Portale di un’istituzione educativa

Progetto di Ingegneria del Software 2021/2022
Professore Davide Rossi
Tutor Fiorela Ciroku

Relazione scelte progettuali e di interfaccia

Gruppo di lavoro:

- Magazzù Gabriele - 0000933823.
- Marchese Renato Junior - 0000949773.
- Monti Stefano - 0000948005.

La relazione delle scelte progettuali e di interfaccia comprende la discussione e le motivazioni delle scelte implementative effettuate.

Sommario

Introduzione

Per la fase di sviluppo del software abbiamo seguito e utilizzato i principi fondamentali dell’ingegneria del software e i pattern dedicati alla scrittura di codice flessibile e mantenibile.
É infatti di fondamentale importanza, nel caso di un software object oriented, utilizzare i **design pattern** affinché il software stesso sia propenso al cambiamento e quindi poco rigido.

Se un domani il software avesse bisogno di manutenzione o aggiornamenti deve essere possibile “mettere mano” sul codice senza che vi siano problematiche legate a questioni di rigidità / fragilità del software.

Si ricorda, infatti, che la fragilità è la tendenza di un programma di “rompersi” in diversi posti quando viene effettuata una singola modifica; solitamente il nuovo problema si crea in un’area che non ha una relazione concettuale con l’area precedentemente modificata. Risolvere questi problemi potrebbe causarne di altri.
D’altra parte, un software presenza rigidità se un singolo cambiamento causa un cascata di conseguenti cambiamenti nei moduli che dipendono dal modulo dove è avvenuto il primo

cambiamento. Più moduli devono essere modificati, più il progetto software sarà rigido.

I principi dell'ingegneria del software e i design pattern ci permettono di creare delle soluzioni efficienti tali per cui il progetto sarà propenso al cambiamento.
Bisogna, però, sempre valutare il trade off tra inserimento di ulteriori pattern e il livello di complessità del codice.
Aggiungere pattern, e di conseguenza anche classi ausiliarie, permette una migliore gestione del progetto, ma aumenta di gran lunga la complessità.

Per quanto possibile sono stati seguiti dunque i principi SOLID e i principi base della programmazione object-oriented.
Sono inoltre stati utilizzati i design pattern **Singleton** e **Façade**.

Gestione delle pagine del sito

Il progetto è stato realizzato seguendo il modello di una single page application: non vi sono presenti più pagine, bensì verrà aggiornata e modificata sempre la medesima.

La classe entry point "Portale.java" (classe alla quale gwt fa riferimento tramite il metodo *onModuleLoad()*) istanzia un oggetto di tipo "Mainpage" e lo aggiunge al RootPanel della pagina del sito.

La classe "Mainpage.java" estende la classe padre Composite di GWT, la quale permette di creare dei Widget personalizzati.
Tale classe presenta un VerticalPanel denominato "mainPanel" al quale verranno aggiunte di volta in volta le pagine necessarie al sito, anche queste widget personalizzate.

Dunque abbiamo fondamentalmente gestito il sito nel seguente modo:

- La classe Portale aggiunge la classe Mainpage al root panel.
- La classe Mainpage aggiunge una determinata classe (widget) al proprio mainPanel, "ripulendo" il pannello, in modo da non avere conflitti con oggetti inseriti precedentemente.

È stato deciso di comune accordo di realizzare una interfaccia utente minimale, senza applicare fogli CSS o ulteriori implementazioni grafiche.

Architettura Client-Server GWT

L'architettura client-server del sito è gestita tramite l'architettura tipica di GWT.
Tre sono le classi fondamentali a cui fare riferimento: GreetingService.java, GreetingServiceAsync.java e GreetingServiceImpl.java.

Come si può capire dai nomi stessi delle classi:

- GreetingService è la classe che si occupa di dichiarare le operazioni che un client può chiedere al server in maniera sincrona.
- GreetingServiceAsync è la classe che si occupa di dichiarare le medesime operazioni in maniera asincrona, usando una AsyncCallback.

- GreetingServiceImpl è la classe che si occupa di implementare le operazioni precedentemente chiamate attraverso le operazioni del Database.

Il client (classe a cui serve usare una determinata operazione del server) userà la classe Singleton (che istanzia un punto di accesso globale a un oggetto di tipo GreetingServiceAsync) per effettuare le chiamate AsyncCallback.

N.B. : GWT supporta operazioni single Thread. É quindi di fondamentale importanza capire come vengono chiamate le operazioni dal server, perché potrebbero risultare problemi derivanti dal tempo di risposta delle singole chiamate.

Pattern

Singleton: Il pattern Singleton serve per assicurarsi che una classe abbia solo un'istanza e fornisca un punto di accesso globale ad essa.
E' un meccanismo globale per trovare la reference ad una istanza.

E' bene utilizzare questo pattern quando:
- una classe nel programma dovrebbe avere una sola istanza disponibile per tutti i client;
- si necessita uno stretto controllo sulle variabili globali.

È stato scelto di utilizzare il pattern Singleton nelle seguenti situazioni:
- Accesso globale al server → classe Singleton.java.
  Questa classe istanzia una variabile globale di tipo GreetingServiceAsync, la quale ci permette di utilizzare le operazioni del server.
- Accesso globale alla sessione attuale di un utente → classe ActualSession.java.
  Questa classe permette di accedere in modo sincronizzato allo Status e all'email della sessione attuale di un singolo utente che interagisce con il portale.
  Sono presenti 6 diversi "Status": uno per ogni tipologia di account, uno per la gestione di un utente non registrato e uno per la gestione di un utente che inserisce una password errata.
- Accesso globale al database → classe UniDB.java.
  Questa classe, quando richiamata, permette di effettuare tutte le operazioni possibili sul database. Lavora in stretto contatto con la classe Singleton e con la classe del server, le cui operazioni sono appunto realizzate nella classe del DB.

Façade: pattern il cui utilizzo consiste nell'essere una "classe facciata" tra il client (classe di alto livello) e un sottosistema (classi / operazioni di basso livello).
Se non si segue un approccio di questo genere bisogna fare attenzione alle dipendenze molto forti che si creano tra il client e il sottosistema.
Infatti in caso di modifiche di un componente del sottosistema sarà necessario modificare anche il client. Nel modo da noi progettato invece sarà semplicemente sufficiente gestire i cambiamenti nelle classi del sottosistema e nella classe facciata senza intaccare in alcun modo la classe del client.

È stato utilizzato principalmente per tutte le operazioni che le varie tipologie di account possono effettuare.

Sono infatti state create le seguenti classi:

- AdminFacade.java → si occupa di gestire le operazioni proprie di un account di tipo admin.
- ProfessorFacade.java → si occupa di gestire le operazioni proprie di un account di tipo professor.
- StrudentFacade.java → si occupa di gestire le operazioni proprie di un account di tipo student.
- SecretaryFacade.java → si occupa di gestire le operazioni proprie di un account di tipo secretary.
- GeneralUserFacade.java → si occupa di gestire tutte le operazioni comuni a più tipologie di account.