Version	Processing Date	Author
1.0	October 29th, 2019	Daniel Schoenicke

Developer Documentation

1. Introduction and Goals

1.1. Task Definition

Note: This task aims to provide an example for the Videoshop. It is written from the perspective of the client of this project (Chair of Software Technology) and therefore can be seen as a requirements specification. You usually cannot expect such a document to be complete or even consistent, which is why you should always ask in case of uncertainty. Wherever information about the domain is provided, we used italic to show the representation in the domain model.

The times when people went to buy their movies physically in the store are mostly over. As we, the Chair of Software Technology, have always had our little secondary business of selling movies to students, this change affects us as well. Therefore, we finally want to

take the leap to move our business into an online shop. We need a software, which can support all of the core aspects of our current shop and automatize processes wherever possible.

Our Shop (*Videoshop*) can have any number of users (*User*) which may interact with it differently. Every visitor of our shop may access the catalog (*VideoCatalog*) and its whole functionality. The catalog contains every article (*Disc*) we offer and distinguishes between DVDs (*Dvd*) and Blu-Rays (*BluRay*). The Discs are stored in an inventory (*Inventory*), where they are represented by items (*InventoryItem*). An item saves the current stock (*quantity*) of the Disc. Whenever something is sold, the quantity of the item has to be reduced (*decreaseQuantity*) accordingly to represent the stock correctly.

Besides normal users in our system (*Customer*), we also want administrative access (*Boss*) to manage our shop. Whenever a customer likes something from our catalog, he may add it (*addDisc*) to his virtual basket (*Cart*, *CartItem*) in any quantity (*quantity*) . The contrary is obviously desired as well, to allow our customers to change their mind (*removeDisc*). During the whole process, the customer shall obviously be able to view his selection and see total price of it (*getPrice*). We are especially interested in the automation of the ordering process, which is why we require support for directly buying the content of the cart (*buy*).

After deciding to buy something, an order (*Order*) with the current time (*dateCreated*) is created. It contains each of the chosen items (*OrderLine*) with their quantity (*quantity*) and price (*price*). If the chosen item is not available in the sufficient quantity (*hasSufficientQuantity*), an error should be shown to the customer. As long as the customer did not pay, the order is registered in the system and assigned a status (*OrderStatus*), but not yet processed (*OPEN*). After receiving the money from the customer (*payOrder*), which he may provide through different methods (*paymentMethod*), the order may be processed further (*PAID*). As the order is shipped to the customer, it should be archived (*completeOrder*, *COMPLETED*), as returns or refunds are ruled out :). Should any unforeseen circumstances occur, we obviously do not want an order to be stuck in the system (*CANCELLED*).

Our shop should obviously provide the means for a visitor to register (*register*). As we do only want registered users to have access to some functionality, a security system is required. We do trust the state-of-the-art authentication mechanism with e-mail (*email*) and a password (*password*). However, as we do not want to force visitors to register, they shall be able to leave a comment (*addComment*) with their opinion (*text*) and a rating (*rating*) for every disc in the catalog.

All in all, we want a nice, fast and secure system, which allows us to administrate all of our customers and the stock. It should support our ordering process and allow us to manage everything related to it. The user experience should be awesome, with a beautiful user interface and a layout, which boosts our sales.

1.2. Quality Demands

To measure the quality of the application, quality demands have to be defined, which have to be fulfilled. _Note: The following descriptions are derived from the ISO/IEC 25010 Software Quality Model.

Maintainability
This characteristic represents the degree of effectiveness and efficiency with which a product or system can be modified to improve it, correct it or adapt it to changes in environment, and in requirements.

Usability
Degree to which a product or system can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use.

Security
Degree to which a product or system protects information and data so that persons or other products or systems have the degree of data access appropriate to their types and levels of authorization.

The following table shows what quality demands have to be fulfilled to which extent. The first column lists the quality demands, while in the following columns an "x" is used to mark the priority.

1 = Not Important .. 5 = Very Important

Quality Demand	1	2	3	4	5
Maintainability				x	
Usability			x		
Security				x	

2. Constraints

2.1. Hardware Specifications

A list of necessary devices / hardware to run and use the application.

- Server

- Computer
- Keyboard
- Mouse

2.2. Software Specifications

A list of necessary software to run and use the application.

The following (or newer) Java version is necessary to run the application:

- Java 11

The following (or newer) browser versions are necessary to use the application:

- Internet Explorer / Edge 10.0
- Firefox 4.0
- Google Chrome 4.0
- Opera 9.6

2.3. Product Usage

This section is going to give an overview of how the product is intended to be used upon completion and under which circumstances.

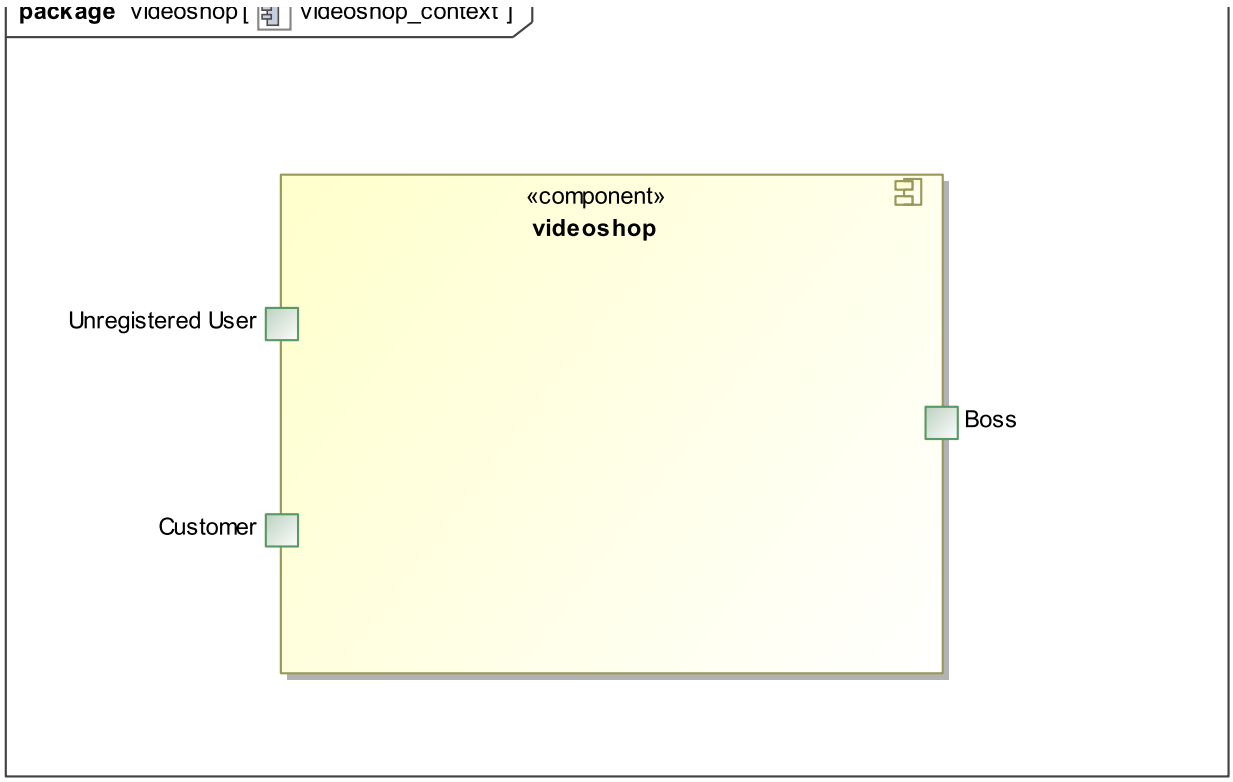
The system is going to be used as a web shop by the Chair of Software Technology to sell movies (discs) to students. The software is supposed to run on a server and be available through the internet (via a browser) to interested customers 24/7.

