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
–	–	–	–
–	–	–	–

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
|   +-- model
|   +-- dto
+-- core                (Sottosistema Core Business)
|   +-- controller
|   +-- service
|   +-- repository
|   +-- model
|   +-- dto
+-- execution           (Sottosistema Esecuzione Mobile)
|   +-- controller
|   +-- service
|   +-- dto
+-- resources           (Sottosistema Risorse e Viabilità)
|   +-- controller
|   +-- service
|   +-- repository
|   +-- model
```

```
|   +-- dto
+-- notification        (Sottosistema Notifiche)
|   +-- service
|   +-- dto
+-- common              (Shared Kernel)
    +-- exception
    +-- util
    +-- model
    +-- enums
```

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:**

  `+ login(request: LoginRequestDTO): ResponseEntity<JwtResponseDTO>`

  Autentica l'utente e restituisce il token.

  - **OCL Pre:** `request.username <> null AND request.username.length > 0`
  - **OCL Pre:** `request.password <> null AND request.password.length > 0`
  - **OCL Post:** `result.statusCode == 200 IMPLIES result.body.token <> null`

  `+ forgotPassword(emailDto: EmailDTO): ResponseEntity<Void>`

  Avvia il reset password.

  - **OCL Pre:** `emailDto.email <> null`

3.1.2. Interface: `AuthService`

Responsabilità: Facade per la logica di sicurezza.

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

  `+ authenticate(username: String, password: String): String`

  Verifica credenziali e genera JWT.

  - **OCL Pre:** `username <> null AND password <> null`
  - **OCL Post:** `result <> null AND result.startsWith("eyJ")`

- **Exception:** `BadCredentialsException` if `!userRepository.exists(usernam hash(password))`

`+ resetPassword(token: String, newPassword: String): void`

Cambia la password dell'utente.

- **OCL Pre:** `tokenStore.isValid(token) == true`
- **OCL Pre:** `newPassword.length >= 8`
- **OCL Post:** `user.passwordHash == hash(newPassword)`

3.1.3. DTO: `LoginRequestDTO`

Responsabilità: Payload per la richiesta di login.

- **Attributes:**
  - `@NotBlank private String username;`
  - `@NotBlank private String password;`

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 role;`
- **Invariants (OCL):**
  - `context JwtResponseDTO inv: self.token <> null`
  - `context JwtResponseDTO inv: self.role <> null`

3.1.5. DTO: `EmailDTO`

Responsabilità: Payload per richieste basate su email.

- **Attributes:**
  - `@Email @NotBlank private String email;`

3.2. Package: `com.heavyroute.users`

3.2.1. Class: `UserController`

Responsabilità: Controller REST per la gestione anagrafica degli utenti.

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

  `+ registerClient(dto: ClientRegistrationDTO): ResponseEntity<UserDTO>`

  Registrazione self-service per un nuovo committente.

  - **OCL Pre:** `dto <> null AND dto.vatNumber <> null`
  - **OCL Post:** `result.body.role == Role.CLIENT`
  - **OCL Post:** `result.body.active == false` (Pending Approval)

  `+ createInternalUser(dto: InternalUserDTO): ResponseEntity<UserDTO>`

  Creazione di un utente staff (solo Gestore Account).

  - **OCL Pre (Security):** `currentUser.role == Role.GA`
  - **OCL Pre:** `dto.role in {PL, TC, DRIVER}`

  `+ changeStatus(id: Long, status: UserStatus): ResponseEntity<Void>`

  Modifica lo stato di un utente (attivazione/disattivazione).

  - **OCL Pre (Security):** `currentUser.role == Role.GA`
  - **OCL Pre:** `userRepository.existsById(id)`

3.2.2. Interface: `UserService`

Responsabilità: Logica di business per il ciclo di vita e la gestione degli utenti.

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

  `+ registerNewClient(dto: ClientRegistrationDTO): User`

  Crea un nuovo committente in attesa.

  - **OCL Pre:** `!userRepository.existsByUsername(dto.username)`
  - **Exception:** `UserAlreadyExistsException`

  `+ createInternalUser(dto: InternalUserDTO): User`

  Crea un utente interno con password temporanea.

  - **OCL Pre:** `!userRepository.existsByEmail(dto.email)`

- **OCL Post:** `result.passwordHash <> null` (password generata)

`+ deactivateUser(id: Long): void`

Disabilita l'accesso al sistema.

- **OCL Post:** `user.active == false`
- **OCL Post:** `tokenStore.invalidateAllSessions(id)`

3.2.3. Entity: `User`

Responsabilità: Entità JPA che rappresenta un utente generico.

