Università degli Studi di Napoli "Parthenope"
Facoltà di Scienze e Tecnologie
Corso di laurea in Informatica

Relazione progetto TaxiToGo Programmazione III

TaxiToGo

Docente
Angelo Ciaramella

Candidato
Emanuele Chiummo
0124002453

Anno Accademico 2023-2024

Indice

Introduzione

Il progetto proposto mira allo sviluppo di un'applicazione dedicata alla gestione efficiente di un servizio di taxi. L'obiettivo principale è fornire un sistema completo e intuitivo che possa essere utilizzato da amministratori, tassisti e clienti al fine di ottimizzare le operazioni legate al trasporto su strada. La piattaforma si baserà su tre ruoli chiave: amministratore, tassista e cliente, ciascuno con funzionalità specifiche atte a garantire un'esperienza d'uso fluida e soddisfacente.

Funzionalità dell'amministratore

L'amministratore avrà il compito di gestire la flotta di taxi e le tariffe delle corse. Tra le principali funzionalità a sua disposizione:

1. **Aggiunta di nuove auto:** L'amministratore potrà facilmente aggiungere nuove vetture alla flotta, garantendo un controllo efficace sulla disponibilità delle risorse.

2. **Aggiornamento delle tariffe:** Sarà possibile modificare e aggiornare le tariffe delle corse, consentendo una flessibilità nella gestione dei costi in base alle necessità e alle dinamiche di mercato.

3. **Visualizzazione delle statistiche:** L'amministratore potrà accedere a dettagliate statistiche riguardanti le tratte effettuate, fornendo un quadro completo delle attività svolte dalla flotta, attraverso una Dashboard.

4. **Gestione delle tratte:** L'amministratore potrà aggiungere nuove tratte, modificare o disattivare quelle esistenti.

5. **Gestione degli utenti:** L'amministratore potrà avere il pieno controllo sull'anagrafica andando ad inserire nuovi utenti, modificarli o impostarli inattivi.

Funzionalità del cliente

Il cliente avrà accesso a un'interfaccia user-friendly per usufruire del servizio di taxi in modo semplice e conveniente. Le principali funzionalità a disposizione del cliente includono:

1. **Prenotazione del taxi:** I clienti potranno prenotare un taxi attraverso due modalità:

    - **Interfaccia Web:** Compilando un form di richiesta e usufruendo del pagamento online.
    - **Prenotazione Email:** Inviando un email all'indirizzo: taxitogo2024@gmail.com e pagare in contanti al completamento della corsa.

Funzionalità del tassista

Il tassista utilizzerà l'applicazione per gestire le chiamate di intervento e semplificare il proprio lavoro. Le principali funzionalità a disposizione del tassista includono:

1. **Accettazione o rifiuto delle chiamate:** Il tassista potrà decidere se accettare o rifiutare una chiamata di intervento, garantendo una gestione efficiente delle proprie disponibilità.

Capitolo 1

Progettazione

Il mondo del trasporto urbano è un elemento cruciale per garantire la mobilità delle comunità, e la gestione efficiente dei servizi di taxi svolge un ruolo fondamentale in questo contesto. Nel quadro di questa dinamica, il progetto TaxiToGo si propone di rivoluzionare il sistema attuale di gestione dei taxi, introducendo una piattaforma avanzata e integrata.

Il panorama attuale del trasporto su strada può presentare inefficienze e limitazioni dovute a processi manuali, scarsa automazione e alla mancanza di strumenti tecnologici dedicati. TaxiToGo si pone come soluzione a questa sfida, offrendo un sistema completo che coinvolge amministratori, tassisti e clienti. L'obiettivo è fornire un'esperienza di servizio fluida, flessibile e all'avanguardia.

Questo documento esplorerà il sistema corrente, mettendo in luce le sue criticità e le opportunità di miglioramento. Successivamente, sarà presentato il sistema proposto, TaxiToGo, evidenziandone le caratteristiche chiave e gli impatti previsti sulle dinamiche del settore dei taxi. Attraverso un'analisi approfondita, cercheremo di dimostrare come TaxiToGo possa rappresentare una svolta significativa nell'ottimizzazione delle operazioni, garantendo una gestione più efficace delle flotte, una maggiore soddisfazione del cliente e una migliore organizzazione complessiva del servizio di taxi....

1.1 Sistema Corrente

Il sistema corrente, prima dell'introduzione di TaxiToGo, è caratterizzato da un processo di gestione dei taxi che spesso si basa su procedure manuali e sistemi tradizionali. L'amministrazione della flotta, la tarificazione delle corse e la gestione delle prenotazioni potrebbero risultare disorganizzate e richiedere un considerevole sforzo umano. La comunicazione tra clienti e tassisti potrebbe essere limitata, causando ritardi e inefficienze. Inoltre, la mancanza di un sistema automatizzato per la registrazione delle tratte e l'analisi delle prestazioni della flotta potrebbe rendere difficile l'ottimizzazione delle operazioni.

