Gestione parcheggi

[1 Introduzione 4](#_Toc1080242746)

[1.1 Informazioni sul progetto 4](#_Toc1008173754)

[1.2 Abstract 4](#_Toc1300567883)

[1.3 Scopo 4](#_Toc1592800038)

[2 Analisi 5](#_Toc292868470)

[2.1 Analisi del dominio 5](#_Toc705037476)

[2.2 Analisi e specifica dei requisiti 5](#_Toc929985182)

[2.3 Use case 8](#_Toc1255931285)

[2.4 Pianificazione 9](#_Toc471564429)

[2.5 Analisi dei mezzi 9](#_Toc562812654)

[2.5.1 Software 9](#_Toc2073412985)

[2.5.2 Hardware 9](#_Toc52540529)

[3 Progettazione 10](#_Toc1527601008)

[3.1 Design dell’architettura del sistema 10](#_Toc1200791227)

[3.1.1 Sitemap 10](#_Toc105382612)

[3.1.2 Schema di rete 10](#_Toc2127391356)

[3.2 Design dei dati e database 11](#_Toc1859426750)

[3.2.1 Diagramma E-R 11](#_Toc306413450)

[3.2.2 Schema procedurale 12](#_Toc1321395416)

[3.2.3 Descrizione database 12](#_Toc8067039)

[3.3 Design delle interfacce 13](#_Toc553438104)

[3.3.1 Pagina di login 13](#_Toc1556288089)

[3.3.2 Pagina di registrazione 14](#_Toc1713742523)

[3.3.3 Pannello di controllo dell’admin 14](#_Toc2040347007)

[3.3.4 Struttura del sito 15](#_Toc2073586967)

[3.3.5 Pagina delle informazioni del parcheggio 16](#_Toc1603310273)

[3.3.6 Pagina di profilo dell’utente 16](#_Toc1853374797)

[3.3.7 Pagina per la modifica del proprio profilo 17](#_Toc1503355980)

[3.4 Design procedurale 18](#_Toc921625136)

[4 Implementazione 19](#_Toc1439577940)

[1.1 Controllers 19](#_Toc904965481)

[1.1.1 Home 19](#_Toc2001867883)

[1.1.1.1 Index() 19](#_Toc300268046)

[1.1.2 Login 19](#_Toc58049716)

[1.1.2.1 logout() 19](#_Toc1447184273)

[1.1.2.2 login() 19](#_Toc593136516)

[1.1.3 Register 20](#_Toc763087192)

[1.1.3.1 register() 20](#_Toc229685807)

[1.1.3.2 verify() 20](#_Toc1849067802)

[1.1.4 Offer 20](#_Toc1234651622)

[1.1.4.1 offri() 20](#_Toc792498461)

[1.1.4.2 index() 20](#_Toc1774997139)

[1.1.5 Research 21](#_Toc1287192151)

[1.1.5.1 index() 21](#_Toc172615821)

[1.1.6 Reserve 21](#_Toc828304718)

[1.1.6.1 index() 21](#_Toc1392574764)

[1.1.6.2 prenota() 21](#_Toc152523529)

[1.1.7 Validator 22](#_Toc540247821)

[1.1.7.1 validateDate(element) 22](#_Toc1698988214)

[1.1.7.2 validateCharAndSpace(element) 22](#_Toc1473918945)

[1.1.7.3 validateCap(element) 22](#_Toc548314860)

[1.1.7.4 validateCarPlate(element) 22](#_Toc104942670)

[1.1.7.5 validateVia(element) 22](#_Toc882723387)

[1.1.7.6 testInput(element) 22](#_Toc114573735)

[1.1.7.7 validatePhoneNumber(element) 23](#_Toc2145289678)

[1.2 Models 23](#_Toc808826706)

[1.2.1 LoginModel 23](#_Toc1717884008)

[1.2.1.1 log(email, password) 23](#_Toc1851180827)

[1.2.2 RegisterModel 24](#_Toc164699038)

[1.2.2.1 register() 24](#_Toc492025496)

[1.2.2.2 verify(mail, nome, cognome) 25](#_Toc1143275120)

[1.2.3 OfferModel 25](#_Toc1069664520)

[1.2.3.1 addOfferta() 26](#_Toc346409731)

[1.2.4 ResearchModel 27](#_Toc1443543166)

[1.2.4.1 getParcheggiDisponibili() 27](#_Toc1127714236)

[1.2.4.2 filterAll(startDate, endDate, disp) 27](#_Toc1793594004)

[1.2.4.3 filterByDisp() 28](#_Toc2036679683)

[1.2.4.4 formatCarAndDate() 29](#_Toc1890801429)

[1.2.5 ReserveModel 29](#_Toc2023279811)

[1.2.5.1 updateParcheggio() 29](#_Toc1738263837)

[1.2.5.2 getParcheggioInfo(id\_parcheggio) 30](#_Toc977969403)

[1.2.5.3 prenota() 30](#_Toc668294624)

[1.2.6 32](#_Toc1365777328)

[5 Test 32](#_Toc117677906)

[5.1 Protocollo di test 32](#_Toc840910445)

[5.2 Risultati test 32](#_Toc46598399)

[5.3 Mancanze/limitazioni conosciute 32](#_Toc1510252670)

[6 Consuntivo 32](#_Toc993433975)

[7 Conclusioni 33](#_Toc586846220)

[7.1 Sviluppi futuri 33](#_Toc1061757236)

[7.2 Considerazioni personali 33](#_Toc319869272)

[8 Bibliografia 33](#_Toc1135161080)

[8.1 Bibliografia per articoli di riviste: 33](#_Toc1166699907)

[8.2 Bibliografia per libri 33](#_Toc1202592659)

[8.3 Sitografia 33](#_Toc1249734815)

[9 Allegati 33](#_Toc1164505937)

# 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

E’ una breve e accurata rappresentazione dei contenuti di un documento, senza notazioni critiche o valutazioni. Lo scopo di un abstract efficace dovrebbe essere quello di far conoscere all’utente il contenuto di base di un documento e metterlo nella condizione di decidere se risponde ai suoi interessi e se è opportuno il ricorso al documento originale.

