Wendy's Oster Shop

Projektdokumentation

Hans-Jörg Schrödl, 0925075

1. April 2016

Stundenliste

Die Zeiterfassung wurde mithilfe von Timetrap durchgeführt. Insgesamt benötigte das Projekt als in etwa 70 Stunden.

Start	Ende	Beschreibung
2016-03-05 15:29:53	2016-03-05 16:04:39	Einlesen in angabe
2016-03-06 12:01:17	2016-03-06 14:54:32	Projekt aufsetzen, Datenbank erstellen
2016-03-06 15:17:23	2016-03-06 17:18:09	Datenbank erstellen
2016-03-06 17:18:28	2016-03-06 17:57:22	Erstellen von DB Initialisierungsskript
2016-03-06 21:32:38	2016-03-06 21:54:11	Fortsetzung DB Initialisierungsskript
2016-03-06 21:54:19	2016-03-06 23:51:08	Erstellen erster DAOs
2016-03-11 16:28:40	2016-03-11 17:50:04	Projektonfiguration für laptop
2016-03-11 17:50:19	2016-03-11 20:30:00	Erstellen erster DAO Unit Tests
2016-03-12 09:00:00	2016-03-12 11:56:04	Artikel DAOs und entsprechende Unit Tests
2016-03-12 11:56:10	2016-03-12 13:30:00	Rechnung DAOs
2016-03-12 16:30:00	2016-03-12 20:30:00	Rechnung DAOS und Article Filter
2016-03-17 10:31:29	2016-03-17 12:28:59	Fortsetzung Artikel Filter
2016-03-18 11:05:12	2016-03-18 14:04:46	Implementiere Artikel Statistik
2016-03-18 15:12:53	2016-03-18 16:53:05	Implementiere Preisanpassung
2016-03-20 18:25:01	2016-03-20 20:30:00	Verbesserung der Filter
2016-03-21 09:35:20	2016-03-21 09:41:15	Verbesserung Artikel Filter
2016-03-21 10:43:19	2016-03-21 16:17:02	Verbesserung Artikel Filter
2016-03-21 19:42:21	2016-03-21 20:40:30	Prototyp für Bildverwaltung
2016-03-21 20:40:37	2016-03-21 21:56:29	Installation von JavaFX
2016-03-22 09:44:23	2016-03-22 13:30:00	Erster UI Prototyp
2016-03-22 16:33:36	2016-03-22 17:13:25	Erstelle Artikeldetails-Maske
2016-03-23 16:00:00	2016-03-23 17:37:33	Refaktoriere Artikel-DAO
2016-03-24 10:30:51	2016-03-24 13:07:22	Erstelle Rechnungsüberblick-Maske
2016-03-24 19:06:49	2016-03-24 21:00:39	Erstelle Rechnungsdetails-Maske
2016-03-24 21:20:46	2016-03-24 22:40:39	Fortsetzung Rechnungsdetails-Maske

