Laurea Magistrale in informatica-Università di Salerno
Corso di *Gestione dei Progetti Software*- Prof.ssa F.Ferrucci

FITDIARY

DECIDE, COMMIT, SUCCEED

Object Design Document

Riferimento	2021_ODD_C07_FitDiary_Fasano-Spinelli_V1.1
Versione	1.1
Data	17/01/2022
Destinatario	Prof.ssa Filomena Ferrucci, Prof.re Fabio Palomba
Presentato da	C07 Team FitDiary: Daniele De Marco, Ilaria De Sio, Rebecca Di Matteo, Daniele Giaquinto, Davide La Gamba, Leonardo Monaco, Simone Spera, Antonio Trapanese.
Approvato da	

Revision History

Data	Versione	Descrizione	Autori
15/12/2021	0.1	Object Design Trade-Off	Rebecca Di Matteo, Simone Spera
15/12/2021	0.2	Componenti Off-the-Shelf	Daniele Giaquinto, Leonardo Monaco
15/12/2021	1.0	Linee guida Doc Interfacce	Davide La Gamba, Ilaria De Sio
17/01/2022	1.1	Modifica descrizione Design Goals e Trade Offs	Davide La Gamba

Team Members

Ruolo	Nome e Cognome	Acronimo	Email
PM	Salvatore Fasano	SF	s.fasano10@studenti.unisa.it
PM	Gianluca Spinelli	GS	g.spinelli18@studenti.unisa.it
TM	Daniele De Marco	DM	d.demarco11@studenti.unisa.it
TM	Ilaria De Sio	IS	i.desio7@studenti.unisa.it
TM	Rebecca Di Matteo	RM	r.dimatteo10@studenti.unisa.it
TM	Daniele Giaquinto	DG	d.giaquinto2@studenti.unisa.it
TM	Davide La Gamba	DL	d.lagamba@studenti.unisa.it
TM	Leonardo Monaco	LM	l.monaco11@studenti.unisa.it
TM	Simone Spera	SS	s.spera7@studenti.unisa.it
TM	Antonio Trapanese	AT	a.trapanese8@studenti.unisa.it

Sommario

1. Introduzione

1.1 Object Design Goals

Rank	ID Design Goal	Descrizione	Origine
1	ODG_1 Robustezza	Il sistema deve essere robusto ed in caso di emergenza deve saper reagire nel modo migliore alle situazioni critiche, questo sarà garantito mediante la gestione del 90% degli errori attraverso apposite eccezioni.	DG_4
2	ODG_2 Riusabilità	Il sistema sarà riusabile per il 90% delle componenti di business logic tramite l'utilizzo di design pattern.	DG_6
3	ODG_3 Manutenibilità	Il sistema dovrà essere caratterizzato da un elevato grado di leggibilità e comprensibilità in modo da garantire interventi di manutenzione semplificati, tramite l'utilizzo dello standard Sun.	DG_6,DG_7,DG_8
4	ODG_4 Sicurezza	Il sistema dovrà fornire un elevato grado di sicurezza mediante comunicazione su protocollo https, corretta gestione di autenticazione ed autorizzazioni tramite JWT e SpringBootSecurity.	DG_5
5	ODG_5 Tempo di riposta	Il sistema dovrebbe, fornire una risposta al frontend ad una richiesta https, in meno di 1 secondo nel 95% dei casi.	DG_1

1.2 Object Design Trade-Off

Trade-off	Descrizione
Tempi di risposta vs Sicurezza	Il sistema al fine di garantire una maggiore sicurezza potrebbe richiedere un tempo di risposta maggiore che, in ogni caso, sarà mantenuto al di sotto di un secondo nell'80% dei casi tale sicurezza sarà garantita da JWT e SpringBootSecurity.
Tempi di risposta vs Robustezza	Il sistema, al fine di garantire una maggiore robustezza, effettuerà dei controlli sui vari input. Tali controlli potrebbero alterare il tempo di risposta a meno di 3 secondi nell'80% dei casi.
Costi di sviluppo vs Utilizzo COTS	Il sistema, al fine di rientrare nel budget prestabilito, farà utilizzo di COTS, per la gestione funzionalità complesse come la sicurezza e pagamenti (Stripe).

1.3 Definizioni, Acronimi e Abbreviazioni

Vengono riportati di seguito 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.

1.4 Riferimenti

Libro: -- Object-Oriented Software Engineering (Using UML, Patterns, and Java) Third Edition Autori: -- Bernd Bruegge & Allen H. Dutoit

RAD -- 2021_RAD_C07_FitDiary_Spinelli-Fasano_V1.0

SDD – 2021_SDD_C07_FitDiary_Spinelli-Fasano_V1.1

1.3 Component Off-the-Shelf

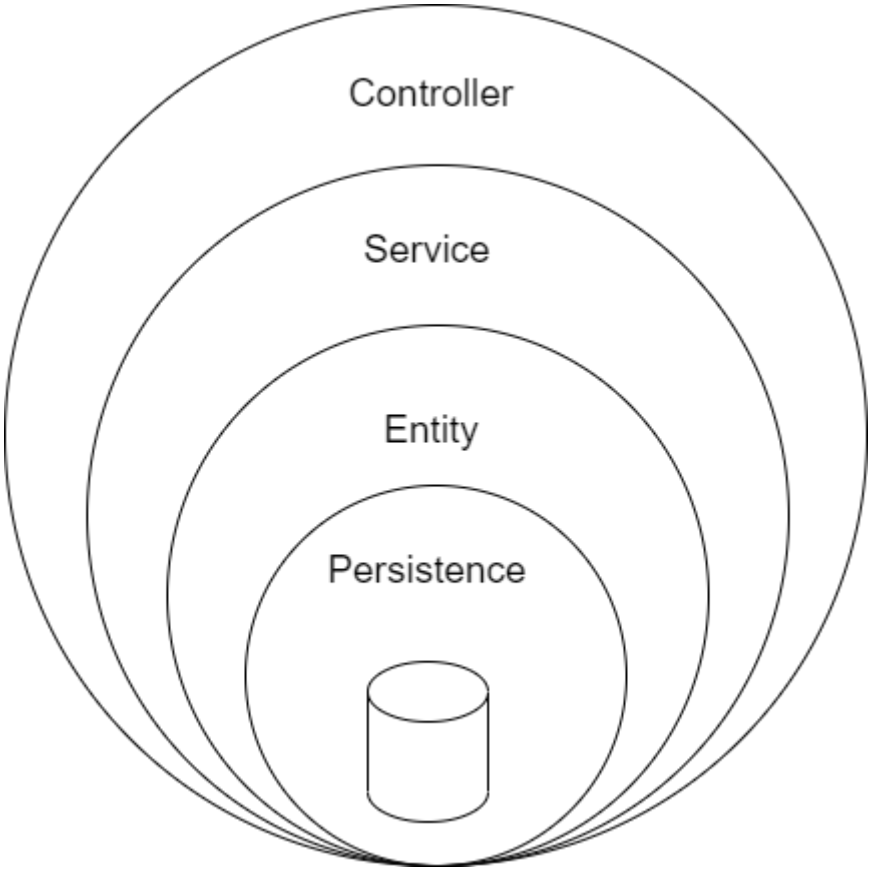
Nella realizzazione della nostra piattaforma andremo ad utilizzare componenti Off-the-Shelf già disponibili per facilitare lo sviluppo del progetto.

- Per la progettazione del lato Backend verrà utilizzato Spring Boot e l'interazione con esso avverrà tramite linguaggio di programmazione Java.
- Per la progettazione del lato Frontend verrà utilizzato ReactJS, una libreria JavaScript, per la progettazione del lato Frontend. Inoltre, per la modellazione dello stile grafico si utilizzerà la libreria Chakra UI (https://chakra-ui.com/guides/integrations/with-cra).
- I dati verranno memorizzati all'interno di un DMBS PostgreSQL mediante JPA, Spring Boot utilizza di default la tecnologia Hibernate. La connessione al DBMS è garantita dal driver JDBC.
- La gestione dei pagamenti avverrà mediante la piattaforma esterna Stripe (https://stripe.com/en-it) grazie all'API da essa fornite.
- Per la manipolazione dei documenti CSV interni al sistema, verrà utilizzato Commons-CSV (https://commons.apache.org/proper/commons-csv/).
- Per la descrizione di modelli di analisi predittiva su un modello di machine learning, verrà utilizzato PMML4S: https://github.com/autodeployai/pmml4s .

Tutte le tecnologie evidenziate possono essere utilizzate gratuitamente, in tal modo si riduce la quantità di lavoro necessario mantenendo inalterati i costi.

