
BOOKVERSE

ODD

1. Introduzione

In tale documento, così come nei documenti del testing, sarà presente solamente documentazione relativa al primo incremento dello sviluppo del sistema. La maggior parte dei requisiti funzionali che vengono realizzati riguardano la pubblicazione della proposta.

1.1 Object design trade-offs

1.1.1 Robustezza vs Velocità di Implementazione

Nel contesto della gestione dei dati in ingresso, attribuiamo notevole importanza al controllo di tali dati. Tuttavia, effettuare questa operazione nel modo più accurato e completo possibile richiederebbe un investimento temporale superiore rispetto a quanto attualmente disponibile per il rilascio della prima versione, completamente funzionante. In considerazione di ciò, affermiamo una scelta che privilegia la tempestività di rilascio a discapito della robustezza iniziale. È importante notare che i controlli sui dati in ingresso saranno oggetto di attenzione e completa implementazione nelle versioni successive del sistema.

1.1.2 Chiarezza vs Velocità di implementazione

Per riuscire a facilitare il più possibile la fase di testing, dovremmo scrivere del codice il più possibile chiaro. Tuttavia, dati i tempi ristretti di sviluppo per questo primo incremento, non sempre potremo mantenere alti i nostri standard.
Nelle versioni successive del sistema, sarà possibile correggere questo aspetto.

1.2 Guidelines and Conventions

1. Le classi hanno dei nomi comuni singolari.
2. I metodi sono denominati con frasi verbali, i campi e i parametri con frasi nominali.
3. Gli Error Status sono restituiti attraverso eccezioni.

1.3 References

- SDD: ci si riferisce al SDD quando si spiega l'organizzazione dei package, dato che quest'ultima è stata creata proprio a partire dalla suddivisione in subsystem.

2 Directories

src è la directory dove è presente tutto il codice del sistema.

All'interno del progetto, nella directory src, sono presenti le seguenti directories:

```
Unset
BookVerse/
└── src
   └── main
    ├── java
    │  ├── PaymenManager
    │  ├── com
    │  │  └── bookverse
    │  │    └── bookverse
    │  │      └── sessionConstants
    │  ├── ebookManager
    │  ├── proposalManager
    │  ├── storageSubSystem
    │  ├── testing
    │  ├── userManager
    │  └── view
    │    ├── author
    │    ├── user
    │    └── validator
    └── webapp
      ├── META-INF
      ├── WEB-INF
      ├── assets
      │  ├── css
      │  ├── images
      │  └── js
      └── templates
```

main > java >PaymentManager

- Bank.java
- BankAdapter.java
- PaymentService.java

main > java > ebookManager

- EBook.java

main > java > storageSubSystem

- AuthorDAO.java
- EBookDAO.java
- FileDAO.java

- GenreDAO.java
- InvalidParameterException.java
- ProposalDAO.java
- UserDAO.java
- ValidatorDAO.java

main > java > userManager

- Author.java
- User.java
- Validator.java
- ValidatorDispatcher.java

main > java > proposalManager

- Proposal.java
- Report.java
- Version.java
- WrongStatusException.java

main > java > testing

Sotto tale directory sono presenti tutti i file per poter inizializzare il database prima di ogni test case durante le attività di testing

- DbResetForFunctionalTesting.java
- RetrieveCredentials.java
- SQLScript.java

main > java > view

- FileDownload.java
- Home.java
- ServletUtils.java

main > java > view > author

- MyJobs.java
- PayProposal.java
- ProposalCorrection.java
- ProposalCreation.java
- Publications.java

- SearchAuthor.java

main > java >view > user

- login.java
- logout.java

main > java >view > validator

- ApproveProposal.java
- PermanentlyRefuse.java
- Proposals.java
- RefuseProposal.java

main > java >com > bookverse > bookverse

- AccessControlFilter.java
- ContextInitializer.java

main > java > com > bookverse > bookverse > sessionConstants

Sotto tale directory è presente un unico file “SessionCostants” che contiene delle costanti che vengono usate per recuperare oggetti dalla sessione.

- SessionCostants.java

main > webapp > META-INF

Tale directory contiene informazioni di metadati sull'applicazione

- context.xml

main > webapp > WEB-INF

- web.xml

main > webapp > assets > css

In css sono presenti tutti i file css per lo stile.

- home.css
- login.css
- navbar.css
- payment.css
- proposal.css
- registrazione.css
- underNavbar.css
- requisiti.css

main > webapp > assets > images

In images sono presenti tutte le immagini del sito web

- bigCheck.png
- bigWrong.png
- buy.png
- cart.png
- downArrow.png
- exit-cross.png
- greenCheck.png
- logo.png
- logout.png
- pdfUploaded.png
- pngUploaded.png
- publish.jpg
- redCheck.png
- upload.png
- x-mark.png

main > webapp > assets > js

In js sono presenti tutti i file javascript per controlli lato client, e chiamate asincrone.