The primary users of the software are students (customers), who supposedly know typical website navigation schemas, as well as administrators (Boss), who do not necessarily have a technical background.

The system shall not need technical maintenance, as the staff of the Chair of Software Technology already has its hands full. Any data shall be stored persistently in a database and be accessible through the application (e.g. no SQL knowledge should be required for a boss).

3. Context and Scope

3.1. Context Diagram



Note: Since Salespoint and Spring Security are used in the implementation, there are no external interfaces.

4. Solution Strategy

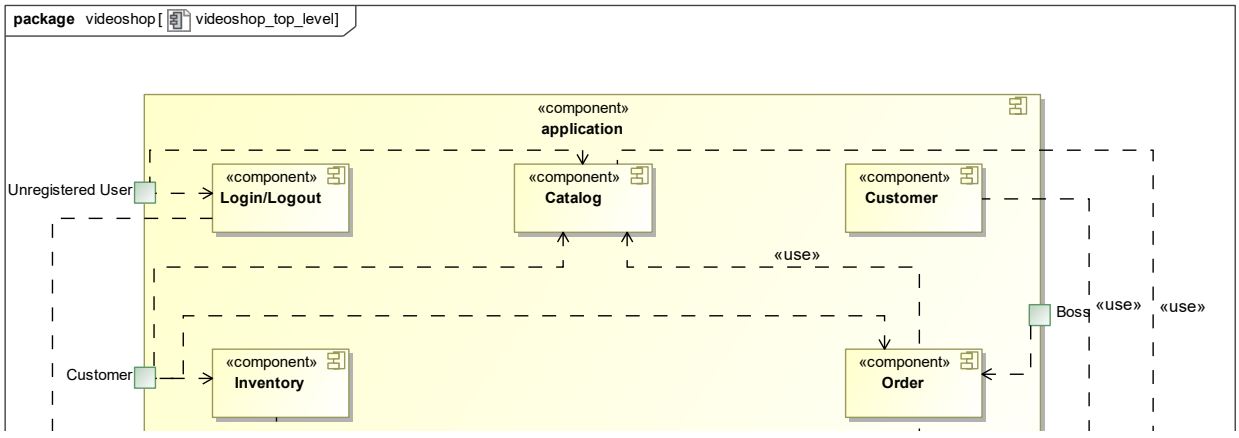
4.1. Quality Demand Fulfillment

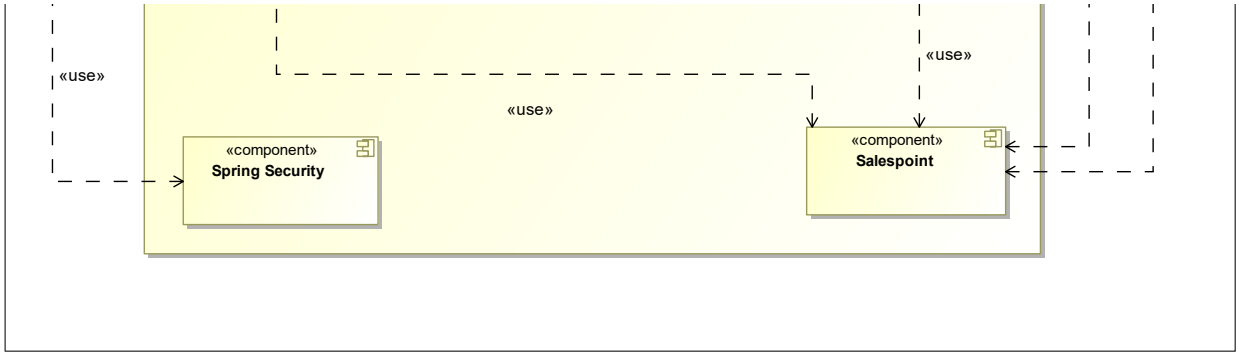
Note: The following table shows the previous defined quality demands and solution approaches to fulfill them.

Quality Demand	Solution approach
Maintainability	• **Modularity** Compose the application out of discrete components such that changes of a component have less impact on other components. • **Reusability** Ensure that components of the system can be reused by other components or systems. • **Modifiabilty** Ensure that the application can be modified or extended without introducing errors or degrading the product quality.

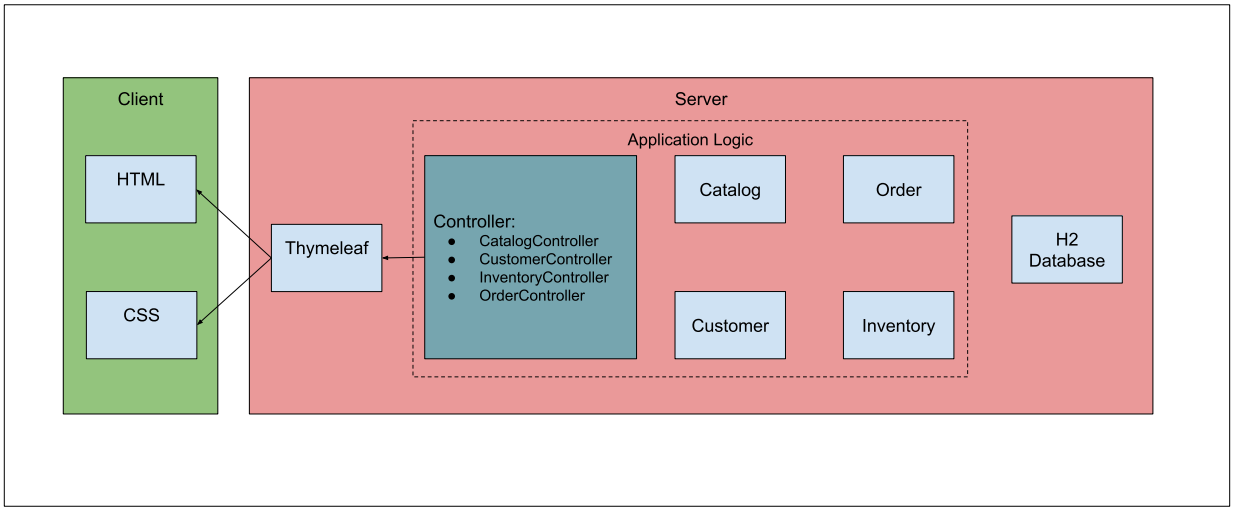
Quality Demand	Solution approach
Usability	• **Learnability** Ensure that the system can be easily used and understood by its users. This can be realized by e.g. unambiguously describing the content of inputs with labels or tooltips. • **User error protection / Error handling** Protect user against making errors. Invalid inputs must not lead to invalid system states. • **User interface aesthetics** Provide a pleasing and satisfying interaction for the user. • **Accessibility** Ensure that people with a wide range of characteristics can fully use the system. This can be realized by e.g. using suitable font sizes and color contrasts.
Security	• **Confidentiality** Ensure that only data can be only accessed by people who are authorized to access them. This can be realized with *Spring Security* and *Thymeleaf* (`sec:authorize` - tag). • **Integrity** Prevent unauthorized modification of data. This can be realized with *Spring Security* (`@PreAuthorize` - annotation). • **Accountability** Traceability of actions or event to a unambiguously entity or person. For this application, every `Order` should be linked to a `Customer`.

