

# Compte-rendu tp 5

??= ⇒ aucune affectation n'est effectuée si le côté gauche n'est pas nul

a.speed ??= 25 ⇒ si a.speed == null, a.speed == 25

**les explications nécessaires pour comprendre démarche**

openssl du tp4

mongoose : <https://mongoosejs.com/docs/guide.html>

mongodb : <https://www.mongodb.com/docs/manual/crud/>

JSON Schema :

## JSON SCHEMA POUR VALIDER LES ENTRÉES

```
const userSchema = {
  schema: {
    body: {
      type: "object",
      properties: {
        name: {type: "string"},
        age: {type: "integer"}
      },
      required: ["name", "age"],
    },
  },
}

app.post('/user', userSchema, async (req, rep) => {
  return {status: 'ok', data: req.body}
})
```

BUT Informatique – R6.A.05 DA – Code & Build d'un service web – Laurent Giustignano – 2023–2024

FASTIFY - JSON SCHEMA

24

## JSON SCHEMA POUR VALIDER LES RÉPONSES

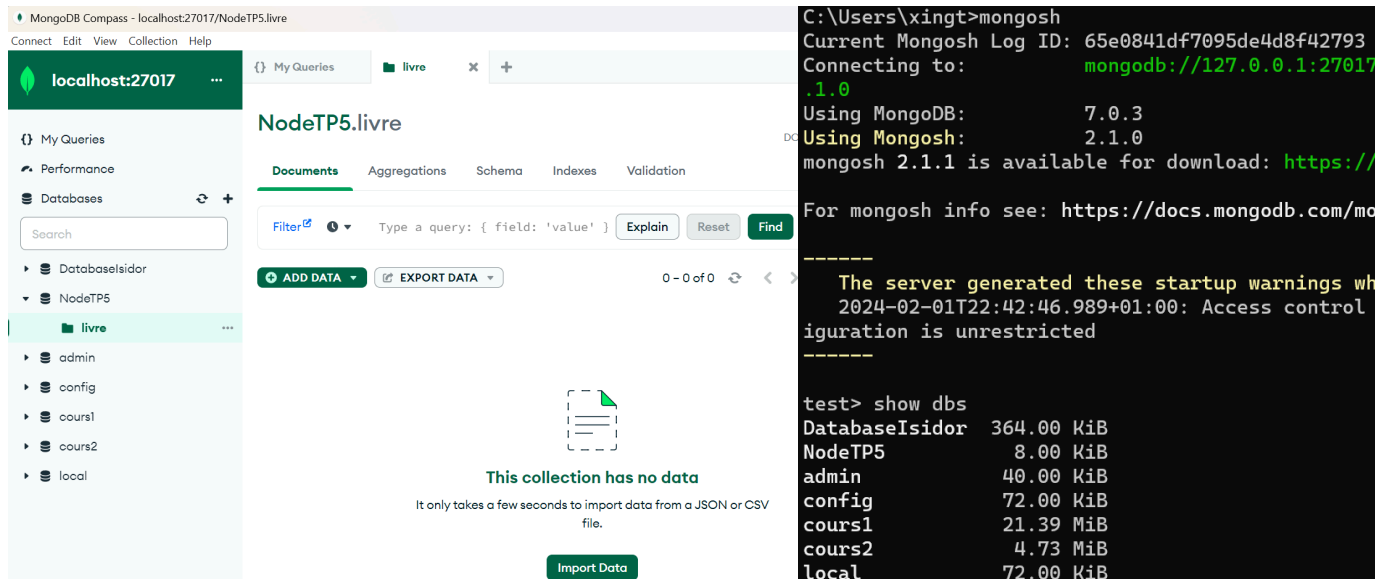
```
const User = {
  type: 'object',
  properties: {
    name: {type: "string"},
    age: {type: "integer"}
  },
}

const getUsers = {
  schema: {
    response: {
      200: {
        type: 'array',
        items: User
      }
    }
  }
}

app.get('/user', getUsers, async (req, res) => {
  return [
    {name: 'Jaime', age: 32},
    {name: 'Cersei', age: 32},
    {name: 'Tyrion', age: 27},
  ],
  taille: 120,
})
```

## vos choix, ce qui a marché facilement

Étape 1 : J'ai créé une base de données, puis j'ai vérifié son existence avec la commande "show dbs".



The image shows two side-by-side screenshots. The left screenshot is from MongoDB Compass, displaying the 'livre' collection in the 'NodeTP5' database. The right screenshot is a terminal window showing the output of the 'mongosh' command, including the current log ID, connection details, and the result of the 'show dbs' command.

