Gestione parcheggi

[1 Introduzione 5](#_Toc27664691)

[1.1 Informazioni sul progetto 5](#_Toc27664692)

[1.2 Abstract 5](#_Toc27664693)

[1.3 Scopo 5](#_Toc27664694)

[2 Analisi 5](#_Toc27664695)

[2.1 Analisi del dominio 5](#_Toc27664696)

[2.2 Analisi e specifica dei requisiti 5](#_Toc27664697)

[2.3 Use case 9](#_Toc27664698)

[2.4 Pianificazione 10](#_Toc27664699)

[2.5 Analisi dei mezzi 10](#_Toc27664700)

[2.5.1 Software 10](#_Toc27664701)

[2.5.2 Hardware 10](#_Toc27664702)

[3 Progettazione 11](#_Toc27664703)

[3.1 Design dell’architettura del sistema 11](#_Toc27664704)

[3.1.1 Sitemap 11](#_Toc27664705)

[3.1.2 Schema di rete 11](#_Toc27664706)

[3.2 Design dei dati e database 12](#_Toc27664707)

[3.2.1 Diagramma E-R 12](#_Toc27664708)

[3.2.2 Schema procedurale 13](#_Toc27664709)

[3.2.3 Descrizione database 13](#_Toc27664710)

[3.3 Design delle interfacce 14](#_Toc27664711)

[3.3.1 Pagina di login 14](#_Toc27664712)

[3.3.2 Pagina di registrazione 15](#_Toc27664713)

[3.3.3 Pannello di controllo dell’admin 15](#_Toc27664714)

[3.3.4 Struttura del sito 16](#_Toc27664715)

[3.3.5 Pagina delle informazioni del parcheggio 17](#_Toc27664716)

[3.3.6 Pagina di profilo dell’utente 17](#_Toc27664717)

[3.3.7 Pagina per la modifica del proprio profilo 18](#_Toc27664718)

[3.4 Design procedurale 19](#_Toc27664719)

[4 Implementazione 20](#_Toc27664720)

[4.1 Controllers 20](#_Toc27664721)

[4.1.1 Home 20](#_Toc27664722)

[4.1.1.1 Index() 20](#_Toc27664723)

[4.1.2 Login 20](#_Toc27664724)

[4.1.2.1 logout() 20](#_Toc27664725)

[4.1.2.2 login() 20](#_Toc27664726)

[4.1.3 Register 21](#_Toc27664727)

[4.1.3.1 register() 21](#_Toc27664728)

[4.1.3.2 verify() 21](#_Toc27664729)

[4.1.4 Offer 21](#_Toc27664730)

[4.1.4.1 offri() 21](#_Toc27664731)

[4.1.4.2 index() 21](#_Toc27664732)

[4.1.5 Research 22](#_Toc27664733)

[4.1.5.1 index() 22](#_Toc27664734)

[4.1.6 Reserve 22](#_Toc27664735)

[4.1.6.1 index() 22](#_Toc27664736)

[4.1.6.2 prenota() 22](#_Toc27664737)

[4.1.7 Validator 23](#_Toc27664738)

[4.1.7.1 validateDate(element) 23](#_Toc27664739)

[4.1.7.2 validateCharAndSpace(element) 23](#_Toc27664740)

[4.1.7.3 validateCap(element) 23](#_Toc27664741)

[4.1.7.4 validateCarPlate(element) 23](#_Toc27664742)

[4.1.7.5 validateVia(element) 23](#_Toc27664743)

[4.1.7.6 testInput(element) 24](#_Toc27664744)

[4.1.7.7 validatePhoneNumber(element) 24](#_Toc27664745)

[4.2 Models 24](#_Toc27664746)

[4.2.1 LoginModel 24](#_Toc27664747)

[4.2.1.1 log(email, password) 24](#_Toc27664748)

[4.2.2 RegisterModel 25](#_Toc27664749)

[4.2.2.1 register() 25](#_Toc27664750)

[4.2.2.2 verify(mail, nome, cognome) 26](#_Toc27664751)

[4.2.3 OfferModel 27](#_Toc27664752)

[4.2.3.1 addOfferta() 28](#_Toc27664753)

[4.2.4 ResearchModel 30](#_Toc27664754)

[4.2.4.1 getParcheggiDisponibili() 30](#_Toc27664755)

[4.2.4.2 filterAll(startDate, endDate, disp) 30](#_Toc27664756)

[4.2.4.3 filterByDisp() 32](#_Toc27664757)

[4.2.4.4 formatCarAndDate() 32](#_Toc27664758)

[4.2.5 ReserveModel 33](#_Toc27664759)

[4.2.5.1 updateParcheggio() 33](#_Toc27664760)

[4.2.5.2 getParcheggioInfo(id\_parcheggio) 34](#_Toc27664761)

[4.2.5.3 prenota() 35](#_Toc27664762)

[4.2.6 Users 36](#_Toc27664763)

[4.2.6.1 checkRuolo() 37](#_Toc27664764)

[4.2.6.2 hasParcheggio() 37](#_Toc27664765)

[4.2.7 MailModel 37](#_Toc27664766)

[4.2.7.1 build() 38](#_Toc27664767)

[4.2.7.2 reservationMail(email, nome, cognome, parcheggio, riservazione) 38](#_Toc27664768)

[4.2.7.3 newUserMail(newUserMail, nome, cognome) 39](#_Toc27664769)

[4.2.8 PDF 39](#_Toc27664770)

[4.2.8.1 createTable(header, parcheggio, riservazione) 40](#_Toc27664771)

[4.2.8.2 createPDF(parcheggio, riservazione) 41](#_Toc27664772)

[5 Test 42](#_Toc27664773)

[5.1 Protocollo di test 42](#_Toc27664774)

[5.2 Risultati test 52](#_Toc27664775)

[5.3 Mancanze/limitazioni conosciute 52](#_Toc27664776)

[6 Consuntivo 1](#_Toc27664777)

[7 Conclusioni 1](#_Toc27664778)

[7.1 Sviluppi futuri 1](#_Toc27664779)

[7.2 Considerazioni personali 1](#_Toc27664780)

[8 Glossario 1](#_Toc27664781)

[9 Bibliografia 1](#_Toc27664782)

[9.1 Sitografia 1](#_Toc27664783)

[10 Allegati 2](#_Toc27664784)

# Introduzione

## Informazioni sul progetto

Allievo: Paolo Weishaupt

Classe: Informatica 4AC presso la Scuola d’Arti e Mestieri di Trevano

Docente responsabile: Guido Montalbetti

Data inizio: 03/09/2019

Data fine: 20/12/2019

## Abstract

*As a society gets bigger and bigger, the organization is one of the key factor of it’s final success. And the organization is needed even for the particulars or, in our case, for the parking places. This project aims to simplify the parking places management by providing a nice and very intuitive UI with everything needed to the user and the administrators.*

## Scopo

Lo scopo del progetto è quello di implementare un metodo per gestire l’affitto dei posteggi presenti a scuola. È prettamente didattico, ma potrebbe venire usato se implementato correttamente. Infatti al momento esiste un metodo per gestirli, ma si è rivelato estremamente inefficace.

# Analisi

## Analisi del dominio

Al momento la gestione dei posteggi viene gestita a Bellinzona dal cantone. Una persona per poter riservare un parcheggio deve fare la richiesta alla direzione del CPT che in seguito verrà girata all’ufficio di competenza del cantone. Questo metodo non è efficiente infatti spesso di lunedì e martedì le auto vengono posteggiate nel piazzale davanti alla scuola e nei prati. Con questo nuovo servizio basterebbe effettuare la registrazione al sito per poter essere poi immediatamente in grado di riservare il posteggio. Le conoscenze richieste sono quindi minime e alla portata di tutti.

## Analisi e specifica dei requisiti

|  |  |
| --- | --- |
| **ID: REQ-001** | |
| **Nome** | Creazione applicativo web |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Il progetto vuole creare un sito Web dove poter gestire i parcheggi disponibili della scuola |
| **Sotto requisiti** | |
| **001** | Le persone potranno ricercare i posteggi disponibili |
| **002** | Le persone potranno riservare i posteggi |
| **003** | Le persone potranno stampare la loro riservazione |

|  |  |
| --- | --- |
| **ID: REQ-002** | |
| **Nome** | Creazione maschera admin |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Il sito dovrà avere una parte del menù riservata agli amministratori |
| **Sotto requisiti** | |
| **001** | Gli amministratori potranno inserire i parcheggi disponibili |
| **002** | Gli amministratori potranno gestire le persone registrate al sito |

|  |  |
| --- | --- |
| **ID: REQ-003** | |
| **Nome** | Funzionalità principali per gli utenti |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Il sito dovrà avere due sezioni principali: Offerta e Ricerca |
| **Sotto requisiti** | |
| **001** | Nella sezione Offerta l’utente dovrà essere in grado di mettere a disposizione il suo parcheggio. |
| **002** | Nella sezione Offerta le persone potranno inserire le date della disponibilità del loro posto auto (mattina, pomeriggio o tutto il giorno) e in più indicare il loro numero di targa oltre ai loro dati personali di base |
| **003** | Nella sezione “Ricerca” si potranno ricercare i parcheggi a disposizione inserendo come parametro il giorno preciso o anche un periodo, p.es. dal 2.12.2019 al 5.12.2019 |
| **004** | Sulla lista dei risultati, fornita dalla ricerca, si potranno ordinare e filtrare i dati |

|  |  |
| --- | --- |
| **ID: REQ-004** | |
| **Nome** | Creazione maschera di login |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Le persone dovranno registrarsi al sito ed inserire i propri dati principali per poter riservare il posto auto |
| **Sotto requisiti** | |
| **001** | I campi minimi da compilare saranno: nome, cognome, email e un recapito telefonico |
| **002** | L’applicativo dovrà prevedere un meccanismo di conferma della registrazione per l’identificazione della persona, p.es. l’invio di un link all’indirizzo di posta elettronica della persona, impostato in fase di registrazione |

|  |  |
| --- | --- |
| **ID: REQ-005** | |
| **Nome** | Specifiche per la riservazione |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Di seguito le specifiche da seguire per la riservazione di un posteggio |
| **Sotto requisiti** | |
| **001** | Il costo del posto auto è di 10 CHF al giorno da conteggiare e fatturare mensilmente (mezza giornata sono 5 CHF) |
| **002** | Al fronte di una riservazione, le persone riceveranno una conferma tramite email |
| **003** | Prevedere un meccanismo di concorrenzialità per evitare che una persona in procinto di riservare il parcheggio, si veda portar la sua riservazione da un’altra persona |

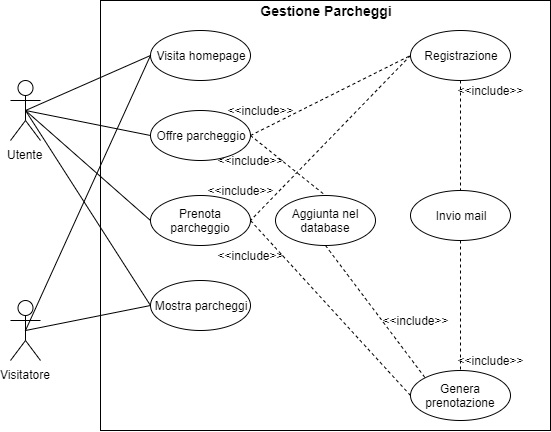
|  |  |
| --- | --- |
| **ID: REQ-006** | |
| **Nome** | Gestione stampe |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Di seguito le specifiche da seguire per la gestione delle stampe |
| **Sotto requisiti** | |
| **001** | Gli amministratori potranno stampare la situazione dei parcheggi per un determinato periodo (a disposizione, riservati) |
| **002** | Gli amministratori potranno stampare la fattura per una persona singola o per tutte le persone, scegliendo un periodo da loro impostato, generalmente una stampa mensile. Il saldo della fattura è di 30 giorni netto |
| **003** | Le persone che avranno riservato il parcheggio potranno stampare la loro riservazione |
| **004** | Gli amministratori potranno stampare i richiami per le persone che non hanno ancora saldato la loro riservazione. A metà di ogni mese, giorno feriale, verranno stampati i richiami delle fatture. Prevedere un meccanismo automatico con notifica all’utente prima della stampa |

|  |  |
| --- | --- |
| **ID: REQ-007** | |
| **Nome** | Gestione fatturazione |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Di seguito le specifiche da seguire per la gestione delle fatturazioni |
| **Sotto requisiti** | |
| **001** | Se dopo il primo e unico richiamo la persona non avesse ancora saldato la fattura, la persona sarà “bloccata” e non avrà più la possibilità di riservare i parcheggi sino al saldo della fattura. In questo caso la persona interessata sarà avvisata via email del suo blocco e del richiamo non ancora saldato |

|  |  |
| --- | --- |
| **ID: REQ-008** | |
| **Nome** | Convalida dati |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | L’applicativo avrà una convalida dei dati |
| **Sotto requisiti** | |
| **001** | La convalida dei dati immessi è una funzionalità dell’applicativo, p.es. l’email e il numero di telefono devono avere un formato corretto |

## Use case

Il diagramma mostra i vari casi d’uso a seconda del tipo di utente. Un utente registrato può avere più interazioni con il sito di un semplice visitatore.



