Code:

1. Middleware: Authentication & Authorization (middleware/auth.js)

```
const jwt = require('jsonwebtoken');
const JWT_SECRET = "yourSecretKey";

function auth(requiredRole) {
  return (req, res, next) => {
    const authHeader = req.headers.authorization;
    if (!authHeader || !authHeader.startsWith('Bearer ')) {
      return res.status(401).json({ success: false, message: "No token provided"
});
    }

    const token = authHeader.split(' ')[1];
    try {
      const decoded = jwt.verify(token, JWT_SECRET);
      req.user = decoded;

      if (requiredRole && req.user.role !== requiredRole) {
        return res.status(403).json({ success: false, message: "Access denied"
});
      }

      next();
    } catch (err) {
      res.status(401).json({ success: false, message: "Invalid token" });
    }
  };
}

module.exports = auth;
```

2. User Schema (models/user.js)

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

const userSchema = new mongoose.Schema({
  name: { type: String, required: true, trim: true },
  email: {
    type: String,
    required: true,
    unique: true,
    lowercase: true,
    match: [/^\S+@\S+\.\S+$/, 'Please enter a valid email']
  },
  password: { type: String, required: true, minlength: 6 },
  role: { type: String, enum: ['user', 'admin'], default: 'user' }
}, { timestamps: true });

module.exports = mongoose.model('User', userSchema);
```

3. Authentication Routes (routes/auth.js)

```
const express = require('express');
const router = express.Router();
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');
const User = require('../models/user');
const JWT_SECRET = "yourSecretKey";

// REGISTER
router.post('/register', async (req, res) => {
  try {
    const { name, email, password, role } = req.body;
    const hashedPassword = await bcrypt.hash(password, 10);
    const newUser = new User({ name, email, password: hashedPassword, role
});
    await newUser.save();
    res.status(201).json({ success: true, message: 'User registered successfully'
});
  } catch (err) {
    res.status(400).json({ success: false, message: err.message });
  }
});

// LOGIN
router.post('/login', async (req, res) => {
  try {
    const { email, password } = req.body;
```

```
        const user = await User.findOne({ email });
        if (!user) return res.status(400).json({ success: false, message: "Invalid
credentials" });

        const isMatch = await bcrypt.compare(password, user.password);
        if (!isMatch) return res.status(400).json({ success: false, message: "Invalid
credentials" });

        const token = jwt.sign({ id: user._id, role: user.role }, JWT_SECRET, {
expiresIn: '1h' });
        res.json({ success: true, token });
    } catch (err) {
        res.status(500).json({ success: false, message: err.message });
    }
});

module.exports = router;
```

4. User Routes (routes/userRoutes.js)

```
const express = require('express');
const router = express.Router();
const User = require('../models/user');
const auth = require('../middleware/auth');

// GET all users (admin only)
router.get('/', auth('admin'), async (req, res) => {
    try {
        const users = await User.find();
        res.json({ success: true, data: users });
    } catch (err) {
        res.status(500).json({ success: false, message: err.message });
    }
});

// DELETE user (admin only)
router.delete('/:id', auth('admin'), async (req, res) => {
    try {
        await User.findByIdAndDelete(req.params.id);
        res.json({ success: true, message: 'User deleted' });
    } catch (err) {
        res.status(500).json({ success: false, message: err.message });
    }
});

// UPDATE user (admin can update anyone, users only their own)
router.put('/:id', auth(), async (req, res) => {
    try {
        if (req.user.role !== 'admin' && req.user.id !== req.params.id) {
            return res.status(403).json({ success: false, message: "You can only
update your own profile" });
        }
        const updatedUser = await User.findByIdAndUpdate(req.params.id, req.body,
{ new: true });
        res.json({ success: true, data: updatedUser });
    } catch (err) {
        res.status(500).json({ success: false, message: err.message });
    }
});

module.exports = router;
```

5. Server Setup (server.js)

```
const express = require('express');
const mongoose = require('mongoose');
const app = express();
app.use(express.json());

// Connect to MongoDB
mongoose.connect("mongodb+srv://<username>:<password>@cluster0.mongodb
.net/myDatabase", {
    useNewUrlParser: true,
    useUnifiedTopology: true
})
.then(() => console.log('✅ MongoDB Connected'))
.catch(err => console.error(err));

// Import Routes
const userRoutes = require('./routes/userRoutes');
const authRoutes = require('./routes/auth');
```

```
app.use('/api/users', userRoutes);
app.use('/api/auth', authRoutes);

