classHidra

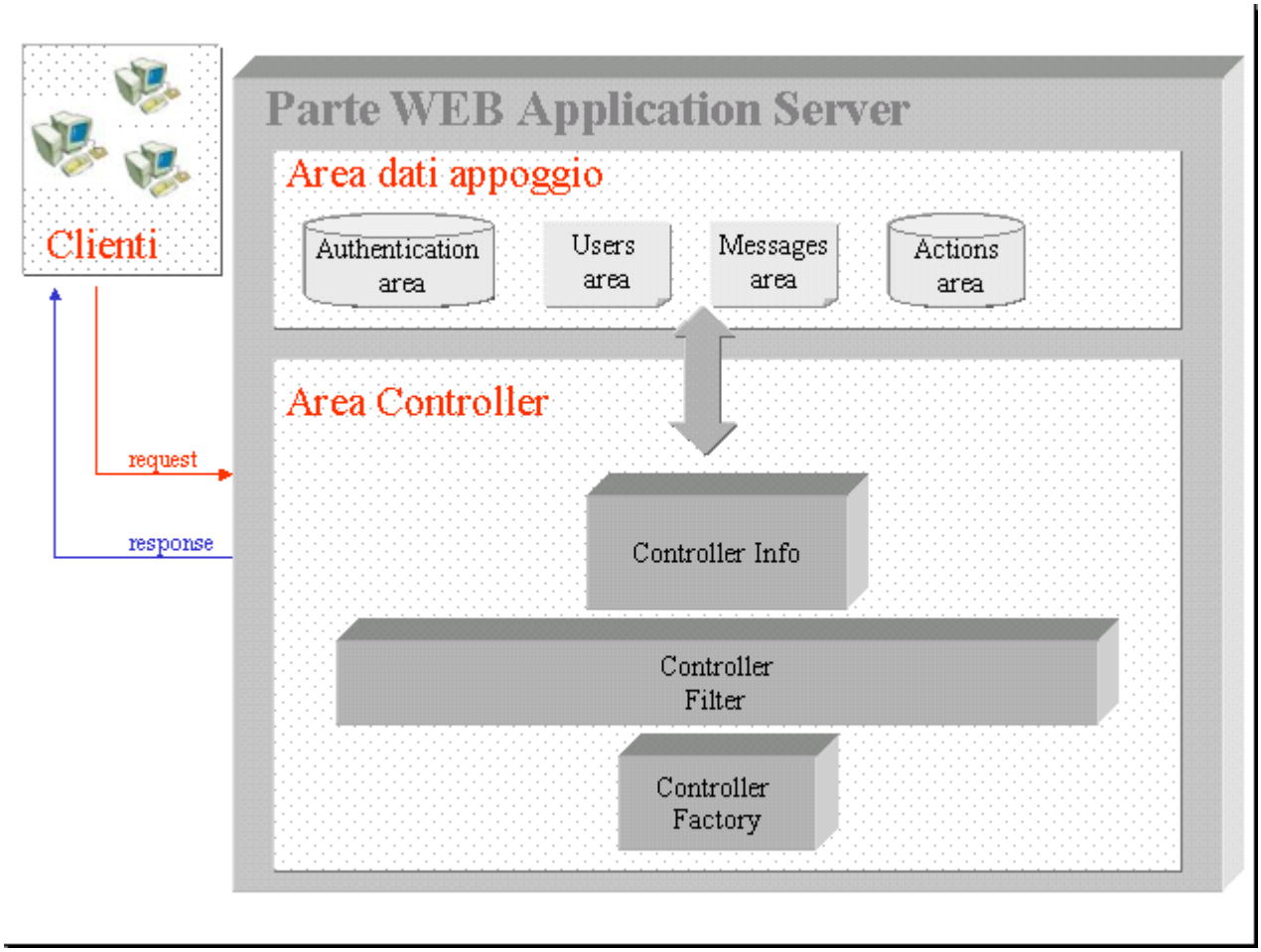
Manuale

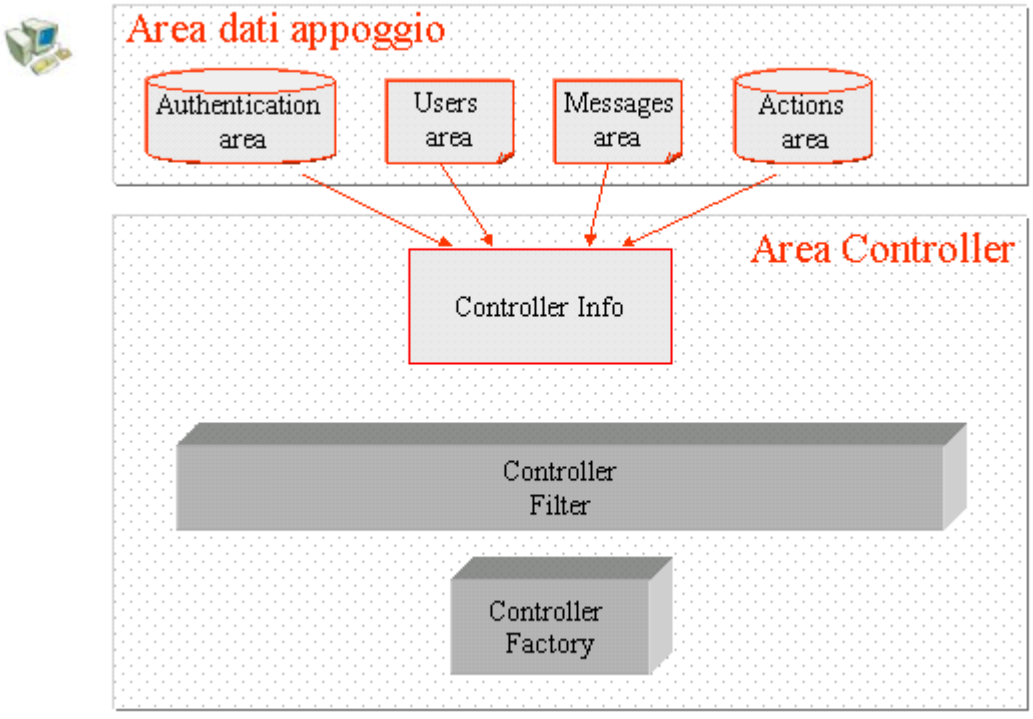
v.1.4.3

1. MVC che cos'è e a cosa permette di fare

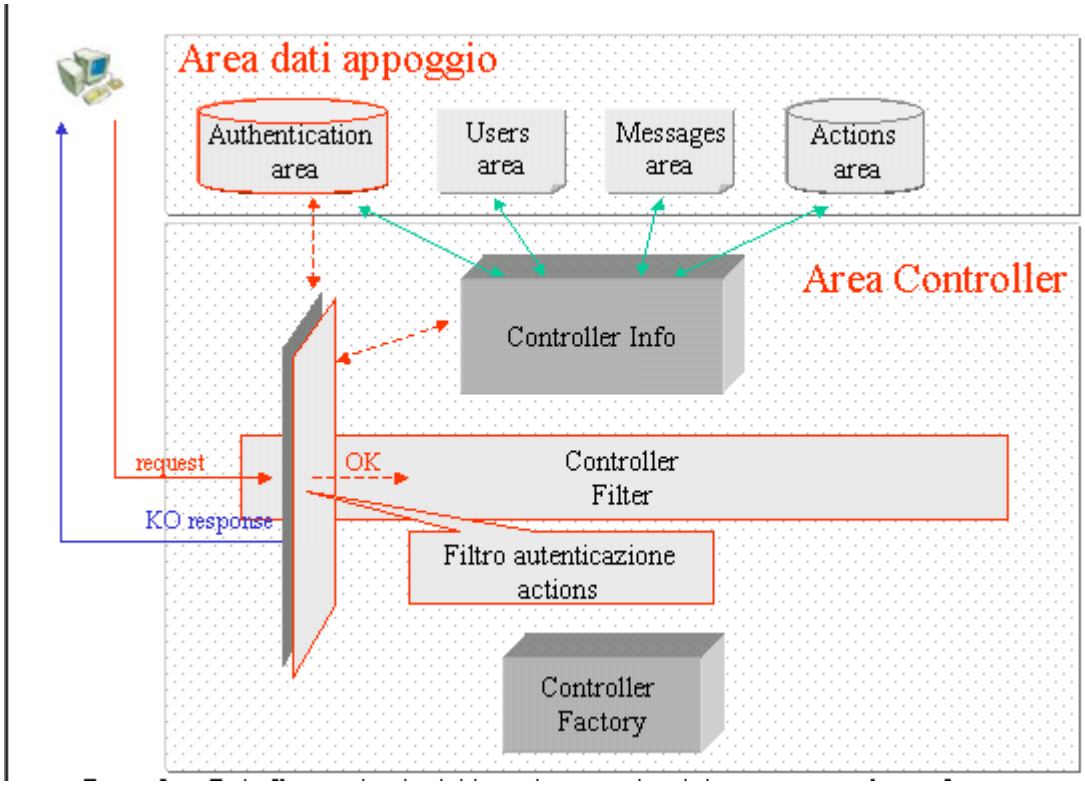
Molto spesso si sente parlare nel mondo Java (e non solo) di applicazioni che rispettano l'MVC, framework MVC e così via. Ma cosa è l'MVC? È l'acronimo di tre parole: Model View Controller. Fino a qui non è difficile.
Sostanzialmente la "filosofia" del MVC è quella di progettare un'applicazione (sia essa web che stand-alone), in modo da dividere l'interfaccia utente (UI), quindi la parte di View, dalla parte che rappresenta i dati dell'applicazione e le regole che governano le modalità con cui questi dati vengono acceduti e modificati (Model) e dalla parte che gestisce le richieste dell'utente e le azioni che devo essere intraprese sul Model (Controller).
