Lucerne University of
Applied Sciences and Arts

HOCHSCHULE
LUZERN

Modul APPE | TA.APPE.F1101

APPE Filialen-Bestellsystem

Systemspezifikation

Horw, 03.06.2011

Projekt	APPE Filialen-Bestellsystem
Dokument	Systemspezifikation
Schule	Hochschule Luzern, Technik & Architektur
Modul	TA.APPE.F1101
Projektteam	**Galliker Thomas** Studiengang Informatik (BB) Panorama 6123 Geiss Tel. +41 79 504 80 70 thomas.galliker@stud.hslu.ch **Gasser Martin** Studiengang Informatik (BB) Mettenwilhöhe 4 6275 Ballwil Tel. +41 79 755 04 32 martin.gasser@stud.hslu.ch **Stocker Elias** Studiengang Informatik (BB) Schybenacherweg 12 6285 Hitzkirch Tel. +41 79 603 10 89 elias.stocker@stud.hslu.ch
Dozenten	**Prof. Gisler Roland, Prof. Hofstetter Jörg, Prof. Olnhoff Thomas**
Letzte Änderung	3. Juni 2011, 11:10:00 Uhr

Änderungsprotokoll

Version	Datum	Autor	Beschreibung
0.1	17.03.2011	gat	Initialversion von Vorlage erstellt
0.2	10.03.2011	gat	Einleitung und Systemübersicht
0.3	22.03.2011	gat	Datenstrukturen dokumentiert
0.4	07.04.2011	gam	Diagramme aktualisiert und Dokument überarbeitet.
0.5	08.04.2011	gam	Review und kleinere Korrekturen
0.6	27.04.2011	gat	Logon Prozess inkl. Flussdiagramm, Event Notification
0.7	28.04.2011	ste/gat	Komponenten- und Klassendiagramm
0.8	28.04.2011	gat	Beschreibung des Leistungsumfangs der Schichten
0.9	29.04.2011	gam	Detailliertere Beschreibung des DB-Schemas
0.10	01.05.2011	gat	Beschreibung der Schnittstellen, Klassen
0.11	01.05.2011	gam	Generelle Überarbeitung des Dokuments
0.12	01.05.2011	gat	Neues Kapitel: Qualitätsmerkmale
0.13	01.05.2011	gam	Review für die Abgabe
0.14	19.05.2011	gat	Design Entscheide: Autorisierung und Verschlüsselung
0.15	20.05.2011	ste	Datenstruktur aufgetrennt in einzelne Diagramme
0.16	23.05.2011	gat	Verteilungsansicht (Diagramm) erstellt
0.17	25.05.2011	gat	Design Entscheid: Verwaltung von Konfigurationsdateien
0.18	26.05.2011	gam	Überarbeiten und Erweitern des Kapitels zur Transaktionssicherheit.
0.19	30.05.2011	gat	Beschreibung der Konfigurationsparameter

Inhalt

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Ziel & Zweck dieses Dokuments

In diesem Dokument ist der Aufbau des Fillial-Bestellsystems (fbs) und dessen Umsysteme beschrieben. Die Systemspezifikation enthält die Softwarearchitektur und beschreibt grundsätzlich **wie** die Software aufgebaut ist. Die Beschreibung **was** das System macht ist im Dokument Kundenanforderungen beschrieben.

1.2 Begriffe & Abkürzungen

Abkürzung	**Erklärung**
HSLU	Hochschule Luzern
APPE	"Applikationsentwicklung"; Modulbezeichnung HSLU
gat	Namenskürzel für Galliker Thomas
ste	Namenskürzel für Stocker Elias
gam	Namenskürzel für Gasser Martin
STASS	Steuerungsausschuss
fbs	**F**illial-**B**estell**s**ystem
DTO	Data Transfer Objects: Objekte für den Datenaustausch zwischen der Businesslogik und der Präsentationsschicht.
Entity	Datenbank Entität bzw. ein Abbild davon, welches in Form einer Klasse existiert
ER Diagram	Entity-Relationship Diagram; zeigt die Beziehungen zwischen Entitäten auf
TLS	Transport Layer Security; Verschlüsselungsprotokoll; weitläufiger bekannt unter der Vorgängerbezeichnung Secure Sockets Layer (SSL); siehe Quelle [10]

Tabelle 1: Abkürzungserklärungen

2 Softwarearchitektur

In diesem Kapitel ist der Aufbau der Applikation beschrieben. Dazu gehören unter anderem eine grobe Systemübersicht, die internen und externen Schnittstellen sowie das Datenmodell.

2.1 Systemübersicht

Das vorliegende System besteht aus einer 3-Layer Architektur. Jede Schicht kann auf die darunterliegende Schicht zugreifen. Zusätzlich gibt es noch ein Common-Projekt für gemeinsamen Code. Unter anderem sind dort die Entities abgelegt.

Der Presentation Layer ist zuständig für die Darstellung sämtlicher Ein-/Ausgabemasken: Von Authentifizierungsmasken bis hin zu Bestelllisten oder Artikellisten. Die Präsentationsschicht kommuniziert direkt mit definierten Funktionen der Logikschicht. Sämtliche Geschäftsprozesse werden in der Logikschicht implementiert. Für die Persistenz der Nutzdaten liegt unter der Logikschicht eine Datenschicht. Allfällige Lese-/Schreiboperationen werden direkt von der Logikschicht an die Datenschicht weitergegeben.

