GUIDED TECHNOLOGY PROJECT

OLINE BOOK STORE

Lucas de Castro - 2021155
CCT College

ABSTRACT

The advantages of saving time, research tool, better selection of products and better information on prices and lower prices are the reasons for creating the Online Book Store which is a web application for selling books online. Through this online platform, the administrator can register new books, create promotions and view their registered books. While customers can search for titles, buy these books and pay with any bank card.

Table of Contents

CHAPTER I: INTRODUCTION

Project Context

According to research conducted by Virgin Media, Irish consumers' online shopping habits have changed significantly in recent years, and this behavior will continue, according to research commissioned by Virgin Media Ireland and independently guided by Amárach Research.

With the current pandemic situation, half of all respondents said they now more often buy products/services online. The survey interviewed more than 1,000 people aged 18 and over in all regions of the country (Amárach Research, 2020). In this scenario, an online store becomes a very viable commercial option.

To take advantage of this scenario, the Online Bookstore will contain some important features, listed below.

The customer will be able to:

- Each customer will be able to add or remove books from the shopping cart.
- The shopping cart will be updated for each interaction with it.
- After paying the order using any credit card, an email will be sent to the customer containing the purchase information
- While viewing the online store, the customer may receive a notification of a book promotion.
- Search the title of any book.

The Admin will be able to:

- The admin will be able to log in in the system.
- Register new books and their authors and book cover image.
- Create promotion and let the customers know it by a Pop up in their screens.
- View the list of all registered books.

System features

- Use of Cache in the Data Base for better performance when loading the store's homepage.
- Login built using JAAS (Java Authentication and Authorization Service).
- Web service providing all registered books and in JSON and XML.
- Responsive Design.

Why is it a good project?

Due to the high growth of online shopping, the virtual book store will be able to take advantage of the current situation. Technologies that will be used to develop the project are JSF (JavaServer Faces), JPA (Java Persistence API), REST and XML.

Main Goal

Develop a web application that will work as an online store for selling books.

Objectives

- Research on technologies
- Define design and diagrams of the project
- Database settings and connections using Postgres.
- Web application development using JSF.
- Complete final report and presentation.

Areas to Cover

This project will have a monolithic structure using the following technologies:

- JSF (JavaServer Faces)
- JPA (Java Persistence API)
- CDI (Context Dependencies Injection)
- HTML (HyperText Markup Language)
- Bootstrap
- JavaScript
- Web service with XML and JSON
- JMS (Java Message Service)
- JAAS (Java Authentication and Authorization Service)
- WebSockets
- Wildfly as local web server for testing and deployment
- PostgreSQL Database

Methodology

The Scrum methodology was adopted for the development of this project. As it is a project carried out by only one developer, the Daily Meeting, Spring Review and Sprint Retrospective ceremonies were adapted for just one person. Basically, these three ceremonies mentioned were joined for only one person.

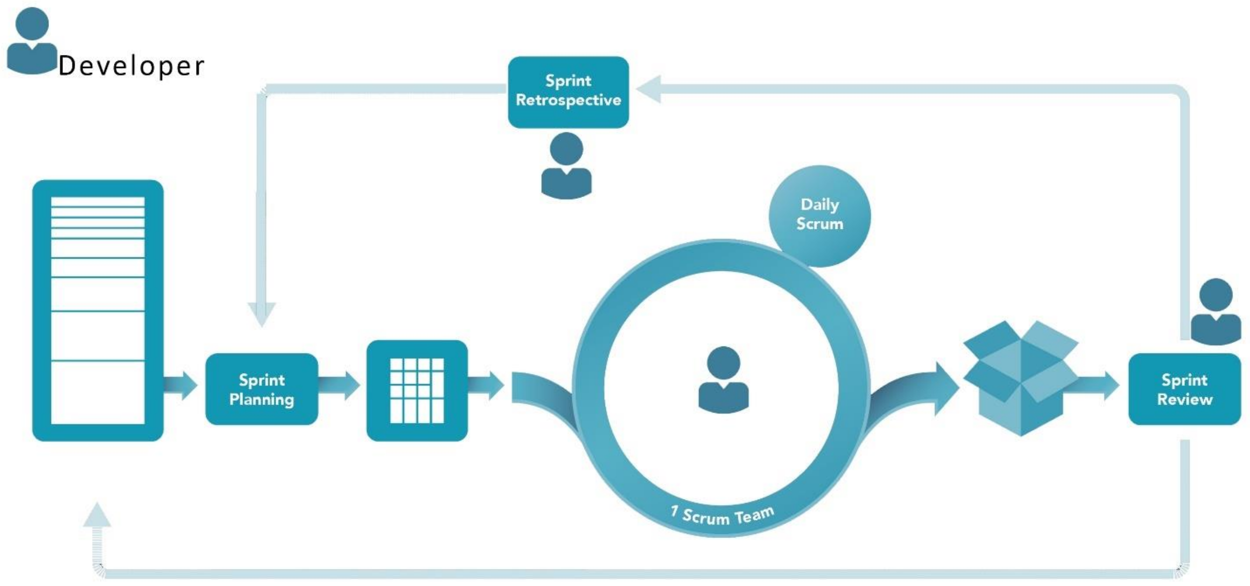
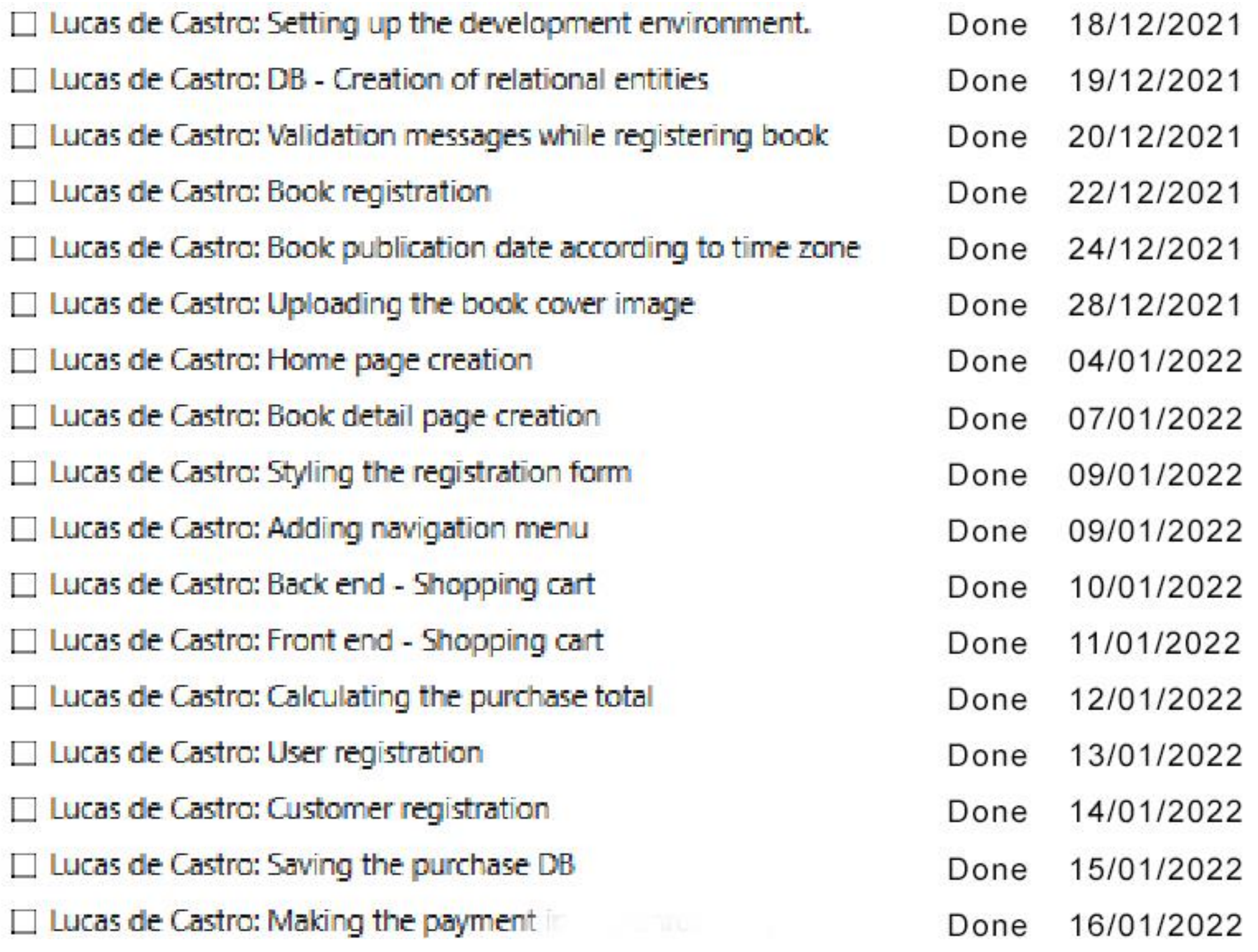


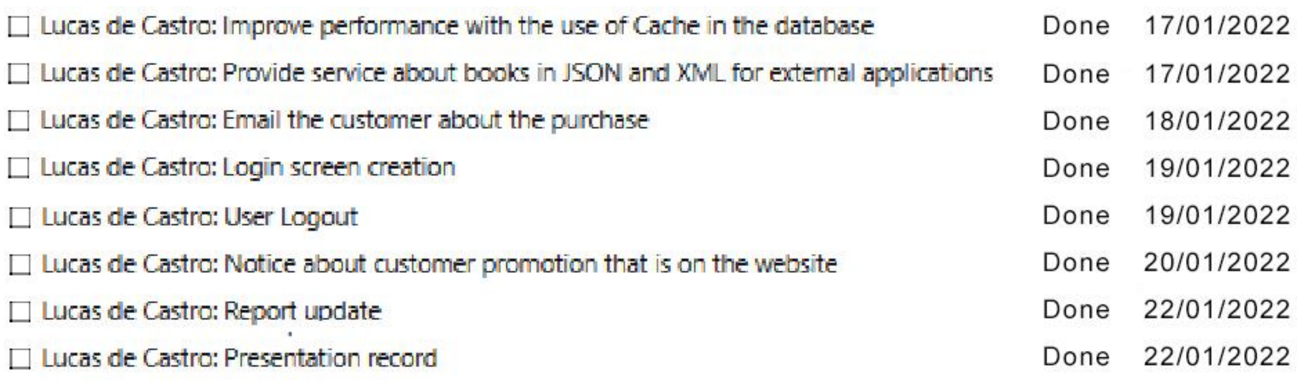
Figure 1 - Scrum Methodology adapted for one person

All the tasks are defined already as the Gantt Chart Task List below:



☐ Lucas de Castro: Setting up the development environment.	Done	18/12/2021
☐ Lucas de Castro: DB - Creation of relational entities	Done	19/12/2021
☐ Lucas de Castro: Validation messages while registering book	Done	20/12/2021
☐ Lucas de Castro: Book registration	Done	22/12/2021
☐ Lucas de Castro: Book publication date according to time zone	Done	24/12/2021
☐ Lucas de Castro: Uploading the book cover image	Done	28/12/2021
☐ Lucas de Castro: Home page creation	Done	04/01/2022
☐ Lucas de Castro: Book detail page creation	Done	07/01/2022
☐ Lucas de Castro: Styling the registration form	Done	09/01/2022
☐ Lucas de Castro: Adding navigation menu	Done	09/01/2022
☐ Lucas de Castro: Back end - Shopping cart	Done	10/01/2022
☐ Lucas de Castro: Front end - Shopping cart	Done	11/01/2022
☐ Lucas de Castro: Calculating the purchase total	Done	12/01/2022
☐ Lucas de Castro: User registration	Done	13/01/2022
☐ Lucas de Castro: Customer registration	Done	14/01/2022
☐ Lucas de Castro: Saving the purchase DB	Done	15/01/2022
☐ Lucas de Castro: Making the payment i	Done	16/01/2022

Figure 2 - Task List of the Project



☐ Lucas de Castro: Improve performance with the use of Cache in the database	Done	17/01/2022
☐ Lucas de Castro: Provide service about books in JSON and XML for external applications	Done	17/01/2022
☐ Lucas de Castro: Email the customer about the purchase	Done	18/01/2022
☐ Lucas de Castro: Login screen creation	Done	19/01/2022
☐ Lucas de Castro: User Logout	Done	19/01/2022
☐ Lucas de Castro: Notice about customer promotion that is on the website	Done	20/01/2022
☐ Lucas de Castro: Report update	Done	22/01/2022
☐ Lucas de Castro: Presentation record	Done	22/01/2022

Figure 3 - Project Task List Continuation

CHAPTER II: LITERATURE REVIEW

The objective of this chapter is to show the academic research carried out that corroborates the decisions chosen throughout the project. The main concepts and patterns of the project proposal are:

- Java EE
- JSF (JavaServer Faces)
- CDI (Contexts and Dependency Injection)
- EJB (Enterprise JavaBeans)
- JAX-RS and JAX-WS
- WebSocket
- JTA (Java Transactions API)
- JPA (Java Persistence API)
- MVC Pattern – DAO
- Database
- JMS (Java Message Service)

Java EE

The decision to use Java EE platform components was made after evaluating the learning curve involving newer technologies such as React with Spring Boot in a microservices architecture. There would not be enough time to develop the project, so the Java EE platform was chosen to run the entire project (Front-End and Back-End) on a monolithic software architecture.

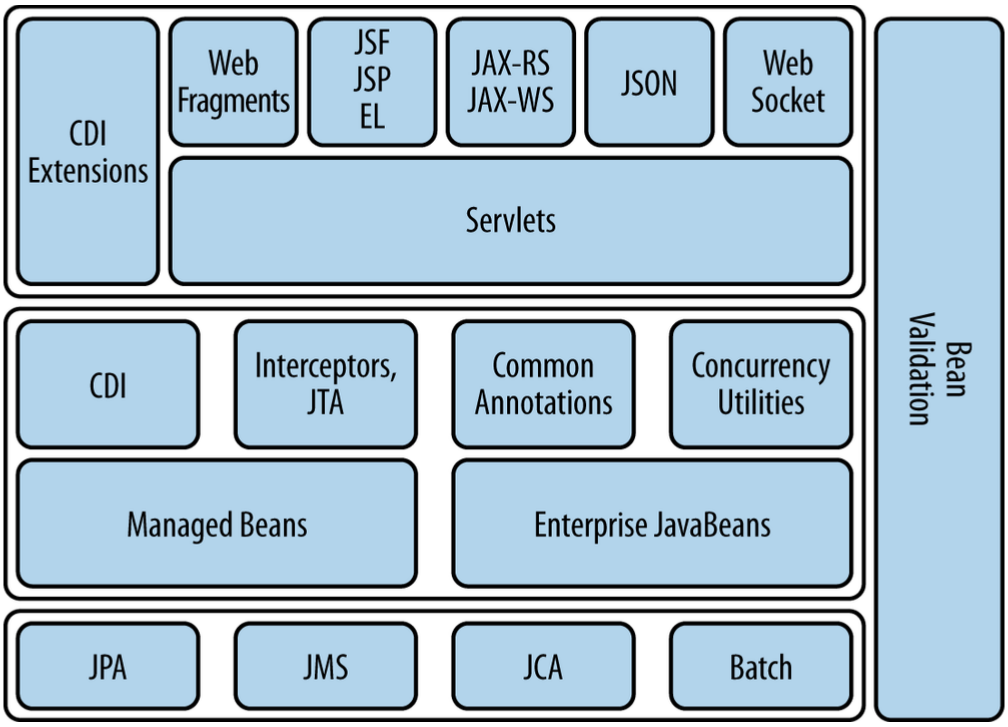


Figure 4 - A summary of all Java EE components

JSF (JavaServer Faces)

JSF brought a user interface-oriented approach, providing the programmer with a rich collection of components, as well as a better separation between application layers. With JSF, the programmer can write views in XHTML and through Data Binding connect them to Java Bean classes. However, JSF is not just about building the page to be shown to the user. It brings with it the MVC pattern to separate the responsibilities of each element to be developed for the web system. Below we see the life cycle of a request using JSF (Figure 5).

