Experiment No. 4

Aim: REST API Design with MongoDB + Mongoose Integration

Code :

index.js

```
require("dotenv").config();
const express = require("express");
const cors = require("cors");
const connectDB = require("./config/db");
const userRoutes = require("./routes/userRoutes");

const app = express();

// Middleware
app.use(cors());
app.use(express.json());

// Connect DB only if not testing
if (process.env.NODE_ENV !== "test") {
  connectDB();
}

// Routes
app.use("/users", userRoutes);

// Start server only if not testing
if (process.env.NODE_ENV !== "test") {
  const PORT = process.env.PORT || 5000;
  app.listen(PORT, () =>
    console.log(`🚀 Server running at http://localhost:${PORT}`)
  );
}

module.exports = app; // ✅ export app for supertest
```

routes / userRoutes.js

```
const express = require("express");
const {
  createUser,
  getUsers,
  getUserById,
  updateUser,
  deleteUser,
} = require("../controllers/userController");

const router = express.Router();

router.post("/", createUser);
router.get("/", getUsers);
router.get("/:id", getUserById);
router.put("/:id", updateUser);
router.delete("/:id", deleteUser);

module.exports = router;

```

Model / User.js

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

const userSchema = new mongoose.Schema(
  {
    name: { type: String, required: true },
    email: { type: String, required: true, unique: true },
    age: { type: Number, required: true },
  },
  { timestamps: true }
);

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

```

Config/db.js

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

const connectDB = async () => {
  console.log(process.env.MONGO_URI);
  try {
    await mongoose.connect(process.env.MONGO_URI);
    console.log("✅ MongoDB Connected");
  } catch (error) {
    console.error("❌ MongoDB Connection Failed:", error.message);
    process.exit(1);
  }
};

module.exports = connectDB;
```

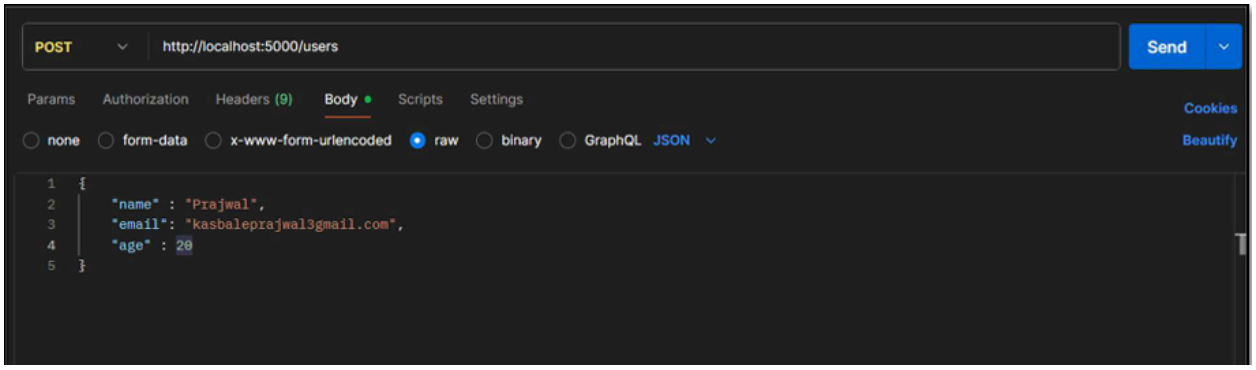
Controller / userController.js

```
const User = require("../models/User");

exports.createUser = async (req, res) => {
  try {
    const user = await User.create(req.body);
    res.status(201).json({ success: true, data: user });
  } catch (err) {
    res.status(400).json({ success: false, message: err.message });
  }
};

exports.getUsers = 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 });
  }
};

exports.getUserById = async (req, res) => {
  try {
    const user = await User.findById(req.params.id);
    if (!user) return res.status(404).json({ success: false, message: "User not found" });
    res.json({ success: true, data: user });
  } catch (err) {
    res.status(500).json({ success: false, message: err.message });
  }
};
```

