Experiment - 4

REST API Design with MongoDB + Mongoose Integration

Aim

To design and implement a REST API integrated with MongoDB using Mongoose, demonstrated through a **Budget Tracker Project** tested with **Postman API**.

Theory

🔹 1. Backend Development with Node.js

- **What is Node.js?**
  Node.js is a **runtime environment** that allows JavaScript to run outside the browser (on a server).
    - Built on Chrome's **V8 Engine**.
    - Handles asynchronous, event-driven requests efficiently.
- **Why Node.js?**
    - High performance for I/O-heavy apps.
    - Uses JavaScript (already known by most developers).
    - Has a huge ecosystem of libraries (npm).

👉 **Example**:
Without Node.js, JavaScript only runs in the browser. With Node.js, we can run:

```
console.log("Hello Backend with Node.js");
```

directly in terminal using `node file.js`.

🔹 2. Express.js

- **What is Express.js?**
  Express is a **framework for Node.js** that simplifies server development.

- **Key Features**:
  - Middleware (functions between request & response).
  - Routing (`/expenses`, `/users`).
  - Error handling.

👉 **Example without Express** (pure Node.js server):

```
const http = require("http");
http.createServer((req, res) => {
  res.write("Hello World");
  res.end();
}).listen(3000);
```

👉 **Example with Express**:

```
import express from "express";
const app = express();

app.get("/", (req, res) => res.send("Hello Express!"));
app.listen(3000, () => console.log("Server running on port 3000"));
```

Clearly, **Express is simpler and more readable**.

🔹 3. REST API Design

- **What is an API?**
  API = Application Programming Interface. It defines how two applications talk.

- **What is REST?**
  REST (Representational State Transfer) is an architectural style that uses **HTTP methods** for CRUD operations.

HTTP Method	Operation	Example in Project
GET	Read	`/expenses` → Get all expenses
POST	Create	`/expenses` → Add new expense
PUT	Update	`/expenses/:id` → Update expense
DELETE	Delete	`/expenses/:id` → Remove expense

👉 **Example**:

- Client: Sends POST `/expenses` with `{ "title": "Lunch", "amount": 150 }`.
- Server: Saves expense in DB and returns the created object.

🔹 4. MongoDB

- **What is MongoDB?**
  A **NoSQL database** that stores data in **BSON (binary JSON)** format.
- **Why MongoDB?**
  - Flexible schema (unlike SQL).
  - Scales well with large data.
  - Works naturally with JSON objects in Node.js.

👉 **Example Document in MongoDB**:

```
{
  "_id": "64a12b45...",
```

```
  "title": "Lunch",
  "amount": 150,
  "category": "Food",
  "date": "2025-08-19T12:00:00Z"
}
```

🔹 5. MongoDB Atlas

- Cloud-hosted version of MongoDB.
- Provides a free cluster, accessible anywhere.
- Includes tools for monitoring, scaling, and backups.

👉 **Steps to Use Atlas**:

1. Create Atlas account.
2. Create Cluster.
3. Add Database → Collection (`expenses`).
4. Copy Connection String (URI).
5. Store in `.env` file as `MONGO_URI`.

🔹 6. Mongoose

- **Why Mongoose?**
  - Node.js talks to MongoDB through queries. Mongoose makes it easier.
  - Adds schemas (rules) to MongoDB.
- **Features**:
  - Schema definition.

- Built-in validators.
- Query helpers.

👉 **Example Schema**:

```
import mongoose from "mongoose";
const expenseSchema = new mongoose.Schema({
  title: { type: String, required: true },
  amount: { type: Number, required: true }
});
```

👉 **Example Query with Mongoose**:

```
const expense = new Expense({ title: "Dinner", amount: 200 });
await expense.save();
```

🔹 7. Postman

- Tool for manually testing APIs.
- Lets developers send GET, POST, PUT, DELETE requests easily.

👉 **Example**:
Send a POST request:

```
{
  "title": "Movie",
  "amount": 300
}
```

Response:

```
{
  "_id": "64a12b45...",
  "title": "Movie",
  "amount": 300,
```

```
  "date": "2025-08-19T12:00:00Z"
}
```

🛠️ Implementation Steps with Commands

Initialize Project

```
npm init -y
```

Install Dependencies

```
npm install express mongoose dotenv cors morgan bcryptjs
npm install --save-dev nodemon jest supertest @babel/core
@babel/preset-env babel-jest cross-env
```

Add Scripts in `package.json`

```
"scripts": {
  "start": "node server.js",
  "dev": "nodemon server.js",
  "test": "jest --watchAll"
}
```

Create `.env` file

```
PORT=5000
MONGO_URI=mongodb+srv://username:password@cluster0.mongodb.net/expense
s
```

Run Project

```
npm start   # Start normally
```

```
npm run dev # Start with nodemon

npm test    # Run tests
```

`.env` and Environment Variables

- A `.env` file is used to store **sensitive information** such as:
  - Database connection string (`MONGO_URI`)
  - Port number (`PORT`)
  - Secret keys or API keys (`API_KEY`)

👉 **Why use `.env`?**

- Keeps sensitive info out of source code.
- Protects secrets when pushing to GitHub.
- Makes app configurable (different keys for dev, test, production).

👉 **Example `.env` file in our project**:

```
PORT=5000
MONGO_URI=mongodb+srv://username:password@cluster0.mongodb.net/expense
s
API_KEY=12345secretkey
```

🔹 2. API Key Authentication

- To **restrict unauthorized access**, we can require clients to send an **API Key** in the request.
- In our project:

- API key is stored in `.env` as `API_KEY`.
- Postman requests must include this in the **Headers**.

👉 **Postman Header Example**:

- **Key** = `x-api-key`
- **Value** = `12345secretkey`

👉 **How it works**:

1. Client sends request → with `x-api-key` header.
2. Server middleware checks if the key matches `.env`.
3. If valid → request continues. If not → `401 Unauthorized`.

