**ODD Object Design**

**Document**

**Ticket4You**

|  |  |
| --- | --- |
| **Versione** 0.1 | |
| **Data** | 05/01/2024 |
| **Destinatario** | Esame di Ingegneria del Software 2023/24 |
| **Presentato da** | Tresy Sorrentino 0512114294  Simona Vigorito 0512115473  Nicolò Sorà 0512115140  Luca Greco 0512106227 |

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Data** | **Versione** | **Descrizione** | **Autori** |
| 27/12/2023 | 0.1 | Prima stesura | Tresy Sorrentino, Simona Vigorito, Nicolò Sorà, Luca Greco |
| 02/01/2024 | 0.2 | Definizione dei package | Tresy Sorrentino, Simona Vigorito, Nicolò Sorà, Luca Greco |
| 03/01/2024 | 0.3 | Definizione e stesura dei design pattern | Tresy Sorrentino, Simona Vigorito, Nicolò Sorà, Luca Greco |
| 04/01/2024 | 0.4 | Implementazione delle interfacce delle classi e del Class Diagram Ristrutturato | Tresy Sorrentino, Simona Vigorito, Nicolò Sorà, Luca Greco |
| 05/01/2024 | 0.5 | Revisione | Tresy Sorrentino, Simona Vigorito, Nicolò Sorà, Luca Greco |

**Team members**

|  |  |  |  |
| --- | --- | --- | --- |
| **Nome** | **Ruolo nel progetto** | **Acronimo** | **Informazioni di contatto** |
| Tresy Sorrentino | Team Member | TS | t.sorrentino15@studenti.unisa.it |
| Simona Vigorito | Team Member | SV | [s.vigorito2@studenti.unisa.it](mailto:s.vigorito2@studenti.unisa.it) |
| Nicolò Sorà | Team Member | NS | n.sora@studenti.unisa.it |
| Luca Greco | Team Member | LG | l.greco26@studenti.unisa.it |

Sommario

[1. Introduzione 5](#_Toc39321)

[1.1 Linee guida per la scrittura del codice 5](#_Toc39322)

[1.2 Definizione, acronimi e abbreviazioni 5](#_Toc39323)

[1.3 Riferimenti e link utili 5](#_Toc39324)

[2. Packages 6](#_Toc39325)

[3. Class Interfaces 8](#_Toc39326)

[4. Class Diagram Ristrutturato 17](#_Toc39327)

[5. Elementi di Riuso 17](#_Toc39328)

[5.1 Design Pattern Usati 17](#_Toc39329)

[6. Glossario 19](#_Toc39330)

# Introduzione

Il progetto di Ticket4You, si propone di rivoluzionare l'esperienza di acquisto e gestione dei biglietti per eventi in Italia. Questa iniziativa nasce dalla consapevolezza che attualmente solo il 40% degli utenti utilizza piattaforme digitali per l'acquisto di biglietti, e il nostro obiettivo è colmare questa lacuna, offrendo un sistema avanzato e intuitivo. La missione di Ticket4You è quella di fornire un servizio completo e interattivo, rendendo la gestione degli eventi più accessibile, efficiente e coinvolgente per gli utenti.

L'obiettivo finale è far diventare Ticket4You una sana abitudine quotidiana per gli utenti, contribuendo così ad aumentare la partecipazione agli eventi e arricchire l'offerta culturale e di intrattenimento in Italia.

## Linee guida per la scrittura del codice

Le linee guida prevedono una serie di convenzioni e di canoni che gli sviluppatori dovrebbero procedere a rispettare nella progettazione delle interfacce del sistema. Per la definizione e costituzione delle linee guida si è provveduto a fare riferimento alla convenzione Java nota come **Sun Java Coding Conventions.**

#### Link a documentazione ufficiale sulle convenzioni

Di seguito una lista di link alle convenzioni usate per definire le linee guida:

* **Java Sun:** https://checkstyle.sourceforge.io/sun\_style.html
* **HTML:** <https://www.w3schools.com/html/html5_syntax.asp>

## Definizione, acronimi e abbreviazioni

Di seguito, sono riportate alcune definizioni presenti nel documento:

* **Package:** raggruppamento di classi, interfacce o file correlati;
* **Design pattern:** template di soluzioni a problemi ricorrenti, impiegati per ottenere riuso e

flessibilità;

* **Interfaccia:** insieme di signature delle operazioni offerte dalla classe;
* **Javadoc:** sistema di documentazione offerto da Java, che viene generato sottoforma di interfaccia in modo da rendere la documentazione accessibile e facilmente leggibile;

## Riferimenti

Di seguito una lista di riferimenti ad altri documenti utili durante la lettura:

* RAD
* SDD
* Test Plan
* Javadoc di Ticket4You

# Packages

In questa sezione viene mostrata la suddivisione del sistema in package, in base a quanto definito nel documento di System Design. Tale suddivisione è motivata dalle scelte architetturali prese e ricalca la struttura di directory standard definita da Maven.

* **src**

### **Main**

* + - **Java,** contiene il package model e il package controller.
    - **Webapp,** contiene le risorse statiche (immagini, etc…), gli script JavaScript, la cartella META-INF, e la cartella WEB-INF che contiene a sua volta le pagine del sito e il file web.xml.
  + **Test,** contiene le classi adibite al testing.

### **Package di Ticket4You**

Nella seguente sezione si procede a mostrare e ad evidenziare la struttura del package principale di Ticket4You. La struttura prevede tre package ottenuta da tre principali scelte:

1. **package Controller:** contiene i sottosistemi per gestire l’accesso, la registrazione, l’evento, i titoli, il calendario e le funzionalità di ricerca di un contenuto;
2. **package Model:** contiene le classi Entity ed i DAO per l’accesso al Database;

*Immagine che contiene Rettangolo, diagramma, linea, Piano

Descrizione generata automaticamente*

*Immagine che contiene testo, schermata, Rettangolo, diagramma

Descrizione generata automaticamente*

*Immagine che contiene testo, schermata, Rettangolo, diagramma

Descrizione generata automaticamente*

*Immagine che contiene testo, schermata, Rettangolo, diagramma

Descrizione generata automaticamente*

*Immagine che contiene testo, schermata, Rettangolo, diagramma

Descrizione generata automaticamente*

Immagine che contiene testo, schermata, Rettangolo, diagramma

Descrizione generata automaticamente

# Class Interfaces Si presentano le interfacce di ogni classe:

### **Javadoc di Ticket4You**

Per motivi di leggibilità si è scelto di creare una repository, hostata tramite GitHub, contenente la JavaDoc di Ticket4You. In tale maniera, chiunque può consultare la documentazione aggiornata dell’intero sistema.

Di seguito, il link al sito in questione: https://github.com/tre02/progetto\_is\_nc16.git

**1.Package Login**

***it.unisa.Ticket4You.login.Login***

|  |  |
| --- | --- |
| **Nome classe** | **LoginServlet** |
| **Descrizione** | La seguente classe consente di gestire le operazioni relativamente l’accesso dell’utente alla piattaforma. |
| **Metodi** | +doPost(HttpServletRequest request,  HttpServletResponse response): void  +doLogin(String email, String password): Utente |
| **Invariante di classe** | / |

|  |  |
| --- | --- |
| **Nome Metodo** | **+doPost(HttpServletRequest request,**  **HttpServletResponse response)** |
| **Descrizione** | Il seguente metodo gestisce la richiesta dell’utente di accesso alla piattaforma |
| **Pre-condizione** | **context**: LoginServlet::  +doPost(HttpServletRequest request, HttpServletResponse response) **pre:** request.getParameter("email") !=null and  request.getParameter("password") !=null |
| **Post-condizione** | **context**: LoginServlet::  +doPost(HttpServletRequest request, HttpServletResponse response)  **post:** / |
| **Nome Metodo** | +doLogin(String email, String password) |
| **Descrizione** | Il seguente metodo provvede a verificare l’esistenza dell’account dell’utente all’interno del database. |
| **Pre-condizione** | **context:** LoginServlet:: doLogin(String email, String password)  **pre**: email!=null && password!=null |
| **Post-condizione** | **context:** LoginServlet:: doLogin(email,password) **post:** UtenteDAO.doLogin(email, password)!=null |