**MongoDB Compass - localhost:27017/NodeTP5.livre**

Connect Edit View Collection Help

localhost:27017

My Queries

livre

NodeTP5.livre

Documents Aggregations Schema Indexes Validation

Filter Type a query: { field: 'value' } Explain Reset Find

ADD DATA EXPORT DATA

0 - 0 of 0

This collection has no data

It only takes a few seconds to import data from a JSON or CSV file.

Import Data

**Terminal Output:**

```
C:\Users\xingt>mongosh
Current Mongosh Log ID: 65e0841df7095de4d8f42793
Connecting to: mongodb://127.0.0.1:27017
Using MongoDB: 7.0.3
Using Mongosh: 2.1.0
mongosh 2.1.1 is available for download: https://
For mongosh info see: https://docs.mongodb.com/mo

-----
The server generated these startup warnings wh
2024-02-01T22:42:46.989+01:00: Access control
igation is unrestricted
-----

test> show dbs
DatabaseIsidor 364.00 KiB
NodeTP5         8.00 KiB
admin           40.00 KiB
config          72.00 KiB
cours1          21.39 MiB
cours2          4.73 MiB
local           72.00 KiB
```

## Étape 2 :

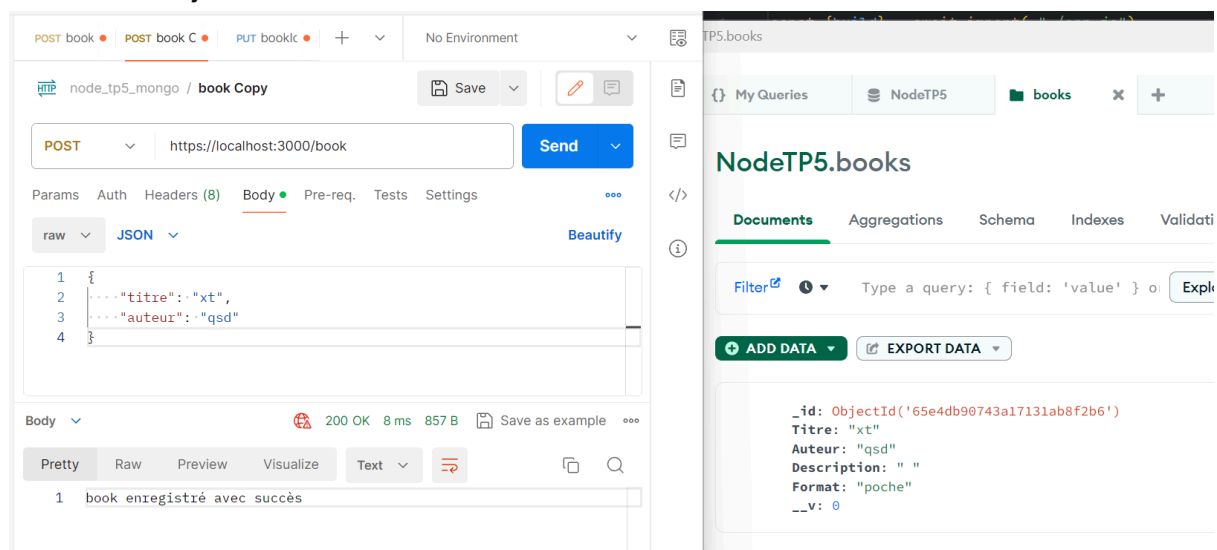
- Création de clés et de certificats en utilisant OpenSSL, puis intégration avec Fastify.
- Création d'un fichier qui se connecte au service MongoDB, puis l'utilise dans le fichier server.js.
- Création d'un Mongoose Schema "!\ type: String".

## Étape 3:

Création de différents fichiers de contrôleur et routes.js.

- addBook

## 1. cas d'ajout d'un livre réussi



The image shows two side-by-side screenshots. The left screenshot is from a REST client, showing a successful POST request to 'https://localhost:3000/book'. The right screenshot is from MongoDB Compass, showing the 'books' collection in the 'NodeTP5' database with the newly added book document.

