UniRent Hub

Object Design Document

UniRentHub

Riferimento	C11_ODD_V_1.0
Versione	1.0
Data	23/01/2024
Destinatario	Prof.ssa Filomena Ferrucci, Prof. Fabio Palomba
Presentato da	C11 Fries Tech Team
Approvato da	Rocco Iuliano, Simone Della Porta

Team Members

Nome	Cognome	Ruolo	Acronimo	Contatto
Rocco	Iuliano	PM	RI	r.iuliano13@studenti.unisa.it
Simone	Della Porta	PM	SDP	s.dellaporta6@studenti.unisa.it
Antonio	Albanese	TM	AA	a.albanese22@studenti.unisa.it
Francesco Pio	Contaldo	TM	FPC	f.contaldo4@studenti.unisa.it
Cristyan	Esposito	TM	CE	c.esposito175@studenti.unisa.it
Iliano	Fasolino	TM	IF	i.fasolino3@studenti.unisa.it
Marco	Greco	TM	MG	m.greco65@studenti.unisa.it
Giuseppe Pio	Sorrentino	TM	GPS	g.sorrentino101@studenti.unisa.it

Revision History

Data	Versione	Descrizione	Autori
19/12/2023	0.1	Scrittura paragrafi 1, 1.2, 1.3, 1.4	CE, IF, FPC
19/12/2023	0.2	Scrittura sezione 2, Definizione package	CE, FPC
22/12/2023	0.3	Definizione Object Design Goals	Tutto il team
22/12/2023	0.4	Scrittura Glossario	CE, GPS
28/12/2023	0.5	Definizione Design Pattern	MG
28/12/2023	0.6	Correzione definizione package, Object Design Goals, Aggiunto Class Diagram Ristrutturato	CE, FPC, IF, AA
29/12/2023	0.7	Definizione Trade-off	GPS
29/12/2023	0.8	Scrittura Class Interface	Tutto il team
29/12/2023	0.9	Correzione sezioni 1, 2, 3, 4, 5	CE, MG, IF, FPC
30/12/2023	1.0	Check e Revisione finale	AA, MG

Sommario

1: Introduzione

UniRentHub punta ad essere una piattaforma che facilita la contrattazione tra uno studente ed un locatore per l'affitto di alloggi, ponendo anche enfasi sulla sostenibilità ambientale.

In questa sezione verranno descritti i vari object design goal, le linee guida utilizzate per la fase di implementazione, definizioni ed acronimi utilizzati nel documento e vari riferimenti utili ad altri documenti.

1.1: Object design goal

Robustezza: la piattaforma dovrà risultare robusta, gestendo al meglio gli imprevisti tramite la gestione degli errori e delle eccezioni.
Usabilità Intuitiva: la piattaforma disporrà di un'interfaccia utente semplice e facile da navigare che permetterà agli utenti di trovare rapidamente l'alloggio adatto alle sue esigenze.
Sicurezza: la piattaforma garantirà la sicurezza delle transazioni per prevenire frodi tramite tecniche di autenticazione, verifica dell'account e tecniche di protezione dei dati.
Manutenibilità: deve garantire una facile applicazione delle modifiche, e per garantire ciò si andrà a disaccoppiare quanto possibile le componenti del sistema tra loro.

1.1.1: Trade-Off

Sicurezza vs. Costi: a causa di tempi e costi limitati il sistema di sicurezza della piattaforma sarà basato solo su email e password. L'autenticazione verrà fatta tramite una pagina apposita e per ogni operazione che necessita la conoscenza dell'identità del cliente.
Persistenza vs. Efficienza: si utilizzerà la memorizzazione persistente dei dati mediante l'uso di un database relazionale, aumentando così il tempo di risposta a discapito dell'efficienza.
Privacy vs. Personalizzazione: si garantirà la protezione dei dati sensibili di un utente, ma al contempo si cercherà di utilizzare i propri dati per offrire un'esperienza personalizzata all'interno della piattaforma.

1.2: Linee guida per la documentazione dell'interfaccia

Di seguito sono elencate le linee guida che includono una lista di regole utili per gli sviluppatori da utilizzare nella realizzazione di classi ed interfacce.

- **Python**: http://www.python.it/
- **Flask:** https://www.geeksforgeeks.org/flask-tutorial/
- **HTML:** https://www.w3schools.com/html/html5_syntax.asp

1.3: Definizioni, acronimi e abbreviazioni

- **CSS**: Cascading Style Sheets
- **DAO**: Data Access Object
- **HTML**: HyperText Markup Language
- **JS:** JavaScript
- **ODD:** Object Design Document
- **SDD**: System Design Document
- **UML**: Unified Modeling Language
- **VENV**: Virtual ENViroments

1.4: Riferimenti

Libro usato come riferimento: Object-Oriented Software Engineering (Using UML, Patterns, and Java) di Bernd Bruegge & Allen H.

Di seguito una lista di riferimenti ai vari documenti utili:
- Requirements Analysis Document
- System Design Document

