Object-Oriented Software Design

a.a.2018-2019


# *PEDAGGIO AUTOSTRADALE*

Gruppo composto da

| NOME | COGNOME | MATRICOLA |
|---|---|---|
| Cimoroni | Marco | 251369 |
| Di Egidio | Laura | 248831 |
| Gentile | Fabio | 248809 |

## A. REQUISITI

Lo scopo del Sistema è quello di fornire un software capace di calcolare il pedaggio autostradale a seconda del percorso svolto. Il sistema inoltre deve fornire anche meccanismi di calcolo differenti a seconda del tipo di calcolo messo a disposizione.

L'utente in base alla normativa stabilita dall'amministratore ha la possibilità di calcolare il percorso effettuato o che vorrà effettuare semplicemente inserendo la sua targa ed il percorso desiderato.

Il sistema deve inoltre fornire un'interfaccia grafica per l'utente e per l'amministratore, il quale dovrà gestire operazioni CRUD riguardanti il casello.

### - Requisiti funzionali

Utente: Conoscenza della tariffa dal casello di entrata al casello d'uscita secondo la norma selezionata dall'amministratore(Norma vigente).

Amministratore: gestione dei caselli (Aggiunta ed eliminazione) e aggiorna la normativa.

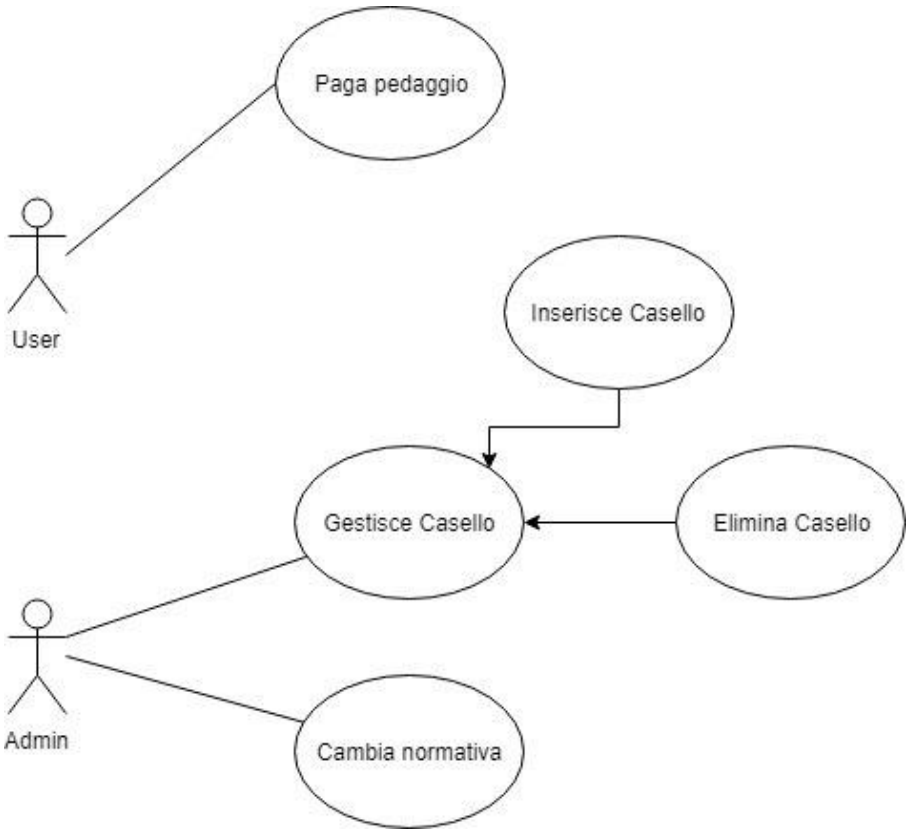
### - Requisiti non funzionali

Il sistema deve essere scalabile

Il sistema deve essere svolto nell' ambiente Java

Il sistema deve essere affidabile

USE CASE DIAGRAM



# B. ARCHITETTURA

## Obiettivi di design

Il principale obiettivo di design del sistema riguardante i requisiti non funzionali è la scalabilità. Il sistema deve essere infatti in grado di poter gestire eventuali normative che porteranno a modifiche all'interno del sistema. Per ora ci è sembrato opportuno mettere a discrezione dell'amministratore la scelta di quale normativa applicare per far calcolare il percorso all'utente.