## Pianificazione

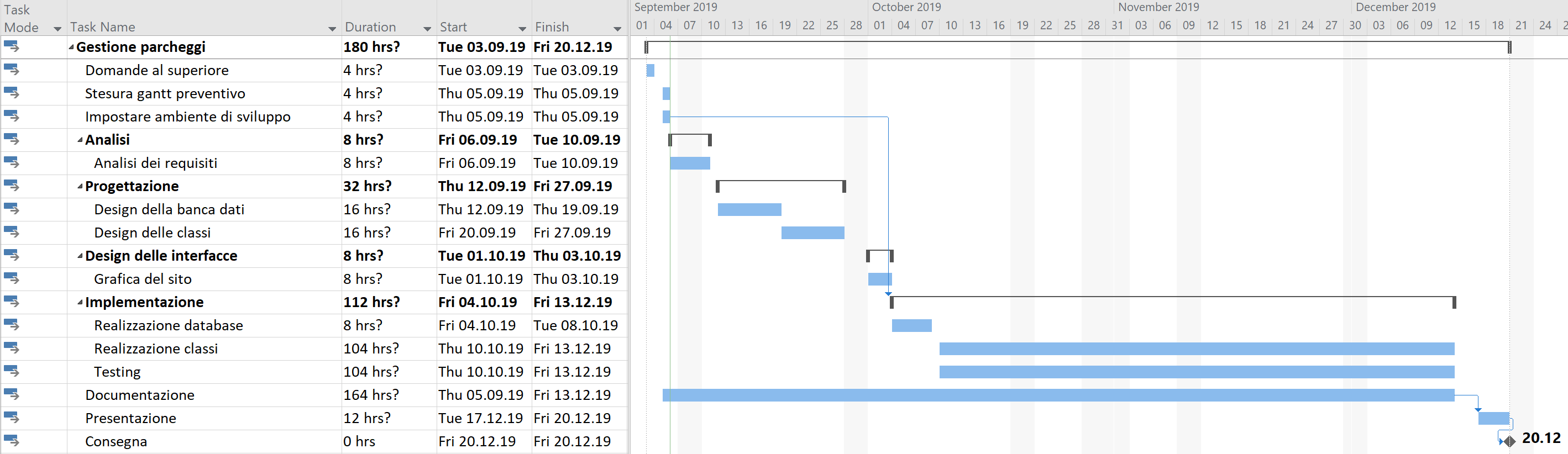


Figura 1 Gantt preventivo

## Analisi dei mezzi

### Software

I software utilizzati per la realizzazione di questo progetto sono:

* Microsoft Word 2016 - usato fino al 13/09/2019
* WPS Office Free 11.1.0.8865
* Visual Studio Code- usato fino al 15/10/2019
* PhpStorm-192.6817.20
* Project
* Google Chrome
* VMWare Workstation - usato fino al 20/09/2019
* XAMPP 7.3.0-0 - usato fino al 13/09/2019
* MySQL Workbench 8.0 CE
* draw.io online
* SourceTree - usato fino al 13/09/2019

### Hardware

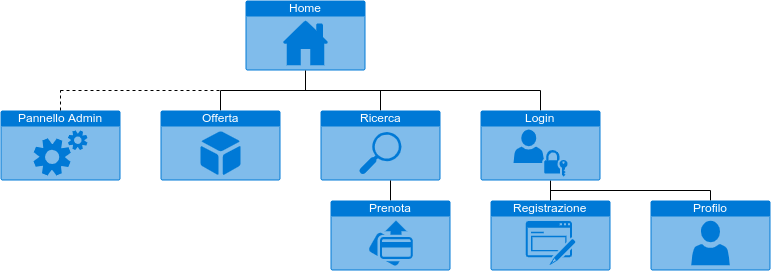
Computer Paolo Weishaupt:

* Modello: Huawei MateBook X Pro
* Processore: Intel Core i7-8550U
* RAM: 8GB
* Display 13.9” 3000x2000 px
* SSD: 512 GB
* OS: Windows 10 Home - usato fino al 10/09/2019
* OS: Deepin 15.11, distro basata su Debian 9 Stretch

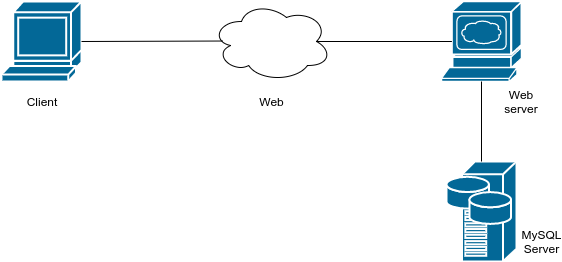
# Progettazione

## Design dell’architettura del sistema

### Sitemap

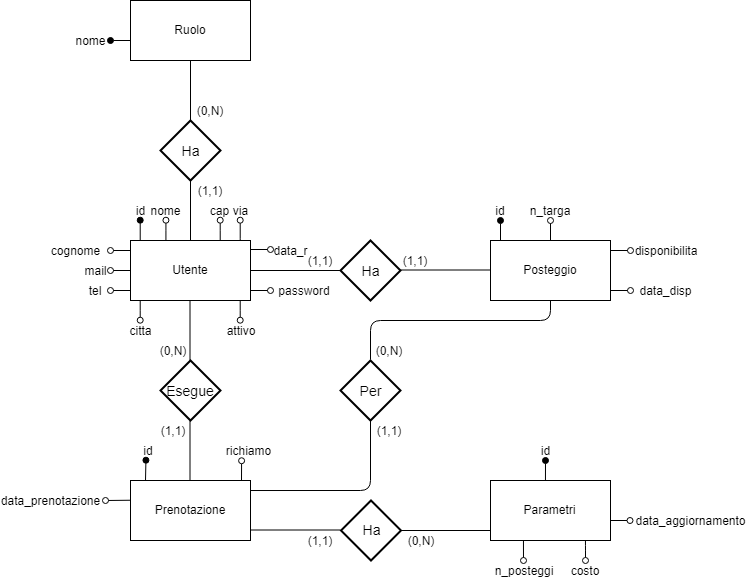


### Schema di rete



## Design dei dati e database

### Diagramma E-R



### Schema procedurale

Parametri(id, n\_posteggi, costo, data\_aggiornamento)

Ruolo(nome)

Utente(id, ruolo(FK), nome, cognome, mail, via, cap, citta, tel, data\_r, attivo, password, id\_posteggio(FK))

Posteggio(id, disponibilita, data\_disp, n\_targa)

Prenotazione(id, id\_utente(FK), id\_posteggio(FK), id\_parametri(FK), richiamo, data\_prenotazione)

### Descrizione database

Il database è composto da 5 entità.

L’entità PARAMETRI serve per contenere il numero di parcheggi disponibili, il costo di affitto di un parcheggio e la data dell’ultimo aggiornamento del costo.

L’entità RUOLO contiene solo un campo ed è il nome del ruolo. Questo perché se in futuro si dovesse avere il bisogno di cambiare i nomi dei ruoli non si dovrà cambiare manualmente per tutti gli utenti.

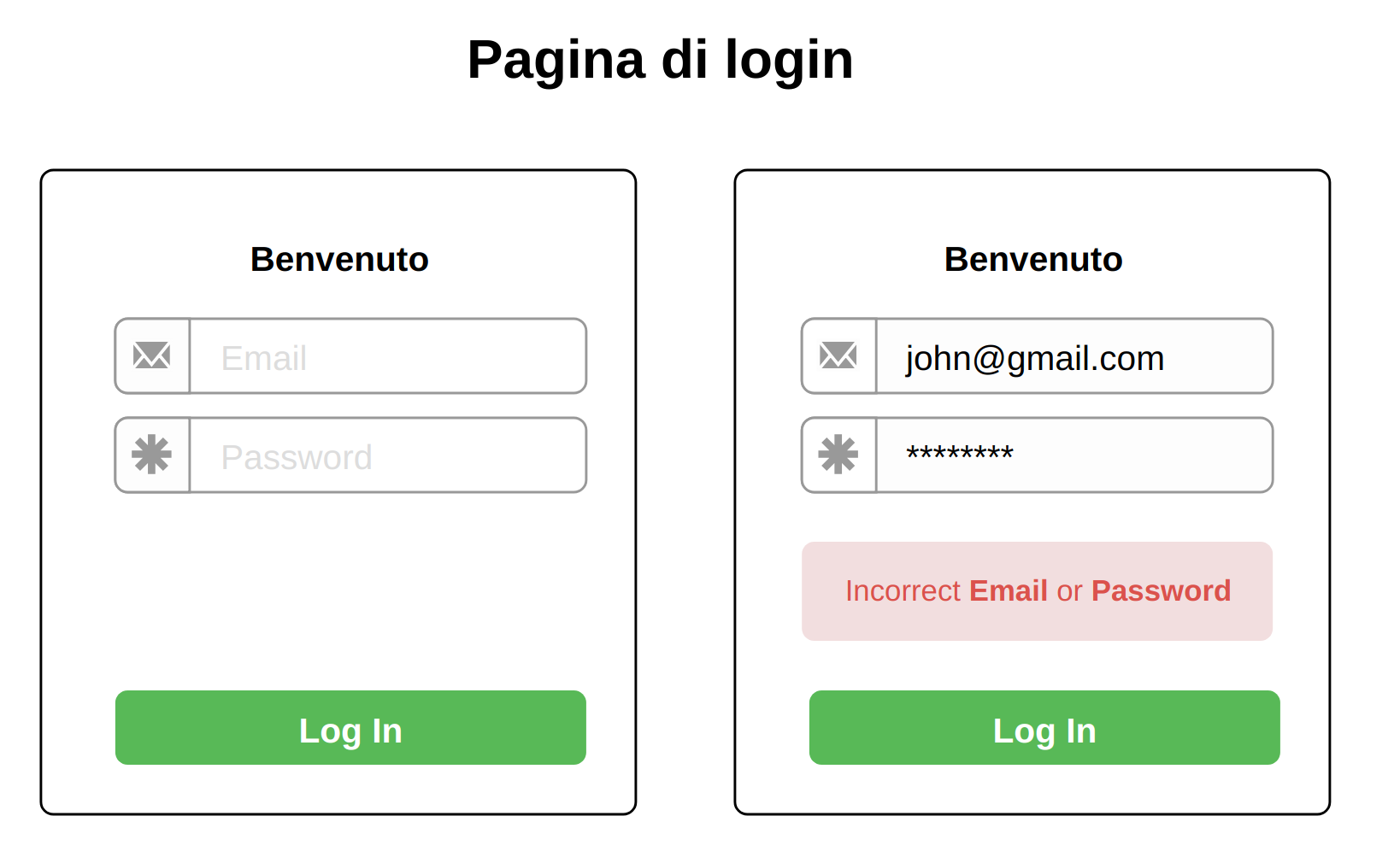
L’entità UTENTE rappresenta gli utenti del db e contiene le loro informazioni. L’attributo attivo serve per la verifica dell’e-mail perché finché non viene verificata il suo valore sarà su false e l’account sarà disabilitato. L’attributo potrà inoltre essere settato a false se l’utente dovesse avere dei richiami su una fattura.

L’entità POSTEGGIO rappresenta un posteggio e contiene i dati necessari al suo riconoscimento. Gli attributi disponibilita, data\_disp e n\_targa verranno utilizzati se il parcheggio dovrà essere messo in offerta.