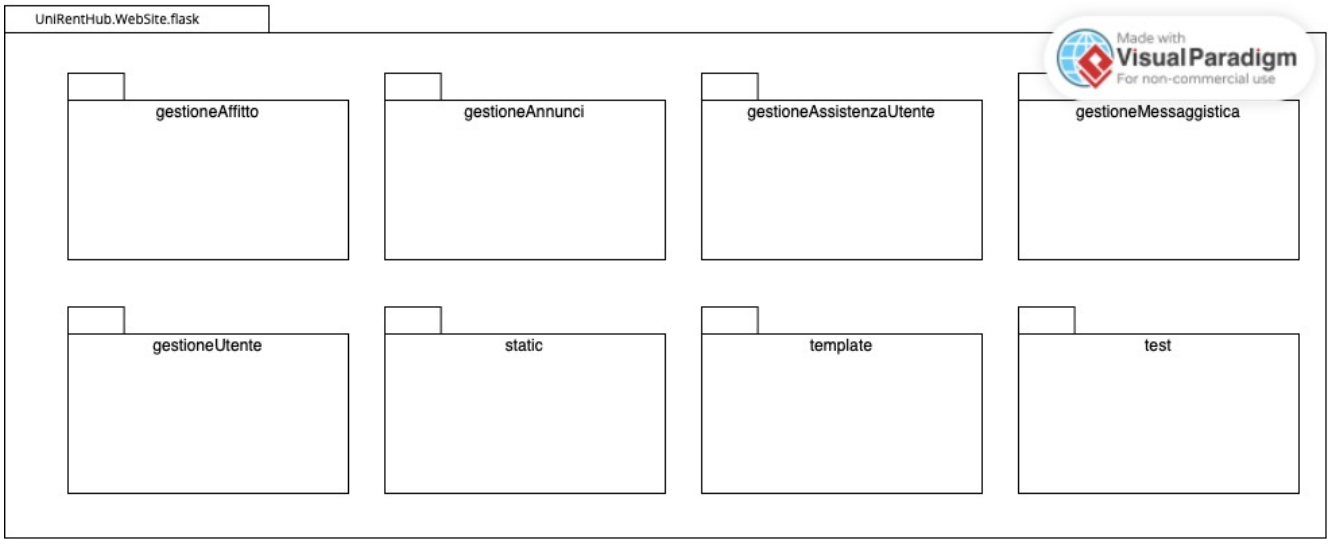
2: Packages

In questa sezione viene descritta la suddivisione dei package dell'intero sistema, basandosi su quanto definito nel SDD. Questa suddivisione rispecchia le scelte architetturali e la directory standard definita da Flask.
La suddivisione è definita di seguito:

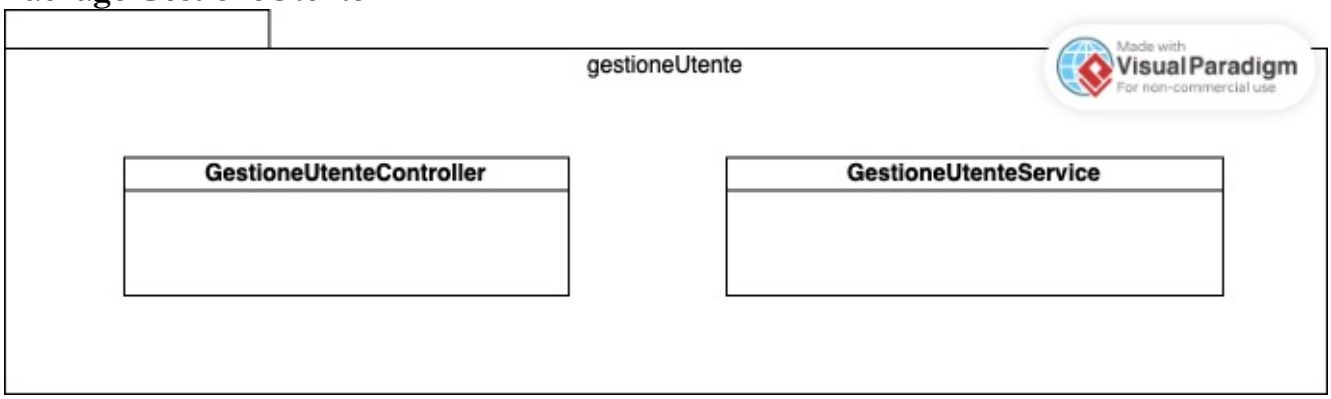
- **.idea**
- **Database**, cartella contenente i file di creazione e popolazione del database
- **flask,** cartella contenente tutti i package ed i file sorgenti
  - **static**, cartella contenente le cartelle dei file CSS, JS e le immagini
    - **css**, cartella con i file CSS
    - **js**, cartella contenente gli script JS
    - **img**, cartella contenente le immagini utilizzate per l'interfaccia grafica
  - **templates**, cartella contenente tutte le pagine html
- **test**, contenente tutti i file di testing
- **.venv**, ambiente virtuale dove sono installate Flask e tutte le librerie utilizzate

Package UniRentHub

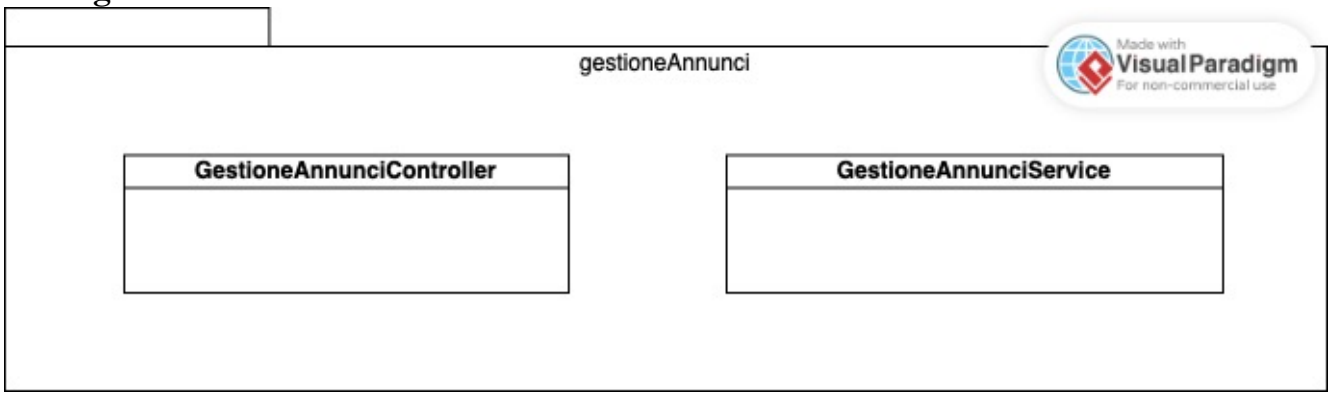
Di seguito vengono mostrati la struttura del package principale della piattaforma. Questa struttura è stata definita andando a separare i vari sottosistemi in package appositi con al proprio interno le varie classi Service e Control, con l'aggiunta delle varie classi Bean e i DAO per l'accesso ai dati del database.



