**Creazione form di login ed inserire use e psw da mysql**

**User Model (Modello Utente)**

Assicurati di avere un modello User (generalmente già presente in Laravel) che rappresenti la tua tabella users. Questo modello si trova tipicamente in app/Models/User.php.

PHP

<?php

namespace App\Models;

use Illuminate\Foundation\Auth\User as Authenticatable;

use Illuminate\Notifications\Notifiable;

class User extends Authenticatable

{

use Notifiable;

protected $fillable = [

'username', // Assicurati che 'username' sia fillable

'password',

];

protected $hidden = [

'password',

];

// Se la tua colonna password non è criptata con bcrypt, potresti volerlo fare.

// Laravel si aspetta password hashate.

// Se stai usando password in chiaro, la sicurezza è un problema.

}

**3. Authentication Controller (Controller di Autenticazione)**

Crea un nuovo Controller per gestire la logica di login. Puoi generarlo con il comando Artisan:

Bash

php artisan make:controller Auth/LoginController

Poi, modifica il file app/Http/Controllers/Auth/LoginController.php in questo modo:

PHP

<?php

namespace App\Http\Controllers\Auth;

use App\Http\Controllers\Controller;

use Illuminate\Http\Request;

use App\Models\User; // Importa il modello User

use Illuminate\Support\Facades\Auth;

use Illuminate\Support\Facades\Hash; // Laravel usa Hash per le password, ma qui useremo la comparazione diretta come nel tuo codice.

class LoginController extends Controller

{

/\*\*

\* Mostra il form di login.

\*

\* @return \Illuminate\View\View

\*/

public function showLoginForm()

{

return view('auth.login'); // Questa sarà la tua vista Blade per il login

}

/\*\*

\* Gestisce la richiesta di login.

\*

\* @param \Illuminate\Http\Request $request

\* @return \Illuminate\Http\RedirectResponse

\*/

public function login(Request $request)

{

$request->validate([

'username' => 'required|string',

'password' => 'required|string',

]);

$username = $request->username;

$password = $request->password;

// Recupera l'utente per username

$user = User::where('username', $username)->first();

// \*\*ATTENZIONE\*\*: La tua logica originale confronta le password in chiaro ($password === $user['password']).

// Questo è ESTREMAMENTE INSUFFICIENTE per la sicurezza in un'applicazione web moderna.

// Laravel si aspetta che le password nel database siano hashate con bcrypt.

// Se le tue password nel database NON sono hashate, questa linea funzionerà,

// ma dovresti ASSOLUTAMENTE implementare hashing delle password (es. bcrypt) per sicurezza.

// La riga commentata sotto è come Laravel gestisce normalmente le password hashate.

// if ($user && Hash::check($password, $user->password)) { // Modo sicuro di Laravel

if ($user && $password === $user->password) { // La tua logica originale (meno sicura)

Auth::login($user); // Effettua il login dell'utente

$request->session()->regenerate();

return redirect()->intended('/index'); // Reindirizza a /index (o a una rotta desiderata dopo il login)

}

return back()->withErrors([

'username' => 'Nome utente o password non validi.',

])->onlyInput('username');

}

/\*\*

\* Effettua il logout dell'utente.

\*

\* @param \Illuminate\Http\Request $request

\* @return \Illuminate\Http\RedirectResponse

\*/

public function logout(Request $request)

{

Auth::logout();

$request->session()->invalidate();

$request->session()->regenerateToken();

return redirect('/'); // Reindirizza alla homepage o alla pagina di login

}

}

**4. Routes (Rotta)**

Definisci le rotte nel file routes/web.php per puntare al tuo controller di login:

PHP

<?php

use Illuminate\Support\Facades\Route;

use App\Http\Controllers\Auth\LoginController; // Importa il controller

/\*

|--------------------------------------------------------------------------

| Web Routes

|--------------------------------------------------------------------------

|

| Here is where you can register web routes for your application. These

| routes are loaded by the RouteServiceProvider within a group which

| contains the "web" middleware group. Now create something great!

