



UNIVERSITÀ DEGLI STUDI DI PERUGIA

RELAZIONE PROGETTO

Sistemi aperti e distribuiti

Creazione di un servizio online di prenotazione delle camere di un albergo

A cura di Daniel Lestini e Joshua Jm Longetti

Obbiettivi:

L'obiettivo è realizzare un web service che permetta agli utenti di effettuare la prenotazione delle camere di un albergo, dando la possibilità di scegliere le date in cui prenotare, e quella di scegliere tra le tipologie diverse di camere: Normal, Comfort e Superior. Sarà inoltre possibile annullare le prenotazioni già effettuate.

Inoltre sarà possibile, per gli utenti registrati come admin, creare nuove camere, prenotare e annullare le prenotazioni, ed anche aggiornare i dati delle camere già presenti o addirittura eliminarle.

Beneficiari: Utenti generici che vogliano prenotare una stanza per fare le cose zozze.

Erogatori: Il gestore del sito nonché padrone dell'hotel nonché io nonché il peggio informatico.

Tecnologia utilizzata:

Il progetto è stato realizzato utilizzando diversi programmi: Nginx, AngularJS, Mysql e Php .

1-Nginx

Utilizzato per la gestione del server. Esso fornisce rapidamente i contenuti statici con un utilizzo efficiente delle risorse di sistema. Grazie ad esso è possibile distribuire contenuti dinamici HTTP su una rete che utilizza i gestori FastCGI per gli script, e può servire come bilanciatore di carico.

Nginx utilizza un approccio asincrono basato su eventi nella gestione delle richieste in modo da ottenere prestazioni più prevedibili sotto stress, in contrasto con il modello del server HTTP Apache che usa un approccio orientato ai thread o ai processi nella gestione delle richieste.

2-AngularJS

Utilizzato per la parte front-end del progetto, è un framework di javascript. Ha l'obiettivo di semplificare lo sviluppo e il test di questa tipologia di applicazioni fornendo un framework lato client con architettura MVC (Model View Controller) e Model-view-viewmodel (MVVM) insieme a componenti comunemente usate nelle applicazioni RIA.

Il framework AngularJS lavora leggendo prima la pagina HTML, che ha incapsulati degli attributi personalizzati addizionali (esempio: *ng-controller*). Angular interpreta questi attributi come delle direttive (comandi) per legare le parti di ingresso e uscita della pagina al modello che è rappresentato da variabili standard JavaScript. Il valore di queste variabili può essere impostato manualmente nel codice o recuperato da risorse JSON statiche o dinamiche.

3-MySQL

È un Relational database management system (RDBMS) composto da un client a riga di comando e un server.

Con esso gestiamo il database dove sono contenute le camere con tutti i loro dati.

4-Php

Utilizzato per la parte back-end del progetto, PHP (acronimo ricorsivo di "PHP: Hypertext Preprocessor", preprocessore di ipertesti; originariamente acronimo di "Personal Home Page") è un linguaggio di scripting interpretato, concepito per la programmazione di pagine web dinamiche. È principalmente utilizzato per sviluppare applicazioni web lato server, ma può essere usato anche per scrivere script a riga di comando o applicazioni stand-alone con interfaccia grafica.

Struttura e funzionamento del progetto:

Il progetto è strutturato principalmente in due sotto progetti:

- Parte front-end che usa Angular
- Parte back-end in Php nativo, senza l'uso di framework.

Il database è gestito con MySQL ed il server è gestito con Nginx.

Il front-end funziona così:

- Ogni "botone" è una chiamata in GET/POST, fatta tramite il service (typescript del componente). È possibile vedere la chiamata tramite la console; Si va su Rete-Header (per vedere come viene mandato il dato), Preview/Response (come torna indietro il risultato).

Una volta fatta la chiamata al server (Nginx), viene chiamato questo Endpoint back-end.

Endpoint back-end che funziona così:

- Chiama l'oggetto e la sua relativa funzione, e ritorna la risposta alla chiamata GET/POST di cui sopra

Codice Database.php:

```
<?php

class Database{
    // db credentials
    private $host = "localhost";
    private $dbName = "albergo";
    private $userName = "root";
    private $psw = "Test1234";
    public $conn;

    // get the database connection
    public function getConnection(){

        $this->conn = null;
        try{
            $this->conn = new PDO("mysql:host=" . $this->host . ";dbname=" . $this->dbName, $this->userName, $this->psw);
            $this->conn->exec("set names utf8");
        }catch(PDOException $exception){
            echo "Connection error: " . $exception->getMessage();
        }
        return $this->conn;
    }
}

?>
```

```
jolong@jolong-Ubruttu: ~
File Modifica Visualizza Cerca Terminale Aiuto
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> describe users
-> ;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| id    | int(11) | NO | PRI | NULL | auto_increment |
| username | varchar(255) | NO | | NULL |
| email | varchar(255) | NO | | NULL |
| password | varchar(255) | NO | | NULL |
| adminUser | tinyint(1) | YES | | NULL |
| token | varchar(255) | YES | | NULL |
+-----+-----+-----+-----+-----+-----+
6 rows in set (0.00 sec)

mysql> describe rooms
-> ;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| id    | int(11) | NO | PRI | NULL | auto_increment |
| name  | varchar(255) | YES | | NULL |
| price | int(11) | YES | | NULL |
| persons | int(11) | YES | | NULL |
| type  | varchar(16) | YES | | NULL |
| occupied_from | date | YES | | NULL |
| occupied_to | date | YES | | NULL |
+-----+-----+-----+-----+-----+-----+
7 rows in set (0.00 sec)

mysql> mysql> clean
-> ;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax
to use near 'clean' at line 1
mysql> 
```

