

# API CINEMA

Express REST MVC • PostgreSQL (Prisma) •  
MongoDB (Mongoose)

Sécurisation avant mise en production

Par Mathis BOHEC & Marie COURTOIS

# Sommaire

I-	<b>Besoin &amp; cas d'usage</b>	
II-	<b>Architecture technique</b>	
III-	<b>Modèles de données</b>	
IV-	<b>Sécurisation (mesures + preuves)</b>	
V-	<b>OWASP (mapping)</b>	
VI-		<b>Outils (SONAR / zap / npm audit)</b>
VII-		<b>Démo</b>
VIII-		<b>Bilan &amp; amélioration</b>

# I- Besoin & cas d'usage

## Expression de besoin

- API pour gérer un catalogue de films
- Consommée par un client web/mobile
- Fonctions clés : consulter, rechercher, s'inscrire, noter

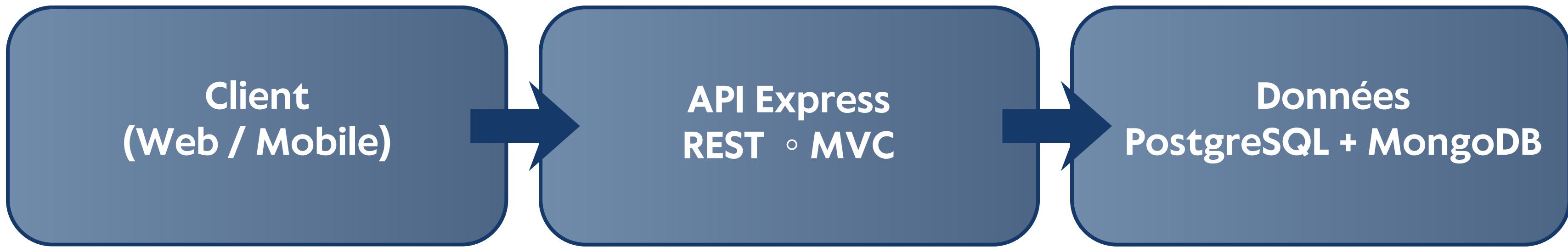
## Rôles

- Visiteur : consulter & rechercher
- Utilisateur : noter + profil
- Admin : CRUD films + gestion users

## Scénarios (exemple)

- Lister/consulter un film (public)
- Filtrer par titre/genre/réalisateur
- Login token JWT
- Noter un film (auth)
- Admin gère le catalogue

# II- Architecture technique ➔ 3-tiers



## Pourquoi SQL + NoSQL ?

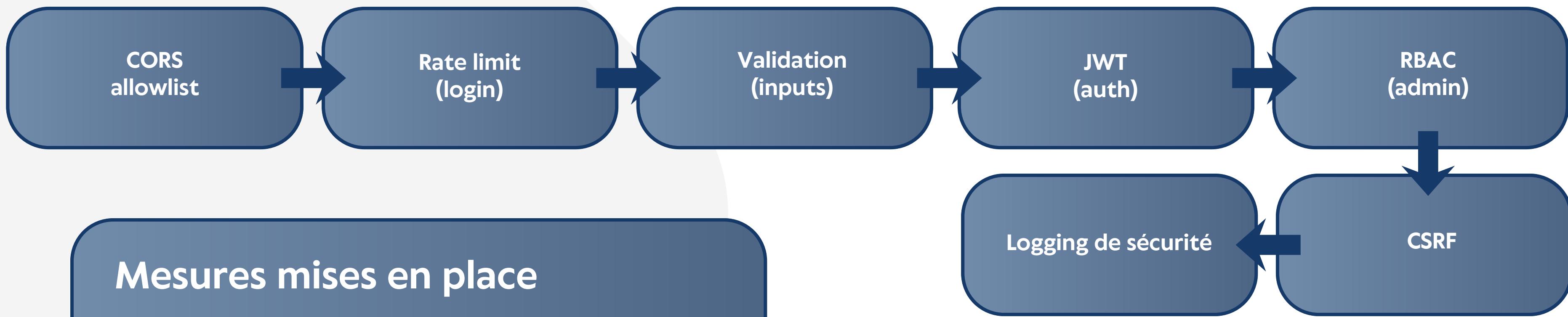
- PostgreSQL : données structurées + relations (films, acteurs, réalisateurs)
- MongoDB : logs/historiques & documents flexibles
- Chacun utilisé là où il est le plus efficace

## Stack sécurité

- CORS allowlist • Helmet • Rate limiting
- JWT (access + refresh) • bcrypt
- Validation des entrées • erreurs centralisées
- CSRF token (1h, usage unique)
- Logging sécurité (accès sensibles / auth)



