Spring 2015

Software Design Document(SDD)

Version 1.0

CMIS 495

Andrew Bernier, Jason Foley, Brett Fry, John Livingston, Christopher Overby, Ina-Marie Sanabria, Charles Schultz

**University of Maryland University College**

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 2/21/2015 | 1.0 | SDD 1.0 | Group D |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

[1. Introduction 1](#_Toc411224379)

[1.1 Purpose 1](#_Toc411224380)

[1.2 Scope 1](#_Toc411224381)

[1.3 Definitions 1](#_Toc411224382)

[2. References 1](#_Toc411224383)

[3. Decomposition Description 2](#_Toc411224384)

[3.1 Module Decomposition 2](#_Toc411224385)

[3.1.1 User Interface 3](#_Toc411224386)

[3.1.2 Services 4](#_Toc411224387)

[3.2 Data Decomposition 5](#_Toc411224388)

[3.2.1 Customer Entity 5](#_Toc411224389)

[3.2.2 Inventory Entity 6](#_Toc411224390)

[3.2.3 Cart Entity 6](#_Toc411224391)

[3.3.4 Transaction Entity 6](#_Toc411224392)

[3.3.5 Finances Entity 6](#_Toc411224393)

[4. Dependency Description 6](#_Toc411224394)

[4.1 Inter-module Dependencies 6](#_Toc411224395)

[4.2 Data Dependencies 7](#_Toc411224396)

[5. Interface Description 8](#_Toc411224397)

[5.1 User Interface Description 8](#_Toc411224398)

[5.1.1 User Interface Controller 10](#_Toc411224399)

[5.1.2 Cart 11](#_Toc411224400)

[5.1.3 Make Payment 12](#_Toc411224401)

[5.1.4 Admin Page 13](#_Toc411224402)

[5.1.5 Modify products page 14](#_Toc411224403)

[5.2 Service Interface Description 15](#_Toc411224404)

[6. Detailed Design 15](#_Toc411224405)

[6.1 Data Detailed Design 15](#_Toc411224406)

[6.1.1 Customer Entity 15](#_Toc411224407)

[6.1.2 CustomerOrder Entity 16](#_Toc411224408)

[6.1.3 Product Entity 18](#_Toc411224409)

[6.1.4 Category Entity 19](#_Toc411224410)

[6.1.5 OrderedProductPK Entity 19](#_Toc411224411)

[6.1.6 Finances Entity 20](#_Toc411224412)

[6.1.7 Transaction Entity 20](#_Toc411224413)

[6.1.8 OrderedProduct Entity 21](#_Toc411224414)

# 1. Introduction

## 1.1 Purpose

The purpose of this document is to show how the web store application will satisfy the requirements outlined in the Software Requirements Specification (SRS) document and create a detailed blueprint that defines the essential development components for a web store for tracking purchases and payments for an online store/business. This document is specifically used during implementation activity of the system.

## 1.2 Scope

The web store software/application will be named xstore.com or xstore. The xstore is designed to act as an enabler for a small business. In this respect, the scope of the application and its intended functionality should be kept small and specific to the problem set.

## 1.3 Definitions

For the scope of this document, the following definitions and lexicon should be noted.

* *concurrent process:*  process that is always running.
* *design entity:* An element (component) of a design that is structurally and functionally distinct from other elements and that is separately named and referenced
* *domain object:* Something that holds information for a specific function.
* *module:* A well-defined component of a software system or part of a system that provides a set of services to other modules
* *service* – Something that performs business logic.
* *user interface*: The graphical element that lets users interface with the system.
* *workflow*: The series of activities that are necessary to complete a task

# 2. References

References for this document include the following:

* IEEE Std. 1016-1998: IEEE Recommended Practice for Software Design Descriptions