1.2 Sistema Proposto (TaxiToGo)

TaxiToGo è un'innovativa piattaforma che propone un sistema di gestione dei taxi completamente automatizzato e integrato. Le funzionalità avanzate offerte mirano a superare le limitazioni del sistema corrente, migliorando l'efficienza, la precisione e la soddisfazione del cliente.

Principali Caratteristiche del Sistema Proposto:

1. **Gestione Integrata della Flotta:** TaxiToGo consente all'amministratore di gestire la flotta in modo centralizzato, aggiungendo o rimuovendo veicoli, controllando le tratte e gestendo gli utenti con facilità.

2. **Tariffazione Dinamica:** Il sistema proposto permette una gestione flessibile delle tariffe, con possibilità di aggiornamenti in tempo reale in risposta alle dinamiche del mercato e alle esigenze aziendali.

3. **Dashboard Statistiche:** TaxiToGo fornisce una dettagliata dashboard statistica per l'amministratore, offrendo una panoramica completa delle prestazioni della flotta e delle tratte effettuate.

4. **Gestione delle Prenotazioni Cliente:** I clienti possono prenotare un taxi in modo conveniente tramite l'interfaccia web o via email, offrendo opzioni di pagamento flessibili.

5. **Gestione delle Chiamate del Tassista:** I tassisti possono accettare o rifiutare le chiamate di intervento, migliorando la gestione delle proprie disponibilità e ottimizzando il tempo di servizio.

6. **Gestione Utenti:** L'amministratore ha il pieno controllo sull'anagrafica degli utenti, potendo inserire nuovi utenti, modificarli o impostarli come inattivi.

7. **Interfaccia Utente Intuitiva:** L'interfaccia utente è progettata per essere intuitiva e user-friendly, garantendo una facile adozione da parte di amministratori, tassisti e clienti.

TaxiToGo, con il suo approccio moderno e tecnologico, si propone di rivoluzionare il settore dei taxi, migliorando l'efficienza operativa e offrendo un'esperienza di servizio ottimizzata per tutte le parti coinvolte.

1.3 Struttura Database

Il database è stato progettato per gestire in modo efficace le informazioni cruciali relative agli utenti, ai taxi, alle posizioni, alle tariffe, alle richieste e alle email. La sua struttura è rappresentata attraverso sei tabelle principali: "email", "taxi", "users", "rates", "locations" e "requests". Le interrelazioni tra queste tabelle sono fondamentali per garantire una gestione coerente delle informazioni e la corretta esecuzione delle operazioni del sistema.

Il sistema è stato realizzato attraverso la piattaforma MySQL Workbench versione 8.0, disegnando il diagramma relazione per poi far generare il db in automatico dal sistema.