|

\*/

// Rotta per mostrare il form di login

Route::get('/login', [LoginController::class, 'showLoginForm'])->name('login');

// Rotta per gestire la sottomissione del form di login

Route::post('/login', [LoginController::class, 'login']);

// Rotta per il logout

Route::post('/logout', [LoginController::class, 'logout'])->name('logout');

// Rotta protetta (richiede autenticazione)

Route::middleware(['auth'])->group(function () {

Route::get('/index', function () {

return view('index'); // Supponendo che tu abbia una vista 'index.blade.php'

})->name('index');

});

// Redirect dalla root (opzionale)

Route::get('/', function () {

return redirect()->route('login');

});

**5. View (Vista Blade)**

Crea un file di vista Blade per il form di login in resources/views/auth/login.blade.php. Puoi adattare il tuo HTML esistente.

Blade

<!DOCTYPE html>

<html lang="it">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Login</title>

<link rel="stylesheet" href="{{ asset('css/style.css') }}"> {{-- Assicurati che il tuo CSS sia nella cartella public/css --}}

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.6/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-4Q6Gf2aSP4eDXB8Miphtr37CMZZQ5oXLH2yaXMJ2w8e2ZtHTl7GptT4jmndRuHDT" crossorigin="anonymous">

</head>

<body>

<div class="container my-5">

<div class="row">

<div class="col-12 text-center">

<h1 class="text-white">Login</h1>

</div>

</div>

</div>

<div class="container">

<div class="row justify-content-center">

<div class="col-md-6">

<form method="post" action="{{ route('login') }}"> {{-- Usa la rotta nominata 'login' --}}

@csrf {{-- Importante per la sicurezza di Laravel --}}

<div class="mb-3 text-center">

<label for="username" class="form-label text-white">Nome Utente</label>

<input type="text" class="form-control @error('username') is-invalid @enderror" id="username" name="username" value="{{ old('username') }}" required>

@error('username')

<div class="invalid-feedback">

{{ $message }}

</div>

@enderror

</div>

<div class="mb-3 text-center">

<label for="password" class="form-label text-white">Password</label>

<input type="password" class="form-control @error('password') is-invalid @enderror" id="password" name="password" required>

@error('password')

<div class="invalid-feedback">

{{ $message }}

</div>

@enderror

</div>

<div class="col-12 text-center mt-5">

<button type="submit" class="btn btn-primary w-50">Accedi</button>

</div>

{{-- Laravel gestisce gli errori tramite le direttive @error --}}

{{-- Se hai un errore generico non legato a un campo specifico, puoi usarlo così: --}}

@if ($errors->any() && !$errors->has('username') && !$errors->has('password'))

<div class="alert alert-danger text-center mt-3" role="alert">

{{ $errors->first() }}

</div>

@endif

</form>

</div>

</div>

</div>

<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.6/dist/js/bootstrap.bundle.min.js" integrity="sha384-j1CDi7MgGQ12Z7Qab0qlWQ/Qqz24Gc6BM0thvEMVjHnfYGF0rmFCozFSxQBxwHKO" crossorigin="anonymous"></script>

</body>

</html>

E crea anche una vista resources/views/index.blade.php per la pagina di destinazione dopo il login (anche solo con un messaggio "Benvenuto!").

Blade

<!DOCTYPE html>

<html lang="it">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Dashboard</title>

</head>

<body>

<h1>Benvenuto, {{ Auth::user()->username }}!</h1>

<p>Sei loggato correttamente.</p>

<form method="POST" action="{{ route('logout') }}">

@csrf

<button type="submit">Logout</button>

</form>

</body>

</html>

**Gestione Password in Laravel**

**Fase 1: Preparare e Eseguire il Comando per Hashare le Password**

1. Creare il Comando Artisan HashPasswords

Se non lo hai già fatto, genera il file del comando:

Bash

php artisan make:command HashPasswords

Questo creerà il file app/Console/Commands/HashPasswords.php.

