ODD
Object Design Document
Stay Up

Riferimento	NC8_ODD_ver0.1
Versione	1.0
Data	30/12/2023
Destinatario	Studenti di Ingegneria del Software 2023/24
Presentato da	Abbate Andrea, Annunziata Nicola, Della Corte Gaetano, Sulipano Raffaele
Approvato da	

Revision History

Data	Versione	Descrizione	Autori
26/12/2023	0.1	Prima Stesura	GD
27/12/2023	0.2	Creazione package diagrams e stesura paragrafo 2	GD
27/12/2023	0.3	Stesura class interfaces per i package autenticazione, prenotazione scheda di allenamento e comunicazione con Personal Trainer	GD
28/12/2023	0.4	Stesura class interfaces per i package registrazione	GD
28/12/2023	0.5	Completamento capitolo 1	GD
29/12/2023	0.6	Aggiornamento class interfaces e scrittura glossario.	GD
29/12/2023	0.7	Aggiornamento class interfaces e scrittura glossario.	GD
30/12/2023	0.8	Aggiunto link al sito javadoc	GD
30/12/2023	1.0	Revisione	Tutto il Team

Team Members

Nome	Ruolo progetto	Acronimo	Informazione di contatto
Giammaria Giordano	Project Manager	GG	giagiordano@unisa.it
Andrea Abbate	Team Member	AA	a.abbate20@studenti.unisa.it
Nicola Annunziata	Team Member	NA	n.annunziata12@studenti.unisa.it
Gaetano Della Corte	Team Member	GD	g.dellacorte13@studenti.unisa.it
Raffaele Sulipano	Team Member	RS	r.sulipano@studenti.unisa.it

Sommario

1 Introduzione

Il sistema che si vuole realizzare ha come obiettivo principale quello di facilitare l'affaccio a nuove persone nel mondo del fitness. Attraverso la piattaforma online viene fornito al nuovo utente registrato la possibilità di cercare il personal trainer adatto alle sue esigenze. Grazie a questo sistema, agevoliamo un nuovo utente iscritto in palestra a confrontarsi subito con esperti del settore e rendere la sua esperienza nel mondo del fitness soddisfacente e con grandi risultati grazie ad una scheda mirata e un'alimentazione sana.
Il sistema ha come obiettivo

1. Facilitare la Registrazione e la Gestione del Profilo Utente:
   Consentire agli utenti di registrarsi facilmente sulla piattaforma, fornendo informazioni personali e preferenze.
2. Gestione degli Abbonamenti:
   Implementare un sistema che permetta agli utenti di visualizzare, scegliere e sottoscrivere abbonamenti offerti dai personal trainer.
3. Gestione delle Sessioni di Allenamento:
   Consentire ai personal trainer di pianificare e gestire sessioni di allenamento per gli utenti.
4. Sicurezza e Privacy:
   Garantire la sicurezza dei dati personali degli utenti e delle transazioni finanziarie.
5. Accessibilità e User Experience:
   Creare un'interfaccia utente intuitiva e accessibile che favorisca un'esperienza positiva per tutti gli utenti, indipendentemente dalle loro competenze tecniche.
6. Aggiornamenti e Manutenzione:
   Pianificare e implementare procedure di manutenzione regolari per garantire il corretto funzionamento del sistema e apportare eventuali miglioramenti o aggiornamenti.