# Sécurisation



## Mesures mises en place

- bcrypt : hash des mots de passe
- JWT : access token (2h) + refresh token
- Helmet : headers HTTP de sécurité
- Erreurs centralisées : pas de fuite d'infos
- Refresh token : rotation + max 5 sessions actives
- CSRF : token 1h + 403 si absent/invalide/expiré + usage unique
- Logging : traces des actions sensibles (userId/role/ip/path)

## Stack sécurité

- Brute force & spam (rate limit)
- Injection (validation + ORM)
- Accès non autorisés (RBAC)
- Misconfiguration (CORS/headers)

# Sécurisation - Extraits de code

## Bcrypt

```
● ● ●  
85 const hashedPassword = await bcrypt.hash(password, 10);  
86  
87 const newUser = await User.create({  
88   username: sanitizedUsername,  
89   email: sanitizedEmail,  
90   password: hashedPassword  
91 });  
92  
93 const userResponse = {  
94   id: newUser._id,  
95   username: newUser.username,  
96   email: newUser.email,  
97   role: newUser.role  
98 };
```

## JWT

```
● ● ●  
147 const token = jwt.sign(  
148   { userId: user._id, role: user.role },  
149   process.env.JWT_SECRET,  
150   { expiresIn: "2h" }  
151 );  
152  
153 res.json({  
154   token,  
155   user: {  
156     id: user._id,  
157     username: user.username,  
158     email: user.email,  
159   },  
160 });  
161
```

# Sécurisation - Extraits de code

## Refresh Token

```
● ○ ●
142 const refreshToken = crypto.randomBytes(64).toString('hex');
143 // Limiter à 5 refresh tokens actifs par utilisateur (rotation)
144 const userWithTokens = await User.findById(user._id).select('+refreshTokens');
145 const refreshTokens = userWithTokens.refreshTokens || [];
146 if (refreshTokens.length >= 5) {
147   refreshTokens.shift();
148 }
149
150 refreshTokens.push(refreshToken);
151
152 await User.findByIdAndUpdate(user._id, { refreshTokens });
153
154 securityLogger.logLoginAttempt(req, true);
155
156 res.json({
157   accessToken,
158   refreshToken,
159   user: {
160     id: user._id,
161     username: user.username,
162     email: user.email,
163   },
164 });
```

## Rate limit

```
● ○ ●
20 app.use(rateLimit({ windowMs: 60_000, max: 100 }));
21 app.use("/auth/login", rateLimit({ windowMs: 15*60_000, max: 10 }));
```

# Sécurisation - Extraits de code

## Validate

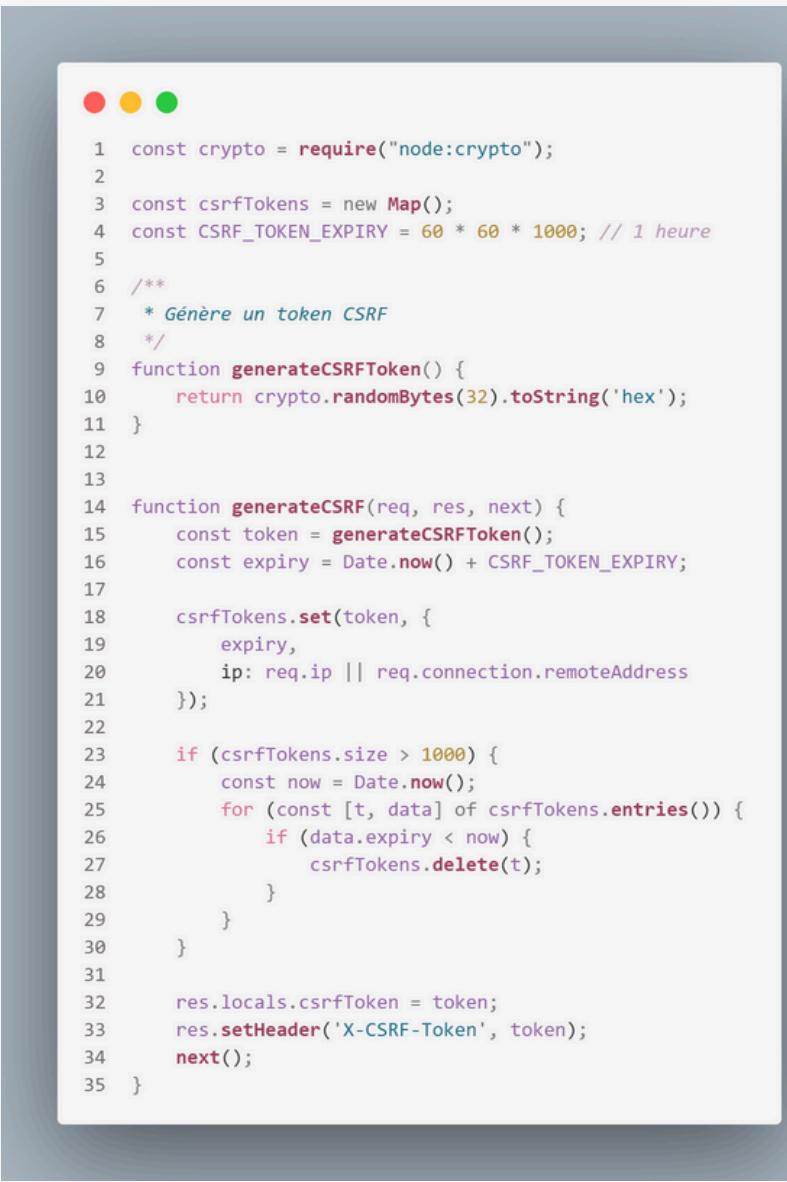
```
● ● ●  
1 const { validationResult } = require("express-validator");  
2  
3 function validate(req, res, next) {  
4   const errors = validationResult(req);  
5   if (!errors.isEmpty()) {  
6     return res.status(422).json({ errors: errors.array() });  
7   }  
8   next();  
9 }  
10
```