In realtà l'MVC dovrebbe essere utilizzato per la realizzazione della maggior parte delle applicazioni, in modo da renderla il più indipendente possibile la parte di visualizzazione, quella relativa ai dati e quella di gestione vera e propria.
Il rispetto del paradigma è di grande aiuto, per esempio, qualora si abbia la necessità di modificare in modo pesante esclusivamente l'aspetto grafico di un portale, senza voler rimettere mano anche alla logica e alla struttura dati dello stesso.
Non sempre comunque il paradigma MVC è necessario: ad esempio, programmi in cui gli oggetti View sono fondamentali ed il collegamento con il Model molto stretto (non ci sono molti controlli e verifiche, ed il contesto è meno importante), si può eliminare il livello Controller.
Esistono molti framework aderenti all'MVC: Struts, JSF, Hibernate, Spring, etc. Attualmente il più diffuso è Struts (alla data in cui viene scritto l'intervento), ma in decadimento a favore delle JSF. I framework disponibili in ambito Java per l'ambiente web sono basati tutti grosso modo sul pattern MVC dal punto di vista della modellazione; dal punto di vista tecnologico ovviamente si basano tutti sulla tecnologia Servlet, che è lo standard per le applicazioni web nel mondo Java. In realtà, più o meno tutti sono orientati a risolvere le problematiche dello strato Controller e talvolta di quello di View piuttosto che del Model. Ci si può chiedere quindi che vantaggio ci possa essere a utilizzare l'MVC di un altro framework, piuttosto che quello di Spring. Il vantaggio può essere quello di riutilizzare le competenze acquisite in altre tecnologie, oppure di integrare senza grosse modifiche delle porzioni applicative già realizzate.

