Partie 3 — Validation Zod + Gestion d'erreurs centralisée

Objectif :

- Valider proprement les entrées utilisateur
- Éviter les “as any”
- Centraliser les erreurs
- Standardiser les réponses API

---

1. Installation Zod

```
npm install zod
```

---

2. Organisation recommandée

Nouvelle structure :

```
src/
 ├── errors/
 │     └── AppError.ts
 ├── schemas/
 │     ├── auth.schema.ts
 │     └── post.schema.ts
 ├── plugins/
 │     └── errorHandler.ts
```

---

3. Création d'une erreur personnalisée

`src/errors/AppError.ts`

```
export class AppError extends Error {
  statusCode: number

  constructor(message: string, statusCode = 400) {
    super(message)
    this.statusCode = statusCode
  }
}
```

---

4. Handler d'erreurs global

`src/plugins/errorHandler.ts`

```
import { FastifyInstance } from "fastify"
import { ZodError } from "zod"
import { AppError } from "../errors/AppError"

export async function errorHandler(app: FastifyInstance) {
  app.setErrorHandler((error, request, reply) => {

    // Erreurs Zod
    if (error instanceof ZodError) {
      return reply.status(400).send({
        error: "Validation error",
        details: error.errors
      })
    }

    // Erreurs applicatives
    if (error instanceof AppError) {
      return reply.status(error.statusCode).send({
        error: error.message
      })
    }

    // Erreurs inattendues
    request.log.error(error)

    return reply.status(500).send({
      error: "Internal Server Error"
    })
  })
}
```

---

5. Enregistrer le plugin

Dans `server.ts` :

```
import { errorHandler } from "./plugins/errorHandler"

app.register(errorHandler)
```

⚠ À enregistrer après les autres plugins.

---

6. Création des schémas Zod

Exemple : Auth

`src/schemas/auth.schema.ts`

```
import { z } from "zod"

export const registerSchema = z.object({
  name: z.string().min(2),
  email: z.string().email(),
  password: z.string().min(6)
})

export const loginSchema = z.object({
  email: z.string().email(),
  password: z.string().min(6)
})
```

---

7. Utilisation dans Controller

Avant (dangereux) :

```
const { email, password } = request.body as any
```

Maintenant (safe) :

```
import { registerSchema } from "../schemas/auth.schema"

export async function register(request, reply) {

  const data = registerSchema.parse(request.body)

  const user = await authService.registerUser(data)

  return user
}
```

- □ Si invalide → ZodError
- □ Capturée automatiquement par le handler global

---

8. Exemple Service avec AppError

`auth.service.ts`

```
import { AppError } from "../errors/AppError"

async registerUser(data) {

  const existing = await prisma.user.findUnique({
    where: { email: data.email }
  })

  if (existing) {
    throw new AppError("Email already used", 409)
  }

  // logique création utilisateur
}
```

---

9. Format de réponse d’erreur standard

Erreurs validation :

```
{
  "error": "Validation error",
  "details": [...]
}
```

Erreurs métier :

```
{
  "error": "Email already used"
}
```

Erreur serveur :

```
{
  "error": "Internal Server Error"
}
```