**1.Package Registrazione**

***it.unisa.Ticket4You.registrazione.Registrazione***

|  |  |
| --- | --- |
| **Nome classe** | **RegistrazioneServlet** |
| **Descrizione** | La seguente classe consente di gestire le operazioni relativamente la registrazione dell’utente alla piattaforma. |
| **Metodi** | + doPost(HttpServletRequest request,  HttpServletResponse response): void  +doRetriveUtente(): Utente |
| **Invarianti di classe** | / |

|  |  |
| --- | --- |
| **Nome Metodo** | **doPost(HttpServletRequest request, HttpServletResponse response)** |
| **Descrizione** | Il seguente metodo gestisce la richiesta di registrazione dell’account dell’utente alla piattaforma. |
| **Pre-condizione** | **context:** RegistrazioneServlet:: doPost(HttpServletRequest request, HttpServletResponse response)  **pre**: request.getParameter("username") !=null and  request.getParameter("firstName")!=null and  request.getParameter("lastName")!=null and  Date.valueOf(request.getParameter("birthday"))!=null and  request.getParameter("email")!=null and  request.getParameter("password")!=null and  request.getParameter("confirmPassword")!=null and  request.getParameter("phone")!=null |
| **Post-condizione** | **context:** RegistrazioneServlet:: doPost(HttpServletRequest request, HttpServletResponse response) **post**: Utente u !=null |
| **Nome Metodo** | +doRegistrazione(Utente utente) |
| **Descrizione** | Questo metodo consente la registrazione di un nuovo utente |
| **Pre-condizione** | **context:** RegistrazioneService:controlloEmail(email) **post:** controlloEmail(email)==false |
| **Post-condizione** | **context:** RegistrazioneService:: doRegistrazione(Utente utente)  **post:** / |

|  |  |
| --- | --- |
| **Nome Metodo** | +controlloEmail(String email): boolean |
| **Descrizione** | Questo metodo consente di verificare che l’indirizzo email inserito al momento della registrazione non sia già presente. |
| **Pre-condizione** | **context:** RegistrazioneService:: controlloEmail(String email)  **pre:** email!=null |
| **Post-condizione** | **context:** RegistrazioneService:: controlloEmail(String email)  **post:** controlloEmail(email)==false |
| **Nome Metodo** | +doRetriveUtente() |
| **Descrizione** | Questo metodo consente di ottenere tutti gli utenti registrati. |
| **Pre-condizione** | **context:** RegistrazioneServlet:: doRetriveUtente()  **pre:** / |
| **Post-condizione** | **context:** RegistrazioneServlet:: doRetriveUtente()  **post:** / |

**1.Package Evento**

***it.unisa.Ticket4You.evento.Evento***

|  |  |
| --- | --- |
| **Nome classe** | **RicercaServlet** |
| **Descrizione** | La seguente servlet consente di gestire le operazioni di ricerca degli eventi |
| **Metodi** | + doGet(HttpServletRequest request,  HttpServletResponse response): void |
| **Invarianti di classe** | / |

|  |  |
| --- | --- |
| **Nome Metodo** | **+doGet(HttpServletRequest request,**  **HttpServletResponse response)** |
| **Descrizione** | Il seguente metodo procede ad effettuare l’opportuna ricerca immessa sulla base dei valori sottomessi dall’utente. |
| **Pre-condizione** | **context:** RicercaServlet: doGet(HttpServletRequest request, HttpServletResponse response)  **pre:** request.getParameter(“selectionInput”)!=null && request.getParameter(“searchBar”)!=null |
| **Post-condizione** | **context:** RicercaServlet:: doGet(HttpServletRequest request, HttpServletResponse response)  **post:** / |

|  |  |
| --- | --- |
| **Nome classe** | **EventoInformationServlet** |
| **Descrizione** | La seguente servlet consente di visualizzare la scheda informativa degli eventi selezionati dall’utente |
| **Metodi** | EventoInformationServlet:  +doGet(HttpServletRequest request,  HttpServletResponse response): void |
| **Invarianti di classe** | / |