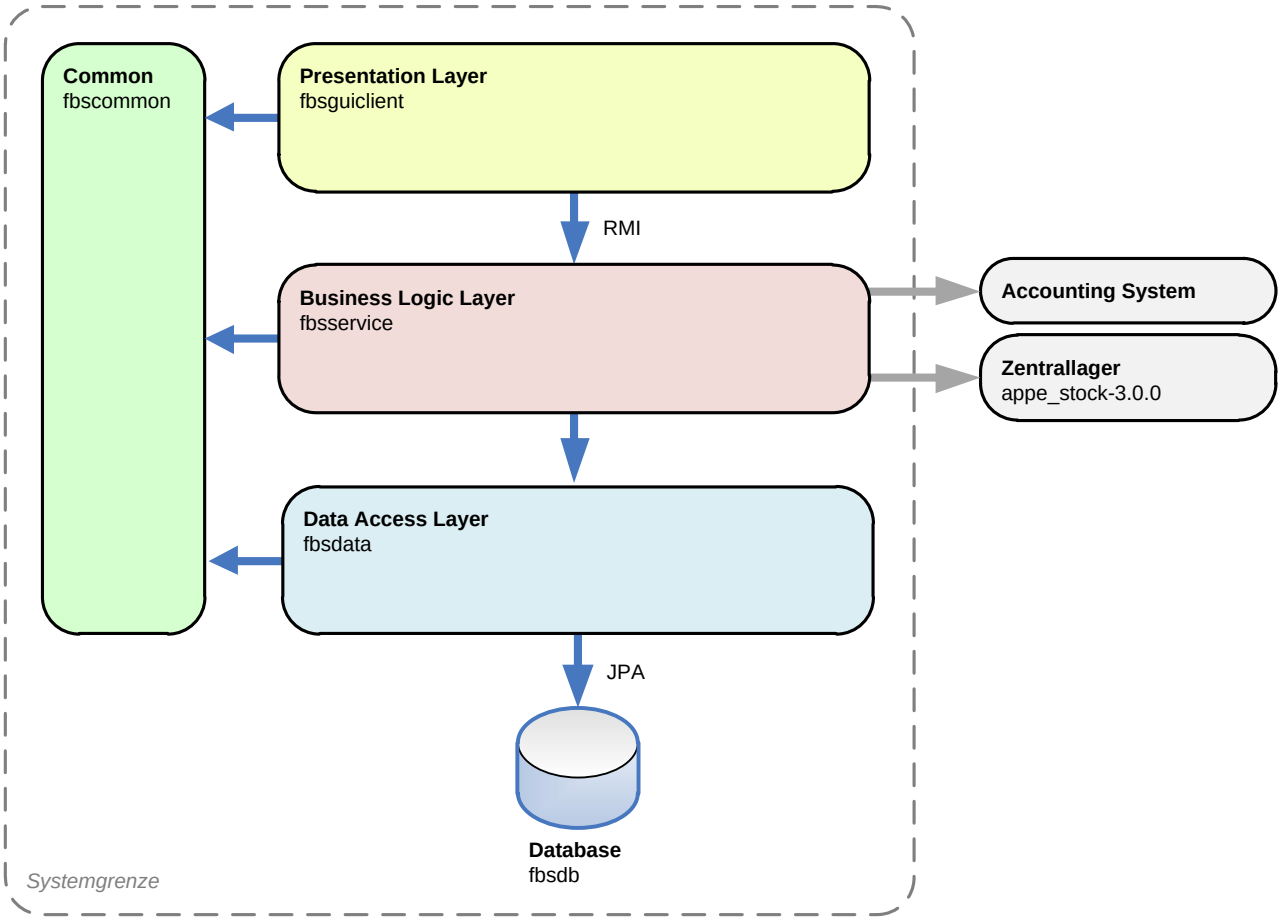

Abbildung 1: Systemübersicht

2.1.1 Beschreibung der Schichten

Presentation Layer fbsguiclient	Die Präsentationsschicht ist die oberste Schicht in der vorliegenden 3-Tier Architektur. Sie stellt die eigentliche Benutzerschnittstelle dar und ermöglicht so einem Benutzer die Interaktion mit dem System. Nach dem der Benutzer sich über das Logon-Form authentifiziert hat, kann er Produktlisten anzeigen lassen, Produkte auswählen und mit einem Klick in den Warenkorb legen. Ferner kann er Warenkorbinhalte in eine Bestellung überführen oder bereits abgeschlossene Bestellungen einsehen. Die Präsentationsschicht nutzt Remote Method Invocation (RMI) um die Kommunikation mit der Business Logik sicherzustellen. Zwischen diesen beiden Schichten werden Datentransferobjekte (DTO) ausgetauscht. Hauptschwerpunkte des Presentation Layers bilden die Kriterien der Software Ergonomie [2]: Aufgabenangemessenheit, Selbstbeschreibungsfähigkeit, Steuerbarkeit, Erwartungskonformität, Fehlertoleranz, Individualisierbarkeit und Lernförderlichkeit.
Business Logic Layer fbsservice	Die Business-Logik-Schicht ist verantwortlich für das korrekte Abarbeiten von Aufträgen. Darunter fallen beispielweise das Füllen des Warenkorbes mit Produkten für einen bestimmten Benutzer, das Umwandeln eines Warenkorbs in eine Bestellung sowie das automatische Nachbestellen von Produkten, deren Mindestlagermenge unterschritten wurde. Die Entkopplung der Geschäftslogik von der Präsentationsschicht auf einen zentralen Dienst macht insbesondere dann Sinn, wenn Anpassungen und Korrekturen an Geschäftsprozessen erwartet werden. Der Business Logic Layer stellt des Weiteren auch Werkzeuge bereit, die zur Umwandlung von Entity-Objekten (aus dem Data Access Layer) und Data Transfer Objects (für die Weitergabe an den Presentation Layer) verwendet werden können.
Data Access Layer fbsdata	Die Datenzugriffsschicht ist der 3. Layer, befindet sich aber zusammen mit dem Business Logic Layer auf dem 2. Tier. Hier werden Klassen bereitgestellt, welche die Datenbank abbilden. Zudem stehen Methoden bereit, welche zum persistieren von Objekten genutzt werden können. Die Verbindung zur darunterliegenden Datenbank wird mit JPA (Java Persistance API) realisiert.
Common fbscommon	Common beherbergt gemeinsamen Code welcher sowohl von der 1. Und der 2. Schicht (Präsentation und Geschäftslogik) genutzt werden. Darunter fallen beispielweise die Schnittstellen-Interfaces.
Database fbsdb	Das 3. Tier ist die Datenbank. Als Datengrundlage wird ein relationales Datenbanksystem verwendet. Das verwendete Produkt ist MySQL 5 mit der Engine InnoDB. Das eingesetzte Schema wird „fbsdb“ benannt und unterstützt das Charset „utf8“.

2.1 Externe Komponenten

Die Applikation verwendet zwei externe Komponenten welche nachfolgend beschrieben sind.

2.1.1 Accounting System

Hinter der Accounting Engine verbirgt sich ein externes Invoicing & Payment System, welches mitunter für das Rechnungswesen, Sales/Order Reporting sowie den Warenaustausch mit Handelspartnern genutzt wird. Die Umsetzung dieser Schnittstelle erfolgt nicht im Rahmen dieses Projekts. Es ist jedoch darauf zu achten, dass die Daten, die dafür benötigt werden, vorhanden sind. Zu einem späteren Zeitpunkt wird ein Service Implementiert, welcher die Schnittstelle umsetzt.
Es ist angedacht, dass der Service entweder direkt auf die DB (fbsdb) zugreifen wird oder mit dem Business-Tier kommuniziert. Die Entscheidung für eine der beiden Varianten wird gefällt, sobald weitere Details über die Schnittstelle bekannt sind.

2.1.2 Komponente Zentrallager (APPE Stock Library 3.0.0)

Diese Komponente wird für Nachbestellungen aus dem Zentrallager verwendet. Die Komponente für das Zentrallager ist ein Legacy-System, welches jedoch (vorerst) nicht abgelöst werden soll. Die APPE Stock Library ist eine Java-Komponente. Es ist also kein Interop (via JNI) nötig.

2.2 Interne Komponenten

Der Java-Code wird gemäss der Softwarearchitektur in Komponenten aufgeteilt. Dadurch ergibt sich nachfolgendes Komponentendiagramm. Die farbliche Markierung der Komponenten korrespondiert mit dem Aufbau des Systems auf Seite 5.

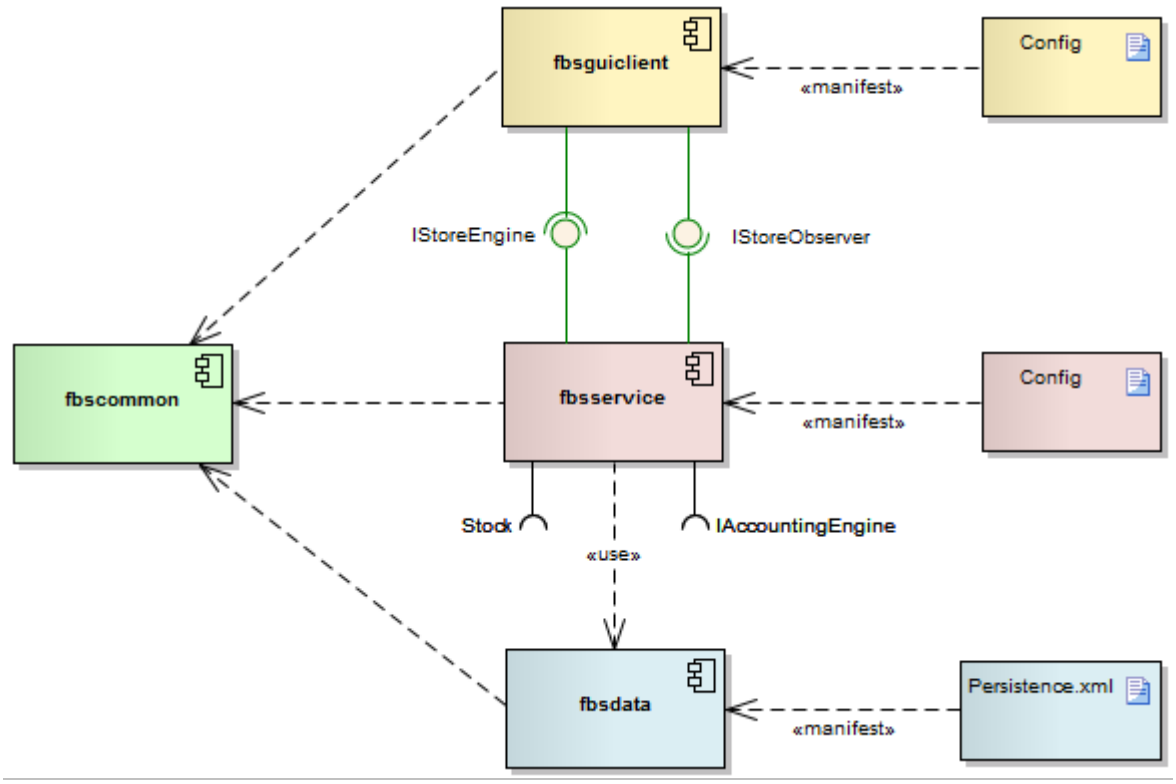

Abbildung 2: Komponentendiagramm

2.2.1 Kommunikation zwischen den Komponenten

PL→BLL	Die Kommunikation wird mittels Remote Method Invocation (RMI) implementiert. RMI basiert auf dem RPC-Prinzip und nutzt TCP/IP als Verbindungsprotokoll. Die für die Entkopplung genutzten Interfaces befinden sich in fbscommon. Data Transfer Objects (DTO) werden als Informationsträger zwischen dem Presentation und dem Business Logic Layer ausgetauscht.
BLL→DAL	Die beiden Schichten werden physisch auf demselben System betrieben, deshalb ist es nicht nötig, eine Kommunikationsverbindung einzurichten.
DAL→DB	Der Data Access Layer nutzt die Java Persistance API (JPA) um Informationen zwischen Domain Objects (Entity Klassen) und den Datenbank Tabellen auszutauschen.

Siehe auch 2.6 Spezifikation der Schnittstellen (Seite 16).

2.3 Die wichtigsten Klassen im Überblick

In der nachfolgenden Abbildung wird das Zusammenspiel von Klassen demonstrativ dargestellt. Der Umfang wurde absichtlich reduziert, um die Verständlichkeit zu erhöhen. Viele Klassen (wie beispielsweise die Provider-Klassen von fbsservice oder die Entity-Klassen in fbsdata) wurden komplett ausgeblendet. Die farbliche Markierung der Klassen korrespondiert mit dem strukturellen Aufbau des Systems auf Seite **Fehler! Textmarke nicht definiert.** dieses Dokuments.

Für Logik, welche über RMI der Präsentationsschicht zur Verfügung gestellt wird, gibt es jeweils eine Klasse in fbsservice sowie ein oder mehrere Interfaces in fbscommon. Die RMI-Interfaces haben alle das „I“-Präfix.