# 3. Decomposition Description

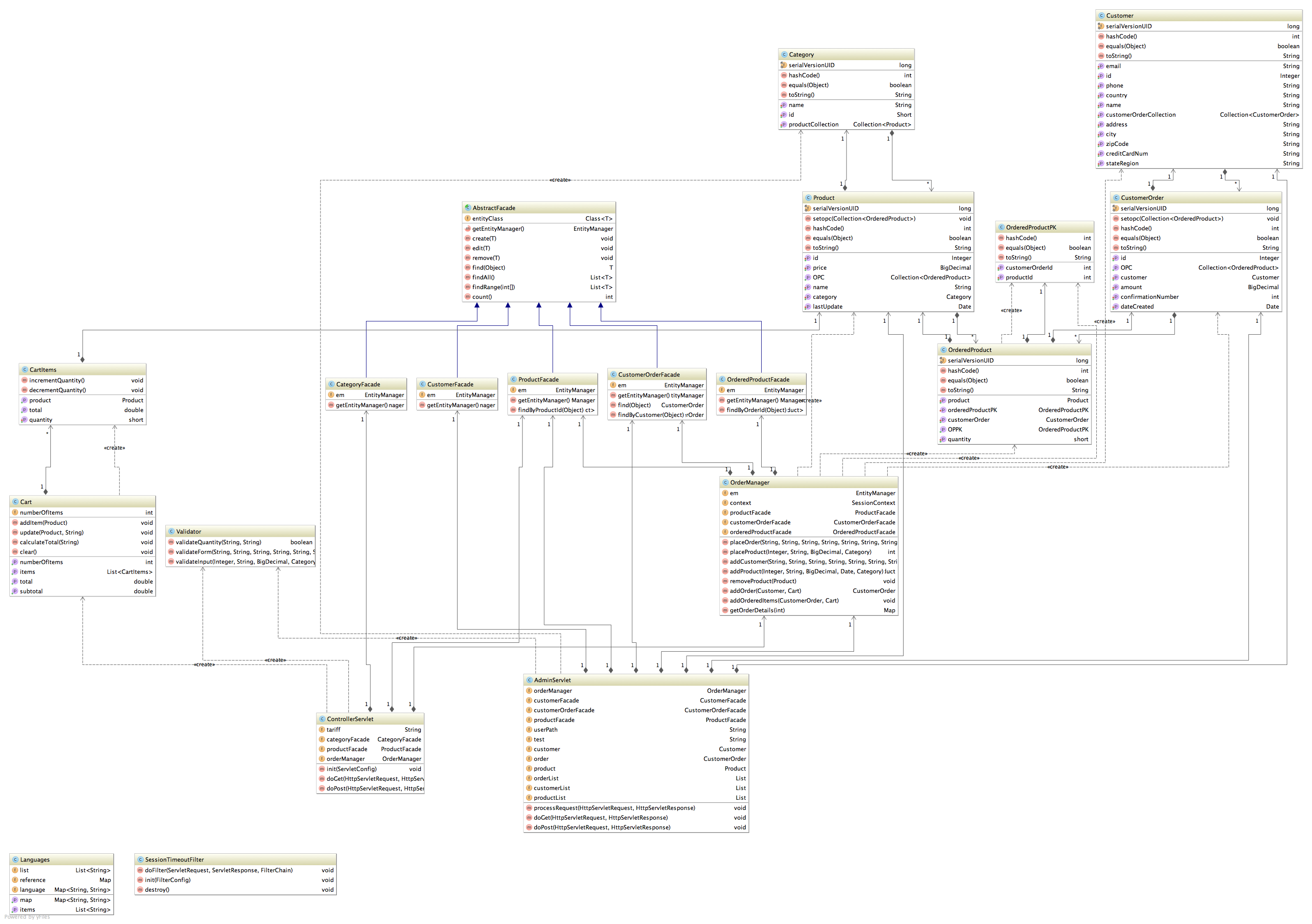


Figure 3.1: Entity Relationship Design Diagram

Figure 3.1 gives the complete entity relationship diagram (ERD) in context for the entire XSTORE.COM. This ERD shows the relationship between the domain objects, their services.

The decomposition description records the division of the software systems into design entities. The following section will describe the way the system has been structured and the purpose and function of each separate entity. The entities and their relationships will also be described in detail. They are divided into modules, concurrent processes, and data.

## 3.1 Module Decomposition

The XSTORE.COM is composed of four different components, which provide specific functionality in the system. These are described at a high level in ***Figure 3.1.1***. The overall design is composed of 4 primary services:

1. User Interface
2. Services
3. Domain Objects
4. Database

In the following representation and design specification, the first three will be covered in depth, as database persistence is covered via each appropriate service. The diagram includes a database reference generically to summarize persistence activities.

The complete architectural context diagram illustrates the human-computer interface for the user of the XSTORE.COM through the user interface, and interaction between domain objects and the user interface via the service layer. This overall design allows for data to be abstracted from the views through functional business logic in the services component, reducing overall complexity.

### 3.1.1 User Interface

This module is for users of the system; since customer, managers, IT workers, and owners of the web store can access the user interface there is a need for multiple user interfaces. Users are able to access the system using this interface exclusively. Access to the user interface does not require any external interface, other than a monitor, keyboard, mouse, printer, and is a single-user set of views.



Figure 3.1.1.1: Architectural Context Diagram – User Interface

#### 3.1.1.1 Cart Items

Displays information inside the cart.

#### 3.1.1.2 Inventory Management

Displays information about specific products, and data entry elements to manage items/products.

#### 3.1.1.3 Make Payment

Displays information about payments specific to a customer.

#### 3.1.1.4 Transaction/Order Management

Displays information about customer transactions in relation to a specific date or order.

#### 3.1.1.5 Customer Information

Displays information about a specific customer including, contact information, payment information, etc.

### 3.1.2 Services

This module is used to house the business logic for processes that the XSTORE.COM needs to conduct where domain objects are involved, such as looking up a previous order or saving a payment. The service module provides output to the user interface, and input from the user interface is used to manage domain objects where appropriate. Persistence of the domain objects to the database is also managed by the service component.



Figure 3.1.2.1: Architectural Context Diagram – Services

#### 3.1.2.1OrderedProducts Entity

This service entity component handles the business logic for checking out and modifying the quantity of items in stock and store users purchase.

#### 3.1.2.3 MakePayment Entity

This service entity component handles the business logic for receiving a payment, adjusting a Finances balance, as well as association to a transaction and customer.

