



A file hosting and sharing application

Studenti:

Oliverio Marco

Rametta Pierpaolo

Gruppo:

Electric Sheep

30 gennaio 2014

Capitolo 1

Descrizione

Deel implementa le funzionalità più comuni ricercate in un servizio di hosting di file quali l'upload di file di varie dimensioni, il download, lo sharing, l'organizzazione gerarchica in cartelle e il versionamento automatico dei file caricati sul sistema, nonché la gestione del proprio account.

1.0.1 Struttura

L'applicazione è stata suddivisa in moduli interdipendenti tra loro, ognuno dei quali si occupa di precise funzionalità chiave. Ogni modulo espone agli altri moduli solo la propria interfaccia, mantenendo privata l'implementazione, così da aumentare la possibilità di riutilizzo e manutenzione del codice. I principali moduli sono mostrati nella figura 1.1.

1.0.2 Domain

Le classi che fanno parte del dominio rappresentano la modellazione del problema svincolata dalle logiche di accesso ai dati, persistenza, navigazione ecc. Le classi sono strutturate come semplici POJO (plain old java object) così da non dipendere da nessun tipo di framework particolare. Le classi si trovano nel package `java org.deel.domain`.

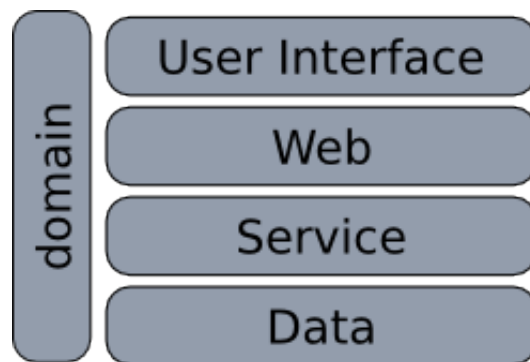


Figura 1.1

1.0.3 Data layer

Si occupa della persistenza dei dati, e di fornire un mapping consistente tra i dati permanenti e i dati manipolati in memoria dall'applicazione. Per salvare i metadati relativi ai file ed i dati relativi agli utenti è stato utilizzato un DBMS, in particolare MySQL. I file veri e propri vengono salvati utilizzando le funzionalità di accesso al filesystem del sistema operativo. Per consentire il mapping tra oggetti del database e oggetti utilizzati dal sistema sono state utilizzate le funzionalità del framework Hibernate. Lo schema E-R dell'applicazione è definito nella figura 1.2.

Per modellare una struttura ad albero rappresentante la struttura organizzativa dei file dell'utente sono state utilizzate le tabelle Folder e FilePath, dove un filepath appartiene ad una sola directory e Folder ha una relazione ricorsiva. Ogni FilePath sarà collegato con un solo File e ciò permette una migliore gestione della condivisione. Per la configurazione di Hibernate sono state utilizzate annotazioni sulle classi del layer Domain. Le relazioni tra le entità sono state mappate in modo bidirezionale, così da consentire la navigazione delle relazioni tra entità senza dover codificare istruzioni sql manualmente.

La strategia di caricamento scelta è stata Lazy, per evitare catastrofici caricamenti ricorsivi di tutto il Database in memoria e eseguire query sql solo quando ve ne sia effettivamente bisogno. Hibernate è stato integrato

