

# DTS NodeJS Capstone Assignment

Name: Atharva Athanikar 72287317E

Batch: DTS Batch-3

## Structure:

- 1) Stocks-trade-api
  - i) Config
    - (a) db.cjs
  - ii) controllers
    - (a) tradeController.cjs
  - iii) middlewares
    - (a) authMiddleware.cjs
  - iv) models
    - (a) trade.cjs
  - v) routes
    - (a) tradeRoutes.cjs
  - vi) tradeApi.cjs

## ./config/db.cjs

```
const mongoose = require("mongoose");
mongoose.connect("mongodb://localhost:27017/stocksTrade", {
  useNewUrlParser: true,
  useUnifiedTopology: true,
})
.then(() => console.log("Connected to MongoDB"))
.catch((err) => console.log("Error connecting to MongoDB:", err));
```

## ./controllers/tradeController.cjs

```
const Trade = require("../models/trade.cjs");

const createTrade = async (req, res) => {
  try {
    const { type, user_id, symbol, shares, price, timestamp } = req.body;
    if (shares < 1 || shares > 100) {
      return res.status(400).json({ message: "Shares value out of range"
});
    }
    if (type !== "buy" && type !== "sell") {
      return res.status(400).json({ message: "Incorrect type provided" });
    }
    const trade = new Trade({ type, user_id, symbol, shares, price,
timestamp });
    await trade.save();
    res.status(201).json(trade);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
};

const getAllTrades = async (req, res) => {
  try {
    const { type, user_id } = req.query;
    let filters = {};
    if (type) filters.type = type;
    if (user_id) filters.user_id = user_id;
    const trades = await Trade.find(filters);
    res.status(200).json(trades);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
};

const getTradeById = async (req, res) => {
  try {
    const trade = await Trade.findById(req.params.id);
    if (!trade) {
      return res.status(404).json({ message: "ID not found" });
    }
    res.status(200).json(trade);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
};

module.exports = { createTrade, getAllTrades, getTradeById };
```

## ./middlewares/authMiddleware.cjs

```
const jwt = require("jsonwebtoken");

const authMiddleware = (req, res, next) => {
  const token = req.header("Authorization")?.split(" ")[1];
  if (!token) return res.status(401).json({ message: "Access denied" });
  try {
    const decoded = jwt.verify(token, "yourSecretKey");
    req.user = decoded;
    next();
  } catch (err) {
    return res.status(400).json({ message: "Invalid token" });
  }
};

module.exports = authMiddleware;
```

## ./models/trade.cjs

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

const tradeSchema = new mongoose.Schema({
  type: { type: String, enum: ["buy", "sell"], required: true },
  user_id: { type: Number, required: true },
  symbol: { type: String, required: true },
  shares: { type: Number, required: true, min: 1, max: 100 },
  price: { type: Number, required: true },
  timestamp: { type: Number, required: true },
});

module.exports = mongoose.model("Trade", tradeSchema);
```

## ./routes/tradeRoutes.cjs

```
const { Router } = require("express");
const router = Router();
const { createTrade, getAllTrades, getTradeById } =
  require("../controllers/tradeController.cjs");
const authMiddleware = require("../middlewares/authMiddleware.cjs");

// POST /trades - Create a new trade
router.post("/trades", authMiddleware, createTrade);

// GET /trades - Get all trades
router.get("/trades", authMiddleware, getAllTrades);

// GET /trades/:id - Get a trade by id
router.get("/trades/:id", authMiddleware, getTradeById);

// Catch-all for disallowed methods on /trades/:id
router.delete("/trades/:id", (req, res) => {
  res.status(405).json({ message: "Method not allowed" });
});

router.patch("/trades/:id", (req, res) => {
  res.status(405).json({ message: "Method not allowed" });
});

module.exports = router;
```

## tradeApi.cjs / Main app

```
const express = require("express");
const { MongoClient } = require("mongodb");
const { v4: uuidv4 } = require("uuid");

const app = express();
app.use(express.json());

const mongoUrl = "mongodb://127.0.0.1:27017/Capstone-Project_DTS";
const dbName = "stocksTrade";
let db, tradesCollection;

MongoClient.connect(mongoUrl, { useUnifiedTopology: true })
  .then((client) => {
    db = client.db(dbName);
    tradesCollection = db.collection("trades");
  });
```