- fileUpload.js
- jquery-3.7.1.min.js
- login.js
- payment.js
- proposals.js
- searchAuthor.js

main > webapp

In webApp sono presenti tutte le jsp.

- confirmationPage.jsp
- correctProposalForm.jsp
- history.jsp
- historyv.jsp
- home.jsp
- homeAuthor.jsp
- homeValidator.jsp
- login.jsp
- myJobs.jsp
- payment.jsp
- proposalForm.jsp
- proposalFormTest.jsp
- proposals.jsp
- publications.jsp
- refuseProposalFormTest.jsp
- register.jsp
- requisiti.jsp

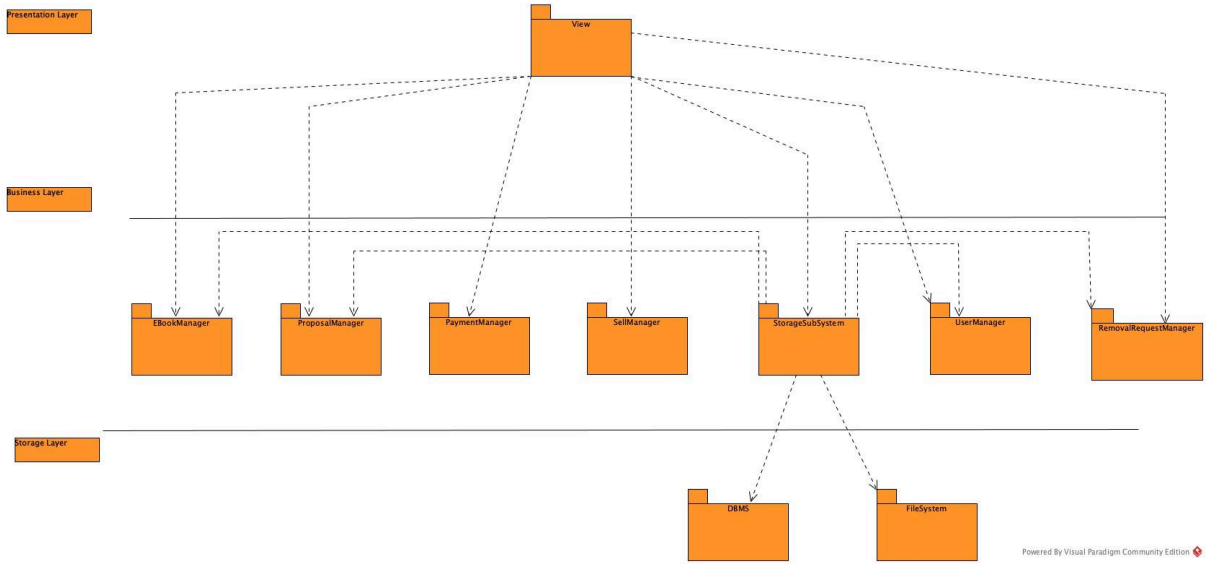
main > webapp > templates

In templates sono presenti le jsp generiche, ovvero la errorPage, e le navbar per ogni tipologia di utente.

- errorPage.jsp
- navbar.jsp
- navbarLogged.jsp
- underNavbarAuthor.jsp
- underNavbarValidator.jsp

3. Packages

Partiamo con un overview, il sistema è stato suddiviso nei seguenti subsystems.



A partire da questi subsystem sono stati quindi prodotti i seguenti packages:

1. com.bookverse
2. ebookManager
3. PaymentManager
4. proposalManager
5. userManager
6. storageManager
7. view

1. All'interno di com.bookverse sono presenti tutte quelle componenti che sono necessarie per il funzionamento del sito, non collocabili in altri package. All'interno di quest'ultimo è presente un package che ospita le costanti della sessione. Sono presenti inoltre le classi che ci consentono di effettuare il controllo degli accessi e una classe che ci consente di assolvere una serie di funzionalità allo startup dell'applicazione.

2. All'interno di ebookManager è presente la classe rappresentativa di un ebook. Quest'ultimo è direttamente mappabile al sottosistema ebookManager.

3. All'interno di PaymentManager è presente tutto ciò che è necessario per realizzare un pagamento. Quest'ultimo è direttamente mappabile al sottosistema PaymentManager.

4. All'interno di proposalManager è presente tutto ciò che ci consente di rappresentare una proposta. Quest'ultimo è direttamente mappabile al sottosistema proposalManager.

5. All'interno di userManager sono presenti le classi che ci consentono di rappresentare le informazioni sull'utente. Quest'ultimo è direttamente mappabile al sottosistema userManager.

6. All'interno di storageManager sono presenti tutte quante le classi necessarie per riuscire a garantire la persistenza dei dati. Quest'ultimo è direttamente mappabile al sottosistema storageManager. Il seguente package possiede delle dipendenze con i package ebookManager, proposalManager, userManager in quanto utilizza le classi che rappresentano i dati per riuscire a garantire la persistenza.

