Personalizzazione dell'interfaccia

- ADT è il metodo consigliabile per personalizzare l'interfaccia
  - facilità di manutenzione
  - velocità nello sviluppo
- Limitazioni degli adt
  - Non sempre ADT sono disponibili
  - In caso di personalizzazioni "pesanti" gli ADT possono non essere indicati
  - Meno adatti a pagine molto complesse e interattive
  - Mancanza di versionamento automatico del codice

→ In questi casi necessario lavorare a livello di JSP

Moduli Fragment

"Fragment module" = "OSGi fragment bundle"

Consentono di fare l'override di intere JSP di un modulo

Strettamente legate alla specifica versione del modulo

Possibili problemi di gestione in fase di upgrade del portale o installazione di patch/fix packs

Moduli Fragment (cont.)

Nel MANIFEST.MF si dichiara con un header su quale modulo si vuole applicare il fragment

- necessario specificare anche l'esatta versione del bundle "target"
- Fragment-Host: <symbolic name del bundle>; bundle-version="..."

Quindi si copia la JSP originale e la inserisce nella stessa struttura di directory nel bundle del Fragment

Quando si fa il deploy, il Fragment va in stato "resolved" e fa l'override della JSP

Esercizio Fragment - 1

Creare un nuovo “Liferay Module Project Fragment”

- project name “app-fragment”
- location “<...>/modules/platform/view-layer”
- selezionare il Liferay runtime
- Host OSGi bunle: selezionare com.liferay.login.web-....jar
- selezionare "META-INF/resources/login.jsp"

Creazione "manuale" del fragment

metodo da usare se non si usa il template "fragment" (ad esempio se non si usa il Liferay IDE/Developer Studio)

- Creare un "Liferay Module Project" ad esempio con un template "api"
- Cancellare gli artefatti non utili al fragment (ad esempio il package com.example.api)
- Aprire una gogo shell (telnet 11311 localhost)
- Verificare la versione del modulo che si vuole personalizzare
  ad esempio per il portlet di login eseguire **"lb -s login.web"**
- Inserire nel bnd.bnd un header utilizzando lo specifico numero di versione rilevato
- Fare riferimento allo snippet "02-fragment/01-bnd.bnd"

Fragment manuale (cont.)

Posizionarsi su “src/main/resources”

Creare una nuova folder “META-INF/resources”

Cercare il file “login.jsp” nella directory dei sorgenti di Liferay

- si trova in “modules/apps/foundation/login/login-web/src/main/resources/META-INF/resources/login.jsp”
- oppure scompattarlo dal portale da “osgi/marketplace/Liferay CE Foundation.lpkg”

Copiarlo nella nuova folder

bnd.bnd

```
Bundle-Name: SMC App Fragment
Bundle-SymbolicName: it.smc.lrtraining.app.fragment
Bundle-Version: 1.0.0

Fragment-Host: com.liferay.login.web;bundle-version="..."

-jsp: *.jsp,*.jspf
-plugin.jsp: com.liferay.ant.bnd.jsp.JspAnalyzerPlugin
```

Nel bnd.bnd viene abilitato il plugin “JspAnalyzerPlugin”: serve per poter utilizzare le tag-lib nelle JSP

La riga “-jsp” serve per dire che le estensioni per i file JSP sono “.jsp” e “.jspf”

Personalizzazione della JSP

Aprire il file "login.jsp" da " “src/main/resources/META-INF/resources”"

Aggiungere un qualsiasi testo all’inizio, ad esempio CUSTOM JSP

Test del Fragment

Deploy

Osservare il log

- viene fatto lo stop e lo start del modulo Host, non del Fragment

Controllare da gogo shell

- il modulo Fragment rimane in stato “Resolved”

Testare la login

Fare undeploy e ritestare

Dipendenze su fragment

nella login.jsp del fragment potrebbe essere necessario introdurre logiche che hanno dipendenze

possibile introdurle nel build.gradle per evitare errori di validazione della JSP

inoltre possibile usare un PortletFilter per iniettare @Reference a servizi direttamente nella request

quindi nella JSP recuperare la reference

Esercizio Portlet Filter - 1

nella login.jsp dello app-fragment, aggiungere lo snippet “03-login.jsp.txt”

creare un nuovo modulo “login-portlet-filter” di tipo “service”

- classe “LoginRenderFilter”
- package “it.smc.lrtraining.login.portlet.filter”
- service “PortletFilter”

snippet "05-LoginRenderFilter.java"

build.gradle: snippet “06-build.gradle”