



**UNIVERSITÀ  
DEGLI STUDI  
DI BERGAMO**

## **Documentazione Progetto**

### **Informatica III - Progettazione e Algoritmi**

Prof.ssa Patrizia Scandurra

Benedeta Lenuzza 1068745

Gloria Pasinetti 1066654

Andrea Lenzi 1066922

Laurea Magistrale in Ingegneria Informatica

Febbraio 2023



# **1 Iterazione 0**

## **1.1 Introduzione**

Lo scopo di questo progetto è quello di creare un sistema distribuito per la gestione dei parcheggi coperti nella città di Bergamo.

L'applicativo è sviluppato per far fronte alle esigenze di un attore:

- L'utente che cerca un parcheggio disponibile.

Ogni parcheggio ha a disposizione un numero limitato di parcheggi che vanno costantemente contati, per cui è in grado di ospitare un numero massimo di utenti contemporaneamente. Per questo motivo, per migliorare l'efficienza sia in termini di costo che di tempo è stato sviluppato un apposito algoritmo grazie al quale si riesce ad abbinare l'utente al parcheggio più adatto alle sue esigenze. Il focus del progetto è quindi quello di gestire la prenotazione di posti auto online effettuata da parte dei clienti ottimizzando le risorse necessarie per effettuare la prenotazione.

## **1.2 Requisiti Funzionali**

Il sistema consente all'utente che cerca parcheggio di sovrintendere agli aspetti principali della gestione dei posti liberi. In particolare, inserendo il nome utente e la password per la login l'utente sarà in grado di:

- Accedere alla propria area personale, visualizzando i suoi dati di accesso.
- Visualizzare i parcheggi disponibili e, attraverso un algoritmo, verificare quali tra questi si adatta al meglio alle proprie esigenze (vicino, posti liberi, posizione, recensione, orario...).
- Visualizzare le recensioni dei singoli parcheggi disponibili all'interno della applicazione.
- Creare recensioni per un determinato parcheggio.
- Possibilità di prenotare il posto auto, specificando orario e quantità di

veicoli.

### 1.3 Analisi dei casi d'uso

Al fine di procedere ad uno sviluppo efficiente, è stato deciso di dividere le specifiche funzionali in tre code di priorità: alta, media, bassa. Nella coda ad alta priorità si trovano i casi d'uso fondamentali al corretto funzionamento dell'applicazione, nella coda a media priorità sono inseriti i casi d'uso riguardanti le funzionalità aggiuntive e nella coda a bassa priorità ci sono le funzionalità non strettamente necessarie.

<b>CODICE</b>	<b>DESCRIZIONE</b>
UC1	Prenotazione parcheggi e visualizzazione dei singoli parcheggi disponibili.

Tabella 1: Casi d'uso ad alta priorità

<b>CODICE</b>	<b>DESCRIZIONE</b>
UC2	Registrazione Utente
UC3	Login Utente
UC4	Logout Utente
UC5	Visualizzazione recensioni
UC6	Creazione di recensioni

Tabella 2: Casi d'uso a media priorità

<b>CODICE</b>	<b>DESCRIZIONE</b>
UC7	Modifica e visualizzazione dati utente

Tabella 3: Casi d'uso a bassa priorità



proprietario. Inoltre, è possibile accedere al sistema tramite una pagina Web dotata di chiare e semplici interfacce.

### **Portabilità**

Il sistema sarà sviluppato come una Web App il che renderà più semplice la portabilità su un'App Android.

## **1.6 Pattern di progettazione**

Il design pattern scelto per lo sviluppo del progetto è il Client-Server dove:

- Esiste almeno un componente con il ruolo di server: i server non conoscono né il numero né l'identità dei client
- Esiste almeno un componente con il ruolo di client: i client conoscono le identità dei server

