Object Design
“Castle Movie Theater”

1. Introduzione

1. Object design trade-offs

- **Usabilità vs. Funzionalità**

  Essendo l’usabilità uno degli obiettivi principali del progetto, si è scelto di limitare il numero di funzionalità al fine di semplificare e snellire l’interazione dell’utente con il sito

- **Rapidità di sviluppo vs. Prestazioni**

  Volendo sviluppare il progetto in tempi rapidi, l’opera di ottimizzazione delle procedure implementate è stata ridotta.

- **Costi vs. Sicurezza**

  Non essendo disponibile alcuna risorsa finanziaria, è stato deciso di non acquistare alcuna licenza HTTPS, e quindi non implementare il protocollo nel sistema

2. Linee guide per l’interfaccia

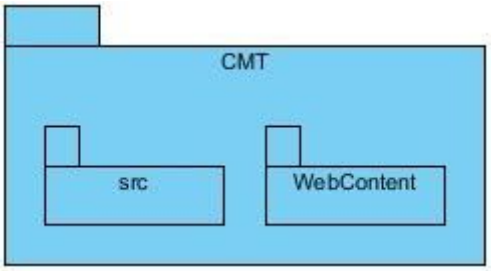
Essendo il sito multi-utente, il sistema deve mostrare e nascondere funzionalità in base al tipo di utente connesso.

3. Definizione, acronimi e abbreviazioni

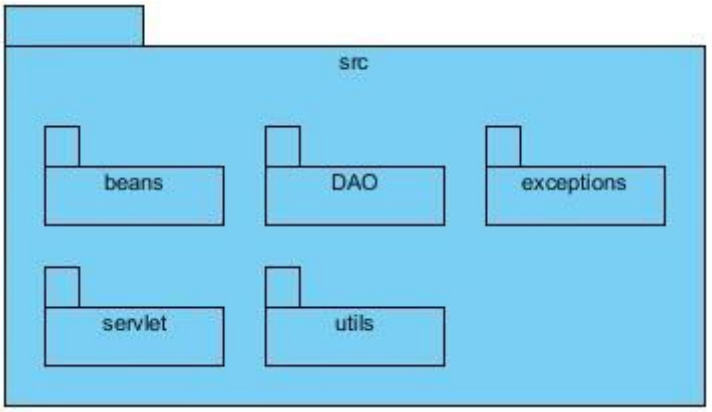
- HTTPS = HyperText Transfer Protocol
- CMT = Castle Movie Theater
- src = Source
- utils = Utilities
- DAO = Data Access Object
- JSP = JavaServer Pages
- CSS = Cascading Style Sheets
- JS = JavaScript