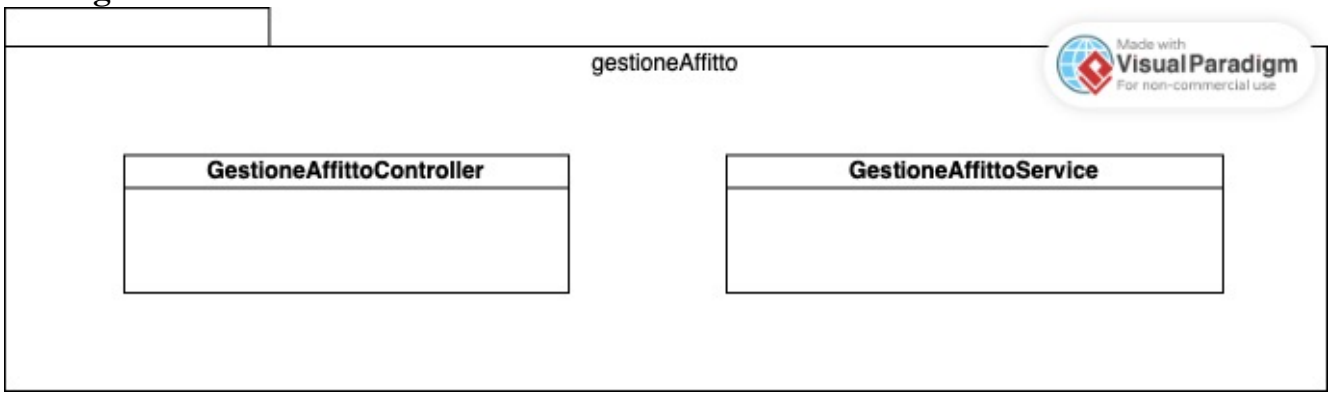
Package GestioneUtente



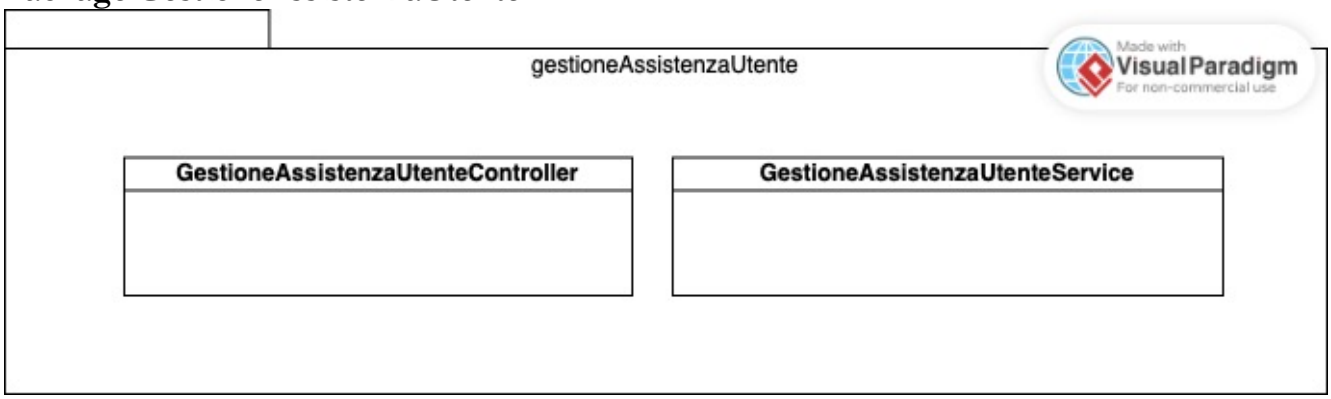
Package GestioneAnnunci



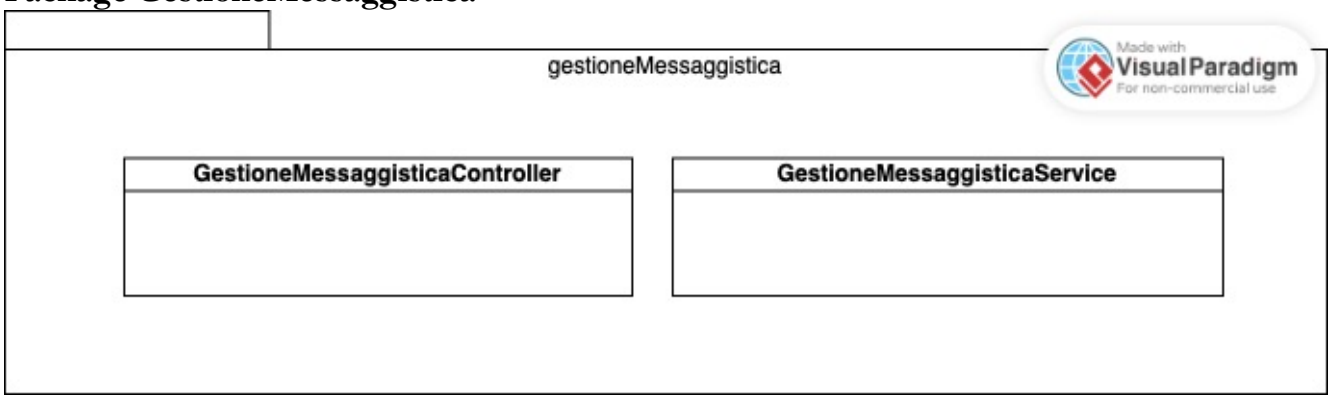
Package GestioneAffitto



Package GestioneAssistenzaUtente



Package GestioneMessaggistica



3: Class Interfaces

Class Interface: Cliente

Nome Classe	GestioneUtenteService
Descrizione	Questa classe implementa i metodi relativi alla gestione dei dati degli utenti, fornendo la registrazione, il login, logout e la modifica dei dati personali.
Metodi	+**registra_cliente**(email:str, nome:str, cognome:str, dataNascita:date, luogoNascita:str, password:str, tipoUtente:str): void +**modifica_password**(cliente: Cliente, password:str, passwordN:str): Boolean +**verifica_account**(cliente:Cliente): Boolean +**is_verificato**(cliente:Cliente): Boolean +**get_cliente_by_email**(email:str): Cliente +**controlla_email_esistente**(email:str): Boolean +**is_valid_email**(email:str): Boolean +**is_valid_password**(password:str): Boolean +**segnala_utente**(email:str, motivo:str): void +**controlla_campi**(nome:str, cognome:str, email:str): Boolean +**update_cliente**(nome:str, cognome:str, email:str, password:str, tipo_utente:str, numero_carta:str, mese_scadenza:int, anno_scadenza:int): void +**update_verificatoservice**(email:str): void
Invariante di classe	/