Ci sono due oggetti principali, room e users:

Room.php:

<?php

```
class Room{
// database connection and table name
private $conn;
private $table_name = "rooms";
// object properties
public $id;
public $name;
public $price;
public $persons;
public $type;
public $occupied_from;
public $occupied_to;
```

```

// constructor with $db as database connection
public function __construct($db){
    $this->conn = $db;
}

// read rooms by type
public function readRooms($roomType) {
    $query = "SELECT
p.id, p.name, p.price, p.persons, p.occupied_from, p.occupied_to
FROM
    " . $this->table_name . " p
WHERE p.type = '" . strtoupper($roomType) . "'
ORDER BY
    p.price DESC";
    // prepare query statement
    $stmt = $this->conn->prepare($query);
    // execute query
    $stmt->execute();
    return $stmt;
}

// read all rooms
public function read(){
    // select all query
    $query = "SELECT
p.id, p.name, p.description, p.price, p.persons, p.occupation
FROM
    " . $this->table_name . " p
ORDER BY
    p.price DESC";

```

```

        // prepare query statement
$stmt = $this->conn->prepare($query);
        // execute query
$stmt->execute();
        return $stmt;
    }

    // create room
    function create(){

        // query to insert record
        $query = "INSERT INTO
        " . $this->table_name . "
        SET
        name=:name,
        price=:price,
        persons=:persons,
        type=:type"; //, occupation=:occupation";
        // prepare query
        $stmt = $this->conn->prepare($query);
        // sanitize
        $this->name=htmlspecialchars(strip_tags($this->name));
        $this->price=htmlspecialchars(strip_tags($this->price));
        $this->persons=htmlspecialchars(strip_tags($this->persons));
        $this->type=htmlspecialchars(strip_tags($this->type));
        // bind values
        $stmt->bindParam(":name", $this->name);
        $stmt->bindParam(":price", $this->price);
        $stmt->bindParam(":persons", $this->persons);
        $stmt->bindParam(":type", $this->type);
    }

```

```

        // execute query
        if($stmt->execute()){
            return true;
        }
        var_dump($stmt->errorInfo());
        return false;
    }

```

```

// used when filling up the update room form
function readOne(){
    // query to read single record
    $query = "SELECT
p.id, p.name, p.description, p.price, p.persons, p.occupation
FROM
" . $this->table_name . " p
WHERE
p.id = ?
LIMIT
0,1";

    // prepare query statement
    $stmt = $this->conn->prepare( $query );
    // bind id of room to be updated
    $stmt->bindParam(1, $this->id);
    // execute query
    $stmt->execute();
    // get retrieved row
    $row = $stmt->fetch(PDO::FETCH_ASSOC);
    // set values to object properties
    $this->name = $row['name'];
    $this->price = $row['price'];
    $this->description = $row['description'];

```

```
$this->persons = $row['persons'];  
$this->occupation = $row['occupation'];  
}
```

```
    // update the room  
    function update(){  
        // update query  
        $query = "UPDATE  
    " . $this->table_name . "  
        SET  
        name = :name,  
        price = :price,  
        persons = :persons,  
        type = :type  
        WHERE  
        id = :id";  
        // prepare query statement  
        $stmt = $this->conn->prepare($query);  
        // sanitize  
        $this->name=htmlspecialchars(strip_tags($this->name));  
        $this->price=htmlspecialchars(strip_tags($this->price));  
        $this->persons=htmlspecialchars(strip_tags($this->persons));  
        $this->type=htmlspecialchars(strip_tags($this->type));  
        $this->id=htmlspecialchars(strip_tags($this->id));  
        // bind new values  
        $stmt->bindParam(':name', $this->name);  
        $stmt->bindParam(':price', $this->price);  
        $stmt->bindParam(':persons', $this->persons);  
        $stmt->bindParam(':type', $this->type);  
        $stmt->bindParam(':id', $this->id);  
        // execute the query
```



```
if($stmt->execute()){  
    return true;  
}  
return false;  
}
```

```
// update the room
```

```
function prenote(){
```

```
    // update query
```

```
    $query = "UPDATE
```

```
    " . $this->table_name . "
```

```
    SET
```

```
    occupied_from = date(:occupied_from),
```

```
    occupied_to = date(:occupied_to)
```

```
    WHERE
```

```
    id = :id";
```

```
    // prepare query statement
```

```
    $stmt = $this->conn->prepare($query);
```

```
        // sanitize
```

```
        $this->occupied_from=htmlspecialchars(strip_tags($this->  
        >occupied_from));
```

```
$this->occupied_to=htmlspecialchars(strip_tags($this->occupied_to));
```

```
$this->id=htmlspecialchars(strip_tags($this->id));
```

```
    // bind new values
```

```
$stmt->bindParam(':occupied_from', $this->occupied_from);  
    $stmt->bindParam(':occupied_to', $this->occupied_to);  
    $stmt->bindParam(':id', $this->id);
```

```
    // execute the query  
    if($stmt->execute()){  
        return true;  
    }
```

```
    return false;  
}
```

```
    // update the room  
function undoPrenote(){
```

```
    // update query  
    $query = "UPDATE  
    " . $this->table_name . "  
        SET  
occupied_from = null,  
occupied_to = null  
        WHERE  
id = :id";
```

```
    // prepare query statement  
    $stmt = $this->conn->prepare($query);
```

```

        // sanitize
$this->id=htmlspecialchars(strip_tags($this->id));

        // bind new values
$stmt->bindParam(':id', $this->id);

        // execute the query
if($stmt->execute()){
    return true;
}

    return false;
}

        // delete the room
function delete(){
    // delete query
$query = "DELETE FROM " . $this->table_name . " WHERE id = :id";

        // prepare query
$stmt = $this->conn->prepare($query);
        // sanitize
$this->id=htmlspecialchars(strip_tags($this->id));

        // bind id of record to delete
$stmt->bindParam(':id', $this->id);
        // execute query

