Università degli Studi di Salerno
Corso di Ingegneria del Software

ResolveIT
System Design Document
Versione 1.2

Data: 20/12/2025

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	Completamento Packages, Class Interface	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 ereditarietà e 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 ritornati 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

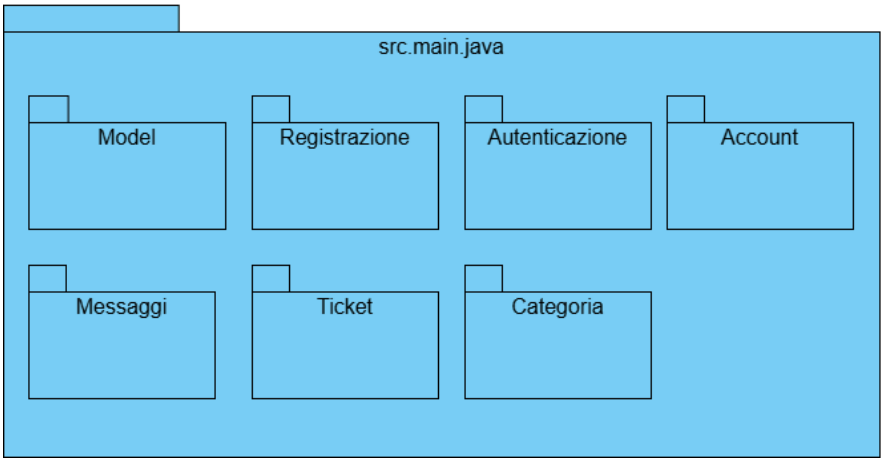
Package ResolveIT

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

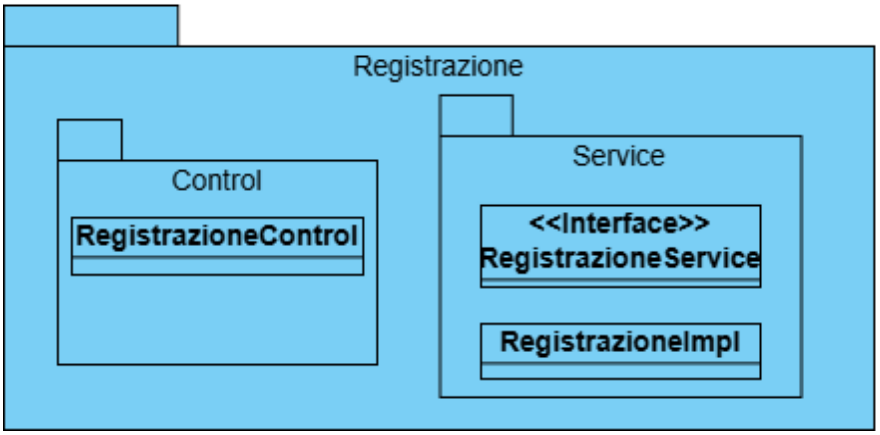
1. Creare un package separato per i sottosistemi individuati, ognuno dei quali contenenti i sub-package Control e Service, i quali contengono rispettivamente le classi Control e Service, e, se presenti, le classi utilizzate solamente dal sub-package.
2. Creare un package separato per le classi che rappresentano il model, contenente le classi entity e DAO 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.

