FPT Education

FPT POLYTECHNIC

cdio™

Conceive Design Implement Operate

THỰC HỌC – THỰC NGHIỆP

NodeJs & Resful Web Service

RESTful Web Service

http://www.poly.edu.vn

Mục tiêu

- Biết được tại sao phải dùng REST API
- Hiểu cấu trúc Rest API
- Tạo được REST API
- Test Rest API

NỘI DUNG

- REST API là gì? Tại sao phải dung Rest API
- Routing và phương thức HTTP
- Tạo REST API
- Test Rest API với Postman

Phần 1: tổng quan và Tạo Restful api

RESTFUL API LÀ GÌ

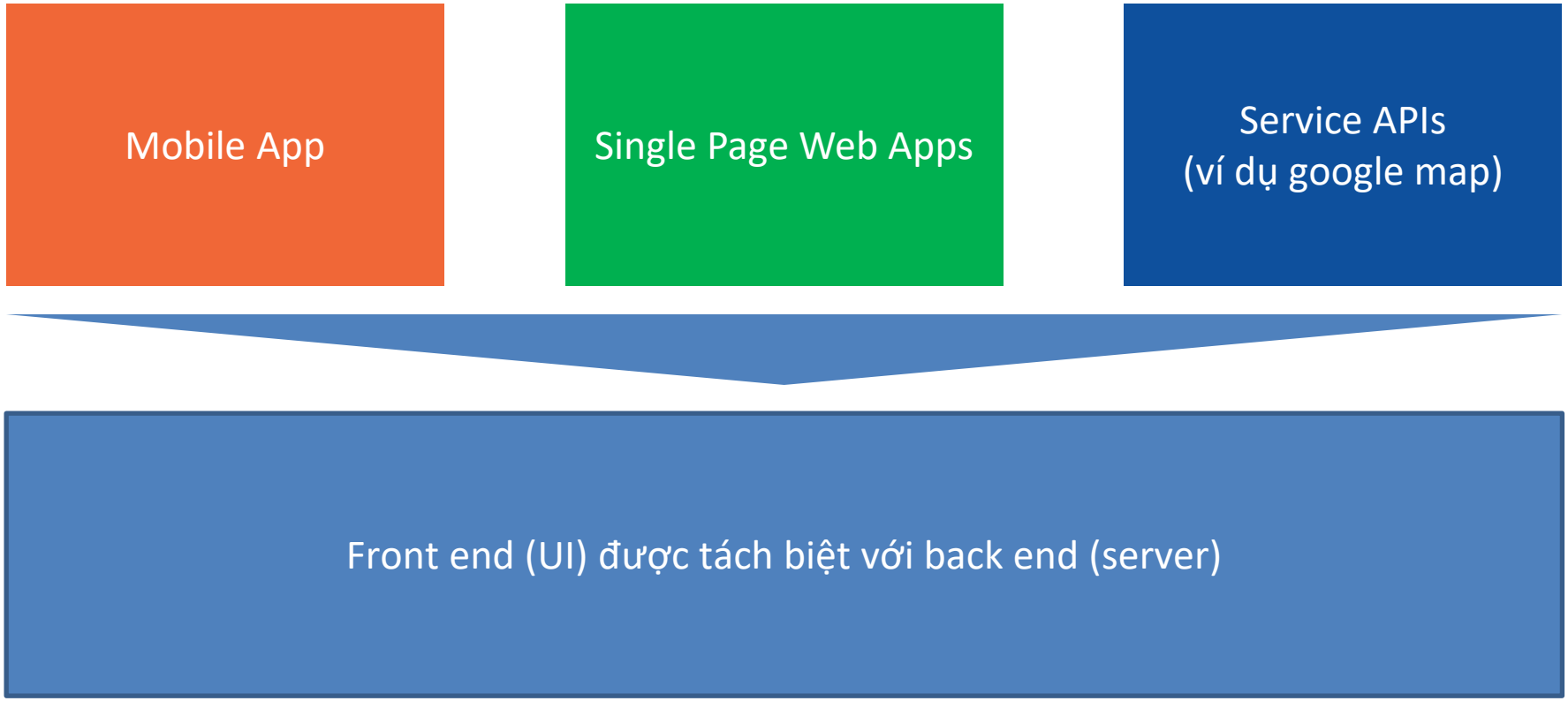
- **RESTful API** là một tiêu chuẩn dùng trong việc thiết kế API cho các ứng dụng web (thiết kế Web services)trong việc quản lý các tài nguyên.
- Tài nguyên hệ thống gồm: Dữ liệu, tệp văn bản, ảnh, âm thanh, video…
- Được truyền tải qua HTTP.