**REST Client:**

POST book C

node\_tp5\_mongo / book Copy

POST https://localhost:3000/book

Send

Params Auth Headers (8) Body Pre-req. Tests Settings

raw JSON Beautify

```
1 {
2   "titre": "xt",
3   "auteur": "qsd"
4 }
```

Body 200 OK 8 ms 857 B Save as example

Pretty Raw Preview Visualize Text

1 book enregistré avec succès

**MongoDB Compass - TP5.books**

My Queries NodeTP5 books

NodeTP5.books

Documents Aggregations Schema Indexes Validation

Filter Type a query: { field: 'value' } Explain

ADD DATA EXPORT DATA

```
_id: ObjectId('65e4db90743a17131ab8f2b6')
Titre: "xt"
Auteur: "qsd"
Description: " "
Format: "poche"
__v: 0
```

## 2. cas d'un livre déjà existant

The screenshot shows a Postman client on the left and the NodeTP5 books web interface on the right. In Postman, a POST request to `https://localhost:3000/book` is shown with a JSON body: `{ "titre": "xt", "auteur": "qsd" }`. The response is a 200 OK status with a body of `book déjà enregistré`. On the right, the NodeTP5 books interface shows a list of documents. The document with `titre: "xt"` and `auteur: "qsd"` is highlighted, showing its details: `_id: ObjectId('65e4db90743a17131ab8')`, `Titre: "xt"`, `Auteur: "qsd"`, `Description: " "`, `Format: "poche"`, and `__v: 0`.

## 3. JSON Schema “/!\ type: ‘string’” - pour valider l'entrée - forcer la valeur de format

The screenshot shows a Postman client on the left and the NodeTP5 books web interface on the right. In Postman, a POST request to `https://localhost:3000/book` is shown with a JSON body: `{ "titre": "yy", "auteur": "qsd", "desc": "", "format": "rtgt" }`. The response is a 400 Bad Request status with a body: `{ "statusCode": 400, "code": "FST_ERR_VALIDATION", "error": "Bad Request", "message": "body/format must be equal to one of the allowed values" }`. On the right, the NodeTP5 books interface shows a list of documents. The document with `titre: "yy"` and `auteur: "qsd"` is highlighted, showing its details: `_id: ObjectId('65e4db90743a17131ab8')`, `Titre: "yy"`, `Auteur: "qsd"`, `Description: " "`, `Format: "poche"`, and `__v: 0`.

```
export const BookInfoSchema :... = {
  type: "object",
  properties: {
    titre: {type: "string"},
    auteur: {type: "string"},
    desc: {type: "string", default: " "},
    format: {
      type: 'string',
      enum: ["poche", "manga", "audio"],
      default: "poche"
    },
  },
  required: ["titre", "auteur"],
}
```

#### 4. JSON Schema - pour valider l'entrée - titre obligatoire

node\_tp5\_mongo / book

POST https://localhost:3000/book

Send

Params Auth Headers (8) Body Pre-req. Tests Settings

raw JSON Beautify

```
1 {
2   "titre": "yy",
3   "auteur": "qsd",
4   "desc": "",
5   "format": "rtgt"
6 }
```

ody 400 Bad Request 3 ms 965 B Save as example

Pretty Raw Preview Visualize JSON

```
1 {
2   "statusCode": 400,
3   "code": "FST_ERR_VALIDATION",
4   "error": "Bad Request",
5   "message": "body must have required property 'titre'"
6 }
```

```
export const BookInfoSchema :... = {
  type: "object",
  properties: {
    titre: {type: "string"},
    auteur: {type: "string"},
    desc: {type: "string", default: ""},
    format: {
      type: 'string',
      enum: ["poche", "manga", "audio"],
      default: "poche"
    },
  },
  required: ["titre", "auteur"],
}
```

