UNIVERSITÀ DEGLI STUDI DI SALERNO

Corso di Laurea in Informatica, prof. A. De Lucia, a.a. 2021-22

Progetto di Ingegneria del Software

Scadenza: 07-10-2021

› Project Proposal

Scadenza: 14-10-2021

› Problem Statement

Scadenza: 28-10-2021

› Requirement Analysis Document

Scadenza: 11-11-2021

› System Design Document

Scadenza: x-y-z

➥ Object Design Document

Scadenza: x-y-z

› Testing Documents

Scadenza: alla consegna del progetto

› Teamwork Report

Shodan [IS-2021/2022-DE LUCIA]

Partecipanti del progetto

Nome	Matricola	E-mail
Antonio Gravino	05121 07161	a.gravino3@studenti.unisa.it
Dario Trinchese	05121 07479	d.trinchese2@studenti.unisa.it
Raffaele Zheng	05121 09015	r.zheng@studenti.unisa.it
Carmine Fabbri	05121 07353	c.fabbri@studenti.unisa.it
Carmine Napolitano	05121 06417	c.napolitano44@studenti.unisa.it

Questo documento tratta esclusivamente l'**Object Design Document** del progetto.
Per ulteriore documentazione valida ai fini dell'esame, consultare la repository *docs*.

Indice dei contenuti

Introduzione 4
- Object Design Trade-offs 4
- Linee guida per la documentazione delle interfacce 5
- Definizioni, acronimi e abbreviazioni 5
- Riferimenti 6

Packages 7
- Back-end Packages 9
- Front-end Packages 11

Class Interface 14
- Control 14
- Utils 16
- Database 16
- Collection 17
- Handler 17
- Service 18
- Model 21

1. Introduzione

Dopo la realizzazione del Requirement Analysis Document e System Design Document, abbiamo descritto in linea di massima quello che sarà il nostro sistema e, quindi, i nostri obiettivi, tralasciando gli aspetti implementativi.

Il presente documento ha lo scopo di produrre un modello capace di integrare in modo coerente e preciso tutte le funzionalità individuate nelle fasi precedenti.

In particolare, definisce le interfacce delle classi, le operazioni, i tipi, gli argomenti e le *signatures* dei sottosistemi definiti nel System Design Document.

Inoltre, sono specificati i *trade-offs* e le linee guida.

1.1 Object Design Trade-offs

- **COMPRENSIBILITA' vs TEMPO**
  Il codice deve essere quanto più comprensibile possibile per facilitare la fase di testing ed eventuali future modifiche. Il codice sarà quindi accompagnato da commenti che ne semplifichino la comprensione. Ovviamente, questa caratteristica aggiungerà un incremento di tempo allo sviluppo del nostro progetto.

- **INTERFACCIA vs USABILITA'**
  L'interfaccia grafica è stata realizzata in modo da essere molto semplice, chiara e concisa, fa uso di *form* e pulsanti disposti in maniera da rendere semplice l'utilizzo del sistema a quanti più utenti possibili.

- **SICUREZZA vs EFFICIENZA**
  La sicurezza, come descritto nei requisiti non funzionali del Requirement Analysis Document, rappresenta uno degli aspetti chiave della piattaforma. Tuttavia, dati i tempi di sviluppo molto limitati, ci limiteremo ad implementare sistemi di sicurezza basati sulla crittografia della password degli utenti.

- **RESPONSE TIME vs HARDWARE:**
  Il sistema garantisce una certa reattività alle richieste, e quindi è in grado di poter comunque offrire una contemporaneità di servizi agli utenti. Ovviamente questa caratteristica sarà limitata dall'hardware del sistema.

- **PRESTAZIONI vs COSTI**

  Il sistema prevede l'utilizzo di fogli di stile semplici e open per mantenere prestazioni elevate, e cioè non s'intende utilizzare librerie grafiche esterne a pagamento, essendo il progetto sprovvisto di budget.