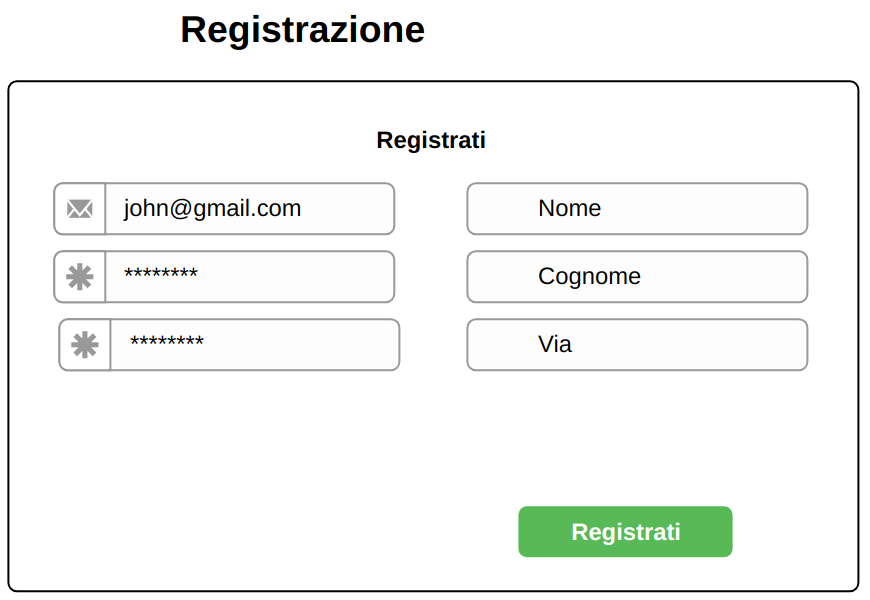
L’entità PRENOTAZIONE serve per gestire le prenotazioni degli utenti di un parcheggio. L’attributo richiamo ha di default il valore false. Se la fattura riguardante quella prenotazione non dovesse essere saldata verrà settato a true e al prossimo richiamo l’utente verrà disabilitato.

## Design delle interfacce

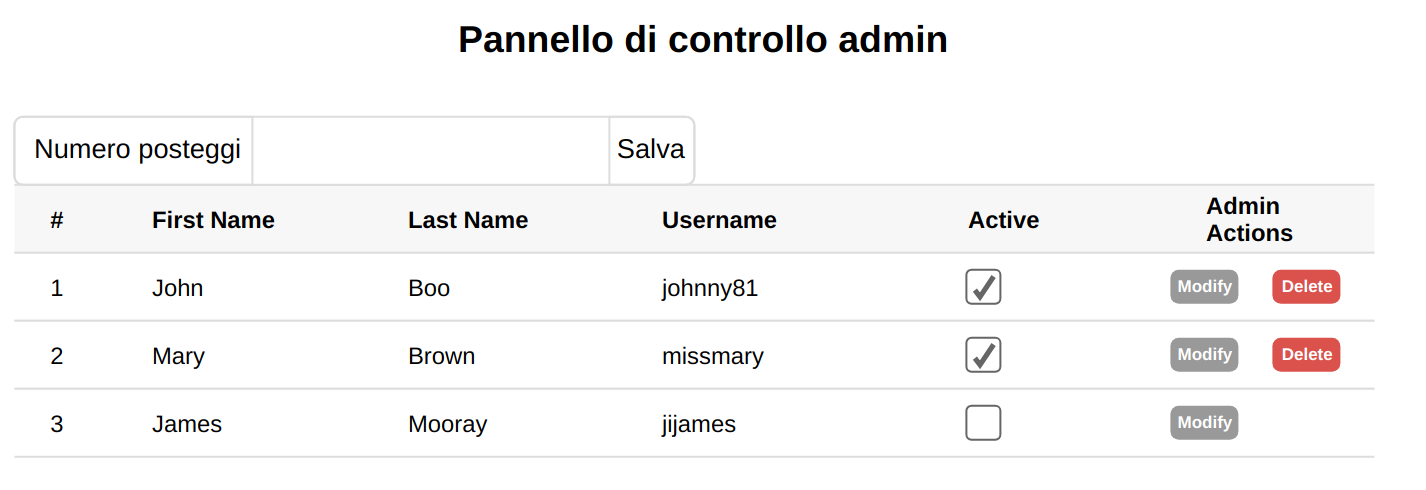
### Pagina di login



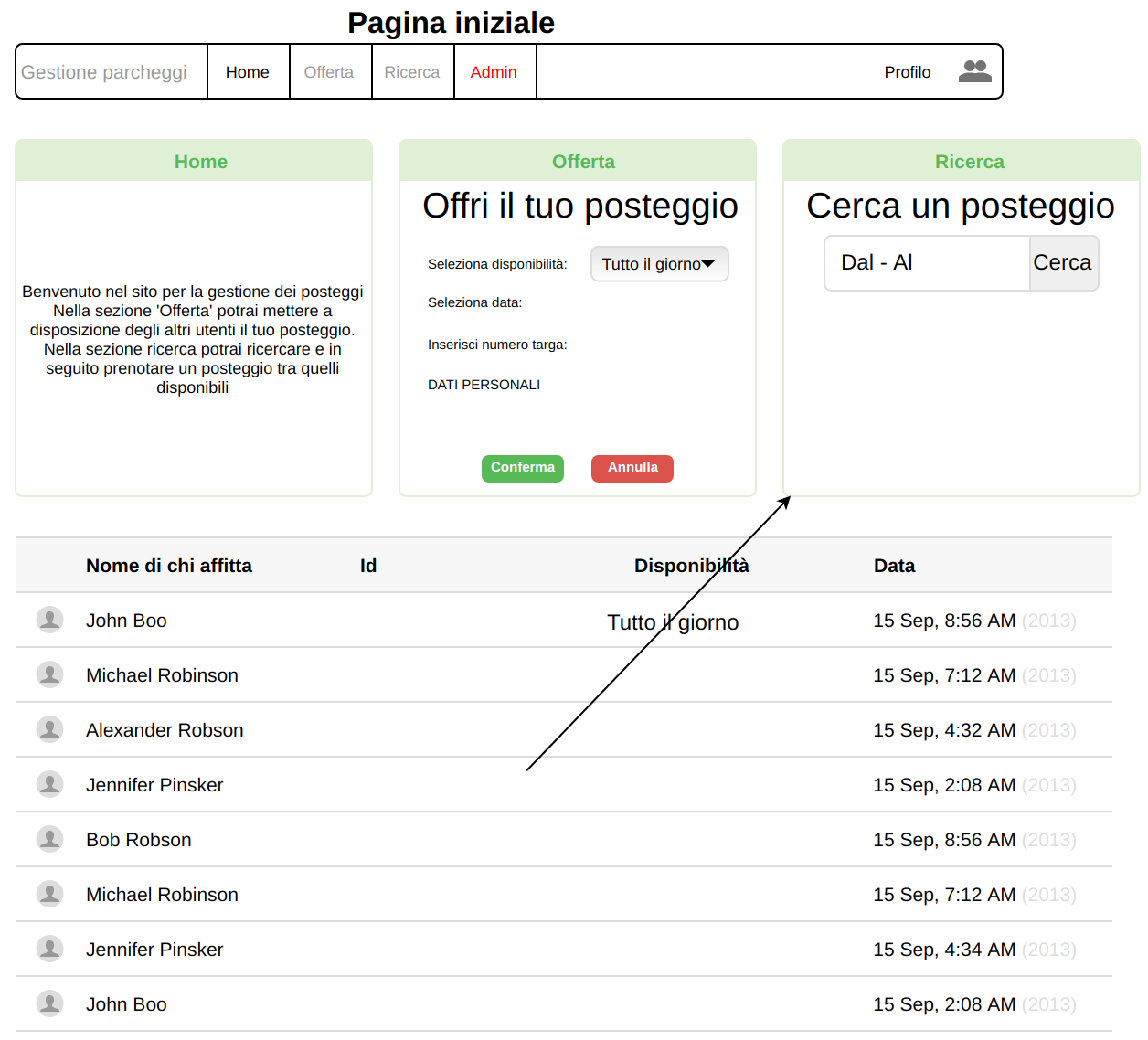
### Pagina di registrazione



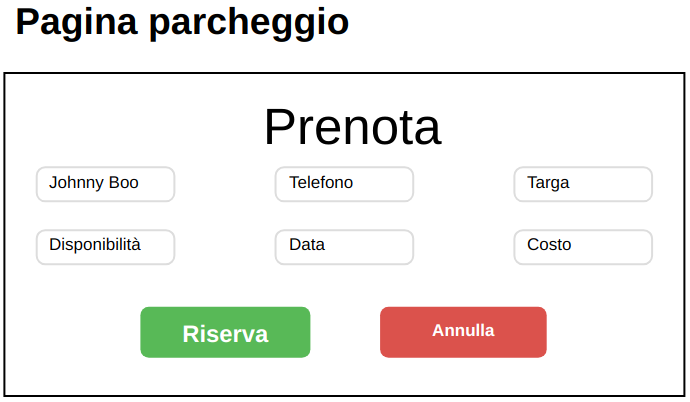
### Pannello di controllo dell’admin



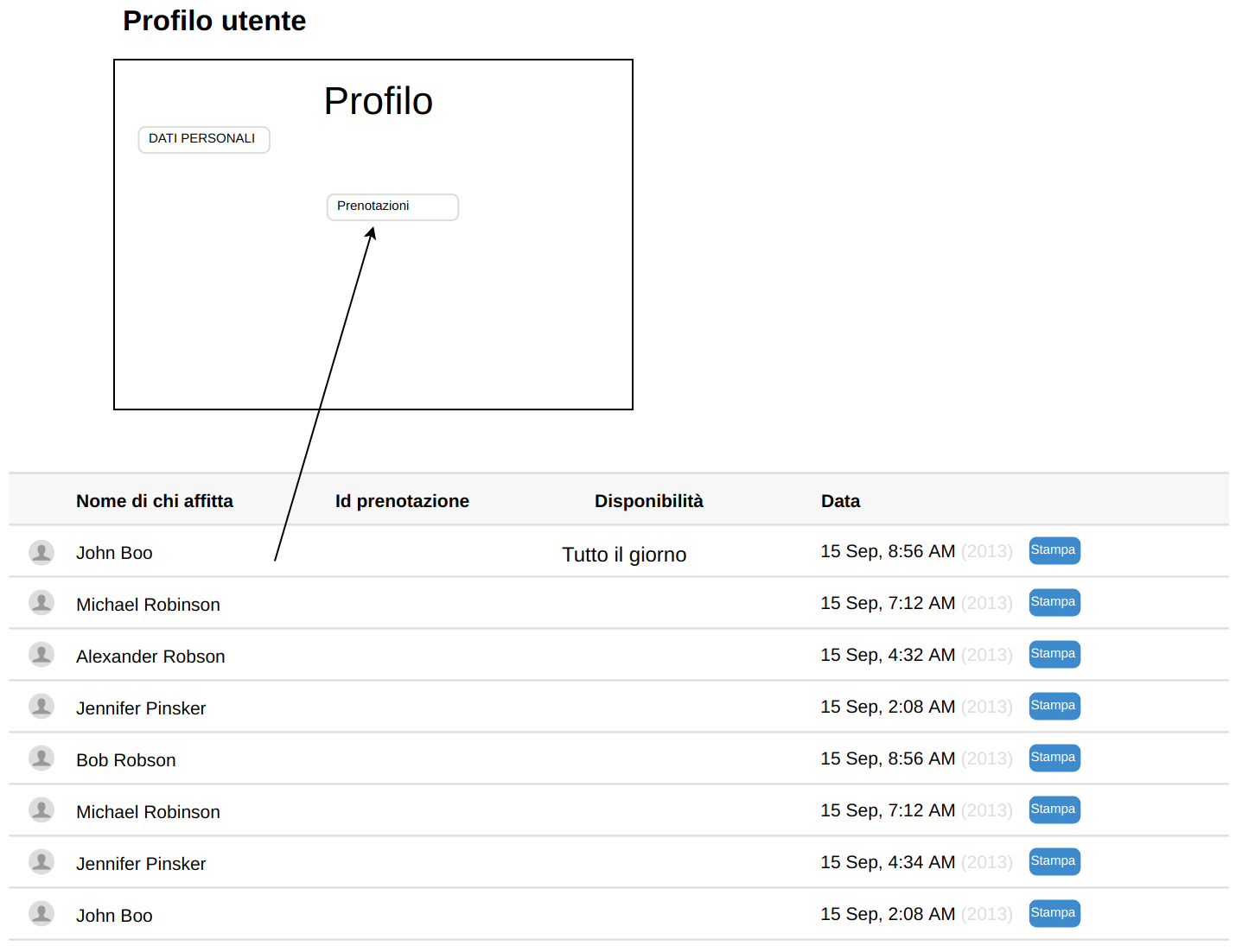
### Struttura del sito



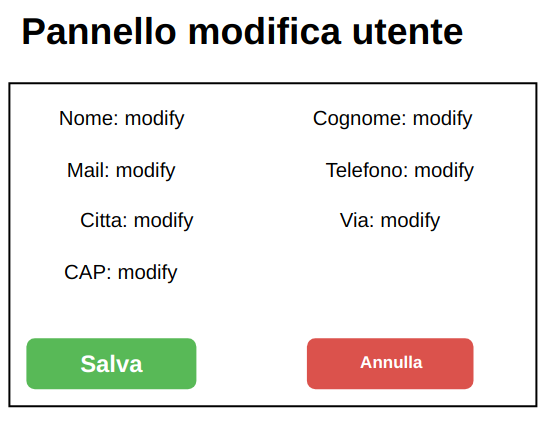
### Pagina delle informazioni del parcheggio