// Start Server
app.listen(3000, () => {
  console.log('🚀 Server running on port 3000');
});
```

Output :

Validating RESTful APIs using Postman -

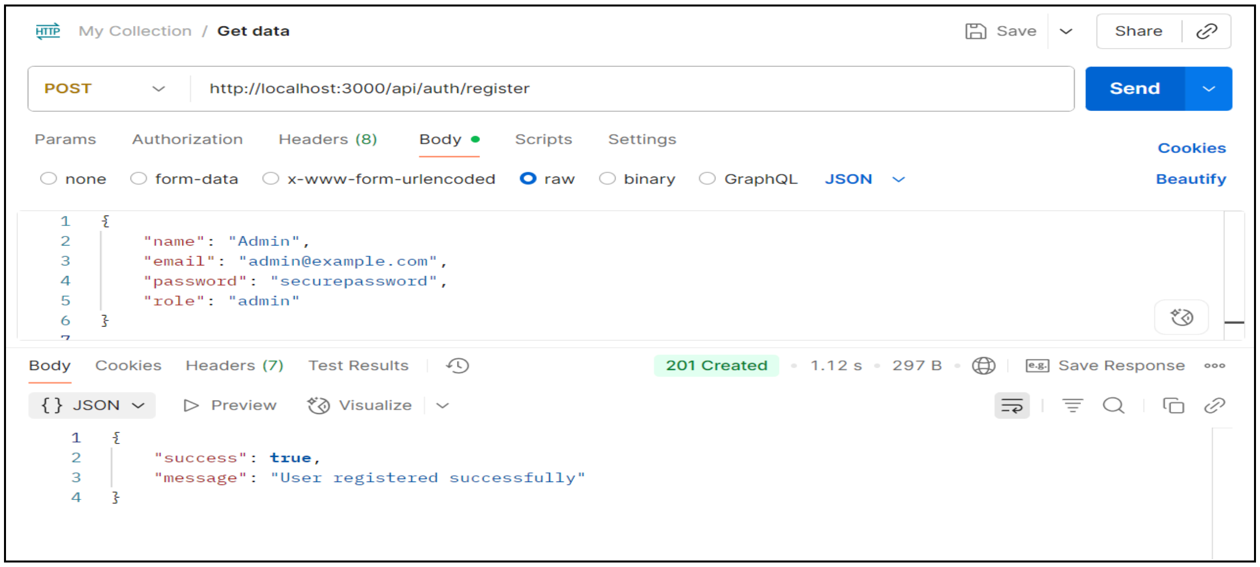
Create Operations: User Registration

The first part of the demonstration involves creating user accounts to populate the database. Two distinct user types will be created to illustrate a typical multi-role system.

1.1. Register an Admin User

The admin user is a privileged account with the authority to access protected routes and perform actions on other users.

- **Endpoint:** POST http://localhost:3000/api/auth/register
- **Body:** Set the request body to raw with type JSON and provide the following payload:
  ```
  {

    "name": "Admin",

    "email": "admin@example.com",

    "password": "securepassword",

    "role": "admin"

  }
  ```
- **Action:** Send the request.
- **Expected Result:** The server should respond with a **201 Created** status, indicating successful user creation.

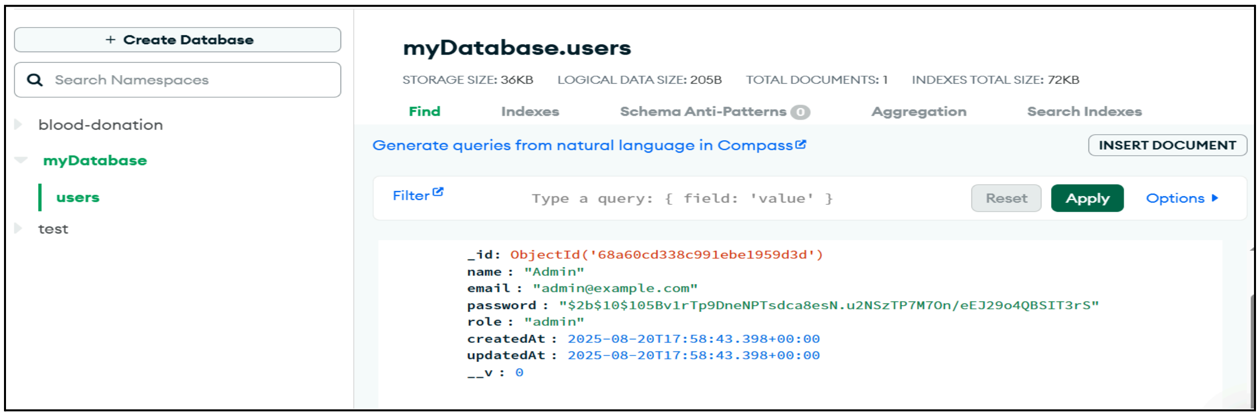


```
POST http://localhost:3000/api/auth/register
```

```
{
    "name": "Admin",
    "email": "admin@example.com",
    "password": "securepassword",
    "role": "admin"
}
```

201 Created • 1.12 s • 297 B

