Design Patterns Used

Our Project is built on the J2EE platform using the Struts2 Framework, inherently based on the Model-View-Controller (MVC) and is organised into the following tiers based on the Design Principle of “Separation of Concerns”.

We have incorporated the 2 main design principles mentioned in the GoF book -

1. Programming to an interface and not to an implementation
2. Favour Object Composition over inheritance

Each tier with its specific purpose and technologies as applicable to that tier is as shown below –

|  |  |  |
| --- | --- | --- |
| **Tier** | **Purpose** | **Technologies Used** |
| Client-tier | User interaction, presentation and devices | Browser |
| Presentation-tier | Session Management, Content creation, format and delivery | HTML, JSP, Freemarker, Velocity, JSON, Servlets and other UI elements |
| Business-tier | Business Logic, Transactions, Data Services | Data Access Object(DAO), Business Service Object(BSO) |
| Integration-tier | Resource Adapters | Java Database Connectivity(JDBC), Connectors |

Our Project uses the following design patterns as applicable to each tier -

|  |  |
| --- | --- |
| **Tier** | **Design Patterns Used** |
| Presentation-tier | Intercepting-Filter  Front-Controller  Dispatcher View |
| Integration-tier | Data Access Object |

**1] MVC Design Pattern**

**Intent :** Logical Separation of concerns based on the responsibilities of each layer – increases cohesiveness and ensures “plug-and-play” of each layer. Changing one layer should not require a change in other layers.

**Model** – A physical or logical representation of the system or its subsystem.

**Implemented Using :** Action classes, Data Access Objects(DAO’s) and Business Service Objects(BSO’s)

**View** – The View manages the graphics and text that make up the display. It interacts with Helpers to get data values with which to populate the display. Additionally, it may delegate activities, such as content retrieval, to its Helpers.

**Implemented Using :** Primarily JSP’s. Also freemarker and velocity templates provided by Struts2 and JSON(Javascript Object Notation) to enable AJAX(Asynchronous JavaScript and XML)

**Controller** –Interacts with a client, controlling and managing the handling of each request. Also implements the FrontController Design Pattern - Provides a centralized controller for managing the handling of a request.

**Implemented Using :** FilterDispatcher provided by Struts2

**2] Decorator/Wrapper/Adapter**

**Intent :** Attach additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality.

**Implemented Using :**

1 .CSS(Cascading Style Sheets) - adding styles to plain html which includes text content, images, tables, etc

2. Jasper Report Generator tool – display report generated in the form of a pdf by using struts2 plugin as a result type

**Benefits:**

1. More flexibility than static inheritance: With decorators, responsibilities can be added and removed at run-time simply by attaching and detaching them. In contrast, inheritance requires creating a new class for each additional responsibility. This gives rise to many classes and increases the complexity of a system.

2. Avoids feature-laden classes high up in the hierarchy. We can define a simple class and add functionality incrementally with Decorator objects. Functionality can be composed from simple pieces. Extending a complex class tends to expose details unrelated to the responsibilities being added.

**3] Iterator**

**Intent :** Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation.

**Implemented Using :** JAVA Iterator API for traversing all kinds of Collections – maps, lists and sets

**Benefits:**

1. It supports variations in the traversal of an aggregate. Complex aggregates may be traversed in many ways. Iterators make it easy to change the traversal algorithm: Just replace the iterator instance with a different one. You can also define Iterator subclasses to support new traversals.

2. It simplifies the Aggregate interface. Iterator's traversal interface obviates the need for a similar interface in Aggregate, thereby simplifying the aggregate's interface.

**4] Bridge**

**Intent :** Decouple an abstraction from its implementation so that the two can vary

independently.

**Implemented Using :** Separated the interface of DAO from its concrete implementation and used iBatis DAOManager in the Service Layer for automated Dependency Injection of the DAO’s leading to Inversion ofControl(IOC)

**Benefits:**

1. Hiding implementation details from clients. One can shield clients from implementation details, like the sharing of implementor objects and the accompanying reference count mechanism (if any).

