This code sets up a simple Node.js web server using Express.js that connects to a MongoDB database. Let me break it down section by section:

1. Initial Setup and Dependencies:

```javascript

const express = require('express');

const app = express();

const cors = require('cors');

require('dotenv').config()

const port = process.env.PORT || 5000;

```

- Imports Express.js framework

- Creates an Express application

- Imports CORS middleware for cross-origin requests

- Loads environment variables from a .env file

- Sets the port (either from environment variable or defaults to 5000)

2. Middleware:

```javascript

app.use(cors());

app.use(express.json());

```

- Enables CORS for all routes

- Parses incoming JSON requests

3. MongoDB Connection Setup:

```javascript

const { MongoClient, ServerApiVersion } = require('mongodb');

const uri = `mongodb+srv://${process.env.DB\_USER}:${process.env.DB\_PASS}@cluster0.swu9d.mongodb.net/?retryWrites=true&w=majority`;

```

- Imports MongoDB client

- Creates a connection string using environment variables for username and password

4. MongoDB Client Configuration:

```javascript

const client = new MongoClient(uri, {

serverApi: {

version: ServerApiVersion.v1,

strict: true,

deprecationErrors: true,

}

});

```

- Creates a MongoDB client with specific server API settings for stability

5. Database Operations (async run function):

```javascript

async function run() {

try {

await client.connect();

const menuCollection = client.db("bistroDb").collection("menu");

const reviewCollection = client.db("bistroDb").collection("reviews");

```

- Connects to MongoDB

- Creates references to "menu" and "reviews" collections in "bistroDb" database

6. API Endpoints:

```javascript

app.get('/menu', async(req, res) =>{

const result = await menuCollection.find().toArray();

res.send(result);

})

app.get('/reviews', async(req, res) =>{

const result = await reviewCollection.find().toArray();

res.send(result);

})

```

- Creates GET endpoint /menu that returns all menu items

- Creates GET endpoint /reviews that returns all reviews

- Both fetch data from MongoDB and send it as response

7. Connection Verification:

```javascript

await client.db("admin").command({ ping: 1 });

console.log("Pinged your deployment. You successfully connected to MongoDB!");

```

- Sends a ping to verify MongoDB connection

8. Root Endpoint and Server Start:

```javascript

app.get('/', (req, res) => {

res.send('boss is sitting')

})

app.listen(port, () => {

console.log(`Bistro boss is sitting on port ${port}`);

})

```

- Creates a root endpoint that returns a simple message

- Starts the server on the specified port

Key Features:

- Uses environment variables for secure database credentials

- Implements async/await for handling asynchronous operations

- Provides basic REST API endpoints

- Connects to MongoDB Atlas (cloud database service)

- Includes error handling with try/catch

Note: The client.close() is commented out in the finally block, meaning the connection stays open as long as the server runs, which is typical for a web server application.

This appears to be a backend for a restaurant/bistro application that serves menu items and customer reviews through API endpoints.

This is a Node.js Express server that connects to MongoDB Atlas and provides basic API endpoints. Here's a detailed explanation:

1. Dependencies and Setup:

```javascript

const express = require('express');

const app = express();

const cors = require('cors');

require('dotenv').config()

const port = process.env.PORT || 5000;

```

- Uses Express.js as the web framework

- Enables CORS for cross-origin requests

- Loads environment variables from .env file

- Sets port from environment variable or defaults to 5000

2. Middleware Configuration:

```javascript

app.use(cors());

app.use(express.json());

```

- Allows requests from different origins

- Parses JSON request bodies

3. MongoDB Connection:

```javascript

const { MongoClient, ServerApiVersion } = require('mongodb');

const uri = `mongodb+srv://${process.env.DB\_USER}:${process.env.DB\_PASS}@cluster0.swu9d.mongodb.net/?retryWrites=true&w=majority`;

const client = new MongoClient(uri, {

serverApi: {

version: ServerApiVersion.v1,

strict: true,

deprecationErrors: true,

}

});

```

- Sets up MongoDB connection using credentials from environment variables

- Configures client with stable API version settings

4. Main Database Operations (run function):

```javascript

async function run() {

try {

await client.connect();

const menuCollection = client.db("bistroDb").collection("menu");

const reviewCollection = client.db("bistroDb").collection("reviews");