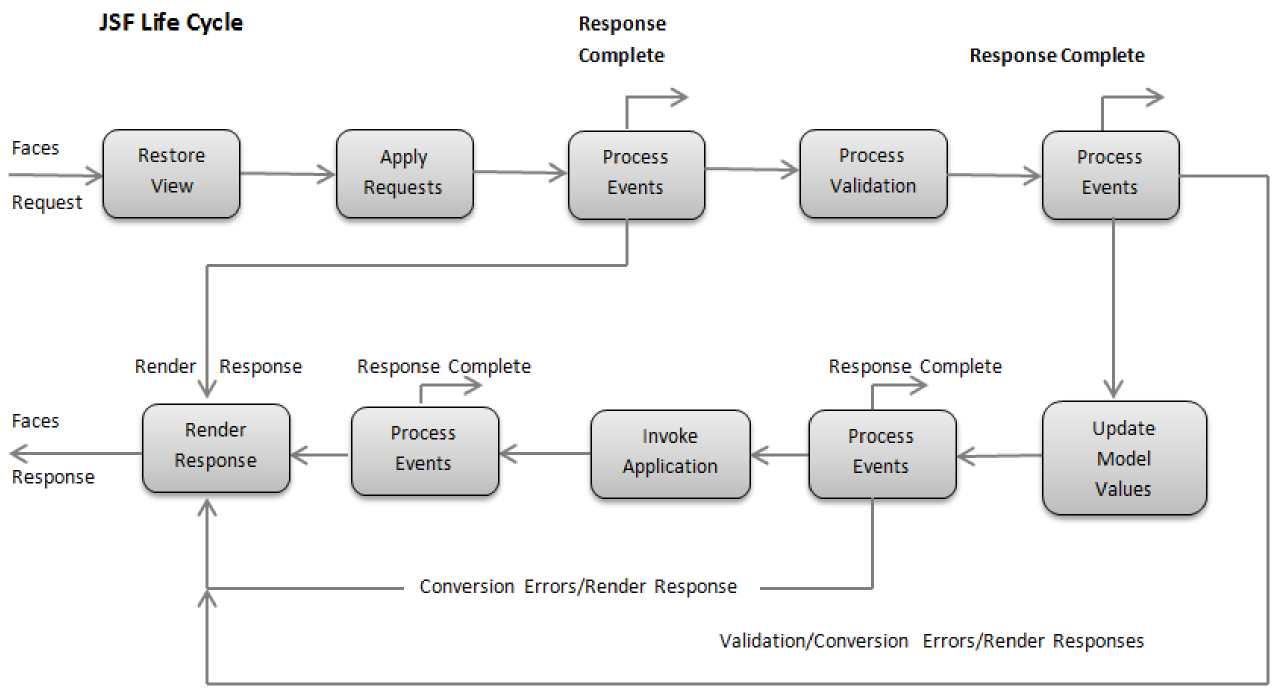


Figure 5 - JSF Life Cycle

CDI (Contexts and Dependency Injection)

The CDI is a component offered by the Java EE platform for managing dependencies between components (known as beans) of an enterprise application (with or without the use of EJBs), associating them with contexts and offering a series of infrastructure services to the application.

EJB (Enterprise JavaBeans)

An EJB typically contains the business logic that acts on the business data. EJB also allows that all parts involving security and transaction can be specified in the form of metadata annotations, or else separately in the Deployment Descriptor which is the web.xml file. Figure 6 shows the EJB container inside a Java EE application.

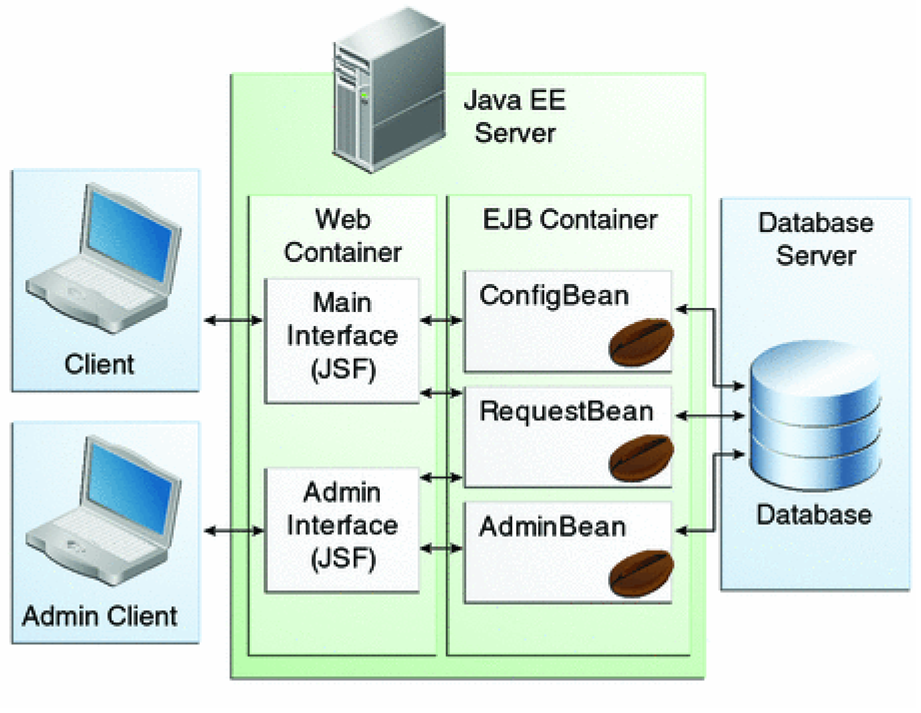


Figure 6 – EJB Container

JAX-RS and JAX-WS

These two components were chosen to be able to display the books in the online store so that other applications or servers can access them. In this way, the monolithic application can scale

JAX-RS (Java API for RESTful Web Services) – is a set of APIs to create web services following the REST architecture (Figure 7).

JAX-WS (Java API for XML Web Services) – is a set of APIs for creating web services in XML. JAX-WS provides a series of annotations that facilitate the development and deployment of both web services clients and servers (endpoints) (Figure 7).

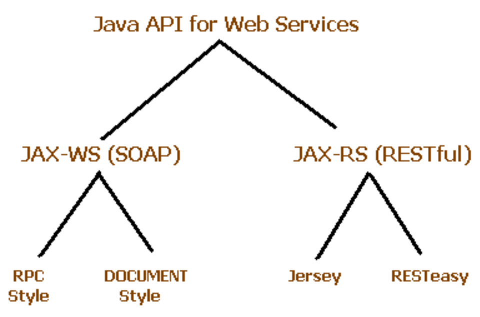


Figure 7 - Webservices created by JAX-RS and JAX-WS

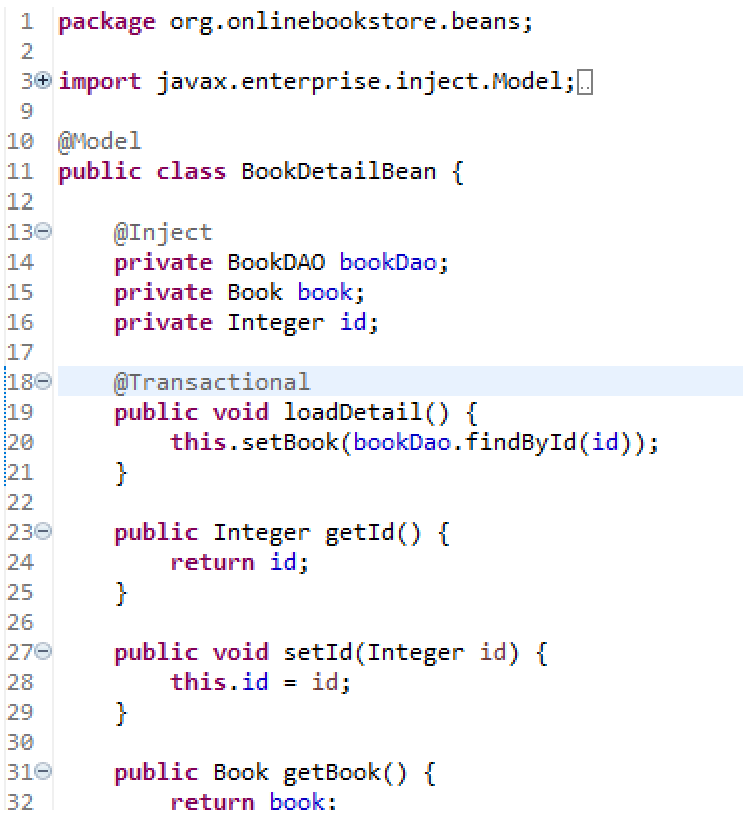
WebSocket

WebSocket was created for implementation in browsers and web servers. The technology can also be used to extend high-performance, distributed applications, and to send and receive messages in larger volumes, using fewer infrastructure resources. In the project, the WebSocket will be used to send promotions created by the Admin.

JTA (Java Transaction API)

JTA is the JAVA EE specification responsible for defining the API needed to control transactions within applications. As transactional control is a feature that is present in almost every project, this specification is used both in JAVA EE containers, such as a Wildfly (server used in the project), and in Servlet Containers.

By adding the @Transactional annotation on top of the method (Figure 8), the application server understands that the method must be executed within a transaction (opening, executing and closing the transaction automatically) in the database. This level of facility only exists because of the CDI. This specification is responsible for defining the behavior of a dependency injection container in JAVA EE.



```java
package org.onlinebookstore.beans;

import javax.enterprise.inject.Model;

@Model
public class BookDetailBean {

    @Inject
    private BookDAO bookDao;
    private Book book;
    private Integer id;

    @Transactional
    public void loadDetail() {
        this.setBook(bookDao.findById(id));
    }

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public Book getBook() {
        return book;
```

Figure 8 - Annotation @Trasancional

JPA (Java Persistence API)

JPA is a specification for storing data in a database. Based on JPA, several implementations are developed and Hibernate was chosen for this project in order to provide an interaction with a relational database, avoiding the development of pure SQL code to manipulate the data present in the database.

MVC Pattern

MVC is a software architecture pattern. MVC suggests a way for you to carry out the division of responsibilities, especially within a web software. The basic principle of MVC is the division of the application into three layers (Figure 9): the user interaction layer (view), the data manipulation layer (model) and the control layer (controller). With MVC, it is possible to separate user interface code from business rules.

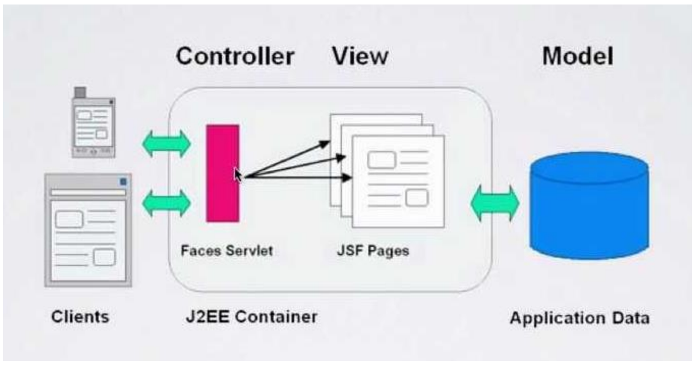


Figure 9 - MVC Pattern

Data-Base

The database chosen was PostgreSQL. PostgreSQL is a tool that acts as a management system for relational databases. Its focus is to allow the implementation of the SQL language in structures.

JMS (Java Message Service)

JMS is a Java API for MOM (Message-Oriented Middleware). A MOM, is a category of software that aims to send and receive messages between distributed applications in an asynchronous, scalable, secure and reliable way. Briefly, using JMS API, two or more applications can communicate by messages with each other. In this project, it will be used to send the promotion to the customers who are viewing the site.

CHAPTER III: SYSTEM ANALYSIS AND DESIGN

The Online Book Store is a web application to sell books. The customer can search, receive promotions while viewing the site and, of course, can buy by paying with any bank card. In the administration part, the admin can log into the system, register new books, view them and create promotions. This chapter shows how these features were achieved. The use of diagrams and flowcharts helps in understanding.

Functional Requirements

The web application, taking the customer view, has just a few requirements:

- Each customer will be able to add or remove books from the shopping cart.
- The customer can see the details about the book.
- After paying the order, an email will be sent to the customer containing the description and the unique ID of the purchase.
- The customer can pay using any bank card.
- The customer can search for a book title.

The web application, by the Admin view:

- The admin will be able to log in in the system.
- Register new books and their authors with its book cover.
- The admin will be able to create promotions and let the customers know through a warning on their screen.

System Requirements:

- Use of Cache in the Data Base for better performance when loading the store's homepage.
- Services available in JSON and XML for viewing books via HTTP request.

Shopping Cart

On the home page, the customer already sees the latest releases and other books. The shopping cart is situated in the upper right corner. The footer of every page contains an input field for searching books by title.

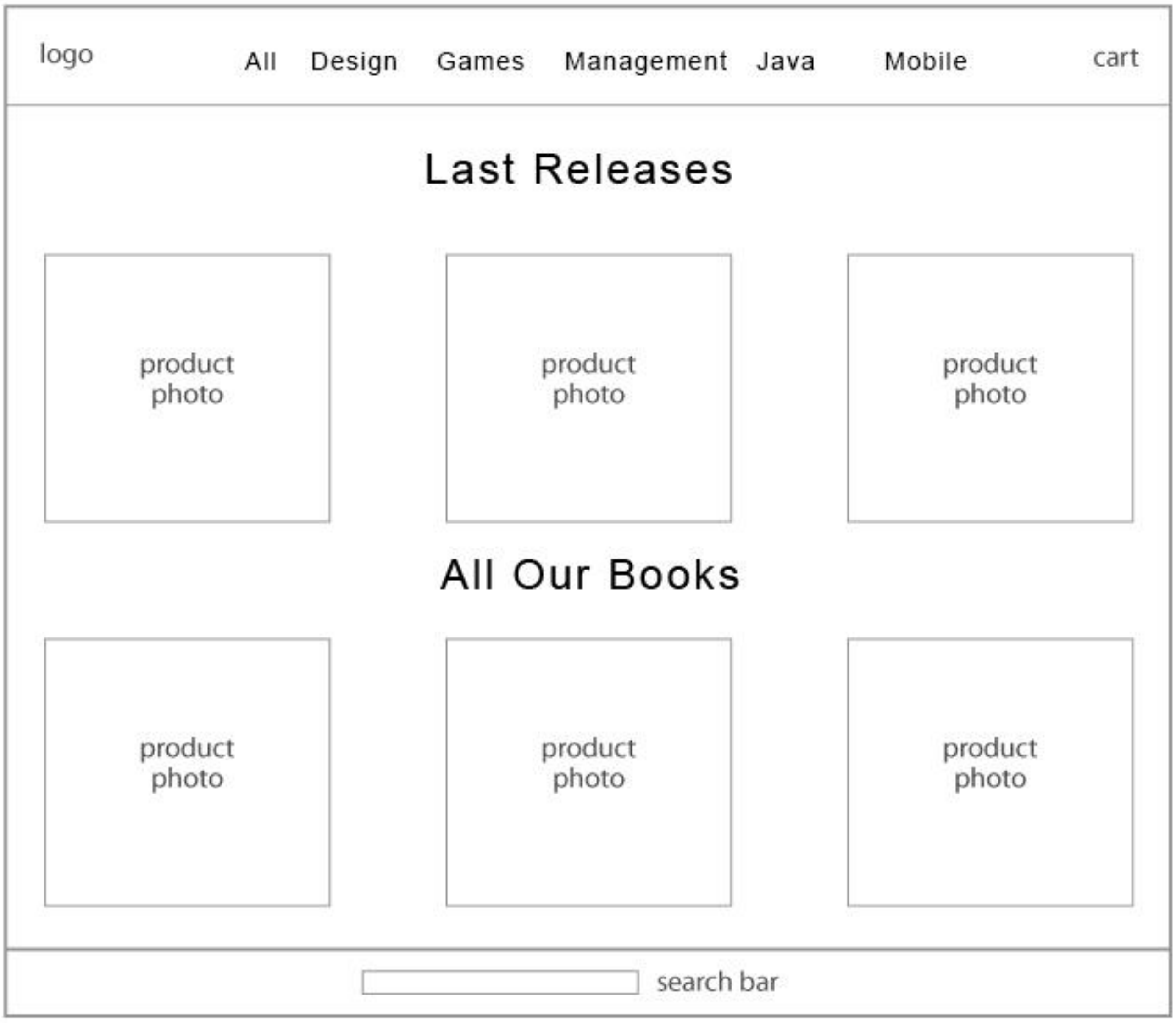


Figure 10 - Home page (index.xhtml)

The book details page makes it possible to add the book to the shopping cart by clicking the Add button.

