CSE 593 Applied Project

Student: Hailun Yan

Advisor: Professor Hessam Sarjoughian

Spring 2009

A Spring-based Lightweight Online Shopping Application

Table of Content

Abstract 3
1 Introduction 4
2 A Lightweight Noninvasive Solution 6
    2.1 Hibernate 6
    2.2 Spring 7
    2.3 SpringMVC 15
    2.4 Maven 15
3 Architectural Overview 16
3 Project Deployment and Demo Screenshots 17
Conclusions 28
References 29

Table of Figure

Figure 1 Architectural blueprint of the Java EE application [9] 4
Figure 2 High level overview of the Spring framework [11] 8
Figure 3 Crosscutting concerns tangled each other [2] 10
Figure 4 Seperation of concern with AOP [2] 11
Figure 5 The concept of calling a method on a transactional Spring AOP proxy [11] 12
Figure 6 High level architecture of the online bookstore web application 16
Figure 7 Online Shpping Application Main Page Screenshot 18
Figure 8 Online Shpping Application Product List Page Screenshot 19
Figure 9 Online Shpping Application Item List Page Screenshot 19
Figure 10 Online Shpping Application Shopping Cart Page Screenshot 20
Figure 11 Online Shpping Application Checkout Summary Page Screenshot 21
Figure 12 Online Shpping Application Login Page Screenshot 22
Figure 13 Online Shpping Application Payment Detail Page Screenshot 23
Figure 14 Online Shpping Application Order Summary Page Screenshot 24
Figure 15 Online Shpping Application Order Confirmation 1 Page Screenshot 25
Figure 16 Online Shpping Application Order Confirmation 2 Page Screenshot 26
Figure 17 Online Shpping Application User Registration Page Screenshot 27
Figure 18 Online Shpping Application User Registration Validation Screenshot 28

Table of Code

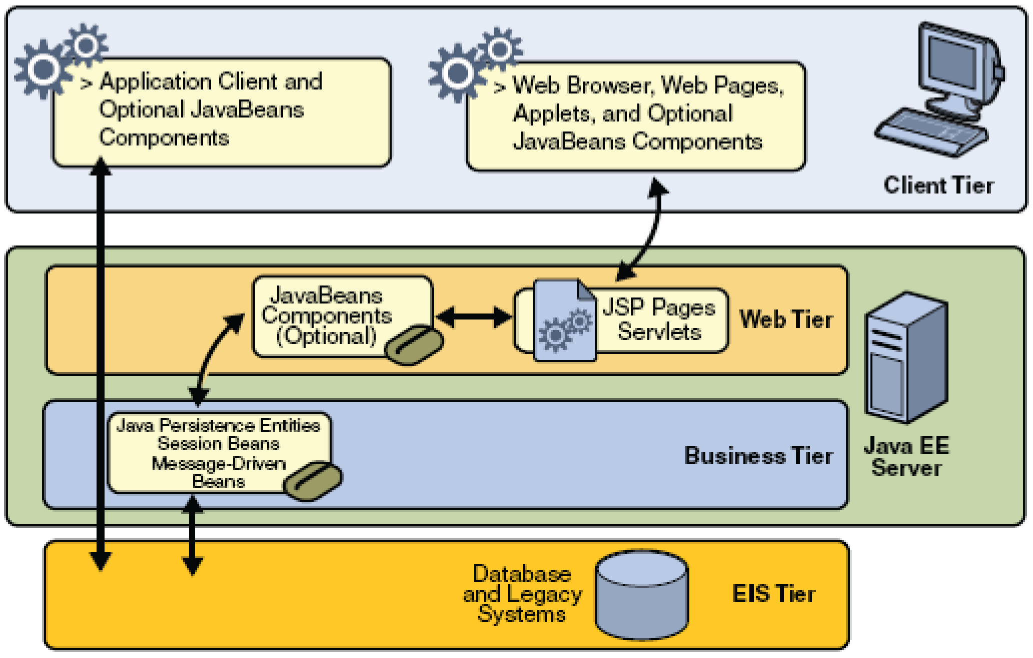
Code 1 checkoutDao Spring Bean Configuration 9
Code 2 Retrieving checkoutDao Spring Bean at Client side 9
Code 3 CheckourService interface definition and its implementation CheckoutServiceImpl 13
Code 4 checkoutService Spring Bean declaration and its transanction management demarcation 14

Abstract

Java Platform, Enterprise Edition (Java EE) is an application platform provided by Sun Microsystems for developing component-based, distributed enterprise applications. Enterprise JavaBeans (EJB) technology is used to implement Java EE components. EJB is highly coupled with application server and hence is not vender independent. In this project, we are going to use some of the well known non-invasive open source frameworks, Spring, Hibernate and SpringMVC to develop an online shopping application. The goal of this project is to demonstrate how these frameworks are capable of working together to make rapid enterprise Java application development easier. With the help of these frameworks, system level crosscutting concerns are handled by frameworks. This makes application developers able to focus on writing the reusable Plain Old Java Objects (POJO) to implement application-specific business logic.

1 Introduction

Java Platform, Enterprise Edition (Java EE) is an application platform provided by Sun Microsystems for developing component-based, distributed enterprise applications. The Java EE platform uses a distributed multi-tiered application model for enterprise applications. Application logic is divided into components according to the kind of functions they provide. The various application components that make up a Java EE application are installed on different machines depending on the tier in the Java EE environment to which the application component belongs. Figure 1 shows the architectural blueprint of the Java EE application. Enterprise JavaBeans (EJB) technology is used to implement Java EE components. Enterprise beans run in the EJB container, a runtime environment within the Application Server. The EJB container provides system-level services such as transactions and security to its enterprise beans. These services enable building and deploying enterprise beans, which form the core of transactional Java EE applications. The Client tier could either be a thin client such as web browser or fat client such as Swing GUI. The EIS tier is the enterprise database and legacy mainframe systems.



> Application Client and Optional JavaBeans Components
> Web Browser, Web Pages, Applets, and Optional JavaBeans Components
Client Tier
JavaBeans Components (Optional)
JSP Pages Servlets
Web Tier
Java Persistence Entities Session Beans Message-Driven Beans
Business Tier
Java EE Server
Database and Legacy Systems
EIS Tier

Figure 1 Architectural blueprint of the Java EE application [9]

Before the availability of EJB, Java developers were responsible for writing the transaction management, authorization, and persistence code themselves. In addition to being error-prone and time-consuming to conceptualize and write, this code was often intertwined with the business logic. EJB partially succeeded in separating cross-cutting concerns from the business logic. Responsibility for handling those concerns moved from the business logic components to the EJB container. However, the first two versions of the framework—EJB 1.0 and EJB 2.0—did this in a fundamentally flawed way [3]. The major flaw of the early specifications of EJB is that they place severe demands on the implementions of the components. EJB 1.0 and 2.0 components must implement interfaces defined by the EJB framework and must often call the EJB framework APIs. This tightly couples the components to the EJB framework, causing the following problems:

- Even though crosscutting concerns are separate from the code and configured in XML configuration files, you cannot ignore them when developing business logic. For example, a persistent EJB component cannot be easily tested without connecting to the database. EJB prevents you from working on one concern at a time.
- Deploying EJB components in the EJB container is a time-consuming operation that often adds complexity of Java EE development. Large application server vendors have traditionally advocated development tools as a way to hide Java EE’s complexity. However, tools for managing Java EE artifacts are themselves complex, as is the code they generate.
- Business logics implemented in EJB technology are not portable between framework versions. There were significant and incompatible changes between EJB 1.0 and EJB 2.0, and between EJB 2.0 and EJB 3.0. To take full advantage of the new and improved features of each release of the specification, you must rewrite your components. This can be quite challenging if you are responsible for maintaining a Java EE application with a lifetime of more than a couple of years.

These problems motivated the enterprise Java community to find better ways of untangling crosscutting concerns. Most large Java EE projects have traditionally used in-house frameworks to hide the platform’s complexity. Many high-quality open source frameworks are now available that offer outstanding documentation and the support of a focused development team, without imposing licensing fees. There is now a clear trend for frameworks to standardize more of the infrastructure that formerly was developed on a per-project basis.

In this project, we are going to use some of the well known open source frameworks, **Spring**, **Hibernate** and **SpringMVC** to develop an online shopping application. The goal of this project is to demonstrate how these frameworks are capable of working together to make rapid enterprise Java application development easier. With the help of these frameworks, system level crosscutting concerns are handled by frameworks. This makes application developers able to focus on writing the reusable Plain Old Java Objects (POJO) to implement application-specific business logic.

2 A Lightweight Noninvasive Solution

Experience shows that developers don't like frameworks that impose excessive constraints on their code. Three novel capabilities of emerging Java EE frameworks that can help developers achieve the goal of a POJO-centric application are transparent persistence, Inversion of Control (IoC), and Aspect Oriented Programming (AOP).

2.1 Hibernate

The object-relational impedance mismatch is a set of conceptual and technical difficulties which are often encountered when a Relational Database Management (RDBM)) system is being used by a program written in an object-oriented programming language or style, particularly when objects and/or class definitions are mapped in a straightforward way to database tables and/or relational schema. The following are some of the major mismatches: [12]

- **Data type differences:** The relational model strictly prohibits by-reference attributes, whereas OO languages embrace and expect by-reference behavior. Scalar types and their operator semantics are also very often subtly to vastly different between the models, causing problems in mapping. A more subtle, but related example is that SQL systems often ignore trailing white space in a string for the purposes of comparison, whereas OO string libraries do not.
- **Structural and integrity differences:** In OO languages, data structures are heavily nested and are difficult to map to relational schemas, where all data is represented in a named set of global, unnested relation variables. The relational model calls for declarative constraints on scalar types, attributes, relation variables, and the database as a whole. Constraints in OO languages are generally not declared as such, but are manifested as exception raising protection logic surrounding encapsulated internal data.
- **Manipulative differences:** The relational model has a relatively small and well defined set of primitive operators and SQL language for usage in the query and manipulation of data, whereas OO languages generally handle query and manipulation through essential OOP concepts such as inheritance, polymorphism, and association.
- **Transactional differences:** Relational database transactions, as the smallest unit of work performed by databases, are much larger than any operations performed by classes in OO languages. Transactions in relational databases are dynamically bounded sets of arbitrary data manipulations, whereas the granularity of transactions in OO languages is typically individual assignments of primitive typed fields.

