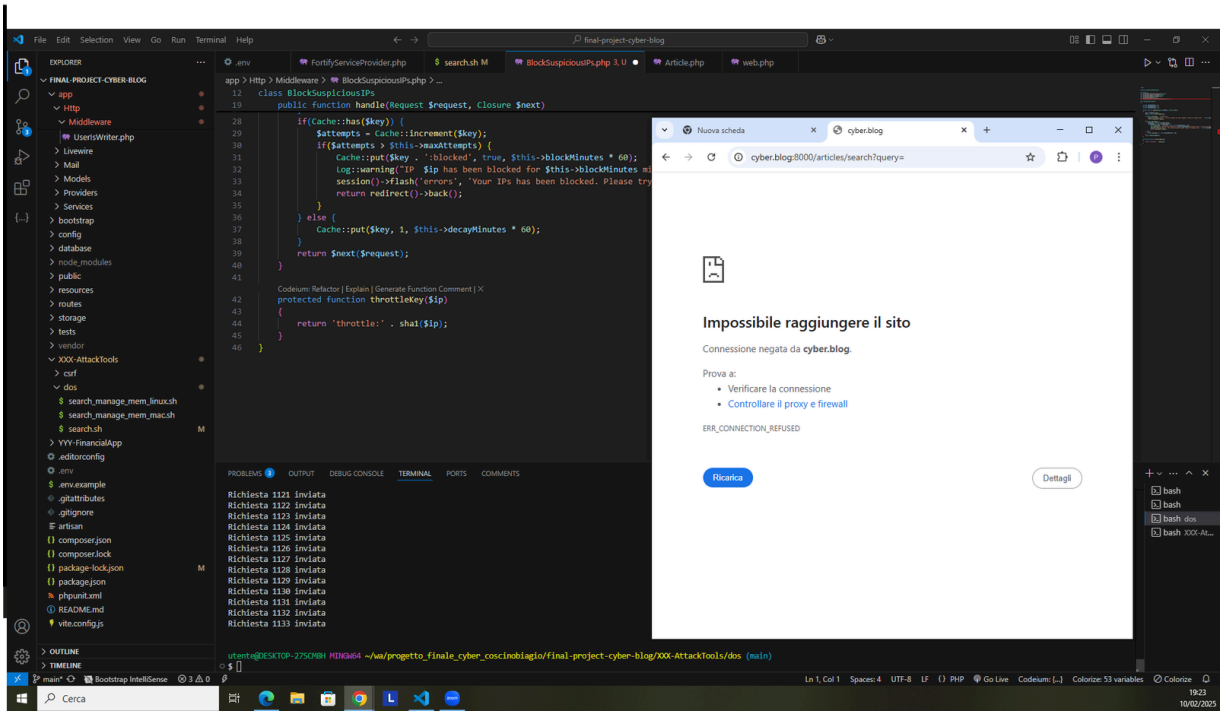
DOCUMENTAZIONE PROGETTO FINALE CYBER SECURITY.

In questa documentazione elencheremo i passaggi effettuati per risolvere le varie problematiche che abbiamo riscontrato nella creazione del progetto.
Iniziamo a parlare della CHALLENGE 1:
Nella prima challenge abbiamo simulato un attacco DoS. nella suddetta challenge abbiamo risolto il problema creando un blocco temporaneo dell'Ip implementandola con una rate Limiter Globale.



CHALLENGE 2:
nella challenge 2 invece abbiamo simulato un attacco CSRF, dove successivamente siamo andati a lavorare sulle rotte cambiando il metodo delle

rotta per eliminare la parte incriminata.

```
    undoArticle');
});

// Admin routes
Route::middleware(['admin', 'admin.local'])->group(function () {
    Route::get('/admin/dashboard', [AdminController::class, 'dashboard'])->name('admin.dashboard');
    Route::patch('/admin/{user}/set-admin', [AdminController::class, 'setAdmin'])->name('admin.setAdmin');
    Route::patch('/admin/{user}/set-revisor', [AdminController::class, 'setRevisor'])->name('admin.
    setRevisor');
    Route::patch('/admin/{user}/set-writer', [AdminController::class, 'setWriter'])->name('admin.setWriter');

    Route::put('/admin/edit/tag/{tag}', [AdminController::class, 'editTag'])->name('admin.editTag');
    Route::delete('/admin/delete/tag/{tag}', [AdminController::class, 'deleteTag'])->name('admin.deleteTag');
    Route::put('/admin/edit/category/{category}', [AdminController::class, 'editCategory'])->name('admin.
    editCategory');
    Route::delete('/admin/delete/category/{category}', [AdminController::class, 'deleteCategory'])->name
    ('admin.deleteCategory');
    Route::post('/admin/category/store', [AdminController::class, 'storeCategory'])->name('admin.
    storeCategory');
    Route::post('/admin/tag/store', [AdminController::class, 'storeTag'])->name('admin.storeTag');
});
```