🔹 3. Middleware for API Key

We add a **custom middleware** to check the key.

```
import dotenv from "dotenv";
dotenv.config();

const apiAuth = (req, res, next) => {
  const apiKey = req.header("x-api-key");
  if (apiKey && apiKey === process.env.API_KEY) {
    next(); // Key valid → continue
  } else {
      res.status(401).json({ message: "Unauthorized: Invalid API Key"
});
  }
};

export default apiAuth;
```

Then in `server.js`:

```
import apiAuth from "./middleware/apiAuth.js";
app.use(apiAuth); // Protect all routes
```

💻 Postman Example with API Key

POST /api/expenses

Headers:

```
x-api-key : 12345secretkey
```

Request Body:

```
{
  "title": "Lunch",
  "amount": 150,
  "category": "Food"
}
```

Response:

```
{
  "_id": "64a12b45...",
  "title": "Lunch",
  "amount": 150,
  "category": "Food",
  "date": "2025-08-19T12:00:00Z"
}
```

👉 If wrong key is given:

```
{ "message": "Unauthorized: Invalid API Key" }
```

Code Snippets

1. models/Expense.js (Mongoose Integration)

👉 Shows schema & proves Mongoose is used.

backend > models > Expense.js > default

```javascript
import mongoose from 'mongoose';

const expenseSchema = new mongoose.Schema({
  title: { type: String, required: true, trim: true },
  amount: { type: Number, required: true },
  category: { type: String, required: true, trim: true },
  date: { type: Date, default: Date.now }
});

export default mongoose.model('Expense', expenseSchema);
```

2. server.js (Entry Point)

👉 Shows Express setup, MongoDB Atlas connection, and middleware integration.

backend > server.js > ...

```
import express from 'express';
import mongoose from 'mongoose';
import dotenv from 'dotenv';
import { auth, errorHandler } from './middleware/errorMiddleware.js';
import expenseRoutes from './routes/expenseRoutes.js';

// Configure dotenv
dotenv.config({ path: 'D:/SEM5/sem5_FSD/FSD_4THEXP_BudgetAPI/.env', debug: true });
console.log('🔍 .env loaded:', process.env);
console.log('🔍 MONGO_URI:', process.env.MONGO_URI);

const app = express();
const PORT = process.env.PORT || 5000;

app.use(express.json());
app.use('/expenses', expenseRoutes);
app.use(errorHandler);

// Connect to MongoDB
mongoose.connect(process.env.MONGO_URI)
  .then(() => console.log('✅ MongoDB Connected'))
  .catch((err) => console.error('❌ DB Error:', err));

// Only start server if not testing
if (process.env.NODE_ENV !== 'test') {
  app.listen(PORT, () => {
    console.log(`🚀 Server running on port ${PORT}`);
  });
}

// Export app for Supertest
export default app;
```

3. routes/expenseRoutes.js (Routing)

backend > routes > expenseRoutes.js > ...

```
import express from 'express';
import { auth } from '../middleware/errorMiddleware.js';
import {
  createExpense,
  getExpenses,
  getExpenseById,
  updateExpense,
  deleteExpense,
  getCategories    // <- new controller
} from '../controllers/expenseController.js';

const router = express.Router();

// --- Categories route must come BEFORE /:id ---
router.get('/categories', auth, getCategories);

// CRUD routes
router.post('/', auth, createExpense);
router.get('/', auth, getExpenses);
router.get('/:id', auth, getExpenseById);
router.put('/:id', auth, updateExpense);
router.delete('/:id', auth, deleteExpense);

export default router;
```

4. Controllers/expenseController.js (REST Logic Example)

1. createExpense

```
if (amount <= 0) throw new Error('Amount must be positive');
const expense = await Expense.create({ title, amount, category });
```

```
export const createExpense = async (req, res) => {
  try {
    const { title, amount, category } = req.body;
    if (amount <= 0) throw new Error('Amount must be positive');
    const expense = await Expense.create({ title, amount, category });
    res.status(201).json(expense);
  } catch (error) {
    res.status(400).json({ message: error.message });
  }
};
```

2. **getExpenses**

```
export const getExpenses = async (req, res) => {
  try {
    const expenses = await Expense.find();
    res.status(200).json(expenses);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
};
```

3. getExpenseById

```
const expense = await Expense.findById(req.params.id);
if (!expense) return res.status(404).json({ message: 'Expense not found' });
```

4. updateExpense

```
if (amount <= 0) throw new Error('Amount must be positive');
const expense = await Expense.findByIdAndUpdate(
```

```
export const updateExpense = async (req, res) => {
  try {
    const { title, amount, category } = req.body;
    if (amount <= 0) throw new Error('Amount must be positive');
    const expense = await Expense.findByIdAndUpdate(
      req.params.id,
      { title, amount, category },
      { new: true, runValidators: true }
    );
    if (!expense) return res.status(404).json({ message: 'Expense not found' });
    res.status(200).json(expense);
  } catch (error) {
    res.status(400).json({ message: error.message });
  }
};
```

5. **deleteExpense**

```
export const deleteExpense = async (req, res) => {
  try {
    const expense = await Expense.findByIdAndDelete(req.params.id);
    if (!expense) return res.status(404).json({ message: 'Expense not found' });
    res.status(204).send();
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
};
```

6. getCategories (Advanced Feature)

```
// --- Updated getCategories ---
export const getCategories = async (req, res) => {
  try {
    const categories = await Expense.aggregate([
      {
        $group: {
          _id: "$category",            // Group by category
          totalAmount: { $sum: "$amount" } // Sum of amounts
        }
      }
    ]);
    res.status(200).json(categories);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
};
```

5. middleware/apiAuth.js (API Key Authentication)

```
backend > middleware > errorMiddleware.js > ...
import jwt from 'jsonwebtoken';

const auth = (req, res, next) => {
  const apiKey = req.header('x-api-key');
  if (!apiKey || apiKey !== process.env.API_KEY) {
    return res.status(401).json({ message: 'Unauthorized' });
  }
  next();
};

const errorHandler = (err, req, res, next) => {
  console.error(err.stack);
  res.status(500).json({ message: 'Internal Server Error' });
};

export { auth, errorHandler };
```