Nome Metodo	**registra_cliente**(email:str, nome:str, cognome:str, dataNascita:date, luogoNascita:str, password:str, tipoUtente:str): void
Descrizione	Questo metodo consente la registrazione di un nuovo cliente alla piattaforma.
Pre-condizione	**context:** GestioneUtenteService::registra_cliente(email, nome, cognome, dataNascita, luogoNascita, password, tipoUtente) **pre**: not controlla_email_esistente(email) AND email != None AND email != “” AND nome != None AND nome != “” AND cognome != None AND cognome != “” AND dataNascita != None AND is_valid_password(password) AND tipoUtente != None
Post-condizione	**context:** GestioneUtenteService::registra_cliente(email, nome, cognome, dataNascita, luogoNascita, password, tipoUtente) **post**: GestioneUtenteDAO.save(email, nome, cognome, dataNascita, luogoNascita, password, tipoUtente)

Nome Metodo	**modifica_password**(cliente: Cliente, password:str, passwordN:str): Boolean
Descrizione	Questo metodo consente la modifica della password di un cliente registrato alla piattaforma.
Pre-condizione	**context**: GestioneUtenteService::modifica_password(cliente, password, passwordN) **pre**: passwordN != password AND is_valid_password(passwordN)
Post-condizione	**context**: GestioneUtenteService::modifica_password(cliente, password) **post**: return True OR False se non ha effettuato la modifica
Nome Metodo	**verifica_account**(cliente:Cliente): Boolean
Descrizione	Questo metodo consente di verificare la mail di un cliente registrato.
Pre-condizione	**context**: GestioneUtenteService::verifica_account(cliente) **pre**: controlla_email_esistente(cliente.email)
Post-condizione	**context**: GestioneUtenteService::verifica_account(cliente) **post**: return True
Nome Metodo	**is_verificato**(cliente:Cliente): Boolean
Descrizione	Questo metodo consente di controllare se un utente ha verificato la sua mail.
Pre-condizione	**context:** GestioneUtenteService::is_verificato(cliente:Cliente) **pre**: cliente is not None
Post-condizione	**context:** GestioneUtenteService::is_verificato(cliente:Cliente) **post**: return true OR false se non è verificato
Nome Metodo	**get_cliente_by_email**(email:str): Cliente
Descrizione	Questo metodo restituisce un cliente registrato in base all'email fornita in input.
Pre-condizione	**context**: GestioneUtenteService::get_cliente_by_email(email) **pre**: email != None AND email != “”

Post-condizione	**context**: GestioneUtenteService::get_cliente_by_email(email) **post**: return Cliente OR None se non esiste
Nome Metodo	**controlla_email_esistente**(email:str): Boolean
Descrizione	Questo metodo consente di verificare se la mail utilizzata è già salvata nel sistema.
Pre-condizione	**context**: GestioneUtenteService::controlla_email_esistente(email) **pre**: email != None AND email != “”
Post-condizione	**context**: GestioneUtenteService::controlla_email_esistente(email) **post**: return True OR False se non esiste
Nome Metodo	**is_valid_email**(email:str): Boolean
Descrizione	Questo metodo verifica se la email inserita nel form di registrazione è valida e non contiene caratteri non consentiti.
Pre-condizione	**context**: GestioneUtenteService::is_valid_email(email) **pre**: email != None AND email != “”
Post-condizione	**context**: GestioneUtenteService::is_valid_email(email) **post**: return True se la mail è valida OR False altrimenti
Nome Metodo	**is_valid_password**(password:str): Boolean
Descrizione	Questo metodo verifica se sono stati utilizzati caratteri speciali e numeri nella password inserita nel form di registrazione.
Pre-condizione	**context**: GestioneUtenteService::is_valid_password(password) **pre**: password != None AND password!= “”
Post-condizione	**context**: GestioneUtenteService::is_valid_password(password) **post**: return True se contiene almeno un numero ed un carattere speciale OR False altrimenti
Nome Metodo	**segnala_utente**(email:str, motivo:str): void
Descrizione	Questo metodo consente ad un utente di poter effettuare una segnalazione relativa ad un altro utente
Pre-condizione	**context:** GestioneUtenteService::segnala_utente(email, motivo) **pre**: controlla_email_esistente(email) AND motivo != None AND motivo != “”

Post-condizione	**context:** GestioneUtenteService::segnala_utente(email, motivo) **post**: GestioneUtenteDAO.save(email, motivo)
Nome Metodo	**controlla_campi**(nome:str, cognome:str, email:str): Boolean
Descrizione	Questo metodo controlla che i parametri nome, cognome ed email siano validi.
Pre-condizione	**context:** GestioneUtenteService::controlla_campi(nome, cognome, email) **pre**: nome != None AND nome != “” AND cognome != None AND cognome != “” AND is_valid_email(email)
Post-condizione	**context:** GestioneUtenteService::controlla_campi(nome, cognome, email) **post**: return True OR False se uno dei parametri non è valido
Nome Metodo	**update_cliente**(nome:str, cognome:str, email:str, password:str, tipo_utente:str, numero_carta:str, mese_scadenza:int, anno_scadenza:int): void
Descrizione	Questo metodo consente di poter aggiornare dei campi di un cliente.
Pre-condizione	**context:** GestioneUtenteService::update_cliente(nome, cognome, email, password, tipo_utente, numero_carta, mese_scadenza, anno_scadenza) **pre**: nome != None AND nome != “” AND cognome != None AND cognome != “” AND is_valid_email(email) AND is_valid_password(password) AND numero_carta != None AND numero_carta != “” AND mese_scadenza != None AND mese_scadenza != “” AND mese_scadenza > 0 AND anno_scadenza != None AND anno_scadenza != “” AND anno_scadenza >= 2024
Post-condizione	**context:** GestioneUtenteService::update_cliente(nome, cognome, email, password, tipo_utente, numero_carta, mese_scadenza, anno_scadenza) **post**: aggiorna i campi di cliente
Nome Metodo	**update_verificatoservice**(email:str): void
Descrizione	Questo metodo aggiorna lo stato di verifica dell’email di un cliente.

