Università degli Studi di Salerno
Corso di Ingegneria del Software

ResolveIT
System Design Document
Versione 1.3

Data: 18/01/2026

Partecipanti:

Nome	Matricola
Samuele Nacchia	0512119128
Andrea Generale	0512119134
Mario Di Feo	0512119320

Scritto da:	MDF, AG, SN

Revision History

Data	Versione	Descrizione	Autore
18/12/2025	1.0	Introduction, Packages and Class interface	AG, SN, MDF
19/12/2025	1.1	Approfondimento Class interfaces	MDF
20/12/2025	1.2.0	Completamento Packages, Class Interface	MDF
27/12/2025	1.2.1	Modifica AutenticazioneImpl e RegistrazioneService	SN
17/01/2026	1.2.2	Modifica Class Interfaces usando Repository	MDF
18/01/2026	1.3	Revisione finale	AG, SN, MDF

Indice

1. Introduction

ResolveIT si propone di fornire un sistema di assistenza clienti personalizzabile per più piattaforme. In questa prima sezione del documento, verranno descritti i trade-offs e le linee guida per la fase di implementazione, riguardanti la nomenclatura, la documentazione e le convenzioni sui formati.

1.1 Object design trade-offs

Riusabilità:
Il sistema deve basarsi sulla riusabilità, attraverso l'utilizzo di design patterns.

Robustezza:
Il sistema deve risultare robusto, reagendo correttamente a situazioni impreviste attraverso il controllo degli errori e la gestione delle eccezioni.

Incapsulamento:
Il sistema garantisce la segretezza sui dettagli implementativi delle classi grazie all'utilizzo delle interfacce, rendendo possibile l'utilizzo di funzionalità offerte da diversi componenti.

Manutenibilità:
L'implementazione avverrà usando un approccio orientato all'information hiding e il sistema si baserà su un'architettura chiusa.

1.2 Interface documentation guidelines

- I nomi delle classi seguiranno specifica UpperCamelCase.
- Le variabili e i metodi seguiranno la specifica lowerCamelCase
- Le classi avranno un nome al singolare.
- Gli errori verranno restituiti tramite eccezioni e non tramite valori di ritorno.

1.3 Definitions, acronyms and abbreviation

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;
- **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;
- **lowerCamelCase:** è la pratica di scrivere frasi in modo tale che ogni parola o abbreviazione nel mezzo della frase inizi con una lettera maiuscola, senza spazi o punteggiatura intermedi (es .methodName());
- **UpperCamelCase**: è la pratica di scrivere frasi in modo tale che ogni parola o abbreviazione inizi con una lettera maiuscola, senza spazi o punteggiatura intermedi (es. ClassName);
- **Javadoc**: sistema di documentazione offerto da Java, che viene generato sotto forma di interfaccia in modo da rendere la documentazione accessibile e facilmente leggibile.

1.4 References

- RAD_ResolveIT (Requirements Analysis Document);
- SDD_ResolveIT (System Design Document);
- TP_ResolveIT (Test Plan);

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 di directory standard definita da Maven.

- .idea: pacchetto contenente le impostazioni del progetto su IntelliJ
- .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
        - resources, contiene i file relativi alle componenti View
            - static, contiene i fogli di stile CSS e gli script JS
            - templates, contiene i file HTML renderizzati da Thymeleaf
    - 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