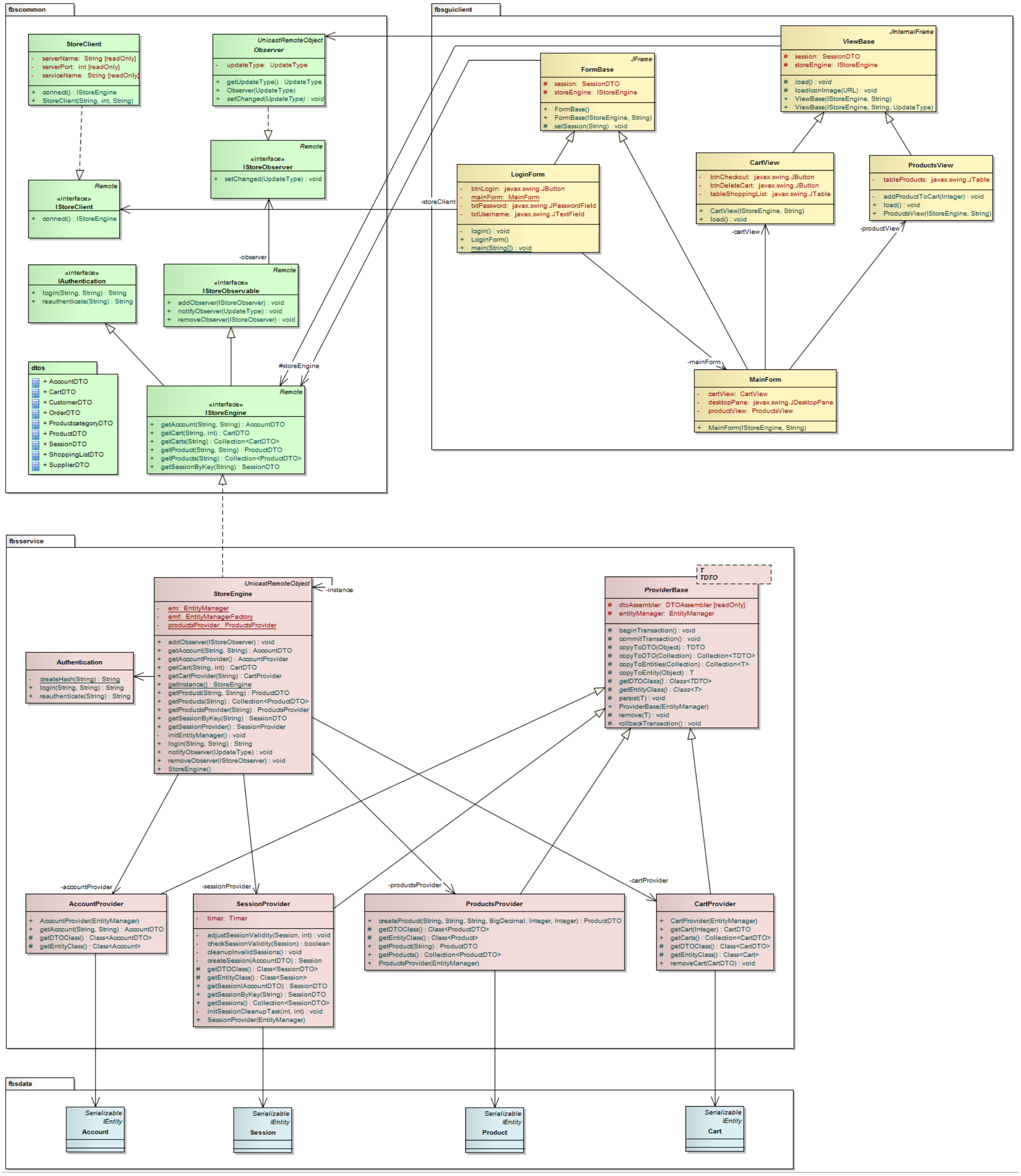

Abbildung 3: Eine Übersicht des Zusammenspiels der Klassen

2.3.1 Beschreibung der wichtigsten Klassen

Forms und Views	Die Benutzeroberfläche setzt das Konzept von MDI (Multi Document Interface) um. Als Container wird ein JFrame Form genutzt. MainForm ist quasi das Hauptfenster für alle weiteren Ansichten (Views). Dieses wird mit Menülisten und anderen Navigationselementen ausgeschmückt. Innerhalb dieser Form können mehrere Views angezeigt werden. Um Produkte aufzulisten wird eine ProductView genutzt; um Bestellungen anzuzeigen wird eine OrderView genutzt, usw. Je nach Berechtigungsstufe, welche der Benutzer hat, werden mehr oder weniger Navigationselemente angezeigt. (Ein Benutzer welcher nicht Administrator ist, soll kein Menüpunkt „Alle offenen Bestellungen anzeigen“ erhalten).
FormBase und ViewBase	Die Basisklassen FormBase und ViewBase enthalten den „gemeinsamen Nenner“ aller Forms bzw. Views. Darin werden einerseits Prozeduren zur Gestaltung der Forms/Views ausgeführt, andererseits aber auch eine Referenz auf die Shopfunktionen (IStoreEngine) sowie der bei der Anmeldung erhaltenen SessionKey hinterlegt.
StoreClient	StoreClient implementiert IStoreClient. Er hat die zentrale Aufgabe des Verbindungsaufbaus von fbsguiclient zu fbsservice. Da er durch ein Interface gedeckt wird, ist es möglich, diese Klasse später z.B. durch SecureStoreClient zu ersetzen. Diese würde dann statt einem reinen RMI-Verbindungsaufbau auch noch ein SSL Zertifikat entgegennehmen.
StoreEngine	Die StoreEngine implementiert sämtliche Methoden, welche durch das Interface IStoreEngine vorgeschrieben werden. Sie ist der zentrale Einstiegspunkt der Anwendung.
Authentication	Klasse zur Authentifizierung von Benutzername und Passwort. Enthält eine Hashing-Methode, welche das Passwort in einen SHA Hash umwandelt. Eine Reauthentifizierungsmethode erlaubt zudem, die Verlängerung einer bestehenden Session.
Provider Klassen	Die Provider Klassen (z.B. ProductProvider) implementieren CRUD Operationen für eine entsprechende Entität (z.B. Product). ProductProvider stellt beispielsweise Methoden zum Erstellen, Löschen oder Aktualisieren eines Products bereit. Provider Klassen setzen auch die Geschäftslogik um. Was in Geschäftsprozessen (z.B. Bestellprozess) formal definiert wurde, muss in den Providerklassen in Form von Code niedergeschrieben werden. Sei dies zum Beispiel, dass der OrderProvider automatisch beim Zentrallager nachbestellt, falls die Mindestlagermenge eines Produkts unterschritten wurde.
ProviderBase	Die Basisklasse ProviderBase enthält den „gemeinsamen Nenner“ aller Provider Klassen. Dazu gehören die Entity Manager Operationen (persist, remove, usw.) sowie die DTO Transformationsoperationen (copyToDTO, copyToEntity, usw.).
Entity Klassen	Sie bilden die Datenbank Entitäten ab und stellen somit Datentypen für Bearbeitungen auf dem Business Layer dar. Dank dieser Entities und dem JPA Entity Manager müssen CRUD Operation nicht via SQL Statements abgesetzt werden.

2.4 Prozesse und Abläufe

2.4.1 Authentifizierung

Der Authentifizierungs- und Autorisierungsprozess gehört zu den wichtigsten Prozessen eines IT Systems. Dieser Prozess befasst sich mit der Übertragung und Verifikation von Benutzer-Credentials (Benutzername- und Passwort-Kombinationen).

Dazu baut der fbsguiclient vorgängig und mit Hilfe von IStoreClient eine Verbindung zu fbsservice auf. Als Antwort für einen korrekten Verbindungsaufbau erhält der Client das zentrale Fassaden-Interface IStoreEngine. Dieses Interface stellt sämtliche Methoden bereit, welche von fbsservice implementiert werden. Darunter auch die von IAuthentication geerbte Login-Methode. Die Login-Methode nimmt Benutzername und Passwort entgegen. Auf fbsservice liest das Account Objekt, welches zum Anmeldeversuch passt. Darin befindet sich neben Benutzername auch ein Passwort-Hash. Die Authentication Klasse vergleicht den Passwort-Hash des übermittelten Passworts mit dem Passwort-Hash des Account Objekts. Falls die beiden Hashes übereinstimmen, wird auf der Datenbank ein Session Objekt erstellt. Für jede Session wird ein eindeutiger Session Key (Universally Unique Identifier) erstellt. Zusätzlich wird die Session mit einem Gültigkeitszeitstempel versehen. So kann später sichergestellt werden, dass ein Benutzer nicht unendlich lange mit dem System authentifiziert sein kann. Die Session Gültigkeit ist über die Konfiguration von fbsservice zu steuern.

Der fbsguiclient erhält nach erfolgreicher Authentifizierung den Session Key zurück. Dieser Key erlaubt ihm den Aufruf sämtlicher Methoden, welche in IStoreEngine bereitgestellt werden. Verwendet der Client einen Session Key, welcher zu keiner aktiven Session passt, so wird der Methodenaufruf in IStoreEngine abgebrochen und mit einer Exception bemerkbar gemacht. Für die Reauthentication (d.h. wiederholten Authentifizieren mittels Session Key) ist alleine die Implementierung von IStoreEngine (also: fbsservice) verantwortlich.

