RESTful API

server.js

```
const express = require('express')
const mongoose = require('mongoose')
const url = 'mongodb://localhost/usersDB'

const app = express()

mongoose.connect(url, {useNewUrlParser:true})
const con = mongoose.connection

con.on('open', () => {
    console.log('connected...')
})

app.use(express.json())

const usersRouter = require('./routes/users')
app.use('/users', usersRouter)

app.listen(9989, () =>{
    console.log('Server started')
})
```

.routes/user.js

```
const express = require('express')
const router = express.Router()
const User = require('./models/users')


        /*  Get */

router.get('/', async(req, res) => {
    try{
        const users = await User.find()
        res.json(users)
    }catch(e){
    res.send('Error'+e)}
})

router.get('/:id', async(req, res) => {
    try{
        const user = await User.findById(req.params.id)
        res.json(user)
```

RESTful API

```
    }catch(e){
    res.send('Error'+e)}
})


        /* post */

router.post('/', async (req, res) => {
    try {
        const users = [];
        for (const userData of req.body) {
            const user = new User({
                name: userData.name,
                tech: userData.tech,
                subscription: userData.subscription
            });
            const savedUser = await user.save();
            users.push(savedUser);
        }
        res.json(users);
    } catch (e) {
        res.status(500).send('Error: ' + e);
    }
});



        /*  Patch  */


router.patch('/:id', async(req, res)=>{
    try{
        const user = await User.findById(req.params.id)
       // user.name = req.body.name
      // user.tech = req.body.tech
        user.subscription = req.body.subscription
        const u1 =  await user.save()
        res.json(u1)
    }catch(e){
        res.send('Error'+e)
    }
})



/* DELETE */
```

RESTful API

```
router.delete('/:id', async (req, res) => {
    try {
        const deletedUser = await User.findByIdAndDelete(req.params.id);



       if (!deletedUser) {
            return res.status(404).send('User not found');
        }
        res.json({ message: 'User deleted successfully', deletedUser });
    } catch (e) {
        res.status(500).send('Error: ' + e);
    }
});



module.exports = router
```