## Strategie adottate per la costruzione del sistema

All'interno del package application.front.fxml troviamo tutte quelle che sono le viste messe a disposizione per l'utente. In questo componente è stata utilizzata la tecnologia JavaFX. Abbiamo ritenuto utile adottare il pattern MVC per una buona "Separation of Concern" gestendo quelli che sono gli input dell' utente e la parte statica del sistema. La componente Model in application.model fa uso delle Properties consentendo quindi di avere una sincronizzazione tra dati in memoria con quelli presenti sull'interfaccia grafica. Per la persistenza dei dati abbiamo scelto l'utilizzo di un database MYSQL.

## Descrizione dell'architettura

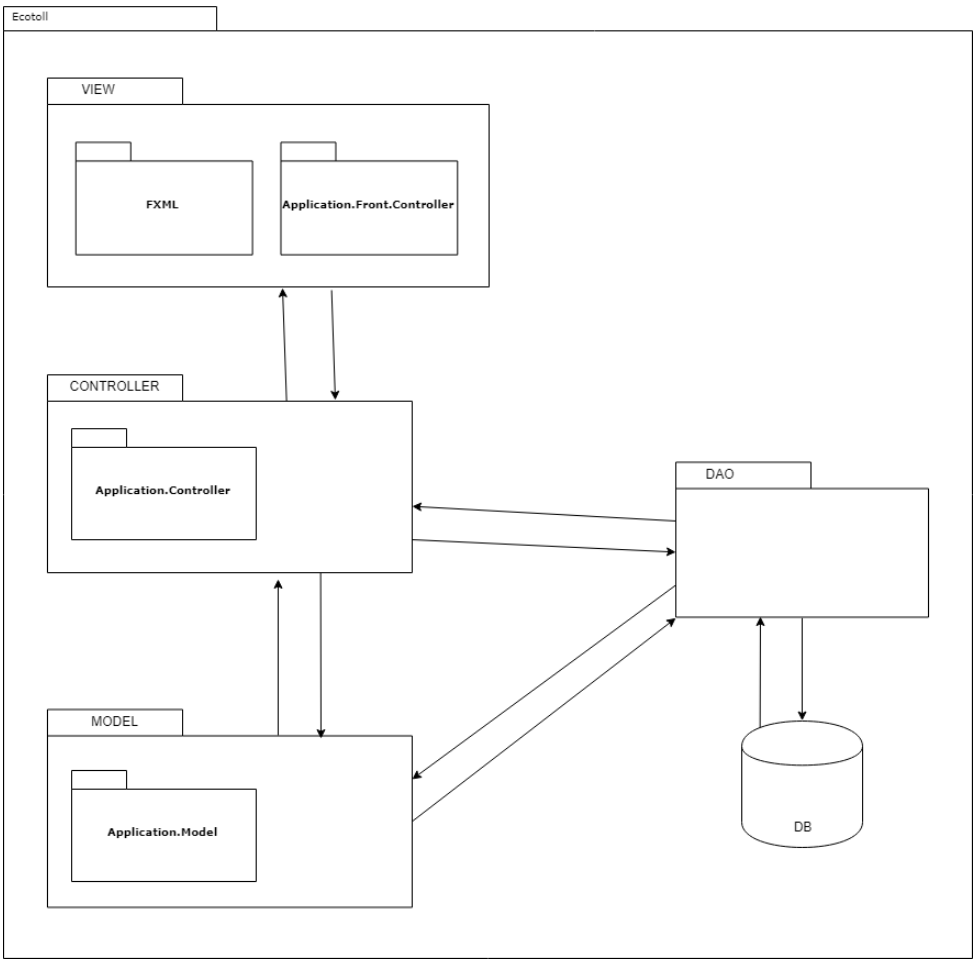
L'architettura prevede tre livelli:

Al primo livello application c'è la classe main della nostra applicazione

Al secondo livello abbiamo il .controller, .front, .dao e .model

Il .front prevede un terzo livello dove sono riportati .Fxml ed i relativi .controller

## PACKAGE DIAGRAM




## Design Patterns

Ciò che ha guidato il nostro progetto è stato sostanzialmente dividere quello che è la parte statica del progetto da ciò che invece nel tempo potrebbe mutare in quanto soggetto a cambiamenti.