Può contenere alcuni o tutti gli elementi seguenti:

* **Background/Situazione iniziale**
* **Descrizione del problema e motivazione**: Che problema ho cercato di risolvere? Questa sezione dovrebbe includere l'importanza del vostro lavoro, la difficoltà dell'area e l'effetto che potrebbe avere se portato a termine con successo.
* **Approccio/Metodi**: Come ho ottenuto dei progressi? Come ho risolto il problema (tecniche…)? Quale è stata l’entità del mio lavoro? Che fattori importanti controllo, ignoro o misuro?
* **Risultati**: Quale è la risposta? Quali sono i risultati? Quanto è più veloce, più sicuro, più economico o in qualche altro aspetto migliore di altri prodotti/soluzioni?

Esempio di abstract:

*As the size and complexity of today’s most modern computer chips increase, new techniques must be developed to effectively design and create Very Large Scale Integration chips quickly. For this project, a new type of hardware compiler is created. This hardware compiler will read a C++ program, and physically design a suitable microprocessor intended for running that specific program. With this new and powerful compiler, it is possible to design anything from a small adder, to a microprocessor with millions of transistors. Designing new computer chips, such as the Pentium 4, can require dozens of engineers and months of time. With the help of this compiler, a single person could design such a large-scale microprocessor in just weeks.*

## 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

I casi d’uso rappresentano l’interazione tra i vari attori e le funzionalità del prodotto.

## Pianificazione

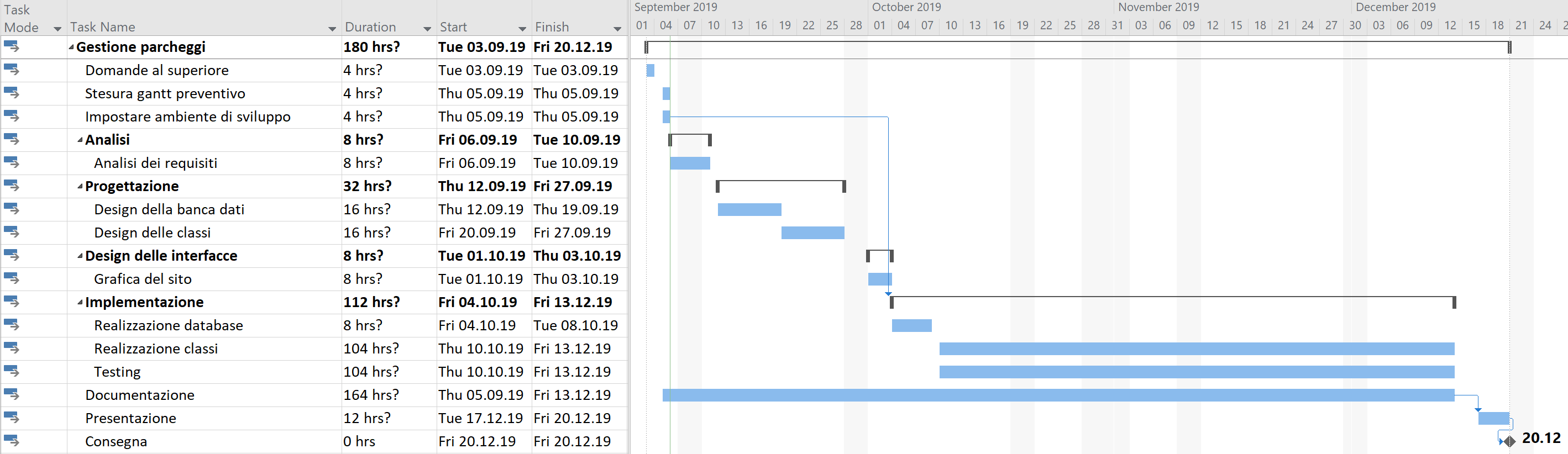


Figura 1 Gantt preventivo

## 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

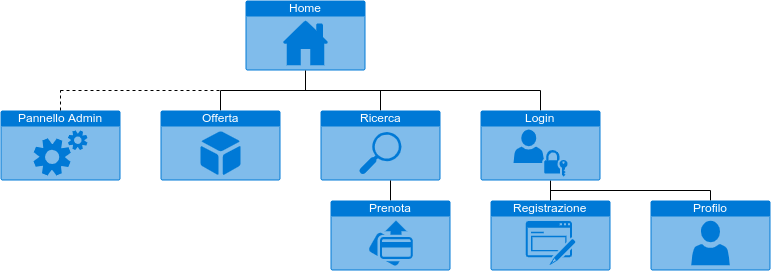
Computer Paolo Weishaupt:

* 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 basato su Debian 9 Stretch

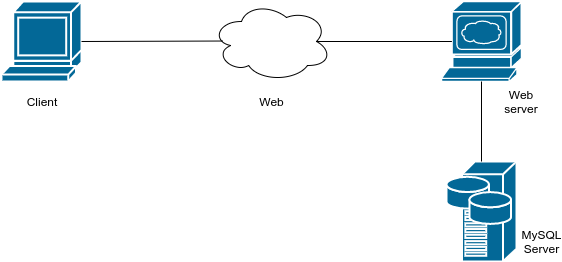
# Progettazione

## Design dell’architettura del sistema

### Sitemap



### Schema di rete

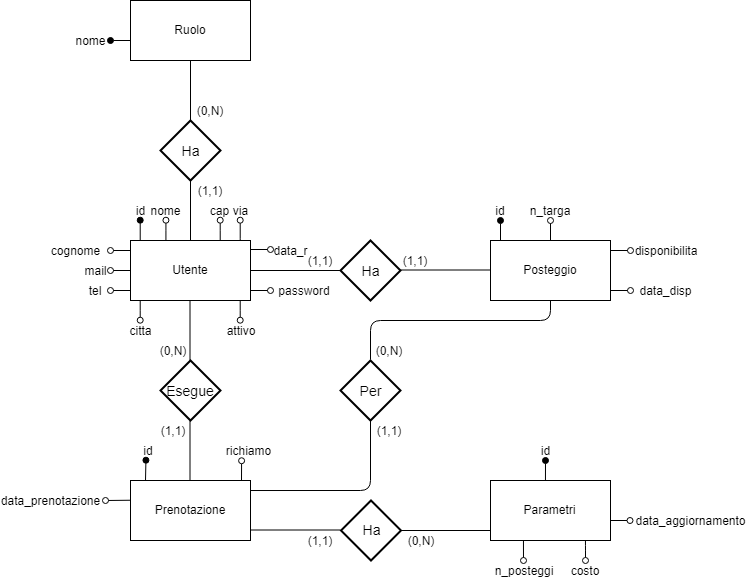


Descrive:

* La struttura del programma/sistema lo schema di rete...
* Gli oggetti/moduli/componenti che lo compongono.
* I flussi di informazione in ingresso ed in uscita e le relative elaborazioni. Può utilizzare *diagrammi di flusso dei dati* (DFD).
* Eventuale sitemap

## 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.

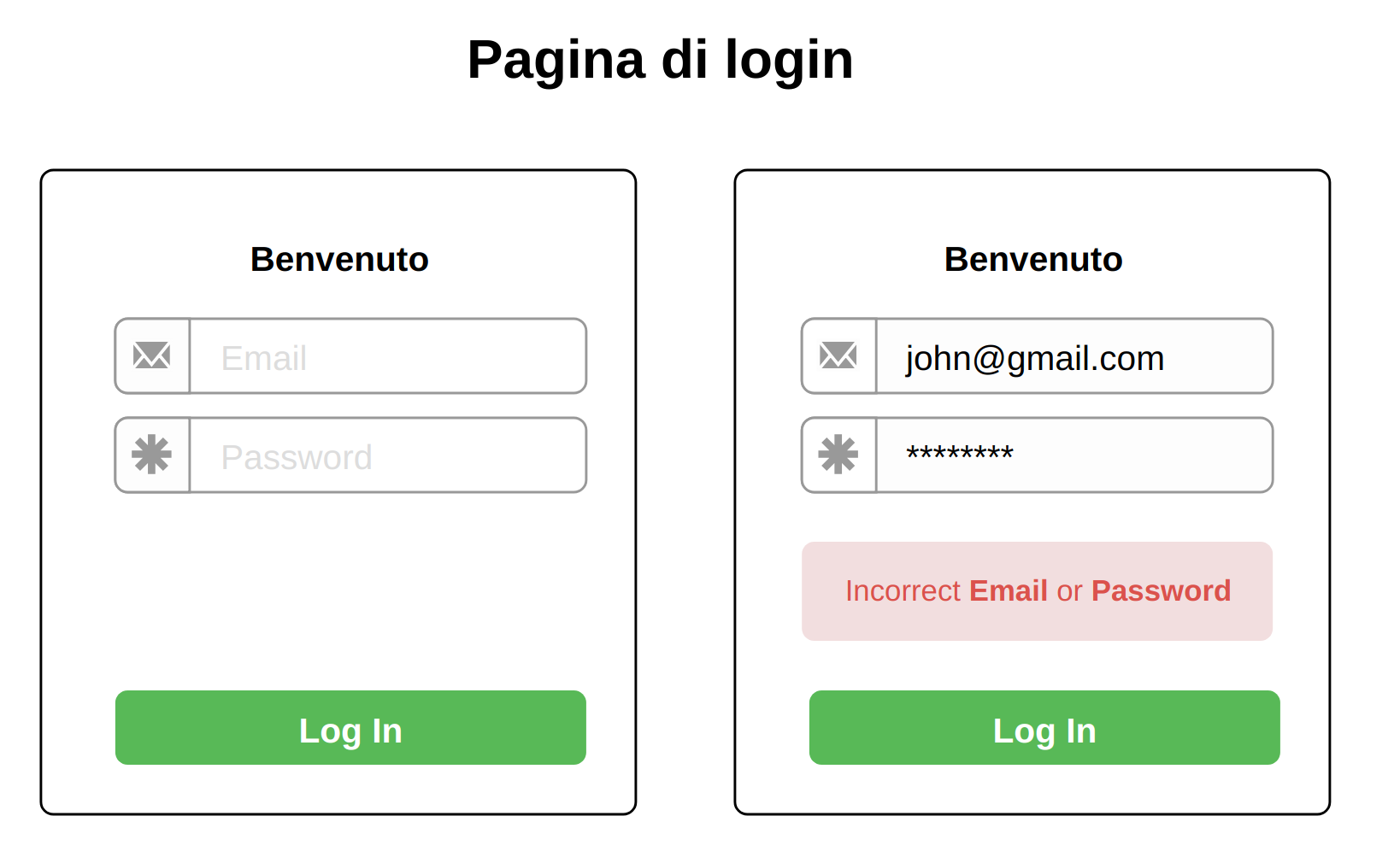
L’entità UTENTE rappresenta gli utenti del db e contiene le loro informazioni. L’attributo attivo serve per la verifica dell’e-mail perché finché non viene verificata il suo valore sarà su false e l’account sarà disabilitato. L’attributo potrà inoltre essere settato a false se l’utente dovesse avere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

