EXPERIMENT

Aim

To validate RESTful APIs using Postman by creating and testing a Budget Tracker API. The experiment demonstrates API key–based authentication, CRUD operations (Create, Read, Update, Delete), and validation of responses through Postman tests.

Theory

A RESTful API (Representational State Transfer) allows communication between client and server using HTTP methods such as GET, POST, PUT, and DELETE.

API Validation ensures:

1. Correctness – Endpoints return correct data and status codes.
2. Authentication & Security – Unauthorized requests are blocked.
3. Schema Compliance – Response follows a defined JSON structure.
4. Error Handling – Invalid inputs return proper error messages.
5. Performance – Response times are within acceptable limits.

Postman is an API testing tool that allows developers to:

- Send HTTP requests with headers, query parameters, and body.
- Validate responses using scripts (JavaScript-based assertions).
- Automate testing with Collections and run them in Newman (CI/CD).

Using a Budget Tracker API, we test CRUD operations with an API Key to validate proper working and security.

Implementation Steps

1. Create Budget Tracker API (Node.js + Express)

```
// server.js
const express = require('express');
const bodyParser = require('body-parser');
const { v4: uuidv4 } = require('uuid');

const API_KEY = 'my-secret-key';
const app = express();
app.use(bodyParser.json());

const budgets = new Map();

// Middleware for API Key
app.use((req, res, next) => {
  const key = req.header('x-api-key');
  if (!key || key !== API_KEY) return res.status(401).json({ error: 'Unauthorized' });
  next();
});

// Create budget
app.post('/budgets', (req, res) => {
  const { name, amount } = req.body;
  if (!name || typeof amount !== 'number') return res.status(400).json({ error: 'Invalid data' });
  const budget = { id: uuidv4(), name, amount };
  budgets.set(budget.id, budget);
  res.status(201).json(budget);
});

// Read all
app.get('/budgets', (req, res) => res.json([...budgets.values()]));
```

Run server:

```
npm init -y
npm i express body-parser uuid
node server.js
```

---

2. Validate using curl (basic check)

```
curl -X POST http://localhost:3000/budgets \
  -H "Content-Type: application/json" \
  -H "x-api-key: my-secret-key" \
  -d '{"name":"Groceries","amount":200}'
```

---

3. Postman Setup

- Create Environment → Variables:
    - base_url = http://localhost:3000
    - api_key = my-secret-key
- Requests in Collection:
    - POST {{base_url}}/budgets (create budget)
    - GET {{base_url}}/budgets (list budgets)
    - GET {{base_url}}/budgets/{{budget_id}}
    - PUT {{base_url}}/budgets/{{budget_id}}
    - DELETE {{base_url}}/budgets/{{budget_id}}
- Add Header: x-api-key: {{api_key}}

---

4. Postman Tests (in Tests tab)

```
// Check status code
pm.test("Status code is 200 or 201", () => {
    pm.expect(pm.response.code).to.be.oneOf([200,201]);
});

// Check response is JSON
pm.test("Response is JSON", () => {
```

```
    pm.response.to.be.json;
});

// Validate fields
let json = pm.response.json();
pm.test("Budget has id, name, amount", () => {
    pm.expect(json).to.have.property("id");
    pm.expect(json).to.have.property("name");
    pm.expect(json).to.have.property("amount");
});
```

Code:

1) server.js

```
// server.js
require('dotenv').config();
const express = require('express');
const bodyParser = require('body-parser');
const { v4: uuidv4 } = require('uuid');
const Joi = require('joi');

const API_KEY = process.env.API_KEY || 'my-secret-key';
const PORT = process.env.PORT || 3000;

const app = express();
app.use(bodyParser.json());

// In-memory store (replace with DB in real projects)
const budgets = new Map();

// Middleware: API Key via header x-api-key or query param api_key
app.use((req, res, next) => {
  const key = req.header('x-api-key') || req.query.api_key;
  if (!key || key !== API_KEY) {
    return res.status(401).json({ error: 'Unauthorized' });
  }
  next();
});

// Validation schema using Joi
const budgetSchema = Joi.object({
  name: Joi.string().min(1).max(100).required(),
  amount: Joi.number().precision(2).min(0).required()
});

// Create budget
app.post('/budgets', (req, res) => {
  const { error, value } = budgetSchema.validate(req.body);
  if (error) return res.status(400).json({ error: error.details[0].message });

```

```
38    const budget = {
39      id,
40      name: value.name,
41      amount: value.amount,
42      createdAt: new Date().toISOString()
43    };
44    budgets.set(id, budget);
45    res.status(201).json(budget);
46  });
47
48  // Read all budgets
49  app.get('/budgets', (req, res) => {
50    res.json(Array.from(budgets.values()));
51  });
52
53  // Read one budget
54  app.get('/budgets/:id', (req, res) => {
55    const b = budgets.get(req.params.id);
56    if (!b) return res.status(404).json({ error: 'Not found' });
57    res.json(b);
58  });
59
60  // Update budget
61  app.put('/budgets/:id', (req, res) => {
62    const existing = budgets.get(req.params.id);
63    if (!existing) return res.status(404).json({ error: 'Not found' });
64
65    const { error, value } = budgetSchema.validate(req.body);
66    if (error) return res.status(400).json({ error: error.details[0].message });
67
68    const updated = {
69      ...existing,
70      name: value.name,
71      amount: value.amount,
72      updatedAt: new Date().toISOString()
```

```
  };
  budgets.set(req.params.id, updated);
  res.json(updated);
});

// Delete budget
app.delete('/budgets/:id', (req, res) => {
  if (!budgets.has(req.params.id)) return res.status(404).json({ error: 'Not found' });
  budgets.delete(req.params.id);
  res.status(204).send();
});

// Health
app.get('/health', (_req, res) => res.json({ status: 'ok' }));

app.listen(PORT, () => {
  console.log(`Budget Tracker API listening on http://localhost:${PORT}`);
});