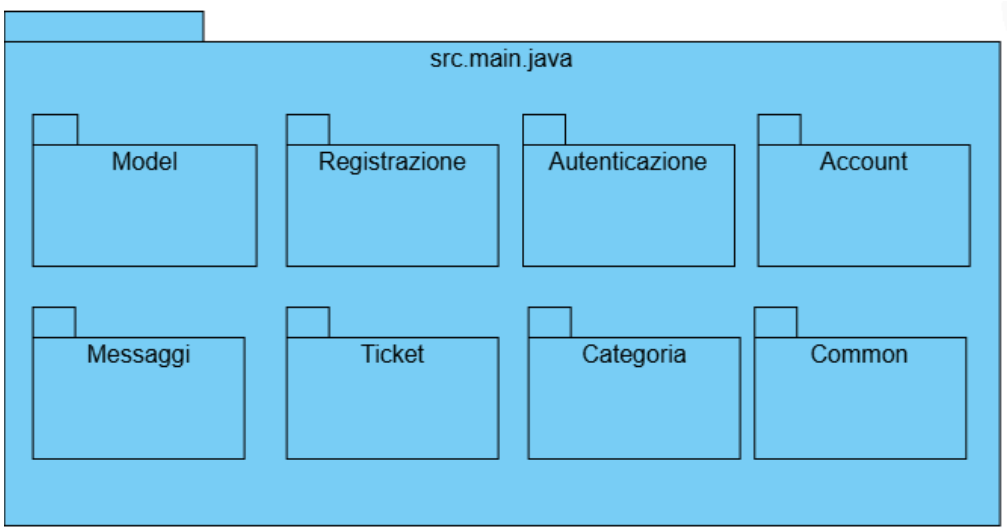
Package ResolveIT

Nella presente sezione si mostra la struttura del package principale di ResolveIT.
La struttura generale è stata ottenuta a partire da due principali scelte:

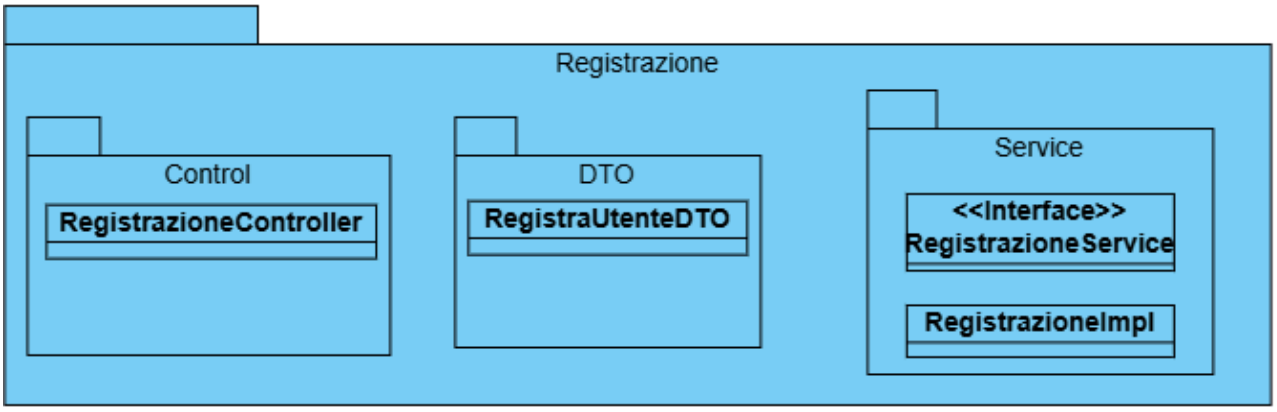
1. Creare un package separato per i sottosistemi individuati, ognuno dei quali contenenti i sub-package Control e Service, i quali contengono rispettivamente le classi Control e Service, e, se presenti, le classi utilizzate solamente dal sub-package.
2. Creare un package separato per le classi che rappresentano il model, contenente le classi entity e repository per l'accesso ai dati persistenti sul database.

La suddivisione precedentemente illustrata ha portato alla creazione di una relazione tra il package Model e tutti gli altri package del sistema.

