CSE4701

Project 2, Part 1

Mike Medved

November 12th, 2023

Table of Contents

Introduction

In the following assignment I am using Prisma running on TypeScript to interface with the MongoDB database. The database is hosted locally on a Docker container as a replica set in order to allow Prisma to work. Below is a snippet from the script used to populate the database in accordance with the instructions provided for the assignment.

```
// Enumerate all JSON files in the data directory and get their contents.
let files = readdirSync('./data')
    .filter(file => file.endsWith('.json'))
    .map(file => `./data/${file}`)
    .map(path => readFileSync(path, 'utf-8'))
    .map(json => JSON.parse(json))
    .map(payload => payload as BookJson);

// Enumerate all JPG files in the data directory and get their binary contents.
let images = readdirSync('./data')
    .filter(file => file.endsWith('.jpg'))
    .map(file => `./data/${file}`)
    .map(path => ({
        bookId: path.split(/(B d+)/g)[1],
        blob: readFileSync(path)
    }));

// Coalesce the books and images into a map of book ID to book and image.
let bookMap: BookMap = files.reduce((acc, book) => {
    acc[book.id] = { book };

    let image = images.find(image => image.bookId === book.id);
    if (image) acc[book.id].image = image.blob;

    return acc;
}, {} as BookMap);

// Create a database record for each book in the above map.
let records = await Promise.all(
    Object.keys(bookMap).map(async bookId => {
        let { book, imageBlob } = bookMap[bookId];
        let { author, ..rest } = book;

        return await client.book.create({
            data: {
                name: book.bookname,
                book_id: bookId,
                image: imageBlob,
                description: rest,
            }
        });
    })
);
```

Note: All table-like result views in the rest of this assignment were achieved by using the built-in *console.table* logging mechanism in JavaScript. They are not necessarily representative of how MongoDB stores the data or anything of the sort, merely for more readable output. Also, the *(index)* column in said table views is a result of the *console.table* function and is not actually an attribute that exists on the documents in the database.

Question 3, Creating the NoSQL Database

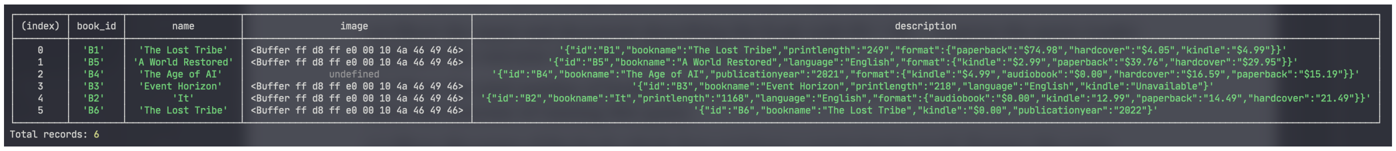
Part C + D, Data Insertion Results

We will use the *findMany* command without any filtering parameters to enumerate all entries in the database. Additionally, we will then use the *count* command to get the number of records in the database. The below code was used to execute these queries against the database.

```
let books = await client.book.findMany({});
console.table(books);

let count = await client.book.count({});
console.log('Total records:', count);
```