Java EE provided two means for accessing persistent data: Java Database Connectivity (JDBC), the Java standard API for relational database management system access; and entity beans, an EJB component type dedicated to modeling a persistent entity. JDBC's error-prone programming model inhibited object-oriented design by forcing developers to work with relational concepts in Java code. Entity beans, despite being advocated by sun and major J2EE vendors, likewise proved to be cumbersome.

Object-relational mapping (ORM) is a programming technique for converting data between incompatible type systems in relational databases and object-oriented programming languages. This creates a virtual object database, which can be used from within the programming language. ORM tool, as a noninvasive framework, provides transparent persistence for pure business objects, known as POJO.

Hibernate is a powerful, high performance open source ORM framework. It lets you develop persistent classes based on object-oriented techniques including association, inheritance, polymorphism, composition, and collections.

2.2 Spring

Spring is a light-weight open source layered Java/JavaEE application platform. The core goal of Spring framework is to make Java/JEE application development easy. As a noninvasive framework, Spring essentially combines IoC and AOP with a service abstraction, to provide a programming model in which application code is implemented in POJOs that are largely decoupled from the Java EE environment and thus reusable in various environments. Figure 2 is a high level overview of the Spring framework.

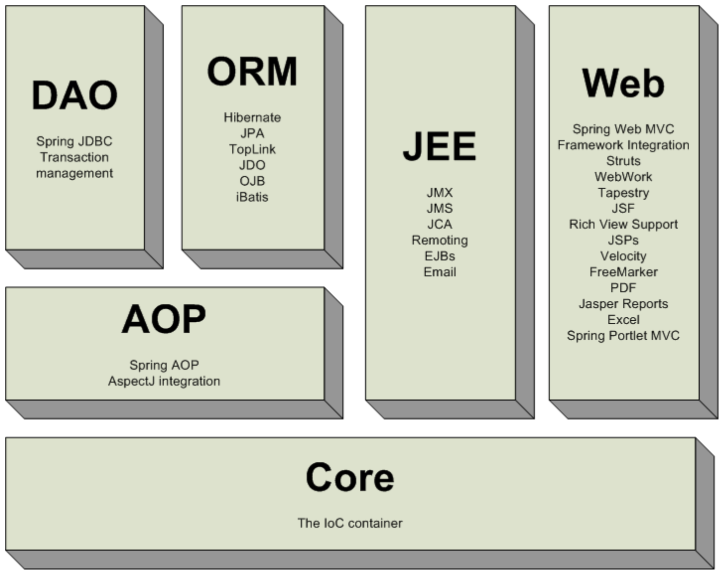


Figure 2 High level overview of the Spring framework [11]

In Spring, a POJO model can be applied to business services through IoC containers. These let business objects be configured at runtime, and enjoy declarative services such as automatic transaction management. IoC is a widely used term that in this case refers to a model in which the framework instantiates application objects and configures them for use. Spring Dependency Injection (DI) is a pure Java type of IoC that does not depend on framework APIs and thus can be applied to objects that aren't aware of the framework. Configuration is via JavaBean properties (setter injection) or constructor arguments (constructor injection). This means that application code doesn't implement any framework interfaces; the framework uses reflection to configure it. The framework injects dependencies such as collaborating objects or configuration parameters, without application classes needing to perform explicit lookup. For example, comparing in the traditional JNDI-based approach to Java EE configuration, DI is a simple but powerful concept. Because the framework is responsible for resolving dependencies on collaborating objects, it can introduce a range of value-adds such as indirection to support hot swapping and codeless generation of proxies that represent remote services.

The following code fragement is from the project source code dataAccessContext-local.xml file. It shows how easy it is to configure the CheckoutDao with Spring DI. Without IoC, we have to manually write code to instantiate a LocalSessionFactoryBean and a BasicDataSource object; assign values to each and every property of these classes respectively; assign the dataSource object to the sessionFactory object's dataSource property. Then, instantiate a CheckoutDaoImpl object, and assign the previously created sessionFactory object to the dataSource property of the newly created CheckoutDaoImpl. This mean whenever we want to change a property of a bean, we have to change the source code, recompile it, and redeploy it to the web server. With Spring IoC, instantiation of an object simply requires configuration and DI. At runtime, Spring will create all these objects, and inject the dataSource object into the sessionFactory, then inject the sessionFactory object into the CheckoutDaoImpl object.

```
<bean id="checkoutDao" class="com.abc.dao.CheckoutDaoImpl">
    <property name="sessionFactory" ref="sessionFactory"/>
</bean>
```

```
<bean id="sessionFactory"
    class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
    <property name="dataSource" ref="dataSource"/>
    <property name="mappingResources">
        <list>
            <value>conf/cse593Project.hbm.xml</value>
        </list>
    </property>
    <property name="hibernateProperties">
        <props>
            <prop key="hibernate.dialect"
                value="net.sf.hibernate.dialect.DB2Dialect"/>
        </props>
    </property>
</bean>
```

```
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource"
        destroy-method="close">
    <property name="driverClassName" value="com.ibm.db2.jcc.DB2Driver"/>
    <property name="url" value="jdbc:db2:BookStore"/>
    <property name="username" value="sa"/>
    <property name="password" value="1234"/>
</bean>
```

Code 1 checkoutDao Spring Bean Configuration

To get a reference of the CheckoutDaoImpl object, all we need is the following two lines of code:

```
ApplicationContext context = new ClassPathXmlApplicationContext(
       new String[] {"dataAccessContext-local.xml"});
CheckoutDaoImpl myDao = (CheckoutDaoImpl)context.getBean("checkoutDao");
```

Code 2 Retrieving checkoutDao Spring Bean at Client side

In the real Spring application development, quite offen, even the above two lines are not necessary. For example, in a web-based Java EE application, the Spring servlet will be triggered by the web server on server startup. Once it is up, the servelt will handle the user http request and the Spring container will instantiate relevant objects, inject the required properties into it and forward it to the servlet. As an application developer, all we need to do is to write POJO to address our specific business logic. Spring will weave these business objects together at load time.

In addition, while spring does not compete with good existing solutions, it does foster integration. For example, Java persistence solutions JDO, Toplink, and Hibernate are great ORM solutions. Spring doesn't need to develop another one, but it does provide first class integration support to all these solutions. The above Spring configuration example is actually a good example of how Spring integrates with Hibernate and Apache database connection pooling solution DBCP seamlessly.

DI goes a long way toward delivering a POJO application model but fails to address some important requirements, such as the ability to apply declarative transaction management, security, custom caching, auditing etc. to selected methods. Traditional solutions to this problem all have substantial disadvantages. Using boilerplate code, e.g, to start and commit or roll back a transaction, results in the same code being used in multiple methods. In addition, design patterns such as the Decorator end up with cut-and-paste code. And objects can only benefit special-purpose frameworks such as EJB, which provide a fixed set of services, by conforming to framework APIs and implicit contracts. The Spring Framework provides a proxy-based AOP solution that complements DI.

Figure 3 illustrates the complexity without the help of AOP. The business objects on the left are too intimately involved with the system services. Not only does each object know that it is being logged, secured, and involved in a transactional context, but also each object is responsible for performing those services for itself.

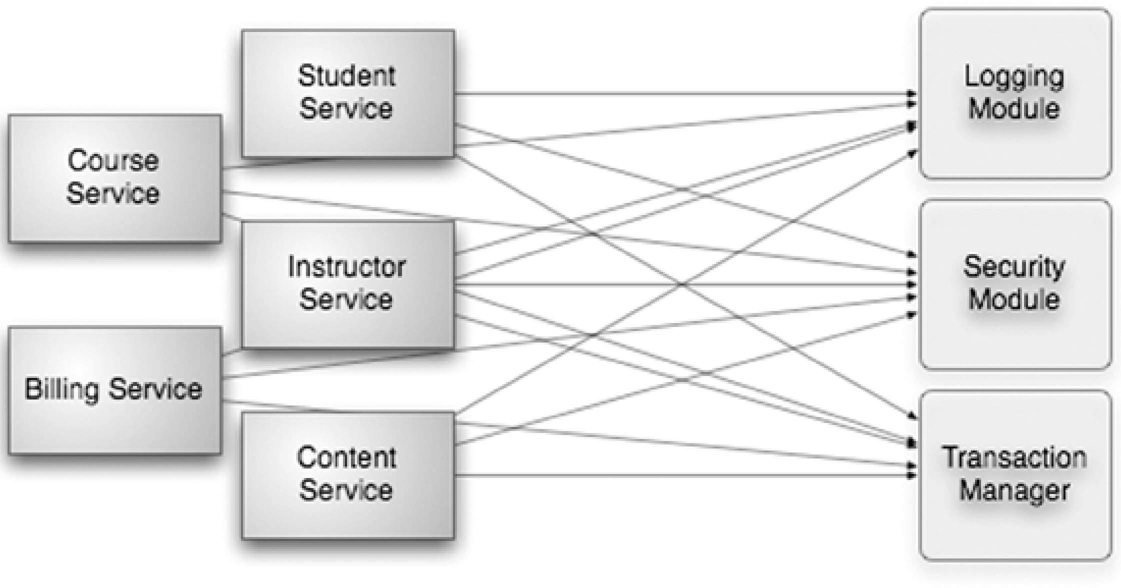


Figure 3 Crosscutting concerns tangled each other [2]

Aspect-oriented programming (AOP) [2] is a programming technique that promotes separation of concerns within a software system. AOP makes it possible to modularize these services and then apply them declaratively to the components that they should affect. This results in components that are more cohesive and that focus on their own specific concerns, completely ignorant of any system services that may be involved. In short, aspects ensure that POJOs remain plain. It may help to think of aspects as blankets that cover many components of an application, as illustrated in figure 4. At its core, an application consists of modules that implement the business functionality. With AOP, you can then cover your core application with layers of functionality. These layers can be applied declaratively throughout your application in a flexible manner without your core application even knowing they exist. This is a powerful concept, as it keeps the security, transaction, and logging concerns from littering the application's core business logic.