#### - deleteBook

##### 1. cas de suppression d'un livre inexistant

node\_tp5\_mongo / book Copy

DELETE https://localhost:3000/book

Send

Params Auth Headers (8) Body Pre-req. Tests Settings

raw JSON Beautify

```
1 {
2   "titre": "x",
3   "auteur": "qsd"
4 }
```

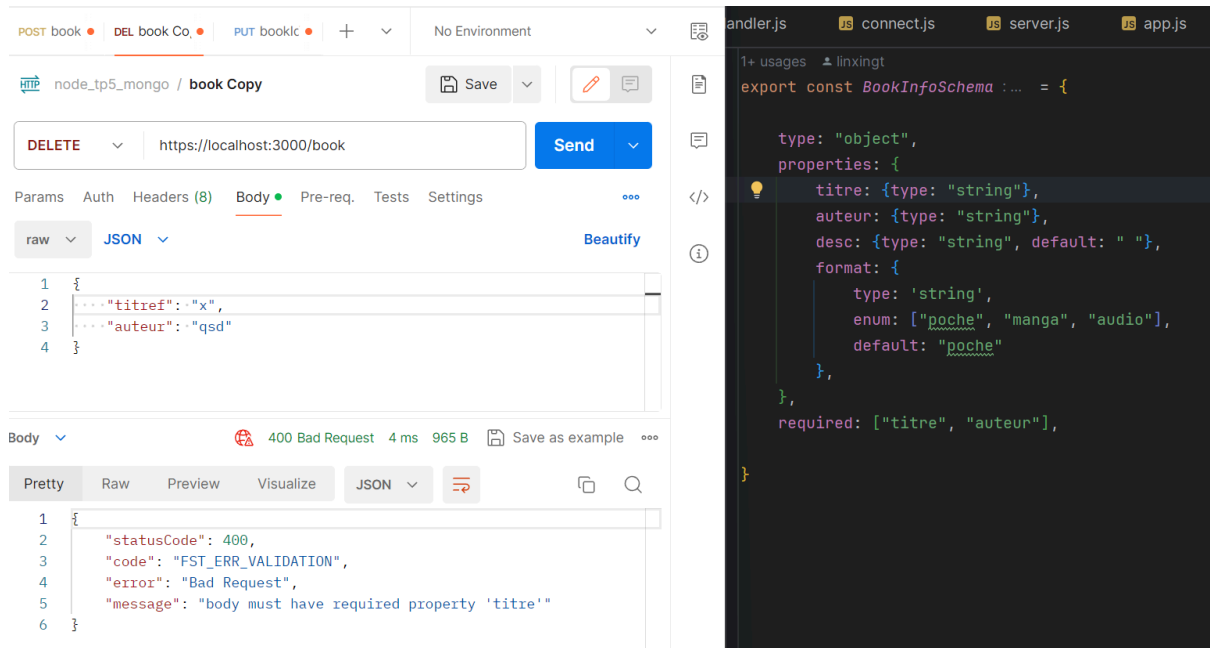
Body 200 OK 7 ms 843 B Save as example

Pretty Raw Preview Visualize Text

```
1 book non existe
```

```
2
3 1+ usages 1 linking
4 export const deleteBook = async (req, res) : Promise<...> => {
5   try {
6     let book : Query<...> & ObtainSchemaGeneric<module:mon... = await Book.find
7       Titre: req.body.titre,
8       Auteur: req.body.auteur
9     })
10    if (book) {
11      const deletedBook : Query<...> & ObtainSchemaGeneric<module:mon... = a
12        filter: {
13          Titre: req.body.titre,
14          Auteur: req.body.auteur
15        })
16      if (deletedBook.deletedCount > 0) {
17        return res.send('book supprimé avec succès');
18      }
19      return res.send('book supprimé raté');
20    }
21    return res.send('book non existe');
22  } catch (error) {
```

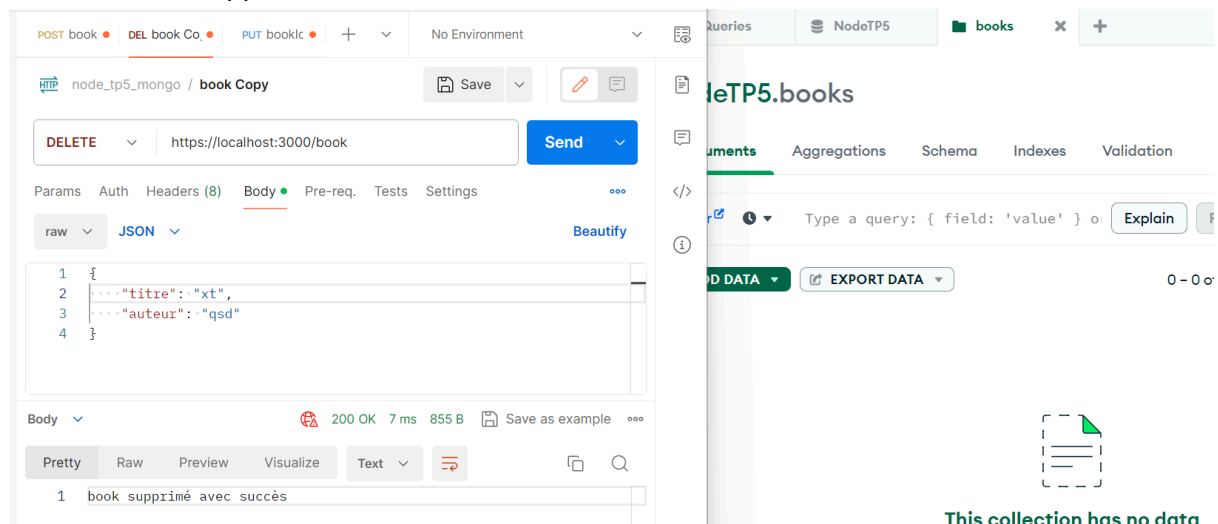
## 2. JSON Schema - pour valider l'entrée - titre obligatoire