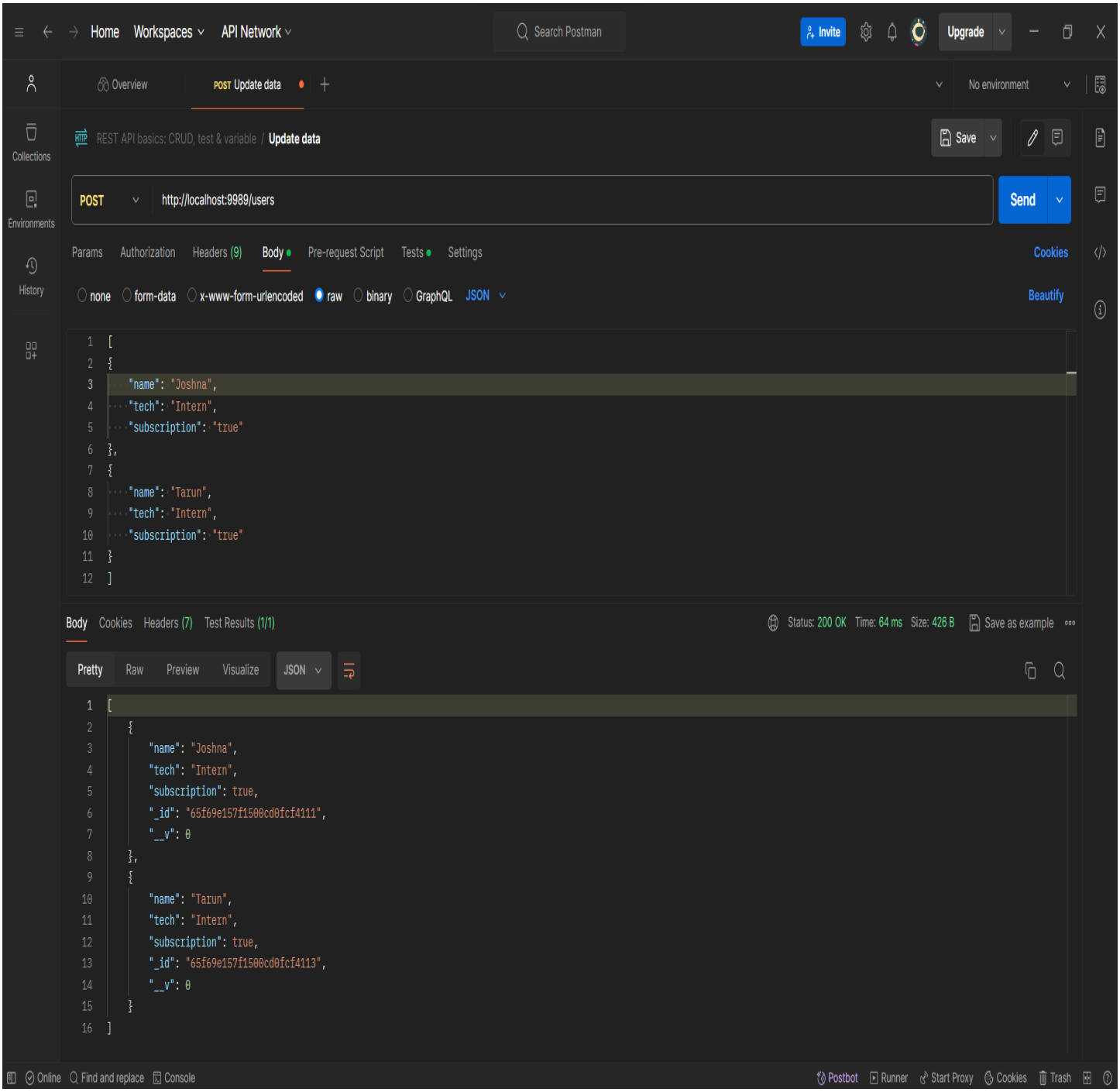
./models/users.js

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

const usersSchema = new mongoose.Schema({
    name: {
        type: String,
        required: true
    },
    tech:{
        type:String,
        required: true
    },
    subscription:{
        type: Boolean,
        required:true,
        default: false
    }
})

