Formation Backend

JOUR 1 - INTRODUCTION ET SÉCURITÉ DES SYSTÈMES ET BDD

Formation GlowCommerce - Niveau BAC +5

Java 21 | Spring Boot 3.2 | PostgreSQL 16 | Docker | Spring Security

Objectifs de la formation

4.5 Jours

Sécurité Backend

Comprendre et appliquer les principes fondamentaux de sécurité : authentification, autorisation, chiffrement et protection des données sensibles.

Compétence C25

DevSecOps & CI/CD

Intégrer la sécurité dès la conception (Shift Left) et automatiser les tests de sécurité dans les pipelines d'intégration continue.

Compétence C18

Monitoring & Incidents

Mettre en place une observabilité complète avec Prometheus/Grafana pour détecter les anomalies et gérer les incidents en temps réel.

Compétence C6

Optimisation Performance

Analyser et optimiser les performances d'une application e-commerce à fort trafic (bases de données, cache, requêtes).

Compétence C19

Tour de table & Présentations

Introduction

Pour chaque participant

- Prénom et nom
- Expérience en développement backend
- Langages & frameworks connus
- Attentes pour cette formation
- Un projet marquant réalisé

Règles du jeu

- Questions libres
- Entraide (binômes)
- Pause (90 min)
- Support en ligne

Collaboration et partage d'expérience sont clés pour progresser ensemble.

Format interactif | Démos live | TPs guidés | Feedback continu

Contexte GlowCommerce

Business Case

Plateforme e-commerce de cosmétiques en pleine croissance

100 000+
Clients Actifs
Croissance de 15% par mois

5 000+
Produits Référencés
Catalogue cosmétique varié

500 / jour
Commandes Moyennes
Pic à 2 000 lors du Black Friday

⚠ ENJEUX TECHNIQUES & BUSINESS

- ✓ Disponibilité 24/7 critique
- ✓ Performance (temps < 2s)
- ✓ Conformité RGPD stricte
- ✓ Sécurité des paiements

High Traffic App

Pourquoi la sécurité est critique ?

Source : IBM Cost of Data Breach Report 2023

Coût moyen d'une brèche

$4.45 Millions

Augmentation de 15% sur 3 ans

Temps de détection & contrôle

277 Jours

Presque 9 mois avant résolution complète

Impact Business GlowCommerce :
Une seule faille peut compromettre la confiance de 100 000+ clients et entraîner des amendes RGPD jusqu'à 20M€ ou 4% du CA mondial.

Exemples Réels & Amendes

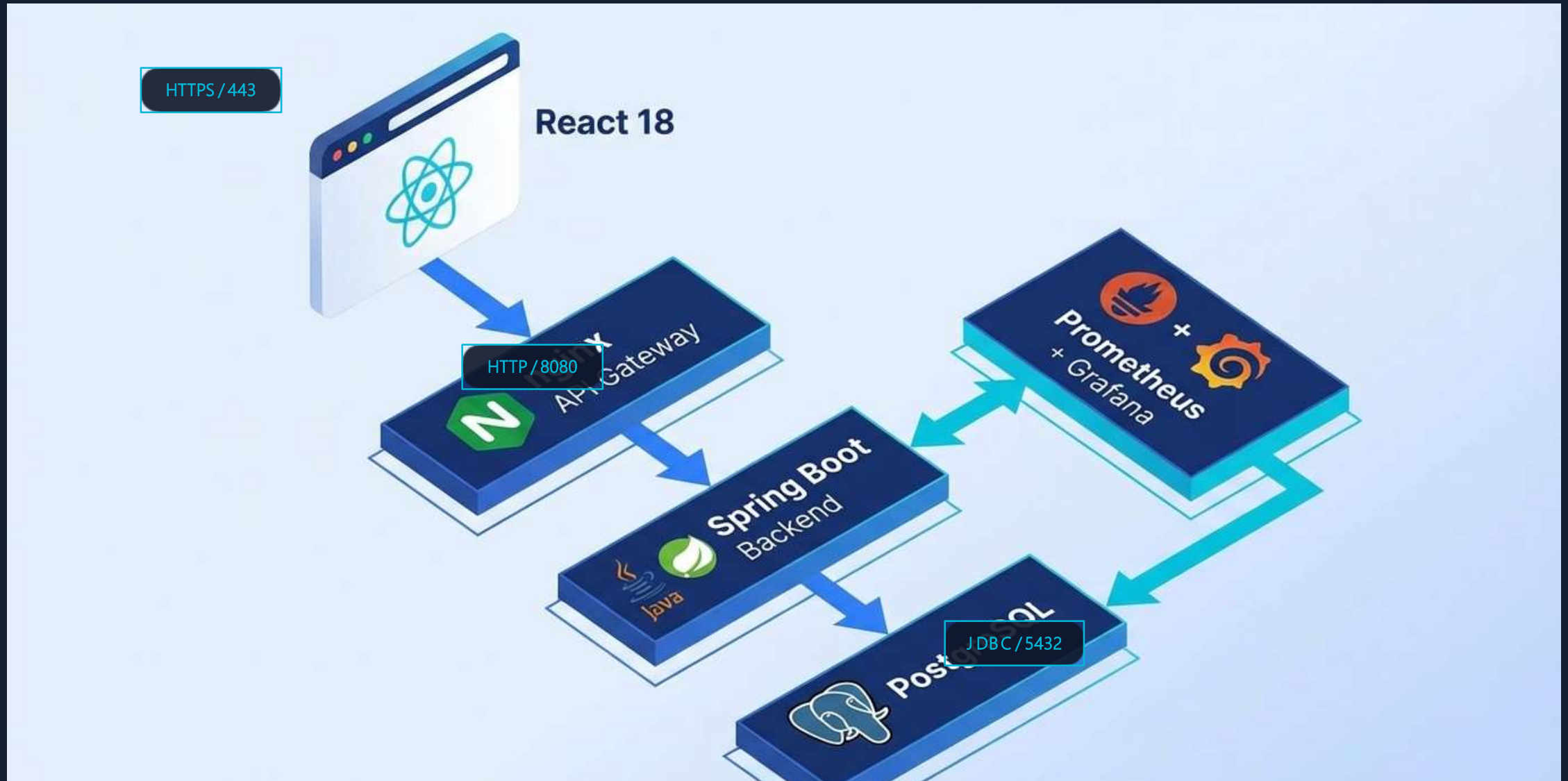
Capital One (2019)	$80M Amende
British Airways (2018)	£20M Amende
Marriott (2018)	£18.4M Amende

Perte de confiance client & Réputation

Chute du Chiffre d'Affaires & Conversion

Vue d'ensemble Architecture

Microservices-ready



Backend Spring Boot

Architecture en couches (Layered Architecture)

Java 21 | Spring Boot 3.2

Controller Layer (API REST)

Point d'entrée de l'application. Gère les requêtes HTTP, valide les entrées, sérialise les réponses JSON et gère les erreurs via @ExceptionHandler.

Service Layer (Business Logic)

Cœur de l'application. Contient la logique métier, les règles de gestion, les transactions (@Transactional) et l'orchestration des données.