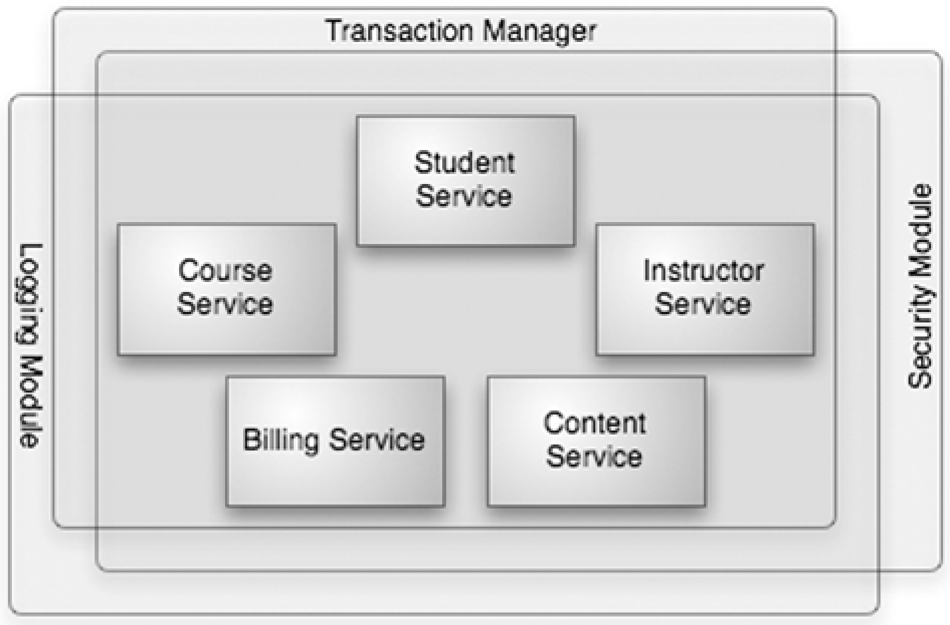


Figure 4 Seperation of concern with AOP [2]

Let's take the transaction management as an example to see how Spring declarative transaction management is made possible with Spring AOP. It may be helpful to begin by considering EJB Container Managed Transaction (CMT) and explaining the similarities and differences with the Spring Framework's declarative transaction management. The basic approach is similar: it is possible to specify transaction behavior down to individual method level. Some of the key differences are:

- Unlike EJB CMT, which is tied to JTA, the Spring Framework's declarative transaction management works in any environment. It can work with JDBC, JDO, Hibernate or other transactions under the covers, with configuration changes only.
- The Spring Framework enables declarative transaction management to be applied to any class, not

merely special classes such as EJBs.

The most important concepts to grasp with regard to the Spring Framework's declarative transaction support are that this support is enabled via AOP proxies, and that the transactional advice is driven by metadata i.e. currently XML or annotation-based. The combination of AOP with transactional metadata yields an AOP proxy that uses a `TransactionInterceptor` in conjunction with an appropriate `PlatformTransactionManager` implementation to drive transactions around method invocations. Figure 5 illustrates the concept of calling a method on a transactional Spring AOP proxy.

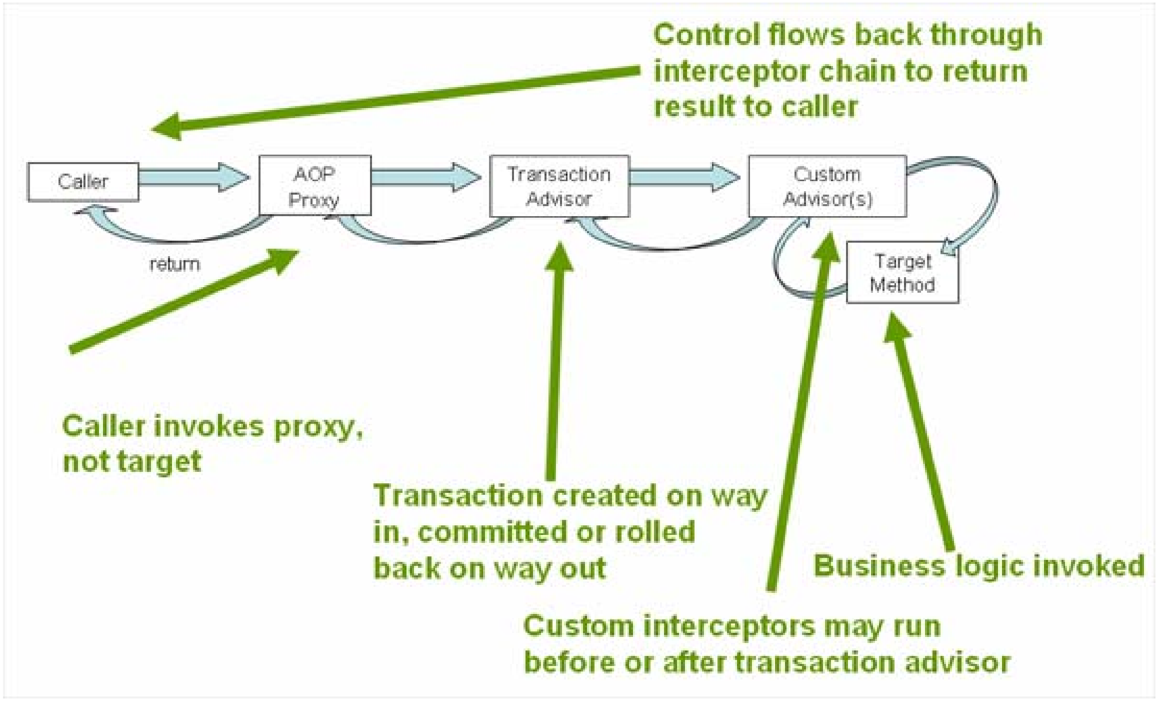

Control flows back through interceptor chain to return result to caller
Caller
AOP Proxy
Transaction Advisor
Custom Advisor(s)
Target Method
return
Caller invokes proxy, not target
Transaction created on way in, committed or rolled back on way out
Business logic invoked
Custom interceptors may run before or after transaction advisor

Figure 5 The concept of calling a method on a transactional Spring AOP proxy [11]

Consider the following interface and its implementation.

```java
package com.abc.service;

import com.abc.BO.Order;

public interface CheckoutService {
    Order findOrderById(long id);
    void addOrder(Order order);
    void updateOrder(Order order);
}
```

```java
package com.abc.service;

import com.abc.BO.Order;

public class CheckoutServiceImpl implements CheckoutService {
    public Order findOrderById(long id) {
        // TODO Auto-generated method stub
        return null;
    }

    public Order findOrderByName(String name) {
        // TODO Auto-generated method stub
        return null;
    }

    public void addOrder(Order order) {
        // TODO Auto-generated method stub
    }
}
```

Code 3 CheckourService interface definition and its implementation CheckoutServiceImpl

Let's assume that the first two methods of the `CheckoutService` interface (`findOrderById` and `findOrderByName`) have to execute in the context of a transaction with read-only semantics, and that the other methods have to execute in the context of a transaction with read-write semantics. The following metadata will configure the actual `CheckoutService` implementation with the right transactions for different methods declaratively.

```
<bean id="checkoutService" class="com.abc.service.CheckoutServiceImpl"/>
<tx:advice id="txAdvice" transaction-manager="txManager1">
   <tx:attributes>
   <tx:method name="find*" read-only="true"/>
   <tx:method name="*"/>
   </tx:attributes>
</tx:advice>
<aop:config>
   <aop:pointcut id="checkoutServicePC"
      expression="execution(* com.abc.service.CheckoutService.*(..))"/>
   <aop:advisor advice-ref="txAdvice" pointcut-ref="checkoutServicePC"/>
</aop:config>
<bean id="txManager1"
   class="org.springframework.orm.hibernate3.HibernateTransactionManager">
   <property name="sessionFactory" ref="sessionFactory"/>
</bean>
<bean id="txManager2"
   class="org.springframework.orm.toplink.TopLinkTransactionManager">
   <property name="sessionFactory" ref="sessionFactory"/>
</bean>
```

Code 4 checkoutService Spring Bean declaration and its transanction management demarcation

In the above metadata snippet, we declared a `checkoutService` service object that we want to make transactional. The transaction semantics that we want to apply are encapsulated in the <tx:advice/> definition. In this definition, we define all methods on starting with “find” are to execute in the context of a read-only transaction, and all other methods are to execute with the default transaction semantics. The “transaction-manager” attribute of the <tx:advice/> tag is set to the name of an implementation of `PlatformTransactionManager` interface, in this case, `HibernateTransactionManager`, that is going to actually drive the transactions. If later on, we switch to Oracle Data persistence solution TopLink, all we need to do is to define a TopLink transaction manager and change the “transaction-manager” attribute of the <tx:advice/> tag to that of TopLink, in this case txManager2. The <aop:config/> definition ensures that the transactional advice defined by the “txAdvice” bean actually executes at the appropriate points in the program. The expression defined within the <aop:pointcut/> element ensures the transactional advice runs for any method defined by the `CheckoutService` service of `com.abc.service` package. Behind the scence, the above configuration is going to affect the creation of a Spring transactional proxy object around the service object. The proxy will be configured with the transactional advice, so that when an appropriate method is invoked on the proxy, a transaction may be started, committed, rolled back, marked as read-only, etc., depending on the transaction configuration associated with that method.

By decoupling a POJO model from Java EE APIs, which are hard to stub at test time, The Spring lightweight container greatly simplify unit testing. It’s possible to unit test in a plain JUnit environment, without any need to deploy code to an application server or to simulate an application server environment. Given the increased popularity of test-driven development, this has been a major factor in lightweight noninvasive frameworks’ popularity.

2.3 SpringMVC

SpringMVC is an elegant, extensible open source web framework for creating enterprise-ready Java web applications [11]. The framework is based on MVC design pattern [11], designed to streamline the full development cycle, from building, to deploying, to maintaining web applications over time.

The MVC architecture is a widely used architectural approach for interactive applications that distributes functionality among application objects so as to minimize the degree of coupling between the objects. To achieve this, it divides applications into three layers: model, view, and controller. Each layer handles specific tasks and has responsibilities to the other layers:

- The *model* represents business data, along with business logic or operations that govern access and modification of this business data. The model notifies views when it changes and lets the view query the model about its state. It also lets the controller access application functionality encapsulated by the model.
- The *view* renders the contents of a model. It gets data from the model and specifies how that data should be presented. It updates data presentation when the model changes. A view also forwards user input to a controller.
- The *controller* defines application behavior. It dispatches user requests and selects views for presentation. It interprets user inputs and maps them into actions to be performed by the model. In a web application, user inputs are HTTP GET and POST requests. A controller selects the next view to display based on the user interactions and the outcome of the model operations.

Similarly as what we illustrated in the CheckoutDao example, with the help of Spring IoC container, we can easily wire Struts Action classes together with the POJOs and service classes.

2.4 Maven

Maven is a software project management and comprehension tool [13]. Based on the concept of a project object model (POM) [13], Maven can manage a project's build, reporting and documentation from a central configuration file. Maven has a large amount of default settings which makes the project build process easy. Maven aims to gather current principles for best practices development, and make it easy to guide a project in that direction.