4.2. Software Architecture





Top Level Architecure of the application



Client Server Model of the application. The client only contains HTML and CSS files. The application logic is implemented on the server.

Note: JavaScript is compiled by the client. You can use JavaScript in your application but make sure, that you don't use it to implement any of the application logic!

HTML-Templates are rendered clientside with their corresponding CSS-Stylesheets. The data shown in the templates is provided by Thymeleaf. Thymeleaf receives the requested data by the controller classes, which are implemented in the backend. These controller classes on the other hand use instances and methods of the model classes. By default, an underlying H2 database saves data persistently.

4.3. Architecture decisions

4.3.1. Design Patterns

- Spring MVC

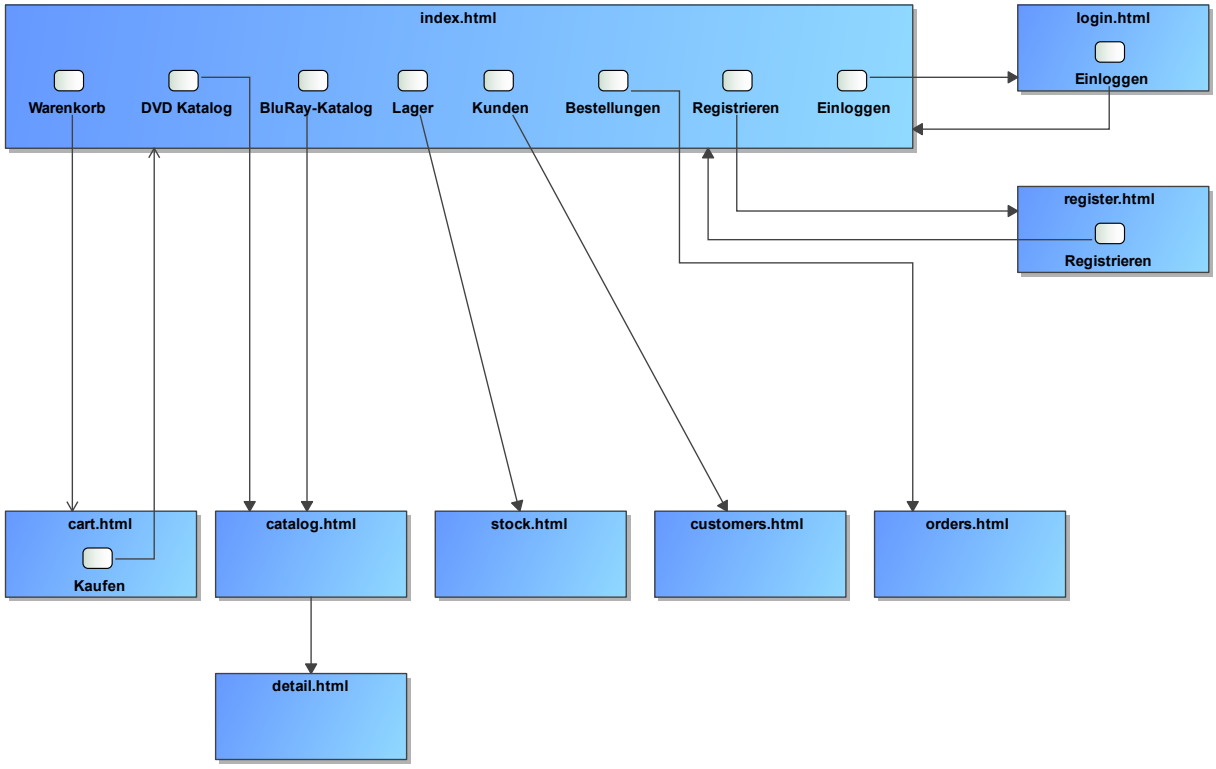
4.3.2. Persistence

The application uses **Hibernate annotation based mapping** to map Java classes to database tables. As a database, **H2** is used. The persistence is deactivated by default. To

activate persistence storage, the following two lines in the file *application.properties* have to be uncommented:

```
# spring.datasource.url=jdbc:h2:./db/videoshop
# spring.jpa.hibernate.ddl-auto=update
```

4.3.3. User Interface



Note: The blue boxes display a HTML-Template. The white boxes within the templates represent buttons, which redirect to the templates, their outgoing arrows point to.

4.4. Use of external frameworks

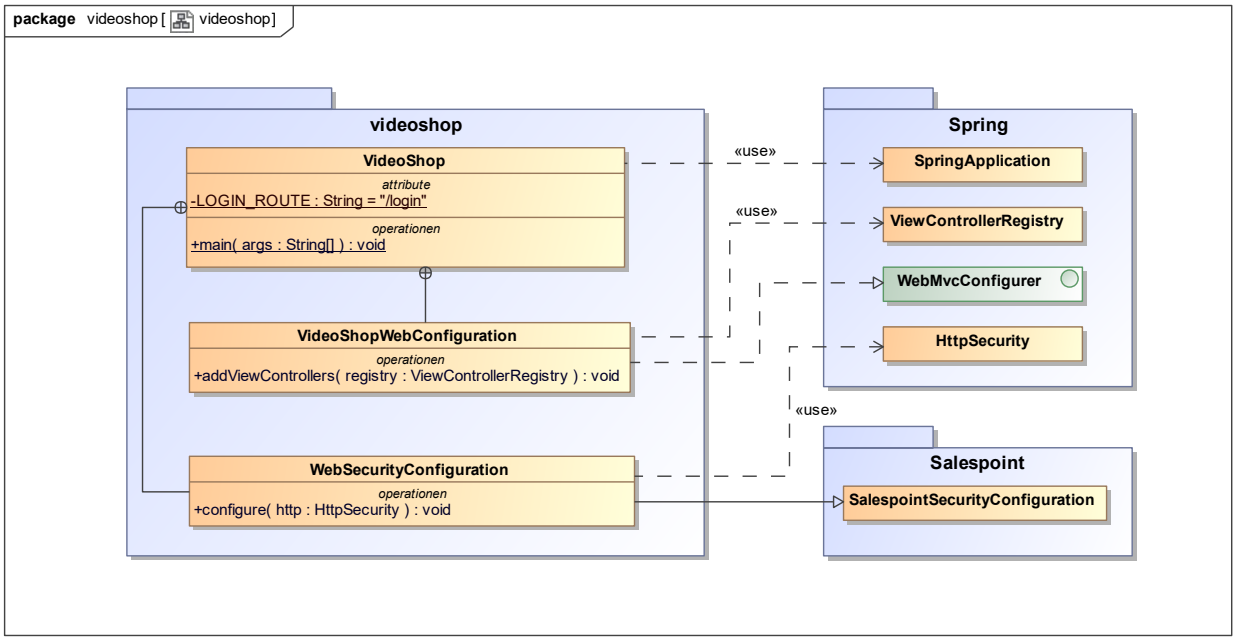
External package	Used by (applications'
salespointframework.catalog	• catalog.Disc • catalog.VideoCatalog • order.OrderController
salespointframework.core	• catalog.CatalogInitializer • customer.CustomerDatal