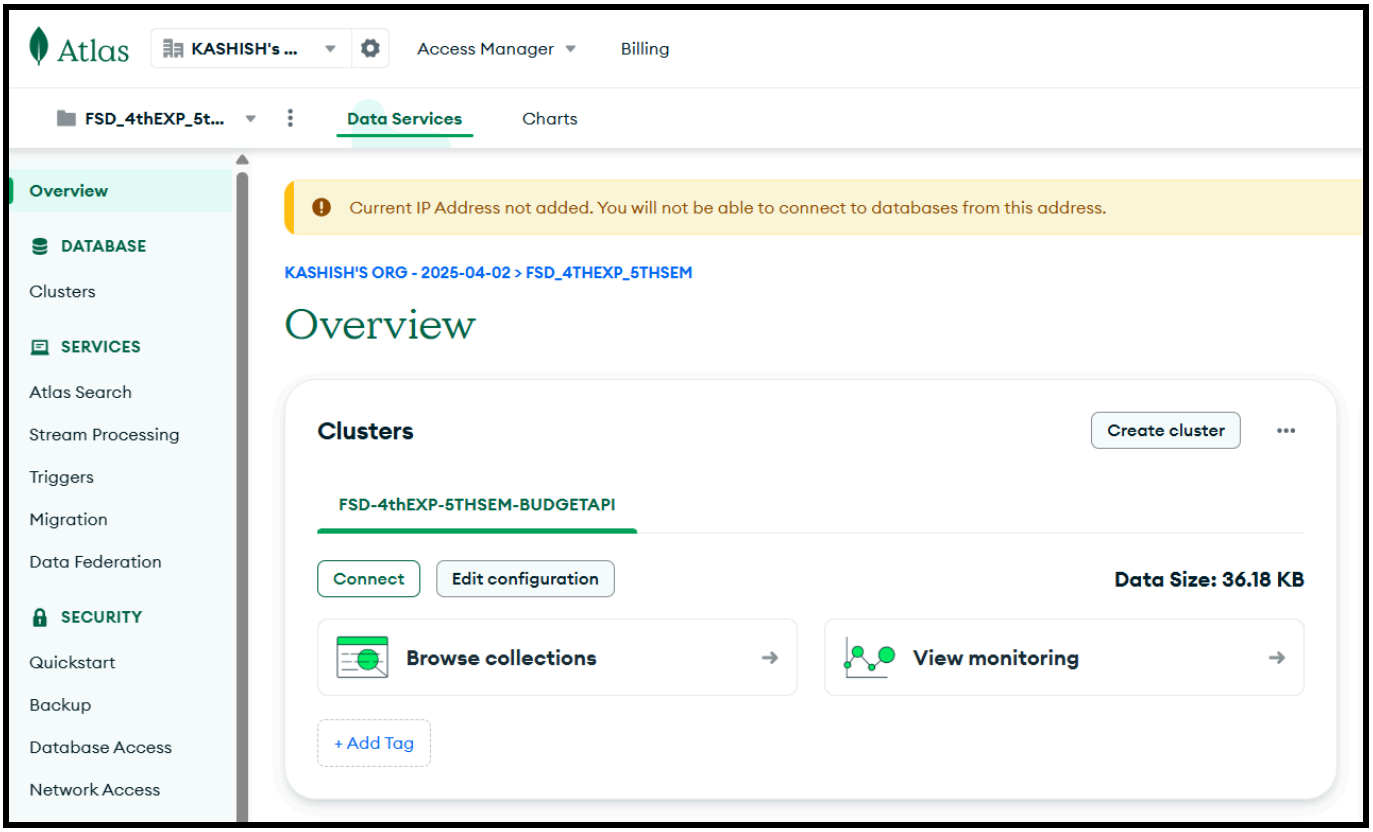
.env Example (Environment Variables)

```
.env
MONGO_URI=mongodb+srv://FSD_4thEXP_5thSEM_BudgetAPI:thisismetrying8@fsd-4thexp-5thsem-budge.tkdvufm.mongodb.net/BudgetAPI?retryWrites=true&w=majority&appName=FSD-4thEXP-5THSEM-BUDGETAPI
PORT=5000
API_KEY=thisismetrying8
```

MongoDB Atlas Setup

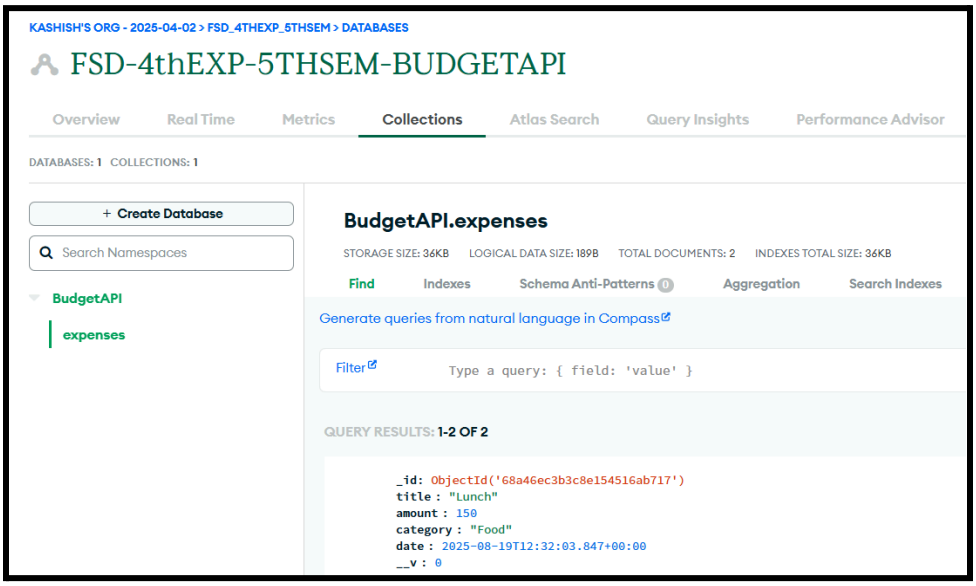
1. **Atlas Dashboard (Cluster Overview)**

This screenshot shows my MongoDB Atlas cluster. The cluster is the cloud-hosted MongoDB database where all my expense data will be stored.”



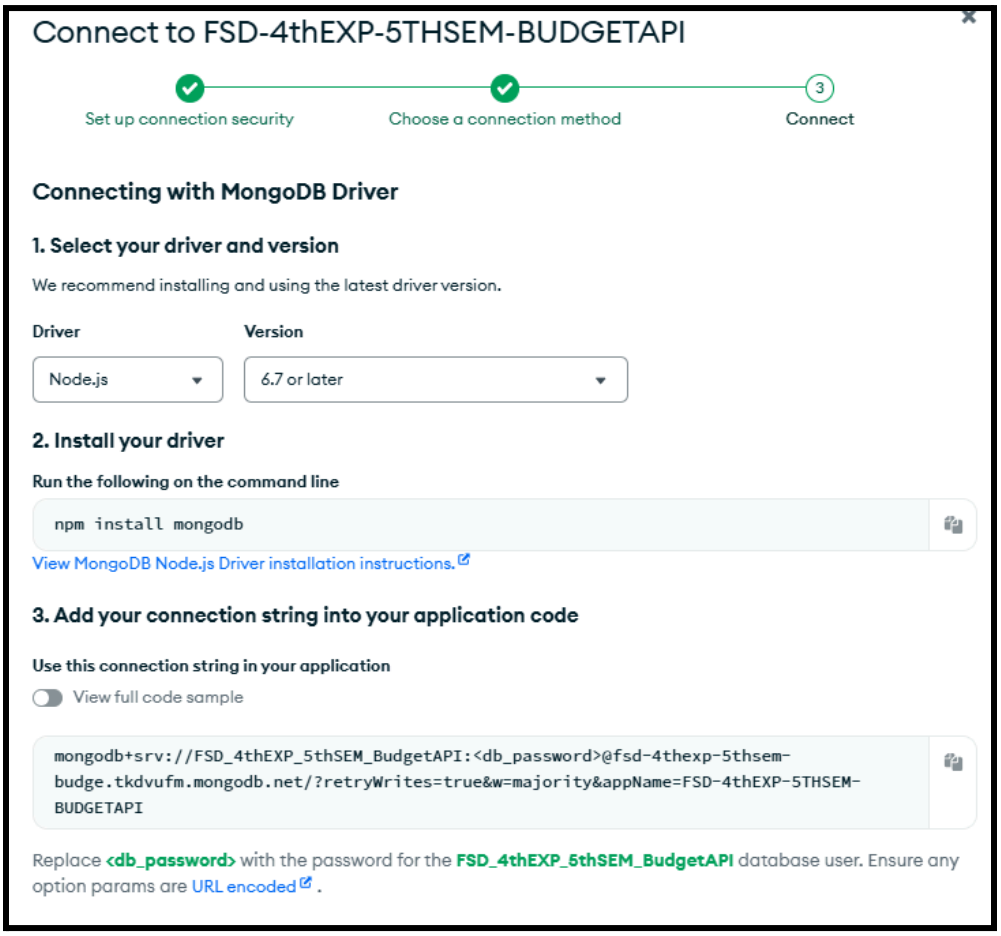
2. **Database → Collections Page**

Here you can see the expenses collection in MongoDB Atlas. This is where my Node.js + Express app stores expense records like title, amount, and category."