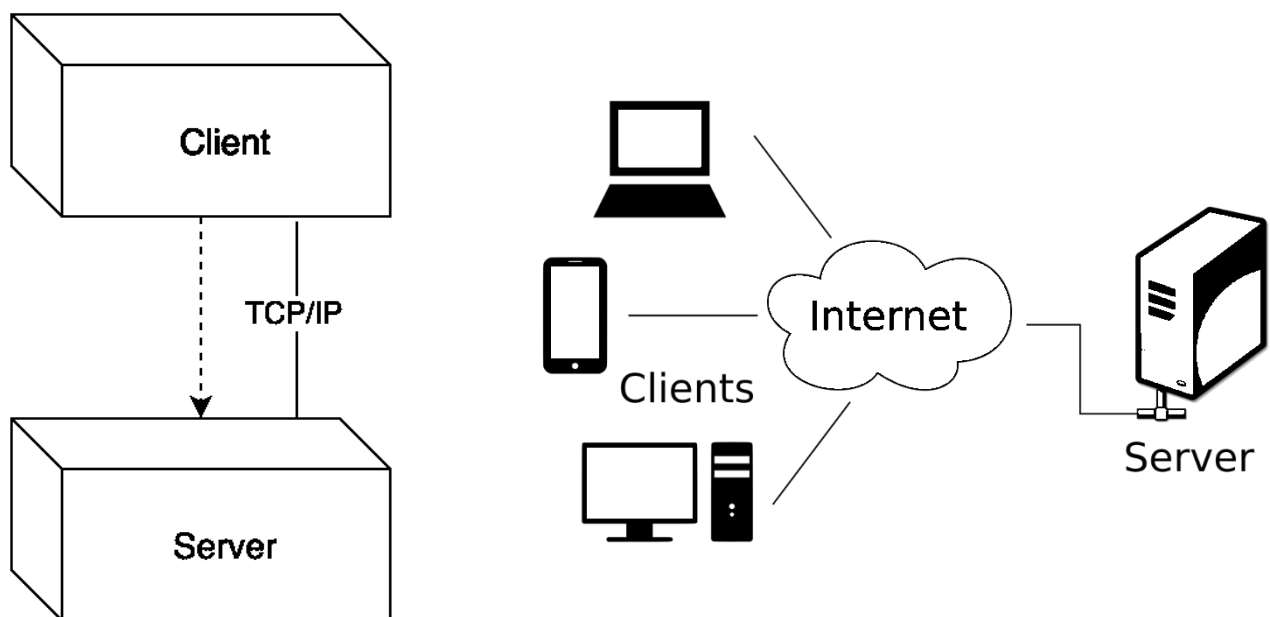


Figura 2: pattern di progettazione

## 1.7 Topologia del sistema

La topologia del sistema, mostrata in Figura 3, evidenzia come il progetto sia stato sviluppato utilizzando un'architettura Three-Tier. L'applicazione è stata dunque suddivisa in tre livelli architetturali, ognuno specializzato in un aspetto del sistema. Questi sono:

- Data, dedicato alla gestione dei dati persistenti;
- Application, che racchiude la logica funzionale;
- Presentation, ossia l'interfaccia utente.

Nello specifico il modulo Data è costituito da un Database gestito con MySQL, all'interno del modulo Application si trova il Web Server usato per gestire la logica funzionale dell'applicazione sviluppata in Java attraverso il framework Spring, e infine, tramite le API esposte dal Server, il modulo Presentation recupera i dati e li rappresenta all'utente tramite una WebApp realizzata in JavaScript, HTML/CSS o un app Android.

