

# Servizio web per la gestione di noleggi auto

*Studente:* Vasiliu Rares  
*Matricola:* 300795

*Docente:* Ferretti Stefano  
*Corso:* Basi Di Dati

Progetto Sessione Estiva A.A. 2021/2022

# Indice

<b>1</b>	<b>Analisi dei requisiti</b>	<b>1</b>
1.1	Specifica dei requisiti della base di dati . . . . .	1
1.2	Specifica della piattaforma . . . . .	1
1.2.1	Profilo . . . . .	2
1.2.2	Amministratore . . . . .	2
1.2.3	Manager . . . . .	3
1.2.4	Addetto . . . . .	4
1.2.5	Socio . . . . .	5
<b>2</b>	<b>Progettazione</b>	<b>7</b>
2.1	Progettazione concettuale . . . . .	7
2.1.1	Manager, addetti e soci . . . . .	7
2.1.2	Filiali . . . . .	7
2.1.3	Veicoli . . . . .	8
2.1.4	Noleggio . . . . .	8
2.1.5	Diagramma E/R completo . . . . .	9
2.2	Progettazione logica . . . . .	9
2.2.1	Eliminazione delle generalizzazioni . . . . .	9
2.2.2	Traduzione delle entità . . . . .	9
2.3	Progettazione fisica (MySQL) . . . . .	10
<b>3</b>	<b>Sviluppo piattaforma</b>	<b>13</b>
3.1	Progettazione . . . . .	13
3.1.1	Routing . . . . .	13
3.1.2	Controller . . . . .	13
3.1.3	Model . . . . .	14
3.1.4	View . . . . .	14
3.1.5	Database . . . . .	14
3.1.6	Utenti . . . . .	14
3.1.7	Grafici . . . . .	15
3.2	Autenticazione . . . . .	15
3.2.1	LoginController::show() . . . . .	15
3.2.2	LoginController::login() . . . . .	16
3.2.3	LoginController::logout() . . . . .	17
3.3	Dashboard . . . . .	17
3.3.1	Admin . . . . .	17
3.3.2	Manager . . . . .	19
3.3.3	Addetto . . . . .	20
3.3.4	Socio . . . . .	22
3.4	Profilo . . . . .	23
3.5	Operazioni . . . . .	25

---

<b>4</b>	<b>Gestione dei noleggi</b>	<b>28</b>
4.1	Check-in . . . . .	28
4.1.1	Selezione socio . . . . .	28
4.1.2	Selezione veicolo . . . . .	28
4.1.3	Creazione noleggio . . . . .	29
4.2	Check-out . . . . .	30
4.2.1	Selezione socio . . . . .	30
4.2.2	Selezione noleggio . . . . .	31
4.2.3	Check-out . . . . .	32
<b>5</b>	<b>Conclusione</b>	<b>33</b>

# Capitolo 1

## Analisi dei requisiti

Si vuole sviluppare una piattaforma per la società di autonoleggi “Gas’N’GO”. Lo scopo principale della piattaforma è consentire la gestione dei noleggi che avvengono nelle varie filiali della società.

### 1.1 Specifica dei requisiti della base di dati

Ogni filiale, identificata dal proprio nome, viene gestita da un manager e almeno un addetto al noleggio, identificati entrambi dal codice fiscale. Per ogni filiale si vuole memorizzare quando è stata aperta, dove si trova, il recapito telefonico e da chi viene gestita. Degli addetti che lavorano ad una filiale si vuole memorizzare la data di assunzione e la data di fine contratto.

Ogni filiale ha diversi veicoli, identificati dalla targa, che possono essere disponibili per il noleggio, attualmente in noleggio, o non più disponibili. Per ogni veicolo si vuole memorizzare marca, modello, colore e costo giornaliero, che deve essere maggiore o uguale a cinque.

I veicoli vengono noleggiati dai soci della società, identificati dal codice fiscale. Per ogni socio si vuole memorizzare la data di iscrizione.

Di ogni noleggio, identificato da un codice, si vuole memorizzare in che filiale è stato fatto, l’addetto che ha svolto il check-in, compresa data e ora dell’operazione, l’addetto che ha svolto il check-out, compresa la data e l’ora dell’operazione, il socio, il veicolo noleggiato e il costo finale del noleggio.

Per ogni manager, addetto e socio si vuole inoltre memorizzare nome, cognome, data di nascita, domicilio e numero di telefono.

Per l’accesso alla piattaforma si vogliono memorizzare (per ogni manager, addetto e socio) delle credenziali che consistono in un’email ed una password.

### 1.2 Specifica della piattaforma

La piattaforma è composto principalmente da tre parti:

- **Dashboard:** contiene una serie di statistiche e un grafico in base all’utente autenticato
- **Profilo:** informazioni dell’utente autenticato
- **Operazioni** varie di *inserimento*, *lettura*, *modifica* e *rimozione*

### 1.2.1 Profilo

Permette di visualizzare le informazioni dell'utente e di modificare le proprie credenziali di accesso.

Nota: l'amministratore non ha informazioni da visualizzare.

### 1.2.2 Amministratore

L'amministratore si occupa di registrare le nuove filiali.

#### Statistiche

- a) Numero di filiali
- b) Numero di soci
- c) Numero di noleggi nel mese corrente
- d) Fatturato nel mese corrente

#### Grafico

Grafico a torta con le performance nel mese corrente delle filiali registrate.

#### Operazioni

- a) Inserire una nuova filiale
- b) Inserire un nuovo account per il manager
- c) Inserire un nuovo manager
- d) Visualizzare l'elenco delle filiali
  - Per ogni filiale nell'elenco, visualizzare
    - Nome della filiale
    - Recapito telefonico
    - Manager (nome, cognome)
    - Fatturato
  - Ordinato per fatturato
- e) Visualizzare per una filiale
  - Le informazioni della filiale
  - Le informazioni del manager
  - Il numero di:
    - Noleggi
    - Addetti
    - Veicolo
- f) Visualizzare l'elenco dei noleggi
  - Per ogni noleggio nell'elenco, visualizzare
    - Codice
    - Filiale (nome)

- Socio (nome, cognome)
    - Veicolo (marca, modello, colore)
    - Durata (in giorni)
    - Costo
  - Ordinato per il costo del noleggio
- g) Visualizzare l'elenco dei soci
- Per ogni socio nell'elenco, visualizzare
    - Email
    - Nome
    - Cognome
  - Ordinato per la data di iscrizione più recente e l'email

### 1.2.3 Manager

Il manager si occupa di assumere gli addetti e registrare i nuovi veicoli. Inoltre deve poter aggiornare i dati degli addetti e dei veicoli.

#### Statistiche

- a) Numero di veicoli disponibili
- b) Numero di addetti
- c) Numero di soci che hanno fatto almeno un noleggio nella filiale che gestisce
- d) Fatturato nel mese corrente

#### Grafico

Grafico a linee con il numero di check-in giornalieri nel mese corrente.

#### Operazioni

- a) Inserire un nuovo account per l'addetto
- b) Inserire un nuovo addetto
- c) Inserire un nuovo veicolo
- d) Visualizzare l'elenco degli addetti
  - Per ogni addetto nell'elenco, visualizzare
    - Email
    - Addetto (nome, cognome)
    - Data di assunzione
    - Numero di noleggi gestiti
  - Ordinato per numero di noleggi
- e) Visualizzare per un addetto le relative informazioni
- f) Modificare le informazioni di un addetto

- g) Visualizzare l'elenco dei noleggi
  - Per ogni noleggio nell'elenco, visualizzare
    - Codice
    - Socio (nome, cognome)
    - Targa
    - Veicolo (marca, modello, colore)
    - Data inizio noleggio
    - Data fine noleggio
    - Costo del noleggio
  - Ordinato per il costo del noleggio
- h) Visualizzare l'elenco dei veicoli
  - Per ogni veicolo nell'elenco, visualizzare
    - Targa
    - Veicolo (marca, modello, colore)
    - Stato
    - Numero di volte in cui è stato noleggiato
  - Ordinato per numero di noleggi
- i) Modificare lo stato di un veicolo
- j) Modificare le informazioni di un veicolo

### 1.2.4 Addetto

L'addetto si occupa della registrazione di un nuovo socio, della fase di check-in e check-out. Inoltre deve poter aggiornare i dati dei soci e terminare la loro iscrizione.

#### Statistiche

- a) Numero di soci aiutati nella fase di check-in o check-out
- b) Numero totale di noleggi in cui ha eseguito sia la fase di check-in che di check-out
- c) Numero totale check-in svolti
- d) Numero totale check-out svolti

#### Grafico

Grafico a linee con il numero di check-in e check-out registrati giornalmente nel mese corrente.

#### Operazioni

- a) Inserire un nuovo account per il socio
- b) Inserire un nuovo socio
- c) Visualizzare l'elenco dei soci
  - Per ogni socio nell'elenco, visualizzare

- Email
  - Nome
  - Cognome
  - Ordinato per la data di iscrizione più recente e l'email
- d) Modificare le informazioni di un socio
- e) Eliminare un socio
- f) Visualizzare l'elenco dei veicoli nella filiale
  - Per ogni veicolo nell'elenco, visualizzare
    - Targa
    - Marca
    - Modello
    - Colore
    - Costo giornaliero
  - Ordinato per marca, modello, colore e costo giornaliero
- g) Inserire un noleggio
- h) Inserire un check-in
- i) Visualizzare l'elenco dei soci con noleggi attivi
  - Per ogni socio nell'elenco, visualizzare
    - Email
    - Nome
    - Cognome
    - Numero di noleggi attivi
  - Ordinato per email
- j) Visualizzare l'elenco dei noleggi attivi di un socio
  - Per ogni noleggio nell'elenco, visualizzare
    - Codice
    - Targa
    - Veicolo (marca, modello, colore)
    - Data inizio noleggio
    - Costo del noleggio
  - Ordinato per la data inizio noleggio
- k) Inserire un check-out

### 1.2.5 Socio

#### Statistiche

- a) Numero di noleggi effettuati
- b) Numero di noleggi attivi
- c) Auto noleggiata più volte
- d) Spesa media



**Grafico**

Grafico a barre con il numero di noleggi effettuati mensilmente nell'anno corrente.