### Pagina di profilo dell’utente

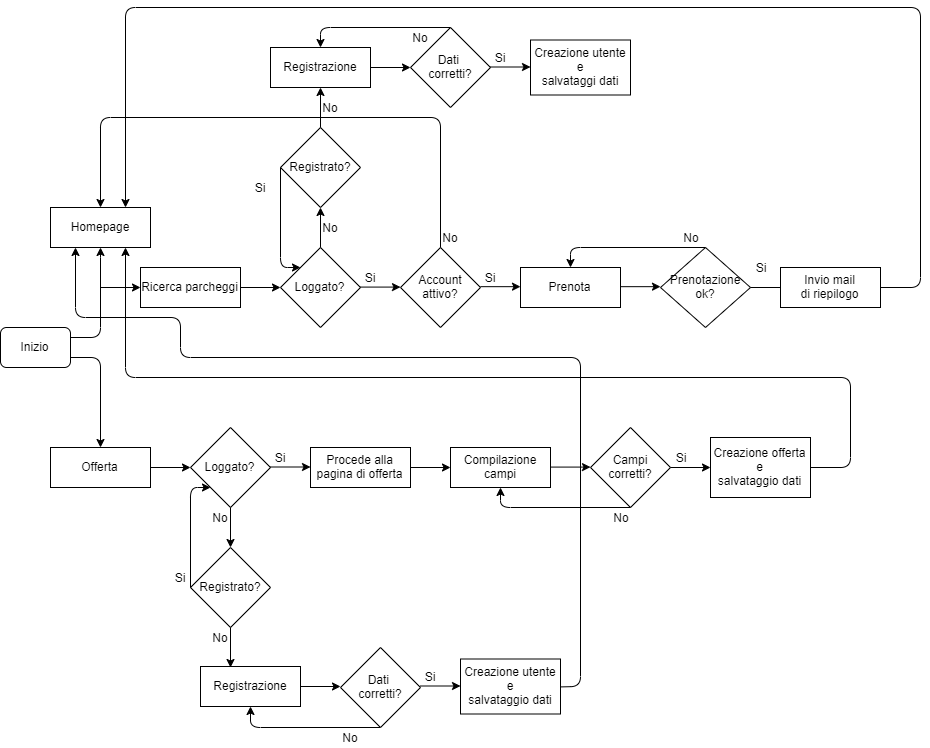


### Pagina per la modifica del proprio profilo



## Design procedurale

Il seguente diagramma spiega il funzionamento del sito.

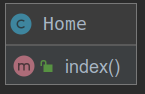


# Implementazione

## 4.1 Controllers

### 4.1.1 Home

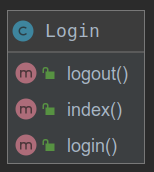
Il controller Home serve per la gestione di tutte le richieste relative alla homepage del sito.



#### 4.1.1.1 Index()

Tramite tramite questo verrà caricata la View della homepage.

### 4.1.2 Login



Il controller Login si occupa di tutta la logica per la gestione del login.

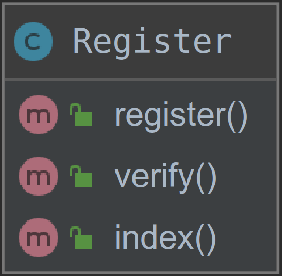
#### 4.1.2.1 logout()

Il metodo logout() si occupa della gestione del logout di un utente. Vengono azzerate tutte le variabili create nella sessione, si distrugge la sessione e poi viene eseguito un redirect alla pagina home.

#### 4.1.2.2 login()

Il metodo login() si occupa della gestione del login degli utenti. Viene salvata e validata la mail inserita dall’utente mentre la password viene cifrata con l’algoritmo sha256. Poi si interroga il database con le informazioni ottenute e se esistono si viene reindirizzati alla pagina home. Se invece non esistono si fa il redirect sulla stessa pagina che mostrerà un messaggio di errore.

### 4.1.3 Register



Il controller Register serve per la gestione della registrazione dei nuovi utenti.

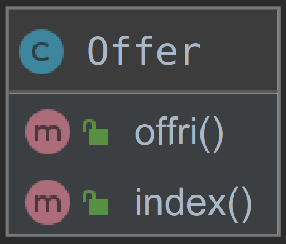
#### 4.1.3.1 register()

Il metodo register() si occupa di richiamare il metodo per la registrazione presente nel corrispondente model.

#### 4.1.3.2 verify()

Il metodo verify() serve per la gestione della verifica e dell’attivazione dell’account appena creato dal nuovo utente.

### 4.1.4 Offer



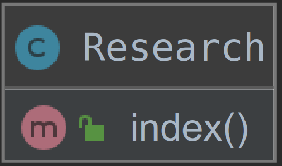
Il controller Offer gestisce la parte dedicata all’offerta di un parcheggio.

#### 4.1.4.1 offri()

Il metodo offri() si occupa di richiamare il metodo di offerta presente nel model.

#### 4.1.4.2 index()

Il metodo index() si occupa del caricamento della pagina di offerta. Per poter accedere a questa pagina l’utente deve avere effettuato il login al sito altrimenti verrà reindirizzato alla pagina per poterlo fare.



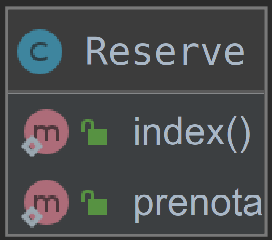
### 4.1.5 Research

Il controller Research si occupa della gestione della pagina di ricerca dei parcheggi disponibili.

#### 4.1.5.1 index()

Il metodo index() serve per caricare la tabella dei parcheggi disponibili. Se non vengono applicati dei filtri si avranno tutti i parcheggi disponibili in lista.

### 4.1.6 Reserve



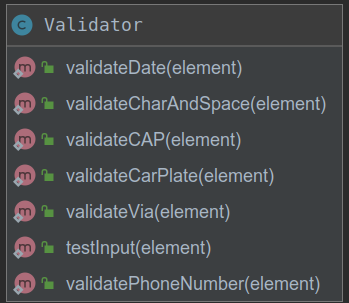
Il controller Reserve si occupa della gestione delle prenotazioni.

#### 4.1.6.1 index()

Il metodo index() prende l’id del parcheggio che si vuole prenotare, riceve le sue informazioni tramite l’apposito metodo nel model e carica la View per la prenotazione passandogli i dettagli del parcheggio.

#### 4.1.6.2 prenota()

Il metodo prenota() richiama il metodo per la prenotazione presente nel Model.



### 4.1.7 Validator

Il controller Validator racchiude i metodi per la validazione dei dati.

#### 4.1.7.1 validateDate(element)

Questo metodo serve per validare la data di un’offerta. Per essere valida una data non deve essere inferiore a quella corrente. Come parametro riceve la data da validare.

#### 4.1.7.2 validateCharAndSpace(element)

Questo metodo serve per controllare la presenza di caratteri speciali e numeri in una stringe che non li accetta. Per essere valida la stringa non deve contenerne. Come parametro riceve la stringa da controllare.

#### 4.1.7.3 validateCap(element)

Questo metodo serve per validare un CAP con il formato svizzero. Il formato accetta numeri composti da 4 cifre. Per essere valido un CAP deve avere 4 e solo 4 cifre. Come parametro riceve il CAP da controllare.

#### 4.1.7.4 validateCarPlate(element)

Questo metodo serve per la validazione di un numero di targa svizzero. Il formato è il seguente:

le due iniziali del cantone di provenienza + spazio + da 1 a 6 cifre.

Esempio: TI 123456

Come parametro riceve la targa da controllare.

#### 4.1.7.5 validateVia(element)

Questo metodo serve per validare una via. Il formato è il seguente:

(stringa + spazio) ripetuti anche più di una volta + da 1 a 3 cifre + un’eventuale lettera.

Esempio: Via Trevano 4

Come parametro riceve la via da validare.

#### 4.1.7.6 testInput(element)

Questo metodo serve per evitare injection. Si esegue un trim della stringa per togliere gli spazi, dopo si fa uno stripslashes per togliere gli slash e infine un htmlspecialchars che converta i caratteri speciali in entità html. Come parametro riceve la string da controllare.

#### 4.1.7.7 validatePhoneNumber(element)

Questo metodo serve per la validazione di un numero di telefono secondo il formato svizzero:

07X XXX XX XX

0041 7X XXX XX XX

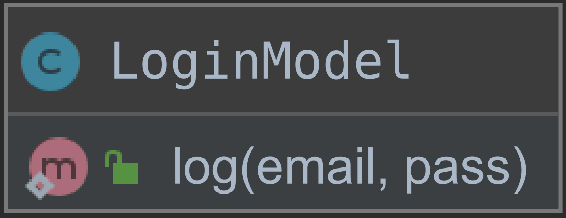
+41 7X XXX XX XX

Per essere valida una data deve essere in uno di questi 3 formati.

Come parametro riceve il numero di telefono da controllare.

## 4.2 Models

### 4.2.1 LoginModel



Questa classe gestisce il login degli utenti.

#### 4.2.1.1 log(email, password)

Questo metodo serve per eseguire il login di un utente.

|  |
| --- |
| $statement = *Database*::*get*()->prepare(**"select** *\** **from utente where mail=:email AND password=:pass"**);  *//inserisco i parametri* $statement->bindParam(**':email'**, $email, \PDO::***PARAM\_STR***); $statement->bindParam(**':pass'**, $pass, \PDO::***PARAM\_STR***);  $statement->execute(); $result = $statement->fetch(\PDO::***FETCH\_ASSOC***); |

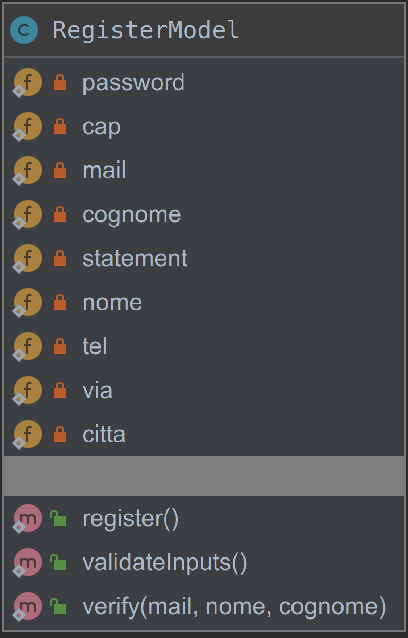
Preparo la query da eseguire inserendo i parametri. La query seleziona l’utente con la mail e la password immessi al momento del login. Con il metodo bindParam() assegno le variabili ai parametri nella query, la eseguo e inserisco il risultato in un array associativo.

|  |
| --- |
| **if**($result > 0)  {  *Auth*::*auth*();  $\_SESSION[**'user\_id'**] = $result[**'id'**];  $\_SESSION[**'ruolo'**] = $result[**'ruolo'**];  $\_SESSION[**'nome'**] = $result[**'nome'**];  $\_SESSION[**'id\_posteggio'**] = $result[**'id\_posteggio'**];   **return**; } **else** {  *Auth*::*logout*();  $\_SESSION[**'loginError'**] = **true**;  **return**; } |

Se il risultato della query è maggiore di 0 significa che esiste un utente registrato con quelle credenziali quindi eseguo l’autenticazione e setto alcune variabili nella sessione che mi serviranno.

Se invece non esiste nessuna corrispondenza nel database eseguo un logout e setto l’errore ‘loginError’ nella sessione che userò per notificare il falliimento della procedura di login.

### 4.2.2 RegisterModel



Questa classe contiene i metodi per la registrazione di un nuovo utente e la verifica della mail e dell’account.

#### 4.2.2.1 register()

Questo metodo esegue la registrazione dei un utente.

|  |
| --- |
| **if**($\_SERVER[**"REQUEST\_METHOD"**] === **"POST"**) {  **self**::*validateInputs*(); } |

Se la richiesta viene eseguita in post richiamo la funzione che raccoglie la validazione degli input. Prima facevo tutto in questo metodo, ma poi ho deciso di dividere le due parti per rendere più leggibile il codice.

|  |
| --- |
| **self**::*$statement* = *Database*::*get*()->prepare(**"insert into utente   (ruolo, nome, cognome, mail, via, cap, citta, tel, data\_r, attivo, password, id\_posteggio  ) values ('user', :nome, :cognome, :mail, :via, :cap, :citta, :tel,** *current\_timestamp***, false, :password, null); "**);  **self**::*$statement*->bindParam(**':nome'**, **self**::*$nome*, PDO::***PARAM\_STR***); **self**::*$statement*->bindParam(**':cognome'**, **self**::*$cognome*, PDO::***PARAM\_STR***); **self**::*$statement*->bindParam(**':mail'**, **self**::*$mail*, PDO::***PARAM\_STR***); **self**::*$statement*->bindParam(**':via'**, **self**::*$via*, PDO::***PARAM\_STR***); **self**::*$statement*->bindParam(**':cap'**, **self**::*$cap*, PDO::***PARAM\_INT***); **self**::*$statement*->bindParam(**':citta'**, **self**::*$citta*, PDO::***PARAM\_STR***); **self**::*$statement*->bindParam(**':tel'**, **self**::*$tel*, PDO::***PARAM\_STR***); **self**::*$statement*->bindParam(**':password'**, **self**::*$password*, PDO::***PARAM\_STR***); |

Preparo la query da eseguire e assegno le variabili ai parametri nella query.

|  |
| --- |
| **try** {  **self**::*$statement*->execute();  *MailModel*::*newUserMail*(**self**::*$mail*, **self**::*$nome*, **self**::*$cognome*);  ViewLoader::*load*(**'login/index'**); } **catch** (PDOException $e) {  ViewLoader::*load*(**'register/index'**); } |

Eseguo la query e se va a buon fine carico la pagina di login invio una mail di verifica all’utente altrimenti ricarico la pagina di registrazione.