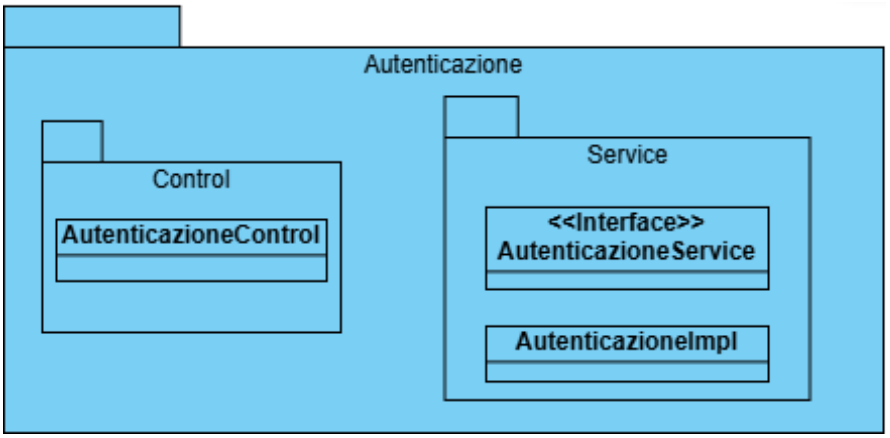
Package Java



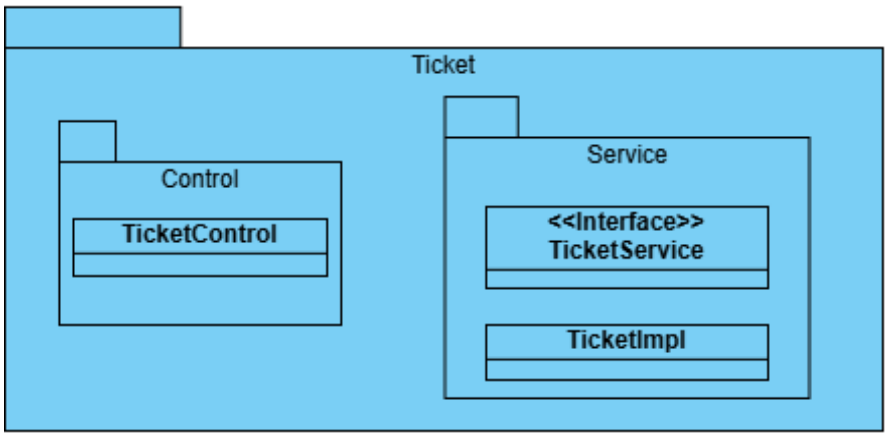
Package Registrazione



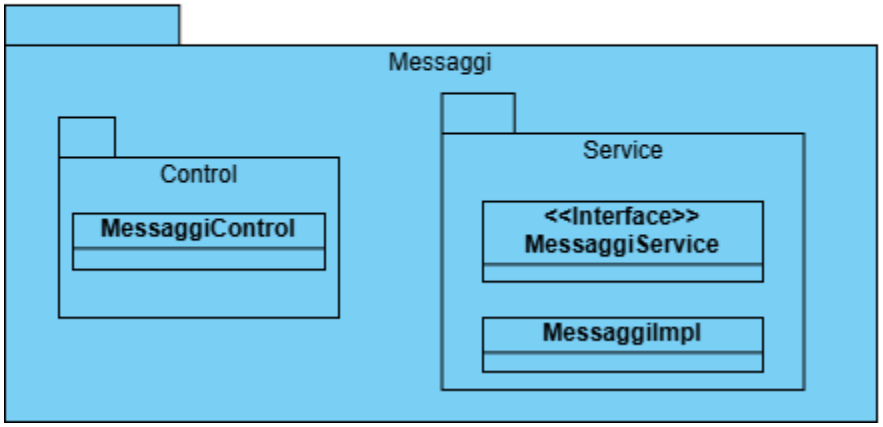
Package Autenticazione



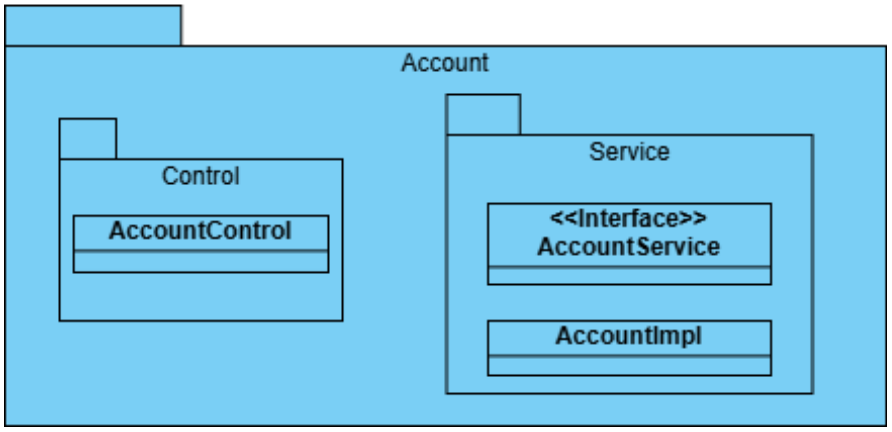
Package Ticket



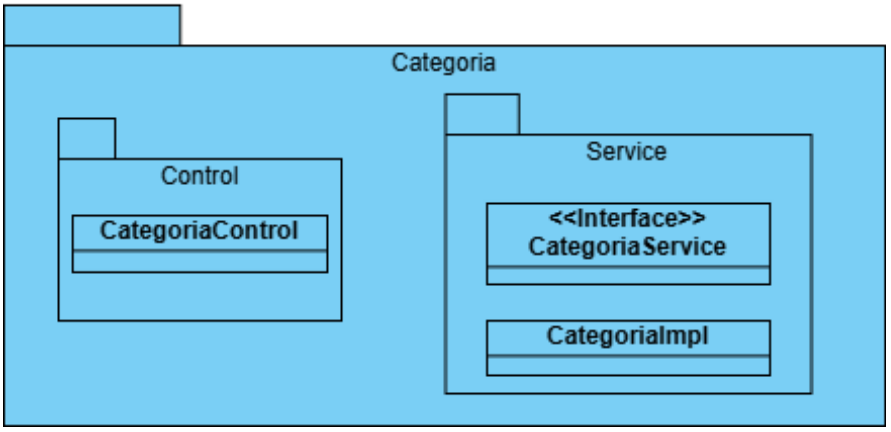
Package Messaggi



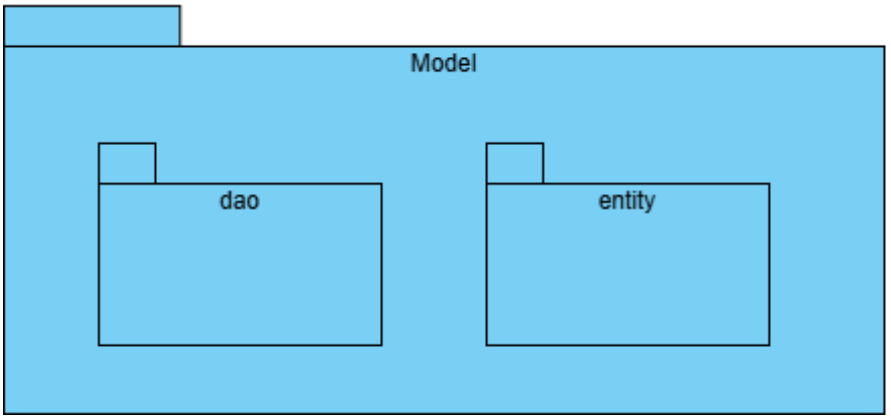
Package Account



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]

Package Registrazione

Nome classe	RegistrazioneImpl
Descrizione	Permette ad un cliente di registrarsi sul sistema e ad un Gestore di registrare un operatore sul sistema
Metodi	+registerUser(User account) : Boolean +registerOperator(Operator account) : Boolean
Invariante di classe	**context** RegistrazioneImpl inv unique_accounts: UserDAO.allInstances()->forAll(u1, u2 | u1 <> u2 implies u1.email <> u2.email)