Repository Layer (Data Access)

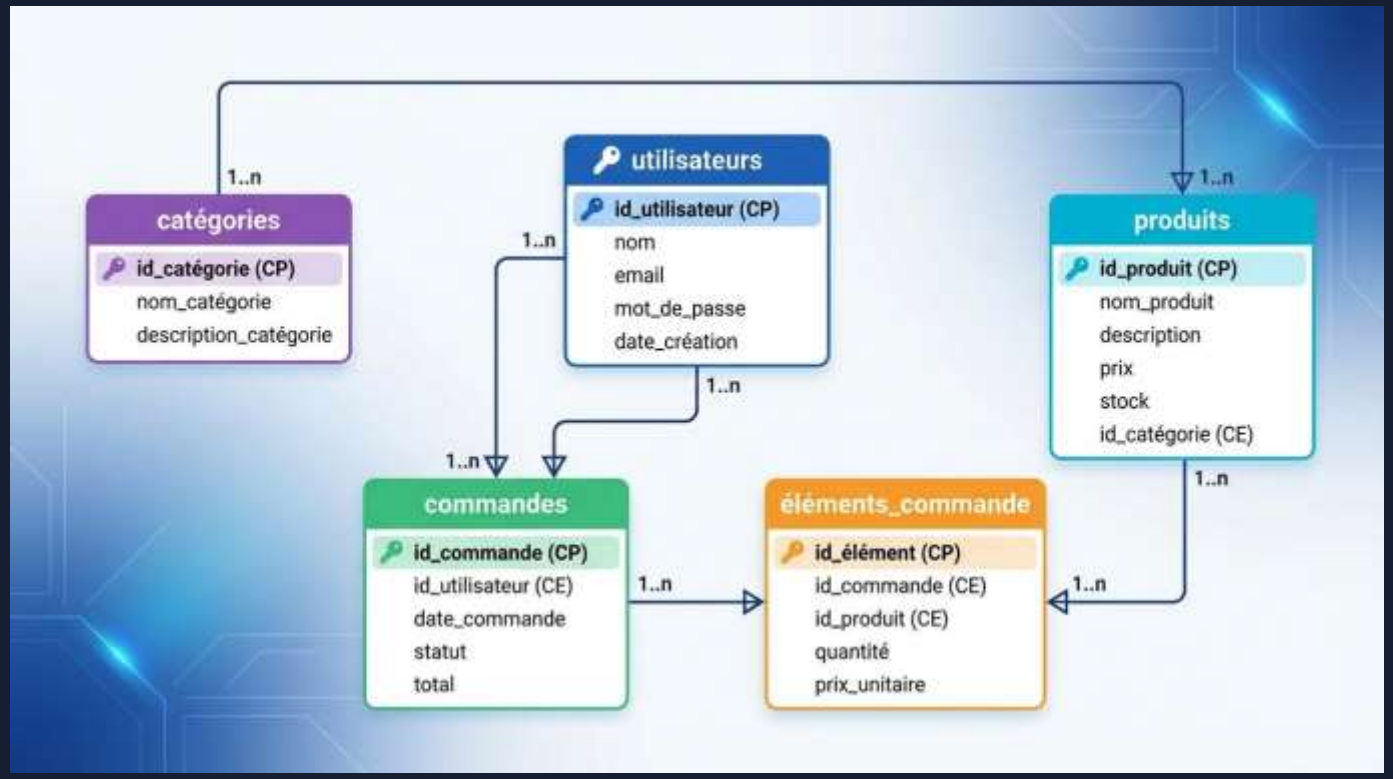
Interface avec la base de données via Spring Data JPA. Exécute les requêtes SQL/JPQL et mappe les résultats vers les entités Java.

Security Layer (Cross-Cutting)

Couche transverse gérant l'authentification (JWT), l'autorisation (Rôles/Permissions), et

Base de données PostgreSQL

Modèle Relationnel



5 Tables Principales Relations Normalisées

Légende & Structure

PK — Primary Key (Clé Primaire)
Identifiant unique (BIGSERIAL)

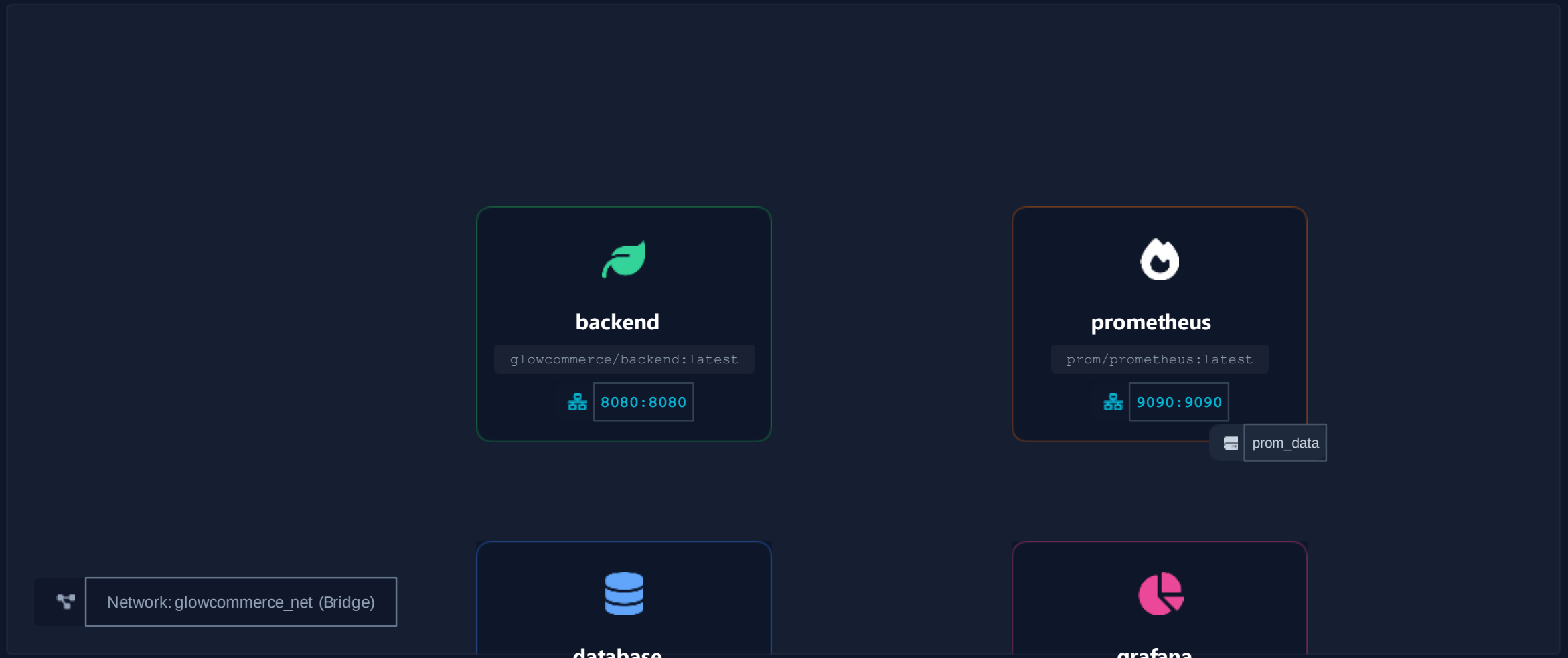
FK — Foreign Key (Clé Étrangère)
Lien référentiel entre tables

Relations (1..n)
Un utilisateur → Plusieurs commandes

IDX — Index & Contraintes
UNIQUE (email), Index performance

users	orders	products
Table Mère	Transactionnel	Catalogue

Architecture Docker



Application Spring Boot

PostgreSQL 16

Monitoring Stack

Prérequis système & outils

Environnement

Infrastructure Container

- ✓Docker Desktop — Version récente
- ✓WSL 2 — Windows
- 8 GB RAM Minimum
- Docker Engine + Compose

