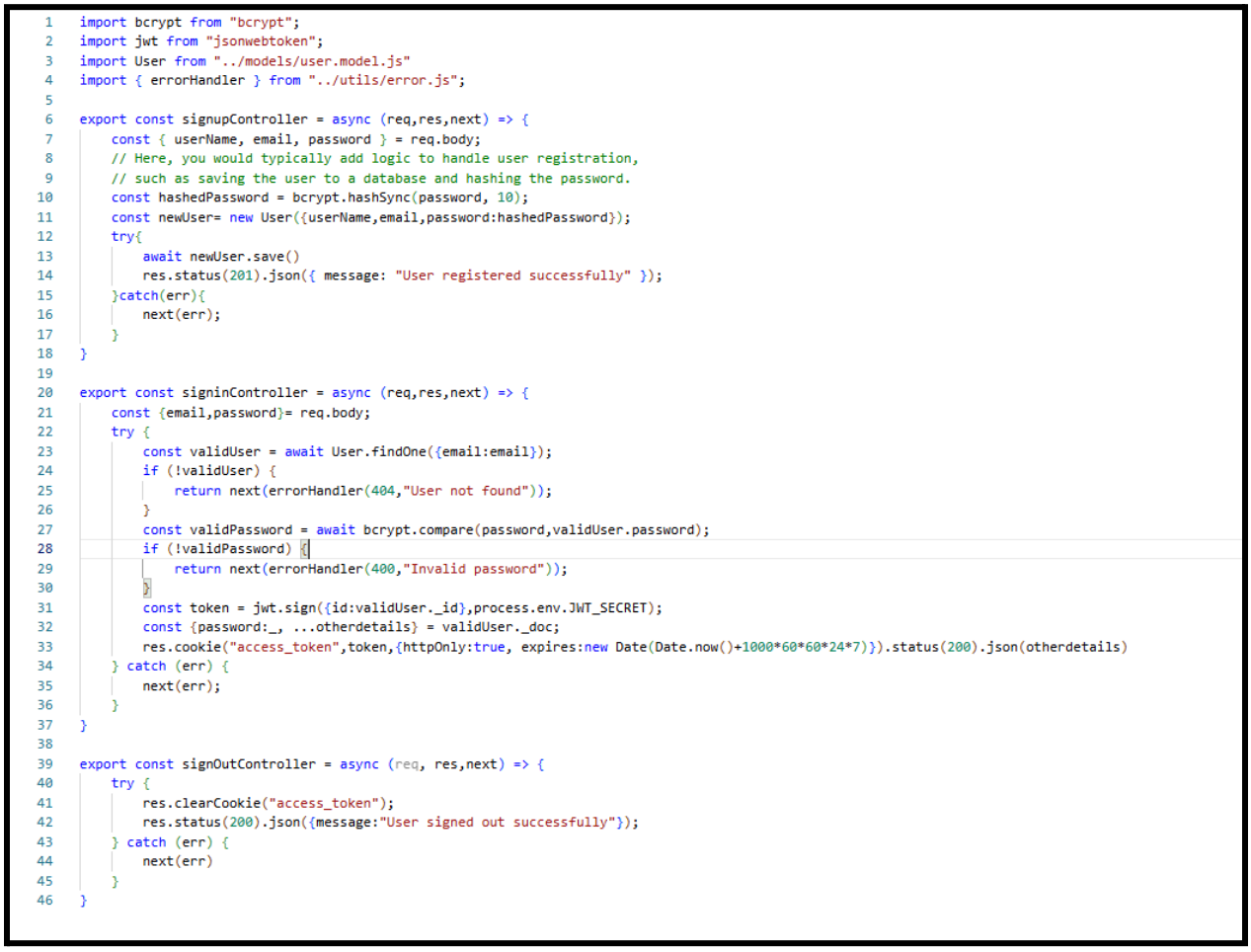
Source Code:

1) auth.controller.js:



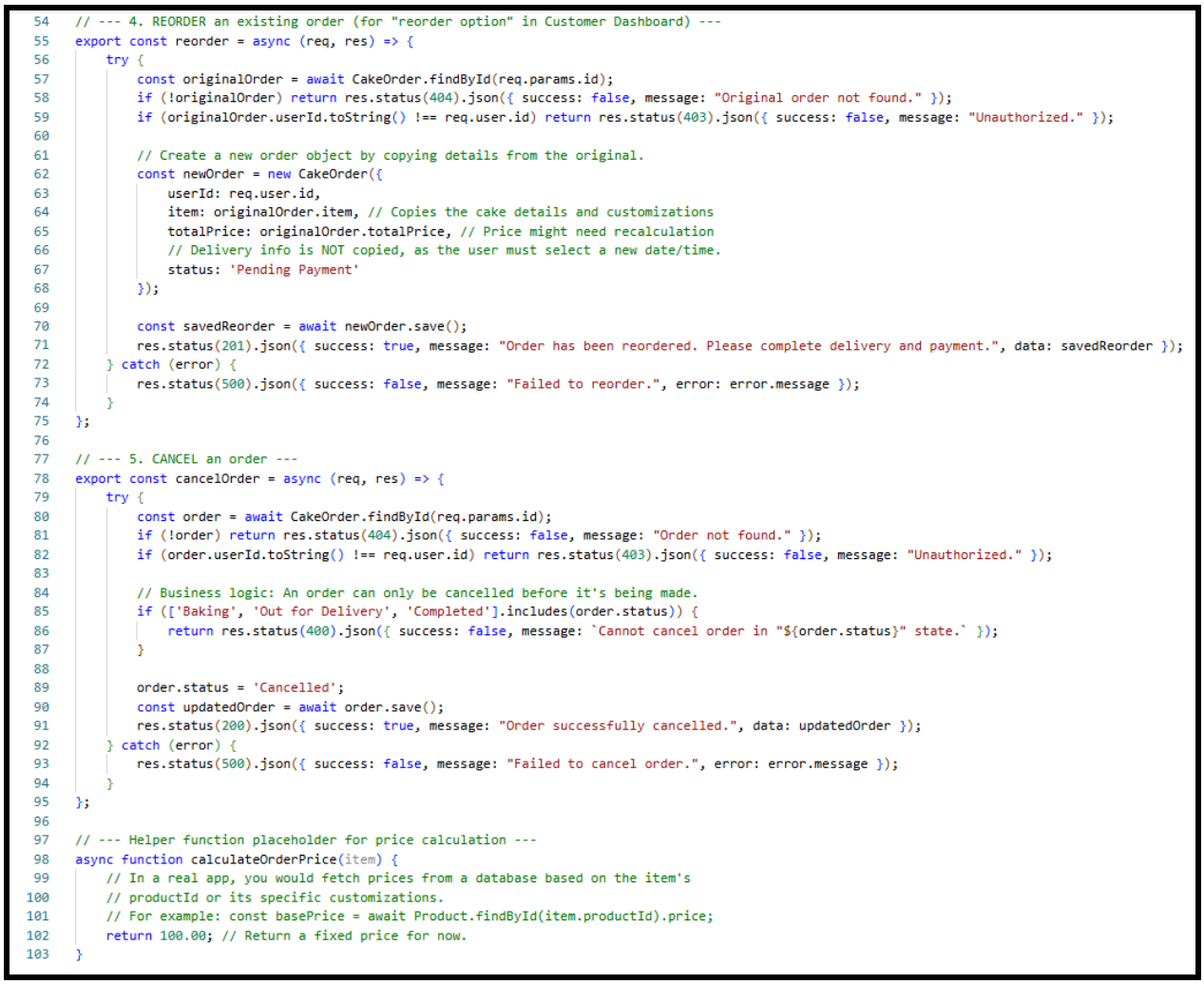
```
import bcrypt from "bcrypt";
import jwt from "jsonwebtoken";
import User from "../models/user.model.js"
import { errorHandler } from "../utils/error.js";

export const signupController = async (req,res,next) => {
    const { userName, email, password } = req.body;
    // Here, you would typically add logic to handle user registration,
    // such as saving the user to a database and hashing the password.
    const hashedPassword = bcrypt.hashSync(password, 10);
    const newUser= new User({userName,email,password:hashedPassword});
    try{
        await newUser.save()
        res.status(201).json({ message: "User registered successfully" });
    }catch(err){
        next(err);
    }
}

export const signinController = async (req,res,next) => {
    const {email,password}= req.body;
    try {
        const validUser = await User.findOne({email:email});
        if (!validUser) {
            return next(errorHandler(404,"User not found"));
        }
        const validPassword = await bcrypt.compare(password,validUser.password);
        if (!validPassword) {
            return next(errorHandler(400,"Invalid password"));
        }
        const token = jwt.sign({id:validUser._id},process.env.JWT_SECRET);
        const {password:_, ...otherdetails} = validUser._doc;
        res.cookie("access_token",token,{httpOnly:true, expires:new Date(Date.now()+1000*60*60*24*7)}).status(200).json(otherdetails)
    } catch (err) {
        next(err);
    }
}

export const signOutController = async (req, res,next) => {
    try {
        res.clearCookie("access_token");
        res.status(200).json({message:"User signed out successfully"});
    } catch (err) {
        next(err)
    }
}
```