Jeder gültige Methodenaufruf verlängert die Gültigkeit der Session um den konfigurierten Wert. Ist eine Session abgelaufen, so wird dem Benutzer das Logon-Form gezeigt. Kann er sich an diesem Formular erfolgreich anmelden, so kann er seine Arbeit ohne Einschränkungen weiterführen.

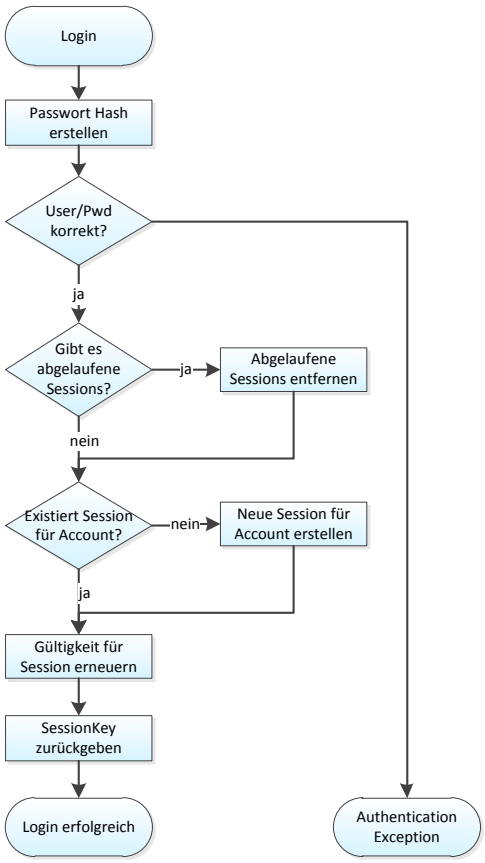

Abbildung 4: Ablauf eines Anmeldevorgangs

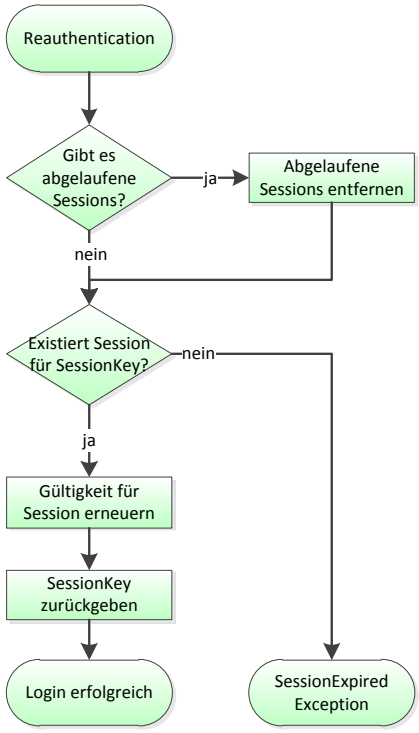

Abbildung 5: Ablauf einer Reauthentifizierung

2.5 Datenstruktur

Die Datengrundlage für das Bestellsystem wird mit Hilfe einer relationalen Datenbank realisiert. Vor dem Erstellen der physischen Datenbank wurde ein Relationales Modell erstellt.
Zur besseren Übersichtlichkeit wurde das Datenmodell in Teildiagramme zerlegt.

2.5.1 Person

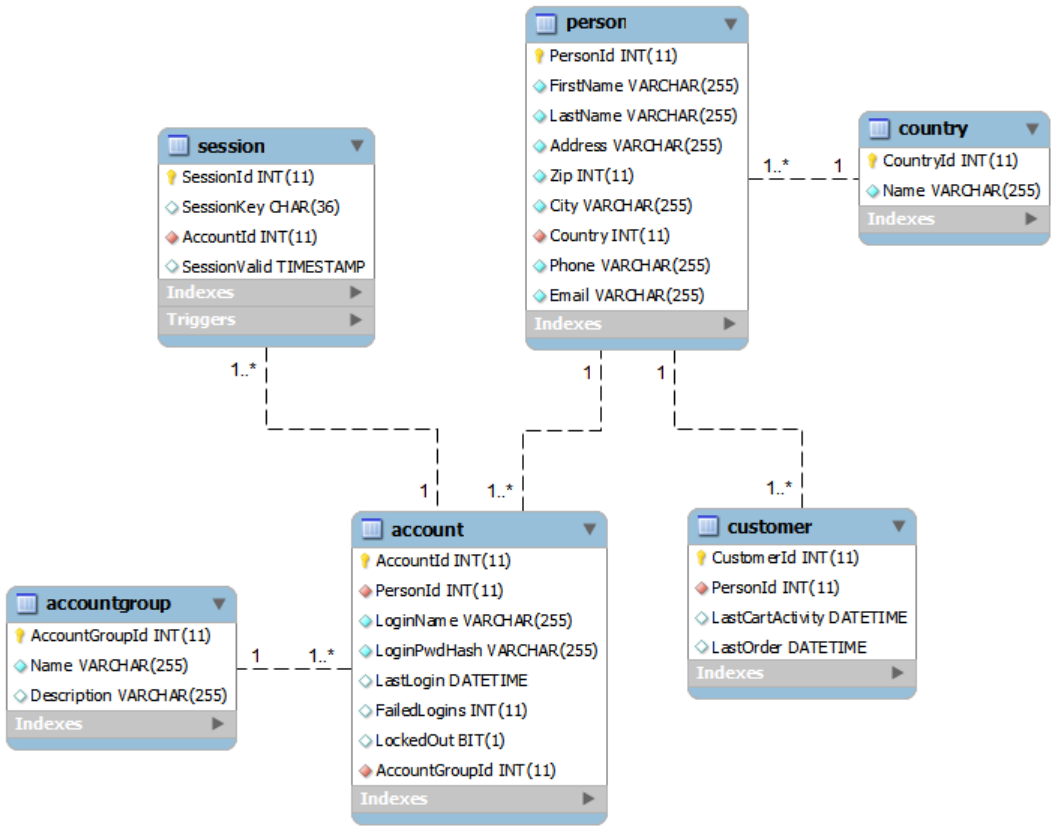

Abbildung 6: Relationen um die Entität „Person“

Account	Ein Benutzeraccount, welcher benötigt wird, um sich am System anzumelden und später Bestellungen zu tätigen. Gespeichert werden Personen- sowie Login Informationen.
AccountGroup	Dient der Gruppierung von Accounts, beispielsweise für Berechtigungen.
Country	Länderliste für Adressen.
Customer	Speichert kundenspezifische Informationen wie zum Beispiel das Datum der letzten Bestellung.
Person	Speichert allgemeine Daten zu Personen wie z.B. Name, Adresse oder E-Mail.

2.5.2 Cart

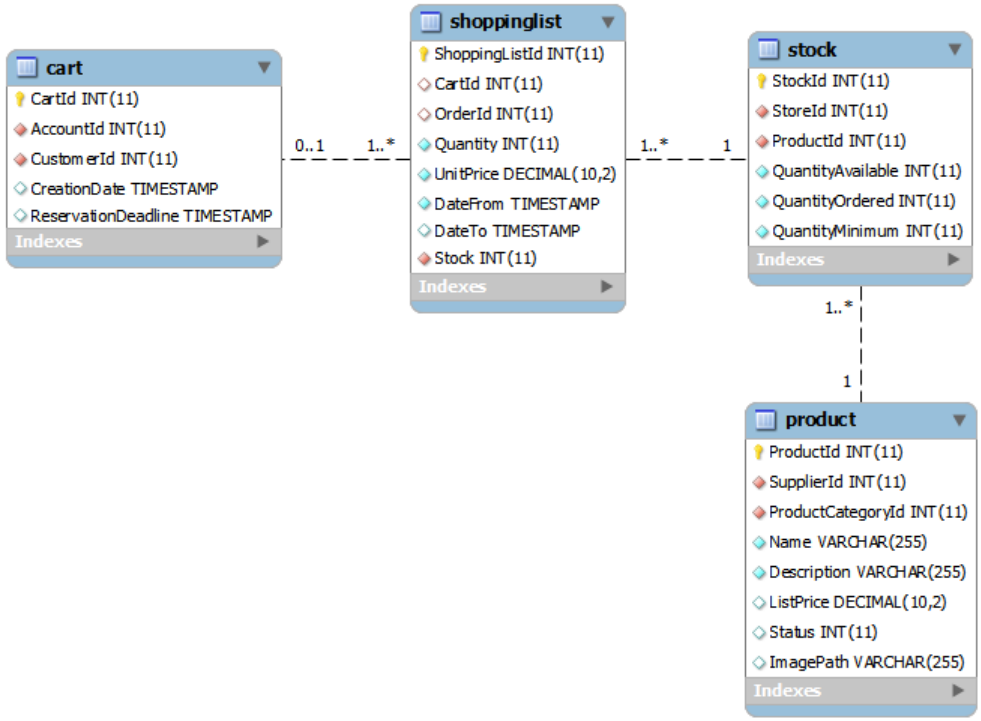

Abbildung 7: Relationen um die Entität „Order“

Cart	Während eines Einkaufvorgangs kann ein Benutzer Produkte in den Warenkorb legen. Auf sein Kommando wird der Inhalt eines Warenkorbs in eine Bestellung überführt. Mithilfe des Attributs „ReservationDeadline“ wird die Gültigkeit des Warenkorbs beschränkt.
Product	Beschreibt die Eigenschaften, welche ein Produkt haben kann.
ShoppingList	Liste der Artikel im Warenkorb oder einer Bestellung mit Mengenangabe und Preis. Beim Bestellen des Warenkorbs werden die Einträge dieser Tabelle an die Bestellung umgehängt. Von- und Bis-Datum werden für Nachbestellungen respektive Abbestellungen benötigt.
Stock	Die Lagerbestände eines bestimmten Ladens (Store).