#### 4.2.2.2 verify(mail, nome, cognome)

Questo metodo serve per verificare la mail di un utente. Prende la mail tramite una richiesta GET ottenuta quando l’utente clicca il link ricevuto per mail.

|  |
| --- |
| **self**::*$statement* = *Database*::*get*()->prepare(**"select mail, nome, cognome   from utente where mail=:mail; "**);  **self**::*$statement*->bindParam(**':mail'**, $mail, PDO::***PARAM\_STR***);  **self**::*$statement*->execute(); $result = **self**::*$statement*->fetch(\PDO::***FETCH\_ASSOC***); |

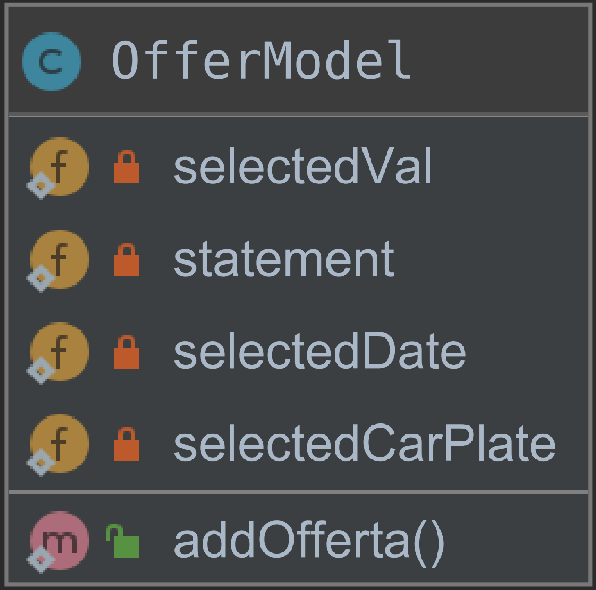
Eseguo una query richiedendo la mail dell’utente dove corrisponde a quella dell’utente da verificare.

|  |
| --- |
| **if**($result > 0){  **self**::*$statement* = *Database*::*get*()->prepare(**"update utente set attivo=true   where mail=:mail and attivo=false  "**);   **self**::*$statement*->bindParam(**':mail'**, $mail, PDO::***PARAM\_STR***);  **self**::*$statement*->execute();   ViewLoader::*load*(**'home/index'**, **array**(**'activationOK'**=>**"Il tuo account è stato attivato con successo!"**)); }**else**{  ViewLoader::*load*(**'home/index'**, **array**(**'activationNO'**=>**"L'URL è invalido o il tuo account è già attivo!"**)); } |

Se c’è una corrispondenza aggiorno il campo attivo dell’utente dove esso è settato a false e alla fine carico la home del sito con un messaggio di conferma dell’attivazione.

Se invece non dovesse essere trovata una corrispondenza significa che l’url usato dall’utente non è valido o che l’account dell’utente è già stato attivato.

### 4.2.3 OfferModel



Questa classe model serve per la gestione delle offerte dei parcheggi.

#### 4.2.3.1 addOfferta()

Questo metodo aggiungere un’offerta.

|  |
| --- |
| **if**(**isset**($\_POST[**'offri'**])){  **self**::*$selectedVal* = $\_POST[**'select\_disp'**];  **self**::*$selectedDate* = *Validator*::*validateDate*($\_POST[**'datepicker'**]);  **self**::*$selectedCarPlate* = *Validator*::*validateCarPlate*($\_POST[**'car\_plate'**]); } |

Se è stato cliccato il pulsante offri prendo il valore degli input e li valido tramite il metodo di validazione del controller Validator.

|  |
| --- |
| **if**(**isset**($\_SESSION[**'dateError'**]) || **isset**($\_SESSION[**'carPlateError'**])) {  *ViewLoader*::*load*(**'offerta/index'**);   **unset**($\_SESSION[**'dateError'**]);  **unset**($\_SESSION[**'carPlateError'**]); } **else** |

Se sono settate le variabili di sessione relative a degli errori nella validazione degli input carico la view di offerta che li mostrerà all’utente e unsetto quelle variabili.

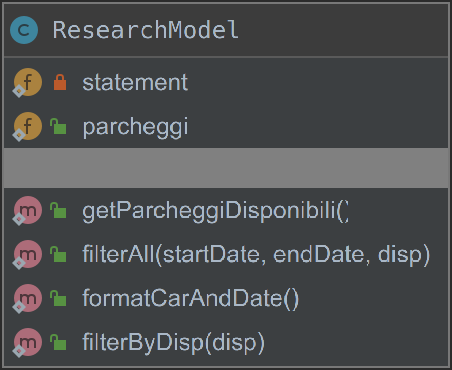
|  |
| --- |
| **else** {  **if**(*Users*::*hasParcheggio*() && $\_SESSION[**'active'**] == **true**)  { |

Se invece non ci sono errori controllo se l’utente ha un parcheggio e la sessione è attiva.

|  |
| --- |
| $inputDate = *date\_create*(**self**::*$selectedDate*); $inputDateFormat = *date\_format*($inputDate, **"Y-m-d H:i:s"**);  **self**::*$statement* = *Database*::*get*()->prepare(**"update posteggio set**  **disponibilita=:disponibilita, data\_disp=:data\_disp, n\_targa=:n\_targa where id=:id; "**);  **self**::*$statement*->bindParam(**":disponibilita"**, **self**::*$selectedVal*, PDO::***PARAM\_STR***);**self**::*$statement*->bindParam(**":data\_disp"**, $inputDateFormat, PDO::***PARAM\_STR***);**self**::*$statement*->bindParam(**":n\_targa"**, **self**::*$selectedCarPlate*, PDO::***PARAM\_STR***);**self**::*$statement*->bindParam(**":id"**, $\_SESSION[**'id\_posteggio'**], PDO::***PARAM\_INT***);  **try** {  **self**::*$statement*->execute();  *ViewLoader*::*load*(**'home/index'**, **array**(**'offertaOK'**=>**"Offerta andata a buon fine"**)); } **catch** (\PDOException $e) {  $\_SESSION[**'addPark'**] = **'Errore nel caricamento dell\'offerta!'**;  *ViewLoader*::*load*(**'offerta/index'**); } |

Se risultano vere le condizioni formatto la data inserita dall’utente e preparo la query da eseguire. La query serve per aggiornare il parcheggio offerto con i dati presi dagli input compilati dall’utente. Una volta eseguita la query se va a buon fine carico la homepage del sito con un messaggio di successo. Se invece la query non va a buon fine carico la view che mostrerà un messaggio di errore.

### 4.2.4 ResearchModel



La classe ResearchModel serve per la gestione della ricerca dei parcheggi disponibili.

#### 4.2.4.1 getParcheggiDisponibili()

Questo metodo ritorna tutti i posteggi disponibili.

|  |
| --- |
| **public static function** getParcheggiDisponibili() {  **self**::*$statement* = *Database*::*get*()->prepare(**"select** *\** **from posteggio where data\_disp is not null"**);  **self**::*$statement*->execute();   **self**::*$parcheggi* = **self**::*$statement*->fetchAll(\PDO::***FETCH\_ASSOC***);   **self**::*formatCarAndDate*(); } |

#### 4.2.4.2 filterAll(startDate, endDate, disp)

Questo metodo serve per eseguire il filtraggio dei dati quando l’utente setta le date tra le quali cercare e la disponibilità.

|  |
| --- |
| **public static function** filterAll($startDate, $endDate, $disp) {  $startDateFormat = *date\_format*(*date\_create*($startDate), **"Y-m-d H:i:s"**);  $endDateFormat = *date\_format*(*date\_create*($endDate), **"Y-m-d H:i:s"**);   **if**($startDateFormat > $endDateFormat)  {  $\_SESSION[**'minDateError'**] = **'Errore: la data iniziale è più grande di quella finale'**;  }  **else** {  **unset**($\_SESSION[**'minDateError'**]);  } |

Il metodo riceve la data di inizio filtraggio, quella di fine filtraggio e la disponibilità selezionata. Formatto le due date per poi controllare se quella di inizio è maggiore di quella di fine. Se risulta vero setto una variabile di errore nella sessione altrimenti la unsetto.

|  |
| --- |
| **if**($disp == **"Tutto"**) {  **self**::*$statement* = *Database*::*get*()->prepare(**"select** *\** **from posteggio where data\_disp between :start\_date and :end\_date and data\_disp is not null"**);  **self**::*$statement*->bindParam(**':start\_date'**, $startDateFormat, \PDO::***PARAM\_STR***);  **self**::*$statement*->bindParam(**':end\_date'**, $endDateFormat, \PDO::***PARAM\_STR***); } **else** {  **self**::*$statement* = *Database*::*get*()->prepare(**"select** *\** **from posteggio where data\_disp between :start\_date and :end\_date and disponibilita = :disp"**);  **self**::*$statement*->bindParam(**':disp'**, $disp, \PDO::***PARAM\_STR***);  **self**::*$statement*->bindParam(**':start\_date'**, $startDateFormat, \PDO::***PARAM\_STR***);  **self**::*$statement*->bindParam(**':end\_date'**, $endDateFormat, \PDO::***PARAM\_STR***); } **self**::*$statement*->execute();  **self**::*$parcheggi* = **self**::*$statement*->fetchAll(\PDO::***FETCH\_ASSOC***);  **self**::*formatCarAndDate*(); |

Se la disponibilità è settata su “Tutti” preparo la query con filtri le due date. Se invece il valore di disponibiilità è diverso da “Tutti” si esegue una query con filtri le date e la disponibilità. Alla fine chiamo il metodo per la formattazione delle date e della targa.

#### 4.2.4.3 filterByDisp()

Questo metodo serve per effettuare ill filtraggio dei parcheggi in base alla sua disponibilità.

|  |
| --- |
| **public static function** filterByDisp($disp) {  **if**($disp == **"Tutto"**)  {  **self**::*$statement* = *Database*::*get*()->prepare(**"select** *\** **from posteggio where data\_disp is not null"**);  }  **else** {  **self**::*$statement* = *Database*::*get*()->prepare(**"select** *\** **from posteggio where disponibilita = :disp"**);  **self**::*$statement*->bindParam(**':disp'**, $disp, \PDO::***PARAM\_STR***);  }  **self**::*$statement*->execute();   **self**::*$parcheggi* = **self**::*$statement*->fetchAll(\PDO::***FETCH\_ASSOC***);   **self**::*formatCarAndDate*(); } |

Se il valore della disponibilità è “Tutti” ritorno tutti i parcheggi mentre se è diverso ritorno i posteggi con quella data disponibilità.

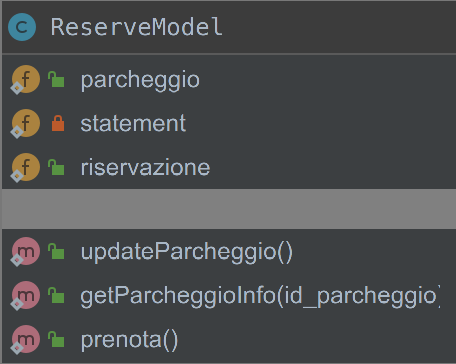
#### 4.2.4.4 formatCarAndDate()

Questo metodo serve per formattare le due date e la targa.

|  |
| --- |
| **public static function** formatCarAndDate() {  **foreach**(**self**::*$parcheggi* **as** &$row)  {  **if**(*is\_null*($row[**'n\_targa'**]))  {  $row[**'n\_targa'**] = **""**;  }  $inputDate = *date\_create*($row[**'data\_disp'**]);  $inputDateFormat = *date\_format*($inputDate, **'d-m-Y'**);  $row[**'data\_disp'**] = $inputDateFormat;  } } |

Per ogni parcheggio presente nella lista dei parcheggi se la targa è nulla le assegno come valore una stringa vuota e formatto la data della disponibilità nel formato che usiamo ovvero giorno-mese-anno.