#### 3.1.2.4 Transaction Entity

This service entity assigns customer information to a transaction that has been determined and holds other metadata, such purchased items, dates and associated payments.

#### 3.1.2.5 CustomerInformation Entity

This service entity assigns Customer information directly, such as phone numbers, name, and address.

## 3.2 Data Decomposition

This module covers all entities that hold data about a particular function in the XSTORE.COM system. These objects are referred to as domain objects.



Figure 3.2.1: Architectural Context Diagram – Domain Objects

### 3.2.1 Customer Entity

This entity holds all information related to a Customer that has used the site and or made a purchase. This information includes metadata such as first and last name, address, credit card number, and so on.

### 3.2.2 Inventory Entity

This entity holds all information about products sold by the establishment. This information is used to manage items in/out of stock

### 3.2.3 Cart Entity

This entity holds all information about an active purchase, include metadata such as product, number of items, cost, and so forth. This entity is the general correlation point for other domain objects within the XSTORE.COM.

### 3.3.4 Transaction Entity

This entity holds information about a payment that correlates directly from a customer to a specific transaction.

### 3.3.5 Finances Entity

This entity holds information about all payments made to the establishment through the XSTORE.COM.

# 4. Dependency Description

Service modules save, delete and update the domain object directly via the database module entries. The user interface module interacts with the service module. The service module controls and manages the domain model objects. This list of dependencies can be seen in a level 1 DFD in Figure 4.1. Additionally, the list of dependencies for modules, services, and data are explained in detail below.

### 4.1 Inter-module Dependencies

For the XSTORE.COM software, there are 4 core modules. These include the user interface, services, domain objects, and database. For the scope of this document, the database module is called directly by the service modules as appropriate, and will not be extracted as a core module for description. In ***Figure 4.1.1***, these are highlighted as dependencies, with the added interaction of the user to the user interface.



Figure 4.1.1: Inter-module Dependency Diagram

1. The User Interface module has a dependency on the service module. Since the user interface is tasked with displaying information from the XSTORE.COM, this dependency allows the business logic from the services module to be represented in the views directly at the user interface. In addition, the user has a dependency on the user interface to represent and interact with those views.
2. The Services module has dependencies on both the user interface module to represent a view, as well as the domain objects module to store information. Since business logic is represented at the service module, but information is stored within the domain objects, this dependency is critical to represent the state of actions and information within the system.
3. The Domain Objects module has a dependency on the services module to process information with business logic, as explained in dependency #2.

## 4.2 Data Dependencies

Data dependencies in the XSTORE.COM are loosely linked through process dependencies to services. This is done to abstract business logic away from both the user interface module, as well as the domain object model. Due to this fact, there is no direct correlation between any of the domain objects. Instead, the processes/services share the dependencies to assist in business logic needs and abstraction between modules.

# 5. Interface Description

## 5.1 User Interface Description

The user interface provides the user with the ability to interact with the XSTORE.COM system, and as a result create records, make purchases, manage inventory, and associations that culminate in an enjoyable experience for the user of the web store. The figure below highlights the overall workflow from a human-computer interaction perspective, and how a daily user of the XSTORE.COM would interact with the system to make a purchase.



Figure 5.1.1: User Interface Workflow

### 5.1.1 User Interface Controller

The Interface Controller is the primary interface for handling calls for the xstore. This screen provides four capabilities:

1. Home Page – This is the launching point for the application and allows the user to browse categories
2. Modify/Update cart items – Shows items added to the cart
3. Checkout – Allows user to make purchase
4. Access to Admin controller and page

The home page user interface element provides the user with the ability to start the workflow when wishing to make a purchase. By finding a product category that is of interest to the customer, allowing the user to easily navigate through the site, and providing language support the user of the XSTORE.COM can then start the process of making purchases. Additionally the products page allows for the user to add items to their shopping cart.