2. Catena Funzionale

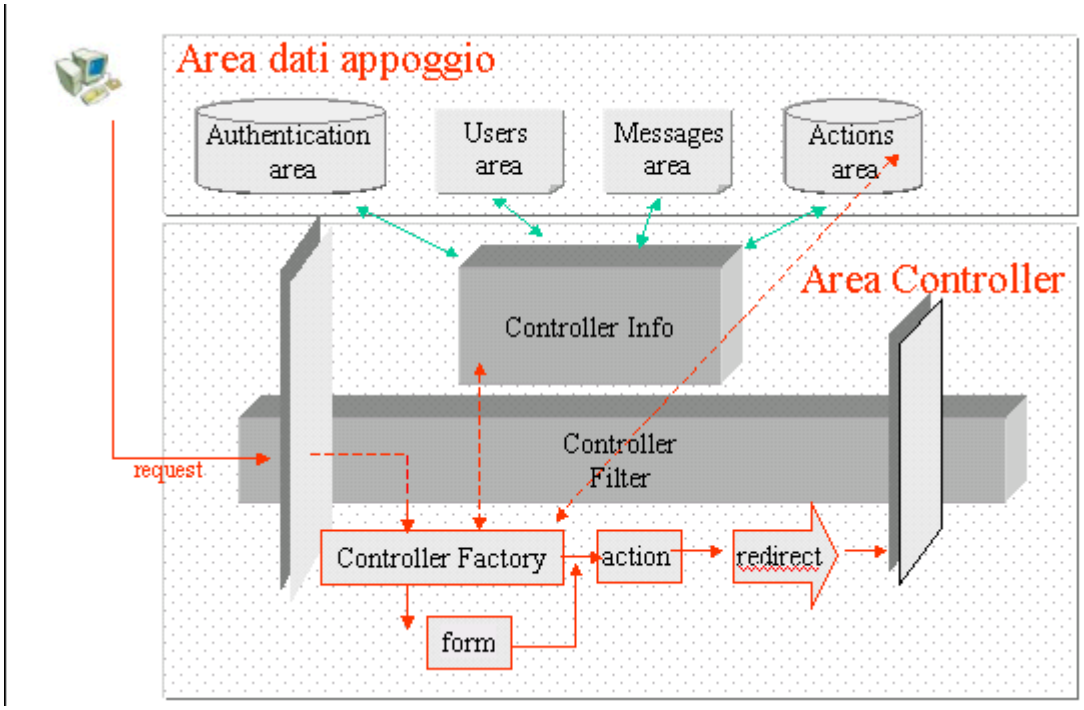




Passo 1: Dalle diverse aree di appoggio si inizializza l'area [**INFO**] del Controller.
La procedura viene eseguita in fase di **init** del servlet, che rappresenta il Controller.



Passo 2: Dal ***client*** arriva la richiesta (***request***) nel tipo **<*nome_azione*>*.bs***.
Viene invocata e coinvolta la parte [***FILTER***] del Controller che recupera il codice ***ID azione*** e ne verifica la congruenza con il primo controllo
[Filtro autenticazione actions]



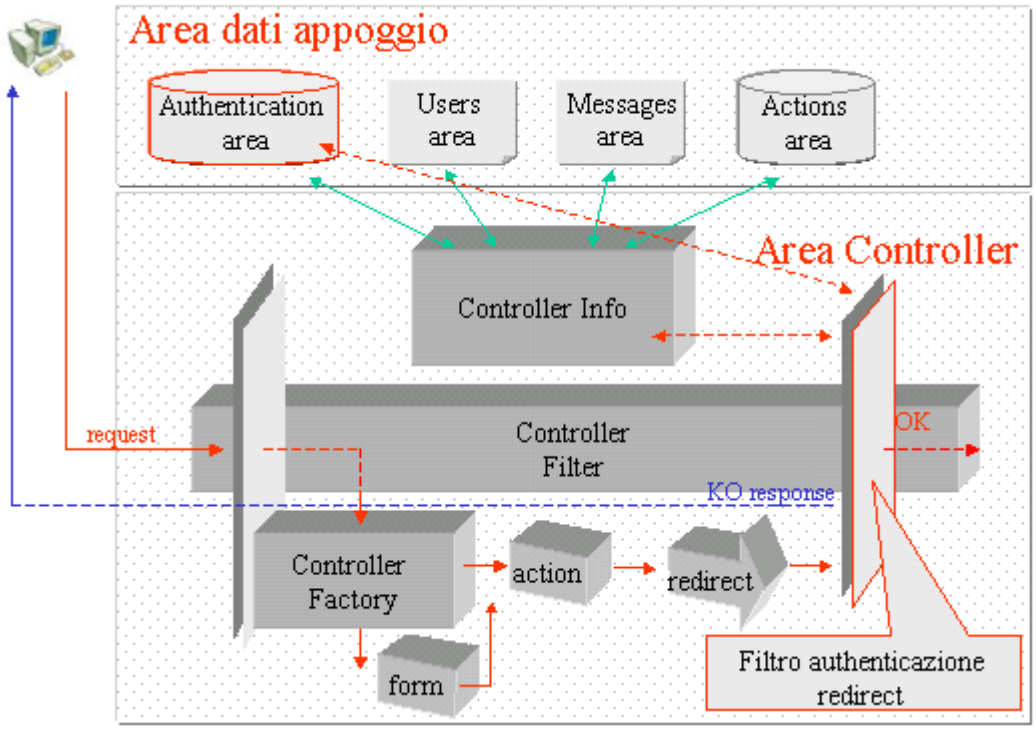
Passo 3: In base al codice ***ID azione*** il [***Controller Factory***] recupera dalla [***INFO – Actions area***] tutte le informazioni necessarie per creare un oggetto :

