Cybersécurité : analyse de logs web

Application web PHP/MySQL avec visualisations Chart.js

Projet personnel réalisé en cours de formation :

Architecture MVC · Base de données MySQL · Graphiques interactifs · Authentification

Projet individuel

Sommaire

I. Introduction 2
II. Analyse du besoin 3
2.1 Objectifs du projet 3
2.2 Fonctionnalités attendues 3
III. Architecture et conception 4
3.1 Structure MVC 4
3.2 Base de données 4
3.3 Chargement des logs 5
IV. Développement 6
4.1 Authentification 6
4.2 Vues et visualisation 7
4.3 Génération de couleurs aléatoires 8
V. Résultat final 9
VI. Conclusion 12

I. Introduction

Ce projet consiste à développer une application web permettant l'analyse et la visualisation de fichiers de logs générés par un serveur web Apache/WAMP. L'enjeu cybersécurité est central : un fichier de logs contient des informations précieuses sur les tentatives d'accès, les adresses IP clientes, les ressources demandées et les comportements anormaux.

L'application est développée en PHP conception MVC (Modèle-Vue-Contrôleur), s'appuie sur une base de données MySQL, et utilise la bibliothèque Chart.js pour produire des représentations graphiques interactives.

Objectif : Concevoir une interface d'administration sécurisée permettant à un administrateur d'importer, d'analyser et de visualiser les logs d'un ou plusieurs serveurs web, avec filtres par IP, par URL et par date.

- Lecture et parsing de fichiers de logs Apache
- Stockage en base de données MySQL
- Authentification administrateur par session PHP
- Visualisation par graphiques interactifs (barres, polaire, ligne)
- Interface responsive avec barre de navigation latérale

II. Analyse du besoin

2.1 Objectifs du projet

Les logs d'un serveur web constituent une source d'information critique en cybersécurité. Leur analyse permet de détecter des comportements suspects (scan de ports, tentatives d'injection, accès à des ressources sensibles), de surveiller la charge du serveur et d'identifier les ressources les plus sollicitées.

2.2 Fonctionnalités attendues

Fonctionnalité	Description
Authentification	Connexion sécurisée par login/mot de passe stockés en BDD
Import des logs	Lecture automatique du fichier access.log au chargement
Analyse par IP	Regroupement et comptage des requêtes par adresse IP
Analyse par URL	Regroupement des accès par ressource demandée
Analyse par date	Regroupement des connexions par tranche horaire
Gestion serveurs	Visualisation des serveurs enregistrés en base de données
Visualisations	Graphiques barres, polaire et ligne via Chart.js

III. Architecture et conception

3.1 Structure MVC

L'application respecte le patron de conception MVC. Chaque couche a un rôle précis et les fichiers sont organisés en conséquence :

Couche	Fichier(s)	Rôle
Modèle	modele.class.php	Connexion PDO, requêtes SQL (INSERT, SELECT, DELETE)
Vue	vue_*.php	Affichage HTML, tableaux de données, graphiques Chart.js
Contrôleur	controleur.class.php	Orchestration : parsing des logs, appels BDD ...
Point d'entrée	index.php	Routage des pages, gestion des sessions, chargement des vues

3.2 Base de données

La base de données **logs_LM_26** est structurée autour de trois tables principales et de trois vues SQL permettant les agrégations nécessaires aux graphiques :

```
drop database if exists logs_lm_26;
create database logs_lm_26;
use logs_lm_26;
```

```
create table admin(
    id int(5) not null auto_increment,
    nom varchar(50),
    prenom varchar(50),
    login varchar(100),
    mdp varchar(255),
    primary key (id)
);

create table serveur(
    idserveur int(5) not null auto_increment,
    ip varchar(50),
    nom varchar(50),
    systeme varchar(50),
    primary key(idserveur)
);

create table logs(
    idlogs int(5) not null auto_increment,
    ip varchar(50),
    datelogs varchar(50),
    url varchar(50),
    idserveur int(5) not null,
    primary key(idlogs),
    foreign key(idserveur) references serveur(idserveur)
);
```

```
drop view if exists ip_nb;
create view ip_nb as (select ip,count(*) as nb from logs group by ip order by nb);

drop view if exists url_nb;
create view url_nb as (select url,count(*) as nb from logs group by url order by nb);

drop view if exists dt_nb;
create view dt_nb as
(select left(datelogs, length(datelogs)-6) as dt,count(*) as nb from logs group by dt order by nb);
```