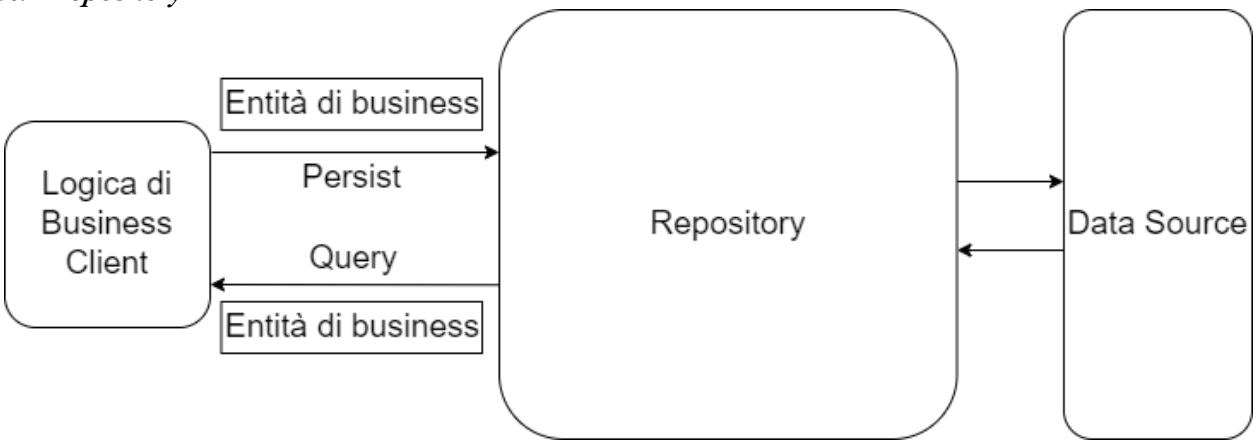
1.4 Design Patterns

1.4.1 Service Layer



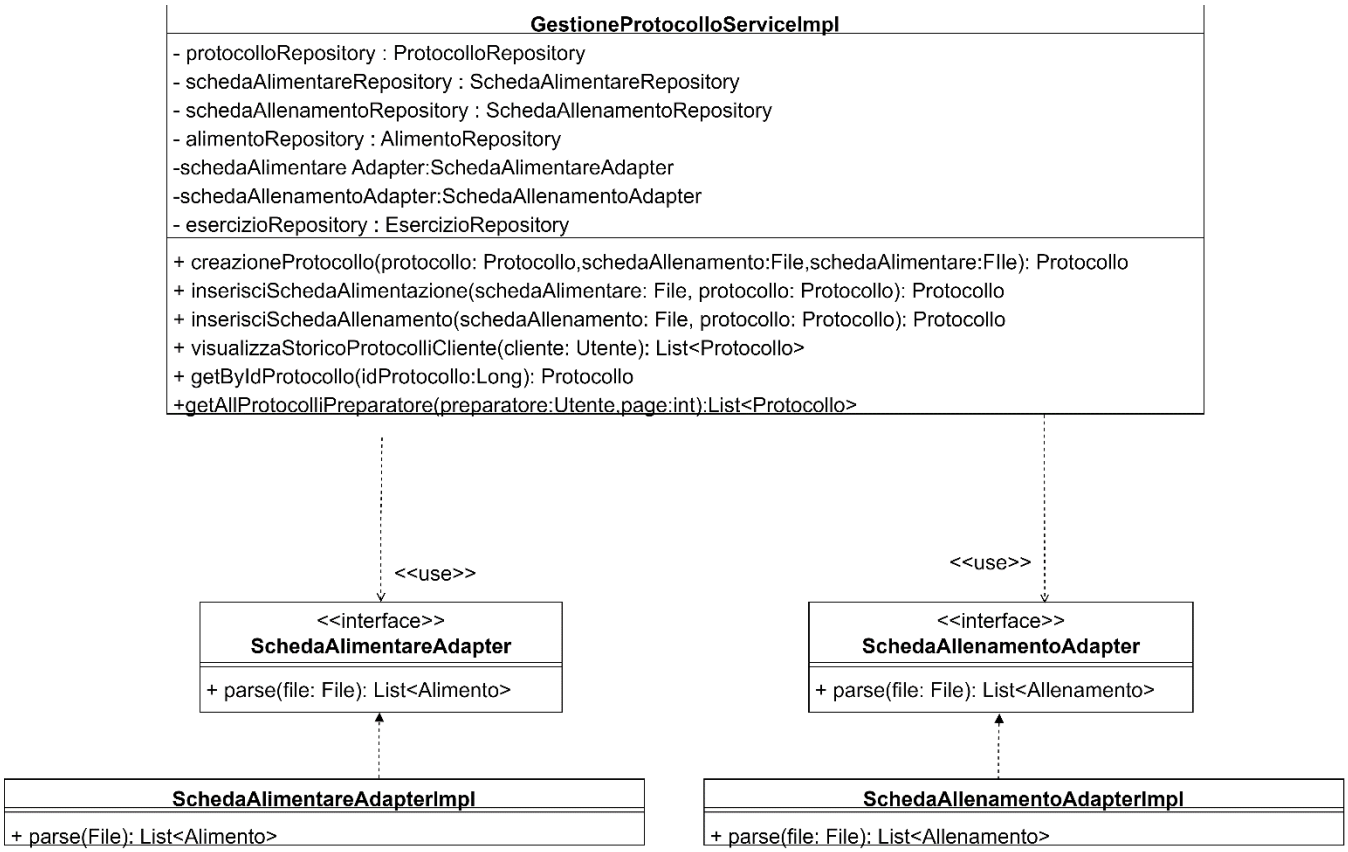
Verrà utilizzato il design pattern "Service Layer" per organizzare la suddivisione dei servizi messi a disposizione del sistema tramite un insieme di livelli logici.
In particolare, i servizi inclusi nel livello Controller sono quelli responsabili della comunicazione Client-Server, il livello Service invece si occupa della logica di business. I servizi del livello Entity saranno quelli che rappresenteranno gli oggetti di rilievo del sistema.
Infine, il livello di Persistenza sarà composto dai dati persistenti.
Questa suddivisione mira a ridurre la complessità nella gestione dei servizi.
La scelta di questo Design Pattern è dovuta alla struttura del nostro sistema, basata su sequenze di richieste di servizi tra il WebServerFrontend e il WebServerBackend, permettendoci di ottenere maggiore manutenibilità.

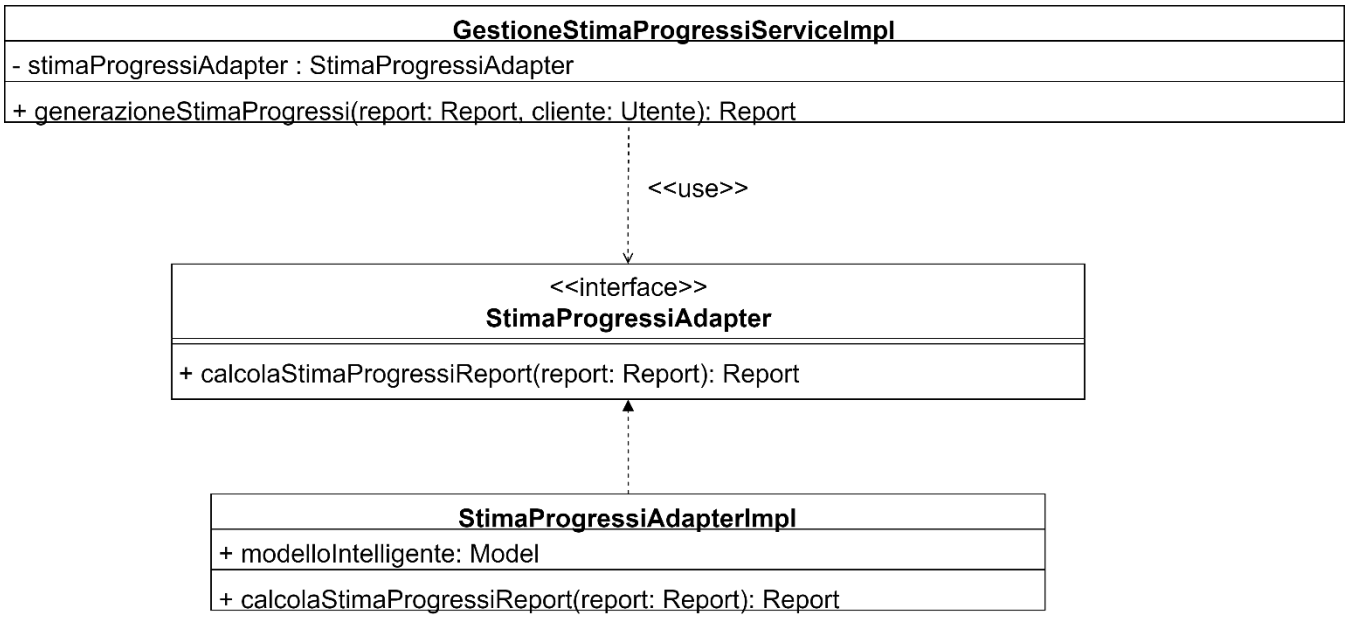
1.4.2 Repository



Dall'analisi della documentazione del framework Spring, del nostro sistema e della tecnologia che permette lo sviluppo di applicazioni Enterprise, abbiamo deciso di adottare il design pattern "Repository". Esso divide la logica di accesso ai dati e la associa alle entità di business nel livello delle Entity. La comunicazione tra la logica di accesso ai dati e la logica di business avviene tramite le interfacce.