```

        console.log(`Connected to database: ${dbName}`);
    })
    .catch((error) => {
        console.error("Error connecting to MongoDB:", error);
        process.exit(1);
    });

const trades = [];
app.post("/trades", (req, res) => {
    const { type, user_id, symbol, shares, price, timestamp } = req.body;
    if (!type || !user_id || !symbol || !shares || !price || !timestamp) {
        return res.status(400).json({ message: "Missing required fields"
    });
    }
    if (type !== "buy" && type !== "sell") {
        return res.status(400).json({ message: "Incorrect type provided"
    });
    }
    if (shares < 1 || shares > 100) {
        return res.status(400).json({ message: "Shares value out of range"
    });
    }
    const newTrade = {
        id: uuidv4(),
        type,
        user_id,
        symbol,
        shares,
        price,
        timestamp,
    };
    trades.push(newTrade);
    res.status(201).json(newTrade);
});

app.get("/trades", (req, res) => {
    const { type, user_id } = req.query;
    let filteredTrades = trades;
    if (type) {
        filteredTrades = filteredTrades.filter((trade) => trade.type ===
type);
    }
    if (user_id) {
        filteredTrades = filteredTrades.filter((trade) => trade.user_id
=== parseInt(user_id));
    }
    res.status(200).json(filteredTrades);
});

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

```

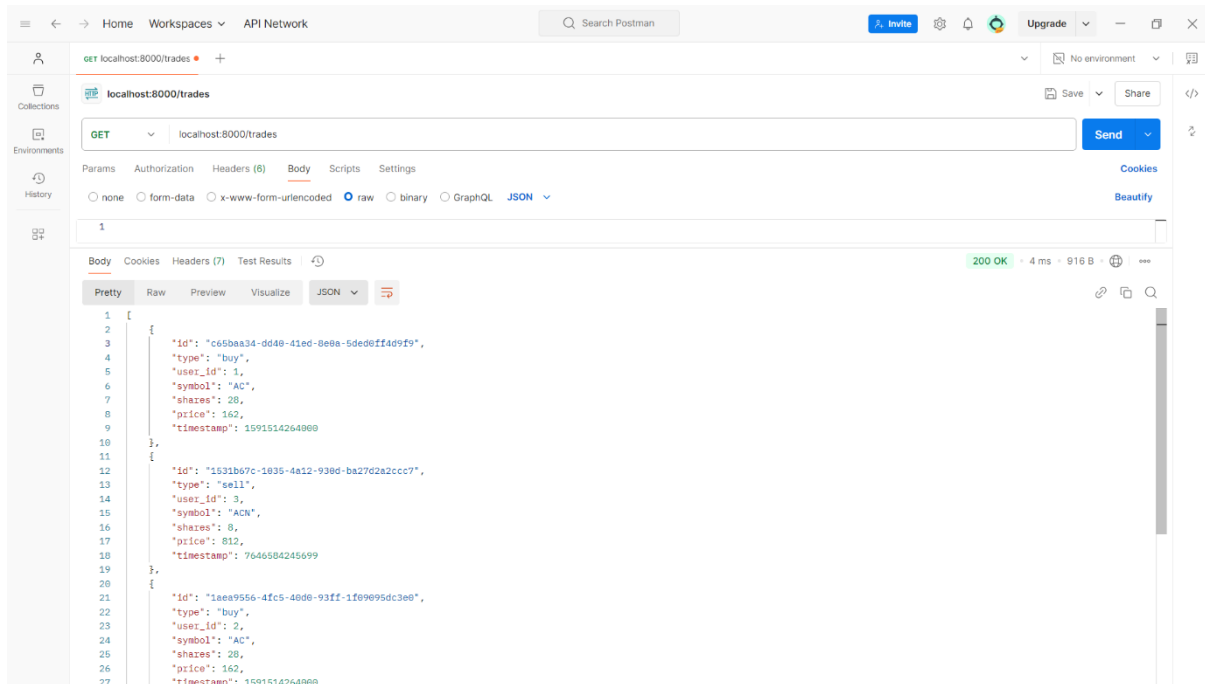
```
    res.status(405).json({ message: "Method Not Allowed" });
  });

app.patch('/trades/:id', (req, res) => {
  res.status(405).json({ message: "Method Not Allowed" });
});