1.3.1 Tabelle

Diagramma EE/R

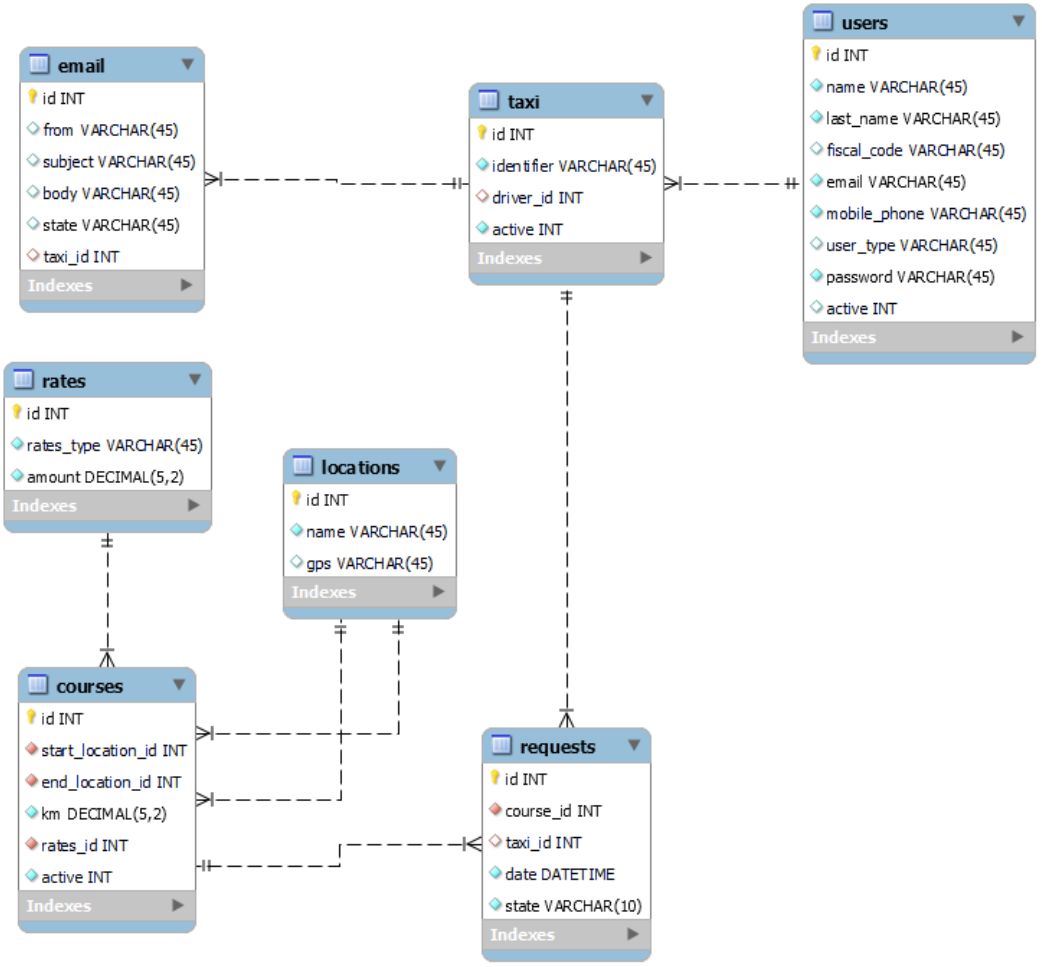


Tabella Email

La tabella "email" memorizza le informazioni relative alle email, con campi quali ID email, mittente, oggetto, corpo, stato e l'ID del taxi associato.

Tabella Taxi

La tabella "taxi" traccia i dati dei taxi, includendo campi per l'ID, l'identificatore, l'ID del conducente e lo stato attivo.

Tabella Users

La tabella "users" conserva le informazioni degli utenti, tra cui ID, nome, cognome, codice fiscale, email, numero di telefono mobile, tipo di utente, password e stato attivo.

Tabella Rates

La tabella "rates" contiene i dettagli delle tariffe, compresi campi per l'ID della tariffa, il tipo di tariffa e la quantità.

Tabella Locations

La tabella "locations" registra informazioni sulle posizioni, con campi per l'ID della posizione, il nome e le coordinate GPS.

Tabella Requests

La tabella "requests" conserva dettagli sulle richieste degli utenti, includendo campi per l'ID della richiesta, l'ID del corso associato, l'ID del taxi associato, la data della richiesta e lo stato.

1.3.2 Relazioni tra le tabelle

Le relazioni tra le tabelle sono essenziali per garantire l'integrità referenziale dei dati all'interno del database TaxiToGo. Ogni relazione è basata su attributi chiave che collegano le tabelle in modo che le informazioni siano correlate e coerenti tra di loro.

Relazione tra "email" e "taxi"

La tabella "email" è collegata alla tabella "taxi" attraverso l'attributo "taxi_id". Questo campo funge da chiave esterna nella tabella "email" e fa riferimento all'ID univoco di un taxi nella tabella "taxi". In termini pratici, questa relazione

consente di associare specifiche email a un taxi particolare, creando una traccia di comunicazioni correlate a ciascun veicolo.

Relazione tra "taxi" e "users"

La tabella "taxi" è connessa alla tabella "users" attraverso l'attributo "driver_id". Qui, "driver_id" funge da chiave esterna che si riferisce all'ID di un utente nella tabella "users". Questa relazione stabilisce un legame tra i dati dei taxi e i dati dei conducenti, consentendo di identificare chiaramente il conducente associato a ciascun taxi.

Relazione tra "taxi" e "requests"

La tabella "taxi" è ulteriormente collegata alla tabella "requests" tramite l'attributo "taxi_id". Questa relazione consente di associare richieste specifiche a un particolare taxi, consentendo una gestione accurata delle richieste di intervento da parte dei conducenti.

Relazione tra "rates" e "courses"

La tabella "rates" è connessa alla tabella "courses" attraverso l'attributo "rates_id". Questo collegamento facilita l'associazione di tariffe specifiche a ciascun corso, garantendo una registrazione dettagliata dei costi associati a ogni percorso effettuato.

Relazioni tra "courses" e "locations"