1.1 Object design goals

Affidabilità vs Tempi di risposta: il sistema garantirà affidabilità sui dati, che saranno sempre consistenti all'interno del nostro database, anche nel caso in cui questa scelta comporti eventuali prolungamenti dei tempi di risposta.
Manutenibilità vs Performance: l'implementazione del sistema favorirà la manutenibilità in modo da permettere allo sviluppatore di apportare modifiche in modo più preciso ed efficace. Quindi si preferirà una buona strutturazione del codice, puntando ad un basso accoppiamento, anche nel caso in cui questo peggiori le performance.
Leggibilità vs Modificabilità: Il nostro sistema prediligerà leggibilità piuttosto che modificabilità, per cui ogni eventuale modifica implementativa, dovrà sempre attenersi strettamente ai criteri di leggibilità individuati in fase di System Design.
Costi vs Estensibilità: Il rispetto dei costi stabiliti prevarrà sull'estensibilità. Quindi, al fine di rispettare i tempi di rilascio, probabilmente non saranno presenti nella prima release le funzionalità di sistema associate a priorità più basse.

1.2 Linee guida per la documentazione dell'interfaccia

Le linee guida includono una lista di regole che gli sviluppatori dovrebbero rispettare durante la progettazione delle interfacce. Per la loro costruzione si è fatto riferimento alla convenzione java nota come **Sun Java Coding Conventions [**Sun, 2009**].**
Link a documentazioni ufficiali sulle convenzioni
Di seguito i link alle convenzioni usati per definire le linee guida:

- **Java Sun https://checkstyle.sourceforge.io/sun_style.html**
- **HTML: https://www.w3schools.com/html/html5_syntax.asp**

1.3 Definizioni, acronimi, e abbreviazioni

- **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;**
- **View:** nel pattern MVC rappresenta ciò che viene visualizzato a schermo da un utente e che gli permette di interagire con le funzionalità offerte dalla piattaforma;
- **Javadoc:** sistema di documentazione offerto da Java, che viene generato sottoforma di interfaccia in modo da rendere la documentazione accessibile e facilmente leggibile.

1.4 Riferimenti

Lista di riferimento ad altri documenti utili durante la lettura:

- Statement of Work.docx

2 Packages

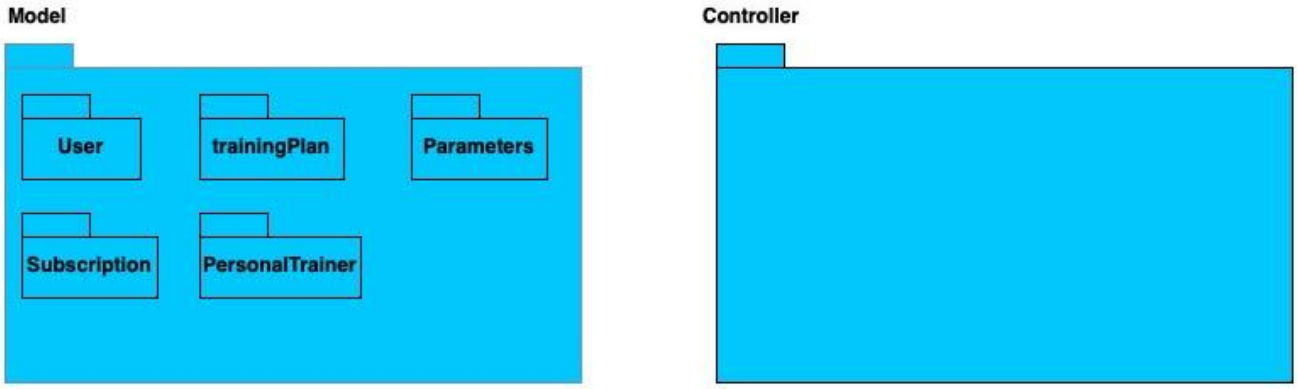
In questa sezione viene mostrata la suddivisione del sistema in package, in base a quanto definito nel documento di System Design. il tutto condizionato dall'architettura MVC e da quella imposta da Maven.

.idea

- **.mvn,** contiene tutti i file di configurazione per Maven
- **src,** contiene tutti i file sorgente

- **main**
- **java,** contiene le classi java relative alle componenti Control e Model
- **webapp:** file relativi alle componenti View del sistema
- **css:** contiene i file CSS
- **test,** contiene tutto il necessario per il testing
- **java,** contiene le classi Java per l'implementazione del testing
- **target,** contiene tutti i file prodotti del sistema di build di Maven.

