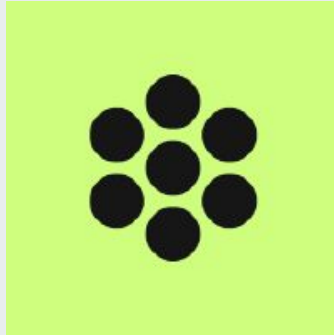




Welcome to Vinyl Vibe!

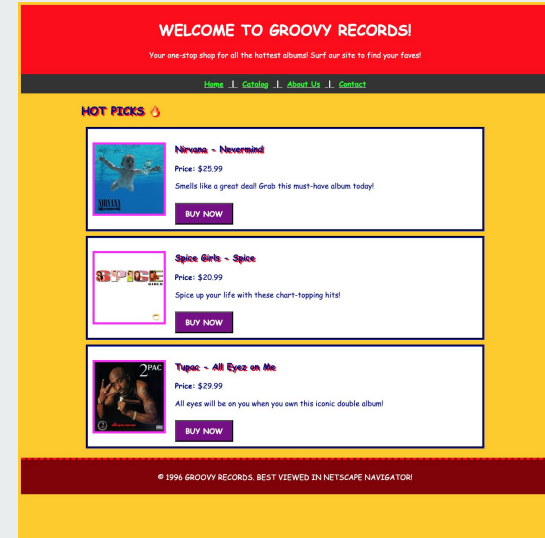




What is Vinyl Vibe?

A bricks and mortar record store owned by retired Roadie Norm Gleeson which required a modern website to replace his 90's hardcoded HTML webpage (best viewed in Netscape Navigator).

Customers from all over the world should be able to purchase vinyl records, turntables, associated accessories and store branded merchandise from Norm's new website.





We elected to divide each section of the Backend up into separate folders that would contain all of the required files for that section, rather than having dedicated 'Models', 'Controllers', 'Middleware', 'Routes' and 'Services' folders.

This helped to compartmentalise how each part of the app would function and work together. It also helped with organisation.

Project Organisation and Directory

```

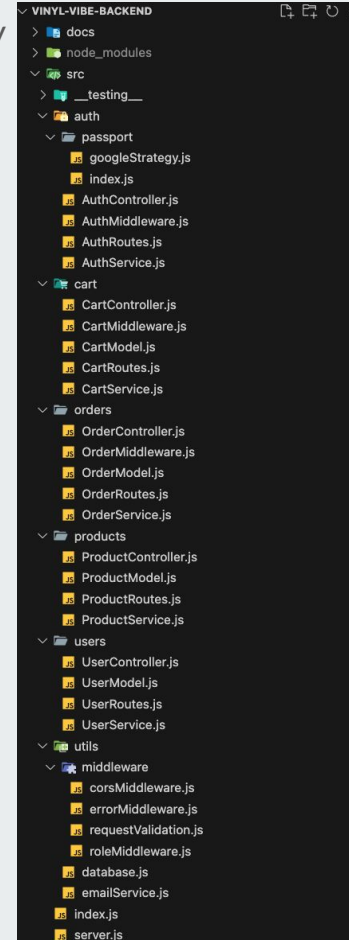
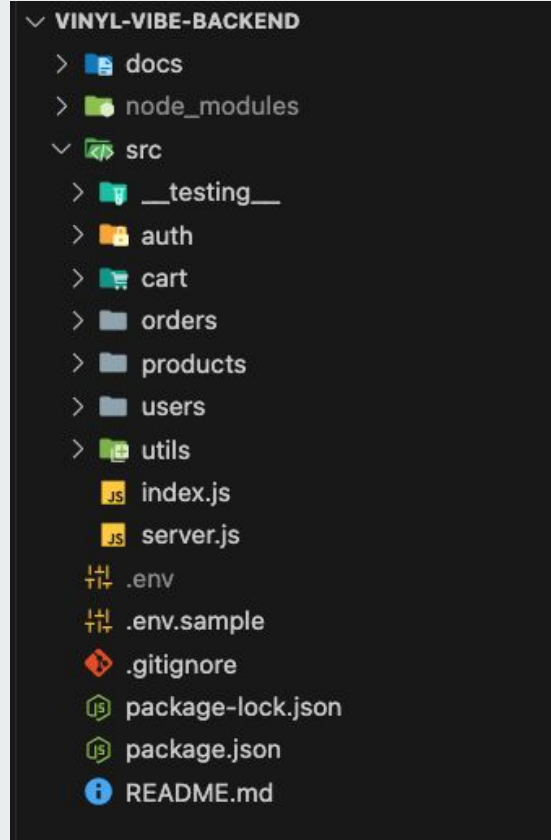
✓ VINYL-VIBE-BACKEND
  > docs
  > node_modules
  ✓ src
    > __testing__
    > auth
    > cart
    > orders
    > products
    > users
    > utils
    index.js
    server.js
    .env
    .env.sample
    .gitignore
    package-lock.json
    package.json
    README.md
```



We elected to divide each section of the Backend up into separate folders that would contain all of the required files for that section, rather than having dedicated 'Models', 'Controllers', 'Middleware', 'Routes' and 'Services' folders.

This helped to compartmentalise how each part of the app would function and work together. It also helped with organisation.