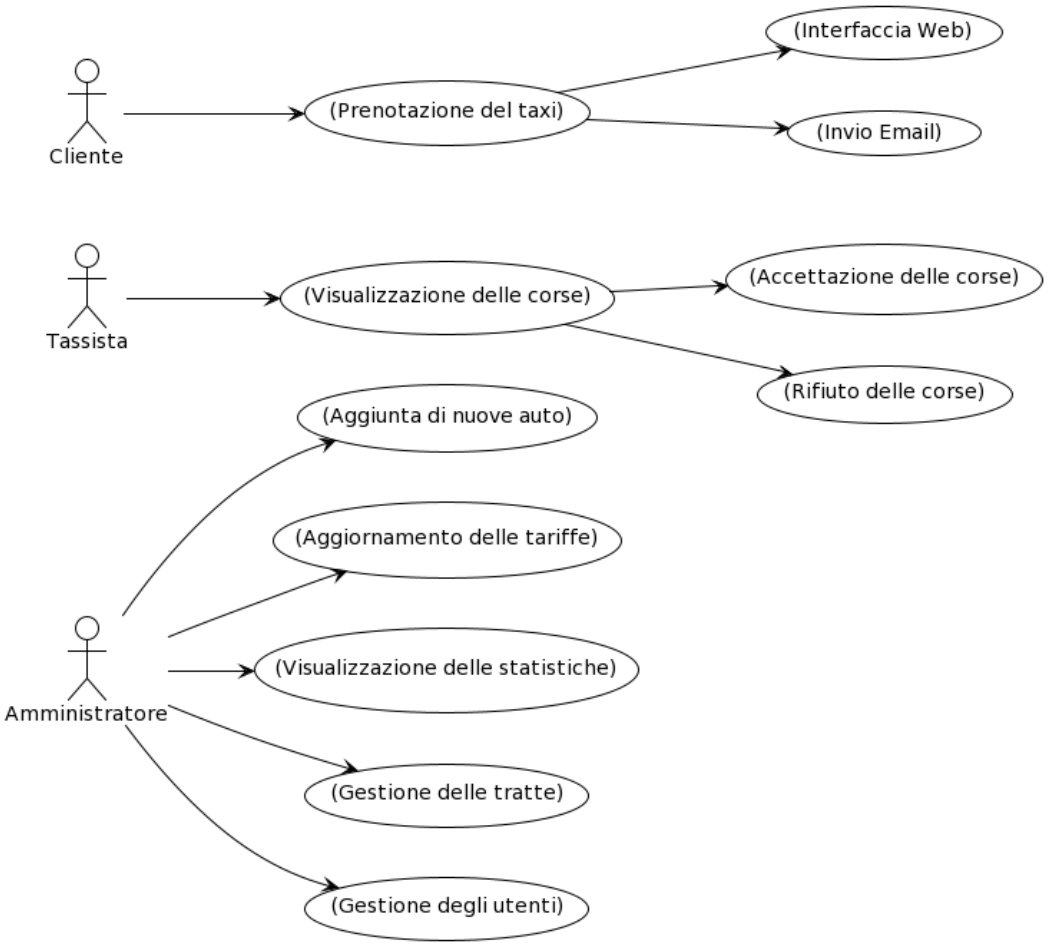
La tabella "courses" è collegata due volte alla tabella "locations". Prima attraverso l'attributo "start_location_id" e successivamente attraverso "end_location_id". Queste relazioni permettono di identificare chiaramente le posizioni di partenza e di arrivo associate a ciascun corso, creando un legame diretto tra i percorsi effettuati e le relative posizioni geografiche.

Relazione tra "courses" e "requests"

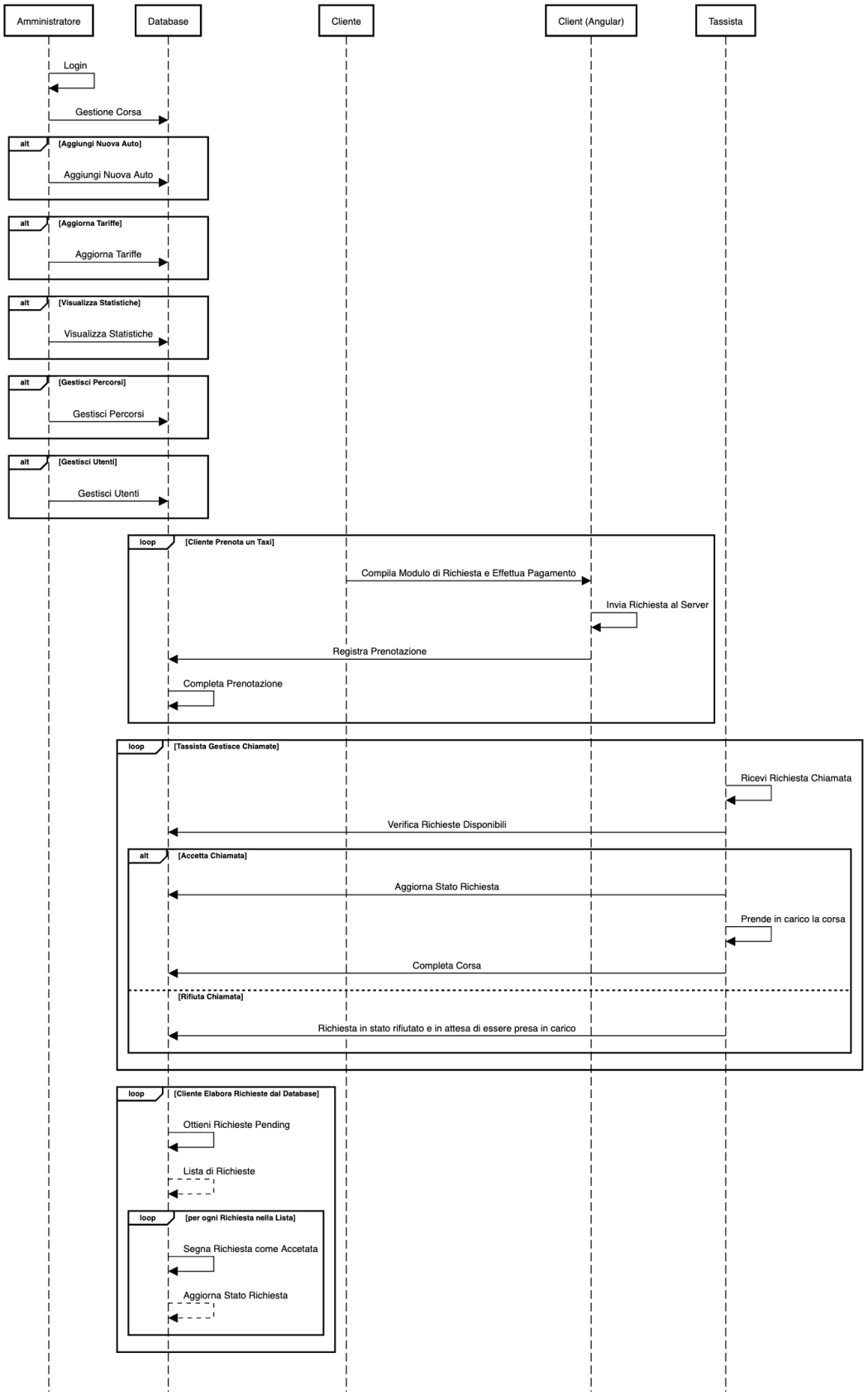
La tabella "courses" è infine collegata alla tabella "requests" mediante l'attributo "course_id". Questa relazione consente di associare ciascuna richiesta di intervento a un corso specifico, fornendo informazioni dettagliate sulla gestione delle corse richieste dagli utenti.