const PORT = 8000;
app.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`);
});
```

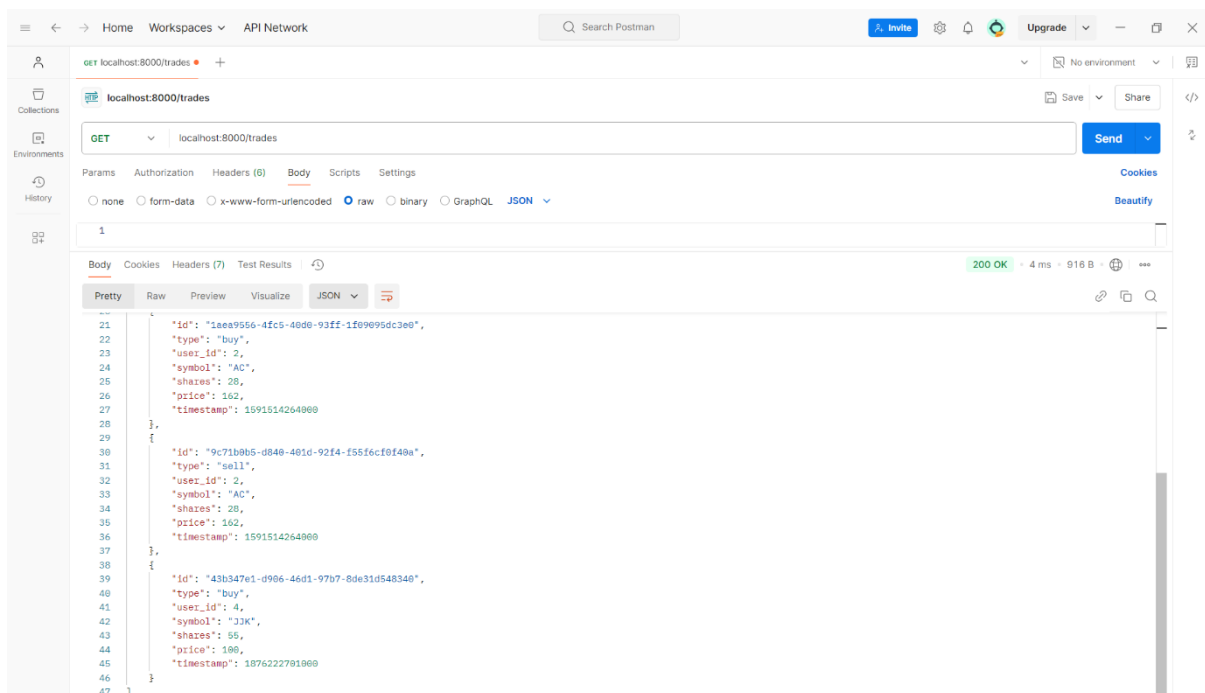
# OUTPUT:

## 1. Each Trade in JSON Entry



Postman interface showing a GET request to `localhost:8000/trades`. The response is a JSON array of 3 trade entries, each with fields: `id`, `type`, `user_id`, `symbol`, `shares`, `price`, and `timestamp`.

```
1 [
2   {
3     "id": "c65baa34-dd48-41ed-8e8a-5ded6ff4d9f9",
4     "type": "buy",
5     "user_id": 1,
6     "symbol": "AC",
7     "shares": 28,
8     "price": 162,
9     "timestamp": 1591514264960
10  },
11  {
12    "id": "1631b67c-1035-4a12-938d-ba27d2a2ccc7",
13    "type": "sell",
14    "user_id": 3,
15    "symbol": "ACN",
16    "shares": 8,
17    "price": 812,
18    "timestamp": 7646584245699
19  },
20  {
21    "id": "1aea9556-4fc5-48d0-93ff-1f09095dc3e0",
22    "type": "buy",
23    "user_id": 2,
24    "symbol": "AC",
25    "shares": 28,
26    "price": 162,
27    "timestamp": 1591514264960
28  }
29 ]
```



Postman interface showing a GET request to `localhost:8000/trades`. The response is a JSON array of 6 trade entries, each with fields: `id`, `type`, `user_id`, `symbol`, `shares`, `price`, and `timestamp`.

