Relazione su scelte progettuali e di interfaccia

Gabriel Riccardo Alsina `Num.Matricola 890823`
Carlotta Carboni `Num.Matricola 893185`
Alessandro Pasi `Num.Matricola 893268`

29/11/2022

Contents

1 Descrizione del progetto

Il progetto per l'anno accademico 2021/2022 consiste nella realizzazione di un sito web per un'istituzione educativa. Il sito web deve avere una homepage che fornisca informazioni per l'università, i dipartimenti e le informazioni di contatto (numero di telefono e indirizzo) accessibili a chiunque lo visiti. Il sito web deve avere un portale accessibile da studenti, docenti, segreteria dell'università e amministratore del sito.
Se uno **studente** accede al portale, può:

- Visualizzare la lista dei corsi disponibili.
- Iscriversi a un corso.
- Registrarsi per un esame di un corso.
- Visualizzare le informazioni personali (nome, cognome, email, corsi ed esami a cui si è iscritti).
- Visualizzare i voti degli esami svolti.

Se un **docente** accede al portale, può:

- **Creare/Modificare/Cancellare un corso**. Per creare un corso, il docente deve inserire il nome del corso, la data di inizio e fine del corso, una descrizione del corso, e opzionale un co-docente o un esame. Un corso può avere solo un esame.
- **Aggiungere/Modificare/Cancellare un esame**. Per creare un esame, il docente deve aggiungere la data, l'ora e durezza dell'esame e l'aula dove si svolgerà l'esame. In un esame, un studente può registrarsi solo una volta. Visualizzare le proprie informazioni personali (nome, cognome, corsi che insegna con relativo esame) Inviare i voti dell'esame alla segreteria

Se la **segreteria** effettua il login al portale, può:

- Visualizzare i dati personali degli studenti.
- Iscriversi a un corso.
- Pubblicare i voti d'esame per gli studenti.

Se l' **amministratore** accede al portale, può:

- Creare account per studenti/docenti/segreteria utilizzando il loro indirizzo email.
- **Aggiungere/Modificare/Visualizzare** le informazioni personali degli studenti e dei docenti.

2 Introduzione

L'applicativo è stato realizzato utilizzando tutte le risorse descritte nelle specifiche del progetto. Abbiamo inoltre cercato di realizzare un prodotto il cui codice fosse **flessibile** e **mantenibile** per rendere eventuali aggiornamenti più semplici da implementare. É stata prestata molta attenzione all'ottimizzazione della leggibilità del codice attraverso i numerosi commenti che spiegano l'utilizzo di ogni singola classe del programma. Ottimizzare tutti questi aspetti non è stato semplice ma comporta una maggiore rendita e un minor dispendio di risorse ed energie per sviluppi futuri.

3 Scelte progettuali

3.1 Client

Dentro la directory Client possiamo trovare tre directory: model, schermate e services.

- **Model**
  Dentro 'model' ci sono tutte le classi relative agli oggetti del nostro progetto più la cartella Serializer che contiene i serializer degli oggetti: questi ci servono per convertire i nostri oggetti in un "flusso" di byte (metodo serialize) per salvarlo nel DB e tradurre dal DB (metodo deserializer) in oggetto.
  Tutti gli oggetti implementano l'interfaccia "Serializable" (per il motivo spiegato sopra), hanno gli attributi che li descrivono, hanno un costruttore e "i metodi get" per leggere gli attributi, mentre i vari comportamenti degli oggetti sono in Services.

- **Services**
  Dentro la directory Services ci sono le interfacce per ogni classe di model, in queste dichiariamo i metodi. Le interfacce implementano "RemoteService", questa definisce l'API generica per i servizi di accesso agli oggetti remoti, che consentono ai processi remoti di accedere ai servizi, in questo modo possono essere richiamate dalle schermate. I servizi di ogni classe sono due, **Service*Oggetto*** e **Service*Oggetto*Async**: i metodi delle interfacce async saranno le API richiamate dalla schermata, il comportamento dei metodi viene specificato nelle classi dentro la cartella Server.

