

Department of Computer Science Institute of Software- and Mutlimedia-Technology

Technical Reference

Salespoint 5

Christopher Bellmann, Thomas Dedek, Stanley Förster,

Paul Henke, Hendrik Neubert, Hannes Weisbach

Technische Universität Dresden

Department of Computer Science

Institute of Software- and Multimedia-Technology

Software Technology Group

3rd April 2012

2nd, revised edition

© 2014

Address

Department of Computer Science Telephone: 0351 463 38442

Institute of Software and Multimedia Technology Fax: 0351 463 38459

TU Dresden birgit.demuth@tu-dresden.de

01062 Dresden http://tu-dresden.de

Content

[2 Preface 4](#_Toc394527956)

[2.1 Typographic Conventions 4](#_Toc394527957)

[2.2 Introduction 4](#_Toc394527958)

[3 Technical Background 5](#_Toc394527959)

[3.1 JPA – Java Persistence API 5](#_Toc394527960)

[3.2 Spring 6](#_Toc394527961)

[3.3 JodaMoney 6](#_Toc394527962)

[3.4 Software Architecture of a Salespoint 5 Application 6](#_Toc394527963)

[3.5 General Design Decisions and Aspects of Salespoint 5 7](#_Toc394527964)

[3.5.1 Notes on Interfaces … 7](#_Toc394527965)

[3.5.2 … implementing Classes, and Naming 7](#_Toc394527966)

[3.6 Configuration of Salespoint 5 8](#_Toc394527967)

[3.7 DataInitilizer or filling Application with Test-Data 8](#_Toc394527968)

[4 Salespoint 5 Components 9](#_Toc394527969)

[4.1 Useraccount 9](#_Toc394527970)

[4.1.1 UserRole 9](#_Toc394527971)

[4.1.2 Login 9](#_Toc394527972)

[4.1.3 Limitation 9](#_Toc394527973)

[4.2 Product 9](#_Toc394527974)

[4.2.1 Product Identifier 9](#_Toc394527975)

[4.2.2 ProductFeature 10](#_Toc394527976)

[4.3 Catalog 10](#_Toc394527977)

[4.4 Accountancy 10](#_Toc394527978)

[5 Bibliography 12](#_Toc394527979)

# Preface

## Typographic Conventions

Two typographic conventions are employed throughout this document to highlight specific phrases. The following paragraphs describe when and why these highlightings are used:

Keywords

The Keyword font is used to denote variable names, class names, type names, java keywords, java package names and so forth.

Termini

Proper names and termini ate printed in termini font.

## Introduction

The Salespoint Framework is intended to minimize developing effort of point-of-sale applications. Salespoint 2010 users complainted about complexity, missing features and bugs. Thus, the decision was made to re-design and re-implement the framework from scratch. In 2013 some parts again were re-implemented with components of the spring project. Our development goal was an easy-to-use framework primarily targeted for educational purposes. As such Salespoint 5 is not tailored to any specific application, but designed with a wide area of applications in mind.

Models and design patterns employed in Salespoint 5 are inspired by “Enterprise Patterns and MDA: Building Better Software with Archetype Patterns and UML” by Jim Arlow [1]. An overview of the functionality of the new features in Salespoint 5 are detailed in this document.

We would like to thank all Salespoint users who submitted their feedback and encourage future users of Salespoint 5 to do the same.

# Technical Background

One of the main reason to use a framework such as Salespoint for educational purpose is to teach students reusability. The Salesoint 5 developers also adhere to that principle. Thus, Salespoint 5 itself uses a number of frameworks and APIs, which are introduced briefly. The software architecture of Salespoint 5 applications is also detailed.

## JPA – Java Persistence API

One of the key features of Salespoint 5 is its integrated persistence layer. The goal is to allow data persistence, while minimising programming effort and training period as well as maximising flexibility for framework users.

The obvious choice was the Java Persistence API (JPA), a Java framework managing relational data in Java Standard Edition or Enterprise Edition applications. Salespoint 5 uses JPA 2.0, developed under JSR 317 and finished in Dec, 2009 [2].