7. All'interno di view sono presenti tutti quanti i package che contengono le servlet e ci consentono di realizzare la logica di controllo. Le servlet sono distribuite all'interno di ciascun package sulla base del ruolo di cui ci permettono di realizzare le funzionalità. Il seguente package, ha una dipendenza con tutti quanti gli altri package dato che realizzando la logica di controllo invoca metodi sulle classi che rappresentano i dati e permettono di effettuare la persistenza dei dati.

Quest'ultimo è mappabile con parte del sottosistema view. Tuttavia, le pagine jsp e i vari asset che vengono utilizzati per realizzare la user interface della nostra applicazione web non sono presenti all'interno di questo package.

4. Specifica delle interfacce

Durante la specifica delle interfacce abbiamo stabilito di non andare a specificare tutti quanti i getter e setter banali che sono presenti all'interno delle classi entity.
Non abbiamo poi effettuato la specifica delle interfacce per tutti quanti i metodi privati delle classi.
Tutte quante le classi che terminano con DAO sono le sole e uniche classi che procedono ad accedere al database.
Laddove non specificato si suppone di utilizzare una tecnica di lazy-evaluation per quanto concerne il caricamento di insiemi di riferimenti di oggetti.
Alcune condizioni che erano particolarmente difficili da esprimere in OCL sono state espresse attraverso il linguaggio naturale.

Proposal	
Descrizione	Classe che rappresenta la proposta all'interno del sistema.
Invariant	**context** Proposal **inv**: (currentStatus() = "refused" || currentStatus() = "permanentlyRefused" || currentStatus() = "approved" || currentStatus() = "pending" || currentStatus() = "completed")

	& proposedBy <> null & getVersions().size() >= 0
makeProposal **static**	Metodo statico che ci consente di creare un oggetto proposal con i campi indicati. **context** Proposal::makeProposal(mainAuthor : Author, coAuthors : Set<Author>) **pre**: ! coAuthors.includes(mainAuthor) & mainAuthor <> null **context** Proposal::makeProposal(mainAuthor : Author, coAuthors : Collection<Author>) **post**: result.oclIsNew() & result.proposedBy = mainAuthor & result.collaborators = coAuthors & result.currentStatus() = “Pending”
approve	Metodo che ci consente di cambiare lo stato di una proposta in “Approved” se si trovava nello stato “Pending”. **context** Proposal::approve() **pre**: currentStatus() = “Pending” **context** Proposal::approve() **post**: currentStatus() = “Approved”
refuse	Metodo che ci consente di cambiare lo stato di una proposta in “Refused” se si trovava nello stato “Pending”. **context** Proposal::refuse() **pre**: currentStatus() = “Pending” **context** Proposal::refuse() **post**: currentStatus() = “Refused”
correct	Metodo che ci consente di cambiare lo stato di una proposta in “Pending” se si trovava nello stato “Refused”. **context** Proposal::correct() **pre**: currentStatus() = “refused” **context** Proposal::correct() **post**: currentStatus() = “pending”
permanentlyRefuse	Metodo che ci consente di cambiare lo stato di una proposta in “PermanentlyRefused” se si trovava nello stato “Pending”. **context** Proposal::permanentlyRefuse() **pre**: currentStatus() = “pending” **context** Proposal::permanentlyRefuse() **post**: currentStatus() = “permanentlyRefused”
pay	Metodo che ci consente di cambiare lo stato di una proposta in “Completed” se si trovava nello stato “Approved”.

	context Proposal::pay() **pre**: currentStatus() = “approved” **context** Proposal::pay() **post**: currentStatus() = “completed”
assignValidator	Metodo che ci consente di assegnare un validator a una proposta che ancora non ne possedeva uno. **context** Proposal::assignValidator(validator : Validator) **pre**: currentStatus() = “Pending” & getAssignedValidator() = null & validator <> null **context** Proposal::assignValidator(validator : Validator) **post**: getAssignedValidator() = validator
isValidParameters static	Il seguente metodo statico restituisce true nel caso in cui i parametri siano validi per creare una nuova versione di una proposta, false nel caso contrario. **context** Proposal::isValidParameters(title : String, genres : Collection<String>, price : Float, description : String) **pre**: – **context** Proposal::isValidParameters(title : String, genres : Collection<String>, price : Float, description : String) **post**: result = (title <> null & title <> “” & title è lungo massimo 30 caratteri & description <> null & description <> “” & description è lungo massimo 500 caratteri & price >= 0 & price <= 500 & genres.size() >= 1)
addVersion	Metodo che permette di aggiungere una versione a una proposta, come ultima versione. **context** Proposal::addVersion(version : Version) **pre**: (currentStatus() = “Pending” || currentStatus() = “Refused”) & version <> null & version.getDate().isAfter(lastVersion().getDate()) **context** Proposal::addVersion(version : Version) **post**: @pre.getVersions().size() = getVersions().size() + 1 & getVersions().includes(version) & getVersions().indexOf(version) = 0
lastVersion	Metodo che permette di recuperare le versioni a partire dalla proposal

	context Proposal::lastVersion() **pre**: getVersions().size() > 0 **context** Proposal::lastVersion() **post**: getVersions().indexOf(result) = 0
currentStatus	Metodo che permette di recuperare lo stato corrente di una proposal **context** Proposal::currentStatus() **pre**: **–** **context** Proposal::currentStatus() **post**: result = self.status