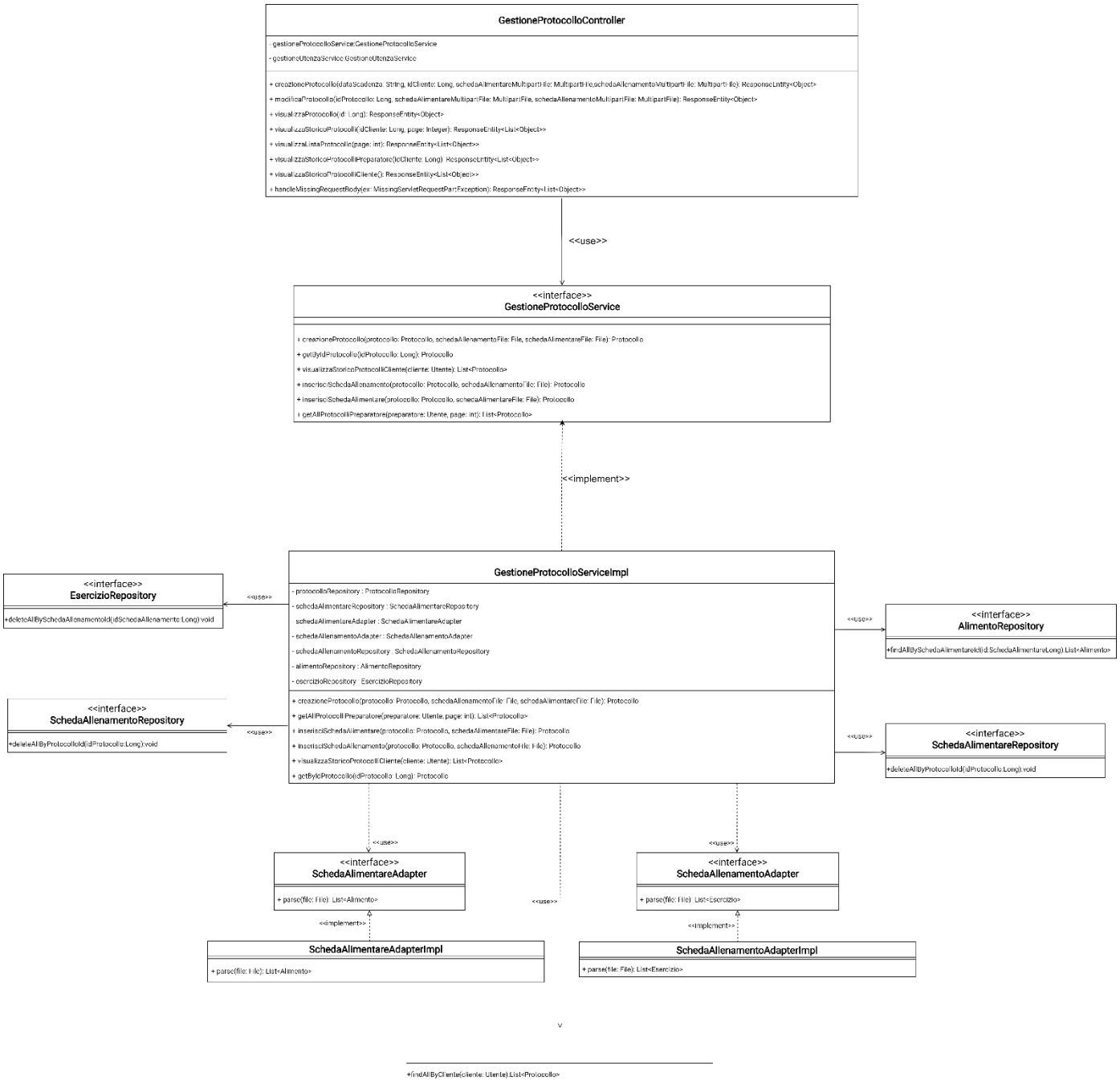
1.4.3 Adapter


GestioneProtocolloServiceImpl
- protocolloRepository : ProtocolloRepository
- schedaAlimentareRepository : SchedaAlimentareRepository
- schedaAllenamentoRepository : SchedaAllenamentoRepository
- alimentoRepository : AlimentoRepository
-schedaAlimentare Adapter:SchedaAlimentareAdapter
-schedaAllenamentoAdapter:SchedaAllenamentoAdapter
- esercizioRepository : EsercizioRepository
+ creazioneProtocollo(protocollo: Protocollo,schedaAllenamento:File,schedaAlimentare:FIle): Protocollo
+ inserisciSchedaAlimentazione(schedaAlimentare: File, protocollo: Protocollo): Protocollo
+ inserisciSchedaAllenamento(schedaAllenamento: File, protocollo: Protocollo): Protocollo
+ visualizzaStoricoProtocolliCliente(cliente: Utente): List<Protocollo>
+ getByIdProtocollo(idProtocollo:Long): Protocollo
+getAllProtocolliPreparatore(preparatore:Utente,page:int):List<Protocollo>
<<use>>
<<use>>
<<interface>>
SchedaAlimentareAdapter
+ parse(file: File): List<Alimento>
<<interface>>
SchedaAllenamentoAdapter
+ parse(file: File): List<Allenamento>
SchedaAlimentareAdapterImpl
+ parse(File): List<Alimento>
SchedaAllenamentoAdapterImpl
+ parse(file: File): List<Allenamento>


GestioneStimaProgressiServiceImpl
- stimaProgressiAdapter : StimaProgressiAdapter
+ generazioneStimaProgressi(report: Report, cliente: Utente): Report
<<use>>
<<interface>>
StimaProgressiAdapter
+ calcolaStimaProgressiReport(report: Report): Report
StimaProgressiAdapterImpl
+ modelloIntelligente: Model
+ calcolaStimaProgressiReport(report: Report): Report

Il design pattern strutturale "Adapter" ci permette di facilitare la collaborazione tra oggetti aventi interfacce differenti, tramite la creazione di classi Adapter che avranno il compito di manipolare i dati.
In particolare, utilizziamo questo pattern per convertire i dati espressi in formato csv in una delle classi del sistema, tra le quali "Scheda allenamento" e "Scheda alimentazione", in modo tale da poter essere facilmente gestibili dalle classi del nostro sistema. Inoltre, un altro Adapter sarà impiegato nella trasformazione dei risultati prodotti dal modulo di intelligenza artificiale in dati manipolabili dal sistema.

1.4.4 Facade



Il design pattern "Facade" è un pattern strutturale che definisce un'interfaccia semplificata per l'accesso ad un insieme di oggetti in un sottosistema più complesso.
Il pattern viene realizzato tramite un'interfaccia che agisce da "facciata" per fornire i metodi richiesti.
In particolare, il nostro sistema utilizza questo design pattern per ogni suo sottosistema, definendo delle interfacce "Service" utilizzabili per accedere ai metodi.
Ciò diminuisce l'accoppiamento e rende il codice facilmente manutenibile, se modifichiamo un componente del layer superiore o inferiore, non dobbiamo modificare tutto il resto.
Inoltre, ciò apporta vantaggi nell'effettuare l'Integration Testing.

1.5 Linee guida per la documentazione delle interfacce

Le linee guida per la documentazione delle interfacce contengono le convenzioni che gli sviluppatori saranno tenuti a seguire durante la progettazione e lo sviluppo delle interfacce del sistema. In particolare, per le seguenti tecnologie verranno utilizzate le indicazioni riportate nei link sottostanti:

- Java Sun e Spring: https://checkstyle.sourceforge.io/sun_style.html
- JavaScript: https://www.w3schools.com/js/js_conventions.asp

2. 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 Client Server.
Il lato Server ha una struttura di directory standard definita da Maven contenente:

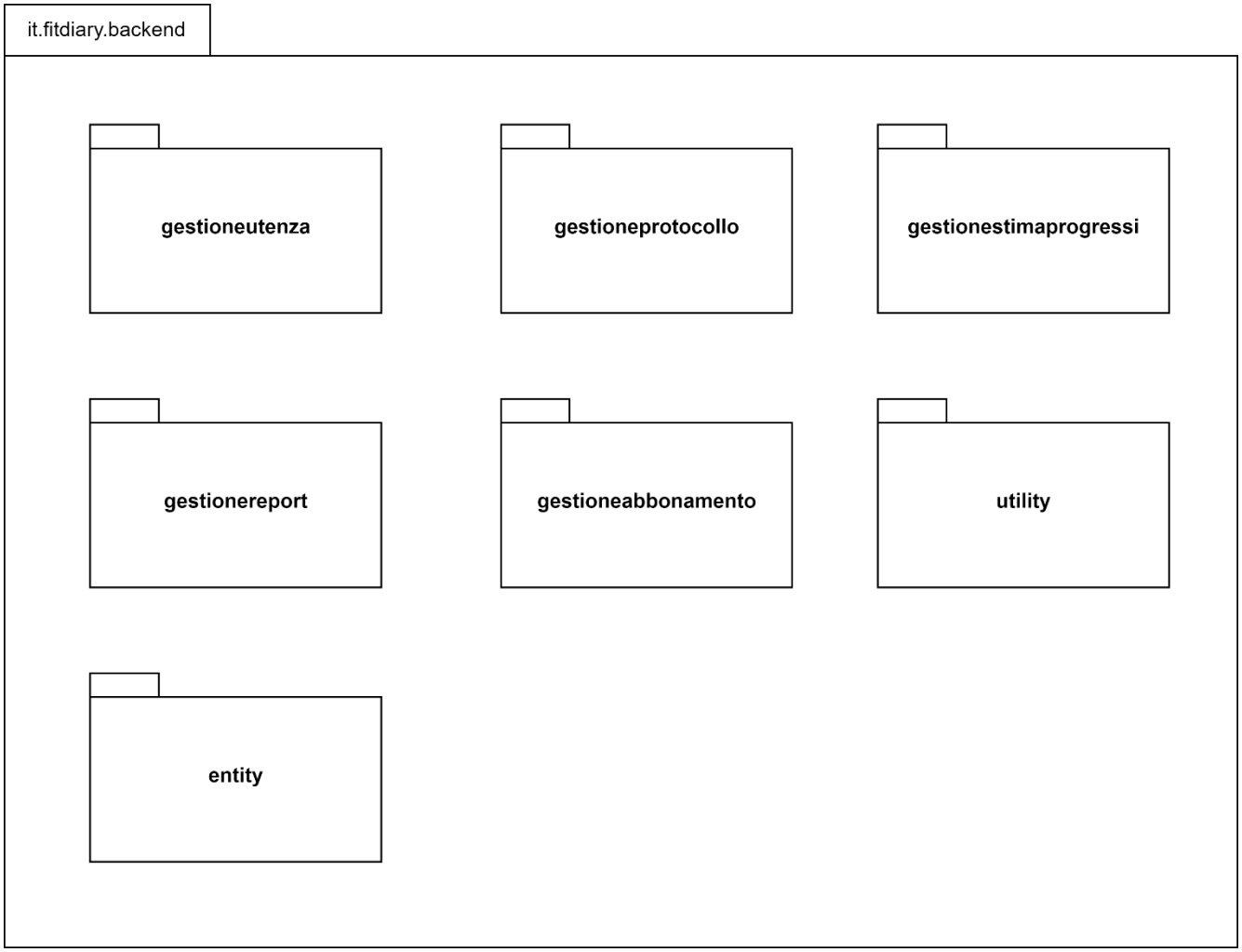
- **.idea**
- **.mvn** contiene tutti i file di configurazione per Maven.
- **src** contiene tutti i file sorgente.
    - **main**
        - **java** contiene le classi Java relative alle componenti Control, Service e Entity
        - **resources** contiene i file relativi alla componente relativa all'agente intelligente.
    - **test**, contiene tutto il necessario per il testing
        - **java** contiene le classi Java per l'implementazione del testing
- **target**, contiene tutti i file prodotti dal sistema di build di Maven.

Il lato Cliente ha struttura di tipo standard React definita da React contenente:

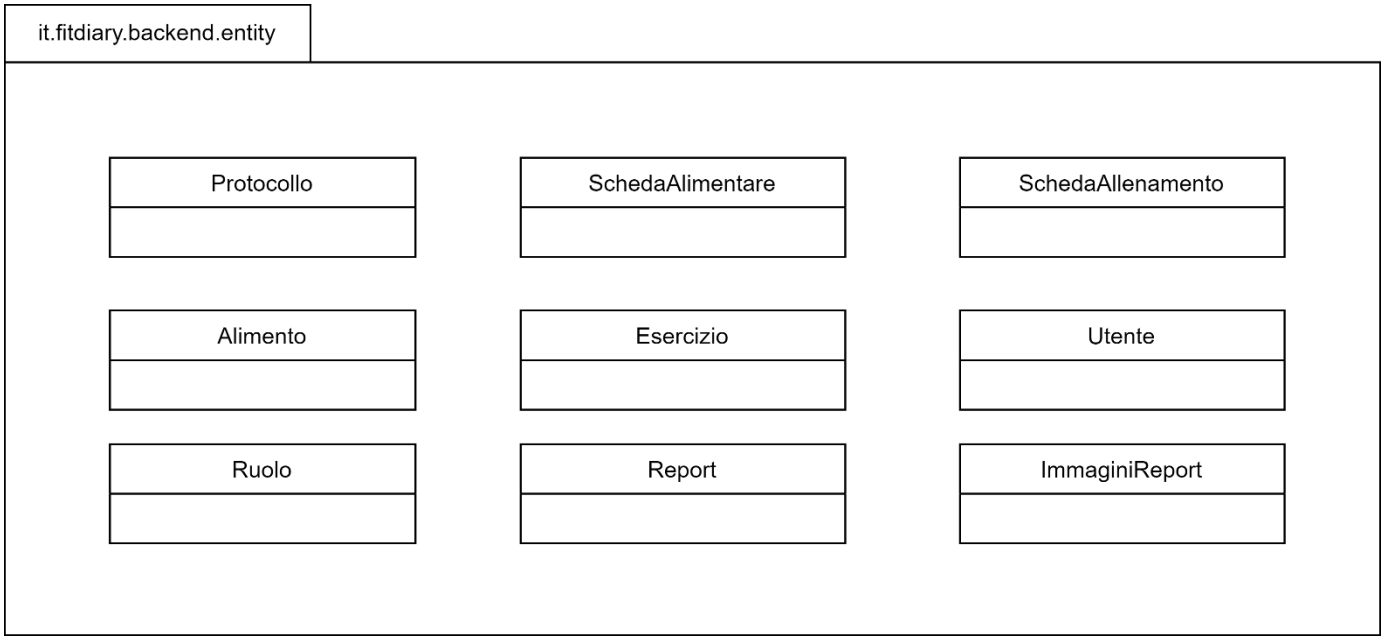
- **.idea**
- **node_modules**
- **public**
- **src**
    - **components**
    - **pages**

2.1 Package FitDiary

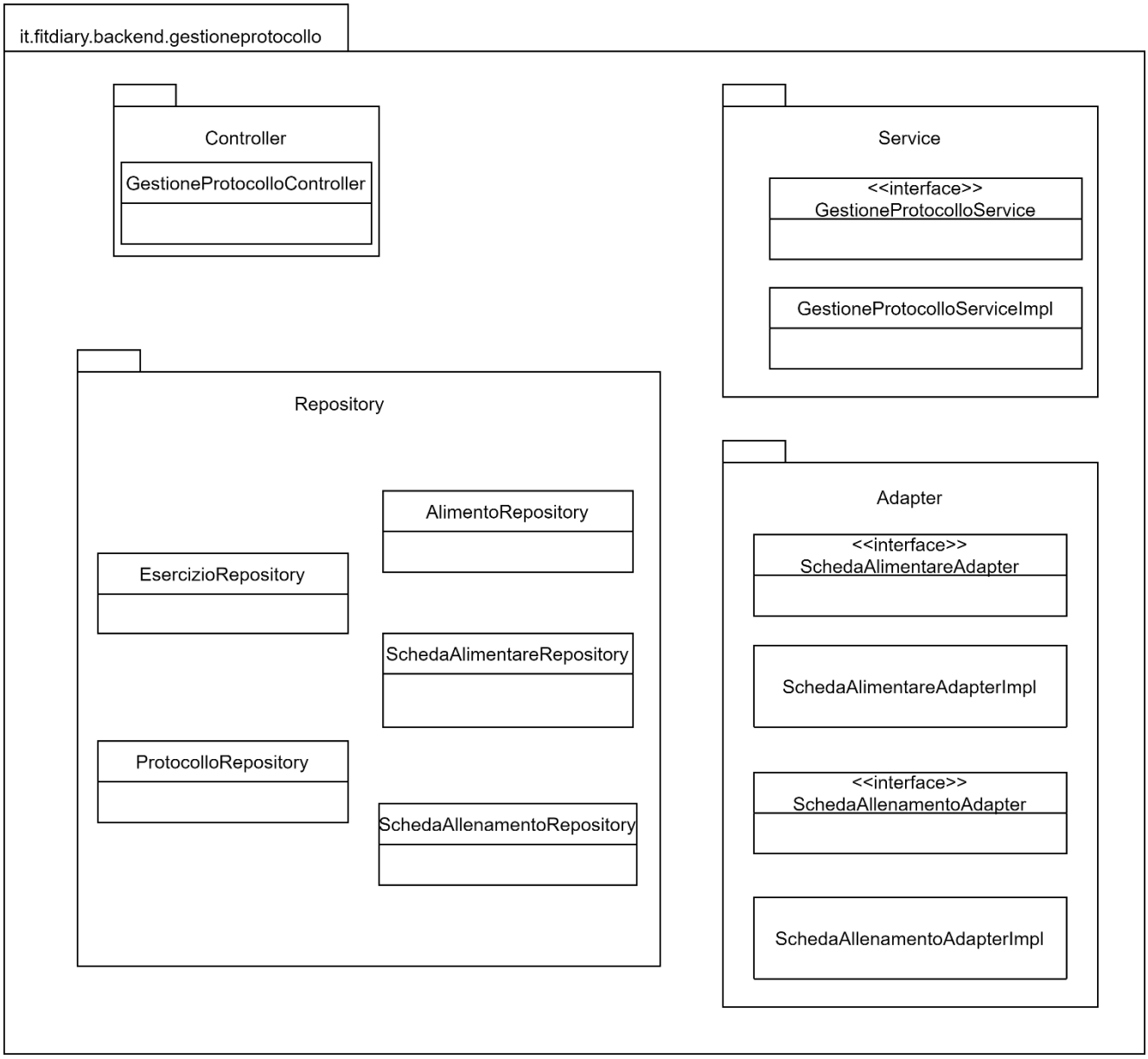
Nella presente sezione si mostra la struttura del package principale di FitDiary.
La struttura generale è stata ottenuta a partire da una principale scelta, creare un package separato per ogni sottosistema, contenente le classi Repository, Service e Controller del relativo sottosistema ed eventuali classi di utilità usate unicamente da esso. È stato inoltre aggiunto un package separato per la gestione di tutte le Entity. In aggiunta, è stato creato un package utility contenente classi utile a tutti i sottosistemi.