Project Organisation and Directory





R14. dbConnect Function

dbConnect function
in VSCode with
comments.

```
src > utils > database.js > dbConnect
1  const mongoose = require("mongoose");  875.5k (gzipped: 234.3k)
2
3
4  /**
5   * Database connection handler
6   *
7   * Why use environment variables for DB connection?
8   * - Security: Keep credentials out of code
9   * - Flexibility: Different DBs for dev/test/prod
10  * - DevOps best practice: Configuration as environment
11  */
12  async function dbConnect() {
13    // Log connection string (without credentials in production)
14    console.log("Attempting to connect to:", process.env.DATABASE_URL);
15
16    // Fallback to local MongoDB if no DATABASE_URL provided
17    // Why use npm_package_name?
18    // - Creates database named after project
19    // - Prevents test/dev database collisions
20    let databaseUrl = process.env.DATABASE_URL;
21
22    if (!databaseUrl) {
23      databaseUrl = `mongodb://127.0.0.1:27017/${process.env.npm_package_name}`;
24    }
25
26    try {
27      // Connect with Mongoose
28      // Why async/await?
29      // - Cleaner error handling
30      // - Ensures connection before server starts
31      // - Prevents race conditions
32      await mongoose.connect(databaseUrl);
33      console.log("Successfully connected to MongoDB!");
34    } catch (error) {
35      // Log error and rethrow
36      // Why rethrow?
37      // - Allows calling code to handle connection failure
38      // - Prevents server from starting with no database
39      console.error("Database connection error:", error);
40      throw error;
41    }
42
43    module.exports = {
44      dbConnect,
45    };
46  }
```

```
const mongoose = require("mongoose");
async function dbConnect() {
  console.log("Attempting to connect to:", process.env.DATABASE_URL);
  let databaseUrl = process.env.DATABASE_URL;

  if (!databaseUrl) {
    databaseUrl = `mongodb://127.0.0.1:27017/
    ${process.env.npm_package_name}`;
  }
  try {
    await mongoose.connect(databaseUrl);
    console.log("Successfully connected to MongoDB!");
  } catch (error) {
    console.error("Database connection error:", error);
    throw error;
  }
}

module.exports = {
  dbConnect,
};
```

JavaScript

The same dbConnect function with comments
removed for brevity and visibility.

The Database Operation that we use is the connection we make to MongoDB.

The conditional statement is the `if` statement we use to creating a local database when there isn't a DATABASE_URL specified in the environment variables

```
JavaScript
const mongoose = require("mongoose");
async function dbConnect() {
  console.log("Attempting to connect to:", process.env.DATABASE_URL);
  let databaseUrl = process.env.DATABASE_URL;

  if (!databaseUrl) {
    databaseUrl = `mongodb://127.0.0.1:27017/
    ${process.env.npm_package_name}`;
  }

  try {
    await mongoose.connect(databaseUrl);
    console.log("Successfully connected to MongoDB!");
  } catch (error) {
    console.error("Database connection error:", error);
    throw error;
  }
}

module.exports = {
  dbConnect,
};
```

The same dbConnect function with comments removed for brevity and visibility.

The database connection handler we've called is 'mongoose'.

This allows us to use environment variables when we make a connection to our database.

We do this for 3 reasons:

Security: Keep sensitive credentials (e.g., DB URLs) out of source code.

Flexibility: Use different databases for development, testing, and production.

DevOps Best Practice: Manage configuration as environment variables for scalability and portability.

```
JavaScript
const mongoose = require("mongoose");
async function dbConnect() {
  console.log("Attempting to connect to:", process.env.DATABASE_URL);
  let databaseUrl = process.env.DATABASE_URL;

  if (!databaseUrl) {
    databaseUrl = `mongodb://127.0.0.1:27017/`;
    databaseUrl += `${process.env.npm_package_name}`;
  }

  try {
    await mongoose.connect(databaseUrl);
    console.log("Successfully connected to MongoDB!");
  } catch (error) {
    console.error("Database connection error:", error);
    throw error;
  }
}

module.exports = {
  dbConnect,
};
```

The same dbConnect function with comments removed for brevity and visibility.



R14. dbConnect Function

Database Operation

We create an async function called dbConnect (explained in more detail in Error 1).

A console log informs us that we're attempting to connect to process.env.DATABASE_URL.

The line 'let databaseUrl = process.env.DATABASE_URL' assigns the environment variable to a local variable, making it reusable and allowing us to pass it to the rest of the function.

module.exports allows the dbConnect function to be exported so it can be imported and used in another file within the project. In our project, this function is called in the 'index.js' file, which handles the connection to the database.

This follows the modular programming approach, where code is split into smaller, reusable pieces, making the project more organised and maintainable.

```
JavaScript
const mongoose = require("mongoose");
async function dbConnect() {
  console.log("Attempting to connect to:", process.env.DATABASE_URL);
  let databaseUrl = process.env.DATABASE_URL;

  if (!databaseUrl) {
    databaseUrl = `mongodb://127.0.0.1:27017/
    ${process.env.npm_package_name}`;
  }

  try {
    await mongoose.connect(databaseUrl);
    console.log("Successfully connected to MongoDB!");
  } catch (error) {
    console.error("Database connection error:", error);
    throw error;
  }
}

module.exports = {
  dbConnect,
};
```

The same dbConnect function with comments removed for brevity and visibility.



R14. dbConnect Function

Database Operation

```
1 DATABASE_URL=mongodb+srv://VVAdminUser:NFK2ABHA@vinylvibecluster.lcapi.mongodb.net/vinyl-vibe-main?retryWrites=true&w=majority&appName=VinylVibeCluster
```

We store the database connection details in environment variables within a .env file, which is excluded from our source code using a .gitignore file.

The .env file contains sensitive information, such as the username, password, and a link to the MongoDB database.