2. Codice del Comando HashPasswords.php

Modifica il file app/Console/Commands/HashPasswords.php per includere la logica di hashing:

PHP

<?php

namespace App\Console\Commands;

use Illuminate\Console\Command;

use App\Models\User; // Importa il modello User

use Illuminate\Support\Facades\Hash; // Importa la facciata Hash

class HashPasswords extends Command

{

/\*\*

\* The name and signature of the console command.

\*

\* @var string

\*/

protected $signature = 'app:hash-passwords'; // Nome del tuo comando

/\*\*

\* The console command description.

\*

\* @var string

\*/

protected $description = 'Hash all plain text passwords in the database.';

/\*\*

\* Execute the console command.

\*/

public function handle()

{

$users = User::all();

foreach ($users as $user) {

// Controlla se la password NON è già hashata (o se non è un hash valido di bcrypt)

// Questo previene di hashare due volte password già hashate

if (!Hash::info($user->password)['algo'] ?? null) {

$user->password = Hash::make($user->password); // Hashing della password

$user->save();

$this->info("Password for user '{$user->username}' has been hashed.");

} else {

$this->info("Password for user '{$user->username}' is already hashed.");

}

}

$this->info('All passwords processed.');

}

}

3. Registrare il Comando in app/Console/Kernel.php

Apri il file app/Console/Kernel.php e aggiungi la classe del tuo comando all'array $commands:

PHP

<?php

namespace App\Console;

use Illuminate\Console\Scheduling\Schedule;

use Illuminate\Foundation\Console\Kernel as ConsoleKernel;

class Kernel extends ConsoleKernel

{

/\*\*

\* The Artisan commands provided by your application.

\*

\* @var array

\*/

protected $commands = [

\App\Console\Commands\HashPasswords::class, // Aggiungi questa riga

];

/\*\*

\* Define the application's command schedule.

\*/

protected function schedule(Schedule $schedule): void

{

// $schedule->command('inspire')->hourly();

}

/\*\*

\* Register the commands for the application.

\*/

protected function commands(): void

{

$this->load(\_\_DIR\_\_.'/Commands');

require base\_path('routes/console.php');

}

}

4. Rigenerare l'Autoloading di Composer

Questo passaggio è fondamentale per permettere a Laravel di trovare la nuova classe del comando:1

Bash

composer dump-autoload

5. Eseguire il Comando per Hashare le Password

Ora puoi eseguire il comando per hashare le password nel tuo database:

Bash

php artisan app:hash-passwords

Dovresti vedere dei messaggi nel terminale che indicano quali password sono state hashate.

**Fase 2: Aggiornare il Controller di Login**

Dopo aver eseguito il comando e aver hashato le password nel database, devi modificare il tuo LoginController per utilizzare il metodo di verifica sicuro.

Codice del Controller LoginController.php

Modifica il file app/Http/Controllers/Auth/LoginController.php:

PHP

<?php

namespace App\Http\Controllers\Auth;

use App\Http\Controllers\Controller;

use Illuminate\Http\Request;

use App\Models\User;

use Illuminate\Support\Facades\Auth;

use Illuminate\Support\Facades\Hash; // Assicurati che sia importato

class LoginController extends Controller

{

/\*\*

\* Mostra il form di login.

\*

\* @return \Illuminate\View\View

\*/

public function showLoginForm()

{

return view('auth.login');

}

/\*\*

\* Gestisce la richiesta di login.

\*

\* @param \Illuminate\Http\Request $request

\* @return \Illuminate\Http\RedirectResponse

\*/

public function login(Request $request)

{

$request->validate([

'username' => 'required|string',

'password' => 'required|string',

]);

$username = $request->username;

$password = $request->password;

$user = User::where('username', $username)->first();

// QUESTA È LA RIGA CORRETTA E SICURA:

if ($user && Hash::check($password, $user->password)) { // Usa Hash::check() per le password hashate

Auth::login($user);

$request->session()->regenerate();

return redirect()->intended('/index');

}

return back()->withErrors([

'username' => 'Nome utente o password non validi.',

])->onlyInput('username');

}

/\*\*

\* Effettua il logout dell'utente.