External package	Used by (applications'
	• inventory.InventoryInitial
salespointframework.inventory	• catalog.CatalogControlle • inventory.InventoryContr • inventory.InventoryInitial
salespointframework.order	order.OrderController
salespointframework.payment	order.OrderController
salespointframework.quantity	• catalog.CatalogControlle • inventory.InventoryInitial • order.OrderController
salespointframework.SalespointSecurityConfiguration	videoshop.WebSecurityConfi
salespointframework.time	catalog.CatalogController
salespointframework.useraccount	• customer.Customer • customer.CustomerDatal • customer.CustomerMana • order.OrderController
springframework.boot	videoshop.VideoShop
springframework.data	• catalog.VideoCatalog • customer.CustomerMana • customer.CustomerRepo
springframework.security	videoshop.WebSecurityConfi

External package	Used by (applications'
springframework.ui	• catalog.CatalogControlle • customer.CustomerContr • inventory.InventoryContr • order.OrderController
springframework.util	• customer.CustomerContr • customer.CustomerDatal • order.OrderController
springframework.validation	customer.CustomerControlle
springframework.web	videoshop.VideoShopWebCo

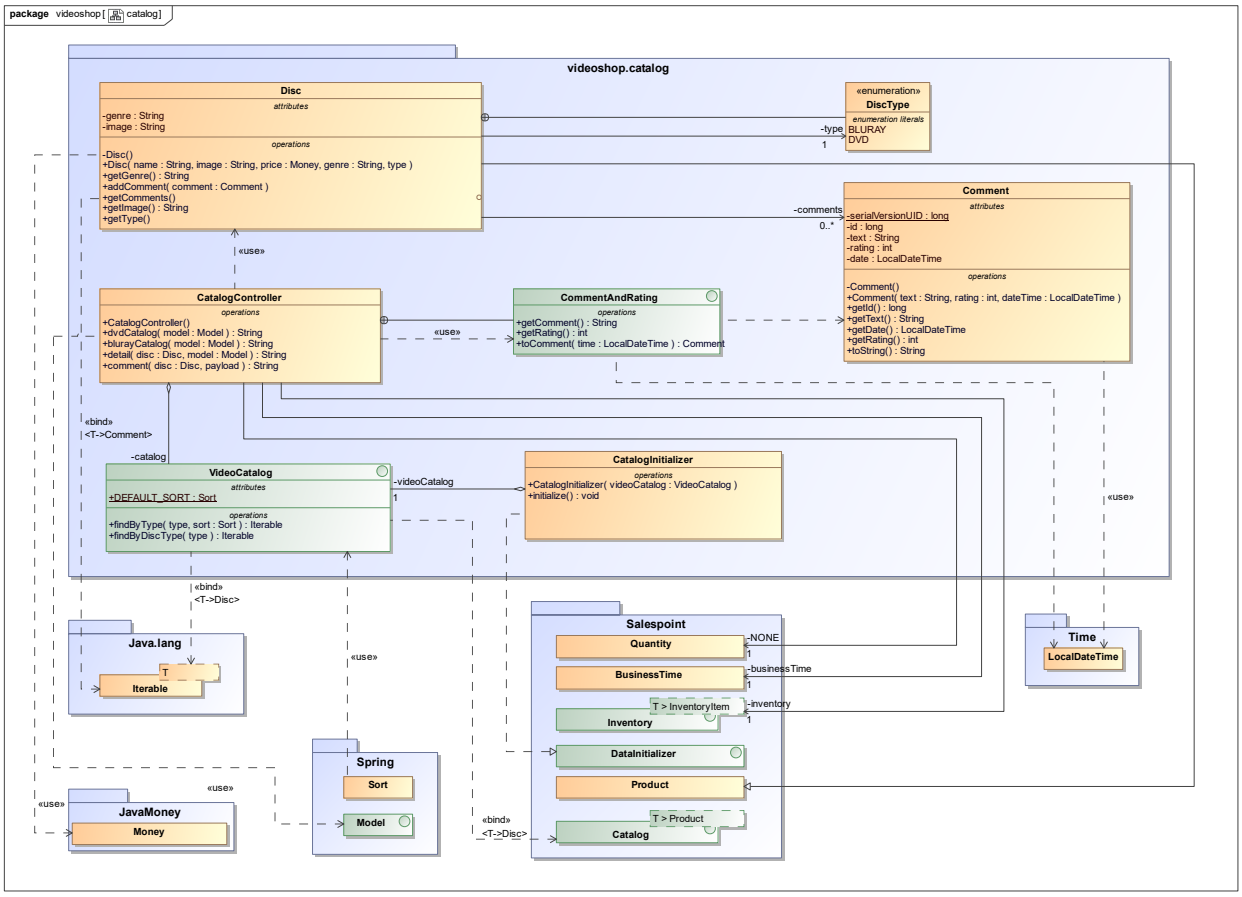
5. Building block view

5.1. Videoshop



Class/Enumeration	Description
VideoShop	The central application class to configure the Spring container and run the application
VideoShopWebConfiguration	Configuration class to route `/login` directly to the `login.html` template
WebSecurityConfiguration	Configuration class to set up basic security and login/logout options