[***form***] contenitore delle dati per la pagina;

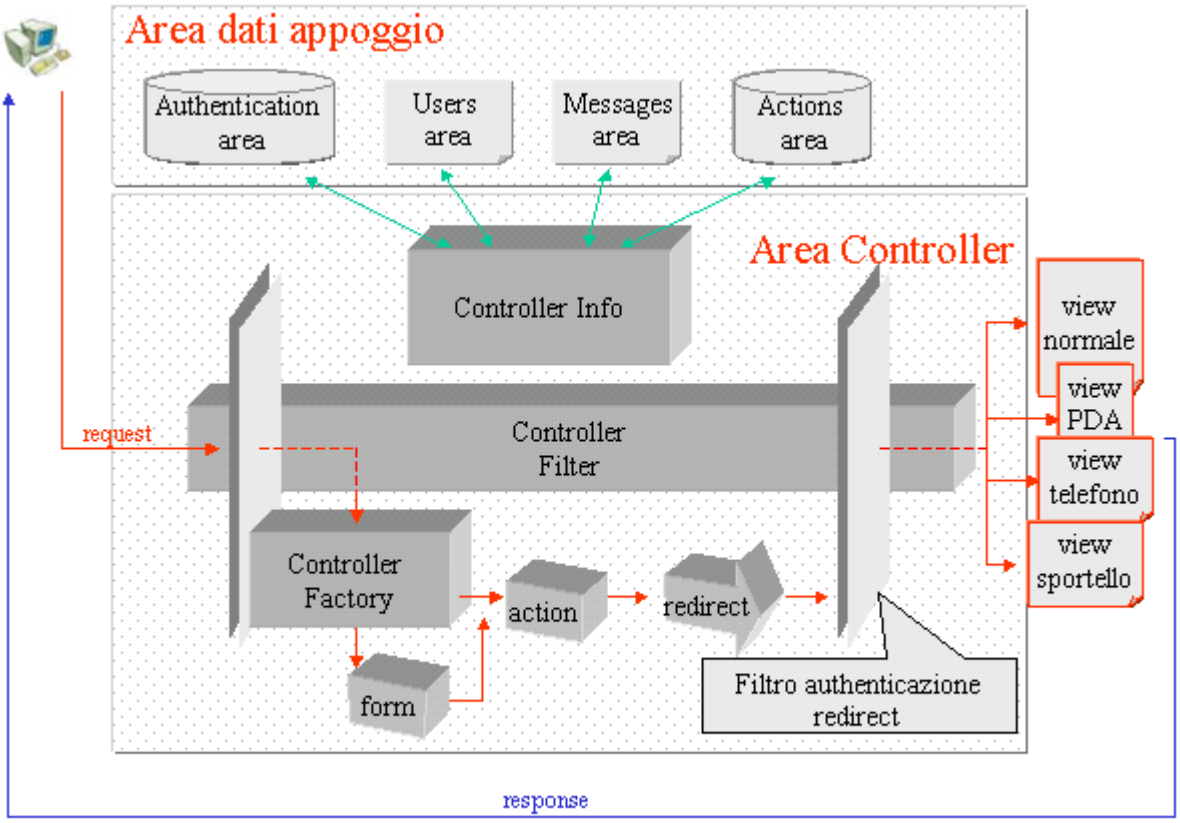
[***action***] parte della logica di Business;

Eseguire suo metodo actionservice (algoritmo della logica di Business) che in questo caso produce un oggetto

[***redirect***] nel caso più semplice si tratta di un ***url*** per la relativa pagina jsp da caricare



Passo 4: Con il [redirect] creato dal actionsevice arriviamo al secondo [Filtro authenticazione Redirect] che verifica se utente che sta lavorando adesso possiede i permessi necessario per vedere la [view – in caso piu semplice jsp, posonno essere altri tipi di risposte: xml, e.c…]. In caso positivo si genera una view – pagina di risposta e si invia al cliente.