Version

Descrizione	Classe che rappresenta la versione di una determinata proposta
Invariant	**context** Version **inv**: getGenres.size() >= 1
makeVersion static	Metodo che ci consente di creare una versione con i parametri indicati **context** Version::makeVersion(title : String, genres : Set<String>, ebookFile : File, fileReport : File, coverImage : File, price : Float, description : String, date : LocalDate) **pre**: title <> null & title <> “” & title.size() <= 30 & author <> null & genres.size() >= 1 & price >= 0 & price <= 500 & description <> null & description.size() <= 500 **context** Version::makeVersion(title : String, genres : Collection<String>, ebookFile : File, coverImage : File, price : Float, description : String, date : Date) **post**: result.oclIsNew() & result.getTitle() = title & result.getGenres()->forAll(g | genres.includes(g)) & result.getGenres().size() = genres.size() & result.getPrice = price & result.getDescription() = description & result.getDate() = date
addReport	Metodo che ci consente di aggiungere un report a una versione pre-esistente **context** Version::addReport(report : File) **pre**: checkReportFormat(report)

	context Version::addReport(report : File) **post**: getReport() = report

Report	
Descrizione	Classe con metodi statici necessaria per verificare l'estensione di un file di report
checkReport static	Metodo che ci consente di effettuare un check rispetto al formato del report **context** Report::checkReportFormat(fileReport : File) **pre**: – **context** Report::checkReportFormat(fileReport : File) **post:** result = (fileReport <> null & fileReport deve terminare con ".pdf")

FileEbook	
Descrizione	Classe con solo un metodo statico, che serve per controllare se il file dell'ebook ha la giusta estensione per il sistema.
checkFile static	Metodo che ci consente di effettuare un check rispetto al file dell'ebook **Context** FileEbook ::checkFile(fileEbook : File) **pre** – **Context** FileEbook ::checkFile(fileEbook : File) **post**: result = (fileEbook <> null & fileEbook deve terminare con ".pdf")

BankAdapter	
Descrizione	Classe che viene utilizzata per effettuare i pagamenti verso la banca. Quest'ultima implementa il pattern adapter, il sistema non deve infatti dipendere dalle api della banca.

checkDataFormat static	Metodo che restituisce true se i campi inseriti sono validi per un pagamento e altrimenti restituisce false **context** BankAdaper::checkDataFormat(cvv : String, scadenza : LocalDate, intestatario : String, numeroCarta : String, price : int) **pre:** – **context** BankAdaper::checkDataFormat(cvv: String, scadenza : LocalDate, intestatario : String, numeroCarta : String, price : int) **post:** result = (cvv <> null & cvv.lenght() = 3 & cvv rispetta il formato *** (soli numeri) & price >= 0 & scadenza <> null & scadenza successiva alla data corrente & intestatario <> null & intestatario <> “” & numeroCarta <> null & numeroCarta rispetta il formato **** **** **** **** (soli numeri))
pay	Metodo che consente di effettuare un pagamento verso la banca, restituisce true se il pagamento va a buon fine, false nel caso contrario. **context** BankAdaper **::** pay(cvv: String, scadenza : Date, intestatario : String, numeroCarta : String) **pre:** checkDataFormat(self) = true **context** BankAdaper **::** pay(cvv: String, scadenza : Date, intestatario : String, numeroCarta : String) **post:** result = esito pagamento

User	
Descrizione	La classe User rappresenta un utente generico registrato all’interno del sistema.
makeUser static	Metodo che ci consente di creare un utente a partire dai parametri passati al metodo. **context** User::makeUser(id : int, name : String, surname : String, email : String, password : String) **pre**: id > 0 & name <> null & name <> “” & surname <> null & surname <> “”

	& email <> null & email <> “” & password <> “” & password <> null **context** User::makeUser(id : int, name : String, surname : String, email : String, password : String) **post**: result.oclIsNew() & result.name = name & result.surname = surname & result.password = password & result.email = email & result.id = id **context** User::makeUser(id : int, name : String, surname : String, email : String) **pre**: id > 0 & name <> null & name <> “” & surname <> null & surname <> “” & email <> null & email <> “” **context** User::makeUser(id : int, name : String, surname : String, email : String) **post**: result.oclIsNew() & result.name = name & result.surname = surname & result.email = email & result.id = id

Author

Descrizione	La classe Author rappresenta un utente reader registrato all’interno del sistema.
	Nessun metodo in particolare da documentare

Validator

Descrizione	La classe Validator rappresenta un utente validator registrato all’interno del sistema.
assignProposal	Metodo che ci consente di assegnare una proposal a un validator

	context Validator::assignProposal(proposal : Proposal) **pre** proposal <> null **context** Validator::assignProposal(proposal : Proposal) **post** getAssignedProposal.includes(proposal) & getAssignedProposal.size() = @Pre.getAssignedProposal.size() + 1