**Operazioni**

- a) Visualizzare l'elenco dei noleggi effettuati
- Per ogni noleggio nell'elenco, visualizzare
    - Codice
    - Filiale
    - Targa
    - Veicolo (marca, modello, colore)
    - Data inizio noleggio
    - Data fine noleggio
    - Costo del noleggio
  - Ordinato per la data di inizio noleggio (noleggio più recente)

## Capitolo 2

# Progettazione

### 2.1 Progettazione concettuale

#### 2.1.1 Manager, addetti e soci

Partiamo dalla modellazione dei vari utenti della piattaforma:

*Ogni filiale, identificata dal proprio nome, viene gestita da un **manager** e almeno un **addetto** al noleggio, identificati entrambi dal codice fiscale. [...] Degli **addetti** che lavorano ad una filiale si vuole memorizzare la data di assunzione e la data di fine contratto. [...] I veicoli vengono noleggiati dai **soci** della società, identificati dal codice fiscale. Per ogni **socio** si vuole memorizzare la data di iscrizione. [...] Per ogni **manager**, **addetto** e **socio** si vuole inoltre memorizzare nome, cognome, data di nascita, domicilio e numero di telefono. Per l'accesso alla piattaforma si vogliono memorizzare (per ogni **manager**, **addetto** e **socio**) delle credenziali che consistono in un'email ed una password.*

Manager, addetti e soci presentano attributi in comune, quali l'insieme di attributi che rappresentano le generalità, e le credenziali d'accesso, vengono quindi generalizzati come persone.

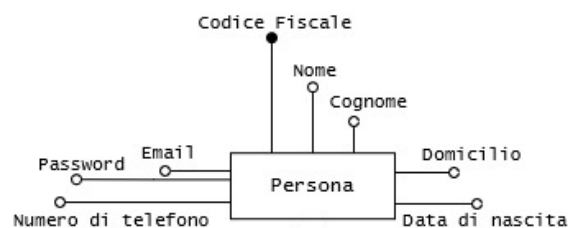


Figura 2.1: Entità persona



Figura 2.2: Entità manager, addetto e socio

#### 2.1.2 Filiali

*Ogni **filiale**, identificata dal proprio nome, viene gestita da un **manager** e almeno un **addetto** al noleggio [...] Per ogni **filiale** si vuole memorizzare quando è stata aperta, dove si trova, il recapito telefonico e da chi viene gestita.*

Sia manager che addetti sono in relazione con la filiale, vengono quindi generalizzati come impiegati.

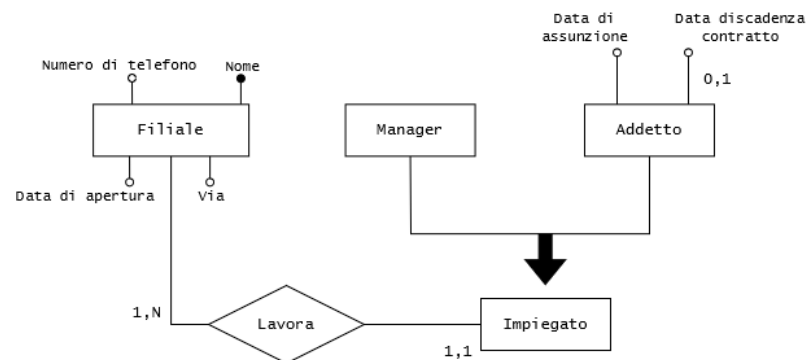


Figura 2.3: Entità filiale in relazione con manager e addetti

### 2.1.3 Veicoli

Ogni **filiale** ha diversi **veicoli**, identificato dalla targa, che possono essere disponibili per il noleggio, attualmente in noleggio, o non più disponibili. Per ogni **veicolo** si vuole memorizzare marca, modello, colore e costo giornaliero, che deve essere maggiore o uguale a cinque.

Ogni filiale deve avere almeno un veicolo che può essere noleggiato. L'attributo "Stato" nella relazione "Fornisce" permette di verificare la disponibilità del veicolo.

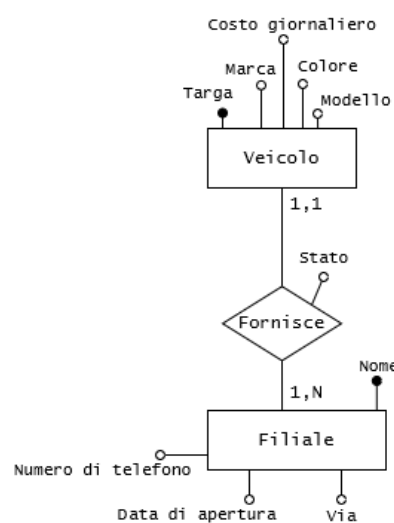


Figura 2.4: Entità filiale in relazione con i veicoli

### 2.1.4 Noleggio

I veicoli vengono noleggiati dai soci della società [...] Di ogni **noleggio**, identificato da un codice, si vuole memorizzare in che **filiale** è stato fatto, [...] il **socio**, il **veicolo** noleggiato e il costo finale del noleggio.

Al fine di evitare valori nulli, dato che viene calcolato durante la fase di checkout, il costo non è un attributo dell'entità noleggio.

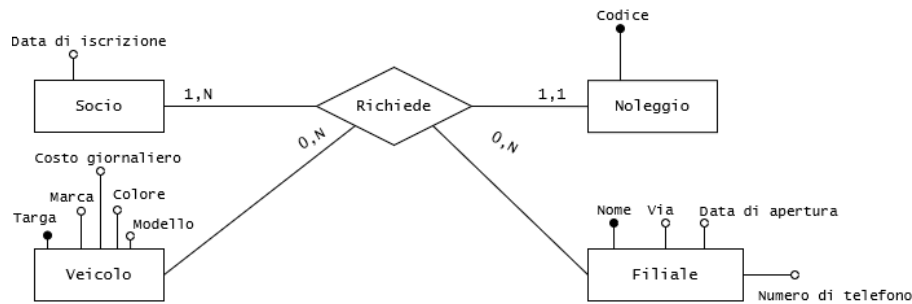


Figura 2.5: Entità noleggio

Di ogni **noleggio**, identificato da un codice, si vuole memorizzare [...] l'**addetto** che ha svolto il check-in, compresa data e ora dell'operazione, l'**addetto** che ha svolto il check-out, compresa la data e l'ora dell'operazione [...] e il costo finale del noleggio.

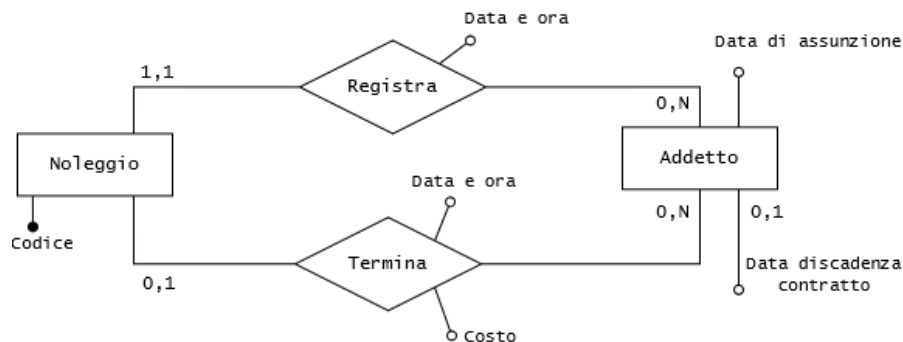


Figura 2.6: Relazioni check-in e check-out

### 2.1.5 Diagramma E/R completo

Figura 2.7

## 2.2 Progettazione logica

### 2.2.1 Eliminazione delle generalizzazioni

A fronte di una migliore gestione nella progettazione della piattaforma, gli attributi che riguardano le credenziali nella generalizzazione "persona" (Figura 2.1) vengono trasformati in una nuova entità, identificata da un codice, chiamata utente, dotata dell'attributo aggiuntivo ruolo per identificare la tipologia di utente. I restanti attributi vengono accorpati nelle entità figlie.

La generalizzazione "impiegato" (Figura 2.3) può essere rimossa senza problemi in quanto non presenta attributi e la tipologia viene gestita nell'entità utente.

### 2.2.2 Traduzione delle entità

In corsivo vengono rappresentate le chiavi esterne, mentre le chiavi primarie vengono sottolineate.