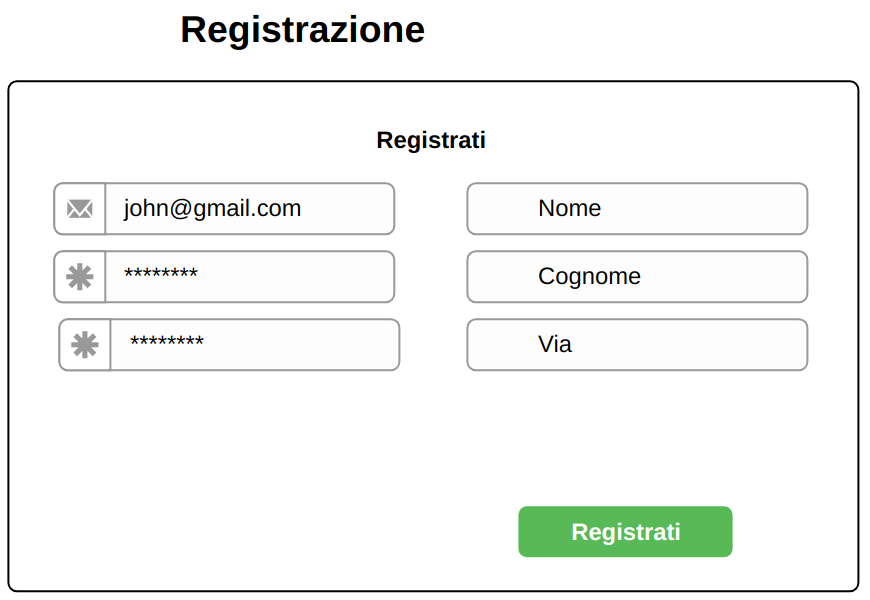
L’entità PRENOTAZIONE serve per gestire le prenotazioni degli utenti di un parcheggio. L’attributo richiamo ha di default il valore false. Se la fattura riguardante quella prenotazione non dovesse essere saldata verrà settato a true e al prossimo richiamo l’utente verrà disabilitato.

