Build a Simple API

1. Choose any backend stack you know (Rails, Express.js, Django, Phoenix, etc.).
2. Create a new API project.
3. Implement endpoints:

GET /health → returns JSON:

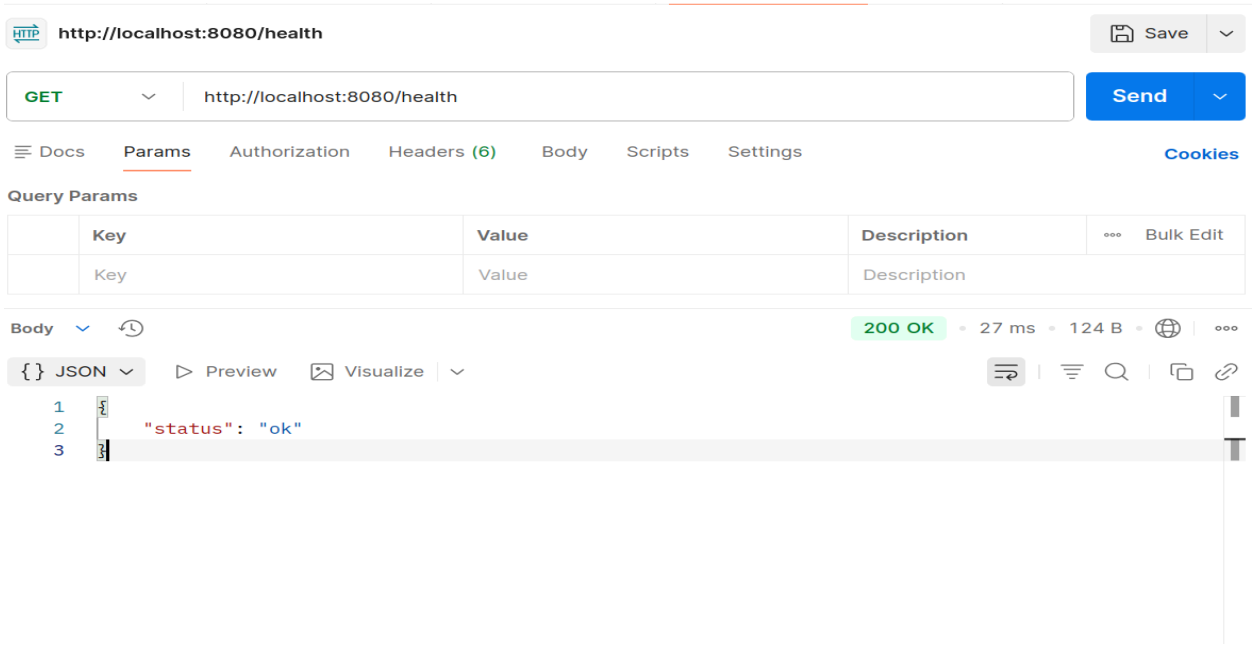
{ "status": "ok" }

GET /users → returns a list of users (can be static/in-memory).