it.fitdiary.backend
gestioneutenza
gestioneprotocollo
gestionestimaprogressi
gestionereport
gestioneabbonamento
utility
entity

2.2 Package Entity



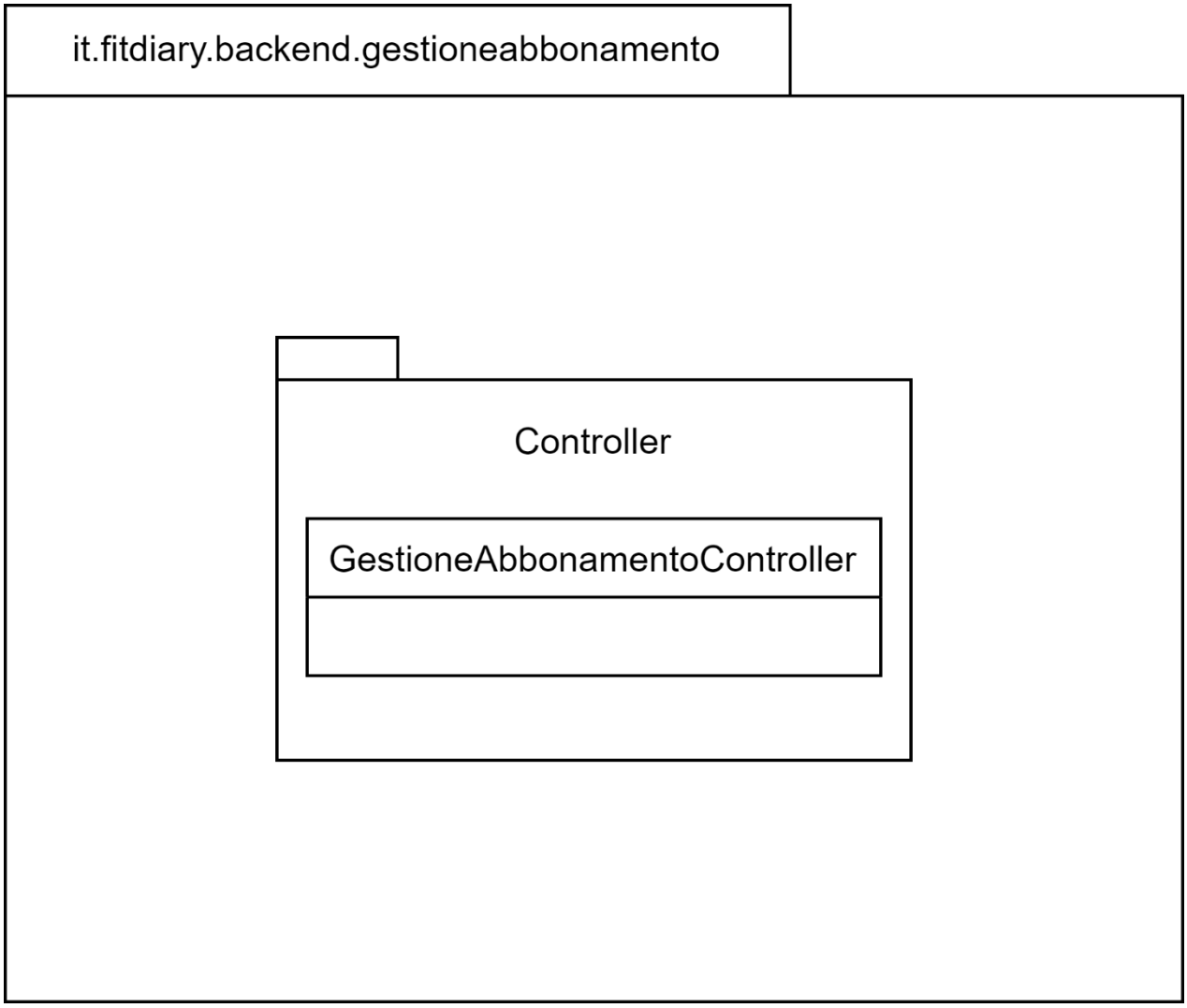
2.3 Package Gestione Protocollo



All' interno di questo package si può osservare la netta divisione interna in quattro sotto-package.

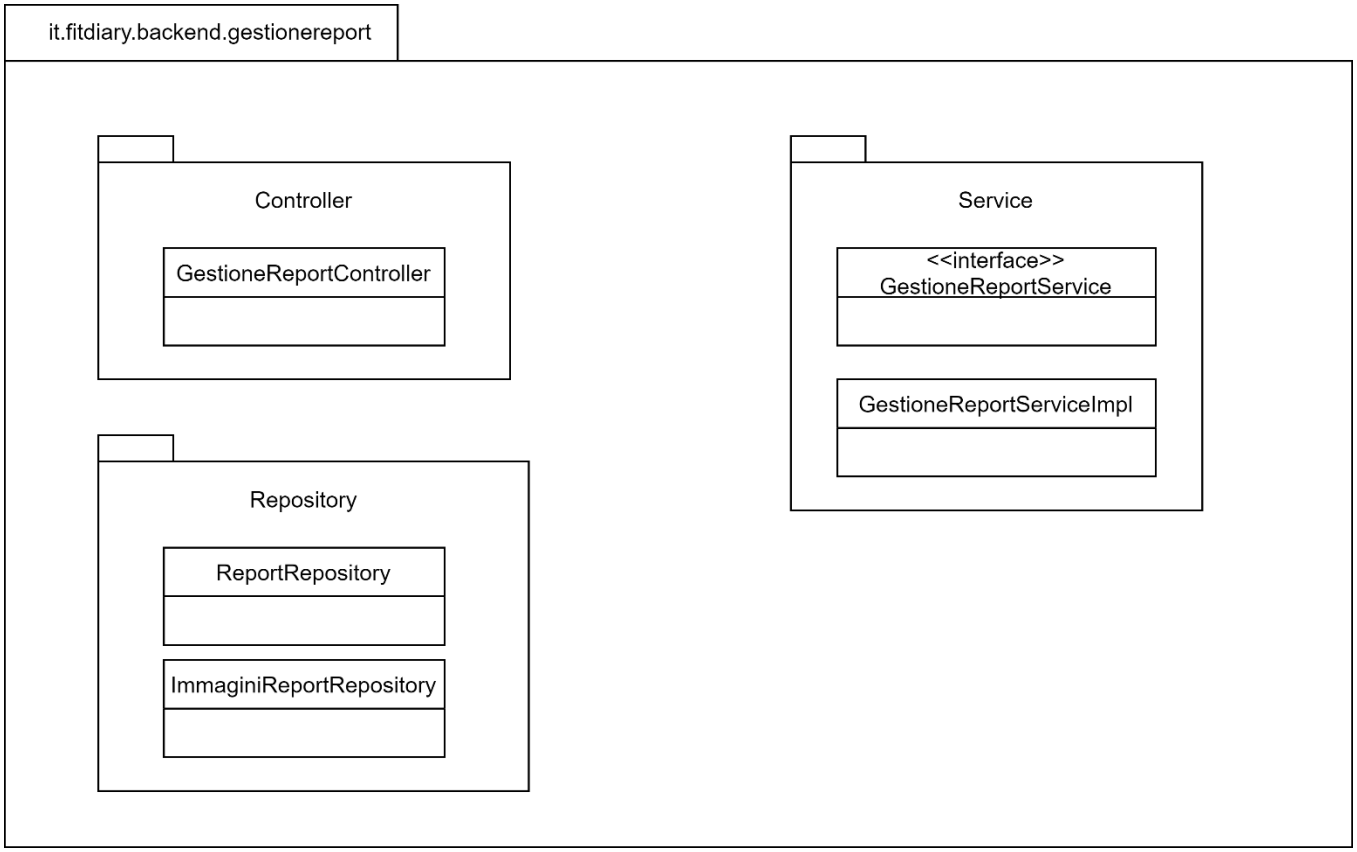
Il sotto-package "Controller" si occuperà della comunicazione fra "Service" e Client, mentre il "Repository" permette di far comunicare il "Service" con i dati persistenti. Il package "Adapter" si occupa di adattare i file delle schede trasformandoli in collezioni di oggetti manipolabili dal sistema.

2.4 Package Gestione Abbonamento



In questo package si può osservare la presenza del package "Controller" per la gestione dei flussi di un abbonamento.