- **Stereotype:** `«Entity»`
- **Attributes:**
  - `- id: Long`
  - `- username: String`
  - `- email: String`
  - `- passwordHash: String`
  - `- role: Role`
  - `- active: Boolean`
- **Invariants (OCL):**
  - `context User inv: self.username.length >= 4`
  - `context User inv: self.email.matches("^.+@.+..+$")`
  - `context User inv: self.role <> null`

3.2.4. DTO: `ClientRegistrationDTO`

- **Attributes:**
  - `@NotBlank private String username;`
  - `@NotBlank private String password;`
  - `@Email private String email;`
  - `@NotBlank private String companyName;`
  - `@NotBlank private String vatNumber;`

3.2.5. DTO: `InternalUserDTO`

- **Attributes:**
  - `@NotBlank private String firstName;`
  - `@NotBlank private String lastName;`
  - `@Email private String email;`
  - `@NotNull private UserRole role;`

3.2.6. DTO: `UserDTO`

- **Attributes:**
  - `private Long id;`
  - `private String username;`
  - `private String email;`
  - `private String role;`
  - `private boolean active;`

3.3. Package: `com.heavyroute.core`

3.3.1. Class: `TransportRequestController`

Responsabilità: API per la gestione delle richieste da parte dei committenti.

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

`+ createRequest(dto: RequestDTO): ResponseEntity<RequestDTO>`
Crea una nuova richiesta.
  - **OCL Pre:** `dto.pickupDate > LocalDate.now()` (Data futura)
  - **OCL Post:** `result.body.status == RequestStatus.PENDING`

`+ getMyRequests(): ResponseEntity<List<RequestDTO»`
Restituisce lo storico.

3.3.2. Class: `TripManagementController`

Responsabilità: API per la gestione operativa dei viaggi (PL/TC).

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

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

Approva richiesta e genera viaggio.

- **OCL Pre (Security):** `currentUser.role == Role.PL`

`+ planResources(tripId: Long, dto: PlanningDTO): ResponseEntity<Void>`

Associa risorse al viaggio.

- **OCL Pre:** `dto.driverId <> null AND dto.vehiclePlate <> null`

3.3.3. Interface: `TripService`

Responsabilità: Facade della logica di business core.

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

`+ approveAndGenerateTrip(requestId: Long): Trip`

Transizione di stato e creazione viaggio.

- **OCL Pre:** `requestRepository.existsById(requestId)`
- **OCL Pre:** `request.status == RequestStatus.PENDING`
- **OCL Post:** `request.status == RequestStatus.APPROVED`
- **OCL Post:** `result.status == TripStatus.PLANNING`
- **Exception:** `IllegalStateException` se stato non valido.

`+ assignResources(tripId: Long, driverId: Long, plate: String): void`

Associa risorse.

- **OCL Pre:** `trip.status == TripStatus.PLANNING`
- **OCL Pre:** `resourceService.isDriverAvailable(driverId, trip.dates)`
- **OCL Pre:** `resourceService.isVehicleAvailable(plate, trip.dates)`

3.3.4. Entity: `TransportRequest`

Responsabilità: Modello persistente richiesta.

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

`- id: Long`

- -- originAddress: String
- -- destinationAddress: String
- -- status: RequestStatus
- -- load: LoadDetails

- **Invariants (OCL):**
  - context TransportRequest inv: self.originAddress <> self.destinationAddress
  - context TransportRequest inv: self.load.weightKg > 0

3.3.5. Entity: Trip

Responsabilità: Modello persistente viaggio.

- **Stereotype:** «Entity»
- **Attributes:**
  - -- id: Long
  - -- tripCode: String
  - -- status: TripStatus
  - -- driver: User
  - -- vehicle: Vehicle
  - -- route: Route
- **Invariants (OCL):**
  - context Trip inv: self.status == CONFIRMED implies (self.driver <> null and self.vehicle <> null)
  - context Trip inv: self.tripCode.matches("^TRP-\d{4}-\d+$")

3.3.6. Entity: Route

Responsabilità: Dettagli del percorso calcolato.

- **Stereotype:** «Entity»
- **Attributes:**
  - -- id: Long
  - -- distanceKm: Double
  - -- durationMinutes: Double
  - -- polyline: String

3.3.7. DTO: `RequestDTO`

- **Attributes:**
  - `@NotBlank private String origin;`
  - `@NotBlank private String destination;`
  - `@NotNull @Future private LocalDate pickupDate;`
  - `@Positive private Double weight;`

3.3.8. DTO: `TripDTO`

- **Attributes:**
  - `private String tripCode;`
  - `private String status;`
  - `private String driverName;`
  - `private String vehiclePlate;`

3.3.9. DTO: `PlanningDTO`

- **Attributes:**
  - `@NotNull private Long driverId;`
  - `@NotNull private String vehiclePlate;`

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: API REST per l'inventario e la viabilità.

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

`+ findAvailable(start: String, end: String): ResponseEntity<List<VehicleDTO>>`

Cerca veicoli liberi per un intervallo di tempo.

– **OCL Pre:** `start <> null AND end <> null`

– **OCL Pre:** `DateUtil.parse(start) < DateUtil.parse(end)`

`+ registerEvent(dto: RoadEventDTO): ResponseEntity<Void>`

Registra un nuovo evento stradale (es. cantiere).

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

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

3.5.2. Interface: `ResourceService`

Responsabilità: Logica di gestione flotta e viabilità.

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

  `+ findVehiclesByAvailability(start: LocalDateTime, end: LocalDateTime): List<Vehicle>`

  Esegue la query di disponibilità incrociando i calendari.