```

2) package.json

```
{} package.json > ...
{
  "name": "budget-tracker-api",
  "version": "1.0.0",
  "description": "Minimal Budget Tracker API for Postman testing",
  "main": "server.js",
  ▷Debug
  "scripts": {
    "start": "node server.js",
    "dev": "nodemon server.js",
    "test:newman": "newman run BudgetTracker.postman_collection.json -e BudgetTracker.postman_envi
  },
  "dependencies": {
    "body-parser": "^1.20.2",
    "dotenv": "^16.3.1",
    "express": "^4.18.2",
    "joi": "^17.9.2",
    "uuid": "^9.0.0"
  },
  "devDependencies": {
    "nodemon": "^3.0.1"
  }
}
```

3) .env

```
$ .env.example
API_KEY=my-secret-key
PORT=3000
```

4) README.md

```
README.md > # Budget Tracker API > ## Setup
# Budget Tracker API

## Setup
1. Copy `.env.example` to `.env` and set `API_KEY`.
2. Install dependencies:
   ```bash
   npm install
```

```
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\\re
  VSCODE_GIT_ASKPASS_NODE: 'C:\\Users\\chhab\\AppData\\Local\\Programs\\Microsoft VS Code\\Co
  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-bud
Server running on port 5000
MongoDB Connected
```
```

Implementation Steps with Postman

1. Create (POST)

- URL: http://localhost:3000/api/transactions
- Method: POST

Headers:

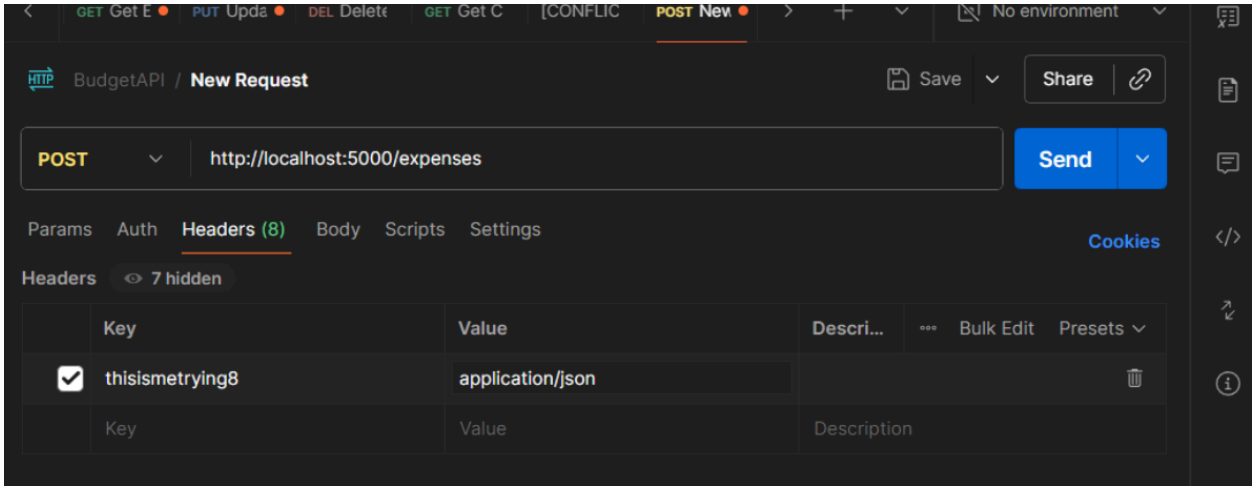
Content-Type: application/json
x-api-key: my-secret-key