1.2 Linee guida per la documentazione delle interfacce

Gli sviluppatori dovranno seguire le seguenti convenzioni per la scrittura del codice:

NAMING CONVENTION

È buona norma utilizzare nomenclature che rispettino le seguenti caratteristiche:

- Descrittivi
- Di uso comune
- Lunghezza medio-corta

VARIABILI

I nomi delle variabili devono cominciare con una lettera minuscola, se il nome della variabile è costituito da più parole, solo l'iniziale delle altre parole sarà maiuscola (es: *nomeVariabile*). In casi di particolare necessità (es. nome variabile lungo) è possibile usare i trattini bassi, con iniziali minuscole.

METODI

I nomi dei metodi devono cominciare con una lettera minuscola, e le parole seguenti con la lettera maiuscola (Camel Case). Il nome del metodo tipicamente consiste di un verbo che identifica una azione, seguito dal nome di un oggetto.

I nomi dei metodi per l'accesso e la modifica delle variabili dovranno essere del tipo *getNomeVariabile()* e *setNomeVariabile()*.

CLASSI E PAGINE

I nomi delle classi e delle pagine devono iniziare con una lettera maiuscola, le parole contenute al suo interno devono cominciare con lettera maiuscola. Il nome deve fornire informazioni utili relative al loro scopo.

Ogni file sorgente contiene una singola classe e deve essere strutturato in un determinato modo:

- L' istruzione package che permette di inserire la classe in un determinato package.
- L' istruzione import che importa le librerie necessarie alla class.
- Una piccola descrizione della classe.
- Il codice effettivo del file sorgente.

1.3 Definizioni, acronimi e abbreviazioni

Da qui in poi…

con “*RAD*” s’intenderà il Requirement Analysis Document,
con “*SDD*” s’intenderà il System Design Document,
con “*ODD*” s’intenderà l’Object Design Document.

1.4 Riferimenti

Questo documento utilizza riferimenti a concetti introdotti ed esplicitati nel RAD e nel SDD di Shodan.

2. Packages

L'implementazione del back-end si declina nei seguenti pacchetti, ognuno dei quali è contenuto nella directory `/java/.`

- *Control*
- *Collection*
- *Database*
- *Handler*
- *Model*
- *Service*
- *Utils*

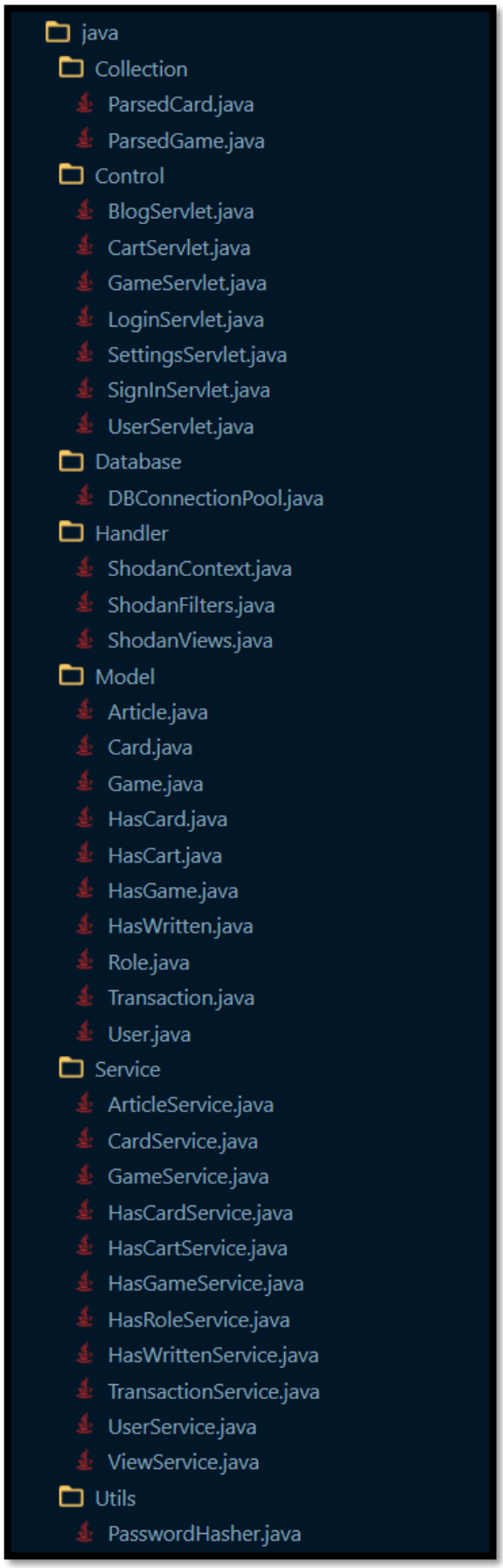
L'implementazione del front-end si declina nei seguenti pacchetti, organizzati in maniera gerarchica.