```

```

        if($stmt->execute()){
            return true;
        }
        return false;
    }

    // search rooms
    function search($keywords){
        // select all query
        $query = "SELECT
p.id, p.name, p.description, p.price, p.persons, p.occupation
        FROM
        " . $this->table_name . " p
        WHERE
        p.name LIKE ? OR p.description LIKE ?
        ORDER BY
        p.price DESC";
        // prepare query statement
        $stmt = $this->conn->prepare($query);
        // sanitize
        $keywords=htmlspecialchars(strip_tags($keywords));
        $keywords = "%{$keywords}%";
        // bind
        $stmt->bindParam(1, $keywords);
        $stmt->bindParam(2, $keywords);
        $stmt->bindParam(3, $keywords);
        // execute query
        $stmt->execute();
        return $stmt;
    }

```

```
}
```

Users.php:

```
<?php
```

```
class User{
// database connection and table name
    private $conn;
    private $table_name = "users";
    // object properties
    public $id;
    public $username;
    public $email;
    public $password;
    public $adminUser;
    public $token;
// constructor with $db as database connection
    public function __construct($db){
        $this->conn = $db;
    }

    // create user
function register($username, $email, $password, $adminUser){
    $this->token = bin2hex(openssl_random_pseudo_bytes(16));

    // query to insert record
    $query = "INSERT INTO
    " . $this->table_name . "
```

SET

```
username=:username, email=:email, password=:password,  
adminUser=:adminUser, token=:token";
```

```
// prepare query
```

```
$stmt = $this->conn->prepare($query);
```

```
// sanitize
```

```
$this->username=htmlspecialchars(strip_tags($this->username));
```

```
$this->email=htmlspecialchars(strip_tags($this->email));
```

```
$this->password=htmlspecialchars(strip_tags($this->password));
```

```
$this->adminUser=htmlspecialchars(strip_tags($this->adminUser));
```

```
// bind values
```

```
$stmt->bindParam(":username", $this->username);
```

```
$stmt->bindParam(":email", $this->email);
```

```
$stmt->bindParam(":password", $this->password);
```

```
$stmt->bindParam(":adminUser", $this->adminUser);
```

```
$stmt->bindParam(":token", $this->token);
```

```
// execute query
```

```
if($stmt->execute()){
```

```
    return true;
```

```
    } else {
```

```
        var_dump($stmt->errorInfo());
```

```
        return false;
```

```
    }
```

```
    }
```

```
// log the user
```

```
function login($username, $password){
```

```

        // query to read single record
        $query = "SELECT
            p.token, p.adminUser
            FROM
            " . $this->table_name . " p
            WHERE
            p.username = ? AND p.password = ?
            LIMIT
            1";

        // prepare query statement
        $stmt = $this->conn->prepare( $query );
        // search for username
        $stmt->bindParam(1, $username);
        $stmt->bindParam(2, $password);
        // execute query
        $stmt->execute();
        // get retrieved row
        $row = $stmt->fetch(PDO::FETCH_ASSOC);
        // set values to object properties
        $this->token = $row['token'];
        $this->adminUser = $row['adminUser'];

        return $row;

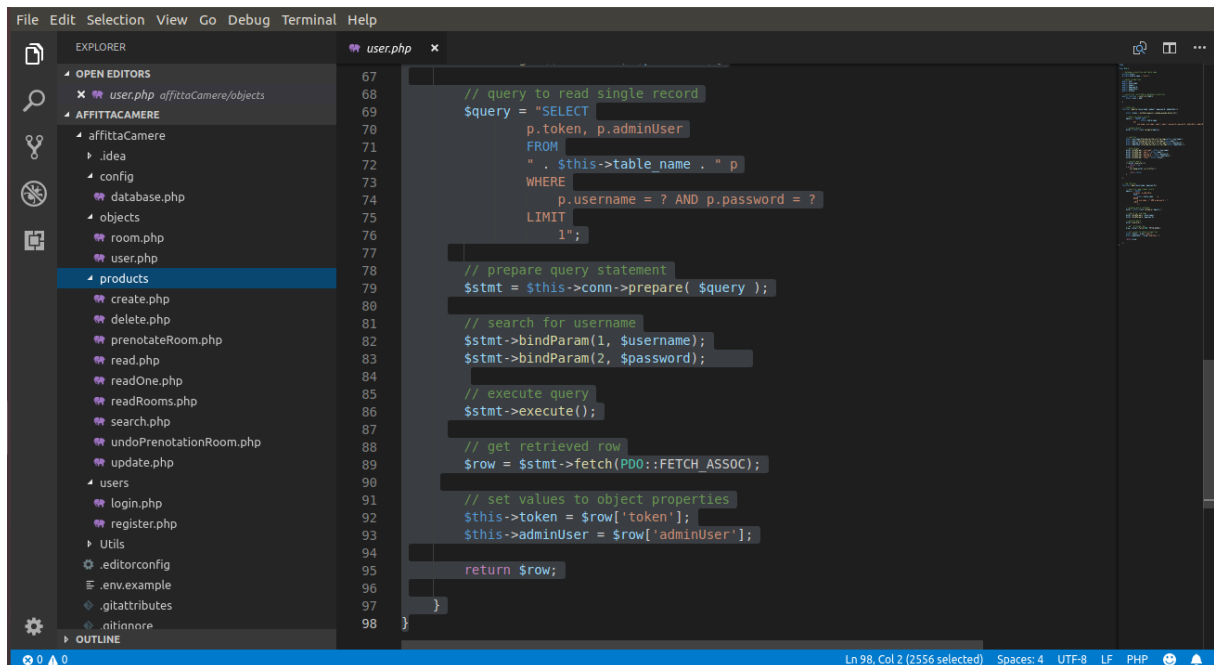
    }

}