Mongoose allows us to access these environment variables and use them to establish a secure connection to the database.

R14. dbConnect Function

Conditional Statement

The conditional statement `if (!databaseUrl)` creates a fallback to a local MongoDB database if no `DATABASE_URL` is provided.

The line `databaseUrl = 'mongodb://127.0.0.1:27017/'` sets a default connection URL for the local MongoDB instance that wasn't created earlier.

The expression `${process.env.npm_package_name}` dynamically retrieves the project name from the `package.json` file and appends it as the database name in the local MongoDB instance.

In our project, since we have provided an environment variable for the database connection, `'databaseUrl'` is not falsy. As a result, the condition `if (!databaseUrl)` evaluates to false, and the program skips the conditional block and moves directly to the `try` block.

```
JavaScript
const mongoose = require("mongoose");
async function dbConnect() {
  console.log("Attempting to connect to:", process.env.DATABASE_URL);
  let databaseUrl = process.env.DATABASE_URL;

  if (!databaseUrl) {
    databaseUrl = `mongodb://127.0.0.1:27017/` +
      `${process.env.npm_package_name}`;
  }

  try {
    await mongoose.connect(databaseUrl);
    console.log("Successfully connected to MongoDB!");
  } catch (error) {
    console.error("Database connection error:", error);
    throw error;
  }
}

module.exports = {
  dbConnect,
};
```

The same `dbConnect` function with comments removed for brevity and visibility.

R14. dbConnect Function

Conditional Statement

The try-catch block allows us to handle errors if a database connection cannot be established.

Since we are using an asynchronous function, we need to use the await keyword.

The line `await mongoose.connect(databaseUrl);` takes the `databaseUrl` variable we created and passes it to Mongoose to establish a connection with the database.

If the connection is successful, we console log a message indicating that we have successfully connected to MongoDB.

```
JavaScript
const mongoose = require("mongoose");
async function dbConnect() {
  console.log("Attempting to connect to:", process.env.DATABASE_URL);
  let databaseUrl = process.env.DATABASE_URL;

  if (!databaseUrl) {
    databaseUrl = `mongodb://127.0.0.1:27017/`;
    ${process.env.npm_package_name}`;
  }
  try {
    await mongoose.connect(databaseUrl);
    console.log("Successfully connected to MongoDB!");
  } catch (error) {
    console.error("Database connection error:", error);
    throw error;
  }
}

module.exports = {
  dbConnect,
};
```

The same dbConnect function with comments removed for brevity and visibility.

R14. dbConnect Function

Conditional Statement

If the database connection is unsuccessful, we move to the catch block.

The console log outputs the specific error that occurred, helping us debug the issue.

The statement throw error re-throws the error, effectively stopping further execution of the function unless it is handled elsewhere.

```
const mongoose = require("mongoose");
async function dbConnect() {
  console.log("Attempting to connect to:", process.env.DATABASE_URL);
  let databaseUrl = process.env.DATABASE_URL;

  if (!databaseUrl) {
    databaseUrl = `mongodb://127.0.0.1:27017/`;
    `${process.env.npm_package_name}`;
  }
  try {
    await mongoose.connect(databaseUrl);
    console.log("Successfully connected to MongoDB!");
  } catch (error) {
    console.error("Database connection error:", error);
    throw error;
  }
}

module.exports = {
  dbConnect,
};
```

JavaScript

The same dbConnect function with comments removed for brevity and visibility.



R15: Error 1

On the first iteration of the signup route, I could create users, but no JWT was being passed to Bruno.

A screenshot of the Bruno API client interface. The top bar shows "VINYL VIBE AUTH API" and "No Environment". Below, a collection named "POST User Sig..." is selected. The request details show a POST to "http://localhost:8080/signup". The request body is a JSON object with "username": "reece2" and "password": "examplePassword". The response tab is active, showing a 200 OK status with a response body that is a JSON object containing a "jwt" field, which is currently empty. A red arrow points from the text "no JWT was being passed to Bruno." to the empty "jwt" field in the response.

VINYL VIBE AUTH API

POST User Sig... +

POST http://localhost:8080/signup

Params Body Headers Auth Vars Script Assert Tests Docs JSON Prettify

```
1 {
2   "username": "reece2",
3   "password": "examplePassword"
4 }
```

Response Headers Timeline Tests

200 OK 46ms 291B

```
1 {
2   "jwt":
3   "user": {
4     "id": "675036e68cd56bf5a66c72ac",
5     "username": "reece2"
6   }
7 }
```

To refresh my memory on how to build the auth section in a MERN stack app, I reviewed and re-did the lecture on 'Express Authentication' that we had done a couple of weeks earlier. I would then adjust the existing code to suit our purpose.

The code had worked in the lesson, so at some point I've made the mistake of making the 'generateJWT' function an 'async' function.

I've carried this mistake into the Auth section of the app.

This was frustrating, because this was also not throwing an error.

```
JavaScript

const jwt = require("jsonwebtoken")

let jwtSecretKey = process.env.JWT_SECRET_KEY;

// async function generateJWT(userDetailsObj)
async function generateJWT(userId, username, roles = null){
  return jwt.sign(
    {
      userId: userId,
      username: username,
      roles: roles
    },
    jwtSecretKey,
    {
      expiresIn: "7d"
    }
  );
}
```

Using the 'async' keyword with 'jwt.sign' causes the function to return a Promise instead of the token directly. This happens because 'async' functions always wrap their return values in a Promise.

Without using 'await' or '.then()' to resolve the Promise, the generated JWT token was created, but it wasn't properly returned to Bruno (or any other consumer of the function).