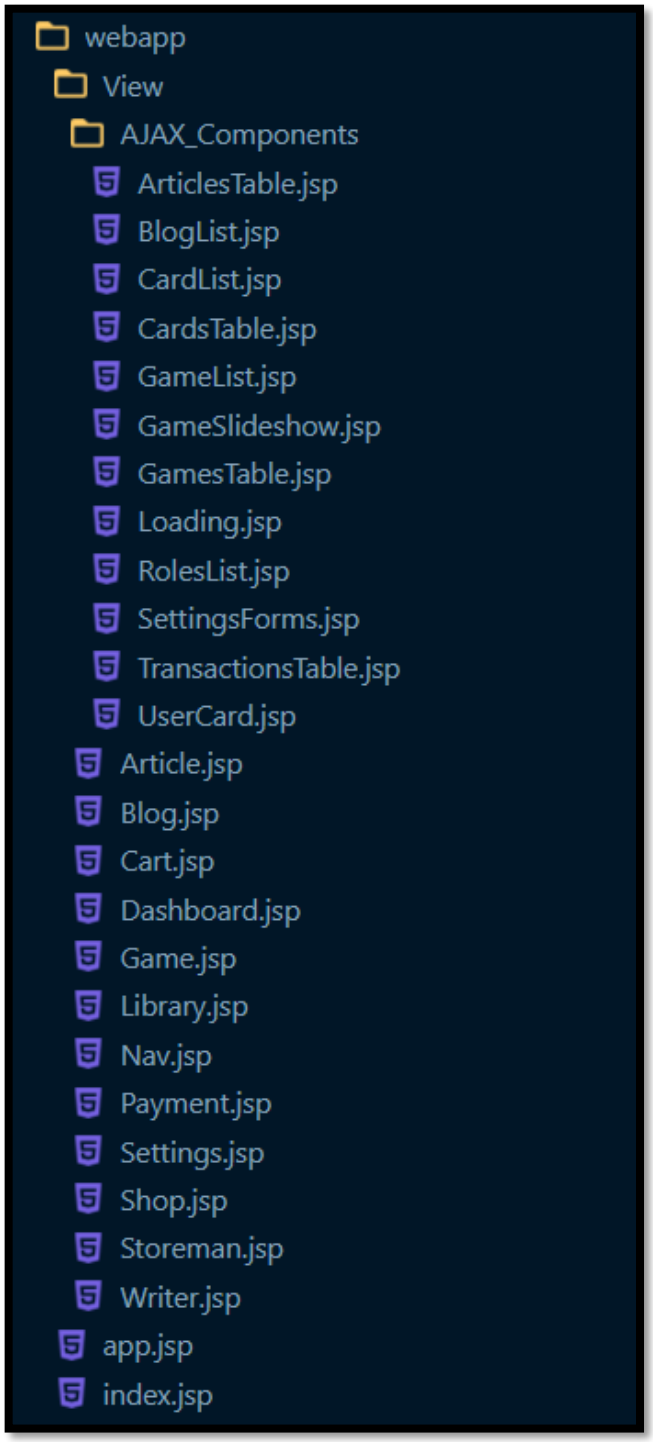
- *View (containers)*
  Coincide con la directory `/webapp/`

  - *View (components)*
    Coincide con la directory `/webapp/View/`

    - *View (sub-components)*
      Coincide con la directory `/webapp/View/AJAX_Components/`

Organizzazione dei packages





2.1 Back-end Packages

Control	
Classe	Descrizione
BlogServlet	Servlet che gestisce gli articoli del blog
CartServlet	Servlet che gestisce il carrello
GameServlet	Servlet che gestisce i titoli del catalogo
LoginServlet	Servlet che gestisce l'accesso alla piattaforma
SignInServlet	Servlet che gestisce la registrazione alla piattaforma
SettingsServlet	Servlet che gestisce la pagina delle impostazioni degli utenti
UserServlet	Servlet che gestisce gli utenti