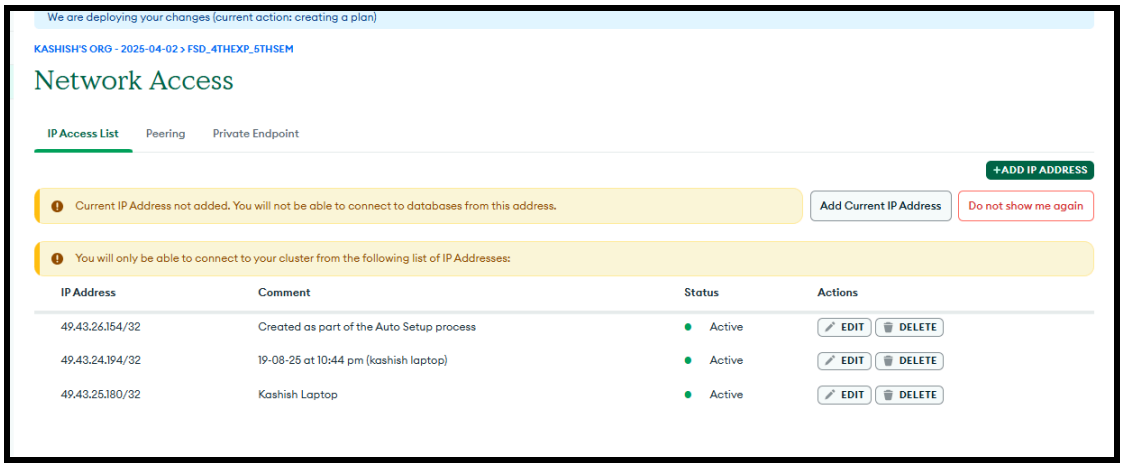
3. **Connection String (Connect → Drivers)**

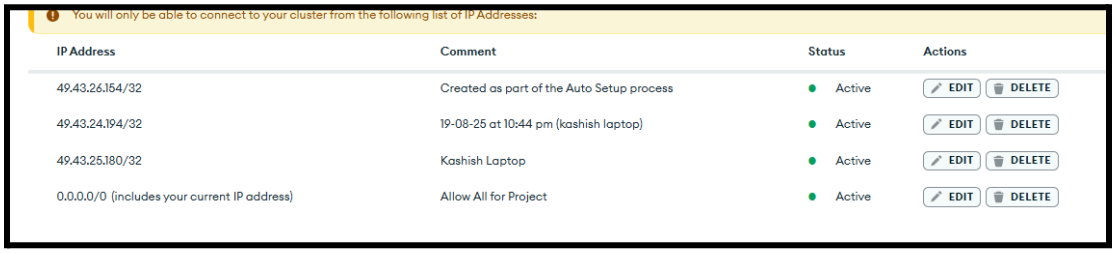
This is the MongoDB connection string. We copy it and store it in `.env` as `MONGO_URI`. Our backend uses Mongoose to connect to this string."



4. **Database User / Network Access (Optional but Impressive)**

- ***To secure the database, MongoDB Atlas requires a database user and IP whitelist. I created a user for authentication and allowed access from my system's IP address."***





"Initially Atlas denied access because my IP was not whitelisted. After adding 0.0.0.0/0 in Network Access, the backend successfully connected."

🔹 CRUD Operations with Postman (Step by Step)