**Utente**(ID, email, password, ruolo)

**Socio**(*userID*, cf, nome, cognome, domicilio, dataDiNascita, numeroDiTelefono, dataDiIscrizione)

**Manager**(*userID*, cf, nome, cognome, domicilio, dataDiNascita, numeroDiTelefono)

**Filiale**(nome, dataDiApertura, via, numeroDiTelefono, *manager*)

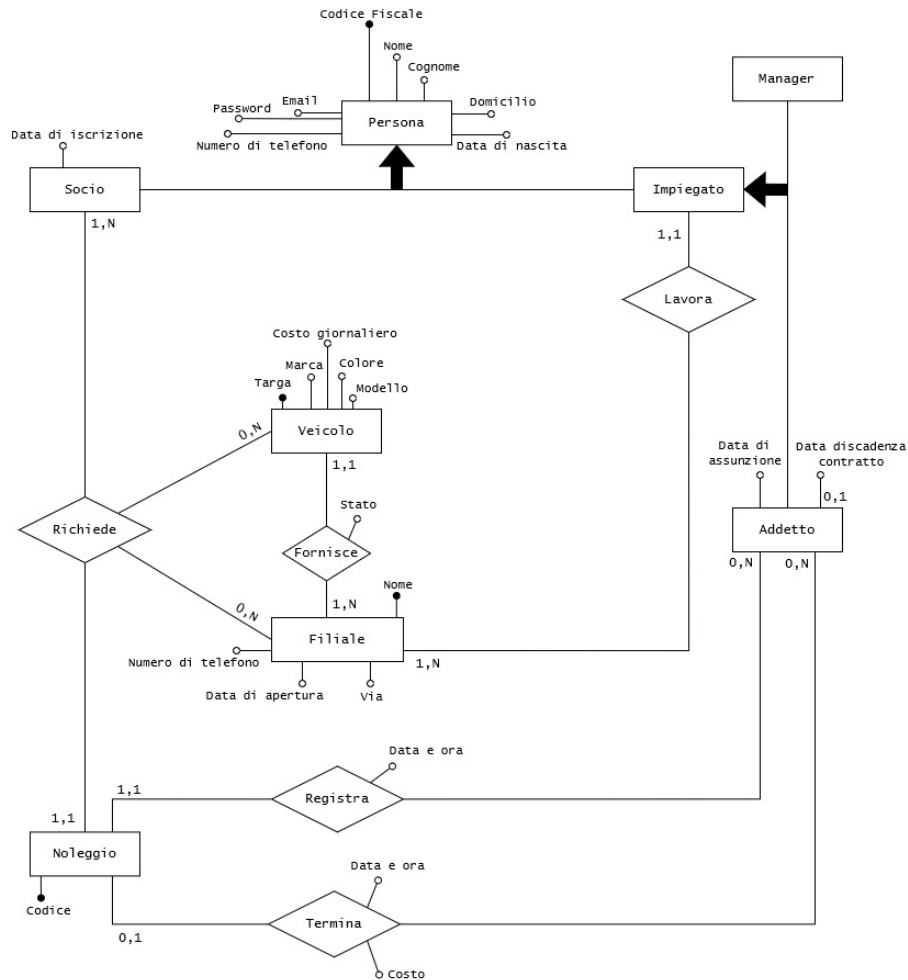


Figura 2.7: Diagramma E/R

**Addetto**(*userID*, *cf*, nome, cognome, domicilio, dataDiNascita, numeroDiTelefono, dataDiAssunzione, dataScadenzaContratto, *filiale*)

**Veicolo**(*targa*, targa, marca, modello, costoGiornaliero, colore, *filiale*)

**Noleggior**(*codiceNoleggior*, *veicolo*, *filiale*, *socio*)

**NoleggiorCheckIn**(*codice*, *addetto*, dataOperazione)

**NoleggiorCheckOut**(*codice*, *addetto*, dataOperazione, costo)

## 2.3 Progettazione fisica (MySQL)

```
CREATE DATABASE gas_n_go;
```

```
CREATE TABLE utente(
  ID int PRIMARY KEY AUTO_INCREMENT,
  email varchar(255) UNIQUE,
  password varchar(255) NOT NULL,
  ruolo int NOT NULL
);
```

```
CREATE TABLE socio(
  user_id int NOT NULL,
```

```
cf char(16) PRIMARY KEY,  
nome varchar(255) NOT NULL,  
cognome varchar(255) NOT NULL,  
data_di_nascita DATE NOT NULL,  
domicilio varchar(255) NOT NULL,  
numero_di_telefono varchar(15) NOT NULL,  
data_di_iscrizione DATE NOT NULL,  
FOREIGN KEY (user_id) REFERENCES utente (ID)  
ON DELETE CASCADE  
ON UPDATE CASCADE  
);
```

---

```
CREATE TABLE manager (  
  user_id int NOT NULL,  
  cf char(16) PRIMARY KEY,  
  nome varchar(255) NOT NULL,  
  cognome varchar(255) NOT NULL,  
  data_di_nascita DATE NOT NULL,  
  domicilio varchar(255) NOT NULL,  
  numero_di_telefono varchar(15) NOT NULL,  
  FOREIGN KEY (user_id) REFERENCES utente (ID)  
  ON DELETE CASCADE  
  ON UPDATE CASCADE  
);
```

---

```
CREATE TABLE filiale(  
  nome varchar(255) PRIMARY KEY,  
  data_di_apertura DATE NOT NULL,  
  via varchar(255) NOT NULL,  
  numero_di_telefono varchar(15) NOT NULL,  
  manager char(16),  
  FOREIGN KEY(manager) REFERENCES manager(cf)  
  ON DELETE SET NULL  
  ON UPDATE CASCADE  
);
```

---

```
CREATE TABLE addetto(  
  user_id int NOT NULL,  
  cf char(16) PRIMARY KEY,  
  nome varchar(255) NOT NULL,  
  cognome varchar(255) NOT NULL,  
  data_di_nascita DATE NOT NULL,  
  domicilio varchar(255) NOT NULL,  
  numero_di_telefono varchar(15) NOT NULL,  
  data_di_assunzione DATE NOT NULL,  
  data_scadenza_contratto DATE,  
  filiale varchar(255) NOT NULL,  
  FOREIGN KEY (user_id) REFERENCES utente (ID)  
  ON DELETE CASCADE  
  ON UPDATE CASCADE,  
  FOREIGN KEY (filiale) REFERENCES filiale (nome)  
  ON DELETE CASCADE  
  ON UPDATE CASCADE  
);
```

---

---

```
CREATE TABLE veicolo(  
    targa char(7) PRIMARY KEY,  
    marca varchar(255) NOT NULL,  
    modello varchar(255) NOT NULL,  
    colore varchar(255) NOT NULL,  
    costo_giornaliero int NOT NULL CHECK (costo_giornaliero >= 5),  
    stato varchar(255) NOT NULL,  
    filiale varchar(255) NOT NULL,  
    FOREIGN KEY (filiale) REFERENCES filiale (nome)  
    ON DELETE CASCADE  
    ON UPDATE CASCADE  
);
```

---

```
CREATE TABLE noleggio (  
    codice_noleggio int PRIMARY KEY AUTO_INCREMENT,  
    veicolo char(7) NOT NULL,  
    filiale varchar(255) NOT NULL,  
    socio char(16),  
    FOREIGN KEY (veicolo) REFERENCES veicolo (targa)  
    ON DELETE CASCADE  
    ON UPDATE CASCADE,  
    FOREIGN KEY (filiale) REFERENCES filiale (nome)  
    ON DELETE CASCADE  
    ON UPDATE CASCADE,  
    FOREIGN KEY (socio) REFERENCES socio (cf)  
    ON DELETE SET NULL  
    ON UPDATE CASCADE  
);
```

---

```
CREATE TABLE noleggiocheckin (  
    noleggio int(11) NOT NULL,  
    addetto char(16),  
    data_operazione datetime NOT NULL,  
    PRIMARY KEY (noleggio, addetto),  
    FOREIGN KEY (noleggio) REFERENCES noleggio (codice_noleggio)  
    ON DELETE CASCADE  
    ON UPDATE CASCADE,  
    FOREIGN KEY (addetto) REFERENCES addetto (cf)  
    ON DELETE CASCADE  
    ON UPDATE CASCADE  
);
```

---

```
CREATE TABLE noleggiocheckout (  
    noleggio int(11) NOT NULL,  
    addetto char(16),  
    data_operazione datetime NOT NULL,  
    costo int NOT NULL,  
    PRIMARY KEY (noleggio, addetto),  
    FOREIGN KEY (noleggio) REFERENCES noleggio (codice_noleggio)  
    ON DELETE CASCADE  
    ON UPDATE CASCADE,  
    FOREIGN KEY (addetto) REFERENCES addetto (cf)  
    ON DELETE CASCADE  
    ON UPDATE CASCADE  
);
```

---

## Capitolo 3

# Sviluppo piattaforma

### 3.1 Progettazione

La piattaforma verrà realizzata utilizzando il linguaggio PHP utilizzando come design pattern il classico Model-View-Control per sparare i dati, e i relativi accessi al DB (model), e la business logic (control) dalla parte di presentazione (view).

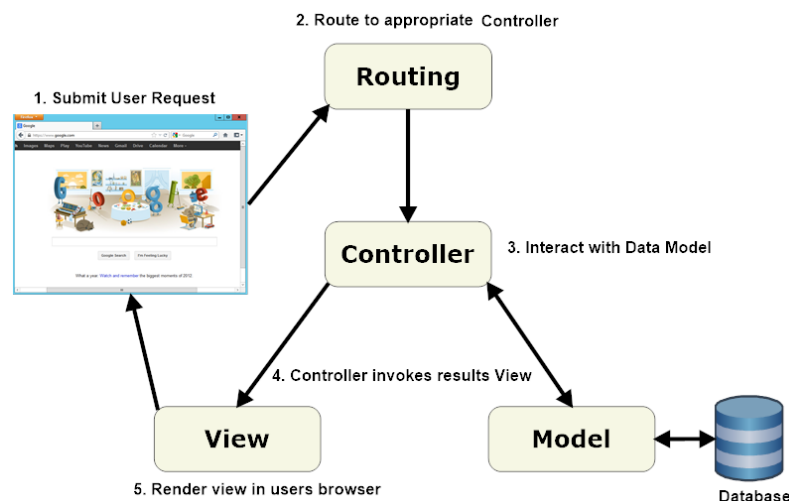


Figura 3.1: Schema funzionamento MVC

#### 3.1.1 Routing

Il routing viene implementato tramite un singolo file (es. *route.php*) ma sono presenti nella cartella *resources/* dei file PHP per ogni route che eseguono le funzioni dichiarate nei controller.