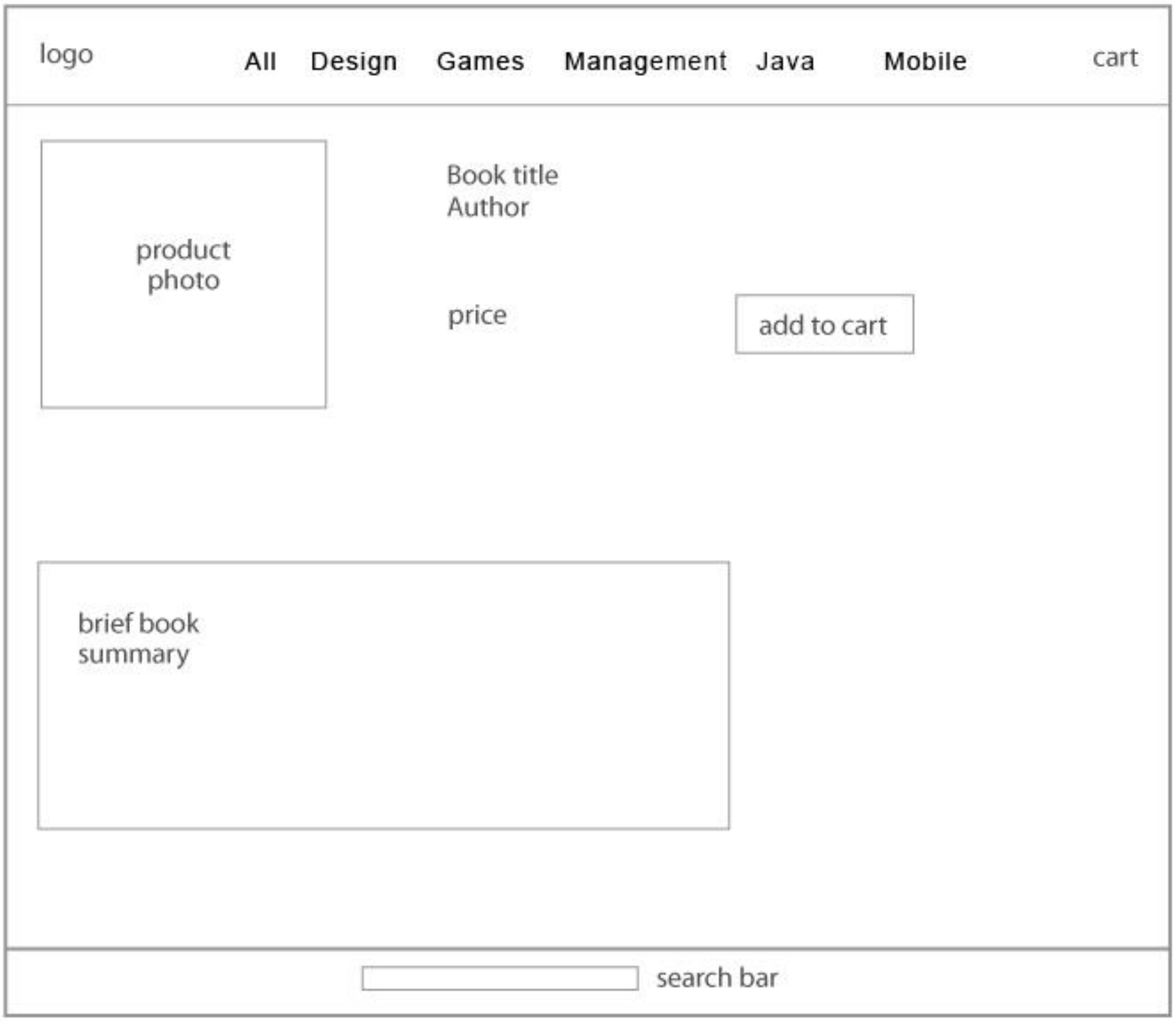


Figure 11 - Detail page (book-detail.xhtml)

By clicking on the button, the book is added to the cart and its quantity can be updated. The Checkout button appears.

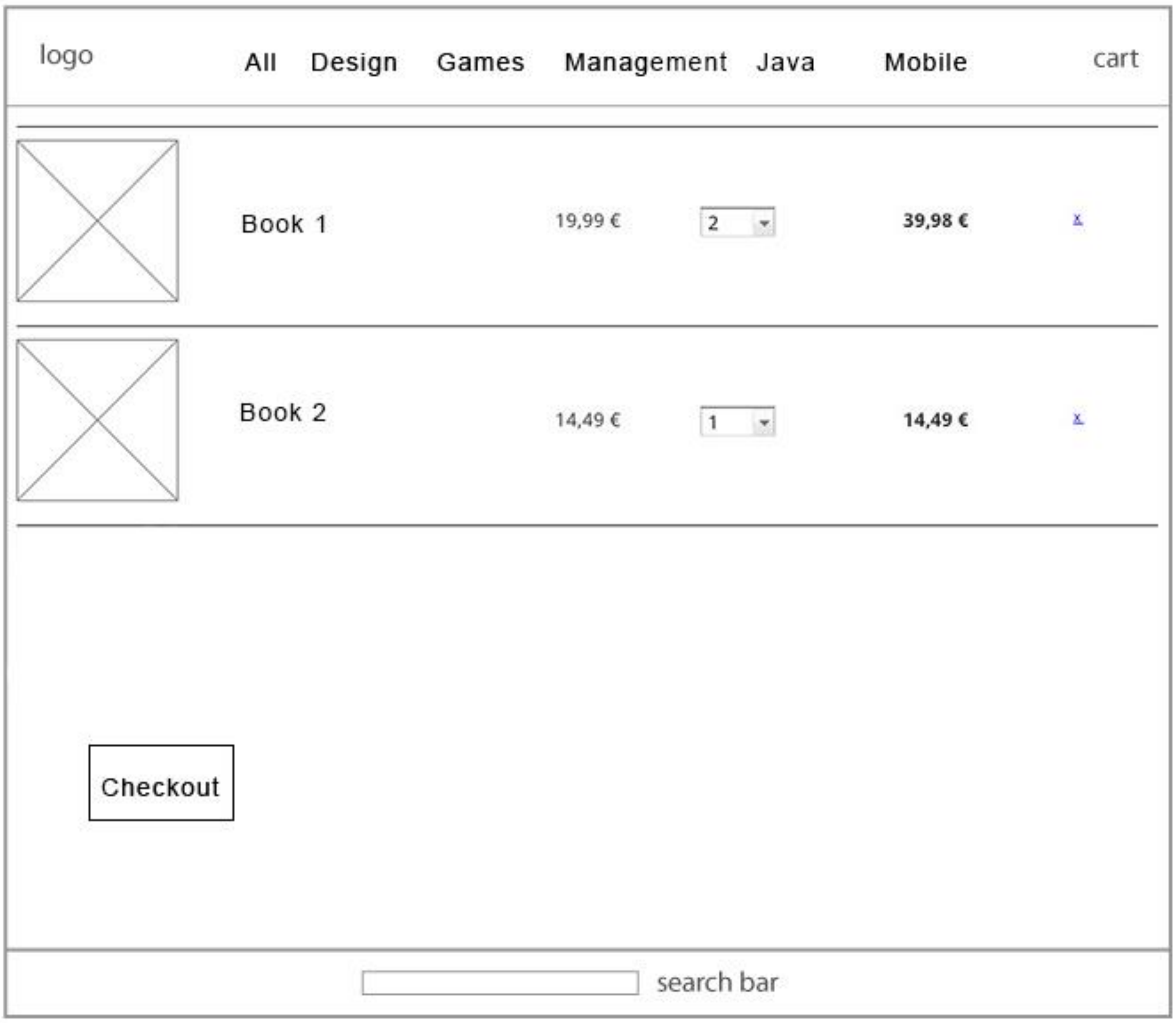


Figure 12 - Review the Cart (cart.xhtml)

The next step is to enter the customer data: full name, email and address.

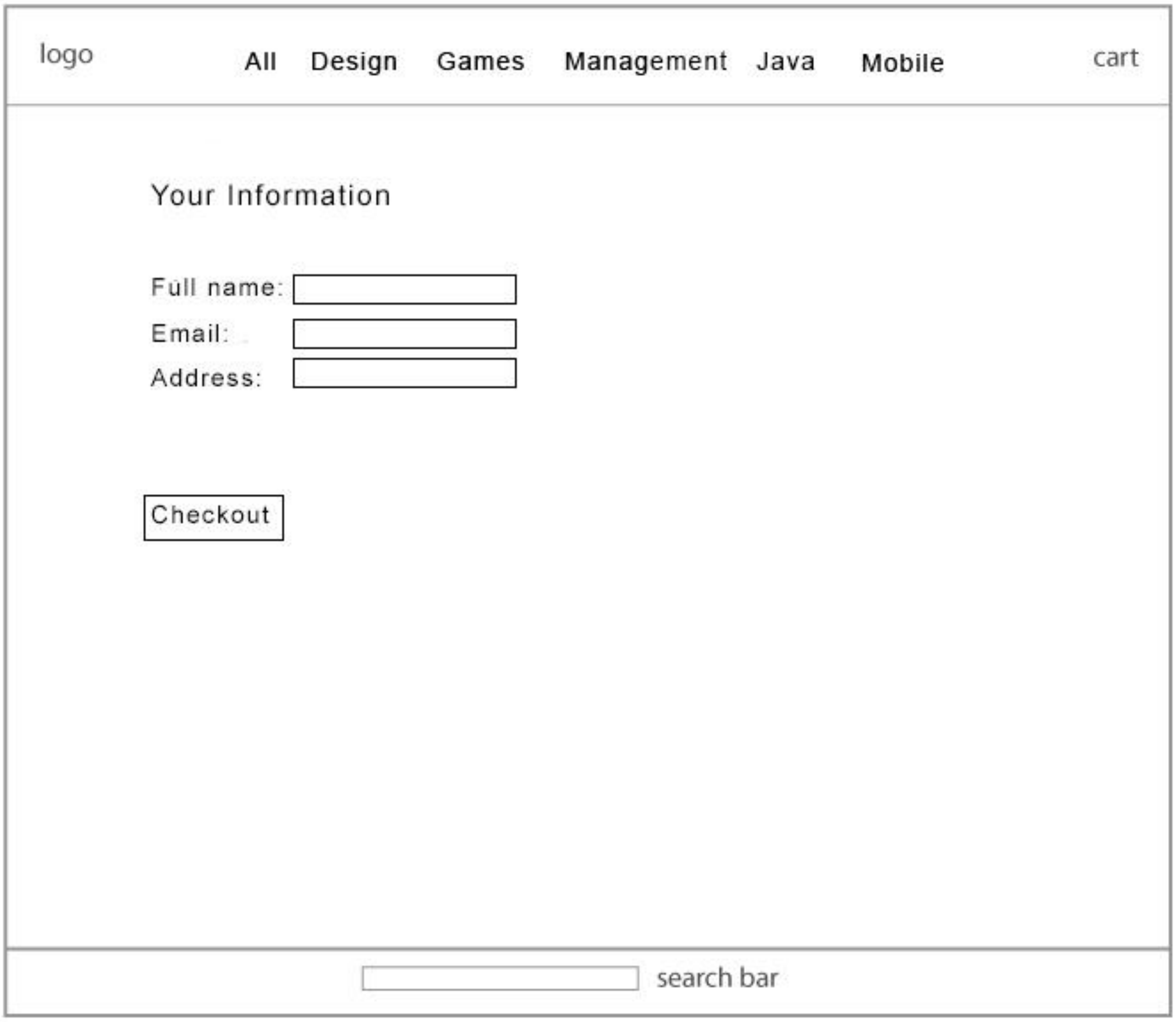


Figure 13 - Customer Information

After this screen, the Paypal screen will appear. Just select the option to pay with any card and enter the information. The Review screen will appear.

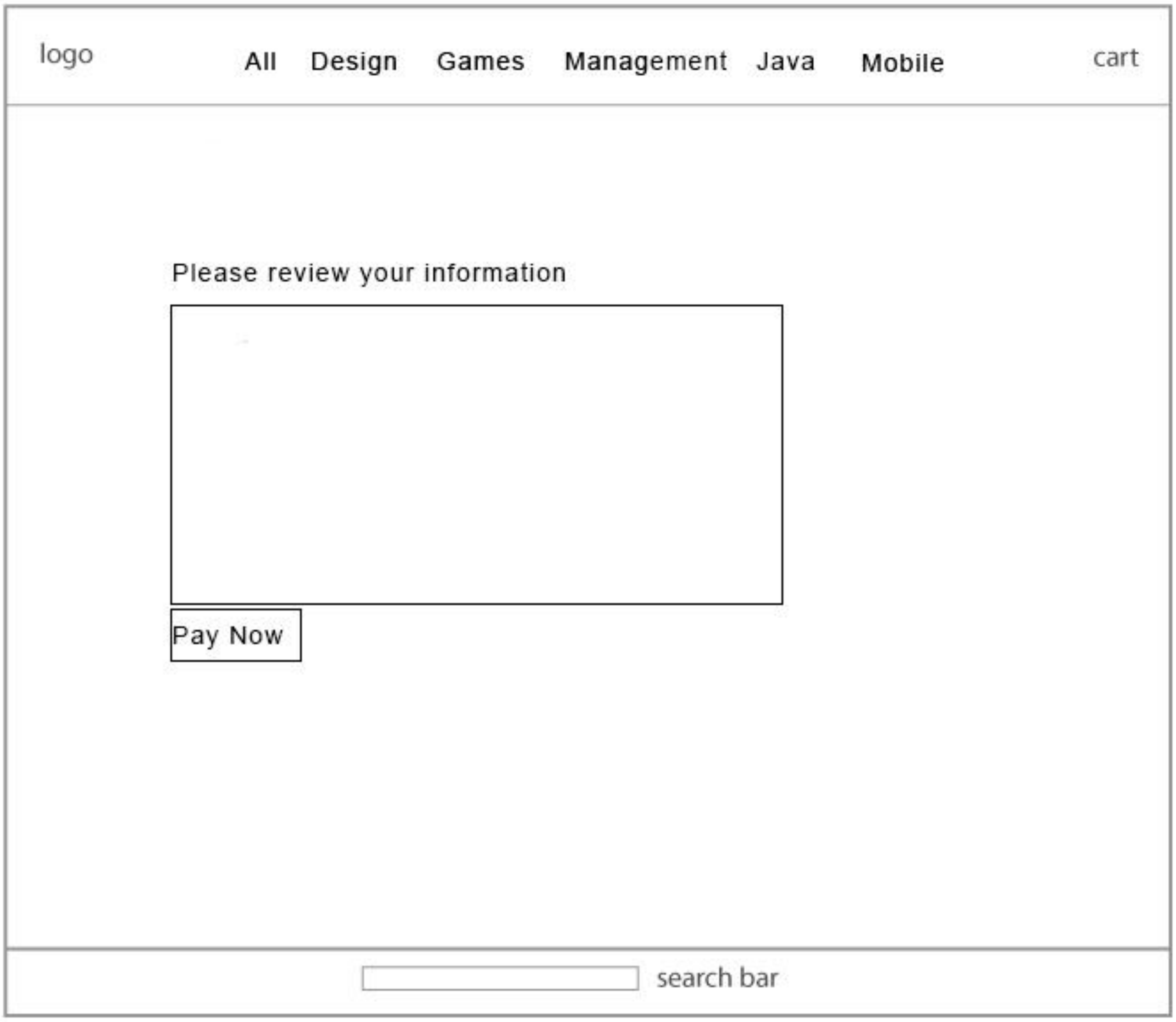


Figure 14 - Review page

After clicking on Pay Now, the payment is made and the next screen is the confirmation screen. The option to receive the purchase data by email is displayed. If the customer wants to receive the information by email, he only needs to click on this option.

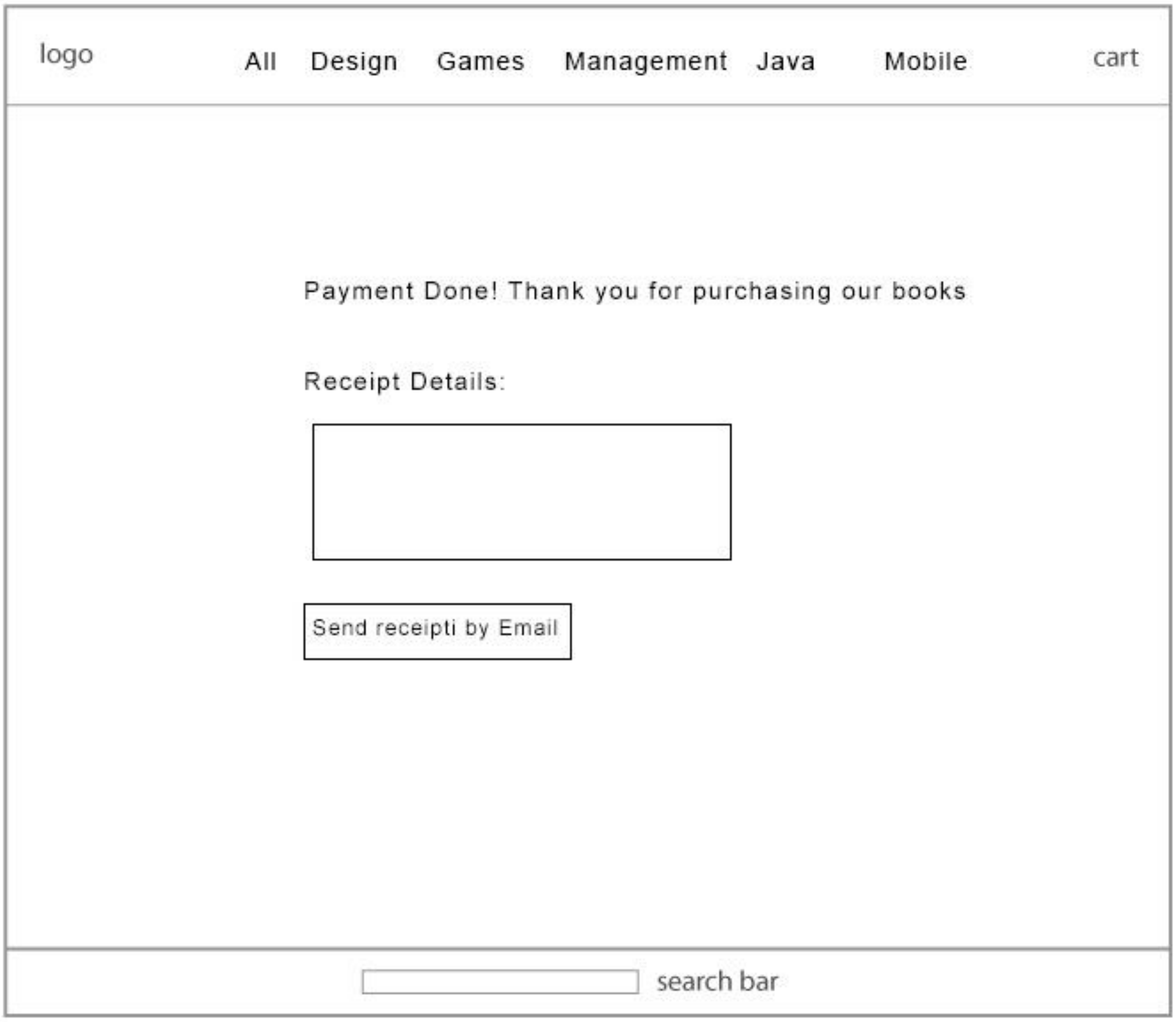


Figure 15 - Payment is done

Now the Admin view: The first difference is the Menu with the options of Register Book, List, Create Promotion, Logout and Home (web application home page).