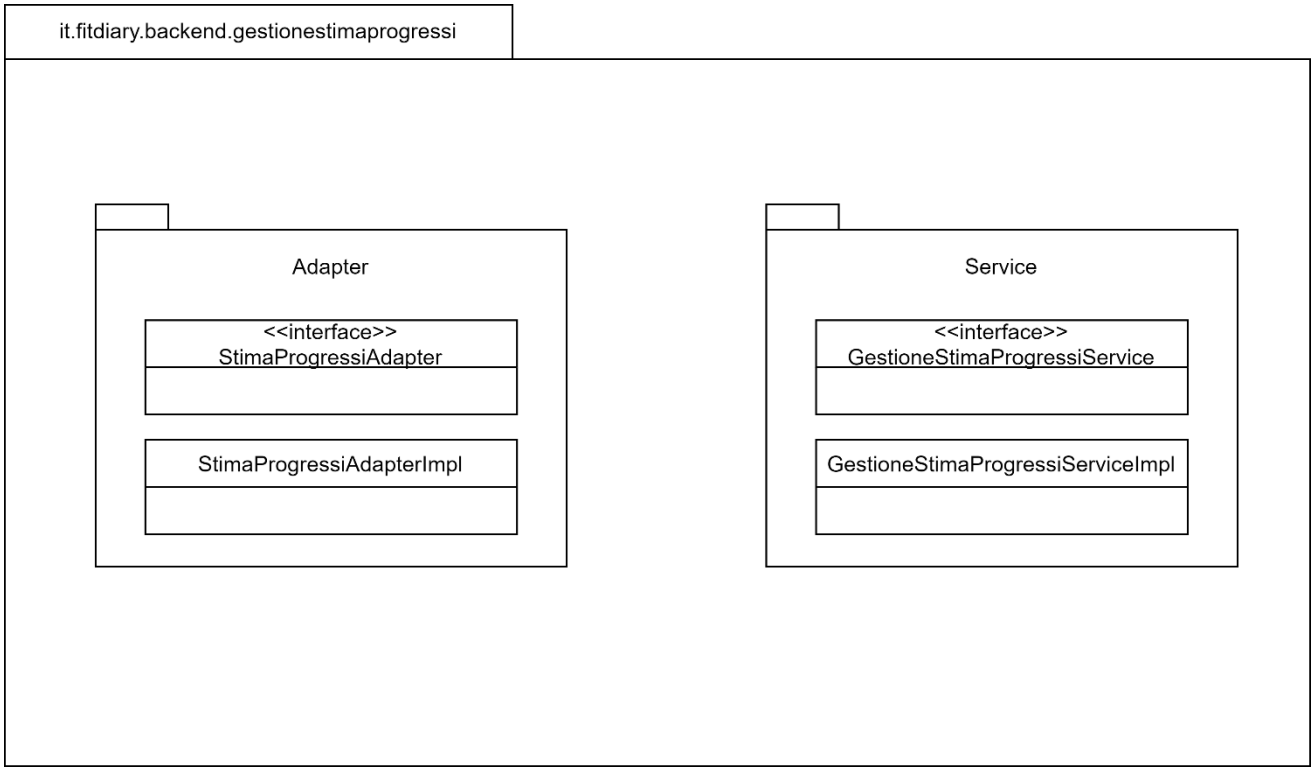
2.5 Package Gestione Report



All' interno di questo package si può osservare la divisione in tre sotto-package.

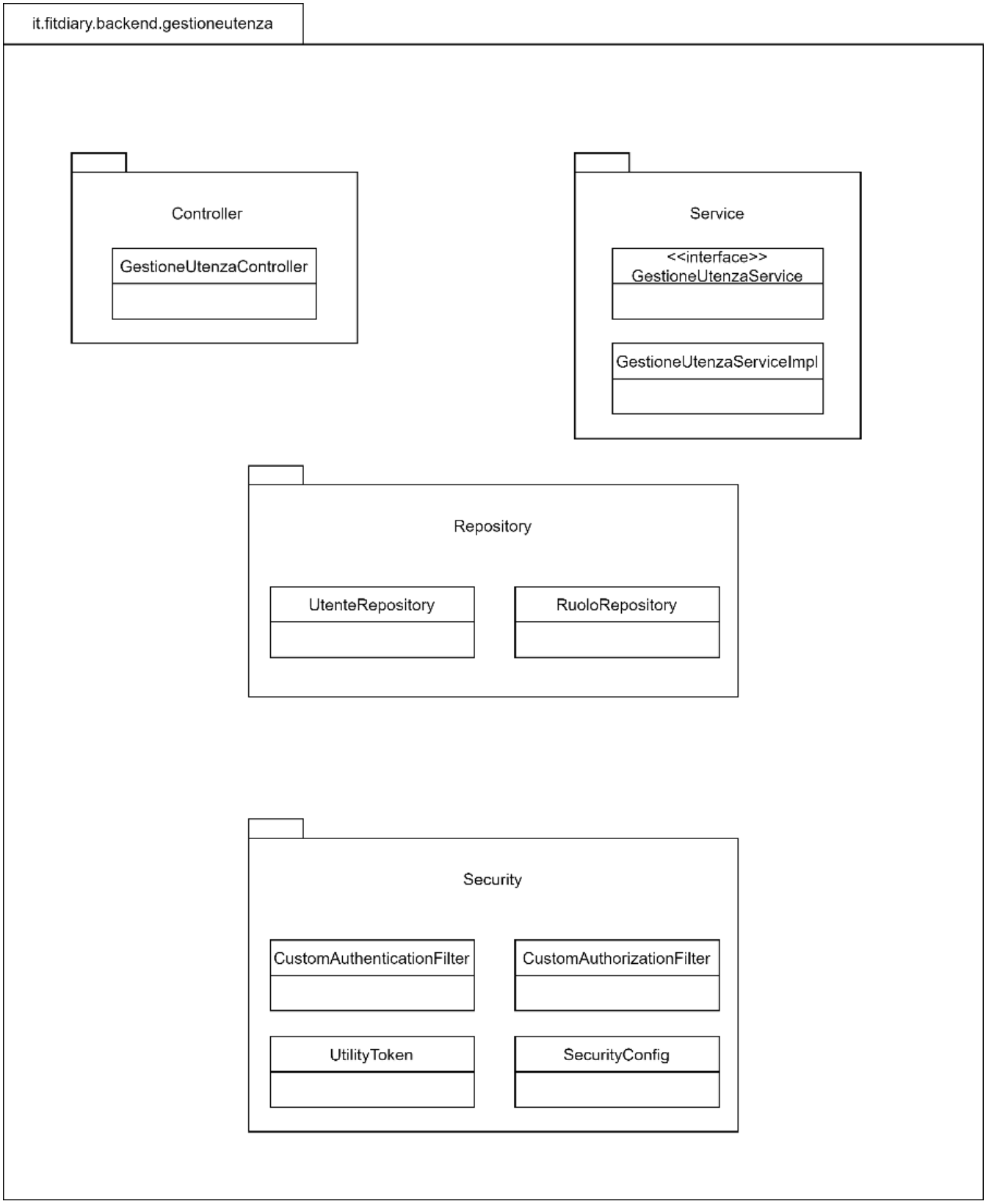
Il sotto-package "Controller" si occuperà della comunicazione fra "Service" e Client, mentre il "Repository" farà comunicare il "Service" con i dati persistenti.

2.6 Package Gestione Stima Progressi



All' interno di questo package si può osservare la singola presenza di un sotto-package "Service" che si occuperà di integrare tutti i servizi forniti dal modello di intelligenza artificiale e di un sotto-package "Adapter" che adatta l'interfaccia del modello di intelligenza artificiale.