Area dati appoggio
Authentication area
Users area
Messages area
Actions area
Area Controller
Controller Info
request
Controller Filter
Controller Factory
action
redirect
form
Filtro authenticazione redirect
view normale
view PDA
view telefono
view sportello
response

3. Per creare una azione nuova

Versioni prima di 1.4.3:

- Censire la nuova azione nel file ***action.xml*** valorizzando :
  - Form
  - Redirect
  - Action
- Creazione fisica degli oggetti :
  - Form java
  - Action java
  - View (jsp, html, xml,...)
- Creazione della struttura del form
  - Campi
  - Validate

Versione 1.4.3 e successive:

- Grazie al implementazione di @annotations non e più necessario dichiarare stream, form, action e redirects nel ***action.xm.*** Nel momento di creazione del'oggetto sarà sufficiente inserire le @annotations che disegnano la struttura: @Stream, @Action, @Bean. Per esempio:

```
@Action (
        path="amanager",
        name="formAmanager",
        redirect="/jsp/amanager/canvas.jsp",
        navigated="true",
        memoryInSession="false",
        reloadAfterAction="false",
        redirects={
                @Redirect(
                        auth_id="amn_id",
                        path="*"
                )
        }
        )

        @Bean (     name="formAmanager")

        @Stream(
                name="def_control_permission",
                applied={
                        @Apply_to_action(action="*")
                }
        )
```

- E anche elementi dalla sezione <action-mapping>

```
@ActionMapping (
```

```
redirects={
        @Redirect(
                path="/jsp/amanager/canvas.jsp",
                descr="Users/Groups/Authentications",
                mess_id="title_fw_amanager"
                )
        }
)
```

- **IMPORTANTE**: al momento di caricamento la sistema DEVE sapere dove si trovano l'elementi con @annotations per analizzare la struttura (altrimenti dovrà rifare la scansione nel tutti pacchetti disponibili che implicherà nel performance di progetto) – per quello nel classhidra_app.properties devono essere specificati i percorsi per il pacchetti necessari (esempio):

```
#Objects with annotations
application.package.annotated=application.web
application.package.annotated.0=it.classhidra.framework.web
…
application.package.annotated.N=…
```

- Creazione fisica degli oggetti :
  - Form java
  - Action java
  - Component.java – che e un'insieme tra Form e Action
  - View (jsp, html, xml,...)
- Creazione della struttura del form
  - Campi
  - Validate

Altri operazioni sono rimasti invariati:

- Valorizzare la parte view (jsp) a partire dai dati contenuti nella form e popolare il file ***messaggi.xml***
  per l'esternalizzazione delle stringhe (multilingua)
- ***Logica di business***
  - Implementare la logica di business in actionservice per l'***action.java***
- ***Richiamare l'azione creata***
  - Se l'azione è invocata dal menu è necessario valorizzare correttamente il file ***menu.xml***
- ***Integration Test***
- ***System e business test***

4. Parte Di configurazione

Per la creazione di una pagina è stato creato un progetto di esempio che permette la visualizzazione di una pagina dove devono essere inseriti una utenza ideale e d una password e selezionando il tasto un tasto si viene rimandati ad un'altra pagina dove vengono visualizzati l'utenza e la password e successivamente una lista statica di nomi.
Per creare una nuova pagina bisogna importare il modello MVC di ClassHidra (scaricabile da http://sourceforge.net/projects/classhidra/) è apportare le seguenti modifiche necessarie per l'esecuzione del programma.
Inizialmente bisogna configurare quattro pagine una denominata "classhidra_app.properties" utile per la configurazione delle pagine infatti si andrà a passare per il nostro caso la pagina con il quale si vuole accedere al programma e la seconda pagina che si vuole visualizzare.

```
# Path for actions.xml
      application.path=classhidra_example
      #application.path.config=/config/
      application.auth.action.enterpoint=prova
      application.auth.tag_navigation.action_excluded=prova;actionLista_prova;
      application.load.resource.mode=thread
      #application.load.resource.mode=normal
      #application.external_loader.class=it.application.external_loaders.exLoader_generic
      application.external_loader.class=
      application.pin=Od+lUoMxjTGv5aP/Sg4yU+IEXkM=

      #application.config.db=classhidra_config

      #transformation Event AfterView: WebSphere
      #include, forward, both
      application.transformation.event.after.elaborationmode=include
      #controller, filter
      application.transformation.event.after.elaborationpoint=controller
      application.transformation.event.after.elaborationwrapper=it.classhidra.core.controller.wrappe
      rs.bsCharResponseWrapper

      #transformation Event AfterView: WebLogic
      #application.transformation.event.after.elaborationmode=both
      #controller, filter
      #application.transformation.event.after.elaborationpoint=filter
      #application.transformation.event.after.elaborationwrapper=it.classhidra.core.controller.wrapp
      ers.bsByteArrayResponseWrapper

      #Objects with annotations
      application.package.annotated=application.web
      application.package.annotated.0=it.classhidra.framework.web
```

