

Compte Rendu

TP Api NodeJS



Noa Bukovec
13/06/2025

Table des matières

1. Introduction	2
Objectif du TP	2
Présentation des 2 versions	2
2. Version JSON	2
Méthode de lecture des données	2
Routes implémentées	2
Tests réalisés	4
Limites constatées.....	6
3. Version SQLite.....	7
Présentation de la base	7
Méthode d'import initial	7
Routes implémentées	8
Gestion des erreurs.....	10
Résultats des tests.....	11
4. Bonus réalisés	13
5. Analyse critique	16
Avantages et inconvénients	16
Difficultés rencontrées et solutions apportées	16
6. Conclusion	17
Bilan du travail accompli	17
Ce que vous avez appris / retenu	17
GitHub.....	17

1. Introduction

Objectif du TP

L'objectif de ce TP était de développer une API REST en Node.js avec Express.js pour manipuler des données de super-héros, d'abord via un fichier JSON puis via une base SQLite.

Présentation des 2 versions

- Version JSON :

Cette version est très basique, elle ne demande pas d'implémenter beaucoup de choses. L'ensemble des données sont stockées en mémoire.

- Version SQLite :

Cette version demande plus de moyen avec la mise en place d'une base de données mais ensuite la recherche dans cette dernière est bien plus facile et permet plus de choses qu'avec la version JSON.

2. Version JSON

Méthode de lecture des données

Les données JSON sont lues avec le module fs, de Node.js en utilisant *promises* pour gérer tout ce qui est asynchrone. Le fichier complet étant chargé en mémoire au démarrage du serveur, cela permet une lecture très rapide.

Mais la méthode consomme beaucoup de mémoire quand le volume de données est important, ce qui est le cas ici avec un fichier JSON de plus de 30 000 lignes.

Les données gardent la même structure que le fichier JSON d'origine, avec des objets imbriqués. Cela facilite le développement, mais rend les recherches et les mises à jour plus complexes.

Routes implémentées

- **GET /heroes** : cela renvoie l'entièreté des données présentes dans le fichier, sans aucun tri.
- **GET /heroes?publisher=Marvel** : cette route renvoie tous les superhéros qui ont comme publieurs Marvel Comics. Ces 2 premières routes sont regroupées ensemble dans une même fonction.

```
app.get('/heroes', (req, res) => {  
  if (req.query.publisher) {  
    const filtered = heroes.filter(h =>
```

```

        h.biography &&
        h.biography.publisher &&
        h.biography.publisher.toLowerCase().includes(req.query.publisher.toLowerCase())
    );
    return res.json(filtered);
}
res.json(heroes);
});

```

- **GET /heroes/search?q=man** : ici, on va rechercher tous les superhéros qui ont man dans leur nom, comme Spider-man, mais ça peut aussi bien être au milieu du nom comme Black Manta.

```

app.get('/heroes/search', (req, res) => {
    if (!req.query.q) return res.status(400).json({ message: 'Paramètre de recherche manquant' });
    const query = req.query.q.toLowerCase();
    const results = heroes.filter(h =>
        h.name.toLowerCase().includes(query)
    );
    res.json(results);
});

```

- **GET /heroes/:id** : cette route permet de récupérer le superhéros dont l'identifiant a été rentré en paramètre.

```

app.get('/heroes/:id', (req, res) => {
    const hero = heroes.find(h => h.id === parseInt(req.params.id));
    if (!hero) {
        return res.status(404).json({ message: 'Héros non trouvé' });
    }
    res.json(hero);
});

```

- **POST /heroes** : avec ceci, on peut ajouter un nouvel héros au fichier.

```

app.post('/heroes', (req, res) => {
    const newHero = {
        id: heroes.length > 0 ? Math.max(...heroes.map(h => h.id)) + 1 : 1,
        ...req.body
    };
    heroes.push(newHero);
    res.status(201).json(newHero);
});

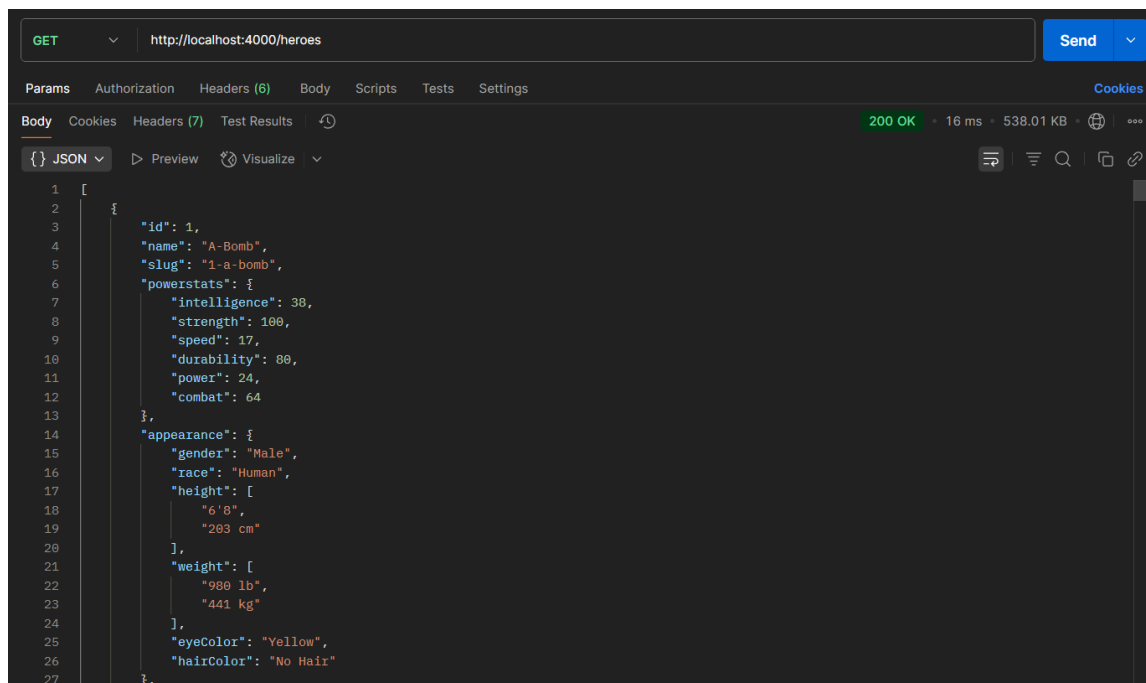
```