```

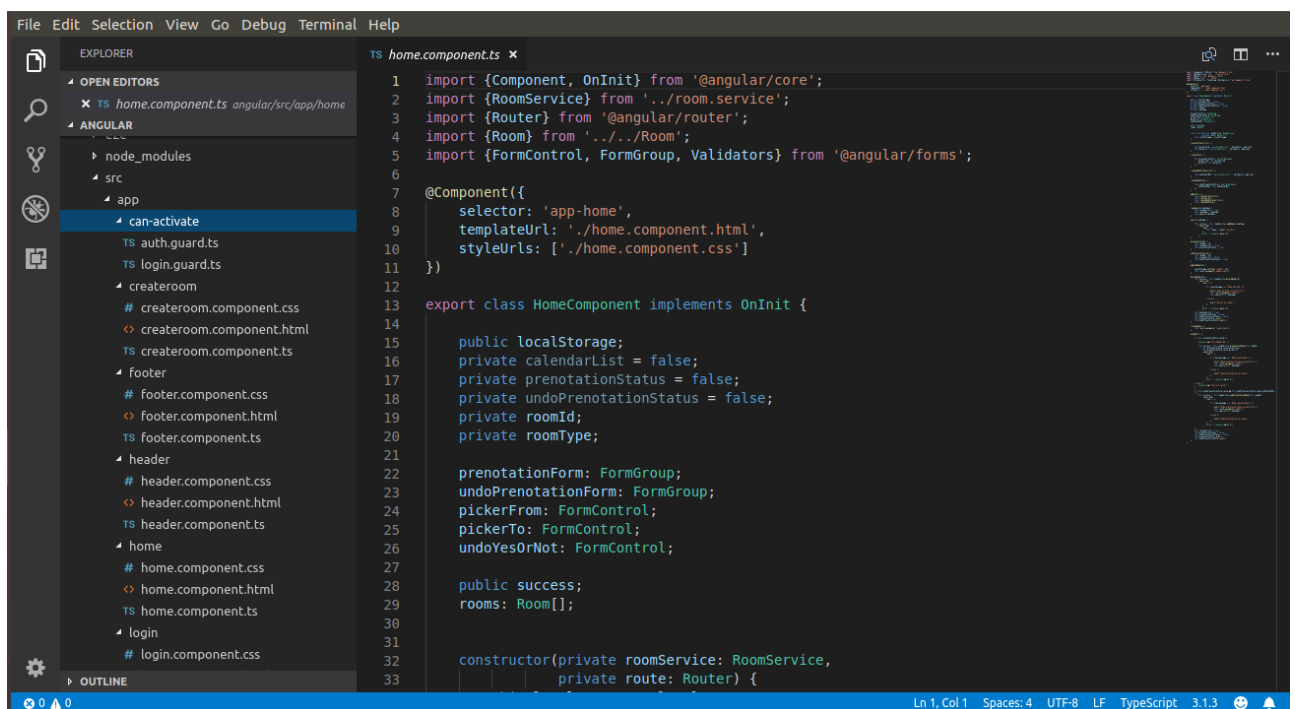
Questi hanno metodi e attributi. Tutte le funzioni che ti permettono di gestire le chiamate che arrivano lato client. In più c'è la generazione di un token che viene emesso al momento della registrazione.

Poi ci sono le sottocartelle con tutte le funzioni e chiamate end point che vengono fatte dal client, ognuna ha la sua funzione. Prende i dati o in GET o in POST a seconda del tipo di chiamata che arriva e attiva la chiamata della funzione relativa. I messaggi sono ciò che viene fatto il JSON



```
67
68 // query to read single record
69 $query = "SELECT
70     p.token, p.adminUser
71 FROM
72     " . $this->table_name . " p
73 WHERE
74     p.username = ? AND p.password = ?
75 LIMIT
76     1";
77
78 // prepare query statement
79 $stmt = $this->conn->prepare( $query );
80
81 // search for username
82 $stmt->bindParam(1, $username);
83 $stmt->bindParam(2, $password);
84
85 // execute query
86 $stmt->execute();
87
88 // get retrieved row
89 $row = $stmt->fetch(PDO::FETCH_ASSOC);
90
91 // set values to object properties
92 $this->token = $row['token'];
93 $this->adminUser = $row['adminUser'];
94
95 return $row;
96
97 }
98 }
```

Angular invece divide ogni componente in HTML, CSS e TYPESCRIPT.



```
1 import {Component, OnInit} from '@angular/core';
2 import {RoomService} from '../room.service';
3 import {Router} from '@angular/router';
4 import {Room} from '../Room';
5 import {FormControl, FormGroup, Validators} from '@angular/forms';
6
7 @Component({
8     selector: 'app-home',
9     templateUrl: './home.component.html',
10    styleUrls: ['./home.component.css']
11 })
12
13 export class HomeComponent implements OnInit {
14
15     public localStorage;
16     private calendarList = false;
17     private prenotazioneStatus = false;
18     private undoPrenotazioneStatus = false;
19     private roomId;
20     private roomType;
21
22     prenotazioneForm: FormGroup;
23     undoPrenotazioneForm: FormGroup;
24     pickerFrom: FormControl;
25     pickerTo: FormControl;
26     undoYesOrNot: FormControl;
27
28     public success;
29     rooms: Room[];
30
31     constructor(private roomService: RoomService,
32                 private route: Router) {
33     }
```