Nei file di route verranno fatti, inoltre, controlli di autenticazione e autorizzazione.

#### 3.1.2 Controller

Lo scopo dei controller è di:

- Validare form e richieste
- Recuperare dai modelli i dati che verranno utilizzati dai componenti nelle view, come ad esempio i link che conducono alle varie operazioni che devono essere visualizzati nella barra di navigazione oppure i campi che costituiscono un form



- Interagire con i modelli per l'aggiunta, modifica e rimozione di dati

Dato che alcune di queste operazioni richiedono la medesima logica in ogni controller è necessario creare una superclasse *Controller* che verrà ereditata dalla maggior parte dei controller.

### 3.1.3 Model

I modelli interagiscono con il database, il loro compito principale è quello di:

- Restituire i risultati delle query
- Inserire, modificare e eliminare i dati nel database

La maggior parte dei modelli rappresenta un'entità del modello E/R. La generalizzazione "persona" (Figura 2.1) verrà modellata creando una superclasse *Person*, ereditata poi dalle classi che rappresentano i manager, gli addetti e i soci, che avrà gli attributi che rappresentano i dati anagrafici. La parte riguardante le credenziali verrà modellata creando una classe *User* che permetterà di gestire l'autenticazione e l'autorizzazione nella piattaforma.

I modelli non devono essere necessariamente delle entità descritte nel modello E/R. Alcuni modelli rappresenteranno tabelle, campi dei form e grafici per una migliore gestione degli stessi.

Infine, i modelli che rappresentano gli utenti della piattaforma dovranno contenere i metodi per il recupero delle statistiche, delle operazioni consentite (che verranno visualizzati nella barra di navigazione sopra menzionata), del profilo e del grafico da mostrare nella dashboard.

### 3.1.4 View

I file di view sono costituiti principalmente da codice HTML con segmenti di codice PHP per la gestione dei dati ricevuti dal controller.

Dato che i dati, in view diverse, vengono rappresentati nello stesso formato è necessario lo sviluppo di componenti per uno sviluppo più fluido. Si pensi ad esempio alle rappresentazioni delle tabelle, esse hanno lo stesso stile e la stessa logica (ciclare e stampare i dati per ogni riga della tabella) in praticamente tutte le view.

### 3.1.5 Database

Per l'accesso al database da parte dei modelli avverrà tramite la classe *Database* che conterrà i metodi per interagire con esso. Nella cartella in cui risiede questa classe sarà presente, inoltre, il file di configurazione per l'effettivo accesso al database.

### 3.1.6 Utenti

La tipologia di utente verrà rappresentata da un numero intero, dove: 0 rappresenta l'admin, 1 rappresenta il manager, 2 rappresenta l'addetto e 3 rappresenta il socio.

L'account admin viene inserito manualmente, con password "12345678":

---

```
INSERT INTO utente
(email, password, ruolo)
VALUES ('admin@a.com',
'$2y$10$bSTUQiWGsZ/lgCtNhPt9MundIRYTVn03XypBOPiFJMjiZo0W5bz8y', 0);
```

---

### 3.1.7 Grafici

Per rappresentare i grafici, oltre alla creazione di un apposito modello, verrà utilizzata la libreria JavaScript "*Chart.js*". Le implementazioni di tale libreria si trovano nella cartella *resource/js/*.

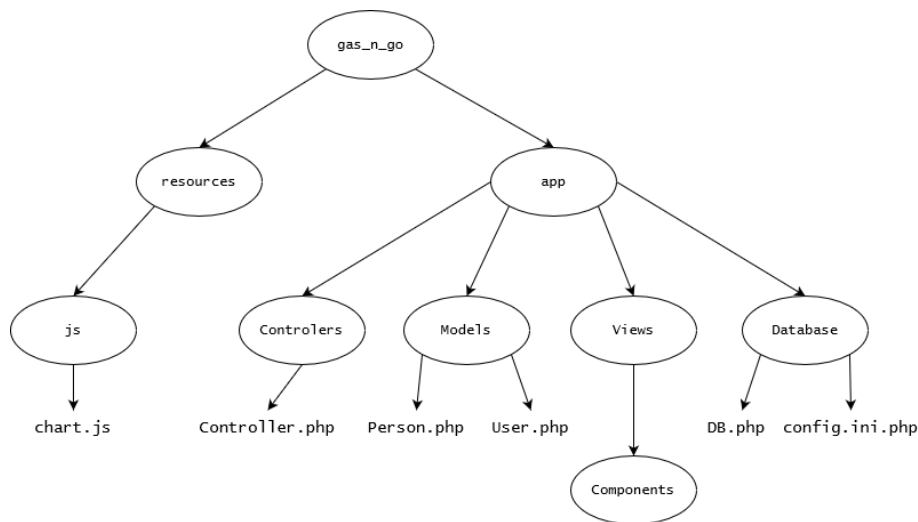


Figura 3.2: Gestione filesystem del progetto

## 3.2 Autenticazione

Il primo passo per accedere alla piattaforma è autenticarsi ad essa. Un utente non autenticato deve essere reindirizzato alla pagina di login. Per fare ciò ogni route include il codice *auth.inc.php* che verifica se è presente la variabile di sessione *user*, in caso:

- **Positivo:** crea la variabile *\$auth* contenente i dati dell'utente autenticato
- **Negativo:** l'utente viene reindirizzato alla pagina di login.

La route *login.php*, al contrario, non deve essere visibile dall'utente autenticato. Viene fatto quindi il controllo inverso. Se la richiesta è valida, viene importata la classe *LoginController* e, essendo la richiesta di tipo GET, viene eseguito il metodo statico *show()*.

### 3.2.1 LoginController::show()

Il metodo *show* deve:

- Prendere i campi email e password
- Passarli alla view di login

Per fare ciò è presente nella classe *User* il metodo statico *getFields()* che ritorna un array contenente due elementi di tipo *Field*. La classe *Field* modella i campi del form che sono composti dagli attributi HTML:

- **type:** viene rappresentato dall'attributo della classe *\$type*
- **name:** viene rappresentato dall'attributo della classe *\$name*
- **placeholder:** viene rappresentato dall'attributo della classe *\$data*
- **required:** viene rappresentato dall'attributo della classe *\$isRequired*

- **value**: viene rappresentato dall'attributo della classe *\$value*

Nel caso del login, dove *getFields()* è definita come:

---

```
public static function getFields()
{
    return array(
        'email' => new Field('email', 'email', 'Email', true),
        'password' => new Field('password', 'password', 'Password', true)
    );
}
```

---

otterremo:

---

```
<input type="email" name="email" id="email" required placeholder="Email" />
<input type="password" name="password" id="password" required
    placeholder="Password" />
```

---

che verrà visualizzato come:

The image shows a web form titled "Accedi". It consists of two text input fields stacked vertically. The first field is labeled "Email" and the second is labeled "Password". Both fields have a light gray border and placeholder text. Below these fields is a solid green button with the word "Login" in white text.

Figura 3.3: resources/login.php

L'attributo *\$value* viene utilizzato per l'aggiornamento dei campi impostandolo con il valore corrente presente nella base di dati. Di fatto, nel costruttore, viene inizializzato come valore nullo e ignorato dal componente che si occupa del render del form.

### 3.2.2 LoginController::login()

Una volta inserite le credenziali, ed eseguito il submit del form, verrà eseguito il metodo *LoginController::login()*.

Come prima cosa il metodo:

- Controlla se i dati provengono da un form tramite il metodo *checkFormSubmit* ereditato dalla classe *Controller*
- Controlla la validità dei dati nei campi del form tramite il metodo *checkFields* ereditato dalla classe *Controller*

Se i controlli falliscono l'utente viene reindirizzato al login.

Una volta verificata la validità del form vengono letti i dati dell'utente dal database tramite il metodo statico *read(\$email)* presente nel modello *User* che esegue la query:

---

```
SELECT *  
FROM utente  
WHERE email = :email  
LIMIT 1
```

---

e restituisce un oggetto di tipo *User*.

Successivamente viene verificata la password tramite il metodo *passwordVerify()* della classe *User*.

Infine, se l'utente è un addetto è necessario controllare se ha un contratto di lavoro valido (non scaduto):

- Viene eseguita la funzione *hasRole()* della classe *User* restituisce un oggetto di tipo *Employee* a seguito dell'esecuzione del metodo statico *Employee::read(\$this->id)*
- Viene eseguita la funzione *hasContractExpired()* della classe *Employee* che controlla se la data di scadenza del contratto è maggiore della data odierna

Se il login è andato a buon fine l'oggetto di tipo *User* viene salvato nella variabile di sessione *\$\_SESSION['user']*.

### 3.2.3 LoginController::logout()

Il logout consiste semplicemente nel distruggere la variabile *\$\_SESSION['user']*.

## 3.3 Dashboard

L'utente correttamente autenticato viene reindirizzato alla pagina *index.php* contenente la dashboard. La route di tale pagina esegue il metodo *DashboardController::show(\$auth)*, dove *\$auth* contiene l'oggetto *User* presente nella variabile di sessione inizializzato nella fase di login.

Il metodo *show* si occupa principalmente di ottenere i dati riguardanti le statistiche e il grafico. Per fare ciò esegue i metodi *getStats()* e *getChart()* presenti in ogni modello che rappresenta un utente.