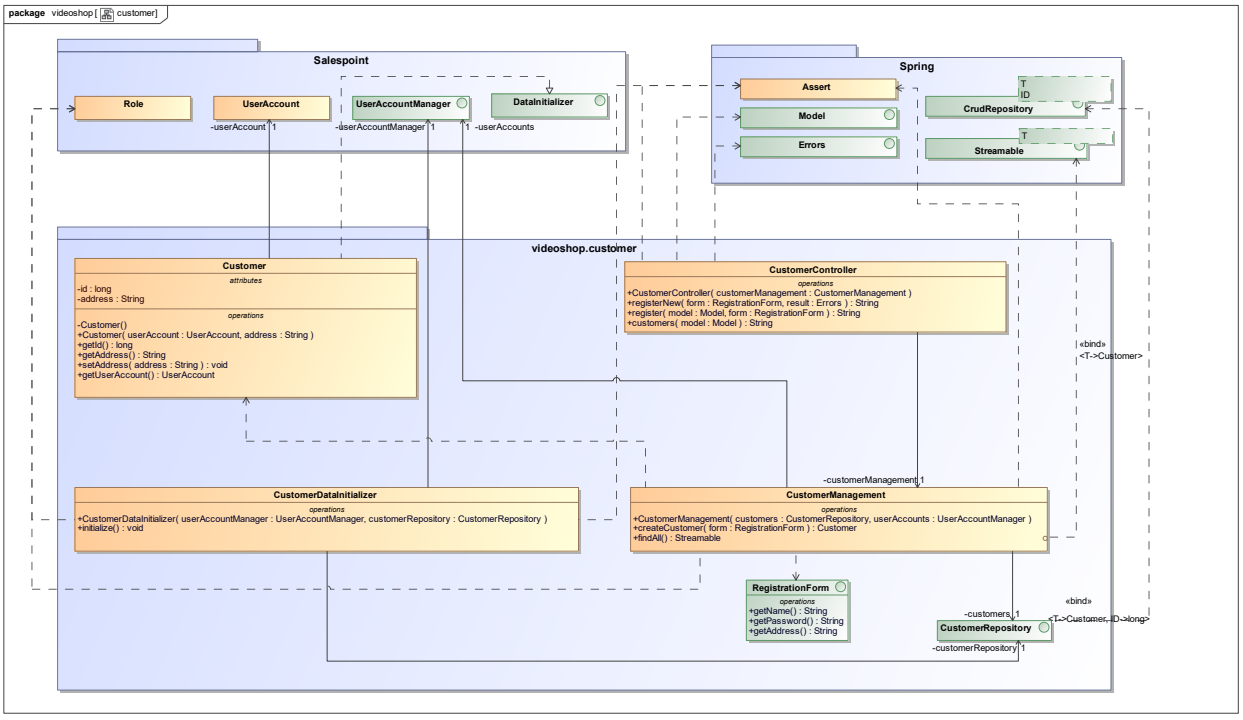
5.2. Catalog



Class/Enumeration	Description
CatalogController	A Spring MVC Controller to handle requests to show `Disc`s and create `Comment`s
CatalogInitializer	An implementation of the DataInitializer to create dummy DVDs and BluRays on application startup

Class/Enumeration	Description
Comment	A comment which can be written for a `Disc`
CommentAndRating	Describes the payload to be expected to add a comment
Disc	Class to describe BluRays and DVDs as the products of the videoshop
DiscType	Enumeration to define a `Disc` as a DVD or a BluRay
VideoCatalog	An extension of Salespoint.Catalog to add videoshop specific queries

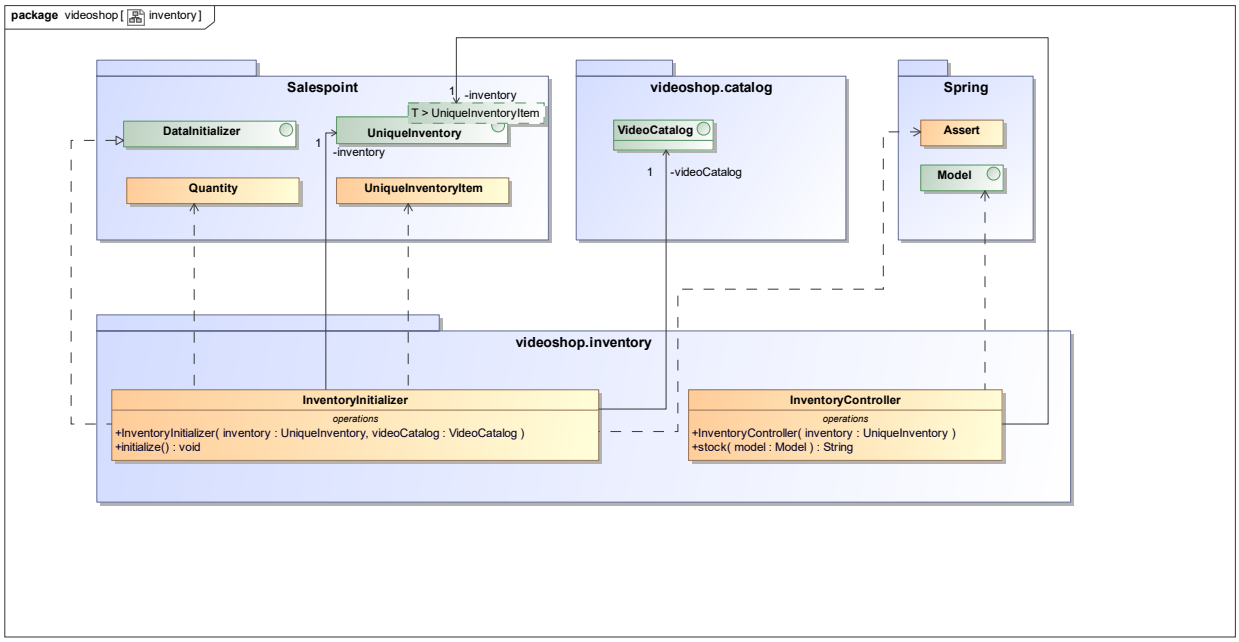
5.3. Customer



Class/Enumeration	Description
Customer	Custom class to extend the Salespoint-UserAccount with an address
CustomerController	A Spring MVC Controller to handle requests to register and show customers

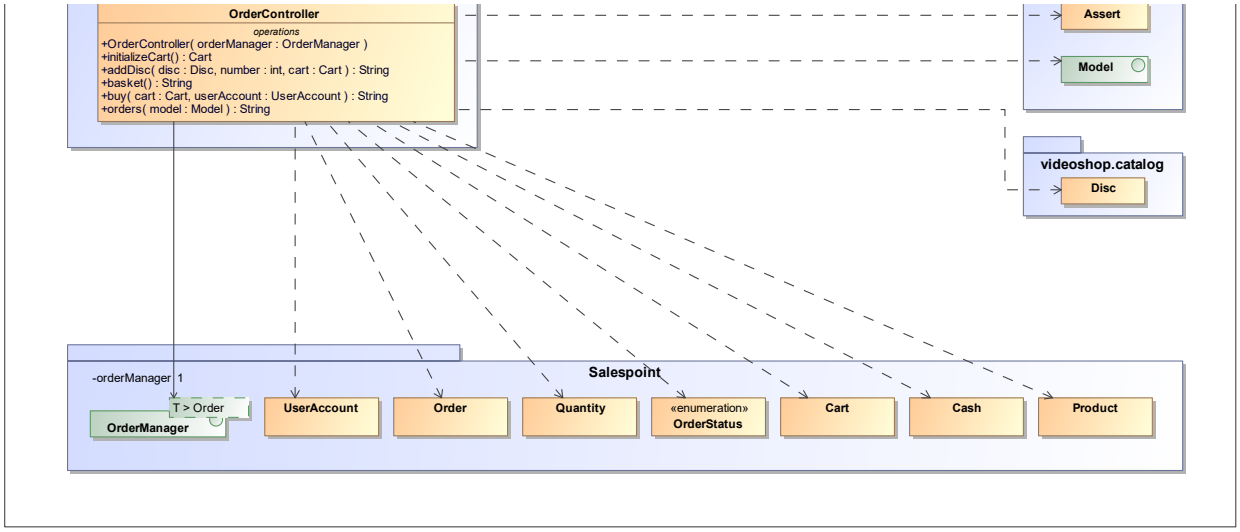
Class/Enumeration	Description
CustomerDataInitializer	An implementation of the DataInitializer to create dummy customers on application startup
CustomerManagement	Service class to manage customers
CustomerRepository	A repository interface to manage Customer-instances
RegistrationFrom	An interface to validate the user input of the registration formular

5.4. Inventory



Class/Enumeration	Description
InventoryController	A Spring MVC Controller to handle the request to show the stock of the shop
InventoryInitilalizer	An implementation of the DataInitializer to create dummy data on application startup

5.5. Order



Class/Enumeration	Description
OrderController	A Spring MVC Controller to handle the cart

