API CINEMA

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

Sécurisation avant mise en production

Par Mathis BOHEC & Marie COURTOIS

Sommaire

I- Besoin & cas d’usage

II- Architecture technique

III- Modèles de données

IV- Sécurisation (mesures + preuves)

V- OWASP (mapping)

VI- Outils (SONAR / zap / npm audit)

VII- Démo

VIII- Bilan & amélioration

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

Client
(Web / Mobile)

→

API Express
REST ◦ MVC

→

Données
PostgreSQL + MongoDB

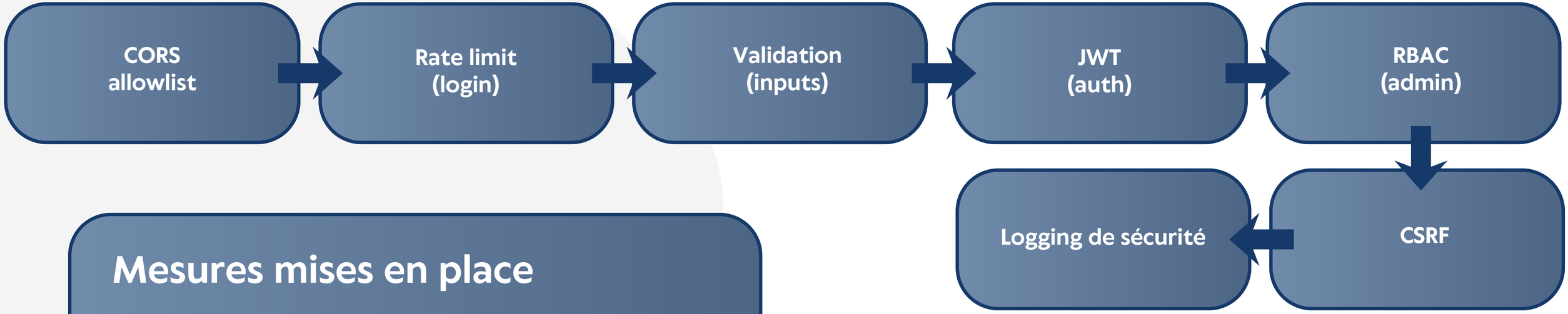
Pourquoi SQL + NoSQL ?

- 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
144  // Limiter à 5 refresh tokens actifs par utilisateur (rotation)
145  const userWithTokens = await User.findById(user._id).select('+refreshTokens');
146  const refreshTokens = userWithTokens.refreshTokens || [];
147  if (refreshTokens.length >= 5) {
148      refreshTokens.shift();
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

```js
const { validationResult } = require("express-validator");

function validate(req, res, next) {
  const errors = validationResult(req);
  if (!errors.isEmpty()) {
    return res.status(422).json({ errors: errors.array() });
  }
  next();
}

```

CORS

```js
app.use(cors({
    origin: function (origin, callback) {
        // Autoriser les requêtes sans origine (mobile apps, Postman, etc.) en développement
        if (!origin && process.env.NODE_ENV !== 'production') {
            return callback(null, true);
        }
        if (allowedOrigins.indexOf(origin) !== -1 || !origin) {
            callback(null, true);
        } else {
            callback(new Error('Non autorisé par CORS'));
        }
    },
    credentials: true,
    allowedHeaders: ["Authorization", "Content-Type", "X-Requested-With"],
    methods: ["GET", "POST", "PUT", "DELETE", "PATCH", "OPTIONS"]
}));
```

Sécurisation - Extraits de code

CSRF

```js
const crypto = require("node:crypto");

const csrfTokens = new Map();
const CSRF_TOKEN_EXPIRY = 60 * 60 * 1000; // 1 heure

/**
 * Génère un token CSRF
 */
function generateCSRFToken() {
    return crypto.randomBytes(32).toString('hex');
}


function generateCSRF(req, res, next) {
    const token = generateCSRFToken();
    const expiry = Date.now() + CSRF_TOKEN_EXPIRY;

    csrfTokens.set(token, {
        expiry,
        ip: req.ip || req.connection.remoteAddress
    });

    if (csrfTokens.size > 1000) {
        const now = Date.now();
        for (const [t, data] of csrfTokens.entries()) {
            if (data.expiry < now) {
                csrfTokens.delete(t);
            }
        }
    }

    res.locals.csrfToken = token;
    res.setHeader('X-CSRF-Token', token);
    next();
}
```