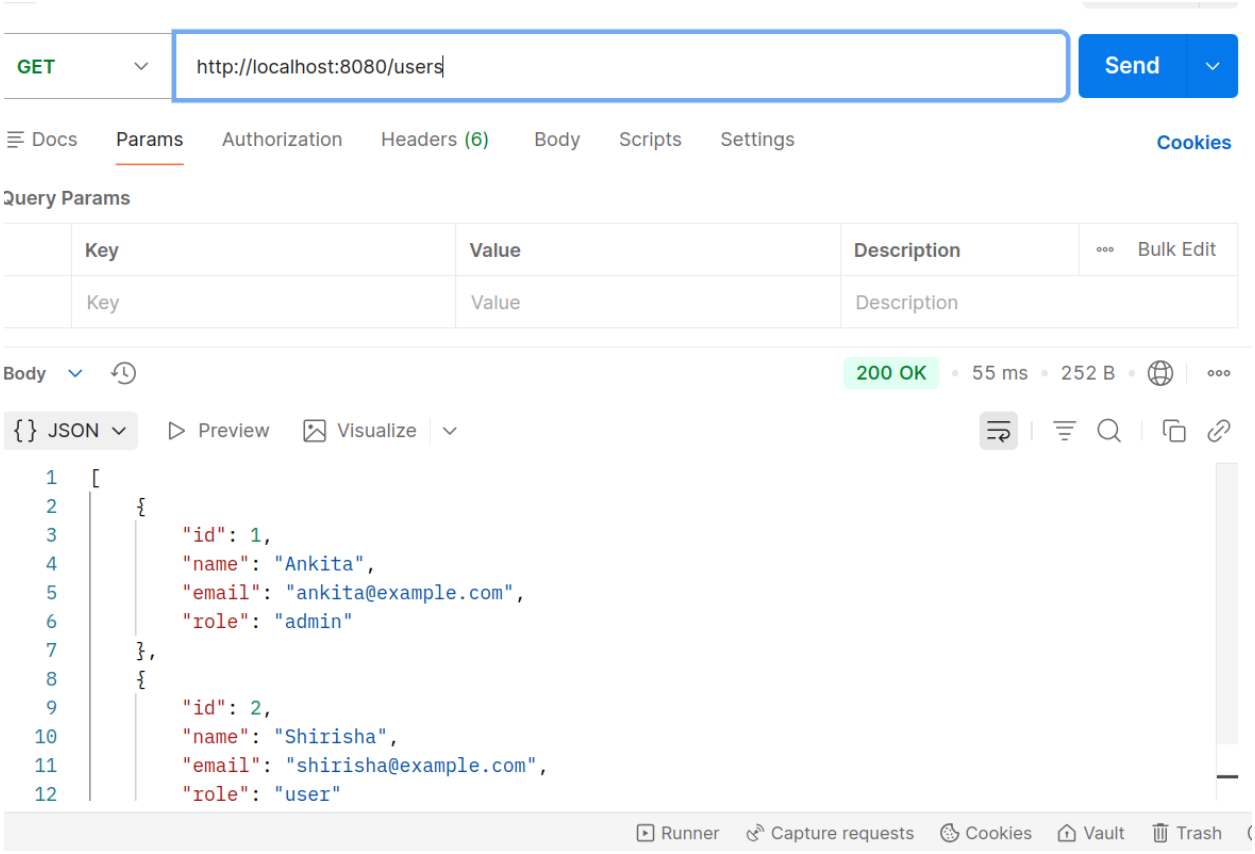
POST /users → accepts JSON body to create a user with at least name & email.

4. Support:
    - Route params (if implementing GET /users/:id)
    - Query params (e.g., GET /users?role=admin)
5. Ensure correct HTTP Status codes (200, 201, 400).
6. Submit: routes file, controller code, sample Postman collection export.