### 3.3.1 Admin

#### Statistiche

Il metodo *getStats()* nella classe *Admin* esegue quattro query:

- a) Numero di filiali:

---

```
SELECT COUNT(*) AS count  
FROM filiale
```

---

- b) Numero di soci:

---

```
SELECT COUNT(*) AS count  
FROM socio
```

---

- c) Numero di noleggi nel mese corrente:

---

```
SELECT COUNT(*) AS count  
FROM noleggiocheckin AS i  
WHERE MONTH(i.data_operazione) = MONTH(CURRENT_DATE())  
AND YEAR(i.data_operazione) = YEAR(CURRENT_DATE())
```

---

d) Fatturato nel mese corrente:

---

```
SELECT CONCAT(SUM(costo), "€") AS revenue
FROM noleggiocheckout
WHERE MONTH(data_operazione) = MONTH(CURRENT_DATE())
AND YEAR(data_operazione) = YEAR(CURRENT_DATE())
```

---

Il risultato delle query viene renderizzato dal componente *cards.php*, ottenendo quindi:

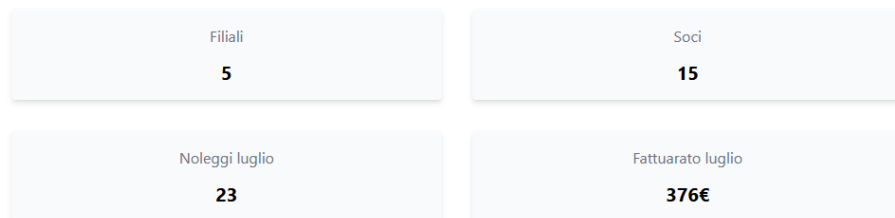


Figura 3.4: Statistiche admin

### Grafico

Il metodo *getChart()* nella classe *Admin* esegue la query per ottenere le performance nel mese corrente delle filiali registrate:

---

```
SELECT f.nome AS branch , (SUM(o.costo)*100)/t.tot AS performance
FROM filiale AS f
LEFT JOIN noleggio AS n ON n.filiale = f.nome
LEFT JOIN noleggiocheckout AS o ON o.noleggio = n.codice_noleggio AND (
    MONTH(o.data_operazione) = MONTH(CURRENT_DATE()) AND
    YEAR(o.data_operazione) = YEAR(CURRENT_DATE())
) CROSS JOIN (
    SELECT SUM(o2.costo) AS tot
    FROM noleggiocheckout AS o2
) t
GROUP BY f.nome
```

---

Il risultato della query viene passato al file *performance\_chart.js*, ottenendo quindi:

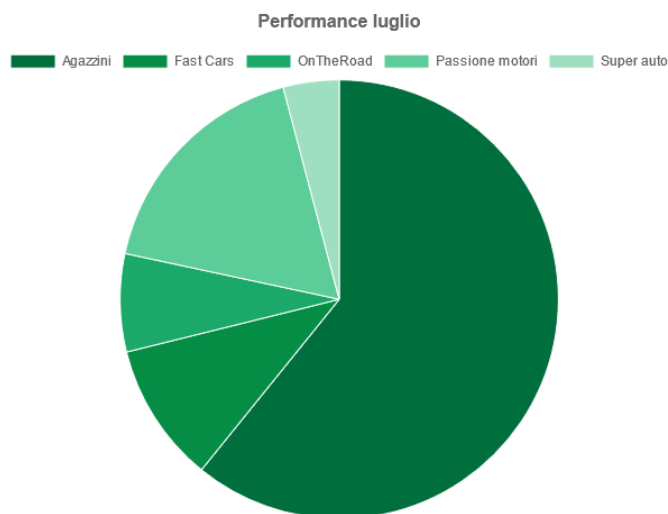


Figura 3.5: Grafico con le performance

### 3.3.2 Manager

Come manager di esempio si considera:

- **Email:** aida.agazzini@gmail.com
- **Password:** 12345678
- **Filiale:** Agazzini

#### Statistiche

Il metodo *getStats()* nella classe *Manager* esegue quattro query:

a) Numero di veicoli disponibili:

---

```
SELECT COUNT(*) AS count
FROM manager AS m
JOIN filiale AS f ON f.manager = m.cf
JOIN veicolo AS v ON v.filiale = f.nome
WHERE m.cf = :cf AND v.stato <> :stato
```

---

dove:

- *:cf* è uguale al codice fiscale del manager autenticato
- *:stato* è uguale a "non\_disponibile"

b) Numero di addetti (con un contratto valido):

---

```
SELECT COUNT(*) AS count
FROM manager AS m
JOIN filiale AS f ON f.manager = m.cf
JOIN addetto AS a ON f.nome = a.filiale
WHERE m.cf = :cf AND
(CURDATE() < a.data_scadenza_contratto OR a.data_scadenza_contratto IS NULL)
```

---

c) Numero di soci che hanno fatto almeno un noleggio, nell'ultimo anno, nella filiale che gestisce:

---

```
SELECT COUNT(DISTINCT s.cf) AS count
FROM manager AS m
JOIN filiale AS f ON f.manager = m.cf
JOIN noleggio AS n ON n.filiale = f.nome
JOIN noleggiocheckin AS i ON i.noleggio = n.codice_noleggio
JOIN socio AS s ON s.cf = n.socio
WHERE m.cf = :cf AND
YEAR(i.data_operazione) = YEAR(CURRENT_DATE())
```

---

d) Fatturato nel mese corrente:

---

```
SELECT DAY(i.data_operazione) AS day, COUNT(*) AS rentals
FROM noleggio AS n
JOIN filiale AS f ON f.nome = n.filiale
JOIN manager AS m ON m.cf = f.manager
JOIN noleggiocheckin AS i ON i.noleggio = n.codice_noleggio
WHERE m.cf = :cf
AND MONTH(i.data_operazione) = MONTH(CURRENT_DATE())
AND YEAR(i.data_operazione) = YEAR(CURRENT_DATE())
GROUP BY DAY(i.data_operazione)
```

---

Si ottiene quindi:

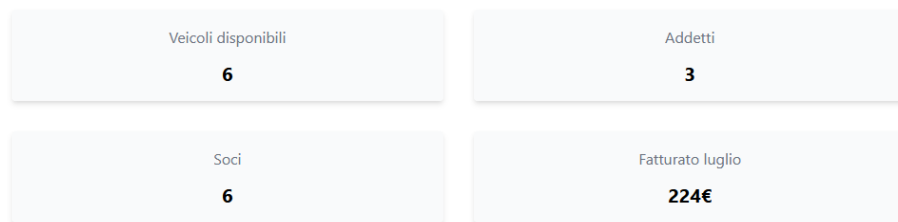


Figura 3.6: Statistiche manager

### Grafico

Il metodo *getChart()* nella classe *Manager* esegue la query per ottenere il numero di check-in giornalieri nel mese corrente:

```
SELECT DAY(i.data_operazione) AS day, COUNT(*) AS rentals
FROM noleggio AS n
JOIN filiale AS f ON f.nome = n.filiale
JOIN manager AS m ON m.cf = f.manager
JOIN noleggiocheckin AS i ON i.noleggio = n.codice_noleggio
WHERE m.cf = :cf
AND MONTH(i.data_operazione) = MONTH(CURRENT_DATE())
AND YEAR(i.data_operazione) = YEAR(CURRENT_DATE())
GROUP BY DAY(i.data_operazione)
```

Si ottiene quindi, per il giorno 05/07:

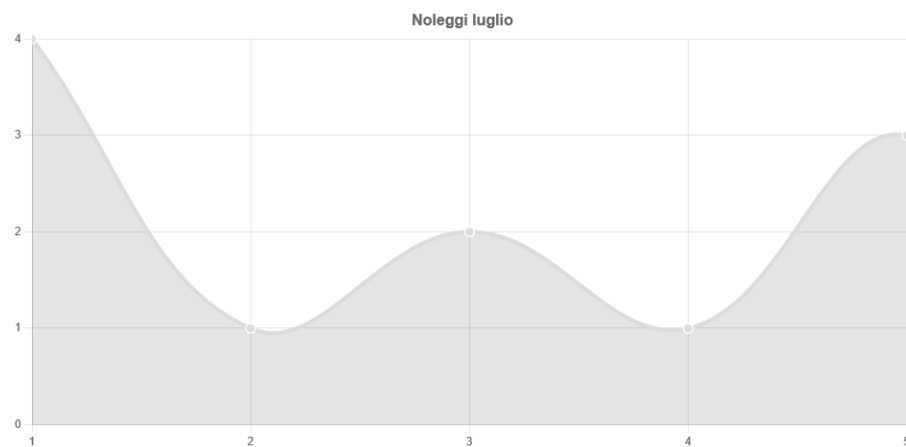


Figura 3.7: Grafico con il numero di check-in

### 3.3.3 Addetto

Come addetto di esempio si considera:

- **Email:** dion.freg@gmail.com
- **Password:** 12345678
- **Filiale:** Agazzini

### Statistiche

Il metodo *getStats()* nella classe *Employee* esegue quattro query:

a) Numero di soci aiutati nella fase di check-in o check-out:

---

```
SELECT COUNT(DISTINCT s.cf) AS count
FROM addetto AS a, socio AS s
JOIN noleggio AS n ON n.socio = s.cf
WHERE a.cf = :cf AND (n.codice_noleggio IN (
    SELECT i.noleggio FROM noleggiocheckin AS i WHERE i.addetto = a.cf
) OR n.codice_noleggio IN (
    SELECT o.noleggio FROM noleggiocheckout AS o WHERE o.addetto = a.cf
))
```

---

b) Numero totale di noleggi in cui ha eseguito sia la fase di check-in che di check-out:

---

```
SELECT COUNT(*) AS count
FROM addetto AS a
JOIN noleggiocheckin AS i ON i.addetto = a.cf
JOIN noleggiocheckout AS o ON o.addetto = a.cf
WHERE a.cf = :cf AND i.noleggio = o.noleggio
```

---

c) Numero totale check-in svolti:

---

```
SELECT COUNT(*) AS count
FROM addetto AS a
JOIN noleggiocheckin AS i ON i.addetto = a.cf
WHERE a.cf = :cf
```

---

d) Numero totale check-out svolti:

---

```
SELECT COUNT(*) AS count
FROM addetto AS a
JOIN noleggiocheckout AS o ON o.addetto = a.cf
WHERE a.cf = :cf
```

---

Si ottiene quindi:

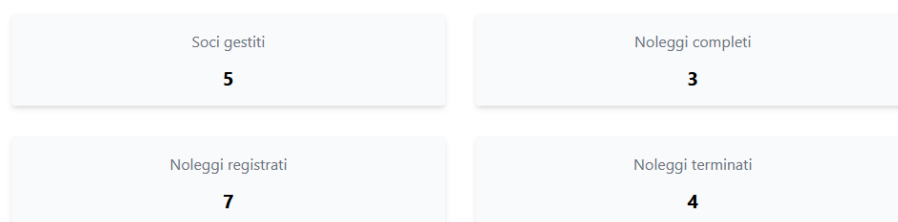


Figura 3.8: Statistiche addetto

### Grafico

Il metodo *getChart()* nella classe *Employee* esegue la query per ottenere numero di check-in e check-out registrati giornalmente nel mese corrente:

---

```
SELECT SUM(rentals) AS rentals, day
FROM (
    SELECT DAY(i.data_operazione) AS day, COUNT(*) AS rentals
    FROM noleggiocheckin AS i
    WHERE i.addetto = :cf
```



```

AND MONTH(i.data_operazione) = MONTH(CURRENT_DATE())
AND YEAR(i.data_operazione) = YEAR(CURRENT_DATE())
GROUP BY DAY(i.data_operazione)

UNION ALL

SELECT DAY(o.data_operazione) AS day, COUNT(*) AS rentals
FROM noleggiocheckout AS o
WHERE o.addetto = :cf
AND MONTH(o.data_operazione) = MONTH(CURRENT_DATE())
AND YEAR(o.data_operazione) = YEAR(CURRENT_DATE())
GROUP BY DAY(o.data_operazione)
) t
GROUP BY day

```

Si ottiene quindi, per il giorno 05/07:

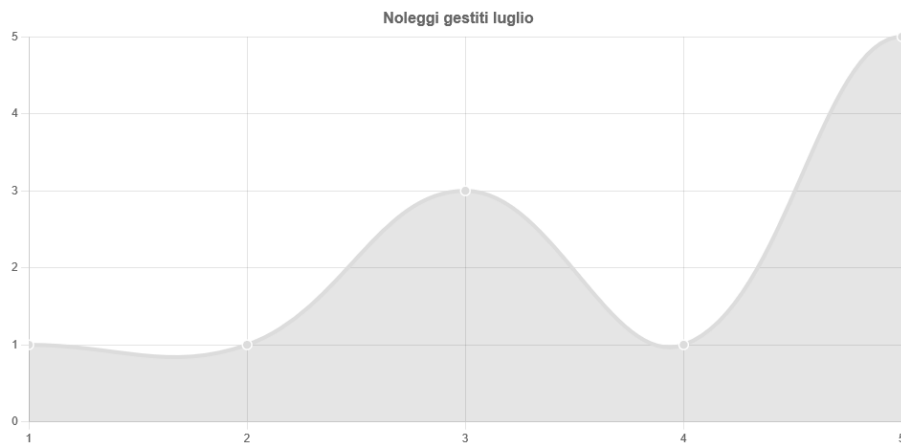


Figura 3.9: Grafico con il numero check-in e check-out registrati

### 3.3.4 Socio

Come socio di esempio si considera:

- **Email:** ludovica.carannante@gmail.com
- **Password:** 12345678

#### Statistiche

Il metodo *getStats()* nella classe *Member* esegue quattro query:

a) Numero di noleggi effettuati:

```

SELECT COUNT(*) AS count
FROM socio AS s
JOIN noleggio AS n ON n.socio = s.cf
AND s.cf = :cf

```

b) Numero di noleggi attivi:

```

SELECT COUNT(*) AS count
FROM socio AS s
JOIN noleggio AS n ON n.socio = s.cf
WHERE n.codice_noleggio NOT IN (

```

```
SELECT noleggio
FROM noleggiocheckout
) AND s.cf = :cf
```

c) Auto noleggiata più volte:

```
SELECT CONCAT(v.marca, " ", v.modello) AS car, COUNT(*) AS rentals
FROM noleggio AS n
JOIN veicolo AS v ON v.targa = n.veicolo
WHERE n.socio = :cf
GROUP BY v.marca, v.modello
ORDER BY rentals DESC LIMIT 1
```

d) Spesa media:

```
SELECT CONCAT(ROUND(AVG(o.costo), 2), "€") AS cost
FROM noleggio AS n
JOIN noleggiocheckout as o on o.noleggio = n.codice_noleggio
WHERE n.socio = :cf
```

Si ottiene quindi:

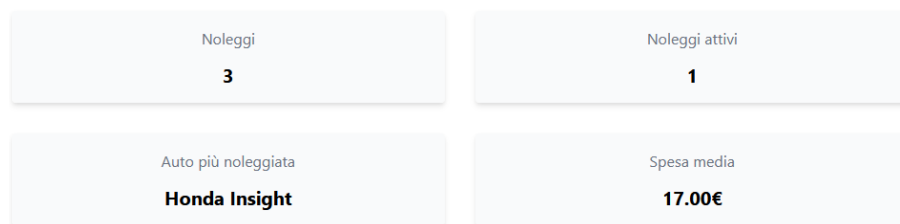


Figura 3.10: Statistiche socio

## Grafico

Il metodo *getChart()* nella classe *Member* esegue la query per ottenere il numero di noleggi effettuati mensilmente nell'anno corrente:

```
SELECT MONTH(i.data_operazione) AS month, COUNT(*) AS rentals
FROM noleggio AS n
JOIN noleggiocheckin AS i ON i.noleggio = n.codice_noleggio
WHERE n.socio = :cf
AND YEAR(i.data_operazione) = YEAR(CURRENT_DATE())
GROUP BY MONTH(i.data_operazione)
```

Si ottiene quindi il grafico in Figura 3.11.

## 3.4 Profilo

Siccome il profilo di manager, addetti e soci presenta i medesimi campi anagrafici, è presente nella superclasse *Person* il metodo *getProfile()* che verrà sovrascritto nelle classi figlie per aggiungere i campi aggiuntivi quali:

- Filiale gestita per il manager
- Data di assunzione e scadenza contratto per l'addetto
- Data di iscrizione per il socio

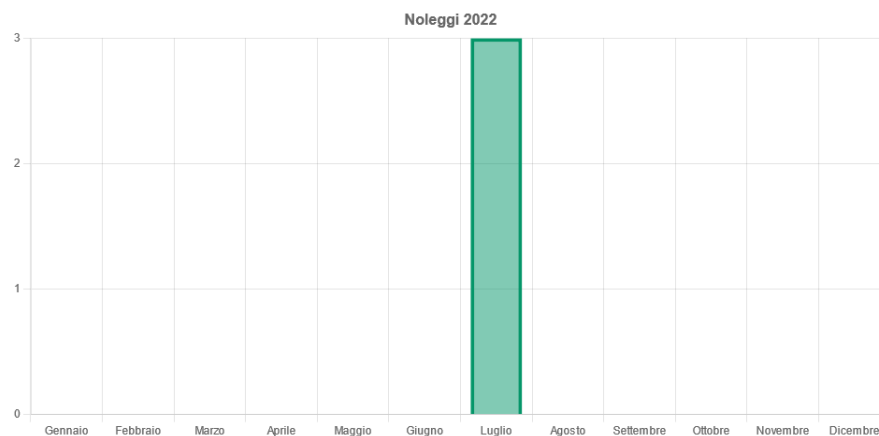


Figura 3.11: Grafico con il numero di noleggi effettuati

Sotto vengono riportati i profili per gli account sopra citati. Come già annunciato, l'amministratore non ha informazioni da mostrare.

INFORMAZIONI	
Manager filiale	Agazzini
Codice fiscale	GZZDAI00L66G336C
Nome	Aida
Cognome	Agazzini
Data di nascita	2000-07-26
Domicilio	Via A.Coari, 166/k
Numero di telefono	0184/691674

Figura 3.12: Profilo manager

INFORMAZIONI	
Codice fiscale	FRGDNG79L19E605C
Nome	Dionigi
Cognome	Freguglia
Data di nascita	1979-07-19
Domicilio	Via Talete, 238
Numero di telefono	059/226978
Data di assunzione	2022-07-01
Data scadenza contratto	Tempo indeterminato

Figura 3.13: Profilo addetto

INFORMAZIONI	
Codice fiscale	CRNLVC90H45C408U
Nome	Ludovica
Cognome	Carannante
Data di nascita	1990-06-05
Domicilio	Via Pisa, 247/f
Numero di telefono	0472/168615
Data di iscrizione	2022-07-01

Figura 3.14: Profilo socio

## 3.5 Operazioni

Le operazioni vengono visualizzate in ogni view in un pannello sulla sinistra renderizzato dal componetene *aside.php*. La lista di link che conducono alle pagine per svolgere le operazioni viene restituita dal metodo *getOperations()* presente in ogni classe che modella una tipologia di utente.