|  |  |
| --- | --- |
| **Nome Metodo** | **+doGet(HttpServletRequest request,**  **HttpServletResponse response)** |
| **Descrizione** | Il seguente metodo procede a mostrare la scheda informativa dell’evento selezionato dall’utente |
| **Pre-condizione** | **context: Evento**InformationServlet: doGet(HttpServletRequest request, HttpServletResponse response) **pre:** request.getSession().getAttribute("eventi")!=null |
| **Post-condizione** | **context:** EventoInformationServlet: doGet(HttpServletRequest request, HttpServletResponse response) **post:** Evento evento!=null |

|  |  |
| --- | --- |
| **Nome Metodo** | **+doGetAddToFav (HttpServletRequest request,**  **HttpServletResponse response)** |
| **Descrizione** | Il seguente metodo procede ad aggiungere agli eventi preferiti l’evento selezionato. |
| **Pre-condizione** | **context:** EventiPreferitiInfoServlet:  +doGetAddToFav (HttpServletRequest request,  HttpServletResponse response)  **pre:** session.getAttribute("FavEvents")!= null |
| **Post-condizione** | **context:** EventiPreferitiInfoServlet: doGet(HttpServletRequest request, HttpServletResponse response)  **post:** / |
| **Nome Metodo** | +doGetAddFavourite (HttpServletRequest request,  HttpServletResponse response) |
| **Descrizione** | Il seguente metodo provvede ad aggiungere gli eventi preferiti selezionati alla sezione dei *“preferiti”.* |
| **Pre-condizione** | **context: EventiPreferiti**InfoServlet:  +doGetAddToFavourite(HttpServletRequest request, HttpServletResponse response) **pre:** session.getAttribute("FavEvents")!= null |
| **Post-condizione** | **context:** EventiServlet: doGet(HttpServletRequest request, HttpServletResponse response) **post:** / |
| **Nome Metodo** | + isExisting(String nomeEvento,List<Evento> EventiList): int |
| **Descrizione** | Il seguente metodo provvede a verificare sulla base del nome se un Evento risulta essere già presente nella libreria del lettore. |
| **Pre-condizione** | **context: EventiPreferiti**InfoServlet: +isExisting(String nome,List<Evento> EventiList): int **pre: Eventi**List!=null  AND  nome!=null |
| **Post-condizione** | **context: EventiPreferiti**InfoServlet: +isExisting(String nome,List<Evento> EventiList): int  **post:** index==i or index==-1 |

**1.Package Titolo**

***it.unisa.Ticket4You.titolo.Titolo***

|  |  |
| --- | --- |
| **Nome classe** | **TitoloServlet** |
| Descrizione | Questa Servlet si occupa di gestire i metodi riferiti ai Titoli. |
| Metodi | TitoloServlet:  +doGet(HttpServletRequest request,  HttpServletResponse response) |
| Invarianti di classe | / |

|  |  |
| --- | --- |
| **Nome Metodo** | **+doGet(HttpServletRequest request,**  **HttpServletResponse response)** |
| **Descrizione** | Il seguente metodo procede a ritornare il Titolo da utilizzare. |
| **Pre-condizione** | **context: Titolo**Servlet: doGet(HttpServletRequest request, HttpServletResponse response) **pre:** request.getSession().getAttribute("Id")!=null |
| **Post-condizione** | **context:** TitoloServlet: doGet(HttpServletRequest request, HttpServletResponse response) **post:Titolo titolo**!=null |