CHALLENGE 3:
Nella challenge corrente abbiamo semplicemente implementato i log nella sezione del login per rendere più sicuro il tutto. Abbiamo implementato il tutto nell'AdminController.

```php
public function setAdmin(User $user)
{
    $user->is_admin = true;
    $user->save();
    Log::info("L'utente $user->name è stato promosso a admin da " . Auth::user()->name);

    return redirect(route('admin.dashboard'))->with('message', "$user->name is now administrator");
}

public function setRevisor(User $user)
{
    $user->is_revisor = true;
    $user->save();
    Log::info("L'utente $user->name è stato promosso a revisore da " . Auth::user()->name);

    return redirect(route('admin.dashboard'))->with('message', "$user->name is now revisor");
}
```

CHALLENGE 4:
Nella challenge 4 abbiamo simulato un attacco SSRF, in poche parole abbiamo fatto si che l'API Key venisse sostituita da una da noi creata per far si che la potessimo rendere sicura lontano dagli attacchi SSRF. infatti abbiamo sostituito l'API key nel .env.

```
VITE_APP_NAME="${APP_NAME}"

NEWSAPI_API_KEY=5fbe92849d5648eabcbe072a1cf91473
```

```php
public function __construct()
{
    $this->httpService = app(HttpService::class);
}

public function fetchNews()
{
    // sicuro

    $allowedUrls = ["https://newsapi.org/v2/top-headlines?country=it ", "https://newsapi.org/v2/
    top-headlines?country=gb", "https://newsapi.org/v2/top-headlines?country=us"];

    if (!in_array($this->selectedApi, $allowedUrls)) {
        return redirect()->route('articles.create')->with('error', 'Invalid URL');
    };

    $apikey = env("NEWSAPI_API_KEY");

    $this->news = json_decode($this->httpService->getRequest($this->selectedApi . "&apiKey=$apikey"),
    true);
```

CHALLENGE 5:

Nella challenge 5 per far si che l'utente leggesse l'articolo in maniera sicura abbiamo usato dei tools come ad esempio BurpSuite. per far si che l'utente leggesse l'articolo in maniera sicura abbiamo creato un meccanismo che filtra il messaggio prima di salvarlo e poi che lo rendesse sicuro in fase di lettura.

```php
/**
 * Store a newly created resource in storage.
 */
public function store(Request $request, HtmlFilterService $htmlservice)
{
    Gate::authorize('create', Article::class);

    $articleData = $request->validate([
        'title' => 'required|unique:articles|min:5',
        'subtitle' => 'required|min:5',
        'body' => 'required|min:10',
        'image' => 'required|image',
        'category' => 'required',
        'tags' => 'required',
    ]);

    $articleData['body'] = $htmlservice->filterHtml($articleData['body']);

    // $article = Article::create($articleData);
    $article = Article::create([
        'title' => $articleData['title'],
        'subtitle' => $articleData['subtitle'],
        'body' => $articleData['body'],
        'image' => $request->file('image')->store('public/images'),
        'category_id' => $articleData['category'],
```

CHALLENGE 6:

Nella challenge 6 abbiamo semplicemente implementato la pagina di profilo di un'utente lasciandola vulnerabile appositamente per capire meglio le proprietà dei fillable.

```php
<?php
// app/Http/Controllers/UserController.php

namespace App\Http\Controllers;

use App\Models\User;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;

class UserController extends Controller
{
    public function store(Request $request)
    {
        // Solo un amministratore può modificare `is_admin`
        if (!Auth::user()->is_admin) {
            // Rimuovi 'is_admin' dalla richiesta, se presente
            $request->merge(['is_admin' => false]);
        }

        $validated = $request->validate([
            'name' => 'required|string|max:255',
            'email' => 'required|email|unique:users,email',
            'password' => 'required|string|min:8|confirmed',
        ]);

        User::create($validated);  // Crea l'utente
    }
}
```

```php
<?php

namespace App\Models;

// use Illuminate\Contracts\Auth\MustVerifyEmail;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Foundation\Auth\User as Authenticatable;
use Illuminate\Notifications\Notifiable;

class User extends Authenticatable
{
    use HasFactory, Notifiable;

    /**
     * The attributes that are mass assignable.
     *
     * @var array<int, string>
     */
    protected $fillable = [
        'name',
        'email',
        'password',
        // 'is_admin',
        // 'is_revisor',
        // 'is_writer'
    ];
```

```php
     */
    protected function casts(): array
    {
        return [
            'email_verified_at' => 'datetime',
            'password' => 'hashed',
        ];
    }

    public function articles()
    {
        return $this->hasMany(Article::class);
    }
}
```