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

RESTful API

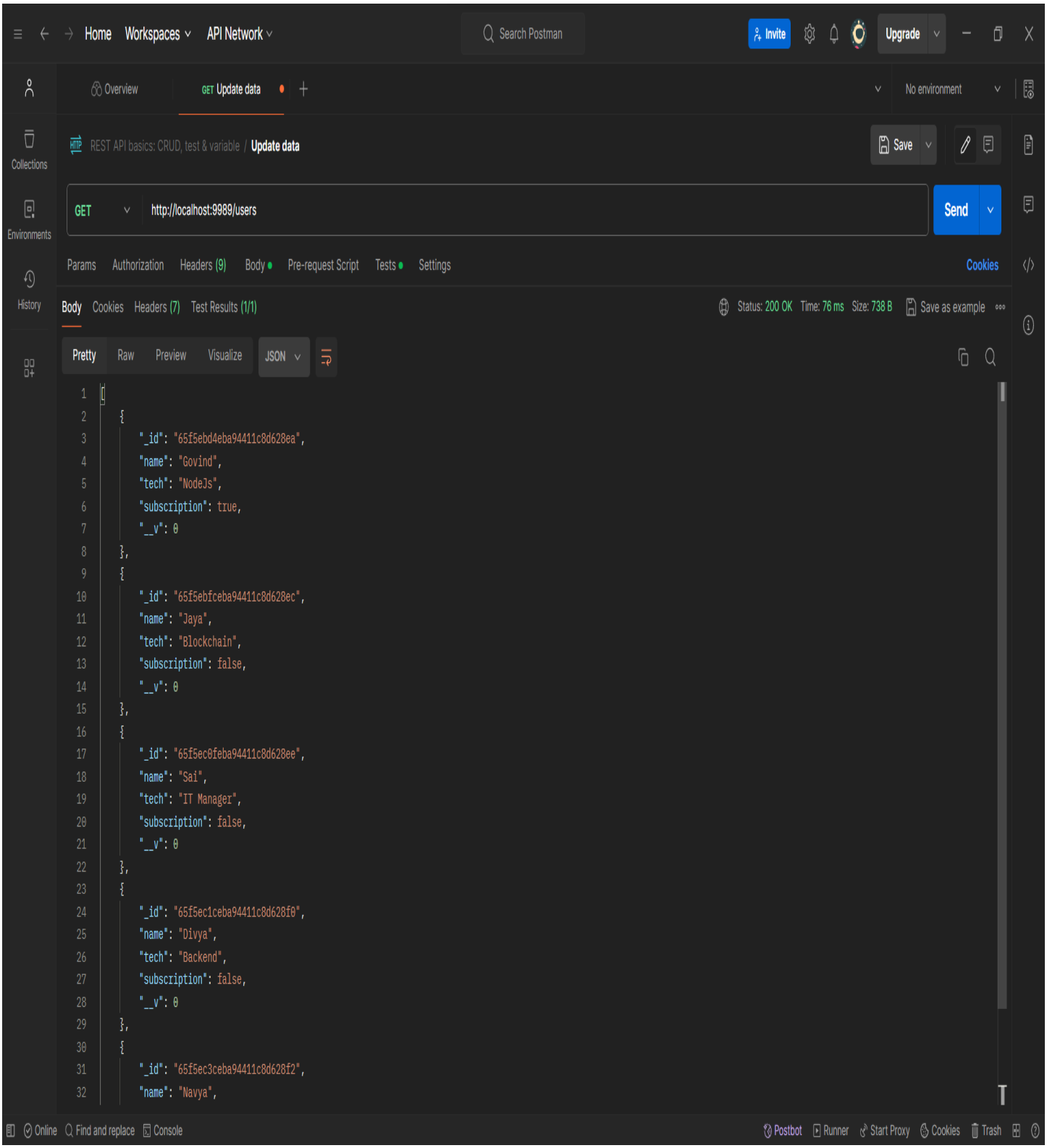
POSTMAN REPORT-

1. Post Request – CREATE



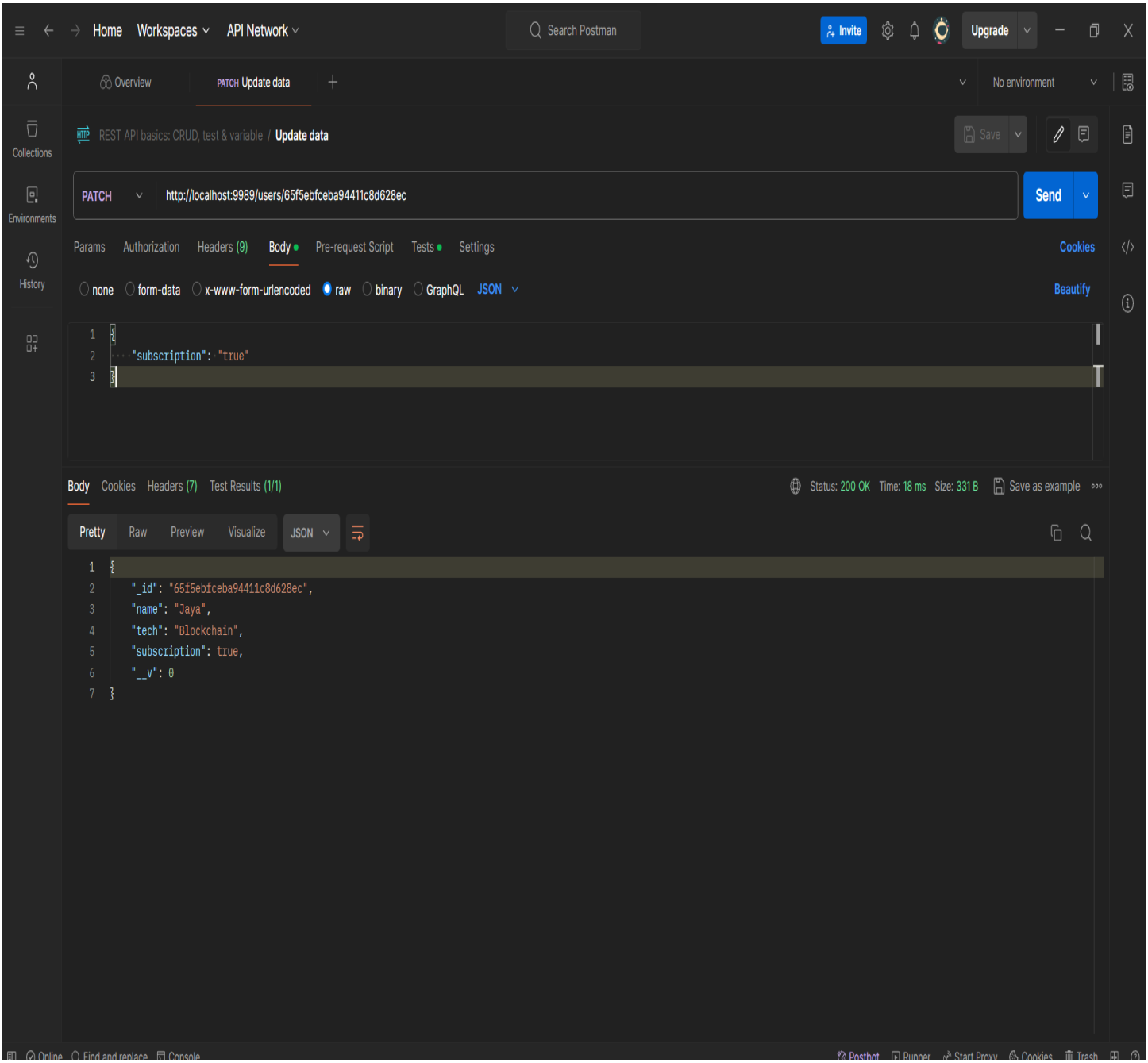