Stack Java & Build

- Java 21 JDK — LTS
- Maven 3.9+
- Git 2.40+
- Git Bash / Terminal

IDE & Outils Recommandés

- IntelliJ IDEA / VS Code
- Postman / Insomnia — API
- DBeaver / pgAdmin — SQL

Commandes de vérification

```
docker --version
docker-compose --version
java -version
mvn -version
git --version
```

Cloner le projet

Git Required

user@glowcommerce:~/dev

```
# 1. Cloner le repository
➜ ~ git clone https://github.com/glowcommerce/backend.git
Cloning into 'backend'... remote: Enumerating objects: 452, done. Receiving objects: 100% (452/452), 2.45 MiB | 1.20 MiB/s, done.

# 2. Se placer sur la branche de formation
➜ ~ cd backend && git checkout formation/day1
Switched to branch 'formation/day1'

# 3. Explorer la structure
➜ backend tree -L 2
➜ backend
```

Structure du projet

```
.
├── backend/ # Code source Spring Boot
│   ├── src/
│   ├── pom.xml # Dépendances Maven
│   └── Dockerfile
├── config/ # Fichiers de configuration
│   ├── postgresql.conf
│   ├── pg_hba.conf # Sécurité réseau BDD
│   └── prometheus.yml
├── docs/
│   ├── slides/
│   └── demos/
├── scripts/ # Utilitaires bash/SQL
└── docker-compose.yml # Orchestration services
```

Configuration initiale Fichier .env

Environment Secrets

backend/.env

```
# PostgreSQL Configuration
POSTGRES_PASSWORD=InitialDevPassword123!
DB_APP_PASSWORD=AppPassword123!

# JWT Security (256-bit Hex Key)
JWT_SECRET=404E635266556A586E3272357538782F413F4428472B4B...

# Spring Boot Profiles
SPRING_PROFILES_ACTIVE=dev
```

Sécurité Critique

Ce fichier contient des secrets sensibles. Il ne doit **JAMAIS** être commité dans Git.
Assurez-vous que .env est bien listé dans votre fichier .gitignore.
En production, utilisez un gestionnaire de secrets (Vault, AWS Secrets Manager).

Démarrage de l'infrastructure

Docker Required

user@glowcommerce:~/backend

```
# 1. Construire les images Docker
➜ backend docker-compose build

Building backend... [+] Building 12.5s (8/8) FINISHED => [internal] load build definition from Dockerfile

# 2. Démarrer les services en arrière-plan
➜ backend docker-compose up -d -d

Creating network "glowcommerce_default" with the default driver Creating glowcommerce-db ... done Creating glowcommerce-backend ... done

# 3. Vérifier que tout tourne
➜ backend docker-compose ps

NAME STATUS PORTS glowcommerce-db Up 0.0.0.0:5432->5432/tcp glowcommerce-api Up 0.0.0.0:8080->8080/tcp

# 4. Suivre les logs de l'application
➜ backend docker-compose logs -f backend
```

Services Démarrés

PostgreSQL 16
Port: 5432

Spring Boot Backend
Port: 8080

Prometheus
Port: 9090

Grafana
Port: 3000

Temps de démarrage estimé :
Environ 2 à 3 minutes lors du premier lancement (téléchargement des images & build Maven).

Vérification de l'installation

Health Checks & Validation des services

Tout doit être au vert

PostgreSQL

READY

```
psql -U postgres -d glowcommerce
```

→ Commande \dt doit lister les 5 tables.
SELECT COUNT(*) FROM products; doit retourner un chiffre.

Backend API

UP

```
curl localhost:8080/actuator/health
```

→ Réponse JSON attendue :
{"status":"UP"}

Prometheus

ACTIVE

http://localhost:9090/targets

→ Vérifier que la cible **backend** est en état (v UP
Scraping toutes les 15s.

Grafana

ONLINE

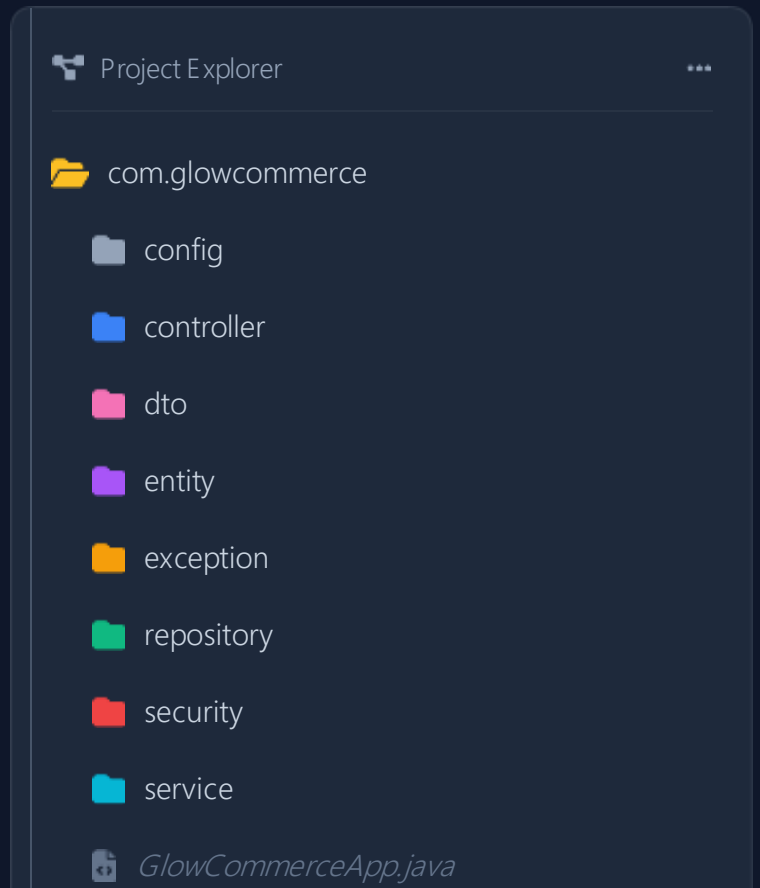
http://localhost:3000

→ Accès à la page de login.
Identifiants par défaut : **admin / admin**.

Structure du Code Backend

src/main/java/com/glowcommerce

Project Explorer



config

Configuration globale de l'application (Beans, CORS, Swagger, Database).

controller

Points d'entrée API REST. Gestion des requêtes HTTP et validation.

`@RestController` `@RequestMapping`

service

Logique métier, règles de gestion et transactions.

`@Service` `@Transactional`

repository

Interface d'accès aux données via Spring Data JPA.

`JpaRepository` `SQL`

entity

Modèles de données mappés sur les tables de la base.

`@Entity` `@Table`

dto

Objets de transfert de données pour découpler API et BDD.

`Record` `Lombok`

security

Filtres JWT, gestion des utilisateurs et permissions.

`JwtFilter` `SecurityConfig`

exception

Gestion centralisée des erreurs et exceptions personnalisées.