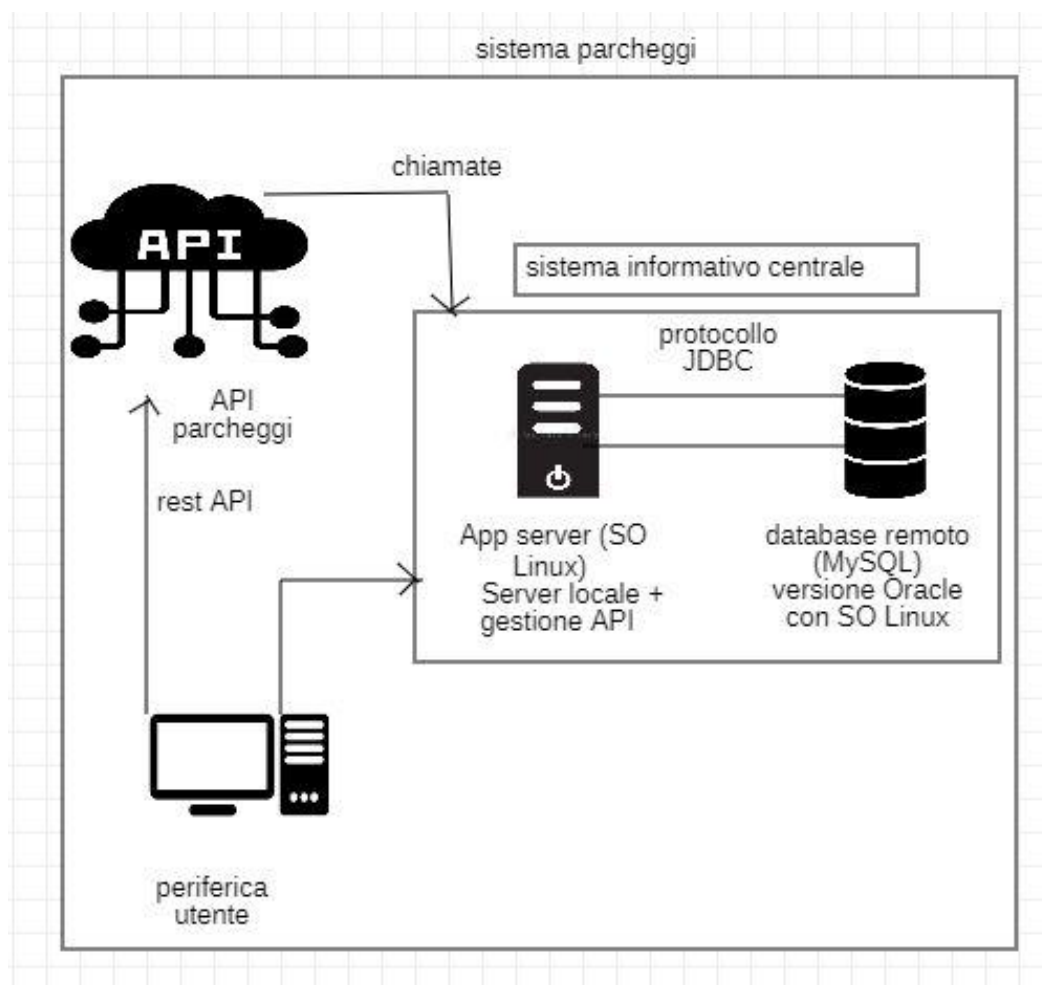


Figura 3: topologia del sistema



## 2 Iterazione 1

Nella prima iterazione si è deciso di implementare i primi casi d'uso ad alta priorità riportati nella tabella 1 di pagina 7:

UC1: Prenotazione parcheggi e visualizzazione dei singoli parcheggi disponibili.

1. Visualizzazione dei parcheggi disponibili
2. Prenotazione di un parcheggio

Di seguito viene riportata una descrizione testuale per ogni caso d'uso implementato in questa iterazione.

### 2.1 UC1.1: Visualizzazione dei parcheggi disponibili

*Breve descrizione:* All'utente che vuole prenotare un parcheggio verrà messa a disposizione un form che dovrà compilare con varie informazioni. In base alle informazioni inserite, tramite un algoritmo verrà calcolato il parcheggio più vicino a lui che rispetta le caratteristiche richieste. Successivamente, il cliente avrà la possibilità di scegliere la propria opzione definitiva prenotando uno o più posti macchina nello specifico parcheggio specificando: Orario, Numero di veicoli, durata della sosta.

1. Visualizzazione dell'elenco dei parcheggi
2. Calcolo della migliore opzione disponibile
3. Visualizzazione dei parcheggi (eventualmente con foto)

*Attori coinvolti:* Sistema, Utente.

*Procedimento:* il Sistema mostra la pagina home dedicata alla visualizzazione di tutti i parcheggi, L'utente inserisce il parametro da prioritizzare nella ricerca, il sistema aggiorna la pagina mostrando le migliori opzioni calcolate tramite un algoritmo di ricerca applicato su tutti i parcheggi precedentemente mostrati, l'utente visualizza il parcheggio a lui più adatto.

### **2.1.1 UC1.1: Visualizzazione dell'elenco dei parcheggi**

*Breve descrizione:* Nella schermata verrà visualizzata una lista dei parcheggi totali registrati nel sistema.

*Attori coinvolti:* Utente, Sistema.

### **2.1.2 UC1.2: Calcolo della migliore opzione disponibile**

*Breve descrizione.* Utente decide il parametro che deve avere più priorità nella ricerca del parcheggio e il sistema risponde con una lista che tiene conto di questo.

*Attori coinvolti:* Utente, Sistema.

*Procedimento:*

1. Il sistema mostra la pagina home della mappa
2. L'utente clicca sulla finestra dei parametri e decide quello che si adatta meglio alle sue necessita
3. Il sistema risponde con una lista che rappresenti al meglio la ricerca dell'utente.

### **2.1.3 UC1.3: Visualizzazione dei parcheggi**

*Breve descrizione:* L'utente dopo aver fatto la ricerca può visualizzare parcheggio per parcheggio al fine di trovare quello che si adatta meglio alle proprie esigenze.

*Attori coinvolti:* Utente, Sistema.