2.5.3 Order

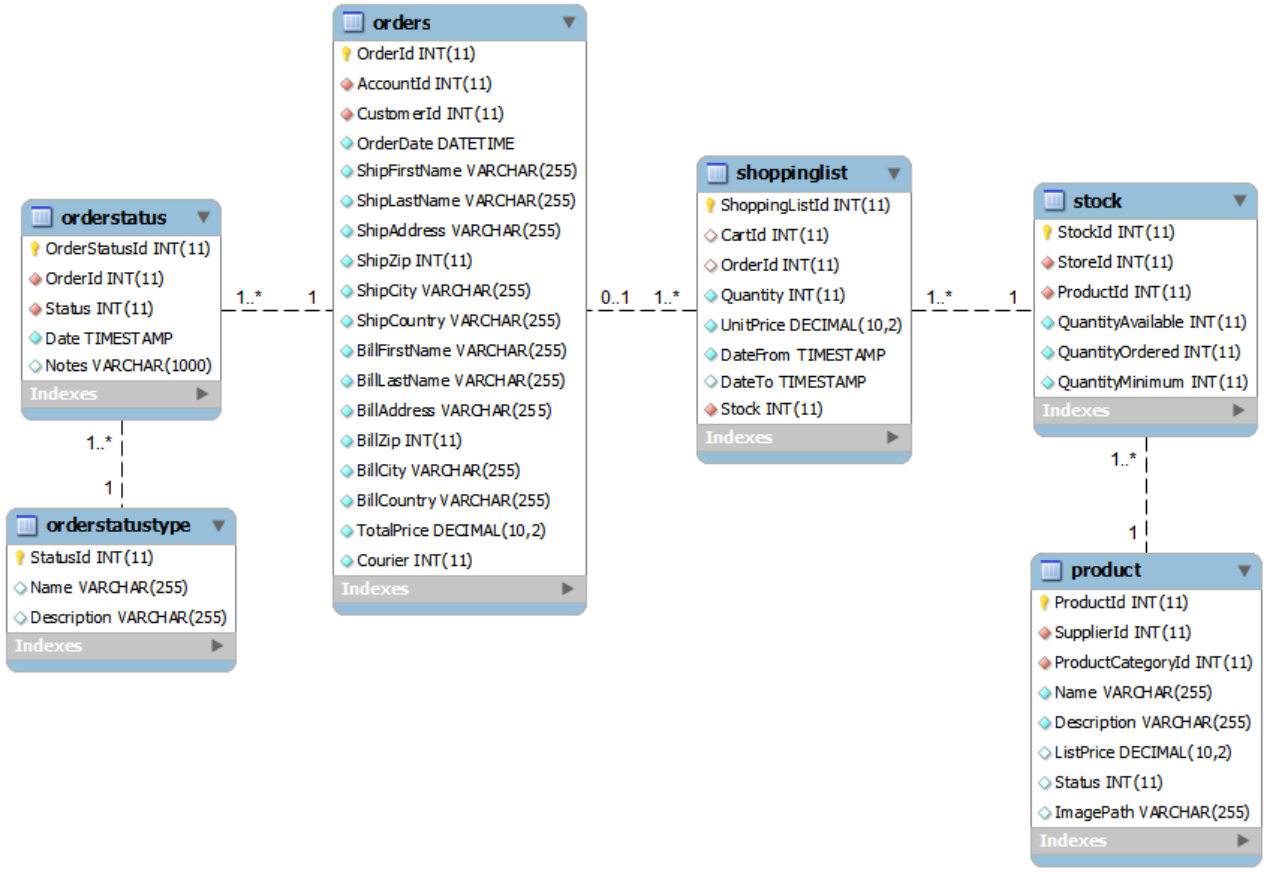

Abbildung 8: Relationen um die Entität „Order“

Orders	Eine Bestellung mit allen hilfreichen Informationen, wie z.B. Liefer- und Rechnungsadresse, Verknüpfung zum Bestellungsinhalt, Verknüpfung zum Bestellungsstatus, Gesamtkosten, … Die Daten werden kopiert damit die korrekte Archivierung gewährleistet werden kann. Andernfalls würden z.B. beim Ändern der Kundenadresse die alten Bestellungen fehlerhaft. (Hinweis: Der Name „Orders“ wurde gewählt weil „Order“ in SQL als Schlüsselwort reserviert ist).
OrderStatus	Status der Bestellung. Es wird z.B. jeweils ein Datensatz angelegt, wenn die Bestellung versendet wurde oder eine Mahnung verschickt wurde. Die Beziehung zu der Orders Tabelle trägt eine 1:m Kardinalität. Zwecks Nachvollziehbarkeit werden sämtliche OrderStatus-Änderungen festgehalten.
OrderStatusType	Eine Liste mit definierten Status Typen, welche ein Order haben kann.
Product	Beschreibt die Eigenschaften, welche ein Produkt haben kann.
ShoppingList	Liste der Artikel im Warenkorb oder einer Bestellung mit Mengenangabe und Preis. Beim Bestellen des Warenkorbs werden die Einträge dieser Tabelle an die Bestellung umgehängt. Von- und Bis-Datum werden für Nachbestellungen respektive Abbestellungen benötigt.

Stock	Der Lagerbestand eines bestimmten Stores für ein bestimmtes Produkt.

2.5.4 Product

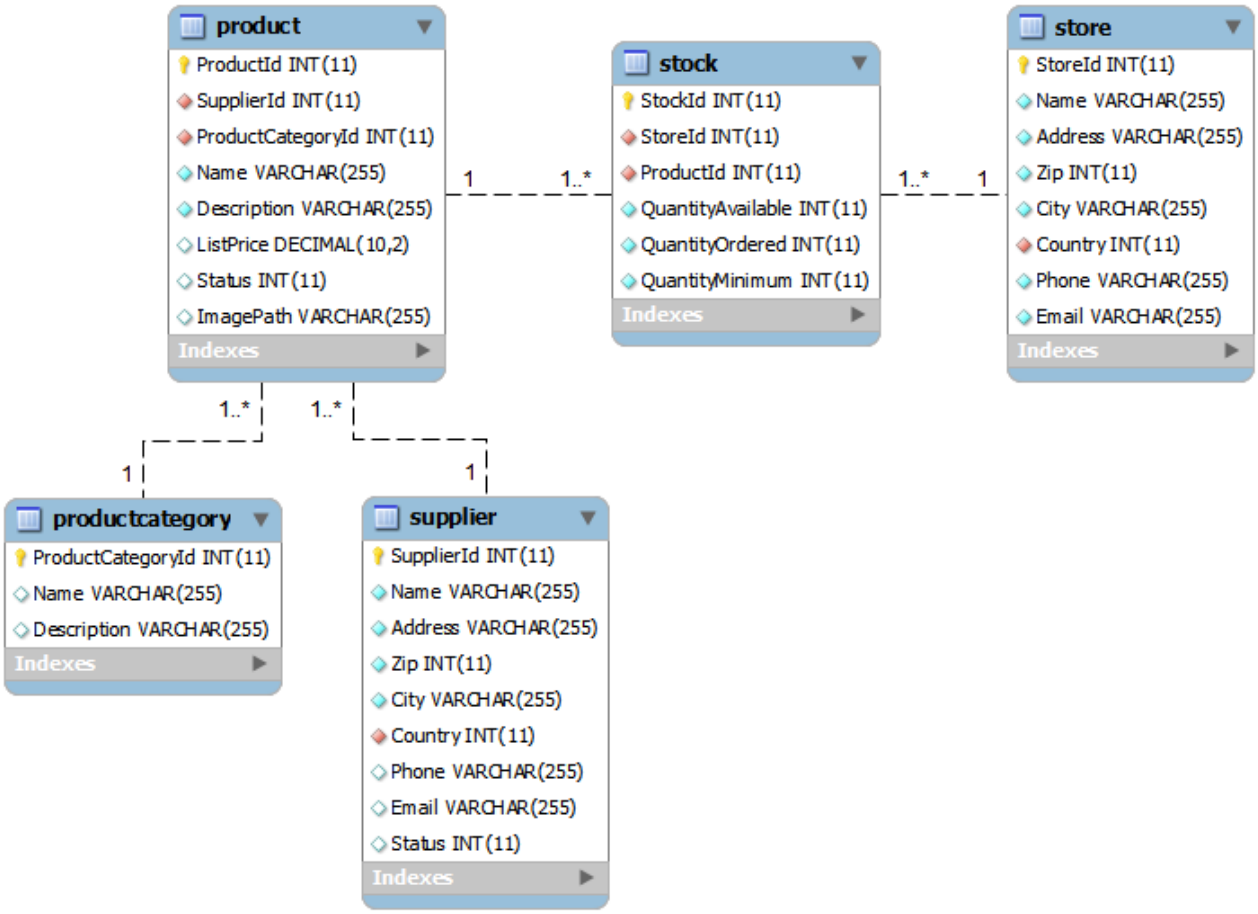

Abbildung 9: Relationen um die Entität „Product“

Product	Beschreibt die Eigenschaften, welche ein Produkt haben kann.
ProductCategory	Beschreibt die Kategorie zu welcher ein Produkt gehören kann. Beispiel: Spielwaren, Backwaren, Autozubehör,…
Stock	Die Lagerbestände eines bestimmten Ladens (Store).
Store	Eine einzelne Ladenfiliale. Sie besitzt ein eigenes Lager (Stock) in welchem die Produkte abgelegt sind.
Supplier	Enthält alle Lieferanten. Auch Lieferanten, welche gegenwärtige keine Produkte geliefert haben, können enthalten sein.