First the Admin needs to log into the system using a simple Login page. After that, the Admin can register a new book.

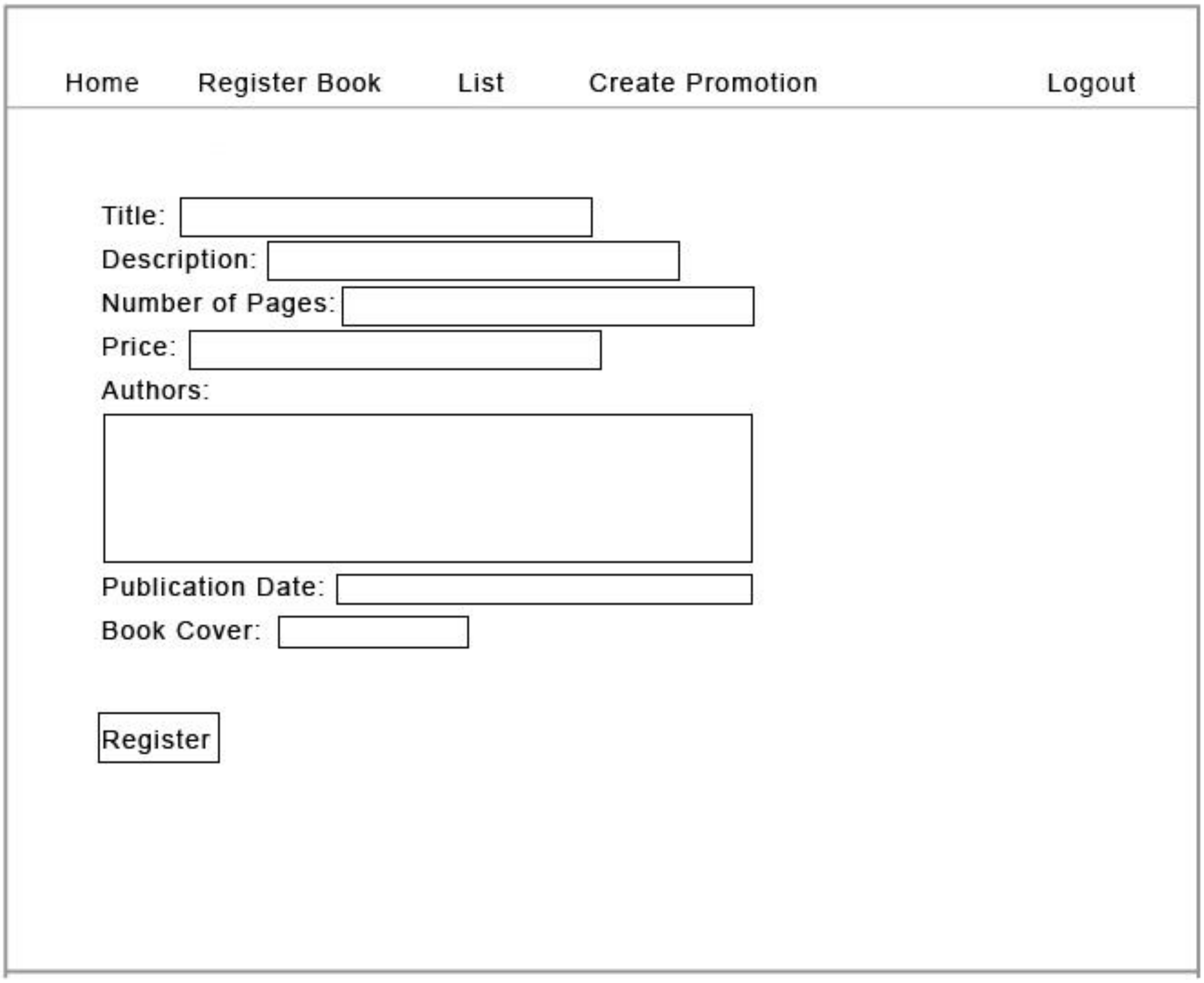
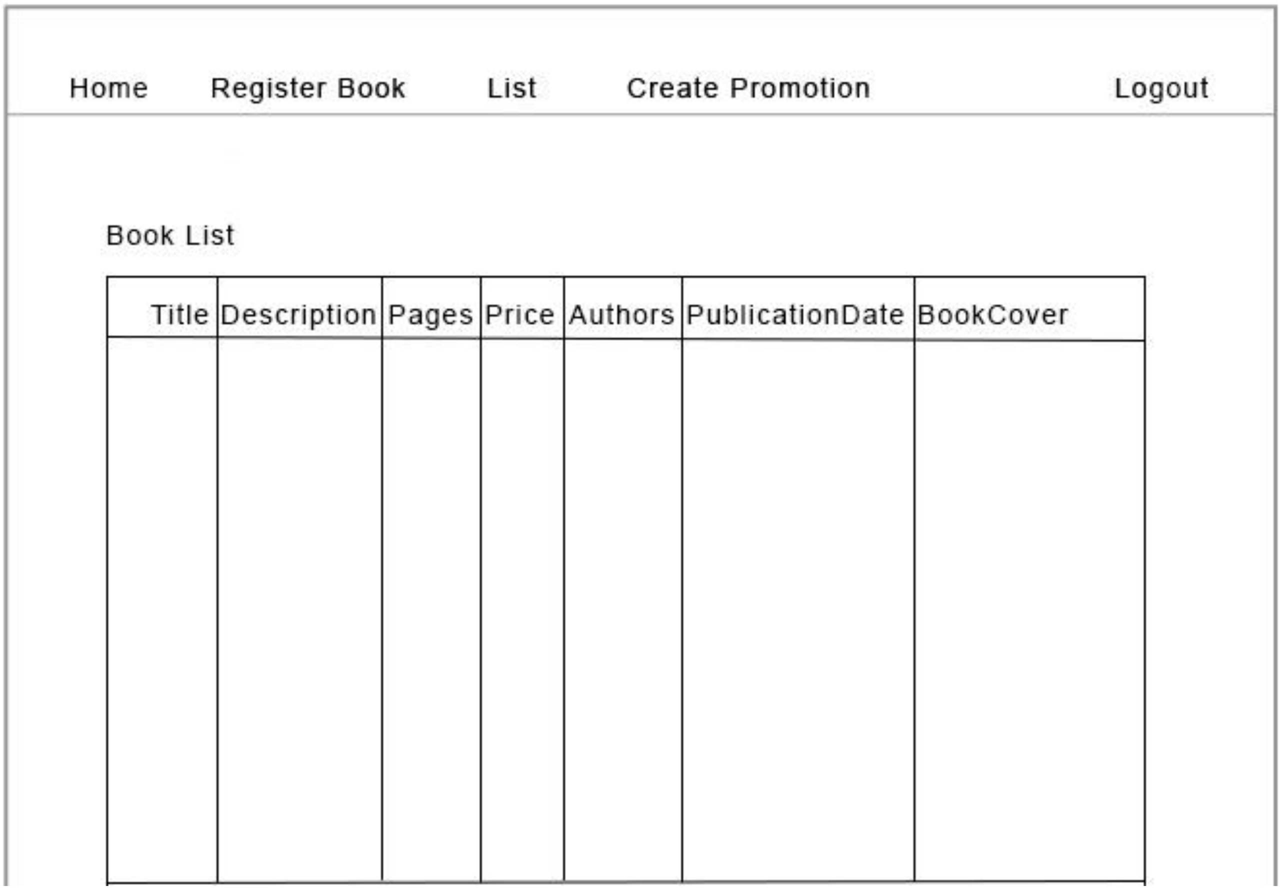


Figure 16 - Register New Book (books/form.xhtml)

Listing all registered books is also another option.



Title	Description	Pages	Price	Authors	PublicationDate	BookCover

Figure 17 - All books (books/list.xhtml)

The admin can also create a promotion and all customers who are browsing the site will receive this notification. The discount will be applied to the price of the book.

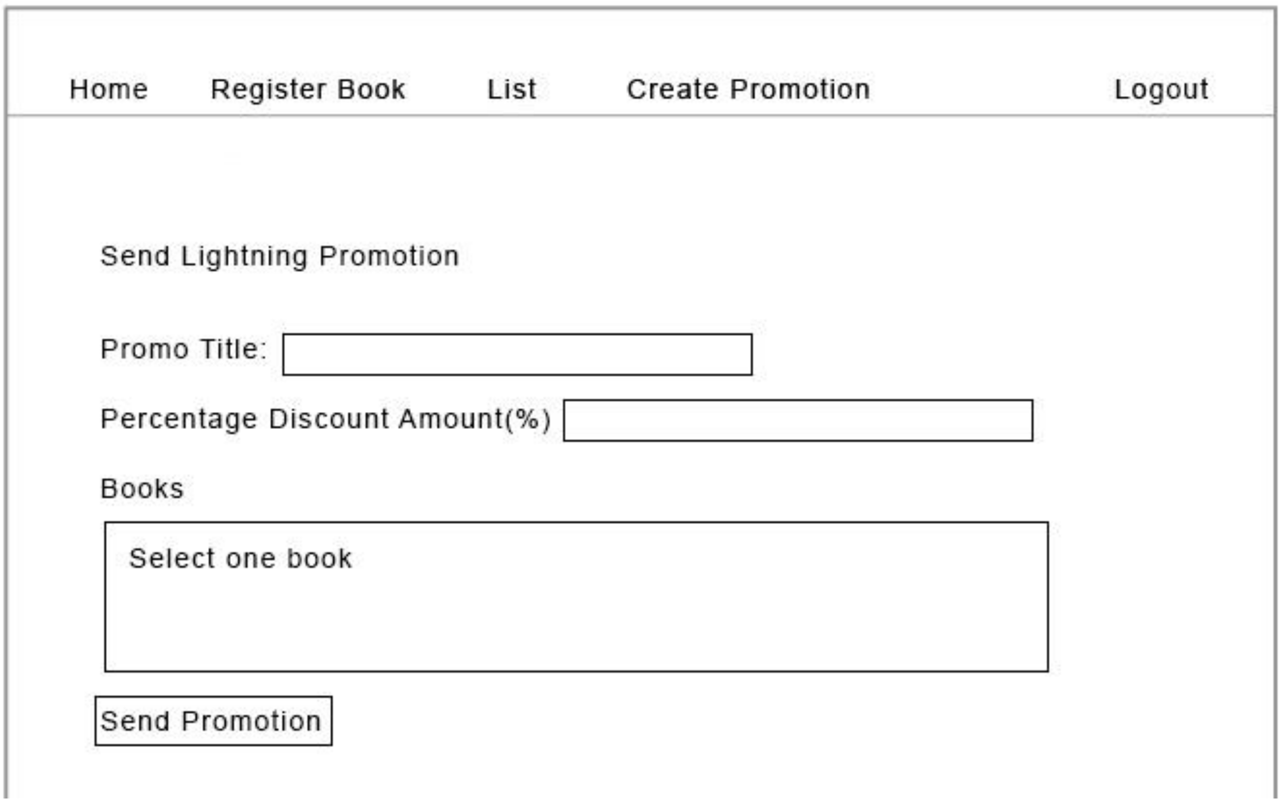


Figure 18 - Promotion page (admin/promo.xhtml)

Database Design

Below is the relational entity diagram.

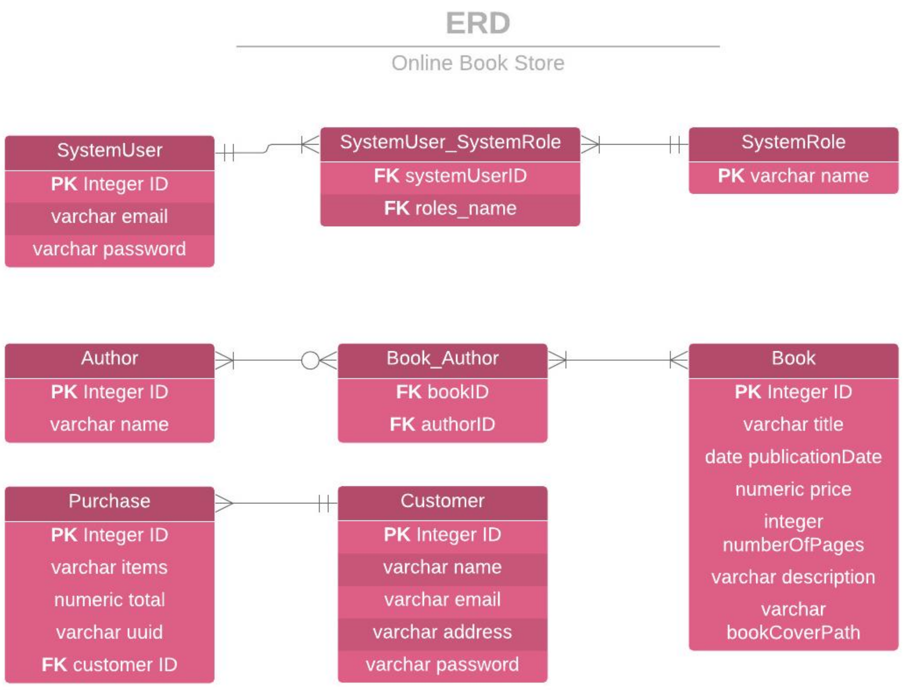


Figure 19 - ERD

Explanation of the Entities

- Table SystemUser: stores admin user data.
- Table SystemRole: stores the roles of each admin user.
- Table Book: stores all the book data.
- Table Author: stores all the author data.
- Table Customer: stores all the customer data.
- Table Purchase: stores all the purchase data. The Item attribute will be filled with the books selected by the class ShoppingCart that uses Session Scope Bean.

Functional Design

The MVC (Model View Controller) design pattern was adopted as the project's architecture, allowing to have an organized and scalable code during development.