3.3 Chargement des logs

À chaque chargement de page, le contrôleur vide la table **logs** puis relit intégralement le fichier *access.log*. Chaque ligne est découpée par espaces pour en extraire l'IP, la date et l'URL demandée, avant insertion en base :

```php
public function chargerLogs($idserveur){
    $this->unModele->deletelogs();
    $fichier = fopen("C:\wamp64\logs\access.log", "r");
    while(!feof($fichier)){
        $ligne = fgets($fichier, 4096);
        if (strlen($ligne) > 40) {
        $tab = explode(" ",$ligne);
        $ip = $tab[0];
        $dt = substr($tab[3], 1);

        if(isset($tab[6])){
        $url = $tab[6];
        }else{
            $url = "URL trop courte";
        }
        $data = array(
            "ip"=>$ip,
            "datelogs"=>$dt,
            "url"=>$url,
            "idserveur"=>$idserveur
        );
        $this->unModele->insertlogs($data);
        }
    }
    fclose($fichier);
}
```

Méthode de 'parsing' et d'insertion des logs dans la base de données

IV. Développement

4.1 Authentification

L'accès à l'application est restreint par un système de session PHP. Le formulaire de connexion envoie les identifiants en POST ; le contrôleur interroge la table **admin** via une requête préparée PDO (protection contre les injections SQL). En cas de succès, les informations de l'utilisateur sont stockées en session ; les pages protégées redirigent vers la connexion si la session est absente.

```php
public function selectAdminWhere($login){
    $requete = "SELECT * FROM admin WHERE login = :login";
    $data = array("login"=>$login);
    $exe = $this->unPdo->prepare($requete);
    $exe->execute($data);
    return $exe->fetch();
}
```

```php
if(isset($_POST['connexion'])){
    $login = $_POST['login'];
    $mdp = $_POST['pass'];

    $unAdmin = $unControleur->selectAdminWhere($login);

    if($unAdmin == null){
        $_SESSION['msg-erreur-connexion'] = 'identifiants incorrect !';
    }else{
        if($unAdmin && password_verify($mdp,$unAdmin['mdp'])){
            $_SESSION['login'] = $unAdmin['login'];
            $_SESSION['nom'] = $unAdmin['nom'];
            $_SESSION['prenom'] = $unAdmin['prenom'];
            header("location:index.php?page=accueil");
        }else{
            $_SESSION['msg-erreur-connexion'] = 'identifiants incorrects !';
        }
    }
}
```

Gestion de l'authentification par session PHP avec requête préparée PDO.

La protection des pages sensibles est centralisée dans le routeur :

```php
$page = $_GET['page'] ?? 'connexion';
$pagesProtegees = ['ip','url','accueil','deconnexion'];
if(!isset($_SESSION['login']) && in_array($page,$pagesProtegees)){
    $page = 'connexion';
}
```

Contrôle d'accès centralisé dans le routeur principal

4.2 Vues et visualisations

Chaque vue PHP récupère ses données via le contrôleur, les formate pour Chart.js et génère le graphique côté client. Trois types de représentation sont utilisés selon la nature de l'analyse :