## Design delle interfacce

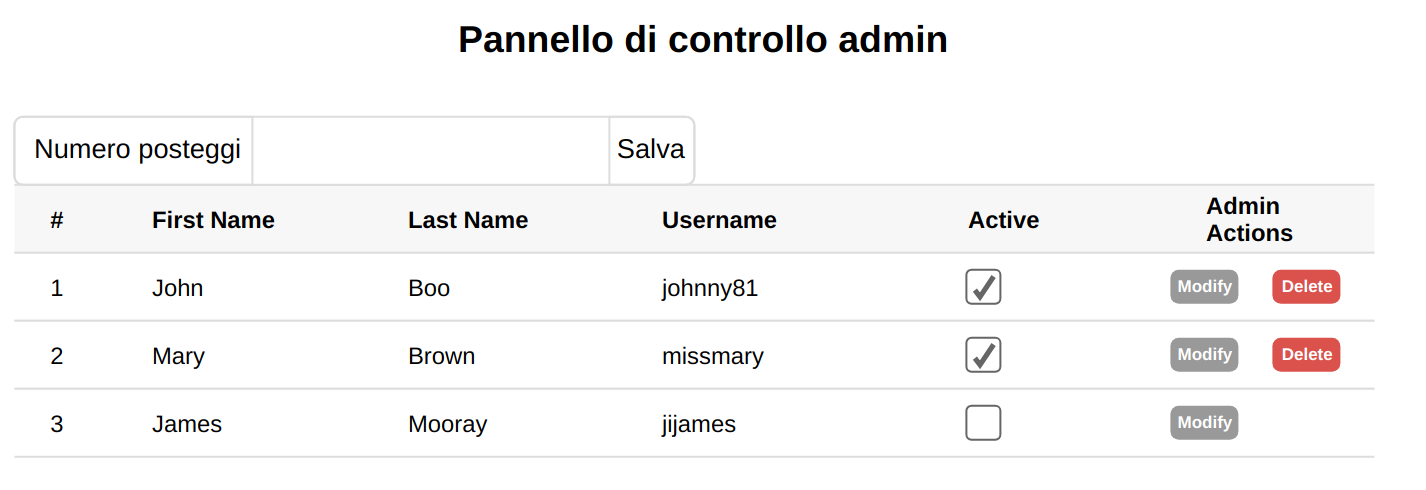
### Pagina di login



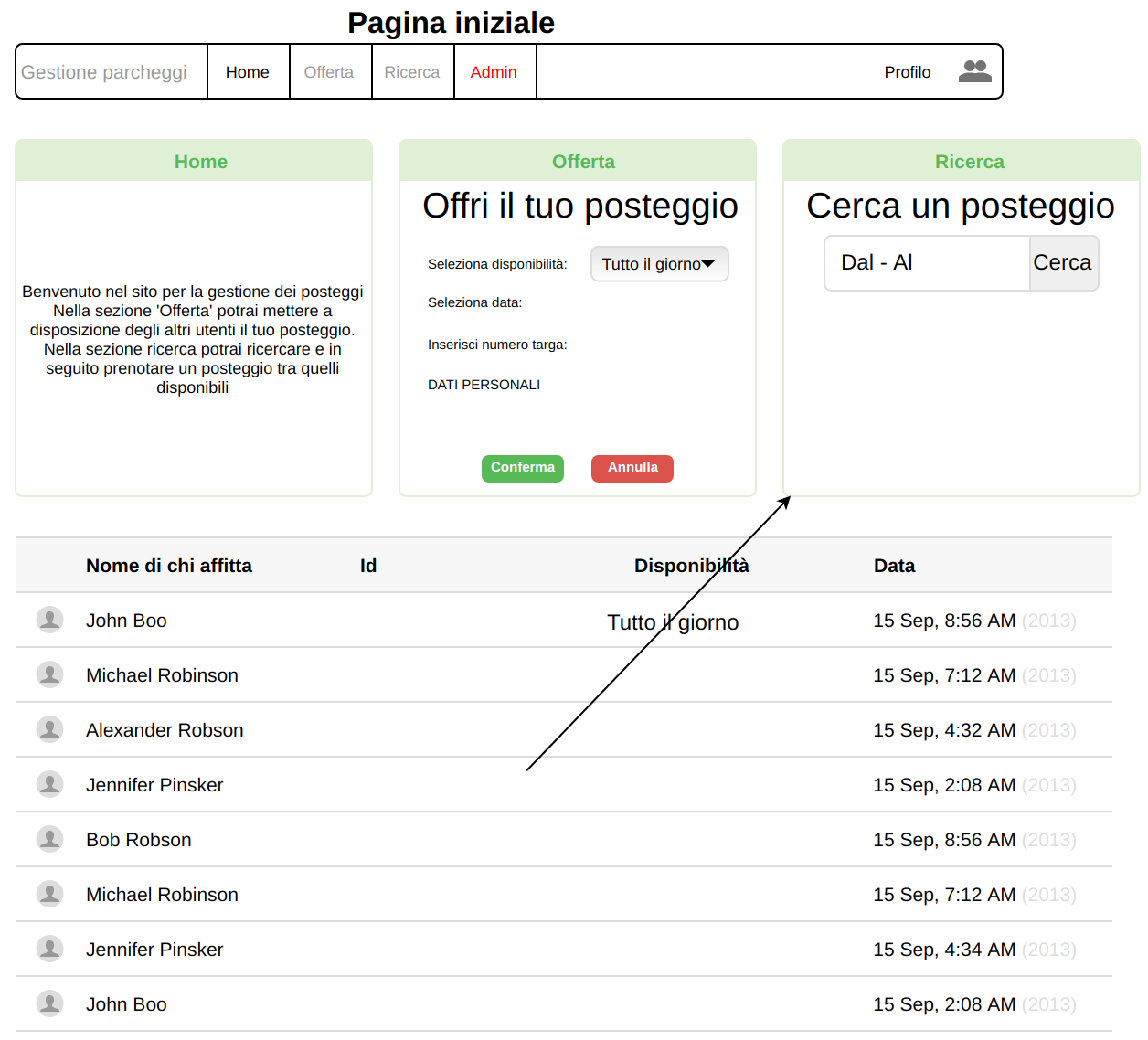
### Pagina di registrazione



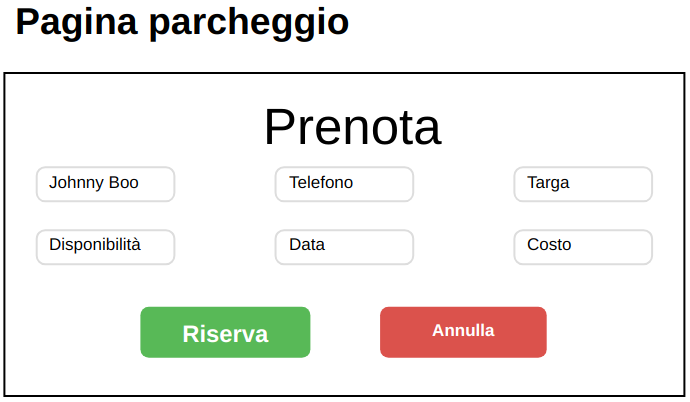
### Pannello di controllo dell’admin



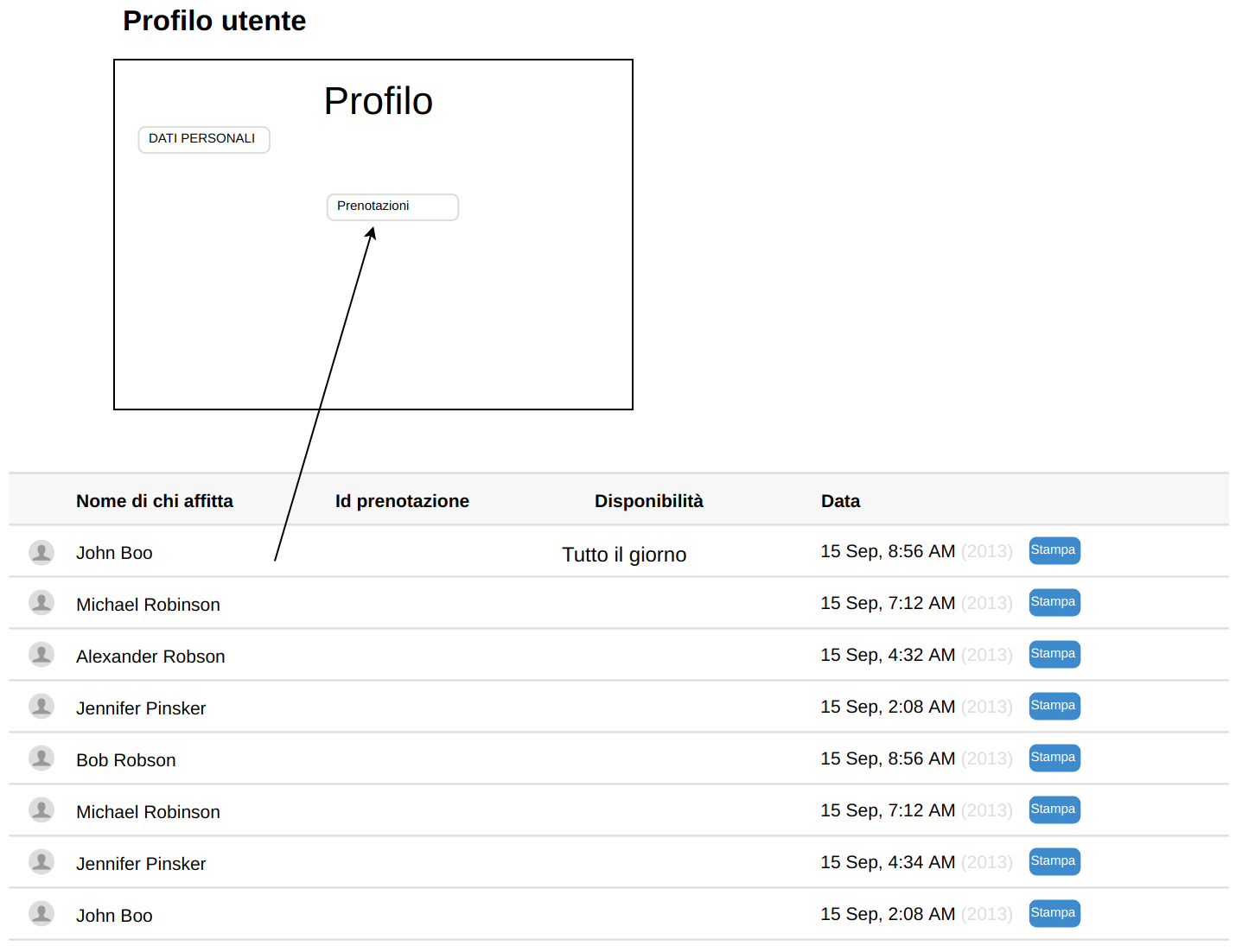
### Struttura del sito



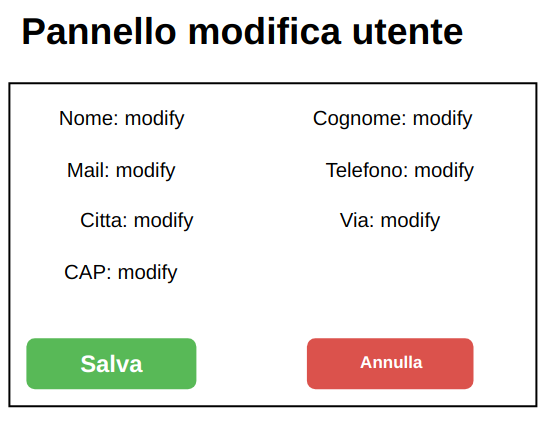
### Pagina delle informazioni del parcheggio



### Pagina di profilo dell’utente



### Pagina per la modifica del proprio profilo



## Design procedurale

Descrive i concetti dettagliati dell’architettura/sviluppo utilizzando ad esempio:

* Diagrammi di flusso e Nassi.
* Tabelle.
* Classi e metodi.
* Tabelle di routing
* Diritti di accesso a condivisioni …

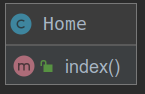
Questi documenti permetteranno di rappresentare i dettagli procedurali per la realizzazione del prodotto.

# Implementazione

## Controllers

### Home

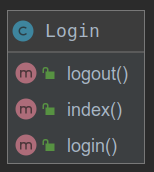
Il controller Home serve per la gestione di tutte le richieste relative alla homepage del sito.



#### Index()

Tramite tramite questo verrà caricata la View della homepage.

### Login



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

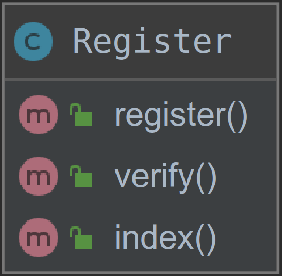
#### logout()

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.

#### 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.

### Register



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

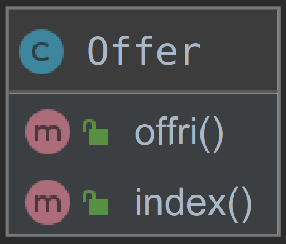
#### register()

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

#### verify()

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

### Offer



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

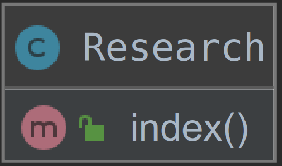
#### offri()

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