ValidatorDispatcher Interface	
Descrizione	La seguente interfaccia definisce un metodo che permette di individuare un validator a cui assegnare una proposal nel sistema.
findFreeValidator	Metodo che ci consente di individuare un validator secondo una euristica definita dall'implementazione. **context** ValidatorDispatcher::findFreeValidator(mainAuthor : Author, coAuthors : Set<Author>) **pre**: mainAuthor <> null & coAuthors <> null & mainAuthor.getId() > 0 & coAuthors->forAll(c | AuthorDAO.findAuthorById(c.getId()) <> null) & findAuthorById(mainAuthor.getId()) <> null **context** ValidatorDispatcher::findFreeValidator(mainAuthor : Author, coAuthors : Set<Author>) **post**: result = validator esistente nel database

E-book	
Descrizione	La classe E-book rappresenta un ebook nel sistema.
Invariants	**context** Ebook **inv:** isInCatalog <> null
makeEbook static	Il seguente metodo ci consente di creare un oggetto ebook copiando i campi dalla proposta(una sorta di copy-constructor). **context** Ebook :: makeEBook(proposal : Proposal) **pre**

	proposal <> null proposal.lastVersion() <> null & proposal.lastVersion().getTitle() <> null & proposal.lastVersion().getTitle() <> “” & proposal.lastVersion().getPrice() >= 0 & proposal.lastVersion().getDescription() <> null & proposal.lastVersion().getDescription() <> “” & proposal.lastVersion().getGenres().size() > 0 **context** Ebook :: makeEBook(proposal : Proposal) **post** self.ocIsNew() & getProposedThrough() = proposal & result.title = proposal.getTitle() & result.description = proposal.getDescription() & result.price = proposal.getPrice() & result.isInCatalog = true & result.ebookFile = proposal.getEbookFile() & result.coverImage = proposal.getCoverImage()

UserDAO

Descrizione	Rappresenta la classe dao relativa allo user, e consente di gestirne la persistenza.
login	Il seguente metodo viene utilizzato per tentare di effettuare il login all’interno del sistema. **context** UserDAO::login(email : String, password : String, role : String) **pre:** email <> null & password <> null & email <> “” & password <> “” & role <> null & (role = “Validator” || role = “Author” || role =”Reader” || role = “CatalogManager”) **context** UserDAO::login(email : String, password : String, role : String) **post:** Se email e password corrispondono a un utente presente nel database e il ruolo indicato è posseduto dall’utente allora: result.email = email & result.password = password & result.currentRole = role, con gli altri campi che corrispondono a quelli della entry sul database altrimenti: result = null

AuthorDAO

Descrizione	Rappresenta la classe dao relativa all' author, e consente di gestirne la persistenza.
findAuthorById	Ci consente di recuperare un autore tramite il suo id. **context** AuthorDAO::findAuthorById(int id) **pre**: id > 0 **context** AuthorDAO::findAuthorById(int id) **post**: Se esiste un autore con l'id specificato, allora il result è un l'Author individuato sul db Se non esiste allora result = null
findAuthorsByEmail	Ci consente di recuperare tutti quanti gli autori la cui email inizia per il parametro passato al metodo. **context** AuthorDAO :: findAuthorsByEmail(email : String) **pre:** email <> null & email <> “” **context** AuthorDAO :: findAuthorsByEmail(email : String) **post:** result = un set con tutti quanti gli autori individuati sul db la cui mail inizia per email
findCoAuthorsForProposal	Ci consente di recuperare tutti quanti i co-autori di una determinata proposta. **context** AuthorDAO :: findCoAuthorsForProposal(proposal : Proposal) **pre:** proposal <> null & proposal.getId() > 0 & ProposalDAO.findById(proposal.getId()) <> null **context** AuthorDAO :: findCoAuthorsForProposal(proposal : Proposal) **post:** result = un set con tutti quanti gli autori che sono co-autori della proposal passata come parametro
findMainAuthorForProposal	Ci consente di recuperare l'autore principale di una proposta. **context** AuthorDAO::findMainAuthorForProposal(proposal : Proposal) **pre**: proposal <> null & proposal.getId() > 0 & ProposalDAO.findById(proposal.getId()) <> null **context** AuthorDAO::findMainAuthorForProposal(proposal : Proposal) **post**: result = author che ha effettuato la proposta(l'author principale)

ValidatorDAO

Descrizione	Rappresenta la classe dao relativa al validator, e consente di gestirne la persistenza, in questo primo incremento implementa il validatorDispatcher ereditandone quindi i contratti.

findFreeValidator	Ci consente di individuare un validator libero all'interno del nostro sistema. **context** ValidatorDAO::findFreeValidator(mainAuthor : Author, coAuthors : Set<Author>) **pre**: – **context** ValidatorDAO::findFreeValidator(mainAuthor : Author, coAuthors : Set<Author>) **post**: result = validator individuato secondo una certa euristica
findValidatorById	Ci consente di recuperare i dati di un validator tramite il suo id. **context** ValidatorDAO::findValidatorById(validatorId : int) **pre**: validatorId > 0 **context** ValidatorDAO::findValidatorById(validatorId : int) **post**: se esiste un validator con il seguente id allora result è uguale al validator con quell'id altrimenti il result = null