  - **OCL Post:** `result->forAll(v | v.status == VehicleStatus.ACTIVE)`
  - **OCL Post:** `result->forAll(v | !v.hasBookingIn(start, end))`

  `+ registerRoadConstraint(event: RoadEventDTO): void`

  Salva l'evento e notifica il Core Business per ricalcoli.

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

3.5.3. Entity: `Vehicle`

Responsabilità: Entità Veicolo (Inventario).

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

  – `- plateNumber: String` (PK)

  – `- model: String`

  – `- maxLoadCapacityKg: Double`

  – `- status: VehicleStatus`

- **Invariants (OCL):**

  – `context Vehicle inv: self.maxLoadCapacityKg > 0`

  – `context Vehicle inv: self.plateNumber.matches("^[A-Z]{2}\d{3}[A-Z]{2}$")`

3.5.4. Entity: `RoadEvent`

Responsabilità: Entità Evento Stradale.

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

  – `- id: Long`

- -- description: String
- -- location: GeoLocation
- -- severity: EventSeverity
- -- validFrom: LocalDateTime
- -- validTo: LocalDateTime

- **Invariants (OCL):**
  - context RoadEvent inv: self.validFrom <= self.validTo
  - context RoadEvent inv: self.location <> null

3.5.5. DTO: `VehicleDTO`

Responsabilità: Transfer Object per dettagli veicolo.

- **Attributes:**
  - -- plateNumber: String
  - -- model: String
  - -- capacity: Double
  - -- status: String

3.5.6. DTO: `RoadEventDTO`

Responsabilità: Transfer Object per creazione eventi.

- **Attributes:**
  - -- description: String
  - -- latitude: Double
  - -- longitude: Double
  - -- severity: String
  - -- validFrom: LocalDateTime
  - -- validTo: LocalDateTime

3.6. Package: `com.heavyroute.notification`

3.6.1. Interface: `NotificationService`

Responsabilità: Facade asincrona per l'invio multicanale di notifiche.

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

  `+ sendEmail(recipient: String, template: String, vars: Map<String, Object>): void`
  Accoda una richiesta di invio e-mail.
  - **OCL Pre:** `recipient <> null AND recipient.matches("^.+@.+\..+$")`
  - **OCL Pre:** `template <> null AND templateExists(template)`

  `+ sendPushNotification(userId: Long, title: String, body: String): void`
  Accoda una richiesta di notifica push.
  - **OCL Pre:** `userId <> null`
  - **OCL Pre:** `title.length <= 100` (Limite provider)

3.6.2. Class: `EmailService`

Responsabilità: Implementazione concreta dell'invio tramite protocollo SMTP.

- **Stereotype:** `«Component»`
- **Attributes:**
  - `- mailSender: JavaMailSender`
  - `- templateEngine: TemplateEngine`

3.6.3. DTO: `NotificationRequest`

Responsabilità: Oggetto messaggio per la coda asincrona (Message Broker).

- **Attributes:**
  - `- type: NotificationType` (EMAIL, PUSH)
  - `- recipient: String`
  - `- payload: String`
  - `- createdAt: LocalDateTime`
- **Invariants (OCL):**
  - `context NotificationRequest inv: self.createdAt <= LocalDateTime.now()`

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: `GlobalExceptionHandler`

Responsabilità: Gestore centralizzato delle eccezioni che traduce gli errori applicativi in risposte HTTP standard (RFC 7807).

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

  `+ handleNotFound(ex: ResourceNotFoundException): ResponseEntity<ProblemDetail>`
  Restituisce HTTP 404.

  `+ handleBusinessError(ex: BusinessRuleException): ResponseEntity<ProblemDetail>`
  Restituisce HTTP 409 Conflict o 422 Unprocessable Entity.

  `+ handleValidation(ex: MethodArgumentNotValidException): ResponseEntity<ProblemDetail>`
  Restituisce HTTP 400 Bad Request con dettagli sui campi invalidi.

3.7.2. Class: `BaseEntity` (Abstract)

Responsabilità: Superclasse per tutte le entità JPA, fornisce ID e campi di auditing automatico.

- **Stereotype:** `«MappedSuperclass»`
- **Attributes:**
  - `# id: Long` (Primary Key)
  - `# createdAt: LocalDateTime`
  - `# updatedAt: LocalDateTime`
  - `# version: Integer` (Optimistic Locking)

3.7.3. Value Object: `GeoLocation`

Responsabilità: Oggetto immutabile che incapsula una coordinata geografica valida.

- **Stereotype:** `«Embeddable»`

- **Attributes:**
  - `+ latitude: Double`
  - `+ longitude: Double`
- **Invariants (OCL):**
  - `context GeoLocation inv: self.latitude >= -90.0 AND self.latitude <= 90.0`
  - `context GeoLocation inv: self.longitude >= -180.0 AND self.longitude <= 180.0`

3.7.4. Enum: `UserRole`

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

- **Values:** `CLIENT, DRIVER, PL (Pianificatore), TC (Coordinator), GA (Gestore Account)`