Additionally to the API itself, which is defined in the javax.persistence package, JPA also consists of Persistence Entities, ORM Metadata and the Java Persistence Query Language (JPQL).

A persistence entity is usually a Plain Old Java Object (POJO), which is mapped to a single table in a database. A row in such a database table corresponds to a specific instance of such an entity. Relational data between entities (and therefore tables) may be specific as annotations in Java source code. Salespoint 5 uses annotations to provide object/relational metadata.

Persistence entities may be related to each other by an inheritance hierarchy. A persistence entity may have a non-persistent superclass. Fields declared by a non-persistent superclass are not stored in the database if an inheriting entity is persisted. Three schemas exist to persist entities with an inheritance relationship: single table, join table and table per class.

The single table strategy stores all instances of classes of an inheritance hierarchy in the same table. The table contains columns for every attribute a persistence entity in the hierarchy declares. The different types are distinguished by a type discriminator column. The discriminator value for each persistence entity in an inheritance hierarchy is generated automatically or can be supplied by the user.

The join table strategy uses a table for the root persistence entity of the inheritance hierarchy. Additionally, a table is added for each persistence entity in the inheritance hierarchy. In the supplementary tables, a foreign key is used to reference a row in the table of the parent persistence entity. Each table contains only columns for fields declared by a specific persistence entity in the inheritance hierarchy, but neither for the entities children nor parents. To reconstruct an object from the database, the different tables have to be joined using this foreign key, thus the name of this strategy.

The table per class strategy creates a table for each persistence entity containing all fields of the class, including inherited fields.

The inheritance strategy of an inheritance hierarchy has to be declared at the root persistence entity. The inheritance strategy may not be changes for a sub-hierarchy, because JPA 2.0 does not require this feature. JPA 2.0 only requires the single table and join table strategies to be implemented. Salespoint 5 uses the single table strategy exclusively.

The query language JPQL, which is similar to SQL, is used to retrieve entity information from the database. In contrast to SQL, JPQL queries act on entity objects instead of database tables. JPA implamentations translate a JPQL statement to SQL statement at run time. It is possible to replace the database management system (DBMS) while keeping the Java classes. Furthermore it is possible to interface directly with the DMBS using Native Queries. Salespoint 5 however, uses the Criteria API [2] [3] to facilitate type safe querying.

Multiple implementations of JPA 2.0 exist, for example TopLink [4] and EclipseLink [3]. The open source persistence and ORM framework Hibernate [5] also supports JPA 2.0. Salespoint 5 uses the JPA 2.0 reference implementation, EclipseLink. No implementation specific code is used in Salespoint 5, therefore it should be possible to interchange EclipseLink with another JPA 2.0 implementation[[1]](#footnote-1).

## Spring

In contrast to earlier versions of the Salespoint Framework, Salespoint 5 obeys the MVC pattern. Salespoint 5 can be seen as the Model of an MVC application, no parts of the View or the Controller are implemented in the framework.

Salespoint 5 is designed as basis for development of web applications, using the Spring Framework [6] to implement Views and Controllers. To further ease the development, Salespoint 5 includes property editors to convert string based representations to Salespoint 5 identifier types. Furthermore, JSP tags to check, if a user is logged in and if a user has a certain capability are included in the framework.

As a big new approach in development with JPA, the Spring Framework with its repository interfaces can make the work very fast, clean and easy. The Crudrepository provides the basic methods for working with the database (CRUD stands for Create, Read, Update and Delete).

## JodaMoney

In Salespoint 5.3.X money class and its related components were replaced by the Joda-Money [7] project. Prices and other money values are represented as org.joda.money.Money or for more precisison as org.joda.money.BigMoney objects.

Due to the fact, that all representing objects are immutable, all arithmetic functions produce a new object (see Listing 1).