```

- Establishes MongoDB connection

- Defines collections for menu and reviews in "bistroDb" database

5. API Routes:

```javascript

app.get('/menu', async(req, res) =>{

const result = await menuCollection.find().toArray();

res.send(result);

})

app.get('/reviews', async(req, res) =>{

const result = await reviewCollection.find().toArray();

res.send(result);

})

```

- GET /menu: Returns all menu items

- GET /reviews: Returns all reviews

- Both endpoints fetch data from MongoDB and return it as JSON

6. Connection Validation:

```javascript

await client.db("admin").command({ ping: 1 });

console.log("Pinged your deployment. You successfully connected to MongoDB!");

```

- Verifies successful database connection

7. Server Configuration:

```javascript

app.get('/', (req, res) => {

res.send('boss is sitting')

})

app.listen(port, () => {

console.log(`Bistro boss is sitting on port ${port}`);

})

```

- Root endpoint returns a simple message

- Starts server on specified port

Key Points:

- Uses async/await for asynchronous operations

- Connects to MongoDB Atlas (cloud database)

- Keeps database connection open (client.close() is commented out)

- Basic error handling with try/catch

- Environment variables secure sensitive data

This appears to be a backend for a restaurant application, likely part of a larger system with a frontend that would consume these API endpoints to display menu items and reviews.

**Express.js Server with MongoDB Integration and Cart Functionality**

## ****1. Initial Setup****

const express = require('express');

const app = express();

const cors = require('cors');

require('dotenv').config();

const port = process.env.PORT || 5000;

* Sets up Express server with CORS and environment variables
* Port defaults to 5000 if not specified in .env

## ****2. Middleware****

app.use(cors());

app.use(express.json());

* Enables cross-origin requests
* Parses JSON request bodies

## ****3. MongoDB Setup****

const { MongoClient, ServerApiVersion, ObjectId } = require('mongodb');

const uri = `mongodb+srv://${process.env.DB\_USER}:${process.env.DB\_PASS}@cluster0.swu9d.mongodb.net/?retryWrites=true&w=majority`;

* Imports MongoDB components including ObjectId for ID handling
* Sets up connection string with credentials from .env

## ****4. MongoDB Client****

const client = new MongoClient(uri, {

serverApi: {

version: ServerApiVersion.v1,

strict: true,

deprecationErrors: true,

}

});

* Configures MongoDB client with stable API settings

## ****5. Database Operations (run function)****

async function run() {

try {

await client.connect();

const menuCollection = client.db("bistroDb").collection("menu");

const reviewCollection = client.db("bistroDb").collection("reviews");

const cartCollection = client.db("bistroDb").collection("carts");

* Connects to MongoDB
* Defines three collections: menu, reviews, and carts

## ****6. API Endpoints****

### a. ****Menu and Reviews (unchanged)****

app.get('/menu', async (req, res) => {

const result = await menuCollection.find().toArray();

res.send(result);

})

app.get('/reviews', async (req, res) => {

const result = await reviewCollection.find().toArray();

res.send(result);

})

* **GET /menu**: Returns all menu items
* **GET /reviews**: Returns all reviews

### b. ****Cart Operations (new)****

app.get('/carts', async (req, res) => {

const email = req.query.email;

const query = { email: email };

const result = await cartCollection.find(query).toArray();

res.send(result);

});

app.post('/carts', async (req, res) => {

const cartItem = req.body;

const result = await cartCollection.insertOne(cartItem);

res.send(result);

});

app.delete('/carts/:id', async (req, res) => {

const id = req.params.id;

const query = { \_id: new ObjectId(id) };

const result = await cartCollection.deleteOne(query);

res.send(result);

});

* **GET /carts**: Retrieves cart items for a specific user (filtered by email query parameter)
* **POST /carts**: Adds a new item to the cart (data from request body)
* **DELETE /carts/:id**: Removes a specific cart item by ID

## ****7. Connection Verification and Server Start****

await client.db("admin").command({ ping: 1 });

console.log("Pinged your deployment. You successfully connected to MongoDB!");

app.get('/', (req, res) => {

res.send('boss is sitting');

})

app.listen(port, () => {

console.log(`Bistro boss is sitting on port ${port}`);