3 Architectural Overview

The course project is a MVC-based online bookstore web application written in Java language based on Spring, Hibernate and Struts frameworks. Figure 6 shows the high level architecture of the web application. This is a classic three-tier Java EE application. However, instead of using EJBs, we use reusable POJOs as business objects. We use open source Apache Tomcat web container to load the web application. Struts, as the application front controller, is used to manage the Presentation layer together with Sun JSP and Servlet technologies. Hibernate deals with ORM issues and works with the POJOs. Spring as the application layer framework wires Struts Action classes with service classes. Sring also works with Hibernate to provide declarative transaction management and persistence services. At the bottom of this architecture, open source database MySql is used to store the data.

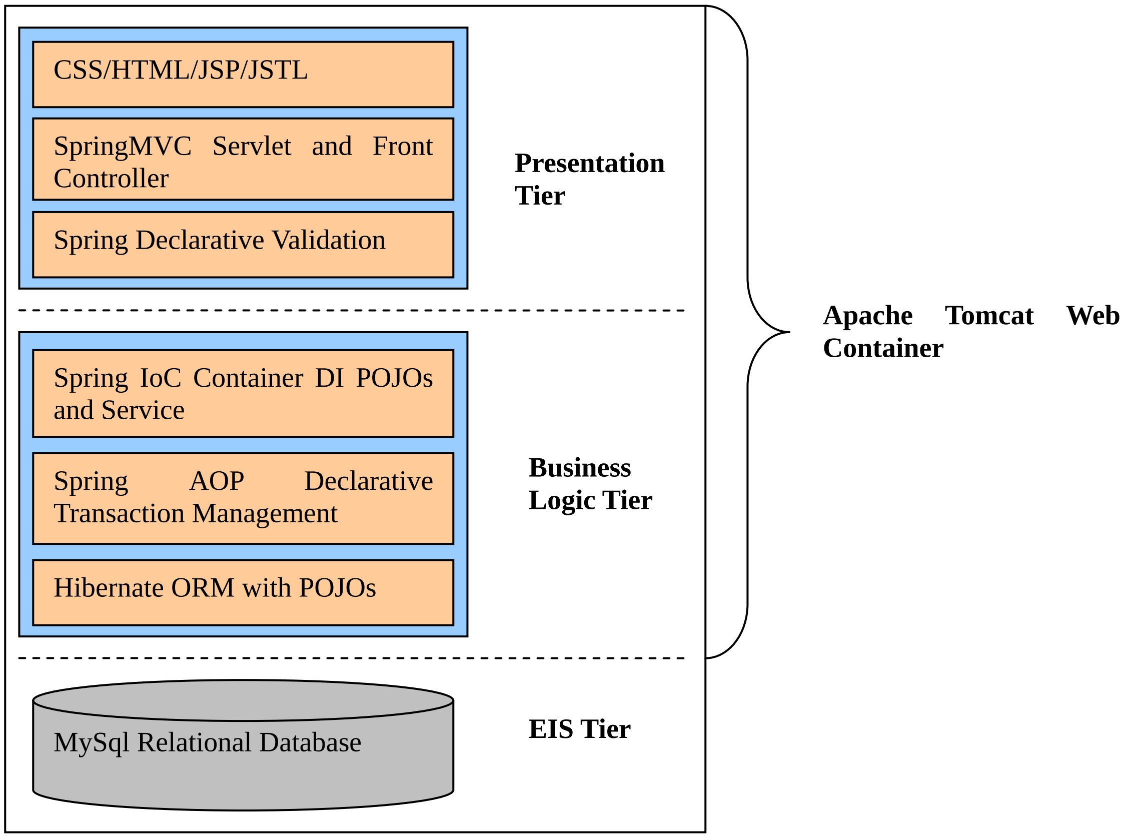

CSS/HTML/JSP/JSTL
SpringMVC Servlet and Front Controller
Spring Declarative Validation
Presentation Tier
Spring IoC Container DI POJOs and Service
Spring AOP Declarative Transaction Management
Hibernate ORM with POJOs
Business Logic Tier
Apache Tomcat Web Container
MySql Relational Database
EIS Tier

Figure 6 High level architecture of the online bookstore web application

Since this is an online shopping application, typical functions of an eCommerce application will be implemented in this application. For example, users are able to search a product from the product catalog, or search by keyword. Other core functions, such as user login, register, and shopping cart functions will also be implemented in this project.

3 Project Deployment and Demo Screenshots

The only prerequisites for building and deploying the project are to install Maven and MySql in your Operating System. After installed Maven and MySql, uncompress the ***mystore-springmvc.zip*** file. The mystore-springmvc folder will be created in the current directory. Then, go to the ***mystore-springmvc*** directory and type the following command in a console or terminal:

>mvn package

Maven will download all required jar libraries(only when these jars are not in your local maven repositories), compile all the source code and create a war file in the ***target*** directory. To deploy the web application to Tomcat web container, simply copy the war file to ***webapps*** directory of Tomcat root directory and start Tomcat.

Before you can start experiencing online shopping in this web store, you need to load some test data into MySql Database. First, create a mystore database. Then, run the mystore-springmvc\src\main\resources\db\mysql\jpetstore-mysql-schema.sql script to create tables used in this application. To load test data in the tables, run the mystore-springmvc\src\main\resources\db\mysql\ **jpetstore-mysql-dataload.sql** script in MySql.

Maven has a web container plugin called Jetty to let you test your web application without deploying it to a real web server or container. To start this jetty, simply type the following command in a console or terminal:

>mvn jetty:run

Then, open a browser and type the following address to start testing the online store application, as shown in Figure 7.

http://localhost:8080/mystore-springmvc

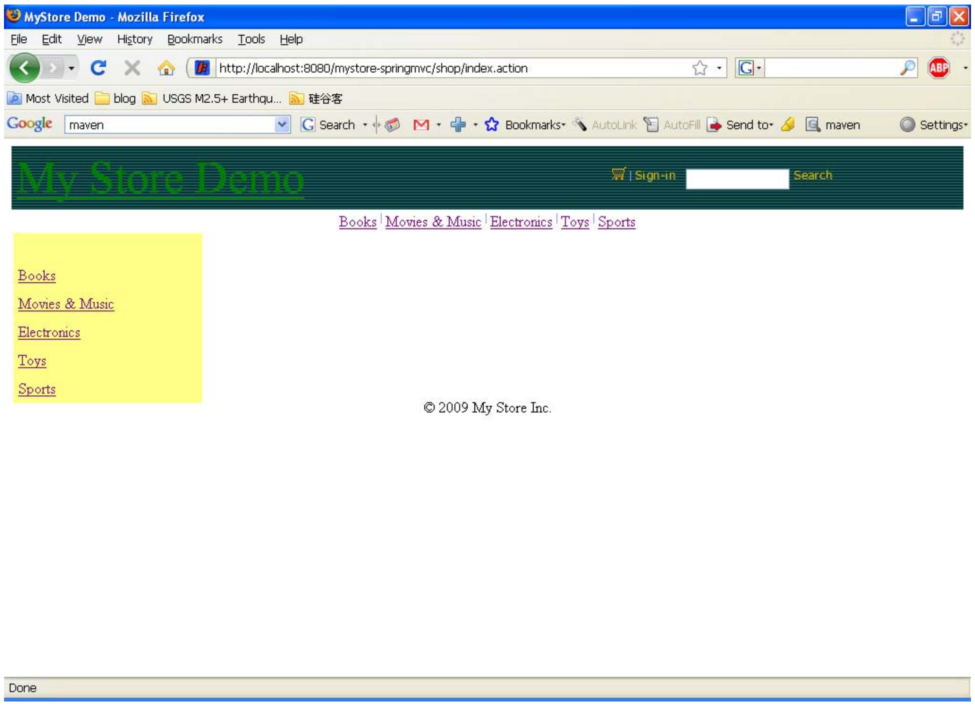
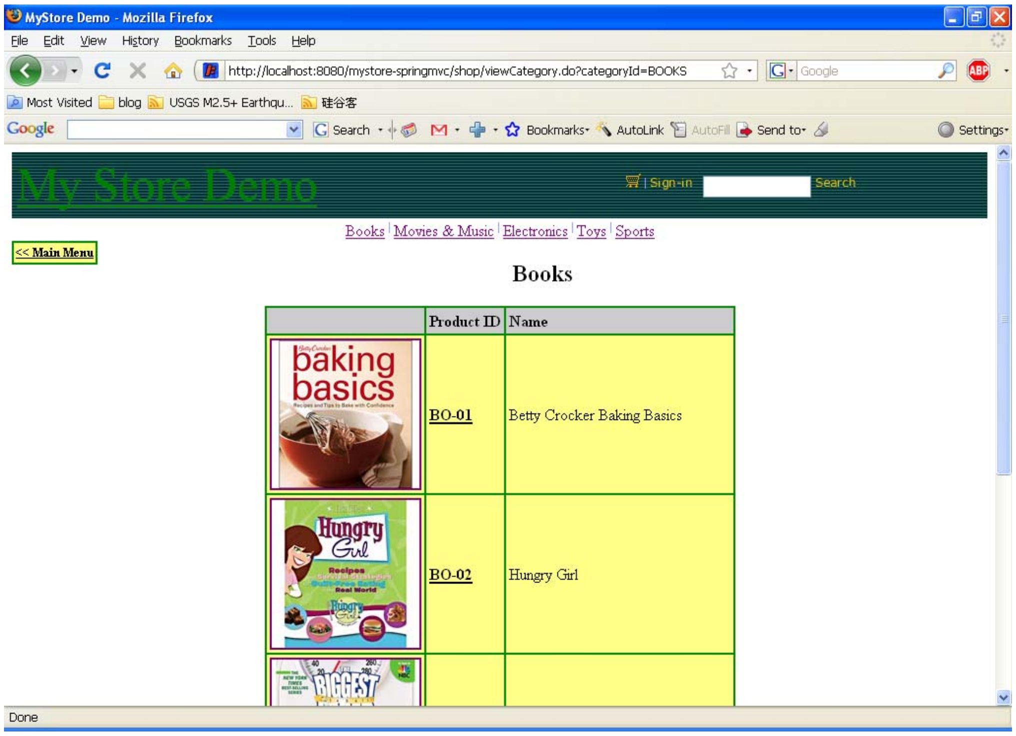

MyStore Demo - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/mystore-springmvc/shop/index.action
My Store Demo
| Sign-in
Search
Books | Movies & Music | Electronics | Toys | Sports
Books
Movies & Music
Electronics
Toys
Sports
© 2009 My Store Inc.
Done

Figure 7 Online Shpping Application Main Page Screenshot

Next, as shown in Figure 8, click one of the product categories, e.g. "Books" link to browse what products are available for sale.