5.6. Traceability between Analysis- and Design Model

*Note: The following table shows the Forward- and Backward Traceability from the Analysis Model to the Design Model and vice versa. If an external class is used in the design model, the kind of usage of this external class is defined in the **Usage**-Column, using one of the following options:*

- Inherictance/Interface-Implementation
- Class Attribute
- Method Parameter

Class/Enumeration (Analysis Model)	Class/Enumeration (Design Model)	Usage
BluRay	• catalog.Disc • catalog.DiscType	
Cart	Salespoint.Cart	Method Parameter
CartItem	Salespoint.CartItem (via Salespoint.Cart)	Method Parameter (via Salespoint.Cart)

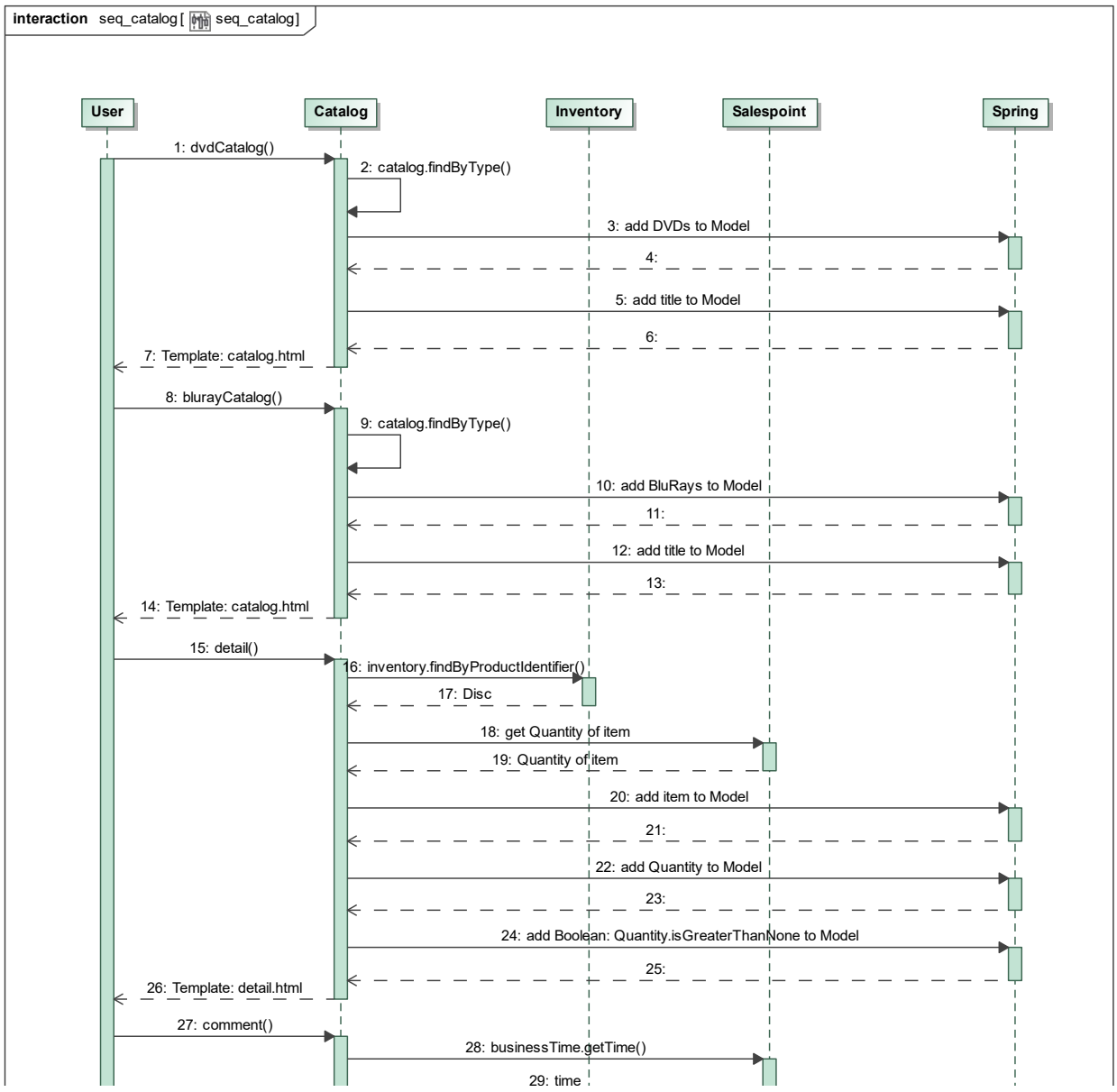
Class/Enumeration (Analysis Model)	Class/Enumeration (Design Model)	Usage
ChargeLine	Salespoint.ChargeLine (via Salespoint.Order)	Method Parameter (via Salespoint.Order)
Comment	catalog.Comment	
Dvd	• catalog.Disc • catalog.DiscType	
Inventory	Salespoint.UniqueInventory	• Class Attribute • Method Parameter
InventoryItem	Salespoint.UniqueInventoryItem	Method Parameter
Order	Salespoint.Order	Method Parameter
OrderLine	Salespoint.Orderline (via Salespoint.Order)	Method Parameter (via Salespoint.Order)
OrderManager	Salespoint.OrderManager<Order>	• Class Attribute • Method Parameter
OrderStatus	Salespoint.OrderStatus	Method Parameter
ROLE/Role	Salespoint.Role	Method Parameter
		• Class

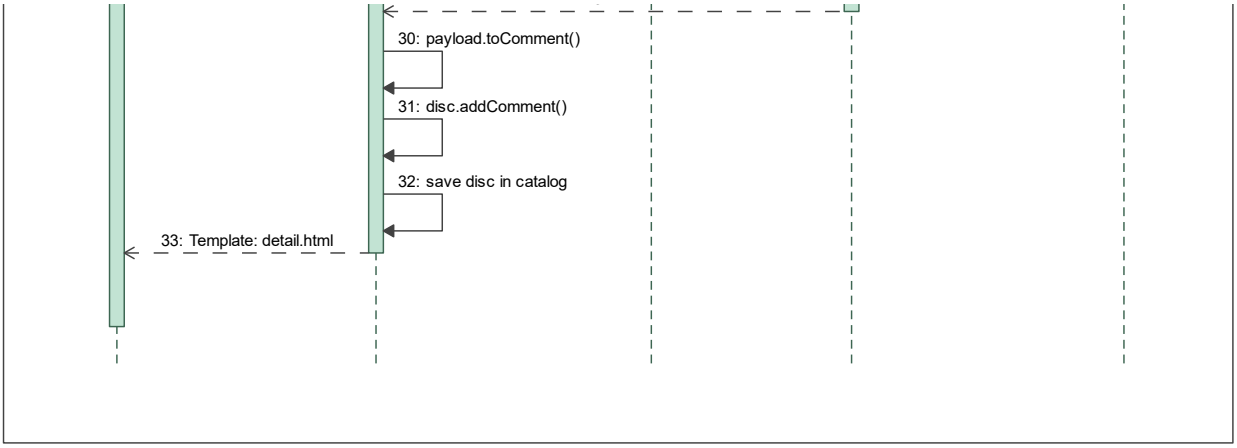
Class/Enumeration (Analysis Model)	Class/Enumeration (Design Model)	Usage
User	• Salespoint.UserAccount • customer.Customer	Attribute • Method Parameter
Videoshop	videoshop.Videoshop	