**1.Package Notifiche**

***it.unisa.Ticket4You.notifiche.Notifiche***

|  |  |
| --- | --- |
| **Nome classe** | **NotificaServlet** |
| Descrizione | Questa Servlet si occupa di gestire i metodi riferiti alle Notifiche. |
| Metodi | NotificaServlet:  +doGet(HttpServletRequest request,  HttpServletResponse response)  +sendNotifica(HttpServletRequest request,  HttpServletResponse response) |
| Invarianti di classe | / |

|  |  |
| --- | --- |
| **Nome Metodo** | **+doGet(HttpServletRequest request,**  **HttpServletResponse response)** |
| **Descrizione** | Il seguente metodo procede ritornare la Notifica da utilizzare. |
| **Pre-condizione** | **context: Notifica**Servlet: doGet(HttpServletRequest request, HttpServletResponse response) **pre:** request.getSession().getAttribute("Notifica")!=null |
| **Post-condizione** | **context:** TitoloServlet: doGet(HttpServletRequest request, HttpServletResponse response) **post:** Notifica notifica!=null |

|  |  |
| --- | --- |
| **Nome Metodo** | **+sendNotifica (HttpServletRequest request,**  **HttpServletResponse response, Boolean tipo)** |
| **Descrizione** | Il metodo invia una notifica via mail all’utente che ha messo l’Evento nei preferiti, distinguendo il tipo di notifica (Terminazione biglietti -> tipo == true, Evento imminente (-24h) -> tipo == false) |
| **Pre-condizione** | **context: Notifica**Servlet: sendNotifica(HttpServletRequest request, HttpServletResponse response, Boolean tipo) **pre:** request.getSession().getAttribute("Notifica")!=null and  Boolean tipo!=null |
| **Post-condizione** | **context:** NotificaServlet: doGet(HttpServletRequest request, HttpServletResponse response, Boolean tipo) **post:**  Notifica notifica!=null |

# Class Diagram Ristrutturato

Immagine che contiene testo, Parallelo, diagramma, schermata

Descrizione generata automaticamente

# Elementi di Riuso

## Design Pattern Usati

Nella sezione successiva, sarà delineato un design pattern impiegato per lo sviluppo di Ticket4You. Il design pattern sarà introdotto con una breve descrizione, evidenziando la problematica risolta nel contesto applicativo di Ticket4You, seguita da una breve spiegazione della soluzione integrata nello sviluppo. In conclusione, sarà presentato un grafico della struttura delle classi che implementano il pattern.

* Il design pattern sarà introdotto da una breve descrizione;
* Problematica da risolvere per il contesto applicativo di Ticket4You;
* Breve spiegazione della risoluzione della problematica presentata nello sviluppo applicativo di Ticket4You;
* Un grafico della struttura delle classi, che ne implementano il pattern;

**Observer**

Il design pattern Observer è un modello comportamentale che permette la comunicazione tra oggetti in un modo non accoppiato. È costituito da un soggetto che tiene traccia degli osservatori interessati ai suoi cambiamenti di stato e notifica automaticamente gli osservatori quando avviene un cambiamento.

Nel contesto di Ticket4You, la webapp deve gestire in modo efficiente gli eventi e i biglietti associati. Tuttavia, è essenziale che l'interfaccia utente sia sempre aggiornata con le informazioni più recenti sugli eventi disponibili e sui biglietti venduti. Inoltre, il sistema deve essere flessibile, consentendo l'aggiunta di nuovi tipi di eventi senza dover modificare pesantemente il codice.

Per affrontare questa problematica, Ticket4You implementerà il design pattern Observer. Il gestore degli eventi fungerà da soggetto osservato, mentre l'interfaccia utente (ad esempio, una pagina web) agirà come osservatore. Quando un nuovo evento viene aggiunto o lo stato di un evento cambia (ad esempio, vendita di biglietti), il gestore degli eventi notificherà automaticamente l'interfaccia utente, consentendole di aggiornare dinamicamente la visualizzazione degli eventi.

*Immagine che contiene testo, schermata

Descrizione generata automaticamente*

# Glossario

|  |  |
| --- | --- |
| **Sigla/Termine** | **Definizione** |
| Package | Raggruppamento di classi ed interfacce. |
| Applicazione Web | Applicazione accessibile attraverso web per mezzo di una rete come ad esempio Internet. |
| Observer pattern | Il design pattern Observer è un modello comportamentale che permette la comunicazione tra oggetti in un modo non accoppiato. È costituito da un soggetto che tiene traccia degli osservatori interessati ai suoi cambiamenti di stato e notifica automaticamente gli osservatori quando avviene un cambiamento. |