The screenshot shows a REST client interface on the left and a code editor on the right. The REST client is configured for a DELETE request to `https://localhost:3000/book`. The request body is a JSON object: `{ "titre": "x", "auteur": "qsd" }`. The response body shows a 400 Bad Request error with the message: `"body must have required property 'titre'"`. The code editor on the right shows a JSON Schema definition for `BookInfoSchema` with the following structure:

```
export const BookInfoSchema : ... = {  
  type: "object",  
  properties: {  
    titre: {type: "string"},  
    auteur: {type: "string"},  
    desc: {type: "string", default: " "},  
    format: {  
      type: "string",  
      enum: ["poche", "manga", "audio"],  
      default: "poche"  
    },  
  },  
  required: ["titre", "auteur"],  
}
```

## 3. cas de suppression réussie



The screenshot shows a REST client interface on the left and a MongoDB interface on the right. The REST client is configured for a DELETE request to `https://localhost:3000/book`. The request body is a JSON object: `{ "titre": "xt", "auteur": "qsd" }`. The response body shows a 200 OK status with the message: `book supprimé avec succès`. The MongoDB interface on the right shows the `books` collection in the `leTP5` database. The collection is empty, with a message: `This collection has no data`.

- search
  - searchAllBook

## 1. cas réussi avec JSON Schema - pour valider la sortie

The image shows a REST client interface on the left and a database interface on the right.

**REST Client Interface:**

- Method: GET
- URL: `https://localhost:3000/book`
- Status: 200 OK, 5 ms, 1.07 KB
- Body (JSON):
 

```

1 [
2   {
3     "Titre": "asz",
4     "Auteur": "qsd",
5     "Description": "",
6     "Format": "manga"
7   },
8   {
9     "Titre": "aa",
10    "Auteur": "qsd",
11    "Description": "",
12    "Format": "poche"
13  },
14  {
15    "Titre": "xx",
16    "Auteur": "qsd",
17    "Description": "",
18    "Format": "poche"
19  },
20  {
21    "Titre": "tt",

```

**Database Interface:**

- Collection: `books`
- Documents:
  - ```

_id: ObjectId('65e4df09743a17131ab8f2ce')
Titre: "asz"
Auteur: "qsd"
Description: ""
Format: "manga"
__v: 0

```
  - ```

_id: ObjectId('65e4df15743a17131ab8f2d1')
Titre: "aa"
Auteur: "qsd"
Description: ""
Format: "poche"
__v: 0

```
  - ```

_id: ObjectId('65e4df1b743a17131ab8f2d4')
Titre: "xx"
Auteur: "qsd"
Description: ""
Format: "poche"
__v: 0

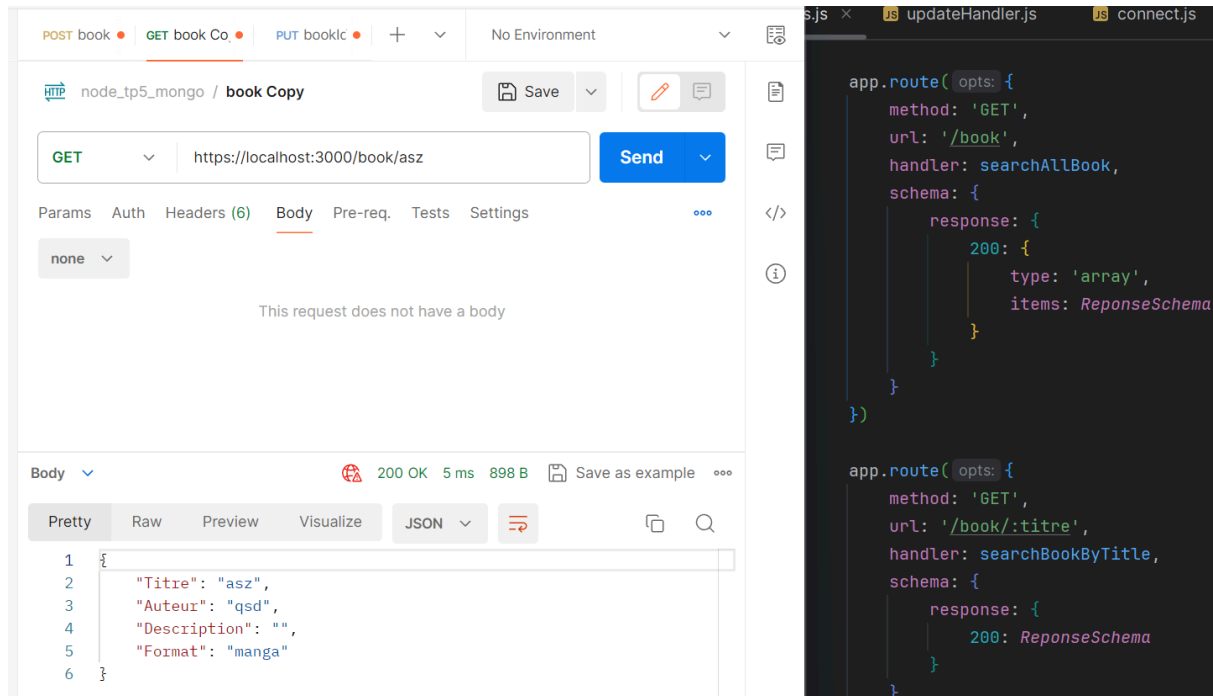
```
  - ```

_id: ObjectId('65e4df1d743a17131ab8f2d7')
Titre: "tt"
Auteur: "qsd"
Description: ""
Format: "poche"
__v: 0