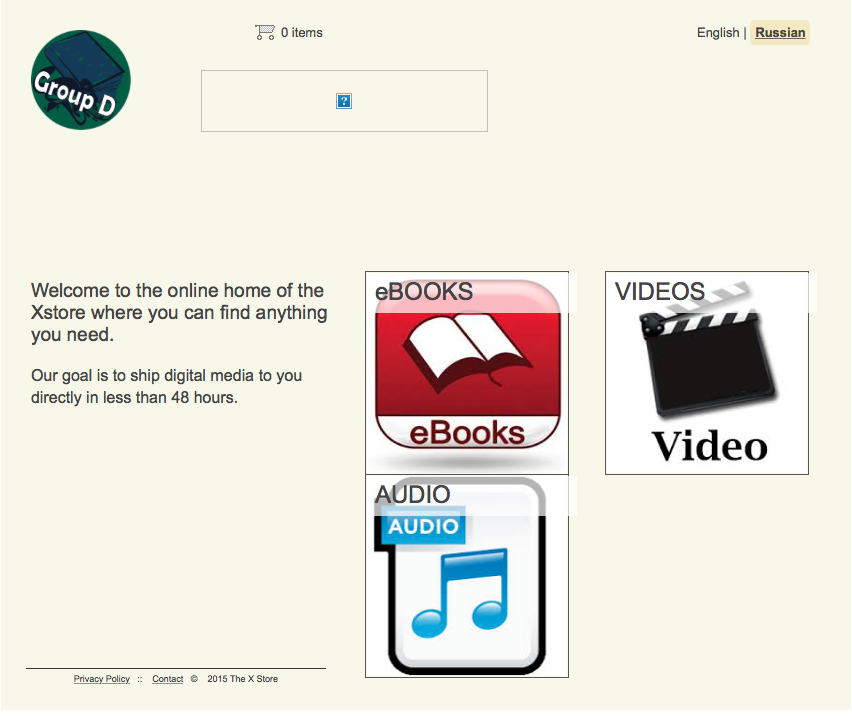


Figure 5.1.1.1: Home Page User Interface

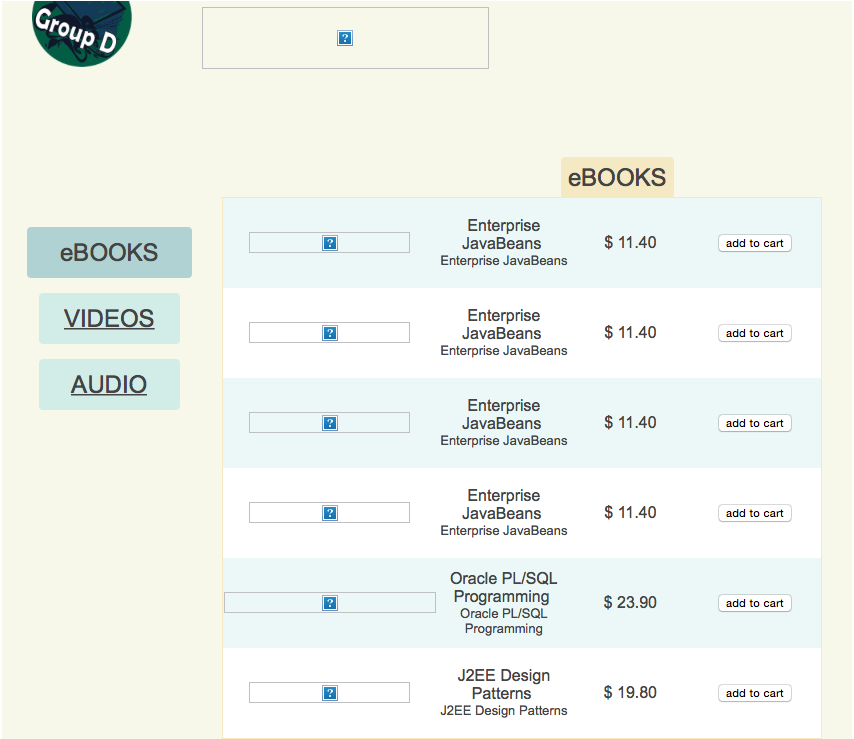


Figure 5.1.1.2: Products (eBooks) User Interface

### 5.1.2 Cart

The cart page is designed to add, remove, and update items the customer wishes to purchase. This portion of the user interface is very simple, and only gives information specific to what is in the cart. One buttons and one link are provided to add, remove, and update the cart based on the choices made by the user.

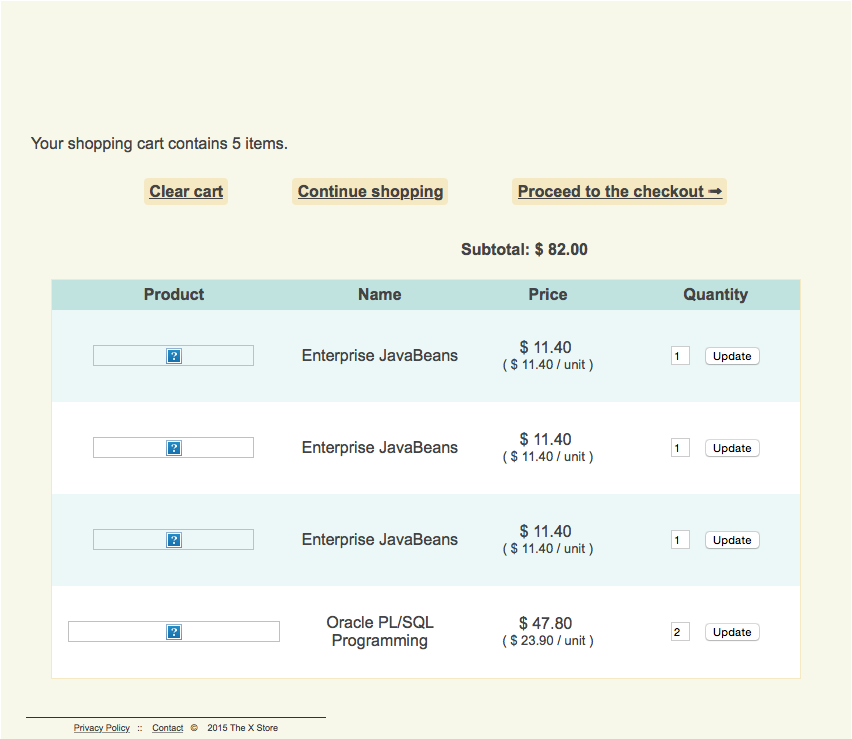


Figure 5.1.2.1: Cart User Interface

### 5.1.3 Make Payment

The payment element of the user interface is designed to encapsulate all elements of a customer’s payment towards their purchase. To accomplish this, the total amount of the items in the cart is provided, along with the delivery fees and taxes. The customer list is provided to associate the payment correctly with the credit card provided by the customer when their information has been taken. Finally, the buttons allow for the user to either process the payment, or cancel the transaction completely.