Nome Metodo	registerUser(User account)
Descrizione	Permette ad un utente di registrarsi sul sistema
Pre-condizione	**context** RegistrazioneImpl::registerUser(account) **pre:** account <> null and UserDAO.findByEmail(account.email) = null
Post-condizione	**context** RegistrazioneImpl::registerUser(account: User) **post:** result = true implies UserDAO.findByEmail(account.email) <> null

Nome Metodo	registerOperator(Operator account)
Descrizione	Permette ad un Gestore di registrare un nuovo account Operatore sul sistema
Pre-condizione	**context** RegistrazioneImpl::registerOperator(account) **pre:** account <> null and OperatorDAO.findByEmail(account.email) = null
Post-condizione	**context** RegistrazioneImpl::registerOperator(account) **post**: Operator.allInstances()->exists(o | o.email = account.email) and result = true

Package Autenticazione

Nome classe	AutenticazioneImpl
Descrizione	Permette a clienti, operatori e gestori del sistema di accedere o uscire dal sistema. Ogni utente accede a diverse pagine home in base al ruolo
Metodi	+login(String email, String password) : Boolean +logOut() : void
Invariante di classe	**context** AutenticazioneImpl **inv** session_validity: self.currentUser <> null implies (UserDAO.exists(self.currentUser) or OperatorDAO.exists(self.currentUser) or GestoreDAO.exists(self.currentUser))

Nome Metodo	+login(String email, String password) : Boolean
Descrizione	Permette ad un Cliente, Operatore o Gestore di autenticarsi al sistema e accedervi
Pre-condizione	**context** AutenticazioneImpl::login(email, password) **pre**: email <> null and password <> null and self.currentUser = null
Post-condizione	**context** AutenticazioneImpl::login(email, password) **post**: result = true implies (self.currentUser <> null and (UserDAO.verify(email, password) or OperatorDAO.verify(email, password) or GestoreDAO.verify(email, password)))

Nome Metodo	+logout() : void
Descrizione	Permette ad un Cliente, Operatore o Gestore di uscire dal sistema
Pre-condizione	**context** AutenticazioneImpl::logOut() **pre**: self.currentUser <> null
Post-condizione	**context** AutenticazioneImpl::logOut() **post**: self.currentUser = null

Package Account

Nome classe	AccountImpl
Descrizione	È responsabile dell'eliminazione degli account, della visualizzazione e modifica dei dati personali e del ripristino password.
Metodi	+removeAccount(Account account): Boolean +updateUser(User account) : void +updateOperator(Operator account) : void
Invariante di classe	**context** AccountImpl **inv** account_consistency: User.allInstances()->forAll(u | UserDAO.exists(u.email)) and Operator.allInstances()-> forAll (o | OperatorDAO.exists(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 (UserDAO.exists(account.email) or OperatorDAO.exists(account.email))
Post-condizione	**context** AccountImpl::removeAccount(account) **post**: result = true implies (not UserDAO.exists(account.email) and not OperatorDAO.exists(account.email))

Nome Metodo	updateUser(User account) : void
Descrizione	Permette ad un Cliente di aggiornare i propri dati personali e le credenziali di accesso.
Pre-condizione	**context** AccountImpl::updateUser(account) **pre**: account <> null and UserDAO.exists(account.email)
Post-condizione	**context** AccountImpl::updateUser(account) **post**: UserDAO.findByEmail(account.email).nome = account.nome and UserDAO.findByEmail(account.email).password = account.password

Nome Metodo	updateOperator(Operator account) : void
Descrizione	Permette ad un Operatore di modificare i propri dati personali e le credenziali di accesso
Pre-condizione	**context** AccountImpl::updateOperator(account) **pre**: account <> null and OperatorDAO.exists(account.email)
Post-condizione	**context** AccountImpl::updateOperator(account) **post**: OperatorDAO.findByEmail(account.email).nome = account.nome and OperatorDAO.findByEmail(account.email).password = account.password

