The NetBeans E-commerce Tutorial - Designing the Application

Tutorial Contents

The application that you design in this tutorial is based on a real-world scenario. After being introduced to the tutorial scenario, you consolidate a high-level list of customer requirements. You then prepare a diagram of the application's business process flow, and a series of *mockups* which help both you and your customer get a clearer picture of how the final application will look to an end-user. Finally, you break down the customer requirements into a set of implementation tasks, and structure your application so that the responsibilities and interactions among functional components are clearly defined.

This tutorial unit discusses the MVC (Model-View-Controller) design pattern. After investigating the benefits that this pattern offers, you set about mapping JSP, Servlet, and other technologies to the MVC architecture, and draft a diagram that illustrates the components of the application in terms of MVC.

This unit makes various references to the book *Designing Enterprise Applications with the J2EE Platform, Second Edition*. This book contains guidelines promoted by Java BluePrints.

Although this tutorial unit does not require use of the NetBeans IDE, it is essential because it lays the groundwork for tasks that will be covered in the following units.

You can view a live demo of the application that you build in this tutorial: NetBeans E-commerce Tutorial Demo Application.

The Scenario

This tutorial is based on the following scenario. Although this is a fictitious scenario, it demonstrates how the software you are about to develop can be applied to real-world business needs. It also serves as a platform from which you can derive customer requirements. Customer requirements should be established as clearly as possible before any design or implementation begins.

The Scenario

A small grocery store, the Affable Bean, collaborates with several local farms to supply a community with organic produce and foods. Due to a long-standing customer base and increasing affluence to the area, the store has decided to investigate the possibility of providing an online delivery service to customers. A recent survey has indicated that 90% of its regular clientele has continuous Internet access, and 65% percent would be interested in using this service.

The grocery store staff have asked you, the Java web developer, to create a website that will enable their customers to shop online. They have also asked that you create an administration console alongside the website, which will allow staff members to keep track of orders.

The store's location is in Prague, in the Czech Republic. Because regular clientele are both English and Czech-speaking, staff have requested that the website support both languages.

The grocery store has already purchased a domain and web hosting plan that provides a Java EE 6-compliant server and MySQL database server. Staff have indicated that one technically-oriented member is able to deploy the application to the production server once it is ready.

Gathering Customer Requirements

The initial phase of any project involves gathering information before making any design or implementation decisions. In its most common form, this involves direct and frequent communication with a customer. Based on the provided scenario, the Affable Bean staff have communicated to you that the application you are to create should fulfill the following requirements:

1. An online representation of the products that are sold in the physical store. There are four categories (dairy, meats, bakery, fruit & veg), and four products for each category, which online shoppers can browse. Details are provided for each product (i.e., name, image, description, price).
2. Shopping cart functionality, which includes the ability to:
   - add items to a virtual shopping cart.
   - remove items from the shopping cart.
   - update item quantities in the shopping cart.
   - view a summary of all items and quantities in the shopping cart.
   - place an order and make payment through a secure checkout process.
3. An administration console, enabling staff to view customer orders.
4. Security, in the form of protecting sensitive customer data while it is transferred over the Internet, and preventing unauthorized access to the administration console.
5. Language support for both English and Czech. (Website only)

The company staff are able to provide you with product and category images, descriptions and price details, as well as any website graphics that are to be used. The staff are also able to provide all text and language translations for the website.

> There are many practices and methods devoted to software development management. Agile software development is one methodology that encourages frequent customer inspection, and places importance on adaptation during the development cycle. Although this is outside the scope of this tutorial, each tutorial unit concludes with a functional piece of software that could be presented to a customer for further communication and feedback.

Preparing Mockups

After gathering customer requirements, you work with the Affable Bean staff to gain a clearer picture of how they expect the website to look and behave. You create a use-case that describes how the application will be used and encapsulates its behavior:

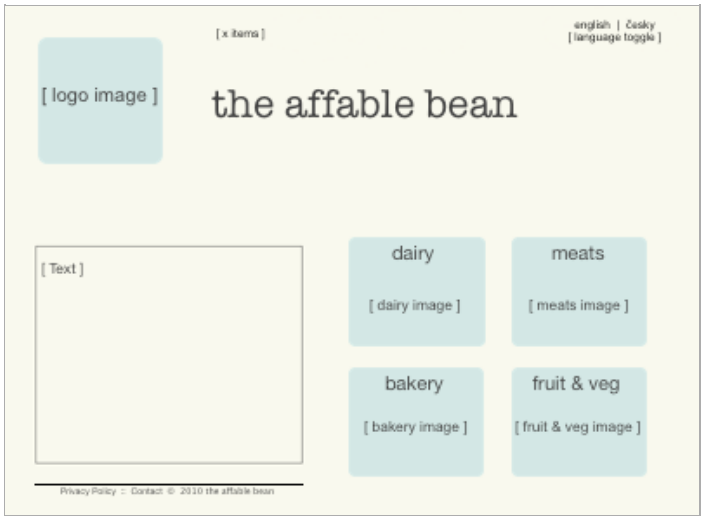
> ### *Use-Case*
> *Customer visits the welcome page and selects a product category. Customer browses products within the selected category page, then adds a product to his or her shopping cart. Customer continues shopping and selects a different category. Customer adds several products from this category to shopping cart. Customer selects 'view cart' option and updates quantities for cart products in the cart page. Customer verifies shopping cart contents and proceeds to checkout. In the checkout page, customer views the cost of the order and other information, fills in personal data, then submits his or her details. The order is processed and customer is taken to a confirmation page. The confirmation page provides a unique reference number for tracking the customer order, as well as a summary of the order.*

You also begin creating mockups. There are numerous ways to go about this task. For example, you could use storyboard software, or create a set of wireframes to relay the relationships between pages. Another common method is known as paper prototyping, where you collaborate with the customer by sketching ideas on paper.

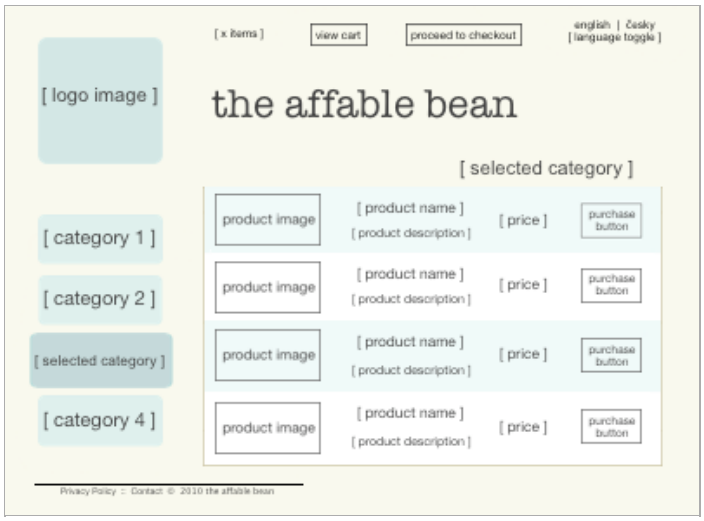
In this scenario, we've produced *mockups* of the primary pages the user expects see when navigating through the website. When we later discuss the MVC design pattern, you'll note that these pages map to the *views* used by the application.

welcome page

The welcome page is the website's home page, and entry point for the application. It

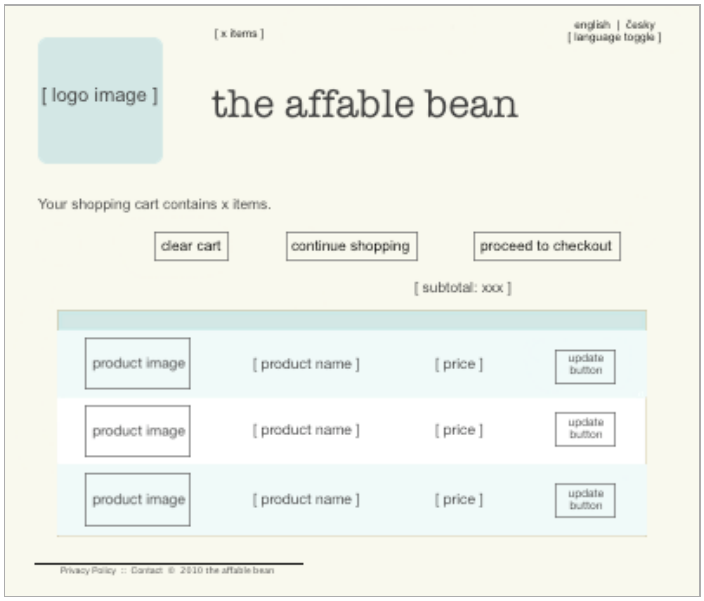


introduces the business and service to the user, and enables the user to navigate to any of the four product categories.