Prestiamo maggiore attenzione alla directory src/main/java ed alla sua struttura in quanto rappresenta il core del progetto.
Mostriamo una visione generale dei package che ne fanno parte:



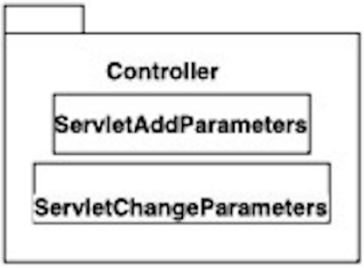
Package:

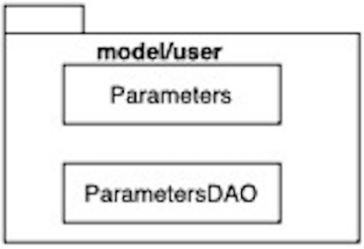
- User
- TrainingPlan
- Parameters
- Subscription
- PersonalTrainer

Ogni package contiene la logica di business per la gestione delle componenti e la gestione di esse nel database. Ognuno contiene dei package i quali hanno le seguenti funzionalità:

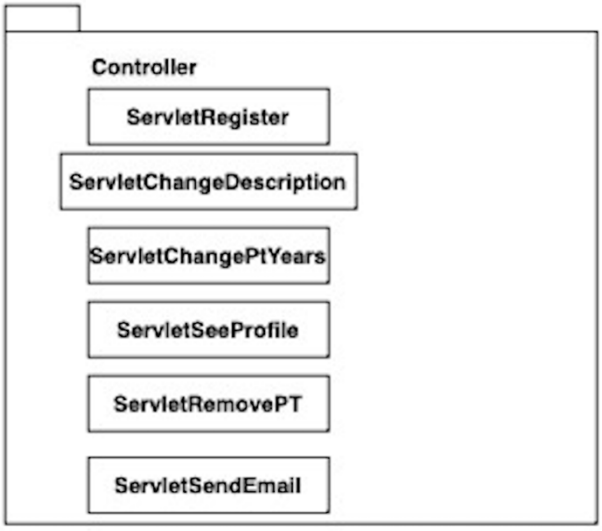
Package controler: contiene le servlet
Package Model: contine i bean ed i DAO.