RESTFUL API LÀ GÌ

- **API: A**pplication **P**rogramming **I**nterface
  - Là một tập các quy tắc và cơ chế mà theo đó, một ứng dụng hay một thành phần sẽ tương tác với một ứng dụng hay thành phần khác
- **REST: RE**presentational **S**tate **T**ransfer
  - Là một dạng chuyển đổi cấu trúc dữ liệu, một kiểu kiến trúc để viết API

TẠI SAU DUNG RESTFUL API

Không phải tất cả các front end – UI chỉ yêu cầu nhận trang html từ server



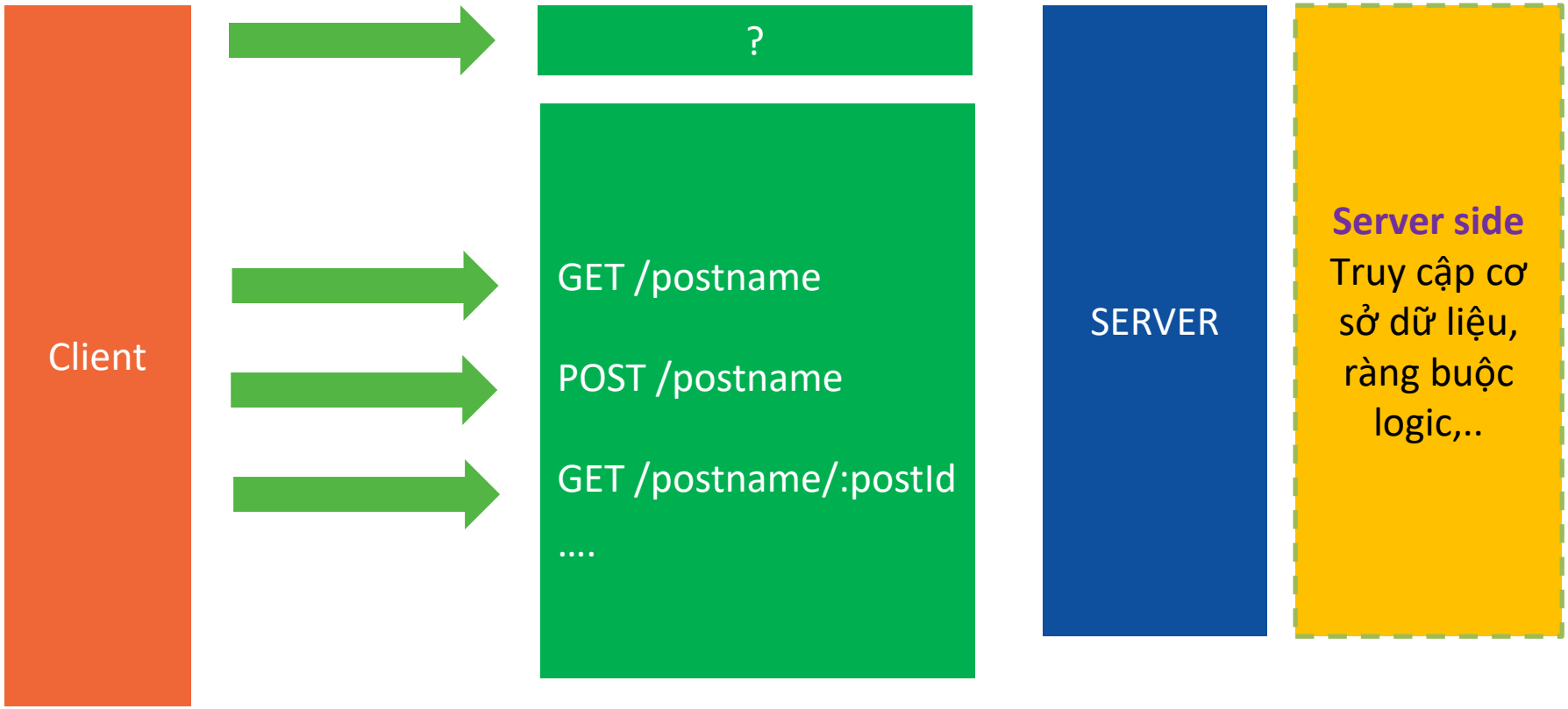
FPT Education
FPT POLYTECHNIC

DẠNG DỮ LIỆU ĐƯỢC DUNG TRONG RESTFUL API

HTML	Text	XML	JSON
Poly Technic	api.js	<name>Poly</name>	{"title": "Poly Technic"}
Dữ lieu + cấu trúc	Dữ liệu	Dữ liệu	Dữ liệu