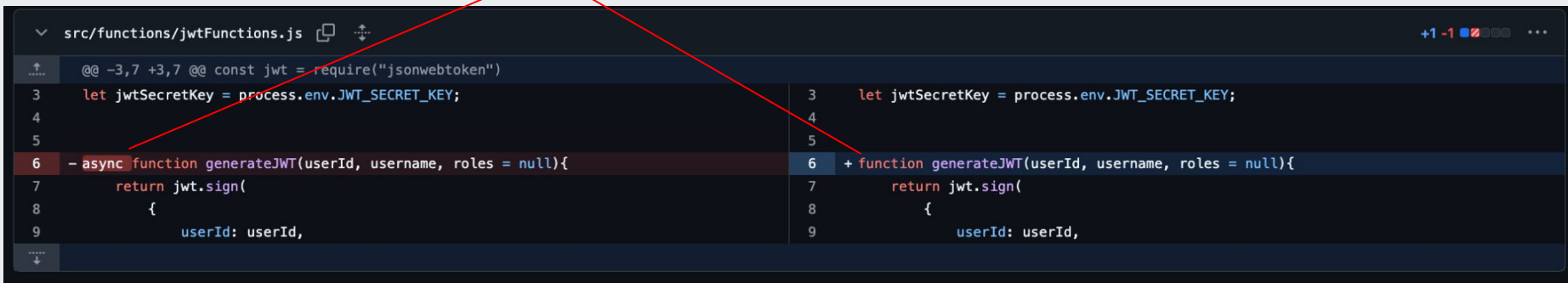
JavaScript

```
const jwt = require("jsonwebtoken")

let jwtSecretKey = process.env.JWT_SECRET_KEY;

// async function generateJWT(userDetailsObj)
async function generateJWT(userId, username, roles = null){
  return jwt.sign(
    {
      userId: userId,
      username: username,
      roles: roles
    },
    jwtSecretKey,
    {
      expiresIn: "7d"
    }
  );
}
```

The fix for this ended up being very simple.



```
src/functions/jwtFunctions.js @@ -3,7 +3,7 @@ const jwt = require("jsonwebtoken")
3  let jwtSecretKey = process.env.JWT_SECRET_KEY;
4
5
6 - async function generateJWT(userId, username, roles = null){
7   return jwt.sign(
8     {
9       userId: userId,
```

```
3  let jwtSecretKey = process.env.JWT_SECRET_KEY;
4
5
6 + function generateJWT(userId, username, roles = null){
7   return jwt.sign(
8     {
9       userId: userId,
```




R15: Error 1

I debugged this by writing comments about what each line of code did to talk through the flow of the function.

I asked myself “Why does it need to be an ‘async’ function?”, and I didn’t have an answer, because it does not need to be an ‘async’ function in this context.

‘jwt.sign’ is synchronous. Therefore it returns the JWT immediately, without a call back.

Once I changed that from an ‘async’ to a synchronous function, the JWT that was previously being wrapped in a Promise was now being passed as a token directly to Bruno.

JavaScript

```
// Importing the 'jsonwebtoken' package for generating and verifying JWTs
const jwt = require("jsonwebtoken");

// Secret key used to sign the JWT; it is stored securely in the .env file
let jwtSecretKey = process.env.JWT_SECRET_KEY;

// Function to generate a JWT (JSON Web Token) with user details
// Takes userId, username, and roles (optional, defaults to null) as arguments
function generateJWT(userId, username, roles = null) {
  return jwt.sign(
    {
      // Payload of the JWT containing user information
      userId: userId,
      username: username,
      roles: roles
    },
    // Secret key used to sign and secure the token
    jwtSecretKey,
    {
      // Token expiration time set to 7 days
      expiresIn: "7d"
    }
  );
}
```



R15: Error 1

BOOM!

And there was much rejoicing!

A screenshot of a REST client interface for "VINYL VIBE AUTH API". The selected request is a POST to "http://localhost:8080/signup" with a JSON body containing "username": "reece2" and "password": "examplePassword". The response is a 200 OK status with a JSON body containing a JWT token and a user object with "id" and "username" fields. A red arrow points from the text "And there was much rejoicing!" to the "jwt" field in the response.

VINYL VIBE AUTH API

POST User Sig... +

POST http://localhost:8080/signup

Params Body Headers Auth Vars Script Assert Tests Docs JSON Prettify

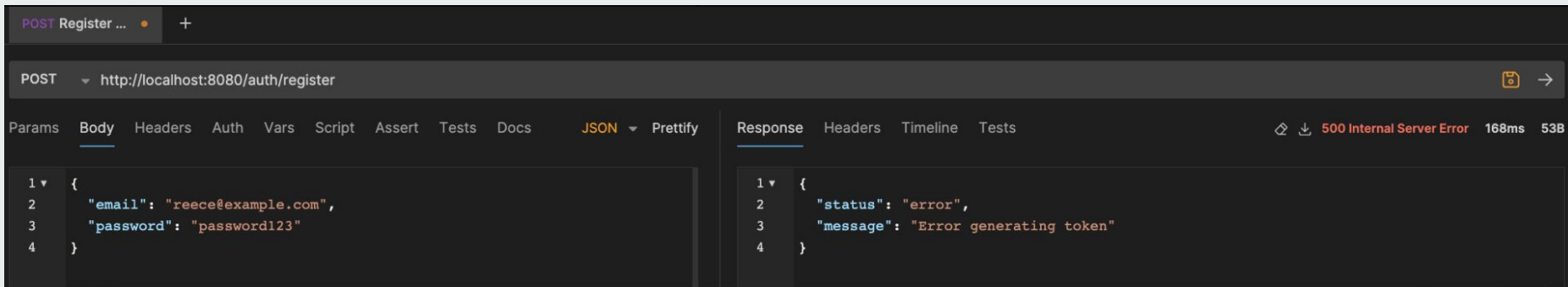
```
1 {
2   "username": "reece2",
3   "password": "examplePassword"
4 }
```

