
BOOKVERSE

SDD

Design goals

Dependability

Robustness

- Il sistema gestisce gli input non validi. **[HIGH]**

Security

- Il sistema deve garantire l'accesso a determinate aree solo agli utenti autorizzati (e.g Le sezioni di validazione degli ebook non devono essere accedute da utenti sprovvisti di account validator) **[HIGH]**

Reliability

- Il sistema deve garantire l'attendibilità dei servizi messi a disposizione (e.g. Se un ordine viene effettuato con successo bisogna garantire che quell'e-book sia disponibile). [**HIGH**]

End user criteria

Usability

- **Responsive**
  Il sistema sarà dotato di interfaccia grafica responsive in modo da potersi adattare a molteplici dispositivi. **[HIGH]**

- **User-Friendly**
  Il sistema è anche dotato di un'interfaccia grafica user friendly che garantisce all'utente un'esperienza piacevole e intuitiva.**[HIGH]**

Utility

- **Error Notification**
  Il sistema deve, in caso di input errato da parte dell'utente durante la compilazione di un form, evidenziare i campi scorretti e far visualizzare un messaggio testuale che indichi come riempire correttamente il campo. **[HIGH]**

Proposed software architecture

La software architecture che abbiamo scelto è un three-tier.
Questo ci consente di distinguere il livello di presentazione, il livello di business e il livello di storage.
In questo modo saremo capaci di distinguere nettamente quelli che sono gli aspetti riguardanti la presentazione, la realizzazione della logica di servizio e quella di persistenza dei dati.
La nostra applicazione sarà inoltre un applicazione client-server. I vari client(dispositivi degli utenti del sistema) potranno infatti usufruire di quelli che sono i servizi che vengono ospitati sul server.

System decomposition

Procediamo a effettuare la decomposizione in subsystem del nostro sistema, raggruppando tutte le funzionalità che riguardano lo stesso contesto.

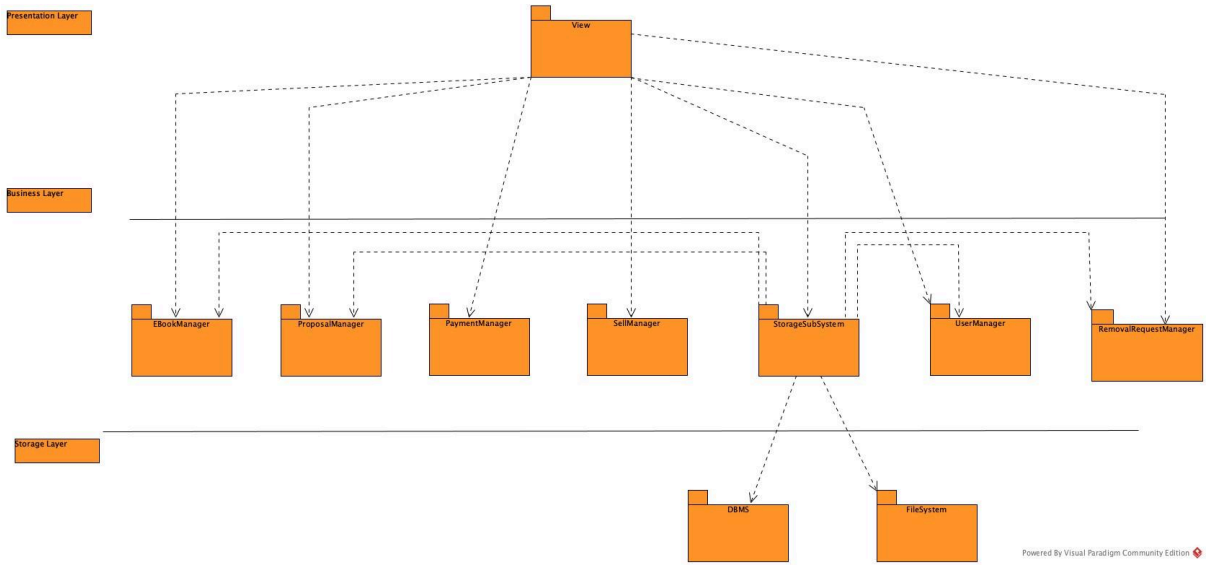
Descriviamo innanzitutto i sottosistemi del livello di presentation.
Il sottosistema **view** contiene tutte quante le pagine del sito e soprattutto le servlet che ci consentono di rispondere alle varie richieste e all'interno delle quali abbiamo implementato la logica di controllo.
La decisione di inserire la logica di controllo direttamente all'interno delle servlet è data dalla semplicità del nostro sistema e della nostra logica di controllo, che non ci porta a fare altro che coordinare l'utilizzo dei servizi offerti dal layer di business.
Il subsystem **view** utilizza i servizi dei subsystem **EBookManager**, **SellManager**, **UserManager**, **RemovalRequestManager** dato che coordina la loro logica di business per realizzare le funzionalità del sistema e che utilizza gli oggetti che rappresentano i dati della nostra applicazione.
view utilizza il servizio di pagamento offerto da **PaymentManager**.
view utilizza anche il servizio di persistenza dei dati offerto da **StorageSubsystem**.