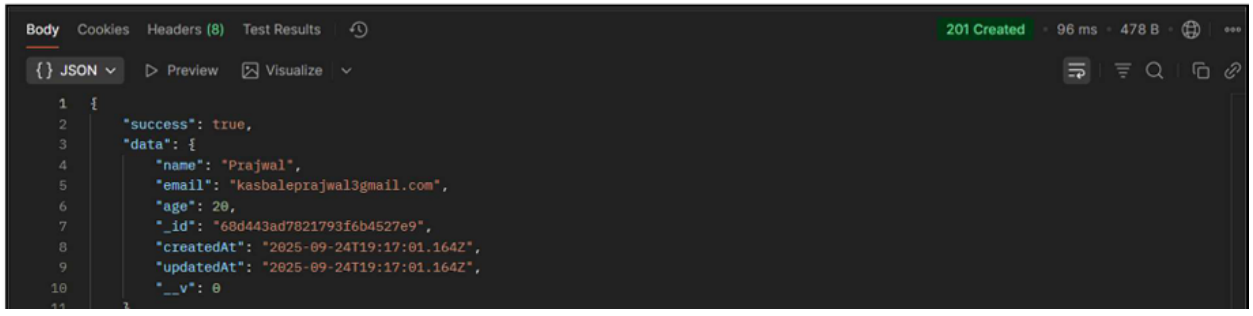
```
exports.updateUser = async (req, res) => {
  try {
    const user = await User.findByIdAndUpdate(req.params.id, req.body, {
      new: true,
      runValidators: true,
    });
    if (!user) return res.status(404).json({ success: false, message: "User not found" });
    res.json({ success: true, data: user });
  } catch (err) {
    res.status(400).json({ success: false, message: err.message });
  }
};

// 🔹 DELETE
exports.deleteUser = async (req, res) => {
  try {
    const user = await User.findByIdAndDelete(req.params.id);
    if (!user) return res.status(404).json({ success: false, message: "User not found" });
    res.json({ success: true, message: "User deleted successfully" });
  } catch (err) {
    res.status(500).json({ success: false, message: err.message });
  }
};
```

Output :
Create User (POST)

Request :-

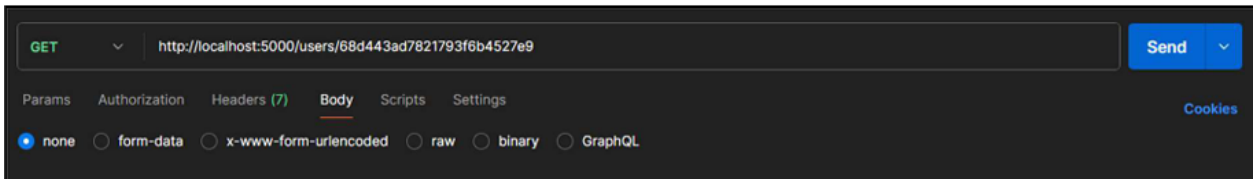


Response : -



Fetch User Data (Get User by ID)