Database	
Classe	Descrizione
DBConnectionPool	File di configurazione dell'accesso al database tramite i driver JDBC

Handler	
Classe	Descrizione
ShodanContext	Servlet che gestisce il contesto generale dell'applicazione
ShodanFilters	Servlet che permette di filtrare l'accesso alle pagine della piattaforma in funzione del ruolo e dello stato di accesso dell'utente
ShodanViews	Servlet che permette di renderizzare view differenti in funzione del ruolo dell'utente

Utils	
Classe	Descrizione
PasswordHasher	Utility d'ausilio che permette di crittografare le password

Collection	
Classe	Descrizione
ParsedCard	Collezione che permette di organizzare i dati di una carta di pagamento in modo sicuro
ParsedGame	Collezione che permette di organizzare i titoli in funzione del possesso da parte dell'utente

Model	
Classe	Descrizione
Article	Modello che delinea le caratteristiche di un articolo sul blog
Card	Modello che delinea le caratteristiche di una carta di pagamento
Game	Modello che delinea le caratteristiche di un titolo del catalogo
HasCart	Modello che delinea l'associamento fra titoli nel carrello e utenti
Role	Modello che delinea le specifiche dei ruoli di un utente
Transaction	Modello che delinea le caratteristiche di una transazione
User	Modello che delinea le caratteristiche di un utente

Service	
Classe	Descrizione
ArticleService	Descrive l'interazione col database per `Article`
CardService	Descrive l'interazione col database per `Card`
GameService	Descrive l'interazione col database per `Game`
HasCartService	Descrive l'interazione col database per `HasCart`

TransactionService	Descrive l'interazione col database per `Transaction`
UserService	Descrive l'interazione col database per `User`
ViewService	Descrive l'interazione col database per ottenere i path corretti delle pagine richieste in funzione del ruolo dell'utente

2.2 Front-end Packages

/webapp/	
JSP	Descrizione
App	View che funge da container principale per i componenti di Shodan
Index	View della landing page di Shodan

/webapp/View/	
JSP	Descrizione
Article	View del componente dell'articolo selezionato
Blog	View del componente del blog
Cart	View del carrello dell'utente
Dashboard	View della dashboard di default di Shodan
Game	View del componente del titolo selezionato
Library	View del componente della libreria dei titoli dell'utente
Nav	View del componente della barra navigazionale
Payment	View del componente delle carte registrare dall'utente

Settings	View del componente delle impostazioni
Shop	View del catalogo della piattaforma
Storeman	View della pagina di amministrazione del cataloghista
Writer	View della pagina di amministrazione dell'articolista

/webapp/View/AJAX_Components	
JSP	Descrizione
ArticlesTable	View del sotto-componente di una tabella che mostra lo storico degli articoli aggiunti sulla piattaforma
BlogList	View del sotto-componente di una lista di articoli
CardList	View del sotto-componente di una lista di carte di pagamento
CardsTable	View del sotto-componente di una tabella che mostra lo storico delle carte di pagamento aggiunte dagli utenti
GameList	View del sotto-componente di una lista di titoli
GamesSlideshow	View del sotto-componente di una lista di titoli organizzati in maniera verticale
GamesTable	View del sotto-componente di una tabella che mostra lo storico dei titoli aggiunti sulla piattaforma
Loading	View del sotto-componente di caricamento della piattaforma
RolesList	View del sotto-componente che permette il Role Switching nella barra navigazionale
SettingsForms	View del sotto-componente dei form relativi alle impostazioni dell'utente
TransactionsTable	View del sotto-componente di una tabella che mostra le transazioni dell'utente

UserCard	View del sotto-componente di una card che mostra i dettagli dell'account di un utente

3. Class Interface

3.1 Control

Nome classe	**BlogServlet**
Descrizione	Servlet che gestisce il blog di Shodan
Signature dei metodi	# doGet(HttpServletRequest, HttpServletResponse) : void action: • blog • article # doPost(HttpServletRequest, HttpServletResponse) : void action: • addArticle • deleteArticle
Pre-condizioni	
Post-condizioni	
Invariante	