```
21 {
22   "id": "1aea9556-4fc5-48d0-93ff-1f09095dc3e0",
23   "type": "buy",
24   "user_id": 2,
25   "symbol": "AC",
26   "shares": 28,
27   "price": 162,
28   "timestamp": 1591514264960
29 },
30 {
31   "id": "9c71b8b5-d840-401d-92f4-f55f6cf0f49a",
32   "type": "sell",
33   "user_id": 2,
34   "symbol": "AC",
35   "shares": 28,
36   "price": 162,
37   "timestamp": 1591514264960
38 },
39 {
40   "id": "43b347e1-d906-46d1-9707-8de31d548340",
41   "type": "buy",
42   "user_id": 4,
43   "symbol": "JJK",
44   "shares": 55,
45   "price": 180,
46   "timestamp": 1876222791960
47 }
```

## 2. POST request to trades

The screenshot shows the Postman interface with a POST request to `localhost:8000/trades`. The request body is a JSON object:

```
1 {
2   "type": "buy",
3   "user_id": 1,
4   "symbol": "AC",
5   "shares": 20,
6   "price": 162,
7   "timestamp": 1591514264000
8 }
9
```

The response is a 201 Created status with a response time of 8 ms and a body size of 375 B. The response body is a JSON object:

```
1 {
2   "id": "017f2e15-012a-4a01-906e-3c49f06ff772",
3   "type": "buy",
4   "user_id": 1,
5   "symbol": "AC",
6   "shares": 20,
7   "price": 162,
8   "timestamp": 1591514264000
9 }
```

The screenshot shows the Postman interface with a POST request to `localhost:8000/trades`. The request body is a JSON object:

```
1 {
2   "type": "buy",
3   "user_id": 2,
4   "symbol": "AC",
5   "shares": 20,
6   "price": 162,
7   "timestamp": 1591514264000
8 }
9
```

The response is a 201 Created status with a response time of 10 ms and a body size of 375 B. The response body is a JSON object:

```
1 {
2   "id": "f63a7c59-7a89-4b49-8254-872a62106347",
3   "type": "buy",
4   "user_id": 2,
5   "symbol": "AC",
6   "shares": 20,
7   "price": 162,
8   "timestamp": 1591514264000
9 }
```



Home Workspaces API Network Search Postman Invite Upgrade

POST localhost:8000/trade: + No environment

localhost:8000/trades Save Share

POST localhost:8000/trades Send

Params Authorization Headers (8) Body Scripts Settings Cookies Beautify

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2   "type": "sell",
3   "user_id": 2,
4   "symbol": "AC",
5   "shares": 28,
6   "price": 162,
7   "timestamp": 1591514264888
8 }
9
```

Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": "7ef933ae-1adb-428f-8e7b-46183be4336d",
3   "type": "sell",
4   "user_id": 2,
5   "symbol": "AC",
6   "shares": 28,
7   "price": 162,
8   "timestamp": 1591514264888
9 }
```

201 Created · 9 ms · 376 B

Home Workspaces API Network Search Postman Invite Upgrade

POST localhost:8000/trade: + No environment

localhost:8000/trades Save Share

POST localhost:8000/trades Send

Params Authorization Headers (8) Body Scripts Settings Cookies Beautify

none form-data x-www-form-urlencoded raw binary GraphQL JSON

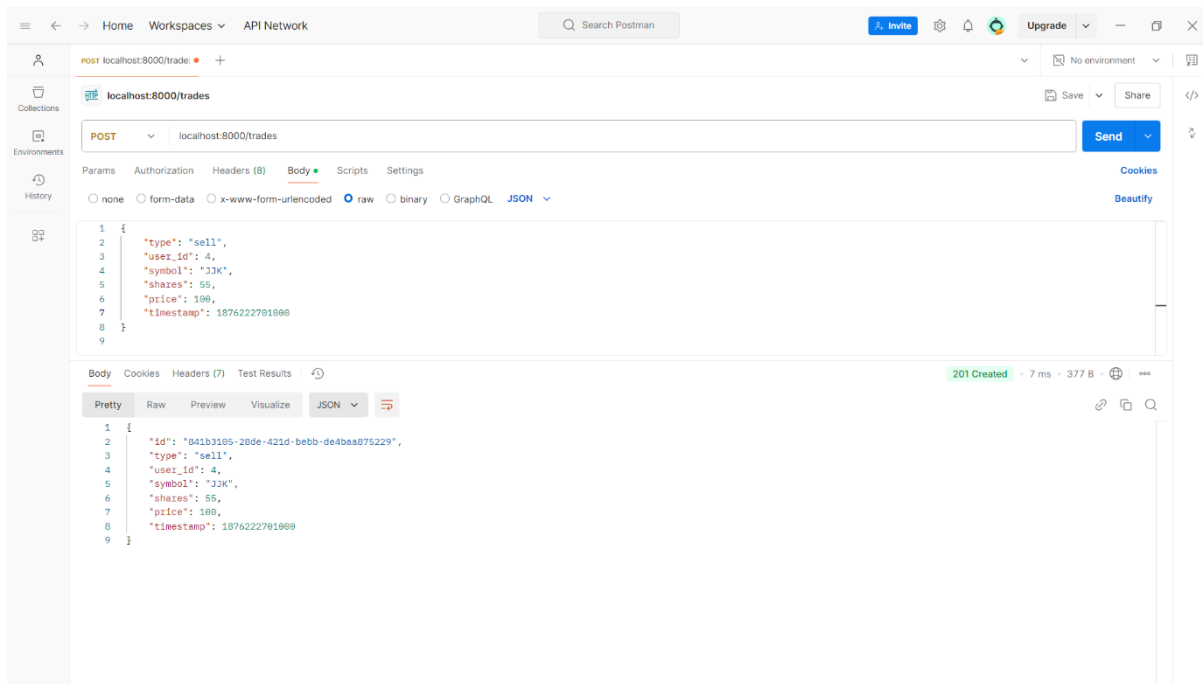
```
1 {
2   "type": "buy",
3   "user_id": 3,
4   "symbol": "ACN",
5   "shares": 8,
6   "price": 812,
7   "timestamp": 7646584245699