`@ControllerAdvice`

Exemple ProductController

Spring Boot 3.2

backend/.../controller/ProductController.java

```java
@RestController
@RequestMapping("/api/products")
@RequiredArgsConstructor
public class ProductController {
    private final ProductService service;

    @GetMapping
    public Page<ProductDTO> getAll(Pageable page) {
        return service.getAllProducts(page);
    }

    @PostMapping
    @PreAuthorize("hasRole('ADMIN')")
    public ProductDTO create(@Valid @RequestBody ProductDTO dto) {
        return service.createProduct(dto);
    }

    @DeleteMapping("/{id}")
    @PreAuthorize("hasRole('ADMIN')")
    public void delete(@PathVariable Long id) {
        service.deleteProduct(id);
    }
}
```

Configuration application.properties

Spring Boot Config

src/main/resources/application.properties

```
#
# ==============================================
# DATABASE CONFIGURATION (PostgreSQL)
#
# ==============================================
spring.datasource.url=jdbc:postgresql://database:5432/glowcommerce
spring.datasource.username=${DB_USERNAME:glowcommerce_app}
spring.datasource.password=${DB_PASSWORD}
spring.datasource.driver-class-name=org.postgresql.Driver

#
# ==============================================
# JPA / HIBERNATE
#
# ==============================================
spring.jpa.hibernate.ddl-auto=validate # Prod safe: validate schema only
spring.jpa.show-sql=false
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect

#
# ==============================================
# HIKARI CONNECTION POOL
#
# ==============================================
spring.datasource.hikari.maximum-pool-size=10
spring.datasource.hikari.minimum-idle=5
spring.datasource.hikari.connection-timeout=30000
spring.datasource.hikari.idle-timeout=600000

#
# ==============================================
```

Sécurité BDD : Enjeux & Menaces

Données Sensibles GlowCommerce

- 100K+ Utilisateurs
- Hash Mots de passe
- Emails & Tél
- Adresses Livraison
- Historique Commandes

Menaces Principales

- Injection SQL
- Accès Non Autorisés
- Vol de Données (Exfiltration)
- Destruction Malveillante
- Déni de Service (DoS)
- Man-in-the-Middle (MITM)

- Perte de Confiance
- Amendes RGPD
- Coûts Légaux & Forensic
- Réputation Dégradée

OWASP Top 10 Database Security Risks

Menaces Critiques

Injection SQL

Insertion de commandes malveillantes via les entrées utilisateur non validées. Permet de voler, modifier ou détruire toute la base de données.

Sensitive Data Exposure

Absence de chiffrement (au repos/transit) pour les PII, données bancaires ou mots de passe. Cible principale des ransomwares.

Broken Authentication

Gestion incorrecte des sessions, mots de passe faibles, ou absence de limitation des tentatives de connexion (brute force).

Broken Access Control

Non-respect du principe du moindre privilège. Un utilisateur standard accédant à des données administratives ou d'autres utilisateurs.

Security Misconfiguration

Configurations par défaut conservées (comptes, mots de passe), ports ouverts inutilement, messages d'erreur trop détaillés.

Configuration par défaut dangereuse

PostgreSQL "Out-of-the-Box"

SUPERUSER Actif

Utilisateur postgres avec tous les droits activé par défaut. Cible privilégiée des attaquants.

Mot de Passe Faible

Souvent vide ou très simple lors de l'installation initiale. Aucune politique de complexité appliquée.

Auth "Trust"

Permet la connexion sans aucun mot de passe si l'on vient d'une IP autorisée. Extrêmement dangereux.

Port 5432 Exposé

Port par défaut connu de tous les scanners. Souvent ouvert sur 0.0.0.0 (toutes les interfaces).

Pas de SSL/TLS

Communications en clair par défaut. Les données transitent sans chiffrement sur le réseau.

Zero Audit & Limites

Aucune trace des actions malveillantes. Pas de limite de connexions (risque DoS).

pg_hba.conf (DANGER)

```
# TYPE DATABASE USER ADDRESS METHOD
host all all 0.0.0.0/0 trust

# 0.0.0.0/0 = Tout Internet
# trust = Pas de mot de passe !
```

Temps moyen pour être piraté avec cette configuration :

~ 5 MINUTES

(Scan automatisé + Brute force)

Principe du Moindre Privilège

Sécurité BDD

Analogie : Entreprise Physique

Chaque employé ne doit avoir QUE les clés strictement nécessaires à son travail.

MAUVAIS (Risqué)

- ✕ Donner les clés MASTER à tous les employés
- ✕ Tout le monde accède au coffre-fort
- ✕ Pas de traçabilité des entrées/sorties
- ⚠ Si une clé est perdue, TOUT est compromis

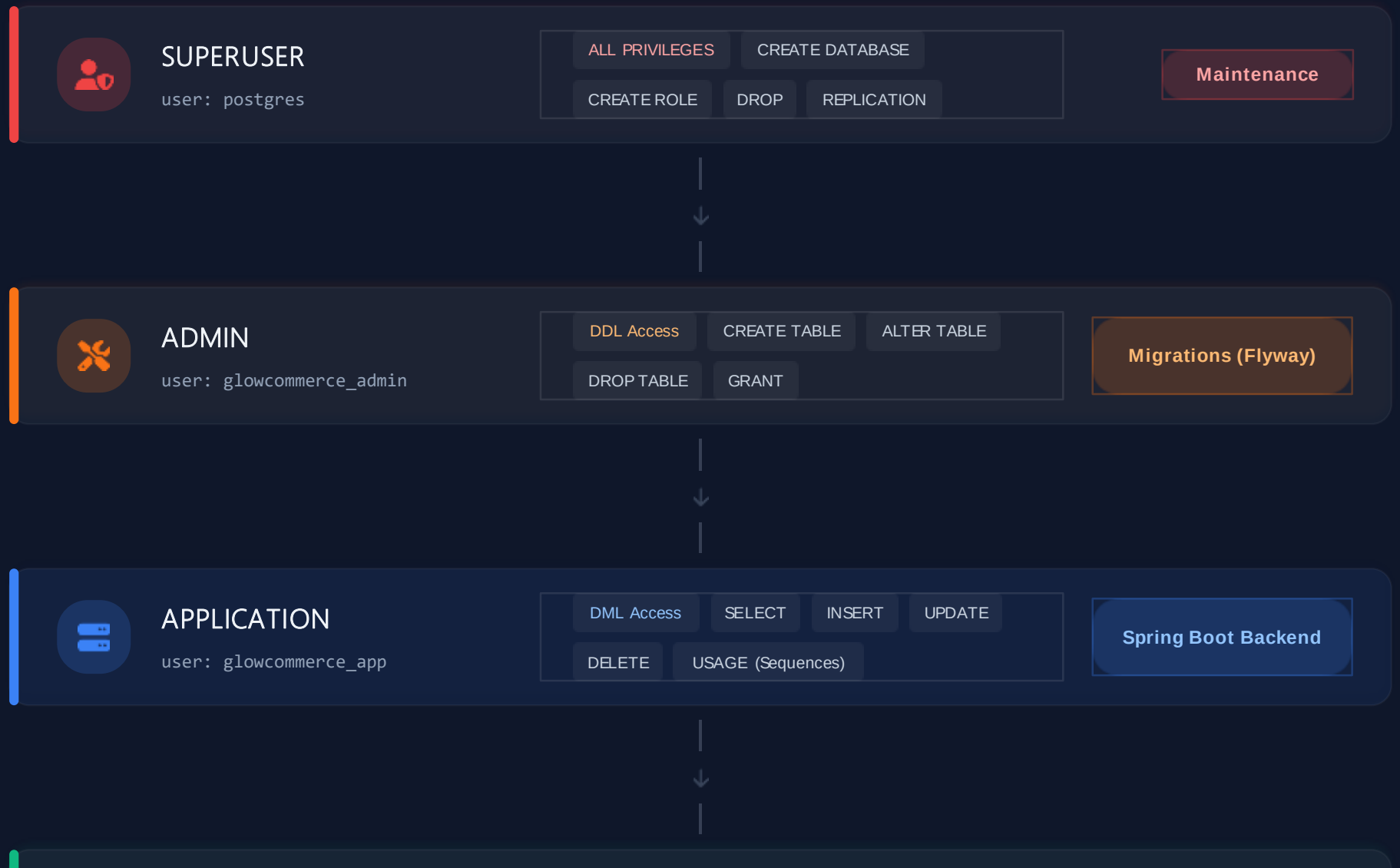
BON (Sécurisé)