- **Schermate**
  Per Schermate si intendono i seguenti file: SchermataStudente, SchermataDocente, SchermataSegreteria, SchermataAdmin. Ogni Schermata fa riferimento ad uno dei quattro utenti disponibili e, al momento del Login da parte dell'utilizzatore, verrà caricata la Schermata corretta richiamando il metodo **accesso** che si occupa di generare l'interfaccia grafica composta da *Menu laterale, nav__user* e *Contenitore user__container* nel

quale vengono mostrate le informazioni a seconda della voce selezionata dal Menu laterale.

3.2 Server

Il server è stato strutturato implementando il **Repository pattern**. I repository sono classi o componenti che incapsulano la logica necessaria per accedere all'origine dei dati. Centralizzano la funzionalità di accesso ai dati comuni, migliorando la manutenibilità e il disaccoppiamento dell'infrastruttura o della tecnologia usata per accedere ai database. Inoltre, dividendo la parte d'implementazione dal repository, ogni classe ha una sola resposabilità (Single responsibility principle dei S.O.L.I.D. principles).

- **Repository**
  Le repository si basono sull'implementazione di Interfacce comuni che dichiarano i metodi getAll, getById e le operazioni CRUD (Create, Remove, Update, Delete). Esistono quattro tipi di interfacceRepository, basate sul come identifichiamo l'oggetto nel database: **RepositoryInt, RepositoryString, RepositoryDoubleInt, RepositoryIntString**.
  Esistono due tipi di classe Repository: ***Oggetto*Repository** e ***Oggetto*RepositoryTest** entrambe le due classi implementano l'interfaccia Repository.

  - ***Oggetto*Repository**
    La classe *Oggetto*Repository, che implementa un'interfaccia repository, serve per richiamare e lavorare con dati "veri" del database, viene richiamata durante l'esecuzione del programma. All'interno di ogni classe all'inzio c'è il metodo **getDB** che serve per prendere e inizializzare il Database. Questo metodo viene sempre richiamato dal metodo **createOrOpenDB** il quale si occupa di chiamare il DB e salvarlo in un HTreeMap.

```
private DB getDb(){
    try{
        synchronized (context) {
            DB db = (DB)context.getAttribute( s: "studentiDb");
            if(db == null) {
                db = DBMaker.fileDB("C:\\MapDB\\studente").closeOnJvmShutdown().checksumHeaderBypass().make();
                context.setAttribute( s: "studentiDb", db);
            }
            return db;
        }
    } catch(Exception e){
        return null;
    }
}
```

Figure 1: Metodo getDb

- ***Oggetto*RepositoryTest**
  Le classi *Oggetto*RepositoryTest lavorano dati "fittizi" salvati in degli Array, e viene utilizzato per gli unitTest

- **Implementazione** Le classi *Oggetto*Implementazione contengono tutti i comportamenti dei metody delle classi presenti nella repository 'Services' implementando l'interfaccia *Oggetto*Service. Ogni classe ha due costruttori, uno senza parametri che sarà utilizzato durante l'esecuzione del programma e un altro con un parametro Boolean che verrà chiamato per i test inizializzando i DB con *Oggetto*RepositoryTest.
  Queste classi hanno tutte un metodo chiamato **chiamaDB()**che chiama il DB. In questo metodo utilizziamo il DP **Singleton** per far in modo che venga istanziato una sola volta il DB. Se non abbiamo ancora istanziato il DB controlliamo di non essere nell'ambiente di test: se non lo siamo inizializziamo il DB richiamando la classe *Oggetto*Repository passandogli il ServeletContext.

```
public void chiamaDB(){
    // Singleton è un modello di progettazione creazionale che consente di garantire che una classe abbia una
    // sola istanza, fornendo al tempo stesso un punto di accesso globale a questa istanza.
    if(!singleton){
        if(!test){
            ServletContext context = this.getServletContext();
            repositoryStudenti = new StudentiRepository(context);
            singleton = true;
        }
    }
}
```

Figure 2: Metodo chiamaDB

3.3 Test

Abbiamo fatto gli UnitTest sui metodi delle classi *Oggetto*Implementazione, i test si basano sul testare l'implementazione dei metodi con i dati delle repository *Oggetto*RepositoryTest.