*Procedimento:*

1. Il sistema mostra le varie opzioni ottimizzate
2. L' Utente sceglie la sua ipotetica scelta definitiva.
3. Il sistema mostra la descrizione del singolo parcheggio.
4. L'utente può decidere se procedere alla prenotazione o cambiare parcheggio scelto.

## **2.2 UC2: Prenotazione di un parcheggio**

*Breve descrizione:* Una volta scelto definitivamente il parcheggio, l'utente può procedere alla sua prenotazione specificando al sistema: il numero di macchine coinvolte, l'orario di arrivo, e il tempo complessivo di sosta del singolo.:

- UC2.1: Effettua Prenotazione
- UC2.2: Conferma prenotazione

*Attori coinvolti:* Utente, Sistema.

*Procedimento:* Il sistema dopo aver mostrato all'utente il singolo parcheggio che si addice meglio alle sue esigenze, mostra sulla schermata un bottone dove l'utente può prenotare il posto auto specificando i parametri descritti precedentemente. Il sistema confermerà la prenotazione e aggiornerà i posti disponibili nel parcheggio.

### **2.2.1 UC2.1: Effettua Prenotazione**

*Breve descrizione:* L'utente decide il parcheggio e lo prenota.

*Attori coinvolti:* Utente, Sistema.

*Procedimento:*

1. L'utente dopo aver visualizzato un parcheggio clicca su prenota
2. Il sistema mostra a schermo la relativa pagina (*prenota.php*) in cui è presente un apposito form.
3. L'utente compila il form e clicca sul bottone *Prenota*.
4. Il sistema aggiorna la pagina mostrando a schermo la prenotazione.

### **2.2.2 UC2.2: Conferma prenotazione**

*Breve descrizione:* Una volta compilato il form il sistema prenota e aggiorna i posti relativi al parcheggio scelto.

*Attori coinvolti:* Utente, Sistema.

*Procedimento:*

1. Il sistema mostra la pagina di prenotazione all'utente
2. L'utente prenota il posto.
3. Il sistema mostra a schermo la relativa pagina (*conferma.php*) dove mostra la avvenuta prenotazione e aggiorna i posti disponibili nel parcheggio scelto dall'utente.

## 2.3 UML Component Diagram

Analizzando i casi d'uso implementati in questa iterazione è stato possibile progettare il Component Diagram mostrato in Figura 4. Il grafico evidenzia le interazioni tra i componenti dal punto di vista delle funzionalità

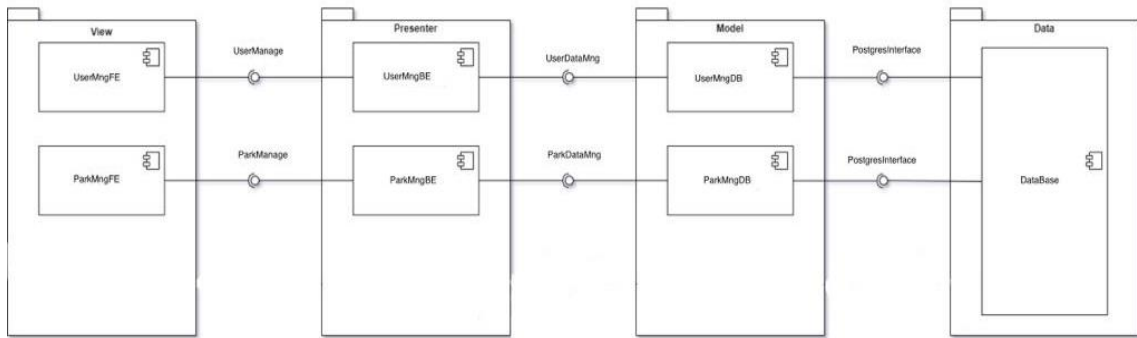


Figura 4: Component Diagram delle parti implementate

## 2.4 UML Class Diagram per i tipi di dato

Il diagramma in Figura 5 mostra i tipi di dato necessari allo sviluppo dell'applicazione.