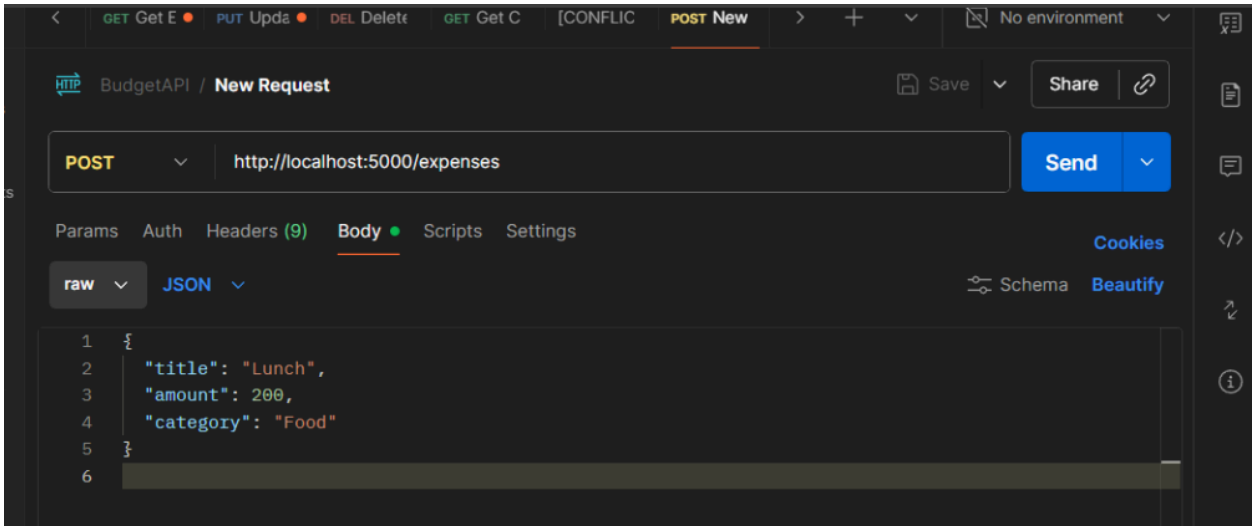
Body (raw JSON):

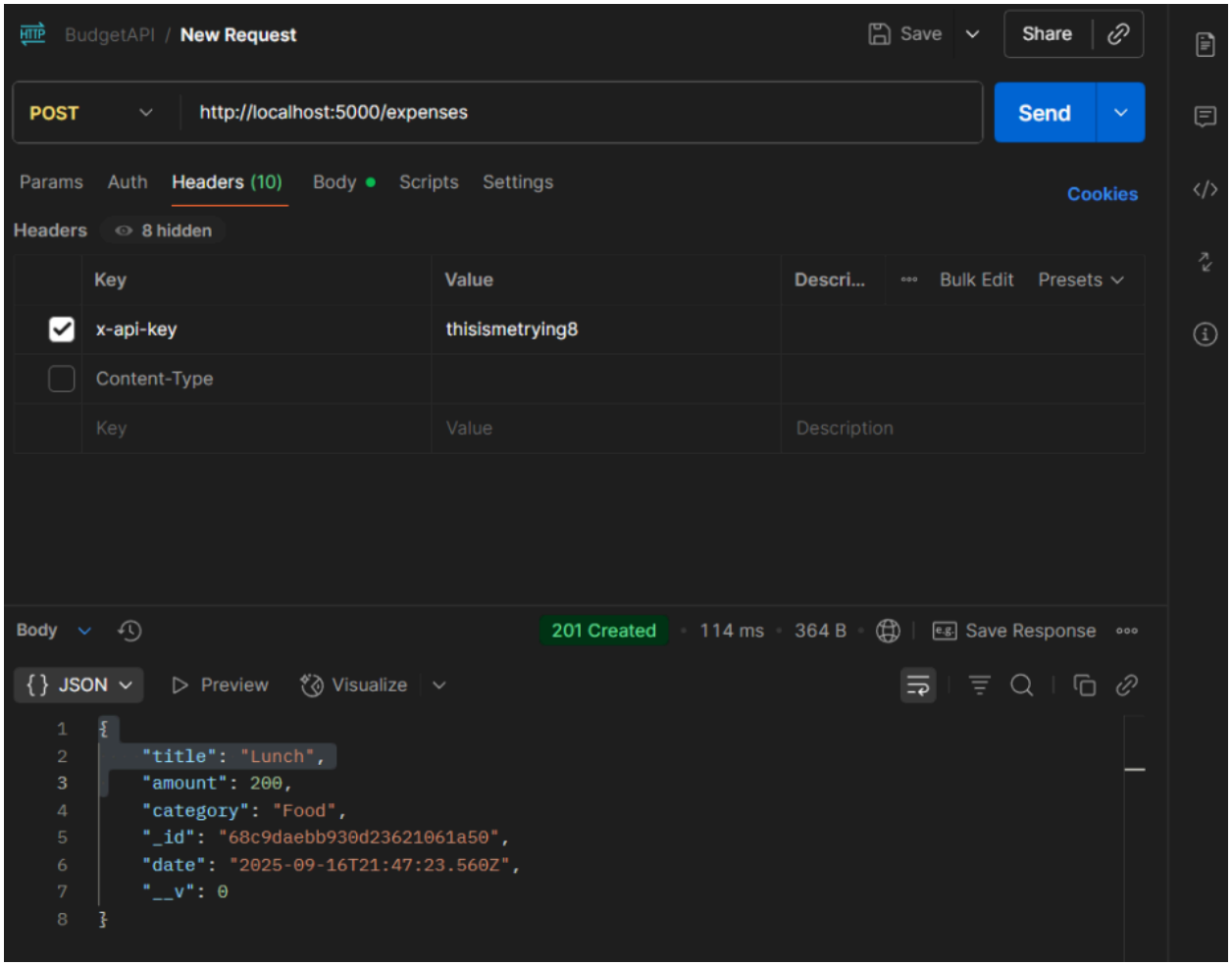
```
{
 "title": "Groceries",
 "amount": 500,
 "type": "expense"
}
```

