EXPERIMENT

AIM

REST API Design with MongoDB + Mongoose Integration.

THEORY

1. What is a REST API?

- REST (Representational State Transfer) is an architectural style for building web services.
- A **REST API** uses HTTP methods (GET, POST, PUT, DELETE) to perform operations on resources.
- Each resource is represented by a **URL (endpoint)**, e.g. /api/users.

Example mappings:

- GET /api/users → Fetch all users
- POST /api/users → Add new user
- PUT /api/users/:id → Update user by ID
- DELETE /api/users/:id → Delete user by ID

Why MongoDB?

- **MongoDB** is a NoSQL database that stores data in flexible **JSON-like documents**.
- Useful for modern applications where schema can change dynamically.
- Example MongoDB document:

3. Why Mongoose?

- **Mongoose** is an ODM library for MongoDB and Node.js.
- It allows developers to:
    - Define **schemas** for data.
    - Create **models** to interact with MongoDB collections.
    - Perform CRUD operations easily.

4. REST API Workflow with MongoDB + Mongoose

1. **Client** sends requests (e.g., GET /api/users).
2. **Express server** receives request.
3. **Route handler** calls a **Mongoose model** to interact with MongoDB.
4. **The database** responds with data.
5. **Express** sends the response back to the client in JSON format.

5. Advantages of this Integration

- **Scalability**: MongoDB handles large datasets.
- **Flexibility**: JSON-like structure matches REST responses.
- **Productivity**: Mongoose simplifies queries with built-in methods.
- **Separation of Concerns**: REST API design keeps client and server independent.

STEPS

1) Create project and install deps

mkdir rest-experiment

cd rest-experiment

npm init -y

npm i express mongoose dotenv morgan express-async-errors

npm i -D nodemon

2) Create `.env` file

MONGO_URI=your_mongodb_uri_here

PORT=5000

NODE_ENV=development

SOURCE CODE

config > JS db.js > ...

```
// config/db.js
const mongoose = require('mongoose');


const connectDB = async () => {
try {
const conn = await mongoose.connect(process.env.MONGO_URI);
console.log(`MongoDB Connected: ${conn.connection.host}`);
} catch (error) {
console.error('MongoDB connection error:', error.message);
process.exit(1);
}
};


module.exports = connectDB;
```

controllers > JS userController.js > ...

```
// controllers/userController.js
const User = require('../models/User');


exports.createUser = async (req, res) => {
const { name, email, age } = req.body;
if (!name || !email) return res.status(400).json({ message: 'Name & email required' });
const userExists = await User.findOne({ email });
if (userExists) return res.status(400).json({ message: 'Email already used' });


const user = await User.create({ name, email, age });
res.status(201).json(user);
};


exports.getUsers = async (req, res) => {
const users = await User.find().select('-__v');
res.json(users);
};


exports.getUserById = async (req, res) => {
const user = await User.findById(req.params.id).select('-__v');
if (!user) return res.status(404).json({ message: 'User not found' });
res.json(user);
};


```

middleware > JS errorMiddleware.js > ...

```js
// middleware/errorMiddleware.js
exports.notFound = (req, res, next) => {
res.status(404).json({ message: `Not Found - ${req.originalUrl}` });
};


exports.errorHandler = (err, req, res, next) => {
const status = res.statusCode === 200 ? 500 : res.statusCode;
res.status(status).json({ message: err.message, stack: process.env.NODE_ENV === 'production' ? r
};
```

models > JS User.js > ...