})

* Verifies MongoDB connection
* Sets up root endpoint and starts server

## ****8. Naming Convention Comment****

/\*\*

\* --------------------------------

\* NAMING CONVENTION

\* --------------------------------

\* app.get('/users') // Get all

\* app.get('/users/:id') // Get one

\* app.post('/users') // Create

\* app.put('/users/:id') // Update (replace)

\* app.patch('/users/:id') // Update (modify)

\* app.delete('/users/:id') // Delete

\*/

* Documents RESTful API conventions (though not all are implemented)

## ****Key Features****

* CRUD operations for cart (Create, Read, Delete)
* User-specific cart filtering using email
* MongoDB ObjectId for precise document identification
* Follows RESTful API patterns
* Secure configuration with environment variables

## ****Summary****

This backend serves as a **restaurant ordering system**, now including shopping cart functionality alongside menu and review features. The cart system allows users to:

* **View** their cart items (GET)
* **Add** items to cart (POST)
* **Remove** items from cart (DELETE)

**Express and MongoDB API Documentation**

**Overview**

This document explains the implementation of an Express server with MongoDB integration. It provides authentication using JWT, role-based access control, and CRUD operations for users, menus, reviews, and carts.

**Project Setup**

* Install required dependencies:
* npm install express cors dotenv mongodb jsonwebtoken
* Create a .env file with:
* DB\_USER=your\_db\_username
* DB\_PASS=your\_db\_password
* ACCESS\_TOKEN\_SECRET=your\_secret\_key

**Middleware Configuration**

* cors(): Enables Cross-Origin Resource Sharing.
* express.json(): Parses incoming JSON requests.

**Database Connection**

* MongoDB is connected using MongoClient.
* Collections:
  + users
  + menu
  + reviews
  + carts

**Authentication & Authorization**

* JWT Token Generation:
* app.post('/jwt', async (req, res) => {
* const user = req.body;
* const token = jwt.sign(user, process.env.ACCESS\_TOKEN\_SECRET, { expiresIn: '1h' });
* res.send({ token });
* });
* Middleware for token verification:
* const verifyToken = (req, res, next) => {
* if (!req.headers.authorization) return res.status(401).send({ message: 'unauthorized access' });
* const token = req.headers.authorization.split(' ')[1];
* jwt.verify(token, process.env.ACCESS\_TOKEN\_SECRET, (err, decoded) => {
* if (err) return res.status(401).send({ message: 'unauthorized access' });
* req.decoded = decoded;
* next();
* });
* };
* Role-based Admin Verification:
* const verifyAdmin = async (req, res, next) => {
* const email = req.decoded.email;
* const user = await userCollection.findOne({ email });
* if (user?.role !== 'admin') return res.status(403).send({ message: 'forbidden access' });
* next();
* };

**API Endpoints**

**Users**

| **Method** | **Endpoint** | **Description** |
| --- | --- | --- |
| GET | /users | Get all users (Admin only) |
| GET | /users/admin/:email | Check if a user is admin |
| POST | /users | Add a new user |
| PATCH | /users/admin/:id | Update a user to admin |
| DELETE | /users/:id | Delete a user (Admin only) |

**Menu**

| **Method** | **Endpoint** | **Description** |
| --- | --- | --- |
| GET | /menu | Get all menu items |

**Reviews**

| **Method** | **Endpoint** | **Description** |
| --- | --- | --- |
| GET | /reviews | Get all reviews |

**Cart**

| **Method** | **Endpoint** | **Description** |
| --- | --- | --- |
| GET | /carts | Get cart items for a user |
| POST | /carts | Add an item to cart |
| DELETE | /carts/:id | Remove an item from cart |

**Server Initialization**

app.listen(port, () => {

console.log(`Server is running on port ${port}`);

});

**Naming Conventions**

* app.get('/users') → Retrieve all users.
* app.get('/users/:id') → Retrieve a user by ID.
* app.post('/users') → Create a new user.
* app.put('/users/:id') → Fully update user data.
* app.patch('/users/:id') → Partially update user data.
* app.delete('/users/:id') → Remove a user.

**Conclusion**

This document outlines the API structure, authentication methods, and access control for managing users, menus, reviews, and carts within a MongoDB-powered Express server.