\*

\* @param \Illuminate\Http\Request $request

\* @return \Illuminate\Http\RedirectResponse

\*/

public function logout(Request $request)

{

Auth::logout();

$request->session()->invalidate();

$request->session()->regenerateToken();

return redirect('/');

}

}

**Fase 3: La Tua Vista Blade per il Login (già fornita)**

La tua vista resources/views/auth/login.blade.php sembra essere già impostata correttamente con la direttiva @csrf e la gestione degli errori. Non sono necessarie modifiche qui.

Blade

<x-layout>

@section('title', 'Homepage')

<div class="container my-5">

<div class="row">

<div class="col-12 text-center">

<h1 class="">Login</h1>

</div>

</div>

</div>

<div class="container">

<div class="row justify-content-center">

<div class="col-md-6">

<form method="post" action="{{ route('login') }}"> {{-- Usa la rotta nominata 'login' --}}

@csrf {{-- Importante per la sicurezza di Laravel --}}

<div class="mb-3 text-center">

<label for="username" class="form-label">Nome Utente</label>

<input type="text" class="form-control @error('username') is-invalid @enderror" id="username" name="username" value="{{ old('username') }}" required>

@error('username')

<div class="invalid-feedback">

{{ $message }}

</div>

@enderror

</div>

<div class="mb-3 text-center">

<label for="password" class="form-label">Password</label>

<input type="password" class="form-control @error('password') is-invalid @enderror" id="password" name="password" required>

@error('password')

<div class="invalid-feedback">

{{ $message }}

</div>

@enderror

</div>

<div class="col-12 text-center mt-5">

<button type="submit" class="btn btn-primary w-50">Accedi</button>

</div>

@if ($errors->any() && !$errors->has('username') && !$errors->has('password'))

<div class="alert alert-danger text-center mt-3" role="alert">

{{ $errors->first() }}

</div>

@endif

</form>

</div>

</div>

</div>

</x-layout>

**Fase 4: Configurazioni Aggiuntive e Debug**

* **Configurazione del Database (.env):** Assicurati che le tue credenziali di database siano corrette nel file .env.

Snippet di codice

DB\_CONNECTION=mysql

DB\_HOST=sql107.infinityfree.com

DB\_PORT=3306

DB\_DATABASE=if0\_38876061\_dario

DB\_USERNAME=if0\_38876061

DB\_PASSWORD=ominoverde87

* **Modello User (app/Models/User.php):** Verifica che il tuo modello User sia corretto e che username sia nei $fillable.
* **Rotte (routes/web.php):** Le tue rotte dovrebbero essere definite correttamente.

PHP

<?php

use Illuminate\Support\Facades\Route;

use App\Http\Controllers\Auth\LoginController;

Route::get('/login', [LoginController::class, 'showLoginForm'])->name('login');

Route::post('/login', [LoginController::class, 'login']);

Route::post('/logout', [LoginController::class, 'logout'])->name('logout');

Route::middleware(['auth'])->group(function () {

Route::get('/index', function () {

return view('index');

})->name('index');

});

Route::get('/', function () {

return redirect()->route('login');

});

* **Pulizia Cache Laravel:** Dopo aver apportato modifiche, è sempre buona norma pulire la cache di Laravel:

Bash

php artisan cache:clear

php artisan config:clear

php artisan route:clear

php artisan view:clear

* **Permessi delle Cartelle:** Assicurati che le cartelle storage e bootstrap/cache abbiano i permessi di scrittura corretti (ad esempio, chmod -R 775 storage bootstrap/cache e chown -R www-data:www-data storage bootstrap/cache o l'utente del tuo server web).
* **APP\_KEY nel .env:** Verifica che la variabile APP\_KEY sia impostata nel tuo file .env. Se manca, generarla con php artisan key:generate.

Seguendo questi passaggi, dovresti risolvere l'errore "419 Page Expired" e implementare un sistema di autenticazione più sicuro con password hashate.