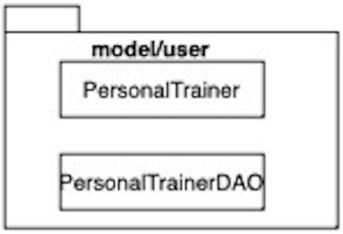
Package Parameters



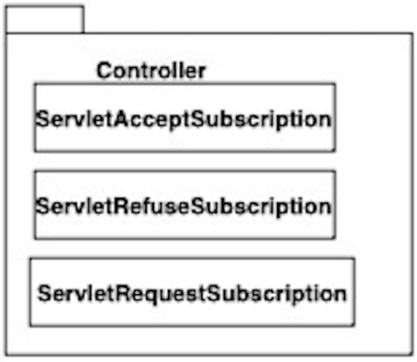


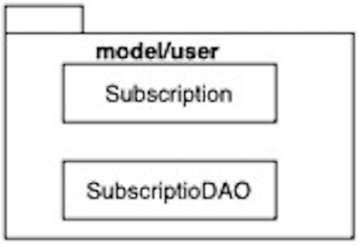
Package Personal Trainer



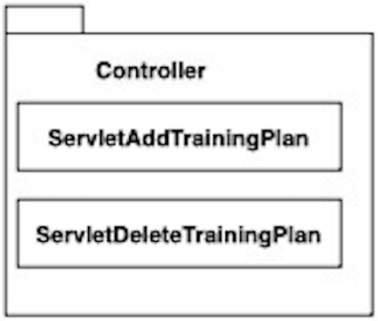


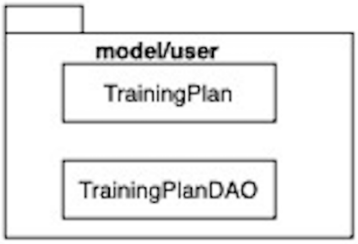
Package Subscription



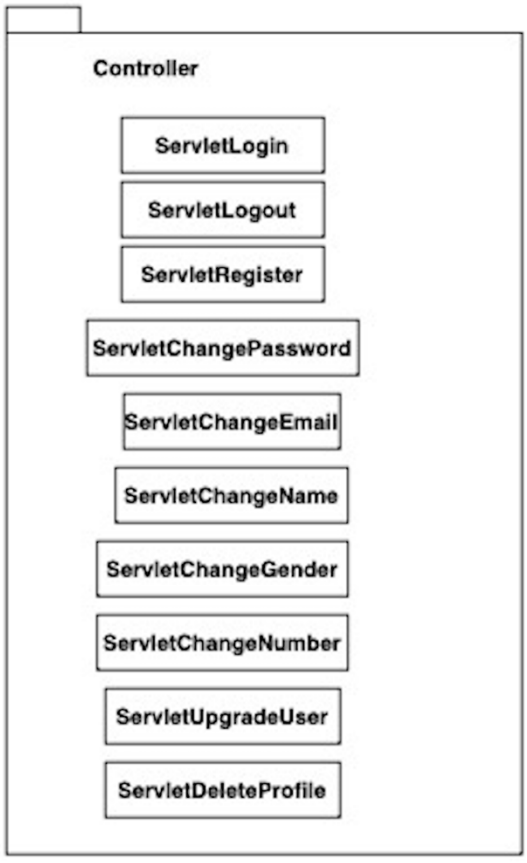


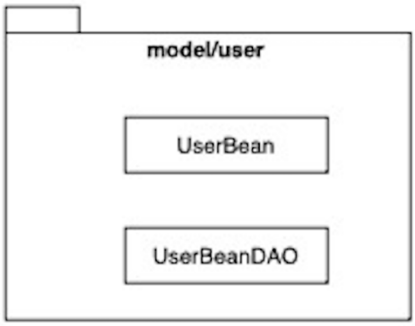
Package Training Plan





Package UserBean





3 Class Interfaces

Di seguito saranno presentate le interfacce di ciascun package.

1 Package Parameters

Nome Classe	Parameters.java
Descrizione	**Questa classe permette di gestire i parametri di un utente.**

Nome Metodo	**getParameters(String email)** : Parameters
Descrizione	**Questo metodo consente di ottenere e restituire i parametri specifici associati all'iscrizione di un atleta e alla selezione di un personal trainer.**
Pre-condizione	**email != null**
Post-condizione	

Nome Metodo	**setParameters(String email, ArrayList<Double> params)** : Parameters
Descrizione	**Questo metodo consente di accettare i nuovi dati forniti dall'utente durante il processo di iscrizione o di aggiornamento delle informazioni personali e di applicare tali modifiche al sistema.**
Pre-condizione	**email != null && !params.isEmpty()**
Post-condizione	

Nome Metodo	**changeParameters(Parameters params)** : Parameters
Descrizione	**Questo metodo consente all'utente di modificare specifici parametri associati alla propria iscrizione o alla scelta del personal trainer.**
Pre-condizione	**params != null**
Post-condizione	

Nome Metodo	**checkParams(Parameters params)** : Boolean
Descrizione	**Questo metodo controlla i pattern dei parameteri inseriti ritornando false nel caso un parametro non rispetti il pattern, true se tutti i pattern sono stati rispettati.**
Pre-condizione	**params != null**
Post-condizione	

2 Package Personal Trainer

Nome Classe	personaltrainer.java
Descrizione	**Questa classe permette di gestire le operazioni relative ad un personal trainer**