Vue	Type de graphique	Données affichées
vue_ip.php	Barres (bar)	Nombre de requêtes par adresse IP
vue_url.php	Ligne (line)	Nombre d'accès par URL demandée
vue_dt.php	Polaire (polarArea)	Nombre de connexions par tranche horaire

```php
<?php
  $logs = $unControleur->selectAll('ip_nb');
  $ip = array();
  $nb = array();
  $cl = array();
  foreach ($logs as $unLog):
?>

<tr>
  <td><?= $unLog['ip'] ?></td>
  <td><?= $unLog['nb'] ?></td>
</tr>
<?php
  $ip[] = $unLog['ip'];
  $nb[] = $unLog['nb'];
  $cl[] = $unControleur->genererCouleur();
?>
<?php endforeach; ?>
<?php
  $chaineIp = '"'.implode('", "', $ip).'"';
  $chaineCl = '"'.implode('", "', $cl).'"';
  $chaineNb = implode(',', $nb);
?>
```

```javascript
<canvas id="bar-chart" width="600" height="400"></canvas>
<script type="text/javascript">
  new Chart(document.getElementById("bar-chart"), {
    type: 'bar',
    data: {
      labels: [<?= $chaineIp ?>],
      datasets: [
        {
          label: "Nombre de connexions",
          backgroundColor: [<?= $chaineCl ?>],
          data: [<?= $chaineNb ?>]
        }
      ]
    },
    options: {
      legend: { display: false },
      title: {
        display: true,
        text: 'Connexions par IP'
      }
    }
  });
</script>
```

Génération des données PHP et injection dans Chart.js (vue IP)

4.3 Génération de couleurs aléatoires

Chaque série de données reçoit une couleur hexadécimale générée aléatoirement par une méthode du contrôleur. L'utilisation de **random_int()** (cryptographiquement sûr en PHP 7+) garantit une bonne distribution des couleurs :

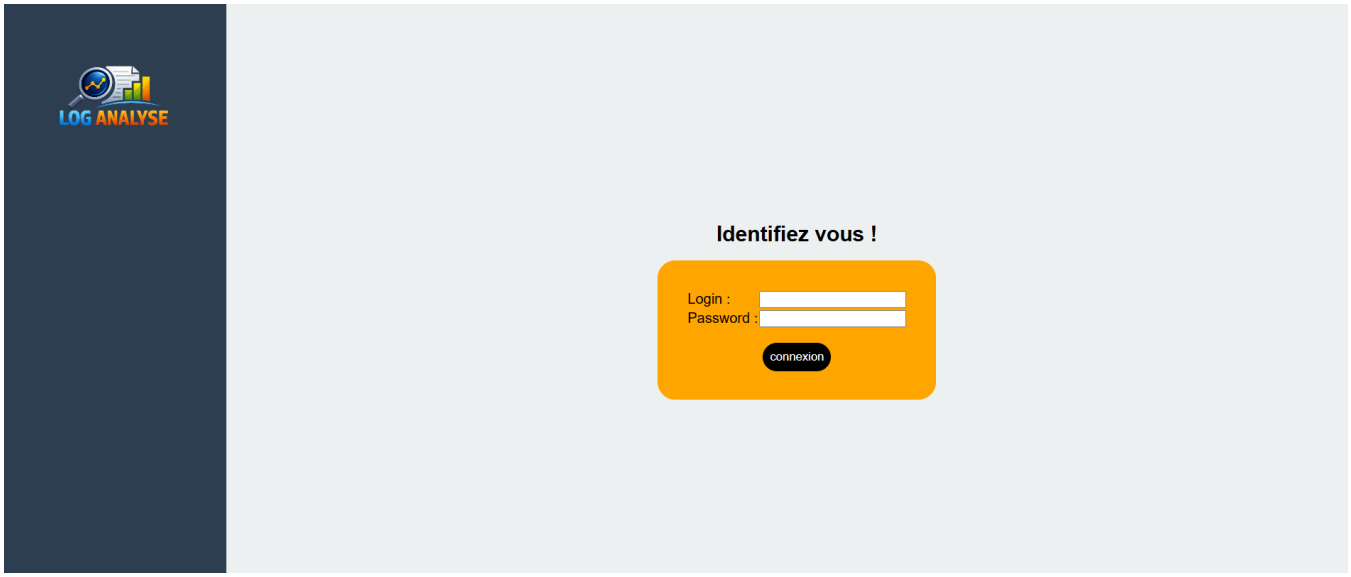
```php
public function genererCouleur(){
    $couleur = "#";
    $chaine = "0123456789ABCDEF";

    for($i=1;$i<=6;$i++){
        $couleur .= $chaine[random_int(0,strlen($chaine)-1)];
    }
    return $couleur;
}
```

Génération d'une couleur hexadécimale aléatoire pour les séries Chart.js.

