**Università degli Studi di Salerno**
Corso di Ingegneria del Software


# Object Design Document - HeavyRoute

## Versione 1.0

**Informazioni Generali**

| | |
|---|---|
| **Corso di Laurea:** | Informatica |
| **Università:** | Università degli Studi di Salerno (UNISA) |
| **Docente:** | Chiar.mo Prof. DE LUCIA Andrea |
| **Data:** | 18/12/2025 |
| **Anno Accademico:** | 2025/2026 |

## Membri Del Gruppo


| | |
|---|---|
| **MANFREDINI Umberto** | Matricola 0512119797 |
| **MANZO Ugo** | Matricola 0512119071 *(Coordinatore)* |
| **ROMANO Pino Fiorello** | Matricola 0512120259 |


## Revision History

| Data | Versione | Descrizione | Autore |
|---|---|---|---|
| 18/12/2025 | 1.0 | Creazione del ODD | Ugo Manzo |
| 04/01/2026 | 1.0.1 | Modifiche Minori | Pino Fiorello Romano |
| 06/01/2026 | 1.0.2 | Correzione discrepanze con il codice | Umberto Manfredini |
| 13/01/2026 | 1.0.3 | Refactoring classi e aggiornamento vincoli OCL | Umberto Manfredini |

# Indice

# 1. Introduction

## 1.1. Object design trade-offs

Durante la fase di Object Design, il team ha dovuto prendere diverse decisioni architetturali che hanno comportato dei compromessi tra obiettivi contrastanti (es. flessibilità o semplicità, performance o manutenibilità). Le principali scelte effettuate sono:

- **Disaccoppiamento API vs Semplicità (DTO Pattern):** Si è scelto di adottare rigorosamente il pattern DTO (*Data Transfer Object*) per tutte le comunicazioni tra client e server, evitando di esporre direttamente le Entità JPA.
  - *Vantaggio:* Garantisce un disaccoppiamento totale tra lo schema del database e il contratto dell'API, permettendo di evolvere i due indipendentemente e di nascondere dati sensibili.
  - *Svantaggio:* Introduce la necessità di creare classi "specchio" e di implementare logiche di mapping (Mapper), aumentando la verbosità del codice e l'overhead di sviluppo.
- **Rich Domain Model vs Anemic Domain Model:** Si è optato per un approccio ibrido tendente all'Anemic Domain Model. La logica di business complessa risiede principalmente nei *Service*, mentre le Entità sono focalizzate sullo stato e contengono solo logica di validazione interna basilare.
  - *Vantaggio:* Si integra perfettamente con il modello di iniezione delle dipendenze di Spring (i Service possono chiamare altri Service/Repository, le Entità no) e semplifica la gestione delle transazioni.
  - *Svantaggio:* Riduce l'incapsulamento della logica di business all'interno degli oggetti del dominio, portando a servizi potenzialmente molto grandi.
- **Statelessness vs Session Management:** L'architettura del backend è stata progettata per essere completamente *stateless*. Lo stato della sessione utente non è mantenuto in memoria sul server, ma è delegato al client tramite token JWT.
  - *Vantaggio:* Massimizza la scalabilità orizzontale e semplifica l'architettura REST.
  - *Svantaggio:* Rende più complessa l'implementazione di funzionalità come la revoca immediata dei permessi o il logout forzato lato server.

## 1.2. Interface documentation guidelines

Per garantire l'uniformità e la manutenibilità del codice, il team di sviluppo adotta le seguenti linee guida per la definizione e la documentazione delle interfacce delle classi:

### Convenzioni di Naming

- **Classi e Interfacce:** `PascalCase` (es. `TransportRequest`). I nomi devono essere sostantivi singolari.
- **Metodi e Variabili:** `camelCase` (es. `calculateRoute`). I metodi devono essere verbi o frasi verbali che indicano chiaramente l'azione.
- **Costanti:** `UPPER_SNAKE_CASE` (es. `MAX_RETRY_ATTEMPTS`).
- **Suffissi Architetturali:** Ogni classe deve avere un suffisso che ne identifica il layer o il ruolo:
  - `...Controller`: Esposizione API REST.
  - `...Service`: Logica di business (Interfaccia).
  - `...ServiceImpl`: Implementazione della logica di business.
  - `...Repository`: Accesso ai dati.
  - `...DTO`: Oggetti di trasferimento dati.
  - `...Exception`: Classi di errore personalizzate.

### Gestione degli Errori

- **Eccezioni vs Codici di Ritorno:** I metodi non devono mai utilizzare codici di errore (es. -1, false) per segnalare fallimenti. Si devono utilizzare esclusivamente le **Eccezioni**.
- **Eccezioni Unchecked:** Si privilegia l'uso di eccezioni unchecked (estensioni di `RuntimeException`) per errori non recuperabili o violazioni di business logic, per non inquinare le firme dei metodi.
- **Global Exception Handling:** I Controller non devono contenere blocchi try-catch ripetitivi. Le eccezioni devono "risalire" ed essere gestite centralmente dal `GlobalExceptionHandler`, che le traduce in risposte HTTP standard.

### Design delle Interfacce

- **Programmazione per Interfacce:** I Controller e gli altri componenti devono dipendere dalle interfacce dei Service (`UserService`), mai dalle loro implementazioni concrete (`UserServiceImpl`), per favorire il disaccoppiamento e il testing.
- **Null Safety:** I metodi che restituiscono collezioni (es. `List`) non devono mai restituire `null`, ma una collezione vuota. I metodi che restituiscono un singolo oggetto opzionale devono utilizzare `java.util.Optional`.

## 1.3. Definitions, acronyms, and abbreviations

**DAO (Data Access Object)** Un pattern architetturale che fornisce un'interfaccia astratta per un qualche tipo di meccanismo di database o di persistenza. In Spring Data, questo ruolo è svolto dai **Repository**.

**DTO (Data Transfer Object)** Un oggetto che trasporta dati tra processi o layer dell'applicazione. Il suo unico scopo è contenere dati, senza logica di business.

**Entity (Entità)** Una classe leggera e persistente che mappa una tabella del database relazionale. In JPA, un'entità rappresenta una riga della tabella.

**JPA (Jakarta Persistence API)** Una specifica Java per la gestione dei dati relazionali nelle applicazioni Enterprise.

**Lazy Loading** Un design pattern di caricamento differito, in cui i dati correlati (es. la lista dei viaggi di un utente) vengono recuperati dal database solo quando sono esplicitamente richiesti.

**Mapper** Un componente o una classe responsabile della conversione dei dati tra oggetti di tipo diverso, tipicamente tra Entity e DTO.

**ORM (Object-Relational Mapping)** Una tecnica di programmazione per convertire dati tra sistemi di tipi incompatibili (oggetti Java e tabelle SQL).

**Service Layer** Il livello dell'applicazione che incapsula la logica di business e definisce i confini delle transazioni.

## 1.4. References

1. System Design Document (SDD) - HeavyRoute, Versione 1.1.

# 2. Packages

Questa sezione descrive la decomposizione dei sottosistemi identificati nel System Design in package Java concreti. L'organizzazione del codice segue una struttura modulare, dove ogni sottosistema funzionale è isolato nel proprio package di primo livello. All'interno di ogni modulo, il codice è ulteriormente organizzato per layer architetturale (Controller, Service, Repository, Model).
La radice del progetto è: `com.heavyroute`.

## 2.1. Organizzazione dei File e Gerarchia

La struttura delle directory del codice sorgente è organizzata come segue:

```
com.heavyroute
+-- auth                (Sottosistema Sicurezza)
|   +-- controller
|   +-- service
|   +-- dto
|   +-- security
+-- users               (Sottosistema Gestione Utenti)
|   +-- controller
|   +-- service
|   +-- repository
|   +-- dto
|   +-- mapper
|   +-- model
|   +-- enums
+-- core                (Sottosistema Core Business)
|   +-- controller
|   +-- service
|   +-- repository
|   +-- dto
|   +-- mapper
|   +-- model
|   +-- enums
+-- execution           (Sottosistema Esecuzione Mobile)
|   +-- controller
|   +-- service
|   +-- dto
+-- resources           (Sottosistema Risorse e Viabilità)
```

```
|   +-- controller
|   +-- service
|   +-- repository
|   +-- dto
|   +-- mapper
|   +-- model
|   +-- enums
+-- notification        (Sottosistema Notifiche)
|   +-- controller
|   +-- service
|   +-- repository
|   +-- dto
|   +-- mapper
|   +-- model
|   +-- enums
+-- common       (Shared Kernel)
    +-- controller
    +-- exception
    +-- model
    +-- config
```

## 2.2. Descrizione dei Package

**com.heavyroute.core** Contiene la logica centrale del dominio: gestione delle richieste, pianificazione dei viaggi e validazione. È il package più complesso e centrale del sistema.

**com.heavyroute.users** Incapsula tutto ciò che riguarda le anagrafiche degli attori. È responsabile della persistenza degli utenti e della gestione dei ruoli.

**com.heavyroute.auth** Gestisce i meccanismi di autenticazione. Contiene i filtri di sicurezza Spring Security, i provider di autenticazione e le utility per la generazione/validazione dei token JWT.

**com.heavyroute.resources** Gestisce l'inventario fisico (Veicoli) e i dati ambientali (Eventi di Viabilità). Agisce come un catalogo consultato dal Core.

**com.heavyroute.execution** Gestisce la macchina a stati per l'avanzamento operativo del viaggio e funge da gateway per l'applicazione mobile dell'autista.

`com.heavyroute.notification` Package infrastrutturale che astrae i dettagli di invio (SMTP, Push Notification Service) offrendo un'interfaccia unificata al resto del sistema.

`com.heavyroute.common` Contiene codice condiviso trasversalmente, come gli handler globali delle eccezioni (`GlobalExceptionHandler`), costanti di sistema e oggetti valore generici.

## 2.3. Dipendenze tra Package

Le dipendenze tra i package sono state progettate per evitare cicli e rispettare la gerarchia dei livelli.

- `core` dipende da:
  - `users`: Per associare i viaggi agli attori (Committente, Autista, PL).
  - `resources`: Per verificare la disponibilità dei veicoli durante la pianificazione.
  - `notification`: Per inviare aggiornamenti di stato.
- `execution` dipende da:
  - `core`: Per aggiornare lo stato delle entità `Trip` gestite nel core.
  - `notification`: Per segnalare imprevisti urgenti.
- `auth` dipende da:
  - `users`: Per recuperare le credenziali (hash password) e i ruoli durante il login.
- `common`: Non ha dipendenze verso altri moduli del dominio ed è utilizzato da tutti.

# 3. Class interfaces

Questa sezione documenta le interfacce pubbliche (API interne) delle classi principali per ciascun sottosistema. L'obiettivo è definire i contratti che permettono ai diversi moduli di interagire, specificando le firme dei metodi, i parametri di input/output e le eccezioni attese.

## 3.1. Package: `com.heavyroute.auth`

### 3.1.1. Class: `AuthController`

**Responsabilità:** Controller REST che gestisce il punto d'ingresso per le richieste di autenticazione.

- **Stereotype:** `«RestController»`
- **Operations:**
  - **`+ registerCustomer(registrationDTO: CustomerRegistrationDTO): ResponseEntity<JwtResponseDTO>`**
    Crea un nuovo committente.
    - **OCL Pre:** `registrationDTO <> null`
    - **OCL Pre:** `registrationDTO.email <> null AND registrationDTO.password.length >= 8`
    - **OCL Pre:** `!customerRepository.existsByEmail(registrationDTO.email)`
    - **OCL Post:** `result.statusCode == 200 IMPLIES result.body.token <> null`
    - **OCL Post:** `customerRepository.count() == customerRepository.count()@pre + 1`
    - **Exception:** `UserAlreadyExistsException` if `customerRepository.existsByEmail(registrationDTO.email)`
  - **`+ authenticateUser(loginRequest: LoginRequestDTO): ResponseEntity<JwtResponseDTO>`**
    Autentica l'utente e restituisce il token.
    - **OCL Pre:** `loginRequest <> null`
    - **OCL Pre:** `loginRequest.username <> null AND loginRequest.username.length > 0`
    - **OCL Pre:** `loginRequest.password <> null AND loginRequest.password.length > 0`

    - **OCL Post:** `result.statusCode == 200 IMPLIES result.body.token <> null`
    - **Exception:** `BadCredentialsException` if `!userRepository.exists (loginRequest.username, loginRequest.password)`

  - `+ forgotPassword(notification: NotificationDTO): ResponseEntity<Void>`
    Avvia il reset password.
    - **OCL Pre:** `notification <> null`
    - **OCL Pre:** `notification.email <> null AND notification.email.length > 0`
    - **OCL Post:** `result.statusCode == 200`
    - **Exception:** `UserNotFoundException` if `!userRepository. existsByEmail(notification.email)`