2.6 Spezifikation der Schnittstellen

2.6.1 Interne Schnittstellen

IStoreEngine	IStoreEngine stellt die zentrale Schnittstelle zum Service (fbsservice) dar. Alle für Shop Clients (fbsguiclient) benötigten Methoden werden hier nach dem Prinzip des Facade-Patterns bereitgestellt. IStoreEngine erweitert u.a. folgende Schnittstellen: • IAuthentication • IStoreObservable
IAuthentication	Eine Authentifizierungsschnittstelle, welche von IStoreEngine erweitert und in der Komponente fbsservice implementiert wird. Sie stellt mitunter Methoden zur Authentifizierung bereit.
IStoreObserver IStoreObservable	Die beiden Schnittstelle IStoreObserver und IStoreObservable haben fbsguiclient's die Möglichkeit, sich auf Events zu abonnieren, welche auf dem Service auftreten können. Dabei wird IStoreObserver überall dort implementiert, wo Client-seitig auf einen Event gewartet werden soll (z.B. Produktliste mit Lagerbestand). IStoreObservable wird Serverseitig implementiert und erlaubt es den IStoreObserver sich beim Server an- und abzumelden. Des Weiteren feuert IStoreObservable Update-Nachrichten an Clients (IStoreObserver). Das können beispielsweise Veränderungen des Lagerbestands sein, falls ein Produkt gekauft wird.
IEntity IDTO	IEntity und IDTO sind Markierungsschnittstellen für Entities bzw. DTO's. Sie kommen dann zum Einsatz, wenn DTO's auf dem Service Layer zu Entities umgewandelt werden müssen. Mit diesen Schnittstellen kann die eindeutige Zuordnung von Entity zu DTO sichergestellt werden.

2.6.2 Externe Schnittstellen

Die Schnittstellen zu externen Komponenten werden bereits in Kapitel 2.1 „Externe Komponenten“ auf Seite 7 dieses Dokuments behandelt.

IAccountingEngine	IAccountingEngine stellt Methoden zur Rechnungsverwaltung bereit. So kann über diese Schnittstelle der Zahlungsstatus offener Rechnungen geprüft werden, wie auch neue Rechnungen gestellt werden.
Stock	Die Schnittstelle „Stock“ wird in der Zentrallager-Komponente bereitgestellt (APPE Stock Library 3.0.0). Sie enthält wichtige Methoden über welche beim Zentrallager Nachbestellungen platziert warden können.

2.7 Softwareverteilung

Nirgends ist die Planung der Softwareverteilung so wichtig wie bei verteilten Systemen. Aus diesem Grund wurde der Thematik erhöhte Priorität geschenkt.
In der Minimalkonfiguration des Filialen-Bestellsystems ist vorgesehen, dass die Präsentationsschicht (fbsguiclient) und die Logik-/Datenschicht (fbsservice, fbsdata und fbsdb) auf physisch getrennten Systemen laufen. In einem erweiterten Szenario kann das DBMS (fbsdb) von der Logikschicht getrennt auf einem separaten System betrieben werden.

Die nachfolgende Grafik versucht den Sachverhalt in einem Deployment Diagramm darzustellen:

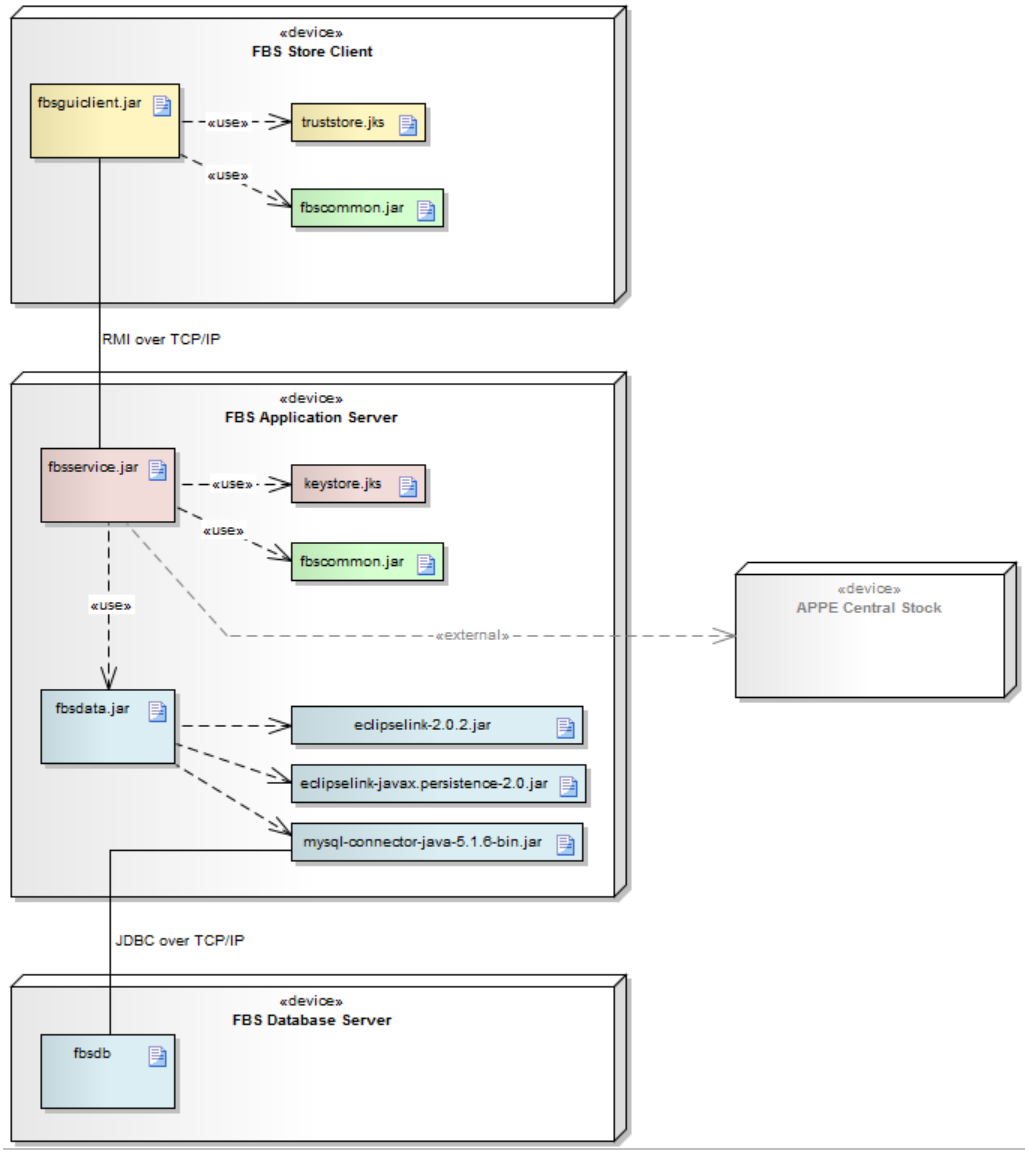

Abbildung 10: Verteilungsansicht

3 Design-Entscheide

In diesem Kapitel sind verschiedene Design-Entscheide mit den Gründen warum wir uns dafür entschieden haben dokumentiert.

3.1 Organisation der Business Logik

Im Business Logic Layer soll es unterschiedliche Funktionsträger geben. Es ist vorgesehen, dass für jede Entität ein gleichwertiger Provider erstellt wird. Beispielsweise gibt es für ein Entity vom Typ „Product“ eine Klasse „ProductProvider“ usw. Dieser Provider trägt sämtliche Methoden zur Manipulation dieses einen Entity Typs. Methoden, welche über alle Provider-Klassen identisch sind, werden in der vererbten ProviderBase-Klasse implementiert. ProviderBase stellt dementsprechend auch Methoden zur Umwandlung von Entity Objekten zu DTOs, sowie allgemeine Methoden des Entity Managers (persist, remove, usw.) bereit.

3.2 Zugriff auf Datenbank

Für den Zugriff auf die Datenbank wird die Java Persistance API (kurz JPA) eingesetzt. Die Instanz des Entity Managers wird auf der Schicht des Business Logic Layers verwaltet. Sämtliche Entity Klassen werden in derselben Instanz des Entity Managers verwaltet. Jede Klasse, welche mit Datenobjekten arbeitet, hat somit dieselbe Referenz auf den Entity Manager. Dies bietet den Vorteil, dass verschiedene Typen von Entitäten in einer einzigen Transaktion manipuliert werden können (Transaktionssicherheit).