```
{
    "success": true,
    "message": "User registered successfully"
}
```



myDatabase.users

STORAGE SIZE: 36KB　LOGICAL DATA SIZE: 205B　TOTAL DOCUMENTS: 1　INDEXES TOTAL SIZE: 72KB

```
_id: ObjectId('68a60cd338c991ebe1959d3d')
name : "Admin"
email : "admin@example.com"
password : "$2b$10$105Bv1rTp9DneNPTsdca8esN.u2NSzTP7M7On/eEJ29o4QBSIT3rS"
role : "admin"
createdAt : 2025-08-20T17:58:43.398+00:00
updatedAt : 2025-08-20T17:58:43.398+00:00
__v : 0
```

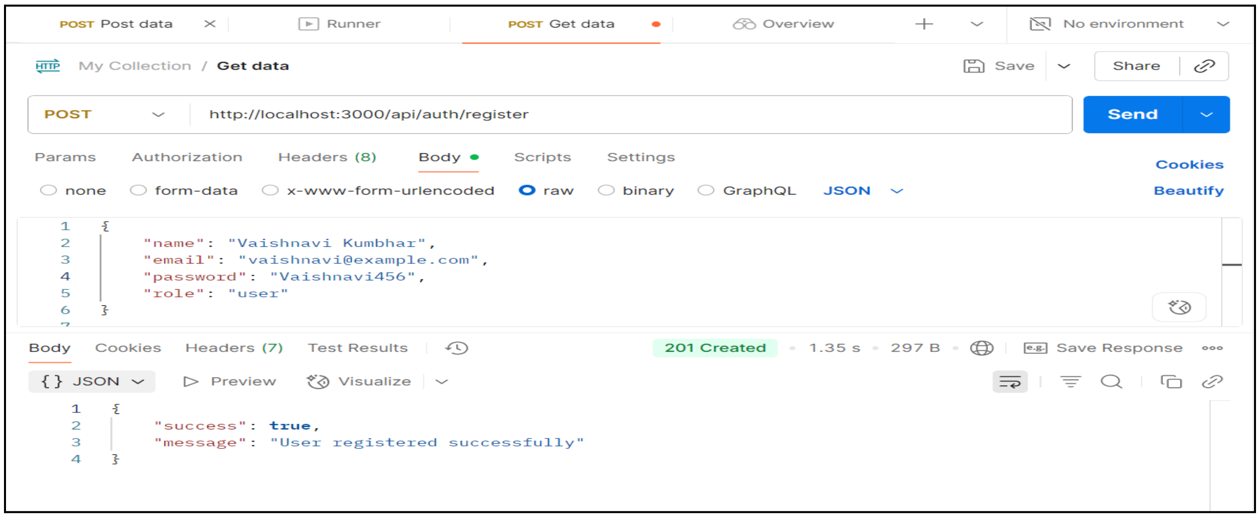
1.2. Register a Regular User

A second, non-admin user is created to serve as a target for the subsequent update and delete operations.

- **Endpoint:** POST http://localhost:3000/api/auth/register
- **Body:** Use the same endpoint but provide a new JSON payload for the regular user:

```
{

"name": "Disha Kulkarni",

  "email": "Disha@example.com",

  "password": "disha645",

   "role": "user"

}
```

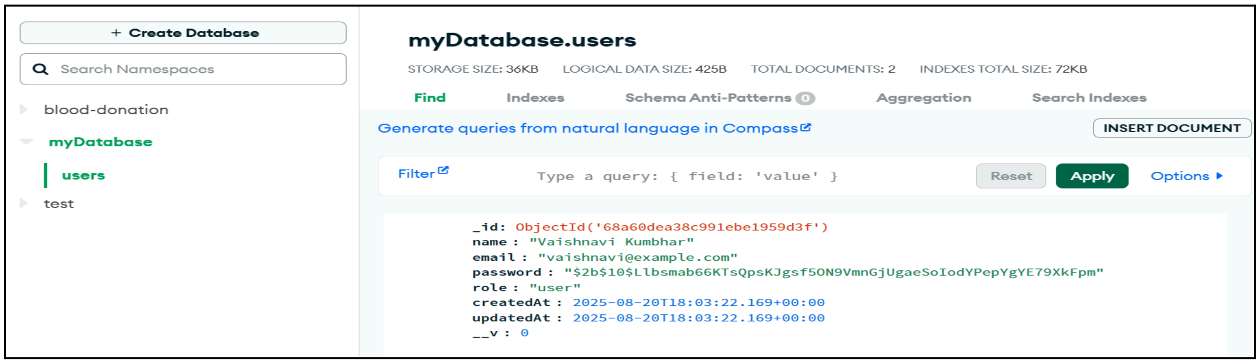
- **Action:** Send the request.
- **Expected Result:** The server should again respond with a **201 Created** status.