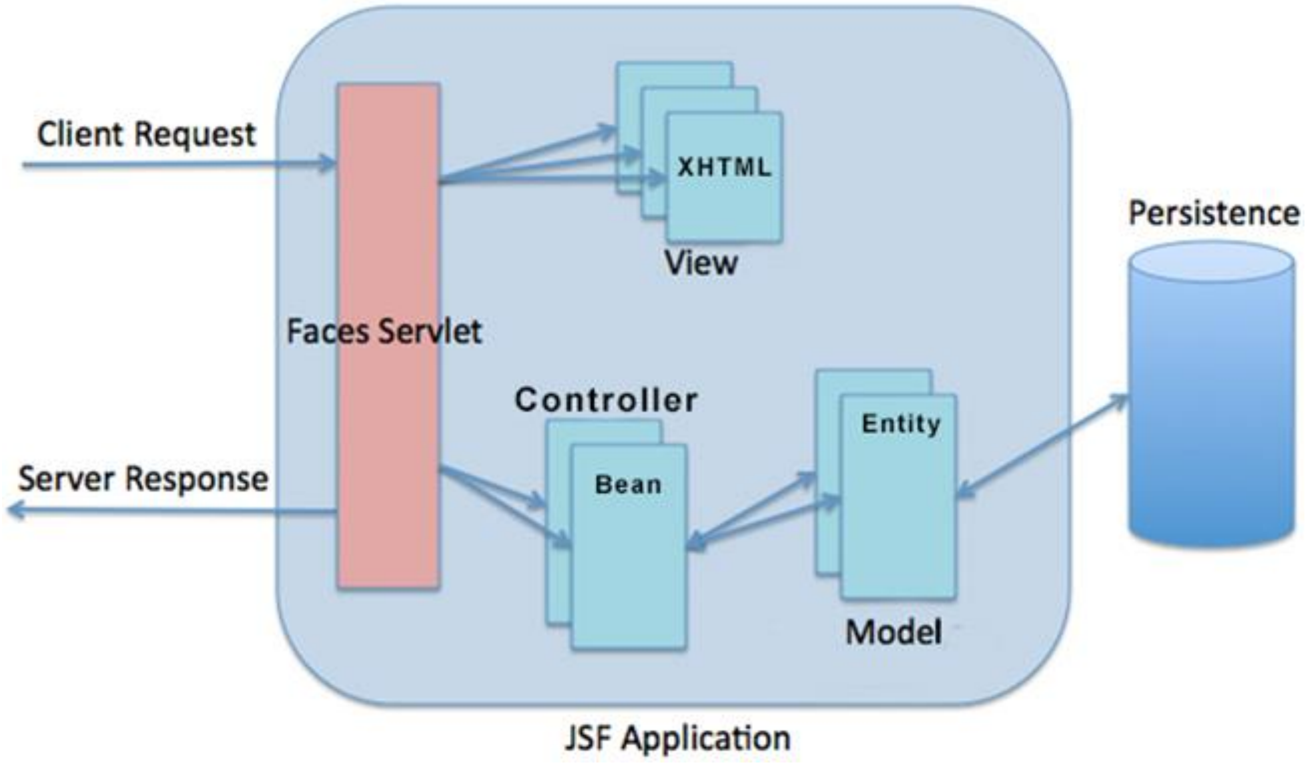


Figure 20 - Design Pattern MVC

CHAPTER IV: IMPLEMENTATION OF THE SYSTEM

This chapter explains, in detail, the development process of the application.

Architecture Considerations

The project was developed using components of the Java EE platform. Following MVC Design Pattern and DAO (Data Access Object).

Technologies Used

The development environment used to build the application were:

IDE (Interface Development Environment)

- Eclipse

Front End

- HTML
- Bootstrap
- Bootstrap Framework
- JavaScript
- XML
- JSON

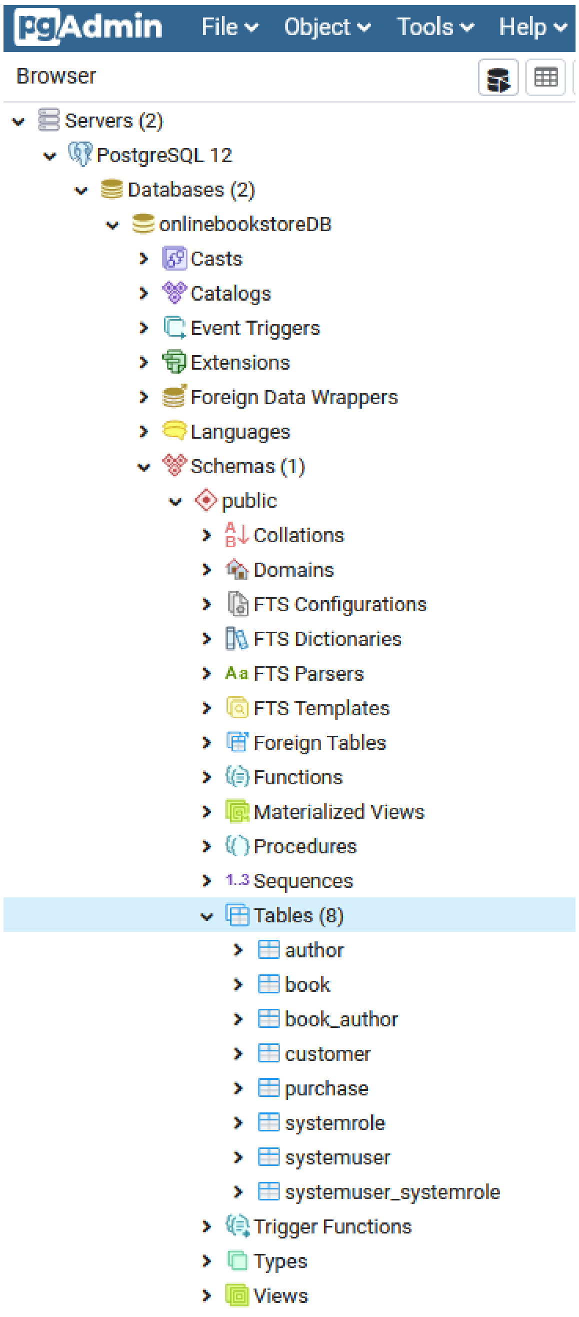
Back End

- JSF
- JSF
- CDI
- JMS
- Websockets
- JAX-RS and JAX-WS

Database

- Relational PostgreSQL Database
- pgAdmin software to manage the data from PostgreSQL

Database Structure:


pgAdmin File Object Tools Help

Browser

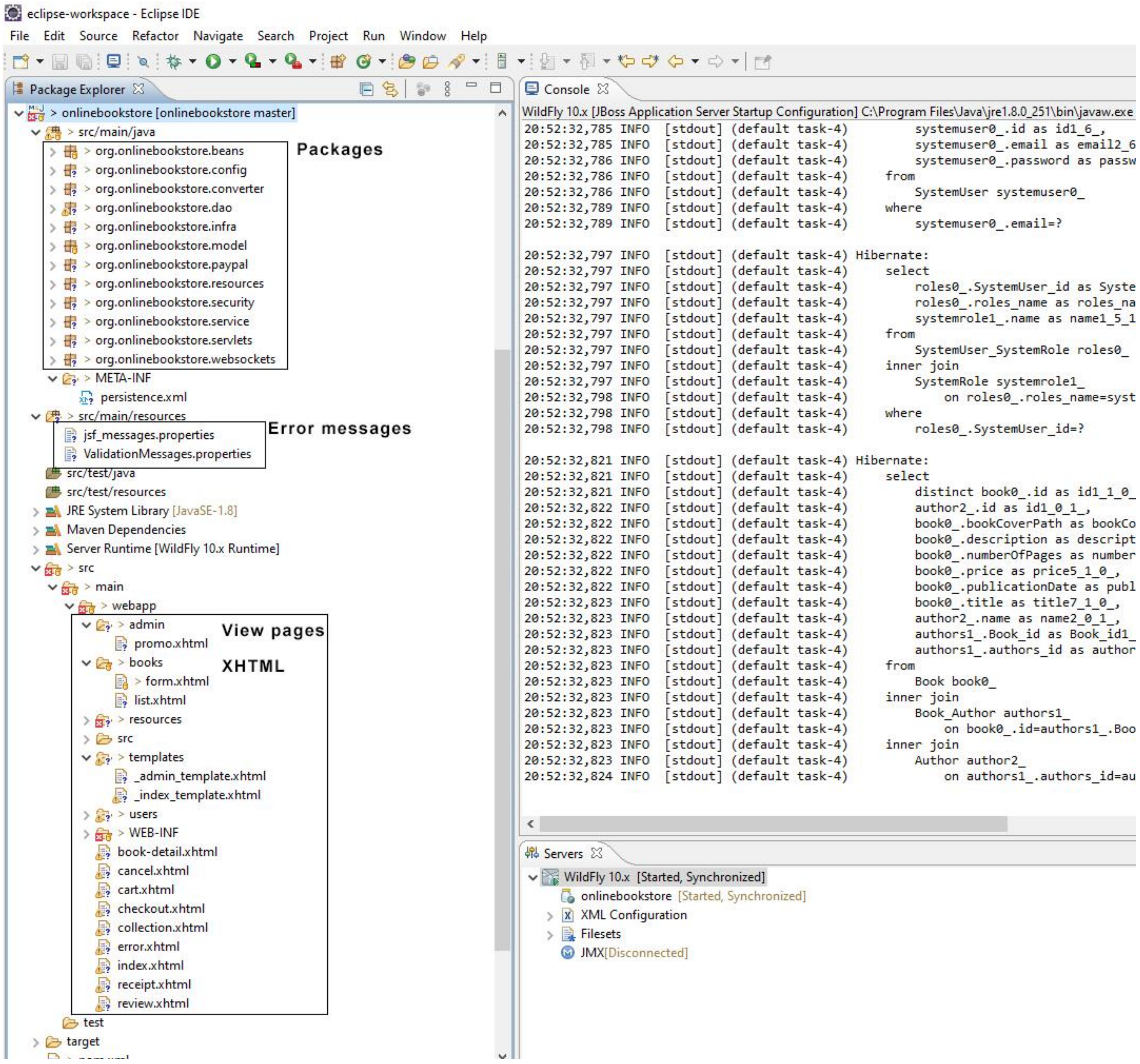
```
Servers (2)
  PostgreSQL 12
    Databases (2)
      onlinebookstoreDB
        Casts
        Catalogs
        Event Triggers
        Extensions
        Foreign Data Wrappers
        Languages
        Schemas (1)
          public
            Collations
            Domains
            FTS Configurations
            FTS Dictionaries
            FTS Parsers
            FTS Templates
            Foreign Tables
            Functions
            Materialized Views
            Procedures
            Sequences
            Tables (8)
              author
              book
              book_author
              customer
              purchase
              systemrole
              systemuser
              systemuser_systemrole
            Trigger Functions
            Types
            Views
```

Figure 21 - pgAdmin interface

Server

- Wildfly Server

From the Eclipse IDE perspective, the project structure is identified as follows:


eclipse-workspace - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
Packages
Error messages
View pages
XHTML
Console
Servers

```
> onlinebookstore [onlinebookstore master]
  > src/main/java
    > org.onlinebookstore.beans
    > org.onlinebookstore.config
    > org.onlinebookstore.converter
    > org.onlinebookstore.dao
    > org.onlinebookstore.infra
    > org.onlinebookstore.model
    > org.onlinebookstore.paypal
    > org.onlinebookstore.resources
    > org.onlinebookstore.security
    > org.onlinebookstore.service
    > org.onlinebookstore.servlets
    > org.onlinebookstore.websockets
    > META-INF
        persistence.xml
  > src/main/resources
      jsf_messages.properties
      ValidationMessages.properties
  src/test/java
  src/test/resources
  JRE System Library [JavaSE-1.8]
  Maven Dependencies
  Server Runtime [WildFly 10.x Runtime]
  > src
    > main
      > webapp
        > admin
            promo.xhtml
        > books
            > form.xhtml
            list.xhtml
        > resources
          src
        > templates
            _admin_template.xhtml
            _index_template.xhtml
        > users
        > WEB-INF
          book-detail.xhtml
          cancel.xhtml
          cart.xhtml
          checkout.xhtml
          collection.xhtml
          error.xhtml
          index.xhtml
          receipt.xhtml
          review.xhtml
      test
  target
```

```
WildFly 10.x [JBoss Application Server Startup Configuration] C:\Program Files\Java\jre1.8.0_251\bin\javaw.exe
20:52:32,785 INFO  [stdout] (default task-4)         systemuser0_.id as id1_6_,
20:52:32,785 INFO  [stdout] (default task-4)         systemuser0_.email as email2_6
20:52:32,786 INFO  [stdout] (default task-4)         systemuser0_.password as passw
20:52:32,786 INFO  [stdout] (default task-4)     from
20:52:32,786 INFO  [stdout] (default task-4)         SystemUser systemuser0_
20:52:32,789 INFO  [stdout] (default task-4)     where
20:52:32,789 INFO  [stdout] (default task-4)         systemuser0_.email=?

20:52:32,797 INFO  [stdout] (default task-4) Hibernate:
20:52:32,797 INFO  [stdout] (default task-4)     select
20:52:32,797 INFO  [stdout] (default task-4)         roles0_.SystemUser_id as Syste
20:52:32,797 INFO  [stdout] (default task-4)         roles0_.roles_name as roles_na
20:52:32,797 INFO  [stdout] (default task-4)         systemrole1_.name as name1_5_1
20:52:32,797 INFO  [stdout] (default task-4)     from
20:52:32,797 INFO  [stdout] (default task-4)         SystemUser_SystemRole roles0_
20:52:32,797 INFO  [stdout] (default task-4)     inner join
20:52:32,797 INFO  [stdout] (default task-4)         SystemRole systemrole1_
20:52:32,798 INFO  [stdout] (default task-4)             on roles0_.roles_name=syst
20:52:32,798 INFO  [stdout] (default task-4)     where
20:52:32,798 INFO  [stdout] (default task-4)         roles0_.SystemUser_id=?

20:52:32,821 INFO  [stdout] (default task-4) Hibernate:
20:52:32,821 INFO  [stdout] (default task-4)     select
20:52:32,821 INFO  [stdout] (default task-4)         distinct book0_.id as id1_1_0_
20:52:32,822 INFO  [stdout] (default task-4)         author2_.id as id1_0_1_,
20:52:32,822 INFO  [stdout] (default task-4)         book0_.bookCoverPath as bookCo
20:52:32,822 INFO  [stdout] (default task-4)         book0_.description as descript
20:52:32,822 INFO  [stdout] (default task-4)         book0_.numberOfPages as number
20:52:32,822 INFO  [stdout] (default task-4)         book0_.price as price5_1_0_,
20:52:32,822 INFO  [stdout] (default task-4)         book0_.publicationDate as publ
20:52:32,823 INFO  [stdout] (default task-4)         book0_.title as title7_1_0_,
20:52:32,823 INFO  [stdout] (default task-4)         author2_.name as name2_0_1_,
20:52:32,823 INFO  [stdout] (default task-4)         authors1_.Book_id as Book_id1_
20:52:32,823 INFO  [stdout] (default task-4)         authors1_.authors_id as author
20:52:32,823 INFO  [stdout] (default task-4)     from
20:52:32,823 INFO  [stdout] (default task-4)         Book book0_
20:52:32,823 INFO  [stdout] (default task-4)     inner join
20:52:32,823 INFO  [stdout] (default task-4)         Book_Author authors1_
20:52:32,823 INFO  [stdout] (default task-4)             on book0_.id=authors1_.Boo
20:52:32,823 INFO  [stdout] (default task-4)     inner join
20:52:32,823 INFO  [stdout] (default task-4)         Author author2_
20:52:32,824 INFO  [stdout] (default task-4)             on authors1_.authors_id=au
```

```
WildFly 10.x  [Started, Synchronized]
    onlinebookstore [Started, Synchronized]
  > XML Configuration
  > Filesets
    JMX[Disconnected]
```

Figure 22 - Eclipse IDE view

Project Packages

The project has several packages. Each of them will be listed below:

- Beans: It's the Controller layer. It has the business rules, redirects to XHTML pages and makes transactions in the database.
- Config: Responsible for JMS configuration, web service path and FacesContext.
- Converter: Author and Date Object Converters between View and Controllers.
- Dao: Data Access Object layer to access the database in the JTA (Java Transaction API) mode.
- Infra: Infrastructure configuration for saving and displaying images and sending email.
- Model: It represents the Entities.
- Paypal: The payment gateway integration.
- Resources: The Web Service to send the books in JSON, or XML, format.
- Security: Current user logged and password encryption.
- Service: PurchaseEmailSender class to send the email containing the purchase data.

- Servlets: To get image from server.
- Websockets: Responsible for keeping in a list the users that are in the application and creating a communication channel for sending promotions.