8 }
9
```

Body Cookies Headers (7) Test Results

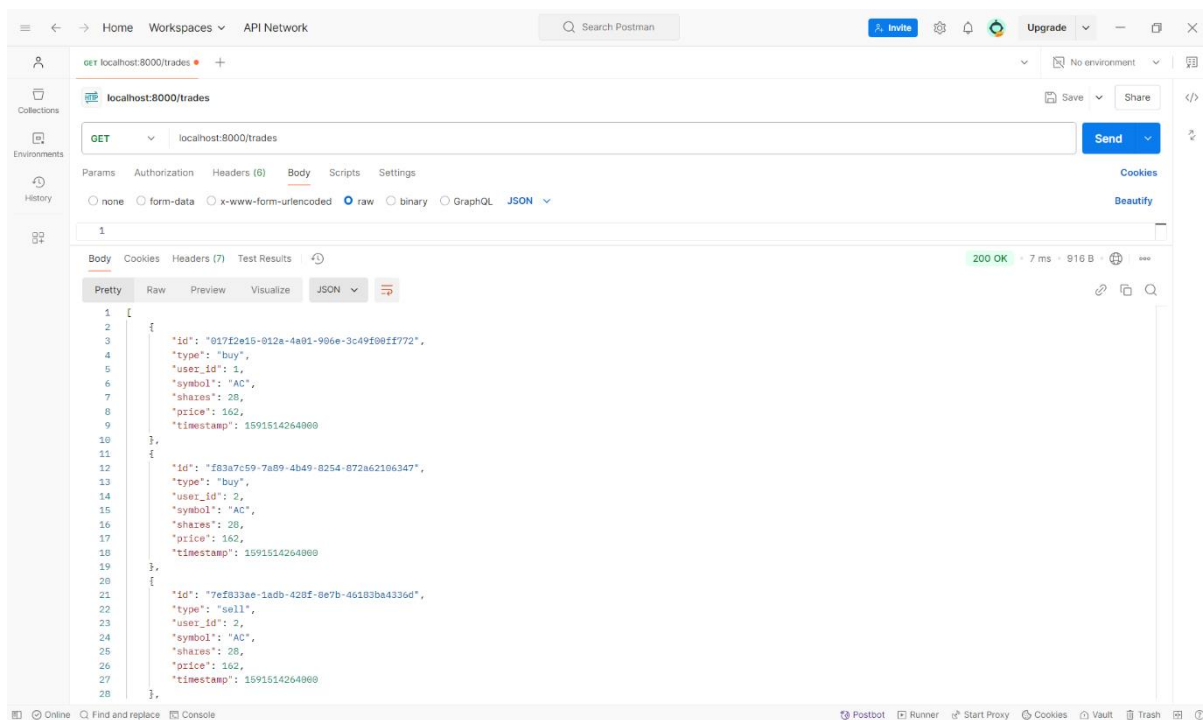
Pretty Raw Preview Visualize JSON

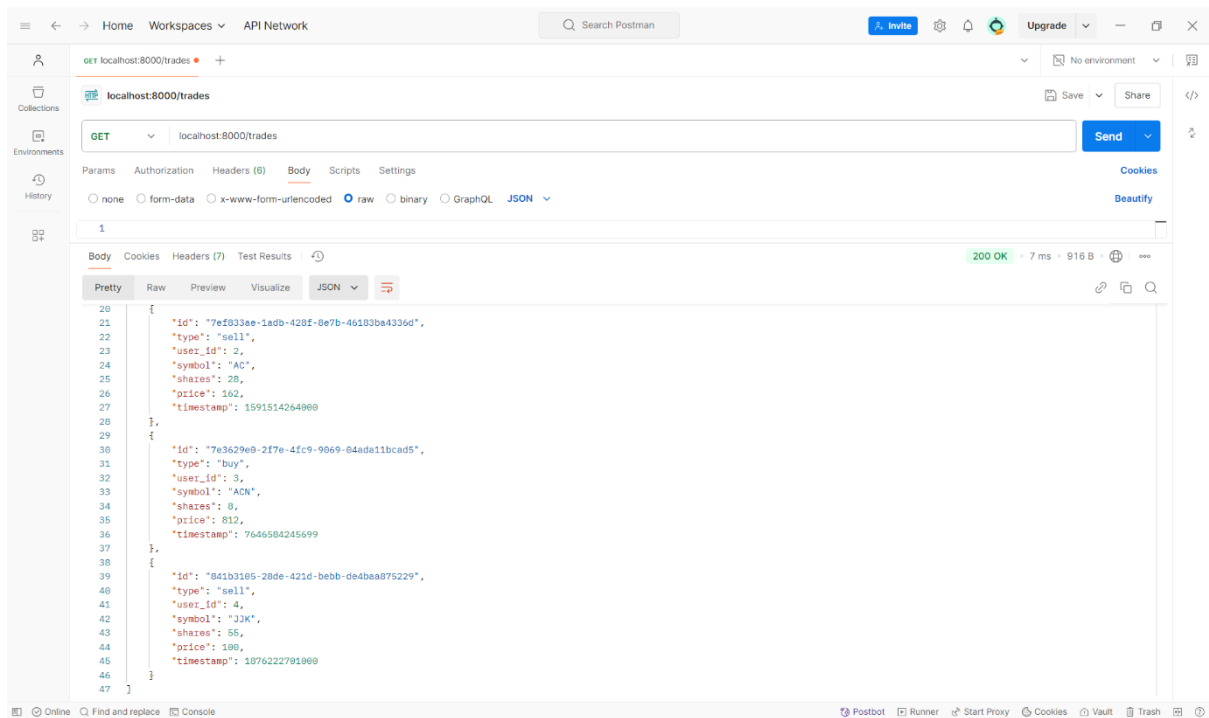
```
1 {
2   "id": "7e3629e9-2f7e-4fc9-9869-94ade11bcd5",
3   "type": "buy",
4   "user_id": 3,
5   "symbol": "ACN",
6   "shares": 8,
7   "price": 812,
8   "timestamp": 7646584245699
9 }
```

201 Created · 8 ms · 375 B

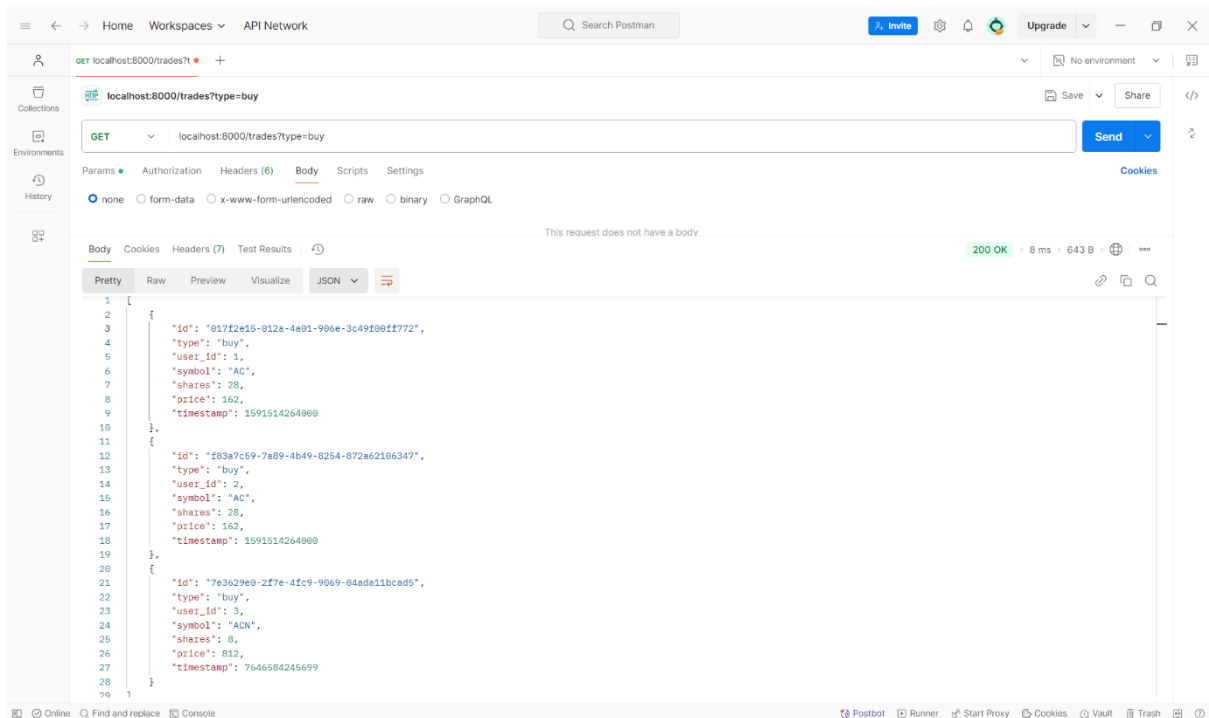


### 3. GET request to Trades





## 4. GET request to /trades?type=buy



## 5. GET request to /trades?user\_id=2

The screenshot shows the Postman interface with a GET request to `localhost:8000/trades?user_id=1`. The response is a 200 OK status with a JSON body containing a single trade object.