### 3.1.2. Interface: `AuthService`

**Responsabilità:** Facade per la logica di sicurezza.

- **Stereotype:** `«Service»`
- **Operations:**
  - `+ login(loginRequest: LoginRequestDTO): JwtResponseDTO`
    Verifica credenziali e genera JWT.
    - **OCL Pre:** `loginRequest <> null`
    - **OCL Pre:** `loginRequest.username <> null AND loginRequest.username.length > 0`
    - **OCL Pre:** `loginRequest.password <> null AND loginRequest.password.length > 0`
    - **OCL Post:** `result <> null AND result.token <> null`
    - **Exception:** `BadCredentialsException` if `!userRepository.exists (loginRequest.username, loginRequest.password)`
  - `+ registerCustomer(registrationDTO: CustomerRegistrationDTO): void`
    Registra un nuovo cliente nel sistema.
    - **OCL Pre:** `registrationDTO <> null`
    - **OCL Pre:** `registrationDTO.email <> null AND registrationDTO.email.length > 0`
    - **OCL Pre:** `registrationDTO.password <> null AND registrationDTO.password.length >= 8`

  - **OCL Post:** `customerRepository.count() == customerRepository.count()@pre + 1`
  - **Exception:** `UserAlreadyExistsException` if `customerRepository.existsByEmail(registrationDTO.email)`

- `+ resetPassword(token: String, newPassword: String): void`
  Cambia la password dell'utente.
  - **OCL Pre:** `token <> null AND token.length > 0`
  - **OCL Pre:** `newPassword <> null AND newPassword.length >= 8`
  - **OCL Post:** `tokenStore.getUser(token).password == newPassword`
  - **Exception:** `InvalidTokenException` if `!tokenStore.isValid(token)`

### 3.1.3. Interface: `UserDetailService`

**Responsabilità:** Recupera i dettagli dell'utente dal database per l'autenticazione.

- **Stereotype:** `«Service»`
- **Operations:**

  `+ loadUserByUsername(username: String): UserDetails`
  Carica i dettagli dell'utente per l'autenticazione.
  - **OCL Pre:** `username <> null AND username.length > 0`
  - **OCL Post:** `result <> null`
  - **OCL Post:** `result.username == username`
  - **Exception:** `UsernameNotFoundException` if `!userRepository.existsByUsername(username)`

### 3.1.4. DTO: `JwtResponseDTO`

**Responsabilità:** Transfer Object per la risposta di login.

- **Attributes:**
  - `private String token;`
  - `private String type = "Bearer";`
  - `private Long id;`
  - `private String username;`
  - `private String email;`
  - `private String role;`

- **Invariants (OCL):**
  - `context JwtResponseDTO inv: self.token <> null AND self.token.length > 0`
  - `context JwtResponseDTO inv: self.type == "Bearer"`
  - `context JwtResponseDTO inv: self.id <> null`
  - `context JwtResponseDTO inv: self.username <> null`
  - `context JwtResponseDTO inv: self.email <> null`
  - `context JwtResponseDTO inv: self.role <> null`

#### 3.1.5. DTO: `LoginRequestDTO`

**Responsabilità:** Transfer Object per le credenziali di accesso.

- **Attributes:**
  - `private String username;`
  - `private String password;`
- **Invariants (OCL):**
  - `context LoginRequestDTO inv: self.username <> null AND self.username.length > 0`
  - `context LoginRequestDTO inv: self.password <> null AND self.password.length > 0`

#### 3.1.6. Class: `JwtAuthenticationFilter`

**Responsabilità:** Intercetta le richieste HTTP per validare il token JWT e impostare il contesto di sicurezza.

- **Stereotype:** `«Security»`
- **Operations:**

  `# doFilterInternal(request: HttpServletRequest, response: HttpServletResponse filterChain: FilterChain): void`

  Logica core del filtro: estrae il token, lo valida e autentica l'utente nel contesto.
  - **OCL Pre:** `request <> null AND response <> null AND filterChain <> null`

- **OCL Post:** `let header = request.getHeader("Authorization") in (header <> null AND header.startsWith("Bearer ") AND jwtUtils.validate(header.substring(7))) IMPLIES SecurityContextHolder.getContext().getAuthentication() <> null`
- **Exception:** `ServletException, IOException` if processing fails.

### 3.1.7. Class: `JwtUtils`

**Responsabilità:** Componente responsabile delle operazioni crittografiche sui token JWT (creazione, parsing e validazione).

- **Stereotype:** `«Security»`
- **Attributes:**
  - `private static final String JWT_SECRET;`
  - `private static final int JWT_EXPIRATION_MS = 86400000;`
- **Operations:**

`+ generateJwtToken(authentication: Authentication): String`

Genera un nuovo JWT per un utente autenticato.

- **OCL Pre:** `authentication <> null`
- **OCL Post:** `result <> null AND result.length > 0`

`+ getUserNameFromJwtToken(token: String): String`

Estrae lo username dal token.

- **OCL Pre:** `token <> null AND token.length > 0`
- **OCL Post:** `result <> null`

`+ validateJwtToken(authToken: String): boolean`

Verifica se il token è valido (firma corretta, non scaduto, formato valido).

- **OCL Pre:** `authToken <> null`
- **OCL Post:** `result == true IMPLIES (isValidSignature(authToken) AND NOT isExpired(authToken))`

### 3.1.8. Class: `SecurityConfig`

**Responsabilità:** Configurazione centrale della sicurezza (Autenticazione, Autorizzazione, CORS). Agisce come "Torre di Controllo" per l'applicazione.

- **Stereotype:** `«Security»`
- **Attributes:**
  - `private final UserDetailsServiceImpl userDetailsService;`
  - `private final JwtAuthenticationFilter jwtAuthenticationFilter;`
- **Operations:**

  **`+ authenticationProvider(): DaoAuthenticationProvider`**

  Configura il provider di autenticazione collegando UserDetailsService e PasswordEncoder.
  - **OCL Post:** `result <> null`
  - **OCL Post:** `result.userDetailsService == self.userDetailsService`

  **`+ authenticationManager(authConfig: AuthenticationConfiguration): AuthenticationManager`**

  Espone il gestore delle autenticazioni per il Controller di Login.
  - **OCL Pre:** `authConfig <> null`
  - **OCL Post:** `result <> null`
  - **Exception:** `Exception` if configuration fails.

  **`+ passwordEncoder(): PasswordEncoder`**

  Definisce l'algoritmo di hashing delle password (BCrypt).
  - **OCL Post:** `result <> null AND result.oclIsKindOf (BCryptPasswordEncoder)`

  **`+ filterChain(http: HttpSecurity): SecurityFilterChain`**

  Definisce la catena dei filtri di sicurezza, le regole CORS, la disabilitazione CSRF e la sessione Stateless.
  - **OCL Pre:** `http <> null`
  - **OCL Post:** `result <> null`
  - **Exception:** `Exception` if configuration fails.

  **`+ corsConfigurationSource(): CorsConfigurationSource`**

  Configura programmaticamente le regole CORS (Cross-Origin Resource Sharing).
  - **OCL Post:** `result <> null`

## 3.2. Package: `com.heavyroute.users`

### 3.2.1. Class: `UserController`

**Responsabilità:** Controller REST per la gestione anagrafica degli utenti.

- **Stereotype:** `«RestController»`
- **Operations:**

`+ getCurrentUser(): ResponseEntity<UserResponseDTO>`

Restituisce il profilo dell'utente attualmente loggato (basato sul token JWT).

- **OCL Pre:** `SecurityContextHolder.getContext().getAuthentication() <> null`
- **OCL Post:** `result.statusCode == 200 IMPLIES result.body.username == currentUser.username`
- **Exception:** `ResourceNotFoundException` if `!userService.existsByUsername(currentUser.username)`

`+ getPending(): ResponseEntity<List<UserResponseDTO»`

Restituisce la lista degli utenti in attesa di approvazione.

- **OCL Pre (Security):** `currentUser.role in {LOGISTIC_PLANNER, ACCOUNT_MANAGER}`
- **OCL Post:** `result.body->forAll(u | u.active == false)`

`+ approve(id: Long): ResponseEntity<UserResponseDTO>`

Approva e attiva un utente registrato.

- **OCL Pre (Security):** `currentUser.role in {LOGISTIC_PLANNER, ACCOUNT_MANAGER}`
- **OCL Pre:** `id <> null`
- **OCL Post:** `result.statusCode == 200 IMPLIES result.body.active == true`

`+ registerClient(dto: CustomerRegistrationDTO): ResponseEntity<UserResponseDTO>`

Registrazione pubblica self-service per un nuovo committente.

- **OCL Pre:** `dto <> null`
- **OCL Post:** `result.statusCode == 200 IMPLIES result.body.role == Role.CUSTOMER`

`+ createInternalUser(dto: InternalUserCreateDTO): ResponseEntity<UserResponseDTO>`

Creazione di un utente staff interno.

- **OCL Pre (Security):** `currentUser.role == Role.ACCOUNT_MANAGER`
- **OCL Pre:** `dto <> null`
- **OCL Post:** `userService.count() == userService.count()@pre + 1`

`+ getInternalUsers(): ResponseEntity<List<UserResponseDTO>>`

Ottiene la lista di tutti gli utenti interni (staff).

- **OCL Pre (Security):** `currentUser.role == Role.ACCOUNT_MANAGER`
- **OCL Post:** `result.statusCode == 200`

`+ updateInternalUser(id: Long, dto: InternalUserUpdateDTO): ResponseEntity<UserResponseDTO>`

Modifica i dati di un utente interno esistente.

- **OCL Pre (Security):** `currentUser.role == Role.ACCOUNT_MANAGER`
- **OCL Pre:** `id <> null AND dto <> null`
- **OCL Post:** `result.statusCode == 200 IMPLIES result.body.id == id`

### 3.2.2. Class: `DriverController`

**Responsabilità:** Controller REST per la gestione specifica degli autisti.

- **Stereotype:** `«RestController»`
- **Operations:**

`+ getAvailableDrivers(): ResponseEntity<List<UserResponseDTO>>`

Recupera la lista degli autisti attualmente disponibili per essere assegnati a un viaggio.

- **OCL Pre (Security):** `currentUser.role == Role.LOGISTIC_PLANNER`
- **OCL Post:** `result.statusCode == 200`
- **OCL Post:** `result.body->forAll(driver | driver.role == Role.DRIVER)`

### 3.2.3. Class: `TrafficCoordinatorController`

**Responsabilità:** Controller REST per le operazioni specifiche del Coordinatore del Traffico.

- **Stereotype:** `«RestController»`

- **Operations:**

  `+ getRoutesToValidate(): ResponseEntity<List<RouteResponseDTO>>`

  Recupera la lista dei percorsi pianificati che richiedono validazione tecnica.

  - **OCL Pre (Security):** `currentUser.role == Role.TRAFFIC_COORDINATOR`
  - **OCL Post:** `result.statusCode == 200`
  - **OCL Post:** `result.body->forAll(r | r.requiresValidation == true)`

  `+ validateRoute(id: Long, approved: boolean): ResponseEntity<Void>`

  Approva o rifiuta un percorso specifico.

  - **OCL Pre (Security):** `currentUser.role == Role.TRAFFIC_COORDINATOR`
  - **OCL Pre:** `id <> null`
  - **OCL Post:** `result.statusCode == 200`
  - **Exception:** `ResourceNotFoundException` if `!routeService.existsById(id)`

### 3.2.4. Interface: `UserService`

**Responsabilità:** Logica di business per la gestione del ciclo di vita degli utenti.

- **Stereotype:** `«Service»`
- **Attributes:**
  - `private final UserRepository userRepository;`
  - `private final CustomerRepository customerRepository;`
  - `private final PasswordEncoder passwordEncoder;`
  - `private final UserMapper userMapper;`
- **Operations:**

  `+ registerNewClient(dto: CustomerRegistrationDTO): UserResponseDTO`

  Gestisce il processo di auto-registrazione per nuovi committenti.

  - **OCL Pre:** `dto <> null`
  - **OCL Pre:** `!userRepository.existsByUsername(dto.username)`
  - **OCL Pre:** `!userRepository.existsByEmail(dto.email)`
  - **OCL Pre:** `!customerRepository.existsByVatNumber(dto.vatNumber)`

- **OCL Post:** `result.active == false` (In attesa di approvazione)
- **Exception:** `UserAlreadyExistException` if duplication found.

**`+ createInternalUser(dto: InternalUserCreateDTO): UserResponseDTO`**

Crea un utente dello staff interno (Back-office) attivo di default.

- **OCL Pre:** `!userRepository.existsByEmail(dto.email)`
- **OCL Pre:** `dto.role in {DRIVER, LOGISTIC_PLANNER, ACCOUNT_MANAGER, TRAFFIC_COORDINATOR}`
- **OCL Post:** `result.active == true`
- **Exception:** `BusinessRuleException` if role is invalid.

**`+ updateInternalUser(id: Long, dto: InternalUserUpdateDTO): UserResponseDTO`**

Aggiorna i dati di un utente interno.

- **OCL Pre:** `userRepository.existsById(id)`
- **OCL Post:** `result.id == id`

**`+ updateCustomer(id: Long, dto: CustomerUpdateDTO): UserResponseDTO`**

Aggiorna i dati di un committente.

- **OCL Pre:** `customerRepository.existsById(id)`
- **OCL Post:** `result.id == id`

**`+ findInactiveUsers(): List<UserResponseDTO>`**

Recupera la lista di tutti gli utenti in attesa di approvazione manuale (active = false).