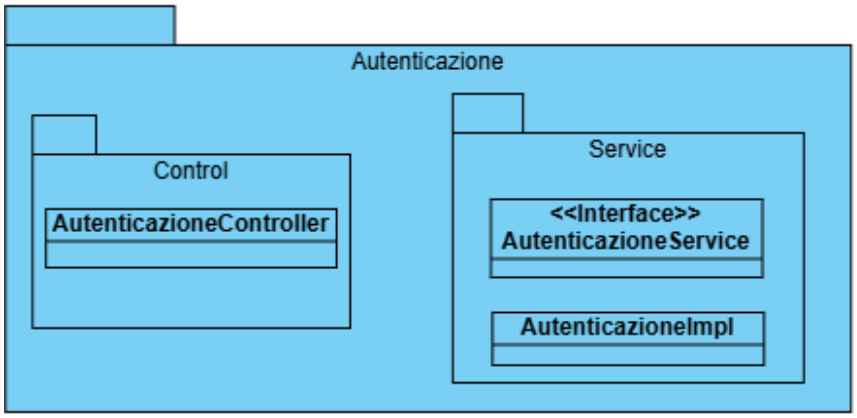
Package Java



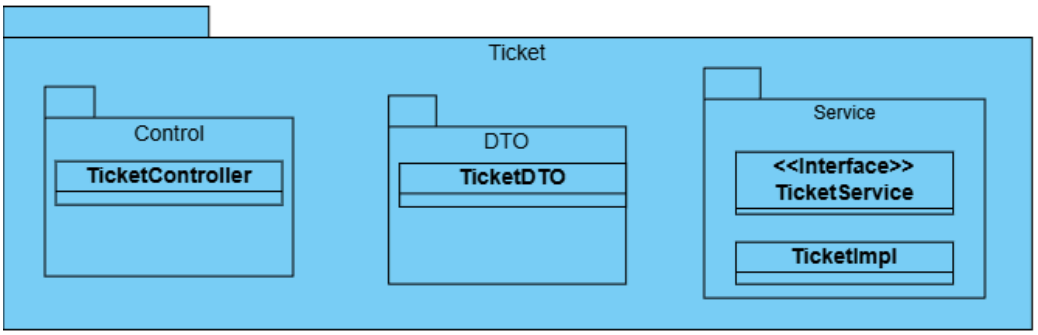
Package Registrazione



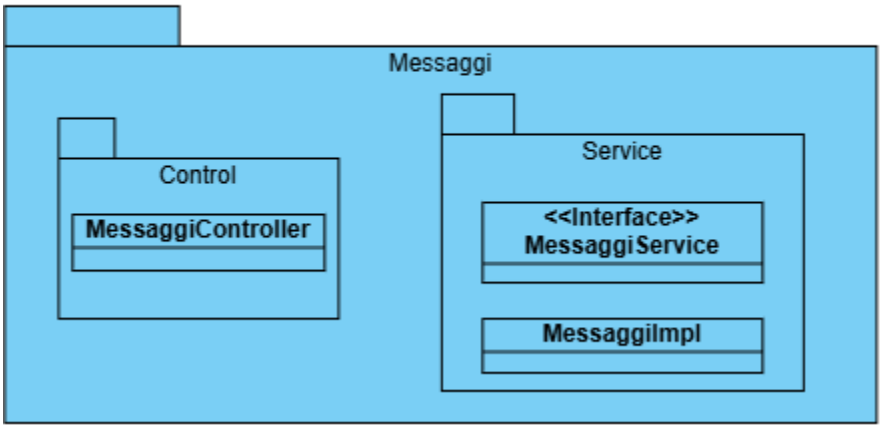
Package Autenticazione



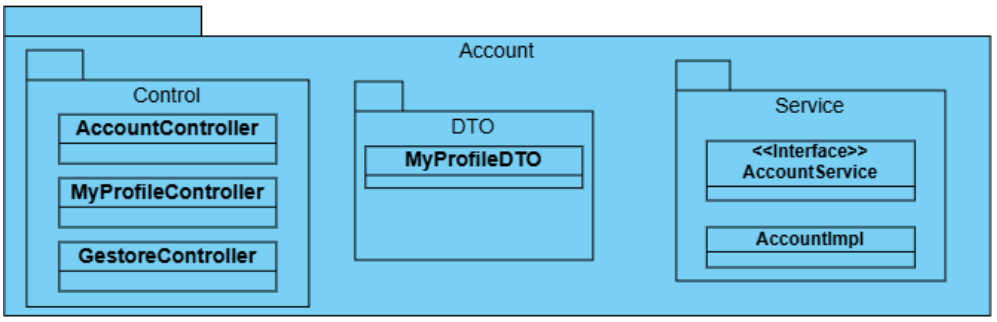
Package Ticket



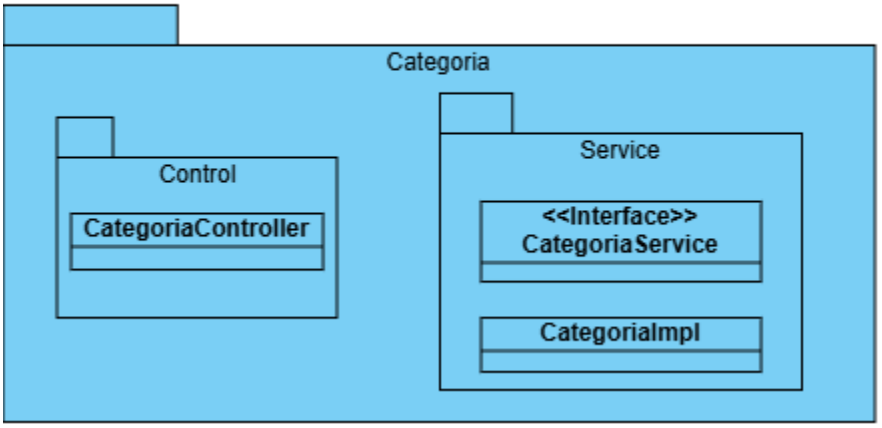
Package Messaggi (Non implementato)