- **DELETE /heroes/:id** : cela nous permet de supprimer un héros associé à un ID

```
app.delete('/heroes/:id', (req, res) => {
  const heroIndex = heroes.findIndex(h => h.id === parseInt(req.params.id));
  if (heroIndex === -1) {
    return res.status(404).json({ message: 'Héros non trouvé' });
  }
  heroes.splice(heroIndex, 1);
  res.status(204).send();
});
```

Tests réalisés

- **GET /heroes**



The screenshot shows a web browser interface with a GET request to `http://localhost:4000/heroes`. The response is a 200 OK status with a 16 ms response time and a 538.01 KB body. The response body is displayed in JSON format, showing an array of hero objects. The first hero object is detailed, showing attributes like `id`, `name`, `slug`, `powerstats`, and `appearance`.

```
{
  "id": 1,
  "name": "A-Bomb",
  "slug": "1-a-bomb",
  "powerstats": {
    "intelligence": 38,
    "strength": 100,
    "speed": 17,
    "durability": 80,
    "power": 24,
    "combat": 64
  },
  "appearance": {
    "gender": "Male",
    "race": "Human",
    "height": [
      "6'8",
      "203 cm"
    ],
    "weight": [
      "980 lb",
      "441 kg"
    ],
    "eyeColor": "Yellow",
    "hairColor": "No Hair"
  }
},
```

- **GET /heroes?publisher=Marvel**

```
GET http://localhost:4000/heroes?publisher=Marvel
200 OK 8 ms 274.08 KB

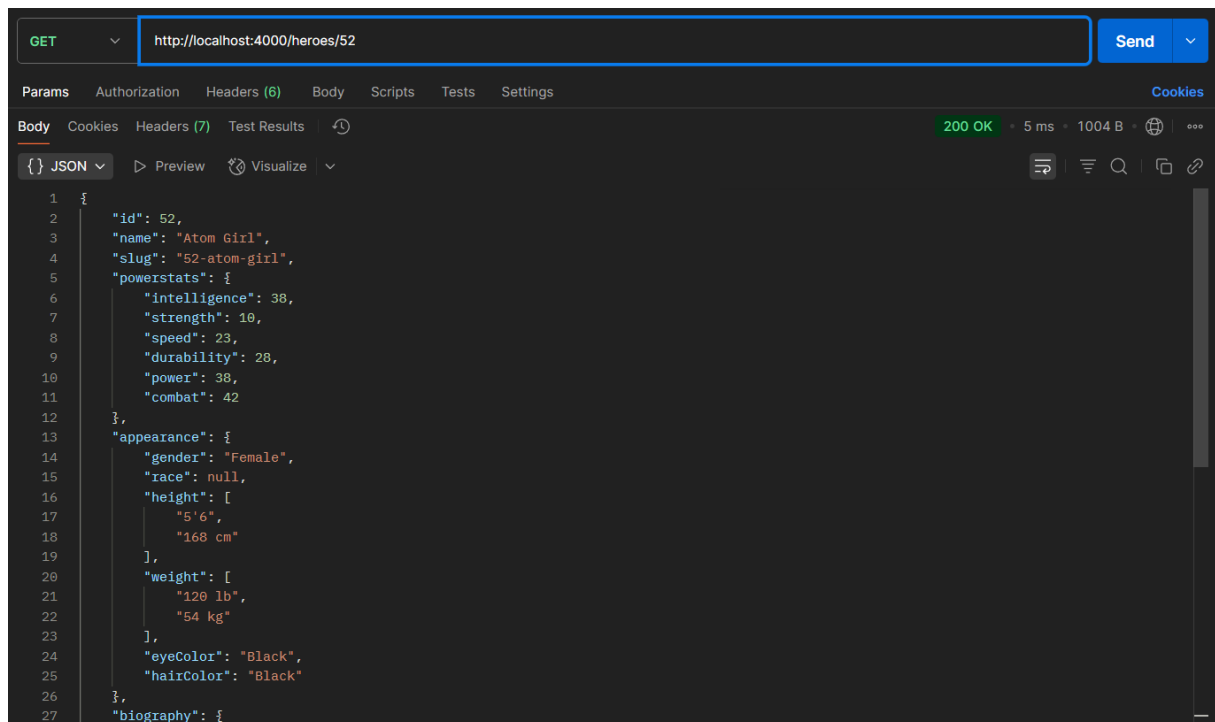
{
  "id": 1,
  "name": "Abraxas",
  "slug": "abraxas",
  "publisher": "Marvel Comics",
  "alignment": "bad",
  "biography": {
    "fullName": "Abraxas",
    "alterEgos": "No alter egos found.",
    "aliases": [
      "-"
    ],
    "placeOfBirth": "Within Eternity",
    "firstAppearance": "Fantastic Four Annual #2001",
    "publisher": "Marvel Comics",
    "alignment": "bad"
  },
  "work": {
    "occupation": "Dimensional destroyer",
    "base": "-"
  },
  "connections": {
    "groupAffiliation": "Cosmic Beings",
    "relatives": "Eternity (\\\"Father\\\")"
  },
  "images": {
    "xs": "xs/5-abraxas.jpg",
    "sm": "sm/5-abraxas.jpg",
    "md": "md/5-abraxas.jpg",
    "lg": "lg/5-abraxas.jpg"
  }
}
```