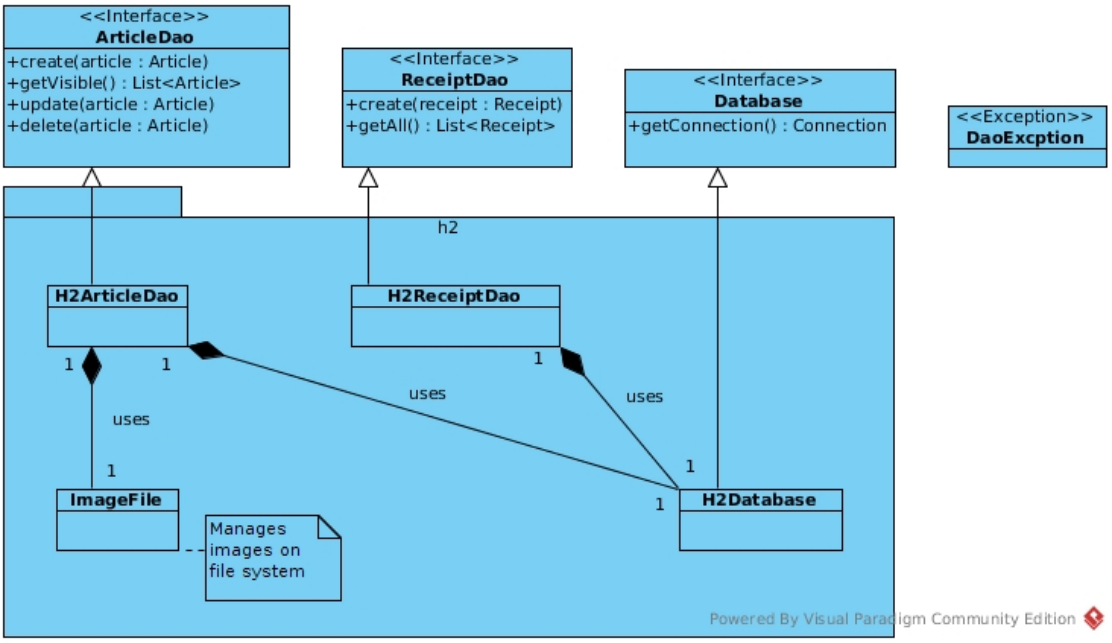
2016-03-25 10:25:48	2016-03-25 13:54:42	Implementiere Erstellen einer neuen Rechnung
2016-03-25 13:54:51	2016-03-25 15:00:00	Verbessere Bildverwaltung
2016-03-28 16:19:43	2016-03-28 18:00:00	Implementiere Bildänderung in Artikel-Details
2016-03-29 11:40:33	2016-03-29 13:36:59	Implementiere Artikelfilter
2016-03-29 17:38:49	2016-03-29 17:58:41	Verbessere Artikel-Tabelle
2016-03-29 17:58:51	2016-03-29 18:50:09	Implementiere Artikel-Statistik
2016-03-29 22:25:21	2016-03-29 23:18:31	Verbessere Artikel-Statistik
2016-03-29 23:18:38	2016-03-30 00:58:59	Implementiere Preisanpassung
2016-03-30 11:02:39	2016-03-30 12:30:00	Fortsetzung Preisanpassung
2016-03-30 18:27:21	2016-03-30 20:04:28	Verbessere Tests, GUI-Feinschliff
2016-03-30 20:04:45	2016-03-30 20:41:49	Verbessere Test-Coverage, DAO-Feinschliff
2016-03-31 15:34:27	2016-03-31 17:54:10	Beginn Projektdokumentation
2016-03-31 17:54:14	2016-03-31 19:30:00	Fortsetzung Projektdokumentation