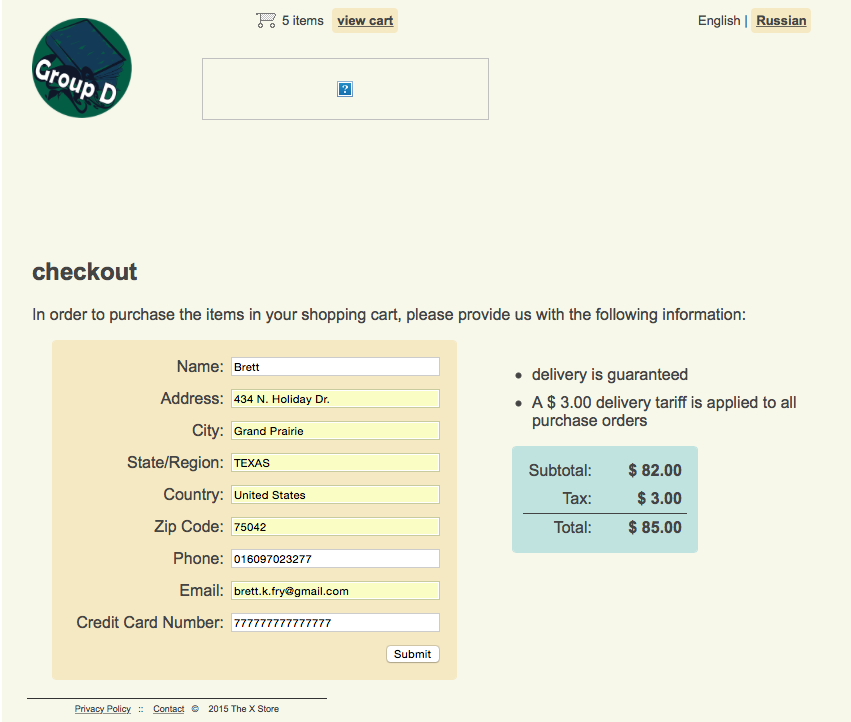


Figure 5.1.3.1: Make Payment User Interface

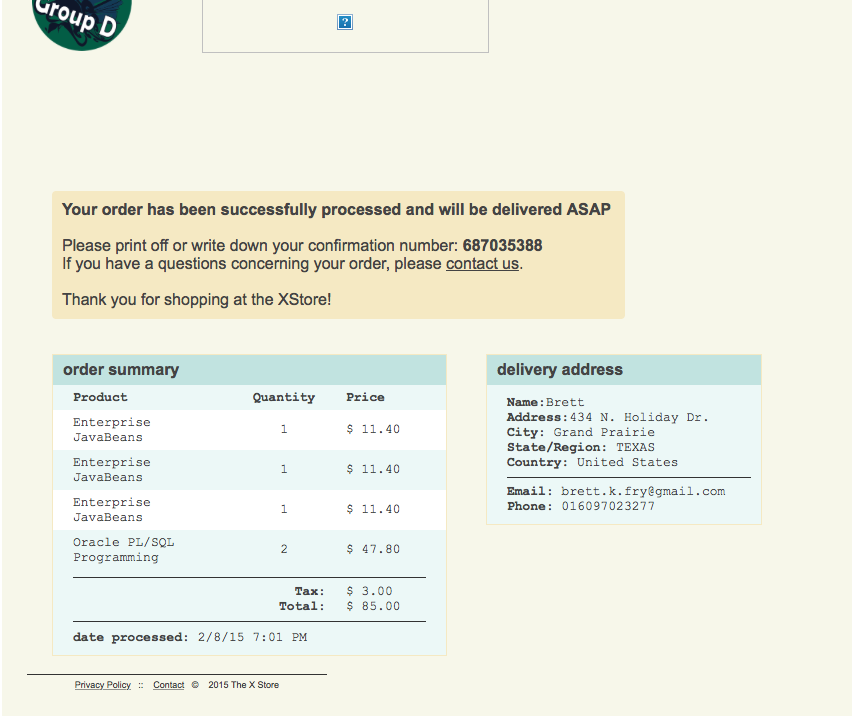


Figure 5.1.3.2: Customer Confirmation User Interface

### 5.1.4 Admin Page

In order to correctly manage the site the administrator need access to update, modify, or delete content rapidly. This can be done using the Admin Page.