Pre-condizione	**context:** GestioneUtenteService::update_verificatoservice(email) **pre**: is_valid_email(email)
Post-condizione	**context:** GestioneUtenteService::update_verificatoservice(email) **post**: aggiorna la verifica del cliente impostandola a True

Class Interface: Dipendente

Nome Classe	GestioneUtenteService
Descrizione	Questa classe implementa i metodi relativi alla gestione dei dati degli utenti, fornendo il login, logout.
Metodi	+**get_homechecker_by_email**(email:str): Homechecker +**check_email_esistente**(email:str): Boolean +**check_ password**(password:str): Boolean +**registra_homechecker**(email:str, nome:str, cognome:str, password:str, tipoDipendente:str): void +**delete_homechecker**(Homechecker:homechecker): Boolean
Invariante di classe	/

Nome Metodo	**get_homechecker_by_email**(email:str): Homechecker
Descrizione	Questo metodo restituisce un homechecker registrato in base all'email fornita in input.
Pre-condizione	**context**: GestioneUtenteService::get_homechecker_by_email(email) **pre**: email != None AND email != ""
Post-condizione	**context**: GestioneUtenteService::get_homechecker_by_email(email) **post**: return Homechecker OR None se non esiste
Nome Metodo	**check_email_esistente**(email:str): Boolean
Descrizione	Questo metodo consente di verificare se la mail utilizzata è già salvata nel sistema.
Pre-condizione	**context**: GestioneUtenteService::check_email_esistente(email) **pre**: email != None AND email != ""

Post-condizione	**context**: GestioneUtenteService::check_email_esistente(email) **post**: return True OR False se non esiste
Nome Metodo	**check_ password**(password:str): Boolean
Descrizione	Questo metodo verifica se sono stati utilizzati caratteri speciali e numeri nella password inserita nel form di registrazione.
Pre-condizione	**context**: GestioneUtenteService::check_password(password) **pre**: password != None AND password!= “”
Post-condizione	**context**: GestioneUtenteService::check_password(password) **post**: return True se contiene almeno un numero ed un carattere speciale OR False altrimenti
Nome Metodo	**registra_homechecker**(email:str, nome:str, cognome:str, password:str, tipoDipendente:str): void
Descrizione	Questo metodo consente la registrazione di un nuovo homechecker alla piattaforma.
Pre-condizione	**context**: GestioneUtenteService::registra_homechecker(email:str, nome:str, cognome:str, password:str, tipoDipendente:str) **pre**: not check_email_esistente(email) AND email != None AND email != “” AND nome != None AND nome != “” AND cognome != None AND cognome != “” AND check_password(password) AND tipoDipendente != None
Post-condizione	**context**: GestioneUtenteService::registra_homechecker(email:str, nome:str, cognome:str, password:str, tipoDipendente:str) **post**: GestioneUtenteDAO.save(email, nome, cognome, password, tipoDipendente)
Nome Metodo	**delete_homechecker**(Homechecker:homechecker): Boolean
Descrizione	Questo metodo permette di rimuovere un homechecker dalla piattaforma.
Pre-condizione	**context**: GestioneUtenteService::delete_homechecker(Homechecker:homechecker) **pre**: /

Post-condizione	**context**: GestioneUtenteService::delete_homechecker(Homechecker:homechecker) **post**: GestioneUtenteDAO.delete(email, nome, cognome, password, tipoDipendente)

Class Interface: Alloggio

Nome Classe	GestioneAnnunciService
Descrizione	Questa classe implementa il sistema di gestione annunci fornendo i metodi per gestire un alloggio pubblicato sul sito
Metodi	+**pubblicazione_alloggio**(Titolo: str, Indirizzo: str, Stanze: int, TipoAlloggio: str, Descrizione: str, Numerobagni: int, NumeroCamereletto: int, PannelliSolari: boolean, PannelliFotovoltaici: boolean, Arredamento: boolean, ClasseEnergetica:str, NumeroOspiti:int, metriquadri:int, prezzo:float, PeriodoMinimo:int, Servizi:Servizio, Tasse:boolean, Immagine:immagine): Boolean +**verifica_campi**(titolo:str, indirizzo:str, descrizione:str, num_bagni:int, num_camere:int, num_ospiti:int, metri_quadri:int, prezzo:double, periodo_minimo:int): Boolean +**inserisci_immagini_service**(immagini:List): str +**ricerca_alloggio**(città:str): List +**visualizza_servizi_alloggio**(id_alloggio:int): List +**modifica_annuncio_byid**(id_allogio:int, tipo_alloggio:str, titolo:str, mq:int, n_camere_letto:int, n_bagni:int, classe_energetica:str, arredamenti:Boolean, data_pubblicazione:date, pannelli_solari:Boolean, pannelli_fotovoltaici:Boolean, descrizione:str, prezzo:double, n_ospiti:int, n_stanze:int, tasse:int): Alloggio +**recensione**(id:int, titolo:str, descrizione:str, voto:int, data:data, email:str): void +**segnala_service**(email:str, emailS:str, motivo:str): void +**creazione_post**(titolo:str, descrizione:str, email:str): void
Invariante di classe	/

Nome Metodo	**pubblicazione_alloggio**(Titolo: str, Indirizzo: str, Stanze: int, TipoAlloggio: str, Descrizione: str, Numerobagni: int, NumeroCamereletto: int, PannelliSolari: boolean, PannelliFotovoltaici: boolean, Arredamento: boolean, ClasseEnergetica:str, NumeroOspiti:int, metriquadri:int, prezzo:float, PeriodoMinimo:int, Servizi:Servizio, Tasse:boolean, Immagine:immagine): Boolean