Figure 1

2) cakeorder.controller.js:

```
import CakeOrder from '../models/cakeorder.model.js';
import Product from '../models/product.model.js';

// --- 1. CREATE a new order ---
export const createOrder = async (req, res) => {
    try {
        // Step 1: Assume req.user.id is available from an authentication middleware.
        const userId = req.user.id;

        // Step 2: (Workflow Point #3) Server-side price calculation logic would go here.
        // This is a placeholder and should be replaced with a real calculation.
        const calculatedPrice = await calculateOrderPrice(req.body.item);

        const newOrder = new CakeOrder({
            userId,
            item: req.body.item,
            totalPrice: calculatedPrice,
            deliveryInfo: req.body.deliveryInfo,
            status: 'Pending Payment' // Initial status before payment is confirmed.
        });

        const savedOrder = await newOrder.save();
        // The next step in the flow would be to initiate payment with this savedOrder._id.
        res.status(201).json({ success: true, message: "Order created and awaiting payment.", data: savedOrder });
    } catch (error) {
        res.status(500).json({ success: false, message: "Failed to create order.", error: error.message });
    }
};

// --- 2. GET all orders for the logged-in user (for "Customer Dashboard") ---
export const getOrderHistory = async (req, res) => {
    try {
        const orders = await CakeOrder.find({ userId: req.user.id }).sort({ createdAt: -1 });
        res.status(200).json({ success: true, data: orders });
    } catch (error) {
        res.status(500).json({ success: false, message: "Failed to fetch order history.", error: error.message });
    }
};

// --- 3. GET details of a single order (for "track live orders") ---
export const getOrderDetails = async (req, res) => {
    try {
        const order = await CakeOrder.findById(req.params.id);
        if (!order) return res.status(404).json({ success: false, message: "Order not found." });
        if (order.userId.toString() !== req.user.id) return res.status(403).json({ success: false, message: "Unauthorized." });
        res.status(200).json({ success: true, data: order });
    } catch (error) {
        res.status(500).json({ success: false, message: "Failed to fetch order details.", error: error.message });
    }
};
```