Response Headers Timeline Tests

200 OK 46ms 2915

```
1 {
2   "jwt": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOiI2NzUwMzZlNjhjZDU2YmY1YTY2YzcyYWMiLCJ1c2VybmFtZSI6InJlZWNlMiIsInJvbGVzIjpudWxsLCJpYXQiOiB3MzZmZMTAxODIsImV4cCI6MTczMzIxNDk4Mn0.bfp9e_C7KLr7LeBhx6FjZSK-GgfGoVioc30ALQsvsQo",
3   "user": {
4     "id": "675036e68cd56bf5a66c72ac",
5     "username": "reece2"
6   }
7 }
```

JSON objects from Bruno were not making it into MongoDB Atlas.

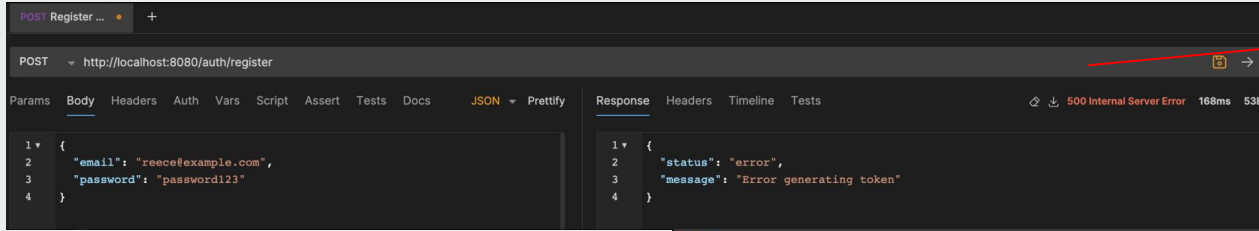


To test the routes I'd made for the Cart, I needed to be a logged in user with a JWT in the Header. In attempting to use the refactored `auth/register` route in Bruno, I would continuously get 500 Internal Server Error.

This was particularly frustrating, as at first I thought this was something to do with the JWT issue I'd had previously. This was more of a knee jerk reaction than anything logical, as at this stage I'm not even able to register a user to generate a JWT. Therefore, the two problems were not related.



R15: Error 2



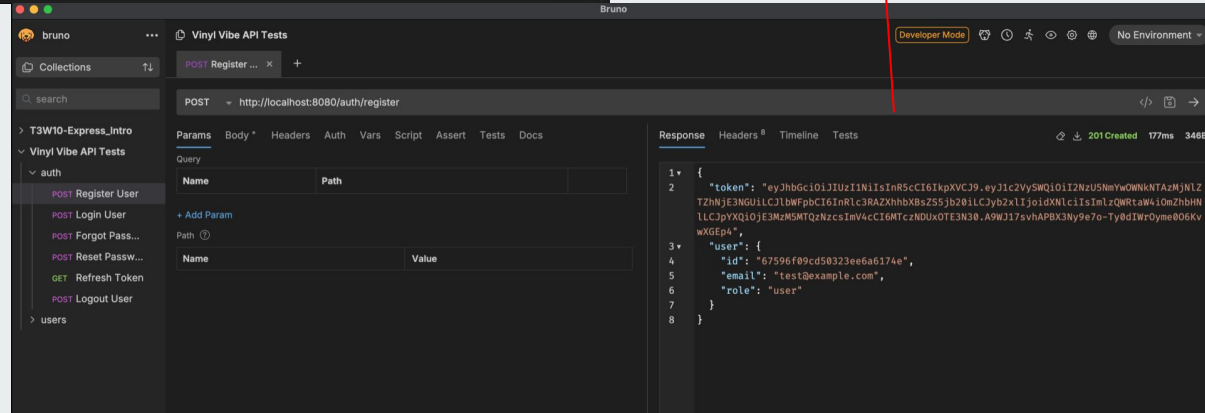
Reece's screen

Damian's screen

To troubleshoot the problem, Damian attempted to use the /auth/register route in Bruno as well. His version worked straight away.

I made sure I was up to date with all commits and did `npm install` to see if there was some packages I was missing.

I was missing some packages, but none that were relevant to this problem.





database.js with dbConnect function

```
src > utils > database.js > dbConnect
1  const mongoose = require("mongoose"); // 875.5k (gzipped: 234.3k)
2
3  /**
4   * Database connection handler
5   *
6   * Why use environment variables for DB connection?
7   * - Security: Keep credentials out of code
8   * - Flexibility: Different DBs for dev/test/prod
9   * - DevOps best practices: Configuration as environment
10  */
11  async function dbConnect() {
12    // Log connection string (without credentials in production)
13    console.log("Attempting to connect to:", process.env.DATABASE_URL);
14
15    // Fall back to local MongoDB if no DATABASE_URL provided
16    // Why use npm_package_name?
17    // - Creates database named after project
18    // - Prevents test/dev database collisions
19    let databaseUrl = process.env.DATABASE_URL;
20
21    if (!databaseUrl) {
22      databaseUrl = `mongodb://127.0.0.1:27017/${process.env.npm_package_name}`;
23    }
24
25    try {
26      // Connect with Mongoose
27      // Why async/await?
28      // - Cleaner error handling
29      // - Ensures connection before server starts
30      // - Prevents race conditions
31      await mongoose.connect(databaseUrl);
32      console.log("Successfully connected to MongoDB!");
33    } catch (error) {
34      // Log error and rethrow
35      // Why rethrow?
36      // - Allows calling code to handle connection failure
37      // - Prevents server from starting with no database
38      console.error("Database connection error:", error);
39      throw error;
40    }
41  }
42
43  module.exports = {
44    dbConnect,
45  };
46
```