6. Runtime view

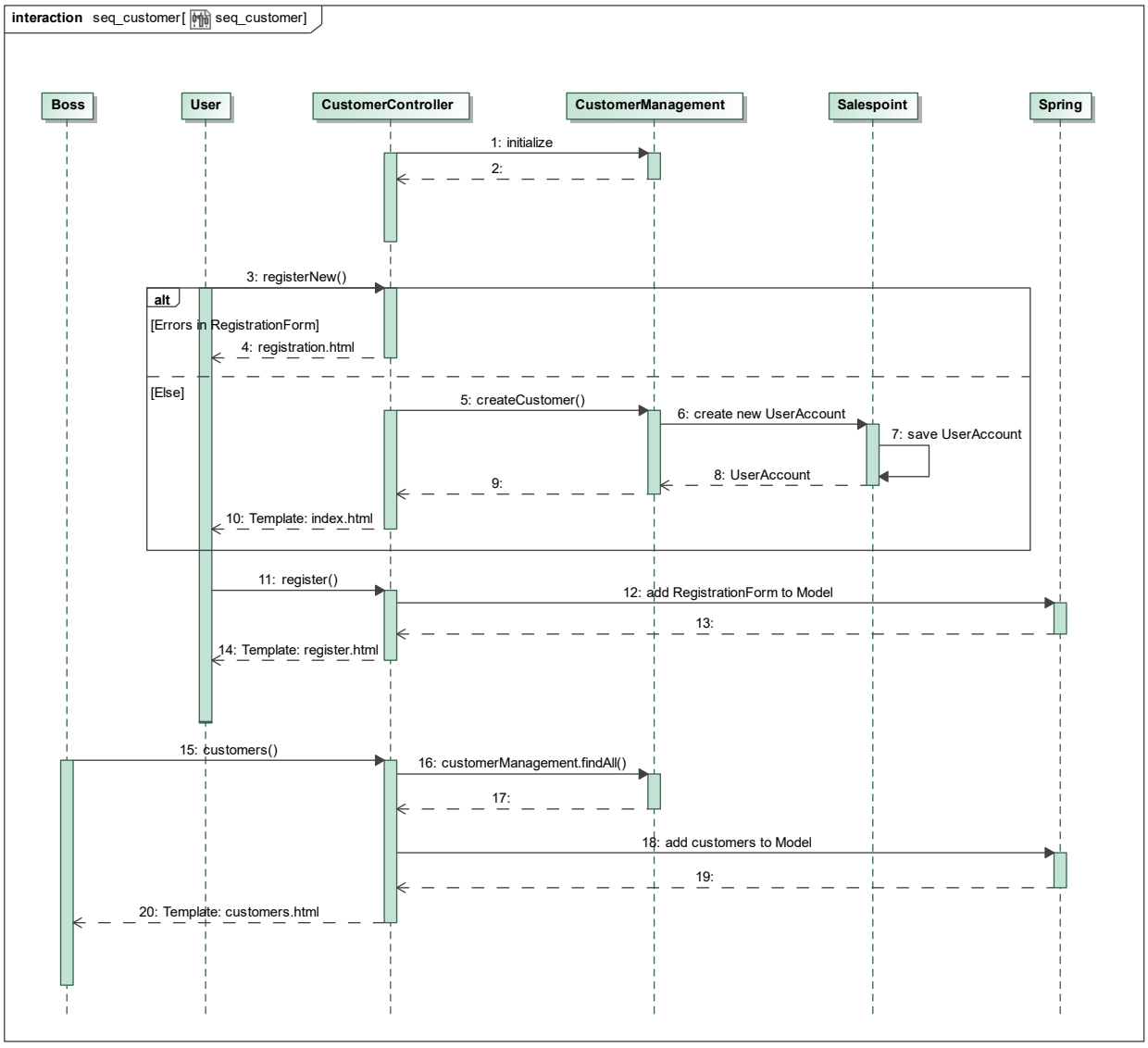
Note: For your developer documentation you only have to create a diagram of one component, which shows the most relevant interactions