I sottosistemi del layer di business sono i seguenti:

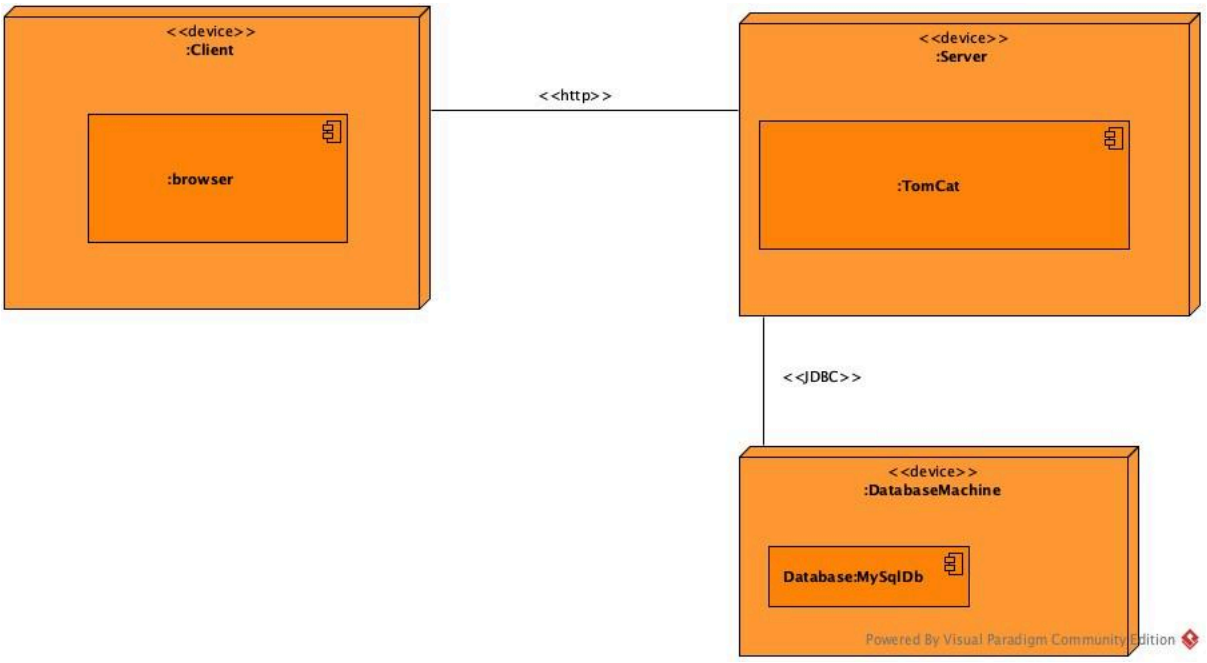
- **EBookSubsystem:** ci permette di gestire gli ebook.
- **ProposalManager:** ci permette di gestire la proposta e il versioning di quest'ultima.
- **SellManager:** contiene le classi necessarie per realizzare tutta quanta la parte d'acquisto all'interno del sito.
- **UserManager:** contiene le classi degli utenti e i suoi ruoli.
- **RemovalRequestManager:** ci permette di gestire gli oggetti necessari alla rimozione di un ebook.
- **PaymentManager**: contiene le classi necessarie per realizzare un pagamento.
- **StorageSubsystem:** Permette di assolvere alla funzionalità di persistenza dei dati.

In particolare lo **StorageSubsystem** utilizza il servizio di persistenza di livello di storage offerto dal **DBMS** e dal **FileSystem**.
Le dipendenze segnalate con gli altri sottostimi riguardano il fatto che le classi presenti all'interno dello storage recuperano i dati e istanziano le classi che sono presenti all'interno dello storage subsystem.

Infine nello Storage layer è presente il **DBMS** e il **FileSystem**.



Mapping Hardware-Software



Persistent Data Management

Procediamo ad individuare tutti quanti i dati/oggetti che devono essere resi persistenti all'interno del sistema:

- User e tutti quanti gli oggetti che rappresentano i ruoli(reader, author, validator, catalogManager).
- RemovalRequest(Le richieste di rimozione).
- EBook(Vengono resi persistenti in maniera da poter formare il catalogo)
- Cart(E' necessario mantenerne la persistenza per riuscire a mantenere la lista dei prodotti che il reader vorrebbe acquistare)
- Proposal, Version, file di report e file di EBook.(Viene reso persistente l'intero storico di una proposal. Validator e Author infatti hanno la possiblità di accedere ai dati della proposal)

Per riuscire a gestire in maniera efficiente le risorse, abbiamo deciso di adottare una modalità di gestione della persistenza ibrida.
Per garantire la persistenza di dati di grandi dimensioni, verrà utilizzato il filesystem(File di report, File che rappresentano le versioni di un ebook, immagini utilizzate come cover del libro).