- **OCL Post:** `result->forAll(u | u.active == false)`

**`+ activateUser(id: Long): UserResponseDTO`**

Attiva un utente specifico (cambia stato active da false a true).

- **OCL Pre:** `userRepository.existsById(id)`
- **OCL Post:** `userRepository.findById(id).active == true`

**`+ deactivateUser(id: Long): void`**

Esegue una cancellazione logica (Soft Delete), impostando active a false.

- **OCL Pre:** `userRepository.existsById(id)`
- **OCL Post:** `userRepository.findById(id).active == false`

**`+ deleteUser(id: Long): void`**

Elimina fisicamente un utente dal database (Hard Delete).

- **OCL Pre:** `userRepository.existsById(id)`

  - **OCL Post:** `!userRepository.existsById(id)`

`+ findAllInternalUsers(): List<User>`

Recupera tutti gli utenti dello staff (escludendo i Customer).

  - **OCL Post:** `result->forAll(u | u.role <> Role.CUSTOMER)`

### 3.2.5. Interface: `DriverService`

**Responsabilità:** Implementazione del servizio per la gestione operativa degli autisti.

- **Stereotype:** `«Service»`
- **Attributes:**
  - `private final DriverRepository driverRepository;`
  - `private final UserMapper userMapper;`
- **Operations:**

`+ findAvailableDrivers(): List<UserResponseDTO>`

Recupera la lista di tutti gli autisti attualmente disponibili (Stato FREE) per l'assegnazione.

  - **OCL Post:** `result <> null`
  - **OCL Post:** `result->size() == driverRepository.findByDriverStatus (DriverStatus.FREE)->size()`

`+ findByDriverStatus(status: DriverStatus): List<Driver>`

Recupera gli autisti filtrati per uno specifico stato operativo.

  - **OCL Pre:** `status <> null`
  - **OCL Post:** `result->forAll(d | d.driverStatus == status)`

### 3.2.6. Interface: `TrafficCoordinatorService`

**Responsabilità:** Implementazione della logica di business per il Coordinatore del Traffico.

- **Stereotype:** `«Service»`
- **Attributes:**
  - `private final RouteRepository routeRepository;`
  - `private final TripRepository tripRepository;`
- **Operations:**

`+ getRoutesToValidate(): List<RouteResponseDTO>`

Recupera i percorsi associati a viaggi in stato di attesa validazione e li converte in DTO per la visualizzazione.

- **OCL Post:** `result <> null`
- **OCL Post:** `result->forAll(r | r.status == TripStatus.WAITING_VALIDATION)`

`+ validateRoute(routeId: Long, approved: boolean): void`

Applica la decisione del coordinatore (Approva/Rifiuta) al viaggio associato al percorso specificato.

- **OCL Pre:** `routeId <> null`
- **OCL Post:** `approved == true IMPLIES route.trip.status == TripStatus.VALIDATED`
- **OCL Post:** `approved == false IMPLIES route.trip.status == TripStatus.MODIFICATION_REQUESTED`
- **Exception:** `RuntimeException` if `!routeRepository.existsById(routeId)`

### 3.2.7. Interface: `UserRepository`

**Responsabilità:** Gestione della persistenza polimorfica per l'entità radice `User`.

- **Stereotype:** `«Repository»`
- **Operations:**

`+ findByUsername(username: String): Optional<User>`

Recupera un utente dato il suo username, ignorando lo stato di attivazione.

- **OCL Pre:** `username <> null AND username.length > 0`
- **OCL Post:** `result <> null`

`+ existsByUsername(username: String): boolean`

Esegue un controllo rapido per verificare se uno username è già occupato.

- **OCL Pre:** `username <> null AND username.length > 0`

`+ existsByEmail(email: String): boolean`

Verifica se un indirizzo email è già associato a un account esistente.

- **OCL Pre:** `email <> null AND email.length > 0`

`+ findByUsernameAndActiveTrue(username: String): Optional<User>`

Restituisce l'utente solo se le credenziali corrispondono e l'account è attivo.

- **OCL Pre:** `username <> null`
- **OCL Post:** `result.isPresent() IMPLIES result.get().active == true`

`+ findByUsernameAndActiveFalse(username: String): Optional<User>`

Recupera un utente specifico che si trova in stato disabilitato o in attesa di approvazione.

- **OCL Pre:** `username <> null`
- **OCL Post:** `result.isPresent() IMPLIES result.get().active == false`

`+ findByActiveFalse(): List<User>`

Recupera la lista di tutti gli utenti che hanno il flag active impostato a false.

- **OCL Post:** `result <> null`
- **OCL Post:** `result->forAll(u | u.active == false)`

### 3.2.8. Interface: `CustomerRepository`

**Responsabilità:** Gestione della persistenza specifica per l'entità `Customer`, inclusa la validazione dei dati fiscali e la gestione delle approvazioni.

- **Stereotype:** `«Repository»`
- **Operations:**

`+ findByActiveFalse(): List<Customer>`

Restituisce la lista di tutti i committenti che sono attualmente in stato "NON ATTIVO", in attesa di approvazione.

- **OCL Post:** `result <> null`
- **OCL Post:** `result->forAll(c | c.active == false)`

`+ existsByVatNumber(vatNumber: String): boolean`

Verifica se una Partita IVA è già presente nel sistema, garantendo l'unicità fiscale.

- **OCL Pre:** `vatNumber <> null AND vatNumber.length > 0`

`+ existsByCompanyName(companyName: String): boolean`

Verifica se una Ragione Sociale è già registrata nel sistema per evitare omonimie.

- **OCL Pre:** `companyName <> null AND companyName.length > 0`

### 3.2.9. Interface: `DriverRepository`

**Responsabilità:** Gestione della persistenza e del recupero delle entità `Driver`. Estende l'interfaccia `JpaRepository`.

- **Stereotype:** `«Repository»`
- **Genaralization:** `JpaRepository<Driver, Long>`
- **Operations:**

  **`+ findAllByDriverStatus(status: DriverStatus): List<Driver>`**
  Restituisce la lista dei conducenti filtrata per uno specifico stato operativo.

  **`+ findAvailableDrivers(): List<Driver>`**
  *(Default Method)* Restituisce la lista dei conducenti che non sono attualmente assegnati ad alcun veicolo (stato `FREE`).

  **`+ findDriversOnTheRoad(): List<Driver>`**
  *(Default Method)* Restituisce la lista dei conducenti che sono attualmente in servizio attivo (stato `ON_THE_ROAD`).

  **`+ findAvailableDriversInArea(minLat: Double, maxLat: Double, minLon: Double, maxLon: Double): List<Driver>`**
  Query JPQL personalizzata che restituisce la lista dei conducenti disponibili (`FREE`) la cui posizione rientra nel rettangolo di delimitazione (*bounding box*) definito dai parametri.

  **`+ findByLicenseNumber(licenseNumber: String): Optional<Driver>`**
  Restituisce un conducente univoco basandosi sul numero di patente fornito.

- **Constraints (OCL):**
  - `context DriverRepository::findByLicenseNumber(licenseNum) pre: licenseNum <> null`
  - `context DriverRepository::findByLicenseNumber(licenseNum) pre: licenseNum.size() > 0`
  - `context DriverRepository::save(driver) pre: driver <> null`
  - `context DriverRepository::save(driver) pre: driver.licenseNumber <> null`

- `context DriverRepository::findAvailableDriversInArea(minLat, maxLat, minLon, maxLon) pre: minLat <> null and maxLat <> null`
- `context DriverRepository::findAvailableDriversInArea(minLat, maxLat, minLon, maxLon) pre: minLon <> null and maxLon <> null`
- `context DriverRepository::findAvailableDriversInArea(minLat, maxLat, minLon, maxLon) pre: minLat <= maxLat`
- `context DriverRepository::findAvailableDriversInArea(minLat, maxLat, minLon, maxLon) pre: minLon <= maxLon`
- `context DriverRepository::findAvailableDriversInArea(minLat, maxLat, minLon, maxLon) pre: minLat >= -90.0 and maxLat <= 90.0`
- `context DriverRepository::findAvailableDriversInArea(minLat, maxLat, minLon, maxLon) pre: minLon >= -180.0 and maxLon <= 180.0`

### 3.2.10. DTO: `CustomerRegistrationDTO`

**Responsabilità:** Data Transfer Object utilizzato per il processo di onboarding (registrazione) di un nuovo cliente aziendale. Raccoglie dati utente e dati fiscali.

- **Stereotype:** `«DTO»`
- **Attributes:**
  - `username: String`
    *Constraints:* `@NotBlank`, `@Size(min=4)`
  - `password: String`
    *Constraints:* `@NotBlank`, `@Size(min=8)`
  - `email: String`
    *Constraints:* `@NotBlank`, `@Email`, `@Pattern(email_regex)`
  - `companyName: String`
    *Constraints:* `@NotBlank`
  - `vatNumber: String`
    *Constraints:* `@NotBlank`, `@Size(min=11, max=11)`, `@Pattern(digits_only)`
  - `address: String`
    *Constraints:* `@NotBlank`
  - `pec: String`
    *Constraints:* `@NotBlank`, `@Email`
  - `phoneNumber: String`
    *Constraints:* `@NotBlank`, `@Pattern(phone_regex)`

  - `firstName: String`
    *Constraints:* `@NotBlank`
  - `lastName: String`
    *Constraints:* `@NotBlank`

#### 3.2.11. DTO: `CustomerUpdateDTO`

**Responsabilità:** Data Transfer Object per l'aggiornamento parziale dei dati del profilo cliente. I campi nulli vengono ignorati.

- **Stereotype:** `«DTO»`
- **Attributes:**
  - `id: Long`
    *Constraints:* `@NotNull` (Necessario per check di sicurezza)
  - `firstName: String` *(Optional)*
  - `lastName: String` *(Optional)*
  - `email: String`
    *Constraints:* `@Email`, `@Pattern(email_regex)`
  - `companyName: String` *(Optional)*
  - `vatNumber: String`
    *Constraints:* `@Size(min=11, max=16)`
  - `address: String` *(Optional)*
  - `phoneNumber: String`
    *Constraints:* `@Pattern(phone_regex)`
  - `password: String`
    *Constraints:* `@Size(min=8)` (Opzionale, solo per reset password)

#### 3.2.12. DTO: `InternalUserCreateDTO`

**Responsabilità:** Data Transfer Object utilizzato dagli Account Manager per la creazione e l'onboarding di nuovo personale interno (Staff). Tutti i campi sono obbligatori.

- **Stereotype:** `«DTO»`
- **Attributes:**
  - `username: String`
    *Constraints:* `@NotBlank`, `@Size(min=4)`

- `email: String`
  *Constraints:* `@NotBlank`, `@Email`, `@Pattern(company_mail_regex)`
- `firstName: String`
  *Constraints:* `@NotBlank`
- `lastName: String`
  *Constraints:* `@NotBlank`
- `role: UserRole`
  *Constraints:* `@NotNull` (Definisce i permessi operativi)
- `password: String`
  *Constraints:* `@NotBlank`, `@Size(min=8)` (Password temporanea obbligatoria)

### 3.2.13. DTO: `InternalUserUpdateDTO`

**Responsabilità:** Data Transfer Object per la modifica dei profili dello staff esistente. Supporta la modifica dei dati anagrafici, la gestione dello stato di attivazione e il reset amministrativo della password.

- **Stereotype:** `«DTO»`
- **Attributes:**
  - `id: Long`
    *Constraints:* `@NotNull` (Richiesto per consistency check)
  - `username: String`
    *Constraints:* `@NotBlank`
  - `email: String`
    *Constraints:* `@NotBlank`, `@Email`, `@Pattern`
  - `role: UserRole`
    *Constraints:* `@NotNull`
  - `firstName: String`
    *Constraints:* `@NotBlank`
  - `lastName: String`
    *Constraints:* `@NotBlank`
  - `phoneNumber: String` *(Optional)*
  - `active: Boolean` *(Optional)*
    Permette di attivare/disattivare l'account dipendente.
  - `password: String` *(Optional)*
    Se valorizzata, sovrascrive la password attuale (Reset). Se `null`, la password resta invariata.

### 3.2.14. DTO: `UserResponseDTO`

**Responsabilità:** Data Transfer Object unificato utilizzato per restituire i dettagli di un utente al client. Aggrega i dati di tutte le possibili sottoclassi (Customer, Driver, Staff) in un'unica struttura piatta; i campi non pertinenti al ruolo specifico saranno restituiti come `null`.

- **Stereotype:** `«DTO»`
- **Attributes:**
  - **Common Fields:** (Presenti per tutti gli utenti)
    - `id: Long`
    - `username: String`
    - `email: String`
    - `firstName: String`
    - `lastName: String`
    - `phoneNumber: String`
    - `role: UserRole`
    - `active: boolean`
  - **Internal Staff Fields:** (Valorizzati solo per Staff e Driver)
    - `serialNumber: String`
    - `hireDate: LocalDate`
  - **Driver Specific Fields:** (Valorizzati solo se role = DRIVER)
    - `licenseNumber: String`
    - `driverStatus: DriverStatus`
  - **Customer Specific Fields:** (Valorizzati solo se role = CUSTOMER)
    - `companyName: String`
    - `vatNumber: String`
    - `pec: String`
    - `address: String`

### 3.2.15. Class: `UserMapper`

**Responsabilità:** Componente stateless responsabile della trasformazione dei dati (Object Mapping) tra il modello di dominio (Entities) e gli oggetti di trasporto (DTO).