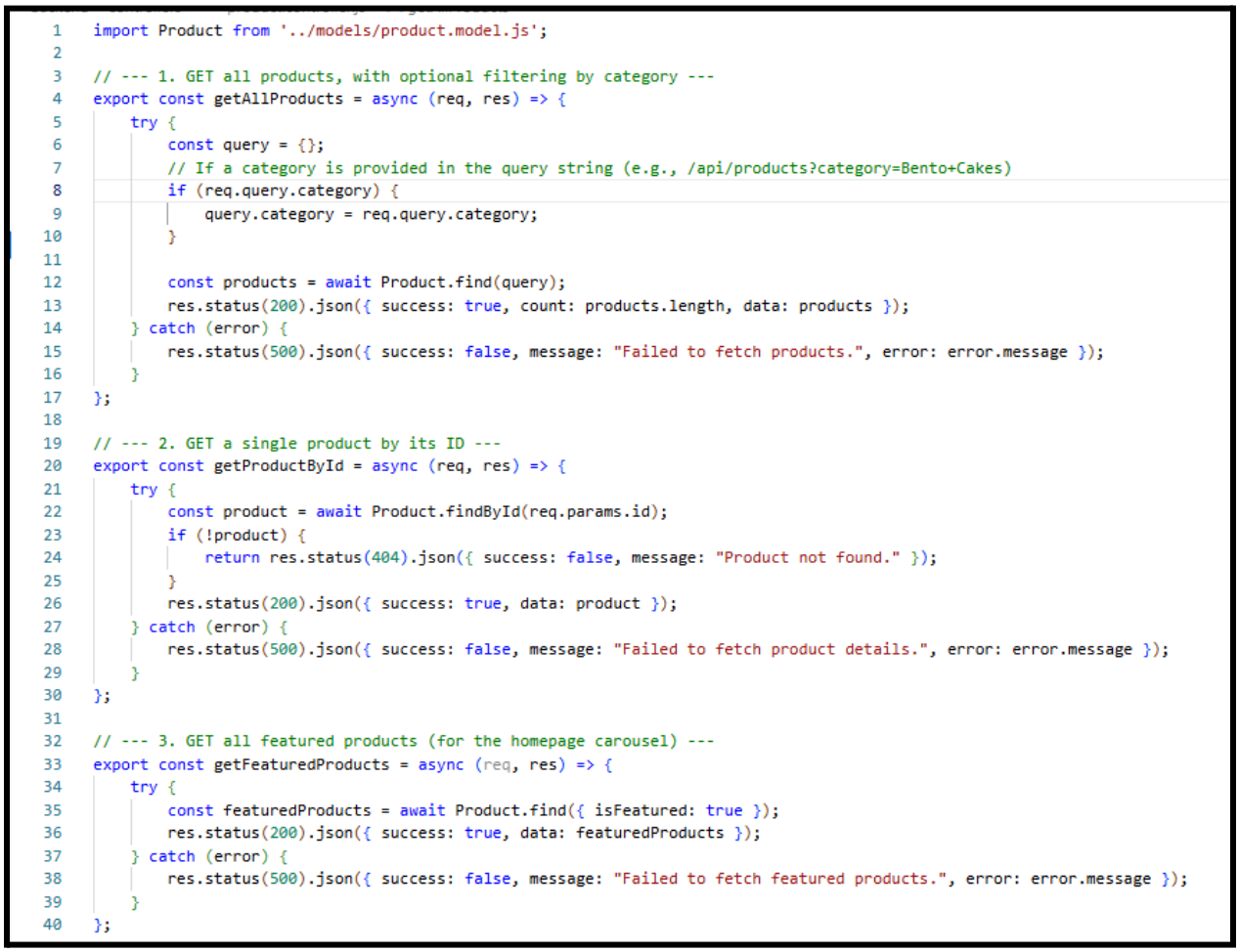
Figure 2



```
// --- 4. REORDER an existing order (for "reorder option" in Customer Dashboard) ---
export const reorder = async (req, res) => {
    try {
        const originalOrder = await CakeOrder.findById(req.params.id);
        if (!originalOrder) return res.status(404).json({ success: false, message: "Original order not found." });
        if (originalOrder.userId.toString() !== req.user.id) return res.status(403).json({ success: false, message: "Unauthorized." });

        // Create a new order object by copying details from the original.
        const newOrder = new CakeOrder({
            userId: req.user.id,
            item: originalOrder.item, // Copies the cake details and customizations
            totalPrice: originalOrder.totalPrice, // Price might need recalculation
            // Delivery info is NOT copied, as the user must select a new date/time.
            status: 'Pending Payment'
        });

        const savedReorder = await newOrder.save();
        res.status(201).json({ success: true, message: "Order has been reordered. Please complete delivery and payment.", data: savedReorder });
    } catch (error) {
        res.status(500).json({ success: false, message: "Failed to reorder.", error: error.message });
    }
};

// --- 5. CANCEL an order ---
export const cancelOrder = async (req, res) => {
    try {
        const order = await CakeOrder.findById(req.params.id);
        if (!order) return res.status(404).json({ success: false, message: "Order not found." });
        if (order.userId.toString() !== req.user.id) return res.status(403).json({ success: false, message: "Unauthorized." });

        // Business logic: An order can only be cancelled before it's being made.
        if (['Baking', 'Out for Delivery', 'Completed'].includes(order.status)) {
            return res.status(400).json({ success: false, message: `Cannot cancel order in "${order.status}" state.` });
        }

        order.status = 'Cancelled';
        const updatedOrder = await order.save();
        res.status(200).json({ success: true, message: "Order successfully cancelled.", data: updatedOrder });
    } catch (error) {
        res.status(500).json({ success: false, message: "Failed to cancel order.", error: error.message });
    }
};

// --- Helper function placeholder for price calculation ---
async function calculateOrderPrice(item) {
    // In a real app, you would fetch prices from a database based on the item's
    // productId or its specific customizations.
    // For example: const basePrice = await Product.findById(item.productId).price;
    return 100.00; // Return a fixed price for now.
}
```

Figure 3

3) product.controller.js:



```
import Product from '../models/product.model.js';

// --- 1. GET all products, with optional filtering by category ---
export const getAllProducts = async (req, res) => {
    try {
        const query = {};
        // If a category is provided in the query string (e.g., /api/products?category=Bento+Cakes)
        if (req.query.category) {
            query.category = req.query.category;
        }

        const products = await Product.find(query);
        res.status(200).json({ success: true, count: products.length, data: products });
    } catch (error) {
        res.status(500).json({ success: false, message: "Failed to fetch products.", error: error.message });
    }
};

// --- 2. GET a single product by its ID ---
export const getProductById = async (req, res) => {
    try {
        const product = await Product.findById(req.params.id);
        if (!product) {
            return res.status(404).json({ success: false, message: "Product not found." });
        }
        res.status(200).json({ success: true, data: product });
    } catch (error) {
        res.status(500).json({ success: false, message: "Failed to fetch product details.", error: error.message });
    }
};

// --- 3. GET all featured products (for the homepage carousel) ---
export const getFeaturedProducts = async (req, res) => {
    try {
        const featuredProducts = await Product.find({ isFeatured: true });
        res.status(200).json({ success: true, data: featuredProducts });
    } catch (error) {
        res.status(500).json({ success: false, message: "Failed to fetch featured products.", error: error.message });
    }
};
```

Figure 4

4) user.controller.js:

```
import User from "../models/user.model.js";
import bcrypt from "bcrypt";
import { errorHandler } from "../utils/error.js";

export const test = (req, res) => {
    res.json({ message: "User route is working!!!" });
}

export const updateUser = async(req,res,next)=>{
    if (req.user.id !== req.params.id) return next(errorHandler(401,"You can update only your account"));
    try {
        if (req.body.password) {
            req.body.password = bcrypt.hashSync(req.body.password, 10);
        }
        const updatedUser = await User.findByIdAndUpdate(req.params.id,{
            $set:{
                userName:req.body.userName,
                email:req.body.email,
                password:req.body.password,
            }
        },{new:true});
         if (!updatedUser) {
      return next(errorHandler(404, "User not found"));
    }

        const {password, ...otherdetails} = updatedUser._doc;
        res.status(200).json(otherdetails);

    } catch (err) {
        next(err);
    }
}

export const deleteUser = async(req,res,next)=>{
   if (req.user.id !== req.params.id) return next(errorHandler(401,"You can delete only your account"));
   try {
     await User.findByIdAndDelete(req.params.id);
     res.clearCookie("access_token");
        res.status(200).json({message:"User deleted successfully"})
   } catch (err) {
    next(err);
   }
}
```