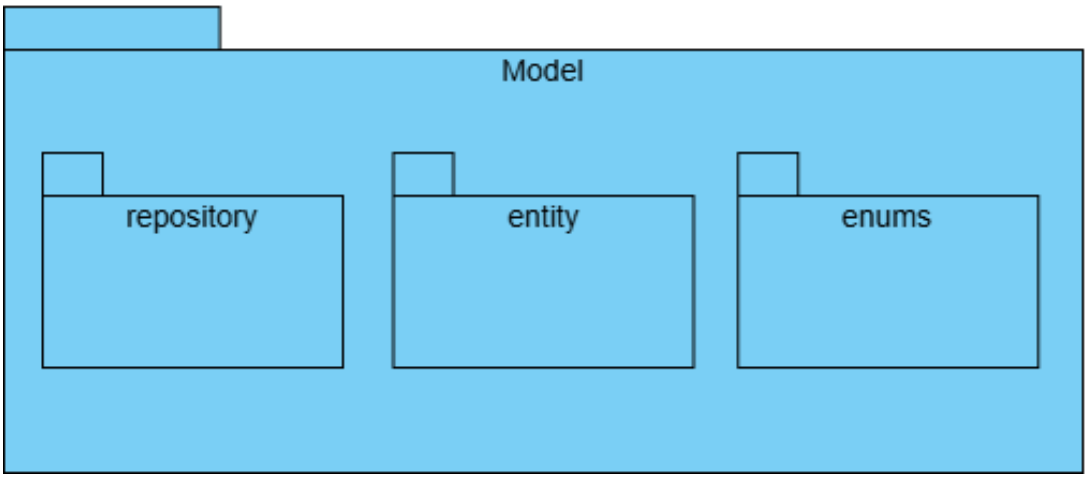
Package Account



Package Categoria



Package Model



3. Class interfaces

Di seguito saranno presentate le interfacce di ciascun package (non sarà presente il package model, le classi controller).

Classi utilizzate	Descrizione
UserDetails	Interfaccia fornita da Spring Security che rappresenta un utente autenticabile. Include username (email), password hashata e collezione di ruoli (Authorities).
RegistraUtenteDTO	Oggetto utilizzato per trasferire i dati necessari alla registrazione di un nuovo utente dal livello di presentazione al sottosistema di Registrazione. I dati contenuti nel DTO vengono validati dal servizio di registrazione prima della creazione delle entità di dominio corrispondenti.

Package Registrazione

Nome classe	RegistrazioneService
Descrizione	Fornisce la logica di business per la creazione di nuovi account. Gestisce la distinzione tra la registrazione autonoma dei Clienti e la registrazione degli Operatori effettuata dai Gestori.
Metodi	+registerClient(RegistraUtenteDTO userDto) : UserDetails +registerOperator(RegistraUtenteDTO userDto) : void
Invariante di classe	**context** RegistrazioneService **inv:** global_unique_email: ClienteRepository.allInstances()->forAll(c | not OperatoreRepository.existsByEmail(c.email) and not GestoreRepository.existsByEmail(c.email)) and OperatoreRepository.allInstances()->forAll(o | not GestoreRepository.existsByEmail(o.email))

Nome Metodo	+registerClient(RegistraUtenteDTO userDto) : UserDetails
Descrizione	Crea un nuovo profilo Cliente. Valida i dati, effettua l'hashing della password e salva l'entità. Restituisce le UserDetails per consentire l'auto-login.
Pre-condizione	**context** RegistrazioneService::registerClient(userDto) **pre:** userDto <> null and userDto.email <> null and not (ClienteRepository.existsByEmail(userDto.email) or OperatoreRepository.existsByEmail(userDto.email) or GestoreRepository.existsByEmail(userDto.email)) and userDto.password = userDto.confermaPassword
Post-condizione	**context** RegistrazioneService::registerClient(userDto) **post:** result <> null and ClienteRepository.existsByEmail(userDto.email) and result.getPassword() <> userDto.password