```
POST http://localhost:3000/api/auth/register
```

```
{
    "name": "Vaishnavi Kumbhar",
    "email": "vaishnavi@example.com",
    "password": "Vaishnavi456",
    "role": "user"
}
```

```
{
    "success": true,
    "message": "User registered successfully"
}
```



myDatabase.users

STORAGE SIZE: 36KB LOGICAL DATA SIZE: 425B TOTAL DOCUMENTS: 2 INDEXES TOTAL SIZE: 72KB

```
_id: ObjectId('68a60dea38c991ebe1959d3f')
name : "Vaishnavi Kumbhar"
email : "vaishnavi@example.com"
password : "$2b$10$Llbsmab66KTsQpsKJgsf5ON9VmnGjUgaeSoIodYPepYgYE79XkFpm"
role : "user"
createdAt : 2025-08-20T18:03:22.169+00:00
updatedAt : 2025-08-20T18:03:22.169+00:00
__v : 0
```

2. Authenticate and Read Operations

After creating the user accounts, the next step is to authenticate to gain access to protected resources and retrieve the full list of users.

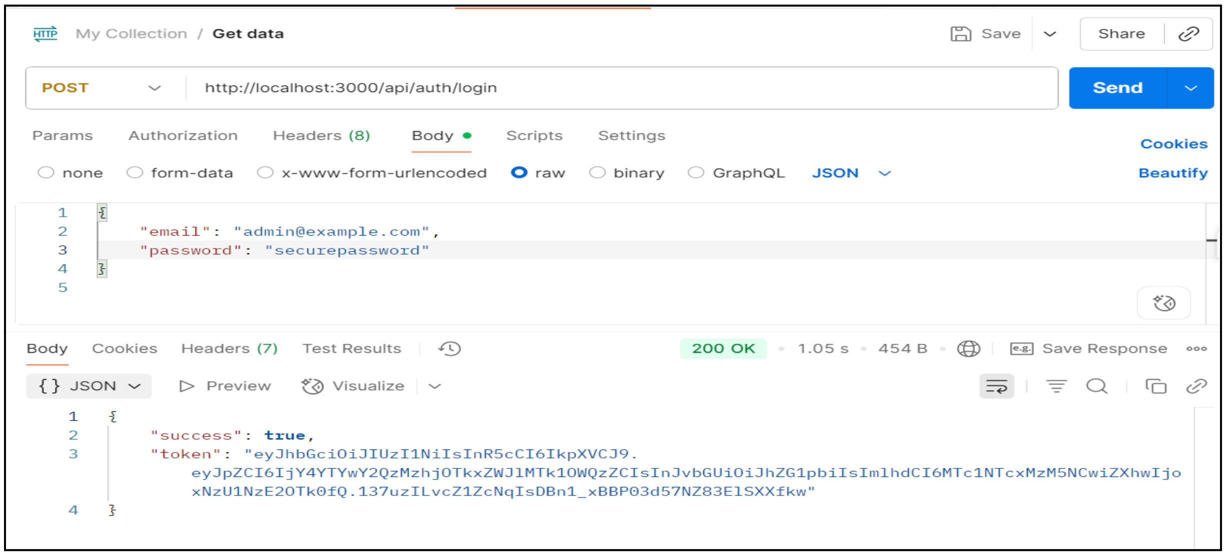
2.1. Log In to Obtain an Admin Token

You must log in as the admin to get a token with the necessary permissions for accessing admin-protected routes.

- **Endpoint:** POST http://localhost:3000/api/auth/login
- **Body:** Provide the email and password for the admin account:

```
{"email": "admin@example.com",
 "password": "securepassword"}
```

- **Action:** Send the request.
- **Expected Result:** The server will return a **200 OK** response with a JWT token. **Copy this token** and keep it ready for the next steps.



POST http://localhost:3000/api/auth/login

```
{
    "email": "admin@example.com",
    "password": "securepassword"
}
```

200 OK • 1.05 s • 454 B