Nome Metodo	**personalTrainerRegistration(UserBean ub)** : PersonalTrainer
Descrizione	**Questo metodo ha lo scopo di consentire ai personal trainer di registrarsi sulla piattaforma, rendendo disponibili i propri servizi agli atleti.**
Pre-condizione	**ub != null**
Post-condizione	

Nome Metodo	**changeDescription(PersonalTrainer pt, String newDescription) :** PersonalTrainer
Descrizione	**Questo metodo consente ai Personal Trainer di aggiornare e personalizzare le informazioni associate al proprio profilo sulla piattaforma, migliorando la visibilità e la precisione delle informazioni.**
Pre-condizione	**pt != null && newDescription !=null**
Post-condizione	

Nome Metodo	**changePTYears(PersonalTrainer pt, Integer newYears) :** PersonalTrainer
Descrizione	**Questo metodo consente al personal trainer di aggiornare il numero di anni di esperienza nel settore della formazione personale.**
Pre-condizione	**pt != null && newYears >= 0**
Post-condizione	

Nome Metodo	**retrieveInfo(String email)** : PersonalTrainer
Descrizione	**Questo metodo consente di ottenere e restituire le informazioni dettagliate associate al profilo di un personal trainer.**
Pre-condizione	**email != null**
Post-condizione	

Nome Metodo	**retrieveAll()** : List<PersonalTrainer>

Descrizione	**Questo metodo consente di recuperare e restituire un elenco completo di tutti i personal trainer disponibili sulla piattaforma.**
Pre-condizione	
Post-condizione	

Nome Metodo	**deletePT(String email)** : Boolean
Descrizione	**Questo metodo consente la rimozione di un personal trainer dalla piattaforma.**
Pre-condizione	**email != null**
Post-condizione	

3 Package Subscription

Nome Classe	subscription.java
Descrizione	**Questa classe gestisce le informazioni legate all'iscrizione di un utente.**

Nome Metodo	**addSubscription(UserBean ub, PersonalTrainer pt, Date dateEnd)** : Subscription
Descrizione	**Questo metodo consente agli utenti di richiedere una nuova iscrizione.**
Pre-condizione	**ub != null && pt != null && dateEnd != null**
Post-condizione	

Nome Metodo	**acceptSubscription(Subscription s)** : Subscription
Descrizione	**Questo metodo viene implementato se la richiesta di iscrizione è valida, e accetta tale richiesta.**
Pre-condizione	**s != null**
Post-condizione	

Nome Metodo	**refuseSubscription(Subscription s)** : Subscription
Descrizione	**Questo metodo rappresenta un'azione nel rifiutare una richiesta di iscrizione da parte di un utente nella piattaforma.**
Pre-condizione	**s != null**
Post-condizione	

Nome Metodo	**getSubscription(UserBean ub)** : Subscription
Descrizione	**Questo metodo consente di recuperare l'istanza della sottoscrizione di un atleta specifico nel sistema, consentendo al sistema di accedere alle**

	informazioni dettagliate relative a quella particolare iscrizione.
Pre-condizione	**ub != null**
Post-condizione	

Nome Metodo	**getAllSubscription()** : ArrayList<Subscription>
Descrizione	**Questo metodo consente di ottenere un elenco completo di tutte le iscrizioni nel sistema.**
Pre-condizione	
Post-condizione	

4 Package Training Plan

Nome Classe	trainingPlan.java
Descrizione	**Questa classe consente di gestire le informazioni relative al piano di allenamento associato a un atleta nel sistema.**

Nome Metodo	**deleteTrainingPlan(String emailUser, String emailPT, String exercisesString)** : Boolean
Descrizione	**Questo metodo consente agli atleti di eliminare il piano di allenamento associato al proprio profilo sulla piattaforma**
Pre-condizione	**emailUser != null && emailPT != null && exercisesString != null**
Post-condizione	