### 4.2.5 ReserveModel



Questa classe gestisce la riservazione dei parcheggi.

#### 4.2.5.1 updateParcheggio()

Questo metodo serve per aggiornare lo stato di un parcheggio una volta prenotato.

|  |
| --- |
| **public static function** updateParcheggio() {  **self**::*$statement* = *Database*::*get*()->prepare(**"update posteggio   set disponibilita=null, data\_disp=null, n\_targa=null  where id=:id;  "**);   **self**::*$statement*->bindParam(**':id'**, $\_SESSION[**'id\_posteggio\_prenotato'**], \PDO::***PARAM\_INT***);  **self**::*$statement*->execute(); } |

Nella query aggiorno i campi del posteggio prenotato settandoli a null. Facendo così il posteggio non risulterà più disponibille.

#### 4.2.5.2 getParcheggioInfo(id\_parcheggio)

Questo metodo serve per ricavare le informazioni del parcheggioo selezionato dalla lista dei parcheggi disponibili.

|  |
| --- |
| **public static function** getParcheggioInfo($id\_parcheggio) {  $\_SESSION[**'id\_posteggio\_prenotato'**] = $id\_parcheggio;   **self**::*$statement* = *Database*::*get*()->prepare(  **"SELECT utente.nome, utente.cognome, utente.tel, parametri.costo, posteggio.data\_disp,  posteggio.disponibilita, posteggio.n\_targa  FROM utente, parametri, posteggio  WHERE posteggio.id = :id  AND utente.id\_posteggio = :id;  "**);  **self**::*$statement*->bindParam(**':id'**, $id\_parcheggio, \PDO::***PARAM\_INT***);  **self**::*$statement*->execute();  **self**::*$parcheggio* = **self**::*$statement*->fetch(\PDO::***FETCH\_ASSOC***); |

Il metodo riceve l’id del parcehggio selezionato. Salvo l’id in una variabile della sessione perchè mi servirà in seguito. La query che dovrò eseguire prende dalla tabella utenti il nome, il cognome e il telefono dell’utente che ha messo a disposizione il parcheggio, dalla tabella parametri il costo base di un parcheggio e dalla tabella posteggio la data della disponibilità, la disponibilità e la targa. Il tutto viene fatto in base all’id del posteggio. Dovrà essere lo stesso nella tabella posteggi e lo stesso assegnato all’utente.

|  |
| --- |
| **if**(*is\_null*(**self**::*$parcheggio*[**'n\_targa'**])) {  **self**::*$parcheggio*[**'n\_targa'**] = **"Targa non fornita"**; } $inputDate = *date\_create*(**self**::*$parcheggio*[**'data\_disp'**]); $inputDateFormat = *date\_format*($inputDate, **'d-m-Y'**); **if**(**self**::*$parcheggio*[**'disponibilita'**] == **"Mattina"** || **self**::*$parcheggio*[**'disponibilita'**] == **"Pomeriggio"**) {  **self**::*$parcheggio*[**'costo'**] /= 2; } **self**::*$parcheggio*[**'data\_disp'**] = $inputDateFormat; |

Dopo aver eseguito la query mi occupo di formattare i valori ottenuti.

Se la targa è nulla setto un messaggio di avviso. Formatto la data con il nostro formato. Se la disponibilità del parcheggio diversa da “Tutto il giorno” stampo il costo diviso due.

#### 4.2.5.3 prenota()

Questo metodo serve per salvare nel database la prenotazione di un parcheggio.

|  |
| --- |
| **public static function** prenota() {  **self**::*$statement* = *Database*::*get*()->prepare(**"insert into prenotazione   (richiamo, data\_prenotazione, id\_utente, id\_posteggio)  values  (false,** *current\_timestamp***, :id\_utente, :id\_posteggio);  "**);   **self**::*$statement*->bindParam(**':id\_utente'**, $\_SESSION[**'user\_id'**], \PDO::***PARAM\_INT***);  **self**::*$statement*->bindParam(**':id\_posteggio'**, $\_SESSION[**'id\_posteggio\_prenotato'**], \PDO::***PARAM\_INT***); |

Preparo la query per inserire una prenotazione nel database e assegno le variabili ai parametri.

|  |
| --- |
| **try** {  **if** (*Auth*::*isAuthenticated*())  {  **self**::*$statement*->execute();   **self**::*$statement* = *Database*::*get*()->prepare(**"select** *\** **from prenotazione  where id\_posteggio=:id\_posteggio and id\_utente=:id\_utente   order by data\_prenotazione desc limit 1"**);   **self**::*$statement*->bindParam(**':id\_utente'**, $\_SESSION[**'user\_id'**], \PDO::***PARAM\_INT***);  **self**::*$statement*->bindParam(**':id\_posteggio'**, $\_SESSION[**'id\_posteggio\_prenotato'**], \PDO::***PARAM\_INT***);  **self**::*$statement*->execute();  **self**::*$riservazione* = **self**::*$statement*->fetch(\PDO::***FETCH\_ASSOC***); |

Se l’utente ha eseguito il login eseguo la query e ne preparo una per riprendere i dati appena inseriti e li salvo nella variabile per la prenotazione.

|  |
| --- |
| **elf**::*$statement* = *Database*::*get*()->prepare(  **"SELECT posteggio.data\_disp, posteggio.disponibilita  FROM posteggio  WHERE posteggio.id = :id;  "**); **self**::*$statement*->bindParam(**':id'**, $\_SESSION[**'id\_posteggio\_prenotato'**], \PDO::***PARAM\_INT***); **self**::*$statement*->execute(); **self**::*$parcheggio* = **self**::*$statement*->fetch(\PDO::***FETCH\_ASSOC***); |

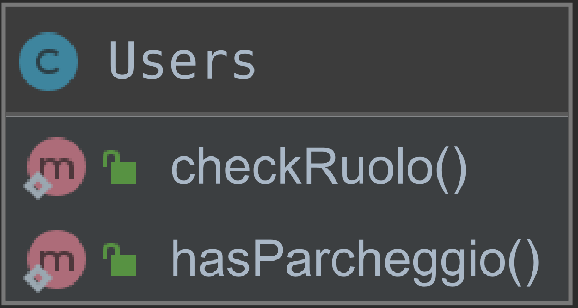
In seguito eseguo un’altra query per ricevere i dati del posteggio prenotato.

|  |
| --- |
| *MailModel*::*reservationMail*($\_SESSION[**'mail'**], $\_SESSION[**'nome'**], $\_SESSION[**'cognome'**], **self**::*$parcheggio*, **self**::*$riservazione*);  **self**::*updateParcheggio*(); **unset**($\_SESSION[**'id\_posteggio\_prenotato'**]); ViewLoader::*load*(**'home/index'**, **array**(**'prenotazioneOK'**=>**"Prenotazione avvenuta"**)); |

Le query precedenti mi servono per poter mandare una mail all’utente con un pdf contenente i dati di essa. Aggiorno il posteggio, unsetto la variabile di sessione contenente l’id del parchggio prenotato e alla fine carico la pagina home con un messaggio di successo.

|  |
| --- |
| }  **else** {  ViewLoader::*load*(**"login/index"**, **array**(**'prenotazioneNO'**=>**"Devi prima**  **registrarti"**));  } } **catch** (PDOException $e) {  ViewLoader::*load*(**'prenotazione/index'**, **array**(**'parcheggio'**=>**self**::*$parcheggio*, **'prenotazioneNO'**=>**"Prenotazione fallita"**)); } |

Se invece l’utente non ha eseguito il login lo redireziono alla pagina per eseguirlo. Se la prenotazione non è andata a buon fine carico la pagina di prenotazione con un messaggio di errore.



### 4.2.6 Users

Questa classe si occupa della gestione degli utenti.

#### 4.2.6.1 checkRuolo()

Questo metodo serve per controllare se un utente è un admin. Ritorna true se lo è, altrimenti false.

|  |
| --- |
| **public static function** checkRuolo() {  **if**(**isset**($\_SESSION[**'ruolo'**]) && $\_SESSION[**'ruolo'**] == **'admin'**)  {  **return true**;  }  **return false**; } |

Se la variabile di sessione è settata e ha il valore “admin” allora ritorna true, altrimenti false.

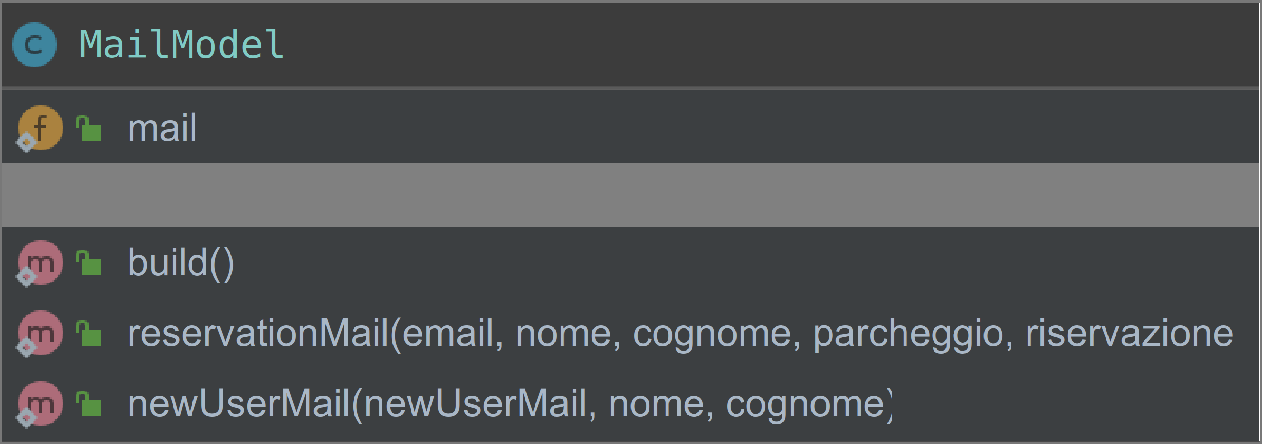
#### 4.2.6.2 hasParcheggio()

Questo metodo serve per controllare se un utente ha un parcheggio. Ritorna true se è vero, altrimenti false.

|  |
| --- |
| **public static function** hasParcheggio() {  **if**(**isset**($\_SESSION[**'id\_posteggio'**]))  {  **return true**;  }  **return false**; } |

Se la variabile di sessione è settata ritorna true, altrimenti false.

### 4.2.7 MailModel



La classe MailModel serve per la creazione e formattazione delle mail da inviare agli utenti.

#### 4.2.7.1 build()

Questo metodo serve per creare le mail e impostargli le informazioni di base.

|  |
| --- |
| **public static function** build() {  **self**::*$mail* = **new** *PHPMailer*(**true**);  **self**::*$mail*->**CharSet** = **'UTF-8'**;  **self**::*$mail*->isSMTP();  **self**::*$mail*->**Host** = **'smtp.gmail.com'**;  **self**::*$mail*->**Port** = 587;  **self**::*$mail*->**SMTPAuth** = **true**;  **self**::*$mail*->**Username** = **'gestioneparcheggi.samt@gmail.com'**;  **self**::*$mail*->**Password** = **'Parcheggi\_Admin\_2019'**;  **self**::*$mail*->setFrom(**'gestioneparcheggi.samt@gmail.com'**, **'Gestione parcheggi'**); } |

#### 4.2.7.2 reservationMail(email, nome, cognome, parcheggio, riservazione)

Questo metodo serve per impostare il testo della mail da inviare quando viene effettuata la prenotazione.

|  |
| --- |
| **public static function** reservationMail($email, $nome, $cognome, $parcheggio, $riservazione){  **self**::*build*();  **self**::*$mail*->addAddress($email, $nome.**" "**.$cognome);  **self**::*$mail*->**Subject** = **"Parcheggio prenotato"**;  **self**::*$mail*->**Body** = **"Ha ricevuto questa mail poichè lei ha effettuato la prenotazione di un posteggio.  Il allegato trova una copia della sua prenotazione."**;   **self**::*$mail*->addStringAttachment(*PDF*::*createPDF*($parcheggio, $riservazione), **'riservazione.pdf'**);  **self**::*$mail*->send(); } |

Alla mail aggiungo il testo del messaggio da inviare e allego il pdf contenente i dati della prenotazione effettuata.