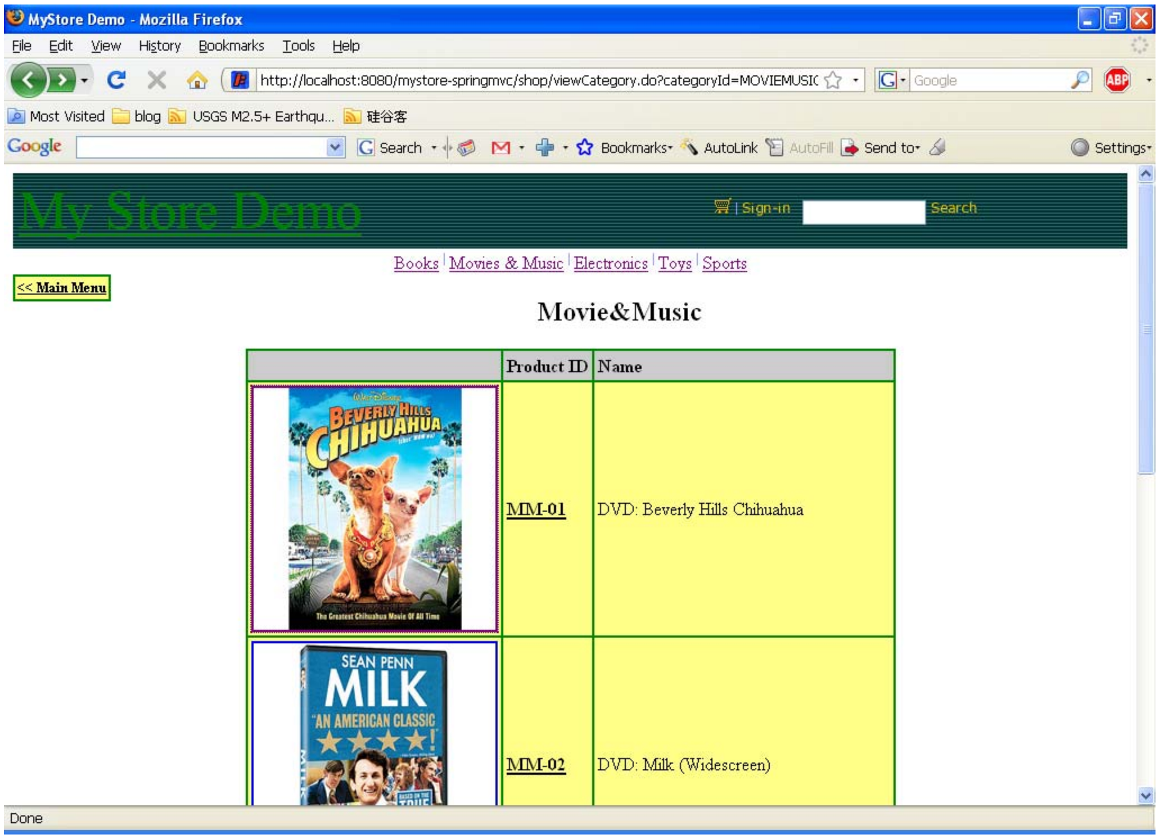

MyStore Demo - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/mystore-springmvc/shop/viewCategory.do?categoryId=BOOKS
My Store Demo
| Sign-in
Search
Books | Movies & Music | Electronics | Toys | Sports
<< Main Menu
Done

Books

	Product ID	Name
	BO-01	Betty Crocker Baking Basics
	BO-02	Hungry Girl

Figure 8 Online Shpping Application Product List Page Screenshot

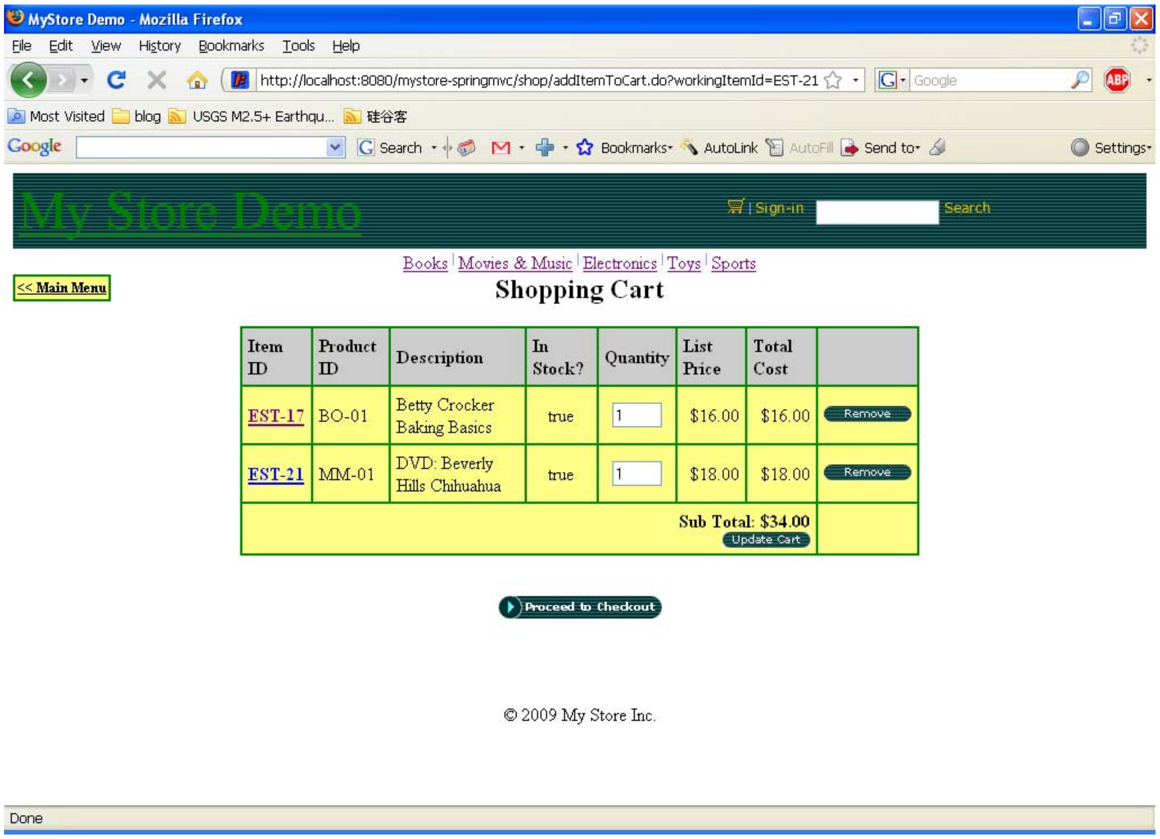
Next, as shown in Figure 9 and 10, click the picture of the book you want to buy to add it to your shopping cart. Then, browse another category, say Move & Music, and add your favoriate movies to your shopping cart.


MyStore Demo - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/mystore-springmvc/shop/viewCategory.do?categoryId=MOVIEMUSIC
My Store Demo
Sign-in
Search
Books | Movies & Music | Electronics | Toys | Sports
<< Main Menu
Done

Movie&Music

	Product ID	Name
	MM-01	DVD: Beverly Hills Chihuahua
	MM-02	DVD: Milk (Widescreen)

Figure 9 Online Shpping Application Item List Page Screenshot


MyStore Demo - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/mystore-springmvc/shop/addItemToCart.do?workingItemId=EST-21
My Store Demo
Sign-in
Search
Books | Movies & Music | Electronics | Toys | Sports
<< Main Menu
Remove
Update Cart
Proceed to Checkout
Done

Shopping Cart

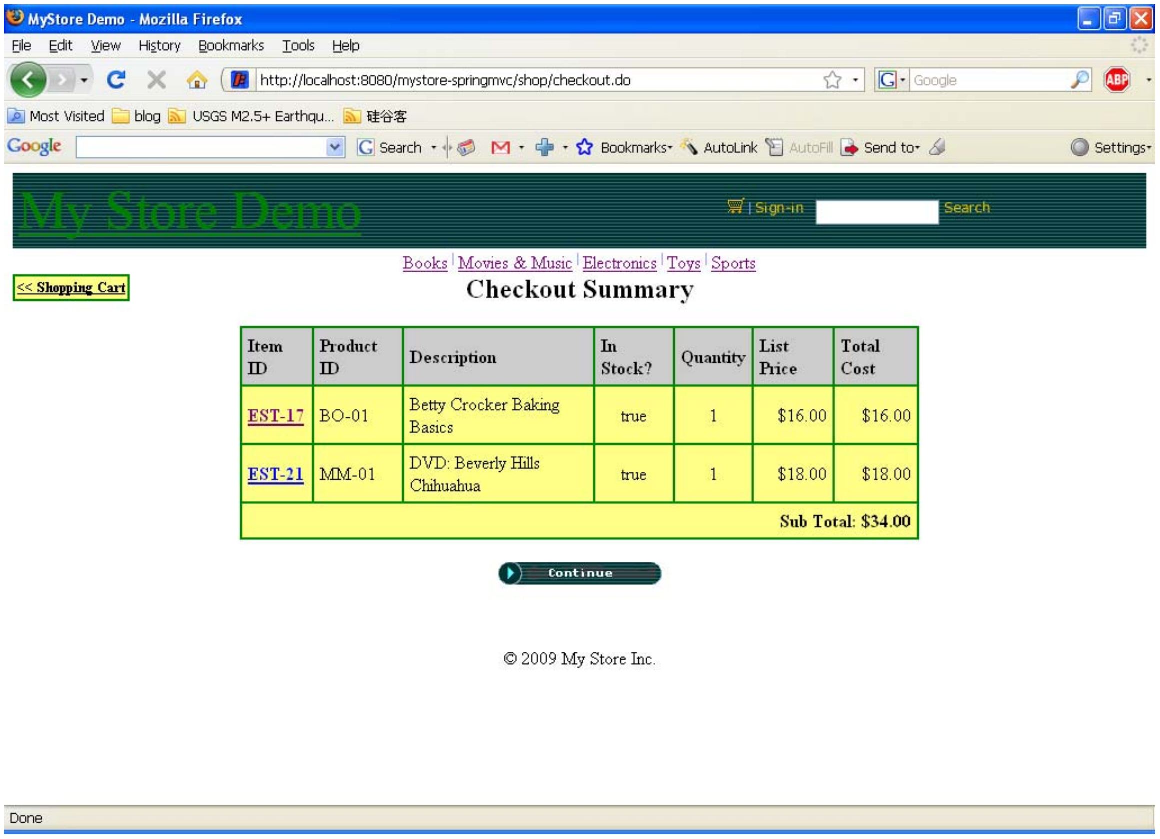
Item ID	Product ID	Description	In Stock?	Quantity	List Price	Total Cost	
EST-17	BO-01	Betty Crocker Baking Basics	true	1	$16.00	$16.00	Remove
EST-21	MM-01	DVD: Beverly Hills Chihuahua	true	1	$18.00	$18.00	Remove
						Sub Total: $34.00	

© 2009 My Store Inc.