Khả năng truy xuất khó nếu chỉ cần lấy dữ liệu	Khả năng truy xuất khó cấu trúc dữ liệu không rỗ ràng	Dễ đọc dữ liệu nhưng dài dòng	Dễ đọc dữ liệu và ngắn gọn. Có thể dễ dàng chuyển đổi bang javascript

RESTFUL API - HTTP

Http cung cấp các phương thức dùng để gọi phía server Restful
Thường dung 5 phương thức cơ bản sau: GET, POST PATCH, PUT, DELETE



RESTFUL API - HTTP

Các phương thức của HTTP

GET	POST	PUT
Nhận dữ liệu từ phía server	Gởi dữ liệu lên server (như thêm mới)	Thường dùng cho việc cập nhật dữ liệu

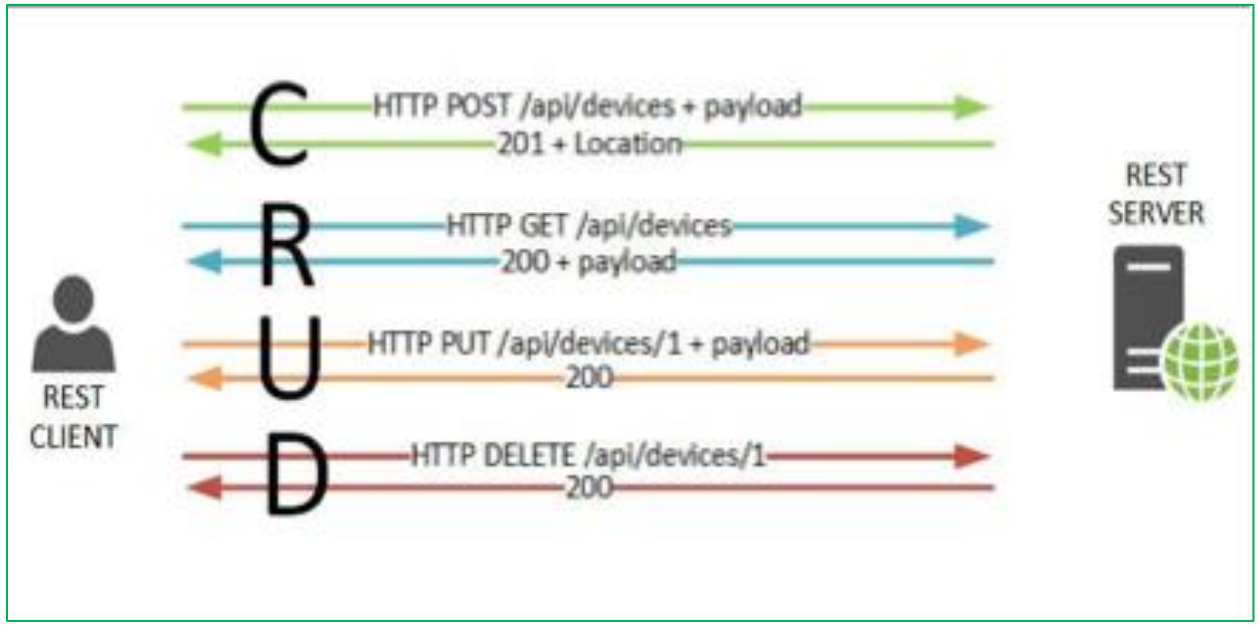
DELETE	PATCH
Xoá dữ lieu phía server	Cập nhật tài nguyên đã tồn tại trên server