I principali Design Patterns utilizzati sono:

DAO FACTORY: Il pattern Data Access Object (DAO) è un pattern architetturale utilizzato per separare i servizi di business dell'application processing layer dalle operazioni di accesso ai dati del data management layer. Il Data Access Object nasconde completamente i dettagli dell'interazione con la sorgente dati. L'interfaccia esposta dal DAO al client non cambia quando l'implementazione dell'origine dati sottostante cambia, e questo consente al DAO di adattarsi a diversi schemi di archiviazione senza dover modificare nulla sugli altri layer.
Il DAO implementa le operazioni CRUD (Create, Read, Update, Delete).

MVC: Model-View-Controller (MVC) è un pattern utilizzato in programmazione per dividere il codice in blocchi dalle funzionalità ben distinte. L'applicazione deve separare i componenti software che

implementano il modello delle funzionalità di business, dai componenti che implementano la logica di presentazione e di controllo che utilizzano tali funzionalità. Vengono quindi definiti tre tipologie di componenti che soddisfano tali requisiti:

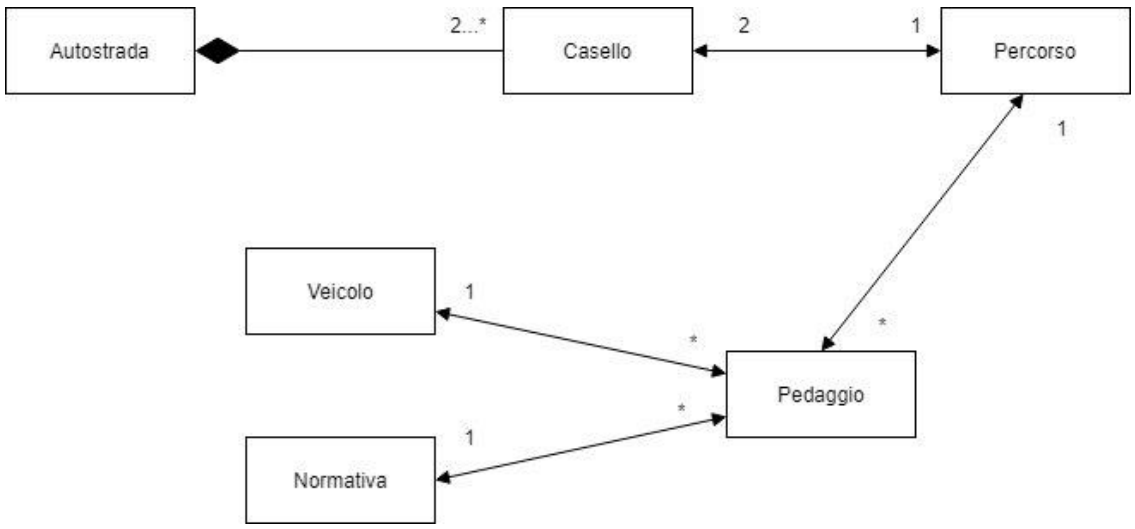
il Model, View, Controller.In particolare:

- Model : Si occupa solo dello stato e della logica (business logic) dell'applicazione. Non si occupa di come lo stato venga rappresentato all'utente o su come sia ricevuto l'input dell'utente (interfaccia).
- View: si occupa di visualizzare i dati all'utente e gestisce l'interazione fra quest'ultimo e l'infrastruttura sottostante.
- Controller: riceve i comandi dell'utente attraverso la View e reagisce eseguendo delle operazioni che possono interessare il Model e che portano generalmente ad un cambiamento di stato della View.

Observer Pattern: Al fine di gestire il Data Binding è stato utilizzando l'observer pattern, sfruttando l'implementazione fornita da JavaFX.

## C. CLASSI E INTERFACCE

### Domain model




Le entità che sono state utili durante lo sviluppo del sistema sono Autostrada, Casello, Normativa, Pedaggio, Percorso, Veicolo.

L'entità Autostrada composta dal suo nome, codice Autostrada (PK) , 'da' e 'a' che rispettivamente indicano l'inizio e la fine dell'autostrada, la lunghezza e la tariffa per km.

Un' autostrada è formata da caselli i quali hanno un proprio nome, un id (PK) , un'altezza e ovviamente l'id dell'autostrada.

La Normativa è utile nel momento in cui dobbiamo andare a calcolare il percorso. L'Entità normativa indica appunto la normativa vigente in quel momento impostata dall'amministratore. Nel momento in cui l'amministratore aggiornerà la normativa i calcoli verranno effettuati con la nuova scelta.