Figure 10 Online Shpping Application Shopping Cart Page Screenshot

Now, you have two products in the cart. If you want to checkout, click the checkout button to continue. Or if you want to update the quantity, you can do that before checkout.

After clicking the Proceed to Checkout button, as shown in Figure 11, you have a chance to look at the checkout summary. If you agree, click continue button.


MyStore Demo - Mozilla Firefox
http://localhost:8080/mystore-springmvc/shop/checkout.do
My Store Demo
Sign-in
Search
Books | Movies & Music | Electronics | Toys | Sports
<< Shopping Cart
Continue
Done

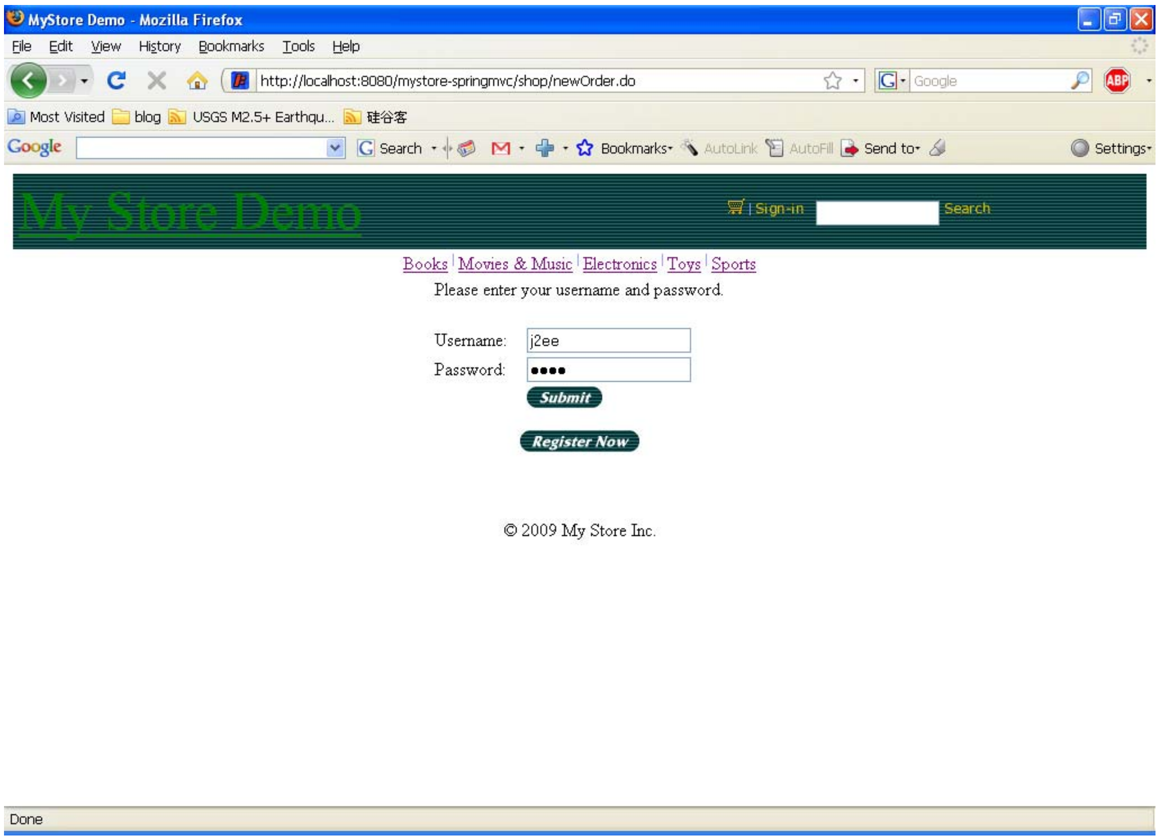
Checkout Summary

Item ID	Product ID	Description	In Stock?	Quantity	List Price	Total Cost
EST-17	BO-01	Betty Crocker Baking Basics	true	1	$16.00	$16.00
EST-21	MM-01	DVD: Beverly Hills Chihuahua	true	1	$18.00	$18.00
						Sub Total: $34.00

© 2009 My Store Inc.

Figure 11 Online Shpping Application Checkout Summary Page Screenshot

Now, fill in payment information and click summit button. You need to login in before you can finish payment and submit your order. To test, as shown in Figure 12, just use the default j2ee user name and j2ee password and click Submit button.

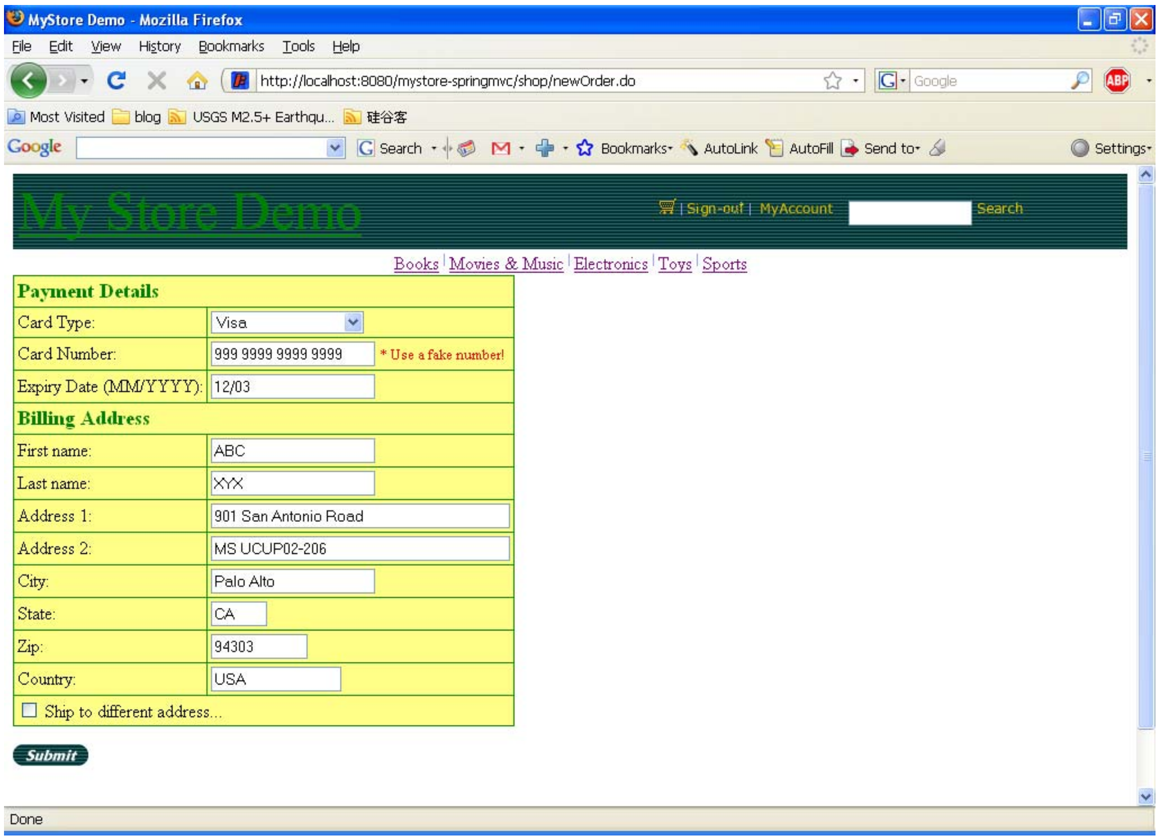

MyStore Demo - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/mystore-springmvc/shop/newOrder.do
My Store Demo
Sign-in
Search
Books | Movies & Music | Electronics | Toys | Sports
Submit
Register Now
© 2009 My Store Inc.
Done

Please enter your username and password.

Username: j2ee
Password: ●●●●

Figure 12 Online Shpping Application Login Page Screenshot

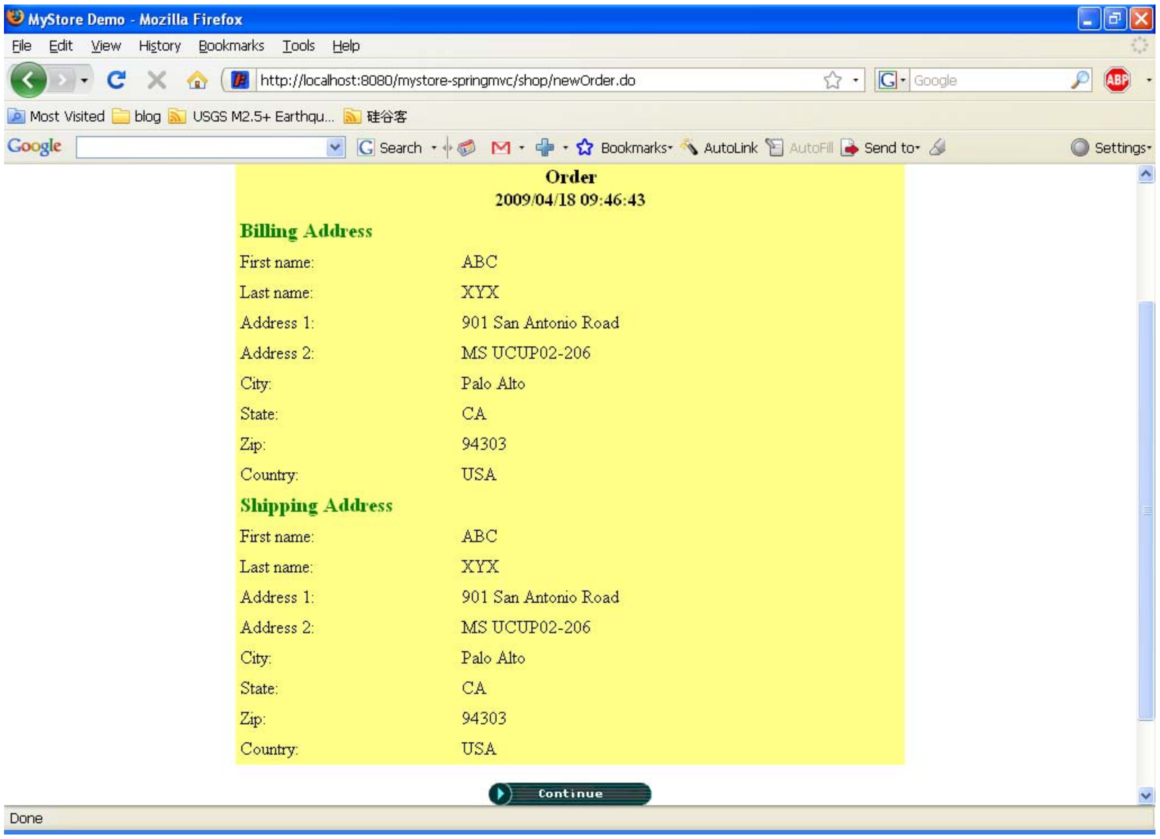
The system already fills in a test information. You can just use that for testing purpose or you can fill in your real payment information. If the shipping address is different from billing address, check the “ship to different address” checkbox. As shown in Figure 13, the system will bring you a new page to fill in shipping address.


