**SDD**

[**Design goals 2**](#_yyh1edoeuez5)

[Dependability: 2](#_upul3ubkyfwy)

[End user criteria: 2](#_9xk7l1ks8hfb)

[**System decomposition 2**](#_ht46h7ms5j9k)

[**Mapping Hardware-Software 3**](#_z7kdrwiwtgbg)

[**Persistent Data Management 3**](#_ky2y0y3jqnxa)

[**Access control and security 4**](#_t4dfu0ut7au4)

[**Global Control Flow 5**](#_n4ptip7ui5d5)

[**Boundary Conditions 5**](#_kuww4gfh3v2d)

[**Subsystem Services Definition 6**](#_ps2p1e9otnsx)

# Design goals

## Dependability:

* Il sistema gestisce gli input non validi.

## End user criteria:

* 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.
* Inoltre il sistema sarà dotato di interfaccia grafica responsive in modo da potersi adattare a molteplici dispositivi.
* Il sistema è anche dotato di un'interfaccia grafica user friendly che garantisce all’utente un'esperienza piacevole e intuitiva.

# System decomposition

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

Descriviamo innanzitutto i Subsystem del layer di business:

* **ProposalSubsystem**: All’interno di questo subsystem sono presenti tutte quante le operazioni che permettono di gestire la proposta, sia per l’Author che per il Validator. Sono quindi presenti le seguenti funzionalità: Creazione, Correzione, Rifiuto, Rifiuto definitivo, Approvazione e Pagamento della proposta.
* **PaymentSubsystem**: E’ presente la funzionalità del pagamento da parte dell’utente.
* **ReaderSubsystem**: Permette di assolvere alle funzionalità di gestione del carrello e della libreria del Reader.
* **RemovalRequestSubsystem**: Permette di creare, assegnare e assolvere una richiesta di rimozione.
* **EBookSubsystem**: Permette la ricerca tramite filtri degli EBooks, la creazione, la rimozione dal catalogo.
* **UserSubsystem**: Permette di assolvere a funzionalità che riguardano gli users, come creazione di un account, login, logout e il controllo degli accessi.

Abbiamo infine un unico subsystem che ci permette di gestire la persistenza dei dati:

* **StorageSubsystem:** Permette di assolvere alla funzionalità di persistenza dei dati. In particolare si occupa di gestire il salvataggio su FileSystem oppure su Database dei dati.

# Mapping Hardware-Software

Utilizzeremo una software architecture three tier. Questo ci darà la possibilità di gestire separatamente il livello di presentation, di application e di storage.

Inoltre utilizzeremo una architettura client-server. Volendo realizzare un applicazione dove si effettuano degli acquisti e si effettua la pubblicazione di elementi che saranno visibili a tutti quanti gli altri utenti, risulta essere la scelta migliore.

In particolare come WebServer utilizzeremo TomCat.

Abbiamo inoltre deciso di andare ad utilizzare un DBMS relazionale e non solo il filesystem di TomCat per mantenere la persistenza dei dati(parte del razionale di questa scelta si trova nella sezione Peristent Data Management). Questo ci permetterà infatti di rimanere all’interno dei nostri tempi di sviluppo più facilmente.

AGGIUNGERE DEPLOYMENT DIAGRAM

# 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(E’ necessario andare a salvarli in maniera persistente così da poter mantenere il catalogo degli ebook)
* Cart(E’ necessario salvare il carrello per fare in modo da salvare quelli che sono i prodotti che il reader vorrebbe acquistare)
* Proposal, Version, file di report e file di EBook.(E’ necessario andare a salvare quelle che sono le proposte e i loro versionamenti per arrivare alle pubblicazioni degli EBook)

Per riuscire a gestire in maniera efficiente le risorse, abbiamo deciso di adottare una modalità di gestione della persistenza ibrida.

Sarà utilizzato quindi un filesystem per garantire la persistenza di file di grandi dimensioni(file di ebook e di report, cover dei libri).