V. Résultat final

Écran de connexion

La page d'accueil affiche un formulaire de connexion centré et stylisé. Sans session active, toutes les routes protégées redirigent vers cette page. Un message d'erreur s'affiche en rouge en cas d'identifiants incorrects.



Interface de connexion sécurisée avec gestion des erreurs de session.

Analyse par IP

La vue **ip_nb** affiche un tableau listant chaque adresse IP ayant accédé au serveur avec le nombre de requêtes associé, ordonné par fréquence (vue SQL **ip_nb**). Un graphique en barres colorées est généré dynamiquement côté client par Chart.js.

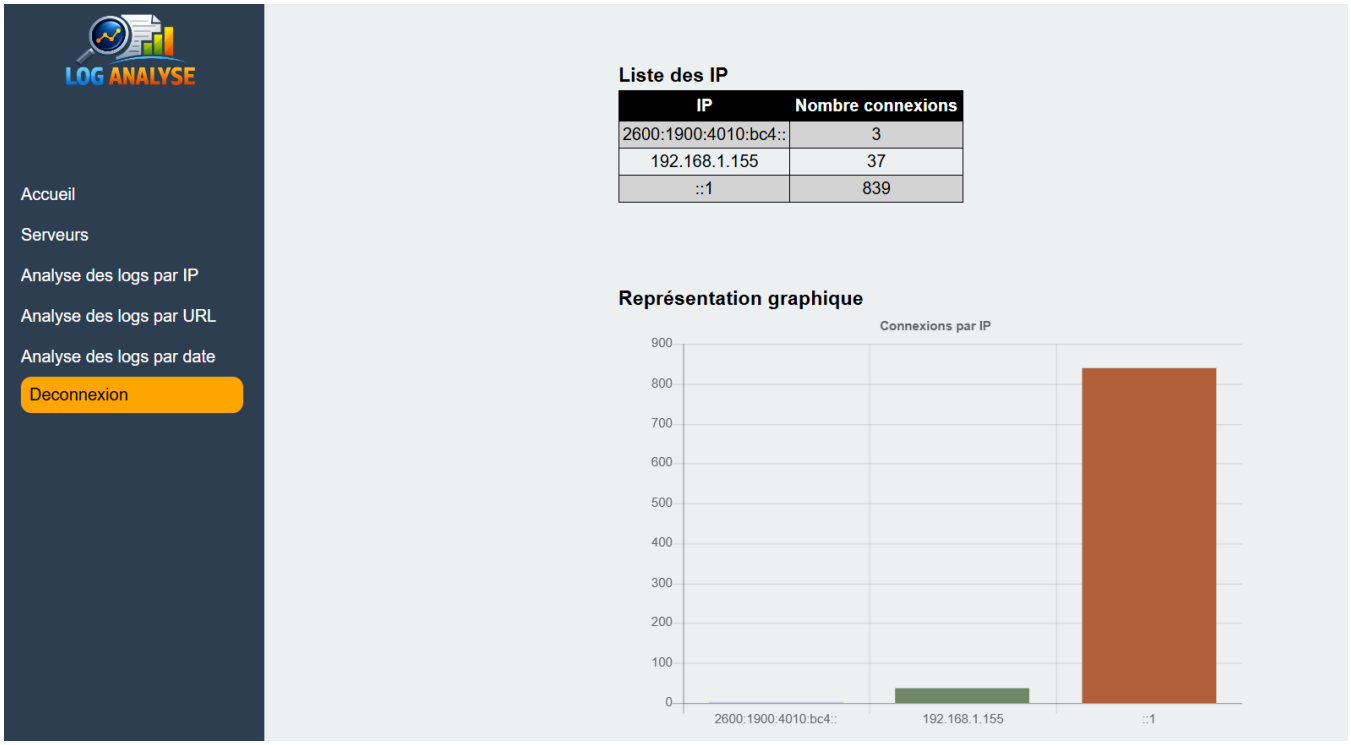
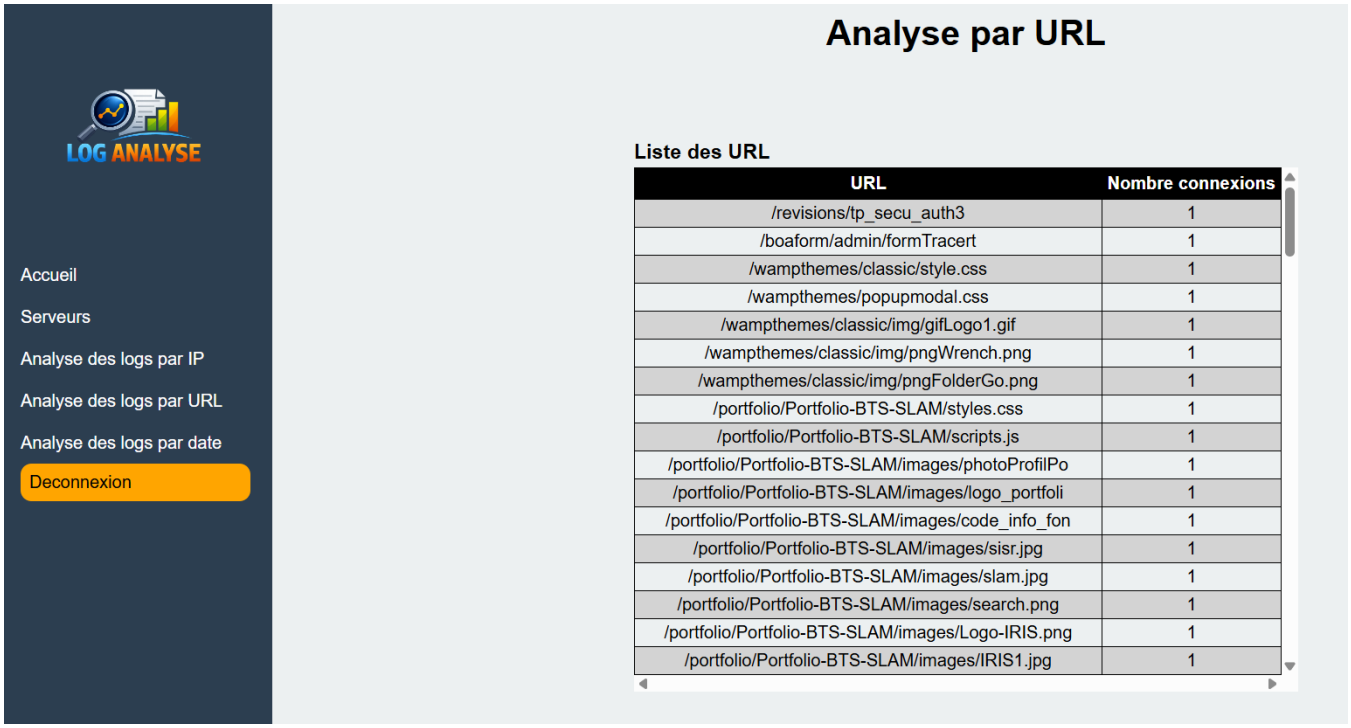


Tableau et graphique en barres des connexions par adresse IP.

Analyse par URL

La vue **url_nb** affiche un tableau listant chaque URL avec le nombre de connexions associé, ordonné par fréquence. Un graphique sous forme de courbe est généré dynamiquement côté client par Chart.js.