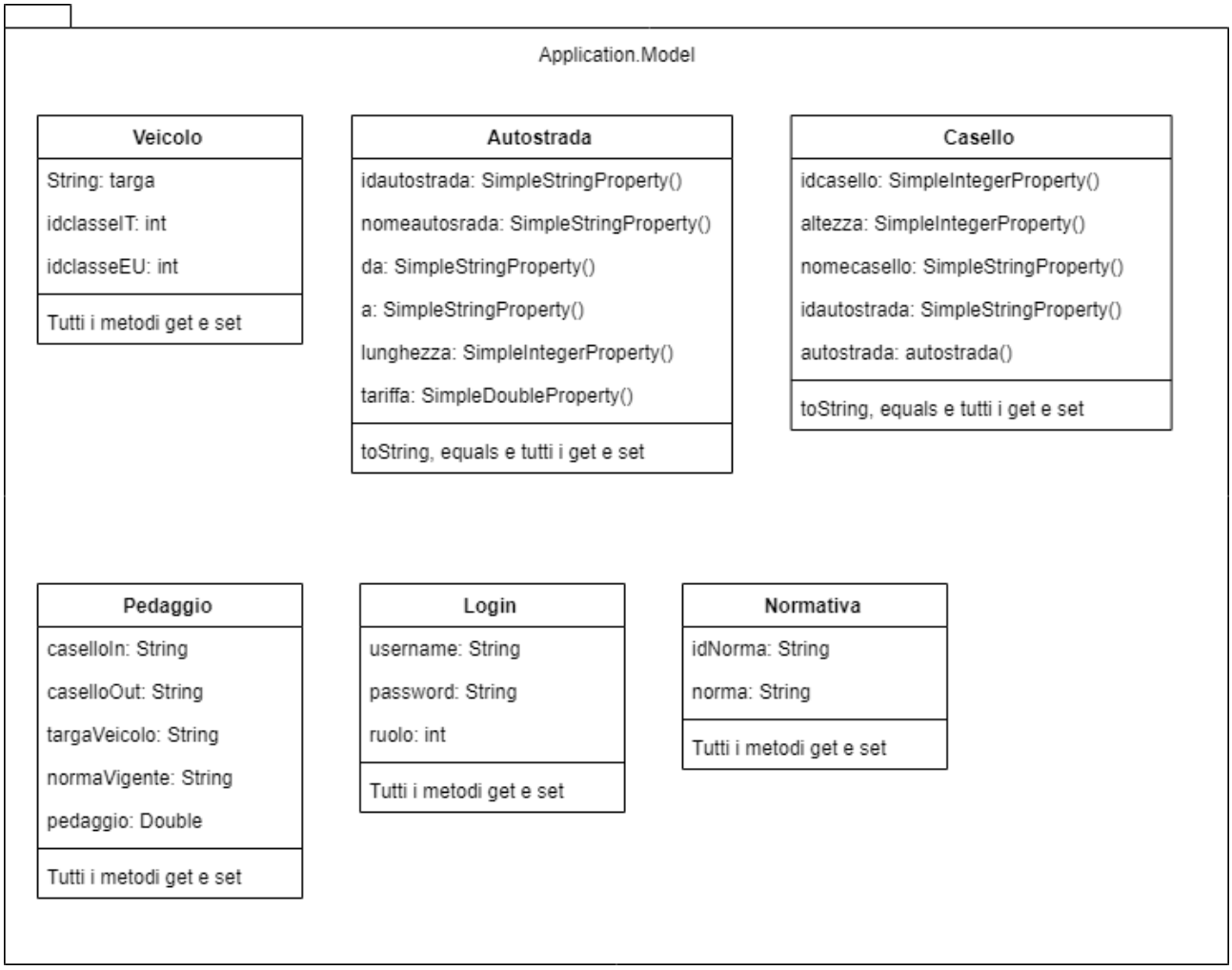
L'entità Veicolo invece è stata scelta per avere la possibilità di inserire la propria targa del veicolo (Considerando ovviamente una situazione in cui il sistema ha registrato tutte le targhe in circolazione) e, da lì, risalire dunque al tipo e alla classe.

Il percorso è stato scelto  per avere la possibilità di gestire caselli in ingresso ed uscita di autostrade differenti (non implementato)

L' entità pedaggio registra il casello in ingresso, in uscita, la targa del veicolo, il costo e la normativa salvandolo sulla Base Dati.