- GET /heroes/search?q=man

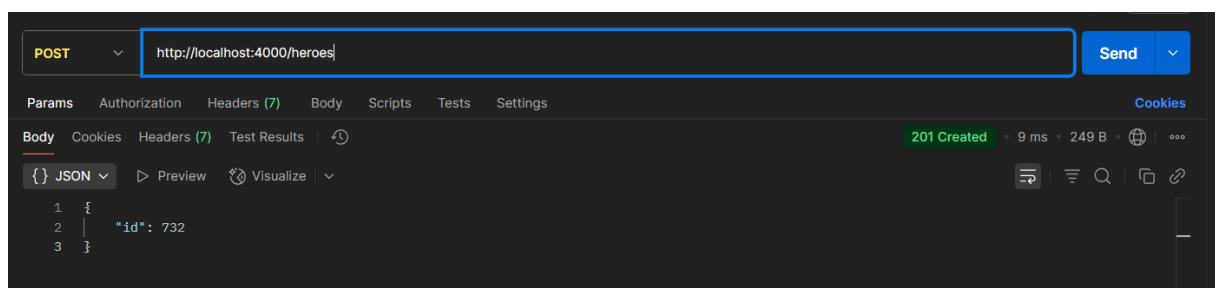
```
GET http://localhost:4000/heroes/search?q=man
200 OK 5 ms 48.02 KB

[
  {
    "id": 6,
    "name": "Absorbing Man",
    "slug": "6-absorbing-man",
    "powerstats": {
      "intelligence": 38,
      "strength": 80,
      "speed": 25,
      "durability": 100,
      "power": 98,
      "combat": 64
    },
    "appearance": {
      "gender": "Male",
      "race": "Human",
      "height": [
        "6'4",
        "193 cm"
      ],
      "weight": [
        "270 lb",
        "122 kg"
      ],
      "eyeColor": "Blue",
      "hairColor": "No Hair"
    }
  }
]
```

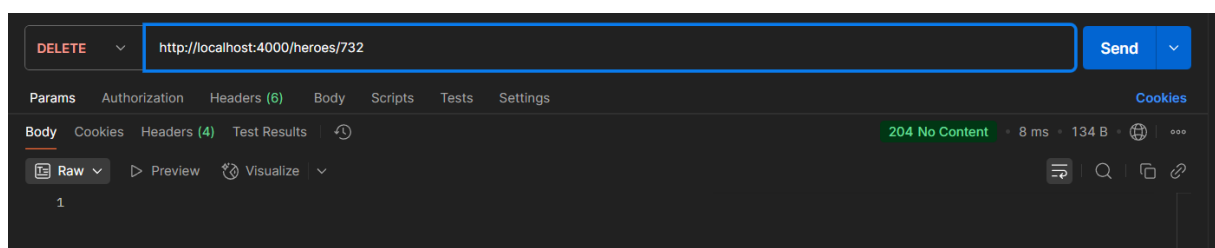
- GET /heroes/:id



- POST /heroes



- DELETE /heroes/:id



Limites constatées

La version JSON a révélé des faiblesses lors de tests plus poussés, et cela notamment lié à l'absence de persistance des modifications, vu que tout changement était perdu au redémarrage du serveur.

3. Version SQLite

Présentation de la base

Voici les commandes pour la création de la base de données :

```
const Database = require('better-sqlite3');
const db = new Database('superheros.db');
db.exec(`CREATE TABLE IF NOT EXISTS heroes (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  name TEXT NOT NULL,
  publisher TEXT,
  gender TEXT,
  race TEXT,
  power TEXT,
  alignment TEXT,
  height_cm INTEGER,
  weight_kg INTEGER,
  createdAt TEXT
);`);
module.exports = db;
```

On peut voir ici qu'on a réduit énormément la quantité d'informations en se limitant à l'essentiel.

Méthode d'import initial

On va importer les données du JSON dans notre base de données grâce à ce script :

```
const fs = require('fs');
const data = JSON.parse(fs.readFileSync('./SuperHerosComplet.json', 'utf-8'));
const insert = db.prepare(`INSERT INTO heroes (name, publisher, gender, race,
power, alignment, height_cm, weight_kg, createdAt)
VALUES (@name, @publisher, @gender, @race, @power, @alignment, @height_cm,
@weight_kg, @createdAt)`);

const count = db.prepare('SELECT COUNT(*) as total FROM heroes').get();
if (count.total === 0) {
  const now = new Date().toISOString();
  for (const hero of data.superheros) {
    insert.run({
      name: hero.name,
      publisher: hero.biography.publisher,
      gender: hero.appearance.gender,
      race: hero.appearance.race,
      power: JSON.stringify(hero.powerstats),
      createdAt: now
    });
  }
}
```



```

    alignment: hero.biography.alignment,
    height_cm: parseInt(hero.appearance.height[1].split(' ')[0]),
    weight_kg: parseInt(hero.appearance.weight[1].split(' ')[0]),
    createdAt: now
  });
}
console.log('Données initiales importées.');
```

Dès lors, on peut toutes sortes d'opérations sur notre base de données.

Routes implémentées

- **GET /heroes** : comme pour JSON, on récupère l'entièreté des données sans aucun filtre.
- **GET /heroes?publisher=DC** : ici on va récupérer les données du superhéros dont le publieur est DC comics. Comme pour la version JSON, ces 2 routes sont sous la même fonction.

```

app.get('/heroes', (req, res) => {
  if (req.query.publisher) {
    const publisher = `%${req.query.publisher}%`;
    const heroes = db.prepare('SELECT * FROM heroes WHERE publisher
LIKE ?').all(publisher);
    return res.json(heroes);
  }
  const heroes = db.prepare('SELECT * FROM heroes').all();
  res.json(heroes);
});
```

- **GET /heroes/search?q=bat** : On va ici recherche l'entièreté des héros donc le nom comporte en son sein la suite de lettres *bat*.

```

app.get('/heroes/search', (req, res) => {
  if (!req.query.q) return res.status(400).json({ message: 'Paramètre de
recherche manquant' });
  const q = req.query.q.toLowerCase();
  const heroes = db.prepare('SELECT * FROM heroes WHERE LOWER(name)
LIKE ?').all(`%${q}%`);
  res.json(heroes);
});
```