---

1. CREATE (POST) — New Expense Add Karna

Method: **POST**

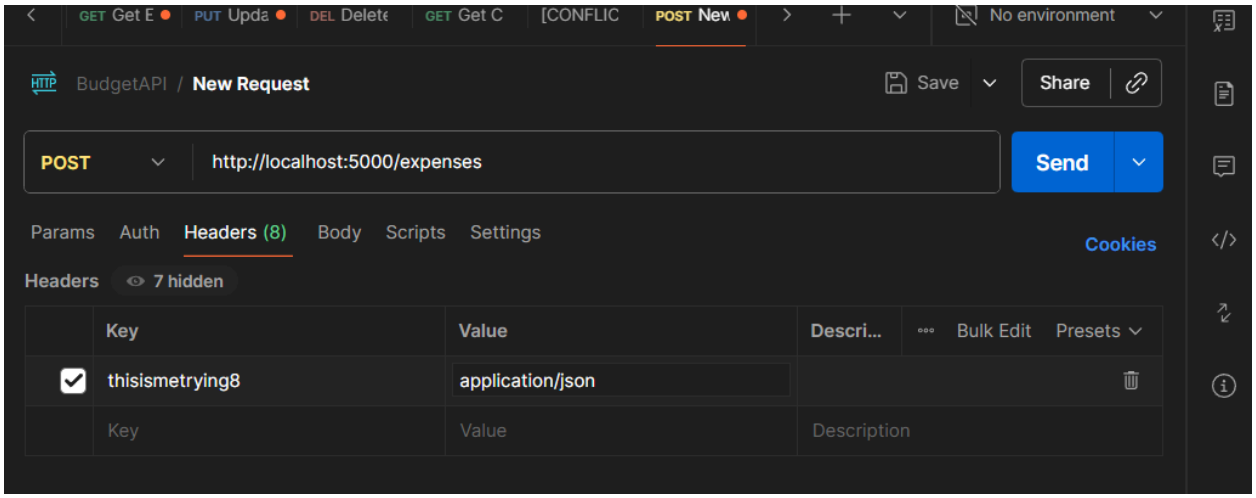
URL:
`http://localhost:5000/expense`

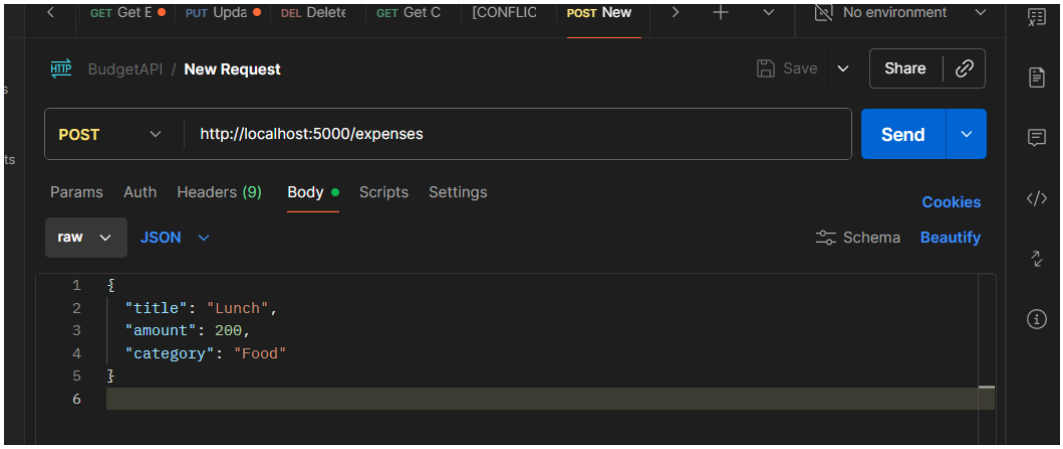
Headers:
`Content-Type`: `application/json`
`x-api-key`: `thisismetrying8`

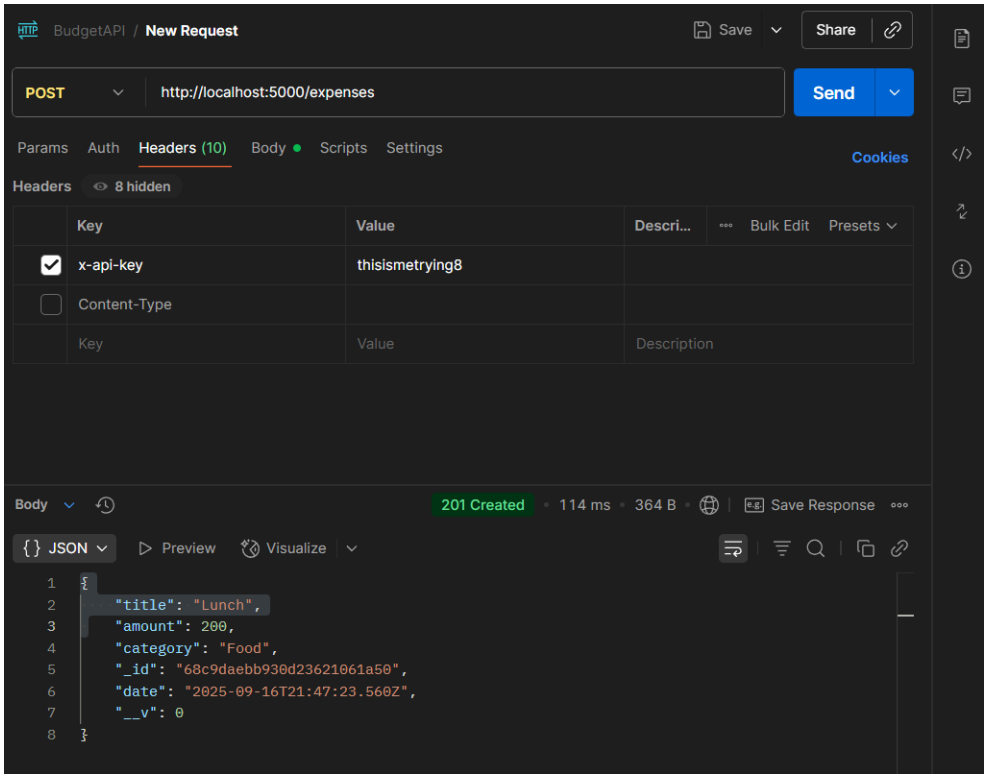
Body (raw, JSON):

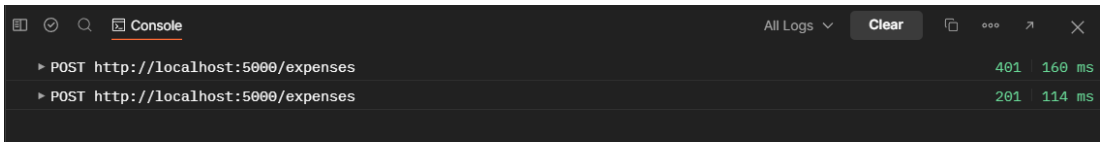
```
{
  "title": "Lunch",
  "amount": 200,
  "category": "Food"
}
```

Send → Response: `201 Created` + JSON with `_id`.







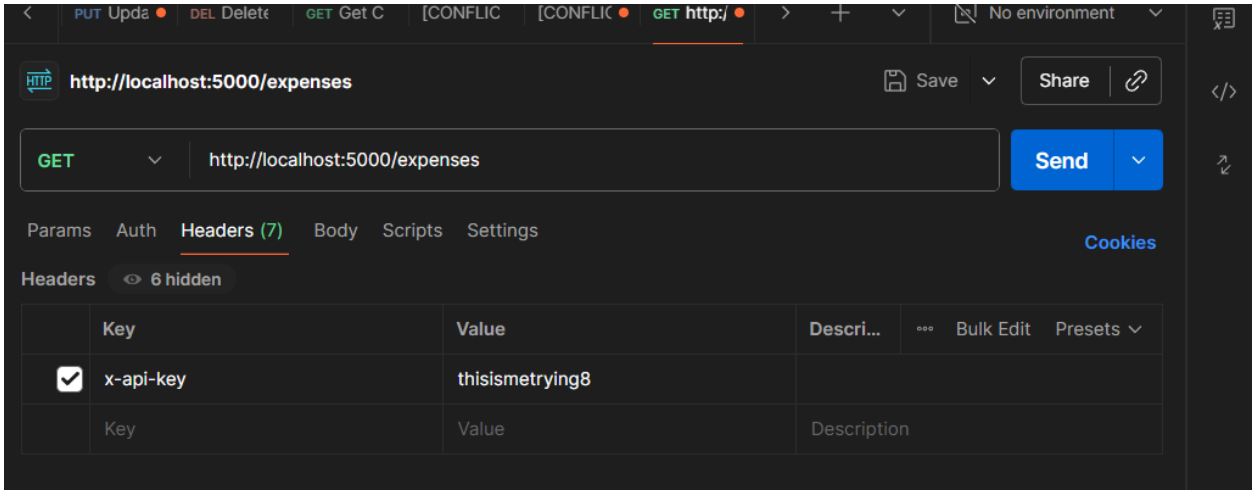


2. READ (GET) — Sabhi Expenses dekhna

Method: **GET**
URL:
`http://localhost:5000/expenses`

Headers:
`x-api-key`: `thisismetrying8`

Send → Response: Array of all expenses.




```
1  [
2      {
3          "_id": "68a46ec3b3c8e154516ab717",
4          "title": "Lunch",
5          "amount": 150,
6          "category": "Food",
7          "date": "2025-08-19T12:32:03.847Z",
8          "__v": 0
9      },
10     {
11         "_id": "68a46ec3b3c8e154516ab719",
12         "title": "Bus",
13         "amount": 50,
14         "category": "Transport",
15         "date": "2025-08-19T12:32:03.861Z",
16         "__v": 0
17     },
18     {
19         "_id": "68c98831cd19dcff21c1df62",
20         "title": "Lunch",
```