Nome Metodo	**getAvailableTrainingPlan(UserBean ub)** : ArrayList<TrainingPlan>
Descrizione	**Questo metodo fornisce una lista di schede di allenamento di un determinato Utente dal giorno corrente in poi.**
Pre-condizione	**ub != null**
Post-condizione	

Nome Metodo	**getAvailablePtTrainingPlan(UserBean ub, PersonalTrainer pt)** : ArrayList<TrainingPlan>

Descrizione	**Questo metodo fornisce una lista di schede di allenamento di un determinato Utente e Personal Trainer dal giorno corrente in poi.**
Pre-condizione	**ub != null && pt != null**
Post-condizione	

Nome Metodo	**getAllTrainingPlans(PersonalTrainer pt)** : ArrayList<TrainingPlan>
Descrizione	**Questo metodo fornisce un elenco di tutte le schede di allenamento di un singolo Personal Trainer.**
Pre-condizione	**pt != null**
Post-condizione	

Nome Metodo	**getAllUserTrainingPlans(UserBean ub)** : ArrayList<TrainingPlan>
Descrizione	**Questo metodo restituisce un elenco completo di tutte le schede di allenamento associati a un utente specifico.**
Pre-condizione	**ub != null**
Post-condizione	

Nome Metodo	**addTrainingPlan(String emailPT, String emailUser, String exercises, Date dateStart, Date dateEnd)** : Boolean
Descrizione	**Questo metodo consente ai Personal Trainer di creare una scheda di allenamento ed associarla ad un utente.**
Pre-condizione	**emailPT != null && emailUser != null && exercises != null && dateStart != null && dateEnd != null**
Post-condizione	

Nome Metodo	**checkTrainingPlan(TrainingPlan tp)** : Boolean
Descrizione	**Questo metodo controlla i pattern dei valori inseriti ritornando false nel caso un parametro non rispetti il pattern, true se tutti i pattern sono stati rispettati.**
Pre-condizione	**tp != null**
Post-condizione	

5 Package UserBean

Nome Classe	userbean.java
Descrizione	**Questa classe è utilizzata per rappresentare e gestire le informazioni dell'utente.**

Nome Metodo	**loginUser(String email, String password)** : UserBean
Descrizione	**Questo metodo consente di loggare un utente registrato.**
Pre-condizione	**email != null && password != null**
Post-condizione	

Nome Metodo	**changeEmail(UserBean ub, String oldemail, String newemail)** : UserBean
Descrizione	**Questo metodo consente agli utenti di modificare l'indirizzo email associato al proprio account.**
Pre-condizione	**ub != null && oldemail != null && newemail != null**
Post-condizione	

Nome Metodo	**changeName(UserBean ub, String newName, String newSurname)** : UserBean
Descrizione	**Questo metodo consente agli utenti di modificare il proprio nome associato al proprio account.**
Pre-condizione	**ub != null && newName != null && newSurname != null**
Post-condizione	

Nome Metodo	**changeGender(UserBean ub, String newGender)** : UserBean
Descrizione	**Questo metodo consente di modificare il genere associato al proprio account.**
Pre-condizione	**ub != null && newGender != null**
Post-condizione	

Nome Metodo	**recoverInfos(String email)** : UserBean
Descrizione	**Questo metodo consente di recuperare le informazioni relative ad un utente.**
Pre-condizione	**email != null**
Post-condizione	

Nome Metodo	**changePsw(UserBean ub, String oldpsw, String newpsw)** : UserBean
Descrizione	**Questo metodo consente di modifica la password associata al proprio account**
Pre-condizione	**ub != null && oldpsw != null && newpsw != null**
Post-condizione	

Nome Metodo	**forgotPsw(String email)** : UserBean
Descrizione	**Questo metodo consente di recuperare la propria password nel caso in cui gli utenti l'abbiano dimenticata.**
Pre-condizione	**email != null**
Post-condizione	