R15: Error 2

The next logical step was to check that the dbConnect function was working and up-to-date. But if it was working for Brad and Damian but not me, that means that the dbConnect function was correctly written and implemented and the problem was somewhere local to me.

dbConnect function imported and called in index.js

DATABASE_URL in my .env file that was being accessed in dbConnect function

```
src > index.js > ...
1  // Purpose:
2  // First point of entry
3  // Initiate the server
4  // Get the port
5  // Tell the server to listen to web traffic
6
7  require("dotenv").config(); // 6.3k (gzipped: 2.8k)
8
9  const { app } = require("./server.js");
10  const { dbConnect } = require("./utils/database.js");
11
12  /**
13   * Server startup sequence
14   *
15   * Why this order?
16   * 1. Load environment variables first - needed for all config
17   * 2. Import configured app - avoid circular dependencies
18   * 3. Connect to database before listening - ensure DB is ready
19   * 4. Start listening only after all setup is complete
20  */
21
22  // Get the PORT value from environment variables
23  // Why use environment variable?
24  // - Different ports for different environments
25  // - Hosting platforms often assign their own port
26  // - Avoid port conflicts in development
27  const PORT = process.env.PORT || 8080;
28
29  // Start server and connect to database
30  // Why async IIFE?
31  // - Ensures database connection before accepting requests
32  // - Proper error handling during startup
33  // - Clean shutdown if startup fails
34  app.listen(PORT, async () => {
35    try {
36      await dbConnect();
37      console.log("Server is running on port " + PORT);
38    } catch (error) {
39      console.error("Failed to start server:", error);
40      process.exit(1); // Exit with error code
41    }
42  });
43
```

IN .env

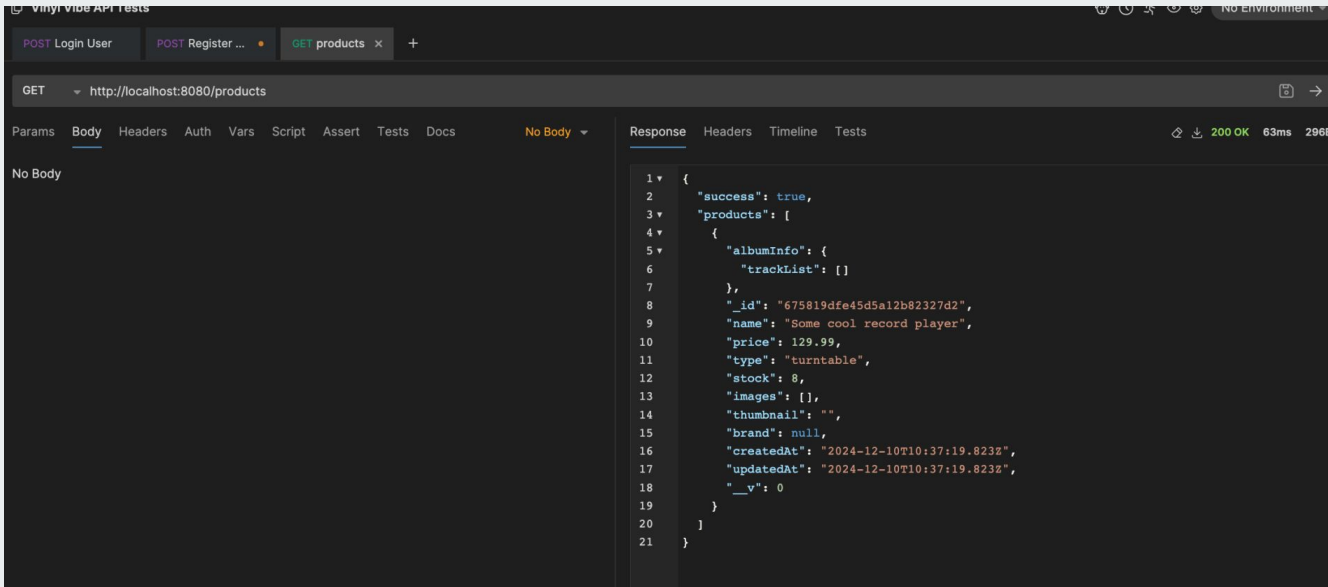
```
1  DATABASE_URL=mongodb+srv://VAdminUser:NFK2ABHA@vinylvibecluster.lcapi.mongodb.net/?retryWrites=true&w=majority&appName=VinylVibeCluster
2  JWT_SECRET=ddbe85ab868f9e76be5c52f75102abf5eb24ef817213a1caffff2a307af5e3357d9de4623e6c4154b2228232194b8866dd0f79a4d8a8153579578789f3b460ea
3  RESEND_API_KEY=re_jUxjiab_n4FLa3UJJKUcAhmFMGFj2Ab
4
5  NODE_ENV=development
6  PORT=8080
7
```



R15: Error 2

As I was working on the Cart Branch which had not yet been merged with Main, I switched over to the Main branch to see if I would get the same error, which I did.