Eccezioni	Se id è già presente nella repository, se l'email è già presente nella repository o le password non corrispondono: throws RuntimeException

Nome Metodo	+registerOperator(RegistraUtenteDTO userDto) : void
Descrizione	Permette ad un Gestore di registrare un nuovo account Operatore sul sistema.
Pre-condizione	**context** RegistrazioneService::registerOperator(userDto) **pre:** userDto <> null and userDto.email <> null and userDto.password <> null and SecurityContextHolder.getContext().getAuthentication() .getAuthorities()->exists(a | a.authority = 'GESTORE') and not (ClienteRepository.existsByEmail(userDto.email) or OperatoreRepository.existsByEmail(userDto.email) or GestoreRepository.existsByEmail(userDto.email))
Post-condizione	**context** RegistrazioneService::registerOperator(userDto) **post**: OperatoreRepository.existsByEmail(userDto.email) and OperatoreRepository.get(userDto.email).password <> userDto.password and OperatoreRepository.get(userDto.email).password = PasswordEncoder.encode(userDto.password)
Eccezioni	Se id è già presente nella repository, se l'email è già presente nella repository o le password non corrispondono: throws RuntimeException

Package Autenticazione

Nome classe	AutenticazioneImpl
Descrizione	Implementa l'interfaccia UserDatailService. Fornisce al framework le credenziali e i ruoli di Clienti, Operatori e Gestori cercandoli nei rispettivi repository. Delega a Spring Security la logica di validazione password, gestione sessione e logout.
Metodi	+loadUserByUsername(String email) : UserDetails
Invariante di classe	**context** AutenticazioneImpl **inv** SecurityContextHolder.getContext().getAuthentication() <> null implies (GestoreRepository.existsByEmail(email) or OperatoreRepository.existsByEmail(email) or ClienteRepository.existsByEmail(email))

Nome Metodo	+loadUserByUsername(String email) : UserDetails
Descrizione	Recupera i dati dell'utente dal database tramite email. Se l'utente è trovato in una delle tre tabelle, restituisce un oggetto UserDetails che include la password hashata e i ruoli (Authorities).
Pre-condizione	**context** AutenticazioneImpl::loadUserByUsername(email) **pre**: email <> null and email.size() > 0
Post-condizione	**context** AutenticazioneImpl::loadUserByUsername(email) **post**: result.getAuthorities() <> null
Eccezioni	Se l'email non è presente nella repository: throws UsernameNotFoundException

Package Account

Nome classe	AccountService
Descrizione	È responsabile dell'eliminazione degli account, della visualizzazione e modifica dei dati personali e del ripristino password.
Metodi	+removeAccount(Account account): void +modifyUser(Cliente account) : boolean +modifyOperator(Operatore account) : boolean
Invariante di classe	**context** AccountImpl **inv** account_consistency: User.allInstances()->forAll(u | ClienteRepository.existsByEmail(u.email) <> NULL) and OperatoreRepository.findAll()-> forAll (o | OperatoreRepository.existByEmail(o.email))

Nome Metodo	removeAccount(Account account): Boolean
Descrizione	Permette ad un Gestore di eliminare un account Utente o Operatore
Pre-condizione	**context** AccountImpl::removeAccount(account) **pre**: account <> null and (ClienteRepository.existsByEmail(account.email) or OperatoreRepository.existsByEmail(account.email))
Post-condizione	**context** AccountImpl::removeAccount(account) **post**: (not ClienteRepository.existsByEmail(account.email) and OperatoreRepository.findByEmail(account.email).attivo = false)
Eccezioni	Se id non è presente in nessuno dei repository o l'account è già disabilitato : throws RuntimeException

Nome Metodo	modifyUser(Cliente account) : void
Descrizione	Permette ad un Cliente di aggiornare i propri dati personali e le credenziali di accesso.
Pre-condizione	**context** AccountImpl::modifyUser(account) **pre**: account <> null and ClienteRepository.existByEmail(account.email)
Post-condizione	**context** AccountImpl::modifyUser(account) **post**:ClienteRepository.findByEmail(account.email).nome = account.nome and ClienteRepository.findByEmail(account.email).password = account.password result = true implies ClienteRepository.findByEmail(account.email).password <> account.password@pre
Eccezioni	Se id non è presente nella repository o le nuove password non corrispondono: throws RuntimeException