Thiết kế API phải theo chuẩn RESTFul

Đơn giản

Dùng danh từ

Dùng đúng phương thức HTTP cung cấp



TẠO RESTFUL API

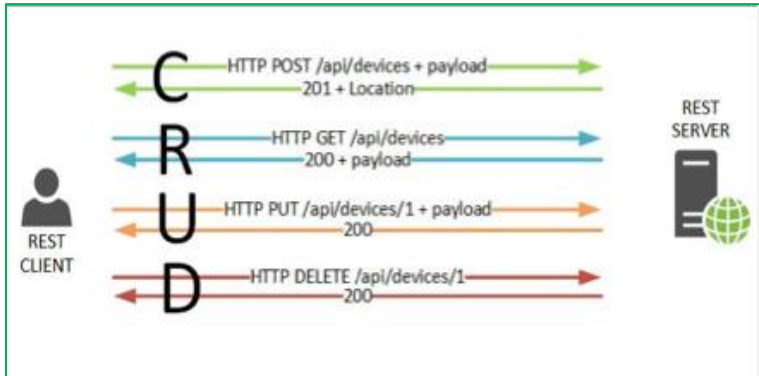
RESTFUL API thực hiện việc phản hồi lại các yêu cầu từ phía client

RESTFUL API với phương thức GET:

```
res.status(200).json({…});
```

RESTFUL API với phương thức POST:

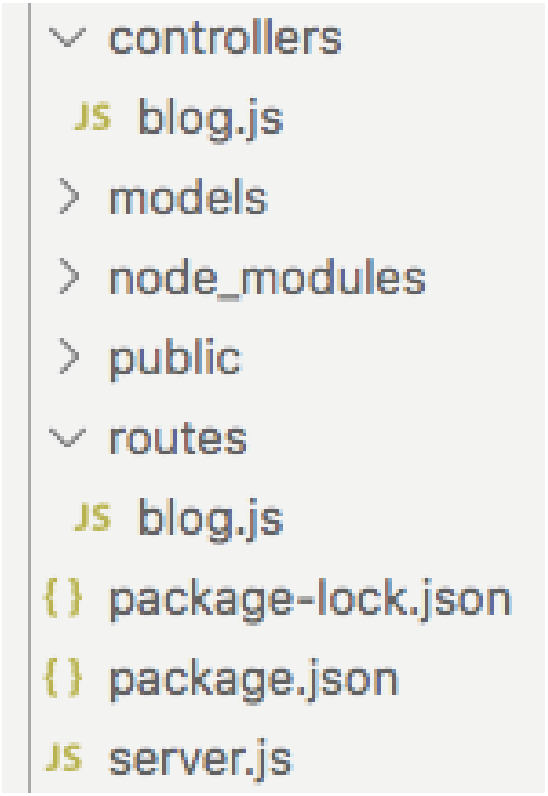
```
const title = req.body.title;
const content = req.body.content;
// Create post in db
res.status(201).json({
message: 'Post created successfully!',…});
```



TẠO RESTFUL API

Tổ chức RESTFUL API cho ứng dụng, giả sử ứng dung quản lý bài viết của 1 blog

- Folder controllers: chứa các hàm và API routes.
- Model: blog, User...
- Routes: Nơi định nghĩa tất cả các API route



TẠO RESTFUL API

Tổ chức RESTFUL API cho ứng dụng, giả sử ứng dung quản lý bài viết của 1 blog

controllers > JS blog.js > ...

```
exports.getPosts = (req, res, next) => {
  res.status(200).json({
    posts: [{ title: 'First Post', content: 'This is the first post!' }]
  });
};

exports.createPost = (req, res, next) => {
  const title = req.body.title;
  const content = req.body.content;
  res.status(201).json({
    message: 'Post created successfully!',
    post: { id: new Date().toISOString(), title: title, content: content }
  });
};
```

TẠO RESTFUL API

Tổ chức RESTFUL API cho ứng dụng, giả sử ứng dung quản lý bài viết của 1 blog

routes > JS blog.js > ...