2. Package

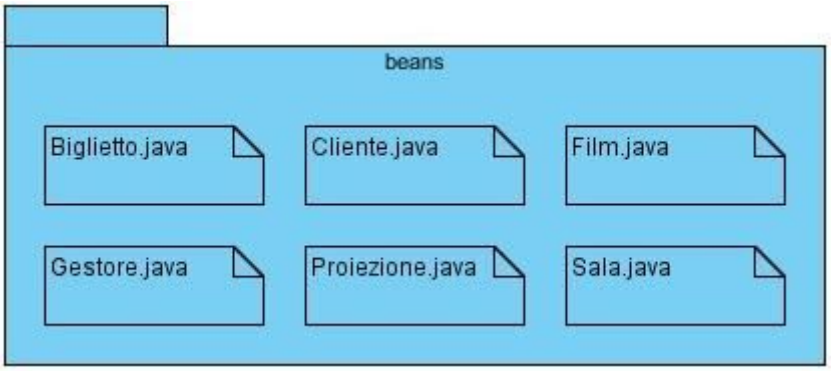
1. Package principale



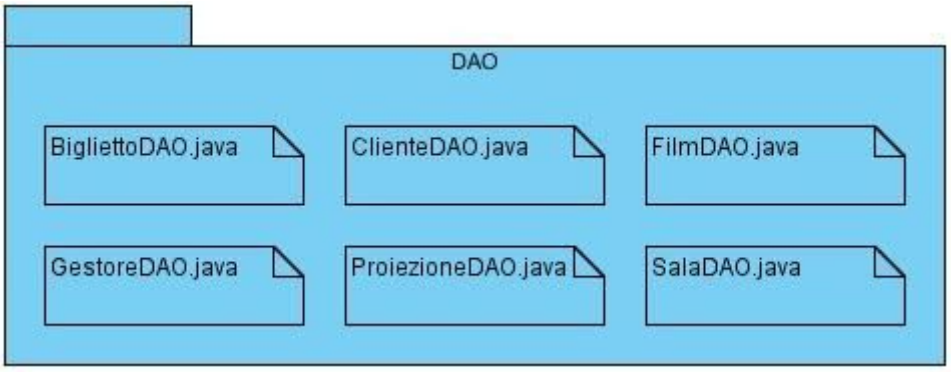
2. src



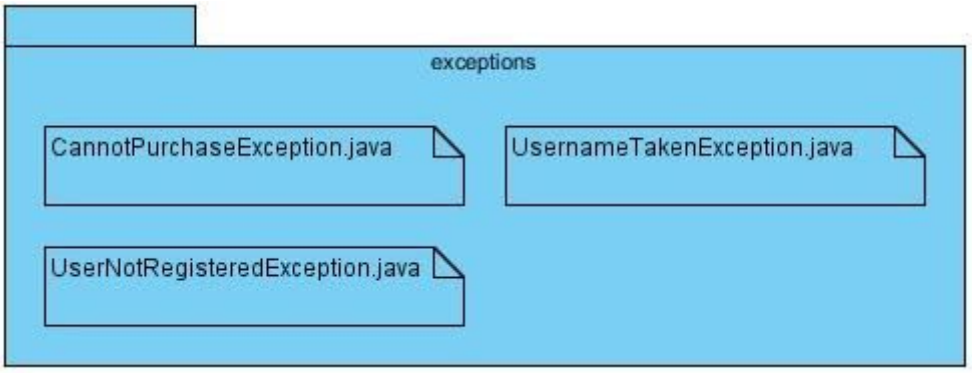
3. beans



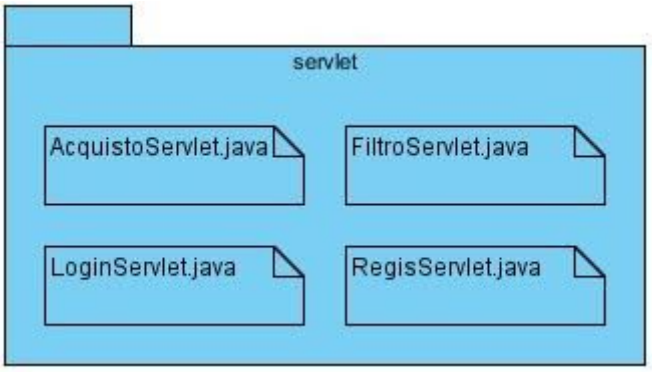
4. **DAO**



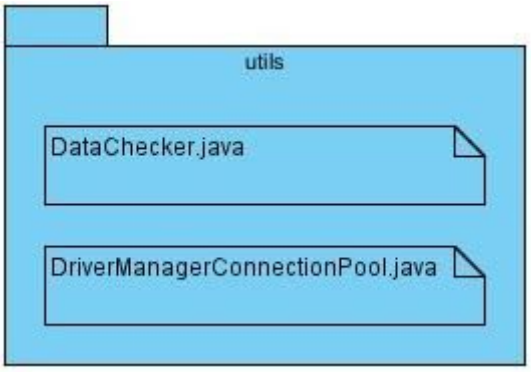
5. **exceptions**



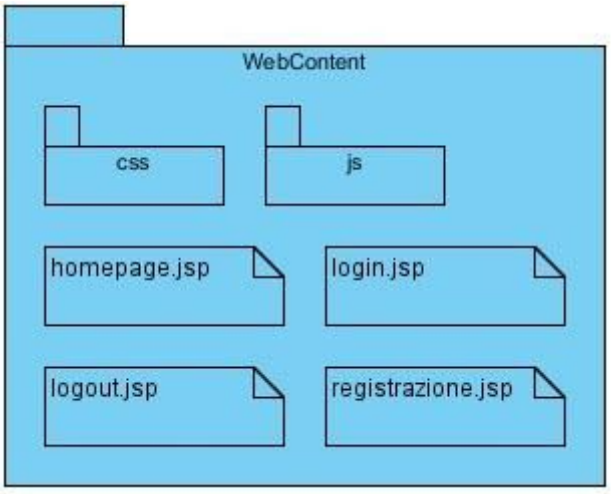
6. **servlet**



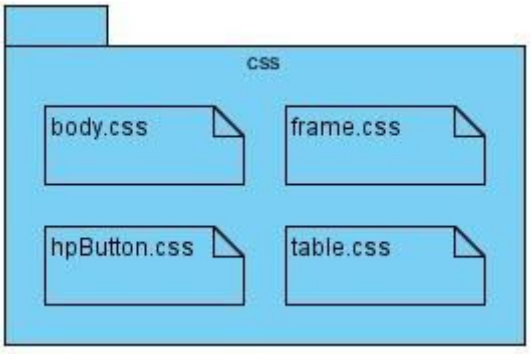
7. **utils**



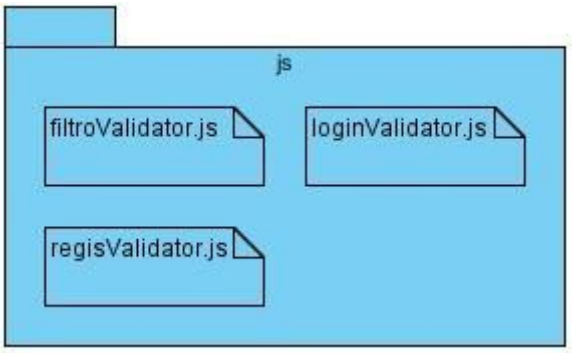
8. **WebContent**



9. **css**



10. js


js
filtroValidator.js
loginValidator.js
regisValidator.js

3. Interfaccia delle classi

1. beans

1. Biglietto

Nome entità:	**Biglietto**
Attributi:	● int **id** ● short **posto** ● int **idCliente** ● int **idProiezione**
Metodi:	● getId(): int ● setId(int id): void ● getPosto(): short ● setPosto(short posto): void ● getIdCliente(): int ● setIdCliente(int idCliente): void ● getIdProiezione(): int ● setIdProiezione(int idProiezione): void

2. Cliente

Nome entità:	**Cliente**
Attributi:	● int **id** ● String **username** ● String **password** ● float **saldo**
Metodi:	● getId(): int ● setId(int id): void ● getUsername():String ● setUsername(String username): void ● getPassword(): String ● setPassword(String password): void ● getSaldo(): float ● setSaldo(float saldo): void

3. **Film**

Nome entità:	**Film**
Attributi:	● int **id** ● String **titolo** ● short **durata** ● String **genere** ● String **regista** ● String **attore1** ● String **attore2** ● String **descrizione** ● String **locandina**