Money value\_1 **=** Money**.**parse**(**“USD 23.07”**);**

Money value\_2 **=** Money**.**parse**(**“USD 18.07”**);**

Money sum **=** value\_1**.**plus**(**value\_2**);**

Listing 1 Example of Money Operations

Joda-Money also supports Currency (see Listing 1). A set of loaded currencies is provided by an instance of CurrencyUnitDataProvider. But new and funky CurrencyUnits can be created to. So with this currencies, money values can be converted from one to another currency.

## Software Architecture of a Salespoint 5 Application

Software often need to be adaptive, flexible and extendable. Using a suitable architecture pattern, such as the Model-View-Controller (MVC) pattern, helps to meet these non-functional requirements. Figure 1 gives an overview of how a Salespoint 5 application is modelled. Salespoint 5, as domain framework, takes the place of the model in the MVC pattern. The model can be extended by sub-classing Salespoint 5 classes or by introducing entirely new classes. Salespoint 5 model classes and sub-classes thereof are transparently stored in a database. If new classes are added to the model and their state is also required to be persistent, the developer also has to facilitate persisting those objects using the JPA-API.

The controller and view are application-specific and have to be implemented by the user. Although Salespoint 5 does not require a specific framework or API like Swing [8] or SWT [9]. However, because Salespoint 5 is intended to be used in conjunction with the Spring MVC [6] framework for SWP at TU Dresden, Salespoint 5 contains supplementary code, easing the development of Spring applications. This supplementary code consists of Spring MVC PropertyEditors, and custom, Salespoint 5 specific JSP-Tags.

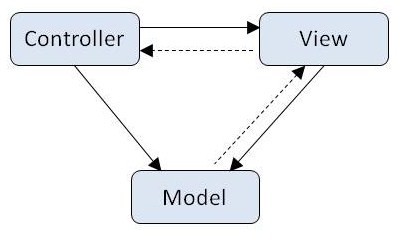


Figure 1 MVC-pattern of a Salespoint 5 application divided into application specific code and framework code.

## General Design Decisions and Aspects of Salespoint 5

This chapter summaries design decisions and aspects common to all Salespoint 5 subsection and details, why those decisions were made.

### Notes on Interfaces …[[2]](#footnote-2)

Integrating a persistence layer into the Salespoint 5 Framework had a great impact on some design decisions made during the development of Salespoint 5. Early on it became obvious that necessities of JPA could dictate the design and implementations of Salespoint 5.

To guard against JPA requirements influencing design decisions, Salespoint 5 strongly follows the programming against interfaces programming style. Although, creating an interface for almost every class violates the KISS (Keep it simple, Stupid! Also, a hard-rock band.) principle, the developers deemed programming against interfaces necessary because Salespoint 5 is intrinsically tied to JPA. Using interfaces allowed us to cleanly define the behaviour of an object, without relying on a specific implementation. The classes itself are, however, not programmed against interfaces. Salespoint 5 usually just implement interfaces, but refer to other classes directly. The reason for this violation of the programming against interfaces paradigm is the generic typing, which would be required, if Salespoint 5 classes would refer to interfaces instead of concrete classes. For example, the PersistentOrder class would require three (cascaded) generic type parameters alone. As a consequence, Salespoint 5 classes refer to each other directly, instead of interface types.

### … implementing Classes, and Naming[[3]](#footnote-3)

Objects, which need to be persisted to safe the current state of an application, are called persistence entities. Usually, a persistence entity is a POJO (Plain Old Java Object). However

## Configuration of Salespoint 5

The configuration for an application can be modified on the application class in the root package (e.g. videoshop.Videoshop for the videoshop project).

Methods, annotated with @Configuration, will be scanned at the beginning of the deployment on the application server (in this case Spring Boot). This configuration files will tell the application server the settings for the application.

By overriding the configuration method, you can specify the login behaviour or security functions. For a basic login strategy the videoshop is a good start. There you can see, that with authorizeRequests() an authorization will be set. Following by matchers, you can specify the pages, this authorization is made on.