Process of elimination, I switched over to the Products route and attempted a Get request, just to see if I could get anything in Bruno to work, and I was successful!





R15: Error 2

Next I tried the POST route on Bruno where I was successfully able to create a new product called “Aldi Cheap Turntable.”
But the question now was where was this document going?

A screenshot of the Bruno API client interface. The top bar shows "Vinyl Vibe API Tests" and "No Environment". Below the top bar, there are tabs for "POST Login User", "POST Register ...", "GET products", and "POST create-p...". The main area shows a POST request to "http://localhost:8080/products". The request body is a JSON object:

```
{  "name": "Aldi Cheap Turntable",  "price": 12.99,  "type": "turntable",  "stock": 8}
```

. The response is a JSON object:

```
{  "success": true,  "message": "Product created successfully",  "product": {    "name": "Aldi Cheap Turntable",    "price": 12.99,    "type": "turntable",    "albumInfo": {      "trackList": []    },    "stock": 8,    "images": [],    "thumbnail": "",    "brand": null,    "_id": "6759763e7582cb013a6b2051",    "createdAt": "2024-12-11T11:23:42.266Z",    "updatedAt": "2024-12-11T11:23:42.266Z",    "__v": 0  }  }
```

. The status is "200 OK" and the response time is "94ms".



R15: Error 2

The obvious first thought was that the routes must be sending data to my local machine rather than to MongoDB Atlas.

```
async function dbConnect() {  
  // Log connection string (without credentials in production)  
  console.log("Attempting to connect to:", process.env.DATABASE_URL);  
  
  // Fallback to local MongoDB if no DATABASE_URL provided  
  // Why use npm_package_name?  
  // - Creates database named after project  
  // - Prevents test/dev database collisions  
  let databaseUrl = process.env.DATABASE_URL;  
  
  if (!databaseUrl) {  
    databaseUrl = `mongodb://127.0.0.1:27017/${process.env.npm_package_name}`;  
  }  
}
```

Our dbConnect function will first attempt to connect to MongoDB through the link stored in DATABASE_URL in the environment variable.

However, if that doesn't work it creates a mongodb database locally when that doesn't work. This line of code names the database after the project you're working on.

I ran 'brew services list' to make sure mongodb-community was running in the background, which it was.

A screenshot of a terminal window with a dark background. The prompt is '~ /ca'. The command 'brew services list' has been executed. The output is a table with four columns: Name, Status, User, and File. The first row shows 'mongodb-community' with status 'started', user 'reecedoyle', and file path '~/Library/LaunchAgents/homebrew.mxcl.mongodb-community.plist'. The second row shows 'postgresql@16' with status 'started', user 'root', and file path '~/Library/LaunchAgents/homebrew.mxcl.postgresql@16.plist'. The terminal shows a green cursor at the bottom.

Name	Status	User	File
mongodb-community	started	reecedoyle	~/Library/LaunchAgents/homebrew.mxcl.mongodb-community.plist
postgresql@16	started	root	~/Library/LaunchAgents/homebrew.mxcl.postgresql@16.plist



R15: Error 2

I was expecting to find a database called “vinyl_vibe_backend” when I went into Mongosh, but frustratingly, it was not there either.

Alphabetically, it should’ve been at the bottom here.

So I was able to create my “Aldi Cheap Turntable” product document that returned 200 success, but it wasn’t going to my local db, and it also wasn’t going to our products db on MongoDB Atlas.

We only had 4 products in the database at the time and it wasn’t there.

```
{
  "_id": ObjectId('6752e1c3f7b2a2a987702a03'),
  "name": "The Dark Side of the Moon",
  "price": 39.99,
  "type": "vinyl",
  "albumInfo": Object {
    stock: 20
  },
  "images": Array (empty)
  "thumbnail": ""
  "brand": null
  "createdAt": 2024-12-06T11:36:35.569+00:00
  "updatedAt": 2024-12-06T11:36:35.569+00:00
  __v: 0
}
```

```
{
  "_id": ObjectId('67582eed92d9e311df854f51'),
  "name": "AT-LP60X",
  "price": 129.99,
  "type": "turntable",
  "albumInfo": Object {
    stock: 8
  },
  "images": Array (empty)
  "thumbnail": ""
  "brand": "Audio-Technica"
  "createdAt": 2024-12-10T12:07:09.997+00:00
  "updatedAt": 2024-12-10T12:07:09.997+00:00
  __v: 0
}
```

```
{
  "_id": ObjectId('675930253880b03b7571192e'),
  "name": "R-51M Bookshelf Speakers",
  "price": 199.99,
  "type": "speaker",
  "albumInfo": Object {
    stock: 12
  },
  "images": Array (empty)
  "thumbnail": ""
  "brand": "Klipsch"
  "createdAt": 2024-12-10T12:12:21.590+00:00
  "updatedAt": 2024-12-10T12:37:59.687+00:00
  __v: 0
}
```

```
{
  "_id": ObjectId('67596bb44c8dc755fd78974'),
  "name": "Audio-Technica AT-LP60X",
  "price": 129.99,
  "type": "turntable",
  "albumInfo": Object {
    stock: 8
  },
  "images": Array (empty)
  "thumbnail": ""
  "brand": null
}
```



R15: Error 2

Damian noticed that there had been a database called “test” underneath our vinyl vibe cluster. When he looked in that, he found my “Aldi Cheap Turntable”!

We learned the hard way that when you don't specify a database in MongoDB, it just creates one and calls it 'test'.