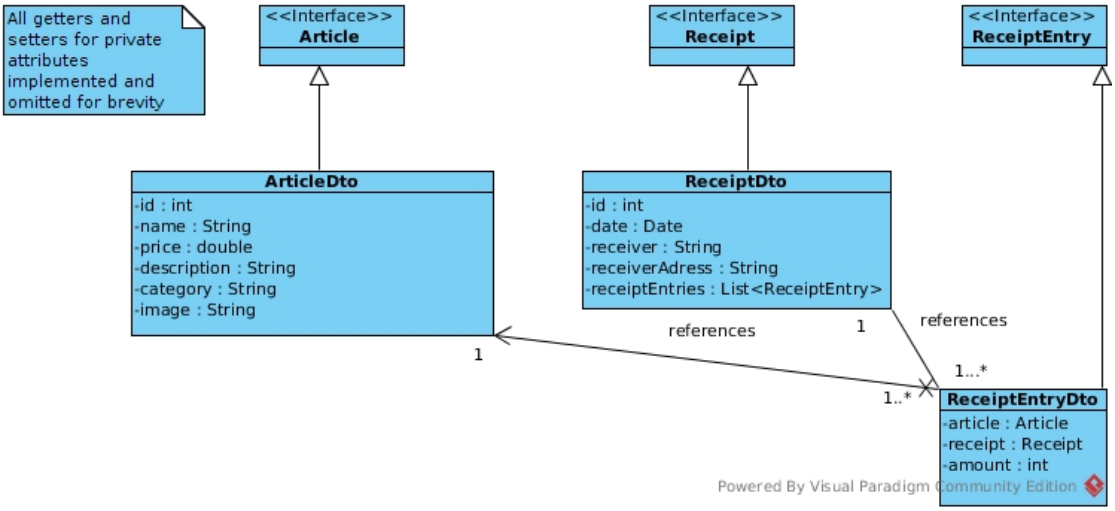
Systemarchitektur

Das Programm wurde entsprechend den Vorgaben in drei Schichten unterteilt. Der Datenaustausch zwischen den Schichten erfolgt über DTOs die im Paket “Entities” zu finden sind.

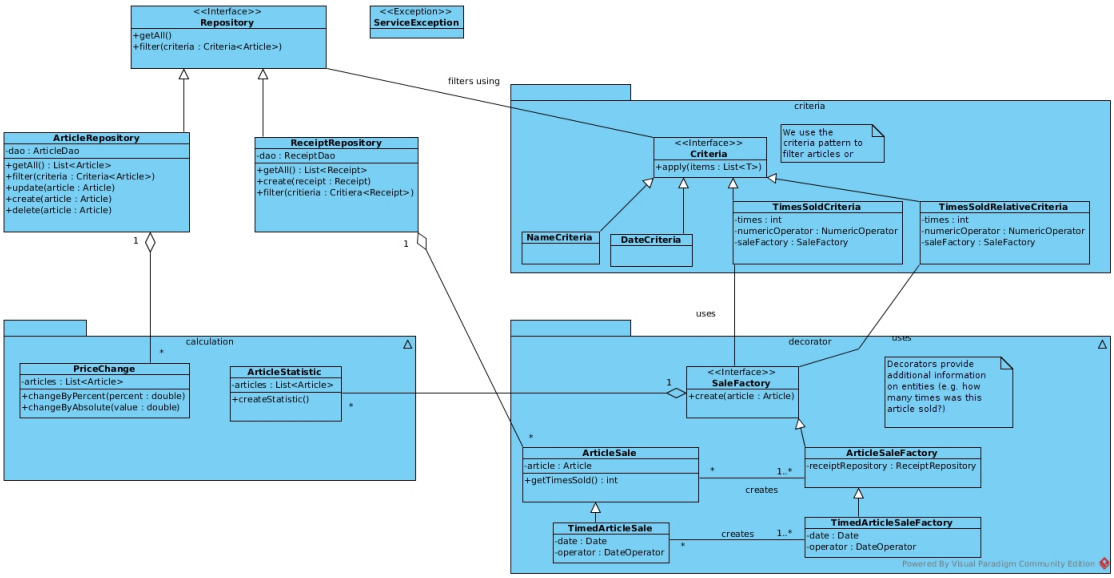
Paketübersicht

Das Paket “Dao” enthält Klassen, die für die Kommunikation mit der Datenbank zuständig sind. Als Design-Vorlage wurde das DAO-Pattern verwendet. Die konkrete Implementierung für die H2-Datenbank befindet sich im Paket “H2”. Weitere Design-Grundsätze die zu beachten sind: Konkrete Datenbank-Verbindungen werden in einer eigenen Klasse gekapselt um die die Erstellung von Unit-Tests zu erleichtern. Auf die Verwendung des Singleton-Patterns verzichtet. Um Kapselung zu wahren wird stattdessen Dependency-Injection (bevorzugt über den Konstruktor) verwendet.