Further, you can easily set an login page with formLogin() and the path to the login page with loginProcessingUrl("/login"). Analogue the logout settings works like login system.

## DataInitilizer or filling Application with Test-Data

Every Application should be tested, so an easy way is, to use test data. So for this goal a recommend solution is, to load on every start of the application a bundle of test data into your system.

As shown in the Videoshop project, a Datainitilizer class is registered and creates a lot of data and finally adds this data to the application. For that purpose the Salespoint has an Interface org.salespointframework.core.DataInitializer, which can be used.

# Salespoint 5 Components

## Useraccount

To manage system accounts, Salespoint 5 has a notation of a user in the form of the User interface. Users are managed by the UserAccountManager, who is also an interface. Every user is uniquely identified by a UserIdentifier, which also serves as primary key attribute for the database. The UML model is depicted in Figure 3.3.

### UserRole

Roles in conjunction with authorization tag hasRole()[[4]](#footnote-4) can be used to change the appearance of a View, depending on a user’s status. For example, a View for a user having an “administrator” role may display different content, for example delete buttons, than for a user not having that role. Thus, roles allow for flexibility and assist in code reuse, when designing the View.

### Login

To reduce code repetition, Salespoint 5 contains code to automate user log in. Using a JSP template, a special login form is generated, which is handled by an interceptor[[5]](#footnote-5). The interceptor verifies the user password and associates the current session with the user using <login> and <logoff>. The session is required, because multiple users can be logged on at the same time.

### Limitation

Due to the fact, that Salespoint use the SecuirtyContext for authentication, the user account cannot so easily be extended. The useraccount provides a username, a password, a list of roles and a boolean, if the useraccount is enabled or not. If this properties don’t meet all requirements, the useraccount has to be expand its features as shown in videoshop project.

## Product

Salespoint 5 is intended as framework for point-of-sale applications. The items for sale are called “products” and represented by instances of classes who implement the Product interface.

A general overview of the Salespoint 5 products subsystem is given in Figure \ref{product\_overview}. To represent different kinds of products, PersistentProduct can be sub-classed; see Section 3.1 for more information.

PersistentProducts are aggregated by PersistentCatalog (see Section \ref{sec:catalog}).

// Bild product\_overview

### Product Identifier

Products are supposed to be an abstraction, like an item on display or a picture in a catalog. ProductInstances are used to represent the actual item you get, when you a buy a product. Products are identified using a ProductIndentifier, whereas ProductInstances are identified by a SerialNumber. ProductInstances can be thought of as identifiable instances of a certain product, which are identical apart from their SerialNumber.

### ProductFeature

To conviently handle products, which are essentially the same but differ in certain aspects, such as color or size Salespoint 5 has the concept of a ProductFeature.

ProductFeatures are specified by a featureType, for example color or size, and a corresponding value, for example “black” or “blue” for the feature “color”.

Additionally a ProductFeature may reference a Money object, to describe an increase or decrease in price of a Product, if it has a certain ProductFeature.

Alternatively, a change in price can be expressed as a percentage of the price of the Product.

An example: A class Shoe extends PersistentProduct and has a Set<ProductFeature> containg the values 36, 37, 38, 39, 40, 41, 42, 43, 44, 45 of the productType} “size”.

The set of ProductFeatures declared in PersistentProduct defines, which ProductFeatures can be aggregated by the corresponding ProductInstance.

An instance of Shoe represents a specific model a vendor might have.

Additionally, a class ShoeInstance may sub-class PersistentProductInstance.

An instance of ShoeInstance represents a specific pair of shoes.

ProductInstance also aggregates ProductFeatures, but in contrast to Product exactly one ProductFeature is allowed for any featureType.

In other words: a shoe has a size - exactly one size.

## Catalog

The Catalog interface was designed to manage Products and ProductFeatures in the system.

It provides functionality to add, remove, and find Products.

Products can be searched by their name or category.

Products and ProductFeatures are more closely described in Section 4.2.