- **Stereotype:** `«Mapper»`

- **Mapping Strategy:** *Unified DTO / Flattening.* Converte la gerarchia polimorfica delle entità in una struttura dati piatta per il frontend.
- **Operations:**

  `+ toDTO(entity: User): UserResponseDTO`

  Converte un'entità `User` (o le sue sottoclassi) nel DTO di risposta unificato. Esegue il *Type Checking* a runtime per identificare il ruolo specifico (Driver, Customer, Staff) e popolare i campi pertinenti (es. patente per i driver, partita IVA per i clienti).

  `+ toDTOList(users: List<User>): List<UserResponseDTO>`

  Operazione batch che converte una lista di entità in una lista di DTO utilizzando il metodo `toDTO`.

  `+ updateInternalUserFromDTO(dto: InternalUserUpdateDTO, entity: User): void`

  Aggiorna i dati di un membro dello staff. Implementa una logica *Null-Safe*: modifica i campi dell'entità solo se i valori corrispondenti nel DTO non sono nulli.

  `+ updateCustomerFromDTO(dto: CustomerUpdateDTO, entity: Customer): void`

  Aggiorna i dati di un cliente aziendale. Gestisce l'aggiornamento simultaneo dei dati anagrafici personali (User) e dei dati fiscali (Customer).

### 3.2.16. Entity: `User`

**Responsabilità:** Classe astratta che rappresenta un utente generico del sistema. Utilizza una strategia di ereditarietà di tipo *JOINED* per distinguere tra i vari attori (es. Driver, Customer).

- **Stereotype:** `«Entity», {abstract}`
- **Generalization:** `BaseEntity`
- **Attributes:**

  `- username: String`

  `- email: String`

  `- password: String`

  `- firstName: String`

  `- lastName: String`

- `- phoneNumber: String`
- `- active: boolean`

- **Operations:**

  `+ hasRole(r: UserRole): boolean`
  Verifica se l'utente possiede il ruolo specificato. Utile per controlli di sicurezza e login.

  `+ getRole(): UserRole {abstract}`
  Metodo astratto polimorfico che costringe le sottoclassi a definire il proprio ruolo specifico nel sistema.

- **Invariants (OCL):**
  - `context User inv: self.username.length() >= 4`
  - `context User inv: self.email.matches("^.+@.+..+$")`
  - `context User inv: self.getRole() <> null`

### 3.2.17. Entity: `InternalUser`

**Responsabilità:** Sottoclasse astratta intermedia che raggruppa attributi e logiche comuni a tutto il personale interno (Staff) del sistema.

- **Stereotype:** `«Entity», {abstract}`
- **Generalization:** `User`
- **Attributes:**
  - `- serialNumber: String`
  - `- hireDate: LocalDate`
- **Invariants (OCL):**
  - `context InternalUser inv: self.serialNumber.length() >= 2`
  - `context InternalUser inv: self.hireDate <= LocalDate.now()`

### 3.2.18. Entity: `Customer`

**Responsabilità:** Entità che rappresenta il committente (Cliente Esterno/Azienda). Eredita le generalità di base dall'entità User.

- **Stereotype:** `«Entity»`

- **Generalization:** `User`
- **Attributes:**
  - `- companyName: String`
  - `- vatNumber: String`
  - `- pec: String`
  - `- address: String`
- **Operations:**

  `+ getRole(): UserRole`

  Restituisce il valore enumerativo `UserRole.CUSTOMER` per identificare i permessi dell'utente.
- **Invariants (OCL):**
  - `context Customer inv: self.vatNumber.length() = 11`
  - `context Customer inv: self.getRole() = UserRole.CUSTOMER`

### 3.2.19. Entity: `Driver`

**Responsabilità:** Sottoclasse concreta che rappresenta l'autista (personale viaggiante). Eredita le caratteristiche del personale interno.

- **Stereotype:** `«Entity»`
- **Generalization:** `InternalUser`
- **Attributes:**
  - `- licenseNumber: String`
  - `- geoLocation: GeoLocation`
  - `- driverStatus: DriverStatus`
  - `- vehicle: Vehicle`
- **Operations:**

  `+ isFree(): boolean`

  Restituisce `true` se l'autista è nello stato `FREE`, altrimenti `false`.

  `+ isOnTheRoad(): boolean`

  Restituisce `true` se l'autista sta attualmente effettuando un viaggio (`ON_THE_ROAD`).

`+ getRole(): UserRole`

Restituisce il valore enumerativo `UserRole.DRIVER`.

- **Invariants (OCL):**
  - `context Driver inv: self.licenseNumber <> null`
  - `context Driver inv: self.getRole() = UserRole.DRIVER`

### 3.2.20. Entity: `AccountManager`

**Responsabilità:** Operatore interno responsabile della gestione amministrativa degli utenti interni (creazione account, gestione accessi).

- **Stereotype:** `«Entity»`
- **Generalization:** `InternalUser`
- **Operations:**

  `+ getRole(): UserRole`

  Restituisce il valore enumerativo `UserRole.ACCOUNT_MANAGER`.

- **Invariants (OCL):**
  - `context AccountManager inv: self.getRole() = UserRole.ACCOUNT_MANAGER`

### 3.2.21. Entity: `LogisticPlanner`

**Responsabilità:** Operatore interno addetto alla pianificazione logistica dei viaggi e all'assegnazione delle risorse (autisti e veicoli) alle richieste.

- **Stereotype:** `«Entity»`
- **Generalization:** `InternalUser`
- **Operations:**

  `+ getRole(): UserRole`

  Restituisce il valore enumerativo `UserRole.LOGISTIC_PLANNER`.

- **Invariants (OCL):**
  - `context LogisticPlanner inv: self.getRole() = UserRole.LOGISTIC_PLANNER`

#### 3.2.22. Entity: `TrafficCoordinator`

**Responsabilità:** Membro dello staff tecnico addetto alla validazione dei percorsi, verifica della fattibilità dei transiti e gestione dei permessi di viabilità.

- **Stereotype:** `«Entity»`
- **Generalization:** `InternalUser`
- **Operations:**

  `+ getRole(): UserRole`
  Restituisce il valore enumerativo `UserRole.TRAFFIC_COORDINATOR`.

- **Invariants (OCL):**
  - `context TrafficCoordinator inv: self.getRole() = UserRole.TRAFFIC_COORDINATOR`

#### 3.2.23. Enum: `UserRole`

Definizione enumerata dei ruoli di sistema per il controllo degli accessi (RBAC).

- **Values:** `CUSTOMER, LOGISTIC_PLANNER, TRAFFIC_COORDINATOR, DRIVER, ACCOUNT_MANAGER`

#### 3.2.24. Enum: `DriverStatus`

Definizione enumerata dei vari stati operativi in cui può trovarsi un autista.

- **Values:** `FREE, ASSIGNED, ON_THE_ROAD, RESTING`

### 3.3. Package: `com.heavyroute.core`

#### 3.3.1. Class: `TransportRequestController`

**Responsabilità:** Controller REST per la gestione del ciclo di vita delle richieste di trasporto. Espone endpoint differenziati per i Committenti (inserimento e storico personale) e per i Pianificatori Logistici (visione globale).

- **Stereotype:** `«RestController»`
- **Operations:**

  `+ createRequest(dto: RequestCreationDTO): ResponseEntity<TransportRequestResponseDTO>`
  Permette ad un cliente autenticato di inserire una nuova richiesta. Il sistema recupera automaticamente l'identità dell'utente dal contesto di sicurezza (JWT).

- **Security:** Richiede ruolo `CUSTOMER`.
- **OCL Pre:** `dto.pickupDate > LocalDate.now()`
- **OCL Post:** `result.status == RequestStatus.PENDING`

**`+ getMyRequest(): ResponseEntity<List<TransportRequestResponseDTO»`**

Restituisce lo storico delle richieste effettuate esclusivamente dall'utente loggato.

- **Security:** Richiede ruolo `CUSTOMER`.

**`+ getAllRequests(): ResponseEntity<List<TransportRequestResponseDTO»`**

Restituisce la lista globale di tutte le richieste presenti nel sistema, necessaria per le operazioni di pianificazione dei viaggi.

- **Security:** Richiede ruolo `LOGISTIC_PLANNER`.

### 3.3.2. Class: `TripManagementController`

**Responsabilità:** Controller REST centrale per la gestione operativa ("Dispatching"). Orchestrano il flusso di lavoro che trasforma una richiesta approvata in un viaggio assegnato e monitorato.

- **Stereotype:** `«RestController»`
- **Operations:**

**`+ getTrips(status: TripStatus[]): ResponseEntity<List<TripDTO»`**

Restituisce la lista di tutti i viaggi, filtrabile opzionalmente per stato.

- **Security:** `hasAnyRole(PL, TC)`

**`+ getTripsToPlan(): ResponseEntity<List<TripDTO»`**

Restituisce la "Worklist" del pianificatore: solo i viaggi in stato `IN_PLANNING` che richiedono assegnazione risorse.

- **Security:** `hasRole(PL)`

**`+ getDriverTrips(driverId: Long): ResponseEntity<List<TripDTO»`**

Endpoint Mobile. Restituisce le assegnazioni specifiche per un autista.

- **Security:** `hasAnyRole(DRIVER, PL)`

**`+ approve(requestId: Long): ResponseEntity<TripDTO>`**

Approva una richiesta di trasporto e genera l'entità `Trip` associata, avviando il calcolo della rotta.

- **Security:** `hasRole(PL)`
- **Post-Condition:** Crea un nuovo Trip in stato `IN_PLANNING`.

`+ planResources(tripId: Long, dto: TripAssignmentDTO): void`

Assegna le risorse fisiche (Autista e Veicolo) al viaggio.

- **Security:** `hasRole(PL)`
- **OCL Pre:** `dto.driverId <> null AND dto.vehiclePlate <> null`

`+ validateRoute(tripId: Long, req: RouteValidationRequest): void`

Permette al Traffic Coordinator di validare o rifiutare il percorso calcolato dall'algoritmo.

- **Security:** `hasRole(TC)`

`+ updateTripStatus(tripId: Long, status: String): void`

Endpoint Mobile. Permette all'autista di segnalare l'avanzamento (es. `IN_TRANSIT`, `DELIVERED`).

- **Security:** `hasAnyRole(DRIVER, PL)`

### 3.3.3. Interface: `TripService`

**Responsabilità:** Facade della logica di business core ("Dispatching Engine"). Orchestra l'intero ciclo di vita del viaggio, dalla generazione automatica post-approvazione richiesta, passando per la pianificazione e validazione, fino al completamento e rilascio delle risorse.

- **Stereotype:** `«Service»`
- **Operations:**

`+ approveRequest(requestId: Long): TripResponseDTO`

Approva una richiesta di trasporto e avvia il processo di generazione del viaggio.

1. Verifica l'idempotenza (restituisce il viaggio esistente se già creato).
2. Invoca `ExternalMapService` per calcolare la rotta iniziale.
3. Crea l'entità `Trip` in stato `IN_PLANNING`.

- **OCL Pre:** `requestRepository.existsById(requestId)`
- **Post:** `result.status == TripStatus.IN_PLANNING`

`+ planTrip(tripId: Long, dto: TripAssignmentDTO): void`

Gestisce l'assegnazione o ri-assegnazione delle risorse (Autista e Veicolo).

- **Logic:** Se erano presenti risorse precedenti, le rilascia (Stato FREE/AVAILABLE).
- **OCL Pre:** `driver.status == FREE`

- **OCL Pre:** `vehicle.status == AVAILABLE`
- **OCL Pre:** `vehicle.capacity >= trip.request.load.weight`
- **Post:** `trip.status == WAITING_VALIDATION`
- **Side Effect:** Invia notifica `ASSIGNMENT` all'autista.

**`+ validateRoute(tripId: Long, approved: boolean, feedback: String): void`**

Registra l'esito della validazione tecnica della rotta da parte del Traffic Coordinator.

- **Post (Approved):** `trip.status == CONFIRMED`, `request.status == PLANNED`

**`+ updateStatus(tripId: Long, newStatus: String): void`**

Gestisce le transizioni di stato operative segnalate dall'app mobile. Include logica di rilascio risorse al completamento.

- **Post (Completed):** `trip.status == COMPLETED`, `driver.status == FREE`, `vehicle.status == AVAILABLE`

**`+ calculateRoute(tripId: Long): void`**

Forza un ricalcolo della rotta tramite il servizio esterno. Utile in caso di modifiche agli indirizzi.

### 3.3.4. Interface: `TransportRequestService`

**Responsabilità:** Logica di business per la gestione del ciclo di vita delle richieste. Agisce da ponte tra il Controller e il Repository, gestendo la validazione, l'assegnazione dello stato iniziale e la trasformazione dei dati (DTO/Entity).