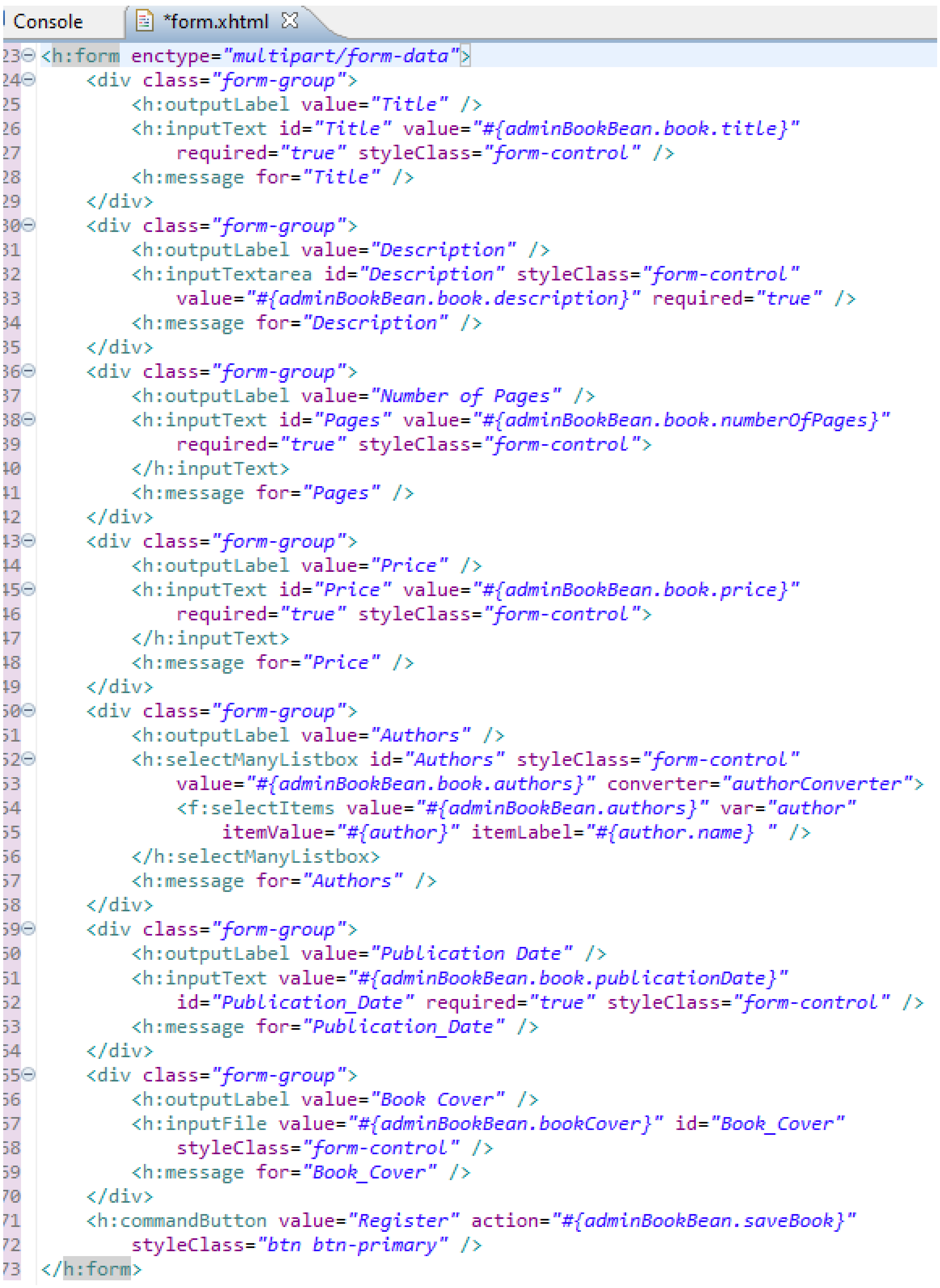
Other important files:

- persistence.xml: File responsible for the configuration of persistence units.
- beans.xml: The beans.xml file is the bean archive descriptor for CDI applications. It can be used for any CDI compliant container, such as Weld which is included in WildFly application server.
- faces-config.xml: to configure as custom JSF validation message.
- Jboss-web.xml: to configure the WildFly server to connect to the database to perform Login.
- web.xml: it is a web application deployment descriptor. It serves to configure what will be the home page, URLs that need login, CACHE of the application.
- pom.xml: to manage the project's dependencies.

Implementation

All other parts of the project basically follow the same flow that was implemented this way, in this example for registering a new book:

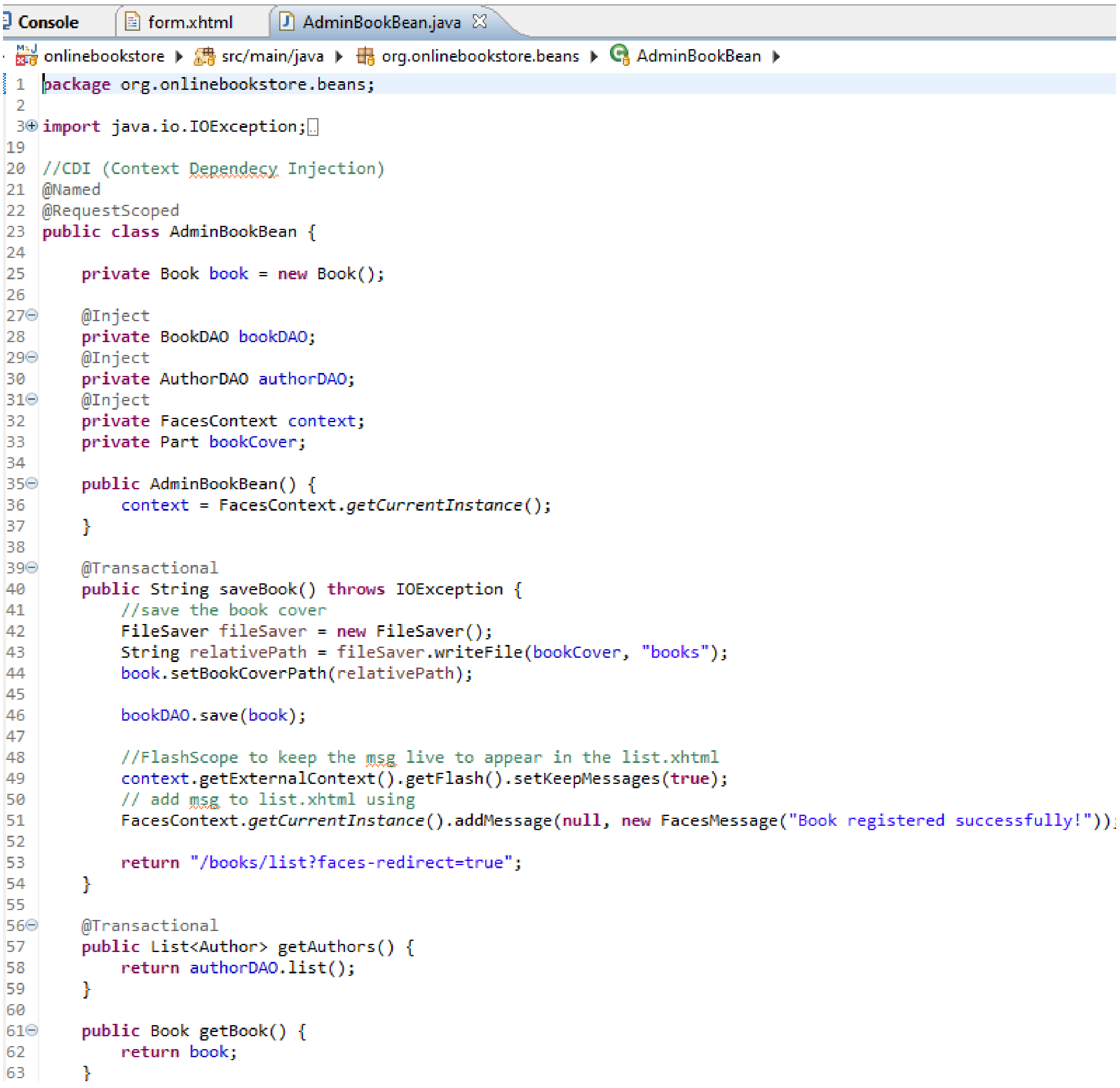
1. The view pages (.XHTML files) are linked with their respective beans using DataBinding or Expression Language. Figure 23 shows the registration form for a book.



```
<h:form enctype="multipart/form-data">
    <div class="form-group">
        <h:outputLabel value="Title" />
        <h:inputText id="Title" value="#{adminBookBean.book.title}"
            required="true" styleClass="form-control" />
        <h:message for="Title" />
    </div>
    <div class="form-group">
        <h:outputLabel value="Description" />
        <h:inputTextarea id="Description" styleClass="form-control"
            value="#{adminBookBean.book.description}" required="true" />
        <h:message for="Description" />
    </div>
    <div class="form-group">
        <h:outputLabel value="Number of Pages" />
        <h:inputText id="Pages" value="#{adminBookBean.book.numberOfPages}"
            required="true" styleClass="form-control">
        </h:inputText>
        <h:message for="Pages" />
    </div>
    <div class="form-group">
        <h:outputLabel value="Price" />
        <h:inputText id="Price" value="#{adminBookBean.book.price}"
            required="true" styleClass="form-control">
        </h:inputText>
        <h:message for="Price" />
    </div>
    <div class="form-group">
        <h:outputLabel value="Authors" />
        <h:selectManyListbox id="Authors" styleClass="form-control"
            value="#{adminBookBean.book.authors}" converter="authorConverter">
            <f:selectItems value="#{adminBookBean.authors}" var="author"
                itemValue="#{author}" itemLabel="#{author.name} " />
        </h:selectManyListbox>
        <h:message for="Authors" />
    </div>
    <div class="form-group">
        <h:outputLabel value="Publication Date" />
        <h:inputText value="#{adminBookBean.book.publicationDate}"
            id="Publication_Date" required="true" styleClass="form-control" />
        <h:message for="Publication_Date" />
    </div>
    <div class="form-group">
        <h:outputLabel value="Book Cover" />
        <h:inputFile value="#{adminBookBean.bookCover}" id="Book_Cover"
            styleClass="form-control" />
        <h:message for="Book_Cover" />
    </div>
    <h:commandButton value="Register" action="#{adminBookBean.saveBook}"
        styleClass="btn btn-primary" />
</h:form>
```

Figure 23 - form.xhtml to register a book

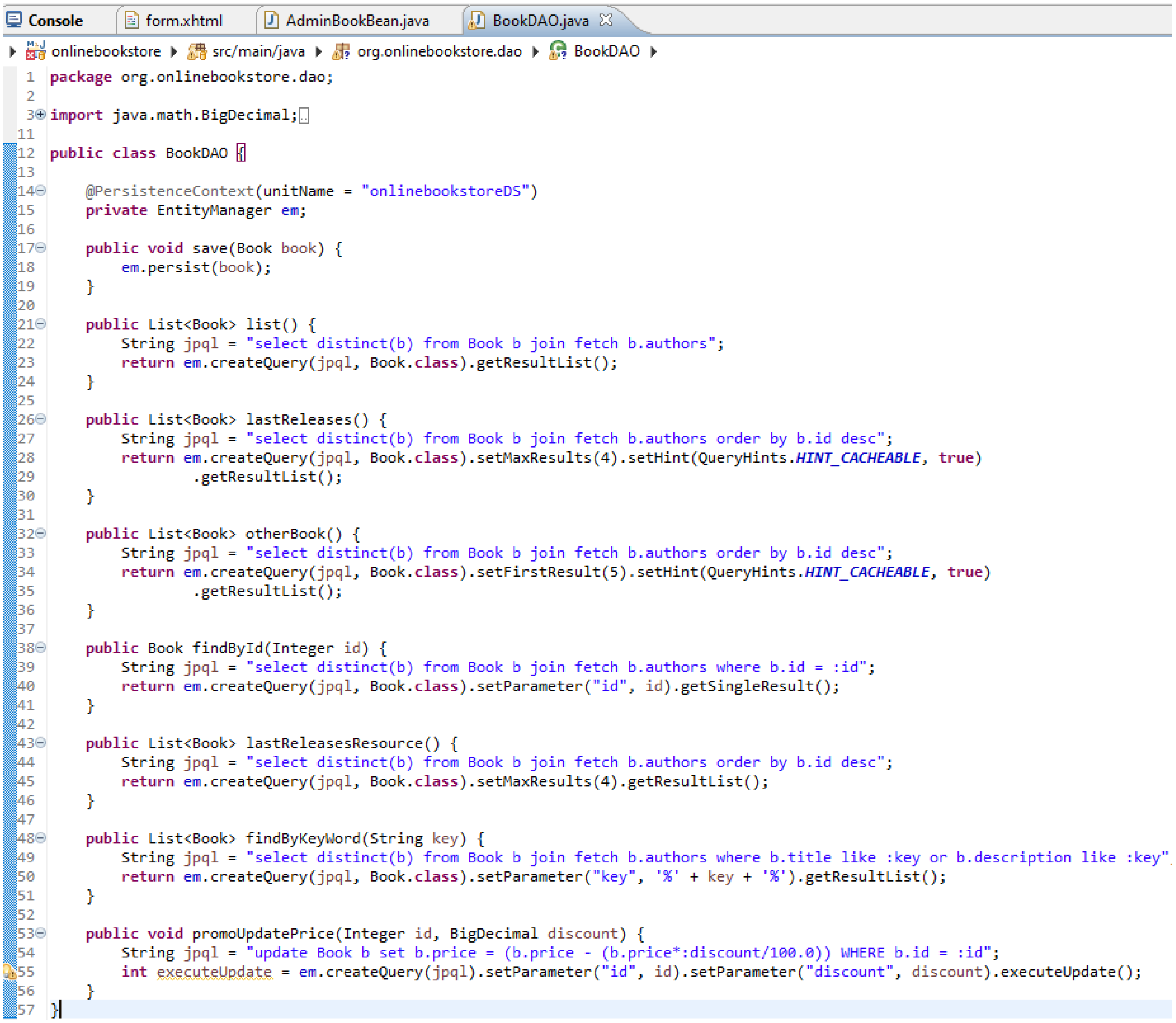
2. In the Controller layer (Figure 24), the CDI links to the view. Here are the application business rules, field types and access for the DAO (Data Access Object) layer.



```java
package org.onlinebookstore.beans;

import java.io.IOException;

//CDI (Context Dependecy Injection)
@Named
@RequestScoped
public class AdminBookBean {

    private Book book = new Book();

    @Inject
    private BookDAO bookDAO;
    @Inject
    private AuthorDAO authorDAO;
    @Inject
    private FacesContext context;
    private Part bookCover;

    public AdminBookBean() {
        context = FacesContext.getCurrentInstance();
    }

    @Transactional
    public String saveBook() throws IOException {
        //save the book cover
        FileSaver fileSaver = new FileSaver();
        String relativePath = fileSaver.writeFile(bookCover, "books");
        book.setBookCoverPath(relativePath);

        bookDAO.save(book);

        //FlashScope to keep the msg live to appear in the list.xhtml
        context.getExternalContext().getFlash().setKeepMessages(true);
        // add msg to list.xhtml using
        FacesContext.getCurrentInstance().addMessage(null, new FacesMessage("Book registered successfully!"));

        return "/books/list?faces-redirect=true";
    }

    @Transactional
    public List<Author> getAuthors() {
        return authorDAO.list();
    }

    public Book getBook() {
        return book;
    }
```