Nome classe	**CartServlet**
Descrizione	Servlet che gestisce il carrello e i pagamenti
Signature dei metodi	# doGet(HttpServletRequest, HttpServletResponse) : void action: • cartPage • quantity # doPost(HttpServletRequest, HttpServletResponse) : void action: • delete • removeItem • pay • addGame
Pre-condizioni	
Post-condizioni	
Invariante	

Nome classe	**GameServlet**
Descrizione	Servlet che gestisce i titoli
Signature dei metodi	# doGet(HttpServletRequest, HttpServletResponse) : void action: • parsed • shop

	• library • game • hasGame • searchGameShop • searchGameLibrary • getGames # doPost(HttpServletRequest, HttpServletResponse) : void action: • addGame • deleteGame • updateGame
Pre-condizioni	
Post-condizioni	
Invariante	

Nome classe	**LoginServlet**
Descrizione	Servlet che gestisce l'accesso alla piattaforma
Signature dei metodi	# doPost(HttpServletRequest, HttpServletResponse) : void
Pre-condizioni	
Post-condizioni	
Invariante	

Nome classe	**SignInServlet**
Descrizione	Servlet che gestisce la registrazione di nuovi utenti
Signature dei metodi	# doPost(HttpServletRequest, HttpServletResponse) : void action: • addArticle • deleteArticle
Pre-condizioni	
Post-condizioni	
Invariante	

Nome classe	**SettingsServlet**
Descrizione	Servlet che gestisce le impostazioni degli utenti
Signature dei metodi	# doGet(HttpServletRequest, HttpServletResponse) : void action: • userCard • settingsForm • transactionsTable • cardsTable # doPost(HttpServletRequest request, HttpServletResponse response) : void

	action: • settingsForms • transactionsTable • cardsTable • updateEmail • updatePassword
Pre-condizioni	
Post-condizioni	
Invariante	

Nome classe	**UserServlet**
Descrizione	Servlet che gestisce gli utenti di Shodan
Signature dei metodi	# doGet(HttpServletRequest, HttpServletResponse) : void action: • role • switchableRoles • cardList • updateBalance • purchase # doPost(HttpServletRequest, HttpServletResponse) : void action: • newCard • logout
Pre-condizioni	
Post-condizioni	
Invariante	

3.2 Utils

Nome classe	**PasswordHasher**
Descrizione	Classe di utility che permette la crittografia delle password
Signature dei metodi	+ hash(String) : String
Pre-condizioni	
Post-condizioni	
Invariante	

3.3 Database

Nome classe	**DBConnectionPool**
Descrizione	Classe che permette l'interazione col database
Signature dei metodi	+ createDBConnection() : Connection + getConnection() : Connection + releaseConnection(Collection) : void
Pre-condizioni	
Post-condizioni	

Invariante	

3.4 Collection

Nome classe	**ParsedCard**
Descrizione	Classe che permette di renderizzare in modo sicuro le carte di pagamento, celando alcuni dati

Signature dei metodi	`+ ParsedCard(Card)`
	`+ getCard() : Card`
	`+ getSafeDigits() : long`
	`+ setCard(Card) : void`
	`+ setSafe_digits(long) : void`

Pre-condizioni	
Post-condizioni	
Invariante	

Nome classe	**ParsedGame**
Descrizione	Classe che permette l'associazione di giochi in funzione del possesso

Signature dei metodi	`+ ParsedGame(Game, boolean)`
	`+ getGame() : Game`
	`+ getOwned() : boolean`
	`+ setGame(Game) : void`
	`+ setOwned(boolean) : void`

Pre-condizioni	
Post-condizioni	
Invariante	

3.5 Handler

Nome classe	**ShodanContext**
Descrizione	Classe che funge da contesto per le servlet del sistema
Signature dei metodi	`+ contextInitialized(ServletContextEvent) : void` `+ contextDestroyed(ServletContextEvent) : void`
Pre-condizioni	
Post-condizioni	
Invariante	

Nome classe	**ShodanFilters**
Descrizione	Classe che permette di filtrare l'accesso alla piattaforma in funzione dello status di login

Signature dei metodi	`+ doFilter(ServletRequest, ServletResponse, FilterChain) : void`
	`+ desotry() : void`
	`+ init() : void`