- **Stereotype:** `«Service»`
- **Operations:**

**`+ createRequest(dto: RequestCreationDTO, username: String): TransportRequestResponseDTO`**

Gestisce l'inserimento di una nuova richiesta nel sistema.

1. Recupera l'entità `User` associata allo username.
2. Mappa i dati tecnici del carico (LoadDetails).
3. Imposta lo stato iniziale a `PENDING`.
4. Persiste la richiesta.

- **OCL Pre:** `userRepository.existsByUsername(username)`
- **OCL Post:** `result.status == RequestStatus.PENDING`

`+ getAllRequests(): List<TransportRequestResponseDTO>`

Recupera l'elenco completo delle richieste presenti nel database. Utilizzato dagli operatori di back-office (Logistic Planner) per l'analisi globale.

`+ getRequestsByClientUsername(username: String): List<TransportRequestResponseDTO>`

Recupera lo storico delle richieste filtrate per uno specifico cliente. Garantisce la *Data Isolation*, assicurando che l'utente veda solo i propri ordini.

- **OCL Pre:** `userRepository.existsByUsername(username)`
- **OCL Post:** `result->forAll(r | r.client.username == username)`

### 3.3.5. Class: `ExternalMapService`

**Responsabilità:** Service di infrastruttura che agisce da *Facade* verso il provider cartografico esterno (Mapbox). Centralizza tutte le chiamate HTTP esterne e implementa logiche di resilienza per la risoluzione degli indirizzi.

- **Stereotype:** `«Service» Mapbox API`
- **Operations:**

`+ calculateFullRoute(origin: String, dest: String): Route`

Calcola il percorso stradale ottimale tra due indirizzi testuali.

1. **Resolution:** Risolve gli indirizzi in coordinate GPS utilizzando una strategia ibrida (Dizionario Locale → API Mapbox).
2. **Routing:** Interroga la *Directions API* per ottenere distanza, durata stimata e la polilinea (geometria del percorso).
3. **Error Handling:** Se l'indirizzo esatto fallisce, tenta un *Fallback* sulla sola città.

- **Exception:** Solleva `BusinessRuleException` se la rotta è impossibile (es. attraverso oceano) o gli indirizzi sono inesistenti.

### 3.3.6. Interface: `RouteRepository`

**Responsabilità:** Gestione della persistenza dei dati geografici e tecnici dei percorsi (distanze, durate, polilinee). Estende `JpaRepository`.

- **Stereotype:** `«Repository»`
- **Generalization:** `JpaRepository<Route, Long>`

- **Operations:**

  `+ findAllByTripStatus(status: TripStatus): List<Route>`

  Esegue una *Derived Query* che attraversa la relazione tra Rotta e Viaggio. Restituisce tutte le rotte associate a viaggi che si trovano nello stato specificato.

  – **Use Case:** Utile per visualizzare sulla mappa globale solo i percorsi attivi (`IN_TRANSIT`) o pianificati, nascondendo quelli storici (`COMPLETED`).

- **Constraints (OCL):**

  – `context RouteRepository::findAllByTripStatus(status) pre: status <> null`

### 3.3.7. Interface: `TransportRequestRepository`

**Responsabilità:** Gestione della persistenza e del recupero delle entità `TransportRequest`. Supporta le dashboard operative filtrando gli ordini per stato di avanzamento e garantisce l'isolamento dei dati per i clienti.

- **Stereotype:** `«Repository»`
- **Generalization:** `JpaRepository<TransportRequest, Long>`
- **Operations:**

  `+ findByRequestStatus(status: RequestStatus): List<TransportRequest>`

  Restituisce l'elenco delle richieste filtrate per lo stato corrente.

  – **Use Case:** Utilizzato dai Pianificatori Logistici per identificare le richieste `PENDING` in attesa di valutazione.

  `+ findAllByClientId(clientId: Long): List<TransportRequest>`

  Restituisce l'intero storico delle richieste sottomesse da uno specifico cliente.

  – **Security:** Metodo fondamentale per la *Data Isolation*: assicura che ogni committente possa accedere in lettura esclusivamente ai propri ordini.

- **Constraints (OCL):**

  – `context TransportRequestRepository::findByRequestStatus(status) pre: status <> null`

- `context TransportRequestRepository::findAllByClientId(id) pre: id <> null`

### 3.3.8. Interface: `TripRepository`

**Responsabilità:** Gestione della persistenza e del recupero delle entità `Trip`. Fornisce query specializzate per il tracciamento dei viaggi tramite codici business, il filtraggio per stato operativo e il supporto alle dashboard mobili per gli autisti.

- **Stereotype:** `«Repository»`
- **Generalization:** `JpaRepository<Trip, Long>`
- **Operations:**

  **`+ findByTripCode(tripCode: String): Optional<Trip>`**
  Recupera un viaggio utilizzando il suo codice business univoco (es. "TRP-2024-0001"). Utile per API pubbliche o tracking dove non si vuole esporre l'ID numerico interno.

  **`+ findByRequestId(requestId: Long): Optional<Trip>`**
  Recupera il viaggio associato a una specifica richiesta di trasporto.
  - **Use Case:** Utilizzato nel Service Layer per garantire l'*Idempotenza*: se esiste già un viaggio per quella richiesta, viene restituito quello esistente invece di crearne uno nuovo.

  **`+ findByDriverIdOrderByCreatedAtDesc(driverId: Long): List<Trip>`**
  Restituisce lo storico completo dei viaggi assegnati a un autista, ordinati dal più recente.
  - **Use Case:** Popola la schermata "I Miei Viaggi" nell'App Mobile del conducente.

  **`+ findByStatus(status: TripStatus): List<Trip>`**
  Restituisce tutti i viaggi che si trovano esattamente nello stato specificato.

  **`+ findByStatusIn(statuses: List<TripStatus>): List<Trip>`**
  Restituisce i viaggi che si trovano in uno degli stati forniti nella lista. Utile per dashboard aggregate (es. visualizzare insieme viaggi `IN_PLANNING` e `WAITING_VALIDATION`).

- **Constraints (OCL):**
  - `context TripRepository::findByTripCode(code) pre: code <> null`

- `context TripRepository::findByDriverIdOrderByCreatedAtDesc(id) pre: id <> null`
- `context TripRepository::findByRequestId(id) pre: id <> null`

### 3.3.9. DTO: `RequestCreationDTO`

**Responsabilità:** Data Transfer Object utilizzato per l'acquisizione dei dati di una nuova richiesta di trasporto. Incapsula i dettagli logistici e le specifiche dimensionali del carico per la valutazione di fattibilità.

- **Stereotype:** `«DTO»`
- **Attributes:**
  - `originAddress: String`
    *Constraints:* `@NotBlank`
  - `destinationAddress: String`
    *Constraints:* `@NotBlank`
  - `pickupDate: LocalDate`
    *Constraints:* `@NotNull`, `@Future` (Non sono ammessi ritiri nel passato)
  - `loadType: String`
    *Constraints:* `@NotBlank` (Es. "Travi in cemento", "Turbina")
  - `weight: Double`
    *Constraints:* `@NotNull`, `@Positive` (Peso in kg)
  - `height: Double`
    *Constraints:* `@NotNull`, `@Positive` (Altezza in metri)
  - `length: Double`
    *Constraints:* `@NotNull`, `@Positive` (Lunghezza in metri)
  - `width: Double`
    *Constraints:* `@NotNull`, `@Positive` (Larghezza in metri)

### 3.3.10. DTO: `TransportRequestResponseDTO`

**Responsabilità:** Data Transfer Object per la visualizzazione dei dettagli di una richiesta esistente. Aggrega le informazioni sullo stato, i dati denormalizzati del cliente richiedente e la struttura annidata con le specifiche tecniche del carico.

- **Stereotype:** `«DTO»`
- **Attributes:**

- **Request Metadata:**
  - `id: Long`
  - `requestStatus: RequestStatus`
  - `pickupDate: LocalDate`
- **Logistics Info:**
  - `originAddress: String`
  - `destinationAddress: String`
- **Client Info:**
  - `clientId: Long`
  - `clientFullName: String`
    (Campo denormalizzato per visualizzazione rapida)
- **Load Details (Nested Object):**
  - `load.type: String`
  - `load.weightKg: Double`
  - `load.height: Double`
  - `load.length: Double`
  - `load.width: Double`

### 3.3.11. DTO: `TripAssignmentDTO`

**Responsabilità:** Data Transfer Object di input utilizzato nelle operazioni di pianificazione operativa. Trasporta gli identificativi delle risorse (Autista e Veicolo) selezionate dal Planner per un determinato viaggio.

- **Stereotype:** `«DTO»`
- **Attributes:**
  - `tripId: Long`
    *Constraints:* `@Positive` (Usato per controllo di consistenza con l'URL)
  - `driverId: Long`
    *Constraints:* `@NotNull`, `@Positive` (ID Autista selezionato)
  - `vehiclePlate: String`
    *Constraints:* `@NotBlank` (Targa veicolo selezionato)

### 3.3.12. DTO: `TripResponseDTO`

**Responsabilità:** Data Transfer Object (Read Model) per la visualizzazione completa dei dettagli del viaggio. Contiene dati denormalizzati e oggetti annidati per ridurre il numero di chiamate API necessarie al client.

- **Stereotype:** `«DTO»`
- **Attributes:**
  - **Core Info:**
    - `id: Long`
    - `tripCode: String` (Codice business visuale)
    - `status: TripStatus`
  - **Driver Info (Denormalized):**
    - `driverId: Long`
    - `driverName: String`
    - `driverSurname: String`
    - `currentLocation: String` *(Optional)*
  - **Vehicle Info (Denormalized):**
    - `vehiclePlate: String`
    - `vehicleModel: String` (Es. "Iveco Stralis")
  - **Client Info (Denormalized):**
    - `clientId: Long`
    - `clientFullName: String`
  - **Nested Objects:**
    - `request: TransportRequestResponseDTO` (Dettagli completi della richiesta originale)
    - `route: RouteResponseDTO` (Dettagli geografici del percorso)

### 3.3.13. DTO: `RouteResponseDTO`

**Responsabilità:** Data Transfer Object (Read Model) ottimizzato per la Dashboard del Traffic Coordinator. Aggrega i dati tecnici del percorso calcolato (distanze, geometria) con il contesto operativo del viaggio (codice ordine, tipologia carico), permettendo al coordinatore di valutare la fattibilità della rotta senza dover consultare più schermate.

- **Stereotype:** `«DTO»`

- **Attributes:**
  - **Identity & Context:**
    - `id: Long` (ID della Rotta)
    - `tripCode: String` (Riferimento Business es. "TRP-2024-X")
    - `plannerName: String` (Chi ha pianificato il viaggio)
    - `status: TripStatus`
  - **Logistics Data:**
    - `origin: String`
    - `destination: String`
    - `loadType: String` (Es. "Turbina" - Fondamentale per valutare i rischi)
  - **Technical Data:**
    - `routeDescription: String`
    - `distance: Double` (in km)
    - `duration: Double` (in minuti)
    - `polyline: String` (Stringa codificata per il rendering su mappa)
  - **Coordinates (Flattened):**
    - `startLat, startLon: Double`
    - `endLat, endLon: Double`

### 3.3.14. DTO: `RouteValidationRequestDTO`

**Responsabilità:** Data Transfer Object di input utilizzato dal Traffic Coordinator per emettere il verdetto tecnico sulla rotta proposta.

- **Stereotype:** `«DTO»`
- **Attributes:**
  - `approved: Boolean`
    *Constraints:* `@NotNull`. `True` conferma la rotta e sblocca il viaggio; `False` lo rimanda in pianificazione.
  - `feedback: String` *(Optional)*
    Note testuali obbligatorie in caso di rifiuto per spiegare al Planner le motivazioni (es. "Altezza sottopasso insufficiente al km 45").

### 3.3.15. Component: `TripMapper`

**Responsabilità:** Componente di mappatura complesso che funge da "Aggregatore". Non si limita a convertire l'entità `Trip`, ma denormalizza i dati recuperando informazioni dalle entità correlate (Autista, Veicolo, Richiesta) per costruire un DTO completo per il frontend.

- **Stereotype:** `«Component»`
- **Dependencies:** `RouteMapper`
- **Operations:**

  `+ toDTO(trip: Trip): TripResponseDTO`

  Converte il viaggio in DTO applicando logiche di *Data Enrichment*:

  – **Driver Vehicle:** Estrae nomi e modelli per evitare che il client debba fare query extra.

  – **Request:** Mappa i dettagli della richiesta originale (inclusi i dettagli del carico) delegando a un metodo interno.

  – **Route:** Delega la conversione dei dati geografici al componente `RouteMapper`.

  `+ toRequestDTO(req: TransportRequest): TransportRequestResponseDTO`

  Helper method interno per mappare i dettagli della richiesta di trasporto e appiattire la struttura nidificata `LoadDetails`.

### 3.3.16. Component: `RouteMapper`

**Responsabilità:** Componente specializzato nella trasformazione dei dati geografici del percorso. Segue il *Single Responsibility Principle* (SRP).