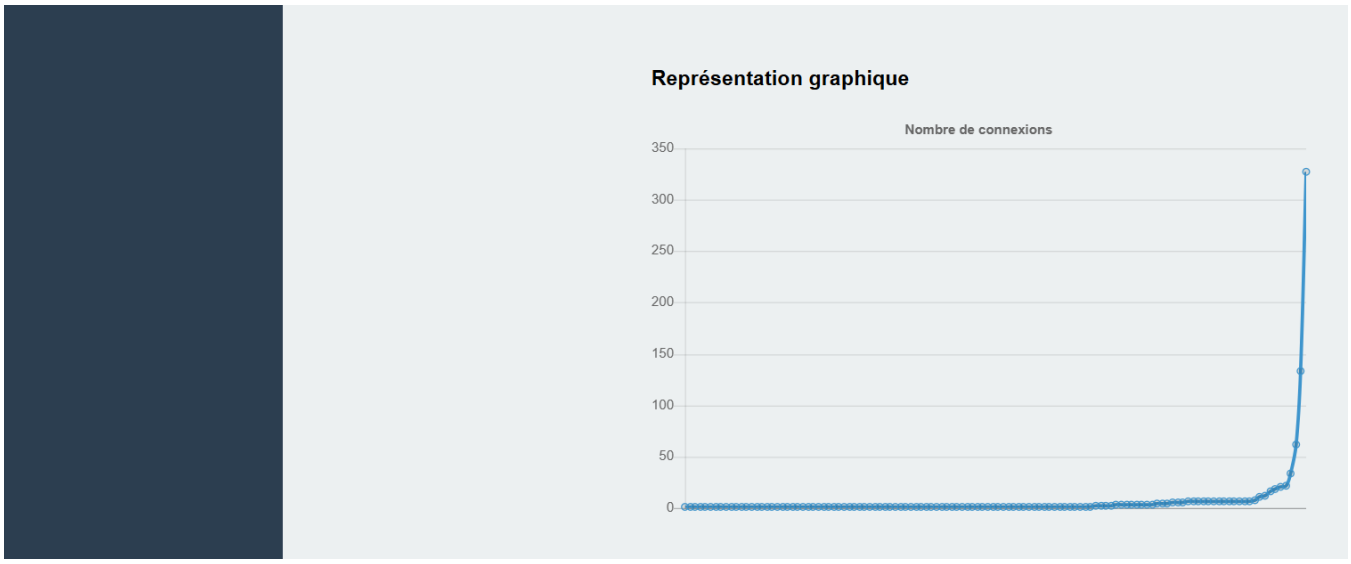
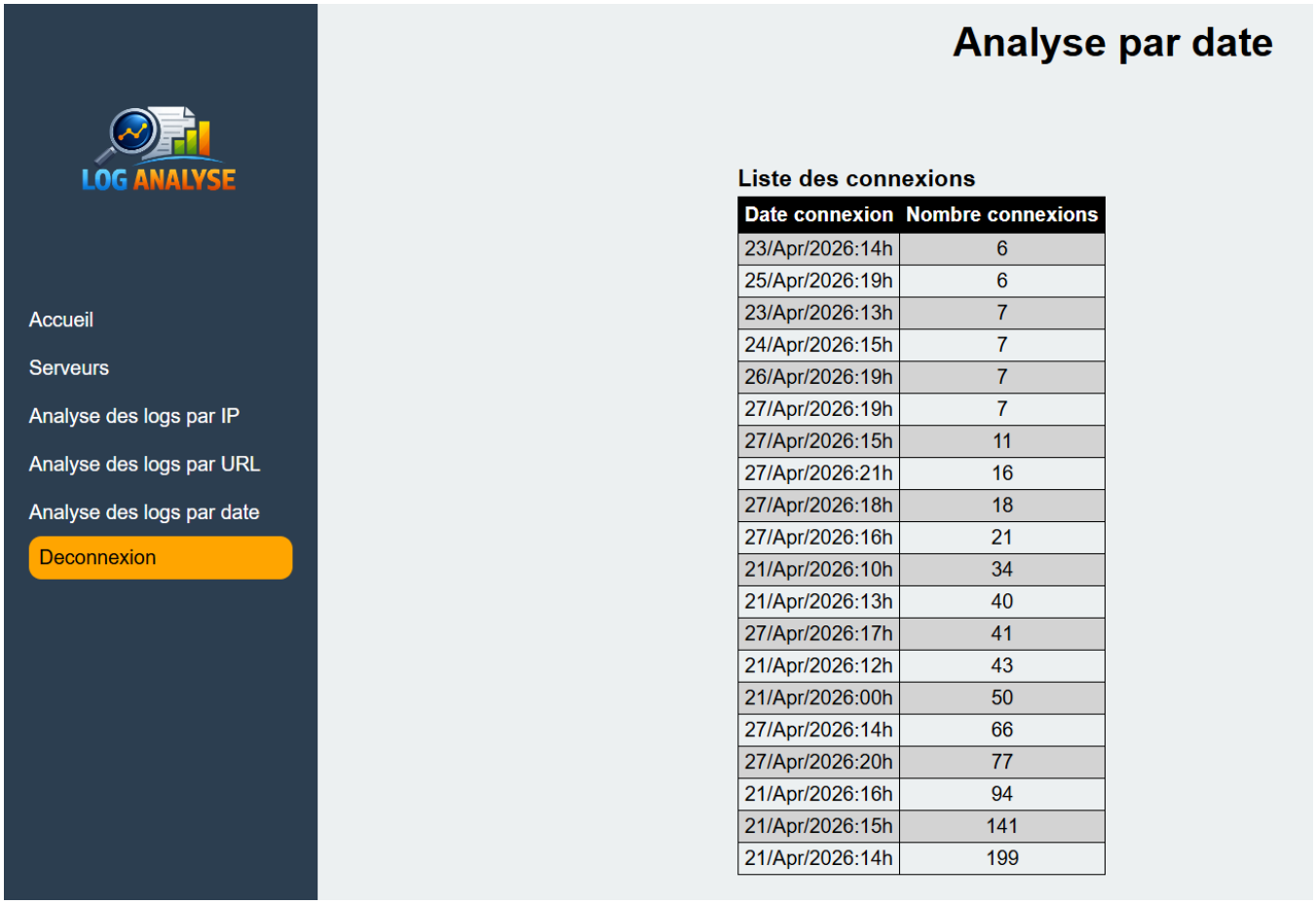