ProposalDAO

Descrizione	Rappresenta la classe dao relativa alla proposal, e consente di gestirne la persistenza.
newProposal	Permette di rendere persistente una nuova proposta sul database, restituisce l'identificativo assegnato alla proposta. Crea anche la relazione tra autori e co-autori all'interno del database. **context** ProposalDAO**::**newProposal(proposal : Proposal) **pre**: proposal <> null & Proposal.isValidParameters(proposal) **context** ProposalDAO**::**newProposal(proposal : Proposal) **post**: findById(proposal.getId()) = proposal & result = proposal.getId() & relazioni create per riuscire a rappresentare quali sono gli autori della proposta
findByCoAuthor	Restituisce tutte quante le proposte che dove tra i co-autori è presente il co-autore identificato dall'id passato al metodo. Vengono recuperate anche tutte quante le versioni relative alla proposta. **context** ProposalDAO::findByCoAuthor(coAuthorId : int) **pre**: coAuthorId > 0 & findAuthorById(coAuthorId) <> null **context** ProposalDAO::findByCoAuthor(coAuthorId : int) **post**: result = set di proposal dove l'autore identificato da coAuthorId è co-autore della proposal

findByMainAuthor	Restituisce tutte quante le proposte dove l'autore principale possiede l'identificativo passato al metodo. **context** ProposalDAO::findByMainAuthor(mainAuthorId : int) **pre**: mainAuthorId > 0 & findAuthorById(mainAuthorId) <> null **context** ProposalDAO::findByMainAuthor(mainAuthorId : int) **post**: result = set di proposal dove l'autore identificato dal seguente id è l'autore principale
findByValidator	Restituisce tutte quante le proposte a cui è assegnato un validator che possiede il seguente identificativo. **context** ProposalDAO::findByValidator(validatorId : int) **pre**: validatorId > 0 & findValidatorById(validatorId) <> null **context** ProposalDAO::findByValidator(validatorId : int) **post**: result = set proposal che sono assegnate al validator identificato da validatorId
newVersion	Ci consente di rendere persistente sul database una nuova versione di una proposta pre-esistente, restituisce l'identificativo generato per la nuova versione. **context** ProposalDAO::newVersion(proposal : Proposal, version : Version) **pre**: proposal <> null & version <> null & proposal.getId() > 0 & findById(proposal.getId()) <> null **context** ProposalDAO::newVersion(proposal : Proposal, version : Version) **post**: findVersionById(version.getId()) <> null
updateVersion	Ci consente di effettuare l'update di una versione pre-esistente. **context** ProposalDAO::updateVersion(version : Version) **pre**: version <> null & version.getId() > 0 & findVersionById(version.getId()) <> null **context** ProposalDAO::updateVersion(version : Version) **post**: findVersionById(version.getId()) = version
assignProposal	Ci consente di rendere persistente sul database l'assegnamento di una proposta a un validator. **context** ProposalDAO::assignProposal(proposal : Proposal, validator : Validator) **pre**: proposal <> null & proposal.getId() > 0 & findById(proposal.getId()) <> null & validator <> null & validator.getId() > 0 & ValidatorDAO.findById(validator.getId()) <> null

	context ProposalDAO::assignProposal(proposal : Proposal, validator : Validator) **post**: viene creata l'associazione sul database tra proposal e validator
findById	Restituisce la proposta identificata dall'id recuperando anche tutte quante le versioni associate alla proposta. **context** ProposalDAO::findById(id : int) **pre**: id > 0 **context** ProposalDAO::findById(id : int) **post**: Se esiste una proposta con il seguente id, allora result è uguale a quella proposta Altrimenti result = null
findVersionById	Restituisce la versione identificata dal seguente id. **context** ProposalDAO::findVersionById(versionId : int) **pre**: versionId > 0 **context** ProposalDAO::findVersionById(versionId : int) **post**: Se esiste una versione con il seguente id, allora result è uguale alla versione identificata da versionId. Altrimenti result = null
updateProposalState	Il metodo ci consente di effettuare il cambio dello stato sul database di una proposta. **context** ProposalDAO::updateProposalState(proposal : Proposal) **pre**: proposal <> null & proposal.getId() > 0 & findById(proposal) <> null & controlla che si possa effettuare il passaggio dallo stato della proposta sul database a quello presente nell'oggetto proposal **context** ProposalDAO::updateProposalState(proposal : Proposal) **post**: findById(proposal.getId()).getCurrentState() = proposal.getCurrentState()