```
{
    "success": true,
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
        eyJpZCI6IjY4YTYwY2QzMzhjOTkxZWJlMTk1OWQzZCIsInJvbGUiOiJhZG1pbiIsImlhdCI6MTc1NTcxMzM5NCwiZXhwIjo
        xNzU1NzE2OTk0fQ.137uzILvcZ1ZcNqIsDBn1_xBBP03d57NZ83E1SXXfkw"
}
```

2.2. Log In as a Regular User

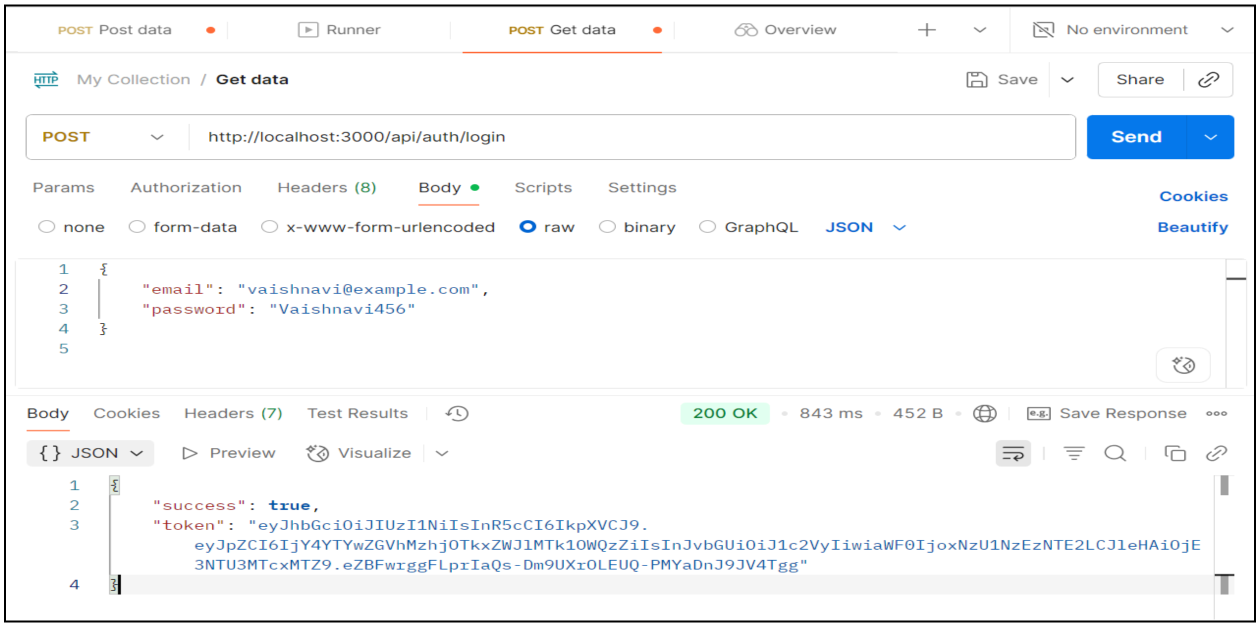
This step demonstrates that a regular user login is identical in process to an admin login, but the token's payload will contain the 'user' role.

- **Endpoint:** POST http://localhost:3000/api/auth/login
- **Body:** Provide the email and password for the regular user account:

{"email": "jane@example.com",

"password": "anotherpassword"}

- **Action:** Send the request.
- **Expected Result:** A **200 OK** response with a JWT token. This token's internal data specifies the user's role as 'user'.



POST http://localhost:3000/api/auth/login

```
{
    "email": "vaishnavi@example.com",
    "password": "Vaishnavi456"
}
```

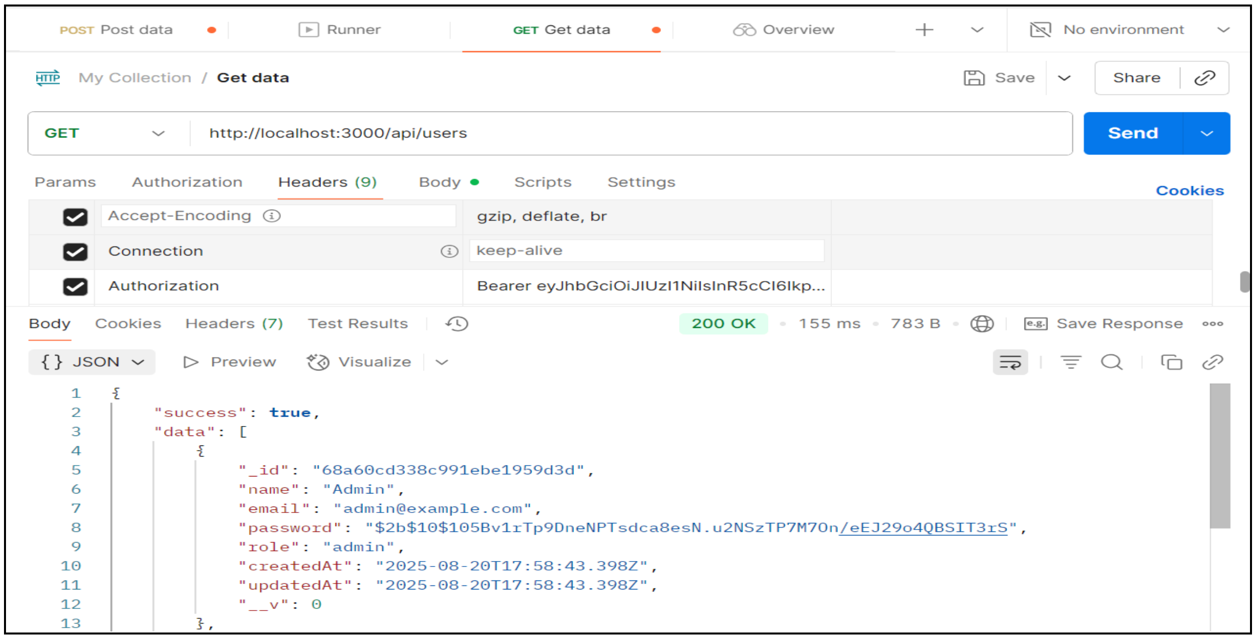
200 OK • 843 ms • 452 B