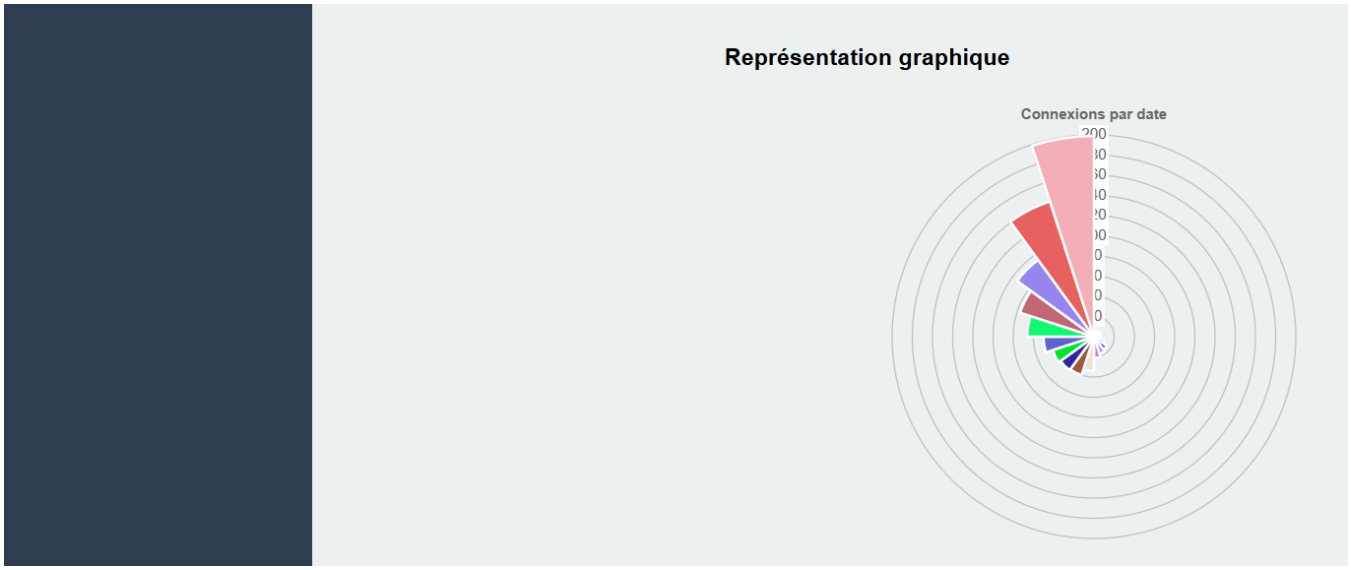
Tableau et graphique sous forme de courbe des connexions aux URL.