EbookDAO

Descrizione	Rappresenta la classe dao relativa all'ebook, e consente di gestirne la persistenza.
newEbook	Ci consente di rendere persistente un nuovo ebook sul database. **context** EbookDAO:: newEbook(ebook : EBook) **pre:** ebook <> null & Proposal.isValidParameters(ebook.getTitle(), ebook.getGenres(), ebook.getDescription(), ebook.getPrice()) & ebook.getCoAuthors() <> null & ebook.getMainAuthor() <> null & AuthorDAO.findAuthorById(ebook.getMainAuthor().getId()) <> null

	& ebook.getCoAuthors()->forAll(c | AuthorDAO.findAuthorById(c.getId())) <> null **context** EbookDAO:: newEbook(ebook : EBook) **post:** findById(ebook.getId()) = ebook & relazioni create tra tabella co-autori e tabella ebook
findById	Ci consente di individuare l'ebook identificato dall'id passato come parametro. **context** EbookDAO::findById(id : int) **pre**: id > 0 **context** EbookDAO::findById(id : int) **post**: Se esiste un ebook con il seguente id allora il result è quell'ebook Altrimenti result = null
findByCoWriter	Il metodo ci permette di ricavare tutti quanti gli ebook che hanno come coAuthor l'autore identificato dall'id passato al metodo. **context** EbookDAO::findByCoWriter(coAuthorId : int) **pre**: coAuthorId > 0 & AuthorDAO.findById(coAuthorId) <> null **context** EbookDAO::findByCoWriter(coAuthorId : int) **post**: result = Set di ebook in cui l'autore identificato da coAuthorId è co-autore
findByMainWriter	Il metodo ci permette di ricavare tutti quanti gli ebook che hanno come autore principale l'autore identificato da mainAuthorId. **context** EbookDAO::findByCoWriter(mainAuthorId : int) **pre**: mainAuthorId > 0 & AuthorDAO.findById(mainAuthorId) <> null **context** EbookDAO::findByCoWriter(mainAuthorId : int) **post**: result = Set di ebook in cui l'autore identificato da mainAuthorId è autore principale

All'interno della nostra implementazione abbiamo deciso di seguire l'euristica di non implementare il controllo delle invarianti e delle post-condizioni per rientrare all'interno dei tempi di implementazione.

5. Mapping Data Storage Schema

Di seguito vengono riportate una serie di informazioni addizionali che non sono inferibili dal resto della documentazione riguardo allo storage schema.

Chiavi primarie:

Per le classi che rappresentano i dati(tutte quante le classi entity) non abbiamo individuato un attributo che potesse essere utilizzato in maniera efficiente come identificatore. Abbiamo quindi deciso di aggiungere un identificativo numerico. Abbiamo deciso di inserire un ID univoco generato automaticamente tramite l'aiuto di SQL. I seguenti ID sono stati aggiunti anche come attributi delle classi.

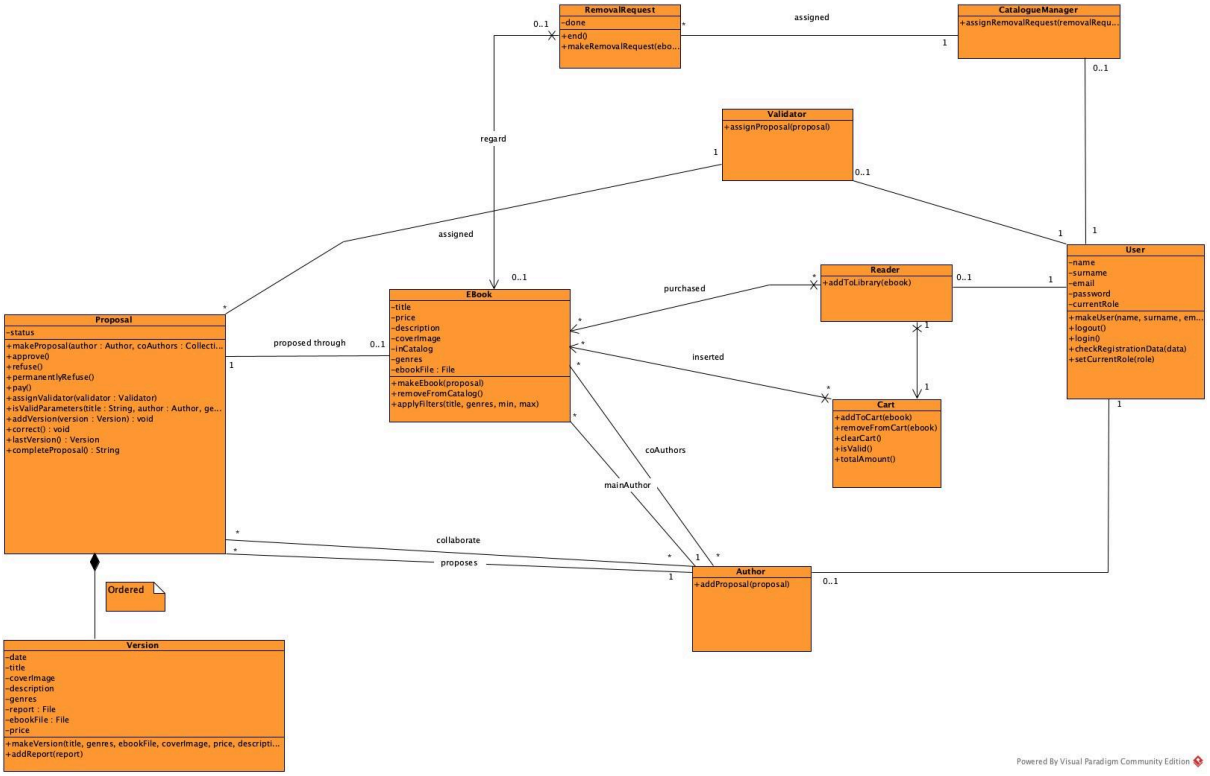
Per quanto concerne le stringhe, è stato necessario andare a stabilire una lunghezza massima per ciascuna di queste ultime.
I limiti imposti sono stati stabiliti guardando a quella che è la requirements elicitation.