- **Stereotype:** `«Component»`
- **Operations:**

  `+ toDTO(route: Route): RouteResponseDTO`

  Converte l'entità `Route` nel relativo DTO.

  – **Geo-Flattening:** Gestisce l'esplosione ("flattening") degli oggetti `GeoLocation` (Start/End) in singoli campi primitivi (‘startLat‘, ‘startLon‘, ecc.) facilitando il rendering delle mappe lato client.

  – **Data Transfer:** Copia le metriche calcolate (distanza, durata) e la polilinea codificata.

### 3.3.17. Entity: `TransportRequest`

**Responsabilità:** Entità che rappresenta la richiesta formale di trasporto inserita dal cliente. Contiene i dati contrattuali (Mittente, Destinatario) e le specifiche logistiche (Carico, Data).

- **Stereotype:** `«Entity»`
- **Generalization:** `BaseEntity`
- **Attributes:**
  - `- client: User`
    Relazione `@ManyToOne` con il committente.
  - `- originAddress: String`
  - `- destinationAddress: String`
  - `- pickupDate: LocalDate`
  - `- requestStatus: RequestStatus`
  - `- load: LoadDetails` (Embedded)
- **Invariants (OCL):**
  - `context TransportRequest inv: self.originAddress <> self.destinationAddress`
  - `context TransportRequest inv: self.load.weightKg > 0`
  - `context TransportRequest inv: self.client <> null`

### 3.3.18. Entity: `Trip`

**Responsabilità:** Entità operativa che rappresenta il viaggio pianificato. Collega la richiesta commerciale alle risorse fisiche (Autista, Veicolo) e al percorso calcolato.

- **Stereotype:** `«Entity»`
- **Generalization:** `BaseEntity`
- **Attributes:**
  - `- tripCode: String`
  - `- status: TripStatus`
  - `- request: TransportRequest`
    Relazione `@OneToOne` obbligatoria.

- `- route: Route`
  Relazione `@OneToOne` con il percorso calcolato.
- `- driver: Driver` (Optional in fase iniziale)
- `- vehicle: Vehicle` (Optional in fase iniziale)

- **Operations:**

`+ generateCode(): void`
Metodo di callback `@PrePersist`. Se il codice viaggio è nullo, ne genera uno univoco nel formato `TRP-YYYY-UUID`.

- **Invariants (OCL):**
  - `context Trip inv: self.status == CONFIRMED implies (self.driver <> null and self.vehicle <> null)`
  - `context Trip inv: self.request <> null`

### 3.3.19. Entity: `Route`

**Responsabilità:** Entità che memorizza i dati tecnici del percorso calcolato dal Routing Engine.

- **Stereotype:** `«Entity»`
- **Generalization:** `BaseEntity`
- **Attributes:**
  - `- description: String`
  - `- routeDistance: Double` (in km)
  - `- routeDuration: Double` (in minuti)
  - `- polyline: String` (LONGTEXT)
  - `- startLocation: GeoLocation` (Embedded)
  - `- endLocation: GeoLocation` (Embedded)
  - `- trip: Trip` (Relazione inversa)

### 3.3.20. Value Object: `LoadDetails`

**Responsabilità:** Oggetto complesso che incapsula le specifiche fisiche e dimensionali del carico (Peso e Dimensioni). Implementa il pattern *Embeddable*, divenendo parte integrante dell'entità `TransportRequest` senza possedere identità propria. Fondamentale per i calcoli di compatibilità veicolo e routing.

- **Stereotype:** «Embeddable», «ValueObject»
- **Attributes:**
  - `- type: String`
    Descrizione qualitativa (es. "Turbina eolica").
  - `- quantity: Integer`
    Numero di unità (Default: 1).
  - `- weightKg: Double`
    Peso totale in kg. Critico per limiti di portata e ponti.
  - `- height: Double`
    Altezza in metri. Critica per tunnel e cavalcavia.
  - `- width: Double`
    Larghezza in metri. Determina la necessità di scorte tecniche.
  - `- length: Double`
    Lunghezza in metri. Critica per curve e manovre.
- **Invariants (OCL):**
  - `context LoadDetails inv: self.weightKg > 0`
  - `context LoadDetails inv: self.height > 0`
  - `context LoadDetails inv: self.width > 0`
  - `context LoadDetails inv: self.length > 0`
  - `context LoadDetails inv: self.quantity >= 1`

### 3.3.21. Enum: `RequestStatus`

Definizione enumerata del ciclo di vita commerciale di una richiesta di trasporto.

- **Values:** `PENDING`, `APPROVED`, `REJECTED`, `PLANNED`, `IN_PROGRESS`, `CANCELLED`, `COMPLETED`

### 3.3.22. Enum: `TripStatus`

Definizione enumerata del ciclo di vita operativo di un viaggio.

- **Values:** `IN_PLANNING`, `WAITING_VALIDATION`, `VALIDATED`, `MODIFICATION_REQUESTED`, `CONFIRMED`, `ACCEPTED`, `IN_TRANSIT`, `PAUSED`, `DELIVERING`, `COMPLETED`, `CANCELLED`

## 3.4. Package: `com.heavyroute.execution`

### 3.4.1. Class: `MobileGatewayController`

**Responsabilità:** API gateway per l'app mobile dell'autista.

- **Stereotype:** `«RestController»`
- **Operations:**

`+ acceptAssignment(tripId: Long): ResponseEntity<Void>`

Conferma incarico da parte dell'autista.

– **OCL Pre (Security):** `currentUser.role == Role.DRIVER`

`+ updateStatus(tripId: Long, dto: StatusUpdateDTO): ResponseEntity<Void>`

Aggiorna stato operativo e posizione GPS.

– **OCL Pre:** `dto.timestamp <= LocalDateTime.now()` (No date future)

`+ reportIncident(tripId: Long, dto: IncidentDTO): ResponseEntity<Void>`

Segnala un problema grave durante il viaggio.

– **OCL Pre:** `dto.description.length > 10`

### 3.4.2. Interface: `ExecutionService`

**Responsabilità:** Logica operativa per la gestione del viaggio in tempo reale.

- **Stereotype:** `«Service»`
- **Operations:**

`+ handleAssignmentAcceptance(tripId: Long, driverId: Long): void`

Processa l'accettazione formale dell'incarico.

– **OCL Pre:** `trip.driver.id == driverId` (Autista corretto)

– **OCL Pre:** `trip.status == TripStatus.CONFIRMED`

– **OCL Post:** `trip.status == TripStatus.ACCEPTED`

`+ trackProgress(tripId: Long, status: TripStatus, loc: GeoLocation): void`

Traccia l'avanzamento del viaggio.

– **OCL Pre:** `trip.status in {ACCEPTED, IN_TRANSIT}`

- **OCL Pre:** `isValidTransition(trip.status, status)` (Macchina a stati)
- **OCL Post:** `trip.lastLocation == loc`

`+ handleIncidentReport(tripId: Long, incident: IncidentDTO): void`

Gestisce la procedura di emergenza per un incidente.

- **OCL Post:** `trip.status == TripStatus.PAUSED`
- **OCL Post:** `notificationService.alertSent == true`

### 3.4.3. DTO: `StatusUpdateDTO`

**Responsabilità:** Transfer Object per la telemetria periodica.

- **Attributes:**
  - `- newStatus: TripStatus`
  - `- latitude: Double`
  - `- longitude: Double`
  - `- timestamp: LocalDateTime`
- **Invariants (OCL):**
  - `context StatusUpdateDTO inv: self.latitude >= -90.0 AND self.latitude <= 90.0`
  - `context StatusUpdateDTO inv: self.longitude >= -180.0 AND self.longitude <= 180.0`

### 3.4.4. DTO: `IncidentDTO`

**Responsabilità:** Transfer Object per la segnalazione di problemi.

- **Attributes:**
  - `- description: String`
  - `- type: IncidentType` (GUASTO, INCIDENTE, ALTRO)
  - `- location: GeoLocation`
  - `- timestamp: LocalDateTime`

## 3.5. Package: `com.heavyroute.resources`

### 3.5.1. Class: `ResourceController`

**Responsabilità:** Controller REST per la gestione centralizzata delle risorse logistiche. Espone endpoint per l'interrogazione della flotta veicoli (con filtri dimensionali), il reperimento degli autisti disponibili e la gestione delle segnalazioni sulla viabilità.

- **Stereotype:** `«RestController»`
- **Base Path:** `/api/resources`
- **Operations:**

`+ getAllVehicles(): ResponseEntity<List<VehicleResponseDTO»`

Restituisce l'elenco completo della flotta aziendale.

– **Security:** Richiede ruolo `LOGISTIC_PLANNER`.

`+ getAvailableVehicles(): ResponseEntity<List<VehicleResponseDTO»`

Restituisce la lista dei veicoli che hanno stato operativo `AVAILABLE`.

– **Security:** Richiede ruolo `LOGISTIC_PLANNER`.

`+ getCompatibleVehicles(w: Double, h: Double, wd: Double, l: Double): List<VehicleDTO>`

Filtro avanzato che restituisce i veicoli disponibili capaci di trasportare un carico con le specifiche dimensioni e peso forniti.

– **Security:** Richiede ruolo `LOGISTIC_PLANNER`.

– **OCL Pre:** `weight >= 0 AND height >= 0 AND width >= 0 AND length >= 0`

`+ getAvailableDrivers(): ResponseEntity<List<UserResponseDTO»`

Restituisce la lista degli autisti che sono attualmente in stato `FREE`.

– **Security:** Richiede ruolo `LOGISTIC_PLANNER`.

`+ addVehicle(dto: VehicleCreationDTO): ResponseEntity<VehicleResponseDTO>`

Aggiunge un nuovo veicolo alla flotta aziendale.

– **OCL Pre:** `dto <> null`

`+ reportEvent(dto: RoadEventCreationDTO): ResponseEntity<RoadEventResponseDTO>`

Registra un nuovo evento stradale (incidente, cantiere, chiusura) che influenza la viabilità.

– **OCL Pre:** `dto <> null`

- **OCL Pre:** `dto.validFrom <= dto.validTo`

**+ `getActiveEvents(): ResponseEntity<List<RoadEventResponseDTO»`**

Restituisce tutte le segnalazioni stradali attualmente attive.

### 3.5.2. Interface: `ResourceService`

**Responsabilità:** Interfaccia di servizio per la gestione operativa delle risorse (flotta veicoli) e della viabilità (eventi stradali). Gestisce il ciclo di vita dei veicoli e fornisce query avanzate per il dispatching.

- **Stereotype:** `«Service»`
- **Operations:**

**+ `createVehicle(dto: VehicleCreationDTO): VehicleResponseDTO`**

Crea e persiste un nuovo veicolo nel sistema.

- **OCL Pre:** `!vehicleRepository.existsByLicensePlate (dto.licensePlate)`
- **OCL Post:** `result <> null AND result.id <> null`

**+ `getAllVehicles(): List<VehicleResponseDTO>`**

Recupera l'intera flotta veicoli indipendentemente dallo stato operativo.

**+ `getAvailableVehicles(): List<VehicleResponseDTO>`**

Restituisce esclusivamente i veicoli con stato `AVAILABLE`. Utile per popolamento rapido di liste di selezione.

- **OCL Post:** `result->forAll(v | v.status == VehicleStatus. AVAILABLE)`

**+ `getAvailableCompatibleVehicles(w: Double, h: Double, wd: Double, l: Double): List<VehicleResponseDTO>`**

Esegue una ricerca mirata per trovare veicoli che siano contemporaneamente `AVAILABLE` e capaci di trasportare le dimensioni specificate.

- **OCL Pre:** `w >= 0 AND h >= 0 AND wd >= 0 AND l >= 0`
- **OCL Post:** `result->forAll(v | v.status == VehicleStatus. AVAILABLE)`

**+ `createRoadEvent(dto: RoadEventCreationDTO): RoadEventResponseDTO`**

Registra un nuovo evento di viabilità (es. cantiere, incidente) che potrebbe impattare il calcolo dei percorsi.

- **OCL Post:** `roadEventRepository.exists(result.id)`

**+ `getActiveEvents(): List<RoadEventResponseDTO>`**

Recupera la lista degli eventi stradali correnti (dove `isActive == true`).

### 3.5.3. Interface: `VehicleService`

**Responsabilità:** Servizio verticale dedicato alla business logic specifica per la gestione della flotta veicoli.

- **Stereotype:** `«Service»`
- **Operations:**

  **`+ findAvailableVehicles(): List<VehicleResponseDTO>`**

  Interroga il database per recuperare esclusivamente i veicoli pronti all'uso (stato `AVAILABLE`), mappandoli nei relativi DTO di risposta.

  – **OCL Post:** `result->forAll(v | v.status == VehicleStatus.AVAILABLE)`

### 3.5.4. Interface: `VehicleRepository`

**Responsabilità:** Gestione della persistenza e del recupero delle entità `Vehicle`. Estende `JpaRepository` includendo query specifiche per il dimensionamento del carico.

