This code is a Node.js application that uses the Express framework to create a RESTful API for managing products in a hypothetical inventory database. It also connects to a MySQL database using the `mysql2` library for data storage and retrieval. Let's break down the code step by step:

1. **Importing Required Modules:*

```
  const express = require("express");
  const mysql = require("mysql2");
```

  - Here, the code imports the necessary Node.js modules: `express` for creating the server and routing, and `mysql2` for connecting to and interacting with a MySQL database.

2. **Creating a MySQL Connection:**

```
     const con = mysql.createConnection({
      host: "localhost",
      user: "root",
      password: "admin123",
      database: "inventory_db"
  });
```

  - This code sets up a connection to a MySQL database named "inventory_db" running on the local machine. It provides connection details such as the host, username, password, and the name of the database.

3. **Creating an Express Application:**

```
  const app = express();
```

  - This line initializes an Express application, which will serve as the server for handling HTTP requests.

4. **Middleware for Parsing JSON Requests:**

```
  app.use(express.json());
```

  - This middleware is added to the Express app to parse incoming JSON data from requests. It allows the application to handle JSON-formatted data in requests.

5. **Defining Routes and Handling HTTP Requests:**

- The code defines various routes and handles different types of HTTP requests (GET, POST, PUT, DELETE) for managing products in the inventory database.

- For example, `app.get("/", ...)` defines a route for the root URL, and `app.get("/products/:id", ...)` defines a route that accepts a product ID as a parameter.

- Inside each route handler, there are corresponding SQL queries executed using the MySQL connection (`con`) to interact with the database. These queries include inserting, updating, selecting, or deleting data in the "products" table.

- The code handles errors that may occur during database operations and responds to the client with appropriate status codes and JSON messages.

6. **Starting the Express Server:**

```
    app.listen(3000, () => {
     console.log("Listening on port 3000");
  });
```

- This code starts the Express server on port 3000. It listens for incoming HTTP requests and routes them to the appropriate handlers based on the defined routes.

In summary, this code creates a RESTful API using Express to perform CRUD (Create, Read, Update, Delete) operations on a MySQL database for managing products. It defines routes to handle these operations and uses the `mysql2` library to interact with the database. The server listens on port 3000 for incoming requests.