# RÉSUMÉ EXPLICATIF DU PROJET

# Explication

L'objectif du projet est de développer une application Java capable d'analyser une adresse IP ou un réseau pour détecter des vulnérabilités, en utilisant plusieurs outils disponibles sur Kali Linux.
L'utilisateur fournit uniquement une cible (IP ou réseau) via une interface web. Cette interface enregistre la demande dans la base de données MySQL dans la table scanner_runs.

L'application Java fonctionne ensuite automatiquement. Elle récupère les entrées dont le statut est "pending" et lance systématiquement quatre outils de scan :

- Nmap
- Masscan
- Nikto
- WhatWeb

Chaque outil analyse la même cible et renvoie des informations différentes : ports ouverts, services détectés, vulnérabilités web, technologies utilisées, etc.

L'application Java convertit chaque résultat en objet "Finding" puis les enregistre dans la table findings. Chaque "Finding" inclut la sévérité, une description, la cible concernée, et surtout l'outil qui a généré le résultat (source_tool).
Une fois tous les outils exécutés, l'application met à jour le statut du scan en "done".

Le site web (et éventuellement une application mobile) peut ensuite consulter la base pour afficher les résultats. L'application Java ne communique qu'avec la base

de données, ce qui simplifie l'architecture et permet au serveur web d'être indépendant.

L'architecture utilisée permet également d'ajouter facilement de nouveaux outils plus tard : il suffit de créer une nouvelle classe Java implémentant l'interface ToolScanner.

# Présentation du squelette du programme Java

## Main class : ScanApp

**Rôle :**

Classe principale qui démarre l'application, récupère les scans en attente dans la base de données et lance tous les outils de scan disponibles (Nmap, Masscan, Nikto, WhatWeb).

### Attributs :

- DatabaseManager dbManager — gestion de la base de données
- List<ToolScanner> scanners — liste des outils de scan utilisés

### Méthodes (entêtes) :

- main(String[] args) — point d'entrée.

- processPendingScans() — exécute tous les scans en attente.
  runSingleScan(ScannerRun run) — lance tous les outils sur une cible donnée.

## Classe : ScannerRun

Rôle :

Représente une exécution de scan (un "run") demandée par le site web.

### Attributs :

- id : identifiant unique.
- target : cible scannée (IP ou réseau).
- startTime : date et heure de début.
- endTime : date et heure de fin.
- status : état (pending | running | done | error).

### Méthodes (entêtes) :

- getters / setters.
- updateStatus(String newStatus).

## Classe : Finding

Rôle :

Représente une vulnérabilité ou information détectée par un outil de scan.

### Attributs :

- id : identifiant unique.
- scannerRunId : référence vers le scan associé.
- sourceTool : outil ayant trouvé la faille (nmap, masscan…).
- severity : niveau (INFO | LOW | MEDIUM | HIGH | CRITICAL).

- title : titre court.
- description : description détaillée.
- target : hôte/port affecté (ex : 192.168.1.12:80).
- details : informations techniques (texte).
- createdAt : date d'enregistrement.

### Méthodes (entêtes) :

- getters / setters.
- summary() — résumé court de la faille.

## Interface : ToolScanner

Rôle :

Standardise le fonctionnement de tous les outils de scan.
Chaque outil doit respecter ce contrat.

### Méthodes (entêtes) :

- String getName() — nom de l'outil.
- List<Finding> scanTarget(String target) — exécute le scan et renvoie des Findings.

Classes scanners : NmapScanner, MasscanScanner, NiktoScanner, WhatWebScanner

Rôle :

Implémentent l'interface ToolScanner pour exécuter ou simuler les outils.

### Attributs (exemples) :

- String nmapBinary, masscanBinary, etc.

Méthodes (entêtes) :

- String getName() — nom de l'outil.
- List<Finding> scanTarget(String target) — exécution ou simulation du scan.

## Classe : DatabaseManager

Rôle :

Gère la connexion MySQL et les opérations d'enregistrement dans la base.

### Attributs :

- Connection connection — connexion JDBC.

### Méthodes (entêtes) :

- connect() — ouvre la connexion.
- close() — ferme la connexion.
- getPendingScans() — récupère les scans en attente.
- updateScanStatus(long runId, String status) — met à jour le statut.
- saveFindings(List<Finding> findings, long runId) — insère les résultats.

# EXEMPLE BDD

## 1) Table des scans : scanner_runs

Chaque ligne = un scan demandé par le site web sur une cible (IP ou réseau).
Ce scan correspond au lancement de tous les outils (Nmap, Masscan, Nikto, WhatWeb).

Je mettrais :

- id : identifiant du scan
- target : la cible (IP, nom de domaine ou réseau, ex : 192.168.1.10 ou 10.0.0.0/24)
- start_time : date/heure du début du scan
- end_time : date/heure de fin
- status : état du scan
    - pending = en attente
    - running = en cours
    - done = terminé
    - error = erreur pendant un outil

.

## 2) Table des vulnérabilités / résultats : findings

Chaque ligne = un résultat trouvé par un des outils.

Je mettrais :

- id : identifiant de la vulnérabilité

- scanner_run_id : l'ID du scan auquel ce résultat appartient (lien vers scanner_runs.id)
- source_tool : le nom de l'outil qui a trouvé ça (nmap, masscan, nikto, whatweb)
- severity : niveau de gravité (INFO, LOW, MEDIUM, HIGH, CRITICAL)
- title : petit titre court (ex : "Port 80 ouvert")
- description : explication un peu plus longue
- target : la cible précise (ex : 192.168.1.10:80)
- details : texte libre avec les détails techniques (sortie de l'outil, bannière, etc.)
- created_at : date/heure d'enregistrement du résultat