```
{
    "success": true,
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
        eyJpZCI6IjY4YTYwZGVhMzhjOTkxZWJlMTk1OWQzZiIsInJvbGUiOiJ1c2VyIiwiaWF0IjoxNzU1NzEzNTE2LCJleHAiOjE
        3NTU3MTcxMTZ9.eZBFwrggFLprIaQs-Dm9UXrOLEUQ-PMYaDnJ9JV4Tgg"
}
```

2.3. Get All Users (Read)

This demonstrates that the admin token can successfully access the protected route to retrieve a list of all users.

- **Endpoint:** GET http://localhost:3000/api/users
- **Headers:** Add an Authorization header with the value Bearer <paste_your_admin_token_here>.
- **Action:** Send the request.
- **Expected Result:** A **200 OK** response containing a JSON array with both the **Admin** and **Jane Doe** user objects. **Important:** From this response, copy the **_id** value of the "Jane Doe" user.

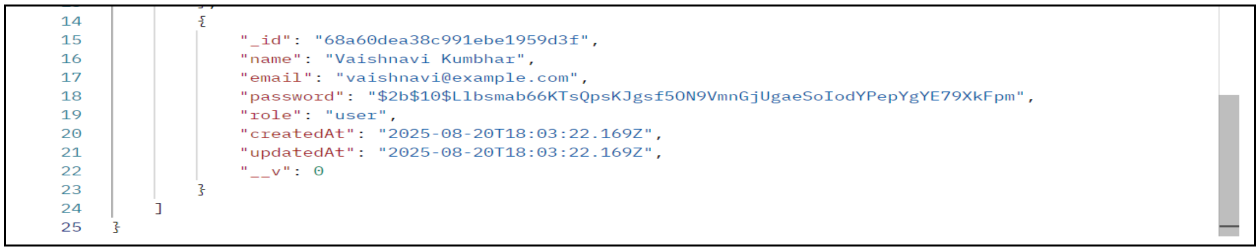


GET http://localhost:3000/api/users

Accept-Encoding	gzip, deflate, br
Connection	keep-alive
Authorization	Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6Ikp...

200 OK • 155 ms • 783 B