- **Stereotype:** `«Repository»`
- **Generalization:** `JpaRepository<Vehicle, Long>`
- **Operations:**

  **`+ findByLicensePlate(plate: String): Optional<Vehicle>`**

  Restituisce un container opzionale con il veicolo associato alla targa fornita, se esistente.

  **`+ existsByLicensePlate(plate: String): boolean`**

  Verifica efficiente dell'esistenza di una targa nel sistema (usato per validazione pre-save).

  **`+ findAllByStatus(status: VehicleStatus): List<Vehicle>`**

  Restituisce la lista completa dei veicoli filtrata per stato operativo (es. AVAILABLE, MAINTENANCE).

  **`+ findCompatibleVehicles(w: Double, h: Double, wd: Double, l: Double, s: VehicleStatus): List<Vehicle>`**

  Query JPQL complessa. Restituisce i veicoli che si trovano nello stato specificato `s` e che possiedono capacità di carico e dimensioni fisiche (maxLoad, maxHeight, maxWidth, maxLength) superiori o uguali a quelle richieste.

- **Constraints (OCL):**

- `context VehicleRepository::findByLicensePlate(plate) pre: plate <> null`
- `context VehicleRepository::save(vehicle) pre: vehicle.licensePlate <> null`
- `context VehicleRepository::findCompatibleVehicles(w, h, wd, l, s) pre: w >= 0 and h >= 0`
- `context VehicleRepository::findCompatibleVehicles(w, h, wd, l, s) pre: wd >= 0 and l >= 0`
- `context VehicleRepository::findCompatibleVehicles(w, h, wd, l, s) pre: s <> null`

### 3.5.5. Interface: `RoadEventRepository`

**Responsabilità:** Gestione della persistenza e del recupero delle entità `RoadEvent`. Estende `JpaRepository` fornendo query specializzate per il filtraggio temporale e geospaziale (Bounding Box).

- **Stereotype:** `«Repository»`
- **Generalization:** `JpaRepository<RoadEvent, Long>`
- **Operations:**

  **`+ findAllActiveEvents(referenceTime: LocalDateTime): List<RoadEvent>`**

  Esegue una query JPQL per restituire gli eventi attivi nell'istante specificato. Considera attivi gli eventi dove `referenceTime` è compreso tra `validFrom` e `validTo` (incluso il caso in cui `validTo` sia `NULL` per eventi a durata indefinita).

  **`+ findBySeverity(severity: EventSeverity): List<RoadEvent>`**

  Restituisce la lista degli eventi filtrata in base al livello di gravità specificato (es. CRITICAL).

  **`+ findEventsInArea(minLat: Double, maxLat: Double, minLon: Double, maxLon: Double): List<RoadEvent>`**

  Query geospaziale che restituisce gli eventi la cui posizione rientra nel rettangolo di delimitazione definito dai parametri. Fondamentale per il Routing Engine.

- **Constraints (OCL):**

- context RoadEventRepository::findAllActiveEvents(referenceTime) pre: referenceTime <> null
- context RoadEventRepository::findBySeverity(severity) pre: severity <> null
- context RoadEventRepository::findEventsInArea(minLat, maxLat, minLon, maxLon) pre: minLat <> null and maxLat <> null
- context RoadEventRepository::findEventsInArea(minLat, maxLat, minLon, maxLon) pre: minLon <> null and maxLon <> null
- context RoadEventRepository::findEventsInArea(minLat, maxLat, minLon, maxLon) pre: minLat <= maxLat
- context RoadEventRepository::findEventsInArea(minLat, maxLat, minLon, maxLon) pre: minLon <= maxLon
- context RoadEventRepository::findEventsInArea(minLat, maxLat, minLon, maxLon) pre: minLat >= -90.0 and maxLat <= 90.0
- context RoadEventRepository::findEventsInArea(minLat, maxLat, minLon, maxLon) pre: minLon >= -180.0 and maxLon <= 180.0
- context RoadEventRepository::save(roadEvent) pre: roadEvent <> null
- context RoadEventRepository::save(roadEvent) pre: roadEvent.location <> null

### 3.5.6. DTO: `VehicleCreationDTO`

**Responsabilità:** Data Transfer Object utilizzato per la registrazione di un nuovo mezzo nella flotta aziendale. Include validazioni rigorose sulle dimensioni fisiche per garantire la coerenza dei dati di pianificazione.

- **Stereotype:** «DTO»
- **Attributes:**
  - `licensePlate: String`
    *Constraints:* `@NotBlank` (Identificativo univoco)
  - `model: String`
    *Constraints:* `@NotBlank`
  - `maxLoadCapacity: Double`
    *Constraints:* `@NotNull`, `@Positive` (Portata in kg)
  - `maxHeight: Double`
    *Constraints:* `@NotNull`, `@Positive` (Altezza in metri)

  - `maxWidth: Double`
    *Constraints:* `@NotNull`, `@Positive` (Larghezza in metri)
  - `maxLength: Double`
    *Constraints:* `@NotNull`, `@Positive` (Lunghezza in metri)
  - `status: VehicleStatus` *(Optional)*

### 3.5.7. DTO: `VehicleResponseDTO`

**Responsabilità:** Data Transfer Object per la visualizzazione dei dettagli veicolo. Arricchisce i dati strutturali con flag booleani di comodo per l'interfaccia utente.

- **Stereotype:** `«DTO»`
- **Attributes:**
  - `id: Long`
  - `licensePlate: String`
  - `model: String`
  - `maxLoadCapacity: Double`
  - `maxHeight: Double`
  - `maxWidth: Double`
  - `maxLength: Double`
  - `status: VehicleStatus`
  - `available: boolean`
    *Computed:* `true` se lo stato è `AVAILABLE`.
  - `inMaintenance: boolean`
    *Computed:* `true` se lo stato è `MAINTENANCE`.

### 3.5.8. DTO: `RoadEventCreationDTO`

**Responsabilità:** Data Transfer Object utilizzato per la segnalazione di nuovi eventi o pericoli stradali. Include validazioni rigorose sulle coordinate GPS per garantire la coerenza spaziale.

- **Stereotype:** `«DTO»`
- **Attributes:**
  - `type: RoadEventType`
    *Constraints:* `@NotNull` (Es. ACCIDENT, CONSTRUCTION)

  - `severity: EventSeverity`
    *Constraints:* `@NotNull` (Es. CRITICAL, LOW)
  - `description: String` *(Optional)*
  - `latitude: Double`
    *Constraints:* `@NotNull`, `Range[-90.0, 90.0]`
  - `longitude: Double`
    *Constraints:* `@NotNull`, `Range[-180.0, 180.0]`
  - `validFrom: LocalDateTime`
    *Constraints:* `@NotNull` (Inizio validità)
  - `validTo: LocalDateTime` *(Optional)*
    Se `null`, l'evento è considerato a tempo indeterminato.

### 3.5.9. DTO: `RoadEventResponseDTO`

**Responsabilità:** Data Transfer Object per la visualizzazione degli eventi stradali. Arricchisce i dati grezzi con flag booleani calcolati per facilitare il rendering su mappa.

- **Stereotype:** `«DTO»`
- **Attributes:**
  - `id: Long`
  - `type: RoadEventType`
  - `severity: EventSeverity`
  - `description: String`
  - `latitude: Double`
  - `longitude: Double`
  - `validFrom: LocalDateTime`
  - `validTo: LocalDateTime`
  - `active: boolean`
    *Computed:* `true` se la data corrente rientra nella finestra di validità.
  - `blocking: boolean`
    *Computed:* `true` se `severity == CRITICAL`. Indica un blocco totale della tratta.

### 3.5.10. Class: `VehicleMapper`

**Responsabilità:** Componente stateless responsabile della trasformazione dei dati tra l'entità `Vehicle` e i DTO specifici (CQRS). Gestisce la logica di arricchimento dei dati per il frontend.

- **Stereotype:** `«Mapper»`
- **Operations:**

  `+ toResponseDTO(entity: Vehicle): VehicleResponseDTO`

  Converte l'entità persistente nel DTO di risposta. Oltre alla mappatura 1:1 dei campi tecnici, calcola i valori dei flag booleani di utilità (`available`, `inMaintenance`) basandosi sullo stato corrente dell'enumerativo `VehicleStatus`.

  `+ toResponseDTOList(entities: List<Vehicle>): List<VehicleResponseDTO>`

  Helper method per la conversione batch di una lista di veicoli utilizzando l'operazione `toResponseDTO`.

  `+ toEntity(dto: VehicleCreationDTO): Vehicle`

  Costruisce una nuova istanza dell'entità `Vehicle` a partire dai dati di input. Utilizza il pattern *Builder* per garantire un'inizializzazione pulita dell'oggetto.

### 3.5.11. Class: `RoadEventMapper`

**Responsabilità:** Componente stateless che gestisce la trasformazione bidirezionale tra l'entità `RoadEvent` e i relativi oggetti di trasporto. Risolve il disallineamento strutturale tra la rappresentazione piatta dei DTO (coordinate separate) e il modello a oggetti dell'entità (Value Object).

- **Stereotype:** `«Mapper»`
- **Operations:**

  `+ toEntity(dto: RoadEventCreationDTO): RoadEvent`

  Costruisce l'entità di dominio a partire dai dati di input. Si occupa di istanziare il Value Object `GeoLocation` incapsulando la latitudine e la longitudine fornite dal DTO.

  `+ toResponseDTO(entity: RoadEvent): RoadEventResponseDTO`

  Genera il DTO di risposta per il frontend. Esegue due operazioni chiave:

  1. **Flattening Geografico:** Estrae le coordinate dall'oggetto embedded `GeoLocation` per esporle come campi primitivi semplici.
  2. **Computed Fields Mapping:** Invoca i metodi di business dell'entità (`isActive`, `isBlocking`) e mappa i risultati booleani nel DTO.

### 3.5.12. Entity: `Vehicle`

**Responsabilità:** Entità che rappresenta un mezzo di trasporto fisico (es. trattore stradale, furgone) presente nell'inventario. Contiene le specifiche tecniche dimensionali per il controllo di compatibilità.

- **Stereotype:** `«Entity»`
- **Generalization:** `BaseEntity`
- **Attributes:**
  - `- licensePlate: String`
  - `- model: String`
  - `- maxLoadCapacity: Double`
  - `- maxHeight: Double`
  - `- maxLength: Double`
  - `- maxWidth: Double`
  - `- status: VehicleStatus`
  - `- currentDriver: Driver`
- **Operations:**

  **`+ isAvailable(): boolean`**
  Restituisce `true` se lo stato del veicolo è `AVAILABLE`.

  **`+ isInMaintenance(): boolean`**
  Restituisce `true` se il veicolo si trova in stato `MAINTENANCE`.

- **Invariants (OCL):**
  - `context Vehicle inv: self.maxLoadCapacity > 0`
  - `context Vehicle inv: self.maxHeight > 0`
  - `context Vehicle inv: self.maxLength > 0`
  - `context Vehicle inv: self.maxWidth > 0`
  - `context Vehicle inv: self.licensePlate <> null`
  - `context Vehicle inv: self.model <> null`
  - `context Vehicle inv: self.status <> null`
  - `context Vehicle inv: self.licensePlate.matches("^[A-Z]{2}\d{3}[A-Z]{2}$")`

### 3.5.13. Entity: `RoadEvent`

**Responsabilità:** Entità che rappresenta un evento avverso o una segnalazione sulla rete stradale (es. incidente, lavori in corso). Gli eventi sono geolocalizzati e possiedono una finestra temporale di validità.

- **Stereotype:** `«Entity»`
- **Generalization:** `BaseEntity`
- **Attributes:**
  - `- description: String`
  - `- location: GeoLocation`
  - `- type: RoadEventType`
  - `- severity: EventSeverity`
  - `- validFrom: LocalDateTime`
  - `- validTo: LocalDateTime`
- **Operations:**

  `+ isActive(): boolean`
  : Restituisce `true` se l'istante attuale rientra nella finestra temporale di validità. Gestisce correttamente eventi a tempo indeterminato (dove `validTo` è `null`).

  `+ isBlocking(): boolean`
  : Restituisce `true` se l'evento è di gravità critica (`CRITICAL`), implicando un blocco totale della viabilità.

- **Invariants (OCL):**
  - `context RoadEvent inv: self.location <> null`
  - `context RoadEvent inv: self.type <> null`
  - `context RoadEvent inv: self.severity <> null`
  - `context RoadEvent inv: self.validFrom <> null`
  - `context RoadEvent inv: self.validTo <> null implies self.validFrom < self.validTo`

### 3.5.14. Enum: `VehicleStatus`

Definizione enumerata degli stati in cui può trovarsi un veicolo.

- **Values:** `AVAILABLE, IN_USE, MAINTENANCE, DECOMMISSIONED`

#### 3.5.15. Enum: `RoadEventType`

Definizione enumerata delle varie tipologie di evento stradale.

- **Values:** `ACCIDENT`, `CONSTRUCTION`, `TRAFFIC_JAM`, `WEATHER_CONDITION`, `POLICE_CHECKPOINT`, `OBSTACLE`

#### 3.5.16. Enum: `EventSeverity`

Definizione enumerata della gravità di un evento stradale.