Durante la fase di requirements elicitation è emerso che non è necessario che il carrello sia condiviso tra più dispositivi di uno stesso utente. 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()**  **setCurrentRole()** | **logout()**  **login()**  **setCurrentRole()** |
| **Reader** |  | **addToLibrary()** |  |  |  |
| **Cart** | **addToCart()**  **RemoveFromCart()**  **clearCart()**  **isValid()**  **totalAmount()** | **addToCart()**  **RemoveFromCart()**  **isValid()**  **clearCart()**  **totalAmount()** |  |  |  |
| **E-Book** | **applyFilters(title, genres, min, max)** | **applyFilters()** |  |  | **removeFromCatalog()** |
| **Author** |  |  | **addProposal()** |  |  |
| **Proposal** |  |  | **makeProposal(author, coAuthors)**  **correct()**  **assign(Validator)**  **pay()**  **lastVersion()**  **isValidParameters()** | **approve() refuse()**  **lastVersion() permanentlyRefuse()**  **completeProposal()**  **addVersion()** |  |
| **Version** |  |  |  | **makeVersion()** |  |
| **Report** |  |  |  | **checkReportFormat()**  **makeReport()** |  |
| **File-Book** |  |  | **checkExtension()** |  |  |

N.B: Abbiamo volutamente omesso i parametri dalle varie operazioni per una maggior chiarezza, nessun firma di un metodo viene infatti disambiguata a partire dai parametri passi.

Dato il numero di attori molto ridotto che partecipano al sistema(5, considerando anche il Guest) probabilmente la migliore scelta per andare a rappresentare la access matrix è l’access control list.

Quest’ultima risulta essere una buona scelta rispetto a utilizzare la global access table oppure la capability. Per il semplice fatto che nella global access table avremmo un sacco di celle vuote, mentre nella capability dovremmo andare per ciascun attore a effettuare la verifica su un numero consistente di coppie(classe, operazione).

Per riuscire a soddisfare al meglio i design goals di sicurezza, abbiamo deciso che le password che vengono conservate e trasmesse nel sistema saranno cryptate con l’algoritmo:...............

# Global Control Flow

Lato client utilizzeremo un control flow basato su eventi, dato che il nostro sistema sarà dotato di un interfaccia grafica.

Lato server invece utilizzeremo un control flow basato su threads. Quest’ultimo ci darà la possibilità di creare un thread per assolvere a ciascuna delle richieste che vengono effettuate al server. Questo ci permetterà di gestire più richieste contemporaneamente e sopratutto di farlo nella maniera più efficiente possibile.

# 

# 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.

# Subsystem Services Definition

Andiamo ora a definire quelli che sono i servizi offerti da ciascun subsystem precedentemente individuato.

* **EBookManager:**
  + **EBookHandler:** ci permette di andare a creare e utilizzare un oggetto della classe EBook.
  + **Search:** ci permette di effettuare una ricerca tramite filtri degli EBook.
* **ProposalManager:**
  + **ProposalApplication:** ci permette di effettuare avanzamento, correzione e pagamento di una proposta.
  + **ProposalElaboration:** ci permette di effettuare rifiuto(definitivo e non) e approvazione di una proposta.
* **PaymentManager:**
  + **Payment:** servizio che ci permette di effettuare un pagamento.
* **ReaderManager:**
  + **CartHandler:** ci permette di eseguire tutte quante le operazioni che riguardano il carrello(aggiunta, rimozione).
  + **Library:** servizio che ci permette di effettuare il download di un ebook e di visualizzare la libreria.
  + **BuyCart:** ci permette di effettuare un acquisto dal carrello.
* **UserManager:**
  + **Authentication:** il servizio permette di effettuare la registrazione, l’autenticazione e il controllo degli accessi.
* **RemovalRequestManager:**
  + **RemovalAdvancement:** il servizio ci permette di effettuare una richiesta di rimozione per un determinato EBook.
  + **EBookRemoval:** il servizio ci permette di effettuare la rimozione di un ebook sulla base di una richiesta di rimozione.