1.4 Progettazione Sviluppo

1.4.1 Use Case



1.4.2 Sequence Diagram



1.4.3 AuthController

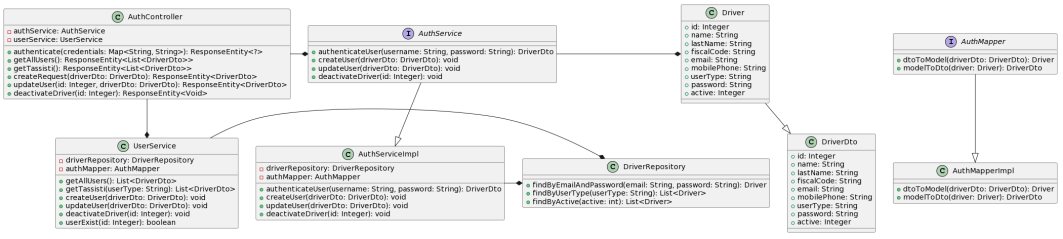
Il sistema illustrato nel diagramma UML rappresenta un'applicazione basata su Spring Boot per la gestione di utenti, in particolare tassisti, tramite un sistema di autenticazione.

Il nucleo dell'applicazione ruota attorno all'entità Driver, che rappresenta un utente all'interno del sistema. Questa entità è divisa in due parti principali: Driver stesso e il suo corrispondente DriverDto, un oggetto di trasferimento dati utilizzato per comunicare con il sistema.

Il AuthController agisce da interfaccia per la gestione delle richieste HTTP relative all'autenticazione e alla gestione degli utenti. Comunica con il AuthService per le operazioni di autenticazione e con il UserService per le operazioni di gestione degli utenti.
Il AuthService è responsabile dell'autenticazione degli utenti, utilizzando il DriverRepository per accedere ai dati degli utenti nel database. Inoltre, offre funzionalità per la creazione, aggiornamento e disattivazione degli utenti, delegando tali operazioni al UserService.

Il UserService, d'altra parte, gestisce le operazioni relative alla gestione degli utenti. Utilizza sia il DriverRepository che il AuthMapper per accedere ai dati e per la conversione tra l'entità Driver e il suo DTO corrispondente.

L'AuthMapper è responsabile della conversione bidirezionale tra Driver e DriverDto, agevolando il trasferimento dei dati tra il sistema e l'interfaccia utente.



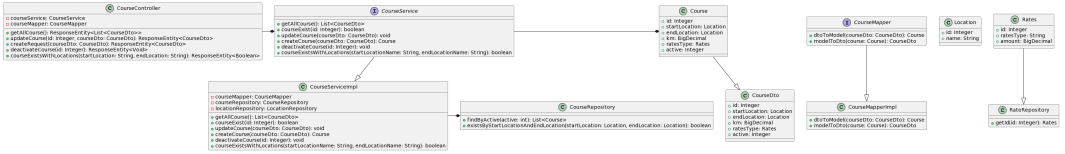
1.4.4 CourseController

Le principali entità coinvolte includono le classi Course e CourseDto, che rappresentano rispettivamente le informazioni dettagliate di una corsa e il suo oggetto di trasferimento dati.