- **GET /heroes/sorted?by=height_cm** : cette route permet de trier les héros en fonction de leur taille, mais on peut aussi mettre plusieurs paramètres (partie Bonus).

```
app.get('/heroes/sorted', (req, res) => {
  const validColumns = ['id', 'name', 'publisher', 'height_cm',
    'weight_kg']; // Ajout de 'id'
  const sortBy = req.query.by ? req.query.by.split(',') : [];

  const invalidColumns = sortBy.filter(col => !validColumns.includes(col));
  if (invalidColumns.length > 0) {
    return res.status(400).json({ message: 'Paramètre de tri invalide' });
  }

  const orderBy = sortBy.length > 0 ? sortBy.join(', ') : 'id';
  const query = `SELECT * FROM heroes ORDER BY ${orderBy}`;

  try {
    const heroes = db.prepare(query).all();
    res.json(heroes);
  } catch (err) {
    res.status(422).json({ message: 'Erreur dans la requête de tri' });
  }
});
```

- **GET /heroes/:id** : comme pour la version JSON, cela nous retourne le héros associé à l'ID.

```
app.get('/heroes/:id', (req, res) => {
  const hero = db.prepare('SELECT * FROM heroes WHERE id
    = ?').get(req.params.id);
  if (!hero) return res.status(404).json({ message: 'Héros non trouvé' });
  res.json(hero);
});
```

- **POST /heroes** : cela nous permet de rajouter une ligne dans notre base de données et de rajouter un héros de ce fait.

```
app.post('/heroes', (req, res) => {
  const { name, publisher, gender, race, power, alignment, height_cm,
    weight_kg } = req.body;

  if (!name) return res.status(400).json({ message: 'Le nom est
    obligatoire' });

  const now = new Date().toISOString();
  const stmt = db.prepare(`
```

```

        INSERT INTO heroes (name, publisher, gender, race, power, alignment,
height_cm, weight_kg, createdAt)
        VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)
    `);

    const info = stmt.run(name, publisher, gender, race, power, alignment,
height_cm, weight_kg, now);
    const newHero = db.prepare('SELECT * FROM heroes WHERE id
= ?').get(info.lastInsertRowid);

    res.status(201).json(newHero);
});

```

- **DELETE /heroes/:id** : et pour finir ce DELETE nous permet de supprimer un héros en renseignant son ID.

```

app.delete('/heroes/:id', (req, res) => {
    const stmt = db.prepare('DELETE FROM heroes WHERE id = ?');
    const result = stmt.run(req.params.id);

    if (result.changes === 0) {
        return res.status(404).json({ message: 'Héros non trouvé' });
    }

    res.status(204).end();
});

```

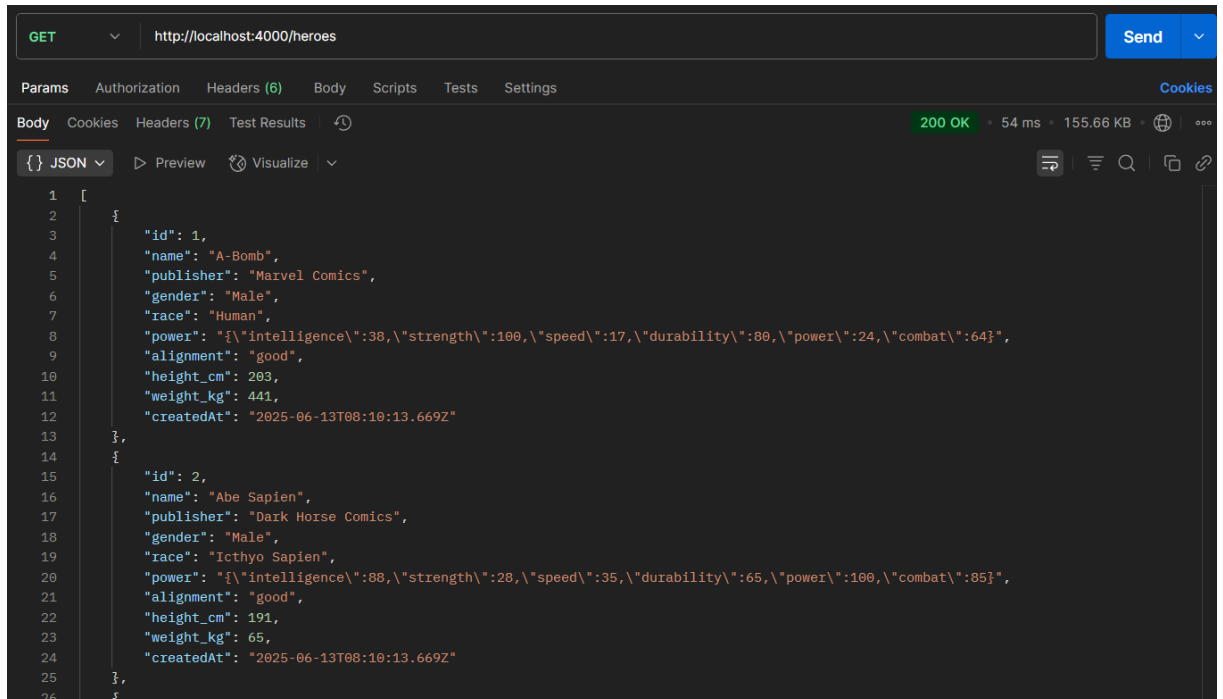
Gestion des erreurs

Différentes erreurs ainsi que leurs gestions ont été implémentés dans le code :

- 400 : Paramètre manquant ou invalide.
- 404 : Ressource non trouvée
- 422 : Données non traitables