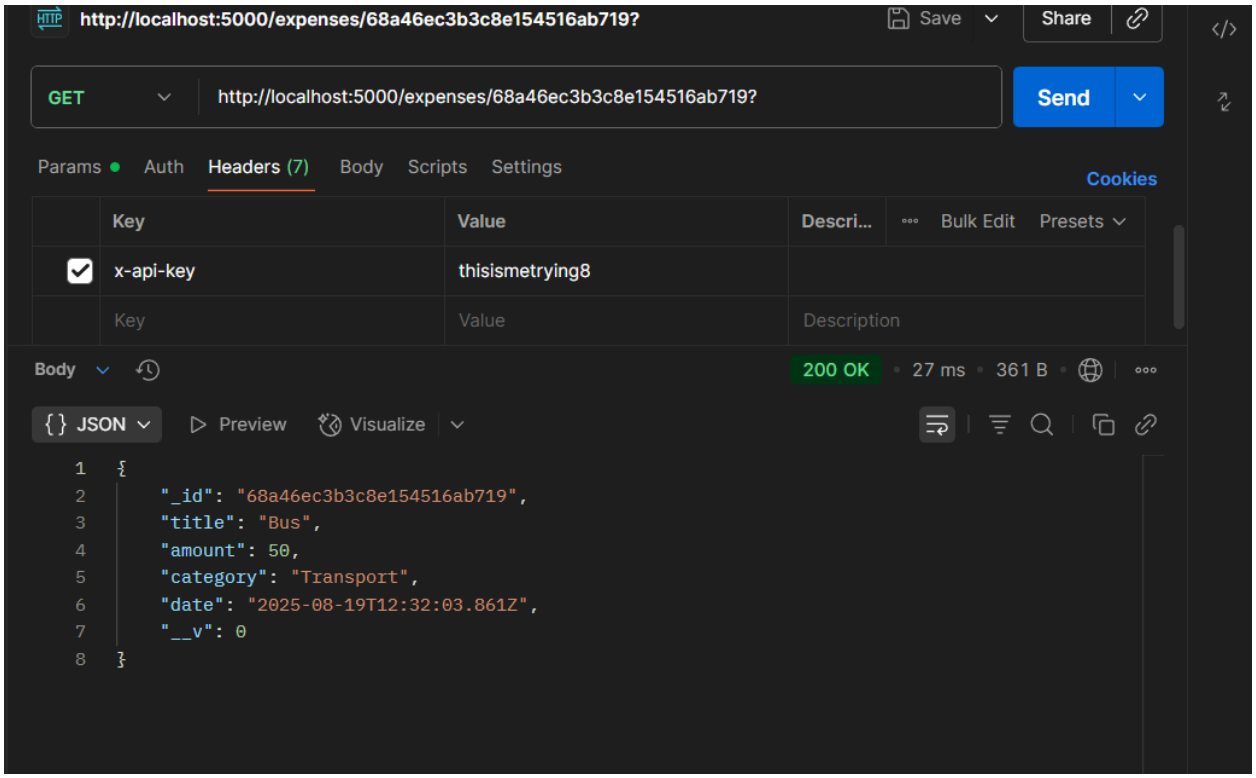

3. READ (GET by ID) — Single Expense dekhna

Method: **GET**

URL:
`http://localhost:5000/expenses/<id>`

👉 `<id>` ko aapko POST response se milta hai (example: `68c9daebb930d23621061a50`).

Headers:
`x-api-key`: `thisismetrying8`
Send → Response: Specific expense object.



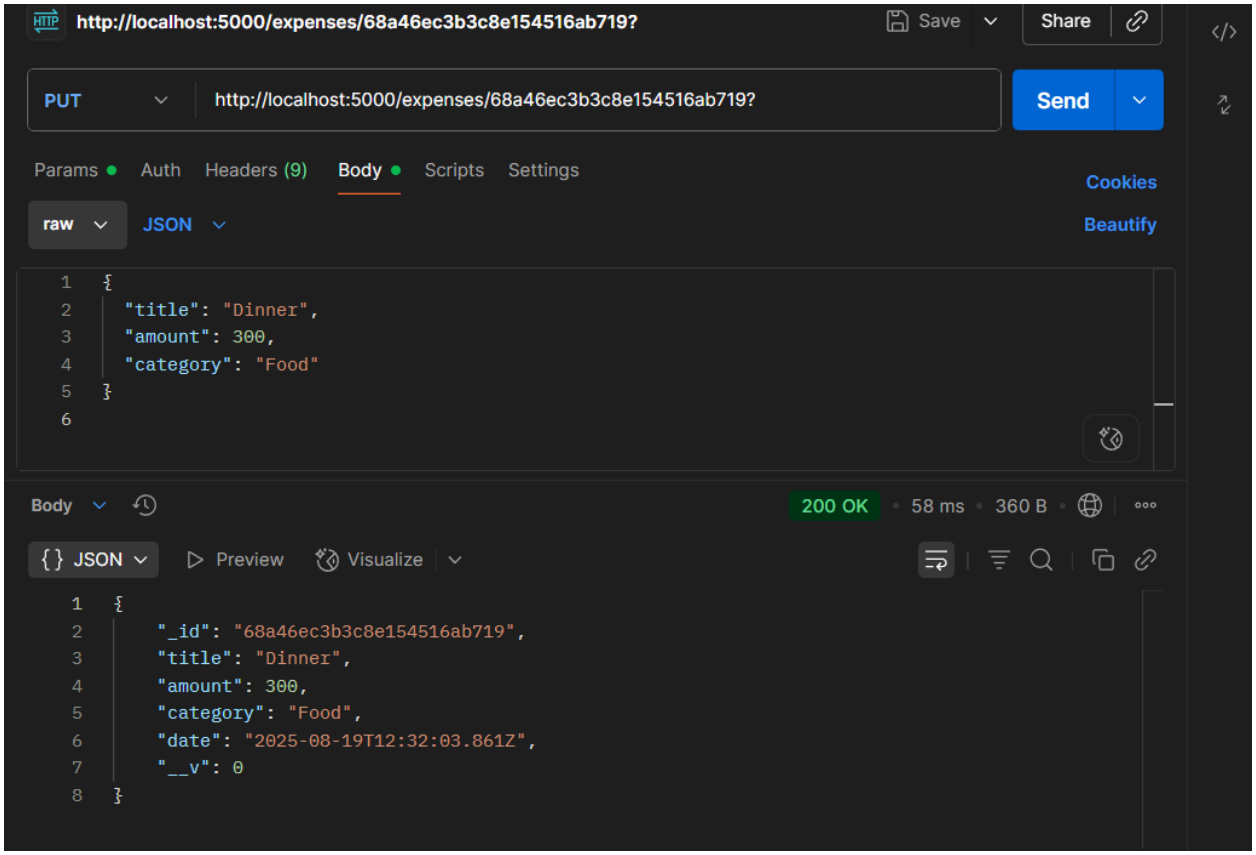
4. UPDATE (PUT) — Expense Update karna

Method: **PUT**
URL:
`http://localhost:5000/expenses/<id>`

👉 `<id>` = jo aap update karna chahte ho.