Descrizione	Questo metodo consente ad un locatore di poter caricare un annuncio
Pre-condizione	**context**: GestioneAlloggioService::CreaAlloggio(Titolo: str, Indirizzo: str, Stanze: int, TipoAlloggio: str, Descrizione: str, Numerobagni: int, NumeroCamereletto: int, PannelliSolari: boolean, PannelliFotovoltaici: boolean, Arredamento: boolean, ClasseEnergetica:str, NumeroOspiti:int, metriquadri:int, prezzo:float, PeriodoMinimo:int, Servizi:Servizio, Tasse:boolean, Immagine:immagine) **pre**: len(titolo) <30 AND len(titolo)>=5, len(indirizzo) <30 AND len(indirizzo)>=5 AND Stanze>0 AND len(descrizione)>30 AND Numerobagni>0 AND NumeroCamereletto>0 AND NumeroOspiti>0 AND MetriQuadri>0 AND prezzo>0 AND PeriodoMinimo>0 AND immaggine.size >= Stanze + Numerobagni + NumeroCamereletto
Post-condizione	**context**: GestioneAlloggioService::CreaAlloggio(Titolo: str, Indirizzo: str, Stanze: int, TipoAlloggio: str, Descrizione: str, Numerobagni: int, NumeroCamereletto: int, PannelliSolari: boolean, PannelliFotovoltaici: boolean, Arredamento: boolean, ClasseEnergetica:str, NumeroOspiti:int, metriquadri:int, prezzo:float, PeriodoMinimo:int, Servizi:Servizio, Tasse:boolean, Immagine:immagine) **post**: GestioneAlloggioDAO.save(Titolo, Indirizzo, Stanze, TipoAlloggio, Descrizione, Numerobagni, NumeroCamereletto, PannelliSolari, PannelliFotovoltaici, Arredamento, ClasseEnergetica, NumeroOspiti, metriquadri, prezzo, PeriodoMinimo, Servizi, Tasse, Immagine)
Nome Metodo	**verifica_campi**(titolo:str, indirizzo:str, descrizione:str, num_bagni:int, num_camere:int, num_ospiti:int, metri_quadri:int, prezzo:double, periodo_minimo:int): Boolean
Descrizione	Questo metodo verifica se tutti i dati passati come parametri siano corretti.
Pre-condizione	**context**: GestioneAnnunciService::verifica_campi(titolo, indirizzo, descrizione, num_bagni, num_camere, num_ospiti, metri_quadri, prezzo, periodo_minimo) **pre**: titolo != None AND titolo != “” AND indirizzo != None AND indirizzo != “” AND descrizione != None AND descrizione != “” AND num_bagni != None AND num_bagni > 0 AND num_camere != None AND num_camere > 0 AND num_ospiti != None AND num_ospiti > 0 AND metri_quadri != None AND metri_quadri > 0 AND prezzo != None AND prezzo > 0 AND periodo_minimo != None AND periodo_minimo > 0

Post-condizione	**context**: GestioneAnnunciService::verifica_campi(titolo, indirizzo, descrizione, num_bagni, num_camere, num_ospiti, metri_quadri, prezzo, periodo_minimo) **post**: return True OR False se qualche campo non è corretto
Nome Metodo	**inserisci_immagini_service**(immagini:List)
Descrizione	Questo metodo permette ad un utente di poter inserire delle immagini al proprio annuncio.
Pre-condizione	**context**: GestioneAnnunciService::inserisci_immagini_service(immagini) **pre**: immagini != None
Post-condizione	**context**: GestioneAnnunciService::inserisci_immagini_service(immagini) **post**: salva le immagini nel database
Nome Metodo	**ricerca_alloggio**(città:str): List
Descrizione	Questo metodo permette di ricercare un alloggio per una determinata città.
Pre-condizione	**context**: GestioneAnnunciService::ricerca_alloggio(città) **pre**: città != None AND città != “”
Post-condizione	**context**: GestioneAnnunciService::ricerca_alloggio(città) **post**: ritorna una lista con gli annunci di alloggi situati in quella determinata città
Nome Metodo	**visualizza_servizi_alloggio**(id_alloggio:int): List
Descrizione	Questo metodo permette di visualizzare tutti i servizi forniti da un alloggio pubblicato.
Pre-condizione	**context**: GestioneAnnunciService::visualizza_servizi_alloggio(id_alloggio) **pre**: id_alloggio != None AND id_alloggio != “”
Post-condizione	**context**: GestioneAnnunciService::visualizza_servizi_alloggio(id_alloggio) **post**: ritorna una lista con i servizi dell’alloggio con id_alloggio

Nome Metodo	**modifica_annuncio_byid**(id_allogio:int, tipo_alloggio:str, titolo:str, mq:int, n_camere_letto:int, n_bagni:int, classe_energetica:str, arredamenti:Boolean, data_pubblicazione:date, pannelli_solari:Boolean, pannelli_fotovoltaici:Boolean, descrizione:str, prezzo:double, n_ospiti:int, n_stanze:int, tasse:int): Alloggio
Descrizione	Questo metodo consente di modificare alcuni parametri di un annuncio.
Pre-condizione	**context**: GestioneAnnunciService::modifica_annuncio_byid(id_allogio, tipo_alloggio, titolo, mq, n_camere_letto, n_bagni, classe_energetica, arredamenti, data_pubblicazione, pannelli_solari, pannelli_fotovoltaici, descrizione, prezzo, n_ospiti, n_stanze, tasse) **pre**: tipo_alloggio != None AND tipo_alloggio != “” AND titolo != None AND titolo != “” AND mq != None AND mq > 0 AND n_camere_letto != None AND n_camere_letto > 0 AND n_bagni != None AND n_bagni > 0 AND classe_energetica != None AND classe_energetica != “” AND arredamenti != None AND arredamenti != “” AND data_pubblicazione != None AND data_pubblicazione != “” AND pannelli_solari != None AND pannelli_solari != “” AND pannelli_fotovoltaici != None AND pannelli_fotovoltaici != “” AND descrizione != None AND descrizione != “” AND prezzo != None AND prezzo > 0 AND n_ospiti != None AND n_ospiti > 0 AND n_stanze != None AND n_stanze > 0 AND tasse != None AND tasse > 0
Post-condizione	**context**: GestioneAnnunciService::modifica_annuncio_byid(id_allogio, tipo_alloggio, titolo, mq, n_camere_letto, n_bagni, classe_energetica, arredamenti, data_pubblicazione, pannelli_solari, pannelli_fotovoltaici, descrizione, prezzo, n_ospiti, n_stanze, tasse) **post**: return Alloggio con le modifiche effettuate
Nome Metodo	**recensione**(id:int, titolo:str, descrizione:str, voto:int, data:data, email:str): void
Descrizione	Questo metodo consente di effettuare la recensione ad un annuncio.
Pre-condizione	**context**: GestioneAnnunciService::recensione(id, titolo, descrizione, voto, data, email) **pre**: titolo != None AND titolo != “” AND descrizione != None AND descrizione != “” AND voto != None AND voto > 0