Pre-condizioni	
Post-condizioni	
Invariante	

Nome classe	**ShodanViews**
Descrizione	Classe che permette di filtrare il tipo di pagine da mostrare all'utente in funzione dei suoi ruoli
Signature dei metodi	`# doGet(HttpServletRequest, HttpServletResponse) : void`
Pre-condizioni	
Post-condizioni	
Invariante	

3.6 Service

Nome classe	**ArticleService**
Descrizione	Classe che permette l'interazione con la tabella "`blog`" del database

Signature dei metodi	`+ ArticleService(Connection)`
	`+ getArticle(int) : Article`
	`+ getAllArticles(int) : ArrayList<Article>`
	`+ addArticle(Article) : boolean`
	`+ findArticle(String, String, String) : Article`
	`+ deleteArticle(int) : boolean`

Pre-condizioni	
Post-condizioni	
Invariante	

Nome classe	**CardService**
Descrizione	Classe che permette l'interazione con la tabella "`cards`" del database

Signature dei metodi	`+ CardService(Connection)`
	`+ getCard(int) : Card`
	`+ getAllCards() : ArrayList<Card>`
	`+ insertCard(Card) : boolean`
	`+ getCardsByOwner(String, String, String) : ArrayList<Card>`
	`+ getCardByNumber(Long) : Card`

Pre-condizioni	
Post-condizioni	
Invariante	

Nome classe	**GameService**
Descrizione	Classe che permette l'interazione con la tabella "`games`" del database

Signature dei metodi	`+ GameService(Connection)`
	`+ getGame(int) : Game`
	`+ getGameByName(String) : Game`
	`+ getAllGamesByUser(int) : ArrayList<Game>`
	`+ getAllDescendingGames(int) : ArrayList<Game>`
	`+ searchGames(String) : ArrayList<Game>`

	+ searchGamesInLibrary(int, String) : ArrayList<Game> + getAllAscendingGames(int) : ArrayList<Game> + addGame(Game) : boolean + updateGame(Game) : boolean + deleteGame(int) : boolean
Pre-condizioni	
Post-condizioni	
Invariante	

Nome classe	**HasCartService**
Descrizione	Classe che permette l'interazione con la tabella "has_cart" del database
Signature dei metodi	+ HasCartService(Connection) + addItem(HasCart) : boolean + removeItem(HasCart) : boolean + removeItemForAll(Game) : boolean + dropCart(User) : boolean + selectCart(User) : ArrayList<Game> + hasInCart(User, Game) : boolean
Pre-condizioni	
Post-condizioni	
Invariante	

Nome classe	**HasGameService**
Descrizione	Classe che permette l'interazione con la tabella "has_game" del database
Signature dei metodi	+ HasGameService(Connection) + addGame(User, Game) : void + hasGame(User, Game) : boolean + removeItem(User, Game) : boolean + removeItemForAll(Game) : boolean
Pre-condizioni	
Post-condizioni	
Invariante	

Nome classe	**ArticleService**
Descrizione	Classe che permette l'interazione con la tabella "blog" del database
Signature dei metodi	+ ArticleService(Connection) + getArticle(int) : Article + getAllArticles(int) : ArrayList<Article> + addArticle(Article) : boolean + findArticle(String, String, String) : Article + deleteArticle(int) : boolean
Pre-condizioni	
Post-condizioni	

Invariante	

Nome classe	**HasRoleService**
Descrizione	Classe che permette l'interazione con la tabella "`has_role`" del database

Signature dei metodi	`+ HasRoleService(Connection)`
	`+ getRoles(int) : ArrayList<Role>`
	`+ getMainRole(ArrayList<Role>) : Role`

Pre-condizioni	
Post-condizioni	
Invariante	

Nome classe	**TransactionService**
Descrizione	Classe che permette l'interazione con la tabella "`transactions`" del database

Signature dei metodi	`+ TransactionService(Connection)`
	`+ getTransactions() : ArrayList<Transaction>`
	`+ getTransactionsByUser(User) : ArrayList<Transaction>`
	`+ insertTransaction(Transaction) : void`

Pre-condizioni	
Post-condizioni	
Invariante	

Nome classe	**UserService**
Descrizione	Classe che permette l'interazione con la tabella "`users`" del database e l'accesso delle sessioni