The screenshot shows the MongoDB Compass interface. On the left, a tree view shows the database structure. The database is named 'vinylvibecluster.lcapi.mongodb.net'. Underneath, there are several collections: 'admin', 'config', 'local', 'test', 'products', 'users', 'vinyl-vibe-main', 'cart', 'orders', 'products', and 'users'. The 'test' database is selected, and the 'products' collection is highlighted. On the right, the document viewer shows two documents. The first document is a 'turntable' with a name 'Some cool record player' and a price of 129.99. The second document is a 'turntable' with a name 'Aldi cheap turntable' and a price of 12.99. Both documents have a stock of 8 and were created/updated on 2024-12-10T10:37:19.823+00:00. The interface includes buttons for 'ADD DATA', 'EXPORT DATA', 'UPDATE', and 'DELETE' at the top right.



R15: Error 2

The issue was that I had the wrong link in my environment variables. I'd missed a message in discord to update the DATABASE_URL link, and therefore was only pointed at the top level, rather than at vinyl-vibe-main, which was where I needed to be pointed.

Original link

```
DATABASE_URL=mongodb+srv://VAdminUser:NFK2ABHA@vinylvibecluster.lcap1.mongodb.net/?retryWrites=true&w=majority&appName=VinylVibeCluster
```

Updated link

```
DATABASE_URL=mongodb+srv://VAdminUser:NFK2ABHA@vinylvibecluster.lcap1.mongodb.net/vinyl-vibe-main?retryWrites=true&w=majority&appName=VinylVibeCluster
```

The difference is that this 'vinyl-vibe-main' just needed to be added.

With the environment variables fixed, I was now able to get to the some errors handled by our Middleware, which is good! That meant the connection was working.

This was the "user already exists" error, which was easily worked around by changing the JSON body with a new email address.

```
> vinyl-vibe-backend@1.0.0 dev
> nodemon src/index.js

[nodemon] 3.1.7
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting node src/index.js
Attempting to connect to: mongodb+srv://VAdminUser:NFK2ABHA@vinylvibecluster.lcap1.mongodb.net/vinyl-vibe-main?retryWrites=true&w=majority&appName=VinylVibeCluster
Successfully connected to MongoDB!
Server is running on port 8080
ERROR ✖ AppError: Error generating token
    at Object.generateToken (/Users/reecedoyle/CA/term-last/vinyl-vibe-backend/src/auth/AuthService.js:30:10)
    at register (/Users/reecedoyle/CA/term-last/vinyl-vibe-backend/src/auth/AuthController.js:28:36)
    at process.processTicksAndRejections (node:internal/process/task_queues:95:5) {
  statusCode: 500,
  status: 'error',
  isOperational: true
}
ERROR ✖ AppError: User already exists
    at Object.createUser (/Users/reecedoyle/CA/term-last/vinyl-vibe-backend/src/users/UserService.js:17:11)
    at process.processTicksAndRejections (node:internal/process/task_queues:95:5)
    at async register (/Users/reecedoyle/CA/term-last/vinyl-vibe-backend/src/auth/AuthController.js:27:20) {
  statusCode: 400,
  status: 'fail',
  isOperational: true
}
ERROR ✖ AppError: Error generating token
    at Object.generateToken (/Users/reecedoyle/CA/term-last/vinyl-vibe-backend/src/auth/AuthService.js:30:10)
    at register (/Users/reecedoyle/CA/term-last/vinyl-vibe-backend/src/auth/AuthController.js:28:36)
    at process.processTicksAndRejections (node:internal/process/task_queues:95:5) {
  statusCode: 500,
  status: 'error',
  isOperational: true
}
```



With my new user and JWT, I was able to confirm through a GET request in Bruno that the products I was viewing matched those in MongoDB Atlas. From then on, all other endpoints worked!

R15: Error 2

```
{
  "success": true,
  "products": [
    {
      "albumInfo": {
        "trackList": []
      },
      "_id": "6752e1c3f7b2a2a987702a03",
      "name": "The Dark Side of the Moon",
      "price": 39.99,
      "type": "vinyl",
      "stock": 20,
      "images": [],
      "thumbnail": "",
      "brand": null,
      "createdAt": "2024-12-06T11:36:35.569Z",
      "updatedAt": "2024-12-06T11:36:35.569Z",
      "__v": 0
    },
    {
      "albumInfo": {
        "trackList": []
      },
      "_id": "67582eed92d9e311df854f51",
      "name": "AT-LP60X",
      "price": 129.99,
      "type": "turntable",
      "stock": 8,
      "images": [],
      "thumbnail": "",
      "brand": "Audio-Technica",
      "createdAt": "2024-12-10T12:07:09.997Z",
      "updatedAt": "2024-12-10T12:07:09.997Z",
      "__v": 0
    }
  ],
}
```

```
_id: ObjectId('6752e1c3f7b2a2a987702a03')
name: "The Dark Side of the Moon"
price: 39.99
type: "vinyl"
albumInfo: Object
stock: 20
images: Array (empty)
thumbnail: ""
brand: null
createdAt: 2024-12-06T11:36:35.569+00:00
updatedAt: 2024-12-06T11:36:35.569+00:00
__v: 0
```

```
_id: ObjectId('67582eed92d9e311df854f51')
name: "AT-LP60X"
price: 129.99
type: "turntable"
albumInfo: Object
stock: 8
images: Array (empty)
thumbnail: ""
brand: "Audio-Technica"
createdAt: 2024-12-10T12:07:09.997+00:00
updatedAt: 2024-12-10T12:07:09.997+00:00
__v: 0
```