Angular, quando parte, chiama index:

```
<!doctype html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Hotel</title>
<base href="/">

<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="icon" type="image/x-icon" href="favicon.ico">

</head>
<body>
  <app-root></app-root>
</body>
</html>
```

Che è la struttura dell'html in generale, ed esso va a chiamare l'app-root

Appheader e footer sono statici e rimangono sempre così, routerauth significa che a seconda di ciò che pigi la root va cambiata su quello.

Approot prende i moduli, le librerie ecc e le passa su tutto il progetto.

I servizi sono ciò che ti permette di servire delle funzioni, appunto. Per esempio login chiama il submit e d esso chiama la funzione. Chiama i dati del form e se esso è valido gli passa i dati e prende la risposta da un'altra funzione. E così via.

La parte front end ha un menù richiamabile con f12

The screenshot shows a web browser at `localhost:4200/home` displaying a hotel booking page. The page features a night view of a grand building, a selection of room types (Normal, Comfort, Superior), and a table of available rooms. The network tab in the developer tools shows a successful GET request to `http://affittacamere.com/products/readRooms.php?type=c...` with a status code of 200 OK. The response headers include `Access-Control-Allow-Origin: *`, `Connection: keep-alive`, `Content-Type: application/json; charset=UTF-8`, `Date: Fri, 09 Nov 2018 14:06:40 GMT`, `Server: nginx/1.14.0 (Ubuntu)`, and `Transfer-Encoding: chunked`.

Select the type of the room you prefer:

Normal Comfort Superior

| Room | Price | Type | Persons | Occupated from | Occupated to | | |
|------|-------|---------|---------|----------------|--------------|---------|----------------|
| A04 | 100 | comfort | 4 | 2018-11-06 | 2018-11-08 | | Annulla Delete |
| A04 | 100 | comfort | 4 | | | Prenota | Delete |
| A04 | 100 | comfort | 4 | | | Prenota | Delete |
| A04 | 100 | comfort | 4 | | | Prenota | Delete |
| A04 | 100 | comfort | 4 | | | Prenota | Delete |

Dove siamo Contattaci

This screenshot is identical to the one above, showing the same web browser interface and network traffic details. It highlights the GET request to the `readRooms.php` endpoint and the corresponding JSON response headers.

Select the type of the room you prefer:

Normal Comfort Superior

| Room | Price | Type | Persons | Occupated from | Occupated to | | |
|------|-------|---------|---------|----------------|--------------|---------|----------------|
| A04 | 100 | comfort | 4 | 2018-11-06 | 2018-11-08 | | Annulla Delete |
| A04 | 100 | comfort | 4 | | | Prenota | Delete |
| A04 | 100 | comfort | 4 | | | Prenota | Delete |
| A04 | 100 | comfort | 4 | | | Prenota | Delete |
| A04 | 100 | comfort | 4 | | | Prenota | Delete |

Dove siamo Contattaci

Questo è il menù dal quale è possibile fare le cose e vedere le chiamate GET, che restituisce l'URL delle chiamate in JSON

Albergo Home Blog Pricing Welcome Logout

Select the type of the room you prefer:

Normal Comfort Superior

New room

Dove siamo Contattaci

localhost:4200/home

67%

Analisi pagina Console Debugger Rete

Tutti HTML CSS JS XHR Caratteri Immagini Media WS Altrc

| Stato | Metodo | File | Dominio | Origine | Tipo | Trasf |
|-------|--------|----------|--------------|---------|------|-------|
| 200 | GET | read... | affittaca... | xhr | json | 918 B |
| 200 | GET | login... | affittaca... | xhr | json | 376 B |

2 richieste 760 B di 1,26 kB trasferiti Completato: 34,02 min

Header Cookie Parametri Risposta Tempi Analisi dello stack

Filtra parametri di richiesta

Stringa query

password: Test1234
username: joshua

Durante la fase di login si intercetta la chiamata con l'avvenuta ricezione del token di 16 cifre alfanumerico.

Albergo Home Blog Pricing Welcome Logout

Select the type of the room you prefer:

Normal Comfort Superior

New room

Dove siamo Contattaci

localhost:4200/home

67%

Analisi pagina Console Debugger Rete

Tutti HTML CSS JS XHR Caratteri Immagini Media WS Altrc

| Stato | Metodo | File | Dominio | Origine | Tipo | Trasf |
|-------|--------|----------|--------------|---------|------|-------|
| 200 | GET | read... | affittaca... | xhr | json | 918 B |
| 200 | GET | login... | affittaca... | xhr | json | 376 B |