## CORS

```
● ● ●  
34 app.use(cors({  
35   origin: function (origin, callback) {  
36     // Autoriser les requêtes sans origine (mobile apps, Postman, etc.) en développement  
37     if (!origin && process.env.NODE_ENV !== 'production') {  
38       return callback(null, true);  
39     }  
40     if (allowedOrigins.indexOf(origin) !== -1 || !origin) {  
41       callback(null, true);  
42     } else {  
43       callback(new Error('Non autorisé par CORS'));  
44     }  
45   },  
46   credentials: true,  
47   allowedHeaders: ["Authorization", "Content-Type", "X-Requested-With"],  
48   methods: ["GET", "POST", "PUT", "DELETE", "PATCH", "OPTIONS"]  
49 }));
```

# Sécurisation - Extraits de code

## CSRF



```
1 const crypto = require("node:crypto");
2
3 const csrfTokens = new Map();
4 const CSRF_TOKEN_EXPIRY = 60 * 60 * 1000; // 1 heure
5
6 /**
7  * Génère un token CSRF
8  */
9 function generateCSRFToken() {
10   return crypto.randomBytes(32).toString('hex');
11 }
12
13
14 function generateCSRF(req, res, next) {
15   const token = generateCSRFToken();
16   const expiry = Date.now() + CSRF_TOKEN_EXPIRY;
17
18   csrfTokens.set(token, {
19     expiry,
20     ip: req.ip || req.connection.remoteAddress
21 });
22
23 if (csrfTokens.size > 1000) {
24   const now = Date.now();
25   for (const [t, data] of csrfTokens.entries()) {
26     if (data.expiry < now) {
27       csrfTokens.delete(t);
28     }
29   }
30 }
31
32 res.locals.csrfToken = token;
33 res.setHeader('X-CSRF-Token', token);
34 next();
35 }
```

## CSRF



```
37 function verifyCSRF(req, res, next) {
38
39 if (req.path.startsWith('/api/user/login') ||
40 req.path.startsWith('/api/user/register') ||
41 req.path.startsWith('/api/user/refresh')) {
42   return next();
43 }
44
45 const token = req.headers['x-csrf-token'] || req.body?.csrfToken;
46
47 if (!token) {
48
49   if (process.env.NODE_ENV === 'development') {
50     return next();
51   }
52   return res.status(403).json({ error: "Token CSRF manquant" });
53 }
54
55 const tokenData = csrfTokens.get(token);
56
57 if (!tokenData) {
58   return res.status(403).json({ error: "Token CSRF invalide" });
59 }
60
61 if (tokenData.expiry < Date.now()) {
62   csrfTokens.delete(token);
63   return res.status(403).json({ error: "Token CSRF expiré" });
64 }
65
66 csrfTokens.delete(token);
67
68
69 next();
70 }
```

Limite : stockage en mémoire → en prod multi-instance, store partagé (Redis)