#### 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.

### Research

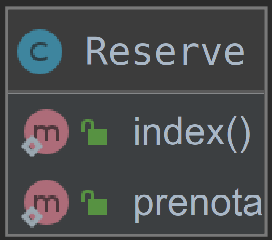


Il controller Research si occupa della gestione della pagina di ricerca dei parcheggi disponibili.

#### 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.

### Reserve



Il controller Reserve si occupa della gestione delle prenotazioni.

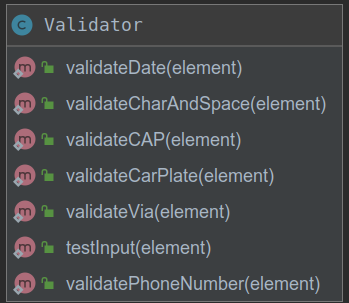
#### index()

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.

#### prenota()

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

### Validator



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

#### 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.

#### 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.

#### 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.

#### 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.

#### 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.

#### 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.

#### 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

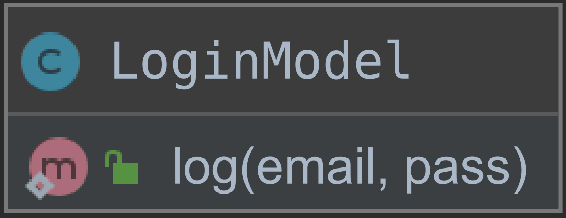
+41 7X XXX XX XX

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

Come parametro riceve il numero di telefono da controllare.

## Models

### LoginModel



Questa classe gestisce il login degli utenti.

#### 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***); |

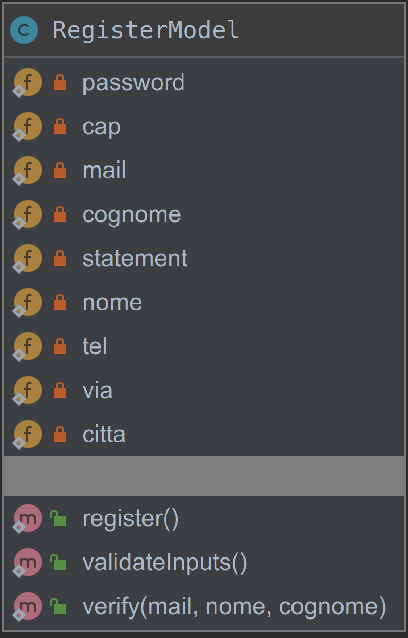
Preparo la query da eseguire inserendo i parametri. La query seleziona l’utente con la mail e la password immessi al momento del login. Con il metodo bindParam() assegno le variabili ai parametri nella query, la eseguo e inserisco il risultato in un array associativo.

|  |
| --- |
| **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.