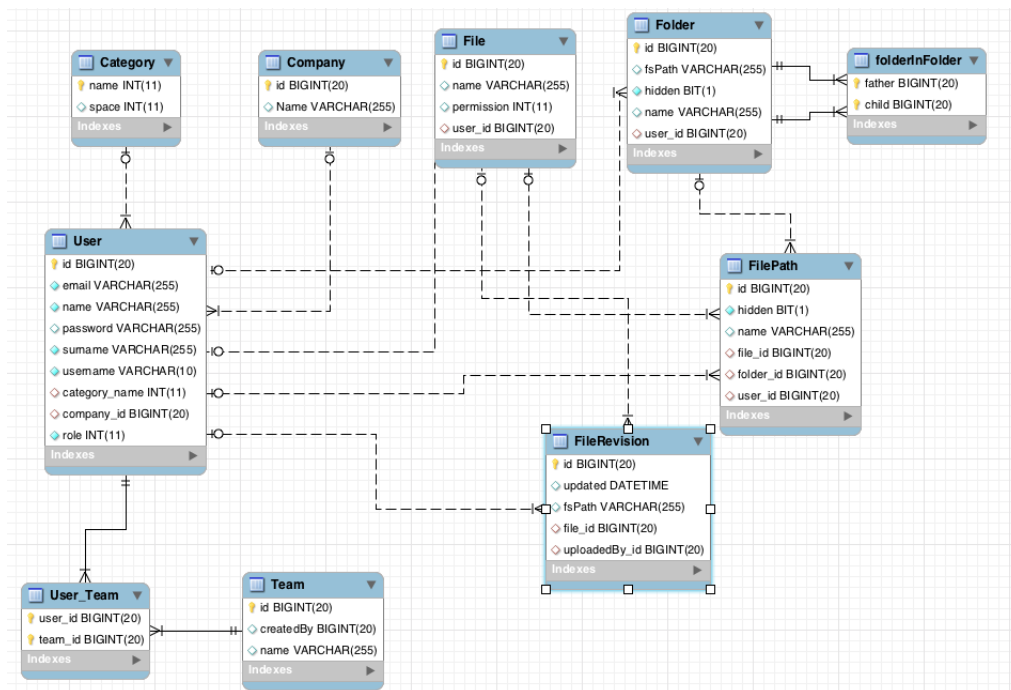


Figura 1.2

nell'applicazione attraverso la configurazione di due bean di spring, in particolare sessionFactory e TransactionManager. Il sessionFactory prevede ad istanziare una sessionFactory di Hibernate e settarla come proprietà dei Data Access Object che andranno poi ad utilizzarla per le comuni operazioni CRUD. Il transaction manager consente invece di definire i boundaries delle transazioni in modo dichiarativo, tramite l'utilizzo dell'annotazione @Transactional. In questo modo il codice è svincolato dal dover far iniziare, committare e rollbackare la transazione in modo esplicito. Il layer Service è stato progettato in modo tale di aver associato un caso d'uso per ogni metodo, rappresentando logicamente una Unit Of Work, che deve preservare, tra le altre, la proprietà di atomicità. Questi infatti sono i metodi che sono stati annotati come @Transactional, in modo che il transaction manager di Spring inizi la transazione al momento della chiamata del metodo, committi al suo completamento o faccia il rollback in caso di errori. Infine per la propagazione delle operazioni sul DB abbiamo deciso, tranne per particolari casi, di non propa-

gare nessuna operazione, così da avere maggiore controllo su inserimenti e rimozioni a livello di business logic.

1.0.4 Service layer

Il service layer si occupa di implementare i principali casi d'uso dell'applicazione, manipolando sia oggetti del domain sia interagendo con il data layer. Il service layer, come il layer web, utilizza la dependency injection del framework Spring. In questo modo l'istanziamento degli oggetti di cui questo layer necessita (ad es. gli oggetti con le funzionalità DAO presenti nel data layer) avviene in maniera automatica e il codice non dipende da essa.

1.0.5 Web Layer

Il web layer si occupa della logica di navigazione e della gestione della sicurezza dell'applicazione.

Per l'implementazione della sicurezza (accesso e autenticazione) è stato utilizzato il modulo Spring Security, che tramite filtri servlet permette un accesso selettivo e autenticato al sistema. Le password vengono salvate nel db utilizzando l'implementazione java di *BCrypt* (le password vengono hashate, saltate e stretchate). Sono stati implementati due diversi role (USER e ADMIN), in modo da avere un'area per la gestione dell'applicazione accessibile solo agli utenti amministratore.

Per quanto riguarda la navigazione sono state utilizzate le funzionalità offerte dal modulo Spring MVC, funzionante su un servlet container (ad es. tomcat). Quando una richiesta arriva all'applicazione essa viene mappata all'opportuno *Controller*. Il mapping viene definito tramite annotazioni sul codice java. Il Controller, nel caso di una richiesta HTML, si occupa di validare l'input (anche questo viene definito grazie ad annotazioni), di chiamare gli opportuni metodi sul layer service, di esporre i dati del modello e di indicare il nome di una vista da visualizzare all'utente. Se invece la richiesta è di tipo JSON, dopo la validazione dell'input e l'interazione con il layer service

il controller ritorna le informazioni richieste al client sotto forma di oggetto JSON. In questo modo abbiamo potuto delegare parte della logica riguardante la User Interface sul lato client, fornendo da una parte una migliore e più reattiva esperienza utente, dall'altra diminuendo in qualche modo il carico di lavoro sul lato server.