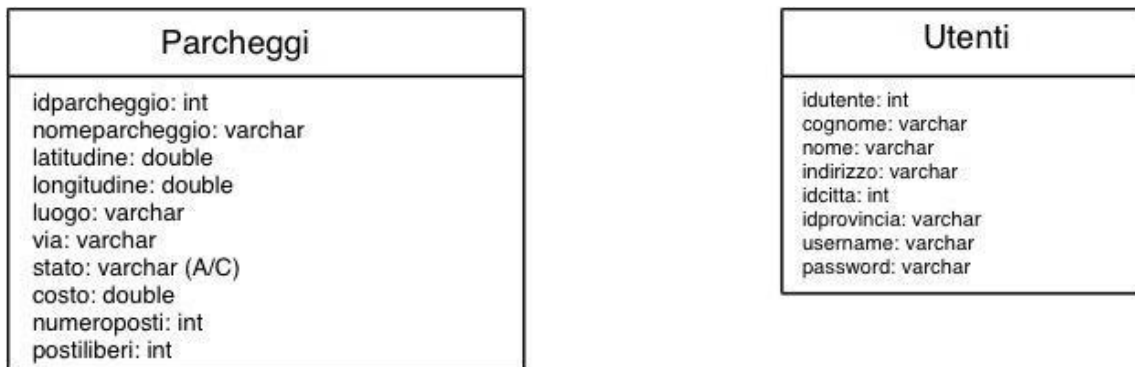


Figura 5: Class Diagram per i tipi di dato

## 3 Iterazione Finale

Nell'ultima iterazione si è deciso di implementare i test statici e dinamici per valutare se il codice funzionasse nella maniera corretta. Si è inoltre deciso di ampliare il class diagram per renderlo più completo e coerente con l'implementazione del codice

### 3.1 Testing

#### 3.1 Analisi Dinamica

Per effettuare l'analisi dinamica dell'applicazione è stato utilizzato PHPUnit: un framework di unit testing per il linguaggio di programmazione PHP che ha permesso di verificare la corretta esecuzione dei casi di test previsti.

```
<?php
include("connessione.php");
session_start();
$idutente=$_SESSION["idutente"];

$cognome=$_POST["cognome"];
$nome=$_POST["nome"];
$indirizzo=$_POST["indirizzo"];
$provincia=$_POST["cmbProvincia"];
$comune=$_POST["cmbCitta"];
mysqli_set_charset($conn,"utf8");
$sql="UPDATE utenti SET cognome='".$cognome."', nome='".$nome."', indirizzo='".$indirizzo."',
if (mysqli_query($conn,$sql)) {
    header("location: mioprofilo_PAC.php?modifica=1");
    $mod = true;
}
else {
    $mod_n = true;
}
mysqli_close($conn);

class test{
public function test_modifica_corretto(){
assertSame(true,$mod);
}
}
```

Figura 6: Codice PHP Test sulla pagina salvamodificheprofilo.php

Figura 7: Risultato del test effettuato sulla pagina salvamodificheprofilo.php

**Gloria Pasinetti**

Cognome:

Nome:

Indirizzo:

Provincia:  ▼

Comune:  ▼

**SALVA MODIFICHE**

Modifica avvenuta correttamente!

In Figura 7 si mostra che il test fatto è andato a buon fine (MODIFICA  
AVVENUTA CORRETTAMENTE!)



```

        $cognome = $riga["cognome"];
        session_start();
        $ins = true;

        $_SESSION['idutente'] = $idutente;
        $_SESSION['username'] = $username;
        $_SESSION['nome'] = $nome;
        $_SESSION['cognome'] = $cognome;

        header("location: index.php");
    }
    else{
        header("location: registrazione.php?e=0");
    }

    else {echo ("LE PASSWORD NON CORRISPONDONO!");}

mysqli_close($conn);
}

$utente = new inserimentoUtente;
$utente -> nuovoUtente();

class test_ins{
    public function test(){
        assertSame(true,$ins);
    }
}

```

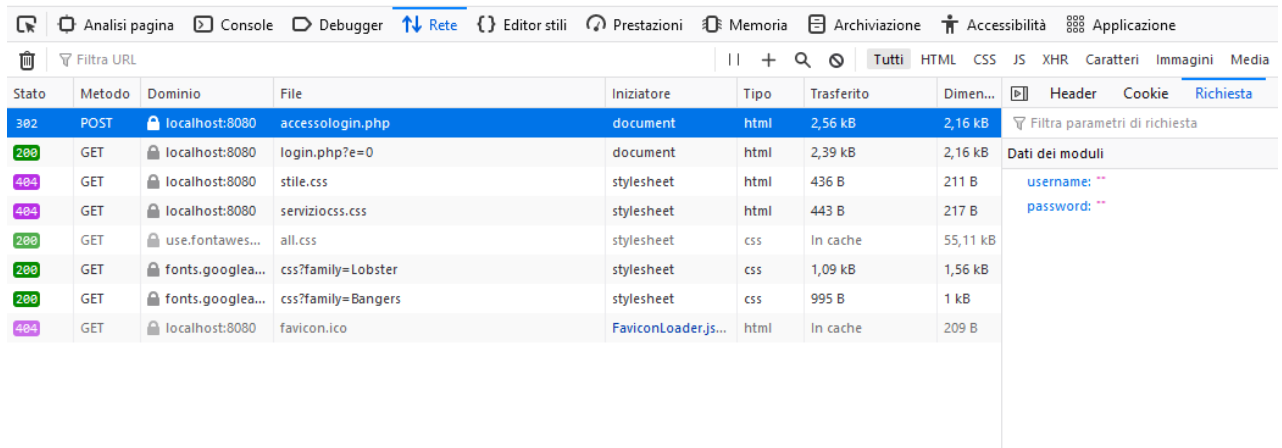
Figura 8: Codice PHP Test sulla pagina inserimentp.php