La classe Course contiene dati essenziali come le posizioni di partenza e arrivo, la distanza percorsa, il tipo di tariffa applicato e lo stato attivo o disattivo della corsa. La classe CourseDto funge da veicolo per trasferire informazioni tra i diversi livelli dell'applicazione, garantendo una gestione efficiente delle operazioni di creazione, aggiornamento e visualizzazione delle corse.

Il servizio di gestione delle corse è implementato attraverso l'interfaccia CourseService, la cui implementazione (CourseServiceImpl) fornisce le funzionalità operative. Tra queste operazioni vi sono la restituzione di tutte le corse attive, la verifica dell'esistenza di una corsa, l'aggiornamento e la creazione di nuove corse, la disattivazione di corse esistenti e la verifica dell'esistenza di corse con specifiche posizioni di partenza e arrivo.

Affiancato a ciò, il sistema coinvolge la classe Rates e il suo repository RateRepository, che gestiscono le informazioni relative alle tariffe applicate alle corse. Questo permette una gestione flessibile delle tariffe, mantenendo l'integrità e la coerenza delle informazioni.



1.4.5 LocationController

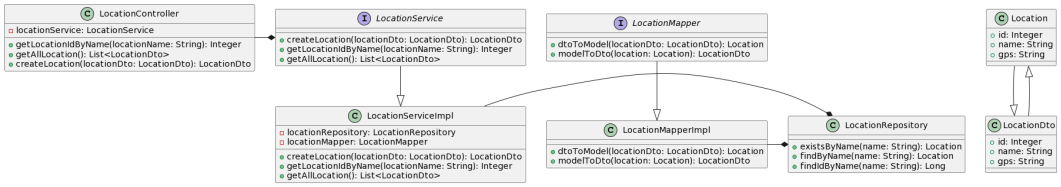
L'entità principale nel modulo di gestione delle posizioni è rappresentata dalla classe Location, che contiene informazioni cruciali come l'identificatore (id), il nome (name), e le coordinate GPS (gps). La classe LocationDto è una rappresentazione di trasmissione dei dati che consente la comunicazione tra l'applicazione client e il backend.

L'interfaccia LocationMapper e la sua implementazione LocationMapperImpl svolgono un ruolo fondamentale nella conversione bidirezionale tra le istanze della classe Location e i relativi oggetti di trasmissione dati (LocationDto). Ciò facilita l'integrazione fluida dei dati tra le diverse componenti del sistema.

Il servizio principale, LocationService, è implementato da LocationServiceImpl. Questo fornisce operazioni essenziali per la gestione delle posizioni, tra cui la creazione di nuove posizioni, l'ottenimento dell'identificatore di una posizione dato il suo nome e l'ottenimento di tutte le posizioni presenti nel sistema.

La classe LocationRepository è responsabile dell'interazione con il database per le operazioni relative alle posizioni. Queste operazioni includono la verifica dell'esistenza di una posizione per nome, la ricerca di una posizione dato il suo nome e l'ottenimento dell'identificatore di una posizione dato il nome.

Infine, il LocationController gestisce le richieste HTTP relative alle posizioni. Fornisce endpoint per ottenere l'identificatore di una posizione dato il suo nome, recuperare tutte le posizioni presenti nel sistema e creare nuove posizioni.



1.4.6 RatesController

La classe centrale è rappresentata da Rates, che incarna le informazioni fondamentali relative alle tariffe del servizio. All'interno di questa classe, troviamo attributi significativi come l'identificatore (id), automaticamente generato per garantire univocità, il tipo di tariffa (ratesType), e l'importo (amount).

Accanto a questa classe principale, abbiamo RateDto, che agisce come un oggetto di trasferimento dati. Questo modello di dati è progettato per agevolare lo scambio di informazioni tra il servizio delle tariffe e il controller.

Inoltre, il diagramma include la classe RateMapper e la sua implementazione RateMapperImpl. Queste componenti sono responsabili della conversione bidirezionale tra oggetti Rates e RateDto.

Per quanto riguarda la persistenza dei dati, il diagramma include anche la classe RateRepository. Essa fornisce un'interfaccia per l'accesso ai dati relativi alle tariffe, sottolineando l'adozione di un approccio basato su repository per la gestione delle operazioni di archiviazione e recupero.

Infine, il servizio delle tariffe è implementato da RateServiceImpl, che utilizza un RateMapper per convertire e gestire le informazioni tra il repository e il controller. La presenza del controller RatesController completa il quadro, fornendo un'interfaccia per l'esposizione delle informazioni sulle tariffe attraverso l'API del sistema.

1.4.7 RequestController

Questa classe rappresenta una richiesta nel sistema, contenente informazioni cruciali come l'identificatore (id), il percorso (course), il taxi assegnato (taxi), la data (date) e lo stato corrente (state). La relazione con la classe Course e Taxi evidenzia l'associazione con le informazioni di percorso e conducente.

