Explanation:

1. **Import necessary modules:**
    - express: For creating the Express.js server.
    - mongoose: For interacting with MongoDB.
2. **Connect to MongoDB:**
    - mongoose.connect() establishes a connection to the MongoDB database.
    - useNewUrlParser and useUnifiedTopology are options for ensuring compatibility with newer MongoDB drivers.
3. **Define a Schema:**
    - userSchema defines the structure of the documents that will be stored in the users collection.
    - name and age are fields within the schema, specifying their data types.
4. **Create a Model:**
    - mongoose.model('User', userSchema) creates a Mongoose model named 'User' based on the userSchema.
5. **Define Routes:**
    - **/users (POST):**
        - Creates a new user document.
        - req.body contains the data sent in the request body (e.g., { name: 'John', age: 30 }).
        - newUser.save() saves the new document to the database.
    - **/users (GET):**
        - Retrieves all user documents from the database using User.find().
6. **Start the Server:**
    - app.listen() starts the Express.js server on the specified port.

To run this example:

1. **Install dependencies:**

```
npm install express mongoose
```

2. **Create a MongoDB database:**
    - Start your MongoDB server.
    - Create a new database (e.g., 'yourDatabaseName') using the MongoDB shell or a database management tool.
3. **Save the code:**
    - Save the code above as a .js file (e.g., server.js).
4. **Run the server:**
    - Execute the following command in your terminal:

```
node server.js
```

Key Concepts:

- **Mongoose:** A powerful Object Data Modeling (ODM) library for MongoDB.[1]
- **Schemas:** Define the structure of documents in your MongoDB collections.
- **Models:** Represent collections of documents in your database.
- **CRUD Operations:** This example demonstrates basic Create (POST) and Read (GET) operations. You can extend this to include Update (PUT/PATCH) and Delete (DELETE) operations.

This example provides a basic foundation for building MongoDB-powered applications with Express.js. You can further enhance it by adding features like authentication, authorization, pagination, and more complex data models. The error message you're encountering (SyntaxError: invalid syntax) indicates that the code is being executed in an environment that doesn't support JavaScript's const keyword. This might be happening if you're attempting to run this Express.js code within a Python environment or a similar context where JavaScript is not the primary language.

To run this Express.js code correctly:

1. **Ensure you have Node.js and npm installed:**
    - Download and install Node.js from the official website (nodejs.org). This will also install npm (Node Package Manager).
2. **Install the mongoose package:**

- Open your terminal or command prompt.
- Navigate to the directory where you'll save the code.
- Run the following command to install the mongoose package:
  npm install mongoose

3. **Create a file:**
   - Save the code above as an .js file (e.g., server.js).
4. **Run the server:**
   - Open your terminal or command prompt.
   - Navigate to the directory where you saved the server.js file.
   - Run the following command:

     node server.js

Explanation:

1. Import necessary modules:

- imports the Express.js framework

```
const express = require('express');
```

- imports the Mongoose library for interacting with MongoDB

```
const mongoose = require('mongoose');
```

2. Connect to MongoDB:

- mongoose.connect(...) establishes a connection to the MongoDB database.
  - Replace 'mongodb://localhost:27017/yourDatabaseName' with the actual connection string to your MongoDB instance.
- The .then() and .catch() blocks handle the connection success and failure, respectively.

```
mongoose = require('mongoose')
mongoose.connect('mongodb://localhost:27017/mydatabase'  )
.then(() => {
  console.log('MongoDB connected successfully')
}).catch(err => {
  console.error('MongoDB connection error:', err)
```

```
})
```

3. Define a Mongoose Schema:

- `const userSchema = new mongoose.Schema({ ... });` defines a schema for the `User` documents.
- The schema defines the structure of the documents, including the fields and their data types.

4. Create a Mongoose Model:

- `const User = mongoose.model('User', userSchema);` creates a Mongoose model named `User` based on the `userSchema`.

5. Define Routes:

- **Create User:**
  - `app.post('/users', ...)` defines a route handler for POST requests to `/users`.
  - It creates a new `User` instance using the data from the request body (`req.body`).
  - `newUser.save()` saves the new user document to the database.
- **Get All Users:**
  - `app.get('/users', ...)` defines a route handler for GET requests to `/users`.
  - `User.find()` retrieves all documents from the `User` collection.

6. Start the Server:

- `app.listen(port, () => { ... });` starts the Express.js server and listens for incoming requests on the specified port.

This example demonstrates a basic integration of Express.js with MongoDB. You can expand upon this by adding more routes, implementing CRUD operations (Create, Read, Update, Delete), implementing more complex data models, and using Mongoose's features like validations, middleware, and more.

I hope this explanation is helpful! Let me know if you have any further questions.