Nome Metodo	**userRegistration(String email, String password, String name, String surname, String telephone, String gender)** : UserBean
Descrizione	**Questa funzionalità permette agli utenti di creare un account nel sistema.**
Pre-condizione	**email != null && password && name != null && surname != null && telephone != null && gender != null**
Post-condizione	

Nome Metodo	**checkEmail(String email)** : UserBean
Descrizione	**Questo metodo verifica se un determinato indirizzo email è già associato a un utente registrato nel sistema.**
Pre-condizione	**email != null**
Post-condizione	

Nome Metodo	**deleteUser(String uEmail)** : void
Descrizione	**Questo metodo consente l'eliminazione di un utente dal sistema.**
Pre-condizione	**uEmail != null**
Post-condizione	

Nome Metodo	**allUsers()** : ArrayList<UserBean>
Descrizione	**Questo metodo consente di recuperare e restituire una lista di tutti gli utenti nel sistema.**
Pre-condizione	
Post-condizione	

Nome Metodo	**requestRolePT(UserBean ub)** : UserBean
Descrizione	**Questo metodo può essere utilizzato quando un utente desidera di iscriversi come personal trainer e viene sottoposto a una valutazione o approvazione da parte degli amministratori del sistema.**
Pre-condizione	**ub != null**
Post-condizione	

Nome Metodo	**retrieveAllPending()** : List<UserBean>
Descrizione	**Questo metodo permette agli amministratori del sistema di visualizzare e gestire tutte le richieste in sospeso.**
Pre-condizione	
Post-condizione	

Nome Metodo	**upgradeToPT(UserBean ub)** : void
Descrizione	**Questo metodo consente di approvare e aggiornare lo stato di un personal trainer (PT).**
Pre-condizione	**ub != null**
Post-condizione	

Nome Metodo	**checkFormat(UserBean ub)** : Boolean
Descrizione	**Questo metodo controlla i pattern dei valori inseriti ritornando false nel caso un parametro non rispetti il pattern, true se tutti i pattern sono stati rispettati.**
Pre-condizione	**ub != null**
Post-condizione	

4 Design Patterns

In questa sezione vengono illustrati i vari design pattern scelti e nello specifico come agiscono e risolvono la problematica individuata.
Per ogni pattern si darà:

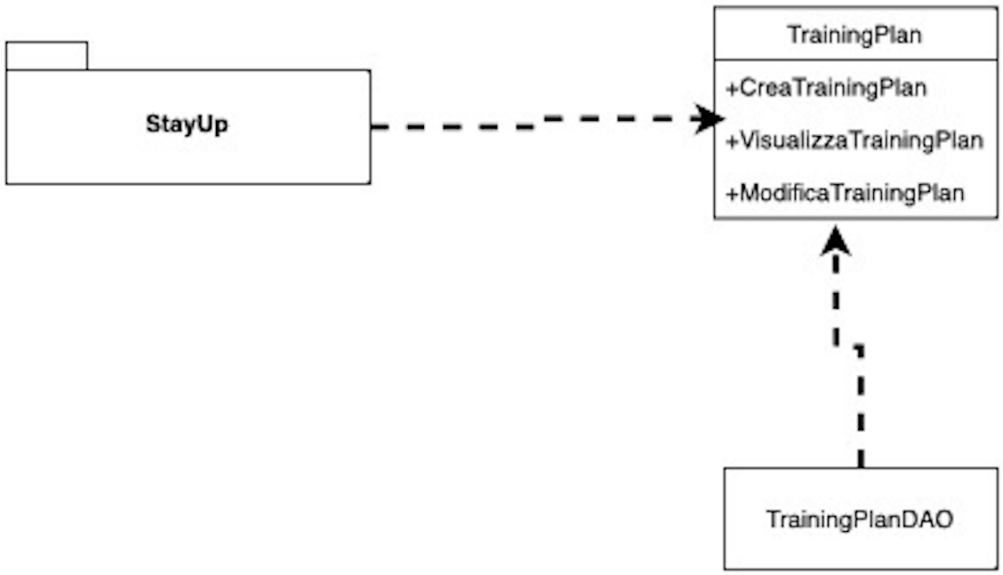
- Un'introduzione teorica.
- Il problema che doveva risolvere all'interno del sistema.
- Una spiegazione di come si è risolto il problema.
- Un grafico della struttura delle classi che implementano il pattern.