category page

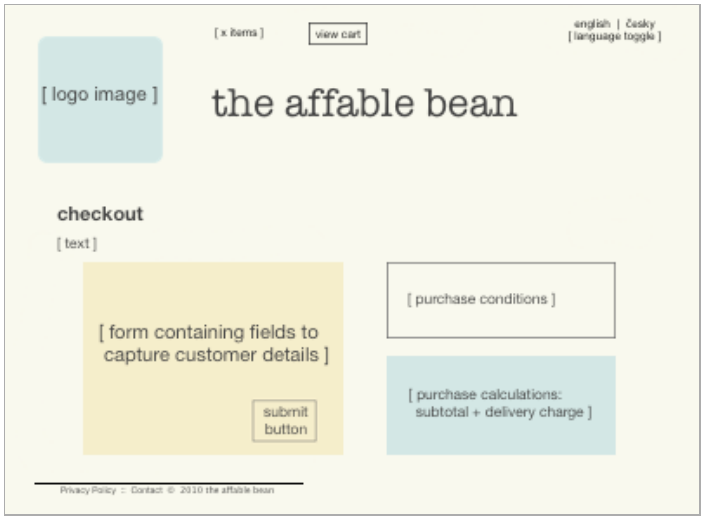
The category page provides a listing of all products within the selected category. From this page, a user is able to view all product information, and add any of the listed products to his or her shopping cart. A user can also navigate to any of the provided categories.



cart page

The cart page lists all items held in the user's shopping cart. It displays product details for each item, and tallies the subtotal for the items in the cart. From this page, a user can:

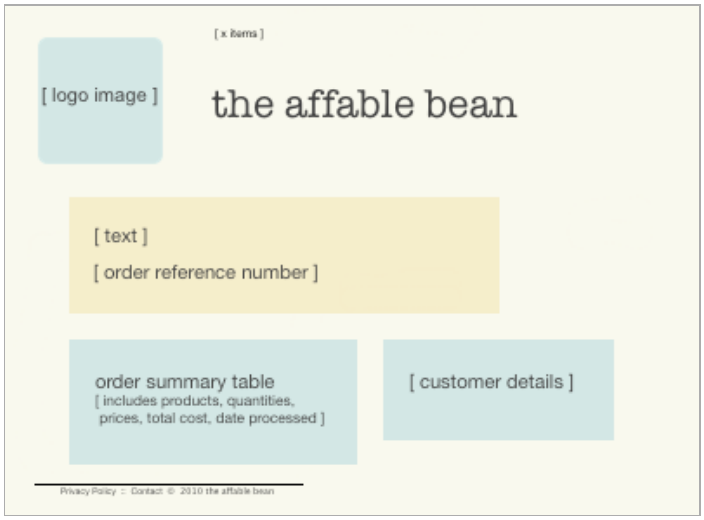
- Clear all items in his or her cart (Clicking 'clear cart' causes the 'proceed to checkout' buttons and shopping cart table to disappear.)
- Update the quantity for any listed item (The price and quantity are updated; the subtotal is recalculated. If user sets quantity to '0', the product table row is removed.)
- Return to the previous category by clicking 'continue shopping'
- Proceed to checkout



checkout page

The checkout page collects information from the customer using a form. This page also displays purchase conditions, and summarizes the order by providing calculations for the total cost.

The user is able to send personal details over a secure channel.



confirmation page

The confirmation page returns a message to the customer confirming that the order was successfully recorded. An order reference number is provided to the customer, as well as a summary listing order details.

Order summary and customer personal details are returned over a secure channel.

