Authentification JWT — Architecture Simple

Objectif :

- Login / Register sécurisé
- Hash password avec bcrypt
- Génération JWT
- Middleware de protection
- Séparation routes / controllers / services / plugins

Architecture :

```
routes → controllers → services → prisma
                        ↓
                      utils (jwt)
                        ↓
                     plugins (auth)
```

---

1. Installation

```
npm install bcrypt jsonwebtoken
npm install -D @types/bcrypt @types/jsonwebtoken
```

---

2. Modèle Prisma

```
model User {
  id        Int      @id @default(autoincrement())
  email     String   @unique
  password  String
  createdAt DateTime @default(now())
}
```

```
npx prisma migrate dev
```

---

3. Structure du projet

```
src/
 ├── prisma.ts
 ├── routes/
 │     └── user.routes.ts
 ├── controllers/
 │     └── user.controller.ts
 ├── services/
 │     └── user.service.ts
 ├── plugins/
 │     └── auth.ts
 ├── utils/
 │     └── jwt.ts
 └── schemas/
       └── user.schema.ts
```

---

4. Utils JWT

`src/utils/jwt.ts`

```
import jwt from "jsonwebtoken"

const JWT_SECRET = process.env.JWT_SECRET || "supersecret"

export function signToken(payload: { id: number }) {
  return jwt.sign(payload, JWT_SECRET, {
    expiresIn: "1h"
  })
}

export function verifyToken(token: string) {
  return jwt.verify(token, JWT_SECRET)
}
```

En production :

- Toujours utiliser une variable d'environnement
- Ne jamais commit le secret

---

5. Service (Logique métier)

`src/services/user.service.ts`

```
import bcrypt from "bcrypt"
import { prisma } from "../prisma"
import { signToken } from "../utils/jwt"

export async function register(email: string, password: string) {
  const hashedPassword = await bcrypt.hash(password, 10)
  return prisma.user.create({
    data: {
      email,
      password: hashedPassword
    }
  })
}

export async function login(email: string, password: string) {
  const user = await prisma.user.findUnique({
    where: { email }
  })

  if (!user) {
    throw new Error("Invalid credentials")
  }

  const valid = await bcrypt.compare(password, user.password)

  if (!valid) {
    throw new Error("Invalid credentials")
  }

  const token = signToken({ id: user.id })

  return { token }
}

export async function getById(id: number) {
  return prisma.user.findUnique({
    where: { id },
    select: {
      id: true,
      email: true
    }
  })
}
```

---

6. Controller

`src/controllers/user.controller.ts`

```

import { FastifyRequest, FastifyReply } from "fastify"
import * as userService from "../services/user.service"
import { registerSchema, loginSchema } from "../schemas/user.schema"

export async function register(request: FastifyRequest, reply: FastifyReply) {
  const { email, password } = registerSchema.parse(request.body)

  const user = await userService.register(email, password)

  return reply.status(201).send(user)
}

export async function login(request: FastifyRequest, reply: FastifyReply) {
  const { email, password } = loginSchema.parse(request.body)

  const result = await userService.login(email, password)

  return reply.send(result)
}
```

---

7. Middleware JWT

`src/plugins/auth.ts`

```
import { FastifyRequest, FastifyReply } from "fastify"
import { verifyToken } from "../utils/jwt"

export async function authMiddleware(
  request: FastifyRequest,
  reply: FastifyReply
) {
  const header = request.headers.authorization

  if (!header) {
    return reply.status(401).send({ message: "Unauthorized" })
  }

  const token = header.split(" ")[1]

  try {
    const decoded = verifyToken(token)
    request.user = decoded
  } catch {
    return reply.status(401).send({ message: "Invalid token" })
  }
}
```

---

8. Extension Type Fastify

Créer `src/fastify.d.ts`

```
import "fastify"

declare module "fastify" {
  interface FastifyRequest {
    user?: any
  }
}
```

9. Routes

`src/routes/user.routes.ts`

```
import { FastifyInstance } from "fastify"
import * as userController from "../controllers/user.controller"
import { authMiddleware } from "../plugins/auth"
import * as userService from "../services/user.service"

export async function userRoutes(app: FastifyInstance) {

  app.post("/register", userController.register)
  app.post("/login", userController.login)

  app.get(
    "/me",
    { preHandler: [authMiddleware] },
    async (request, reply) => {
      const user = await userService.getById(request.user.id)
      return reply.send(user)
    }
  )

}
```

10. Bonnes pratiques sécurité

- Ne jamais retourner le password
- Utiliser select dans Prisma
- JWT court (15min–1h)
- Ajouter Refresh Token en prod
- Ajouter rate limit sur /login
- Utiliser variable d’environnement