# Sécurisation - Extraits de code

## RBAC

```
● ● ●
24 logSensitiveAccess(req, resource, action) {
25   const logData = {
26     timestamp: new Date().toISOString(),
27     event: 'SENSITIVE_ACCESS',
28     userId: req.user?.userId || 'N/A',
29     role: req.user?.role || 'N/A',
30     resource,
31     action,
32     ip: req.ip || req.connection.remoteAddress,
33     userAgent: req.get('user-agent'),
34     method: req.method,
35     path: req.path
36   };
37
38   console.log(`[SECURITY] 🔒 Accès sensible: ${action} sur ${resource} par utilisateur ${logData.userId} (${logData.role})`);
39 },
```

# Sécurisation - Extraits de code

## Helmet (headers sécurisés)

	Headers	Preview	Response	Initiator	Timing	Cookies
Content-Security-Policy			default-src 'self';base-uri 'self';font-src 'self' https: data:;form-action 'self';frame-ancestors 'self';img-src 'self' data:;object-src 'none';script-src 'self';script-src-attr 'none';style-src 'self' https: 'unsafe-inline';upgrade-insecure-requests			
Content-Type			application/json; charset=utf-8			
Cross-Origin-Opener-Policy			same-origin			
Cross-Origin-Resource-Policy			same-origin			
Date			Wed, 17 Dec 2025 13:55:00 GMT			
Etag			W/"14-7437k4ydMXNsPABXZFR1dLo4GpM"			
Keep-Alive			timeout=5			
Origin-Agent-Cluster			?1			
Referrer-Policy			no-referrer			
Strict-Transport-Security			max-age=31536000; includeSubDomains			
Vary			Origin			
X-Content-Type-Options			nosniff			
X-Dns-Prefetch-Control			off			
X-Download-Options			noopen			
X-Frame-Options			SAMEORIGIN			
X-Permitted-Cross-Domain-Policies			none			
X-Ratelimit-Limit			100			
X-Ratelimit-Remaining			97			

# Sécurisation - Extrait de Sonar

The screenshot shows the SonarQube security review interface. On the left, a summary indicates 0.0% Security Hotspots Reviewed. Below this, there are three buttons: To review (highlighted), Fixed, and Safe. A filter icon shows 1 result. The main area displays 1 Security Hotspot to review, with a Review priority of Medium. A specific issue is highlighted: Denial of Service (DoS). The description states: "Make sure the regex used here, which is vulnerable to super-linear runtime due to backtracking, cannot lead to denial of service." The code snippet shown is:

```
const mongoose = require("mongoose");
const User = require("../models/User.js");

// Fonctions utilitaires de validation et sanitisation
function isValidEmail(email) {
    const emailRegex = /^[^@\s]+@[^\s@]+\.[^\s@]+\$/;
    return emailRegex.test(email);
}

function sanitizeString(str) {
    if (typeof str !== "string") return null;
```

A callout box points to the regex pattern `/[^@\s]+@[^\s@]+\.[^\s@]+\$/` with the text: "Make sure the regex used here, which is vulnerable to super-linear runtime due to backtracking, cannot lead to denial of service."

Hotspot : risque ReDoS (backtracking regex) → fix : limitation de longueur avant regex

# OWASP

## A01 Broken Access Control

- Mesures : JWT + RBAC + logs accès sensibles
- Preuve : 403 sur /admin + logging SENSITIVE\_ACCESS

## A07 Identification & Authentication Failures

- Mesures : JWT exp + rate limit login
- Preuve : expiresIn + 10/15min

## A05 Misconfiguration / A02 Crypto

- A05 : CORS allowlist + Helmet
- A02 : bcrypt + HTTPS local

# Outils de sécurité

## SONARQUBE

- Bugs :
- Vulnérabilités :
- Hotspots :
- Corrections :

## OWASP ZAP

- Scan : baseline + routes ciblées
- Alertes : CORS / headers / auth
- Avant/Après :
- Faux positifs :

## npm audit

- Vulnérabilités : [à compléter)
- Actions : audit fix + upgrades
- Risques acceptés : [si besoin)
- Objectif : composants à jour

# Sécurisation - Extraits de code

## Ce qui est bon

- Architecture MVC claire + séparation responsabilités
- RBAC admin/user + routes protégées
- Validation + erreurs centralisées
- Outils : Sonar / ZAP / npm audit (preuves)

## A améliorer

- Stocker hash des refresh tokens (au lieu de clair)
- CSRF : store partagé Redis en prod multi-instance
- Logs structurés + centralisation (ELK / Loki)
- Tests d'intégration sécurité (RBAC/injection;bruteforce)

**Merci pour  
votre écoute !**

**Mathis BOHEC & Marie COURTOIS**