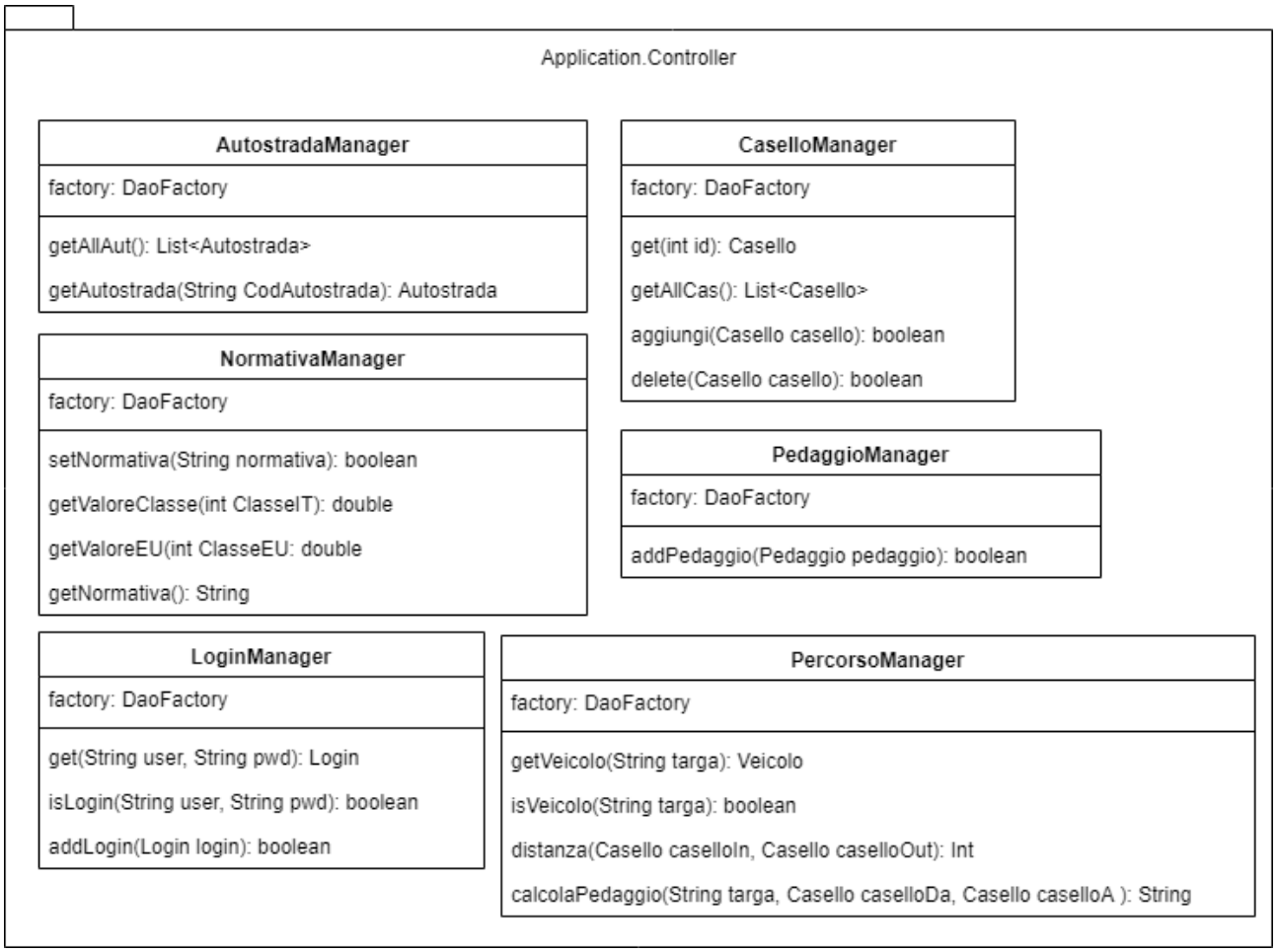
# Descrizione dettagli di design

## Model Package




Il package Applicatio.Model si occupa solo dello stato e della logica (business logic) dell'applicazione. Non si occupa di come lo stato venga rappresentato all'utente o su come sia ricevuto l'input dell'utente (interfaccia).