Nome Metodo	modifyOperator(Operatore account) : void
Descrizione	Permette ad un Operatore di modificare i propri dati personali e le credenziali di accesso
Pre-condizione	**context** AccountImpl::modifyOperator(account) **pre**: account <> null and OperatoreRepository.existsByEmail(account.email)
Post-condizione	**context** AccountImpl::modifyOperator(account) **post**: OperatoreRepository.findByEmail(account.email).nome = account.nome and OperatoreRepository.findByEmail(account.email).password = account.password result = true implies OperatoreRepository.findByEmail(account.email).password <> account.password@pre
Eccezioni	Se id non è presente nella repository o le nuove password non corrispondono: throws RuntimeException

Package Ticket

Nome classe	TicketService
Descrizione	Si occupa della gestione dei ticket, creazione, eliminazione, assegnazione e risoluzione.
Metodi	+addTicket (Ticket ticket, Cliente cliente) : void +deleteTicket (Long ticketId) : void +assignTicket (Long ticketId, Operatore operatore) : void +resolveTicket(Long ticketId) : void +releaseTicket (Long ticketId) : void +getTicketUtente(Cliente cliente): List<Ticket> +getTicketDisponibili(): List<Ticket> +getTicketInCarico(Operatore operatore): List<Ticket>
Invariante di classe	**context** TicketImpl **inv** session_validity: self.currentUser <> null implies (ClienteRepository.existsByEmail(self.currentUser.email) or OperatoreRepository.existsByEmail(self.currentUser.email))

Nome Metodo	addTicket (TicketDTO ticket, Cliente autore) : void
Descrizione	Permette ad un Cliente di inviare un nuovo ticket di supporto tecnico
Pre-condizione	**context** Ticket::addTicket(ticket, autore) **pre**: ticket <> null and not TicketRepository.existsById(ticket.id)
Post-condizione	**context** Ticket::addTicket(ticket, autore) **post**: TicketRepository.existsById(ticket.id)
Eccezioni	Se la categoria selezionata non è valida, l'allegato è di dimensione troppo grande o di un formato non accettato, il titolo non rispetta il formato previsto o la descrizione supera il limite di caratteri consentito : throws RuntimeException

Nome Metodo	deleteTicket (Long ticketId) : void
Descrizione	Permette ad un Cliente di annullare un ticket di supporto da lui inviato
Pre-condizione	**context** Ticket::deleteTicket(ticket) **pre**: ticket <> null and TicketRepository.existsById(ticket.id) and ticket.stato = “aperto”
Post-condizione	**context** Ticket::deleteTicket(ticket) **post**: TicketRepository.findById(ticket.id).stato = “ANNULLATO” [eliminazione logica]
Eccezioni	Se id non è presente nella repository o il ticket è in uno stato diverso da “aperto”: throws RuntimeException

Nome Metodo	releaseTicket (Long ticketId) : void
Descrizione	Permette ad un Operatore di rilasciare un ticket da lui preso in carico e riportarlo come aperto
Pre-condizione	**context** Ticket::releaseTicket(ticket) **pre**: ticket <> null and ticket.stato = “in_corso”
Post-condizione	**context** Ticket::releaseTicket(ticket) **post**: ticket.operatoreAssegnato = null and ticket.stato = “aperto”
Eccezioni	Se l’id non è presente nella repository o il ticket è in uno stato diverso da “aperto”: throws RuntimeExceptionthrows RuntimeException

Nome Metodo	assignTicket (Long ticketId, Operatore operatore) : void
Descrizione	Permette ad un Operatore di prendere in carico un ticket di supporto per risolverlo.
Pre-condizione	**context** Ticket::assignTicket(operatore, ticket) **pre**: ticket <> null and operatore <> null and ticket.stato = “aperto” and OperatoreRepository.existsByEmail(operatore.email)
Post-condizione	**context** Ticket::assignTicket(operatore, ticket) **post**: ticket.operatoreAssegnato = operatore and ticket.stato = “in_corso”
Eccezioni	Se l’id non è presente nella repository, il ticket è in uno stato diverso da “aperto” o l’account operatore non è valido: throws RuntimeException

