

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

The screenshot displays the Postman application interface. At the top, the navigation bar includes 'Home', 'Workspaces', and 'API Network'. A search bar labeled 'Search Postman' is present. On the right, there are buttons for 'Invite', 'Upgrade', and a minus sign. The left sidebar shows 'Collections' and 'Environments'. The main area is titled 'REST API basics: CRUD, test & variable / Update data'. Below this, the request method is 'POST' and the URL is 'http://localhost:9989/users'. The 'Body' tab is selected, showing a JSON array of two objects. The 'Send' button is visible. Below the request, the 'Body' tab of the response is selected, showing a JSON array of two objects with additional fields like '_id' and '__v'. The status bar at the bottom indicates 'Status: 200 OK', 'Time: 64 ms', and 'Size: 426 B'.

```
1 [
2 {
3   "name": "Joshna",
4   "tech": "Intern",
5   "subscription": "true"
6 },
7 {
8   "name": "Tarun",
9   "tech": "Intern",
10  "subscription": "true"
11 }
12 ]
```

Body Cookies Headers (7) Test Results (1/1) Status: 200 OK Time: 64 ms Size: 426 B Save as example

```
1 [
2 {
3   "name": "Joshna",
4   "tech": "Intern",
5   "subscription": true,
6   "_id": "65f69e157f1508cd0fcf4111",
7   "__v": 0
8 },
9 {
10  "name": "Tarun",
11  "tech": "Intern",
12  "subscription": true,
13  "_id": "65f69e157f1508cd0fcf4113",
14  "__v": 0
15 }
16 ]
```

RESTful API

2. Get Request – READ

The screenshot shows the Postman interface with a GET request to `http://localhost:9089/users` successfully executed. The response is a JSON array of 5 user objects. The status is 200 OK, with a response time of 76 ms and a size of 738 B.

Request:

- Method: GET
- URL: `http://localhost:9089/users`

Response:

```
1 {
2   {
3     "_id": "65f5ebd4eba94411c8d628ea",
4     "name": "Govind",
5     "tech": "NodeJS",
6     "subscription": true,
7     "__v": 0
8   },
9   {
10    "_id": "65f5ebfceb94411c8d628ec",
11    "name": "Jaya",
12    "tech": "Blockchain",
13    "subscription": false,
14    "__v": 0
15  },
16  {
17    "_id": "65f5ec0feba94411c8d628ee",
18    "name": "Sai",
19    "tech": "IT Manager",
20    "subscription": false,
21    "__v": 0
22  },
23  {
24    "_id": "65f5ec1ceba94411c8d628f0",
25    "name": "Divya",
26    "tech": "Backend",
27    "subscription": false,
28    "__v": 0
29  },
30  {
31    "_id": "65f5ec3ceba94411c8d628f2",
32    "name": "Navya",
```

Postman Interface Details:

- Top Bar: Home, Workspaces, API Network, Search Postman, Invite, Upgrade.
- Left Sidebar: Overview, REST API basics: CRUD, test & variable / Update data, Collections, Environments, History.
- Request Tab: Params, Authorization, Headers (9), Body (selected), Pre-request Script, Tests, Settings.
- Response Tab: Body (selected), Cookies, Headers (7), Test Results (1/1).
- Response Format: Pretty (selected), Raw, Preview, Visualize, JSON.
- Status Bar: Online, Find and replace, Console, Postbot, Runner, Start Proxy, Cookies, Trash.

RESTful API

3. Patch Request – UPDATE

The screenshot displays the Postman interface for a PATCH request. The request is directed to the endpoint `http://localhost:9989/users/65f5ebfceb94411c8d628ec`. The request body is a JSON object that updates the `subscription` field to `true` for the user with ID `65f5ebfceb94411c8d628ec`.

Request Details:

- Method:** PATCH
- URL:** `http://localhost:9989/users/65f5ebfceb94411c8d628ec`
- Body Type:** raw (JSON)
- Body Content:**

```
1 {  
2   "subscription": "true"  
3 }
```

Response Details:

- Status:** 200 OK
- Time:** 18 ms
- Size:** 331 B
- Body Content (JSON):**

```
1 {  
2   "_id": "65f5ebfceb94411c8d628ec",  
3   "name": "Jaya",  
4   "tech": "Blockchain",  
5   "subscription": true,  
6   "__v": 0  
7 }
```

RESTful API

4. Delete Request – Destroy

The screenshot displays the Postman interface for a DELETE request. The top navigation bar includes 'Home', 'Workspaces', and 'API Network'. The left sidebar shows 'Collections' and 'Environments'. The main workspace is titled 'REST API basics: CRUD, test & variable / Update data'. The request method is set to 'DELETE' and the URL is 'http://localhost:9989/users/65f69e157f1500cd0fc4113'. The 'Send' button is visible. Below the URL bar, tabs for 'Params', 'Authorization', 'Headers (9)', 'Body', 'Pre-request Script', 'Tests', and 'Settings' are shown. The 'Query Params' table is empty. The 'Body' tab is selected, showing a JSON response in 'Pretty' format. The status bar indicates 'Status: 200 OK', 'Time: 18 ms', and 'Size: 383 B'.

DELETE `http://localhost:9989/users/65f69e157f1500cd0fc4113` **Send**

Params **Authorization** **Headers (9)** **Body** **Pre-request Script** **Tests** **Settings** **Cookies**

Query Params

Key	Value	Description	...	Bulk Edit
Key	Value	Description		

Body **Cookies** **Headers (7)** **Test Results (1/1)** **Status: 200 OK** **Time: 18 ms** **Size: 383 B** **Save as example**

Pretty **Raw** **Preview** **Visualize** **JSON**

```
1 {
2   "message": "User deleted successfully",
3   "deletedUser": {
4     "_id": "65f69e157f1500cd0fc4113",
5     "name": "Tarun",
6     "tech": "Intern",
7     "subscription": true,
8     "__v": 0
9   }
10 }
```

Online Find and replace Console Postbot Runner Start Proxy Cookies Trash

RESTful API

MongoDB Compass

The screenshot displays the MongoDB Compass web interface for a local host. The left sidebar shows a tree view of databases and collections, with 'usersDB' expanded and 'users' selected. The main panel shows the 'usersDB.users' collection with 9 documents and 1 index. The 'Documents' tab is active, displaying a list of documents. Each document is shown in a JSON format. The bottom status bar indicates the current shell is 'MongoSh'.

localhost

My Queries usersDB users

usersDB.users

9 DOCUMENTS 1 INDEXES

Documents Aggregations Schema Indexes Validation

Filter Type a query: { field: 'value' } or Generate query

ADD DATA EXPORT DATA UPDATE DELETE

1 - 9 of 9

```
{ "_id": ObjectId("65f5ebd4eba94411c8d628ea"), "name": "Govind", "tech": "NodeJs", "subscription": true, "__v": 0 }
```

```
{ "_id": ObjectId("65f5ebfceb94411c8d628ec"), "name": "Jaya", "tech": "Blockchain", "subscription": true, "__v": 0 }
```

```
{ "_id": ObjectId("65f5ec0feba94411c8d628ee"), "name": "Sai", "tech": "IT Manager", "subscription": false, "__v": 0 }
```

```
{ "_id": ObjectId("65f5ec1ceba94411c8d628f0"), "name": "Divya", "tech": "Backend", "subscription": false, "__v": 0 }
```

```
{ "_id": ObjectId("65f5ec3ceba94411c8d628f2"), "name": "Navya", "tech": "Transaction Risk Operator", "subscription": false, "__v": 0 }
```

> MONGOSH

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)

RESTful API

- "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>