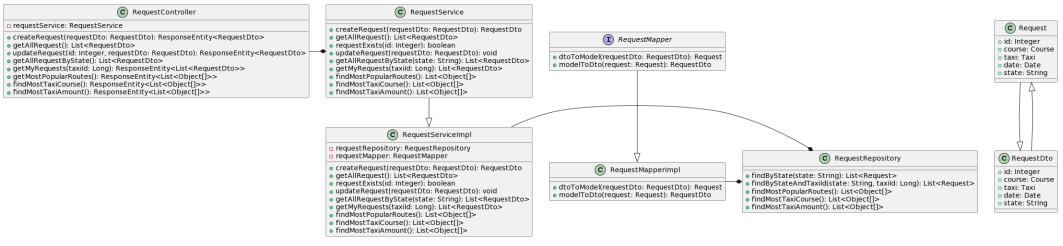
RequestDto: Un oggetto di trasferimento dati che riflette la struttura della classe Request, facilitando la comunicazione tra il servizio e il controller.

RequestMapper e RequestMapperImpl: Queste interfacce e implementazioni forniscono metodi per la conversione bidirezionale tra oggetti Request e RequestDto, garantendo una gestione efficiente della trasformazione dei dati.

RequestRepository: Questo repository fornisce un'interfaccia per l'accesso ai dati relativi alle richieste. Include metodi per recuperare richieste in base allo stato, al conducente e per analizzare le richieste più popolari.

RequestService e RequestServiceImpl: Questi componenti gestiscono la logica aziendale legata alle richieste, inclusa la creazione, l'aggiornamento e il recupero delle richieste. Utilizzano il RequestRepository e il RequestMapper per accedere e gestire i dati.

RequestController: Il controller espone i vari endpoint per gestire le richieste, inclusi metodi per creare, recuperare, aggiornare e analizzare le richieste. Comunica con il RequestService per gestire le operazioni.



1.4.8 TaxiController

Questa classe rappresenta un taxi nel sistema, contenente informazioni cruciali come l'identificatore (id), l'identificatore univoco del taxi (identifier), il conducente assegnato (driver), e lo stato di attività (active).

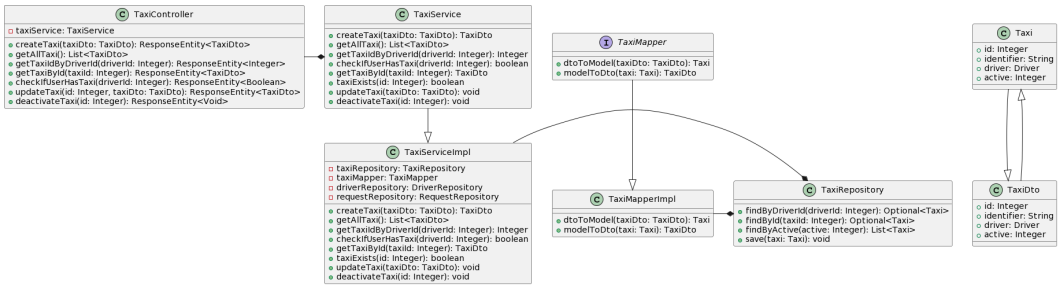
TaxiDto: Un oggetto di trasferimento dati che riflette la struttura della classe Taxi, facilitando la comunicazione tra il servizio e il controller.

TaxiMapper e TaxiMapperImpl: Queste interfacce e implementazioni forniscono metodi per la conversione bidirezionale tra oggetti Taxi e TaxiDto, garantendo una gestione efficiente della trasformazione dei dati.

TaxiRepository: Questo repository fornisce un'interfaccia per l'accesso ai dati relativi ai taxi. Include metodi per recuperare taxi in base al conducente, all'identificatore del taxi e allo stato di attività.

TaxiService e TaxiServiceImpl: Questi componenti gestiscono la logica legata ai taxi, inclusa la creazione, l'aggiornamento e il recupero dei taxi. Utilizzano il TaxiRepository e il TaxiMapper per accedere e gestire i dati.

TaxiController: Il controller espone vari endpoint per gestire i taxi, inclusi metodi per creare, recuperare, aggiornare e disattivare i taxi. Comunica con il TaxiService per gestire le operazioni.



Capitolo 2

Sviluppo

2.1 Framework e Software utilizzati

Nell'ambito dello sviluppo del progetto, sono stati impiegati diversi strumenti e framework. Questi strumenti sono stati selezionati per le loro caratteristiche, funzionalità e capacità di facilitare lo sviluppo del software, nonchè i più diffusi nei contesti lavorativi enterprise.

2.1.1 Software Database

MySQL Workbench è uno strumento di progettazione e amministrazione di database MySQL che fornisce un'interfaccia grafica per creare, modificare e gestire schemi di database. La versione 8.0 di MySQL Workbench è stata utilizzata per la progettazione del database, la gestione degli schemi e l'esecuzione di query SQL.

2.1.2 Framework Back-End

Spring Boot è un framework per lo sviluppo di applicazioni Java basato su Spring Framework, progettato per semplificare il processo di creazione di applicazioni Java enterprise. Spring Boot semplifica la configurazione dell'applicazione, riducendo la necessità di configurazioni manuali e fornendo funzionalità come l'iniezione di dipendenze e la gestione delle transazioni.