2 richieste 760 B di 1,26 kB trasferiti Completato: 34,02 min

Header Cookie Parametri Risposta Tempi Analisi dello stack

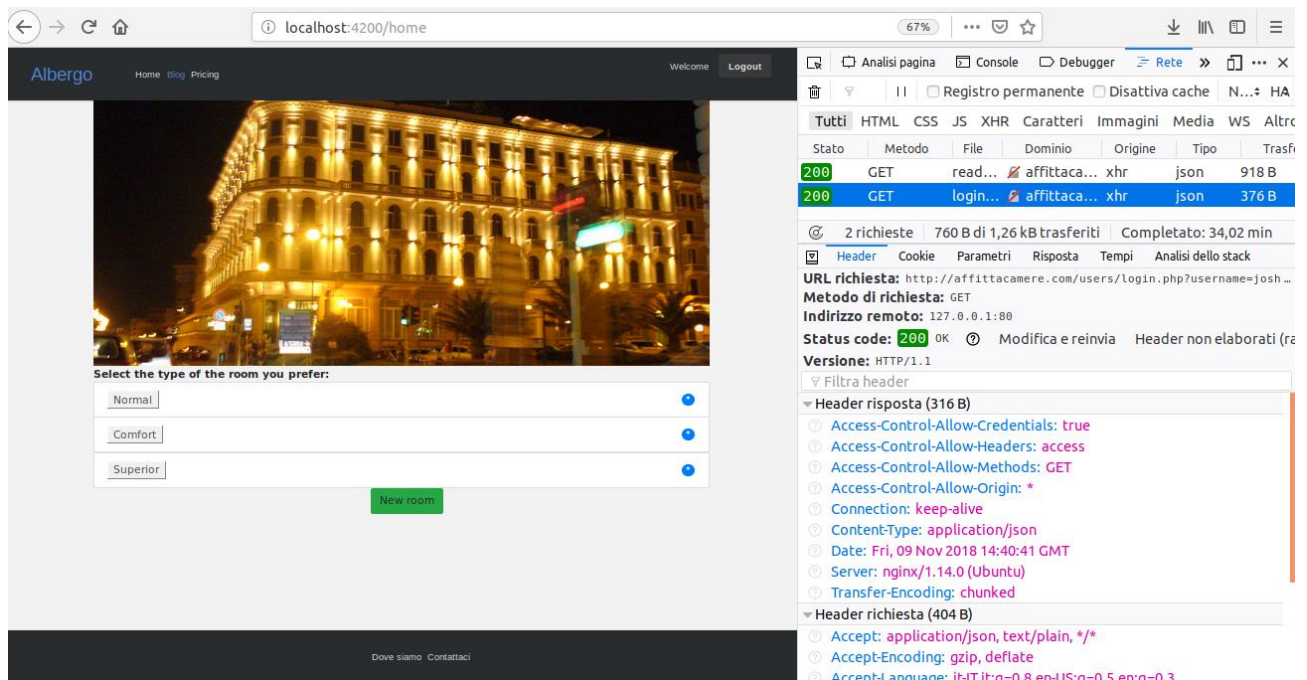
Filtra proprietà

JSON

token: 3a373953322508a53496d01c4863ecd9
adminUser: 1

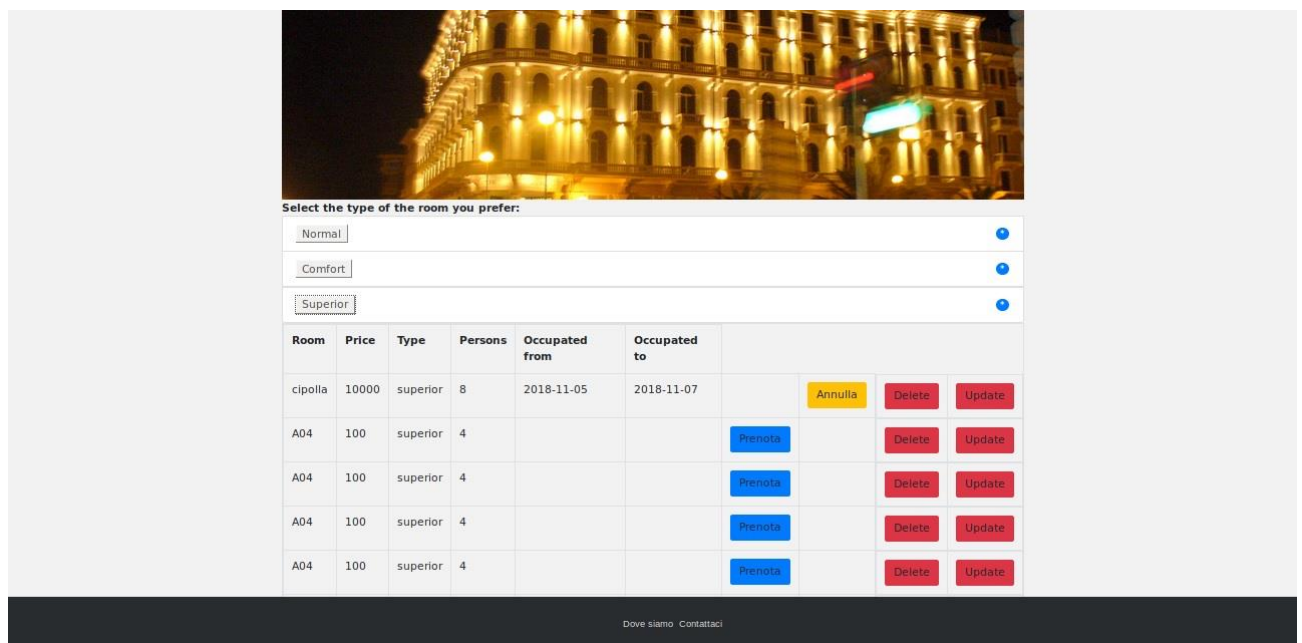
Payload risposta

```
1 {\"token\": \"3a373953322508a53496d01c4863ecd9\", \"adminUser\": \"1\"}
```



Da qui possiamo fare tutte cose e guardare le risposte alle chiamate come ad esempio il token nel login o le informazioni riguardanti le camere che abbiamo selezionato. Ci viene inoltre fornito l'URL che ci permette di vedere le chiamate in JSON.