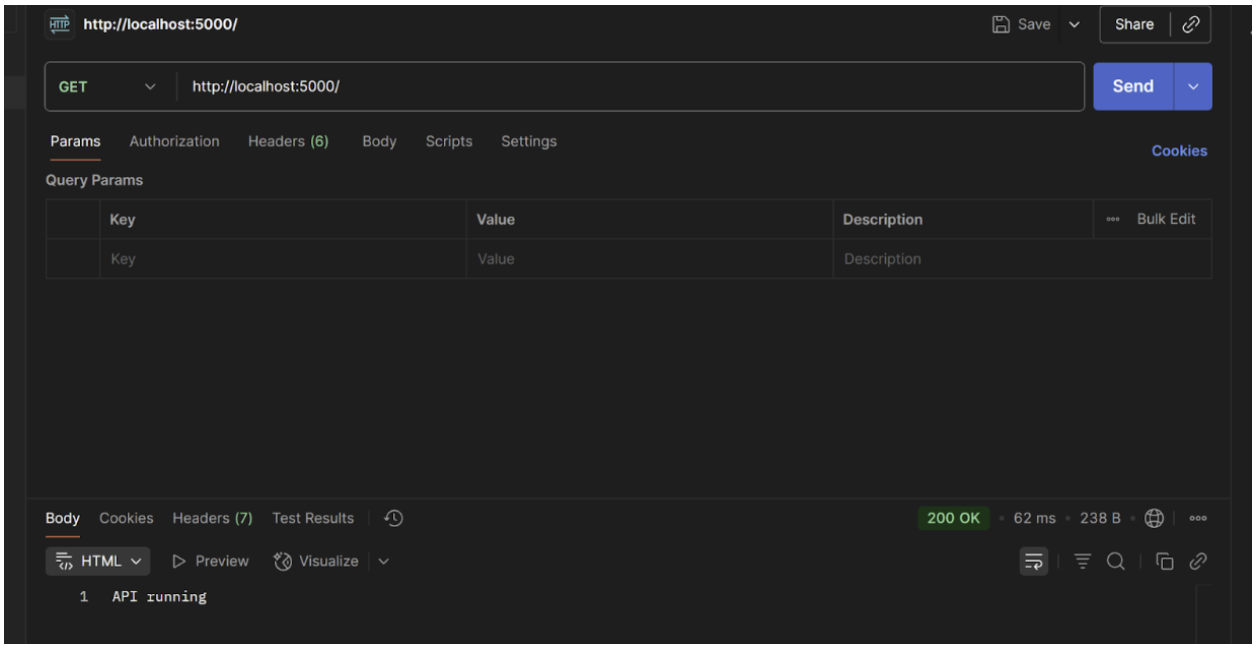
```js
// models/User.js
const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({
name: { type: String, required: true },
email: { type: String, required: true, unique: true },
age: { type: Number, default: null },
createdAt: { type: Date, default: Date.now }
});

// hide __v when converting to JSON
userSchema.set('toJSON', { transform: (doc, ret) => { delete ret.__v; return ret; } });
module.exports = mongoose.model('User', userSchema);
```

routes > JS userRoutes.js > ...

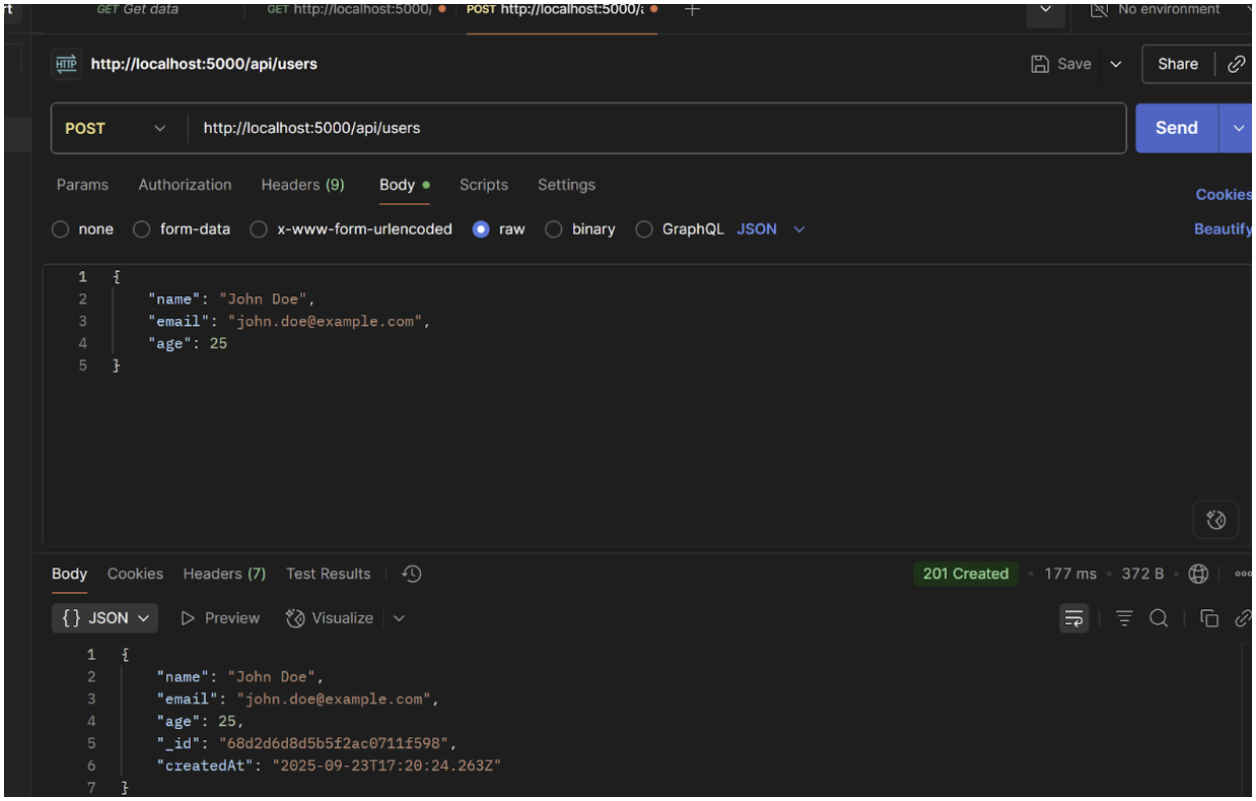
```js
// routes/userRoutes.js
const express = require('express');
const router = express.Router();
const controller = require('../controllers/userController');


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


module.exports = router;
```