Durante la fase di requirements elicitation è emerso che non risulta necessario che il carrello rimanga condiviso tra i vari dispositivi di uno stesso account. Di conseguenza la persistenza del carrello sarà garantita attraverso l'utilizzo della sessione. Questo ci permetterà di risparmiare accessi e spazio di storage nel database.

La persistenza del resto dei dati sarà gestita attraverso un database relazionale. La scelta ricade in un database per l'eterogeneità dei dati e per non doversi curare del meccanismo degli accessi concorrenti.
La scelta di un database relazionale permette di recuperare in maniera efficiente e veloce i dati utili a svolgere le funzionalità del sito.

Access control and security

	Guest	**Reader**	**Author**	**Validator**	**Catalogue Manager**
Removal Request			**makeRemovalRequest()**		**end()**
Catalogue Manager			**assignRemovalRequest()**		
Validator			**assignProposal()**		
User	**makeUser(name, surname, email, password)**	**logout()** **login()** **setCurrentRole()**	**logout()** **login()** **setCurrentRole()**	**logout()** **login()** **setCurrentRol**	**logout()** **login()** **setCurrentRole()**

				e()	
Reader		**addToLibrary()**			
Cart	**addToCart()** **RemoveFromCart()** **clearCart()** **isValid()** **totalAmount()**	**addToCart()** **RemoveFromCart()** **isValid()** **clearCart()** **totalAmount()**			
E-Book	**applyFilters(title, genres, min, max)**	**applyFilters()**			**removeFromCatalog()**
Author			**addProposal()** **addEbook()**		
Proposal			**makeProposal(author, coAuthors)** **correct()** **assign(Validator)** **pay()** **lastVersion()** **isValidParameters()**	**approve()** **refuse()** **lastVersion()** **permanentlyRefuse()** **completeProposal()** **addVersion()**	
Version				**makeVersion()**	
Report				**checkReportFormat()** **makeReport()**	
File-EBook			**checkExtension()**		

N.B: Abbiamo volutamente omesso i parametri dalle varie operazioni per una maggior chiarezza, non risulta infatti ad andare a disambiguare sulla base dei parametri passati.

Per riuscire a garantire delle performance migliori e non aggiungere a tutti quanti i metodi della logica cross-cutting, abbiamo deciso di non aggiungere al livello di business il controllo degli accessi.
Abbiamo infatti pensato di realizzare attraverso un interceptor un filtro che ci consenta di andare a mappare una determinata funzionalità del sistema ai ruoli che vi hanno accesso. Dato che una volta ricevuta una richiesta, la domanda alla quale sarà necessario rispondere è: “chi ha accesso a questa funzionalità?” abbiamo deciso di implementare la seguente matrice degli accessi come capability list.

Per riuscire a garantire una maggiore sicurezza, abbiamo deciso che le password saranno salvate all’interno del database in forma cryptata con algoritmo: SHA512.

Global Control Flow

Lato client sarà presente un control flow basato su eventi. Il nostro sistema sarà infatti dotato di un interfaccia grafica essendo esso un applicazione web.
Lato server verrà utilizzato un control flow basato su threads. Quest'ultimo ci consentirà di rispondere a molteplici richieste che arrivano al server contemporaneamente, è infatti un requisito scontato da considerarsi la possiblità di poter utilizzare la piattaforma contemporaneamente da parte di più attori.

Boundary Conditions

Le *boundary conditions* del sistema sono le seguenti:

- **Installazione del sistema:** l'installazione del sistema verrà eseguita da un addetto che ha il compito di riempire la base di dati con gli account dei validator e dei catalogManager. In particolare, il server container Apache Tomcat, sul quale verrà fatto il deployment del sistema, verrà installato su di una macchina remota, per questo ci si affiderà ad un'azienda di web hosting.
- **Avvio del sistema:** il sistema verrà avviato da un addetto, quindi viene attivato il DBMS MySQL, la cui connessione col sistema verrà stabilita mediante driver JDBC.
- **Spegnimento del sistema:** Il sistema dovrà essere spento da un addetto. Prima di fare ciò si deve far attenzione ad eseguire tutte le transazioni iniziate, salvare tutti i dati critici ed infine come ultima cosa andare a chiudere nel modo corretto le varie connessioni con il DB.