- ✓ Clé bureau pour les employés standard
- ✓ Clé salle serveur uniquement pour les IT
- ✓ Clé coffre réservée au directeur
- ✓ Badge personnel avec logs d'accès

Séparation des rôles | Contrôle d'accès | PostgreSQL Security

Architecture utilisateurs PostgreSQL

Principe du Moindre Privilège



Mots de passe sécurisés

Best Practices

À éviter absolument

- admin123 (Trop commun)
- password! (Dictionnaire)
- 123456 (Suite logique)

Caractéristiques d'un mot de passe fort

- **Longueur :** ≥ 16 caractères (32 recommandés)
- **Complexité :** A-Z, a-z, 0-9, symboles (#@!$%)
- **Unicité :** Aléatoire et jamais réutilisé

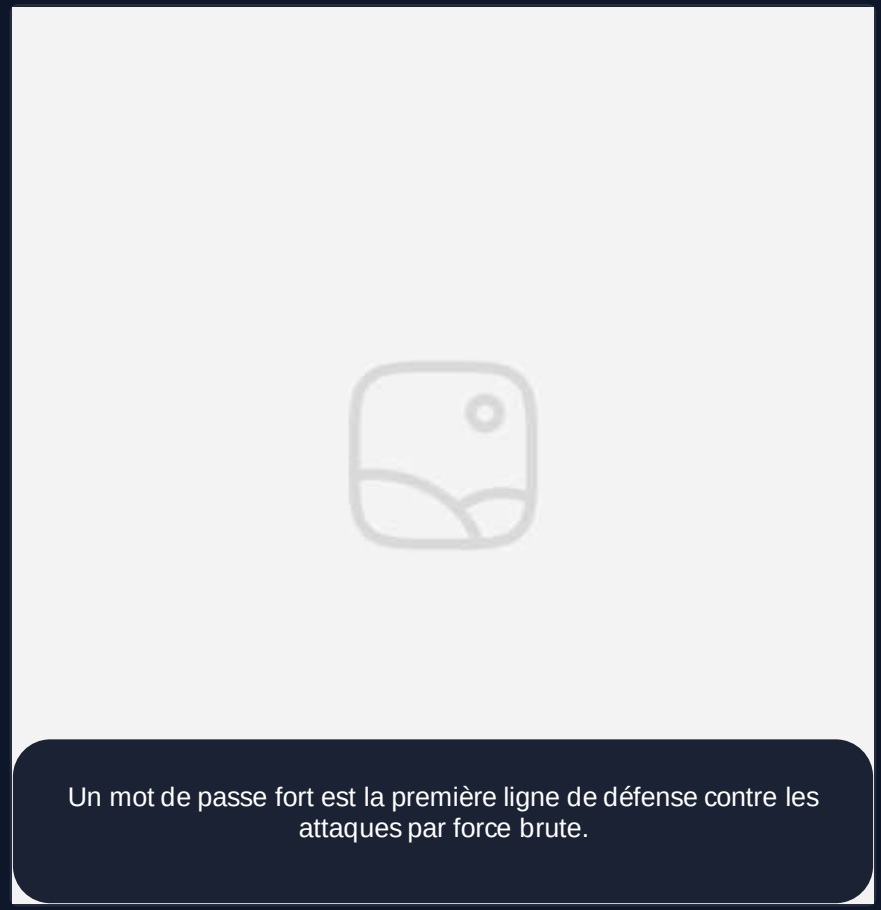
Génération sécurisée

Via OpenSSL :

```
openssl rand -base64 32
```

Via pwgen :

```
pwgen -s 32 1
```



Un mot de passe fort est la première ligne de défense contre les attaques par force brute.

Création des Utilisateurs Commandes SQL

PostgreSQL 16

init_users.sql

```sql
-- 1. Utilisateur Application (Spring Boot)
CREATE USER glowcommerce_app WITH
PASSWORD 'xK8mN3pQ7rL2vC9wB4yT6aU1E5oI0sD8' -- 32 chars aléatoires
NOSUPERUSER -- Pas de privilèges root
NOCREATEDB -- Ne peut pas créer de BDD
NOCREATEROLE -- Ne peut pas créer d'utilisateurs
NOINHERIT -- Pas d'héritage implicite
LOGIN -- Autoriser la connexion
CONNECTION LIMIT 20; -- Limiter les connexions simultanées

COMMENT ON ROLE glowcommerce_app IS 'App User - CRUD limité';

-- 2. Utilisateur Read-Only (Analytics/BI)
CREATE USER glowcommerce_readonly WITH
PASSWORD 'yL9nP4rQ8sC2vB7wE3aT5oI1uD6fG0hJ'
NOSUPERUSER NOCREATEDB NOCREATEROLE NOINHERIT
LOGIN
CONNECTION LIMIT 5;

-- 3. Sécuriser l'utilisateur postgres (Superuser)
ALTER USER postgres WITH PASSWORD 'zA1bC2dE3fG4hI5jK6lM7nO8pQ9rS0tU1vW2xY3z';
```

Pour vérifier la création des utilisateurs et leurs attributs, utilisez la méta-commande psql :

Attribution des privilèges Commandes GRANT

PostgreSQL Security

init_permissions.sql

```sql
-- 1. Connexion & Usage de base
GRANT CONNECT ON DATABASE glowcommerce TO glowcommerce_app;
GRANT USAGE ON SCHEMA public TO glowcommerce_app;

-- 2. Droits CRUD sur les tables existantes
GRANT SELECT, INSERT, UPDATE, DELETE
ON ALL TABLES IN SCHEMA public TO glowcommerce_app;

-- 3. Gestion des séquences (IDs) & Futures tables
GRANT USAGE, SELECT ON ALL SEQUENCES IN SCHEMA public TO glowcommerce_app;
ALTER DEFAULT PRIVILEGES IN SCHEMA public
GRANT SELECT, INSERT, UPDATE, DELETE ON TABLES TO glowcommerce_app;

-- 4. Utilisateur Read-Only (BI / Analytics)
GRANT SELECT ON ALL TABLES IN SCHEMA public TO glowcommerce_readonly;
```

Limitations Intentionnelles de Sécurité

Notez ce qui est **ABSENT** volontairement pour l'application :
✕ DROP (Suppression de tables), ✕ TRUNCATE (Vider les tables), ✕ CREATE (Création d'objets).
Seul le rôle ADMIN (utilisé par Flyway) possède ces droits.