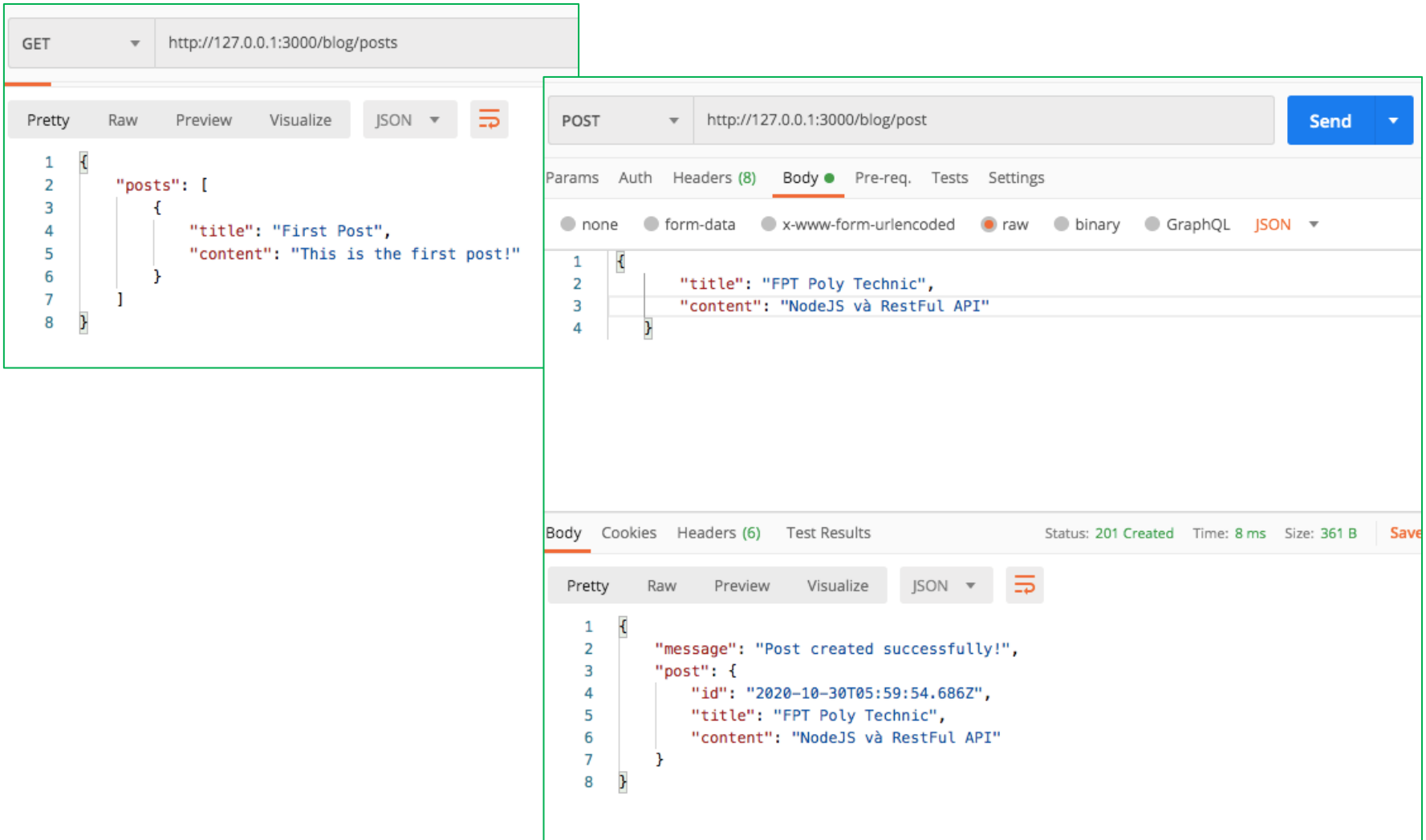
```
const express = require('express');

const blogController = require('../controllers/blog');

const router = express.Router();

// GET /blog/posts
router.get('/posts', blogController.getPosts);

// POST /blog/post
router.post('/post', blogController.createPost);

module.exports = router;
```