6.1. Catalog

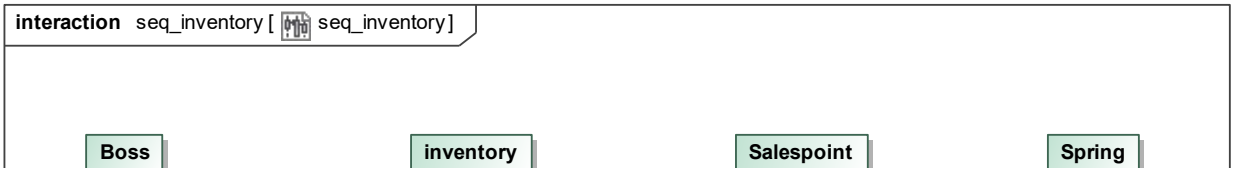


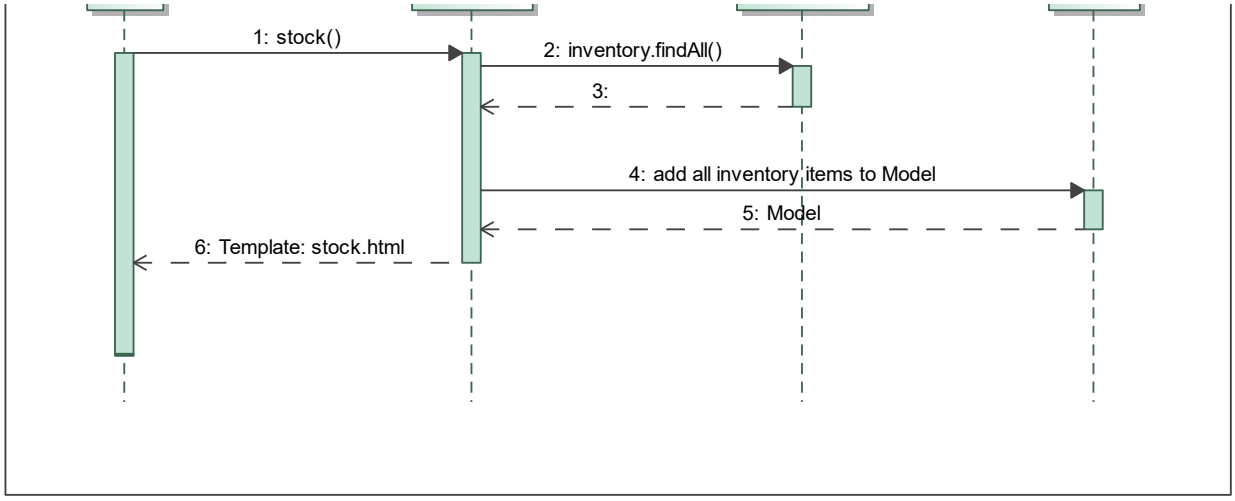


6.2. Customer

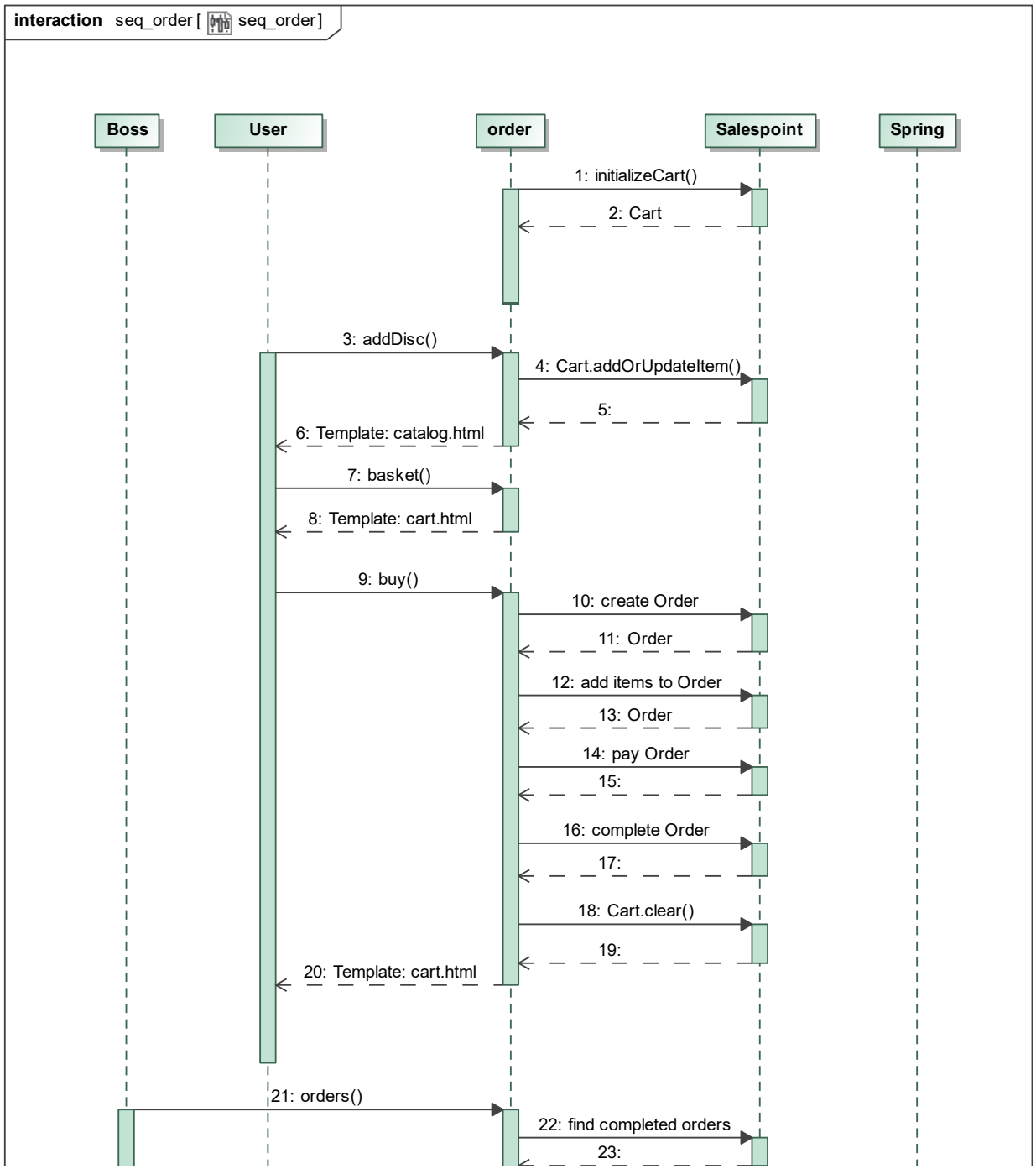


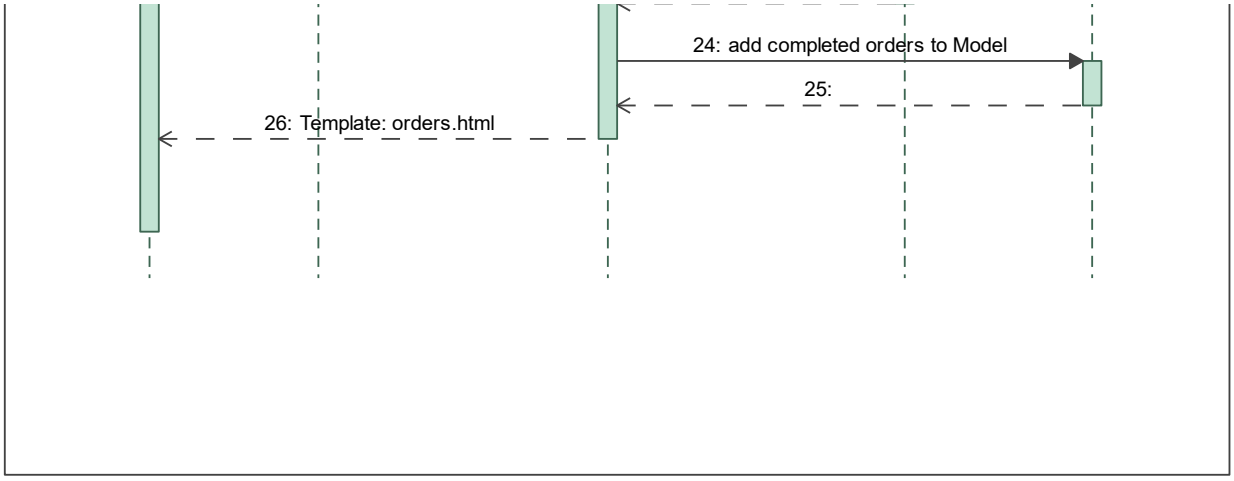
6.3. Inventory




1: stock()
2: inventory.findAll()
3:
4: add all inventory items to Model
5: Model
6: Template: stock.html

6.4. Order


interaction seq_order [seq_order]
Boss
User
order
Salespoint
Spring
1: initializeCart()
2: Cart
3: addDisc()
4: Cart.addOrUpdateItem()
5:
6: Template: catalog.html
7: basket()
8: Template: cart.html
9: buy()
10: create Order
11: Order
12: add items to Order
13: Order
14: pay Order
15:
16: complete Order
17:
18: Cart.clear()
19:
20: Template: cart.html
21: orders()
22: find completed orders
23:



7. Technical debt

7.1. Quality Gates

*Note: In this section, all failed Quality Gates are listed. These ratings go from **A** (best) to **E** (worst). This chapter should only be written at the end of your project.*

Quality Gate	Actual Value	Goal
Reliability	C	A
Coverage	0.0%	50.0%

7.2. Issues

*Note: In this section, all SonarQube issues of the priority **Blocker**, **Critical** and **Major** are listed, as well as common **Minor**-Issues*

Priority	Description	Location	Corresponding Quality Gate
Major	The return value of "orElseGet" must be used	videoshop.InventoryInitializer line 66	Reliability
	Assign this magic number *X*	• 17 appearances within catalog.CatalogInitializer	

Priority	Description	Location	Corresponding Quality Gate
Minor	to a well-named constant, and use the constant instead	• 1 appearance within inventory.InventoryInitializer • 1 appearance within order.OrderController	None
Minor	Lines should not be longer than 120 characters	• 1 appearance within catalog.Disc • 1 appearance within customer.Customer • 1 appearance within customer.RegistrationForm	None