```

## - searchBookByTitle

### 1. cas réussi avec JSON Schema - pour valider la sortie



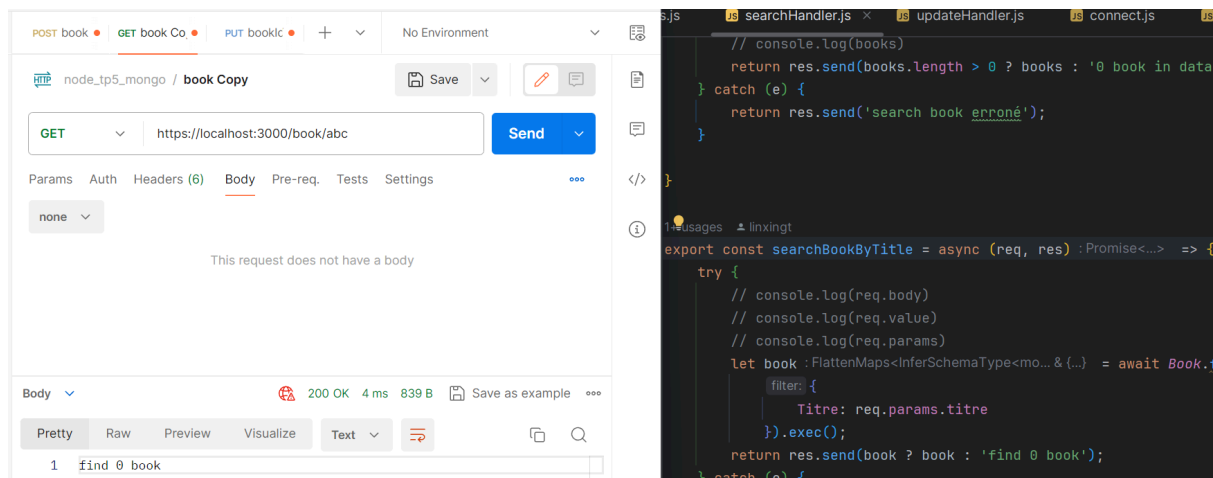
The screenshot shows a REST client interface on the left and a code editor on the right. The REST client is configured with a GET request to `https://localhost:3000/book/asz`. The response is a 200 OK status with a JSON body:

```
{
  "Titre": "asz",
  "Auteur": "qsd",
  "Description": "",
  "Format": "manga"
}
```

The code editor on the right shows the `searchAllBook` handler in `updateHandler.js`:

```
app.route({
  method: 'GET',
  url: '/book',
  handler: searchAllBook,
  schema: {
    response: {
      200: {
        type: 'array',
        items: ResponseSchema
      }
    }
  }
})
```

### 2. recherche d'un livre inexistant



The screenshot shows a REST client interface on the left and a code editor on the right. The REST client is configured with a GET request to `https://localhost:3000/book/abc`. The response is a 200 OK status with a JSON body:

```
{
  "find 0 book"
}
```

The code editor on the right shows the `searchBookByTitle` handler in `searchHandler.js`:

```
// console.log(books)
return res.send(books.length > 0 ? books : '0 book in data')
} catch (e) {
  return res.send('search book erroné');
}

export const searchBookByTitle = async (req, res) : Promise<...> => {
  try {
    // console.log(req.body)
    // console.log(req.value)
    // console.log(req.params)
    let book : FlattenMaps<InferSchemaType<mo... & {...}> = await Book.
      .filter({
        Titre: req.params.titre
      }).exec();
    return res.send(book ? book : 'find 0 book');
  } catch (e) {
    // console.log(e)
  }
}
```