Package Ticket

Nome classe	TicketCollection
Descrizione	Si occupa della gestione dei ticket, creazione, eliminazione, assegnazione e risoluzione.
Metodi	+addTicket (Ticket ticket) : Boolean +deleteTicket (Ticket ticket) : Boolean +assignTicket (Operatore operatore, Ticket ticket) : Boolean +resolveTicket(Ticket ticekt) : Boolean
Invariante di classe	**context** TicketCollection **inv** session_validity: self.currentUser <> null implies (UserDAO.exists(self.currentUser.email) or OperatorDAO.exists(self.currentUser.email))

Nome Metodo	addTicket (Ticket ticket) : Boolean
Descrizione	Permette ad un Cliente di inviare un nuovo ticket di supporto tecnico
Pre-condizione	**context** Ticket::addTicket(ticket) **pre**: ticket <> null and not TicketDAO.exists(ticket.id)
Post-condizione	**context** Ticket::addTicket(ticket) **post**: result = true implies TicketDAO.exists(ticket.id)

Nome Metodo	deleteTicket (Ticket ticket) : Boolean
Descrizione	Permette ad un Cliente di annullare un ticket di supporto da lui inviato
Pre-condizione	**context** Ticket::deleteTicket(ticket) **pre**: ticket <> null and TicketDAO.exists(ticket.id) and ticket.stato = “aperto”
Post-condizione	**context** Ticket::deleteTicket(ticket) **post**: result = true implies not TicketDAO.exists(ticket.id)

Nome Metodo	releaseTicket (Ticket ticket) : Boolean
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 = “assegnato”
Post-condizione	**context** Ticket::releaseTicket(ticket) **post**: result = true implies (ticket.operatoreAssegnato = null and ticket.stato = “aperto”)

Nome Metodo	assignTicket (Operatore operatore, Ticket ticket) : Boolean
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 OperatorDAO.exists(operatore.email)
Post-condizione	**context** Ticket::assignTicket(operatore, ticket) **post**: result = true implies (ticket.operatoreAssegnato = operatore and ticket.stato = “assegnato”)

Nome Metodo	resolveTicket(Ticket ticekt) : Boolean
Descrizione	Permette ad un Operatore di contrassegnare come risolto.
Pre-condizione	**context** Ticket::resolveTicket(ticket) **pre**: ticket <> null and ticket.stato = “assegnato”
Post-condizione	**context** Ticket::resolveTicket(ticket) **post**: result = true implies ticket.stato = “risolto”

Package Categoria

Nome classe	CategoriaImpl
Descrizione	Responsabile della creazione, modifica ed eliminazione delle categorie dei ticket**.**
Metodi	+addCategoria(Categoria categoria) : Boolean +updateCategoria(String nome, Categoria categoria) : Boolean +deleteCategoria (Categoria categoria) : Boolean
Invariante di classe	**context** CategoriaImpl **inv** unique_category_name: CategoriaDAO.allInstances()->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::updateCategoria(nome, categoria) **pre:** categoria <> null and CategoriaDAO.exists(categoria.nome)
Post-condizione	**context** CategoriaImpl::addCategoria(categoria) **post:** result = true implies CategoriaDAO.exists(categoria.nome)

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**: account <> null and UserDAO.exists(account.email)
Post-condizione	**context** CategoriaImpl::updateCategoria(nome, categoria) **post:** result = true implies (CategoriaDAO.exists(nome) and not CategoriaDAO.exists(categoria.nome@pre))

Nome Metodo	deleteCategoria(Categoria categoria) : Boolean
Descrizione	Permette ad un Gestore di eliminare una categoria presente nel sistema
Pre-condizione	**context** CategoriaImpl::deleteCategoria(categoria) **pre:** categoria <> null and CategoriaDAO.exists(categoria.nome)
Post-condizione	**context** CategoriaImpl::deleteCategoria(categoria) **post**: result = true implies not CategoriaDAO.exists(categoria.nome)