4 Scelte di interfaccia

4.1 Sito

- **Homepage**
  La homepage è la pagina iniziale su cui viene portato l'utilizzatore del sito. A questa pagina possono accedere sia utenti registrati che non. Nella home

sono stati inseriti tutti i dati che sono stati ritenuti necessari per un utente fittizio quali: News ed Eventi organizzati dall'università, Informazioni sul numero di Corsi ed Utenti iscritti al portale, Informazioni sull'università e Informazioni di contatto. Il design presenta elementi ed animazioni moderne che puntano a dare un'immagine all'avanguardia soddisfando anche l'occhio dell'utilizzatore.
Il file attraverso il quale viene creata la pagina è il seguente src/com/university/client/schermate/**Index.java**.
Il metodo che si occupa di ciò è *aggiungiContenuto()*. Esso carica l'html precedentemente scritto all'interno del *VerticalPanel homepage__panel* facendo anche 3 richieste al Server per ricevere il numero di Studenti, Docenti e Corsi presenti nel DB.

- **Login**
  La pagina di login è accessibile tramite la Navbar e solo un utente registrato sarà in possesso delle credenziali per eseguire l'accesso al portale. Ci sono 4 tipologie di utenti in grado di accedere alla piattaforma e sono: Studente, Docente, Segreteria e Admin. Quest'ultimo ha il compito di creare gli account e fornire le credenziali per tutti gli altri utenti.
  Il file attraverso il quale viene creata la pagina è il seguente src/com/university/client/schermate/**University.java**.
  Il metodo che si occupa di ciò è *cacicaLogin()*. Esso crea un FormPanel login_panel attraverso il quale l'user può inserire le proprie credenziali di accesso che verranno inviate al server per la verifica attraverso la pressione del pulsante "Login".

- **Navbar**
  La navbar è un componente dinamico e permette all'utente di spostarsi in modo semplice e guidato all'interno del sito web. Essa è visualizzabile in tutte le pagine del sito, sia dalla homepage che dal portale una volta effettuato il login. Lo stile applicato a questo menu è sempre in chiave moderna così da accentuare sempre di più l'immagine solida che vogliamo collegare all'università.
  Il file attraverso il quale viene creato il componente è il seguente src/com/university/client/schermate/**University.java**.
  Il metodo che si occupa di ciò è *Navbar()*. Esso crea un *HorizontalPanel navbar__panel* che contiene 4 Button utili al raggiungimento di tutte le pagine e sezioni del sito web.

4.2 Portale

- **Menù laterale**
  Il menu laterale è un componente dinamico disponibile solamente una volta eseguito l'accesso al portale tramite un qualsiasi utente, serve per permettergli di visualizzare ed usufruire di tutte le funzionalità del profilo. Questo menu viene generato direttamente all'interno di ogni file

"Schermata": SchermataStudente, SchermataDocente, SchermataSegreteria, SchermataAdmin. Questa navbar laterale è costituita da un *VerticalPanel nav__user* e viene popolata da un numero di *Button* dipendente dalla quantità di funzionalità generali disponibili per ogni utente.

- **CellTable**
  Le tabelle sono elementi dei quali è stato fatto grande uso una volta eseguito il login al portale. Sono state utilizzate per permettere di visualizzare ai vari utenti delle liste di oggetti, come per esempio corsi, esami, studenti e docenti. Queste *CellTable* vengono generate all'interno dei file Schermata e sono state scelte grazie alle loro proprietà e flessibilità in quanto permettono di mostrare in modo semplice e chiaro tutti i dati necessari, riconoscibili grazie al titolo personalizzabile di ogni colonna della griglia.