The PersistentCatalog is an implementation of the Catalog interface.

Additionally PersistentCatalog provides an update()-method to update and merge existing PersistentProducts to the database.

The find() methods request the database in the form of CriteriaQuerys which will be processed by JPA and results are returned in the form of Iterables. The reason for this is to make returned objects immutable without making it difficult to iterate over these results.

## Accountancy

The accountancy package contains functionality supporting book keeping.

AccountancyEntry is a representation of an accounting entry.

Accountancy aggregates AccountancyEntrys.

Every AccountancyEntry is uniquely identified by an AccountancyEntryIdentifier.

PersistentAccountancyEntry implements AccountancyEntry and serves as persistence entity, while PersistentAccountancy implements Accountancy and provides transparent access to the JPA layer.

AccountancyEntryIdentifier is used as primary key attribute, when entities are stored in the database.

By implementing and sub-classing the AccountancyEntry interface, the notion of different accounts, as known from double-entry bookkeeping, can be realised.

As can be seen in Figure Bild{accountancy\_overview}, PersistentAccountancyEntry is sub-classed to create a second “account” used to store payment information, namely ProductPaymentEntry.

Payment information also includes a user identifier referencing the buyer, an order identifier referring to the \code{Order} which was payed, and a PaymentMethod describing the money transfer.

The attributes are named userIdentifier, orderIdentifier, and paymentMethod respectively.

The inheritance hierarchy of PaymentMethod is depicted in Figure Bild{payment\_overview}.

To create a new account, AccountancyEntry has to be sub-classed.

Every (persisted) object of such a class belongs to the same account.

Accessing per-account entries is facilitated by specifiying the desired class type when calling get() or find() methods of Accountancy as explained in Section 3.1.

# Bibliography

|  |  |
| --- | --- |
| [1] | J. Arlow, Enterprise Patterns and MDA: Building Better Software with Archetype Patterns and UML, Addison-Wesley Professional, December 22, 2003. |
| [2] | J. C. Process, „JSR-000317 Java Persistence 2.0,“ [Online]. Available: https://jcp.org/aboutJava/communityprocess/final/jsr317/. [Zugriff am 30 Mai 2014]. |
| [3] | Eclipse.org, „Eclipselink,“ [Online]. Available: https://www.eclipse.org/eclipselink/. [Zugriff am 30 Mai 2014]. |
| [4] | Oracle, „Oracle TopLink,“ [Online]. Available: http://www.oracle.com/technetwork/middleware/toplink/overview/index.html?ssSourceSiteId=otncn. [Zugriff am 30 Mai 2014]. |
| [5] | RedHat, „Hibernate. Everything data.,“ [Online]. Available: http://hibernate.org/. [Zugriff am 30 Mai 2014]. |
| [6] | S. Framework, „Spring Framework,“ [Online]. Available: http://projects.spring.io/spring-framework/. [Zugriff am 30 Mai 2014]. |
| [7] | Joda.org, „Joda-Money,“ [Online]. Available: www.joda.org/joda-money. [Zugriff am 18 Mai 2014]. |
| [8] | Oracle, „Swing,“ [Online]. Available: http://docs.oracle.com/javase/7/docs/technotes/guides/swing/index.html. [Zugriff am 5 Juni 2014]. |
| [9] | Eclipse.org, „SWT: The Standard Widget Toolkit,“ [Online]. Available: https://www.eclipse.org/swt/. [Zugriff am 5 Juni 2014]. |

1. This document will be updated, as soon as another JPA provider is tested with Salespoint 5. [↑](#footnote-ref-1)
2. continued in the next sub-section [↑](#footnote-ref-2)
3. started at the preceeding sub-section [↑](#footnote-ref-3)
4. Expression-Based Access Control, Part IV. Authorization [http://docs.spring.io/spring-security/site/docs/3.0.x/reference/el-access.html] [↑](#footnote-ref-4)
5. An Interceptor is a Spring concept. [↑](#footnote-ref-5)