Figure 24 - CDI for form.xhtml

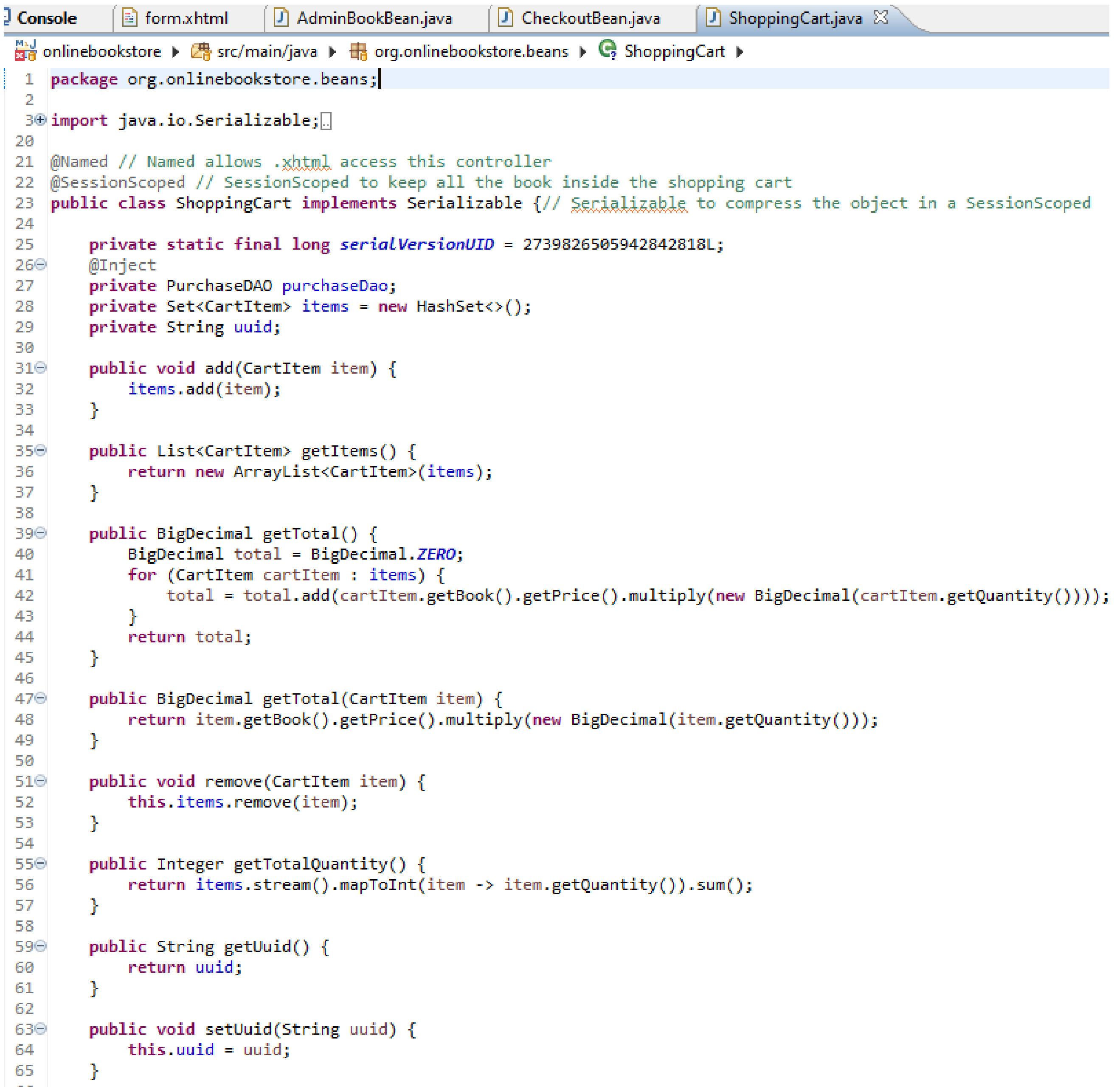
3. DAO layer (Figure 25) to register a new book in the database using Entity Manager by Data Source settings on WildFly server.



```java
package org.onlinebookstore.dao;

import java.math.BigDecimal;

public class BookDAO {

    @PersistenceContext(unitName = "onlinebookstoreDS")
    private EntityManager em;

    public void save(Book book) {
        em.persist(book);
    }

    public List<Book> list() {
        String jpql = "select distinct(b) from Book b join fetch b.authors";
        return em.createQuery(jpql, Book.class).getResultList();
    }

    public List<Book> lastReleases() {
        String jpql = "select distinct(b) from Book b join fetch b.authors order by b.id desc";
        return em.createQuery(jpql, Book.class).setMaxResults(4).setHint(QueryHints.HINT_CACHEABLE, true)
                .getResultList();
    }

    public List<Book> otherBook() {
        String jpql = "select distinct(b) from Book b join fetch b.authors order by b.id desc";
        return em.createQuery(jpql, Book.class).setFirstResult(5).setHint(QueryHints.HINT_CACHEABLE, true)
                .getResultList();
    }

    public Book findById(Integer id) {
        String jpql = "select distinct(b) from Book b join fetch b.authors where b.id = :id";
        return em.createQuery(jpql, Book.class).setParameter("id", id).getSingleResult();
    }

    public List<Book> lastReleasesResource() {
        String jpql = "select distinct(b) from Book b join fetch b.authors order by b.id desc";
        return em.createQuery(jpql, Book.class).setMaxResults(4).getResultList();
    }

    public List<Book> findByKeyWord(String key) {
        String jpql = "select distinct(b) from Book b join fetch b.authors where b.title like :key or b.description like :key"
        return em.createQuery(jpql, Book.class).setParameter("key", '%' + key + '%').getResultList();
    }

    public void promoUpdatePrice(Integer id, BigDecimal discount) {
        String jpql = "update Book b set b.price = (b.price - (b.price*:discount/100.0)) WHERE b.id = :id";
        int executeUpdate = em.createQuery(jpql).setParameter("id", id).setParameter("discount", discount).executeUpdate();
    }
}
```

Figure 25 - BookDAO

The shopping cart has a Session Bean (Figure 26). This type of bean exists while the user is browsing the site. In this way, items can be added to the cart and all information is kept.


Console
form.xhtml
AdminBookBean.java
CheckoutBean.java
ShoppingCart.java
onlinebookstore ▸ src/main/java ▸ org.onlinebookstore.beans ▸ ShoppingCart ▸

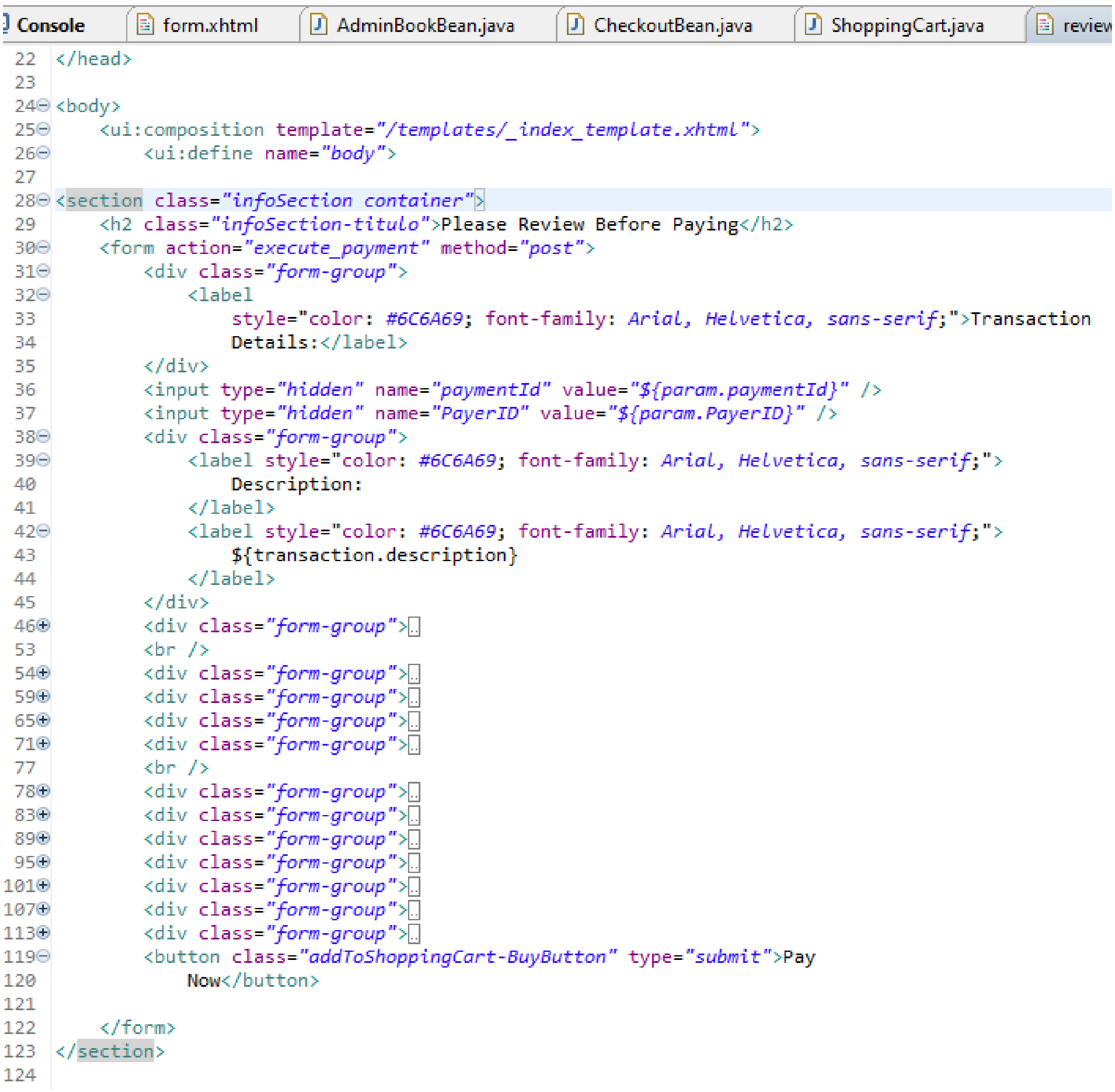
```java
package org.onlinebookstore.beans;

import java.io.Serializable;
@Named // Named allows .xhtml access this controller
@SessionScoped // SessionScoped to keep all the book inside the shopping cart
public class ShoppingCart implements Serializable {// Serializable to compress the object in a SessionScoped

    private static final long serialVersionUID = 2739826505942842818L;
    @Inject
    private PurchaseDAO purchaseDao;
    private Set<CartItem> items = new HashSet<>();
    private String uuid;

    public void add(CartItem item) {
        items.add(item);
    }

    public List<CartItem> getItems() {
        return new ArrayList<CartItem>(items);
    }

    public BigDecimal getTotal() {
        BigDecimal total = BigDecimal.ZERO;
        for (CartItem cartItem : items) {
            total = total.add(cartItem.getBook().getPrice().multiply(new BigDecimal(cartItem.getQuantity())));
        }
        return total;
    }

    public BigDecimal getTotal(CartItem item) {
        return item.getBook().getPrice().multiply(new BigDecimal(item.getQuantity()));
    }

    public void remove(CartItem item) {
        this.items.remove(item);
    }

    public Integer getTotalQuantity() {
        return items.stream().mapToInt(item -> item.getQuantity()).sum();
    }

    public String getUuid() {
        return uuid;
    }

    public void setUuid(String uuid) {
        this.uuid = uuid;
    }
```

Figure 26 - ShoppingCart

Payment Gateway – Paypal

Paypal is one of the most popular payment gateways. The Paypal SDK (Software Dev Kit) was used to implement all the custom integration within the application. In this way, the customer has the feeling of security because he/her will still be in the Online Book Store application. Several bank cards can be used by the customer to make the payment.

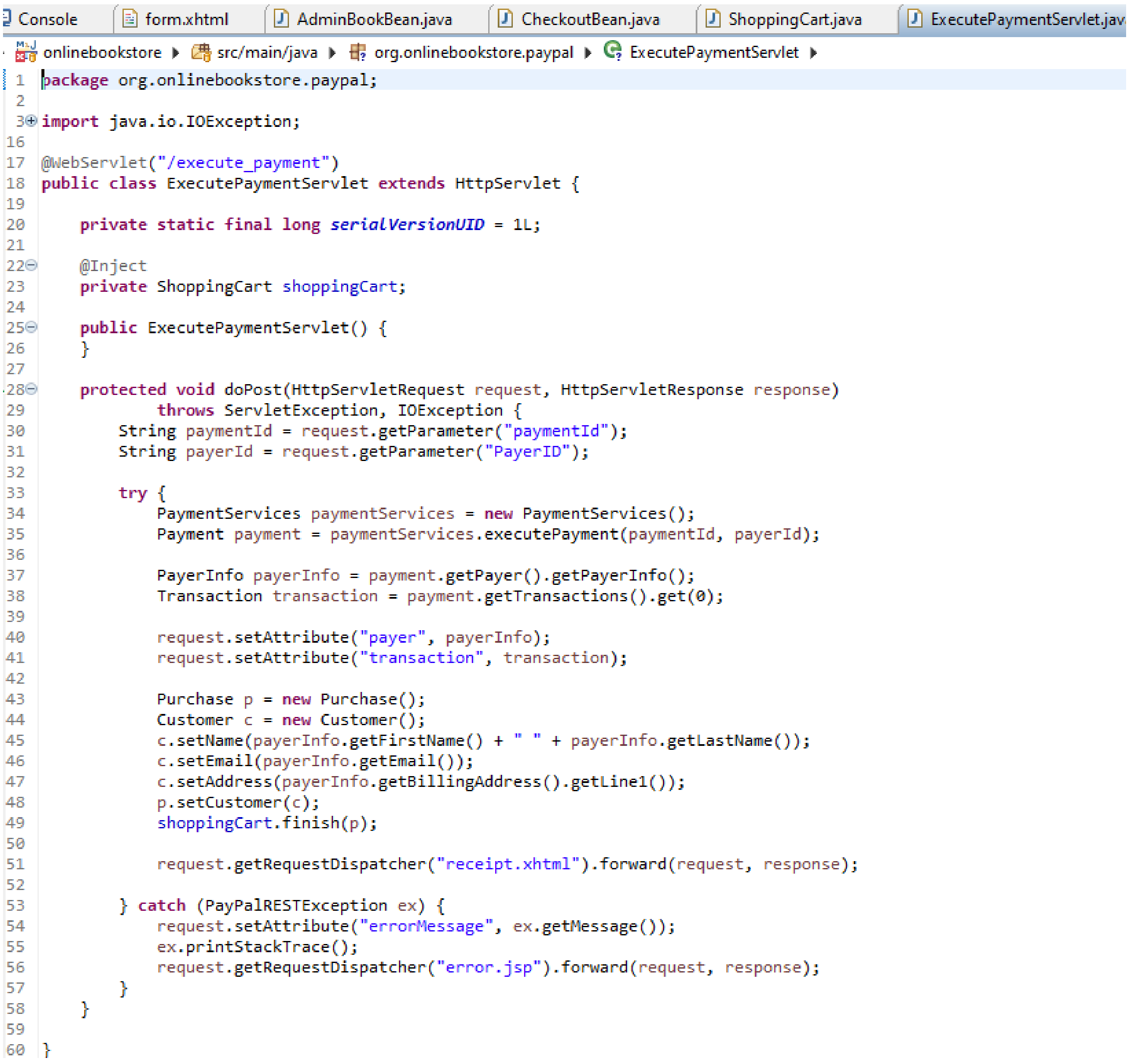
1. This is the payment confirmation page (Figure 27). The WebServlet execute_payment is executed when the "Pay Now" button is clicked.



```
</head>

<body>
    <ui:composition template="/templates/_index_template.xhtml">
        <ui:define name="body">

<section class="infoSection container">
    <h2 class="infoSection-titulo">Please Review Before Paying</h2>
    <form action="execute_payment" method="post">
        <div class="form-group">
            <label
                style="color: #6C6A69; font-family: Arial, Helvetica, sans-serif;">Transaction
                Details:</label>
        </div>
        <input type="hidden" name="paymentId" value="${param.paymentId}" />
        <input type="hidden" name="PayerID" value="${param.PayerID}" />
        <div class="form-group">
            <label style="color: #6C6A69; font-family: Arial, Helvetica, sans-serif;">
                Description:
            </label>
            <label style="color: #6C6A69; font-family: Arial, Helvetica, sans-serif;">
                ${transaction.description}
            </label>
        </div>
        <div class="form-group">
        <br />
        <div class="form-group">
        <div class="form-group">
        <div class="form-group">
        <div class="form-group">
        <br />
        <div class="form-group">
        <div class="form-group">
        <div class="form-group">
        <div class="form-group">
        <div class="form-group">
        <div class="form-group">
        <div class="form-group">
        <button class="addToShoppingCart-BuyButton" type="submit">Pay
            Now</button>

    </form>
</section>