Tests validation des privilèges

Vérification du principe de moindre privilège

User Context
glowcommerce_app

root@glowcommerce-db:~$ psql -U glowcommerce_app -d glowcommerce --test-suite

SYSTEM SECURE

Opération	Commande SQL (Exemple)	Résultat Technique	Statut
Lecture (SELECT)	SELECT * FROM products LIMIT 5;	5 rows returned (0.02s)	Autorisé
Écriture (INSERT)	INSERT INTO orders (id...) VALUES...;	INSERT 0 1	Autorisé
Modification (UPDATE)	UPDATE users SET last_login = NOW();	UPDATE 1	Autorisé
Destruction (DROP)	DROP TABLE users;	ERROR: must be owner of table users	Refusé
Nettoyage (TRUNCATE)	TRUNCATE TABLE orders;	ERROR: permission denied for table orders	Refusé
Admin (CREATE DB)	CREATE DATABASE hack_db;	ERROR: permission denied to create database	Refusé

Conclusion du test : L'utilisateur applicatif est correctement confiné. Il peut manipuler les données (CRUD) mais ne peut **jamais** altérer la structure ou détruire massivement les données.

Tests validation des privilèges (Read-Only)

Vérification de l'accès en lecture seule stricte

User Context
glowcommerce_readonly

```
root@glowcommerce-db:~$ psql -U glowcommerce_readonly -d glowcommerce --verify
```

• SYSTEM SECURE

Opération	Commande SQL (Test)	Résultat Technique	Statut
Lecture (SELECT)	`SELECT count(*) FROM users;`	100452 rows returned	Autorisé
Écriture (INSERT)	`INSERT INTO products VALUES (...);`	ERROR: permission denied for table products	Refusé
Modification (UPDATE)	`UPDATE orders SET status = 'PAID';`	ERROR: permission denied for table orders	Refusé
Suppression (DELETE)	`DELETE FROM users WHERE id=1;`	ERROR: permission denied for table users	Refusé

Conclusion du test : Le profil Read-Only est parfaitement isolé. Toute tentative de modification de données (INSERT, UPDATE, DELETE) est systématiquement bloquée par le moteur de base de données.

Révoquer privilèges PUBLIC Nettoyage des droits par défaut

PostgreSQL Hardening

psql console - clean_public.sql

```sql
-- 1. Révoquer le droit de créer des objets dans le schéma public
REVOKE CREATE ON SCHEMA public FROM PUBLIC;

-- 2. Révoquer tous les droits sur les tables existantes
REVOKE ALL ON ALL TABLES IN SCHEMA public FROM PUBLIC;

-- 3. Révoquer les droits sur les séquences (auto-increment)
REVOKE ALL ON ALL SEQUENCES IN SCHEMA public FROM PUBLIC;

-- 4. Révoquer les droits sur les fonctions
REVOKE ALL ON ALL FUNCTIONS IN SCHEMA public FROM PUBLIC;
```

Vérification

```sql
SELECT count (*) FROM information_schema.role_table_grants
WHERE grantee ='PUBLIC' AND table_schema ='public' ;

-- Résultat attendu :
```

Audit des privilèges Requêtes d'inspection

SQL Audit

audit_queries.sql

```sql
-- 1. Lister les privilèges explicites sur les tables
SELECT grantee, table_name, privilege_type
FROM information_schema.role_table_grants
WHERE table_schema = 'public'
AND grantee IN ('glowcommerce_app', 'glowcommerce_readonly');

-- 2. Vérifier les attributs des rôles (Superuser, Create DB...)
SELECT rolname, rolsuper, rolcreaterole, rolconnlimit
FROM pg_roles WHERE rolname NOT LIKE 'pg_%';
```

terminal (psql)

```
glowcommerce=> \dp products
Access privileges
Schema | Name | Type | Access privileges
--------+----------+-------+-----------------------------------
 public | products | table | glowcommerce_app=arwd/postgres
 | | | glowcommerce_readonly=r/postgres
```

Légende \dp : **r** = SELECT (read) **a** = INSERT (append) **w** = UPDATE (write) **d** = DELETE

Authentification BCrypt - Principes

Sécurité BDD

Qu'est-ce que BCrypt ?

- KDF (Key Derivation Function) basée sur Blowfish
- Algorithme lent par conception (protège contre brute force)
- Génération automatique d'un sel (salt) unique pour chaque hash
- Facteur de coût (cost factor) ajustable pour suivre la puissance CPU

Pourquoi l'utiliser ?

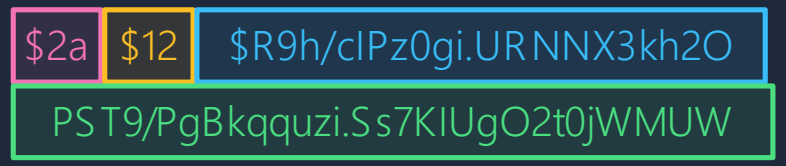
- Résistant aux attaques par dictionnaire et brute force
- Salt unique : deux mêmes mots de passe génèrent des hashes différents
- Adaptatif : on peut augmenter la sécurité avec le temps
- Éprouvé et standard de l'industrie depuis 1999

✕ À NE JAMAIS UTILISER

- **MD5** (Cassé depuis 2004 - Collisions)
- **SHA-1** (Cassé depuis 2017 - SHAttered)
- **SHA-256 simple** (Sans salt = vulnérable aux rainbow tables)
- **Mots de passe en clair** (Interdit par RGPD)

Anatomie d'un hash BCrypt

Sécurité BDD



Version de l'algorithme

$2a = Standard actuel

Facteur de coût (Cost)

$12 = 2^{12} = 4096 itérations

Salt (Sel)

22 caractères aléatoires uniques

Hash final

31 caractères générés

Avantage clé : Même mot de passe = Hashes différents (grâce au salt unique)

Flux de Hashing

Mot de passe
Input utilisateur

Génération Salt
Aléatoire cryptographique

Fonction BCrypt
BCrypt(pwd, salt, cost)

Hash Stocké
Base de données

Facteur de Coût BCrypt

Équilibre Sécurité vs Performance

Cost Factor	Itérations (2^cost)	Temps Hashage	Recommandation	
10	1 024	~50 ms	Trop Rapide	Vulnérable Brute Force
12	4 096	~200 - 300 ms	✓ Recommandé	Standard 2024-2026
14	16 384	~800 - 1000 ms	Haute Sécurité	Impact UX sensible
16	65 536	~3 - 4 sec	Trop Lent	Risque DoS / Timeout
18	262 144	~12 sec	Inutilisable	Web interactif

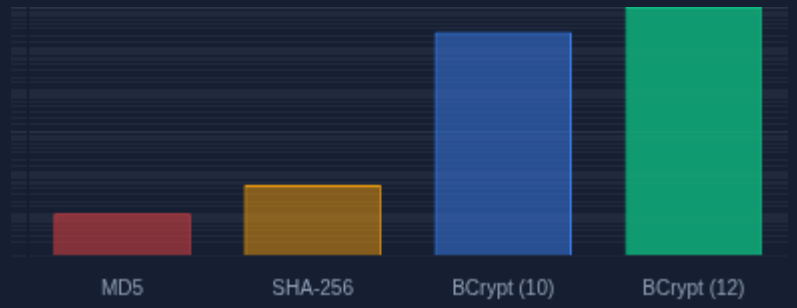
Le bon choix : Un hash doit prendre entre **200ms et 500ms** sur votre serveur de production. Cela rend les attaques par force brute impraticables (trop lentes) tout en restant imperceptible pour l'utilisateur légitime.