Also, you agree with staff on the following rules, which apply to multiple pages:

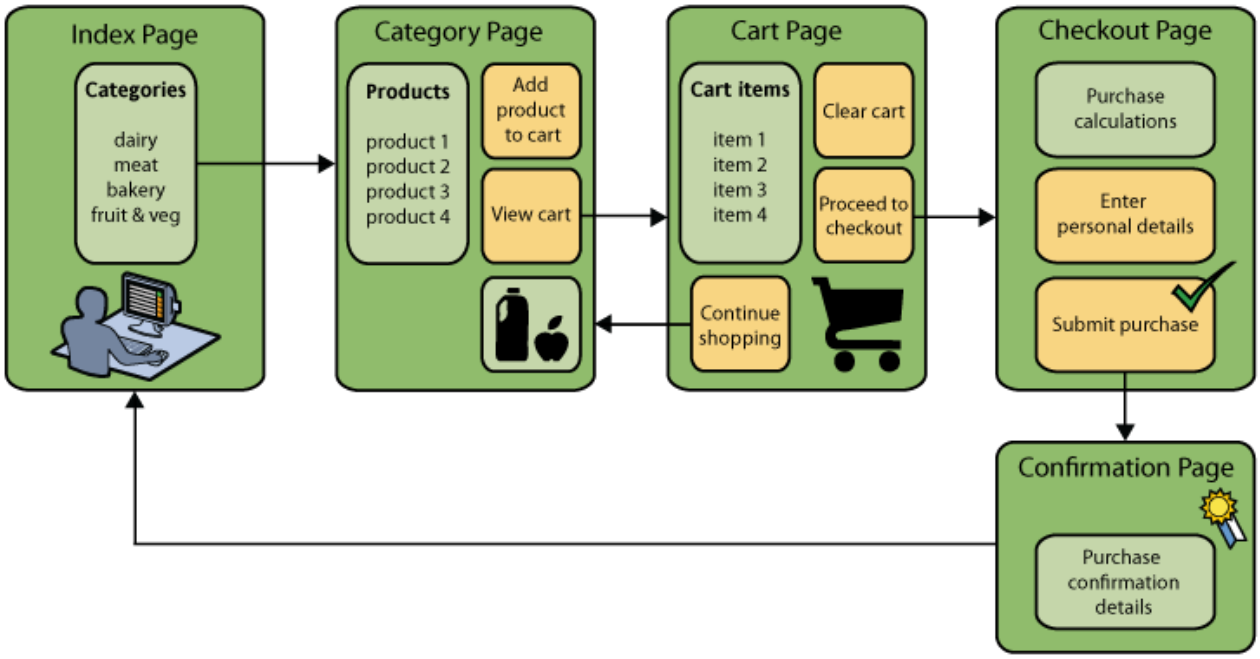
- The user is able to proceed to checkout from any page, provided that:
  - The shopping cart is not empty
  - The user is not already on the checkout page
  - The user has not already checked out (i.e., is on the confirmation page)
- From all pages, the user is able to:
  - View the status of his or her shopping cart (if it is not empty)
  - Return to the welcome page by clicking the logo image
- The user is able to select the language (English or Czech) to view the page in for all pages except the confirmation page.

Note: Although not presented here, you would equally need to work with the client to produce use-cases and mockups, and establish rules for the administration console. The NetBeans E-commerce Tutorial focuses on developing the store front (i.e., the website). However, Unit 11, Securing the Application demonstrates how to create a login mechanism to access the administration console. Also, you can examine the provided implementation of the administration console by downloading the completed application.

The Business Process Flow

To help consolidate the relationships between the proposed mockups and better illustrate the functionality that each page should provide, you prepare a diagram that demonstrates the process flow of the application.

The diagram displays the visual and functional components of each page, and highlights the primary actions available to the user in order to navigate through the site to complete a purchase.



Determining the Architecture

Before you start coding, let's examine the ways in which you can architect the project. Specifically, you need to outline the responsibilities among functional components, and determine how they will interact with each other.