OUTPUT

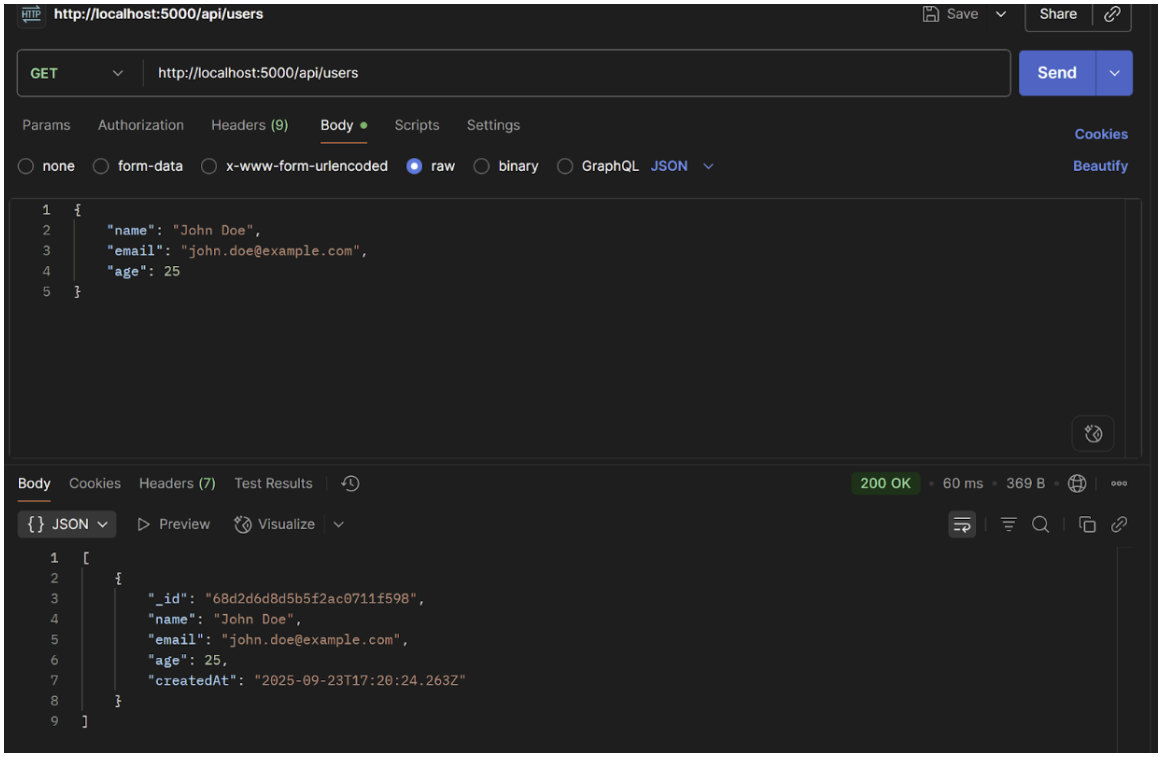
Request 1: Check if server is alive



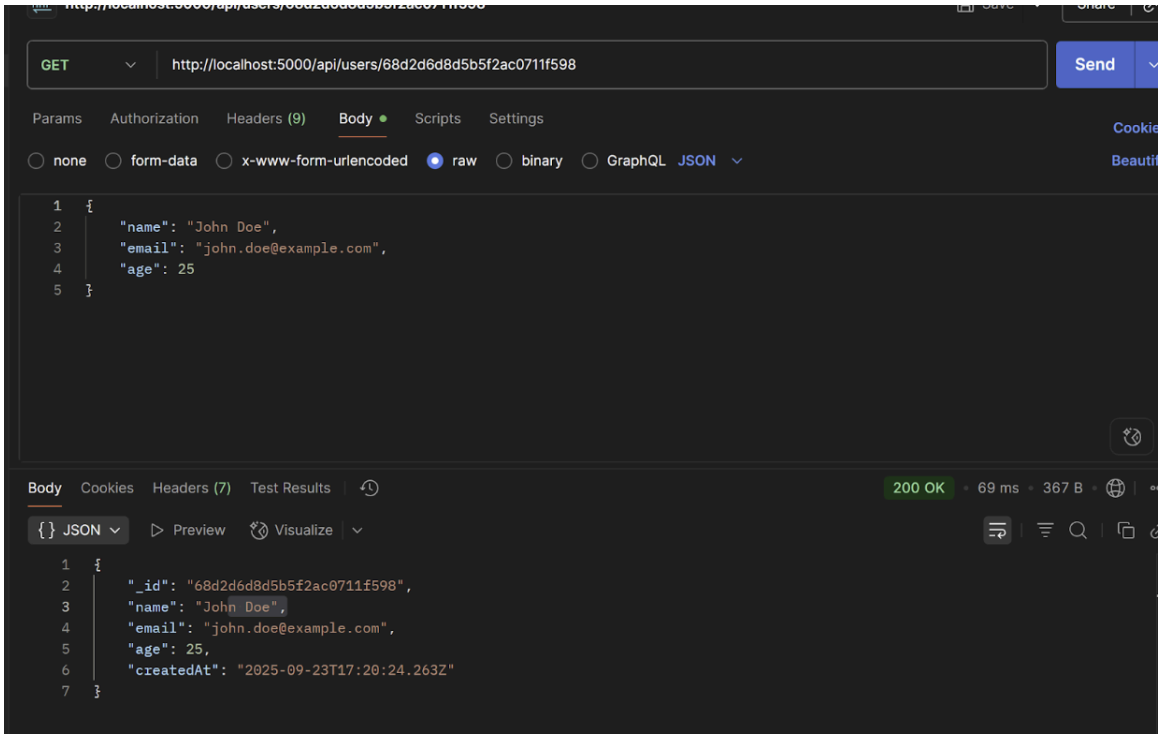
Request 2: Create a new user



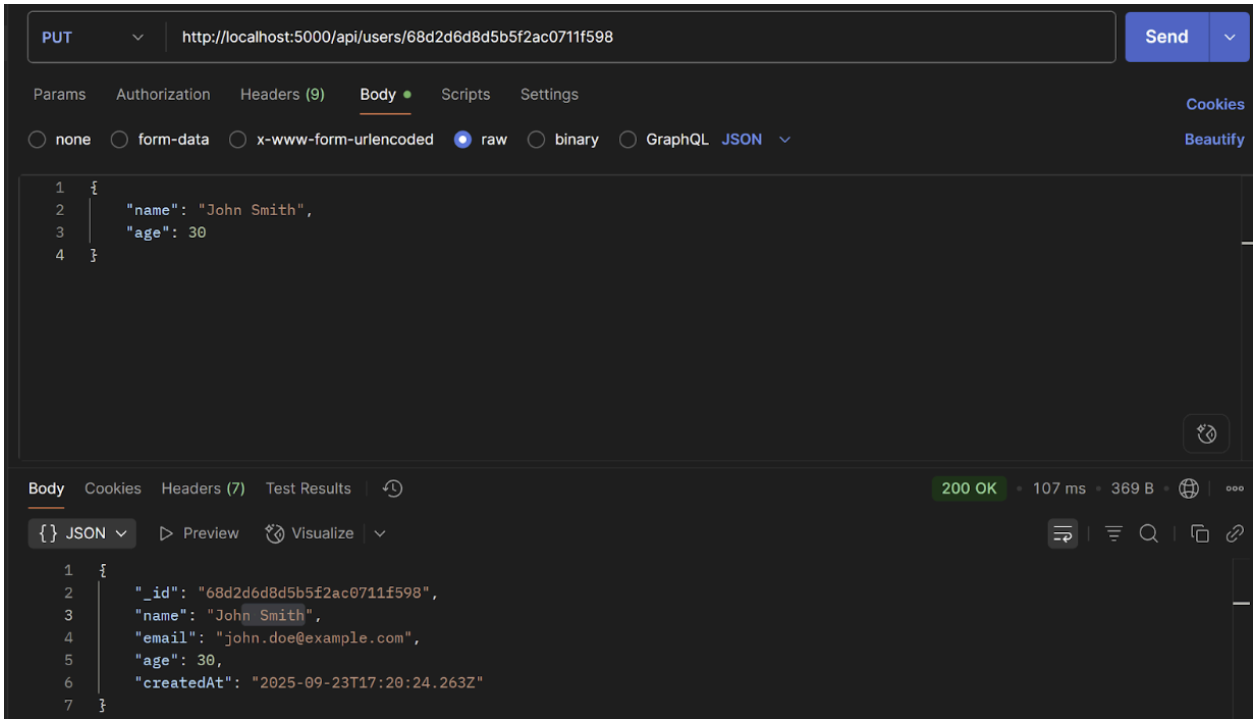
Request 3: Get all users



Request 4 : Get user by ID



Request 5: Update user



CONCLUSION

This experiment demonstrated how to design a RESTful API using **Node.js, Express, MongoDB, and Mongoose**. It showed how CRUD operations can be performed efficiently with well-structured endpoints and how Mongoose simplifies interaction with MongoDB. The integration provides a scalable and flexible way to build modern backend applications.