Évolution (Loi de Moore)

La puissance de calcul double tous les 18 mois. Pour maintenir le même niveau de sécurité (temps de hashage constant), il faut augmenter le coût.

- **2020** Cost 10-11
- **2024** **Cost 12 (Actuel)**
- **2028** Cost 13-14
- **2032** Cost 14-15

Impact sur Brute Force



Temps pour tester 1 milliard de mots de passe (GPU Cluster)

Implémentation Spring Security Encoder BCrypt

Spring Boot 3.2

config/SecurityConfig.java

```java
@Configuration
public class SecurityConfig {

@Bean
public PasswordEncoder passwordEncoder() {
// Facteur de coût 12 = 4096 itérations
// Temps moyen : ~200ms par hash
return new BCryptPasswordEncoder(12);
}
}
```

Performance & Sécurité

Le facteur 12 offre un excellent compromis. Il est assez lent pour empêcher les attaques par force brute (brute-force), mais assez rapide pour ne pas dégrader l'expérience utilisateur lors du login.

service/AuthService.java

```java
@Service
@RequiredArgsConstructor
public class AuthService {

// Injection du bean configuré à gauche
private final PasswordEncoder passwordEncoder;

public void register(RegisterRequest request) {

// Encodage avant sauvegarde en BDD
String hashedPassword =
passwordEncoder.encode(
request.getPassword()
);

// ... sauvegarde user ...
}
}
```

Vérification mot de passe Login sécurisé

Spring Security

service/AuthService.java

```java
@Service
public class AuthService {

    private final PasswordEncoder passwordEncoder;

    public LoginResponse login(LoginRequest request) {
        // 1. Récupérer l'utilisateur (via email)
        User user = userRepository.findByEmail(request.getEmail())
            .orElseThrow(() -> new BadCredentialsException("Invalid credentials"));

        // 2. Vérifier le mot de passe : Brut vs Hash BCrypt
        boolean isMatch = passwordEncoder.matches(
            request.getPassword(), // "password123"
            user.getPasswordHash() // "$2a$12$R9h..."
        );

        if (!isMatch) {
            throw new BadCredentialsException("Invalid credentials");
        }

        // 3. Générer le JWT...
        return jwtService.generateToken(user);
    }
```

Modèle de données Entité User JPA

Spring Data JPA

src/main/java/com/glowcommerce/entity/User.java

```java
@Entity
@Table(name = "users")
public class User {

    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    // Email unique pour l'identification
    @Column(unique = true, nullable = false)
    private String email;

    // Hash BCrypt : toujours 60 caractères (ex: $2a$12$...)
    @Column(name = "password_hash", nullable = false, length = 60)
    private String passwordHash;

    @Enumerated(EnumType.STRING)
    private Role role; // ADMIN, CUSTOMER

    @Column(name = "created_at", updatable = false)
    private LocalDateTime createdAt;

    @PrePersist
    protected void onCreate() { this.createdAt = LocalDateTime.now(); }
}
```

Validation des Données Mots de passe robustes

Bean Validation (JSR 380)

backend/src/main/java/com/glowcommerce/dto/RegisterRequest.java

```java
package com.glowcommerce.dto;

import jakarta.validation.constraints.*;
import lombok.Data;

@Data
public class RegisterRequest {

    @Email
    @NotBlank
    private String email;

    @NotBlank(message = "Mot de passe requis")
    @Size(min=12, max=100, message = "12-100 caractères")
    @Pattern(
      regexp = "^(?=.*[a-z])(?=.*[A-Z])(?=.*\\d)(?=.*[@$!%*?&]).+$",
      message = "Doit contenir maj, min, chiffre, spécial"
    )
    private String password;
}
```

Chiffrement des Données Sensibles

Conformité RGPD

Protection des données personnelles (PII) au repos en utilisant l'algorithme standard de l'industrie. Les données sont chiffrées avant d'être persistées.

Rotation des Clés (Key Rotation)

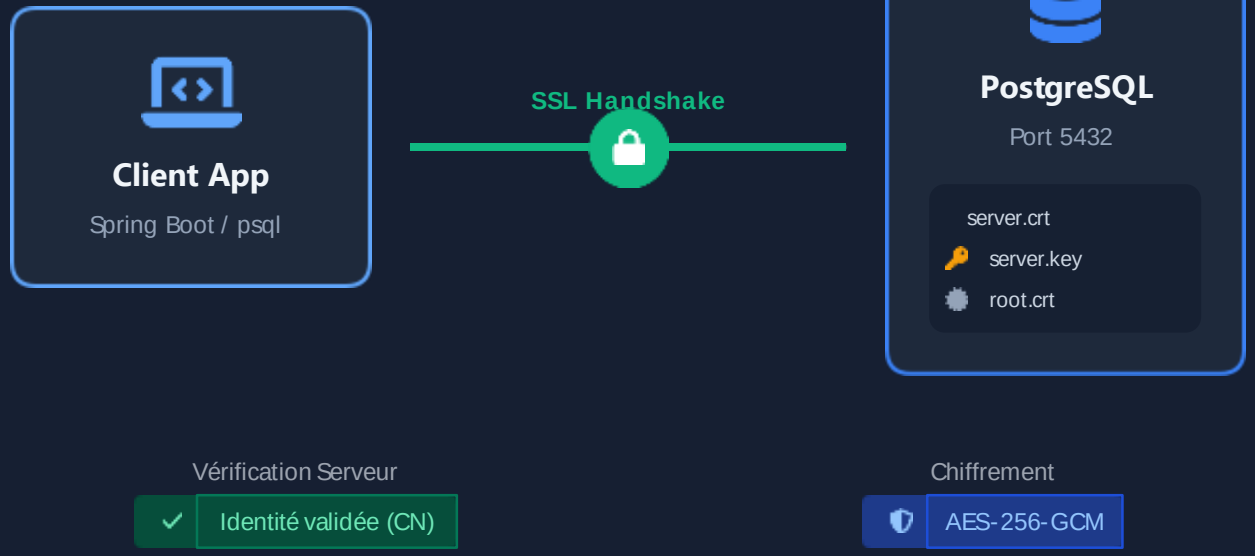
Les clés de chiffrement (DEK) sont renouvelées périodiquement. L'ancienne clé reste disponible pour le déchiffrement, la nouvelle est utilisée pour les nouvelles écritures.

Journalisation des Accès (Audit Logs)

Chaque opération de déchiffrement est loggée. On sait QUI a accédé aux données sensibles et QUAND.

Sécuriser les Connexions (SSL/TLS)

Chiffrement en Transit



postgresql.conf

```
ssl = on
ssl_cert_file = 'server.crt'
ssl_key_file = 'server.key'
# Requis: chmod 600 sur server.key
```

pg_hba.conf

```
# TYPE DATABASE USER ADDRESS METHOD
hostssl all all 0.0.0.0/0 scram-sha-256
# Force SSL pour toutes les connexions
```

Modes de vérification Client (JDBC/LibPQ)

- **require** : Chiffrement obligatoire, pas de vérif certificat.
- **verify-ca** : Vérifie que le certificat est signé par la CA.
- **verify-full** : Vérifie CA + Nom d'hôte (Recommandé).