```
1 {
2   {
3     "id": "017f2e15-012a-4a01-906e-3c49f0eff772",
4     "type": "buy",
5     "user_id": 1,
6     "symbol": "AC",
7     "shares": 20,
8     "price": 162,
9     "timestamp": 1591514264000
10  }
11 }
```

Body: 200 OK · 7 ms · 372 B

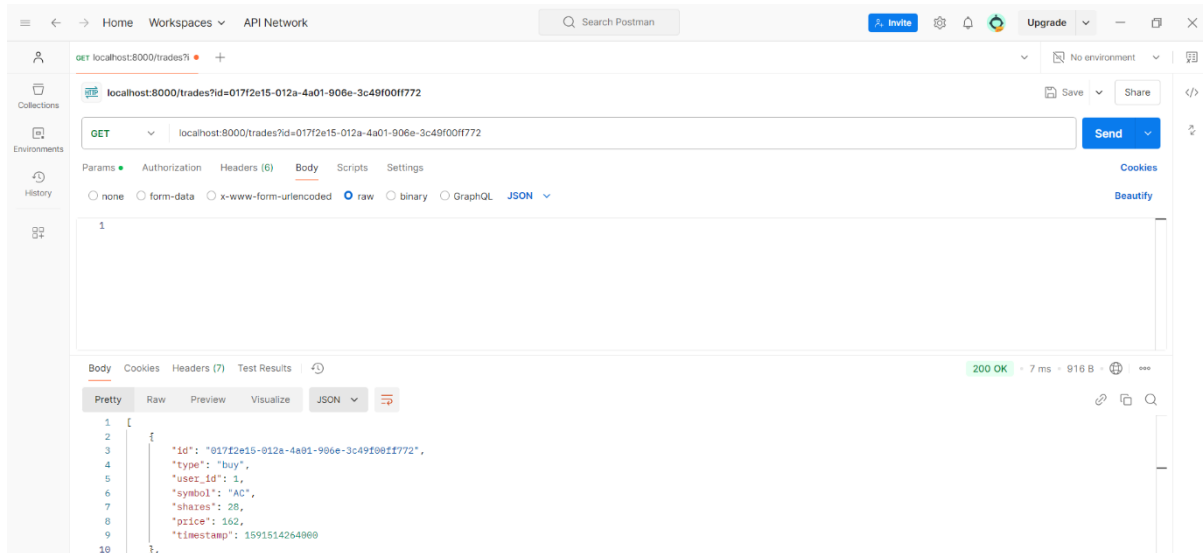
The screenshot shows the Postman interface with a GET request to `localhost:8000/trades?user_id=2`. The response is a 200 OK status with a JSON body containing two trade objects.