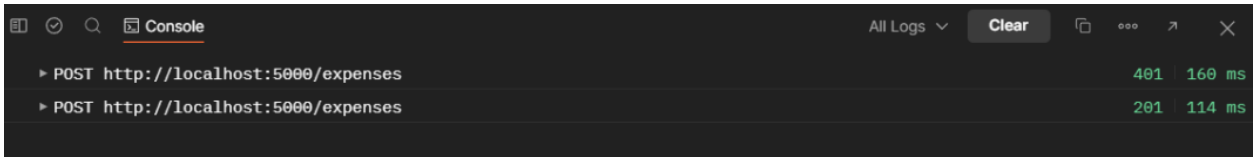
Expected Response:

```
{
 "id": "abc123",
 "title": "Groceries",
 "amount": 500,
 "type": "expense"
}
```


GET Get E
PUT Upda
DEL Delete
GET Get C
[CONFLIC
POST New
No environment
BudgetAPI / New Request
Save
Share
POST
http://localhost:5000/expenses
Send
Params
Auth
Headers (8)
Body
Scripts
Settings
Cookies
Headers
7 hidden
Key
Value
Descri...
Bulk Edit
Presets
thisismetrying8
application/json
Key
Value
Description


GET Get E
PUT Upda
DEL Delete
GET Get C
[CONFLIC
POST New
No environment
BudgetAPI / New Request
Save
Share
POST
http://localhost:5000/expenses
Send
Params
Auth
Headers (9)
Body
Scripts
Settings
Cookies
raw
JSON
Schema
Beautify
{
"title": "Lunch",
"amount": 200,
"category": "Food"
}


BudgetAPI / New Request
Save
Share
POST
http://localhost:5000/expenses
Send
Params
Auth
Headers (10)
Body
Scripts
Settings
Cookies
Headers
8 hidden
Key
Value
Descri...
Bulk Edit
Presets
x-api-key
thisismetrying8
Content-Type
Key
Value
Description
Body
201 Created
114 ms
364 B
Save Response
JSON
Preview
Visualize
{
"title": "Lunch",
"amount": 200,
"category": "Food",
"_id": "68c9daebb930d23621061a50",
"date": "2025-09-16T21:47:23.560Z",
"__v": 0
}


Console
All Logs
Clear
POST http://localhost:5000/expenses
401
160 ms
POST http://localhost:5000/expenses
201
114 ms

---

2. Read (GET)

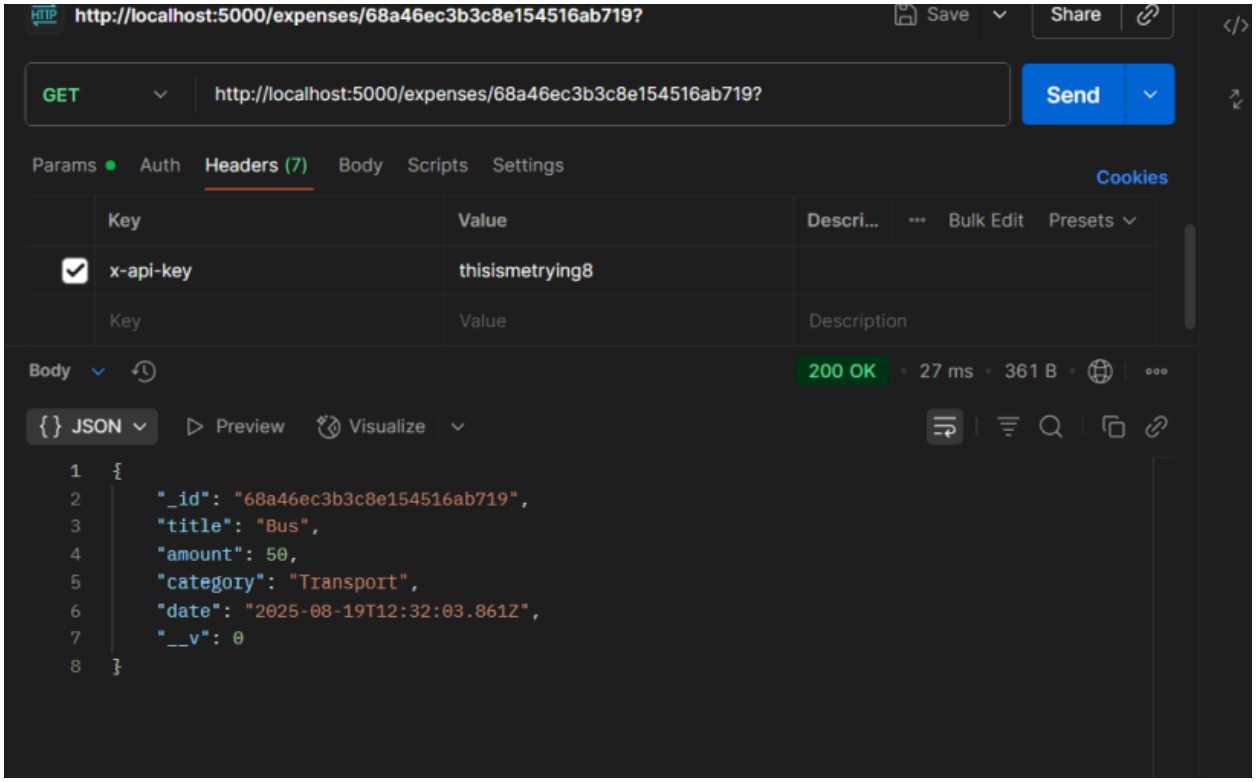
- URL: http://localhost:3000/api/transactions
- Method: GET