FPT Education
FPT POLYTECHNIC

TEST RESTFUL API VỚI POSTMAN

Tải và cài đặt tool postman

Thực hiên test API với postman




demo

PHẦN 2: ỨNG DUNG THỰC TIỄN

ĐẶC TẢ BLOG

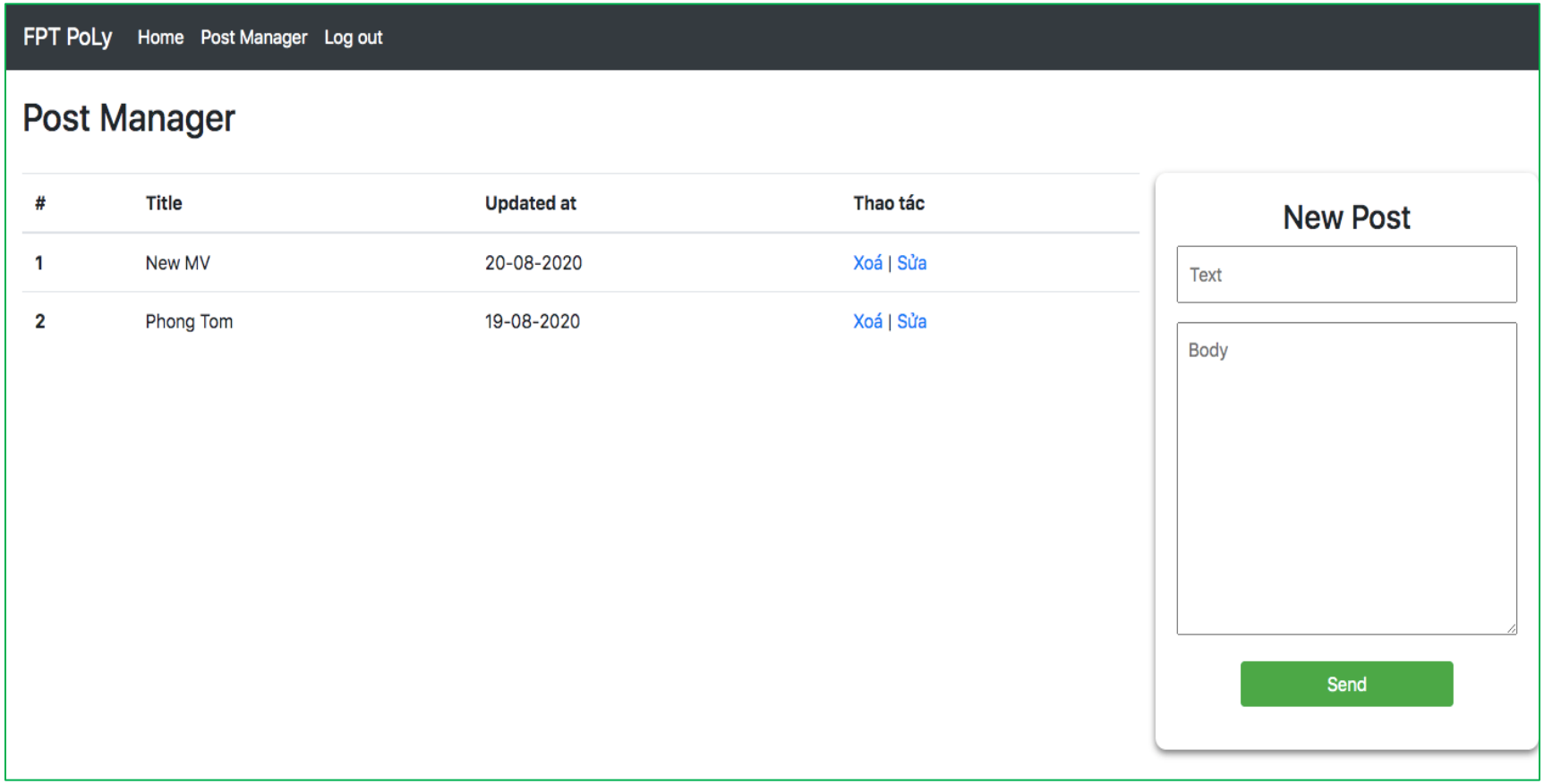
- Xây trang blog cho phép người dùng xem và bình luận bài viết
- Có nhiều chủ đề (Công nghệ, cuộc sống,…), Mỗi chủ đề có nhiều bài viết.
- Người dùng có thể bình luận trên nhiều bài viết.

Blog có 3 loại người dùng cơ bản:

- Admin: sau khi đăng nhập, có quyền CRUD chủ đề, CRUD bài viết, CRUD Bình luận
- Client (người dùng đã đăng ký, đăng nhập): bài viết theo chủ đề, bình luận bài viết
- Anonymous (người dùng chưa đăng nhập): xem danh sách bài viết