Résultats des tests

- GET /heroes

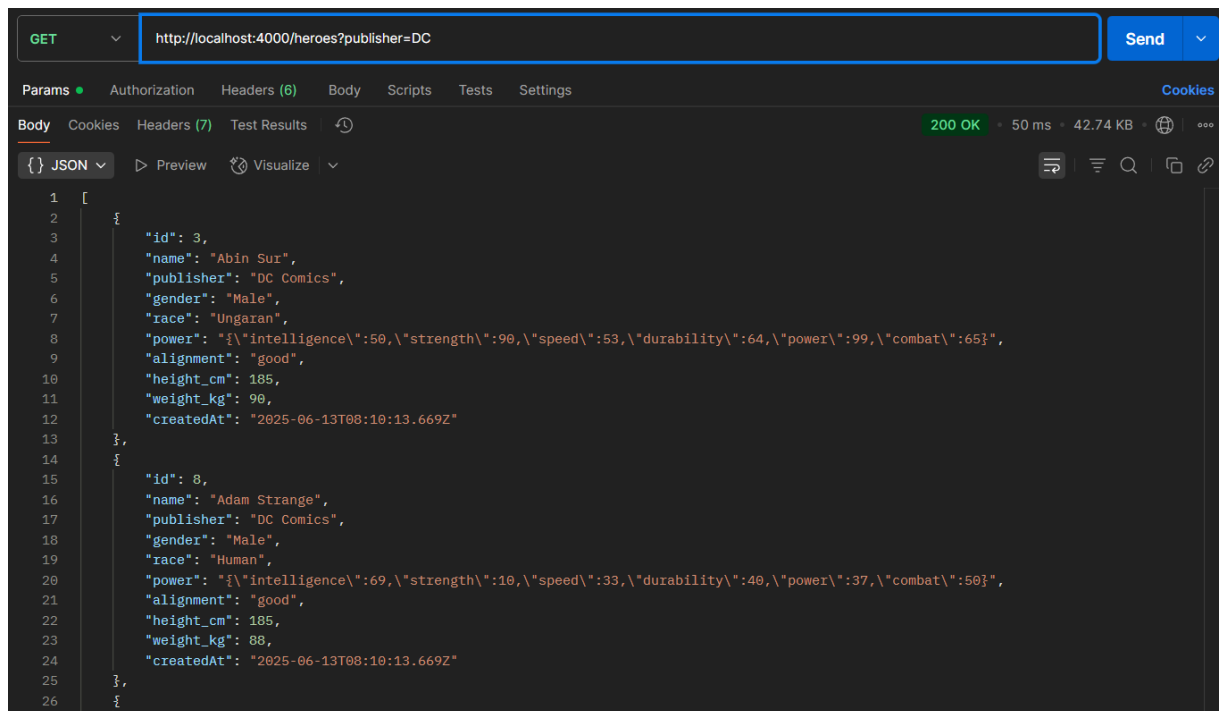


GET http://localhost:4000/heroes

200 OK • 54 ms • 155.66 KB

```
{
  "id": 1,
  "name": "A-Bomb",
  "publisher": "Marvel Comics",
  "gender": "Male",
  "race": "Human",
  "power": {
    "intelligence": 38,
    "strength": 100,
    "speed": 17,
    "durability": 80,
    "power": 24,
    "combat": 64
  },
  "alignment": "good",
  "height_cm": 203,
  "weight_kg": 441,
  "createdAt": "2025-06-13T08:10:13.669Z"
},
{
  "id": 2,
  "name": "Abe Sapien",
  "publisher": "Dark Horse Comics",
  "gender": "Male",
  "race": "Ichthyo Sapien",
  "power": {
    "intelligence": 88,
    "strength": 28,
    "speed": 35,
    "durability": 65,
    "power": 100,
    "combat": 85
  },
  "alignment": "good",
  "height_cm": 191,
  "weight_kg": 65,
  "createdAt": "2025-06-13T08:10:13.669Z"
}
```