### RegisterModel



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

#### 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.

#### 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***); |

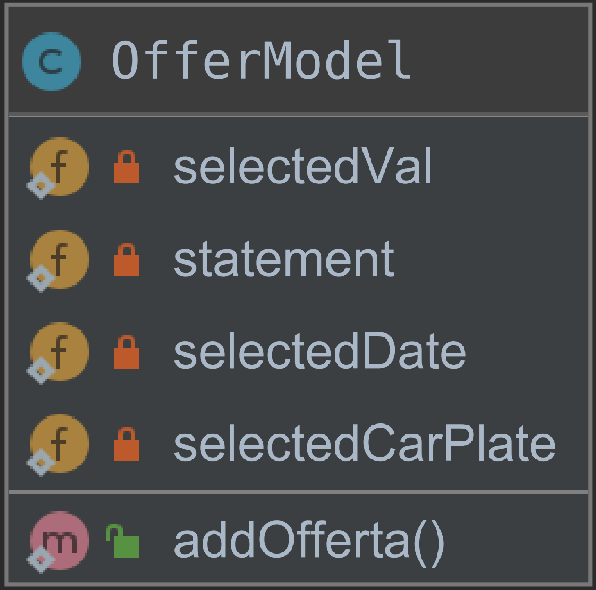
Eseguo una query richiedendo la mail dell’utente dove corrisponde a quella dell’utente da verificare.

|  |
| --- |
| **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.

### OfferModel



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

#### 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 cliccatoo il pulsante offri prendo il valore degli input e li valido tramite il metodi 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.

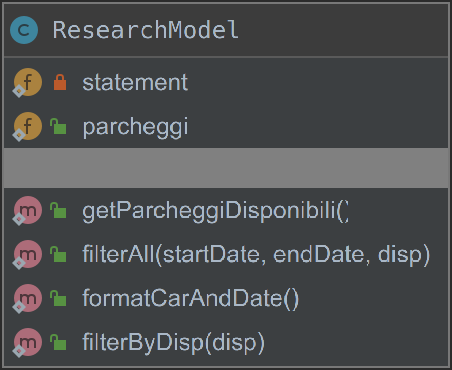
|  |
| --- |
| **else** {  **if**(*Users*::*hasParcheggio*() && $\_SESSION[**'active'**] == **true**)  { |

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.

### ResearchModel



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

#### 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*(); } |

#### 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.

#### 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.

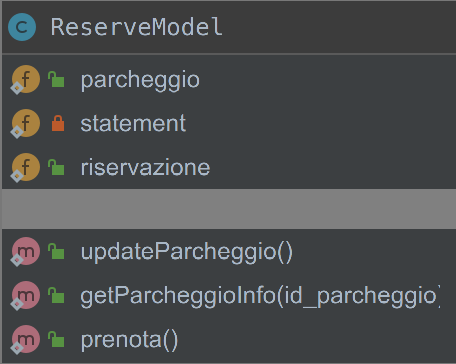
#### formatCarAndDate()

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.

### ReserveModel



Questa classe gestisce la riservazione dei parcheggi.

#### 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 asettandoli a null. Facendo così il posteggio non risulterà più disponibille.

#### 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.

#### 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.

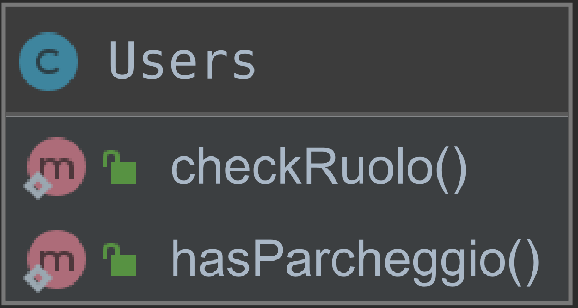
|  |
| --- |
| *MailModel*::*reservationMail*($\_SESSION[**'mail'**], $\_SESSION[**'nome'**], $\_SESSION[**'cognome'**], **self**::*$parcheggio*, **self**::*$riservazione*);  **self**::*updateParcheggio*(); **unset**($\_SESSION[**'id\_posteggio\_prenotato'**]); ViewLoader::*load*(**'home/index'**, **array**(**'prenotazioneOK'**=>**"Prenotazione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

### Users



Questa classe si occupa della gestione degli utenti.

#### 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.

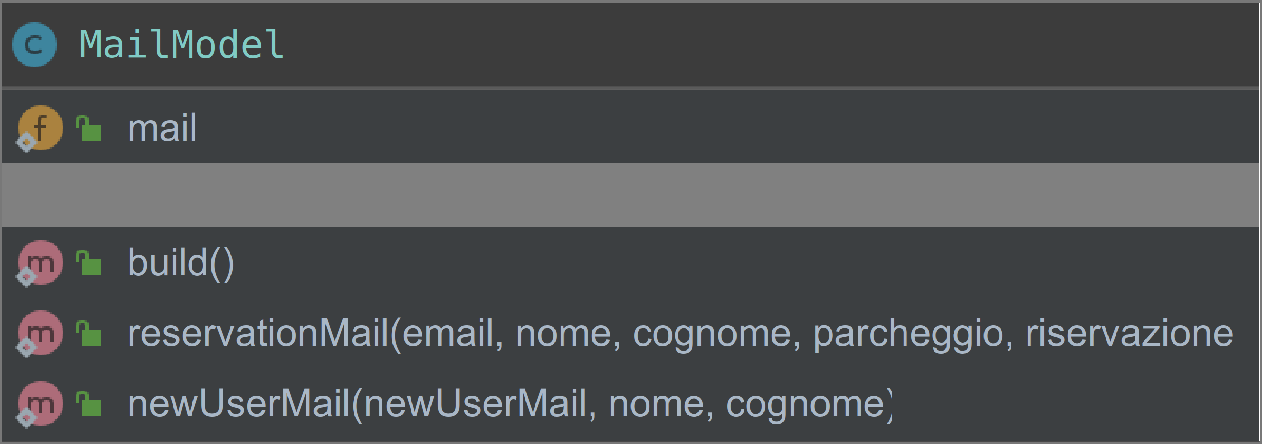
#### hasParcheggio()

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.