Singleton

Descrizione del problema: una delle problematiche riscontrate nel nostro progetto, è l'eccessiva istanziazione della classe relativa alla connessione al database.
Soluzione: Il Singleton, design pattern creazionale, ha lo scopo di garantire che venga creata un'unica istanza di una determinata classe. Dunque, l'istanziazione avverrà solamente alla prima chiamata della classe.
Conseguenze: I principali vincoli riguardanti tale scelta riguardano l'impossibilità di sfruttare l'ereditarietà per la classe, quindi, potrebbe esserci difficoltà in alcune scelte di testing.

Façade

Il Façade è un design pattern che permette, implementando una interfaccia semplificata di accedere a sottosistemi più complessi. In questo modo si può nascondere al sistema la complessità delle librerie, dei framework o dei set di classi che si stanno usando. Si garantisce così un alto disaccoppiamento e si rende la piattaforma più facile da aggiornare, poiché basterà cambiare l'implementazione dei metodi dell'interfaccia per implementare le modifiche. StayUp, essendo un sistema complesso, sfrutta il design pattern Façade per implementare tutta la sua logica di business e rendere più facile l'interfacciarsi con essa.
Di seguito un esempio di Façade nel sistema:



Data Access Object

Descrizione del problema: Essendo StayUp una web application che presenta un database molto vasto, quindi ha bisogno di poter interagire con database in modo rapido e sicuro con numerosi query per quella che è la moltitudine di dati da gestire. Per questo motivo utilizziamo questo pattern per snellire il codice dei beans e astrarre la logica di business dalla gestione della persistenza dei dati, così da non avere un alto accoppiamento tra essi.
Soluzione: Tramite un'interfaccia DAO si ha l'accesso al Layer di Persistenza, nascondendo all'utente i dettagli di implementazione dei metodi. Inseriamo inoltre un nuovo layer che si occupa solo della gestione dei dati persistenti, così da avere una gestione centralizzata solo per i dati persistenti.
Conseguenze: Gli oggetti di business non essendo a conoscenza dell'implementazione del DAO garantiscono una possibile migrazione verso nuovo database, comportando modifiche solo al layer di persistenza.
Si risolvono così problemi dovuti a futuri cambiamenti.
Inoltre, migliora la leggibilità e la trasparenza del codice, grazie al basso accoppiamento tra i Layer. La centralizzazione della gestione dei dati, in un livello sperato, rende più efficiente la manutenzione e la gestione dell'applicazione.

5 GLOSSARIO

Sigla/Termine	Definizione
Package	Raggruppamento di classi ed interfacce.
Controller	Classe che si occupa di gestire le richieste effettuate dal client
Service	Classe che implementa la logica di business, viene utilizzata dal controller o da un altro sottosistema.
Model	Parte del design architetturale MVC che fornisce al sistema i metodi per accedere ai dati utili.
MVC	Model-View-Controller: design architetturale che permette di separare la logica di presentazione dalla logica di business alla base del sistema.

Façade	Un oggetto che permette attraverso un'interfaccia più semplice l'accesso a sottosistemi che espongono interfacce complesse e molto diverse tra loro.
Singleton	È un design pattern creazionale che ha lo scopo di garantire che di una determinata classe venga strutturata una sola istanza e di fornire un punto di accesso globale a tale istanza