Un'altra pagina denominata " classhidra_auth.properties" serve per configurare JAAS authentication in caso se vi serve.

```
      application.auth.manager=it.classhidra.core.tool.jaas_authentication.auth
      _manager
      application.auth.policy=bs0_policy
      application.auth.jaas_systemname=java.security.auth.login.config
      application.auth.jaas_fileconfig=jaas.config
```

La terza pagina che bisogna configurare denominata "classhidra_db.properties"utile per la connessione con il database.

```
      application.db.connectiontype=pooldatasourcev51
      application.db.datasource=java:comp/env/jdbc/databaseDB2

      application.db.id.1=connectionODBC
      application.db.connectiontype.1=drivermanager
      application.db.driver.1=sun.jdbc.odbc.JdbcOdbcDriver
      application.db.url.1=jdbc:odbc:dleasing
```

```
application.db.user.1 = AppServer
application.db.password.1 = Utente00e
```

La quarta pagina denominata “classhidra_log.properties” serve per configurare log-manager interno.

```
application.log.stub=
application.log.level=DEBUG
application.log.pattern=
application.log.path=
application.log.name=classhidra
application.log.maskname=application_
application.log.maskformat=yyyyMMdd_hhmmssss
application.log.maxlength=1000000
application.log.maxfiles=10
application.log.flashrate=0
application.log.flashsize=1024
application.log.write2console=true
```

Nel caso se si utilizza un gestore di log esterno – basta specificare il percorso per il gestore:

```
application.log.stub=it.classhidra.core.tool.log.stubs.stub_log
```

che servirà come un ponte tra sistema e nuovo gestore. Nello stub basterà sovrascrivere il metodo:

```
public void write(HashMap hm){
	try{
		Object mess = hm.get(iStub.log_stub_mess);
		Object exception = hm.get(iStub.log_stub_exception);
		Object throwable = hm.get(iStub.log_stub_throwable);
		Object request = hm.get(iStub.log_stub_request);
		Object servletcontext = hm.get(iStub.log_stub_servletcontext);
		Object level = hm.get(iStub.log_stub_level);

		…

	}catch(Exception e){
	}
}
```

Successivamente è necessario aggiungere nella cartella “WEB_INF” la pagina “web.xml” che permette il richiamo di tutte le pagine utili per il funzionamento del programma.
Infatti, andrà a richiamare la pagina dei filtri con il suo relativo indirizzo, la pagina che compie i controlli e quella degli errori.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web
Application 2.3//EN" "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app id="WebApp">
  <display-name>leasing</display-name>
  <filter>
	<filter-name>bsFilter</filter-name>
	<display-name>bsFilter</display-name>
	<filter-class>it.classhidra.core.controller.bsFilter</filter-
class>
	<init-param>
		<param-name>(nome)</param-name>
		<param-value>(valore)</param-value>
	</init-param>
  </filter>
  <filter-mapping>
	<filter-name>bsFilter</filter-name>
	<url-pattern>/actions/*</url-pattern>
```

```
  </filter-mapping>
  <filter-mapping>
        <filter-name>bsFilter</filter-name>
        <url-pattern>/*</url-pattern>
  </filter-mapping>
  <servlet>
        <servlet-name>bsController</servlet-name>
        <display-name>bsController</display-name>
        <servlet-
class>it.classhidra.core.controller.bsController</servlet-class>
        <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
        <servlet-name>bsController</servlet-name>
        <url-pattern>/Controller</url-pattern>
  </servlet-mapping>