2.7 Package Gestione Utenza

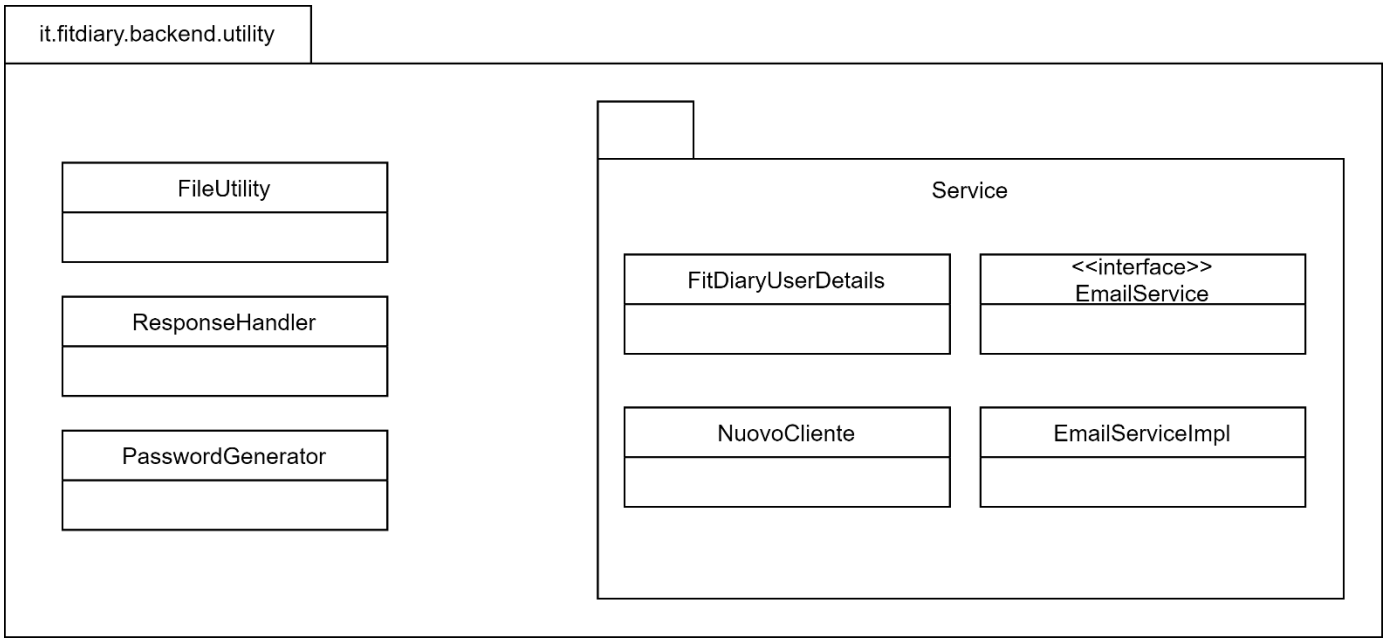

it.fitdiary.backend.gestioneutenza
Controller
GestioneUtenzaController
Service
<<interface>>
GestioneUtenzaService
GestioneUtenzaServiceImpl
Repository
UtenteRepository
RuoloRepository
Security
CustomAuthenticationFilter
CustomAuthorizationFilter
UtilityToken
SecurityConfig

All' interno di questo package si può osservare la divisione interna in quattro sotto-package.

Il sotto-package "Controller" si occuperà della comunicazione fra "Service" e Client, il "Repository" permette di far comunicare il "Service" con i dati persistenti.

Il sotto-package "Security" si occupa del controllo delle autenticazioni e autorizzazioni delle operazioni nel flusso di navigazione del sistema.

2.8 Package Utility



Il package utility comprende le classi utili a tutti i package del sistema, in particolare la classe ResponseHandler si occuperà di formattare le risposte restituite dalle classi Controller. FileUtility e PasswordGenerator forniscono funzionalità per gestire file e creare password. Le classi nel sotto-package "Service" forniscono funzionalità ai metodi dei sotto-package "Service"

3. Class Interfaces

3.1 Package Gestione Protocollo

Nome Classe	GestioneProtocolloService
Descrizione	**Permette di gestire le operazioni relative al protocollo.**
Metodi	+ creazioneProtocollo(Protocollo protocollo, File schedaAllenamentoFile, FIle schedaAlimentareFile): Protocollo +getByIdProtocollo(Long idProtocollo): Protocollo +inserisciSchedaAllenamento(Protocollo procollo,File SchedaAllenamentoFile): Protocollo +getAllProtocolliPreparatore(Utente preparatore, int page):List<Protocollo> +inserisciSchedaAlimentare(Protocollo procollo,File SchedaAllimentareFile): Protocollo + visualizzaStoricoProtocolliCliente(Long idCliente): List<Protocollo>
Invariante di classe	**n/a**

Nome Metodo	+ creazioneProtocollo(Protocollo protocollo, File schedaAllenamento, File schedaAlimentare): Protocollo
Descrizione	**Questo metodo permette di creare un protocollo per un Cliente.**
Pre-condizione	**context**: GestioneProtocolloService::creazioneProtocollo(Protocollo protocollo, File schedaAllenamentoFile, File schedaAlimentareFile) **pre**: protocollo <> null and (schedaAllenamento <> null or schedaAlimentazione <> null)
Post-condizione	**context:** GestioneProtocolloService::creazioneProtocollo(Protocollo protocollo, File schedaAllenamento, FIle schedaAlimentare) post: protocollo.getId() <> null

Nome Metodo	+ getByIdProtocollo(Long idProtocollo):Protocollo
Descrizione	**Questo metodo permette di visualizzare un determinato protocollo.in base all'id**
Pre-condizione	n/a
Post-condizione	**context:** GestioneProtocolloService:: getByIdProtocollo(Long idProtocollo) post: protocollo <> null

Nome Metodo	+ visualizzaStoricoProtocolliCliente(Long idCliente): List<Protocollo>
Descrizione	**Questo metodo consente di visualizzare lo storico dei protocolli di un cliente.**
Pre-condizione	n/a
Post-condizione	**context:**GestioneProtocolloService::visualizzaStoricoProtocolliCliente (idCliente: Long) **post: listaProtocolli <> null**

Nome Metodo	+inserisciSchedaAllenamento(Protocollo procollo,File SchedaAllenamentoFile):Protocollo
Descrizione	**Questo metodo permette di inserire una nuova scheda allenamento.**
Pre-condizione	**context**: GestioneProtocolloService:: +inserisciSchedaAllenamento(Protocollo procollo,File SchedaAllenamentoFile): **pre**: protocollo <> null and (schedaAllenamento <> null
Post-condizione	**context:** GestioneProtocolloService:: +inserisciSchedaAllenamento(Protocollo procollo,File SchedaAllenamentoFile):post: protocollo <> @pre.protocollo

Nome Metodo	+inserisciSchedaAlimentare(Protocollo procollo,File SchedaAlimentareFile):Protocollo
Descrizione	**Questo metodo permette di inserire una nuova scheda allenamento.**
Pre-condizione	**context**: GestioneProtocolloService:: +inserisciSchedaAlimentare (Protocollo procollo,File SchedaAlimentareFile):**pre**: protocollo <> null and (schedaAlimentare <> null
Post-condizione	**context:** GestioneProtocolloService:: +inserisciSchedaAlimentare (Protocollo procollo,File SchedaAlimentareFile):post: protocollo <> @pre.protocollo

Nome Metodo	+getAllProtocolliPreparatore(Utente preparatore, int page):List<Protocollo>
Descrizione	**Questo metodo permette di recuperare 50 protocolli assegnati da un preparatore**
Pre-condizione	**context**: GestioneProtocolloService:: getAllProtocolliPreparatore(Utente preparatore, int page) **pre**: Utente<> null
Post-condizione	**context:** GestioneProtocolloService:: getAllProtocolliPreparatore(Utente preparatore, int page):post: List<Protocollo> <> null

3.3 Package Gestione Report

Nome Classe	GestioneReportService
Descrizione	**Permette di gestire le operazione relative al Report.**
Metodi	+ getById(Long idReport):Report + inserimentoReport(Report report,ArrayList<String> urlImmagini): Report +search(Long idCliente, LocalDateTime data):Report
Invariante di classe	**n/a**

Nome Metodo	+ getById(Long idReport):Report
Descrizione	**Questo metodo consente di avere i dati di un report**
Pre-condizione	**context:** GestioneReportService:: getById(Long idReport) **pre:** idReport <> null
Post-condizione	**context:** GestioneReportService::visualizzazioneStoricoProgressi(Utente cliente) **post:** listReports <> null

Nome Metodo	+ search(Long idCliente,LocalDateTime data): Report
Descrizione	**Questo metodo consente di visualizzare un protocollo.**
Pre-condizione	**context**: GestioneReportService:: search(Long idCliente,LocalDateTime data) **pre**: idCliente <> null and data <> null
Post-condizione	**context:** GestioneReportService::visualizzazioneReportProtocollo(Long id) **post:** report <> null

Nome Metodo	+ inserimentoReport (Report report, ArrayList<String> urlImmagini): Report
Descrizione	**Questo metodo consente di inserire un nuovo report.**
Pre-condizione	**context**: GestioneReportService:: inserimentoReportProtocollo(Report report, ArrayList<String> urlImmagini) **pre**: report <> null and urlImmagini <> null
Post-condizione	**context:** GestioneReportService:: inserimentoReportProtocollo(Report report) **post:** report <> null

3.4 Package Gestione Stima Progressi

Nome Classe	GestioneStimaProgressiService
Descrizione	**Permette di gestire le operazione relative alla Stima Progressi.**
Metodi	+ generazioneStimaProgressi(Report report): Report
Invariante di classe	**n/a**

Nome Metodo	+ generazioneStimaProgressi(Report report): Report
Descrizione	**Questo metodo permette di generare la stima progressi associata ad un report.**
Pre-condizione	**context**: GestioneStimaProgressiService:: generazioneStimaProgressi(Report report) **pre**: report <> null
Post-condizione	**n/a**

3.5 Package Gestione Utenza

Nome Classe	GestioneUtenzaService
Descrizione	**Permette di gestire le operazione relative all'Utenza.**
Metodi	+inserisciCliente(Long idPreparatore,String nome,String cognomen,String email):Utente +getById(Long id):Utente +loadUserByUsername(String email):FitDiaryUserDetails +existsByPreparatoreAndId(Utente preparatoer,Long idCliente):Boolean + visualizzaListaUtenti(): List<Utente> + delateUtenteById(Long idUtente): void + disattivaOrAttivaUtente(long id): Utente + registrazione (Utente utente): Utente
Invariante di classe	**n/a**