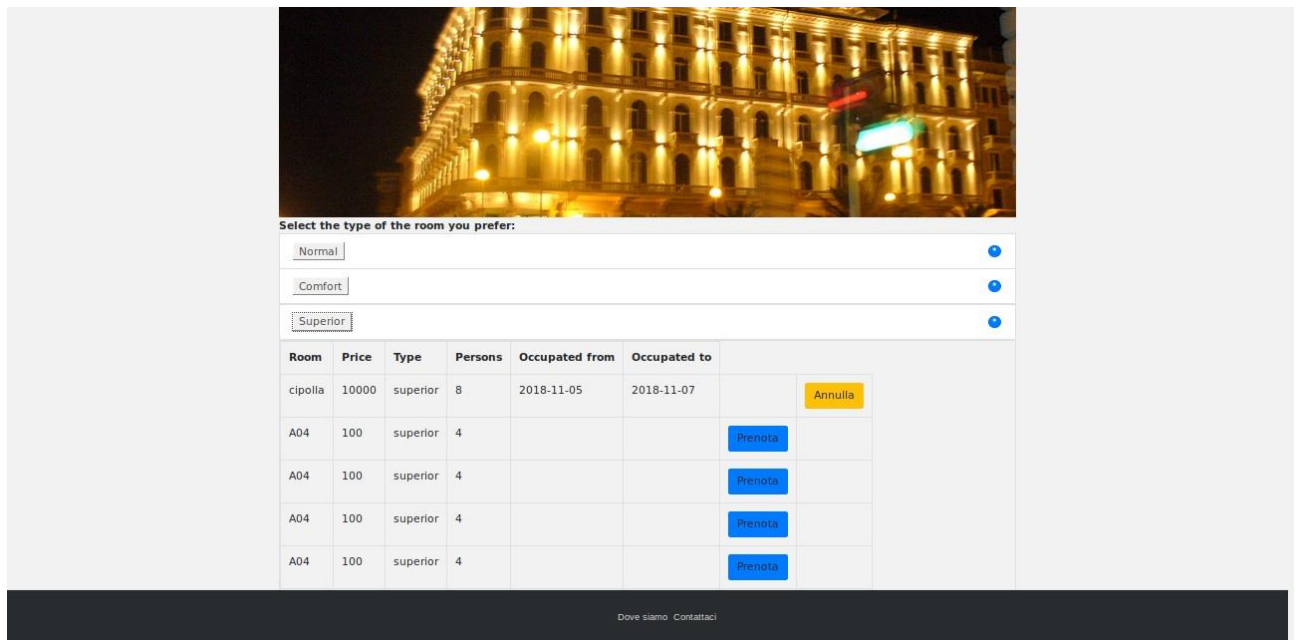
Ora seguiranno un po' di immagini del progetto che sono più esplicative perché un'immagine vale mille parole anche in informatica.



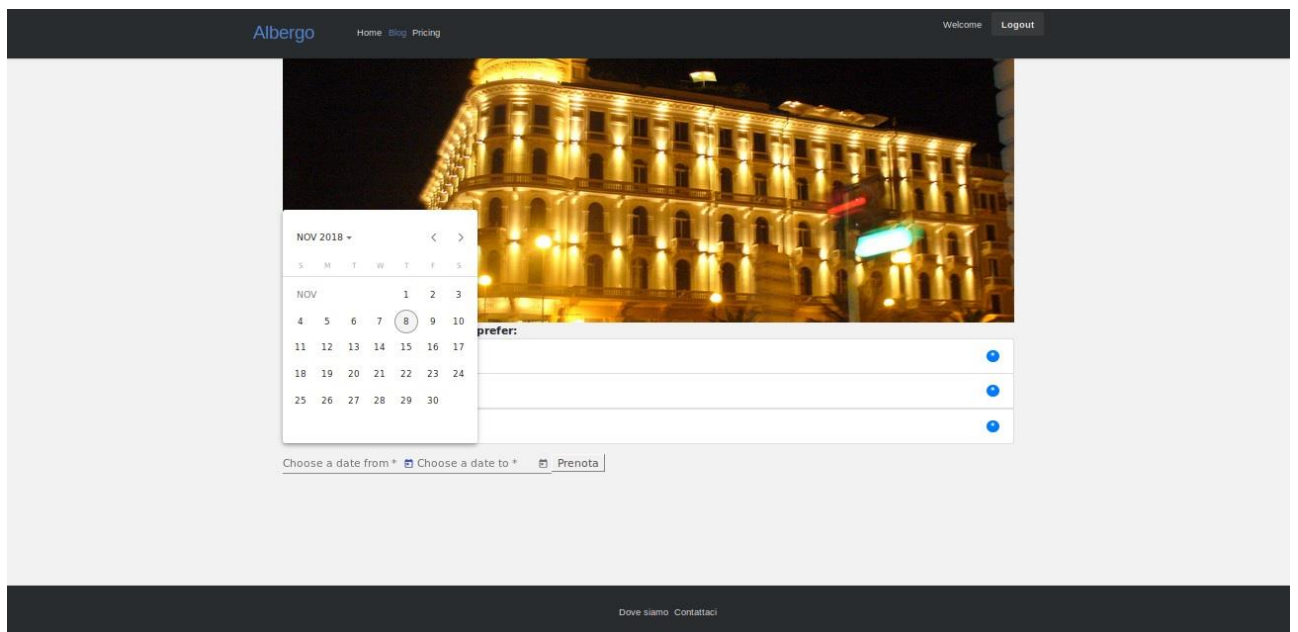
Da questo menù è possibile cancellare le camere presenti, modificarne i parametri come il nome, il numero di persone che la camera può ospitare o il prezzo e perfino effettuare prenotazioni o annullare quelle già presenti.