### 3.2 Test API tramite Browser

La verifica del buon funzionamento delle API REST esposte dai vari controller è stata effettuata tramite la funzionalità di analisi della pagina offerta dal Browser. In particolare, è stato testato il funzionamento di una chiamata GET (prenotazione parcheggio) e di due chiamate POST (usata per il login e per trovare la posizione)

### 3.2.1 Test chiamata POST Login

La chiamata POST mostrata in Figura 10 consente alla WebApp di salvare nel database le informazioni di una singola iscrizione passando i valori inseriti dall'utente nei relativi campi. In questo caso abbiamo deciso di utilizzare una chiamata POST perché dovevamo salvare una password e quindi non volevamo che venisse visualizzata nell'URL quando i parametri venivano passati durante la chiamata.



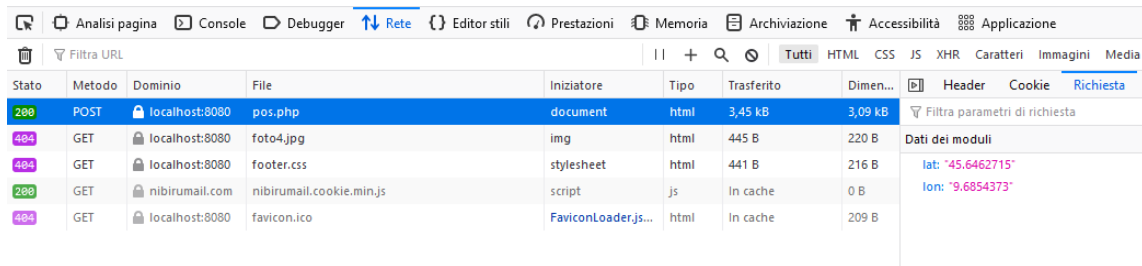
The screenshot shows the Chrome DevTools Network tab with the 'Rete' (Network) panel selected. The first request is a POST to 'accesslogin.php' with a status of 302. Subsequent requests for 'login.php?e=0', 'stile.css', 'serviziocss.css', 'all.css', 'css?family=Lobster', 'css?family=Bangers', and 'favicon.ico' are shown with their respective status codes (200 or 404) and details.

Stato	Metodo	Dominio	File	Iniziatore	Tipo	Trasferito	Dimen...	Header	Cookie	Richiesta
302	POST	localhost:8080	accesslogin.php	document	html	2,56 kB	2,16 kB	Filtra parametri di richiesta		
200	GET	localhost:8080	login.php?e=0	document	html	2,39 kB	2,16 kB	Dati dei moduli		
404	GET	localhost:8080	stile.css	stylesheet	html	436 B	211 B	username: ""		
404	GET	localhost:8080	serviziocss.css	stylesheet	html	443 B	217 B	password: ""		
200	GET	use.fontawes...	all.css	stylesheet	css	In cache	55,11 kB			
200	GET	fonts.googlea...	css?family=Lobster	stylesheet	css	1,09 kB	1,56 kB			
200	GET	fonts.googlea...	css?family=Bangers	stylesheet	css	995 B	1 kB			
404	GET	localhost:8080	favicon.ico	FaviconLoader.js...	html	In cache	209 B			

Figura 10: Chiamata POST login

### 3.2.2 Test chiamata POST Posizione

La chiamata POST mostrata in Figura 11 consente alla WebApp di salvare nel database le informazioni di una singola iscrizione passando i valori inseriti dall'utente nei relativi campi. In questo caso abbiamo deciso di utilizzare una chiamata POST perché dovevamo salvare una longitudine e una latitudine per salvare la posizione.