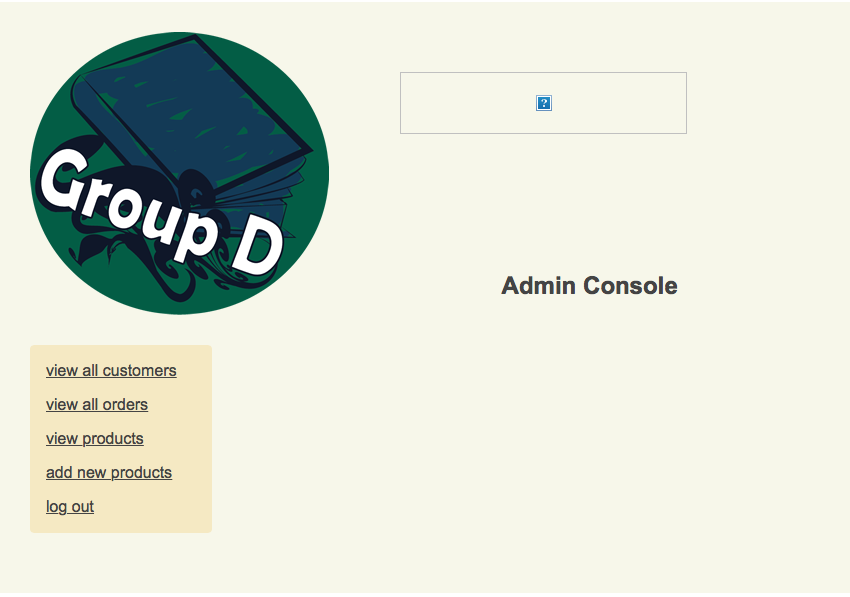


Figure 5.1.4.1: Admin User Interface

### 5.1.5 Modify products page

In order to correctly manage the site the administrator need access to update, modify, or delete items or products sold on the site.