Esempio operazioni addetto:

---

```
public static function getOperations()
{
    return array(
        'hireEmployee' => new Operation('Assumi_laddetto', 'employees/create.php'),
        'addCar' => new Operation('Aggiungi_veicolo', 'cars/create.php'),
        'employees' => new Operation('Addetti', 'employees/index.php'),
        'rentals' => new Operation('Noleggi', 'branches/rentals/index.php'),
        'cars' => new Operation('Veicoli', 'cars/index.php'),
    );
}
```

---

Le barre di navigazione si presentano, per ogni tipologia di utente, come in Figura 3.15.

La verifica dell'autorizzazione per l'accesso alle pagine di operazione viene fatta nei file di route, dichiarando in un array i ruoli che possono visualizzare tale pagina e includendo il file *authorization.inc.php*.

L'aggiunta di un nuovo manager, addetto o socio richiede inoltre l'inserimento di delle credenziali per la creazione dell'utente. Se l'utente da inserire è un manager devono essere specificati anche i dati di una filiale.

Ad esempio, il form per la registrazione di un utente si presenta come nella Figura 3.16. Come per il login, i campi vengono restituiti da un metodo statico *getFields()* presente sia nella classe *Member* che *User* (come visto precedentemente).

Per aggiornare, invece, i dati attraverso un form viene eseguito il metodo **non statico** *getUpdateFields()*.

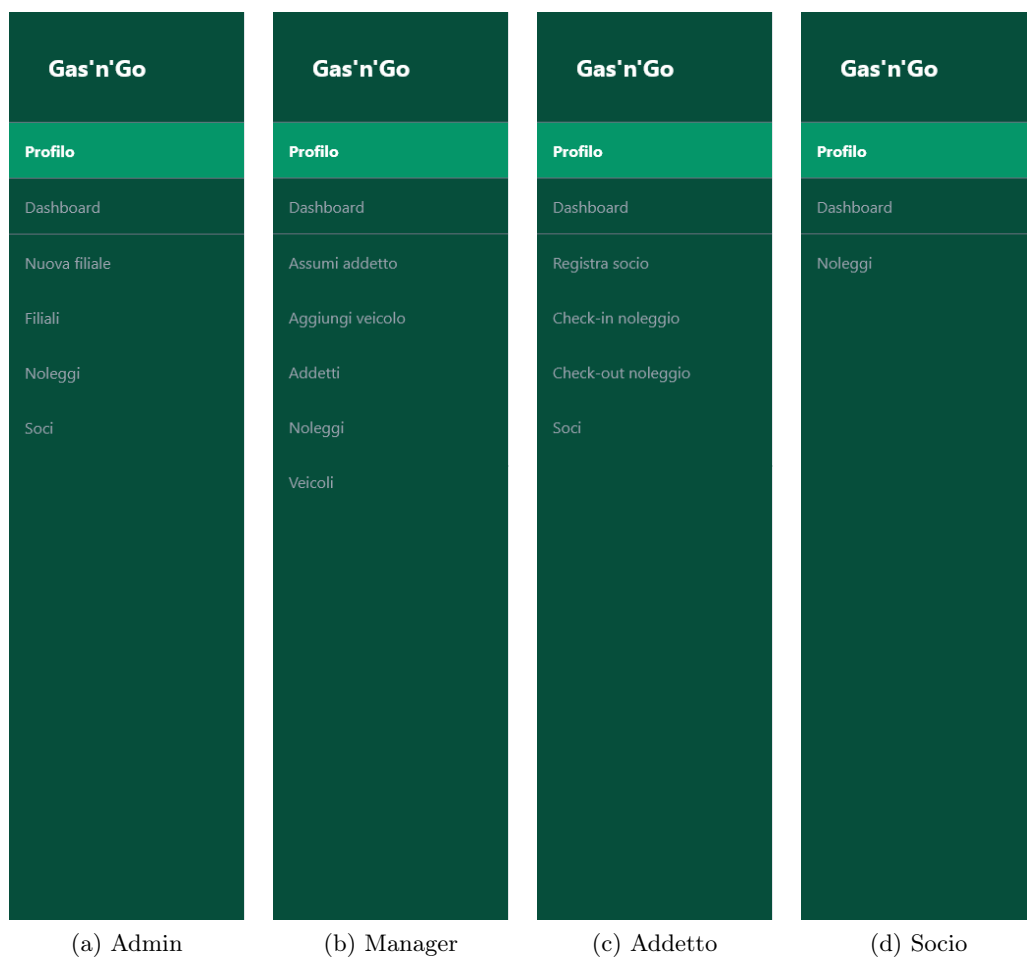
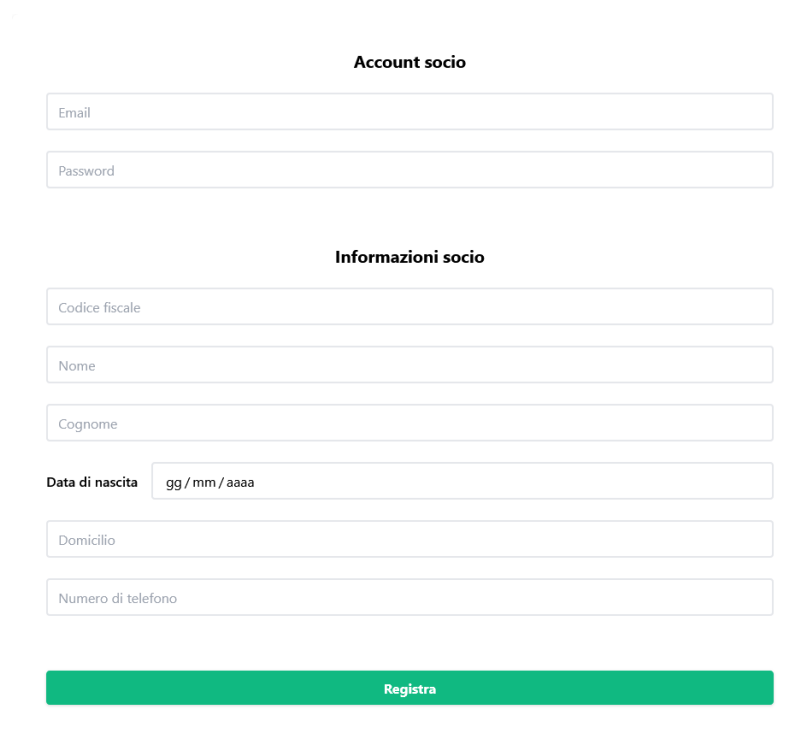


Figura 3.15: Pannelli laterali di navigazione



The form is titled "Account socio" and "Informazioni socio". It contains several input fields for user registration. The "Account socio" section has fields for "Email" and "Password". The "Informazioni socio" section has fields for "Codice fiscale", "Nome", "Cognome", "Data di nascita" (with a date format hint "gg/mm/aaaa"), "Domicilio", and "Numero di telefono". A green "Registra" button is at the bottom.

**Account socio**

Email

Password

**Informazioni socio**

Codice fiscale

Nome

Cognome

Data di nascita gg/mm/aaaa

Domicilio

Numero di telefono

**Registra**

Figura 3.16: resources/members/create.php

## Capitolo 4

# Gestione dei noleggi

### 4.1 Check-in

Questa fase viene gestita dal controller *CheckinController*.

#### 4.1.1 Selezione socio

La fase di check-in inizia collegandosi alla pagina *resources/rentals/checkin/member.php* per selezionare il socio che richiede un noleggio. Nota: dopo la creazione di un socio si passa direttamente alla selezione di un veicolo da noleggiare, saltando di fatto questa parte.

La selezione del socio avviene eseguendo il metodo *selectMember(User \$auth)* nel controller. Il metodo restituisce alla view una tabella contenente email, nome e cognome dei soci disponibili. I dati vengono restituiti dal metodo statico *readAll()* presente nella classe *Member*, il quale esegue la query:

---

```
SELECT u.email, s.nome, s.cognome
FROM socio AS s
JOIN utente AS u ON u.ID = s.user_id
ORDER BY s.data_di_iscrizione DESC, u.email
```

---

L'addetto visualizzerà quindi la tabella:

EMAIL	NOME	COGNOME	
<b>annamaria.amantea@gmail.com</b>	Annamaria	Amantea	<a href="#">Seleziona</a>
<b>avaccaroni@gmail.com</b>	Amalia	Vaccaroni	<a href="#">Seleziona</a>
<b>eliana.petroni@gmail.com</b>	Eliana	Petroni	<a href="#">Seleziona</a>
<b>liliana.ansione@gmail.com</b>	Liliana	Ansione	<a href="#">Seleziona</a>
<b>paol.ricc@gmail.com</b>	Paolo	Ricco	<a href="#">Seleziona</a>

Figura 4.1: Tabella per la selezione del socio

#### 4.1.2 Selezione veicolo

Dopo aver selezionato un socio si passa alla selezione del veicolo.

La route della pagina è *resources/rentals/checkin/car.php* e richiede il parametro *email* nella query string. La route esegue il metodo *selectCar(User \$auth)* nel controller. Tale metodo per prima cosa controlla la validità del parametro email, per fare ciò prova a leggere un membro dal database, tramite il metodo statico della classe *Member readFromEmail(\$email)*. Se il valore è nullo reindirizza l'utente alla pagina di selezione di un socio, altrimenti restituisce alla view una tabella contenente le auto disponibili nella filiale dell'addetto tramite il metodo *getAvailableCarsTable(\$branch, \$action)* della classe *Car*. Tale metodo richiede come secondo parametro un azione da compiere sugli elementi della tabella, in questo caso l'operazione è quella di selezione. La query che viene eseguita è la seguente:

---

```
SELECT targa, marca, modello, colore, costo_giornaliero
FROM veicolo
WHERE stato = :stato AND filiale = :filiale
ORDER BY marca ASC, modello ASC, colore ASC, costo_giornaliero DESC
```

---

dove *:stato* è uguale a disponibile.