Nella pagina di login è possibile effettuare l'accesso sia come amministratori che come utenti. Verrà effettuato un controllo per verificare l'effettiva presenza dell'utente già registrato, ed in caso positivo verrà fatto effettuare l'accesso riportando un messaggio di avvenuto login. In caso contrario si riceverà un messaggio di errore e si rimarrà nella pagina di login per effettuare altri tentativi.

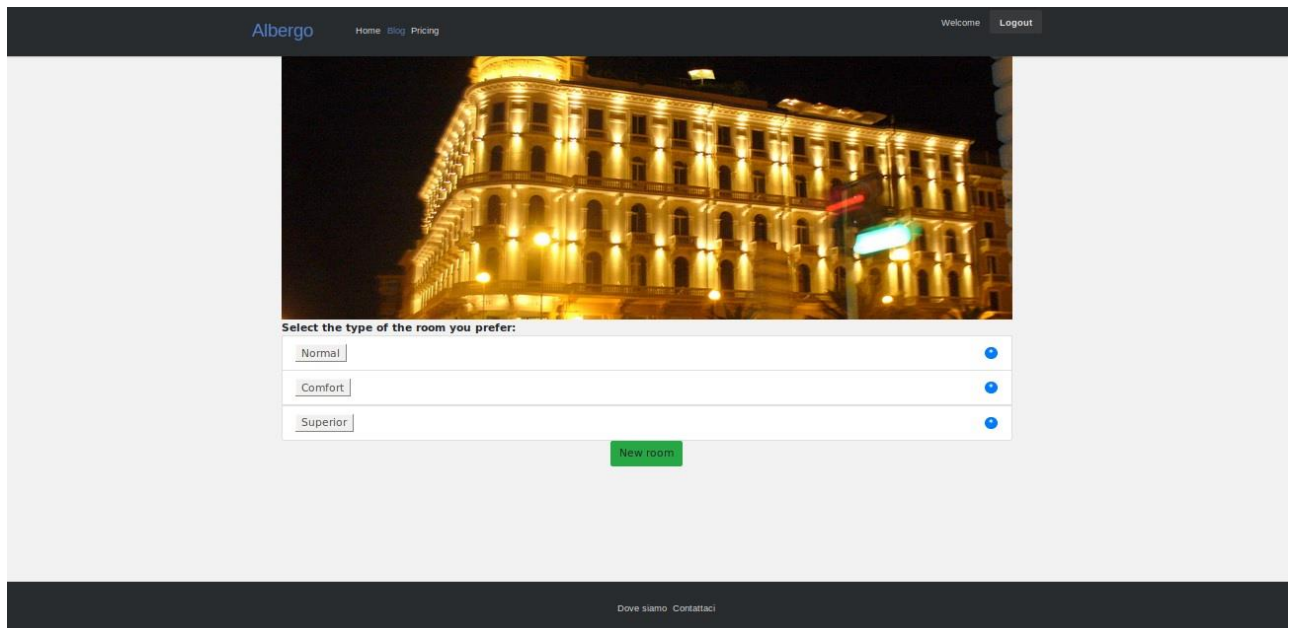
- La pagina di registrazione del sito permette di registrarsi come admin o come utente. All'inserimento di un nuovo utente il sistema genererà un token randmico di lunghezza 16.
- Da tutte le pagine è inoltre possibile effettuare il login se non si è ancora effettuato l'accesso, oppure effettuare il logout tornando così alla pagina di login.



Come già accennato l'utente può prenotare le camere servendosi degli appositi bottoni o annullare le precedenti prenotazioni.



L'utente può prenotare servendosi del calendario. In questo modo si evitano errori di digitazione della data.



A differenza dell'utente l'admin ha un bottone in più con il quale può accedere al menù di modifica delle camere presenti.

Quella dell'update della camera non c'entra perchè troppo grande.

Ora inseriremo un altro pò di codice così a caso, giusto per finire in bellezza:

La funzione Read Rooms che permette di fare le cose:

ReadRooms.php

```
<?php
// required headers
header("Access-Control-Allow-Origin: *");
header("Content-Type: application/json; charset=UTF-8");

// include database and object files
include_once '../config/database.php';
include_once '../objects/room.php';

// instantiate database and room object
$database = new Database();
```

```

$db = $database->getConnection();

// initialize object
$room = new Room($db);

// get roomType
$type=isset($_GET["type"]) ? $_GET["type"] : "";

// query rooms
$stmt = $room->readRooms($type);

$num = $stmt->rowCount();

// check if more than 0 record found
if($num>0){
    // rooms array
    $rooms_arr=array();
    $rooms_arr["records"]=array();
    // retrieve our table contents
    while ($row = $stmt->fetch(PDO::FETCH_ASSOC)){
        extract($row);
        $room_item = array(
            "id" => $id,
            "name" => $name,
            "price" => $price,
            "type" => html_entity_decode($type),
            "persons" => $persons,
            "occupied_from" => $occupied_from,

```



```
"occupied_to" => $occupied_to  
);
```

```
array_push($rooms_arr["records"], $room_item);  
}  
echo json_encode($rooms_arr);  
}  
else{  
    echo json_encode(  
array("message" => "No rooms found.")  
);  
}  
?>
```

E questi erano alcuni dei file di configurazione più importanti per il progetto

Fine relazione, grazie per l'attenzione.