3.3 Abbildung von Customer und Account in der DB

Im Bestellsystem gibt es zwei Arten von Personen. Einerseits die Kunden, für welche Bestellungen erfasst werden, andererseits die Mitarbeiter, welche Bestellungen erfassen. Es kann natürlich auch sein, dass ein Mitarbeiter etwas für sich selber bestellt. Dadurch kann dies (in Java) nicht mit Vererbung abgebildet werden, da hierfür Mehrfachvererbung benötigt würde.
Wir haben uns entschieden, allgemeine Informationen, welche für Mitarbeiter und Kunden benötigt werden in der Tabelle Person abzulegen, kundenspezifische Informationen sind in der Tabelle ‚Customer‘ und Daten welche zum Einloggen in die Anwendung benötigt werden sind in der Tabelle ‚Account‘ zu finden.
Vorteile:

- Die Datenintegrität kann einfacher gewährleistet werden. Bei einer Lösung mit einer Tabelle, welche alle Felder enthält, ist es schwieriger zu prüfen, ob z.B. alle Informationen vorhanden sind, die von einem Kunden benötigt werden.
- Es können Fremdschlüssen auf Kunden und Sachbearbeiter gemacht werden
- Es gibt keine redundanten Daten. Bei einer Lösung mit zwei Tabellen wäre dass der Fall.

Nachteile:

- Etwas komplexeres DB-Schema
- Das Mapping im Code wird aufwändiger

3.4 Datenaustausch mit Data Transfer Objects

Wir haben uns für den Einsatz von DTOs entschieden, da dadurch von der Business Logik bestimmt werden kann, was an die Präsentationslogik übergeben werden soll. Dadurch können z.B. sensitive Daten gefiltert werden. Als positiver Nebeneffekt kann die zu übertragende Datenmenge reduziert werden.

3.5 Generische Entity-DTO Assemblierung

Durch den Einsatz von DTOs entsteht auch ein erhöhter Programmieraufwand. Um diesen zu minimieren haben wir uns für den Einsatz eines Frameworks zur generischen Assemblierung und Disassemblierung entschieden. Die Komplexität innerhalb der Komponente fbsservice ist jedoch dadurch gestiegen.

3.6 Kommunikation zwischen PL und BLL

Die Kommunikation zwischen Presentation Layer (fbsguiclient) und Business Logic Layer (fbsservice) wird mit Hilfe von Java Remote Methode Invocation (kurz RMI) realisiert. Infolgedessen ist es notwendig, die Business Logik mit einem Interface zu abstrahieren. Der Einsatz von vielen Interfaces (für jeden Provider eines) wäre zu umständlich und kompliziert, daher wurde entschieden, als Eintrittspunkt in die Applikation ein Facade Interface zu erstellen. Dieses vereint alle Provider hinter einer einheitlichen Schnittstelle.

3.7 Sichere Übertragung zwischen PL und BLL

Da die beiden Kommunikationspartner fbsguiclient und fbsservice sensible Informationen austauschen, bedarf es hier einer Sicherung des Übertragungskanals. Konkret heisst dies, dass RMI mit Transport Layer Security (TLS) abgesichert werden muss. Die Implementationen der Client-Komponenten sollen durch eine Schnittstelle (IStoreClient) so entkoppelt werden, dass wahlweise zwischen einem unsicheren und einem sicheren Client-Connector gewählt werden kann. Serverseitig sollen grundsätzlich sichere und unsichere Client-Anfragen akzeptiert werden. Es ist denkbar, dass das Verwenden von TLS über ein Config-Eintrag erzwungen werden kann.

3.8 Authentifizierung mit Hash-Funktion

Die Benutzerauthentifizierung wird über eine Schnittstelle (IAuthentication) gesteuert, welche in der StoreEngine auf fbsservice implementiert wird. Das GUI muss folglich Loginname und Passwort vom Benutzer entgegennehmen. Die Credentials werden im Authorization Modul auf dem Service Layer ausgewertet.
Damit in der Datenbank keine Kennwörter in Klartext gespeichert werden müssen, wird in der Account Tabelle lediglich der Hash-Wert des Passworts abgelegt. Der verwendete SHA-Algorithmus wurde so gestaltet, dass es keine effiziente Möglichkeit gibt, aus dem Hash auf das gespeicherte Passwort zurückzuschliessen. Der Authentifizierungsprozess wird unter 2.4.1 Authentifizierung (Seite 11) detailliert erklärt.

3.9 Autorisierung in StoreEngine

Nach erfolgreicher Authentifizierung hat ein Client grundsätzlich die Möglichkeit, jede Methode im bereitgestellten Interface IStoreEngine aufzurufen. Nicht jeder Benutzer hat jedoch die Berechtigung, jede dieser Methoden aufzurufen. Um dieser Anforderung gerecht zu werden, wird in StoreEngine mittels einer separaten Klasse die Identität des Benutzers mit den geforderten Benutzerrechten verglichen. Genügen die Rechte des Benutzers den definierten Voraussetzungen, so wird die gewünschte Methode ausgeführt. Andernfalls wird eine UnauthorizedException zurückgegeben.
An sich kein weltbewegender Ablauf. Bemerkenswert dieses Design Entscheids ist, dass keine Autorisierungen innerhalb der Provider-Klassen (ProductProvider, OrderProvider,…) vorgenommen werden. Diese Klassen dürfen davon ausgehen, dass der Aufrufer bereits vorgängig in StoreEngine auf ihre Autorität geprüft wurden.

3.10 Session Handling mit SessionKey‘s

Beim ersten Anmelden erhält der Benutzer einen SessionKey, welchen Ihn bemächtigt, das System zu nutzen. Dieser Schlüssel ist mit einem herkömmlichen Türschlüssel zu vergleichen: Er erlaubt dem Benutzer gewisse Aktionen – abhängig von seiner Autorisierung - durchführen zu können. Methoden, welche in IStoreEngine deklariert sind, verlangen (neben den Methodenparametern auch) die Angabe des SessionKeys. Jede erfolgreiche Ausführung einer Methode endet in einer automatischen Reauthentifizierung (siehe dazu auch Abbildung 5: Ablauf einer Reauthentifizierung, Seite 11).
Ein einfaches Beispiel zum beschriebenen Sachverhalt: Ein Benutzer möchte ein Produkt aus dem Sortiment löschen. Er ruft die entsprechende Methode der StoreEngine auf. Dort wird mit Hilfe der Provider-Klassen geprüft, ob der Benutzer über die nötigen Berechtigungen verfügt. Falls dies der Fall ist, so wird die Aktion ausgeführt – ansonsten eine Fehlermeldung zurückgegeben.

3.11 Transaktionssicherheit

Die Business Logik muss sicherstellen, dass bestimmte Aktionen atomar ablaufen. Zu diesem Zweck stellt der EntityManager des JPA Framework Transaktionen bereit. JPA unterstützt auch verschachtelte Transaktionen wenn die Datenbank dies nicht unterstützt. In diesem Fall wird einfach für die äusserste Transaktion eine Datenbanktransaktion verwented.
Beim Fillialbestellsystem benötigen wir verschachtelte Transaktionen, da einige Methoden direkt aufgerufen werde können oder auch indirekt durch eine andere Methode. In beiden Fällen soll die Transaktionssicherheit gewährleistet werden. Die Grundregel zur Verwendung von Transaktionen in fbsservice lautet wie folgt: Jede öffentliche Methode eines Providers muss transaktionssicher sein. Interne (private) Methoden müssen zwingend von der der aufrufenden Methode gesichert werden.

Ein Beispiel für eine transaktionssichere Methode „Warenkorb löschen“:

```
Anfang der Transaktion
      Entferne Produkte aus Warenkorb
      Lege Produkte zurück ans Lager
      Lösche Warenkorb
Ende der Transaktion
```

3.12 Verwaltung von Konfigurationsdateien

Jede eigenständige Komponente, welche eine individuelle Konfiguration zulässt, erhält ihre eigene Konfigurationsdatei. Diese Properties-Datei muss zwingend ausserhalb der kompilierten Komponente liegen, sodass Änderungen durch den Benutzer vorgenommen werden können. Komponenten, welche andere Komponenten Nutzen müssen auch deren Konfigurationsdatei zugänglich machen. So nutzt beispielsweise fbsservice die Bibliothek fbscommon und muss deswegen nicht nur fbsservice.properties, sondern auch fbscommon.properties bereitstellen.
Die Bedeutung der Attribute der jeweiligen Konfigurationsdateien ist im Betriebshandbuch zu finden.

Environment-Anforderungen

FBS Client	Prozessorleistung	1GHz
	Arbeitsspeicher	512MB RAM
	Festplattenspeicher	20MB
	Betriebssysteme	Windows XP/Vista/7, Linux, Mac OS X
	Bildschirmauflösung	1440x800 (mindestens)
	Vorausgesetzte Software	Java SE Runtime Environment, 1.6.0_21
FBS Application Server	Prozessorleistung	1GHz[1]
	Arbeitsspeicher	1GB RAM[2]
	Festplattenspeicher	20MB
	Betriebssysteme	Windows XP/Vista/7, Windows Server 2008, Linux, Mac OS X
	Netzwerk	TCP Port 3333 (default) TCP Port 4444 (default bei SSL)
	Vorausgesetzte Software	Java SE Runtime Environment, 1.6.0_21
FBS DB Server	Prozessorleistung	1GHz
	Arbeitsspeicher	1GB RAM
	Festplattenspeicher	200MB[3]
	Betriebssysteme	Windows XP/Vista/7, Windows Server 2008, Linux, Mac OS X
	Netzwerk	TCP Port 3306 (MySQL default)
	Vorausgesetzte Software	Java SE Runtime Environment, 1.6.0_21 MySQL Server 5.5.10

[1] Die Prozessorleistung ist abhängig von der Aktivität der verbundenen Clients.
[2] Pro aktive Client-Verbindung werden zusätzliche 10MB Arbeitsspeicher benötigt.
[3] Der benötigte Festplattenspeicher ist abhängig von der Menge gespeicherter Datensätze.