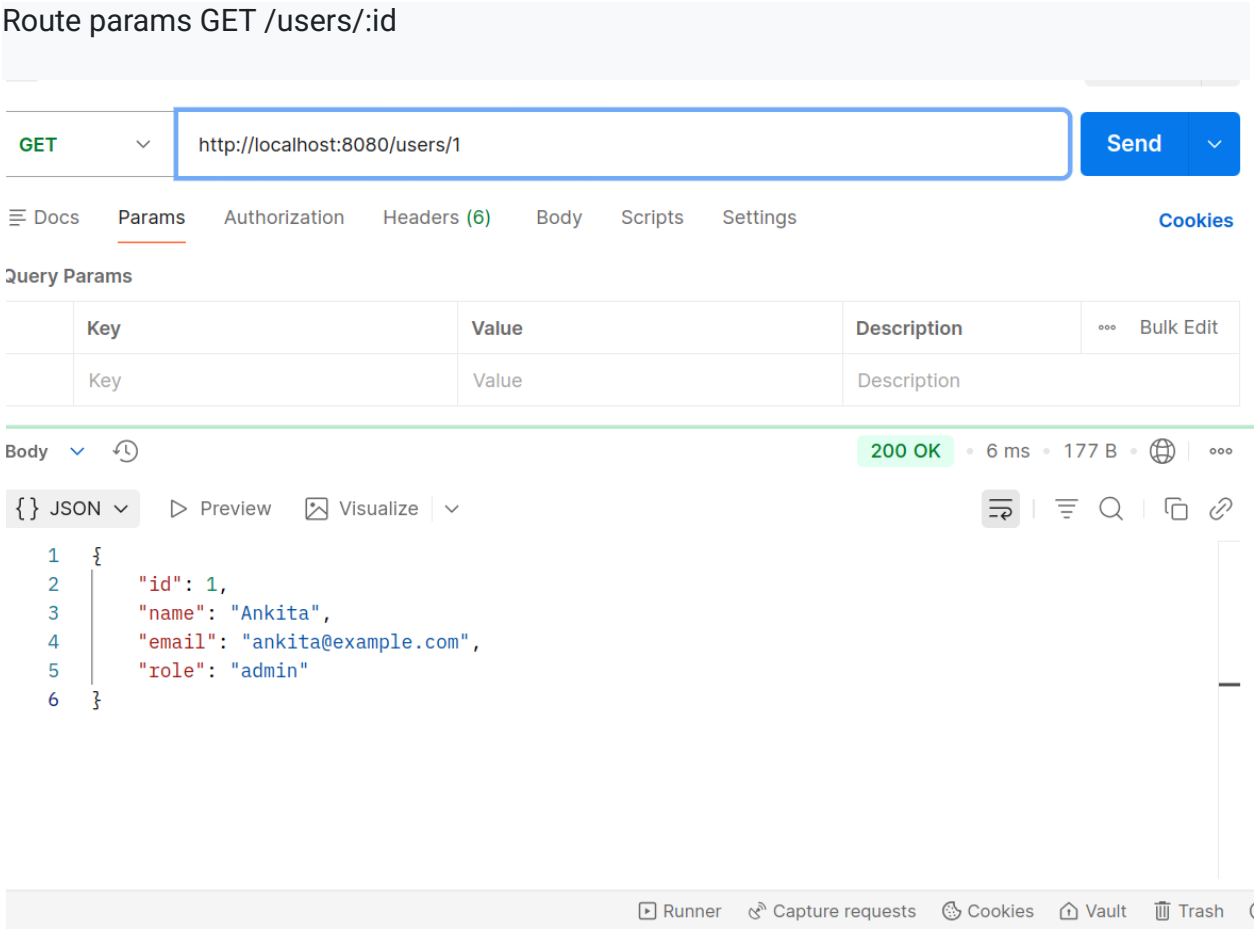
GET /health → returns JSON:



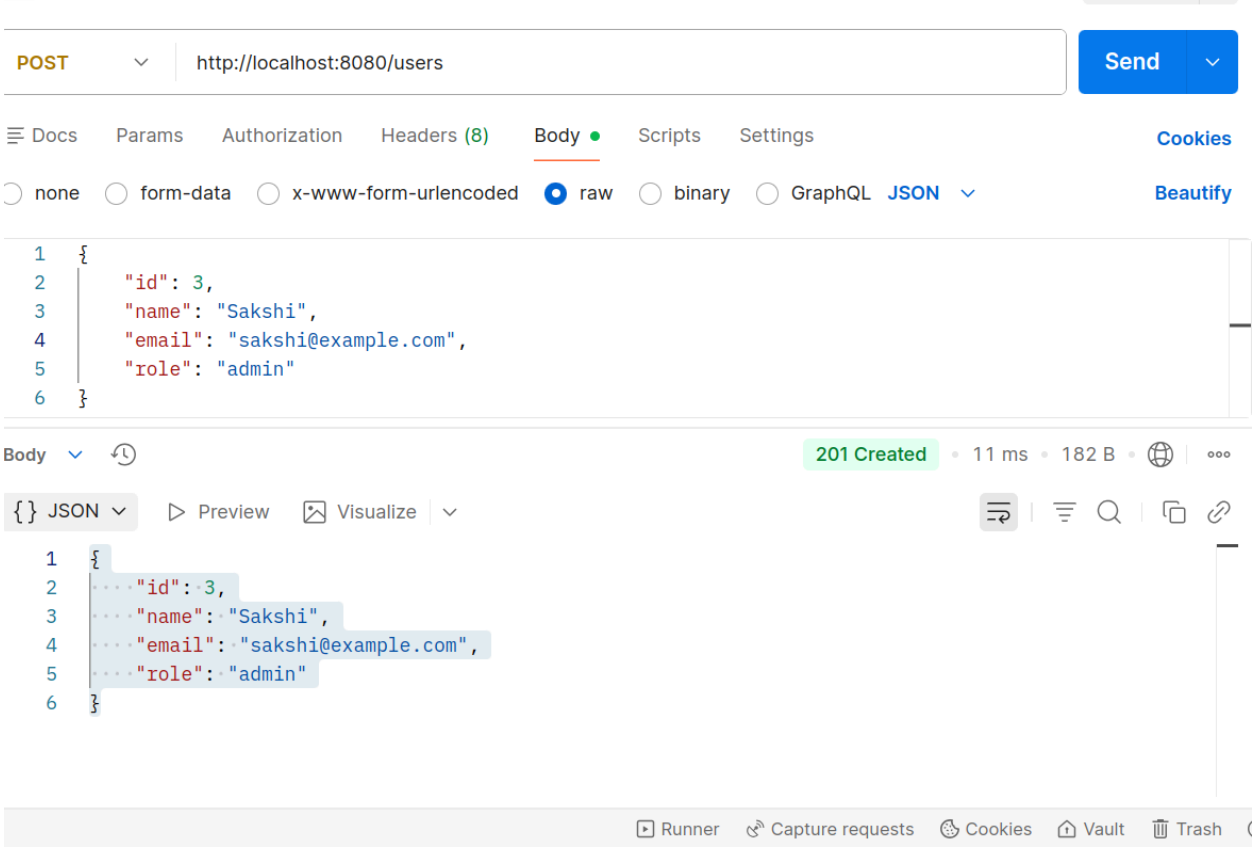
GET /users → returns a list of users (can be static/in-memory



