Node-Express assessment

1) Install and start

Bash/ or VS code terminal

NPM install

NPM start

- NPM install: downloads the packages.

- NPM start : runs server.js and starts the API.

2) Environment file

Create a file named .env in the project root:

env

PORT=3000

MONGODB_URI=mongodb://localhost:27017/express-lessons

- PORT` is the server port.

- MONGODB_URI` is the database address. Change it if you use MongoDB Atlas.

3) Main file (server.js)

javascript

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

const mongoose = require("mongoose");

require("dotenv").config();
```

```
const app = express();
const PORT = process.env.PORT || 3000;

app.use(express.json());
app.use(express.urlencoded({ extended: true }));

const connectDB = async () => {
  try {
    await mongoose.connect(process.env.MONGODB_URI || "mongodb://localhost:27017/express-lessons");
    console.log("MongoDB connected successfully");
  } catch (error) {
    console.error("MongoDB connection error:", error.message);
    console.log("Server will continue running, but database operations will fail.");
  }
};

connectDB();

app.use("/product", require("./Routes/productRoutes"));
app.use("/order", require("./Routes/orderRoutes"));

app.get("/", (req, res) => {
  res.json({ message: "Welcome to Express API" });
});

app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}`);
});
```

- Connects to MongoDB.
- Adds middleware to read JSON.
- Mounts product and order routes.
- Starts the server.

4) Models (database schemas)

Product model — `Models/productModel.js`

javascript

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

const productSchema = new mongoose.Schema({
    name: { type: String, required: true, trim: true },
    price: { type: Number, required: true, min: 0 },
    category: { type: String, required: true, trim: true },
    stock: { type: Number, required: true, min: 0, default: 0 }
}, { timestamps: true });

module.exports = mongoose.model("Product", productSchema);
```

- Defines product fields: name, price, category, stock.
- Adds timestamps.

Order model — `Models/orderModel.js`

javascript

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

const orderSchema = new mongoose.Schema({
   customerName: { type: String, required: true, trim: true },
   productId: { type: mongoose.Schema.Types.ObjectId, ref: "Product", required: true },
   quantity: { type: Number, required: true, min: 1 },
   totalAmount: { type: Number, required: true, min: 0 },
   sender: {
      name: { type: String, required: true, trim: true, default: "gulleid mohamed farah" },
      id: { type: String, required: true, default: "4867444" }
   },
   receiver: {
      name: { type: String, required: true, trim: true, default: "hooyo hinda hussein handulle" },
      id: { type: String, required: true, default: "4115165" }
   },
   status: { type: String, enum: ["pending", "processing", "completed", "cancelled"], default: "pending" }
}, { timestamps: true });

module.exports = mongoose.model("Order", orderSchema);
```

- Defines order fields and default sender/receiver data.
- Adds status with allowed values.

---

5) Controllers (business logic)

Product controller — `Controllers/productController.js`

Key functions:

- getAllProducts — list all products.
- getProductById — get one product.
- createProduct — add a product.
- updateProduct — edit a product.
- deleteProduct — remove a product.

Order controller — `Controllers/orderController.js`

Key functions:

- getAllOrders — list all orders (includes sender/receiver).
- getOrderById — get one order (includes sender/receiver).
- createOrder — make an order; uses defaults for sender/receiver if missing.
- updateOrder — edit order (can change sender/receiver).
- updateOrderStatus — change status.
- deleteOrder — remove an order.

6) Routes (API endpoints)

Product routes — `Routes/productRoutes.js`

text

```
GET    /product            -> getAllProducts
GET    /product/:id        -> getProductById
POST   /product/create     -> createProduct
PUT    /product/update/:id -> updateProduct
```

DELETE /product/:id -> deleteProduct

Order routes — `Routes/orderRoutes.js`

```text
GET /order -> getAllOrders
GET /order/:id -> getOrderById
POST /order/create -> createOrder
PUT /order/update/:id -> updateOrder
PUT /order/changestatus/:id -> updateOrderStatus
DELETE /order/:id -> deleteOrder
```

- These routes connect URLs to controller functions.

---

7) Sample requests (use Postman or curl)

Create a product

bash

POST http://localhost:3000/product/create

Content-Type: application/json

Create an order (with sender/receiver)

bash

POST http://localhost:3000/order/create

Content-Type: application/json

{

"customerName": "gulleid Mohamed ",

"productId": "<PUT_PRODUCT_ID_HERE>",

"quantity": 2,

"sender": {

"name": "gulleid mohamed farah",

"id": "4867444"

},

"receiver": {

"name": "hooyo hinda hussein handulle",

"id": "4115165"

}

}

```

If you do not send `sender` or `receiver`, the defaults above are used.

**Update order status**

bash

PUT http://localhost:3000/order/changestatus/<ORDER_ID>

Content-Type: application/json
```

{ "status": "completed" }

8) What happens when server runs

1. server.js loads .env and connects to MongoDB.
2. It registers product and order routes.
3. When you call an endpoint, the route calls the controller.
4. The controller reads/writes data through the Mongoose models.
5. Responses include sender and receiver data for orders.

9) Defaults for sender/receiver

-Sender: name `gulleid mohamed farah`, id `4867444`

- Receiver: name `hooyo hinda hussein handulle`, id `4115165`
- These are used automatically if you do not pass them in the request body.

10) Quick checks

- Server running? Look for: Server is running on port 3000
- DB connected? Look for: MongoDB connected successfully
- Root endpoint: ‘http://localhost:3000` should return a JSON welcome message.