- GET /heroes?publisher=DC

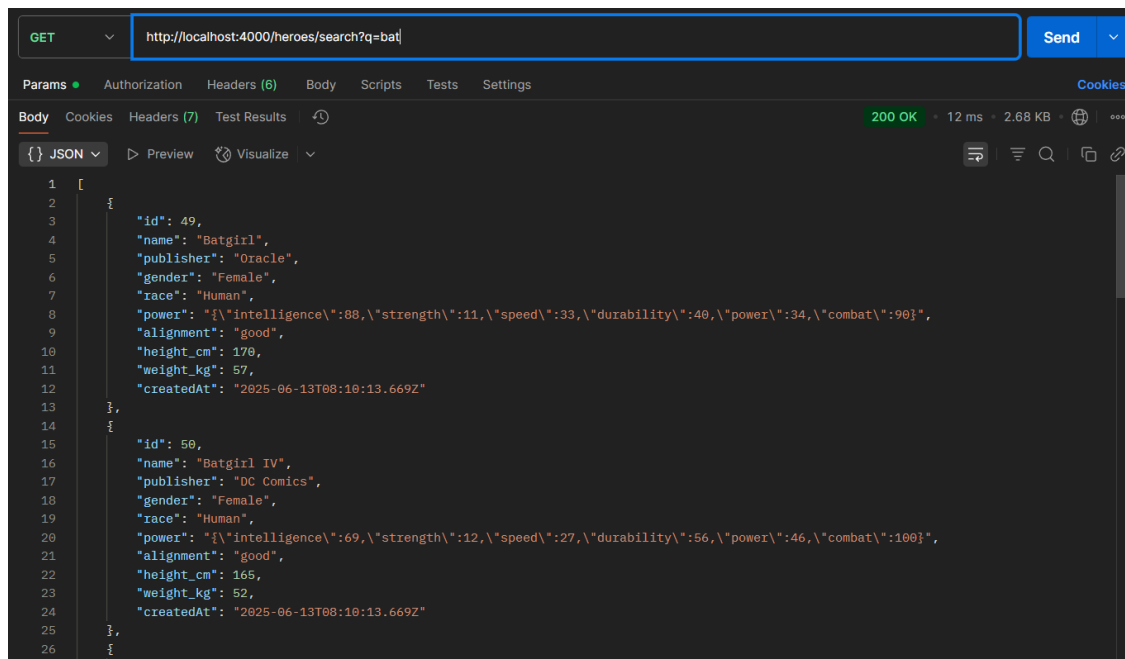


GET http://localhost:4000/heroes?publisher=DC

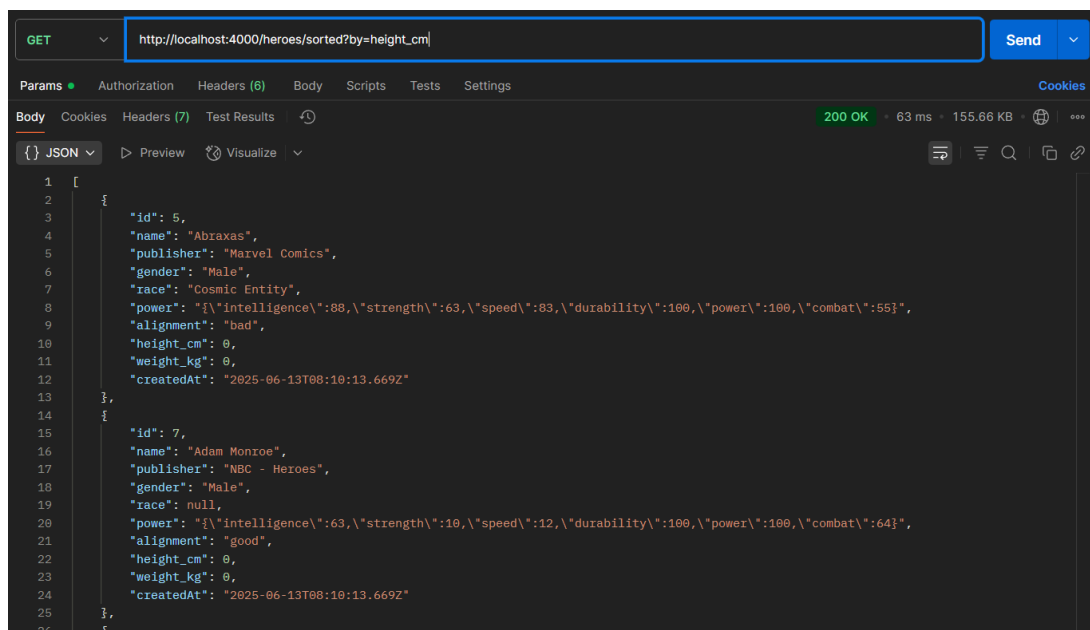
200 OK • 50 ms • 42.74 KB

```
{
  "id": 3,
  "name": "Abin Sur",
  "publisher": "DC Comics",
  "gender": "Male",
  "race": "Ungaran",
  "power": {
    "intelligence": 50,
    "strength": 90,
    "speed": 53,
    "durability": 64,
    "power": 99,
    "combat": 65
  },
  "alignment": "good",
  "height_cm": 185,
  "weight_kg": 90,
  "createdAt": "2025-06-13T08:10:13.669Z"
},
{
  "id": 8,
  "name": "Adam Strange",
  "publisher": "DC Comics",
  "gender": "Male",
  "race": "Human",
  "power": {
    "intelligence": 69,
    "strength": 10,
    "speed": 33,
    "durability": 40,
    "power": 37,
    "combat": 50
  },
  "alignment": "good",
  "height_cm": 185,
  "weight_kg": 88,
  "createdAt": "2025-06-13T08:10:13.669Z"
}
```