## - updateBook

### 1. cas de mise à jour réussi

The screenshot shows a Postman client on the left and a MongoDB Atlas interface on the right. In Postman, a PUT request to `https://localhost:3000/bookId/65e4df09743a17131a...` with a JSON body `{ "titre": "xta", "auteur": "abc" }` was sent, resulting in a 200 OK response. The response body in Postman shows `book update avec succès`. The MongoDB Atlas interface on the right shows the `books` collection with two documents. The first document has `_id: ObjectId('65e4df09743a17131ab8f2ce')`, `Titre: "xta"`, `Auteur: "abc"`, `Description: " "`, `Format: "poche"`, and `__v: 0`. The second document has `_id: ObjectId('65e4df15743a17131ab8f2d1')`, `Titre: "aa"`, `Auteur: "gsd"`, `Description: ""`, `Format: "poche"`, and `__v: 0`.

```
PUT https://localhost:3000/bookId/65e4df09743a17131a...
{
  "titre": "xta",
  "auteur": "abc"
}
```

200 OK 12 ms 852 B

book update avec succès

NodeTP5.books

Documents Aggregations Schema Indexes Valid

Filter Type a query: { field: 'value' } o Ex

ADD DATA EXPORT DATA

`_id: ObjectId('65e4df09743a17131ab8f2ce')`  
`Titre: "xta"`  
`Auteur: "abc"`  
`Description: " "`  
`Format: "poche"`  
`__v: 0`

`_id: ObjectId('65e4df15743a17131ab8f2d1')`  
`Titre: "aa"`  
`Auteur: "gsd"`  
`Description: ""`  
`Format: "poche"`  
`__v: 0`

### 2. cas où rien n'a été changé

The screenshot shows a Postman client on the left and a MongoDB Atlas interface on the right. In Postman, a PUT request to `https://localhost:3000/bookId/65e4df09743a17131a...` with a JSON body `{ "titre": "xta", "auteur": "abc" }` was sent, resulting in a 200 OK response. The response body in Postman shows `pas de modification`. The MongoDB Atlas interface on the right shows the `books` collection with two documents. The first document has `_id: ObjectId('65e4df09743a17131ab8f2ce')`, `Titre: "xta"`, `Auteur: "abc"`, `Description: " "`, `Format: "poche"`, and `__v: 0`. The second document has `_id: ObjectId('65e4df15743a17131ab8f2d1')`, `Titre: "aa"`, `Auteur: "gsd"`, `Description: ""`, `Format: "poche"`, and `__v: 0`.

```
PUT https://localhost:3000/bookId/65e4df09743a17131a...
{
  "titre": "xta",
  "auteur": "abc"
}
```

200 OK 5 ms 847 B

pas de modification

eTP5.books

Documents Aggregations Schema Indexes Valid

Filter Type a query: { field: 'value' } o E

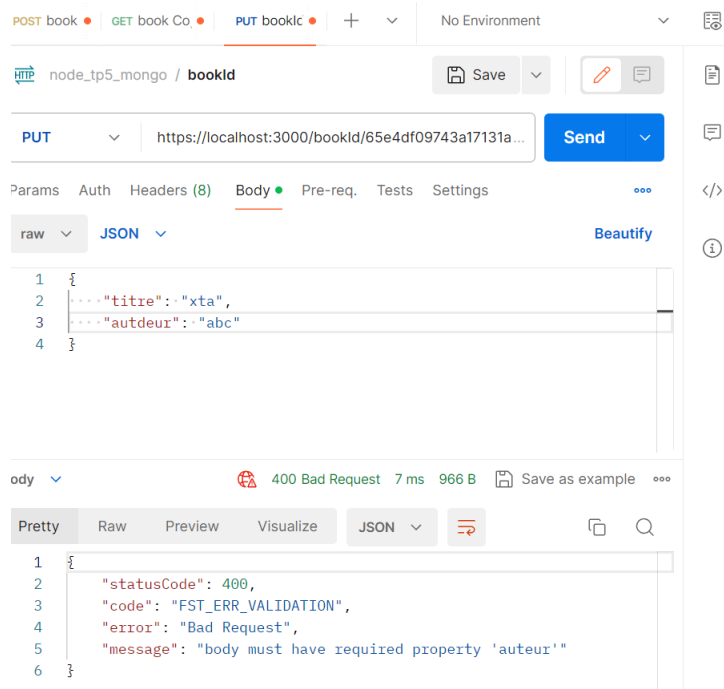
D DATA EXPORT DATA

`_id: ObjectId('65e4df09743a17131ab8f2ce')`  
`Titre: "xta"`  
`Auteur: "abc"`  
`Description: " "`  
`Format: "poche"`  
`__v: 0`