L'addetto visualizzerà quindi la tabella:

TARGA	MARCA	MODELLO	COLORE	COSTO GIORNALIERO	
AW100DR	Honda	Insight	Nera	9	<a href="#">Seleziona</a>
VB039MN	Rover	216	Grigia	12	<a href="#">Seleziona</a>
LL120XB	Skoda	Felicia	Nera	8	<a href="#">Seleziona</a>

Figura 4.2: Tabella per la selezione di un veicolo

### 4.1.3 Creazione noleggio

La selezione del veicolo porta alla route *resources/rentals/checkin/checkin.php* che richiede due parametri:

- *email* socio
- *targa* veicolo

La route esegue il metodo *store(User \$auth)* nel controller. Come prima, viene controllata la validità dei parametri della query string.

Se entrambi i parametri sono validi si avranno gli oggetti *\$member* e *\$car*. Per la creazione del noleggio e del check-in è necessario creare due array contenenti i valori da inserire. Per il noleggio sono

---

```
$rentalData = array(
    'veicolo' => $car->getPlate(),
    'filiale' => $employee->getBranch(),
    'socio' => $member->getCF(),
);
```

---

mentre per il check-in va determinata anche la data e l'ora corrente:

---

```
$checkinData = array('addetto' => $employee->getCF());

date_default_timezone_set('Europe/Rome');
$checkinData['data_operazione'] = date('Y-m-d_H:i:s');
```

---



Infine si passano questi dati al metodo statico *createCheckin(\$rentalData, \$checkinData)* della classe *Rental*.

### **createCheckin(\$rentalData, \$checkinData)**

Il metodo inizia con l'istanziare la classe *DB* ed eseguire il metodo *begin()* così da poter effettuare un roll back in caso di errore.

Prima dell'inserimento del check-in è necessario inserire un nuovo noleggio:

---

```
INSERT INTO noleggio
(veicolo, filiale, socio)
VALUES
(:veicolo, :filiale, :socio)
```

---

Se l'operazione è andata a buon fine otteniamo il codice noleggio con il metodo della classe *DB* *lastInsertId()* e procediamo con l'inserimento del check-in

---

```
INSERT INTO noleggiocheckin
(noleggio, addetto, data_operazione)
VALUES
(:noleggio, :addetto, :data_operazione)
```

---

L'ultima operazione è quella dell'aggiornamento dello stato del veicolo che passa da "disponibile" a "in noleggio". Per fare ciò si esegue il metodo statico della classe *Car* *updateState(\$db, \$plate, \$state)*. Tale metodo richiede l'istanza del database in quanto se si verifica un errore è possibile fare un roll back ed annullare gli inserimenti del noleggio e del check-in. La query di aggiornamento è la seguente:

---

```
UPDATE veicolo
SET stato = :stato
WHERE targa = :targa
```

---

Gli stati che un veicolo può avere sono definiti nella classe *Car* e sono

---

```
const ON_RENTAL = 'in_noleggio';
const AVAILABLE = 'disponibile';
const UNAVAILABLE = 'non_disponibile';
```

---

## **4.2 Check-out**

Questa fase viene gestita dal controller *CheckoutController*.

### **4.2.1 Selezione socio**

Come per il check-in, come prima cosa l'addetto seleziona un socio, che in questo caso deve avere un noleggio attivo nella filiale dell'addetto. Quest'ultimo si deve collegare alla pagina *resources/rentals/checkout/member.php*.

La selezione del socio avviene eseguendo il metodo *selectMember(User \$auth)* nel controller. Il metodo restituisce alla view una tabella contenente email, nome, cognome e numero di noleggi attivi dei soci con almeno un noleggio attivo nella filiale. I dati vengono restituiti dal metodo statico *readActiveMembersRentals(\$branch)* presente nella classe *Rental*, il quale esegue la query:

---

```
SELECT u.email, s.nome, s.cognome, COUNT(*) AS count
FROM socio AS s
```

---

---

```

JOIN utente AS u ON u.ID = s.user_id
JOIN noleggio AS n ON n.socio = s.cf
WHERE n.codice_noleggio NOT IN(
    SELECT noleggio FROM noleggiocheckout
) AND n.filiale = :filiale
GROUP BY s.nome, s.cognome, u.email
ORDER BY u.email

```

---

L'addetto visualizzerà quindi la tabella:

EMAIL	NOME	COGNOME	NUMERO NOLEGGI	
ange.guer@gmail.com	Angelo	Guerrieri	1	<a href="#">Seleziona</a>
ludovica.carannante@gmail.com	Ludovica	Carannante	1	<a href="#">Seleziona</a>
silvestro.ghio@gmail.com	Silvestro	Ghio	1	<a href="#">Seleziona</a>

Figura 4.3: Tabella per la selezione del socio

#### 4.2.2 Selezione noleggio

Dopo aver selezionato un socio si passa alla selezione del noleggio da terminare.

La route della pagina è *resources/rentals/checkout/rental.php* e richiede il parametro *email* nella query string. La route esegue il metodo *selectRental(User \$auth)* nel controller. Tale metodo per prima cosa controlla la validità del parametro email e se il valore è valido restituisce alla view una tabella contenente i noleggi attivi effettuati nella filiale dell'addetto tramite il metodo *getActiveMemberRentalsTable(\$cf, \$branch, \$action)* della classe *Rental* che esegue la query:

---

```

SELECT n.codice_noleggio, v.targa,
CONCAT(v.marca, " ", v.modello, " ", v.colore) AS car,
i.data_operazione AS inizio_noleggio,
CONCAT(
    (v.costo_giornaliero * (DATEDIFF(NOW(), i.data_operazione)+1)),
    "$\mbox{\texteuro}$"
) AS cost
) AS cost
FROM noleggio AS n
JOIN veicolo AS v ON v.targa = n.veicolo
JOIN noleggiocheckin AS i ON i.noleggio = n.codice_noleggio
JOIN socio AS s ON s.cf = n.socio
WHERE n.codice_noleggio NOT IN (SELECT noleggio FROM noleggiocheckout)
AND s.cf = :cf AND n.filiale = :filiale
ORDER BY i.data_operazione

```

---

L'addetto visualizzerà quindi, per il socio "Silvestro Ghio" nel giorno 08/07, la tabella:

CODICE	TARGA	VEICOLO	INIZIO NOLEGGIO	COSTO	
9	BK283BD	Ford Transit Bianca	2022-07-05 10:46:13	56€	<a href="#">Termina</a>

Figura 4.4: Tabella per la selezione di un veicolo

### 4.2.3 Check-out

La selezione del veicolo porta alla route *resources/rentals/checkout/checkout.php* che richiede il parametro *codice\_noleggio*. La route esegue il metodo *store(User \$auth)* nel controller. Come prima, viene controllata la validità del codice noleggio. Per fare ciò si deve controllare che il noleggio non sia già stato terminato, viene quindi eseguito il metodo *isActive()* della classe *Rental* il quale esegue la query:

---

```
SELECT *
FROM noleggio
WHERE codice_noleggio NOT IN (
    SELECT noleggio
    FROM noleggiocheckout
) AND codice_noleggio = :codice
```

---

e si controlla se il risultato presenta delle righe.

Per la registrazione del checkout viene eseguito il metodo statico della classe *Rental* *createCheckout(\$checkoutData, \$plate)* il quale richiede anche la targa così da poter aggiornare lo stato del veicolo e renderlo disponibile per ulteriori. *\$checkoutData* è un array contenente il codice fiscale dell'addetto, il codice noleggio, la data e l'ora attuale e il costo del noleggio calcolato come:

$$(\text{numero di giorni} + 1) * \text{costo giornaliero}$$

il +1 rappresenta il giorno nel quale è iniziato il noleggio, se il noleggio è giornaliero il numero di giorni è pari a zero.

I dati riguardanti il costo giornaliero, la targa del veicolo e la data di inizio noleggio vengono restituiti dal metodo *getCheckinInfo()* della classe *Rental*, il quale esegue la query:

---

```
SELECT i.data_operazione AS date, v.costo_giornaliero AS price
FROM noleggiocheckin AS i
JOIN noleggio AS n ON n.codice_noleggio = i.noleggio
JOIN veicolo AS v ON v.targa = n.veicolo
WHERE n.codice_noleggio = :codice
```

---

#### **createCheckout(\$checkoutData, \$plate)**

Il metodo inizia con l'istanziare la classe *DB* ed eseguire il metodo *begin()* così da poter effettuare un roll back in caso di errore.

Come prima cosa viene inserito il noleggio:

---

```
INSERT INTO noleggiocheckout
(noleggio, addetto, data_operazione, costo)
VALUES
(:noleggio, :addetto, :data_operazione, :costo)
```

---

Se l'operazione è andata a buon fine si aggiorna lo stato del veicolo con il metodo utilizzato anche nella fase di check-in, si esegue quindi il metodo:

---

```
Car::updateState($db, $plate, Car::AVAILABLE)
```

---

Il veicolo ora è nuovamente disponibile per il noleggio. Si conclude così la fase di check-out.

## Capitolo 5

# Conclusione

I dati presenti nel DB sono stati generati tramite il tool: <http://www.gaffuri.it/generatore/>.

Il diagramma E/R è stato creato tramite il tool: <https://www.diagrams.net/>