligen Ports auf die selben Ports der Host-Maschine gemappt. Damit kann man jeweils über die localhost-URL auf die jeweiligen Services zugreifen.

11.1.4. Deployen

Java Applikation werden als war-Dateien auf dem Server deployet. Dieses kann erstellt werden, indem ein Maven Build ausgeführt wird. Dafür navigiert man in das Verzeichnis des Servers und führt folgenden Befehl aus:

```
mvn install
```

Listing 11.4: Befehl um den Maven Build zu starten

Das setzt ein installiertes Maven voraus (https://maven.apache.org/).

Während des Maven Builds werden die Tests ausgeführt, die einen Datenbank Zugriff benötigen. Dieser kann entweder in der Datei 'src/test/resources/hibernate.cfg.xml' umkonfiguriert werden (siehe Konfiguration) oder vor dem Ausführen des Maven Builds kann die Vagrant Maschine gestartet werden (siehe Vagrant).

Nach einer erfolgreichen Ausführung des Maven Builds produziert Maven die Datei 'target/bpo-server-version.war'. Diese kann nun in das entsprechende Verzeichnis von Tomcat verschoben werden, womit die Applikation deployet werden sollte.

War das Deployment erfolgreich, ist die Applikation über die entsprechende URL des Servers erreichbar (Beispielsweise: 'http://localhost:8080/bpo-server/').

11.2. Cockpit

11.2.1. Konfiguration

Der einzige Konfigurationsparameter ist die URL des Servers, von welchem die Informationen abgerufen werden. Dies muss in den Dateien '*resource.js' vorgenommen werden.

11.2.2. Deployment

Um ein frisch ausgechecktes Repository des Cockpits zu starten, müssen folgende Schritte ausgeführt werden:

```
npm install
bower install
http-server -a localhost -p 3000 app/
```

Listing 11.5: Schritte zum Bereitstellen des Cockpits

Dafür müssen npm (https://www.npmjs.com/) und bower (http://bower.io/) installiert sein.

11.3. Client

11.3.1. Entwicklungsumgebung

Für die Entwicklung des Android Clients wird die offizielle Android IDE 'Android Studio' empfohlen. Android Studio kann unter folgendem Link heruntergeladen werden:

https://developer.android.com/sdk/index.html

11.3.2. Deployment

Android Studio verwendet 'gradle' als Build Management Tool. Beim Starten der Applikation wird die '.apk' direkt auf ein virtuelles oder physisches Device installiert. Der Build kann aber auch manuell über die Konsole gestartet werden. Eine detailierte Anleitung findet man unter: 'http://developer.android.com/tools/building/building-cmdline.html#ManualReleaseMode'

Beachte

Der Android Emulator, welcher eine virtuelle Maschine als Device erstellt, kann nicht parallel mit Vagrant verwendet werden. Versucht man dennoch den Android Emulator zu starten erscheint folgende Fehlermeldung:

```
ioctl(KVM_CREATE_VM) failed: 16 Device or resource busy
ko:failed to initialize KVM
```

Listing 11.6: Fehlermeldung wenn der Android Emulator nebst Vagrant gestartet wird

Deswegen empfiehlt es sich direkt auf einem Tablet zu entwickeln, was auch performanter ist.