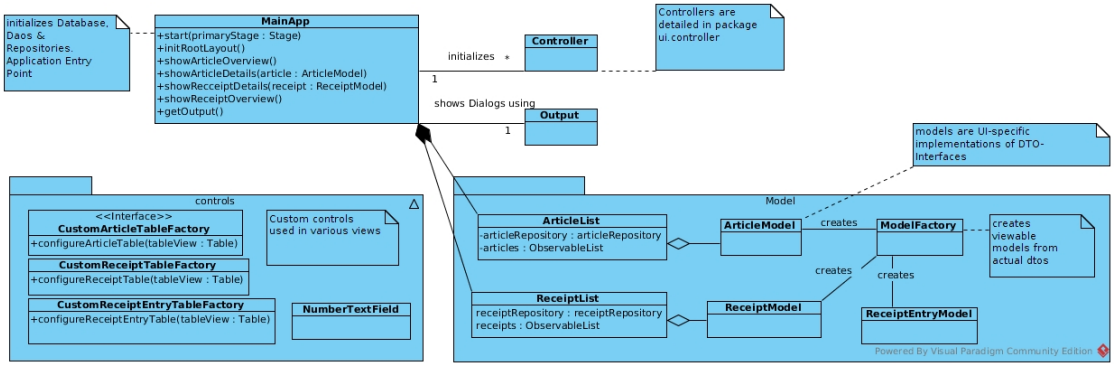


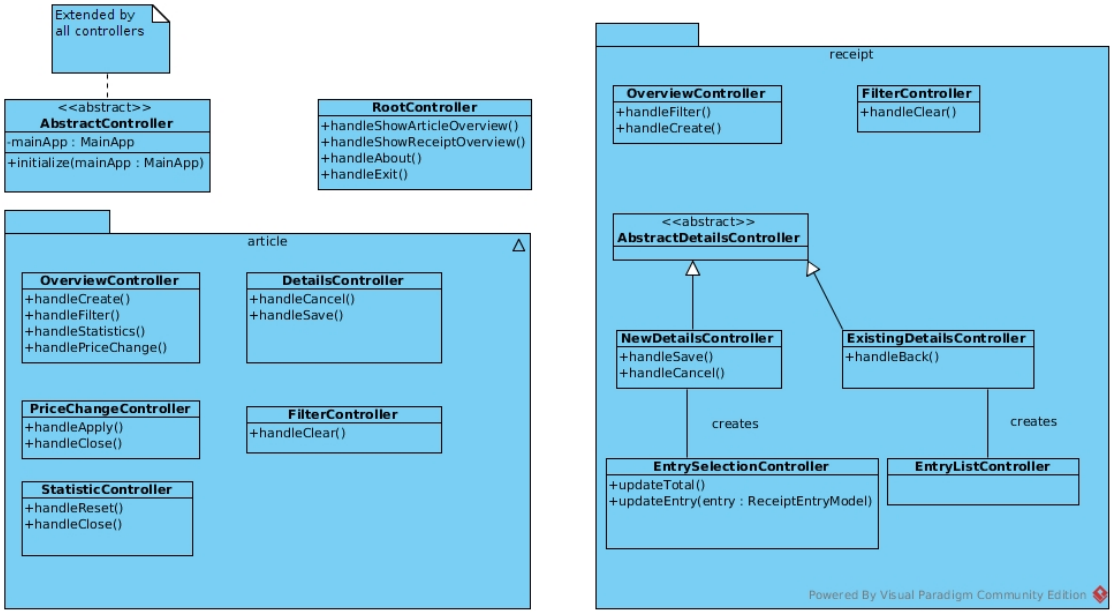


Die Service-Schicht enthält Klassen, die für die Verwaltung von Daten zur Laufzeit sowie für die Bearbeitung von spezifischen Anwendungsfällen zuständig sind. Um die Suche nach Artikeln und Rechnung zu implementieren wurde das Criteria-Pattern verwendet. Um DTOs mit abgeleiteten Daten auszustatten (e.g wie oft ein Artikel in einem Zeitraum verkauft wurde) wurde das Decorator-Pattern zusammen mit dem Factory-Pattern verwendet.



Die UI-Schicht wurde dem Model-View-Controller-Pattern folgend aufgebaut. Die graphische Benutzeroberfläche wurde in FXML deklariert während Logik in den unterliegenden Controllern (die wiederum auf diverse Komponenten der Service-Schicht zugreifen) abgewickelt wird.
Controller selbst sind entsprechend der GUI gegliedert. Der zentrale Punkt des Programms ist die Klasse “MainApp”. Diese ist für die Initialisierung aller Services sowie der GUI zuständig.