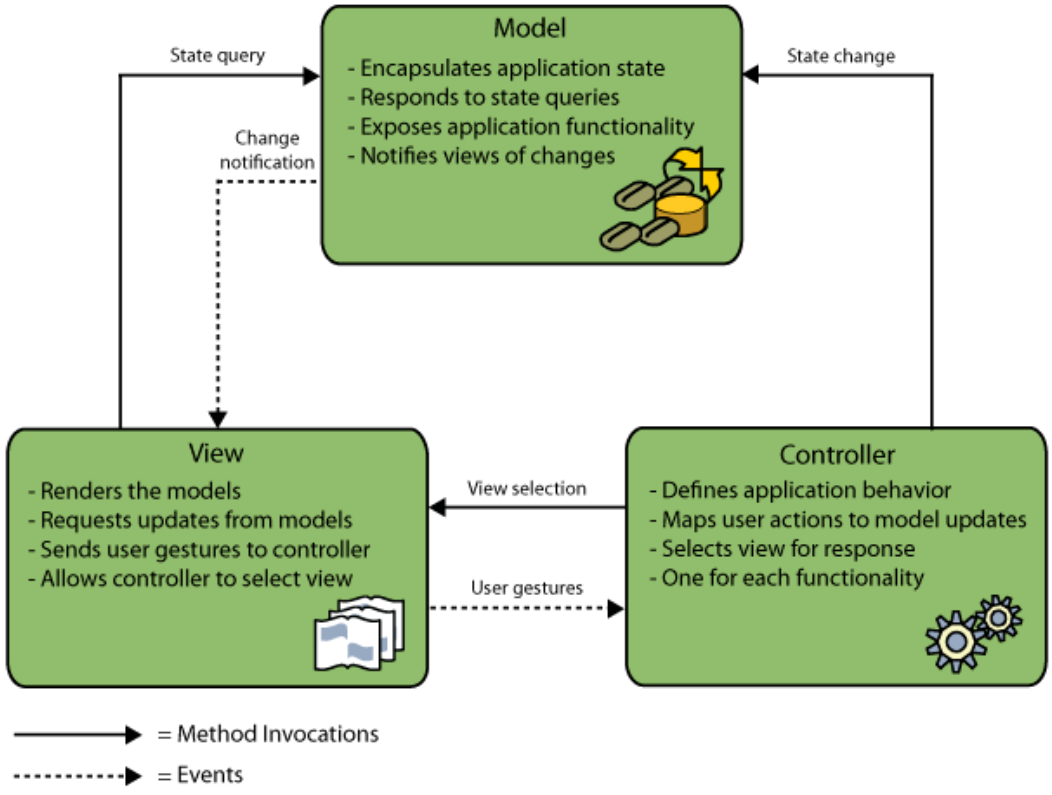
When you work with JSP technologies, you can code all of your business logic into JSP pages using scriptlets. Scriptlets are snippets of Java code enclosed in <% %> tags. As you may already be aware, JSP pages are compiled into servlets before they are run, so Java code is perfectly valid in JSP pages. However, there are several reasons why this practice should be avoided, especially when working in large projects. Some reasons are outlined in Designing Enterprise Applications with the J2EE Platform, Second Edition as follows:[1]

- **Scriptlet code is not reusable:** Scriptlet code appears in exactly one place: the JSP page that defines it. If the same logic is needed elsewhere, it must be either included (decreasing readability) or copied and pasted into the new context.

- **Scriptlets mix logic with presentation:** Scriptlets are islands of program code in a sea of presentation code. Changing either requires some understanding of what the other is doing to avoid breaking the relationship between the two. Scriptlets can easily confuse the intent of a JSP page by expressing program logic within the presentation.
- **Scriptlets break developer role separation:** Because scriptlets mingle programming and Web content, Web page designers need to know either how to program or which parts of their pages to avoid modifying.
- **Scriptlets make JSP pages difficult to read and to maintain:** JSP pages with scriptlets mix structured tags with JSP page delimiters and Java language code.
- **Scriptlet code is difficult to test:** Unit testing of scriptlet code is virtually impossible. Because scriptlets are embedded in JSP pages, the only way to execute them is to execute the page and test the results.

There are various design patterns already in existence which provide considerable benefits when applied. One such pattern is the MVC (Model-View-Controller) paradigm, which divides your application into three interoperable components:[2]

- **Model:** Represents the business data and any business logic that govern access to and modification of the 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.
- **View:** 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.
- **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.