#### 4.2.7.3 newUserMail(newUserMail, nome, cognome)

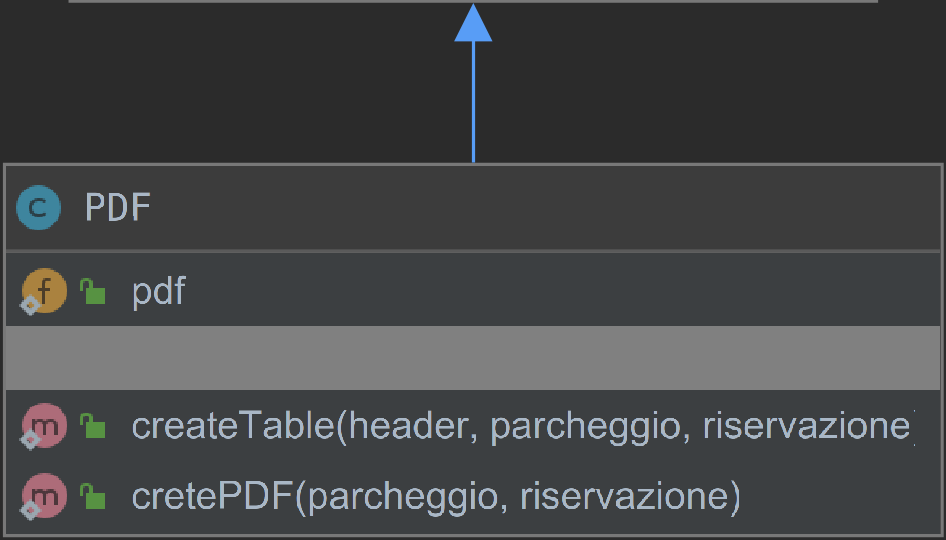
Questo metodo serve per impostare il testo della mail da inviare quando un utente effettua la registrazione.

|  |
| --- |
| **public static function** newUserMail($newUserMail, $nome, $cognome){  **self**::*build*();  **self**::*$mail*->addAddress($newUserMail, $nome.**" "**.$cognome);  **self**::*$mail*->**Subject** = **"Registrazione effettuata"**;  **self**::*$mail*->**Body** = **"Il tuo account è stato creato con successo!  Per poter essere in grado di usare il tuo account clicca sul seguente link per attivarlo:  https://samtinfo.ch/gestioneparcheggi2019/register/verify/?mail="**.$newUserMail.**"&nome="**.$nome.  **"&cognome="**.$cognome;  **self**::*$mail*->send(); } |

Alla mail aggiungo il testo del messaggio di benvenuto. Oltre a questo aggiungo il link per effettuare la verifica dell’account.



### 4.2.8 PDF



La classe PDF estende la classe FPDF. La classe FPDF appartiene all’omonima libreria per la creazione di file pdf.

#### 4.2.8.1 createTable(header, parcheggio, riservazione)

Funzione che crea la tabella con i dati del parcheggio e della riservazione.

|  |
| --- |
| $dateFormat = *date\_format*(*date\_create*($parcheggio[**'data\_disp'**]), **'d-m-Y'**); $prenDateFormat = *date\_format*(*date\_create*($riservazione[**'data\_prenotazione'**]), **'d-m-Y'**); |

Formatto le date nel nostro formato.

|  |
| --- |
| *// Va a capo* **self**::*$pdf*->Ln(10); *// Larghezza delle colonne* $w = **array**(45, 45, 45, 45); *// Header della tabella* **self**::*$pdf*->SetFont(**'Arial'**,**'B'**,12); **for**($i=0;$i<*count*($header);$i++)  *// Inserisco l'header* **self**::*$pdf*->Cell($w[$i],7,$header[$i],1,0,**'C'**); **self**::*$pdf*->Ln(); *// Font del body* **self**::*$pdf*->SetFont(**'Arial'**,**'B'**,10); *// Dati* **self**::*$pdf*->Cell($w[0],6,$riservazione[**'id'**],**'LR'**); **self**::*$pdf*->Cell($w[1],6,$prenDateFormat,**'LR'**); **self**::*$pdf*->Cell($w[2],6,$parcheggio[**'disponibilita'**],**'LR'**,0,**'R'**); **self**::*$pdf*->Cell($w[3],6,$dateFormat,**'LR'**,0,**'R'**); **self**::*$pdf*->Ln();  *// Chiusura della riga* **self**::*$pdf*->Cell(*array\_sum*($w),0,**''**,**'T'**); |

Vado a capo, setto la larghezza delle colonne, setto il font da usare e inserisco I dati dell’header. In seguito setto il font da usare nel corpo della tabella, inserisco I dati e chiudo la riga.

#### 4.2.8.2 createPDF(parcheggio, riservazione)

Funzione che serve per creare un PDF.

|  |
| --- |
| **self**::*$pdf* = **new** *FPDF*(); **self**::*$pdf*->AddPage(); **self**::*$pdf*->SetFont(**'Arial'**,**'B'**,13); **self**::*$pdf*->Cell(60,10,**'Dati della riservazione'**); |

Creo un nuovo oggetto FPDF, gli aggiungo una pagina, setto il font da usare e gli dico in che cella iniziare a scrivere.

|  |
| --- |
| $header = **array**(**'ID prenotazione'**, **'Data prenotazione'**, **'Disponibilita'**, **'Data disponibilita'**); **self**::*createTable*($header, $parcheggio, $riservazione); **return self**::*$pdf*->Output(**'S'**); |

Creo l’header da passare alla funzione che crea la tabella e la richiamo passandogli anche il parcheggio e la riservazione. Ritorno il PDF creato.

# Test

## Protocollo di test

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case:  **Riferimento**: | TC-001  REQ-001 ID-001 | Nome: | Ricerca dei parcheggi disponibili |
| Descrizione: | Gli utenti devono essere in grado di poter eseguire una ricerca dei parcheggi disponibili. | | |
| Prerequisiti: | Pagina di ricerca funzionante  Database funzionante  Collegamento al database | | |
| Procedura: | 1. Spostarsi sulla barra di navigazione del sito nella sezione ricerca 2. Eseguire un eventuale filtraggio dei parcheggi in base alla data e alla disponibilità | | |
| Risultati attesi: | Il risultato dovrebbe essere una tabella mostrante tutti I parcheggi disponibili. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case:  **Riferimento**: | TC-002  REQ-001 ID-002 | Nome: | Riservazione di un parcheggio |
| Descrizione: | Gli utenti devono essere in grado di poter riservare un parcheggio. | | |
| Prerequisiti: | Pagina di login funzionante  Pagina di ricerca funzionante  Pagina di riservazione funzionante  Database funzionante  Connessione al database funzionante  Account utente verificato | | |
| Procedura: | 1. Spostarsi sulla barra di navigazione del sito nella sezione ricerca. 2. Scegliere il parcheggio che si vuole riservare 3. Cliccare il pulsante “prenota” a destra della tabella 4. Ora ci ritroveremo in una pagina che riassume tutti i dati del parcheggio 5. Cliccare ancora una volta prenota | | |
| Risultati attesi: | Il risultato dovrebbe essere il redirect alla pagina home del sito che ci mostrerà una notifica verde in altro a destra. Inoltre, nella casella delle mail l’utente dovrà ricevere una mail con il pdf della ricevuta della prenotazione. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case:  **Riferimento**: | TC-003  REQ-001 ID-003 | Nome: | Stampa della prenotazioni |
| Descrizione: | Gli utenti devono essere in grado di poter stampare le loro riservazioni | | |
| Prerequisiti: | Aver effettuato una prenotazione | | |
| Procedura: | **Stampa della riservazione ricevuta per mail**   1. Entrare nella propria casella e-mail 2. Aprire la mail ricevuta dopo la prenotazione 3. Scaricare l’allegato e stamparlo   **Stampa delle vecchie prenotazioni sul sito**   1. Effettuare il login e dirigersi sul proprio profilo 2. Nella lista delle prenotazioni effettuate selezionare quella desiderata e stamparla | | |
| Risultati attesi: | La stampa del primo metodo funziona.  La stampa della riservazione deve partire. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case:  **Riferimento**: | TC-004  REQ-002 ID-001 | Nome: | Inserimento numero di parcheggi disponibili |
| Descrizione: | Gli amministratori devono poter essere in grado di inserire il numero esatto di parcheggi disponibili | | |
| Prerequisiti: | Pagina di login funzionante  Admin dashboard funzionante  Collegamento al database funzionante | | |
| Procedura: | 1. Spostarsi sulla barra di navigazione del sito nella sezione Admin dashboard avendo effettuato il login con un account amministratore 2. Inserire nell’apposito campo il numero di parcheggi | | |
| Risultati attesi: | Il campo nel database si aggiorna e il numero massimo di parcheggi disponibili diventa quello settato dell’amministratore. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case:  **Riferimento**: | TC-005  REQ-002 ID-002 | Nome: | Gestione degli utenti |
| Descrizione: | Gli amministratori devono essere in grado di poter gestione gli account degli utenti normali dalla dashboard admin. | | |
| Prerequisiti: | Pagina di login funzionante  Admin dashboard funzionante  Collegamento al database funzionante | | |
| Procedura: | 1. Spostarsi sulla barra di navigazione del sito nella sezione Admin dashboard avendo effettuato il login con un account amministratore 2. Selezionare un utente dalla tabella degli utenti 3. Modificare il nome dell’utente 4. Salvare | | |
| Risultati attesi: | Il nome di quell’utente deve cambiare | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case:  **Riferimento**: | TC-006  REQ-003 ID-001 | Nome: | Offerta del parcheggio |
| Descrizione: | Gli utenti dovranno essere in grado di mettere a disposizione degli altri utenti il loro parcheggio | | |
| Prerequisiti: | Pagina di offerta funzionante  Database funzionante  Collegamento al database | | |
| Procedura: | 1. Effettuare il login come utente. L’utente deve avere un parcheggio 2. Entrare nella sezione Offerta 3. Compilare i campi 4. Premere il pulsante offri | | |
| Risultati attesi: | Verrà eseguito un redirect alla pagina home con un messaggio di successo mostrato in verde. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case:  **Riferimento**: | TC-007  REQ-003 ID-002 | Nome: | Compilazione del form di offerta |
| Descrizione: | Gli utenti devono inserire la disponibilità del loro parcheggio, la data e il loro numero di targa | | |
| Prerequisiti: | Pagina di offerta funzionante  Database funzionante  Collegamento al database | | |
| Procedura: | 1. Effettuare il login come utente. L’utente deve avere un parcheggio 2. Entrare nella sezione Offerta 3. Scegliere la disponibilità del parcheggio dall’apposita select 4. Inserire la data della disponibilità con il datepicker 5. Inserire il proprio numero di targa | | |
| Risultati attesi: | L’utente è in grado di inserire i valori. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case:  **Riferimento**: | TC-008  REQ-003 ID-003 | Nome: | Filtraggio dei parcheggi disponibili |
| Descrizione: | Nella sezione “Ricerca” si potranno ricercare i parcheggi a disposizione inserendo come parametro il giorno preciso o anche un periodo, p.es. dal 2.12.2019 al 5.12.2019 | | |
| Prerequisiti: | Pagina di ricerca funzionante  Database funzionante  Collegamento al database | | |
| Procedura: | 1. Spostarsi nella pagina di ricerca 2. Inserire una data di inizio ricerca 3. Inserire una data di fine ricerca maggiore a quella di inizio 4. Inserire la disponibilità 5. Cliccare il pulsante cerca | | |
| Risultati attesi: | La pagina mostrerà solo i parcheggi specificati nel filtro dall’utente. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case:  **Riferimento**: | TC-009  REQ-003 ID-004 | Nome: | Ordinamento dei risultati |
| Descrizione: | Gli utenti devono essere in grado di ordinare i dati nella tabella dei parcheggi disponibili | | |
| Prerequisiti: | Pagina di ricercar funzionante  Database funzionante  Collegamento al database | | |
| Procedura: | 1. Spostarsi nella pagina di ricerca 2. Cliccare l’header della sezione per la quale si vuole ordinare(p. es. Data disponibilità) | | |
| Risultati attesi: | La tabella ordinerà i dati in ordine ascendente e se ricliccato discendente. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case:  **Riferimento**: | TC-010  REQ-004 ID-001 | Nome: | Campi della maschera di login |
| Descrizione: | I campi minimi da compilare saranno: nome, cognome, e-mail e un recapito telefonico | | |
| Prerequisiti: | Pagina di login funzionante  Database funzionante  Collegamento al database | | |
| Procedura: | 1. Spostarsi nella pagina di login cliccando in alto a destra 2. Compilare tutti i campi tranne uno di quelli obbligatori a rotazione | | |
| Risultati attesi: | Il login non funziona e si rimane alla pagina di login con gli errori effettuati. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case:  **Riferimento**: | TC-011  REQ-004 ID-002 | Nome: | Conferma registrazione |
| Descrizione: | L’applicativo dovrà prevedere un meccanismo di conferma della registrazione per l’identificazione della persona, p.es. l’invio di un link all’indirizzo di posta elettronica della persona, impostato in fase di registrazione | | |
| Prerequisiti: | PHPMailer funzionante  Pagina di registrazione funzionante  Connessione a internet | | |
| Procedura: | 1. Effettuare la registrazione al sito 2. Nella mail ricevuta cliccare il link fornito | | |
| Risultati attesi: | L’utente deve essere reindirizzato alla pagina home del sito con un messaggio di successo in verde. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case:  **Riferimento**: | TC-012  REQ-005 ID-001 | Nome: | Fatturazione |
| Descrizione: | Fatturazione mensile | | |
| Prerequisiti: | Database funzionante  Collegamento al database | | |
| Procedura: |  | | |
| Risultati attesi: | Ogni mese l’utente riceverà una mail con il saldo da pagare | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case:  **Riferimento**: | TC-013  REQ-005 ID-002 | Nome: | Mail alla riservazione |
| Descrizione: | Gli utenti riceveranno una mail di conferma alla riservazione di un parcheggio | | |
| Prerequisiti: | Pagina di prenotazione e ricerca funzionante  Database funzionante  Collegamento al database | | |
| Procedura: | 1. Effettuare una prenotazione | | |
| Risultati attesi: | L’utente riceverà una mail con un pdf della riservazione. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case:  **Riferimento**: | TC-014  REQ-005 ID-003 | Nome: | Concorrenzialità durante la prenotazione |
| Descrizione: | Prevedere un meccanismo di concorrenzialità per evitare che una persona in procinto di riservare il parcheggio, si veda portar la sua riservazione da un’altra persona. | | |
| Prerequisiti: | Pagina di ricerca e prenotazione funzionante  Database funzionante  Collegamento al database | | |
| Procedura: | 1. Scegliere il parcheggio da prenotare 2. Eseguire il login o la registrazione | | |
| Risultati attesi: | Un altro utente non vedrà disponibile il parcheggio che l’utente registrato sta vedendo | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case:  **Riferimento**: | TC-015  REQ-006 ID-001 | Nome: | Stampa delle riservazioni per periodo |
| Descrizione: | Gli amministratori devono essere in grado di stampare lo stato dei parcheggi per un determinato periodo | | |
| Prerequisiti: | Admin dashboard funzionante  Database funzionante  Collegamento al database | | |
| Procedura: | 1. Eseguire il login come amministratore 2. Spostarsi nella dashboard 3. Immettere il range di date per le quali stampare 4. Cliccare stampa | | |
| Risultati attesi: | Il risultato dovrebbe essere un foglio pdf stampabile | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case:  **Riferimento**: | TC-016  REQ-006 ID-002 | Nome: | Stampa fatture |
| Descrizione: | Gli amministratori potranno stampare la fattura per una persona singola o per tutte le persone, scegliendo un periodo da loro impostato, generalmente una stampa mensile. Il saldo della fattura è di 30 giorni netto | | |
| Prerequisiti: | Admin dashboard funzionante  Database funzionante  Collegamento al database | | |
| Procedura: | 1. Eseguire il login come amministratore 2. Spostarsi nella dashboard 3. Selezionare se si vuole fare una stampa per ogni persona o solo per una 4. Impostare il range di date 5. Cliccare stampa | | |
| Risultati attesi: | Il risultato dovrebbe essere un foglio pdf stampabile | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case:  **Riferimento**: | TC-017  REQ-006 ID-003 | Nome: | Stampa riservazione utenti |
| Descrizione: | Gli utenti devono essere in grado di poter stampare la propria riservazione | | |
| Prerequisiti: | Aver effettuato almeno una prenotazione | | |
| Procedura: | Vedere TC-003 | | |
| Risultati attesi: | Vedere TC-003 | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case:  **Riferimento**: | TC-018  REQ-006 ID-004 | Nome: | Stampa richiami |
| Descrizione: | Gli amministratori potranno stampare i richiami per le persone che non hanno ancora saldato la loro riservazione. A metà di ogni mese, giorno feriale, verranno stampati i richiami delle fatture. Prevedere un meccanismo automatico con notifica all’utente prima della stampa | | |
| Prerequisiti: | Admin dashboard funzionante  Database funzionante  Collegamento al database | | |
| Procedura: | 1. Eseguire il login come amministratore 2. Spostarsi nella dashboard 3. Cliccare il pulsante stampa richiami | | |
| Risultati attesi: | Il risultato deve essere un pdf stampabile. L’utente riceverà una mail che avvisa della stampa. Ogni metà mese verrà generato un pdf stampabile dei richiami. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case:  **Riferimento**: | TC-019  REQ-007 ID-001 | Nome: | Gestione richiami |
| Descrizione: | Se dopo il primo e unico richiamo la persona non avesse ancora saldato la fattura, la persona sarà “bloccata” e non avrà più la possibilità di riservare i parcheggi sino al saldo della fattura. In questo caso la persona interessata sarà avvisata via email del suo blocco e del richiamo non ancora saldato | | |
| Prerequisiti: | Aver eseguito almeno una prenotazione | | |
| Procedura: | 1. Non pagare la fattura mensile 2. Provare a eseguire una riservazione | | |
| Risultati attesi: | L’utente non deve essere in grado di prenotare il parcheggio e riceverà una mail. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case:  **Riferimento**: | TC-020  REQ-008 ID-001 | Nome: | Convalida degli input |
| Descrizione: | Gli input devono accettare solo certi tipi di dati | | |
| Prerequisiti: | Sito funzionante | | |
| Procedura: | 1. Spostarsi in una pagina che ha degli input 2. Immettere un valore diverso da quello specificato | | |
| Risultati attesi: | L’utente deve essere notificato con una notifica di errore rossa dell’errore di immissione. | | |