CSRF

```js
function verifyCSRF(req, res, next) {

    if (req.path.startsWith('/api/user/login') ||
        req.path.startsWith('/api/user/register') ||
        req.path.startsWith('/api/user/refresh')) {
        return next();
    }


    const token = req.headers['x-csrf-token'] || req.body?.csrfToken;

    if (!token) {

        if (process.env.NODE_ENV === 'development') {
            return next();
        }
        return res.status(403).json({ error: "Token CSRF manquant" });
    }

    const tokenData = csrfTokens.get(token);

    if (!tokenData) {
        return res.status(403).json({ error: "Token CSRF invalide" });
    }

    if (tokenData.expiry < Date.now()) {
        csrfTokens.delete(token);
        return res.status(403).json({ error: "Token CSRF expiré" });
    }

    csrfTokens.delete(token);

    next();
}
```

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

Sécurisation - Extraits de code

RBAC

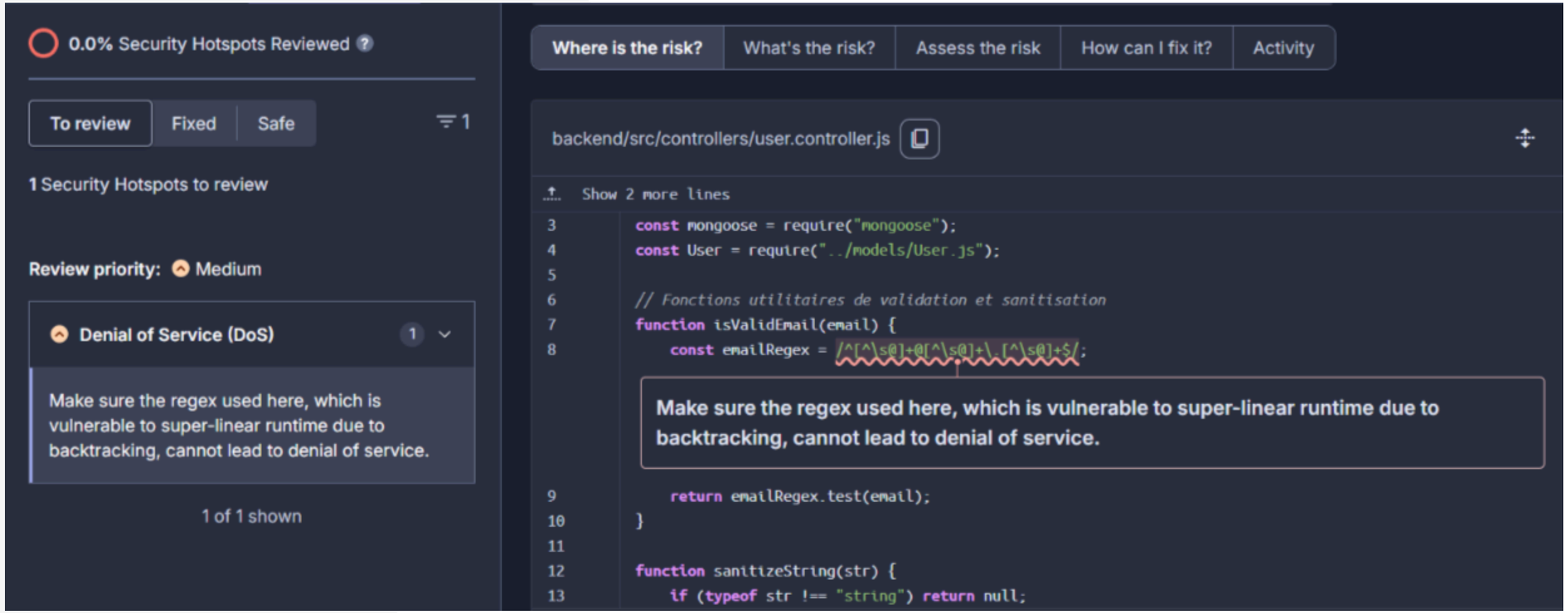
```
logSensitiveAccess(req, resource, action) {
    const logData = {
        timestamp: new Date().toISOString(),
        event: 'SENSITIVE_ACCESS',
        userId: req.user?.userId || 'N/A',
        role: req.user?.role || 'N/A',
        resource,
        action,
        ip: req.ip || req.connection.remoteAddress,
        userAgent: req.get('user-agent'),
        method: req.method,
        path: req.path
    };

    console.log(`[SECURITY] 🔒 Accès sensible: ${action} sur ${resource} par utilisateur ${logData.userId} (${logData.role})`);
},
```

Sécurisation - Extraits de code

Helmet (headers sécurisés)

Sécurisation - Extrait de Sonar



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