Post-condizione	**context**: GestioneAnnunciService::recensione(id, titolo, descrizione, voto, data, email) **post**: GestioneAnnunciService::doSave(id, titolo, descrizione, voto, data, email)
Nome Metodo	**segnala_service**(email, emailS, motivo): void
Descrizione	Questo metodo consente di effettuare la segnalazione al locatore di annuncio.
Pre-condizione	**context**: GestioneAnnunciService::segnala_service(email, emailS, motivo) **pre**: email != None AND email != "" AND emailS != None AND emailS != "" AND motivo != None AND motivo != ""
Post-condizione	**context**: GestioneAnnunciService::segnala_service(email, emailS, motivo) **post**: GestioneAnnunciService::doSave(email, emailS, motivo)
Nome Metodo	**creazione_post**(titolo:str, descrizione:str, email:str): void
Descrizione	Questo metodo consente di effettuare la creazione di post da parte di un utente studente.
Pre-condizione	**context**: GestioneAnnunciService::creazione_post(titolo, descrizione, email) **pre**: titolo != None AND titolo != "" AND descrizione != None AND descrizione != ""
Post-condizione	**context**: GestioneAnnunciService::creazione_post(titolo, descrizione, email) **post**: GestioneAnnunciService::doSave(titolo, descrizione, email)

Class Interface: Affitto

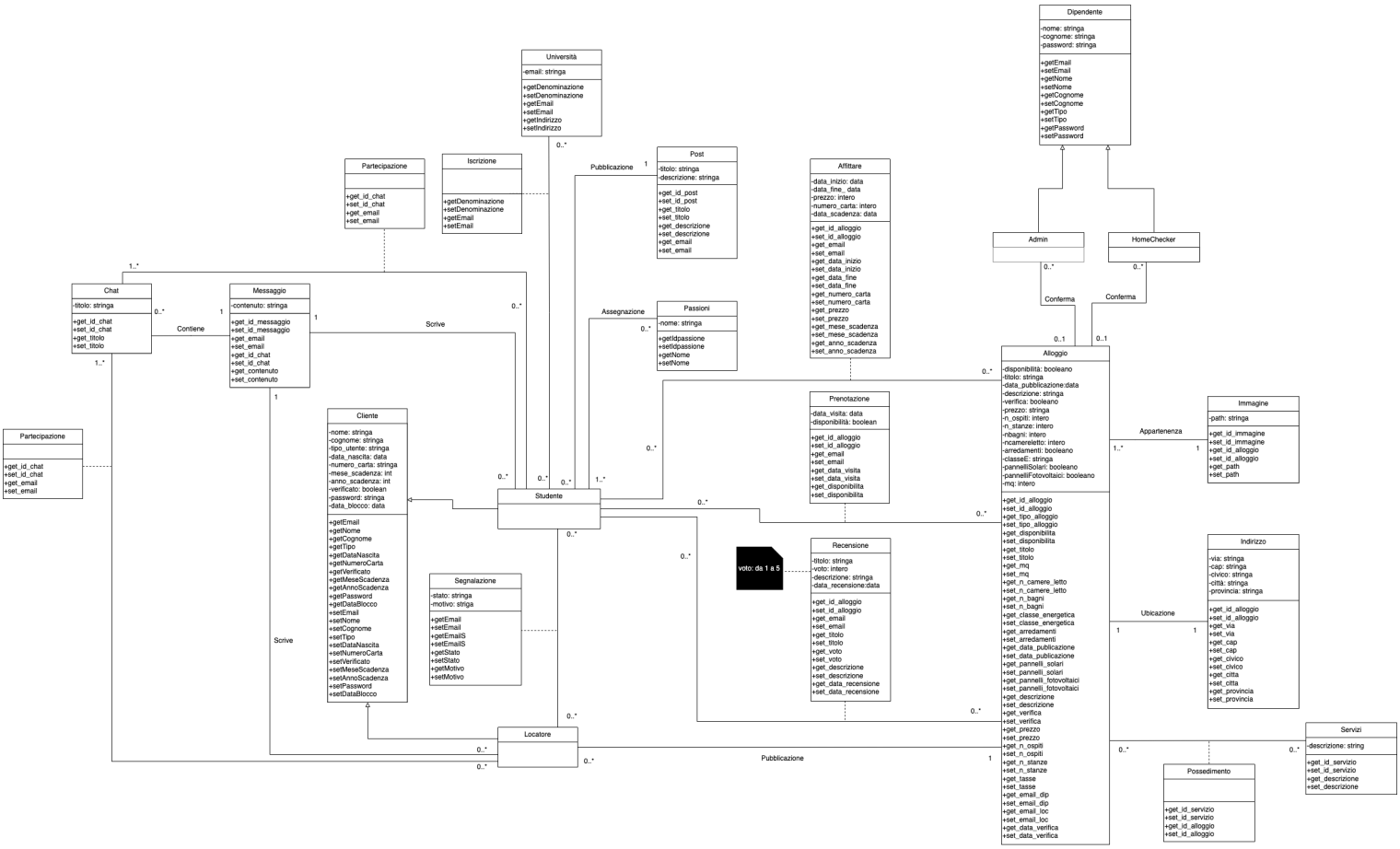
Nome Classe	GestioneAffittoService
Descrizione	Questa classe implementa il sistema di gestione affitto fornendo i metodi per effettuare l'affitto di un appartamento

Metodi	+**prenota_visita_alloggio**(email:str, data_visita:date, id_alloggio: int): Boolean +**affitto_alloggio_cliente**(email: str, data_inizio:date, data_fine:date, prezzo:float, numero_carta:int, data_scadenza:date, id_alloggio:int): Boolean +**insert_data_byId**(prenotazione: Prenotazione): void +**delete_data_by_Id**(data_visita:data): void +**ottieni_date_per_alloggio_byId**(id_alloggio:int): List
Invariante di classe	/

Nome Metodo	**prenota_visita_alloggio**(email:str, data_visita:date, id_alloggio: int): Boolean
Descrizione	Questo metodo consente ad uno Studente di prenotare una visita per l'alloggio selezionato
Pre-condizione	**context**: GestioneAffittoService::prenota_visita_alloggio(email, data_visita, id_alloggio) **pre**: email != None AND email != "" AND data_visita != None AND data_visita != "" AND id_alloggio != None
Post-condizione	**context**: GestioneAffittoService::prenota_visita_alloggio(email, data_visita, id_alloggio) **post**: GestioneAffittoService.save(email, data_visita, id_alloggio)
Nome Metodo	**affitto_alloggio_cliente**(email: str, data_inizio:date, data_fine:date, prezzo:float, numero_carta:int, data_scadenza:date, id_alloggio:int): Boolean
Descrizione	Questo metodo consente ad uno Studente di affittare un alloggio con conseguente pagamento
Pre-condizione	**context**: GestioneAffittoService::affitto_alloggio_cliente(email, data_inizio, data_fine, prezzo, numero_carta, data_scadenza, id_alloggio) **pre**: email != None AND email != "" AND data_inizio != None AND data_fine != None AND data_inizio < data_fine AND prezzo != None AND prezzo != "" AND numero_carta != None AND data_scadenza != None AND id_alloggio != None
Post-condizione	**context**: GestioneAffittoService::affitto_alloggio_cliente(email, data_inizio, data_fine, prezzo, numero_carta, data_scadenza, id_alloggio) **post**: Disponibilità != (data_inizio + data_fine)