Metodi:	● getId():int ● setId(int id): void ● getDurata(): short ● setDurata(short durata): void ● getGenere(): String ● setGenere(String genere): void ● getRegista(): String ● setRegista(String regista): void ● getAttore1(): String ● setAttore1(String attore1): void ● getAttore2(): String ● setAttore2(String attore2): void ● getDescrizione(): String ● setDescrizione(String descrizione): void ● getLocandina(): String ● setLocandina(String locandina): void

4. **Gestore**

Nome entità:	**Gestore**
Attributi:	● int **id** ● String **username** ● String **password**
Metodi:	● getId():int ● setId(int id): void ● getusername(): String ● setusername(String username): void ● getPassword(): String ● setPassword(String password): void

5. **Proiezione**

Nome entità:	**Proiezione**
Attributi:	● int **id** ● int **data** ● short **orario** ● float **costo** ● int **idSala** ● int **idFilm**
Metodi:	● getId(): int ● setId(int id): void ● getData(): int ● setData(int data): void ● getOrario(): short ● setOrario(short orario): void ● getCosto(): float ● setCosto(float costo): void ● getIdSala(): int ● setIdSala(int idSala): void ● getIdFilm(): int ● setIdFilm(int idFilm): void

6. **Sala**

Nome entità:	**Sala**
Attributi:	● int **id** ● byte **numeroFile** ● byte **postiFila**
Metodi:	● getId(): int ● setId(int id): void ● getNumeroFile(): byte ● setNumeroFile(byte numeroFile): void ● getPostiFila(): byte ● setPostiFila(byte NumeroFile): void

2. DAO

1. BigliettoDAO

Nome entità:	**BigliettoDAO**
Metodi:	● getPosti(int idProiezione): ArrayList<Short> ● getLastId(): int ● addBiglietto(Biglietto biglietto): void

- **Metodo getPosti**

Nome metodo:	**getPosti**
Descrizione:	Dato in input l'id di una proiezione, restituisce la lista dei posti occupati per quest'ultima
Precondizione:	● idProiezione >= 0
Postcondizione:	Ritorna una lista contenente i posti occupati per quella proiezione