Headers:
`Content-Type`: `application/json`

`x-api-key: thisismetrying8`

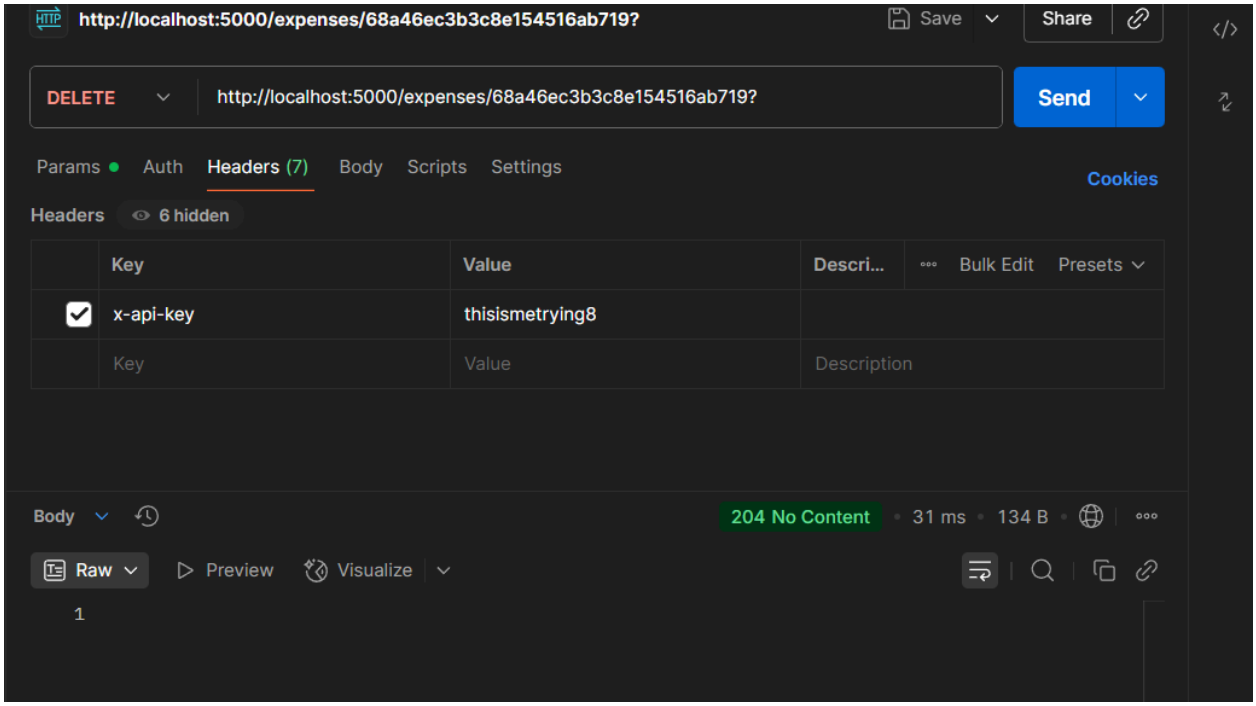


5. DELETE — Expense Delete karna

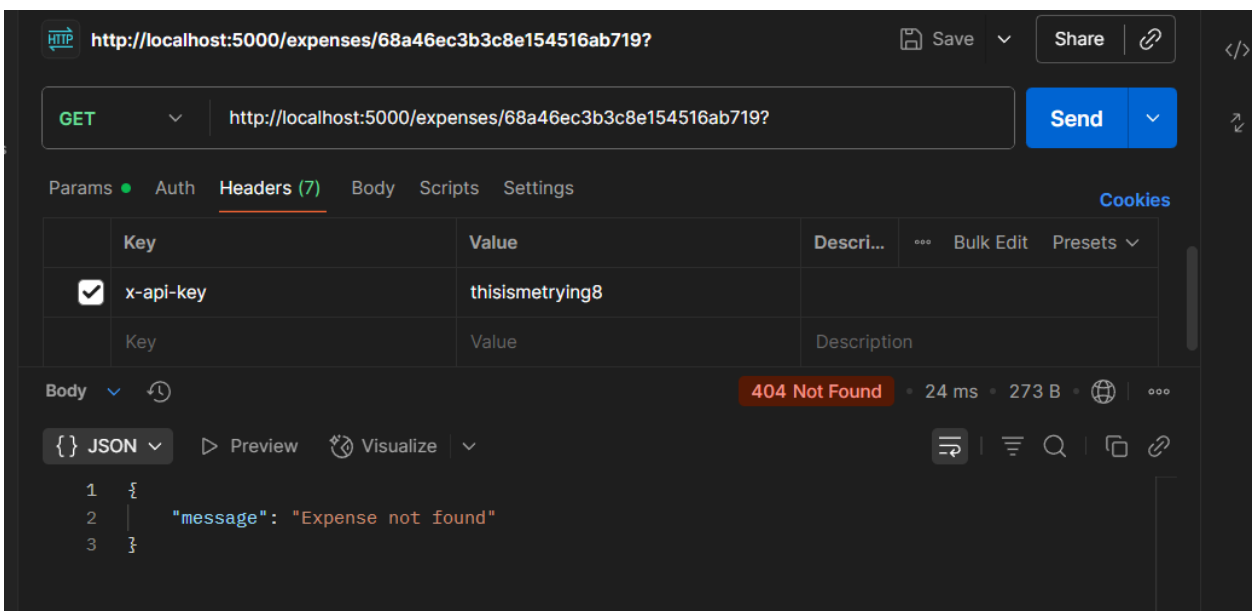
Method: **DELETE**
URL:
`http://localhost:5000/expenses/<id>`

👉 `<id>` = jo delete karna ho.

Headers:
`x-api-key: thisismetrying8`
Send → Response: `{ "message": "Expense deleted successfully" }` (ya similar).



To verify we can see it by using (READ (GET by ID)) - the expense of bus does not exist, hence it is deleted.