MyStore Demo - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/mystore-springmvc/shop/newOrder.do
My Store Demo
| Sign-out | MyAccount
Search
Books | Movies & Music | Electronics | Toys | Sports
Submit
Done

Payment Details	
Card Type:	Visa
Card Number:	999 9999 9999 9999 * Use a fake number!
Expiry Date (MM/YYYY):	12/03
Billing Address	
First name:	ABC
Last name:	XYX
Address 1:	901 San Antonio Road
Address 2:	MS UCUP02-206
City:	Palo Alto
State:	CA
Zip:	94303
Country:	USA
☐ Ship to different address...	

Figure 13 Online Shpping Application Payment Detail Page Screenshot

Finally, as shown in Figure 14, after submitting the payment and shipping information, it shows the order summary page before you submit the order.


MyStore Demo - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/mystore-springmvc/shop/newOrder.do
Continue
Done

Order
2009/04/18 09:46:43

Billing Address

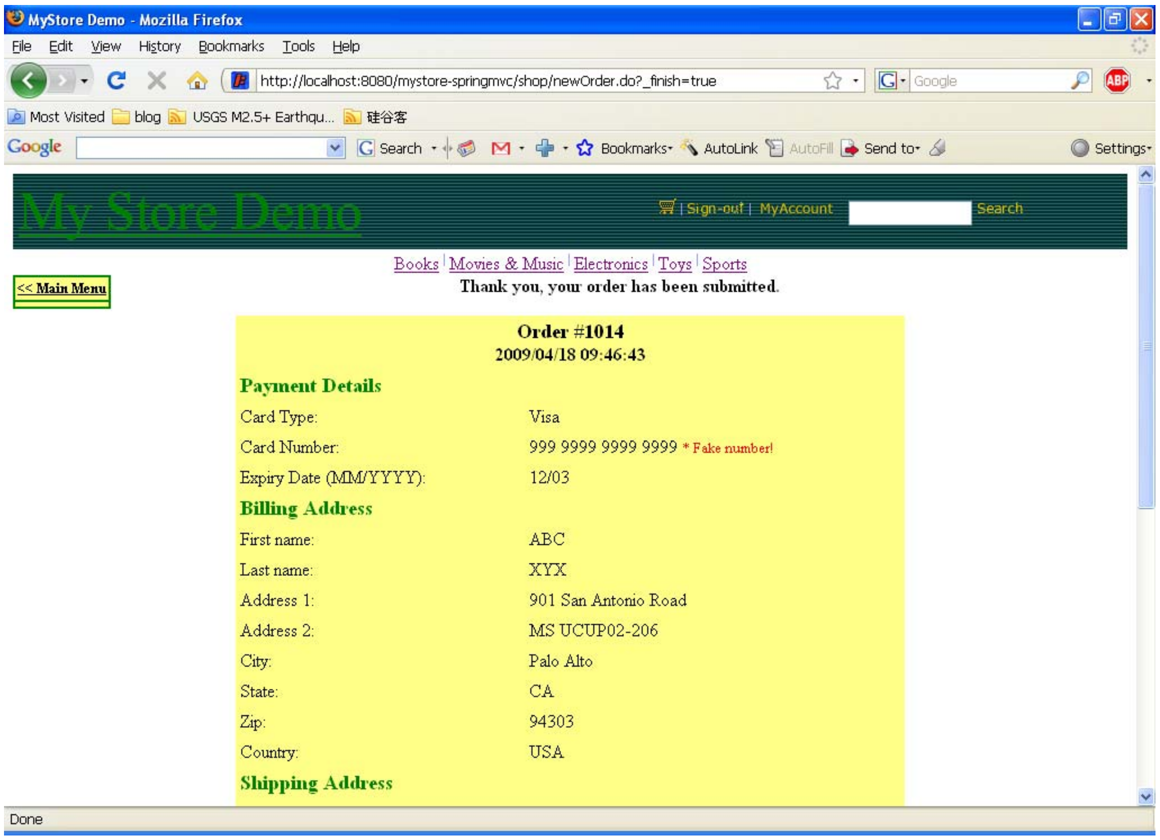
First name:	ABC
Last name:	XYX
Address 1:	901 San Antonio Road
Address 2:	MS UCUP02-206
City:	Palo Alto
State:	CA
Zip:	94303
Country:	USA

Shipping Address

First name:	ABC
Last name:	XYX
Address 1:	901 San Antonio Road
Address 2:	MS UCUP02-206
City:	Palo Alto
State:	CA
Zip:	94303
Country:	USA

Figure 14 Online Shpping Application Order Summary Page Screenshot

As shown in Figure 15 and 16, if all information is correct, click Continue to finish the order. And a confirmation page with your order details will be displayed.


MyStore Demo - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/mystore-springmvc/shop/newOrder.do?_finish=true
Most Visited blog USGS M2.5+ Earthqu... 硅谷客
Search Bookmarks AutoLink AutoFill Send to Settings
My Store Demo
Sign-out | MyAccount Search
Books | Movies & Music | Electronics | Toys | Sports
<< Main Menu
Done

Thank you, your order has been submitted.

Order #1014
2009/04/18 09:46:43

Payment Details

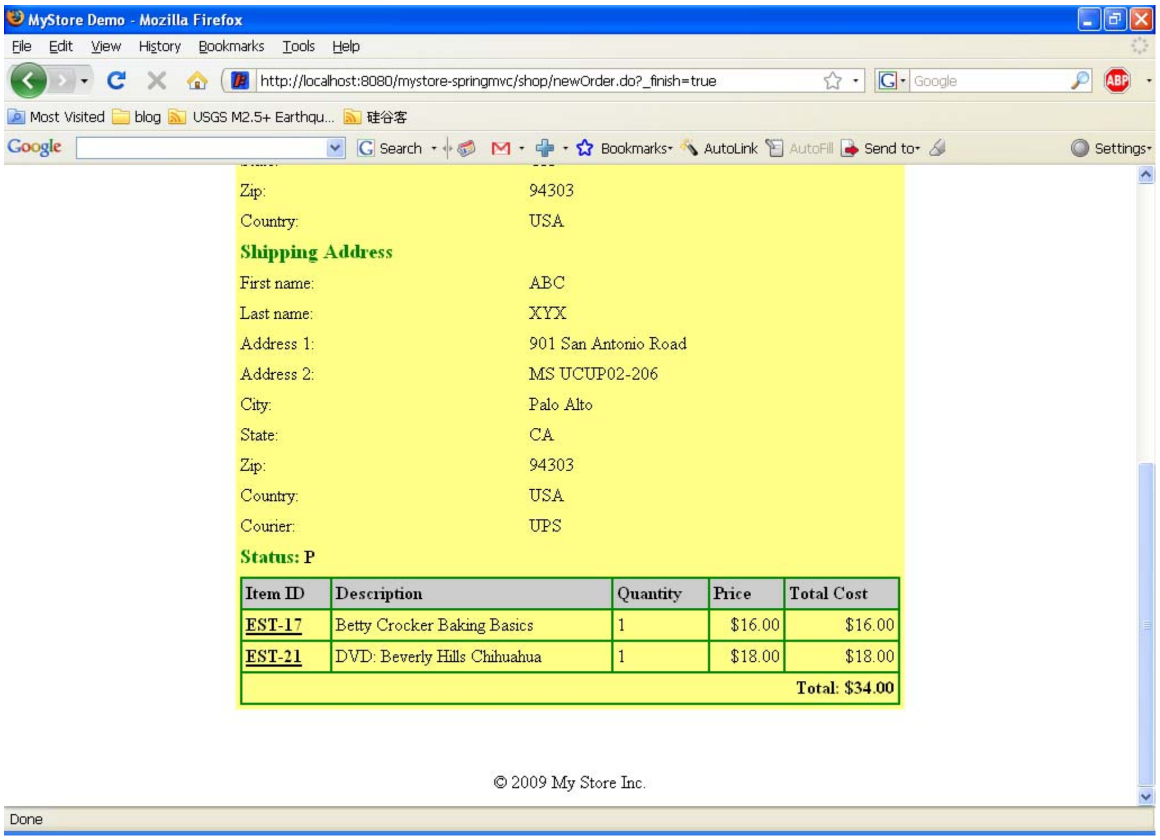
Card Type:	Visa
Card Number:	999 9999 9999 9999 * Fake number!
Expiry Date (MM/YYYY):	12/03

Billing Address

First name:	ABC
Last name:	XYX
Address 1:	901 San Antonio Road
Address 2:	MS UCUP02-206
City:	Palo Alto
State:	CA
Zip:	94303
Country:	USA

Shipping Address

Figure 15 Online Shpping Application Order Confirmation 1 Page Screenshot


MyStore Demo - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/mystore-springmvc/shop/newOrder.do?_finish=true
Most Visited blog USGS M2.5+ Earthqu... 硅谷客
Google Search Bookmarks AutoLink AutoFill Send to Settings
Done

Zip:	94303
Country:	USA

Shipping Address

First name:	ABC
Last name:	XYX
Address 1:	901 San Antonio Road
Address 2:	MS UCUP02-206
City:	Palo Alto
State:	CA
Zip:	94303
Country:	USA
Courier:	UPS