### MailModel



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

#### 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'**); } |

#### 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.

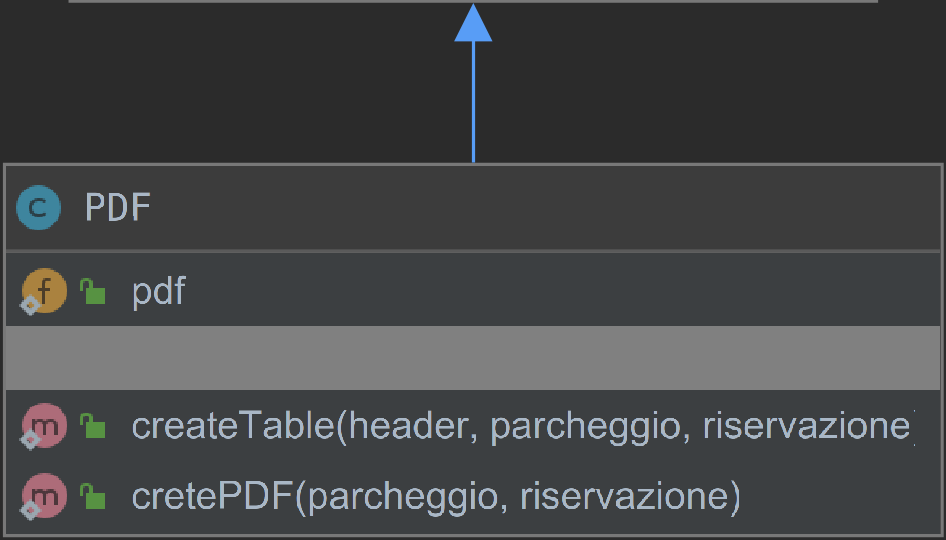
#### newUserMail(newUserMail, nome, cognome)

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.

### PDF



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

# Test

## Protocollo di test

Definire in modo accurato tutti i test che devono essere realizzati per garantire l’adempimento delle richieste formulate nei requisiti. I test fungono da garanzia di qualità del prodotto. Ogni test deve essere ripetibile alle stesse condizioni.

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-001  REQ-012 | **Nome:** | Import a card with KIC, KID and KIK keys, but not shown with the GUI |
| **Descrizione:** | Import a card with KIC, KID and KIK keys with no obfuscation, but not shown with the GUI | | |
| **Prerequisiti:** | Store on local PC: Profile\_1.2.001.xml (appendix n\_n) and Cards\_1.2.001.txt (appendix n\_n).  PIN (OTA\_VIEW\_PIN\_PUK\_KEY) and ADM (OTA\_VIEW\_ADM\_KEY) user right not set. | | |
| **Procedura:** | 1. Go to “Cards manager” menu,  in main page click “Import Profiles” link, Select the “1.2.001.xml” file, Import the Profile 2. Go to “Cards manager” menu,  in main page click “Import Cards” link, Select the “1.2.001.txt” file, Delete the cards,  Select the “1.2.001.txt” file, Import the cards 3. Research the “41795924770” Card, Click the imsi card link Check the card details 4. Execute the SQL: SELECT imsi, dir, keyset, cntr, rawtohex(kickey), rawtohex(kidkey), rawtohex(kikkey), rawtohex(chv), rawtohex(dap)FROM otacardkey a where imsi='340041795924770' ORDER BY keyset; | | |
| **Risultati attesi:** | Keys visible in the DB (OtaCardKey) but not visible in the GUI (Card details) | | |

## Risultati test

Tabella riassuntiva in cui si inseriscono i test riusciti e non del prodotto finale. Se un test non riesce e viene corretto l’errore, questo dovrà risultare nel documento finale come riuscito (la procedura della correzione apparirà nel diario), altrimenti dovrà essere descritto l’errore con eventuali ipotesi di correzione.

## Mancanze/limitazioni conosciute

Descrizione con motivazione di eventuali elementi mancanti o non completamente implementati, al di fuori dei test case. Non devono essere riportati gli errori e i problemi riscontrati e poi risolti durante il progetto.

# Consuntivo

Consuntivo del tempo di lavoro effettivo e considerazioni riguardo le differenze rispetto alla pianificazione (cap 1.7) (ad esempio Gannt consuntivo).

# Conclusioni

Quali sono le implicazioni della mia soluzione? Che impatto avrà? Cambierà il mondo? È un successo importante? È solo un’aggiunta marginale o è semplicemente servita per scoprire che questo percorso è stato una perdita di tempo? I risultati ottenuti sono generali, facilmente generalizzabili o sono specifici di un caso particolare? ecc

## Sviluppi futuri

Migliorie o estensioni che possono essere sviluppate sul prodotto.

## Considerazioni personali

Cosa ho imparato in questo progetto? ecc

# Bibliografia

## Bibliografia per articoli di riviste:

1. Cognome e nome (o iniziali) dell’autore o degli autori, o nome dell’organizzazione,
2. Titolo dell’articolo (tra virgolette),
3. Titolo della rivista (in italico),
4. Anno e numero
5. Pagina iniziale dell’articolo,

## Bibliografia per libri

1. Cognome e nome (o iniziali) dell’autore o degli autori, o nome dell’organizzazione,
2. Titolo del libro (in italico),
3. ev. Numero di edizione,
4. Nome dell’editore,
5. Anno di pubblicazione,
6. ISBN.

## Sitografia

1. URL del sito (se troppo lungo solo dominio, evt completo nel diario),
2. Eventuale titolo della pagina (in italico),
3. Data di consultazione (GG-MM-AAAA).

**Esempio:**

* http://standards.ieee.org/guides/style/section7.html, *IEEE Standards Style Manual*, 07-06-2008.

# Allegati

Elenco degli allegati, esempio:

* Diari di lavoro
* Codici sorgente/documentazione macchine virtuali
* Istruzioni di installazione del prodotto (con credenziali di accesso) e/o di eventuali prodotti terzi
* Documentazione di prodotti di terzi
* Eventuali guide utente / Manuali di utilizzo
* Mandato e/o Qdc
* Prodotto
* …