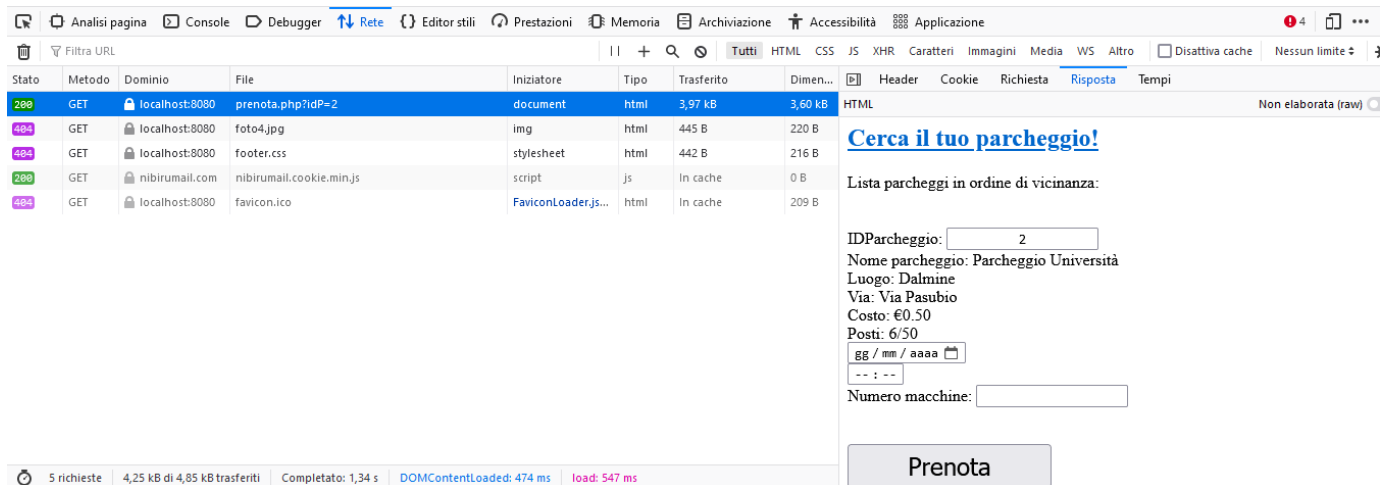
Stato	Metodo	Dominio	File	Iniziatore	Tipo	Trasferito	Dimen...	Header	Cookie	Richiesta
200	POST	localhost:8080	pos.php	document	html	3,45 kB	3,09 kB			
404	GET	localhost:8080	foto4.jpg	img	html	445 B	220 B			
404	GET	localhost:8080	footer.css	stylesheet	html	441 B	216 B			
200	GET	nibirumail.com	nibirumail.cookie.min.js	script	js	In cache	0 B			
404	GET	localhost:8080	favicon.ico	FaviconLoader.js...	html	In cache	209 B			

Dati dei moduli  
lat: "45.6462715"  
lon: "9.6854373"

Figura 11: Chiamata POST posizione

### 3.2.3 Test chiamata GET Prenotazione parcheggio

La chiamata GET mostrata in Figura 12 consente alla WebApp di recuperare dal database le informazioni del parcheggio che si vuole prenotare.



Stato	Metodo	Dominio	File	Iniziatore	Tipo	Trasferito	Dimen...	Header	Cookie	Richiesta	Risposta	Tempi
200	GET	localhost:8080	prenota.php?idP=2	document	html	3,97 kB	3,60 kB					
404	GET	localhost:8080	foto4.jpg	img	html	445 B	220 B					
404	GET	localhost:8080	footer.css	stylesheet	html	442 B	216 B					
200	GET	nibirumail.com	nibirumail.cookie.min.js	script	js	In cache	0 B					
404	GET	localhost:8080	favicon.ico	FaviconLoader.js...	html	In cache	209 B					

Cerca il tuo parcheggio!

Lista parcheggi in ordine di vicinanza:

IDParcheggio:

Nome parcheggio: Parcheggio Università

Luogo: Dalmine

Via: Via Pasubio

Costo: €0.50

Posti: 6/50

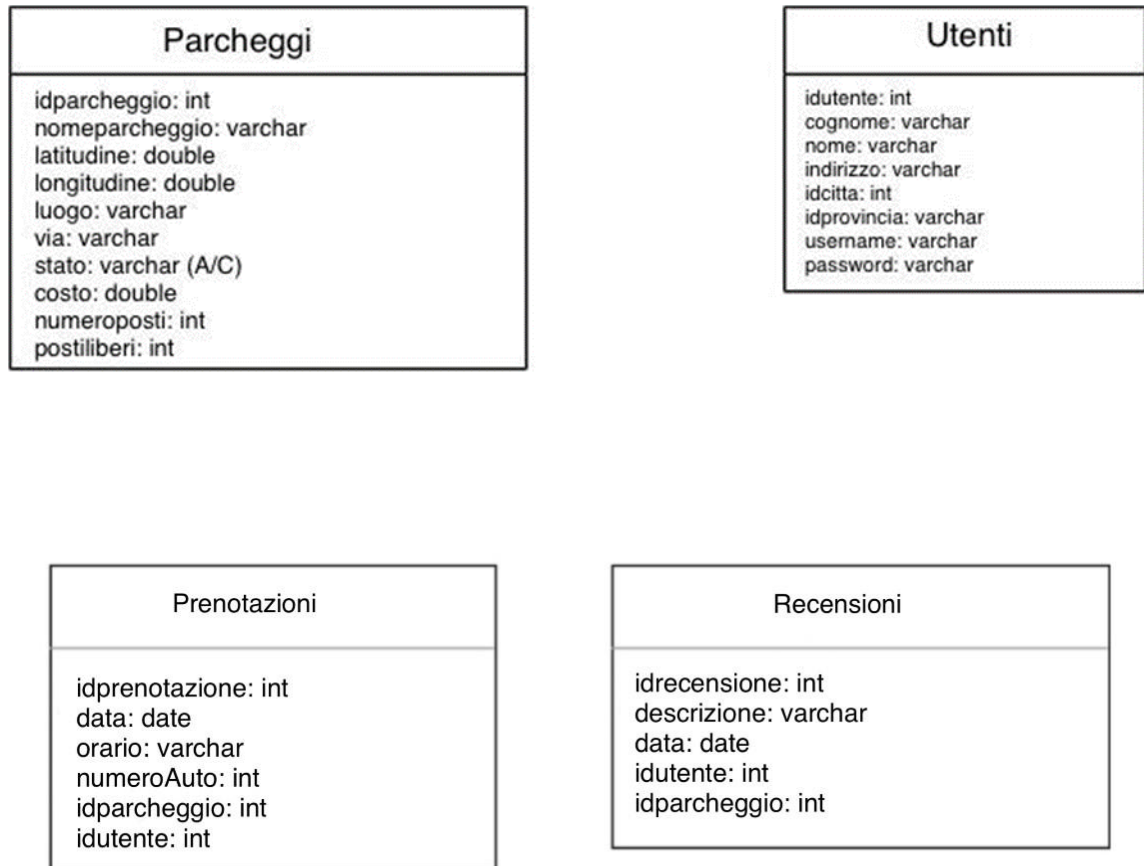
Numero macchine:

Prenota

5 richieste | 4,25 kB di 4,85 kB trasferiti | Completato: 1,34 s | DOMContentLoaded: 474 ms | load: 547 ms

Figura 12: Chiamata GET visualizza parcheggio selezionato

### 3.3 UML Class Diagram per i tipi di dato



Il diagramma in Figura 13 mostra i tipi di dato necessari allo sviluppo dell'applicazione.

### 3.4 Algoritmo

L'algoritmo utilizzato nel progetto serve per trovare il parcheggio aperto e coperto più vicino alla posizione dell'utente.

CODICE JAVASCRIPT PER TROVARE LA POSIZIONE DELL'UTENTE

```
var x = document.getElementById("test");  
  
function trovaCoordinate() {  
if (navigator.geolocation) {  
navigator.geolocation.getCurrentPosition(showPosition);  
} else {  
x.innerHTML = "Geolocalizzazione non supportata dal browser.";  
}  
}  
  
function showPosition(position) {  
var lat;  
var lon;  
var distanzaLat;  
var distzanLon;  
lat = position.coords.latitude;  
lon = position.coords.longitude;  
document.getElementById("lat").innerHTML = lat ;  
document.getElementById("lon").innerHTML = lon ;  
}  
}
```

## CODICE PHP PER TROVARE PARCHEGGIO APERTO PIU' VICINO ALLA POSIZIONE DELL'UTENTE

```
<?php
include("connessione.php");
$lat= $_POST["lat"];
$lon = $_POST["lon"];
//echo $lat . "<br>";
//echo $lon;

$sql= "SELECT idparcheggio, nomeparcheggio from parcheggio where stato= 'A' order by
(latitudine-$lat), (longitudine-$lon) ";
$ris=mysqli_query($conn,$sql);
while($riga= mysqli_fetch_array($ris)){
    echo "<div style='border:3px solid #760c0c; padding: 20px;'>";
    $idparcheggio = $riga["idparcheggio"];

    echo "<a href='prenota.php?idP=$idparcheggio'> - " . $riga["nomeparcheggio"] . "<br>";
    //echo $riga["longi"] . "<br>";
    echo "</div>";
}

?>
```