```

Figure 27 - review.xhtml

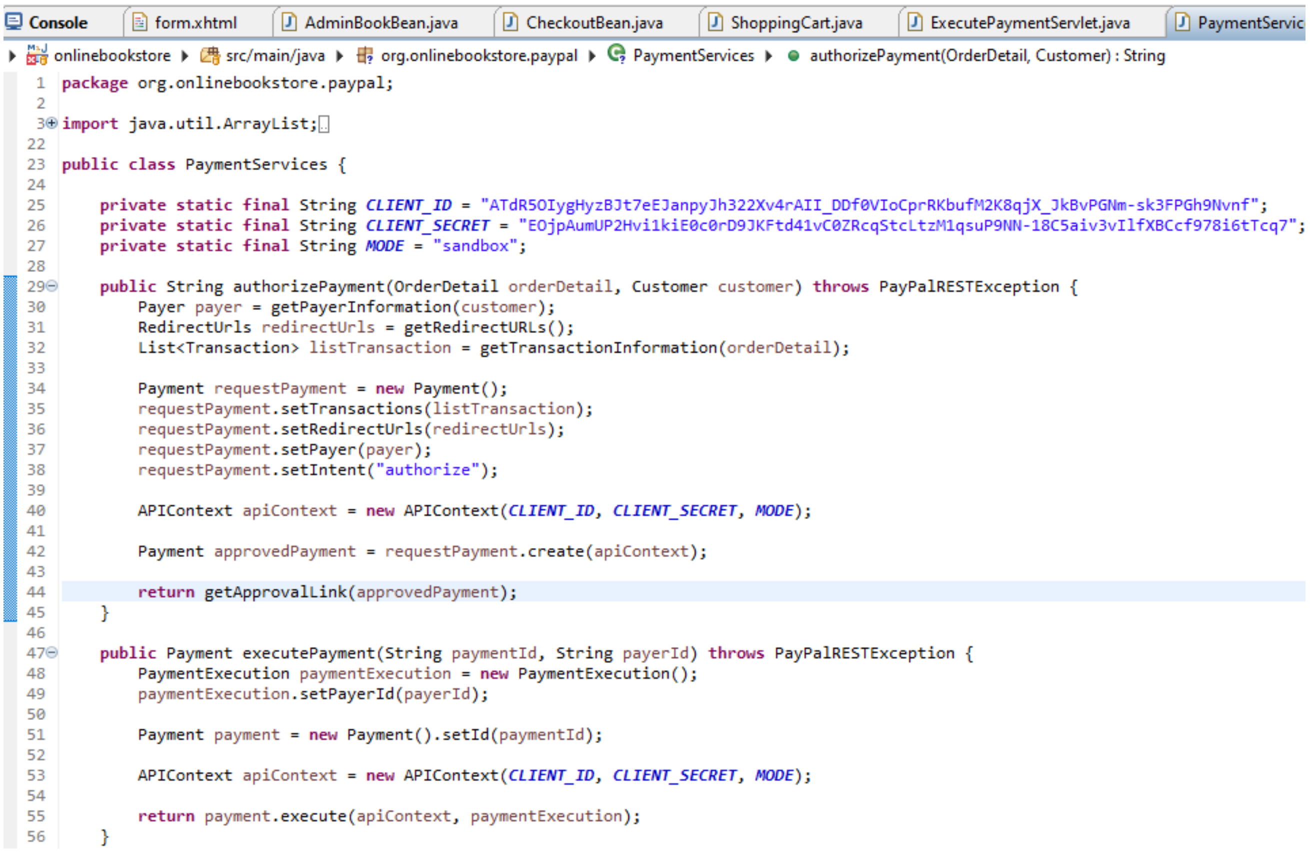
2. After confirmation of the data by the customer, the data is sent to Paypal and the payment is made(Figure 28).



```java
package org.onlinebookstore.paypal;

import java.io.IOException;

@WebServlet("/execute_payment")
public class ExecutePaymentServlet extends HttpServlet {

    private static final long serialVersionUID = 1L;

    @Inject
    private ShoppingCart shoppingCart;

    public ExecutePaymentServlet() {
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        String paymentId = request.getParameter("paymentId");
        String payerId = request.getParameter("PayerID");

        try {
            PaymentServices paymentServices = new PaymentServices();
            Payment payment = paymentServices.executePayment(paymentId, payerId);

            PayerInfo payerInfo = payment.getPayer().getPayerInfo();
            Transaction transaction = payment.getTransactions().get(0);

            request.setAttribute("payer", payerInfo);
            request.setAttribute("transaction", transaction);

            Purchase p = new Purchase();
            Customer c = new Customer();
            c.setName(payerInfo.getFirstName() + " " + payerInfo.getLastName());
            c.setEmail(payerInfo.getEmail());
            c.setAddress(payerInfo.getBillingAddress().getLine1());
            p.setCustomer(c);
            shoppingCart.finish(p);

            request.getRequestDispatcher("receipt.xhtml").forward(request, response);

        } catch (PayPalRESTException ex) {
            request.setAttribute("errorMessage", ex.getMessage());
            ex.printStackTrace();
            request.getRequestDispatcher("error.jsp").forward(request, response);
        }
    }

}
```

Figure 28 – ExecutePaymentServlet

3. For the execution of the payment, the data of the payee (the owner of the Online Book Store – Appendix A) is kept in the PaymentService class (Figure 29). Payments are being made in the Sandbox development environment provided by Paypal. For the production environment, just change the values of the MODE variable from "sandbox" to "live" and update the variables CLIENT_ID and CLIENT_SECRET to real data from the real Paypal account.


Console
form.xhtml
AdminBookBean.java
CheckoutBean.java
ShoppingCart.java
ExecutePaymentServlet.java
PaymentServic
onlinebookstore ▸ src/main/java ▸ org.onlinebookstore.paypal ▸ PaymentServices ▸ authorizePayment(OrderDetail, Customer) : String

```java
package org.onlinebookstore.paypal;

import java.util.ArrayList;

public class PaymentServices {

    private static final String CLIENT_ID = "ATdR5OIygHyzBJt7eEJanpyJh322Xv4rAII_DDf0VIoCprRKbufM2K8qjX_JkBvPGNm-sk3FPGh9Nvnf";
    private static final String CLIENT_SECRET = "EOjpAumUP2Hvi1kiE0c0rD9JKFtd41vC0ZRcqStcLtzM1qsuP9NN-18C5aiv3vIlfXBCcf978i6tTcq7";
    private static final String MODE = "sandbox";

    public String authorizePayment(OrderDetail orderDetail, Customer customer) throws PayPalRESTException {
        Payer payer = getPayerInformation(customer);
        RedirectUrls redirectUrls = getRedirectURLs();
        List<Transaction> listTransaction = getTransactionInformation(orderDetail);

        Payment requestPayment = new Payment();
        requestPayment.setTransactions(listTransaction);
        requestPayment.setRedirectUrls(redirectUrls);
        requestPayment.setPayer(payer);
        requestPayment.setIntent("authorize");

        APIContext apiContext = new APIContext(CLIENT_ID, CLIENT_SECRET, MODE);

        Payment approvedPayment = requestPayment.create(apiContext);

        return getApprovalLink(approvedPayment);
    }

    public Payment executePayment(String paymentId, String payerId) throws PayPalRESTException {
        PaymentExecution paymentExecution = new PaymentExecution();
        paymentExecution.setPayerId(payerId);

        Payment payment = new Payment().setId(paymentId);

        APIContext apiContext = new APIContext(CLIENT_ID, CLIENT_SECRET, MODE);

        return payment.execute(apiContext, paymentExecution);
    }
```

Figure 29 - Settings for the Paypal payee

Paypal Diagram

The complete diagram of all WebServlets in the Paypal payment process (Figure 30):

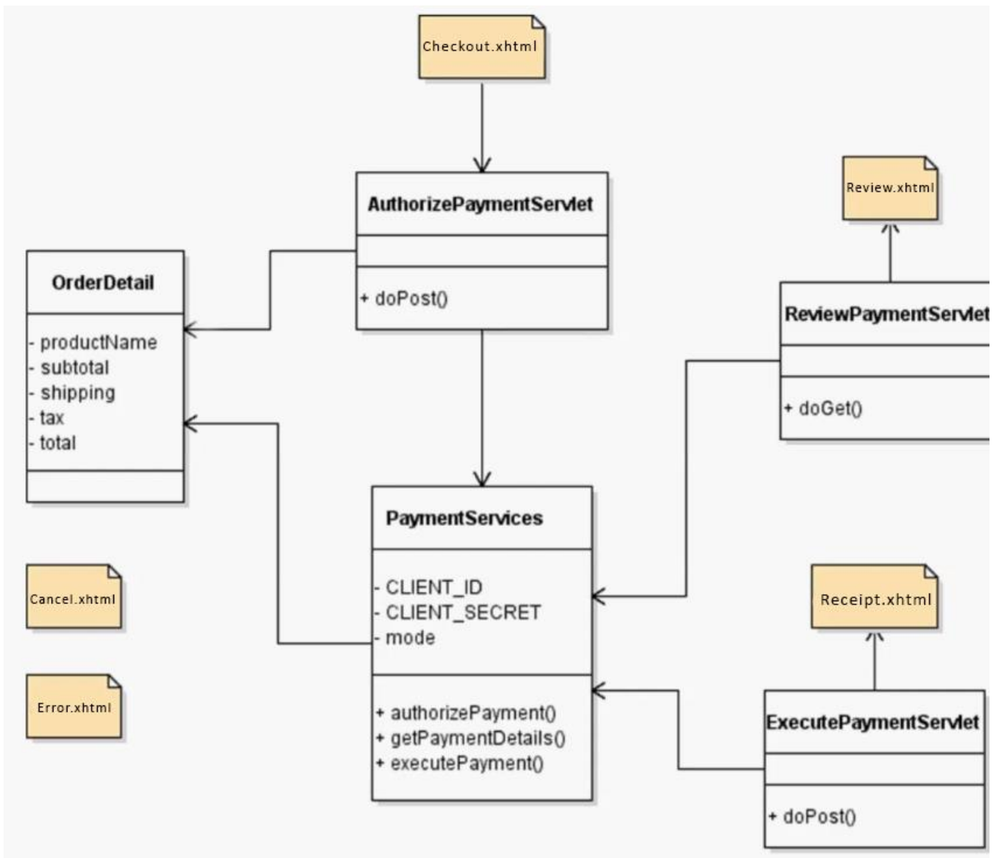


Figure 30 - Weservlets from Paypal SDK

Full flowchart of the steps for a Paypal payment (Figure 31):

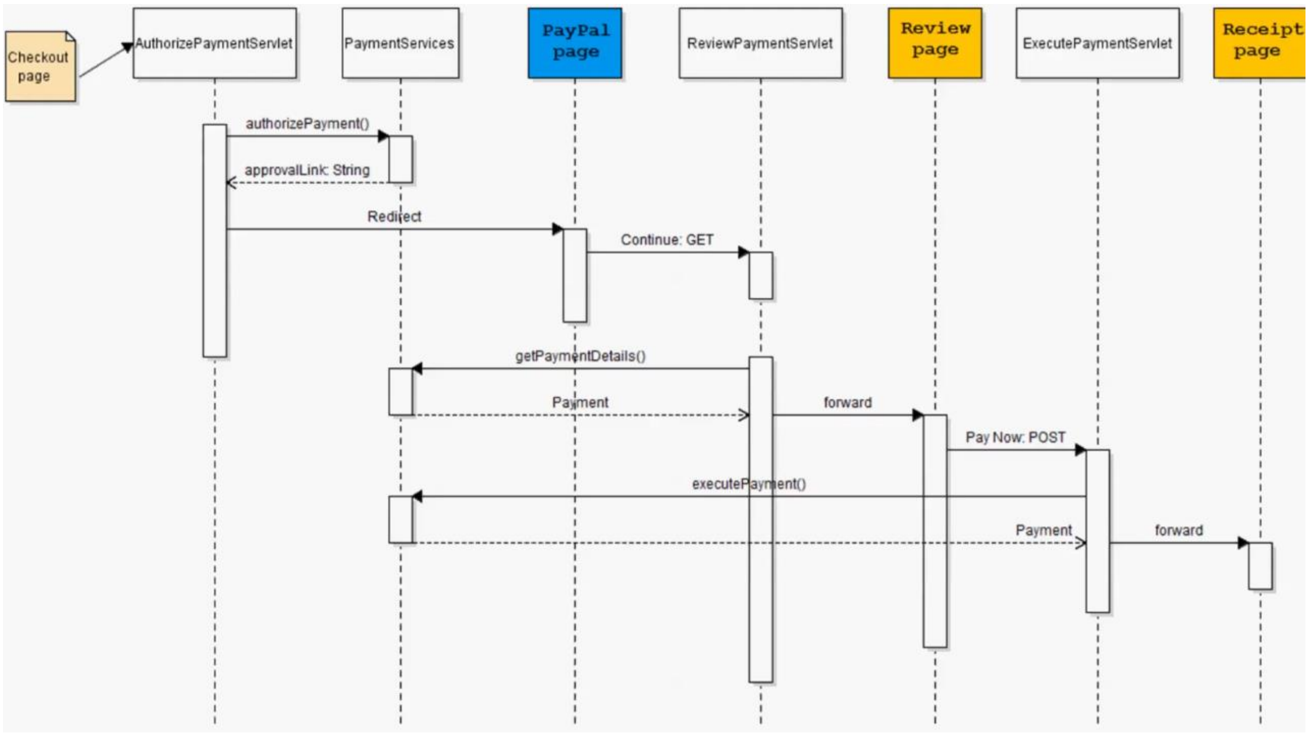


Figure 31 - Steps of the Paypal payment

CHAPTER V: TESTING AND EVALUATION

In this chapter, the final test and evaluation process of the Online Book Store will be presented, although the test and evaluation process took place throughout the entire development of the project. The chapter is divided into sections.

Access to the System - Admin

On the login screen, the Admin needs to enter a valid email and password. After clicking on the Login button, he is redirected to the Register Books screen. This function is working correctly.

Registering a New Book - Admin

After logging in, the admin can register a new book by filling in the fields with the expected values. If you fill in an incorrect value, warnings will appear indicating what is wrong. This function is working correctly.

Listing Books - Admin

Once logged in, the Admin can see all the books registered by clicking the List button. This function is working correctly.

Creating Promotion - Admin

Once logged in, the Admin can create promotions by writing a title, the discounted percentage and choosing the book that will have the promotion. All the customers on the website will see this promotion. The new book price is updated. This function is working correctly.

Logout - Admin

Once logged in, Admin can log out by clicking Logout button. This function is working correctly.

Search by Book - Customer

The customer can search in the search bar located in the footer. After filling in the field and clicking on the Search button, the book list page referring to the search will appear. This function is working correctly.

Top Menu - Customer

The customer can click on a topic on the top menu and a list of books on that subject will appear. This function is working correctly.

Detail Page - Customer

After clicking on any book, the customer is redirected to the book's detail page. This function is working correctly.

Shopping Cart - Customer

After clicking the Buy button on the book's detail page, the customer can increase the number of units for that book or remove the book. This function is working correctly.

Customer Form - Customer

After clicking on the Checkout button, the customer needs to fill in the name, email, address and click on Checkout. Email format validations are performed. This function is working correctly.

Paypal Pages - Customer

Features regarding Paypal have been tested and are all working correctly.

Review Page - Customer

On the purchase review page, the customer confirms the data displayed on the page. This function is working correctly.

Payment Done Page - Customer

After reviewing the purchase data and clicking on the Pay Now button, the payment is made and the Payment Done Page is displayed. The Send Receipt by Email button is displayed. If the customer clicks on Send Receipt by Email, the customer will receive an email containing the details of his/her purchase and the Home Page is displayed. This function is working correctly.

CAPTER VI: CONCLUSIONS AND FURTHER WORK

Conclusions

The main objective of the project was to develop an online shopping store and, according to the requirements, the Online Book Store project is working 100% as expected.

Despite being just a prototype, with some adjusts, the project can get closer to a final product that could be available for public access.

Further Work

There are some relevant improvements to add to the system:

- Create books categories.
- Authors registration page for Admin to use.
- Customer registration page.
- Exclusive page for customers.
- Storage of images in a cloud service.
- Add more images for each book (slide show).
- Show the Table of Contents for each book.
- Add more Payment Gateways.
- Create the Gift Card option.
- Create an Invoice template in the Paypal Business Account and provide it by Receipt by Email for customers purchases.
- Add advertising banners to make more money.

APPENDIX A: LINKS

Project available on GitHub

GitHub link: https://github.com/lucasc88/onlinebookstore

Admin Credentials

Link: http://localhost:8080/onlinebookstore/books/form.xhtml

Email: lucas@onlinebookstore.ie

Password: 123

Paypal SandBox Credit Card

Card Type: MasterCard

Card Number: 5232109159002185

Expiration Date: 10/2024

CVV: 996

Paypal SandBox Business Account

Link: https://www.sandbox.paypal.com/signin

Email: sb-wpjru8982299@business.example.com

Password: vv<6iU*!

APPENDIX B

SQL Commands to Initialize the Database

- insert into systemrole (name) values ('ADMIN');
- insert into SystemUser(email, password) values ('lucas@onlinebookstore.ie', 'pmWkWSBCL51Bfkhn79xPuKBKHz//H6B+mY6G9/eieuM=');
- insert into systemuser_systemrole values (1, 'ADMIN');
- insert into author(name) values (' Theodore OBrien'), ('Joshua Bloch'), ('Jon Duckett'), ('Elisabeth Robson'),('Eric Freeman'),('Eric Lengyel'),('Harrison Ferrone'),('Robert K. Wysocki'), ('Project Management Institute'),('Herbert Schildt'),('John P. Doran'),('Peter Spath'),('John Horton');

References

Amárach Research, S., 08/10/2020. *Average online monthly spend hits €245 - Research*. [Online]

Available at: https://www.virginmedia.ie/about-us/press/2020/average-online-monthly-spend-hits-245-euro/ [Accessed 19 01 2021].

Scholtz, B. and Arjan Tijms (2018). The Definitive Guide to JSF in Java EE 8. Berkeley, Ca Apress.

Keith, M., Merrick Schincariol and Massimo Nardone (2018). Pro JPA 2 in Java EE 8 : an in-depth guide to Java persistence API's. New York, Ny: Springer Science+Business Media.

Kodali, R.R., Wetherbee, J., Massimo Nardone and Chirag Rathod (2018). Beginning EJB in Java EE 8 : building applications with Enterprise JavaBeans. Berkeley, California: Apress.

Paypal SandBox [Online] Available at: https://developer.paypal.com/developer/accounts/[Accessed 17 01 2021].

JavaTPoint, 02/10/2020. *JMS Tutorial*. [Online] Available at: https://www.javatpoint.com/jms-tutorial [Accessed 15 01 2021].

Burke, B. (2014). RESTful Java with JAX-RS 2.0: [designing and developing distributed web services]. Sebastopol, Ca: O'reilly.

Baeldung. 02/09/2019. *A Guide to the Java API for WebSocketTutorial* [Online] Available at: https://www.baeldung.com/java-websockets [Accessed 15 01 2021].

Baeldung. 22/05/2021. *Guide To The Java Authentication And Authorization Service (JAAS)* [Online] Available at: https://www.baeldung.com/java-websockets [Accessed 15 01 2021].