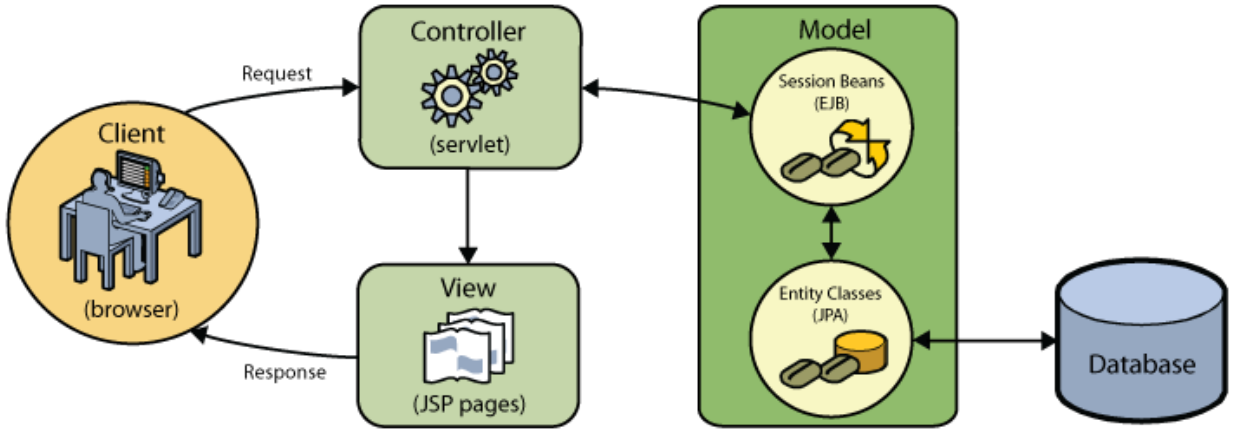
Adhering to the MVC design pattern provides you with numerous benefits:

- **Separation of design concerns:** Because of the decoupling of presentation, control, and data persistence and behavior, the application becomes more flexible; modifications to one component have minimal impact on other components. You can, for example, create new views without needing to rewrite the model.
- **More easily maintainable and extensible:** Good structure can reduce code complexity. As such, code duplication is minimized.

- **Promotes division of labor:** Developers with different skill sets are able to focus on their core skills and collaborate through clearly defined interfaces.

Note: When JSP technology was first introduced in 1999, the early specifications included a description of two model architectures: Model 1 and Model 2. Model 1 involves implementing business logic directly within JSP pages, whereas Model 2 applies the MVC pattern. For more information on Model 1 and Model 2 architectures, see Designing Enterprise Applications with the J2EE Platform, section 4.4.1: Structuring the Web Tier.

You can apply the MVC pattern to the application that you develop for the Affable Bean company. You can use a servlet as a *controller* to handle incoming requests. The pages from the business process flow diagram can be mapped to *views*. Finally, the business data, which will be maintained in a database, can be accessed and modified in the application using EJB session beans with JPA entity classes. These components represent the *model*.



Planning the Project

In order to plan the project, you need to extrapolate functional tasks from the customer requirements. The tasks that we produce will structure the implementation plan for the project, and form the outline for tutorial units that follow. In practice, the more capable you are of identifying tasks and the work they entail, the better you'll be able to stick to the schedule that you and your customer agree upon. Therefore, begin with a high-level task list, then try to drill down from these tasks dividing each task into multiple sub-tasks, and possibly dividing sub-tasks further until each list item represents a single unit of work.

- ± Set up the development environment
- ± Prepare the data model for the application
- ± Create front-end project files
- ± Organize the application front-end
- ± Create a controller servlet
- ± Connect the application to the database
- ± Develop the business logic
- ± Add language support
- ± Create administration console
- ± Secure the application

Send Us Your Feedback

See Also

Online Resources

- Java BluePrints

- J2EE Patterns Catalog
- Java BluePrints Solutions Catalog
- Java BluePrints: Model-View-Controller
- Web-Tier Application Framework Design
- The Java EE 5 Tutorial - Chapter 3: Getting Started with Web Applications

Technical Articles

- Servlets and JSP Pages Best Practices
- Design Patterns for Building Flexible and Maintainable J2EE Applications

Books

- Core Servlets and JavaServer Pages, Volume 1: Core Technologies, 2nd Edition
- Core Servlets and JavaServer Pages, Volume 2: Advanced Technologies, 2nd Edition

References

1. ^ For a more extensive list, see Designing Enterprise Applications with the J2EE Platform, section 4.2.6.8: Using Custom Tags to Avoid Scriptlets.
2. ^ For more information on the MVC pattern, see Designing Enterprise Applications with the J2EE Platform, section 11.1.1: Model-View-Controller Architecture.