4 Qualitätsmerkmale

Im Projektauftrag wird explizit erwähnt, dass die Qualitätsmerkmale des zu erstellenden Softwareprodukts dokumentiert werden. Diesem Auftrag wird in den nächsten Kapiteln Folge geleistet.

4.1 Korrektheit

Ein Softwaresystem ist dann (funktional) korrekt, wenn es sich genauso verhält, wie in der Produktanforderung festgelegt wurde.
Problem: Die Produktanforderung lässt oft Spielraum für Interpretationen. Diese gilt es zu minimieren um ein gemeinsames Verständnis für das zu erstellenden System zu erlangen. An Meilensteinbesprechungen sollen die Produktanforderungen jeweils mit den bereits existierenden Prototypen und Mock-Ups verglichen werden. So können Missverständnisse frühestmöglich eliminiert werden.

4.2 Zuverlässigkeit

Software-Zuverlässigkeit ist definiert als "Wahrscheinlichkeit der fehlerfreien Funktion eines Computer-Programms in einer spezifizierten Umgebung in einer spezifizierten Zeit" [8].
Die Metriken für Zuverlässigkeit von Software basieren auf "Fehlerhäufigkeit im Verhältnis zur Anzahl der ausgeführten Testfälle in einem bestimmten Zeitraum". Die Testfälle können in Unit-, Komponenten- oder Systemtests definiert werden. Damit eine statistische Relevanz erreicht werden kann, bietet es sich an, die Tests zu automatisieren (z.B. mit Build Systemen). Mit sog. Regressionstests [8] kann die relative Zuverlässigkeit über einen bestimmten Zeitraum beobachtet werden.

4.3 Robustheit

Ein Softwareprodukt ist robust, wenn es auch unter unvorhergesehenen Umständen vernünftig reagiert (z.B. auf zufällige oder beabsichtige Angriffe). Die Robustheit kann erhöht werden in dem solche Ausnahmefälle in Form von Testfällen in einem Testkonzept dokumentiert und später in Form von Unit-/Systemtests implementiert werden.
Nebst der Robustheit des Softwareprodukts soll auch er Entwicklungs- und Wartungsprozess so robust wie möglich sein. Ein gut dokumentiertes System ist auch dann wartbar, wenn sich das Personal innerhalb des Entwicklungsteams ändert. Hardware-/Systemausfälle können ebenfalls an der Robustheit des Entwicklungsprozess' kratzen.

4.4 Informationssicherheit

Informationssicherheit bedeutet Sicherheit im Sinne von risikoarmen Benutzen von Software. Dazu gehören folgende Aspekte:

- **Confidentiality:** Das System ist verantwortlich für die Verschlüsselung von sensiblen Daten wie z.B. Benutzername- und Passwort-Kombinationen. Sowohl während der Übermittlung zwischen Client und Server wie auch überall dort, wo diese Informationen gespeichert werden (Datenbank, Backups, Logs). Der Zugriff auf diese Informationen muss eingeschränkt werden.
  Die technischen Möglichkeiten sind sehr umfangreich und fordern meistens grosse Anpassungen an der Architektur. Die beiden Kommunikationsträger fbsguiclient und fbsservice werden für den Einsatz von SSL-over-RMI vorbereitet.

- **Integrity:** Informationen dürfen keinesfalls unbemerkt manipuliert werden können - weder während der Übertragung zwischen den Schichten noch in persistenten Datenablagen. Die Signierung und Verschlüsselung von übertragenen Objekten sowie das Führen von Transaktionsregistern kann die Datenintegrität stärken. Datenbanksystemen bieten mit "Referenzieller Integrität" gute Möglichkeiten, Integrität und Konsistenz der Daten zu gewährleisten. In der vorliegenden Datenbank wurden alle Relationen zwischen Entitäten auf konsistentes Verhalten überprüft. Das Datenbanksystem verhindert somit das Löschen von Datensätzen, welche mit anderen Entitäten verknüpft sind.

- **Availability:** Die Verfügbarkeit der darunterliegenden IT Systeme obliegt (oft) dem Betreiber der Infrastruktur. (Ausgenommen: SaaS, was aber in diesem Projekt nicht der Fall ist). Die Software muss seinerseits sicherstellen, dass die benötigten Resourcen nicht unökonomisch oder gar absichtlich aufgebraucht werden (sog. DoS Attacken). Im vorliegenden Softwareprodukt könnte beispielsweise die Implementation von StoreEngine um einen ThreadPool erweitert werden. Dieser würde Client-Anfragen über eine Queue in separate Threads auslagern, sodass Anfragefluten (bis zu einem bestimmten Mass) abgedämpft werden können.
  Eine Zusätzliche Steigerung der Verfügbarkeit würde ein Datenbank-Cluster herbeiführen: Mehrere MySQL Instanzen auf physisch und geographisch getrennten Systemen könnten die Leistung eines international genutzten Shop Systems markant steigern. Für die Synchronisation zwischen den Datenbanksystemen können gängige Technologien genutzt werden.

- **Authenticity & Autorisation:** Die Identität eines Benutzers muss sichergestellt werden, vor dieser mit dem System arbeiten darf. Der Authentifizierungsprozess darf vom Anfragesteller nicht beeinflusst werden können. Der Prozess hat also auf einem vertrauenswürdigen, gesicherten und physisch getrennten System zu erfolgen (fbsservice). Nach erfolgter Authentifizierung gilt es herauszufinden, zu welchen Aktionen ein Benutzer berechtigt ist (Autorisierung).

- **Non-repudiation:** Die Unbestreitbarkeit von Aktionen hat vor allem rechtliche Relevanz: Käufer noch Verkäufer müssen einen vereinbarten Vertrag einhalten können. Transaktionen dürfen nicht manipuliert werden können. Im E-Commerce werden deshalb Technologien eingesetzt, welche mit digitalen Zertifikaten sicherstellen, dass keine vertragsrelevanten Dokumente gefälscht werden können.

4.5 Effizienz

Effizienz ist ein Mass für ein Ergebnis unter Berücksichtigung der eingesetzten Mittel. In der Softwareentwicklung bedeutet dies oft ein Tradeoff zwischen "Speicherbedarf und Rechenaufwand" oder "Entwicklungsaufwand und Softwarequalität" [5]. Als Beispiel für gute Effizienz in diesem Projekt wurde die Klasse StoreEngine als Singleton-Klasse implmentiert. So kann verhindert werden, dass das zeitraubende Laden des Entity Managers für jeden Client erneut vollbracht werden muss. Als Alternative dazu wurde ins Auge gefasst, dass jede Provider-Klasse (ProductProvider, OrderProvider, usw.) ihre eigenen EntityManagers haben und diese bei jedem Methodenaufruf über IStoreEngine neu instanziert würden. Das hat sich aber als zu ineffizient erwiesen. Zuviele unnötige Datenbankaufrufe wären mit dieser Lösung gemacht worden.

4.6 Wartbarkeit

Zur Wartbarkeit einer Software gehören u.a. eine aktuelle System-Dokumentation, Dokumentation von Source Code sowie Versionisierung von Configuration Items.

4.7 Portierbarkeit

Da in diesem Projekt das Java Runtime Environment zu Einsatz kommen soll, ist auch die Portierbarkeit auf andere Java-kompatible Systeme gegeben.

4.8 Kompatibilität

Um die Kompatibilität des Systems zu erhöhen, wurden das Gesamtsystem in Teilsysteme zerlegt und diese mit definierten Schnittstellen versehen. So können beispielsweise auf dem Business Logic Layer (fbsservice) Änderungen vorgenommen werden, welche auf andere Komponenten (z.B. fbsguiclient) keinen Einfluss haben.
Die Client-Applikation fbsguiclient greift über eine Schnittstelle IStoreClient auf den Service zu. Dies ermöglicht beispielsweise später die einfache(re) Implementierung eines SSL-basierten RMI-Clients.

5 Quellen

Ref	Quellenangabe / URL	Letzter Zugriff
[1]	http://en.wikipedia.org/wiki/Multitier_architecture	24.04.2011
[2]	http://de.wikipedia.org/wiki/Software-Ergonomie	24.04.2011
[3]	http://de.wikipedia.org/wiki/Observer_(Entwurfsmuster)	24.04.2011
[4]	www.swe.informatik.uni-goettingen.de/.../II-Software-Qualitaet-6-auf-1.pdf	01.05.2011
[5]	http://de.wikipedia.org/wiki/Effizienz	01.05.2011
[6]	http://en.wikipedia.org/wiki/Information_security	01.05.2011
[7]	http://de.wikipedia.org/wiki/Software-Zuverl%C3%A4ssigkeit	01.05.2011
[8]	http://de.wikipedia.org/wiki/Regressionstest	01.05.2011
[9]	http://de.wikipedia.org/wiki/Hashfunktion	01.05.2011
[10]	http://de.wikipedia.org/wiki/Transport_Layer_Security	19.05.2011