Il nostro sistema fa anche utilizzo del file system per persistere una parte dei dati. In particolare piuttosto che utilizzare dei blob nel database abbiamo ritenuto necessario andare a salvare nel filesystem cover dei libri, file di report e file di ebook che si riferiscono a ciascuna versione.
Per fare in modo che le prestazioni del filesystem non degradino, è di fondamentale importanza mantenere un'organizzazione del filesystem.
Piuttosto che mantenere tutti quanti i file di ebook di tutte quante le versioni all'interno di una singola cartella, abbiamo deciso di creare una cartella per ospitare tutte quante le versioni della proposta. Tale cartella ha come nome l'identificativo della proposta.
I nomi dei file associati a ogni versione vengono quindi salvati all'interno del database. I file associati a ogni versione sono identificati come "tipoDiFile_" + versionId + ".extension". Ad esempio nel caso del file di ebook della versione 3 di una proposta avremo ebookFile_3.pdf..

6. Ristrutturazione e Ottimizzazione

Di seguito sono riportate le modifiche effettuate al class diagram con il loro razionale:

- Modifiche alla versione e ai file: Abbiamo deciso di non utilizzare dei riferimenti alle classi FileEBook e Report, all'interno delle quali abbiamo lasciato solamente i metodi statici per effettuare la verifica dell'estensione dei due file.
  Queste classi possono essere infatti mappate come la classe File di java che rappresenta proprio la "vista logica di un file". Quindi abbiamo aggiunto due attributi di tipo File a Version, che rappresentano proprio i due file.
- Riferimento da EBook a FileEBook: Precedentemente avevamo la relazione da EBook a FileEBook. Quest'ultima ci permetteva di ottenere l'ultima versione del FileEBook senza dover passare prima per la proposta e successivamente per l'ultima versione.
  Per mantenere il riferimento al file, abbiamo aggiunto un campo a EBook, ovvero ebookFile per rappresentare questa relazione.
  Anche quest'ultimo risulta essere un trade-off, se parte delle informazioni vengono duplicate, possiamo recuperare l'ebook più velocemente.
- Riferimenti da EBook a Author: Abbiamo deciso di mantenere queste due relazioni all'interno del class diagram in modo da non dover passare per la Proposal per recuperare gli autori. Trade off molto simile a quello visto precedentemente.
- Attributi ridondanti tra Versione e EBook: Gli attributi dell'ebook(title, description, price, ecc.) potrebbero essere recuperati a partire dall'ultima version della proposta. Abbiamo considerato che queste informazioni vengono riportate spesso. E' vero simile pensare infatti che un attività molto ricorrente sia recuperare un certo sottoinsieme degli ebook presenti all'interno del sistema.
  Con uno sguardo al futuro abbiamo ritenuto un trade-off necessario mantenere questi dati duplicati.



7. Design Pattern utilizzati

All’interno del nostro progetto sono stati utilizzati dei design pattern per riuscire a implementare alcune funzionalità del sistema.

Ad esempio abbiamo ritenuto utile andare a utilizzare un pattern di tipo interceptor per riuscire a realizzare il filtro che verifica i permessi per eseguire una determinata funzionalità. In questo modo andremo ad intercettare la richiesta e a scartarla nel caso in cui non si possieda il permesso necessario per eseguire la funzionalità richiesta.
Abbiamo deciso infatti di implementare a livello delle servlet il controllo degli accessi, in quanto realizzare i controlli attraverso questo pattern ci ha consentito di non aggiungere logica cross-cutting(in questo caso il controllo degli accessi) all’interno del livello di business(in particolare entity e dao).

Il secondo pattern che abbiamo deciso di utilizzare è un pattern adapter/bridge per incapsulare la logica di pagamento. Quest’ultima infatti è realizzata attraverso dei servizi che non implementiamo direttamente noi, ma che ci vengono offerti da enti di terze parti. Risulta quindi cruciale per evitare di diventare dipendenti rispetto a quel servizio e la sua implementazione utilizzare un pattern per disaccoppiare.