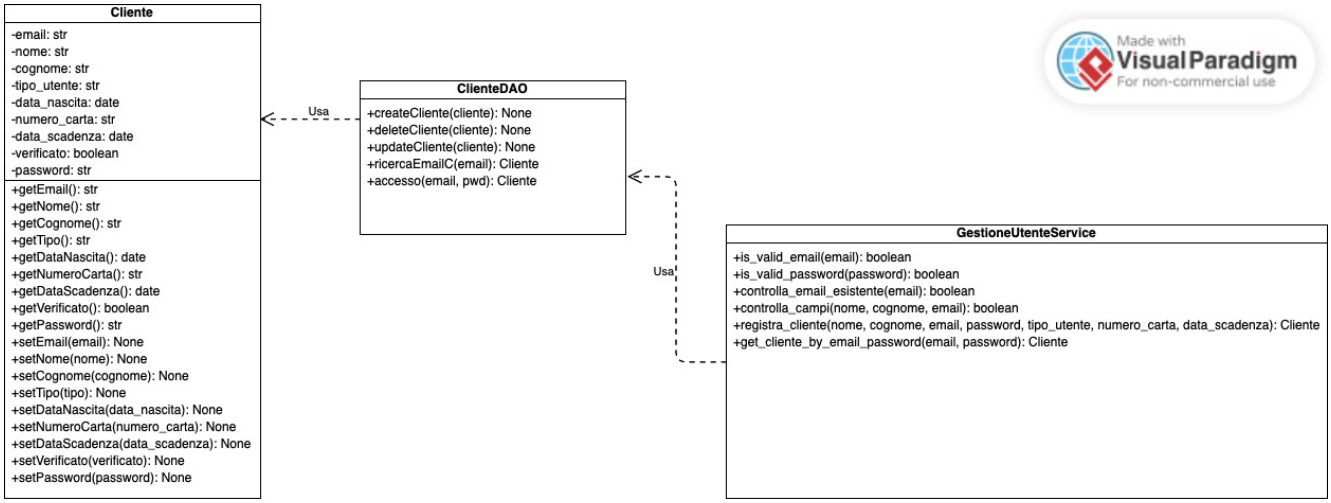
Nome Metodo	**insert_data_byId**(prenotazione: Prenotazione): void
Descrizione	Questo metodo consente ad un locatore di aggiungere delle date ad un alloggio.
Pre-condizione	**context**: GestioneAffittoService::insert_data_byId(prenotazione) **pre**: prenotazione != None
Post-condizione	**context**: GestioneAffittoService::insert_data_byId(prenotazione) **post**: prenotazione is not None
Nome Metodo	**delete_data_by_Id**(data_visita:data): void
Descrizione	Questo metodo consente ad un locatore di eliminare una data di visita ad un suo annuncio.
Pre-condizione	**context**: GestioneAffittoService::delete_data_by_Id(data_visita) **pre**: data_visita != None AND data_visita != “”
Post-condizione	**context**: GestioneAffittoService::delete_data_by_Id(data_visita) **post**: GestioneAffittoService.doDelete(data_visita)
Nome Metodo	**ottieni_date_per_alloggio_byId**(id_alloggio:int): List
Descrizione	Questo metodo permettere di avere una lista di date di visita per un determinato alloggio
Pre-condizione	**context**: GestioneAffittoService::ottieni_date_per_alloggio_byId(id_alloggio) **pre**: id_alloggio != None AND id_alloggio != “”
Post-condizione	**context**: GestioneAffittoService::delete_data_by_Id(data_visita) **post**: return List con tutte le date di quell’alloggio

4: Class Diagram Ristrutturato



5: Design Patterns

Il design pattern scelto da utilizzare nel nostro sistema è il DAO. Un DAO è un pattern che offre un'interfaccia astratta al database. UniRentHub è una piattaforma web e punta alla gestione degli affitti da parte di studenti e alla gestione degli annunci da parte dei locatori, e per questo presenta un database molto ricco, e quindi si ha la necessità di interagire col database in modo quanto più rapido e sicuro con molte query per la grande quantità di dati che si dovrà gestire.
È stato scelto di utilizzare i DAO perché ci permette di separare gli oggetti che si occupano di attività di controllo e gestione della logica di business dalla logica di accesso ai dati. Infatti, le componenti che si occupano della gestione della logica di business non devono accedere direttamente al database perché comporterebbe una scarsa manutenibilità.
Di seguito un esempio di come verrà utilizzato il design pattern DAO:



6: Glossario

Di seguito un'utile raccolta di termini chiave e relative definizioni per l'aiuto della comprensione di concetti specifici. Una guida rapida e chiara attraverso il linguaggio tecnico, fornendo una panoramica esaustiva e accessibile a tutti. Ha lo scopo di fornire chiarezza ed una migliore comprensione dei termini essenziali che sono stati utilizzati all'interno del ODD.

- **Package**: raccolta di risorse o file correlati e organizzati in una directory. Può includere codice sorgente, librerie, dati, configurazioni o altri elementi utilizzati per facilitare lo sviluppo, la distribuzione o l'esecuzione di un'applicazione o di un sistema software.
- **Design patterns**: Rappresentano modelli architetturali o strutturali che forniscono approcci standardizzati e efficaci per risolvere specifici problemi di progettazione del software. Essi offrono soluzioni già testate e documentate che possono essere riutilizzate per migliorare l'organizzazione, la manutenibilità e l'estendibilità del codice.
- **DAO**: implementazione del pattern architetturale che ha il compito di fornire un accesso astretto ai dati persistenti.
- **Controller**: classe che si occupa di gestire le richieste del client.
- **Service**: classe che implementa la logica di business e viene utilizzata dal controller o da altri sottosistemi.