Nome Metodo	+ getById(Long id): Utente
Descrizione	**Questo metodo consente di recupreare i dati di un utente**
Pre-condizione	**context**: GestioneUtenzaService:: getById(Long id): Utente **pre**: id <> null
Post-condizione	**context: GestioneUtenzaService:: getById(Long id): Utente** **post: Utente <> null**

Nome Metodo	+inserisciCliente(Long idPreparatore,String nome,String cognome,String email):Utente
Descrizione	**Questo metodo consente di inserire i dati personali di un Utente.**
Pre-condizione	**context**: GestioneUtenzaService:: inserisciCliente(Long idPreparatore,String nome,String cognomen,String email):Utente **pre**:idPreparatore <>null and nome <> null and cognome <>null and email <> null
Post-condizione	**context: GestioneUtenzaService:: inserisciCliente(Long idPreparatore,String nome,String cognomen,String email):Utente post: utente <> null**

Nome Metodo	+ visualizzaListaUtenti(): List<Utente>
Descrizione	**Questo metodo permette di visualizzare la lista degli utenti.**
Pre-condizione	n/a
Post-condizione	**context: GestioneUtenzaService:: visualizzaListaUtenti()** **pre:listaUtenti <> null**

Nome Metodo	+loadUserByUsername(String email):FitDiaryUserDetails
Descrizione	**Questo metodo consente di recupreare i dati di un utente tramite la sua email**
Pre-condizione	**context**: GestioneUtenzaService:: loadUserByUsername(String email):FitDiaryUserDetails **pre**: email<>null
Post-condizione	**context: GestioneUtenzaService:: loadUserByUsername(String email):FitDiaryUserDetails** **post: FitDiaryUserDetails<>null**

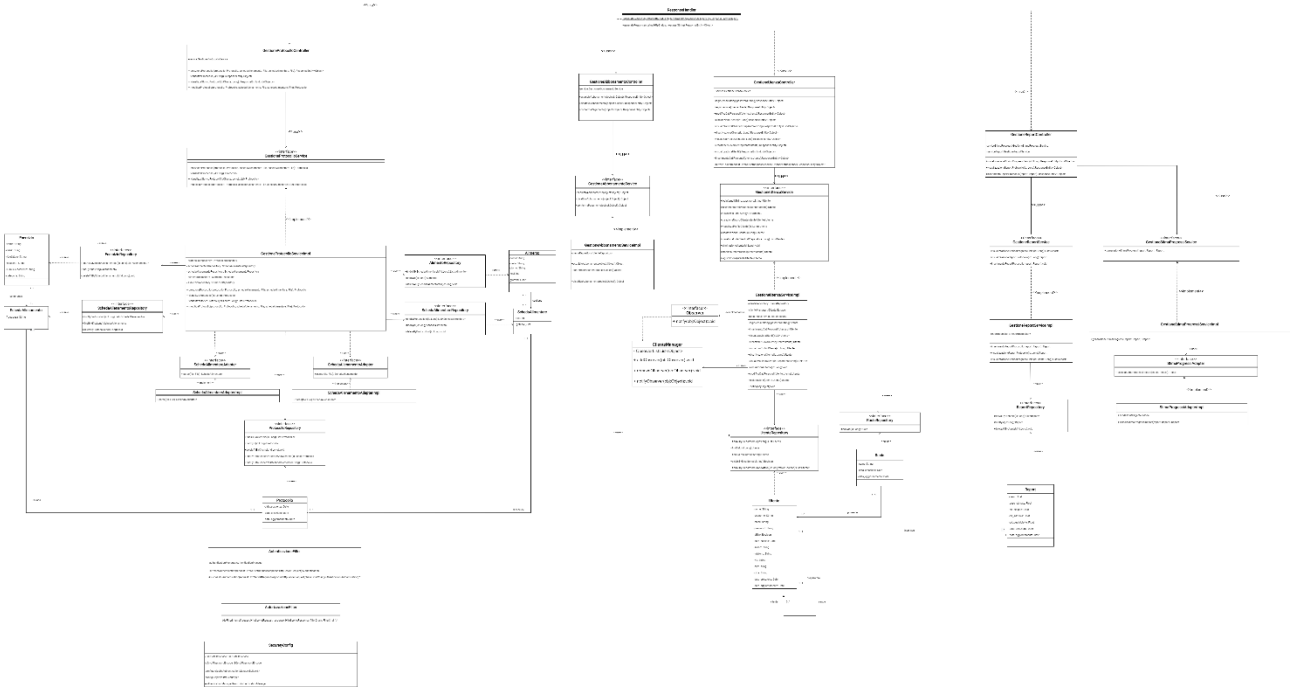
Nome Metodo	+existsByPreparatoreAndId(Utente preparatore,Long idCliente):Boolean
Descrizione	**Questo metodo permette di sapere se un cliente fa della lista clienti di un preparatore**
Pre-condizione	**context**: GestioneUtenzaService:: existsByPreparatoreAndId(Utente preparatore,Long idCliente) **post**: preparazione<>null and idCliente <> null
Post-condizione	**context: GestioneUtenzaService:: existsByPreparatoreAndId(Utente preparatore,Long idCliente)** **post: Boolena <> null**

Nome Metodo	+ disattivazioneOrAttivaUtente (Long id): Utente
Descrizione	**Questo metodo consente di attivare o disattivare un Cliente.**
Pre-condizione	n/a
Post-condizione	**context: GestioneUtenzaService:: disattivazioneOrAttivaUtente(Long id)** **post: Utente.attivo == false or Utente.attivo == true**

Nome Metodo	+ delateUtenetById(Long idUtente): void
Descrizione	**Questo metodo consente di eliminare un Utente.**
Pre-condizione	n/a
Post-condizione	**context: GestioneUtenzaService:: delateUtenetById(Long idUtente)** **post: idUtente <> null**

Nome Metodo	+ registrazione(Utente utente): Utente
Descrizione	**Questo metodo consente di registrare un nuovo preparatore.**
Pre-condizione	**context**: GestioneUtenzaService::registrazione(Utente utente) **pre**: utente<>null
Post-condizione	**context: GestioneUtenzaService::registrazione(Utente utente)** **post: utente<>null**

4. Class Diagram



Per consultare al meglio il documento -clicca qui-

5. Glossario

Sigla/Termine	Definizione
Componenti Off-the-Shelf	Componenti hardware e software disponibili sul mercato per l'acquisto da parte di aziende di sviluppo interessate a utilizzarli nei loro progetti.
Backend	Con il termine backend o back-end, nell'ambito del web-publishing, si indica l'interfaccia con la quale il gestore di un sito web dinamico ne gestisce i contenuti e le funzionalità.
Frontend	In un servizio al pubblico offerto attraverso una rete telematica o telefonica, l'insieme delle applicazioni e dei programmi informatici con cui l'utente interagisce direttamente (contrapposto a backend).
Spring Boot	Framework Java Spring per creare applicazioni basate sul framework Java Spring, maggiormente utilizzate per creare microservizi.
ReactJS	Una libreria Open-Source frontend in linguaggio Javascript per la creazione di interfacce utente mantenuto da Meta e da una comunità di singoli sviluppato e aziende.
JavaScript	JavaScript è un linguaggio di programmazione orientato agli oggetti e agli eventi, comunemente utilizzato nella programmazione Web lato client
PostgreSQL	PostgreSQL è un completo DBMS ad oggetti rilasciato con licenza libera.
Hibernate	Hibernate (talvolta abbreviato in H8) è una piattaforma middleware open source per lo sviluppo di applicazioni Java, attraverso l'appoggio al relativo framework, che fornisce un servizio di Object-relational mapping (ORM) ovvero gestisce la persistenza dei dati sul database attraverso la rappresentazione e il mantenimento su database relazionale di un sistema di oggetti Java.

JPA	Le JPA sono un framework per il linguaggio di programmazione Java, che si occupa della gestione della persistenza dei dati di un DBMS relazionale nelle applicazioni che usano le piattaforme Java Platform, Standard Edition e Java Enterprise Edition.
JDBC	JDBC è un connettore per database che consente l'accesso e la gestione della persistenza dei dati sulle basi di dati da qualsiasi programma scritto con il linguaggio di programmazione Java, indipendentemente dal tipo di DBMS utilizzato.
DBMS	Sistema software progettato per consentire la creazione, la manipolazione e l'interrogazione efficiente di database, ospitato su architettura hardware dedicata oppure su semplice computer.
Preparatore	Il professionista che si occupa della preparazione fisica e/o alimentare del cliente.
Cliente	Colui che usufruisce del servizio fornito dal preparatore.
Admin	Rappresenta la figura amministrativa del sistema, in grado di cancellare e visualizzare gli utenti che utilizzano il sistema.
Protocollo	Documento prodotto da un preparatore per un cliente contenente scheda di allenamento e/o alimentazione. Per essere creato deve contenere almeno una scheda (di allenamento o di alimentazione). Modificare un protocollo consiste nell'aggiungere o modificare una scheda.
Report	Rappresenta i progressi di un cliente, raggiunti a fine protocollo espressi in termini di dati numerici come peso, circonferenze del corpo, eventualmente foto che il cliente può allegare e contiene la stima dei progressi prevista per quel report.
Stima Progressi	Previsione del peso ad un mese di distanza dalla consegna del report.