`_id: ObjectId('65e4df15743a17131ab8f2d1')`  
`Titre: "aa"`  
`Auteur: "gsd"`  
`Description: ""`  
`Format: "poche"`  
`__v: 0`



### 3. JSON Schema - pour valider l'entrée - auteur obligatoire



POST book • GET book Co. • PUT bookId • + No Environment

node\_tp5\_mongo / bookId

PUT https://localhost:3000/bookId/65e4df09743a17131a... Send

Params Auth Headers (8) Body • Pre-req. Tests Settings

raw JSON Beautify

```
1 {
2   "titre": "xta",
3   "auteur": "abc"
4 }
```

ody 400 Bad Request 7 ms 966 B Save as example

Pretty Raw Preview Visualize JSON

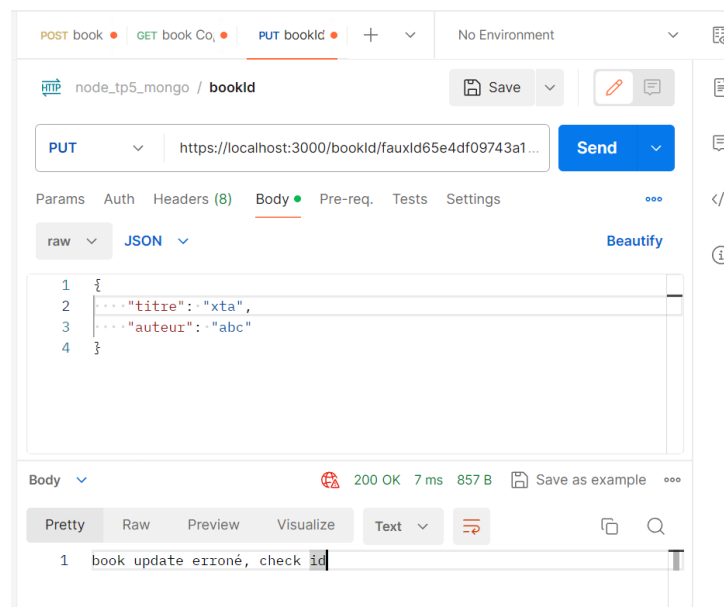
```
1 {
2   "statusCode": 400,
3   "code": "FST_ERR_VALIDATION",
4   "error": "Bad Request",
5   "message": "body must have required property 'auteur'"
6 }
```

```
method: 'GET',
url: '/book/:titre',
handler: searchBookByTitle,
schema: {
  response: {
    200: ReponseSchema
  }
})

app.route( opts: {
  method: 'PUT',
  url: '/bookId/:id',
  handler: updateBook,
  schema: {
    body: BookInfoSchema
  },
})

app.route( opts: {
  method: 'DELETE',
  url: '/book',
  handler: deleteBook,
  schema: {
    body: BookInfoSchema
  },
})
```

### 4. cas avec un ID incorrect



POST book • GET book Co. • PUT bookId • + No Environment

node\_tp5\_mongo / bookId

PUT https://localhost:3000/bookId/fauxId65e4df09743a1... Send

Params Auth Headers (8) Body • Pre-req. Tests Settings

raw JSON Beautify

```
1 {
2   "titre": "xta",
3   "auteur": "abc"
4 }
```

Body 200 OK 7 ms 857 B Save as example

Pretty Raw Preview Visualize Text

```
1 book update erroné, check id
```

```
let resu : UpdateResult<Document> = await B
  filter: { _id: req.params.id },
  update: {
    $set: {
      Titre: req.body.titre,
      Auteur: req.body.auteur,
      Description: req.body.desc,
      Format: req.body.format
    }
  }
).exec()
console.log(resu)
if (resu.modifiedCount > 0)
  return res.send('book update avec succ
else if (resu.matchedCount > 0 && resu.mod
  return res.send('pas de modification'
else if (resu.matchedCount === 0)
  return res.send('pas de book correspon
  return res.send('book update raté');
} catch (error) {
  console.error(error);
  res.send('book update erroné, check id');
```

### vos difficultés, comment vous les avez surmontées

Au début, je me suis confondue entre JSON Schema et Mongoose Schema. J'avais pensé que les deux étaient les mêmes choses, puis j'ai trouvé plusieurs différences entre eux et j'ai enfin réussi à utiliser JSON Schema pour valider les entrées.

### qu'est qu'on pourrait améliorer

Pour moi, ce serait d'utiliser JSON Schema pour valider les entrées et les sorties dans une seule route, car cette fois-ci, je n'ai fait qu'un JSON Schema par route.