1.0.6 User Interface

E' il layer più vicino all'utente e l'unico con cui interagisce direttamente. Una volta che il controller indica il nome di una vista e i dati da esporre, in modo totalmente configurabile questo nome viene mappato ad una tecnologia di visualizzazione. Nel nostro caso abbiamo utilizzato i tag JSTL utilizzati per renderizzare i contenuti da esporre in una pagina html finale. In questo modo è facile aggiungere nuove tecnologie di rendering senza modificare il layer web. Per migliorare l'esperienza utente parte di questa logica è stata portata sul lato client. Del codice javascript in questo caso esegue le richieste in modo asincrono al server grazie ad AJAX, e, tramite un meccanismo di callback, aggiorna dinamicamente il contenuto della pagina. Si è utilizzato jQuery come libreria.

Capitolo 2

Sviluppo

2.0.7 Ambiente, IDE e testing

Come ambiente di sviluppo sono stati utilizzati diversi tool e strategie. Si è proceduto nello sviluppo del software anche seguendo una metodologia di tipo Test Driven. Si è cioè proceduto prima alla creazione di unit test per la funzione da implementare per poi solo in un secondo momento scrivere la reale implementazione. L'utilizzo della dependency injection insieme a l'utilizzo di tool per lo unit testing come *JUnit*, *Spring Test Framework* e *mockito* hanno consentito di scrivere test piccoli, mirati, utili e facili da mettere in piedi.

Come IDE si è scelto di utilizzare *Eclipse*, come tool di building e per la gestione delle dipendenze *Maven*.

2.0.8 Caso d'uso d'esempio

In figura 2.1 è mostrato il diagramma di sequenza semplificato del caso d'uso riguardante l'upload di un file da parte dell'utente. E' possibile fare l'upload di un file trascinandolo direttamente dal Sistema. Una volta droppato il file, parte una richiesta asincrona fatta dal client al server.

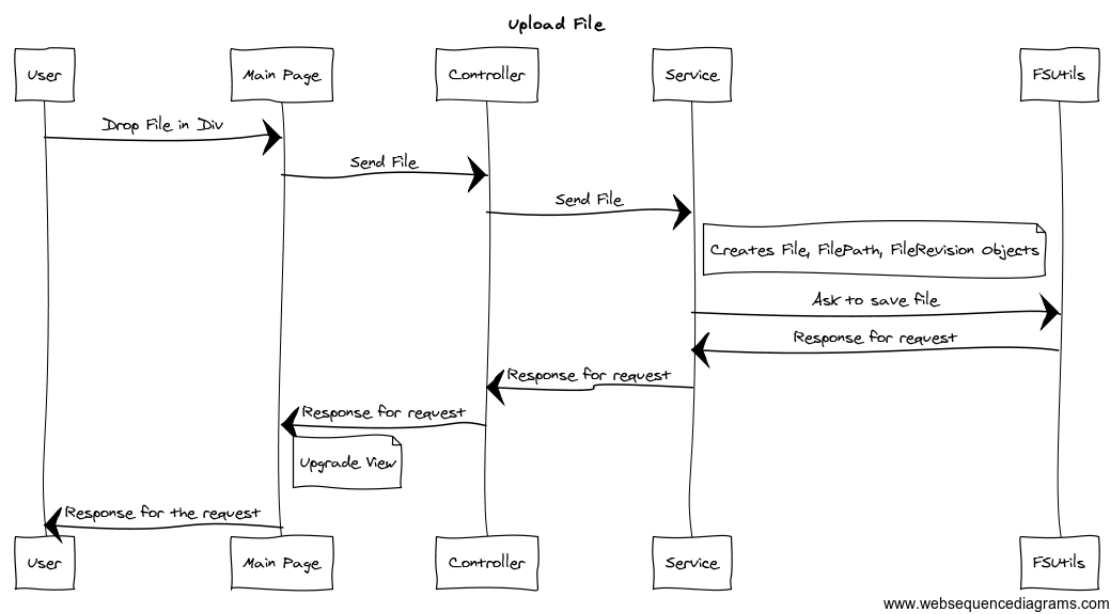


Figura 2.1