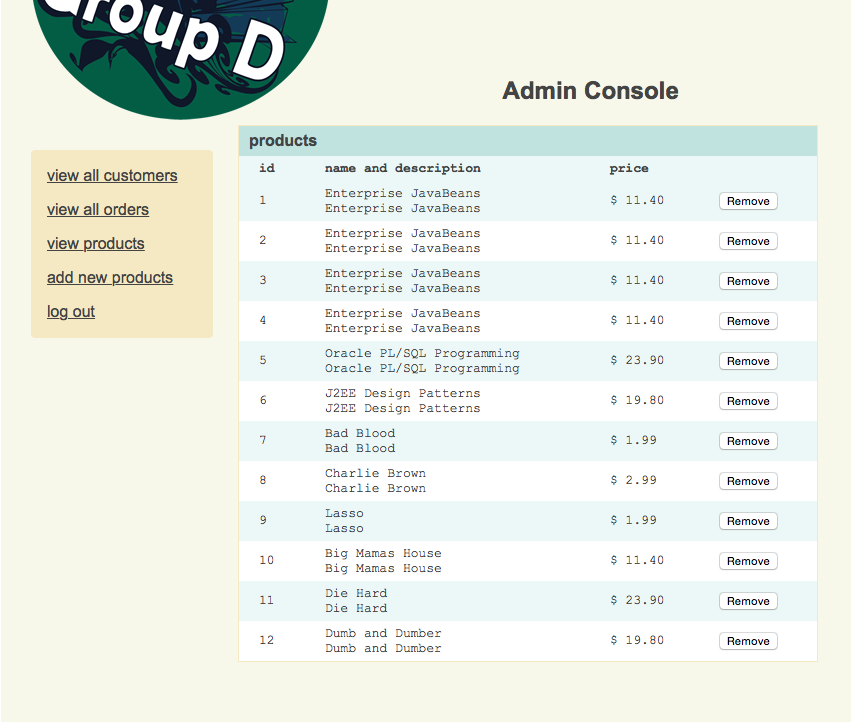


Figure 5.1.5.1: Remove product User Interface

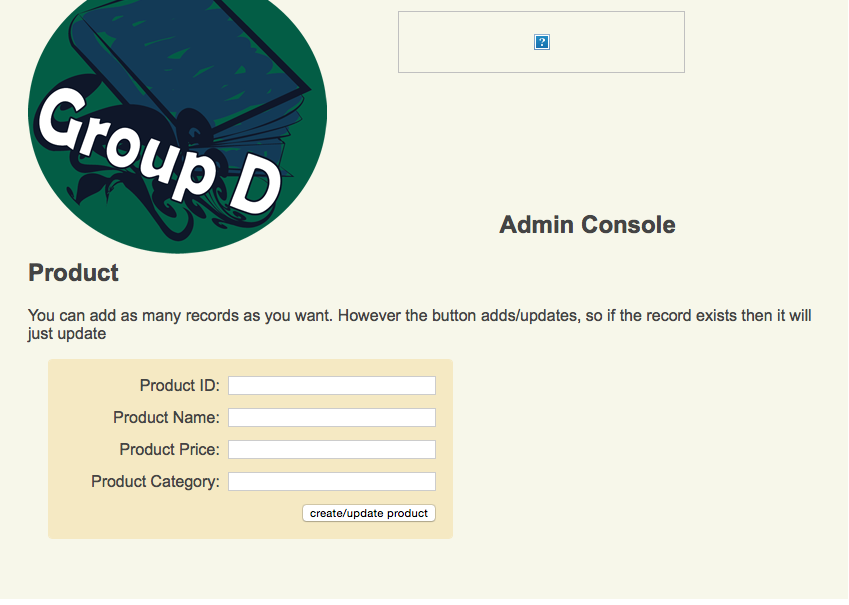


Figure 5.1.5.2: Add/Update User Interface

### 5.2 Service Interface Description

Service interfaces (domain object, service, and concurrent service) relationships can be found in the Entity Relationship Diagram in section 3 (Decomposition Description). Interface descriptions can be found with detailed descriptions and class models in section 6.1 (Data Detailed Design)

# 6. Detailed Design

## 6.1 Data Detailed Design

### 6.1.1 Customer Entity

The Customer entity is a domain object that is used to represent information about a customer. This domain object stores information about a customer, and is primarily composed of attributes and accessor methods.

Twenty-two methods are present in the customer entity:

1. int getId(void) – Returns the unique id for a Customer object.
2. void setId(int id) – Allows manual setting or changing of a customer id.
3. String getFirstName() – Returns the first name of a Customer Object.
4. void setFirstName(string firstName) – Sets the customer’s first name based on the user input.
5. String getMiddleName() – Returns the middle name of a Customer Object.
6. void setMiddleName(string middleName) – Sets the customer’s middle name based on user input.
7. String getLastName() – Returns the last name of a Customer Object.
8. void setLastName(String lastName) – Sets the customer’s last name based on user input.
9. String getStreet() – Returns the street portion of the address of a Customer Object.
10. void setStreet(String street) – Sets the customer’s street address based on user input.
11. String getCity() – Returns the city portion of the address of a Customer Object.
12. void setCity (String city) – Sets the customer’s city based on user input.
13. String getState() – Returns the State portion of the address of a Customer Object.
14. void setState(String state) – Sets the customer’s State based on user input.
15. String getCountry() – Returns the Country portion of the address of a Customer Object.
16. void setCountry(String country) – Sets the customer’s Country based on user input.
17. int getZip() – Returns the zip code portion of the address of a Customer Object.
18. void setZip(int zipCode) – Sets the customer’s zip code based on user input.
19. int getPhoneNumber() – Returns a representation of a customer’s phone number based on country.
20. void setPhoneNumber(int phoneNumber) – Sets the customer’s phone number based on user input.
21. int getCCNumber() – Retrieves the customer’s credit card number.
22. void setCCNumber(int ccNumber) Sets the customer’s credit card number based on user input.

Eleven attributes are present in this entity:

1. int id – Unique id for a customer.
2. String firstName – First name of a customer.
3. String middleName – Middle name of a customer
4. String lastName – Last name of a customer.
5. String street – street address of a customer
6. String city – city of a customer
7. String state - state of a customer
8. String country – country of a customer
9. int zipCode – zipCode of a customer
10. int phoneNumber – Phone number of a customer.
11. int ccNumber – Credit card number associated with a customer.

### 6.1.2 CustomerOrder Entity

The CustomerOrder entity is a domain object that is used to represent information about a customer's orders. This domain object stores information about an order, and is primarily composed of attributes, mutator, and accessor methods.

Twelve methods are present in the customer entity:

1. Integer getId()
2. void setId(Integer id) – is a void method (mutator method) that allows for the driver to change the quantity
3. BigDecimal getAmount()–method (accessor method) that s dollar amount
4. void setAmount(BigDecimal amount) – is a void method (mutator method) that allows for the driver to change the dollar amount
5. Date getDateCreated()– method (accessor method) that s dateCreated
6. void setDateCreated(Date dateCreated) – is a void method (mutator method) that allows for the driver to change the dateCreated
7. int getConfirmationNumber()–method (accessor method) that s confirmation number
8. void setConfirmationNumber(int confirmationNumber) –is a void method (mutator method) that allows for the driver to change the confirmation number
9. Collection<OrderedProduct> getOPC()– method (accessor method) that s ordered Product Collection
10. void setopc(Collection<OrderedProduct> opc) – is a void method (mutator method) that allows for the driver to change the ordered Product Collection
11. Customer getCustomer()– method (accessor method) that s Customer object
12. void setCustomer(Customer customer) – is a void method (mutator method) that allows for the driver to change the Customer object

Six attributes are present in this entity:

1. int id – unique customer id
2. BigDecimal amount –dollar amount of order
3. Date dateCreated –date that order was created
4. int confirmationNumber –unique confirmation number
5. Customer customer – customer related to purchase
6. Collection<OrderedProduct> opc – collection of prices for objects in order

### 6.1.3 Product Entity

The Product entity is a domain object that is used to represent information about a customer's orders. This domain object stores information about an order, and is primarily composed of attributes, mutator, and accessor methods.