Below are the results of the above code being executed, note that the image binary data was truncated, and the description JSON data was stringified so it is readable. You may need to zoom in as the image is quite wide.



```
┌─────────┬─────────┬─────────────────────┬──────────────────────────────────────────┬──────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────┐
│ (index) │ book_id │        name         │                  image                   │                                                      description                                                             │
├─────────┼─────────┼─────────────────────┼──────────────────────────────────────────┼──────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────┤
│    0    │  'B1'   │  'The Lost Tribe'   │ <Buffer ff d8 ff e0 00 10 4a 46 49 46>   │       '{"id":"B1","bookname":"The Lost Tribe","printlength":"249","format":{"paperback":"$74.98","hardcover":"$4.05","kindle":"$4.99"}}'       │
│    1    │  'B5'   │ 'A World Restored'  │ <Buffer ff d8 ff e0 00 10 4a 46 49 46>   │     '{"id":"B5","bookname":"A World Restored","language":"English","format":{"kindle":"$2.99","paperback":"$39.76","hardcover":"$29.95"}}'     │
│    2    │  'B4'   │  'The Age of AI'    │                undefined                 │ '{"id":"B4","bookname":"The Age of AI","publicationyear":"2021","format":{"kindle":"$4.99","audiobook":"$0.00","hardcover":"$16.59","paperback":"$15.19"}}' │
│    3    │  'B3'   │  'Event Horizon'    │ <Buffer ff d8 ff e0 00 10 4a 46 49 46>   │          '{"id":"B3","bookname":"Event Horizon","printlength":"218","language":"English","kindle":"Unavailable"}'          │
│    4    │  'B2'   │        'It'         │ <Buffer ff d8 ff e0 00 10 4a 46 49 46>   │ '{"id":"B2","bookname":"It","printlength":"1168","language":"English","format":{"audiobook":"$0.00","kindle":"12.99","paperback":"14.49","hardcover":"21.49"}}' │
│    5    │  'B6'   │  'The Lost Tribe'   │ <Buffer ff d8 ff e0 00 10 4a 46 49 46>   │                 '{"id":"B6","bookname":"The Lost Tribe","kindle":"$0.00","publicationyear":"2022"}'                 │
└─────────┴─────────┴─────────────────────┴──────────────────────────────────────────┴──────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────┘
Total records: 6
```

Figure 1: Results for Part C and D

Question 4, Querying the NoSQL Database

Part A, Retrieving a Single Document By ID

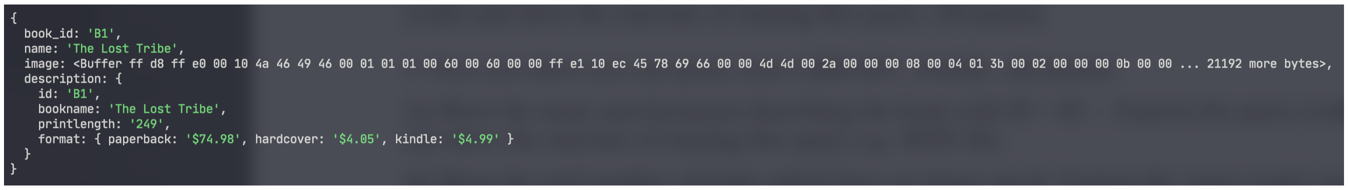
Show the name and document describing the book with ID = B1.

In order to retrieve a single document by ID, we will use the *findUnique* command with the ID as a parameter. The below code was used to execute this query against the database.

```
let book = await client.book.findUnique({
    where: { book_id: 'B1' }
});

console.log(book);
```

Below is the result of the above query. Note again that the binary image data has been truncated.



```
{
  book_id: 'B1',
  name: 'The Lost Tribe',
  image: <Buffer ff d8 ff e0 00 10 4a 46 49 46 00 01 01 01 00 60 00 60 00 00 ff e1 10 ec 45 78 69 66 00 00 4d 4d 00 2a 00 00 00 08 00 04 01 3b 00 02 00 00 00 0b 00 00 ... 21192 more bytes>,
  description: {
    id: 'B1',
    bookname: 'The Lost Tribe',
    printlength: '249',
    format: { paperback: '$74.98', hardcover: '$4.05', kindle: '$4.99' }
  }
}
```

Figure 2: Results for Part A

Part B, Retrieving The Number of Filtered Documents

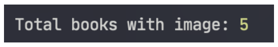
Show the total number of books which have an image saved.

In order to retrieve the number of filtered documents, we will use the *count* command with a filter parameter. The below code was used to execute this query against the database.

```
let count = await client.book.count({
    where: {
        image: { not: null }
    }
});

console.log('Total books with image:', count);
```

Below is the output of the above code:



```
Total books with image: 5
```

Figure 3: Results for Part B

Part C, Retrieving a Document Without a Field

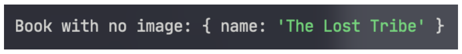
Show the name of the book which does not have a cover image saved.