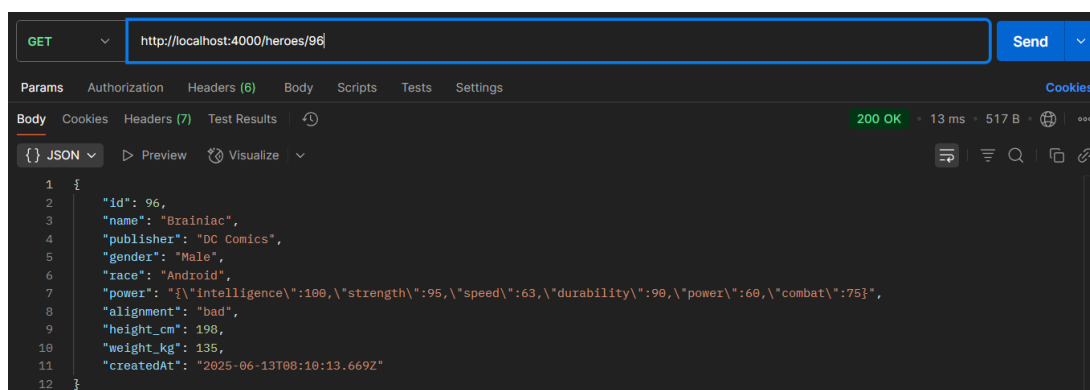
- GET /heroes/search?q=bat



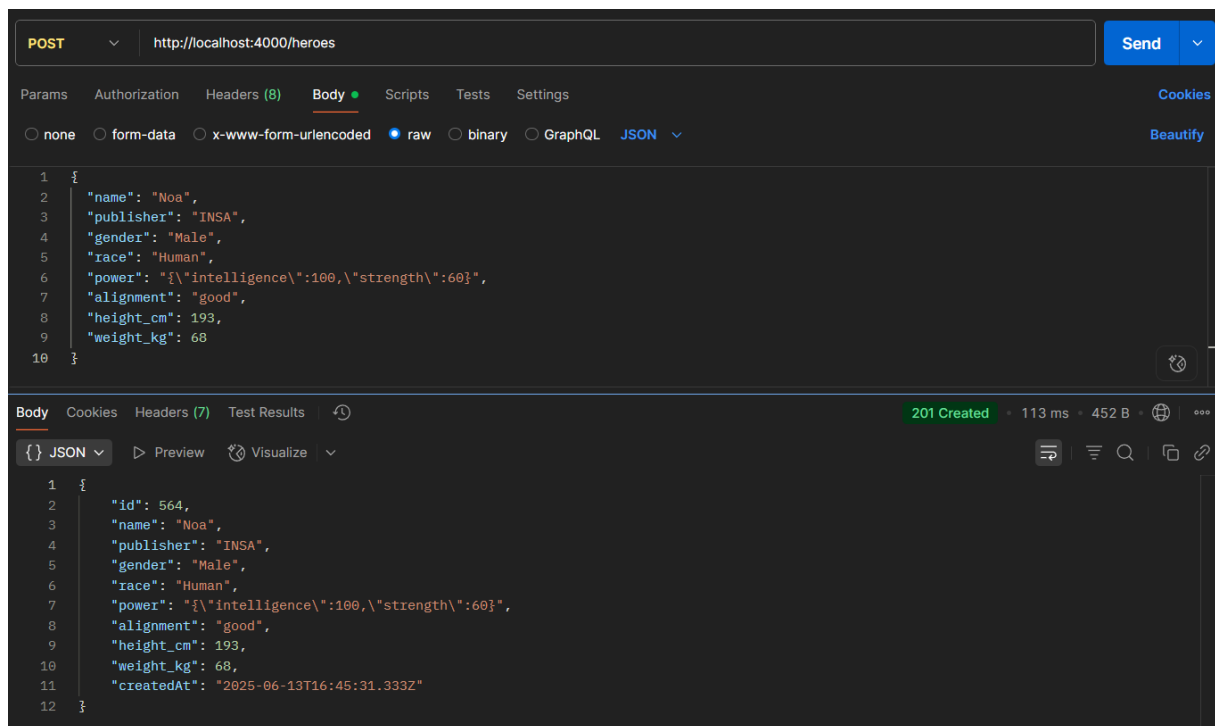
- GET /heroes/sorted?by=height_cm



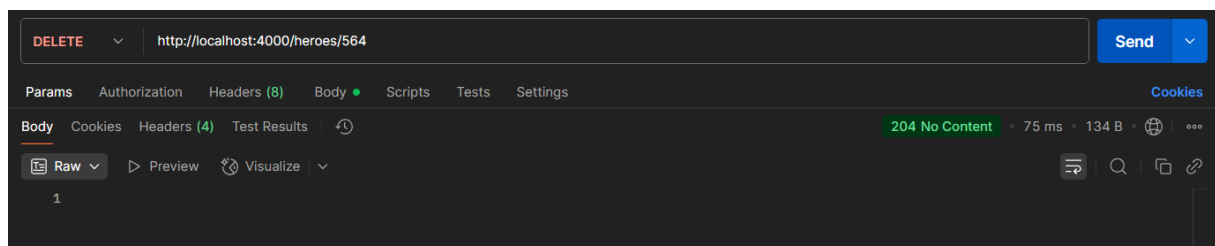
- GET /heroes/:id



- POST /heroes



- DELETE /heroes/:id



4. Bonus réalisés

Il y avait 3 bonus de disponibles :

- Export CSV
- Tri multichamps
- Statistiques

Voici les codes pour ces derniers

- Export CSV :

```
app.get('/heroes/export', (req, res) => {
  let query = 'SELECT * FROM heroes';
  const params = [];

  if (req.query.publisher) {
    query += ' WHERE publisher LIKE ?';
  }
});
```

```

    params.push(`%${req.query.publisher}%`);
  }

  const heroes = db.prepare(query).all(...params);

  // Convertir en CSV
  let csv =
'id,name,publisher,gender,race,power,alignment,height_cm,weight_kg,createdAt\n';
  heroes.forEach(hero => {
    csv +=
`${hero.id},${hero.name},${hero.publisher},${hero.gender},${hero.race},${hero.
power},${hero.alignment},${hero.height_cm},${hero.weight_kg},${hero.createdAt}
\n`;
  });

  res.header('Content-Type', 'text/csv');
  res.attachment('heroes.csv');
  res.send(csv);
});

```

Test :

The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** http://localhost:4000/heroes/export?publisher=DC
- Status:** 200 OK
- Time:** 15 ms
- Size:** 24.16 KB
- Body:** A CSV file named 'heroes.csv' containing 14 rows of data. The first row is the header: 'id,name,publisher,gender,race,power,alignment,height_cm,weight_kg,createdAt'. The subsequent rows contain data for various DC characters, including Abin Sur, Adam Strange, Alan Scott, Alfred Pennyworth, Amazo, Animal Man, Anti-Monitor, Aquababy, Aqualad, Aquaman, Atlas, Atom Girl, and Atom II.

- Tri multichamps

```

app.get('/heroes/sorted', (req, res) => {
  const validColumns = ['id', 'name', 'publisher', 'height_cm',
'weight_kg']; // Ajout de 'id'
  const sortBy = req.query.by ? req.query.by.split(',') : [];

```

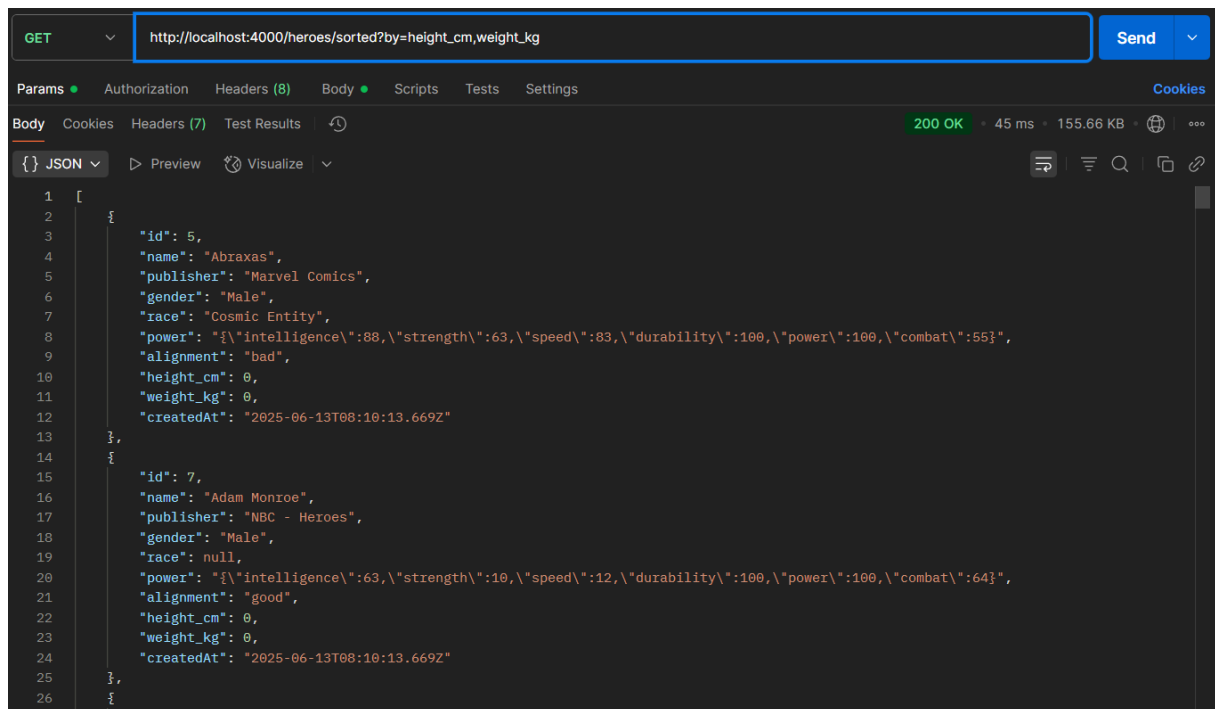
```

const invalidColumns = sortBy.filter(col => !validColumns.includes(col));
if (invalidColumns.length > 0) {
  return res.status(400).json({ message: 'Paramètre de tri invalide' });
}

const orderBy = sortBy.length > 0 ? sortBy.join(', ') : 'id';
const query = `SELECT * FROM heroes ORDER BY ${orderBy}`;

try {
  const heroes = db.prepare(query).all();
  res.json(heroes);
} catch (err) {
  res.status(422).json({ message: 'Erreur dans la requête de tri' });
}
});