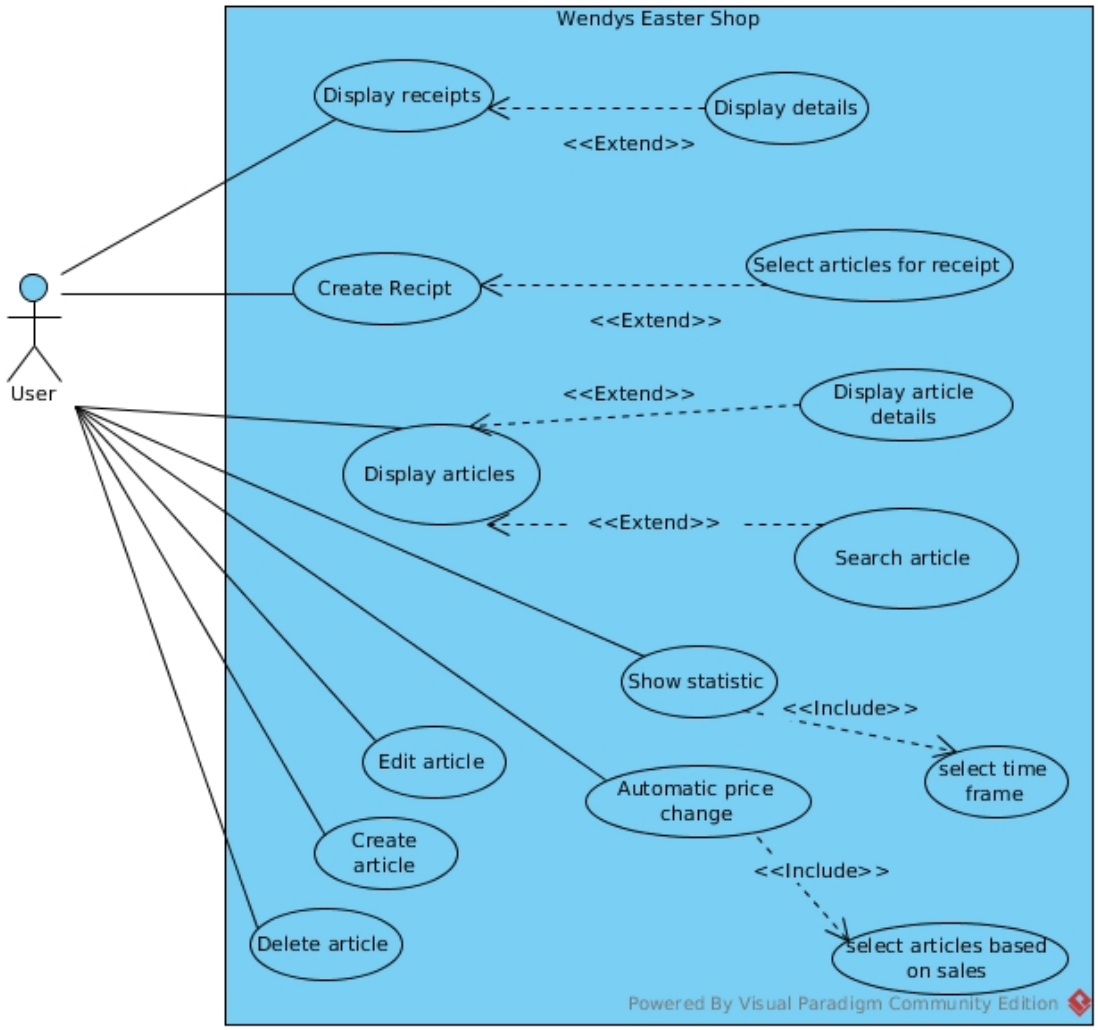


Test-Prinzipien

Die Unit-Tests der Anwendung folgen eigenen Design-Prinzipien die hier nur kurz erläutert werden sollen. Mocks (aus dem Framework mockito) werden eingesetzt um Klassen isoliert testen zu können. Hier kommt die exzessive Verwendung von Dependency-Injection zugute. Test-Klassen folgen der Benennung *KlassenNameTests*. Test Methoden sind wie folgt benannt: *Methode_Szenario_ErwartetesErgebnis*. Prinzipiell wurde versucht, den Grundlagen aus dem Buch "The Art of Unit Testing" von Roy Osherove zu folgen.