Figure 5

5) auth.route.js:

```
1  import express from "express";
2  import { signinController,signOutController,signupController } from "../controllers/auth.controller.js";
3
4  const Router = express.Router();
5
6  Router.post("/signup",signupController);
7  Router.post("/signin",signinController);
8  Router.post("/signout",signOutController);
9
10 export default Router;
```

Figure 6

6) cakeorder.route.js:

```
1  import express from 'express';
2  const router = express.Router();
3  import {
4      createOrder,
5      getOrderHistory,
6      getOrderDetails,
7      cancelOrder,
8      reorder
9  } from '../controllers/cakeorder.controller.js';
10
11 // In a real app, an authentication middleware would be placed here to protect all order routes
12 // and add the 'req.user' object to requests. e.g., router.use(authMiddleware);
13
14 import { verifyToken } from '../utils/verifyUser.js'; // Adjust the path if it's different
15 router.use(verifyToken);
16
17 // POST /api/orders - Creates a new order.
18 router.post('/', createOrder);
19
20 // GET /api/orders - Gets the order history for the logged-in user.
21 router.get('/', getOrderHistory);
22
23 // GET /api/orders/:id - Gets the details of a single, specific order for tracking.
24 router.get('/:id', getOrderDetails);
25
26 // POST /api/orders/:id/reorder - Creates a new order based on a previous one.
27 router.post('/:id/reorder', reorder);
28
29 // PATCH /api/orders/:id/cancel - Marks a specific order as cancelled.
30 router.patch('/:id/cancel', cancelOrder);
31
32 export default router;
33
```

Figure 7

7) product.route.js:

```
1   import express from 'express';
2   const router = express.Router();
3   import {
4       getAllProducts,
5       getProductById,
6       getFeaturedProducts
7   } from '../controllers/product.controller.js';
8
9   // GET /api/products - Gets all products.
10  // Can be filtered with a query, e.g., /api/products?category=Birthday+Cakes
11  router.get('/', getAllProducts);
12
13  // GET /api/products/featured - Gets only the featured products for the homepage.
14  router.get('/featured', getFeaturedProducts);
15
16  // GET /api/products/:id - Gets a single product by its unique ID.
17  router.get('/:id', getProductById);
18
19  export default router;
```

Figure 8

8) user.route.js:

```
1   import express from "express"
2   import {test,updateUser,deleteUser} from "../controllers/user.controller.js"
3   import { verifyToken } from "../utils/verifyUser.js";
4
5   const UserRouter = express.Router();
6
7   UserRouter.get("/", test );
8   UserRouter.put("/update/:id", verifyToken, updateUser)
9   UserRouter.delete("/delete/:id", verifyToken, deleteUser)
10
11  export default UserRouter
```

Figure 9

9) server.js:

```
import dotenv from "dotenv";
dotenv.config();

import express from 'express';
import cookieParser from 'cookie-parser'; // 1. Import cookie-parser
import connectDB from './config/db.js';

// --- Import all your modular routers ---
import authRouter from "./routes/auth.route.js";
import userRouter from "./routes/user.route.js";
import productRouter from "./routes/product.router.js";
import cakeOrderRouter from "./routes/cakeorder.router.js"; // 2. Corrected router name

// Connect to the database
connectDB();

// Initialize the Express app
const app = express();

// Set the port from environment variables
const PORT = process.env.PORT || 5000;

// --- Middleware ---
app.use(express.json()); // 3. Modern replacement for bodyParser
app.use(cookieParser()); // 4. Use cookie-parser for auth tokens

// --- Define All API Routes ---
app.use('/api/auth', authRouter);
app.use('/api/user', userRouter);
app.use('/api/products', productRouter);
app.use('/api/orders', cakeOrderRouter); // 5. Updated route for orders

// A simple default route to confirm the server is running
app.get('/', (req, res) => {
    res.send('Welcome to the Cuppie Cake API!');
});

// Start listening for requests
app.listen(PORT, () => {
    console.log(`Server is running in ${process.env.NODE_ENV || 'development'} mode on port ${PORT}`);
});
```

Figure 10

Output:

1) signup:

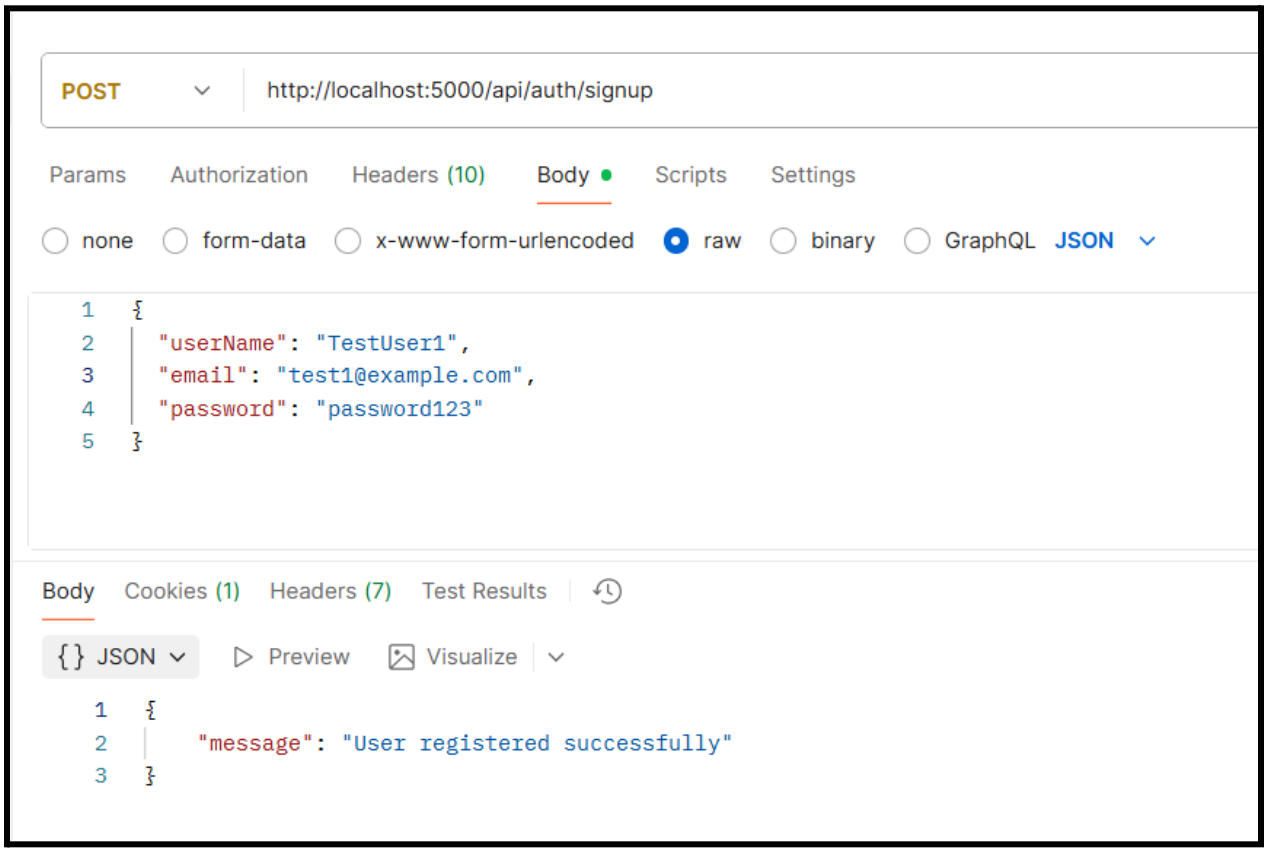


Figure 1

2) signin:

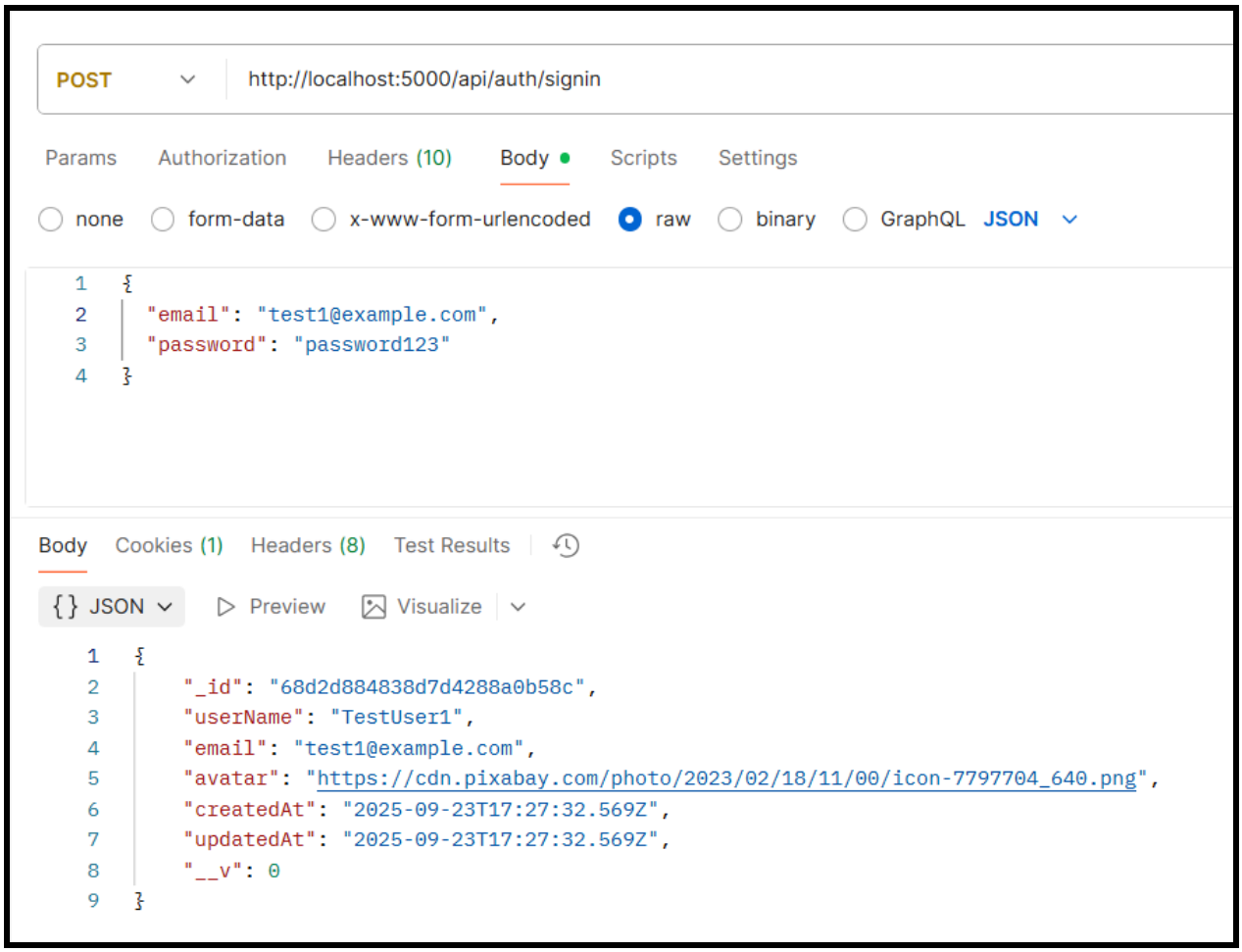

POST
http://localhost:5000/api/auth/signin
Params
Authorization
Headers (10)
Body
Scripts
Settings
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
{
"email": "test1@example.com",
"password": "password123"
}
Body
Cookies (1)
Headers (8)
Test Results
{} JSON
Preview
Visualize
{
"_id": "68d2d884838d7d4288a0b58c",
"userName": "TestUser1",
"email": "test1@example.com",
"avatar": "https://cdn.pixabay.com/photo/2023/02/18/11/00/icon-7797704_640.png",
"createdAt": "2025-09-23T17:27:32.569Z",
"updatedAt": "2025-09-23T17:27:32.569Z",
"__v": 0
}

Figure 2

3) get products:

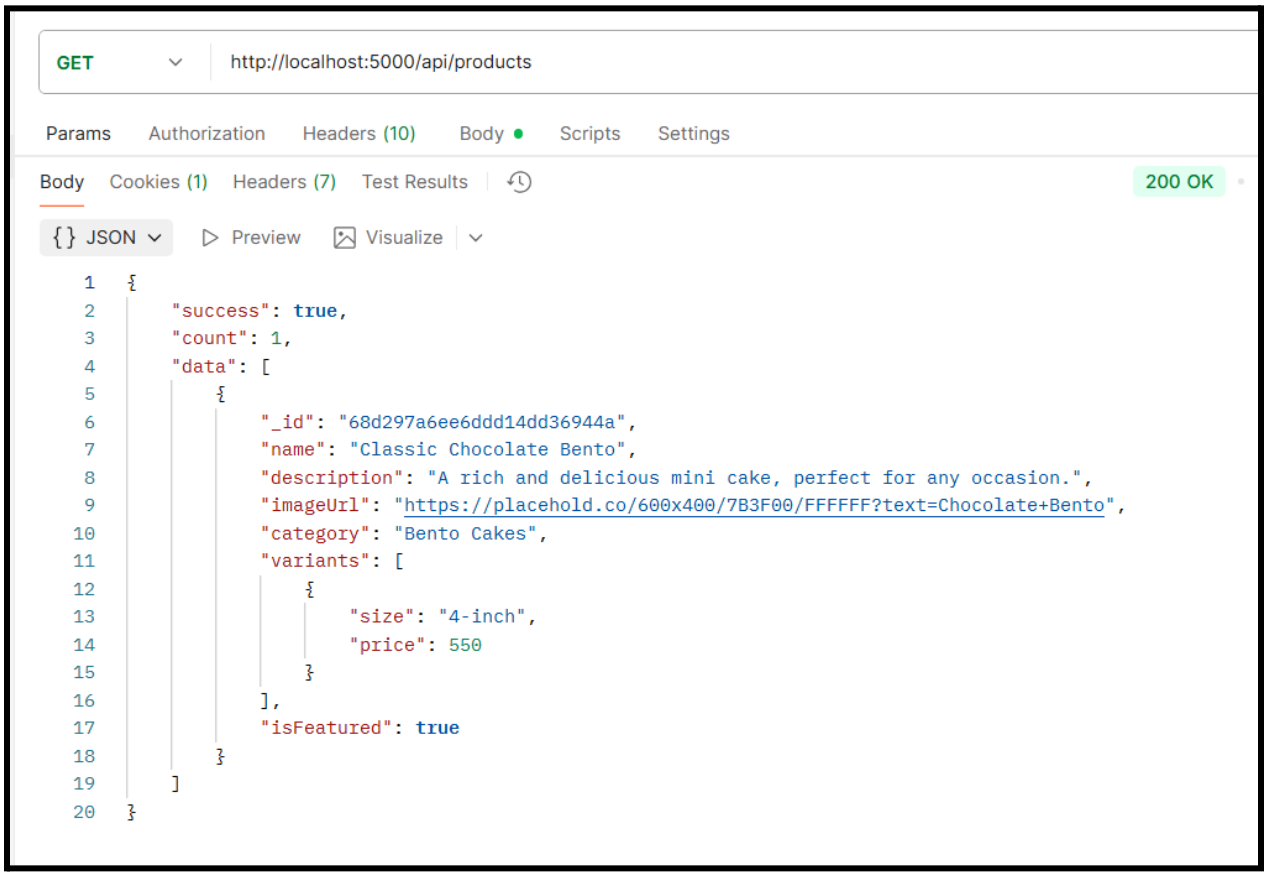


Figure 3

4) get featured products:

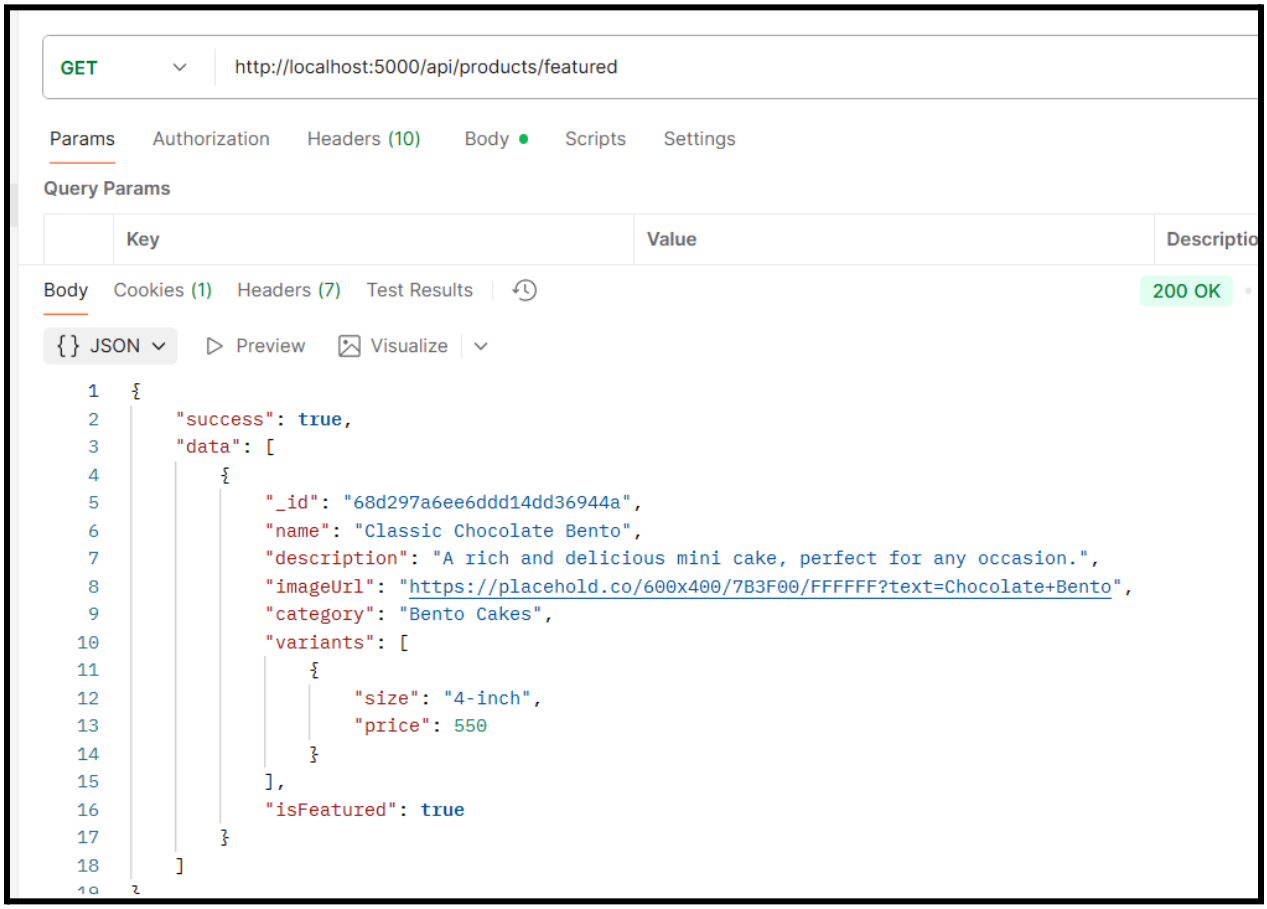


Figure 4

5) create order:

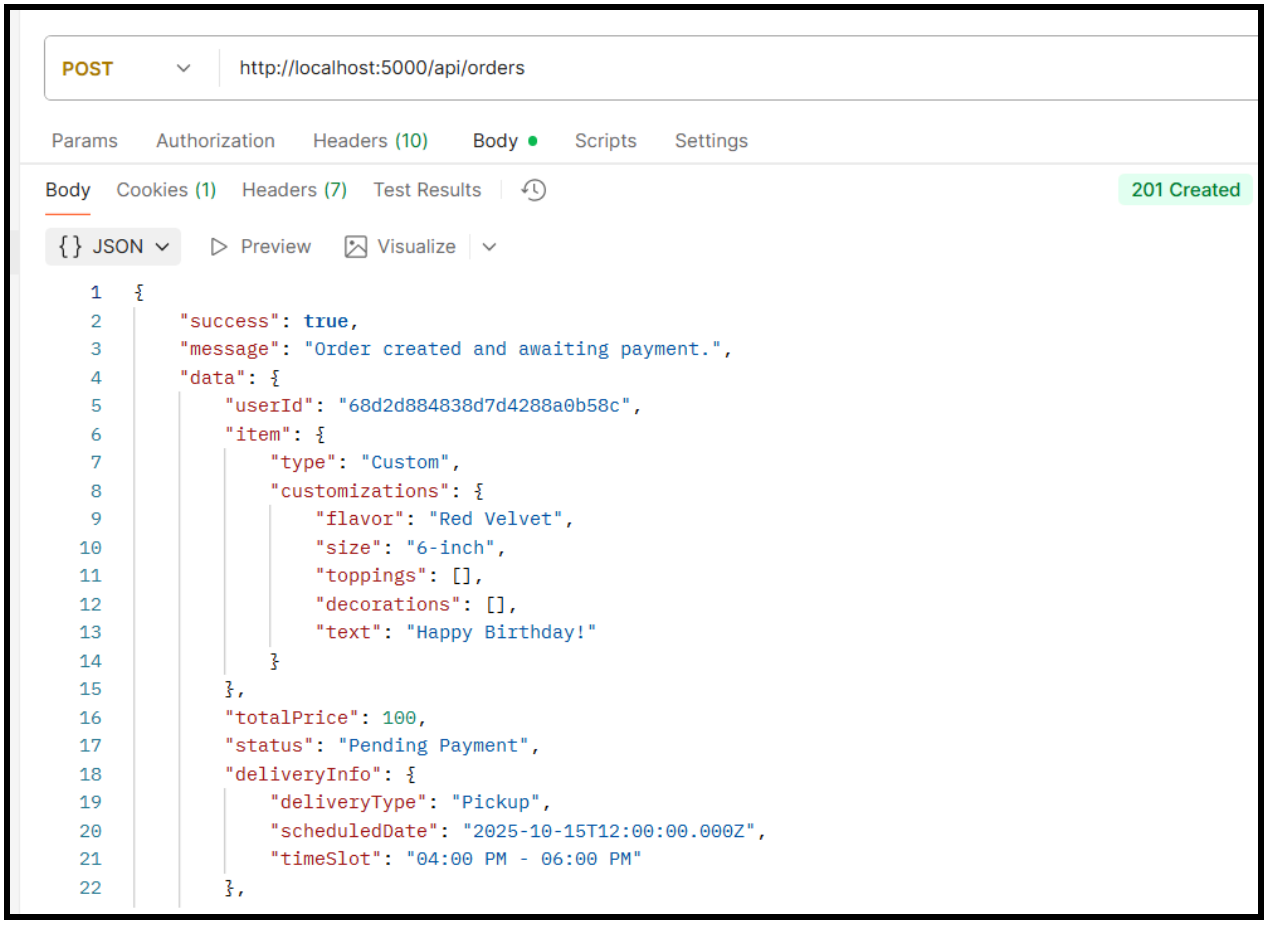

POST
http://localhost:5000/api/orders
Params
Authorization
Headers (10)
Body
Scripts
Settings
Body
Cookies (1)
Headers (7)
Test Results
201 Created
JSON
Preview
Visualize
{
"success": true,
"message": "Order created and awaiting payment.",
"data": {
"userId": "68d2d884838d7d4288a0b58c",
"item": {
"type": "Custom",
"customizations": {
"flavor": "Red Velvet",
"size": "6-inch",
"toppings": [],
"decorations": [],
"text": "Happy Birthday!"
}
},
"totalPrice": 100,
"status": "Pending Payment",
"deliveryInfo": {
"deliveryType": "Pickup",
"scheduledDate": "2025-10-15T12:00:00.000Z",
"timeSlot": "04:00 PM - 06:00 PM"
},

Figure 5

6) order history:

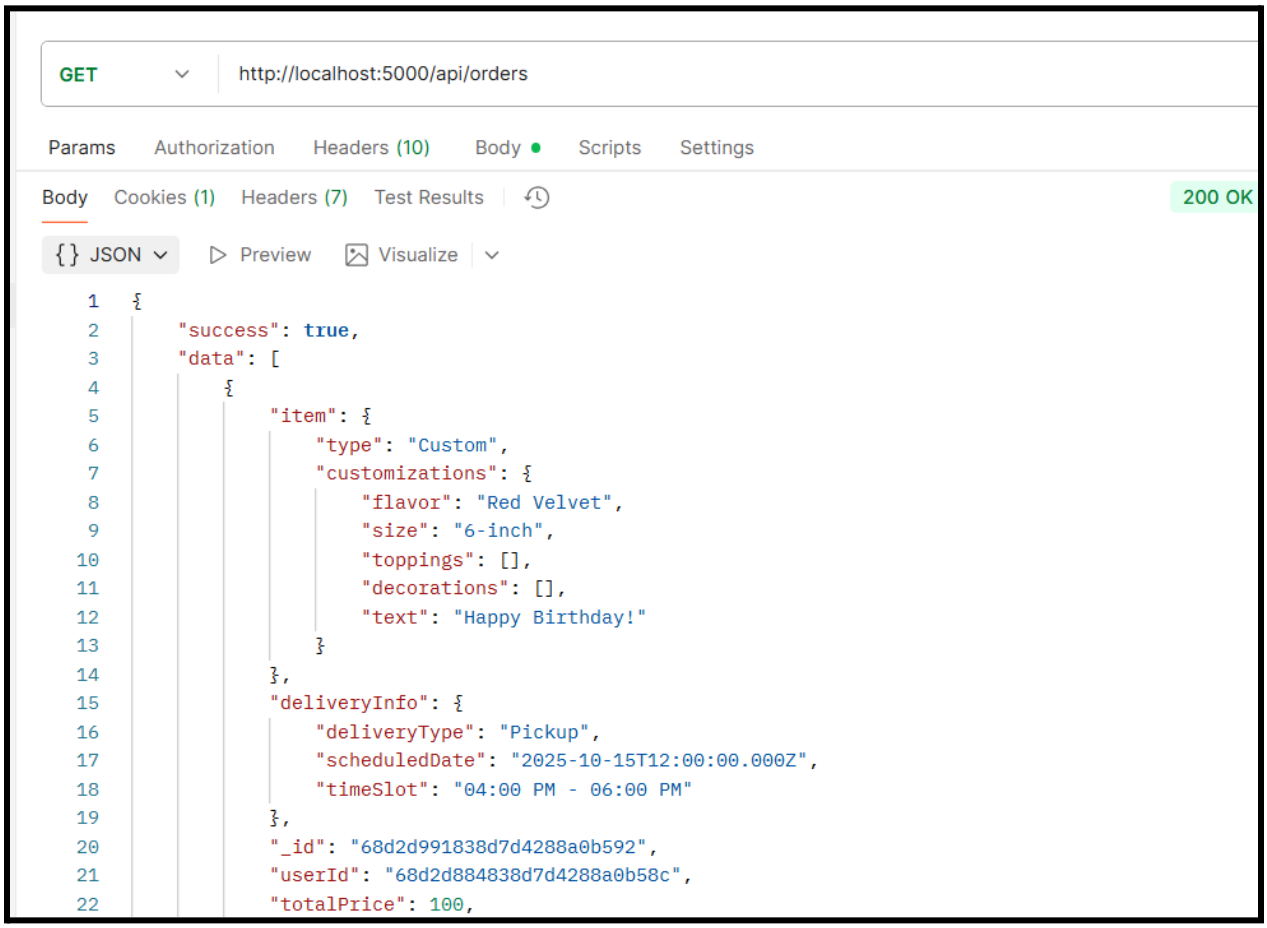


Figure 6

7) specific order:

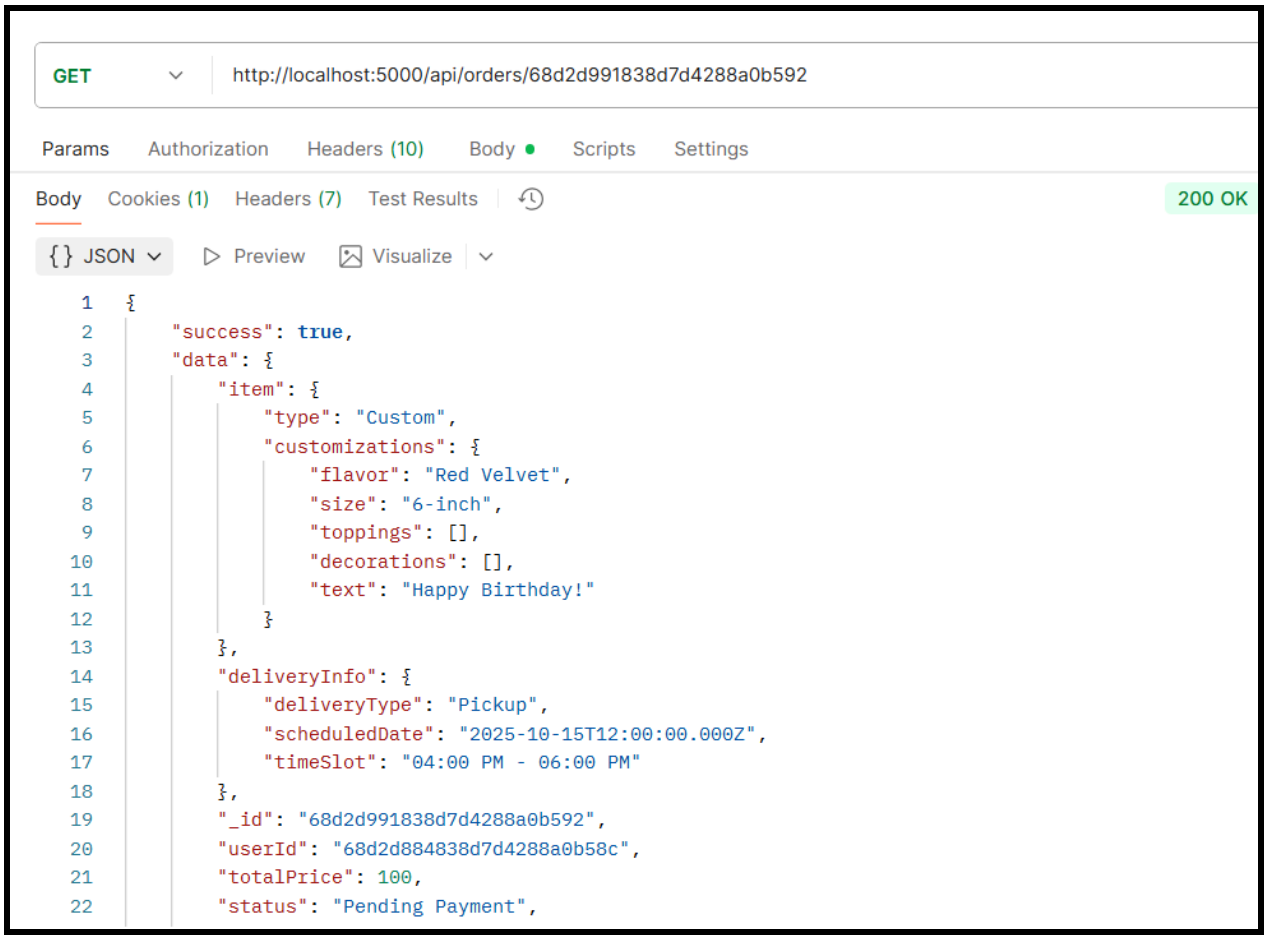

GET
http://localhost:5000/api/orders/68d2d991838d7d4288a0b592
Params
Authorization
Headers (10)
Body
Scripts
Settings
Body
Cookies (1)
Headers (7)
Test Results
200 OK
JSON
Preview
Visualize
{
"success": true,
"data": {
"item": {
"type": "Custom",
"customizations": {
"flavor": "Red Velvet",
"size": "6-inch",
"toppings": [],
"decorations": [],
"text": "Happy Birthday!"
}
},
"deliveryInfo": {
"deliveryType": "Pickup",
"scheduledDate": "2025-10-15T12:00:00.000Z",
"timeSlot": "04:00 PM - 06:00 PM"
},
"_id": "68d2d991838d7d4288a0b592",
"userId": "68d2d884838d7d4288a0b58c",
"totalPrice": 100,
"status": "Pending Payment",

Figure 7