FPT Education
FPT POLYTECHNIC

GIAO DIỆN QUẢN TRỊ BLOG



CÁC BƯỚC THỰC HIỆN

DATABASE

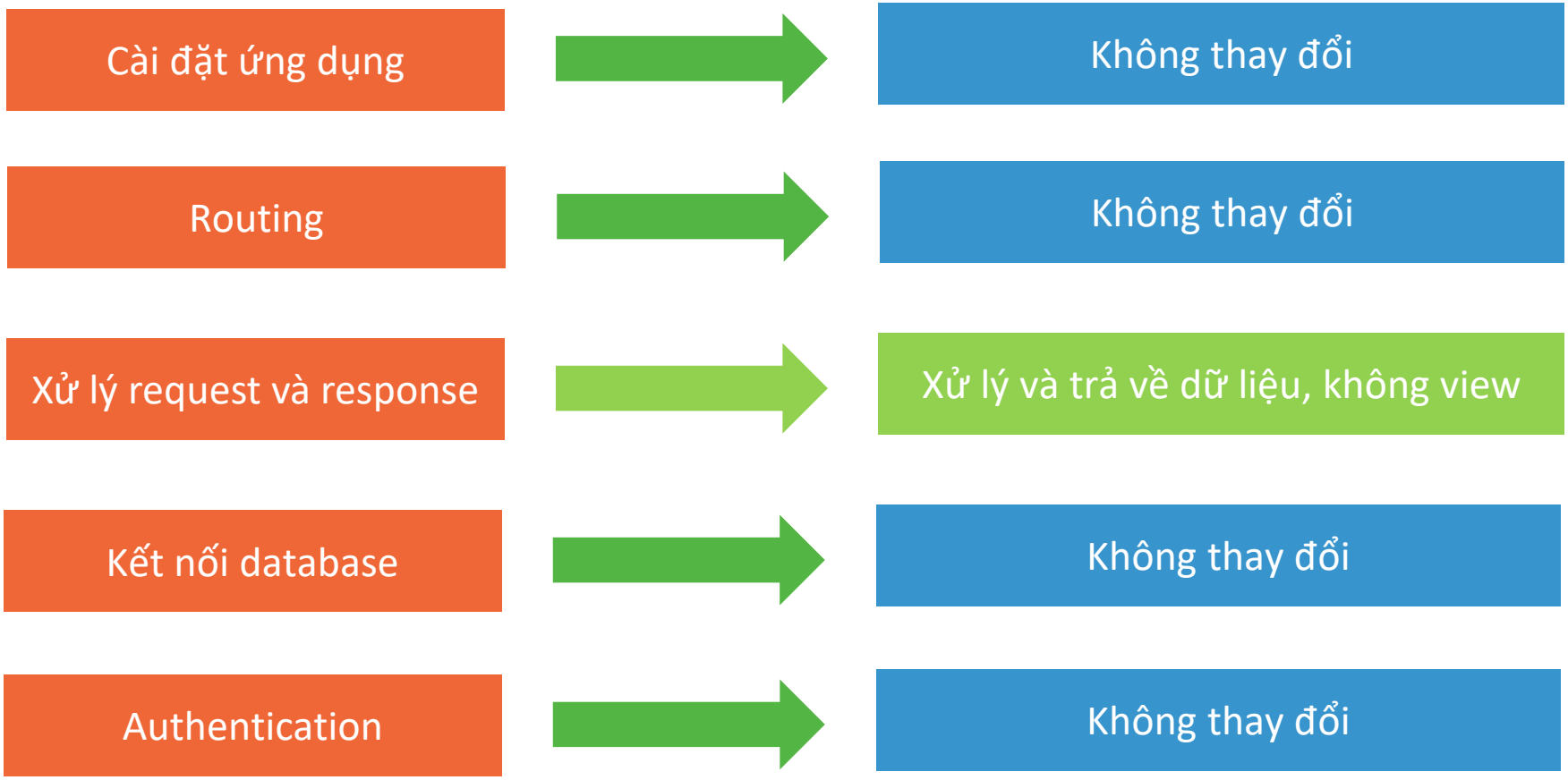
- Có những thực thể nào nào, mỗi thực thể có những field nào
- Mối quan hệ giữa các thực thể
- Dùng kỹ thuật nào để mô phỏng các mối quan hệ đó?

BACKEND

- Tạo Model, kết nối đến database
- Thực hiện validation
- Viết API (service) cho frontend

TEST

- Test tất cả RESTFUL API đã viết bằng postman


Cài đặt ứng dụng
Không thay đổi
Routing
Không thay đổi
Xử lý request và response
Xử lý và trả về dữ liệu, không view
Kết nối database
Không thay đổi
Authentication
Không thay đổi

POST (BÀI VIẾT) API

Dựa theo chuẩn RESTFul API, ta thiết kế 5 APIs cơ bản cho post như sau:

HTTP method	Route	Body	Access
GET	{host}/blog/posts	Không	Public
GET	{host}/blog/posts/:id	Không	Public
POST	{host}/blog/posts	Có	Admin
PUT	{host}/blog/posts/:id	Có	Admin
DELETE	{host}/blog/posts/:id	Không	Admin