- **Values:** `LOW`, `MEDIUM`, `CRITICAL`

### 3.6. Package: `com.heavyroute.notification`

#### 3.6.1. Class: `NotificationController`

**Responsabilità:** Controller REST per la gestione delle notifiche lato client (Frontend). Espone endpoint per la consultazione dello storico avvisi e per l'aggiornamento dello stato di lettura (acknowledgment).

- **Stereotype:** `«RestController»`
- **Operations:**

`+ getMyNotifications(userId: Long): ResponseEntity<List<NotificationDTO»`

Restituisce la lista di tutte le notifiche indirizzate all'utente specificato.

– **OCL Pre:** `userId <> null`

`+ markRead(id: Long): ResponseEntity<Void>`

Endpoint `PATCH` che marca una specifica notifica come "letta". Restituisce uno status HTTP 204 (No Content) in caso di successo.

– **OCL Pre:** `id <> null`

#### 3.6.2. Interface: `NotificationService`

**Responsabilità:** Interfaccia per la gestione del ciclo di vita delle notifiche utente. Gestisce la creazione, la persistenza e l'aggiornamento di stato (lettura) degli avvisi di sistema.

- **Stereotype:** `«Interface»`
- **Operations:**

`+ send(userId: Long, title: String, msg: String, type: NotificationType, refId: Long): void`

Crea e persiste una nuova notifica nel database con stato iniziale `UNREAD`.

- **OCL Pre:** `userId <> null`
- **OCL Pre:** `title <> null AND message <> null`
- **Note:** Predisposto per l'integrazione futura con sistemi Push (es. Firebase).

`+ getNotificationsForUser(userId: Long): List<NotificationDTO>`

Recupera lo storico delle notifiche per un utente specifico, ordinate cronologicamente dalla più recente.

- **OCL Pre:** `userId <> null`

`+ markAsRead(notificationId: Long): void`

Aggiorna lo stato di una notifica specifica impostandolo su `READ`.

- **OCL Pre:** `notificationId <> null`
- **OCL Post:** `notificationRepository.findById(notificationId).status == READ`
- **Exception:** Solleva `ResourceNotFoundException` se l'ID non esiste.

### 3.6.3. Interface: `NotificationRepository`

**Responsabilità:** Gestione della persistenza e del recupero delle entità `Notification`. Estende `JpaRepository` fornendo query ottimizzate per la dashboard utente.

- **Stereotype:** `«Repository»`
- **Generalization:** `JpaRepository<Notification, Long>`
- **Operations:**

`+ findByRecipientIdOrderByCreatedAtDesc(recipientId: Long): List<Notification>`

Restituisce la lista completa delle notifiche indirizzate a un utente specifico.

`+ countByRecipientIdAndStatus(recipientId: Long, status: NotificationStatus): long`

Esegue un conteggio efficiente delle notifiche che si trovano in uno specifico stato.

- **Constraints (OCL):**
  - `context NotificationRepository::findByRecipientIdOrderByCreatedAtDesc(id)`
    `pre: id <> null`
  - `context NotificationRepository::countByRecipientIdAndStatus(id, status)`
    `pre: id <> null`
  - `context NotificationRepository::countByRecipientIdAndStatus(id, status)`
    `pre: status <> null`

### 3.6.4. DTO: `NotificationDTO`

**Responsabilità:** Data Transfer Object utilizzato per trasmettere i dettagli di una notifica al client (Frontend). Rappresenta un avviso persistente visualizzabile nella dashboard utente.

- **Stereotype:** `«DTO»`
- **Attributes:**
  - `id: Long`
    Identificativo univoco della notifica.
  - `title: String`
    Titolo breve dell'avviso (es. "Nuovo Viaggio Assegnato").
  - `message: String`
    Corpo del messaggio con i dettagli.
  - `type: NotificationType`
    Categoria della notifica (utile per icone o colori distintivi nella UI).
  - `status: NotificationStatus`
    Stato di lettura (`READ`, `UNREAD`).
  - `createdAt: LocalDateTime`
    Timestamp di creazione.
  - `referenceId: Long`
    Riferimento opzionale all'ID dell'entità correlata (es. ID del viaggio o dell'evento), utile per creare link di navigazione ("Deep Linking").

### 3.6.5. Class: `NotificationMapper`

**Responsabilità:** Componente stateless responsabile della trasformazione dei dati (Object Mapping) per il modulo notifiche. Isola il modello di dominio (Entity) dalla rappresentazione JSON inviata al client (DTO).

- **Stereotype:** «Mapper»
- **Operations:**

  `+ toDTO(entity: Notification): NotificationDTO`

  Converte l'entità persistente `Notification` nel DTO di risposta. Esegue una mappatura diretta dei campi informativi (titolo, messaggio), dello stato di lettura e dei metadati temporali (`createdAt`), includendo l'eventuale `referenceId` per permettere al frontend di collegare la notifica all'oggetto specifico (es. aprire il dettaglio del viaggio).

### 3.6.6. Entity: `Notification`

**Responsabilità:** Entità persistente che rappresenta un singolo messaggio di notifica destinato ad un utente. Mantiene il contenuto informativo e traccia lo stato di lettura (Read/Unread).

- **Stereotype:** «Entity»
- **Generalization:** `BaseEntity`
- **Attributes:**

  – `- recipientId: Long`
  ID dell'utente destinatario. Disaccoppiato dall'entità User per modularità.

  – `- title: String`

  – `- message: String`

  – `- type: NotificationType`

  – `- status: NotificationStatus`

  – `- referenceId: Long`
  (Opzionale) ID dell'oggetto di dominio a cui la notifica si riferisce (es. ID del viaggio).

- **Invariants (OCL):**

  – `context Notification inv: self.recipientId <> null`

  – `context Notification inv: self.title <> null`

  – `context Notification inv: self.message <> null`

  – `context Notification inv: self.type <> null`

  – `context Notification inv: self.status <> null`

### 3.6.7. Enum: `NotificationStatus`

Definisce lo stato di lettura della notifica all'interno del suo ciclo di vita.

- **Values:** `UNREAD, READ`

### 3.6.8. Enum: `NotificationType`

Definisce la categoria semantica e la priorità della notifica. Utilizzato dal frontend per decidere lo stile visivo (colore, icona) e dal backend per eventuali logiche di routing (es. le notifiche URGENT potrebbero inviare anche un SMS).

- **Values:** `INFO, ALERT, ASSIGNMENT, URGENT`

## 3.7. Package: `com.heavyroute.common`

Questo package contiene i componenti trasversali utilizzati da tutti gli altri moduli: gestione errori, superclassi JPA e Value Object di dominio.

### 3.7.1. Class: `DebugController`

**Responsabilità:** Controller di diagnostica e manutenzione per ambienti di sviluppo e test (Dev/QA). Fornisce funzionalità per l'ispezione dello stato del sistema e il ripristino dell'ambiente (Reset) senza riavvio.

- **Stereotype:** `«RestController»`
- **Profile Constraint:** `!prod` (Componente attivo SOLO in ambienti non produttivi)
- **Base Path:** `/api/debug`
- **Operations:**

  `+ resetDatabase(): ResponseEntity<String>`

  Esegue una procedura atomica di *Hard Reset* del database:

  1. Elimina tutti i dati operativi (Viaggi, Richieste, Veicoli, Utenti) rispettando l'ordine dei vincoli referenziali.
  2. Invoca il componente `DataSeeder` per ripopolare il database con i dati di default.

  – **Transaction:** Atomica (Rollback completo in caso di errore).

  `+ dumpDatabase(): ResponseEntity<Map<String, Object»`

  Restituisce un'istantanea (Snapshot) JSON dello stato attuale del database. Utile per il debugging remoto.

  - **Security Post:** `result.users->forAll(u | u.password == "[HIDDEN]")` (Le password vengono oscurate prima della serializzazione).

`+ getUsersByRole(role: UserRole): ResponseEntity<List<User»`

Endpoint di utilità per filtrare rapidamente gli utenti in base al ruolo.

  - **Security Post:** `result->forAll(u | u.password == "[HIDDEN]")`

`+ getRequestsByStatus(status: RequestStatus): ResponseEntity<List<TransportRequest»`

Endpoint di utilità per visualizzare tutte le richieste di trasporto che si trovano in uno specifico stato (es. `PENDING`).

### 3.7.2. Class: `GlobalExceptionHandler`

**Responsabilità:** Gestore centralizzato (AOP) delle eccezioni applicative. Intercetta gli errori e li normalizza nel formato standard RFC 7807 (*Problem Details*), garantendo risposte API coerenti e sicure.

- **Stereotype:** `«Exception»`
- **Operations:**

`+ handleMethodArgumentNotValid(...): ResponseEntity<Object>`

Gestisce errori di validazione sui DTO (es. `@NotNull`). Restituisce **400 Bad Request** con una mappa dettagliata `campo -> errore`.

`+ handleNotFound(ex: ResourceNotFoundException): ProblemDetail`

Gestisce il mancato ritrovamento di entità. Restituisce **404 Not Found**.

`+ handleBusinessRule(ex: BusinessRuleException): ProblemDetail`

Gestisce violazioni logiche (es. stato non valido). Restituisce **409 Conflict**.

`+ handleUserAlreadyExist(ex: UserAlreadyExistException): ProblemDetail`

Gestisce tentativi di registrazione duplicata. Restituisce **409 Conflict**.

`+ handleBadCredentials(ex: BadCredentialsException): ProblemDetail`

Gestisce errori di login. Restituisce **401 Unauthorized** con messaggio generico per prevenire l'enumerazione degli utenti (Security best practice).

`+ handleDatabaseConstraint(ex: DataIntegrityViolationException): ResponseEntity`

Analizza l'errore SQL sottostante (es. *Data too long*, *Duplicate entry*) e lo traduce in un messaggio utente comprensibile. Restituisce **409 Conflict** o **400 Bad Request**.

### 3.7.3. Class: `BaseEntity`

**Responsabilità:** Superclasse astratta per la persistenza. Fornisce l'identificativo univoco (Primary Key) e gestisce automaticamente il ciclo di vita temporale (Auditing) tramite Spring Data JPA.

- **Stereotype:** `«MappedSuperclass», {abstract}`
- **Attributes:**
  - `- id: Long`
    *Constraints:* `@Id`, `@GeneratedValue(IDENTITY)`
  - `- createdAt: LocalDateTime`
    *Constraints:* `@CreatedDate`, `@Column(updatable=false)`
  - `- updatedAt: LocalDateTime`
    *Constraints:* `@LastModifiedDate`

### 3.7.4. Value Object: `GeoLocation`

**Responsabilità:** Value Object che incapsula una coordinata geografica (GPS). Implementa il pattern *Embeddable* per essere incluso in altre entità e applica una validazione difensiva nel costruttore per garantire la coerenza dei dati geografici.

- **Stereotype:** `«Embeddable», «ValueObject»`
- **Attributes:**
  - `- latitude: Double`
  - `- longitude: Double`
- **Operations:**

  `+ GeoLocation(lat: Double, lon: Double)`
  Costruttore che applica le invarianti di dominio. Solleva `IllegalArgumentException` se i parametri sono nulli o violano i limiti geografici terrestri.

  `+ toString(): String`
  Restituisce la rappresentazione testuale formattata "lat, lon".

- **Invariants (OCL):**

- `context GeoLocation inv: self.latitude <> null and self.longitude <> null`
- `context GeoLocation inv: self.latitude >= -90.0 and self.latitude <= 90.0`
- `context GeoLocation inv: self.longitude >= -180.0 and self.longitude <= 180.0`

### 3.7.5. Class: `DataSeeder`

**Responsabilità:** Componente di inizializzazione (Bootstrap) che popola il database con dati dimostrativi (Seed Data) all'avvio dell'applicazione, se il database risulta vuoto. Fondamentale per avere un ambiente di sviluppo pronto all'uso.

- **Stereotype:** `«Component»`
- **Realization:** `CommandLineRunner`
- **Profile Constraint:** `!test` (Disattivato durante i test automatici)
- **Operations:**

  `+ run(args: String[]): void`

  Metodo di entry-point eseguito automaticamente da Spring Boot. Esegue una transazione atomica che crea:

  - **Staff:** Planner, Coordinatori e Account Manager con credenziali predefinite.
  - **Flotta:** Veicoli pesanti con specifiche dimensionali reali.
  - **Utenza:** Autisti e Clienti Aziendali (es. Hitachi, Ansaldo).
  - **Operatività:** Un set di richieste di trasporto pendenti per testare gli algoritmi di assegnazione.

### 3.7.6. Class: `CorsConfig`

**Responsabilità:** Configurazione globale per la gestione delle politiche *Cross-Origin Resource Sharing* (CORS). Abilita le chiamate API provenienti da domini o porte differenti (es. Frontend Angular/React).

- **Stereotype:** `«Configuration»`
- **Operations:**

  `+ corsConfigurer(): WebMvcConfigurer`

  Registra un Bean che definisce le regole di accesso HTTP:

- **Path:** `/**` (Tutti gli endpoint)
- **Origins:** `*` (Qualsiasi sorgente - Modalità Sviluppo)
- **Methods:** `GET, POST, PUT, DELETE, OPTIONS`