- **Metodo getLastId**

Nome metodo:	**getLastId**
Descrizione:	Restituisce l'id dei biglietti maggiore
Precondizione:	//
Postcondizione:	Ritorna l'id dei biglietti maggiore, -1 se non ve ne è nessuno

- **Metodo addBiglietto**

Nome metodo:	**addBiglietto**
Descrizione:	Dato in input un oggetto di tipo “Biglietto”, procede ad inserirlo all’interno del DB
Precondizione:	● Biglietto != null
Postcondizione:	//

2. ClienteDAO

Nome entità:	**ClienteDAO**
Metodi:	● getId(String username, String password): int ● isRegistered(String username): boolean ● getLastId(): int ● addCliente(Cliente cliente): void ● spend(int id, float ammontare): void

- **Metodo getId**

Nome metodo:	**getId**
Descrizione:	Dati in input username e password, restituisce l’id del Cliente associato
Precondizione:	● username != null ● password != null
Postcondizione:	Ritorna l’id del Cliente associato, -1 se non ha trovato corrispondenze

- **Metodo isRegistered**

Nome metodo:	**isRegistered**
Descrizione:	Dati in input un username, ritorna un booleano il cui valore dipende dalla presenza o meno dell’argomento nel DB
Precondizione:	● username != null
Postcondizione:	Ritorna TRUE se è presente un Cliente col username

	specificato, FALSE altrimenti

- **Metodo getLastId**

Nome metodo:	**getLastId**
Descrizione:	Restituisce l'id dei biglietti maggiore
Precondizione:	//
Postcondizione:	Ritorna l'id dei biglietti maggiore, -1 se non ve ne è nessuno

- **Metodo addCliente**

Nome metodo:	**addCliente**
Descrizione:	Dato in input un oggetto di tipo "Cliente", procede ad inserirlo all'interno del DB
Precondizione:	• Cliente != null
Postcondizione:	//

- **Metodo spend**

Nome metodo:	**spend**
Descrizione:	Dato in input un id ed un ammontare, scala ammontare dal saldo del cliente associato all'id.
Precondizione:	• id >= 0 • ammontare >= 0
Postcondizione:	//

3. FilmDAO

Nome entità:	**FilmDAO**
Metodi:	• findFilm(String titolo, String genere, String regista, String attore): ArrayList<Film>

- **Metodo findFilm**

Nome metodo:	**findFilm**
Descrizione:	Dati in input titolo, genere, regista e attore, restituisce una lista contenente tutti i film i cui attributi combaciano con quelli inseriti
Precondizione:	• titolo != null • genere != null • regista != null • attore != null
Postcondizione:	Ritorna una lista contenente tutti i film che rispettano il filtro

4. **GestoreDAO**

Nome entità:	**GestoreDAO**
Metodi:	• getId(String username, String password): int • isRegistered(String username): boolean

- **Metodo getId**

Nome metodo:	**getId**
Descrizione:	Dati in input username e password, restituisce l'id del Gestore associato
Precondizione:	• username != null • password != null
Postcondizione:	Ritorna l'id del Gestore associato, -1 se non ha trovato corrispondenze

- **Metodo isRegistered**

Nome metodo:	**isRegistered**
Descrizione:	Dati in input un username, ritorna un booleano il cui valore dipende dalla presenza o meno dell'argomento nel DB

Precondizione:	● username != null
Postcondizione:	Ritorna TRUE se è presente un Gestore col username specificato, FALSE altrimenti

5. ProiezioneDAO

Nome entità:	**ProiezioneDAO**
Metodi:	● getProiezioni(int idFilm, boolean includePast): ArrayList<Proiezione>

● **Metodo getProiezioni**

Nome metodo:	**getProiezioni**
Descrizione:	Dati in input un idFilm e includePast, restituisce la lista di tutte le proiezioni associate al film indicato
Precondizione:	● idFilm >= 0
Postcondizione:	Ritorna la lista delle proiezioni associate al film indicato

6. SalaDAO

Nome entità:	**SalaDAO**
Metodi:	● getSala(int id): Sala

● **Metodo getSala**

Nome metodo:	**getSala**
Descrizione:	Dato in input un id, restituisce la Sala associata
Precondizione:	● idSala >= 0
Postcondizione:	Ritorna la Sala associata all'id, null se ve ne è nessuna

3. Servlet

Nome servlet:	**AcquistoServlet**
Descrizione:	Permette di acquistare biglietti per una proiezione ad un Cliente
Precondizione:	Nella sessione deve essere presente l'attributo "id", mentre il form deve contenere parametri di "totale", "posti" e "idProiezione"
Postcondizione:	Il Cliente ha acquistato dei biglietti