BLOG MODEL

```
var db=require('./database');
var posts=[];
module.exports = class Post {
constructor() {
}
//thêm một bài viết
static addPost(newPost) {
db.query('insert into tblPost SET ?',newPost, function(err, data) {
if (err) throw err;
return true;
})
}
//xoa bài viết theo mã
static delPost(id) {
db.query(`delete from tblPost where postId= ${id}`, function(err, data) {
if (err) throw err;
return true;
})
}
//trả về tất cả bài viết
static fetchAll() {
let sql = `SELECT * FROM tblPost`;
db.query(sql, function(err, data) {
if (err) throw err;
posts=data;
});
return posts;
}
```

POST CONTROLLER

- **Import post model**
- **Cập nhật API trả về tất cả các bài post hiện hành**

```
const Post = require('../models/post');
exports.getPosts = (req, res, next) => {
const posts = Post.fetchAll();
res.status(200).json({
posts: posts
});
};
```

POST CONTROLLER

- **Import post model**
- **Cập nhật API thêm mới một bài post**

```
exports.createPost = (req, res, next) => {
const title = req.body.title;const content = req.body.content;
const newPost={ title: title, content: content,create_date: new Date().toISOString() };
if(Post.addPost(newPost)){
res.status(201).json({
message: 'Post created successfully!',
post: newPost
});}
else{
res.status(500).json({
message:”Có lỗi xay ra khi thêm mới post."
});}};
```

DELETE VÀ PUT

- **Post model:**
  - Thêm hàm delete, findById, update với cấu trúc như bài học trước

- **Post controller:**
  - exports.update = (req, res) => { };
  - exports.delete = (req, res) => { };

MODEL VỚI SEQUELIZE

- **Cập nhật lại Post model:**

```
const Sequelize = require('sequelize');

const sequelize = require('../util/database');

const Post = sequelize.define('tblPost', {
  postId: {
    type: Sequelize.INTEGER,
    autoIncrement: true,
    allowNull: false,
    primaryKey: true
  },
  title: Sequelize.STRING,
  content: {
    type: Sequelize.STRING,
    allowNull: false
  },
  create_date: {
    type: Sequelize.DATE,
    allowNull: false,
  }
},
{ timestamps: false }
);
module.exports = Post;
```

MODEL VỚI SEQUELIZE

- **Cập nhật lại Post controller: them mới**

controllers > JS blog.js > createPost > exports.createPost

```
exports.createPost = (req, res, next) => {
  const title = req.body.title;
  const content = req.body.content;
  const post = new Post({ title: title, content: content,create_date: new Date().toISOString() });
  post
    .save()
    .then(result => {
      res.status(201).json({
        message: 'Post created successfully!',
        post: result
      });
    })
    .catch(err => {
      if (!err.statusCode) {
        err.statusCode = 500;
      }
      next(err);
    });
```

MODEL VỚI SEQUELIZE

- **Cập nhật lại Post controller: update**

> controllers > JS blog.js > ...

```
exports.updatePost = (req, res, next) => {
  const postId = req.params.postId;
  const title = req.body.title;
  const content = req.body.content;

  Post.findById(postId)
    .then(post => {
      if (!post) {
        const error = new Error('Không tim thấy bài viết - post.');
        error.statusCode = 404;
        throw error;
      }
      post.title = title;
      post.content = content;
      return post.save();
    })
    .then(result => {
      res.status(200).json({ message: 'Post update thành công!', post: result });
    })
    .catch(err => {
      if (!err.statusCode) {
        err.statusCode = 500;
      }
      next(err);
    });
};
```

MODEL VỚI SEQUELIZE

- **Lấy bài viết – Post theo Id**

```
// GET /blog/posts
router.get('/posts', blogController.getPosts);
router.get('/posts/:postId', blogController.getPostById);
```

controllers > JS blog.js > createPost

```
exports.getPostById = (req, res, next) => {
  const postId = req.params.postId;
  Post.findByPk(postId)
    .then(post => {
      if (!post) {
        const error = new Error('Không tìm thấy bài viết- post.');
        error.statusCode = 404;
        throw error;
      }
      console.log(post);
      res.status(200).json({ message: 'Post được tìm thấy.', post: post });
    })
    .catch(err => {
      if (!err.statusCode) {
        err.statusCode = 500;
      }
      next(err);
    });
};
```


demo

SUMARRY

- ☑ Biết được tại sao phải dùng REST API
- ☑ Hiểu cấu trúc Rest API
- ☑ Tạo được REST API
- ☑ Test Rest API


thank
you!