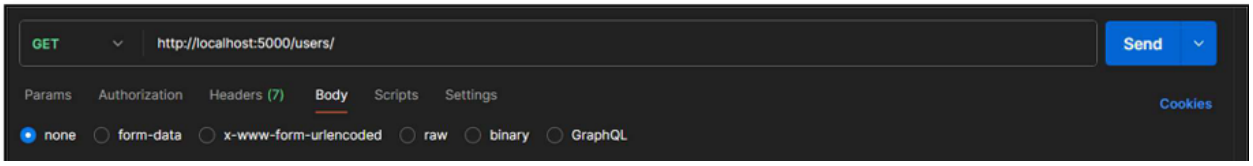
Request :-



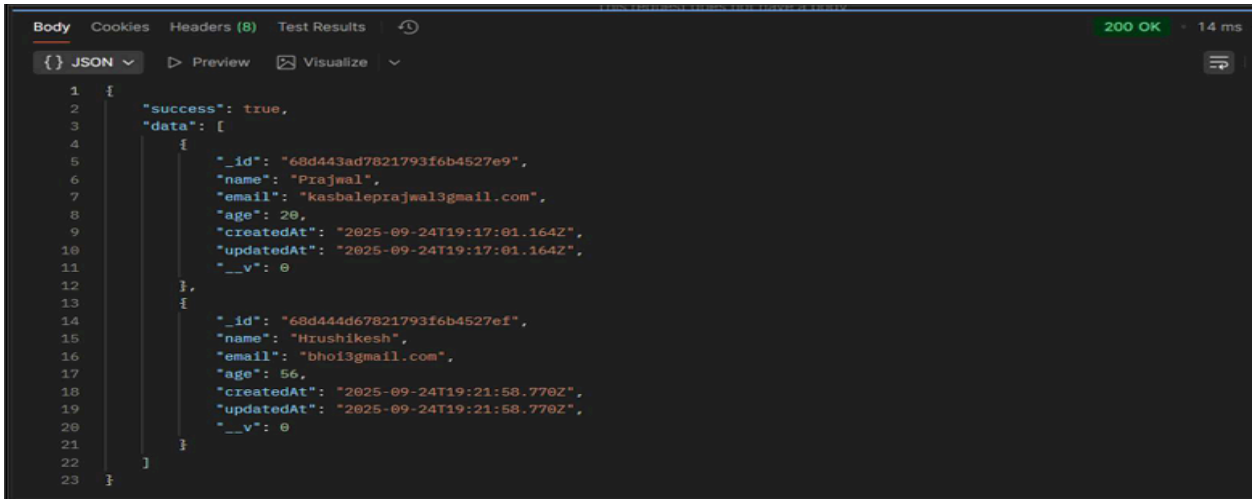
Response :-

Fetch All User (Get)