Nome Metodo	resolveTicket(Long ticketId) : void
Descrizione	Permette ad un Operatore di contrassegnare come risolto.
Pre-condizione	**context** Ticket::resolveTicket(ticket) **pre**: ticket <> null and ticket.stato = “in_corso”
Post-condizione	**context** Ticket::resolveTicket(ticket) **post**: ticket.stato = “risolto”
Eccezioni	Se l’id non è presente nella repository o il ticket è in uno stato diverso da “aperto”: throws RuntimeExceptionthrows RuntimeException

Nome Metodo	getTicketUtente(Cliente cliente): List<Ticket>
Descrizione	Recupera tutti i ticket inviati da uno specifico cliente.
Pre-condizione	**context** Ticket::getTicketUtente(cliente) **pre**:cliente <> null
Post-condizione	**context** Ticket::getTicketUtente(cliente) **post**: post: result->forAll(t | t.cliente = cliente)

Nome Metodo	getTicketDisponibili(): List<Ticket>
Descrizione	Recupera i ticket aperti in attesa di assegnazione.
Pre-condizione	**context** Ticket::getTicketDisponibili() **pre**: //
Post-condizione	**context** Ticket::getTicketDisponibili() **post**: post: result->forAll(t | t.stato = "aperto")

Nome Metodo	getTicketInCarico(Operatore operatore): List<Ticket>
Descrizione	Recupera i ticket attualmente in lavorazione da un operatore.
Pre-condizione	**context** Ticket::getTicketInCarico(operatore) **pre**: operatore <> null
Post-condizione	**context** Ticket::getTicketInCarico(operatore) **post**: result->forAll(t | t.operatoreAssegnato = operatore and t.stato = "in_corso")

Package Categoria

Nome classe	CategoriaImpl
Descrizione	Responsabile della creazione, modifica ed eliminazione delle categorie dei ticket**.**
Metodi	+addCategoria(Categoria categoria) : void +updateCategoria(String nome, Categoria categoria) : void +disableCategoria (Categoria categoria) : void +enableCategoria (Categoria categoria) : void
Invariante di classe	**context** CategoriaImpl **inv** unique_category_name: CategoriarRepository.FindAll()->isUnique(c | c.nome)

Nome Metodo	addCategoria(Categoria categoria) : Boolean
Descrizione	Permette ad un Gestore di aggiungere una nuova categoria sul sistema
Pre-condizione	**context** CategoriaImpl:: addCategoria(categoria) **pre:** categoria <> null and CategoriaRepository.findByNome(categoria.nome) = NULL
Post-condizione	**context** CategoriaImpl::addCategoria(categoria) **post:** CategoriaRepository.findByNome(categoria.nome) <> NULL
Eccezioni	Se id è già presente nella repository, o il nome è già presente: throws RuntimeException Se la categoria non rispetta il formato richiesto: throws IllegalArgumentException

Nome Metodo	updateCategoria(String nome, Categoria categoria) : Boolean
Descrizione	Permette ad un Gestore di modificare una categoria esistente sul sistema
Pre-condizione	**context** CategoriImpl::updateCategoria(nome, categoria) **pre:** categoria <> null and CategoriaRepostiroy.findByNome(categoria.nome).stato = true
Post-condizione	**context** CategoriaImpl::updateCategoria(nome, categoria) **post:** CategoriaRepository.findByNome(nome)<> NULL and CategoriaRepository.findByNome(categoria.nome@pre) = NULL
Eccezioni	Se id non è presente nella repository, o il nome è già presente: throws RuntimeException Se la categoria non rispetta il formato richiesto: throws IllegalArgumentException

Nome Metodo	disableCategoria(Categoria categoria) : Boolean
Descrizione	Permette ad un Gestore di disabilitare una categoria presente nel sistema
Pre-condizione	**context** CategoriaImpl::deleteCategoria(categoria) **pre:** categoria <> null and CategoriaRepostiroy.findByNome(categoria.nome).stato = true
Post-condizione	**context** CategoriaImpl::deleteCategoria(categoria) **post**: CategoriaRepostiroy.findByNome(categoria.nome).stato = false
Eccezioni	Se id non è presente nella repository o la categoria è già disabilitata: throws RuntimeException