RESTful API

2. Get Request – READ



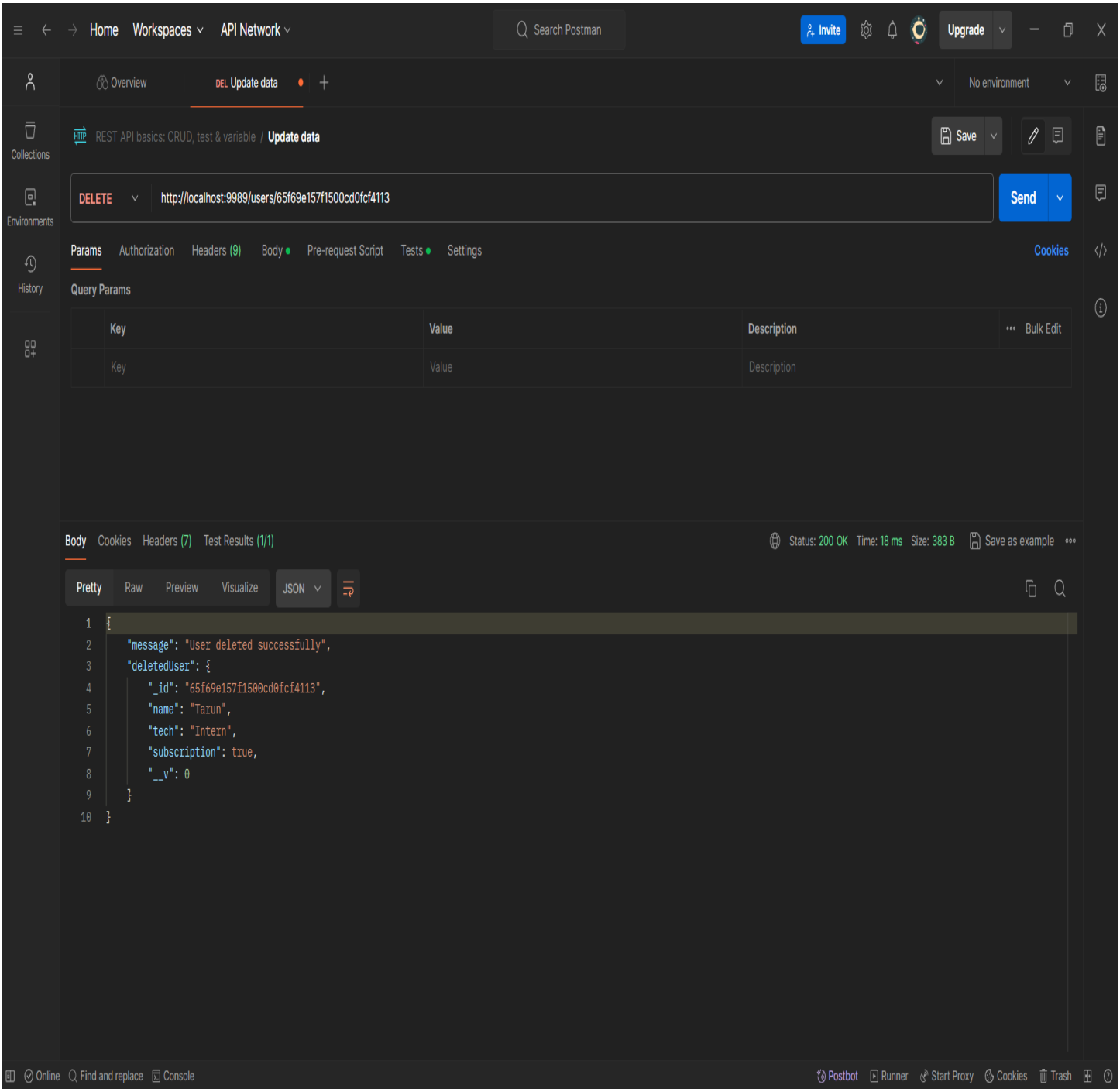
RESTful API

3. Patch Request – UPDATE



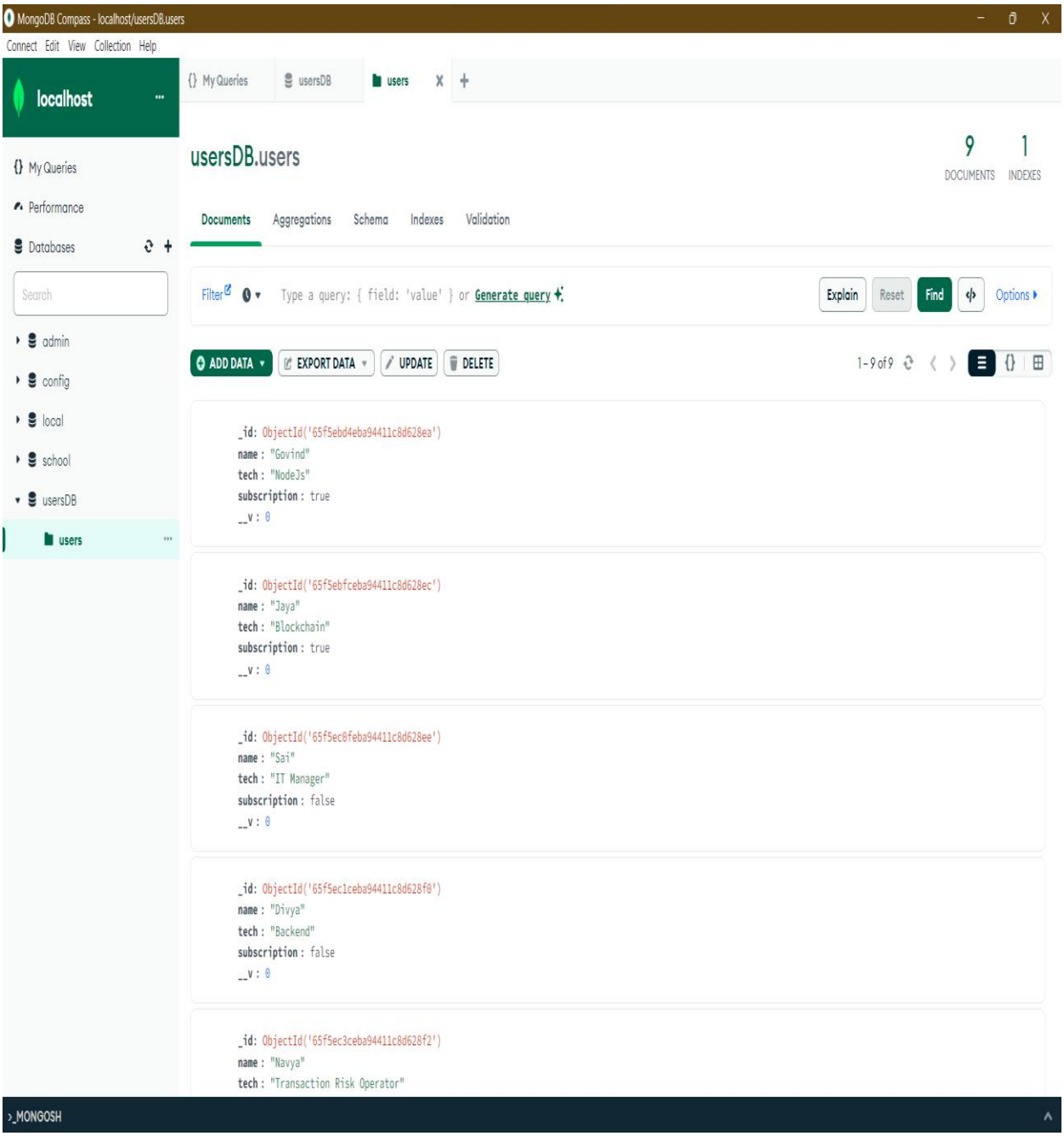
RESTful API

4. Delete Request – Destroy



RESTful API

MongoDB Compass



RESTful API

Readme File

Users API Documentation

Overview: This API provides CRUD (Create, Read, Update, Delete) operations for managing user data.