```
1 [
2   {
3     "id": "f03a7c59-7a09-4b49-8254-872a62106347",
4     "type": "buy",
5     "user_id": 2,
6     "symbol": "AC",
7     "shares": 20,
8     "price": 162,
9     "timestamp": 1591514264000
10  },
11   {
12     "id": "7ef033ae-1adb-428f-8e7b-46183ba4336d",
13     "type": "sell",
14     "user_id": 2,
15     "symbol": "AC",
16     "shares": 20,
17     "price": 162,
18     "timestamp": 1591514264000
19   }
20 ]
```

Body: 200 OK · 7 ms · 509 B

Postbot  
Ctrl Alt P

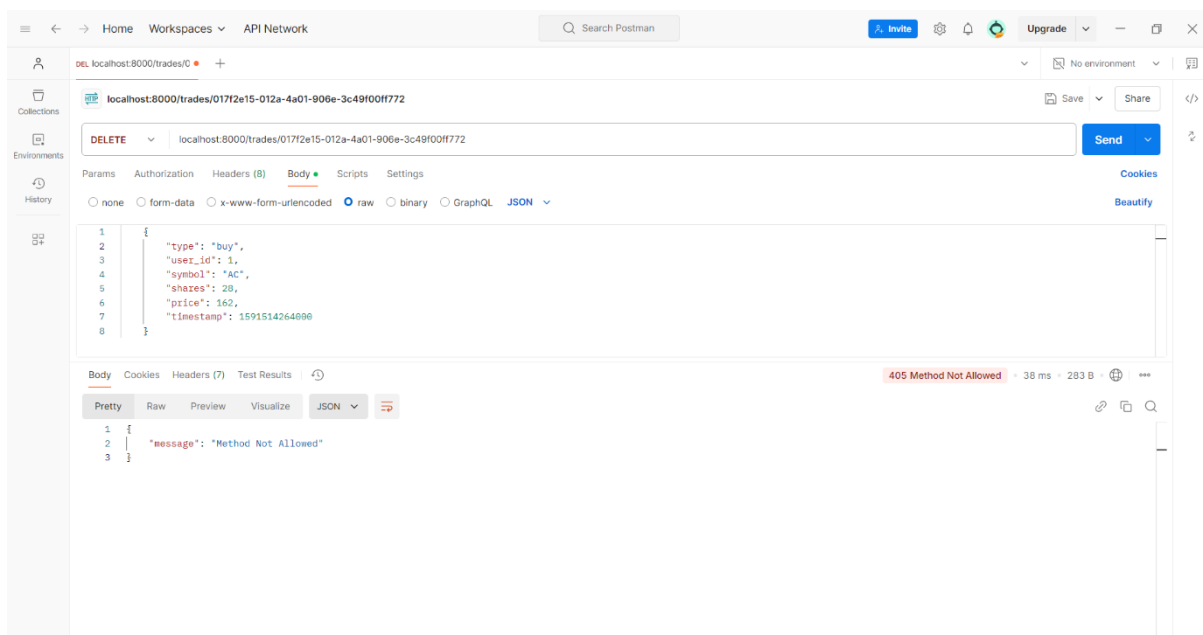
## 6. GET request to /trades/:id



Postman interface showing a GET request to `localhost:8000/trades?id=017f2e15-012a-4a01-906e-3c49f00ff772`. The response is a 200 OK status with a 7 ms response time and 916 B of data. The body is a JSON object representing a trade.

```
1 {
2   {
3     "id": "017f2e15-012a-4a01-906e-3c49f00ff772",
4     "type": "buy",
5     "user_id": 1,
6     "symbol": "AC",
7     "shares": 28,
8     "price": 162,
9     "timestamp": 1591514264800
10  },
11 }
```

## 7. DELETE, PATCH request to /trades/:id



Postman interface showing a DELETE request to `localhost:8000/trades/017f2e15-012a-4a01-906e-3c49f00ff772`. The response is a 405 Method Not Allowed status with a 38 ms response time and 283 B of data. The body is a JSON object with a message.

```
1 {
2   {
3     "type": "buy",
4     "user_id": 1,
5     "symbol": "AC",
6     "shares": 28,
7     "price": 162,
8     "timestamp": 1591514264800
9   }
10 }
```

```
1 {
2   "message": "Method Not Allowed"
3 }
```