```



- Statistiques

```

app.get('/heroes/stats', (req, res) => {
  const stats = {
    countByPublisher: db.prepare('SELECT publisher, COUNT(*) as count FROM
heroes GROUP BY publisher').all(),
    avgHeight: db.prepare('SELECT AVG(height_cm) as avg FROM
heroes').get().avg,
    avgWeight: db.prepare('SELECT AVG(weight_kg) as avg FROM
heroes').get().avg,

```

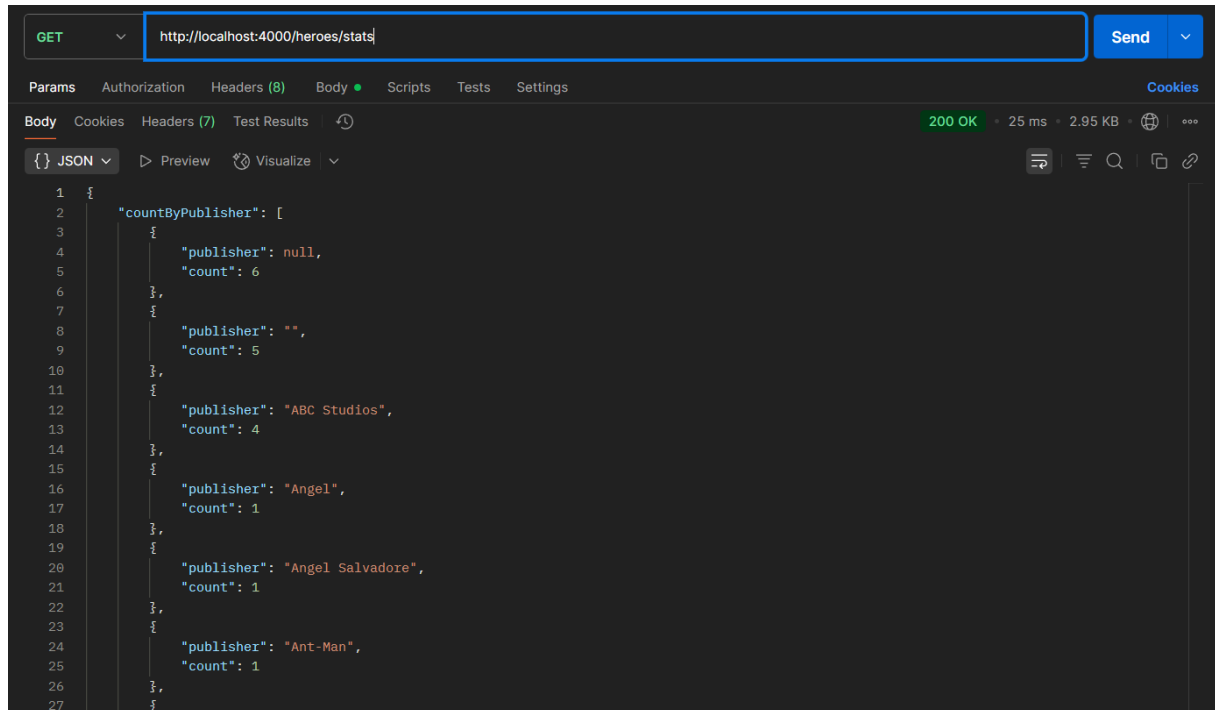


```

        alignmentDistribution: db.prepare('SELECT alignment, COUNT(*) as count
FROM heroes GROUP BY alignment').all()
    };

    res.json(stats);
});

```



5. Analyse critique

Avantages et inconvénients

Les avantages de la version JSON est que c'est très simple à comprendre et à faire mais coté inconvénients, niveau persistance on en a une absence, car on est sur la mémoire volatile.

Du coté de la version SQLite, on a aucun problème de persistance vu qu'on est sur une base de données, on peut cibler nos requêtes ce qui est bien plus efficace et niveau fonctionnalités, c'est bien plus intéressant que la version JSON.

Difficultés rencontrées et solutions apportées

Du coté des difficultés, les principales que j'ai pu avoir son lié à l'écriture du code et le fait de faire fonctionner les commandes GET ou POST et à force de modification et d'aide avec GitHub Copilot, les problèmes se sont régler.

6. Conclusion

Bilan du travail accompli

L'ensemble des demandes ont été réalisé, que ce soit la mise en place des APIs, la configuration de la base de données SQLite, mais également l'entièreté des routes.

Ce que vous avez appris / retenu

Lors de ce projet, j'ai appris de nombreuses choses sur la mise en place d'API mais aussi sur la manière de lier une base de données à cela.

GitHub

Voici le [lien](#) du repo GitHub sur lequel tout est présent.