Anwendungsfälle

Anwedungsfalldiagramm



Anwendungsfallbeschreibungen

Titel	**Alle Artikel anzeigen**
Scope:	System
Level:	User-goal
Aktoren:	User
Kurzbeschreibung:	Der User lässt sich alle Artikel anzeigen.

Preconditions:	Keine
Hauptszenario:	Der User lässt sich alle vorhandenen Artikel anzeigen. Diese Anzeige der einzelen Artikel in dieser Übersicht beinhaltet zumindest Namen, Preis, eine Kurzbeschreibung sowie eine Kategorie, zu der ein Artikel gehört.
Fehlerszenario:	Kann keine Verbindung zur Datenbank hergestellt werden erhält der User eine Fehlermeldung.
Nachbedingung:	Der User kann nach Artikeln suchen und einzelne Artikel ändern bzw. löschen.

Titel	**Artikel suchen**
Scope:	System
Level:	User-goal
Aktoren:	User
Kurzbeschreibung:	Der Nutzer schränkt die Anzahl der angezeigten Artikel ein.
Preconditions:	Keine
Hauptszenario:	Der User sucht einen bestimmten Artikel. Dabei soll die Auswahl über mindestens ein Attribut (z.B. Name) erfolgen.
Fehlerszenario:	Keines.
Nachbedingung:	Der User kann einzelne Artikel ändern bzw. löschen.

Titel	**Artikel ändern**
Scope:	System

Level:	User-goal
Aktoren:	User
Kurzbeschreibung:	Der User ändert die Daten eines Artikels.
Preconditions:	Keine.
Hauptszenario:	Der User kann Beschreibung, Kategorie und Preis eines Artikels ändern. Auch das Bild kann geändert werden. Der Name muss nicht geändert werden können. Der Vorgang soll auch abgebrochen werden können. In diesem Fall soll der Nutzer verständigt werden, falls Änderungen stattgefunden haben. Die entsprechenden Daten innerhalb von Rechnungen dürfen sich dadurch nicht verändern.
Fehlerszenario:	Gibt der Nutzer ungültige Daten ein wird eine entsprechende Fehlermeldung angezeigt. Können die Daten aufgrund eines Datenbankfehlers nicht gespeichert werden soll eine entsprechende Fehlermeldung angezeigt werden.

Titel	**Einzelne Artikel löschen**
Scope:	System
Level:	User-goal
Aktoren:	User
Kurzbeschreibung:	Der User löscht einen Artikel.
Preconditions:	Keine.
Hauptszenario:	Der User wählt einen Artikel aus der gelöscht werden soll. Das System sollte eine Bestätigung anfordern. Bestätigt der User den Vorgang wird der Artikel aus der Datenbank gelöscht.
Fehlerszenario:	Kann keine Verbindung zur Datenbank hergestellt werden erhält der User eine Fehlermeldung.

Titel	**Statistik anzeigen**
Scope:	System
Level:	User-goal
Aktoren:	User
Kurzbeschreibung:	Der User fordert eine Graphik an, die zeigt wie oft einzelne Artikel verkauft wurden.
Preconditions:	Keine
Hauptszenario:	Der Nutzer lässt sich zu einem oder allen Artikeln anzeigen, wie oft diese verkauft wurden. Dabei soll sich der Zeitraum beschränken lassen, das heißt nur Verkäufe aus den letzten x Tagen werden berücksichtigt. Die Darstellung soll mithilfe eines geeigneten Charts erfolgen.