Route params GET /users/:id



POST /users → accepts JSON body to create a user with at least name & email.



What is a REST API?

REST API means
Representational State Transfer – Application Programming Interface

In simple words:

> A REST API is a way for two programs to talk to each other over the internet using HTTP.

Your **frontend** (browser, mobile app, Postman) sends a request
Your **backend** (Go server) sends a response

They talk using:

- URLs
- HTTP methods
- JSON data

🧩 Example of REST API

Imagine an **Expense Tracker** app.

You want to:

- Add expense
- View expenses
- Update expense
- Delete expense

These become API endpoints:

Action	HTTP Method	URL	What it does
Add expense	POST	`/expenses`	Create new expense
Get all expenses	GET	`/expenses`	Get list
Get one expense	GET	`/expenses/5`	Get expense with id 5
Update expense	PUT	`/expenses/5`	Modify expense
Delete expense	DELETE	`/expenses/5`	Remove expense

📤 Example Request (POST)

Client sends:

POST /expenses

JSON:

```
{
  "amount": 500,
  "category": "Food",
  "description": "Lunch"
}
```

📥 Example Response

Server replies:

```
{
  "id": 10,
  "amount": 500,
  "category": "Food",
  "description": "Lunch",
  "status": "saved"
}
```

This is how REST API works.

📌 What is MVC?

MVC means:

> **Model – View – Controller**

It is a way to **organize backend code** so it is clean, readable, and scalable.

Instead of writing everything in one file, we divide responsibilities.

🧱 MVC Structure

```
Project
│
├── models
│     └── expense.go
```

```
│
├── controllers
│   └── expense_controller.go
│
├── views
│   └── (JSON output in APIs)
│
└── main.go
```

🧩 Model (M)

Model represents **data and database structure**

Example: `models/expense.go`

```
type Expense struct {
    ID int
    Amount int
    Category string
    Description string
}
```

It defines what an expense looks like.

🧩 Controller (C)

Controller contains **business logic**

It:

- Receives request
- Calls model
- Sends response

Example:

```
func GetExpenses(w http.ResponseWriter, r *http.Request) {
    expenses := models.GetAllExpenses()
    json.NewEncoder(w).Encode(expenses)
```

}

🧩 View (V)

In REST APIs, the **view is JSON output**.

Example JSON sent to client:

```
[
  {
    "id": 1,
    "amount": 500,
    "category": "Food"
  }
]
```

No HTML pages – just JSON.

🔁 How MVC + REST API Work Together

User calls:

GET /expenses

Flow:

Client (Postman / App)
↓
Controller (handles route)
↓
Model (gets data from DB)
↓
Controller
↓
View (JSON response)
↓
Client

🧠 Why MVC + REST is powerful

Problem	Without MVC	With MVC
Code mess	Everything mixed	Clean structure
Debugging	Hard	Easy
Team work	Confusing	Each dev works on a layer
Scaling	Difficult	Easy