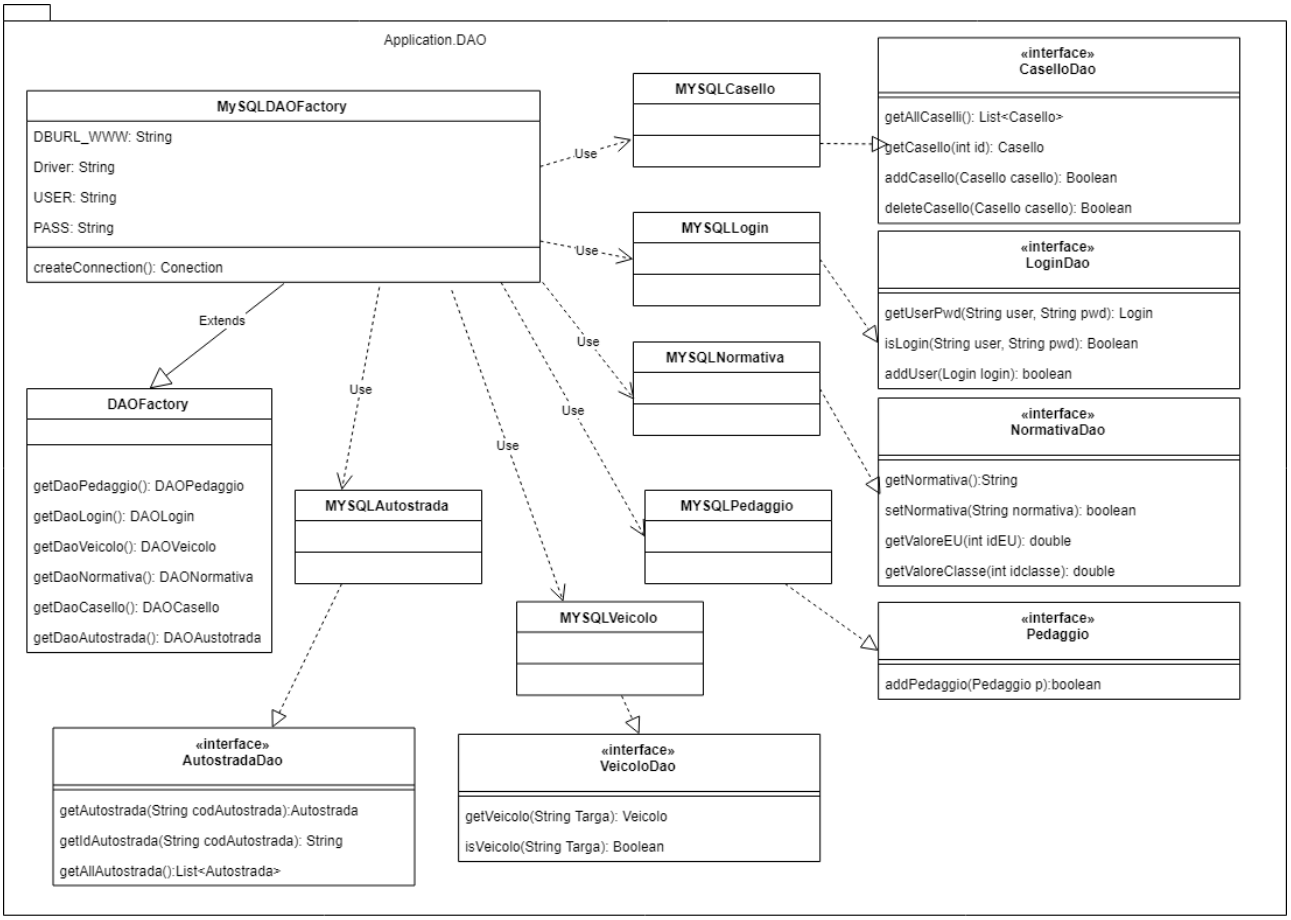
## Manager Package




Il package Application.controller fornisce alla View del nostro sistema tutte le funzionalità necessarie all'utilizzo del sistema. Rappresenta il Controller nel pattern MVC.

Il package application.front.controller e application.front.fxml sono la View del nostro sistema. In fxml ci sono tutti i form di javaFX e nel controller i relativi controller dei file fxml. Le classi controller JavaFX effettuano chiamate alle componenti del Controller che andranno ad eseguire le funzionalità richieste dall'utente ed eventualmente a modificare le componenti del Model.

## DAO PACKAGE




Per isolare il package relativo alla persistenza dei dati è stato scelto il pattern architetturale DAO (Data Access Object). I vari oggetti DAO ,che rappresentano un' entità tabellare e forniscono gli opportuni metodi per effettuare query, sono generati attraverso l'utilizzo di una Factory diminuendo così l'accoppiamento tra le componenti DAO e Manager.