In order to write this query, we will use the *findFirst* command with a filter parameter checking for a document wherein the *image* field is equal to undefined. Additionally, we will set the *select* field to only output the name. The below code was used to execute this query against the database.

```
let book = await client.book.findFirst({
    where: { image: undefined },
    select: { name: true }
});

console.log('Book with no image:', book);
```

Below is the output of the above code:



```
Book with no image: { name: 'The Lost Tribe' }
```

Figure 4: Results for Part C

Part D, Printing All Values of a Field in the Database

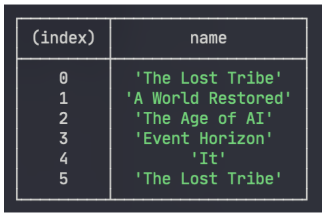
Print out all the book names in LIBRARY_NoSQL

In order to write this query, we will use the *findMany* command with a select parameter to only output the name field. The below code was used to execute this query against the database.

```
let books = await client.book.findMany({
    select: { name: true }
});

console.table(books);
```

Below is the output of the above code (note that the index column is a byproduct of the *console.table* command, and does not actually exist in the database):



```
┌─────────┬─────────────────────┐
│ (index) │        name         │
├─────────┼─────────────────────┤
│    0    │  'The Lost Tribe'   │
│    1    │ 'A World Restored'  │
│    2    │  'The Age of AI'    │
│    3    │  'Event Horizon'    │
│    4    │        'It'         │
│    5    │  'The Lost Tribe'   │
└─────────┴─────────────────────┘
```

Figure 5: Results for Part D

Part E, Retrieving Documents With a Field

Print out the name and ID of the books available in kindle, and the price of the kindle version for each.

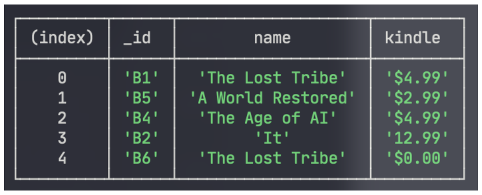
In order to be able to execute this query with Prisma, I opted to use the *findRaw* command as the nested JSON path filtering was not working for me. Thus, I used the below code snippet to query the database. Below, you can see I abstracted out the creation of the *$and* block as I reused it for both of the possible Kindle paths (in the root, and under format). Additionally, I used the projection block to filter only the attributes I wanted to return as per the problem. In this case, I also included both the root and format paths for the Kindle attribute.

```
const compileConditionalRule = (path: string) => ({
    $and: [
        { [path]: { $exists: true } },
        { [path]: { $ne: 'Unavailable' } },
    ]
});

let books = await client.book.findRaw({
    filter: {
        $or: [
            compileConditionalRule('description.format.kindle'),
            compileConditionalRule('description.kindle'),
        ]
    },
    options: {
        projection: {
            id: true,
            name: true,
            'description.kindle': true,
            'description.format.kindle': true,
        }
    }
})

console.table(books);
```

Below is the output of the above code:



```
┌─────────┬──────┬───────────────────┬─────────┐
│ (index) │ _id  │       name        │ kindle  │
├─────────┼──────┼───────────────────┼─────────┤
│    0    │ 'B1' │ 'The Lost Tribe'  │ '$4.99' │
│    1    │ 'B5' │'A World Restored' │ '$2.99' │
│    2    │ 'B4' │  'The Age of AI'  │ '$4.99' │
│    3    │ 'B2' │       'It'        │ '12.99' │
│    4    │ 'B6' │ 'The Lost Tribe'  │ '$0.00' │
└─────────┴──────┴───────────────────┴─────────┘
```

Figure 6: Results for Part E

Part F, Retrieving Documents With An Ambiguous Field Value

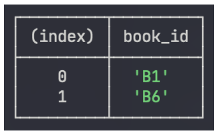
Show the ID of both books named "The Lost Tribe"

In order to write this query, we will use the *findMany* command to retrieve all documents with the name field equalling "The Lost Tribe". Additionally, I will only select the *book_ id* field as per the problem. The below code was used to execute this query against the database.

```
let books = await client.book.findMany({
    where: {
        name: {
            equals: 'The Lost Tribe'
        }
    },
    select: {
        book_id: true
    }
})

console.table(books);
```

Below is the output of the above code:



```
┌─────────┬─────────┐
│ (index) │ book_id │
├─────────┼─────────┤
│    0    │  'B1'   │
│    1    │  'B6'   │
└─────────┴─────────┘
```

Figure 7: Results for Part F