Système d'Audit Logs

Traçabilité

Table audit_logs

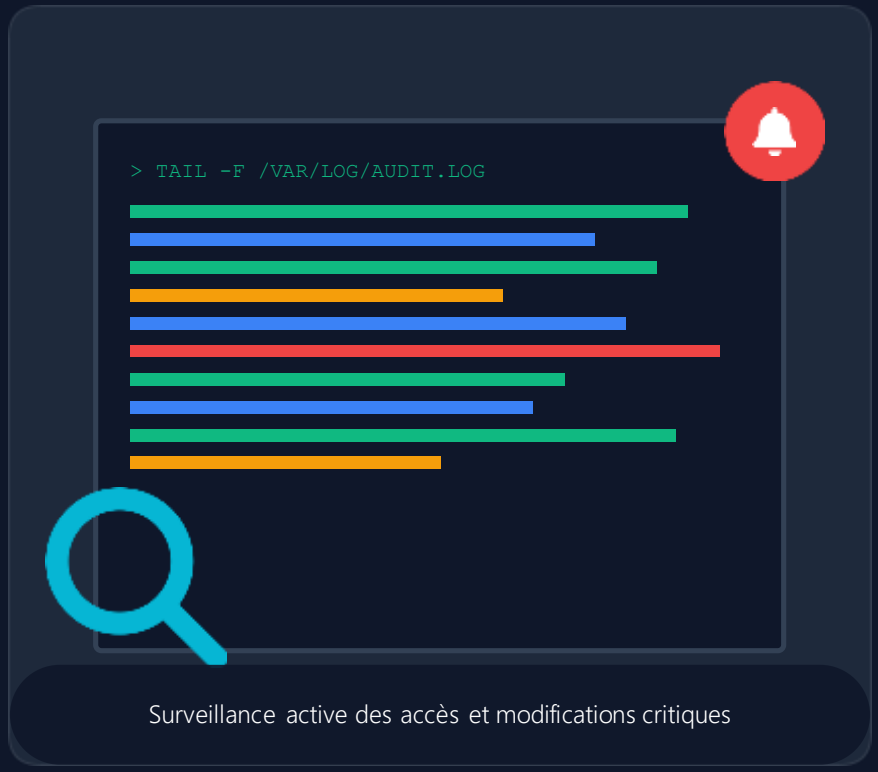
- **user_id** : Qui ?
- **action** : Quoi ? (INSERT/UPDEL)
- **table_name** : Où ?
- **timestamp** : Quand ?
- **old_data** : Valeur avant (JSON)
- **new_data** : Valeur après (JSON)
- **ip_address** : Origine ?

Triggers Automatiques

- **BEFORE/AFTER** sur opérations d'écriture
- Ciblage des tables sensibles : **users**, **orders**, **products**
- Garantie d'immuabilité (impossible de contourner via API)

Conformité & Rétention

- Rétention : **90 jours minimum** (RGPD/Légal)
- Corrélation avec logs applicatifs & SIEM



Surveillance active des accès et modifications critiques

Forensics

Alertes Sécurité

Conformité

DEMO 1-1 : Hardening PostgreSQL

LIVE SESSION

1 **Exécuter le script SQL**
Lancement de hardening_postgres.sql sur une base vierge.

2 **Création des Rôles**
Définition de glowcommerce_app et _readonly.

3 **Permissions & Limites**
Application GRANT/REVOKE et Connection Limits.

4 **Validation & Tests**
Vérification des accès et refus avec psql.

```
postgres@db:~$ psql -U postgres -d glowcommerce -f hardening.sql

CREATE ROLE glowcommerce_app... OK
CREATE ROLE glowcommerce_readonly... OK
REVOKE PUBLIC ON SCHEMA public... DONE
GRANT SELECT, INSERT, UPDATE... DONE
ALTER USER postgres WITH PASSWORD... SECURED

postgres@db:~$ psql -U glowcommerce_app -d glowcommerce
psql (16.1)
Type "help" for help.

glowcommerce=> SELECT count(*) FROM products;
count
-------
154
(1 row)

glowcommerce=> DROP TABLE products;
ERROR: permission denied for table products
glowcommerce=>
```

DEMO 1-2 : Système Audit

LIVE SESSION

1 **Création Table Audit**

Table audit_logs (user, action, old/new, IP).

2 **Fonction PL/pgSQL**

Implémentation de audit_changes().

3 **Déploiement Triggers**

Surveillance users, orders, products.

4 **Tests & Monitoring**

Vérification logs et export Grafana.

```
postgres@db:~$ psql -U glowcommerce_app -d glowcommerce
psql (16.1)

glowcommerce=> CREATE TABLE audit_logs (...);
CREATE TABLE

glowcommerce=> \i audit_triggers.sql
CREATE FUNCTION audit_changes... OK
CREATE TRIGGER trg_audit_users... OK
CREATE TRIGGER trg_audit_products... OK

glowcommerce=> INSERT INTO products (name) VALUES ('Serum');

INSERT 0 1

glowcommerce=> SELECT action, user, table_name FROM audit_logs;

 action | user | table_name
--------+----------------+------------
 INSERT | glowcommerce_a | products
(1 row)

glowcommerce=>
```

TP 1-1

Hardening BDD PostgreSQL

Exercice Pratique

45 min

Temps estimé

Objectifs Pédagogiques

- Comprendre et appliquer le principe de moindre privilège.
- Savoir séparer les rôles (Admin, App, ReadOnly).
- Sécuriser l'accès aux données par défaut.

Instructions & Tâches

1. Créer les rôles **glowcommerce_app** et **glowcommerce_readonly** avec des mots de passe forts.
2. Révoquer tous les privilèges du rôle **PUBLIC** sur le schéma public.
3. Attribuer les privilèges spécifiques (GRANT) selon la matrice de sécurité définie.
4. Effectuer des tests de connexion (psql) et valider les interdictions (ex: DROP TABLE).

Livrables attendus — Script SQL (hardening.sql) — Rapport de validation

TP 1-2

Système d'Audit

Exercice Pratique

45 min

Temps estimé

Objectifs Pédagogiques

- Implémenter une traçabilité complète des données.
- Surveiller les tables sensibles (Users, Orders, Products).
- Maîtriser les Triggers PostgreSQL (PL/pgSQL).

Instructions & Tâches

1. Créer la table audit_logs (user, action, timestamp, old/new data).
2. Implémenter la fonction trigger générique audit_changes().
3. Activer les triggers AFTER INSERT OR UPDATE OR DELETE sur 3 tables.
4. Simuler des modifications et valider les captures dans les logs.

Livrables & Bonus

Script SQL (audit.sql) **Rapport de validation**

★ BONUS Créer un tableau de bord Grafana (Métriques : Actions/min)

Récapitulatif Jour 1

Compétences acquises & Validation

Progression Formation

Hardening PostgreSQL

Séparation des rôles (App/ReadOnly), principe du moindre privilège, révocation des droits PUBLIC.

Authentification Sécurisée

Implémentation de BCrypt (cost=12) et protection contre les attaques par force brute.

Chiffrement & Confidentialité

Données sensibles chiffrées au repos (AES-256) et activation SSL/TLS pour le transit.

Audit & Conformité RGPD

Mise en place de logs d'audit (Triggers) et traçabilité des accès aux données sensibles.

Prochaines Étapes

01 DevSecOps & CI/CD

Intégration de tests de sécurité automatisés dans le pipeline GitHub Actions.

02 Monitoring Avancé

Création de dashboards Grafana pour la surveillance sécurité en temps réel.

03 Gestion d'Incidents

Simulation d'attaques et procédures de réponse (Incident Response).