**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;

});

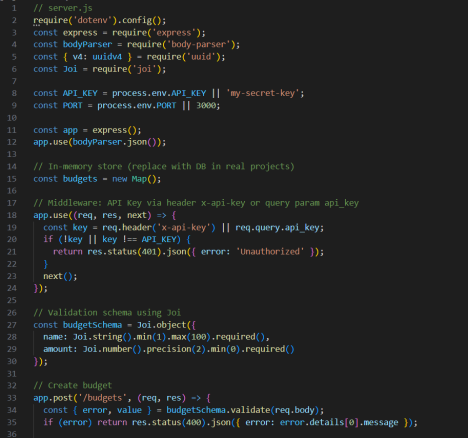
// Validate fields

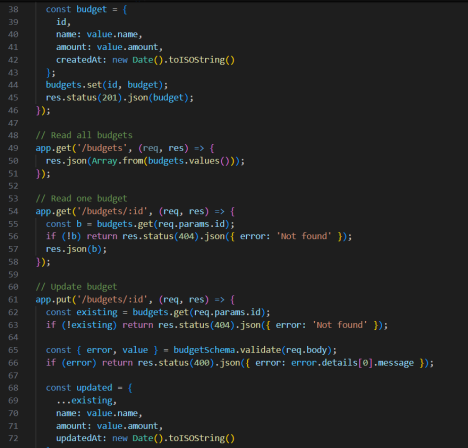
let json = pm.response.json();

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:

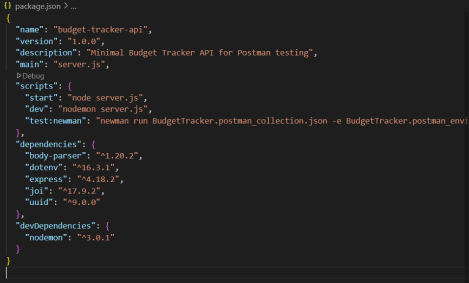
1) server.js





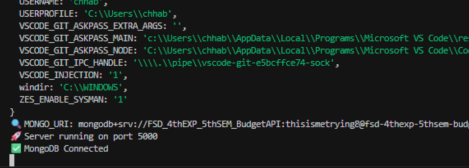
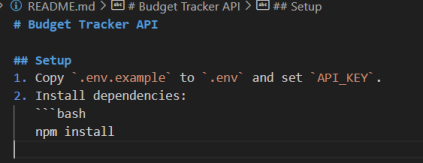


2) package.json

3) .env



4) README.md



**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:

{

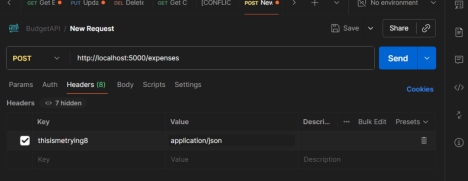
"id": "abc123",

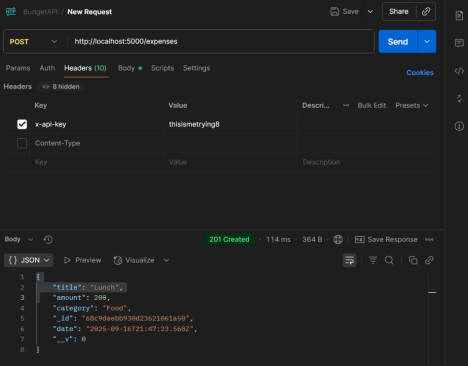
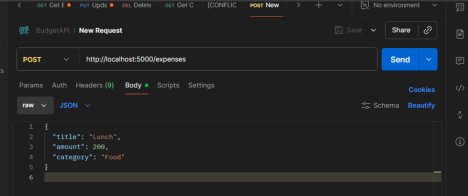
"title": "Groceries",

"amount": 500,

"type": "expense"

}





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",

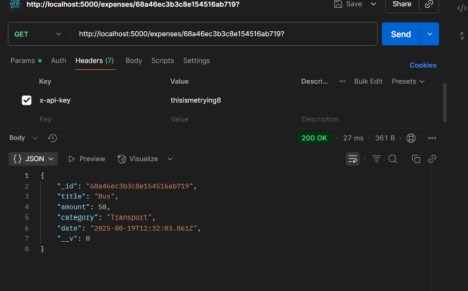
"title": "Salary",

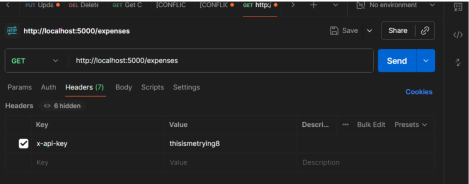
"amount": 10000,

"type": "income"

}

]





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."



Code:









**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.