```
{
    "success": true,
    "data": [
        {
            "_id": "68a60cd338c991ebe1959d3d",
            "name": "Admin",
            "email": "admin@example.com",
            "password": "$2b$10$105Bv1rTp9DneNPTsdca8esN.u2NSzTP7M70n/eEJ29o4QBSIT3rS",
            "role": "admin",
            "createdAt": "2025-08-20T17:58:43.398Z",
            "updatedAt": "2025-08-20T17:58:43.398Z",
            "__v": 0
        },
```



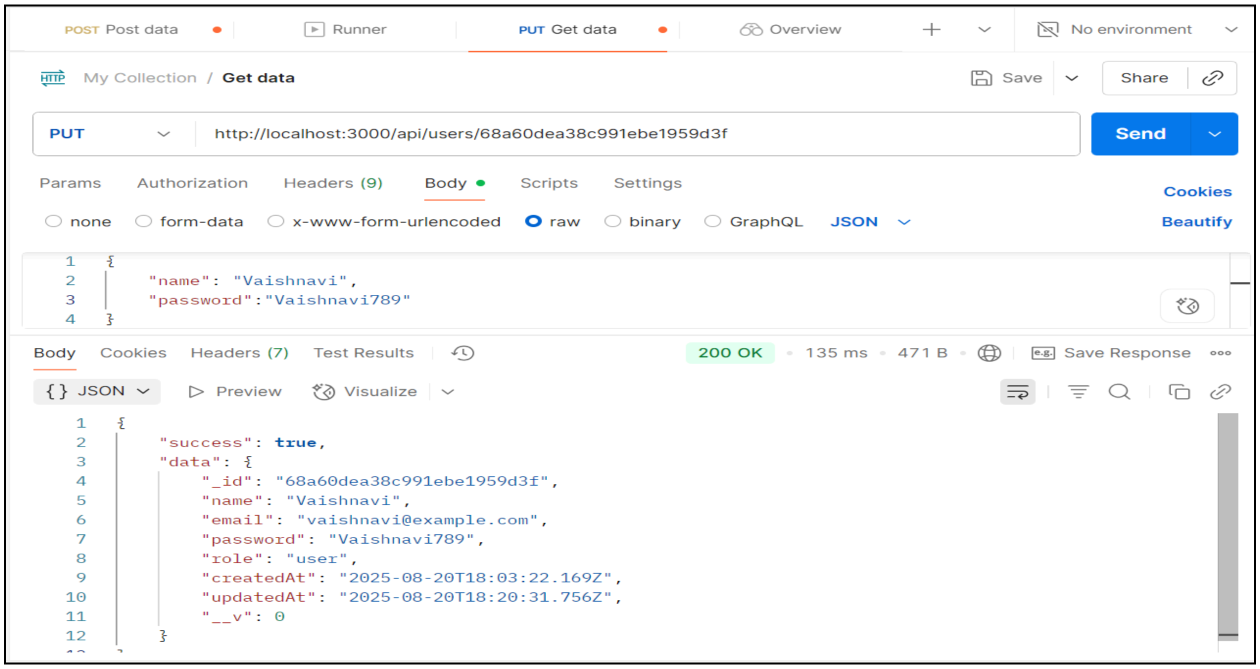
```
        {
            "_id": "68a60dea38c991ebe1959d3f",
            "name": "Vaishnavi Kumbhar",
            "email": "vaishnavi@example.com",
            "password": "$2b$10$Llbsmab66KTsQpsKJgsf5ON9VmnGjUgaeSoIodYPepYgYE79XkFpm",
            "role": "user",
            "createdAt": "2025-08-20T18:03:22.169Z",
            "updatedAt": "2025-08-20T18:03:22.169Z",
            "__v": 0
        }
    ]
}
```

3. Update Operation

This part of the demonstration shows the ability to modify an existing user's data using the PUT method.

3.1. Update the Regular User's Name

- **Endpoint:** PUT http://localhost:3000/api/users/<paste_jane_doe_id_here>
- **Headers:** Use the same Authorization header with the admin token.
- **Body:** Provide the updated data in a JSON payload. In this case, only the name is changed:
  { "name": "Jane Smith" }
- **Action:** Send the request.
- **Expected Result:** The server will respond with a **200 OK** status and the updated user object, showing the name Jane Smith.



PUT http://localhost:3000/api/users/68a60dea38c991ebe1959d3f