2. Decoupling interface and implementation. An implementation is not bound permanently to an interface. The implementation of an abstraction can be configured at run-time. It's even possible for an object to change its implementation at run-time. Decoupling Abstraction and Implementor also eliminates compile-time dependencies on the implementation.

**5] Composite**

**Intent :** Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.

**Implemented Using :** JUnit Test Cases

**Benefits:**

1. Makes the client simple: Clients can treat composite structures and individual objects uniformly. Clients normally don't know (and shouldn't care) whether they're dealing with a leaf or a composite component.

2. Makes it easier to add new kinds of components. Newly defined Composite or Leaf subclasses work automatically with existing structures and client code. Clients don't have to be changed for new Component classes.

**6] Chain of Responsibility/Intercepting Filter**

**Intent :** Avoid coupling the sender of a request to its receiver by giving more than one object a chance to handle the request. Chain the receiving objects and pass the request along the chain until an object handles it.

**Implemented Using :** The HTTP Request from the client(browser) passing through a number of Interceptors before being handled by the Action class. Each of these interceptors have the capability of either passing the request onto the next interceptor in the chain of sending a response back to the client. A number of interceptors including 1 custom made – AuthenticationInterceptor has been used for this purpose

**Benefits:**

1. Reduced coupling. The pattern frees an object from knowing which other object handles a request. An object only has to know that a request will be handled "appropriately." Both the receiver and the sender have no explicit knowledge of each other, and an object in the chain doesn't have to know about the chain's structure.

2. Added flexibility in assigning responsibilities to objects. Chain of Responsibility gives you added flexibility in distributing responsibilities among objects. You can add or change responsibilities for handling a request by adding to or otherwise changing the chain at run-time. You can combine this with subclassing to specialize handlers statically.

**7] DispatcherView**

**Intent :** Some of the responsibilities of a Controller include managing the choice of and dispatching to an appropriate View. This behavior may be partitioned into a separate component, referred to as a Dispatcher

**Implemented Using :** The FrontController provided by struts2 i.e. the FilterDispatcher provides a “Dispatcher” result type which dispatches control to the specified result page based on the result name.

**Benefits:**

1. Centralizes Control and Improves Reuse and Maintainability: Control processing is handled in a central place for multiple requests. It is easier to manage these activities and perform dispatching from a centralized point, since a central access point means code is reused across multiple requests, reducing duplication and easing maintenance.

2. Improves Application Partitioning: Use of helpers results in a cleaner separation of the view from an application's business processing. The helpers, in the form of JavaBeans(JSP 1.0+) and Custom tags (JSP 1.1+), provide a place for business logic to be factored out of the JSP, where scriptlet code quickly becomes cumbersome and unwieldy in large projects.

**8] Data Access Object(DAO)**

**Intent :** An object that encapsulates and abstracts access to data from a persistent store or an external system

**Implemented Using :** Using dao interfaces and their concrete implementations. Each dao interacts with the DB using sql and populates data into objects. The Action class and dao objects are connected using Business Service Objects(BSO’s)

**Benefits:**

1. Enables Transparency: Business objects can use the data source without knowing the specific details of the data source's implementation. Access is transparent because the implementation details are hidden inside the DAO.

2. Enables Easier Migration: A layer of DAOs makes it easier for an application to migrate to a different database implementation. The business objects have no knowledge of the underlying data implementation. Thus, the migration involves changes only to the DAO layer.

**9] Mediator**

**Intent :** Define an object that encapsulates how a set of objects interact. Mediator promotes loose coupling by keeping objects from referring to each other explicitly, and it lets you vary their interaction independently.

**Implemented Using :** The Business Service Objects connects the Action and DAO Objects and thereby reduces the coupling between them. This lets the concrete implementation of DAO class methods to vary irrespective of their usage in the Action classes. The DAO object is abstracted from the Action class using the Service layer

**Benefits:**

1. It limits subclassing. A mediator localizes behavior that otherwise would be distributed among several objects. Changing this behavior requires subclassing Mediator only; Colleague classes canbe reused as is.

2. It simplifies object protocols. A mediator replaces many-to-many interactions with one-to-manyinteractions between the mediator and its colleagues. One-to-many relationships are easier tounderstand, maintain, and extend.