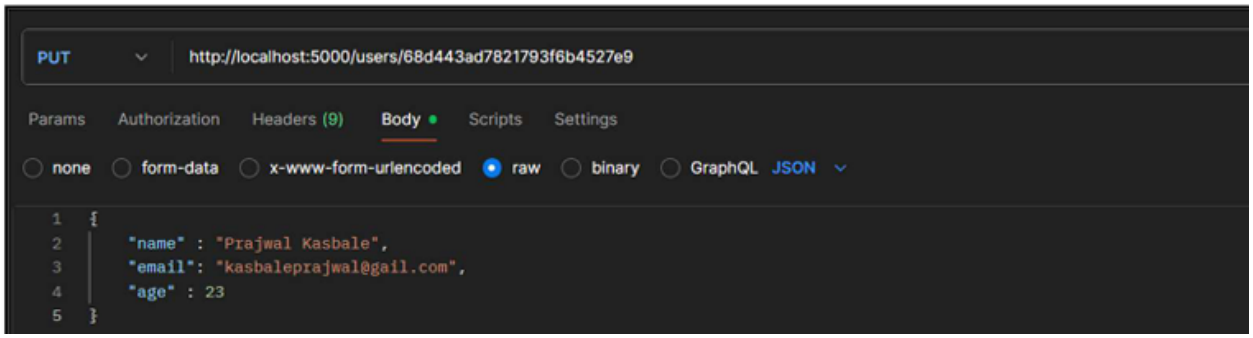
Request :-



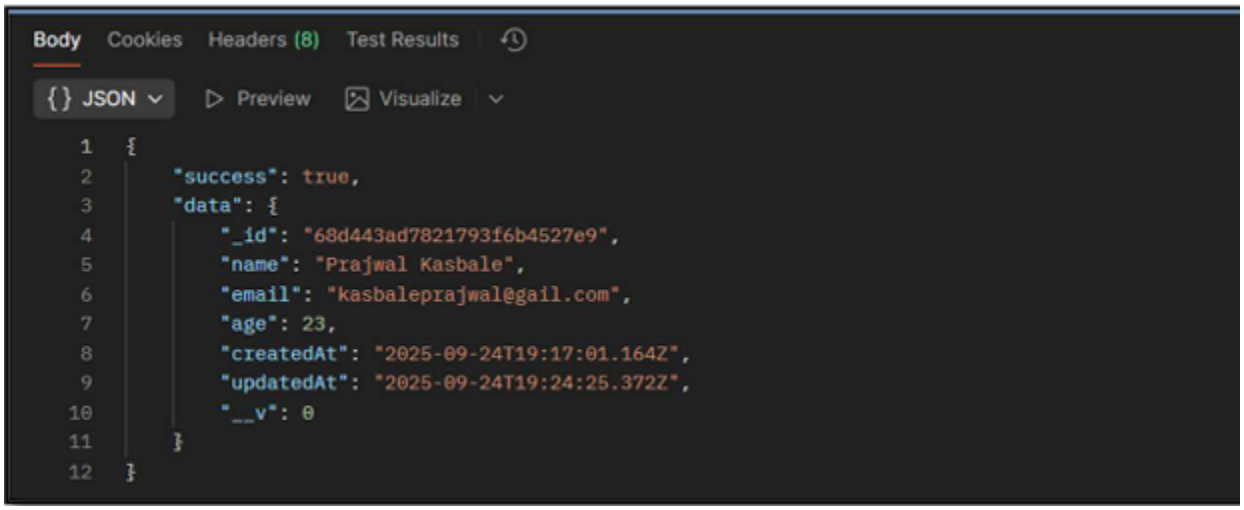
Response :-



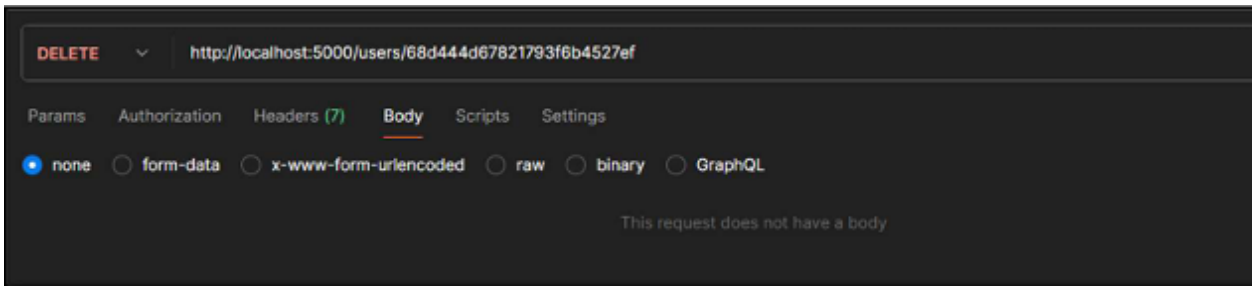
Update User (PUT)
Request :-



Response :-

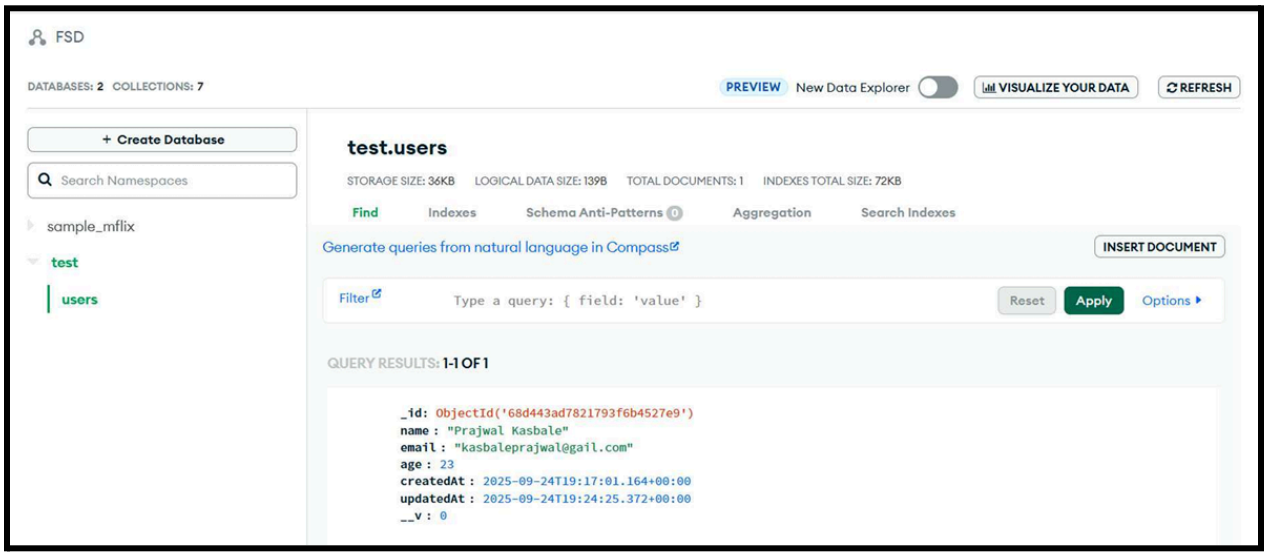


Delete User (DELETE)
Request :-



Response :-

Mongo Atlas Server



Conclusion :-

Designing a **REST API with MongoDB and Mongoose** enables efficient data handling through a flexible schema and powerful querying capabilities. By integrating Mongoose, developers gain schema validation, middleware, and model-based interactions, simplifying database operations. A well-structured RESTful design—combined with proper routing, error handling, and security practices—ensures the API is **scalable, maintainable, and ready for production**.