30% Extra – Supertest (Automated Testing)

Why testing is important

"Normally we test APIs manually in Postman, but here I used Supertest to automate it. The test case calls `POST /expenses` and checks if an expense is created successfully. When I run `npm test`, the case passes, proving the API works automatically without manual effort."

```
tests
    expense.test.js                U
```

```
Watch Usage: Press w to show more.
PS D:\SEM5\sem5_FSD\FSD_4THEXP_BudgetAPI\backend> npm test

> fsd_4thexp_budgetapi@1.0.0 test
> jest --watchAll
  console.log
    [dotenv@17.2.1][DEBUG] No encoding is specified. UTF-8 is used by default

      at _debug (node_modules/dotenv/lib/main.js:135:11)

  console.log
    [dotenv@17.2.1] injecting env (3) from ..\.env -- tip: 🔐 prevent building .env in docker: https
x.com/prebuild

      at _log (node_modules/dotenv/lib/main.js:139:11)
```

```
      at Object.log (server.js:10:9)

  console.log
    ✅ MongoDB Connected

      at log (server.js:21:23)

 PASS  tests/expense.test.js
  Expense API
    √ should create a new expense (370 ms)
    √ should get all expenses (48 ms)
    √ should get a single expense by id (48 ms)
    √ should update an expense (53 ms)
    √ should delete an expense (55 ms)
    √ should update an expense (53 ms)
    √ should delete an expense (55 ms)
    √ should delete an expense (55 ms)
    √ should get distinct categories (59 ms)
    √ should get distinct categories (59 ms)


Test Suites: 1 passed, 1 total
Test Suites: 1 passed, 1 total
Tests:       6 passed, 6 total
Tests:       6 passed, 6 total
Snapshots:   0 total
Time:        2.224 s, estimated 4 s
Ran all test suites.

Watch Usage
 › Press f to run only failed tests.
 › Press o to only run tests related to changed files.
 › Press p to filter by a filename regex pattern.
 › Press t to filter by a test name regex pattern.
 › Press q to quit watch mode.
 › Press Enter to trigger a test run.
```

Code :

```javascript
// tests/expense.test.js
import request from 'supertest';
import mongoose from 'mongoose';
import app from '../server.js'; // make sure server.js exports your Express app
import Expense from '../models/Expense.js';

const API_KEY = 'thisismetrying8';

beforeAll(async () => {
  const url = process.env.MONGO_URI || 'mongodb://127.0.0.1/budget-api-test';
  await mongoose.connect(url, { useNewUrlParser: true, useUnifiedTopology: true });
});

afterAll(async () => {
  await mongoose.connection.close();
});

beforeEach(async () => {
  await Expense.deleteMany({});
});

describe('Expense API', () => {
  it('should create a new expense', async () => {
    const res = await request(app)
      .post('/expenses')
      .set('x-api-key', API_KEY)
      .send({ title: 'Lunch', amount: 150, category: 'Food' });

    expect(res.statusCode).toBe(201);
    expect(res.body.title).toBe('Lunch');
    expect(res.body.amount).toBe(150);
    expect(res.body.category).toBe('Food');
  });
```

```javascript
it('should get all expenses', async () => {
  await Expense.create({ title: 'Lunch', amount: 150, category: 'Food' });

  const res = await request(app)
    .get('/expenses')
    .set('x-api-key', API_KEY);

  expect(res.statusCode).toBe(200);
  expect(res.body.length).toBe(1);
  expect(res.body[0].title).toBe('Lunch');
});

it('should get a single expense by id', async () => {
  const expense = await Expense.create({ title: 'Lunch', amount: 150, category: 'Food' });

  const res = await request(app)
    .get(`/expenses/${expense._id}`)
    .set('x-api-key', API_KEY);

  expect(res.statusCode).toBe(200);
  expect(res.body._id).toBe(expense._id.toString());
});

it('should update an expense', async () => {
  const expense = await Expense.create({ title: 'Lunch', amount: 150, category: 'Food' });

  const res = await request(app)
    .put(`/expenses/${expense._id}`)
    .set('x-api-key', API_KEY)
    .send({ title: 'Dinner', amount: 200, category: 'Food' });

  expect(res.statusCode).toBe(200);
  expect(res.body.title).toBe('Dinner');
  expect(res.body.amount).toBe(200);
});
```

```javascript
  it('should delete an expense', async () => {
    const expense = await Expense.create({ title: 'Lunch', amount: 150, category: 'Food' });

    const res = await request(app)
      .delete(`/expenses/${expense._id}`)
      .set('x-api-key', API_KEY);

    expect(res.statusCode).toBe(204);

    const check = await Expense.findById(expense._id);
    expect(check).toBeNull();
  });

  it('should get distinct categories', async () => {
    await Expense.create({ title: 'Lunch', amount: 150, category: 'Food' });
    await Expense.create({ title: 'Bus', amount: 50, category: 'Transport' });

    const res = await request(app)
      .get('/expenses/categories')
      .set('x-api-key', API_KEY);

    expect(res.statusCode).toBe(200);

    // Map response objects to their _id (category) values
    const categories = res.body.map(c => c._id);
    expect(categories).toEqual(expect.arrayContaining(['Food', 'Transport']));
  });
});
```

backend > babel.config.js > ...

```javascript
export default {
  presets: [
    [
      "@babel/preset-env",
      {
        targets: { node: "current" }
      }
    ]
  ]
};
```

```
PS D:\SEM5\sem5_FSD\FSD_4THEXP_BudgetAPI\backend> npm start

> fsd_4thexp_budgetapi@1.0.0 start
> node server.js

[dotenv@17.2.1][DEBUG] No encoding is specified. UTF-8 is used by default
[dotenv@17.2.1] injecting env (3) from ..\.env -- tip: write to custom object with { processEnv: myObject }
.env loaded: {
```

```
  USERNAME: 'chhab',
  USERPROFILE: 'C:\\Users\\chhab',
  VSCODE_GIT_ASKPASS_EXTRA_ARGS: '',
  VSCODE_GIT_ASKPASS_MAIN: 'c:\\Users\\chhab\\AppData\\Local\\Programs\\Microsoft VS Code\\res
  VSCODE_GIT_ASKPASS_NODE: 'C:\\Users\\chhab\\AppData\\Local\\Programs\\Microsoft VS Code\\Cod
  VSCODE_GIT_IPC_HANDLE: '\\\\.\\pipe\\vscode-git-e5bcffce74-sock',
  VSCODE_INJECTION: '1',
  windir: 'C:\\WINDOWS',
  ZES_ENABLE_SYSMAN: '1'
}
MONGO_URI: mongodb+srv://FSD_4thEXP_5thSEM_BudgetAPI:thisismetrying8@fsd-4thexp-5thsem-budg
Server running on port 5000
MongoDB Connected
```