Setting Up Express with MongoDB

1. **Install Node.js and npm**:

Ensure that Node.js and npm are installed on your system. You can download and install them from the official Node.js website.

2. **Create a New Project Directory**:

Use your terminal or command prompt to create a new directory for your project. Navigate into this directory.

3. **Initialize a Node.js Project**:

Run `npm init -y` in the terminal to initialize a new Node.js project with default settings. This will create a `package.json` file in your project directory.

4. **Install Express**:

Install Express as a dependency for your project by running `npm install express` in the terminal.

5. **Install Mongoose**:

Mongoose is a MongoDB object modeling tool that provides a straight-forward, schema-based solution for modeling application data. Install Mongoose by running `npm install mongoose` in the terminal.

6. **Create Your Express Application**:

RESTful API

Write your Express application code in a JavaScript file (e.g., `app.js` or `server.js`). Require Express and Mongoose, and set up your Express application, including defining routes, middleware, and connecting to MongoDB.

7. **Define MongoDB Schemas and Models**:

Use Mongoose to define schemas and models for your MongoDB collections. This allows you to define the structure of your data and interact with MongoDB using JavaScript objects.

8. **Implement CRUD Operations**:

With Express and Mongoose set up, implement CRUD (Create, Read, Update, Delete) operations in your Express application to interact with MongoDB. Use Mongoose methods such as `save()`, `find()`, `findOneAndUpdate()`, and `deleteOne()` to perform database operations.

Endpoints:

1. *GET /users*

- Retrieves a list of all users.
- Request Type: GET
- URL: http://localhost:9989/users
- Response Format: JSON array of user objects

2. *GET /users/:id*

- Retrieves a user by their ID.
- Request Type: GET
- URL: http://localhost:9989/users/:id
- Replace ":id" with the user's ID.
- Response Format: JSON object representing the user

3. *POST /users*

- Adds a new user to the database.
- Request Type: POST
- URL: http://localhost:9989/users
- Request Body: JSON object with the following fields:
    - "name": User's name (required)

- "tech": User's technology (required)
- "subscription": User's subscription status (true/false, default: false)

- Response Format: JSON object representing the newly created user

4. PATCH /users/:id

- Updates the subscription status of a user.
- Request Type: PATCH
- URL: http://localhost:9989/users/:id
- Replace ":id" with the user's ID.
- Request Body: JSON object with the following field:
  - "subscription": Updated subscription status (true/false)
- Response Format: JSON object representing the updated user

5. DELETE /users/:id

- Deletes a user from the database.
- Request Type: DELETE
- URL: http://localhost:9989/users/:id
- Replace ":id" with the user's ID.
- Response Format: JSON object representing the deleted user

Usage:

1. Use any HTTP client (e.g., Postman) to send requests to the API endpoints.
2. Set the request type (GET, POST, PATCH, DELETE) and provide necessary request data.
3. Replace placeholders (e.g., ":id") with actual values (e.g., user's ID).
4. Send the request and observe the response.

Using Postman for Testing

1. **Download and Install Postman**:
If you haven't already, download and install Postman on your system.

2. **Open Postman**:
Launch the Postman application.

3. **Create Requests**:

RESTful API

In Postman, create requests to test your Express API endpoints. Set the request type (GET, POST, PUT, DELETE), specify the request URL (e.g., `http://localhost:9989/users`), and provide any necessary request body or parameters.

4. **Send Requests**:
Send the requests and observe the responses from your Express API. Verify that the responses match your expectations and that your API endpoints are functioning correctly.

5. **Test CRUD Operations**:
Test each CRUD operation (Create, Read, Update, Delete) using Postman. For example, create a new user, retrieve user data, update user information, and delete users.

6. **Verify Error Handling**:
Test error handling by sending requests with invalid data or to non-existent endpoints. Ensure that your Express application returns appropriate error responses.

URL ： http://localhost:9989/users