Titel	**Preisanpassung der Artikel**
Scope:	System
Level:	User-goal
Aktoren:	User
Kurzbeschreibung:	Der User möchte die Preise mehrerer Artikel anpassen.
Preconditions:	Keine.

Hauptszenario:	Basierend auf verschiedenen Kriterien, die der User spezifiziert sollen Artikel ausgewählt werden. Der Preis dieser Artikel soll entsprechend der Auswahl des Nutzers um einen Absolutbetrag oder einen prozentuellen Wert erhöht oder verringert werden. Die Auswahlkritierien sollen mindestens umfassen: Artikeln die in den letzten x Tagen mindestens y mal verkauft wurden, Artikel die in den letzten x Tagen mehr als y mal verkauft wurden, Auswahl der x meistverkauften Artikel in den letzten y Tagen, Auswahl der x am wenigsten verkauften Artikeln in den letzten y Tagen. Der User soll eine Meldung erhalten ob der Vorgang durchgeführt werden soll und es soll ausgegeben werden, wie viele Artikel von den Änderungen betroffen wären.
Fehlerszenario:	Gibt der Nutzer ungültige Preisänderungen an (Reduktion um mehr als 100%) soll eine entsprechende Fehlermeldung ausgegeben werden.

Titel	**Alle Rechnungen anzeigen**
Scope:	System
Level:	User-goal
Aktoren:	User
Kurzbeschreibung:	Der User kann eine Übersicht aller vorhandenen Rechnungen anzeigen.
Preconditions:	Keine
Hauptszenario:	Der User zeigt eine Übersicht über alle vorhandenen Rechnungen an. Die Übersicht soll relevante Daten (wie z.B. Ausststellungsdatum) beinhalten.
Fehlerszenario:	Kann keine Verbindung zur Datenbank hergestellt werden erhält der User eine Fehlermeldung.
Nachbedingung:	Der User kann nach Rechnungen suchen, Rechnungen erstellen und einzelne Artikel ändern.

Titel	**Rechnungen suchen**
Scope:	System
Level:	User-goal
Aktoren:	User
Kurzbeschreibung:	Der User kann nach Rechnungen suchen.
Preconditions:	Keine
Hauptszenario:	Der Nutzer soll nach einzelnen Rechnungen suchen können. Die Suche soll dabei über mindestens ein Attribut (z.b. Ausstellungsdatum) erfolgen.
Fehlerszenario:	Keines.
Nachbedingung:	Der User kann nach Rechnungen erstellen und einzelne Artikel ändern.

Titel	**Rechnungsdetails anzeigen**
Scope:	System
Level:	User-goal
Aktoren:	User
Kurzbeschreibung:	Der User kann Details einzelner Rechnungen einsehen.
Preconditions:	Keine
Hauptszenario:	Der Nutzer soll sich alle Daten einer bestimmten Rechnung ansehen können. Dabei sollen auch eine Liste aller Artikel, die in Rechnung gestellt wurden angezeigt werden. Der Gesamtbetrag der Rechnung soll auch angezeigt werden.

Fehlerszenario:	Kann keine Verbindung zur Datenbank hergestellt werden erhält der User eine Fehlermeldung.
Nachbedingung:	Keine.

Titel	**Rechnung erstellen**
Scope:	System
Level:	User-goal
Aktoren:	User
Kurzbeschreibung:	Der User kann neue Rechnungen erstellen.
Preconditions:	Keine
Hauptszenario:	Der Nutzer soll eine Auswahl von Artikeln in Rechnung stellen können. Dabei sollen relevante Daten wie z.B. Kundenname und Adresse angegeben werden. Der Gesamtbetrag soll angezeigt werden. Das erstellen einer Rechnung soll, falls alle Eingaben korrekt sind, nochmals vom Nutzer bestätigt werden.
Fehlerszenario:	Versucht der Nutzer eine Rechnung mit ungültigen Daten zu erstellen so soll eine entsprechende Fehlermeldung ausgegeben werden.
Nachbedingung:	Der User kann nach Rechnungen erstellen und einzelne Artikel ändern.