Signature dei metodi	`+ UserService(Connection)`
	`+ getIdByUsername(String) : int`
	`+ getUser(int) : User`
	`+ findUserByEmail(String) : boolean`
	`+ updateUser(User) : boolean`
	`+ insertUser(User) : boolean`
	`+ deleteUser(User) : boolean`
	`+ getUserBySession(String) :User`
	`+ insertSession(String, User) : boolean`
	`+ destroySession(User) : void`

Pre-condizioni	
Post-condizioni	
Invariante	

Nome classe	**ViewService**
Descrizione	Classe che permette l'interazione con la tabella "views" del database
Signature dei metodi	`+ ViewService(Connection)` `+ getView(Role, RequestedView) : String`

Pre-condizioni	
Post-condizioni	
Invariante	

3.7 Model

Nome classe	**Article**
Descrizione	Classe che descrive gli articoli
Signature dei metodi	+ Article(String, String, String, User) + Article(int, String, String, String, User) + getId() : int + getTitle() : String + getShortTitle() : String + getHtml() : String + getAuthor() : User + setId() : int + setTitle() : String + setShortTitle() : String + setHtml() : String + setAuthor() : User
Pre-condizioni	
Post-condizioni	
Invariante	

Nome classe	**Card**
Descrizione	Classe che descrive le carte di pagamento
Signature dei metodi	+ Card(int, String, long, String, Date, User) + getCard_id() : int + getCard_type() : String + getCard_number() : long + getCard_owner() : String + getCard_date() : Date + getOwner() : User + setCard_id(int) : void + setCard_type(String) : void + setCard_number(long) : void + setCard_owner(String) : void + setCard_date(Date) : void + setOwner(User) : void
Pre-condizioni	
Post-condizioni	
Invariante	

Nome classe	**Game**
Descrizione	Classe che descrive i titoli
Signature dei metodi	+ Game(int, int, String, String, String, Date, String) + getId() : int + getPrice() : int

	+ getName() : String + getDescription() : String + getImage() : String + getRelease() : Date + getLandscape() : String + setId(int) : void + setPrice(int) : void + setName(String) : void + setDescription(String) : void + setImage(String) : void + setRelease(Date) : void + setLandscape(String) : void
Pre-condizioni	
Post-condizioni	
Invariante	

Nome classe	**HasCart**
Descrizione	Classe che associa titoli nel carrello agli utenti
Signature dei metodi	+ HasCart(int, int) + getUserId() : int + getGameId() : int + setUserId(int) : void + setGameId(int) : void
Pre-condizioni	
Post-condizioni	
Invariante	

Nome classe	**HasGame**
Descrizione	Classe che associa titoli agli utenti in funzione del possesso
Signature dei metodi	+ HasGame(int, int) + getUserId() : int + getGameId() : int + setUserId(int) : void + setGameId(game) : void
Pre-condizioni	
Post-condizioni	
Invariante	

Nome classe	**Role**
Descrizione	Classe che associa ruoli agli utenti
Signature dei metodi	+ Role(int, String) + getUserId() : int + getRoleName() : String + getParsedRoleName() : String + setUserId(int) : void + setRoleName(String) : void + setParsedRoleName(String) : void
Pre-condizioni	

Post-condizioni	
Invariante	

Nome classe	**Transaction**
Descrizione	Classe che descrive le transazioni
Signature dei metodi	+ Transaction(User, Game, Date, int) + getUser() : User + getGame() : Game + getTransaction_date() : Date + getTransaction_price() : int + setUser(User) : void + setGame(Game) : void + setTransaction_date(Date) : void + setTransaction_price(int) : void
Pre-condizioni	
Post-condizioni	
Invariante	

Nome classe	**User**
Descrizione	Classe che descrive gli utenti
Signature dei metodi	+ User(int, String, String, String, int, String, ArrayList<Role>) + getId() : int + getName() : String + getPassword() : String + getEmail() : String + getSession() : String + getMoney() : int + getRoles() : ArrayList<Role> + setId(int) : void + setName(String) : void + setPassword(String) : void + setEmail(String) : void + setMoney(int) : void + setSession(String) : void + setRoles(ArrayList<Role>) : void
Pre-condizioni	
Post-condizioni	
Invariante	