Analyse par date / heure

La vue **dt_nb** agrège les logs par tranche horaire (format tronqué à l'heure). Les données sont représentées sous forme de graphique polaire, particulièrement adapté à la visualisation de distributions cycliques dans le temps.



Date connexion	Nombre connexions
23/Apr/2026:14h	6
25/Apr/2026:19h	6
23/Apr/2026:13h	7
24/Apr/2026:15h	7
26/Apr/2026:19h	7
27/Apr/2026:19h	7
27/Apr/2026:15h	11
27/Apr/2026:21h	16
27/Apr/2026:18h	18
27/Apr/2026:16h	21
21/Apr/2026:10h	34
21/Apr/2026:13h	40
27/Apr/2026:17h	41
21/Apr/2026:12h	43
21/Apr/2026:00h	50
27/Apr/2026:14h	66
27/Apr/2026:20h	77
21/Apr/2026:16h	94
21/Apr/2026:15h	141
21/Apr/2026:14h	199



Graphique polaire des connexions regroupées par nombre et date/ tranche horaire.

VI. Conclusion

La réalisation de cette application d'analyse de logs en PHP constitue une expérience concrète alliant développement web et problématiques de cybersécurité. L'architecture MVC adoptée garantit une séparation claire des responsabilités et facilite la maintenance du code. L'utilisation de PDO avec des requêtes préparées assure une protection efficace contre les injections SQL.

Bilan technique

- Conception d'une architecture MVC en PHP orienté objet
- Modélisation et requêtes SQL avancées (vues, agrégations, clés étrangères)
- Sécurisation des accès : sessions PHP, requêtes préparées PDO, usage BpCrypt
- Parsing de fichiers texte structurés (format Common Log Format Apache)
- Intégration d'une bibliothèque JavaScript tierce (Chart.js) dans un contexte PHP
- Conception d'une interface avec barre de navigation latérale

Compétences mobilisées

- Analyser un besoin métier et le traduire en modèle de données relationnelles.
- Organiser un projet PHP en couches logiques indépendantes (MVC).
- Sécuriser une application web contre les vulnérabilités courantes (injections SQL, accès non authentifiés).
- Produire des visualisations de données exploitables par un administrateur non technique.

Perspectives d'évolution

- Détection automatique des IPs suspectes (seuil de requêtes anormal).
- Sélection dynamique du serveur et du fichier de logs à analyser.
- Export des données en CSV ou PDF directement depuis l'interface.
- Ajout de filtres temporels (plage de dates) sur les analyses.

> *Ce projet illustre concrètement les mécanismes essentiels du développement web sécurisé en PHP, tout en produisant un outil fonctionnel et directement exploitable dans un contexte professionnel d'administration système ou de SOC (Security Operations Center).*