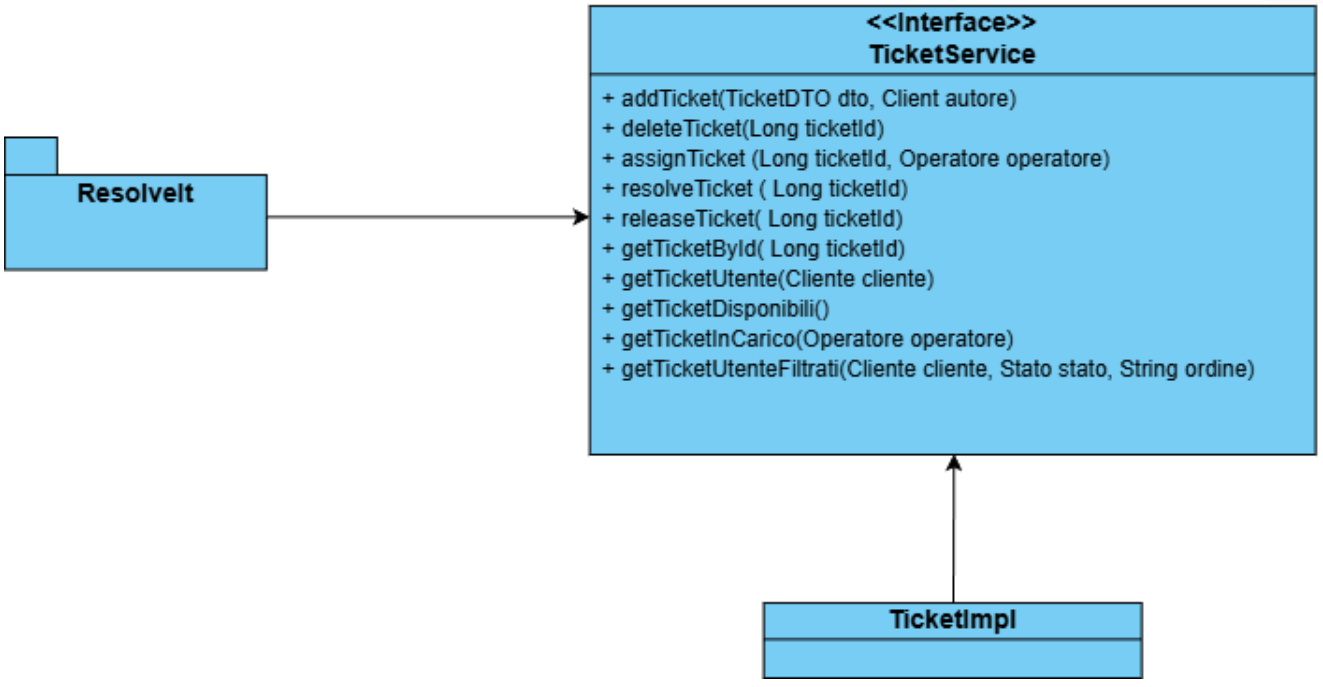
Nome Metodo	enableCategoria(Categoria categoria) : Boolean
Descrizione	Permette ad un Gestore di riattivare una categoria presente nel sistema e precedentemente disabilitata
Pre-condizione	**context** CategoriaImpl::deleteCategoria(categoria) **pre:** categoria <> null and CategoriaRepostiroy.findByNome(categoria.nome).stato = false
Post-condizione	**context** CategoriaImpl::deleteCategoria(categoria) **post**: CategoriaRepostiroy.findByNome(categoria.nome).stato = true
Eccezioni	Se id non è presente nella repository o la categoria è già abilitata: throws RuntimeException

4. Design Pattern

Facade Pattern

Il Facade è un design pattern che permette, implementando un' 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 manutenibile e più aggiornabile, poiché basterà cambiare l'implementazione dei metodi dell'interfaccia per implementare le modifiche. ResolveIT sfrutta il design pattern Facade per implementare tutta la sua logica di business e rendere più facile interfacciarsi con essa. Nello specifico ResolveIT utilizza il Facade per ogni suo sottosistema, implementandolo attraverso delle interfacce che sono usate per accedere ai metodi interni.

Di seguito un esempio di Facade nel sistema ResolveIT:



5. Glossario

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
Facade	Un design pattern che permette, attraverso interfacce un più semplice accesso ai sottosistemi.
MVC	Model-View-Controller: design architetturale che permette di separare la logica di presentazione dalla logica di business alla base del sistema.
DTO	Oggetto per incapsulare i dati di input da passare ad un service.
Thymeleaf	Template engine per HTML per rendere la pagina parametrizzabile al fine di generare pagine dinamiche