```
{
    "name": "Vaishnavi",
    "password":"Vaishnavi789"
}
```

200 OK • 135 ms • 471 B

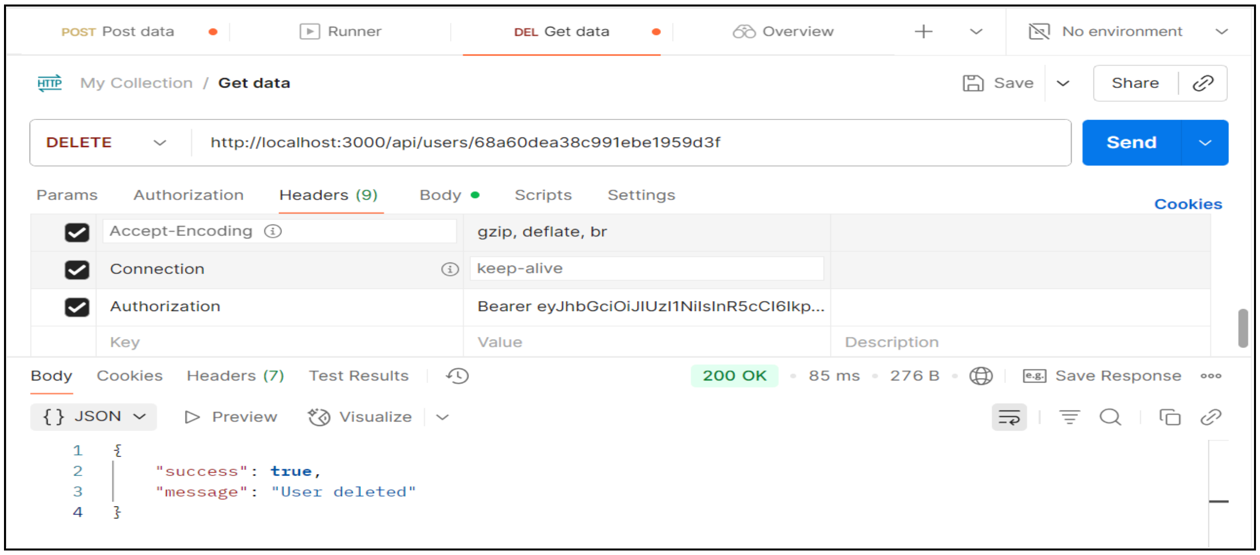
```
{
    "success": true,
    "data": {
        "_id": "68a60dea38c991ebe1959d3f",
        "name": "Vaishnavi",
        "email": "vaishnavi@example.com",
        "password": "Vaishnavi789",
        "role": "user",
        "createdAt": "2025-08-20T18:03:22.169Z",
        "updatedAt": "2025-08-20T18:20:31.756Z",
        "__v": 0
    }
```

4. Delete Operation

This final part demonstrates the deletion of a user from the database and verifies that the operation was successful.

4.1. Delete the Regular User

- **Endpoint:** DELETE http://localhost:3000/api/users/<paste_jane_doe_id_here>
- **Headers:** Use the Authorization header with the admin token.
- **Action:** Send the request.
- **Expected Result:** A **200 OK** response with a User deleted success message.



DELETE http://localhost:3000/api/users/68a60dea38c991ebe1959d3f

✓	Accept-Encoding	gzip, deflate, br
✓	Connection	keep-alive
✓	Authorization	Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6Ikp...

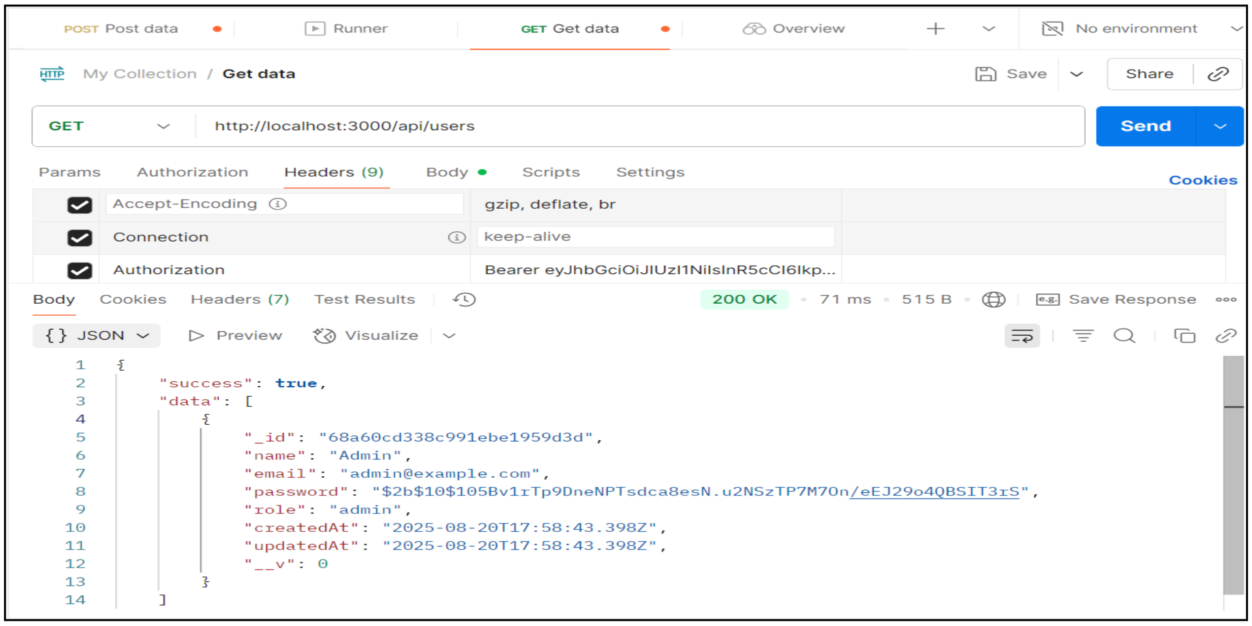
200 OK • 85 ms • 276 B

```
{
    "success": true,
    "message": "User deleted"
}
```

4.2. Verify Deletion

A final GET request confirms that the user has been permanently removed from the database.

- **Endpoint:** GET http://localhost:3000/api/users
- **Headers:** Use the Authorization header with the admin token.
- **Action:** Send the request.
- **Expected Result:** A **200 OK** response with a JSON array containing only the **Admin** user object.



GET http://localhost:3000/api/users

✓	Accept-Encoding	gzip, deflate, br
✓	Connection	keep-alive
✓	Authorization	Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6Ikp...

200 OK • 71 ms • 515 B

```
{
    "success": true,
    "data": [
        {
            "_id": "68a60cd338c991ebe1959d3d",
            "name": "Admin",
            "email": "admin@example.com",
            "password": "$2b$10$105Bv1rTp9DneNPTsdca8esN.u2NSzTP7M70n/eEJ29o4QBSIT3rS",
            "role": "admin",
            "createdAt": "2025-08-20T17:58:43.398Z",
            "updatedAt": "2025-08-20T17:58:43.398Z",
            "__v": 0
        }
    ]
```