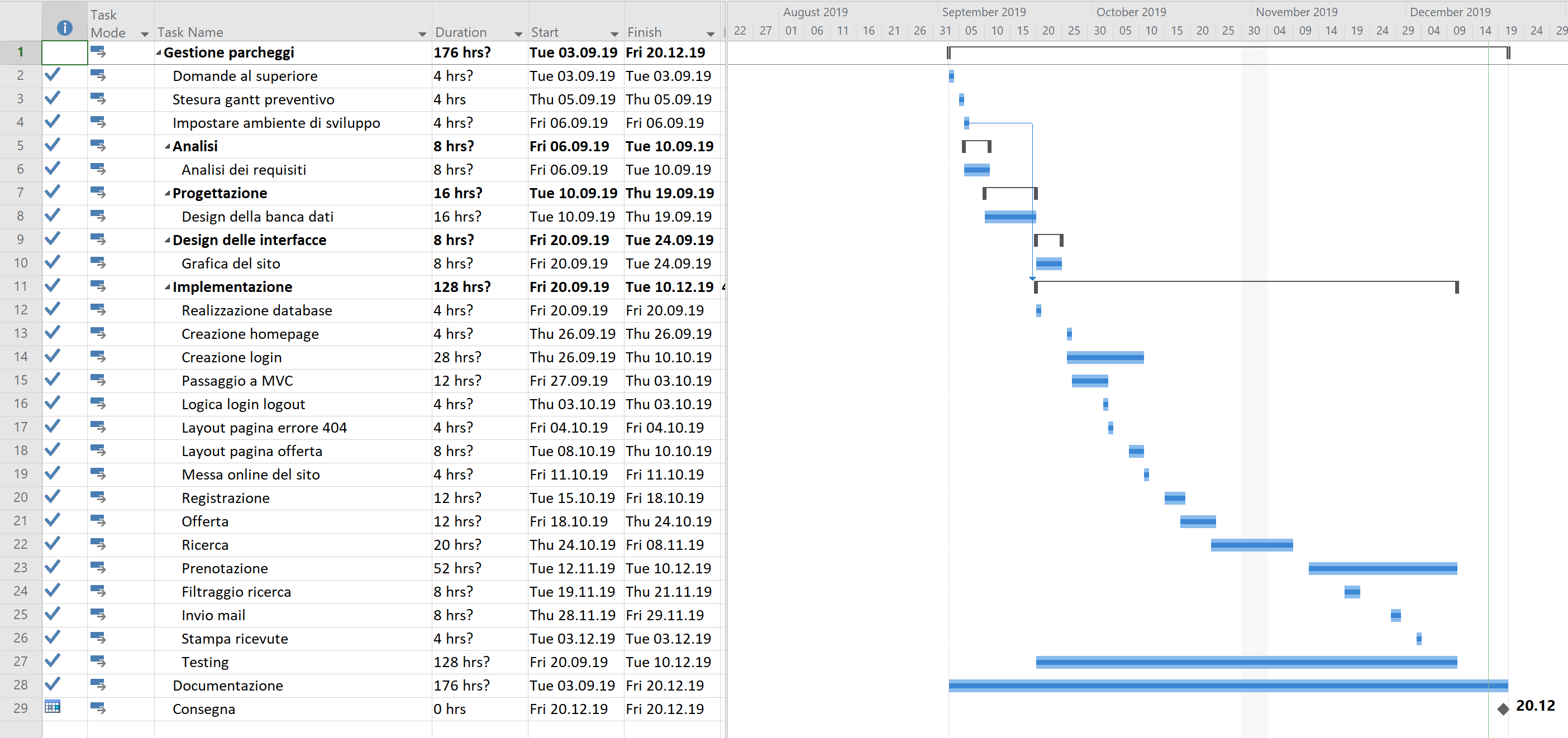
## Risultati test

|  |  |  |
| --- | --- | --- |
| **Test Case** | **Risultato** | **Note** |
| TC-001 | Funzionante |  |
| TC-002 | Funzionante |  |
| TC-003 | Funzionante parzialmente | L’utente può stampare il pdf che riceve per mail, ma non ha una pagina di profilo dalla quale avere uno storico delle prenotazioni. Dovrebbe andare a cercare le vecchie mail. |
| TC-004 | Non funzionante | Non implementato. |
| TC-005 | Non funzionante | Non implementato. |
| TC-006 | Funzionante |  |
| TC-007 | Funzionante |  |
| TC-008 | Funzionante |  |
| TC-009 | Funzionante |  |
| TC-010 | Funzionante |  |
| TC-011 | Funzionante |  |
| TC-012 | Non funzionante | Non implementato |
| TC-013 | Funzionante |  |
| TC-014 | Non funzionante | Non implementato |
| TC-015 | Non funzionante | Non implementato. |
| TC-016 | Non funzionante | Non implementato. |
| TC-017 | Funzionante parzialmente | Vedere risultato TC-003. |
| TC-018 | Non funzionante | Non implementato. |
| TC-019 | Non funzionante | Non implementato. |
| TC-020 | Funzionante |  |

## Mancanze/limitazioni conosciute

Il progetto non è stato portato a termine. Manca il pannello di gestione degli amministratori, la pagina del profilo dell’utente dove venivano mostrati i suoi dati e le sue vecchie prenotazioni, manca la gestione delle fatturazioni, manca la possibilità dell’amministratore di scegliere quante ricevute stampare, manca la gestione della concorrenzialità durante una riservazione.

# Consuntivo



# Conclusioni

Questo progetto secondo me può essere un valido rimpiazzo per l’attuale metodo di gestione dei posteggi qui a scuola. Ovviamente prima di poter entrare completamente in produzione deve essere completato in tutte le sue parti. Non è stato un lavoro pesante anzi, mi ha interessato molto. Sicuramente una cosa sulla quale dovrò lavorare è la gestione dei tempi perché non credo che avrei finito il progetto, ma qualche attuale mancanze forse avrei potuto implementarla.

## Sviluppi futuri

Negli sviluppi futuri si potrebbero aggiungere le parti che mancano.

## Considerazioni personali

Questo progetto è stato il primo sito che ho sviluppato e caricato online con la gestione di un database. Da questo punto di vista è stata una cosa abbastanza semplice da assimilare. Credo che per il secondo semestre rimarrò su questo tipo di progetto.

# 8 Glossario

|  |  |
| --- | --- |
| distro | Distribuzione personalizzata |
| sha256 | Algoritmo di cifratura a 256 bit |
| php | PHP Hypertext Preprocessor, linguaggio di programmazione lato server |
| query | Azione da eseguire sul database |
| GET | Richiesta eseguita per ricevere dati tramite l’URI |

# 9 Bibliografia

## 9.1 Sitografia

* <http://www.fpdf.org/>, libreria per la creazione dei pdf
* <https://github.com/PHPMailer/PHPMailer>, libreria per l’invio di e-mail
* <https://stackoverflow.com/>, per la risoluzione di molti dei problemi sorti durante il corso del progetto
* <https://php.net/>, documentazione di php
* <https://github.com/filippofinke/php-mvc/>, template mvc utilizzato

# 10 Allegati

Allegati cartacei:

* Diari di lavoro
* Quaderno dei compiti
* Abstract
* Credenziali

Allegati digitali:

* Codice sorgente del prodotto su GitHub:
  + <https://github.com/PaoloWeishaupt/gestioneparcheggi2019>
* Documentazione digitale assieme a diari, Quaderno dei Compiti e Abstract