Nome servlet:	**FiltroServlet**
Descrizione:	Restituisce una lista di film in programmazione filtrata per attributi
Precondizione:	Il form deve contenere parametri di "titolo", "genere", "regista", "attore"
Postcondizione:	Viene restituita la lista di film in programmazione che soddisfa il filtro

Nome servlet:	**LoginServlet**
Descrizione:	Permette l'autenticazione nel sito a utenti registrati
Precondizione:	Il form deve contenere parametri di "username" e "password"
Postcondizione:	L'utente è autenticato

Nome servlet:	**RegisServlet**
Descrizione:	Permette ad un utente di registrarsi come Cliente
Precondizione:	Il form deve contenere parametri di "username", "password", "confermaPassword", "saldo" e "trattamento" (dati personali)
Postcondizione:	L'utente è autenticato e registrato

4. Utils

1. DataChecker

Nome entità:	**DataChecker**
Descrizione:	Controlla la validità dei dati inseriti dall'utente
Attributi:	● Map<Field, String> **errorMsgMap** ● Map<Page, String> **pageMap**
Metodi:	● writeErrorMessage(HttpServletResponse response, String msg, String page): void ● checkBetween(String string, int min, int max): boolean ● checkIsNumber(String string): boolean ● checkFiltro(HttpServletResponse response, String titolo, String genere, String regista, String attore): boolean ● checkRegistrazione(HttpServletResponse response, String username, String password, String saldo): boolean ● checkLogin(HttpServletResponse response, String username, String password): boolean

- **Metodo writeErrorMessage**

Nome metodo:	**writeErrorMessage**
Descrizione:	Stampa a schermo un messaggio d'errore e reindirizza alla pagina indicata
Precondizione:	● response != null ● msg != null ● page != null
Postcondizione:	//

- **Metodo checkBetween**

Nome metodo:	**checkBetween**
Descrizione:	Controlla se la lunghezza della stringa indicata è compresa tra un minimo e un massimo
Precondizione:	• string != null • min >= 0 • max >= min
Postcondizione:	Ritorna TRUE se compresa tra min e max, FALSE altrimenti

- **Metodo checkIsNumber**

Nome metodo:	**checkIsNumber**
Descrizione:	Controlla se la stringa indicata è un numero
Precondizione:	• string != null
Postcondizione:	Ritorna TRUE se contiene un numero, FALSE altrimenti

- **Metodo checkFiltro**

Nome metodo:	**checkFiltro**
Descrizione:	Controlla se le stringhe passate come argomento rispettano il proprio formato
Precondizione:	• response != null • titolo != null • genere != null • regista != null • attore != null
Postcondizione:	Ritorna TRUE se rispettano il formato, FALSE altrimenti

- **Metodo checkRegistrazione**

Nome metodo:	**checkRegistrazione**
Descrizione:	Controlla se le stringhe passate come argomento rispettano il proprio formato
Precondizione:	● response != null ● username != null ● password != null ● saldo != null
Postcondizione:	Ritorna TRUE se rispettano il formato, FALSE altrimenti

- **Metodo checkLogin**

Nome metodo:	**checkLogin**
Descrizione:	Controlla se le stringhe passate come argomento rispettano il proprio formato
Precondizione:	● response != null ● username != null ● password != null
Postcondizione:	Ritorna TRUE se rispettano il formato, FALSE altrimenti

2. DriverManagerConnectionPool

Nome entità:	**DriverManagerConnectionPool**
Descrizione:	Gestisce la connessione tra il server ed il DB
Attributi:	● List<Connection> **freeDbConnections**
Metodi:	● createDBConnection(): Connection ● getConnection(): Connection ● releaseConnection(Connection connection): void

- **Metodo createDBConnection**

Nome metodo:	**createDBConnection**
Descrizione:	Crea e restituisce un nuovo oggetto connection tramite cui comunicare con il DB
Precondizione:	//
Postcondizione:	Ritorna un oggetto di tipo connection

- **Metodo getConnection**

Nome metodo:	**getConnection**
Descrizione:	Ritorna una connection, o prelevandola dalla lista delle connessioni libere o creandone una nuova in caso questa sia vuota
Precondizione:	//
Postcondizione:	Ritorna un oggetto di tipo connection

- **Metodo releaseConnection**

Nome metodo:	**releaseConnection**
Descrizione:	Inserisce la connessione “conn” passata in argomento alla lista delle connessioni libere
Precondizione:	• connection != null
Postcondizione:	//