Il tutto è stato implementato in Spring Tool Suite (STS) un ambiente di sviluppo integrato basato su Eclipse, progettato per facilitare lo sviluppo di applicazioni Spring.

2.1.3 Framework Front-End

Angular è un framework di sviluppo front-end ampiamente utilizzato per la creazione di applicazioni web dinamiche e interattive. La versione 17.0.10 di Angular è stata scelta per sfruttare le ultime funzionalità e miglioramenti del framework. Angular offre un'architettura basata su componenti, agevolando la modularità e la manutenzione del codice. Inoltre, fornisce un'ampia gamma di strumenti per la gestione dello stato dell'applicazione, la gestione delle route e la manipolazione del DOM.

TypeScript è un linguaggio di programmazione sviluppato da Microsoft che estende JavaScript aggiungendo tipizzazione statica. Nel contesto di Angular, TypeScript è la scelta predefinita per lo sviluppo, fornendo un sistema di tipi che migliora la robustezza e la manutenibilità del codice.

Il tutto è stato implementato utilizzando Visual Studio Code come editor.

2.2 Design Pattern

Il progetto è stato sviluppato seguendo i principi fondamentali dell'ingegneria del software e integrando vari design pattern al fine di migliorare la struttura, la manutenibilità e la flessibilità del sistema. L'applicazione è stata realizzata utilizzando il framework Spring Boot, il quale adotta ampiamente i concetti di Inversion of Control (IoC) e injection of dependencies per facilitare lo sviluppo.

Inversion of Control

IoC sta per "Inversion of Control" (Inversione di Controllo). È un principio di progettazione del software in cui il controllo del flusso dell'applicazione viene invertito rispetto al normale flusso di esecuzione. Invece che il programmatore

controlli il flusso dell'applicazione, il controllo viene delegato a un framework o un contenitore che gestisce la sequenza di chiamate.

Un esempio concreto di IoC in Spring è l'uso di annotazioni come @Autowired. Questa annotazione viene utilizzata per l'iniezione automatica delle dipendenze. Invece di creare manualmente le istanze degli oggetti dipendenti, il framework Spring si occupa di fornire automaticamente le dipendenze necessarie.

2.2.1 Sigleton Pattern

Il Singleton Pattern è utilizzato nella classe DriverRepository, TaxiRepository, CourseRepository, LocationRepository, e RateRepository. Assicura che una sola istanza di ciascuna di queste classi esista nell'applicazione, fornendo un'unica porta d'accesso a tali istanze. Questo design pattern è adatto per i repository, in quanto garantisce che ci sia una sola istanza di ciascun repository e che essa sia condivisa da tutti i componenti che ne necessitano.

2.2.2 Factory Method Pattern

il Factory Method Pattern è utilizzato nella classe RequestService. Il metodo createRequest funge da "factory method" per la creazione di nuovi oggetti Request. Questo design pattern separa la logica di creazione degli oggetti dalla logica principale del servizio, facilitando l'estensione e la personalizzazione della creazione degli oggetti.

2.2.3 Model-View-Controller

Utilizzato all' interno dei controller del progetto. Dove il controller gestisce le richieste HTTP, chiama il servizio appropriato e restituisce una risposta.

2.2.4 Repository Pattern

Utilizzo i repository per implementare le interfacce che estendono JPARepository, separando la logica di accesso ai dati.

2.2.5 Mapper Pattern

Per gestire la conversione tra oggetti DTO e oggetti di modello. Pattern di mappatura che aiuta a mantenere separati i modelli di dominio dai modelli di presentazione.

2.2.6 Observer

Utilizzato per la creazione del server email, la classe emailListener implementa l'interfaccia ≪MessageCountListener≫, EmailListener in questo caso osserva i cambiamenti nel conteggio dei messaggi nella cartella IMAP.

2.2.7 Strategy

Utilizzato per la creazione del server email, nel metodo extractTextFromMultipart di EmailListener il codice varia in base al tipo di contenuto (plain text, html o multipart).

2.2.8 Dependency Injection

Utilizzato in molte classi del progetto, utilizzando le annotation @Autowired, per iniettare automaticamnte l'istanza del servizio quando viene instanziata l'implementazione.

2.2.9 DTO (Data transfer Object) Pattern

Sono presenti diverse classi DTO come DriverDto, TaxiDto, RequestDto, ecc., utilizzate per trasferire dati tra il livello di persistenza dei dati e il livello di presentazione. Inoltre, sono presenti mapper dedicati (DriverMapper, TaxiMapper, ecc.) per convertire tra entità JPA e DTO. Questo design pattern aiuta a separare il modello di dominio (entità JPA) dalla logica di presentazione, migliorando la modularità e la manutenibilità del codice.

Bibliografia

[1] Documentazione ufficiale SpringBoot
https://spring.io/

[2] Documentazione ufficiale Angular
https://angular.io/

[3] Implementazione Server Email
http://tinyurl.com/4umjryne