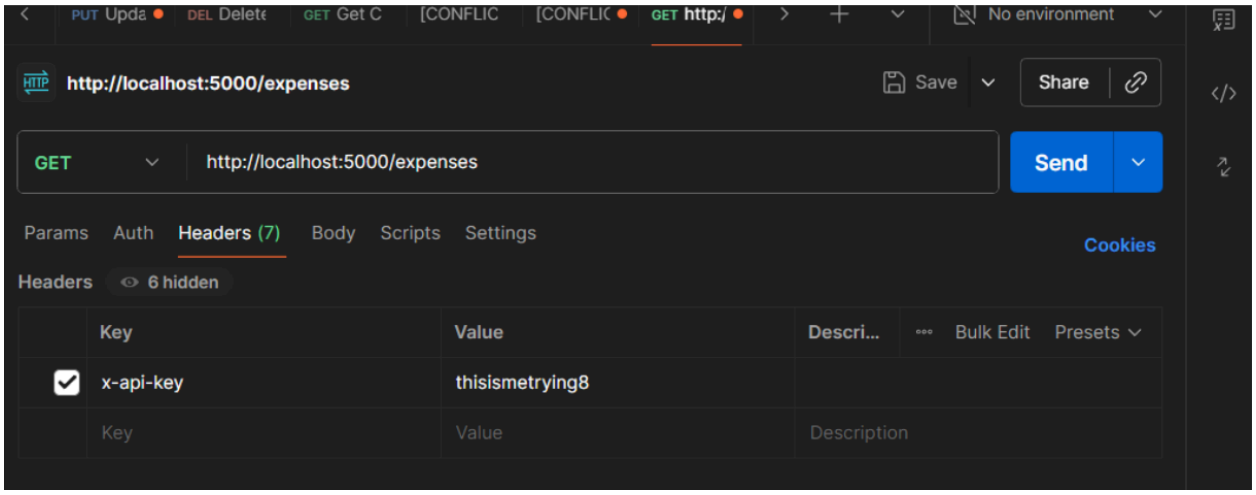
Headers:

x-api-key: my-secret-key

Expected Response:


```
[
  {
    "id": "abc123",
    "title": "Groceries",
    "amount": 500,
    "type": "expense"
  },
  {
    "id": "xyz789",
    "title": "Salary",
    "amount": 10000,
    "type": "income"
  }
]
```


http://localhost:5000/expenses/68a46ec3b3c8e154516ab719?
Save
Share
GET
http://localhost:5000/expenses/68a46ec3b3c8e154516ab719?
Send
Params
Auth
Headers (7)
Body
Scripts
Settings
Cookies
Key
Value
Descri...
Bulk Edit
Presets
x-api-key
thisismetrying8
Key
Value
Description
Body
200 OK
27 ms
361 B
{} JSON
Preview
Visualize
{
"_id": "68a46ec3b3c8e154516ab719",
"title": "Bus",
"amount": 50,
"category": "Transport",
"date": "2025-08-19T12:32:03.861Z",
"__v": 0
}




```
[
    {
        "_id": "68a46ec3b3c8e154516ab717",
        "title": "Lunch",
        "amount": 150,
        "category": "Food",
        "date": "2025-08-19T12:32:03.847Z",
        "__v": 0
    },
    {
        "_id": "68a46ec3b3c8e154516ab719",
        "title": "Bus",
        "amount": 50,
        "category": "Transport",
        "date": "2025-08-19T12:32:03.861Z",
        "__v": 0
    },
    {
        "_id": "68c98831cd19dcff21c1df62",
        "title": "Lunch",
```

3. Update (PUT / PATCH)

- URL: http://localhost:3000/api/transactions/abc123
- Method: PUT

Headers:

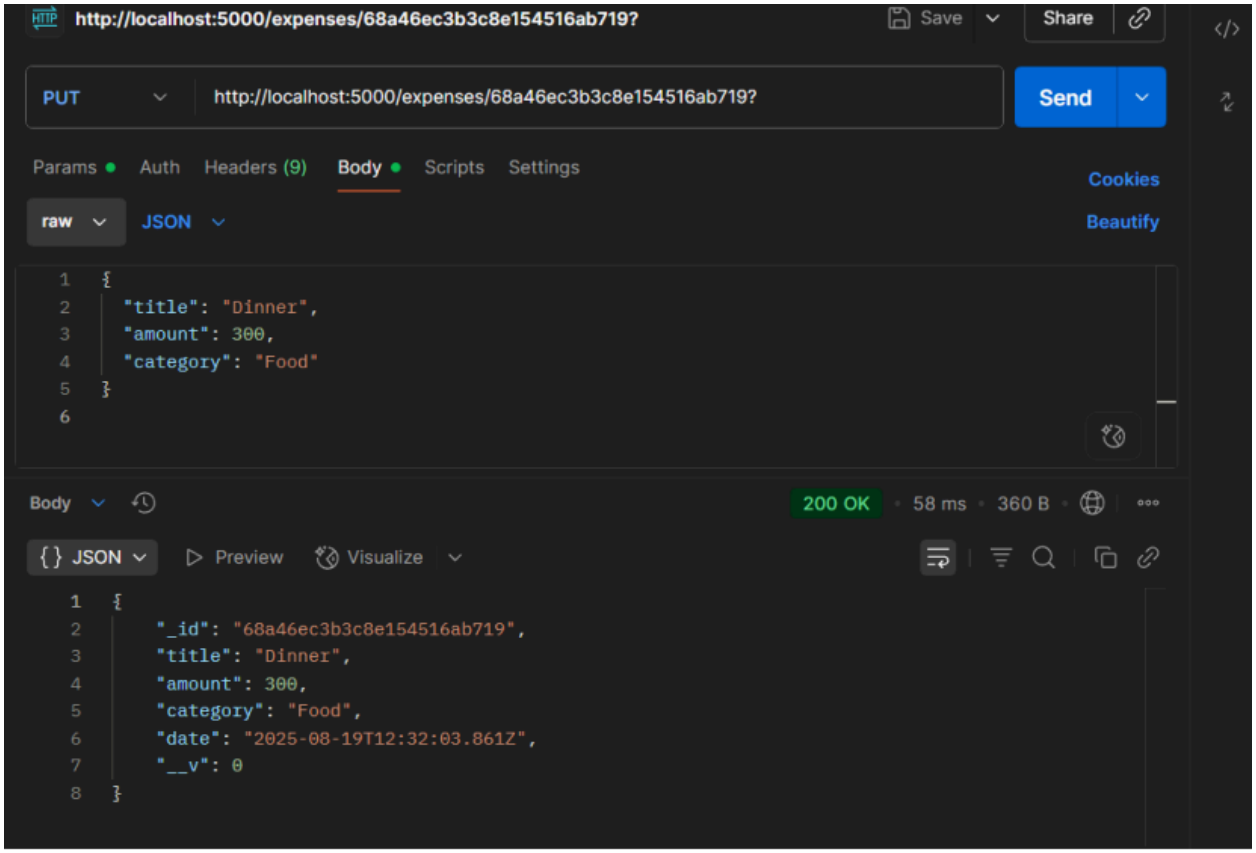
Content-Type: application/json
x-api-key: my-secret-key

Body (raw JSON):

```
{
 "title": "Groceries and Fruits",
 "amount": 600,
 "type": "expense"
}
```

Expected Response:

```
{
 "id": "abc123",
 "title": "Groceries and Fruits",
 "amount": 600,
 "type": "expense"
}
```



---

4. Delete (DELETE)

- URL: http://localhost:3000/api/transactions/abc123
- Method: DELETE

Headers:

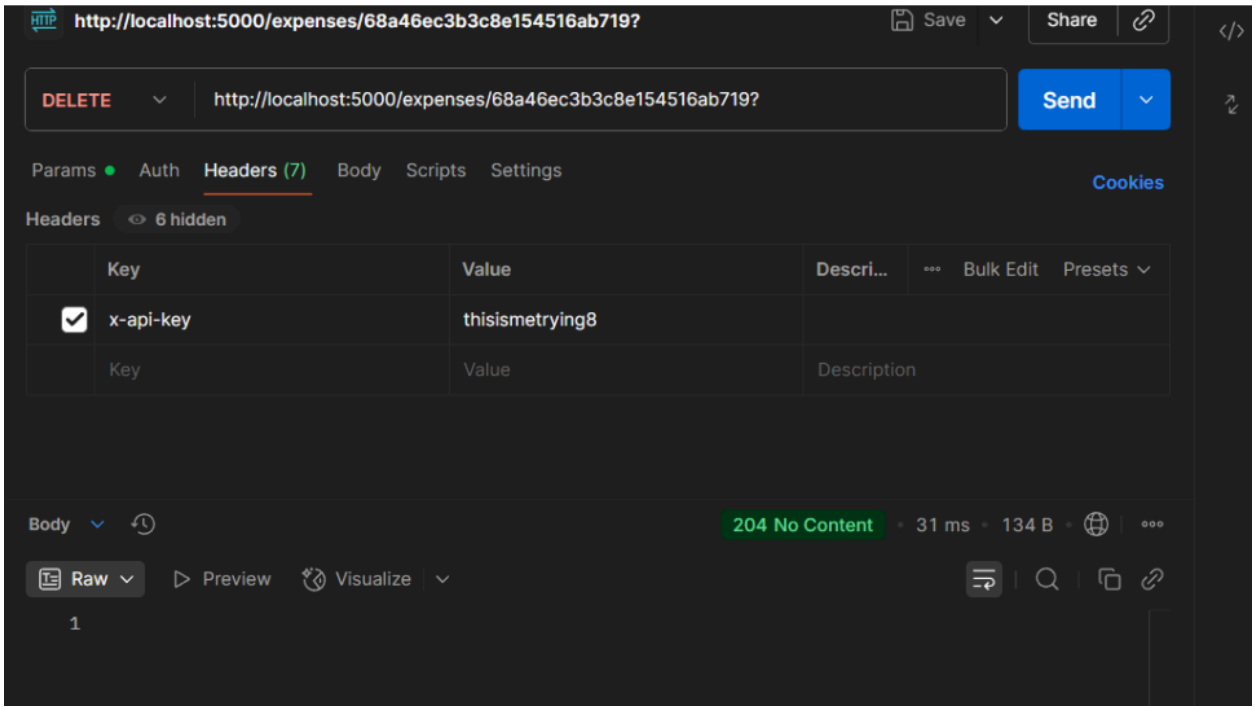
x-api-key: my-secret-key

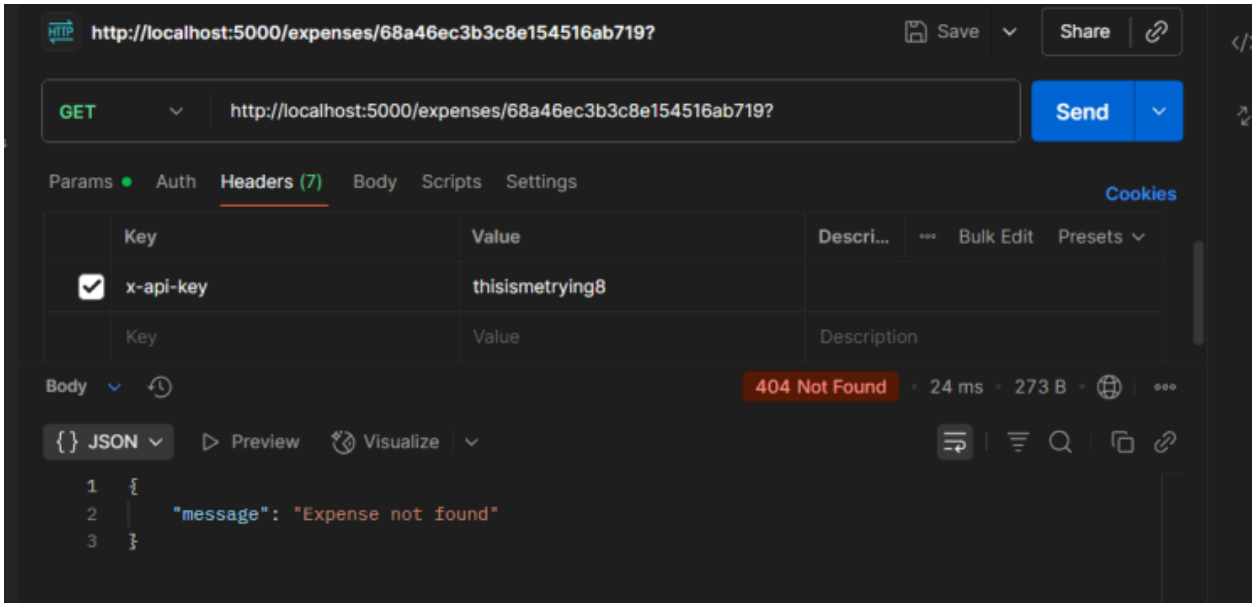
-

Expected Response:

{
"message": "Transaction deleted successfully"

- }





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."

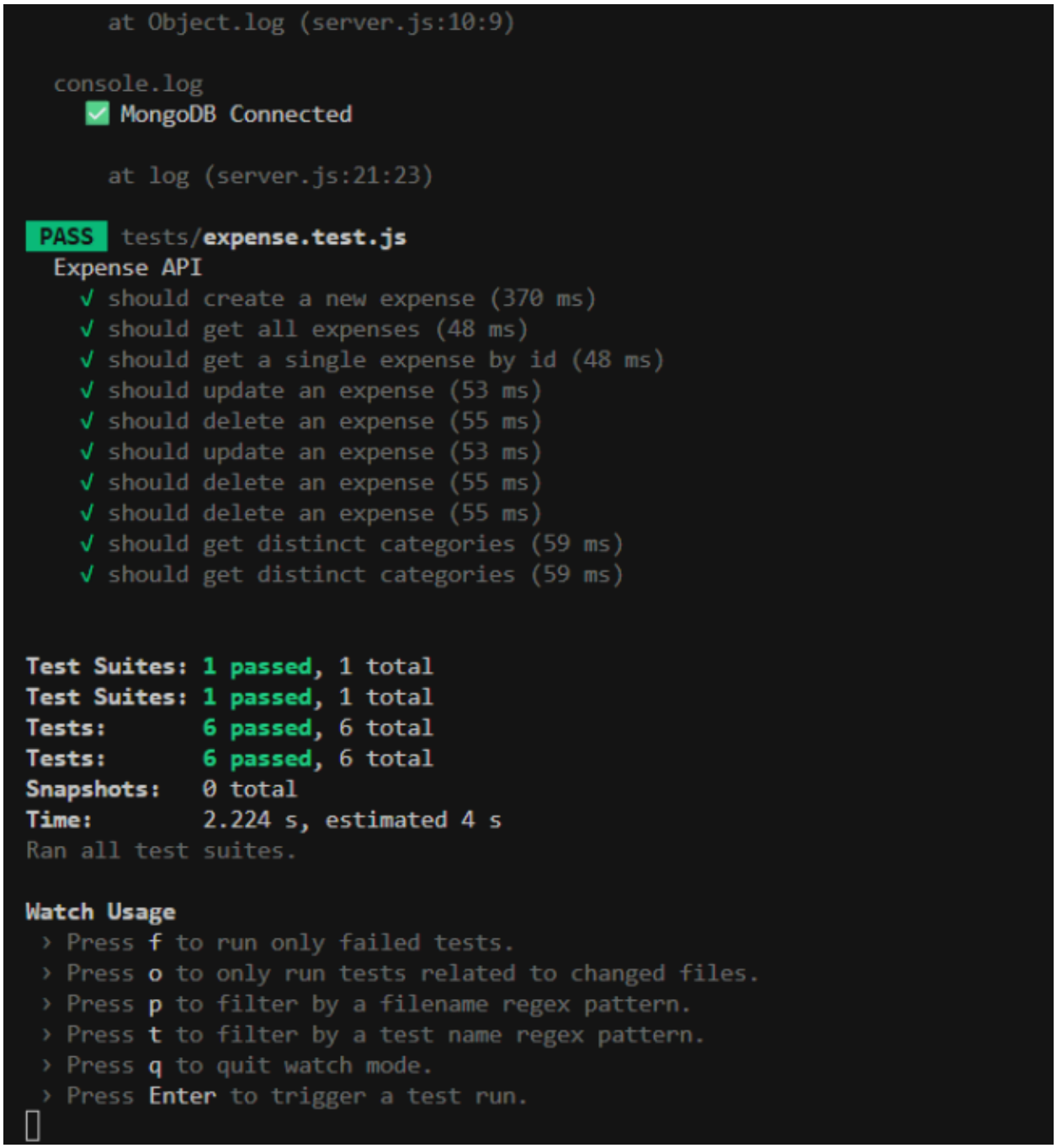

```
Watch Usage: Press w to show more.
PS D:\SEM5\sem5_FSD\FSD_4THEXP_BudgetAPI\backend> npm test

> 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:

```
// 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');
  });
```

```
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);
});
```

```
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' });

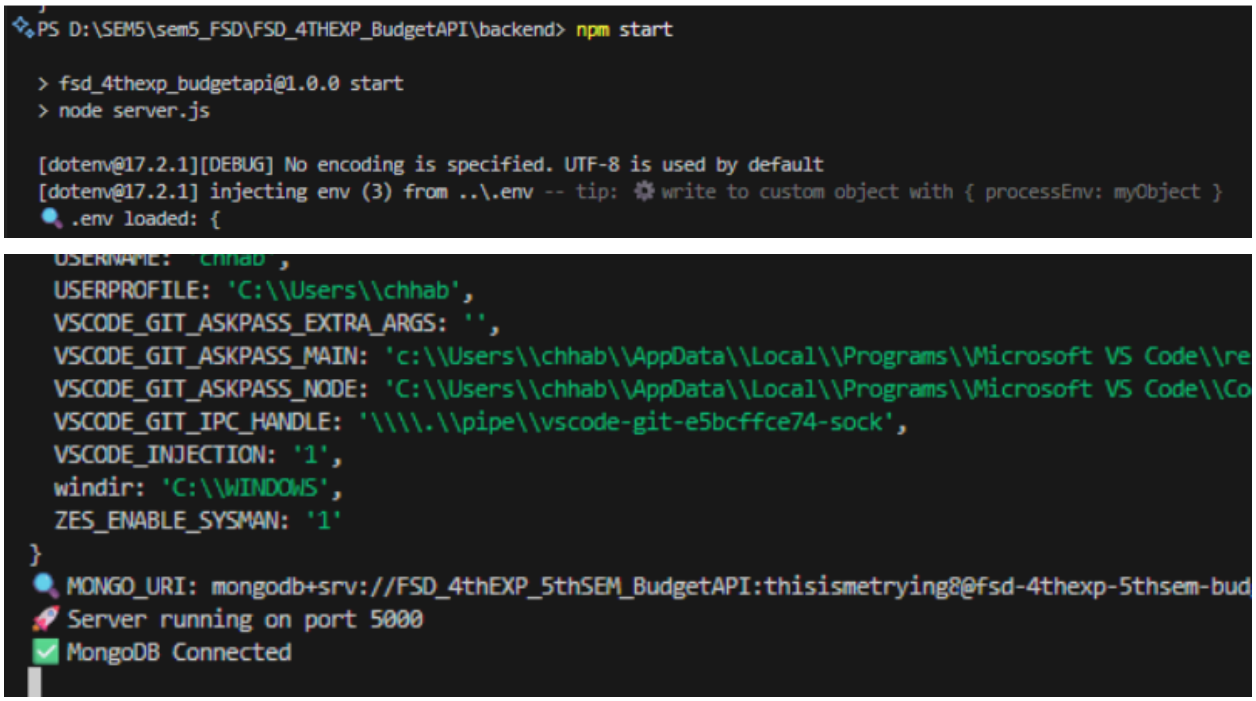
  const res = await request(app)
    .get('/expenses/categories')
    .set('x-api-key', API_KEY);

  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 > ...

```
1  export default {
2    presets: [
3      [
4        "@babel/preset-env",
5        {
6          targets: { node: "current" }
7        }
8      ]
9    ]
10 };
11
```



Conclusion

- Successfully created and validated a RESTful Budget Tracker API using Postman.
- Demonstrated CRUD operations secured with an API Key.
- Verified correctness, authentication, schema validation, and error handling with Postman test scripts.
- This experiment shows how Postman helps developers automate and ensure reliability, security, and performance of APIs.