Twelve methods are present in the customer entity:

1. Integer getId(void) – returns unique id for Product object
2. String getName(void) – returns unique name for Product object
3. void setName(String name) – sets Product name
4. BigDecimal getPrice(void) – returns price for Product object
5. void setPrice(BigDecimal price) – sets price for Product object
6. Date getLastUpdate(void) – returns date of last update to the Product object
7. void setLastUpdate(Date lastUpdate) – sets date of last update for Product object
8. Category getCategory(void) – returns Category object for Product object
9. void setCategory(Category category) – sets Category object for Product object
10. Collection<OrderedProduct> getOPC()– returns Collection of OrderedProduct objects
11. void setopc(Collection<OrderedProduct> opc) – sets OrderedProduct Collection
12. String toString(void) – returns formatted information for Product Object

Six attributes are present in this entity:

1. Integer id – unique id for Product object
2. String name – name for Product object
3. BigDecimal price – price for Product object
4. Date lastUpdate – date of last alteration to Product object
5. Category category – Category object assigned to Product object
6. Collection<OrderedProduct> opc – Collection object assigned to Product

### 6.1.4 Category Entity

The Category entity is a domain object used to represent information about a distinct category. This entity mostly consists of attributes and accessor methods.

Nine methods are present in the category entity:

1. Short getId() – this method returns the id of a particular category
2. void setId(short id) – this method allows the driver to change the id of a particular category
3. String getName() – this method returns the name of a given category
4. void setName(String name) – this method allows the driver to change the name of a category
5. Collection<Product> getProductCollecion() – this method returns a product collection
6. void setProductCollecion(Collection<Product> productCollection) – this method allows the driver to change the product collection
7. int hashCode() – this method returns the hash code of a categories id.
8. Boolean equals(Object object) – this method returns true or false depending if the parameter equals the category
9. String toString() – this method returns the string value of category

Three attributes are present in this entity:

1. short id – Unique id for a category
2. String name – Unique name for a category
3. Collection<Product> productCollection – Collection of products

### 6.1.5 OrderedProductPK Entity

The OrderedProductPK entity is a domain object used to represent information about a distinct ordered product and its primary key. This entity mostly consists of attributes and accessor methods.

Nine methods are present in the category entity:

int getCustomerOrderId() – this method returns a customer’s order id

void setCustomerOrderId(int customerOrderId) – this method allows the driver to change the order id

int getProductId() – this method returns a project id

void setProductId(int productId) – this method allows a driver to change a product id

int hashCode() – this method returns a customer order id hash code

Boolean equals(Object object) – this method returns true or false depending on whether the object is equal to order id

String toString() – this method returns the string format of a ordered product

Two attributes are present in this entity:

int customerId – Unique id for a customer

int productId – Unique id for a product

### 6.1.6 Finances Entity

The Finances entity is a domain object that is used to represent financial information for the totality of the establishment. This domain object stores information about a total finances, and is primarily composed of attributes and accessor methods.

Two methods are present in this entity:

1. void expense(Payment) – Calculates the expenditure of funds based on a Payment (negative).
2. void profit(Payment) – Calculates the addition of finds based on a Payment (positive).

Two attributes are present in this entity:

1. double money – Total money available to the establishment.
2. Payment[] payments – Managed collection of Payment domain objects.

### 6.1.7 Transaction Entity

The Transaction entity is a domain object that is used to represent payment information for a purchase. This domain object stores information about a payment, and is primarily composed of attributes and accessor methods. represents the class diagram for Payment.

Six methods are present in this entity:

1. int getId() – Returns the unique id for a payment.
2. double getTotal() – Returns the total amount for the payment that is to be paid in full for a purchase
3. void setTotal(double) – Sets the total amount that the payment requires to be paid in full for a purchase.
4. void makePayment(double) – The amount being paid toward/applied to the purchase cost.
5. int getCustomerId() – Returns the unique id for the correlated customer.
6. void setCustomerId(int customerId) – Sets the unique id for the specified customer.

Four attributes are present in this entity:

1. int id – Unique id for a payment.
2. double paid – Amount paid towards the total.
3. double total – Total amount required to be paid in full for a purchase
4. int customerId – Unique id for the correlated customer.

### 6.1.8 OrderedProduct Entity

The OrderedProduct entity is a domain object used to represent information about a distinct ordered product. This entity mostly consists of attributes and accessor methods.

Ten methods are present in the category entity:

1. void setCustomerOrderId(int customerOrderId) – this method allows the driver to change the order id
2. short getQuantity() – this method returns the quantity
3. void setQuantity(short quantity) – this method allows a driver to change the quantity amount
4. Product getProduct() – this method returns a product
5. void setProduct(Product product) – this method allows a driver to set a product
6. CustomerOrder getCustomerOrder() – this method returns a customer’s order
7. Void setCustomerOrder(CustomerOrder customerOrder) – this method allows a driver to set a customer’s order
8. Int hashCode() – this method returns the hash value of a ordered product
9. Boolean equals(Object object) – this method returns true or false depending on whether the object matches the ordered product
10. String toString() – this method returns the string representation of a ordered product

Four attributes are present in this entity:

1. OrderedProductPK oProductPK – Unique ordered product
2. short quantity – Unique product quantity
3. Product product – Unique product type
4. CustomerOrder customerOrder – Unique customer order