  <welcome-file-list>
        <welcome-file>index.html</welcome-file>
        <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
  <error-page>
        <error-code>404</error-code>
        <location>errors.jsp</location>
  </error-page>
</web-app>
```

L'ultima parte di configurazione del nostro sistema è denominata "**bsController.tld**" che si trova nella sottocartella "tlds" nella cartella "WEB_INF" e severe come libreria per Tag Library proprietaria.

5. Creazione

Successivamente si passa alla creazione della pagine del nostro modello MVC che nel nostro caso si compone in tre fasi:

- gestione e creazione delle pagine in Java denominate “Form” che permette la creazione e gestione delle variabili
- creazione delle pagine “Action” che permettono la gestione delle azioni eseguite all’interno della pagina
- creazione delle pagine JSP dove sono contenute tutte le parti grafiche e i collegamenti con le pagine.

La prima pagina che si deve creare è quella la “Form”infatti questa ci permette di creare e gestire le variabili utili per compiere determinate, funzioni all’interno della pagina.

Il nostro esempio di una pagina Form denominata “FormProva” si trova al seguente indirizzo: “prova.src.it.application.prova.web.formbean.formProva.java”.

La nostra form di prova inizia con l’importazione delle classi utili per il corretto funzionamento del programma, successivamente di iniziano a creare le variabili utili, cioè quelle che servono per compiere le funzioni prescelte per le nostre pagine. Successivamente si andranno a creare due vettori (Elements, elements_show) e altre due variabili (page, current_page)che servono per identificare i vari stati delle pagine.

Eseguita la creazione delle variabili si andrà ad instanziare le variabili e i vettori da noi create si andranno a creare i metodi get e set per poter utilizzare le variabili all’esterno della nostra classe.

Dopo la creazione della pagina in Java denominata Form si passa alla creazione delle Action queste vanno create una per ogni pagina infatti gestiscono la azioni che possono essere eseguite da esse.

Nel nostro esempio abbiamo creato due action che si trovano in “prova.src.it.application.prova.web.formaction” in questa cartella si trovano due tipi di action nella prima denominata “Actionprova” utile per la pagina di prova , che anche la pagina iniziale, e la seconda Action è “ActionLista_prova”.

Per la prima action (Actionprova), all’inizio si importano le classi che servono per il funzionamento della nostra action, successivamente si andrà ad estendere la nostra classe e si andrà a creare un metodo denominato “actionservice” in questa pagina non compirà nessuna azione.

Nella seconda pagina denominata “ActionLista_prova” come nella precedente andremo a richiamare le classi esterne e ad estendere la classe principale, ma a differenza della precedente faremo compiere delle funzioni particolari.
Infatti all’inizio del metodo “`actionservice`” incominceremo con una funzione che ci permette di importare dalla pagina precedente lo user e la password successivamente è stato creato una istruzione if che permette di richiamare il metodo “`loadDates`”, una funzione che è stata creata per poter stampare a video una lista di nomi creata al suo interno. Successivamente sono stati creati altri due metodi che permettono uno, denominato “`popolateShow`”,di popolare la tabella con all’interno i nomi immessi precedentemente nel metodo “`loadDates`” e l’altro metodo, denominato “`formProva find`”, permette l’inserimento di una funzione che permette di ricordare la pagina precedente ed attraverso dei bottoni permettono il ritorno alla precedente pagina senza perdere i dati immessi.

Infine dopo la creazione delle pagine in Java per la gestione delle pagine sono state create le pagine in Jsp e in Javascript.
Le pagine in Javascript che si trovano “prova.WebContent.javascript” permettono il funzionamento del tasto nella prima pagina e gestiscono i collegamenti tra una pagina e l’altra.
Invece le pagine in JSP che si trovano “prova.WebContent.jsp” la prima, denominata “prova”, inizia richiamando il metodo “`bsController″` che effettua il controllo di eventuali errori presenti , successivamente richiama le pagine Javascript associate e prosegue con la creazione di tag di testo presenti nella nostra pagina e con un bottone che permette il collegamento alla seconda pagina creata.
La seconda pagina incomincia come la pagina di prova e prosegue stampando a video i dati inseriti precedentemente e con una successiva tabella contenente i dati presente nella action.