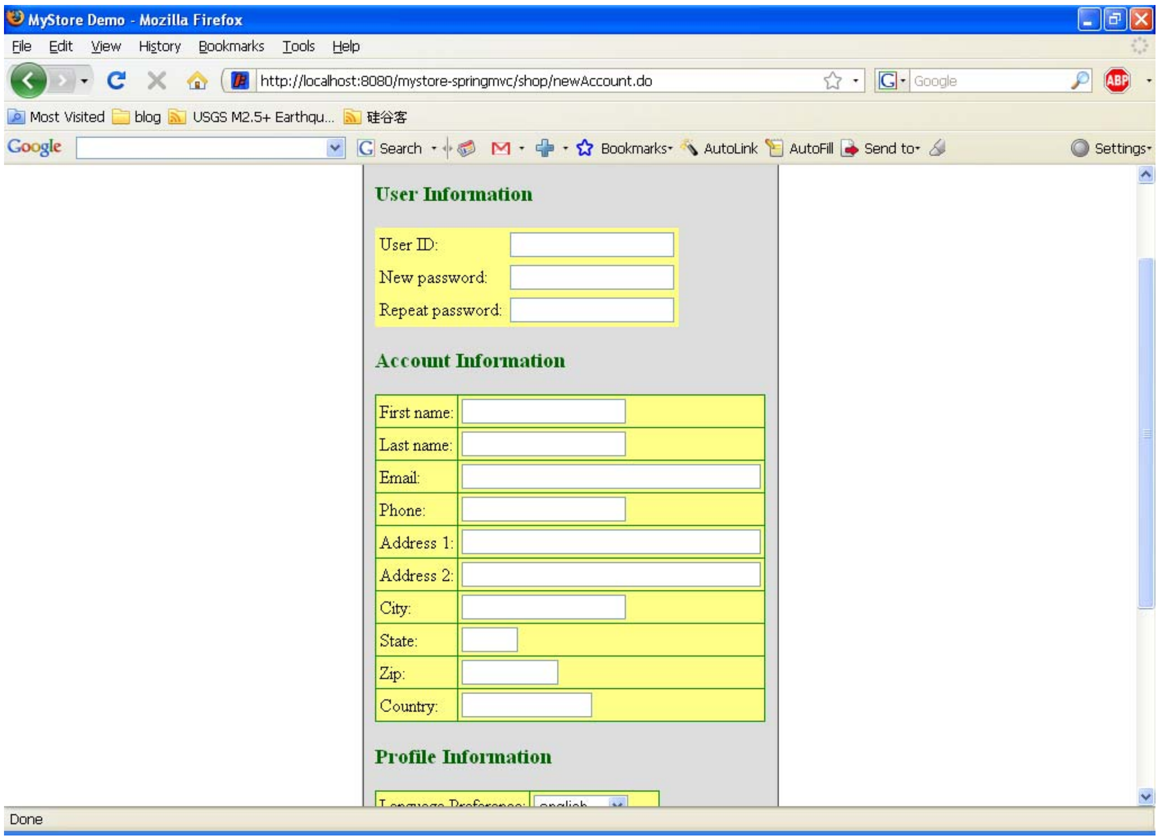
Status: P

Item ID	Description	Quantity	Price	Total Cost
EST-17	Betty Crocker Baking Basics	1	$16.00	$16.00
EST-21	DVD: Beverly Hills Chihuahua	1	$18.00	$18.00
				Total: $34.00

© 2009 My Store Inc.

Figure 16 Online Shpping Application Order Confirmation 2 Page Screenshot

In addition to the typical shopping function, the web application also lets you register new accounts or update existing accounts. As shown in Figure 17, to register a new account, click Sign-in link on top of the main page. Then click the “Register Now” button to open the register page.



MyStore Demo - Mozilla Firefox

http://localhost:8080/mystore-springmvc/shop/newAccount.do

User Information

User ID:	
New password:	
Repeat password:	

Account Information

First name:	
Last name:	
Email:	
Phone:	
Address 1:	
Address 2:	
City:	
State:	
Zip:	
Country:	

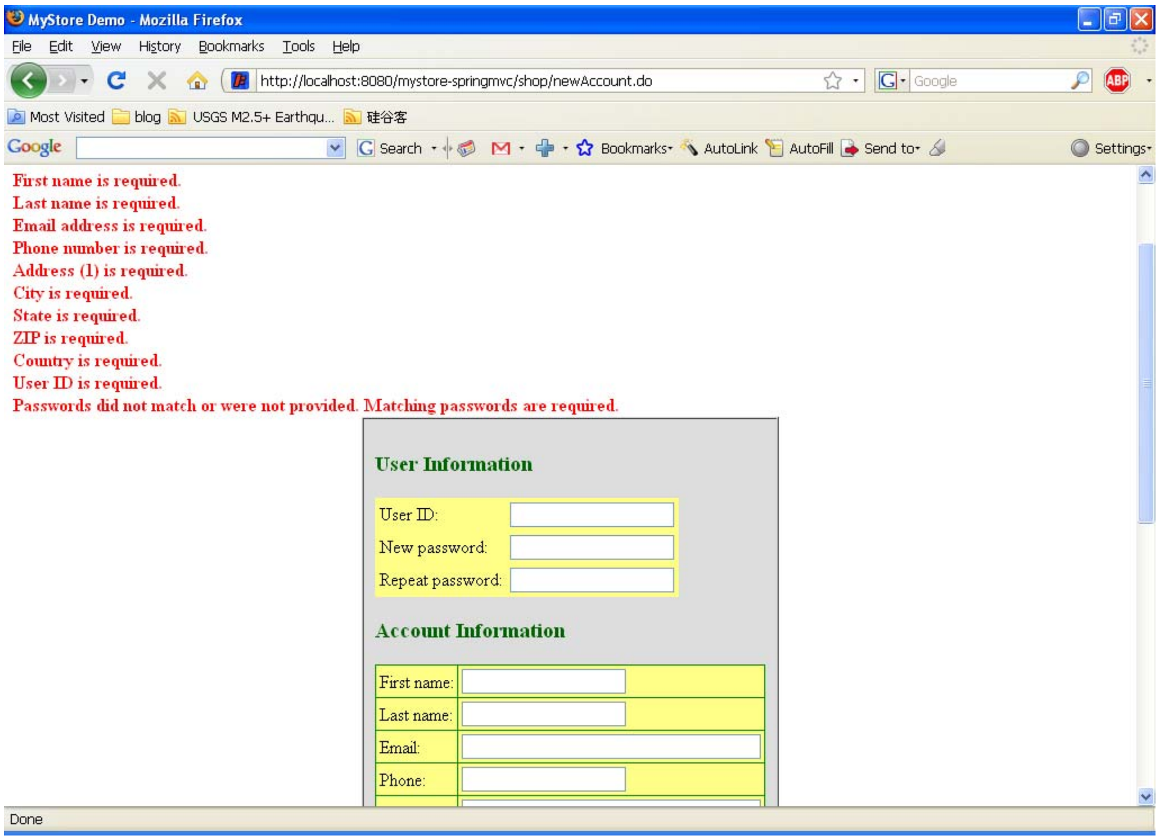
Profile Information

Language Preference: english

Done

Figure 17 Online Shpping Application User Registration Page Screenshot

As shown in Figure 18, when you submit the page, the system will validate the information you put before a new account is able to be created.


MyStore Demo - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/mystore-springmvc/shop/newAccount.do
Done

First name is required.
Last name is required.
Email address is required.
Phone number is required.
Address (1) is required.
City is required.
State is required.
ZIP is required.
Country is required.
User ID is required.
Passwords did not match or were not provided. Matching passwords are required.

User Information

User ID:
New password:
Repeat password:

Account Information

First name:
Last name:
Email:
Phone:

Figure 18 Online Shpping Application User Registration Validation Screenshot

Conclusions

Java EE is a Java based enterprise level solution for developing distributed web application. It has been widely adopted all over the world in the past 10 years. However, due to its vendor-dependent component EJB and the invasive feature of its framework, this technology has been gradually deprecated. In the last couple of years, a lot of open source alternative frameworks have come out to address these issues. Among these frameworks, the combination of Spring, Hibernate and SpringMVC are becoming increasing popular due to its vendor-independent and lightweight features. In this project, we developed an online shopping application based on these frameworks. We first introduced each of these frameworks and briefly explains the advantages they have over the traditional Java EE technology. Next, we analyzed some code snippets from the project source code to demonstrate the issues these frameworks addressed. Finally, we explained the design and deployment process of the online shopping application and provided a series of screenshots for a typical online shopping process. We concluded that these open source alternative solutions are truly lightweight and non-invasive as they simply require developers to write POJO business objects and move the code for handling cross-cutting concerns into these frameworks.

References

1. Dijkstra, E.W. 1982. On the role of scientific thought. In Selected Writings on Computing: A Personal Perspective, 60-66. Springer-Verlag.
2. Craig Walls, Ryan Breidenbach. Spring in Action 2nd Edition. Manning Publishing Co. 2008
3. Laufer, K. A hike through post-EJB J2EE Web application architecture. Computing in Science & Engineering. Volume 7, Issue 5, Sept.-Oct. 2005 Page(s):80 - 88
4. Laufer, K. A hike through post-EJB J2EE Web application architecture. Part II. Computing in Science & Engineering. Volume 8, Issue 2, March-April 2006 Page(s):79 – 87
5. Arthur,J., Azadegan,S. Spring framework for rapid open source J2EE Web application development: a case study. Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, 2005 and First ACIS International Workshop on Self-Assembling Wireless Networks. SNPD/SAWN 2005. Sixth International Conference on 23-25 May 2005 Page(s):90 - 95
6. Johnson, R. J2EE development frameworks. Computer Volume 38, Issue 1, Jan. 2005 Page(s):107 - 110
7. Chris Richardson. Untangling enterprise Java**.** Queue. Volume 4, Issue 5 (June 2006). Component Technologies. Pages:36 – 44. 2006. ISSN: 1542-7730
8. Elizabeth J. O'Neil. Object/relational mapping 2008: hibernate and the entity data model (edm). SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data. June 2008
9. The Java EE 5 Tutorial. http://java.sun.com/javaee/5/docs/tutorial/doc/
10. Yu Ping, Kontogiannis, K., Lau, T.C. Transforming legacy Web applications to the MVC architecture. Software Technology and Engineering Practice, 2003. Eleventh Annual International Workshop on 19-21 Sept. 2003 Page(s):133 – 142
11. The Spring Framework Reference Documentation.
    http://static.springframework.org/spring/docs/2.5.x/reference/index.html
12. Object-relational impedance mismatch
    http://en.wikipedia.org/wiki/Object-Relational_impedance_mismatch
13. Apache Maven
    http://maven.apache.org/