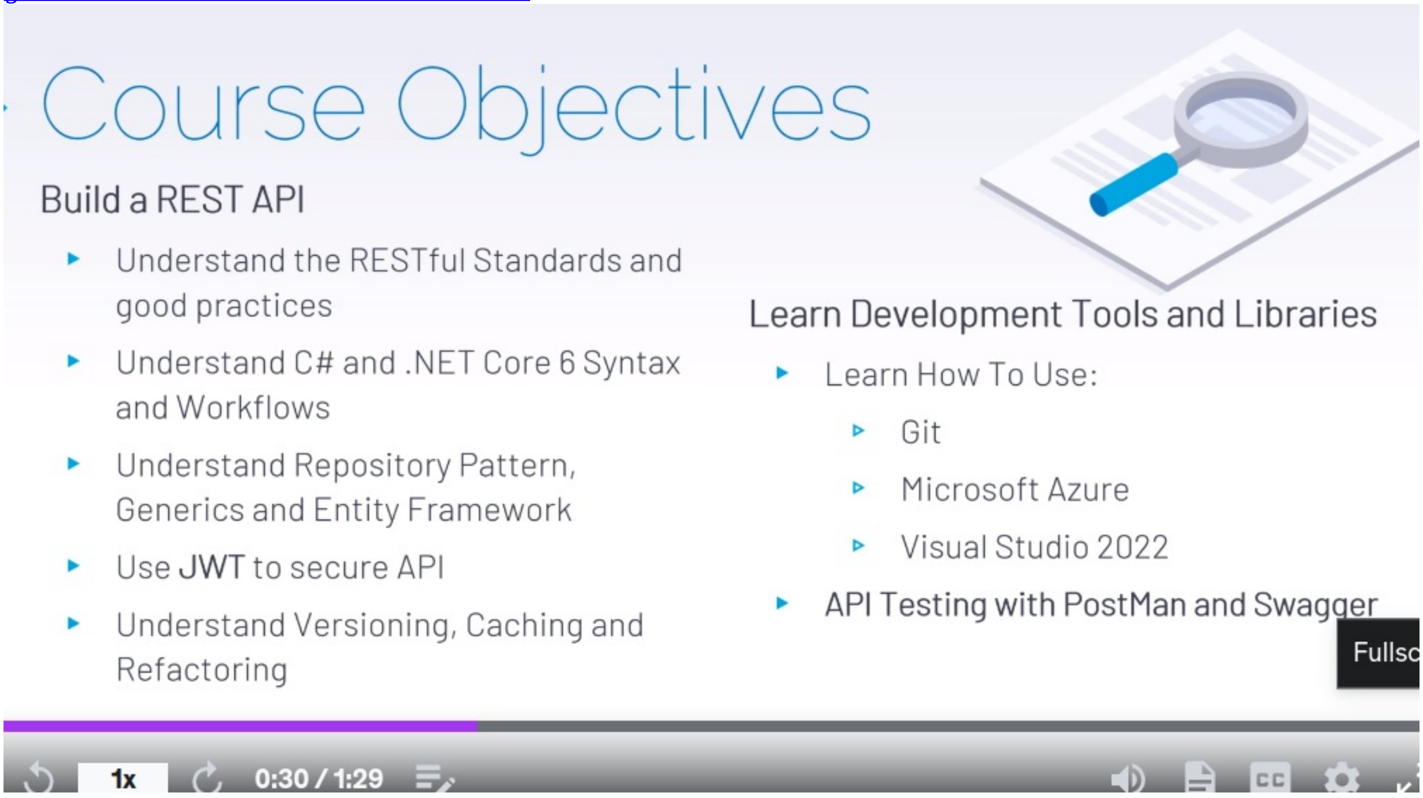
Udemy course: Ultimate ASP.NET pt. 1

Introduction and Environment Configuration

https://www.udemy.com/course/ultimate-aspnet-5-web-api-development-guide/learn/lecture/31123480#overview



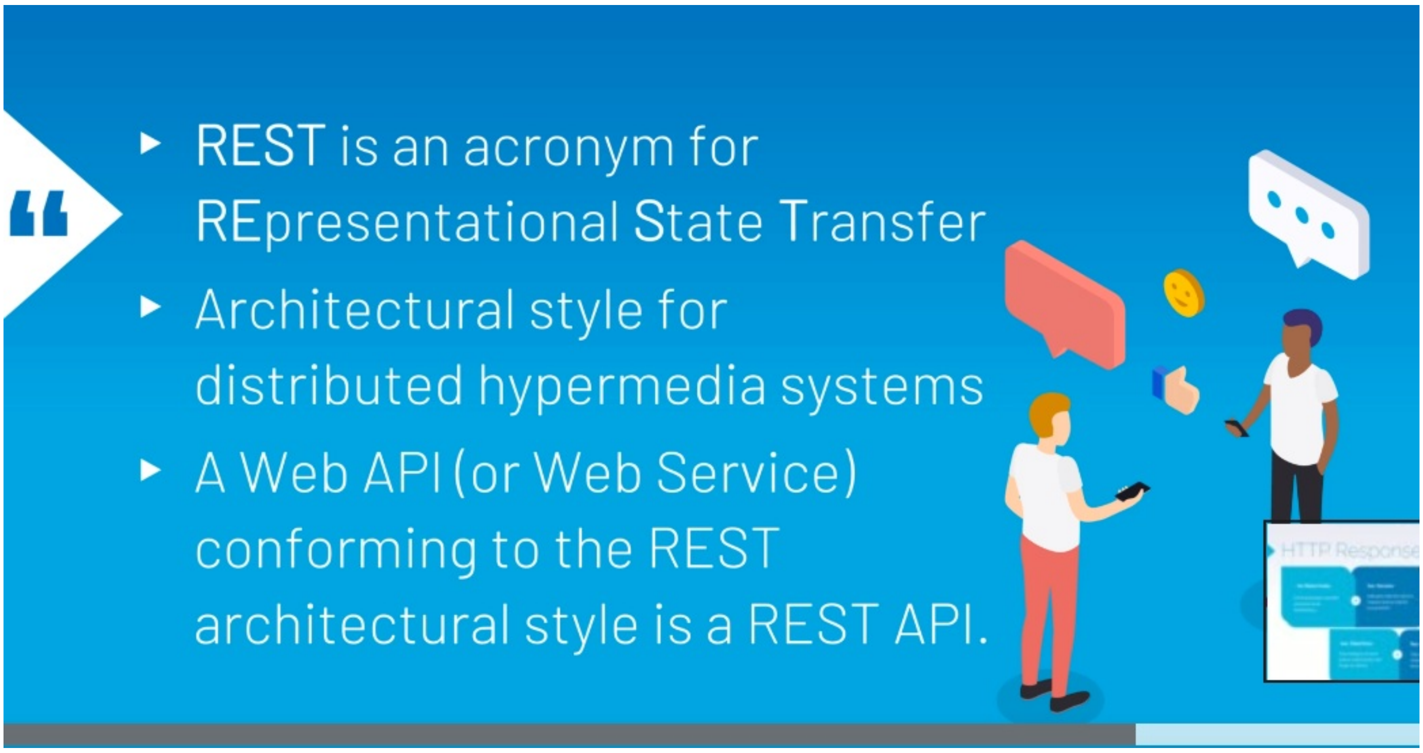
Course Objectives

Build a REST API

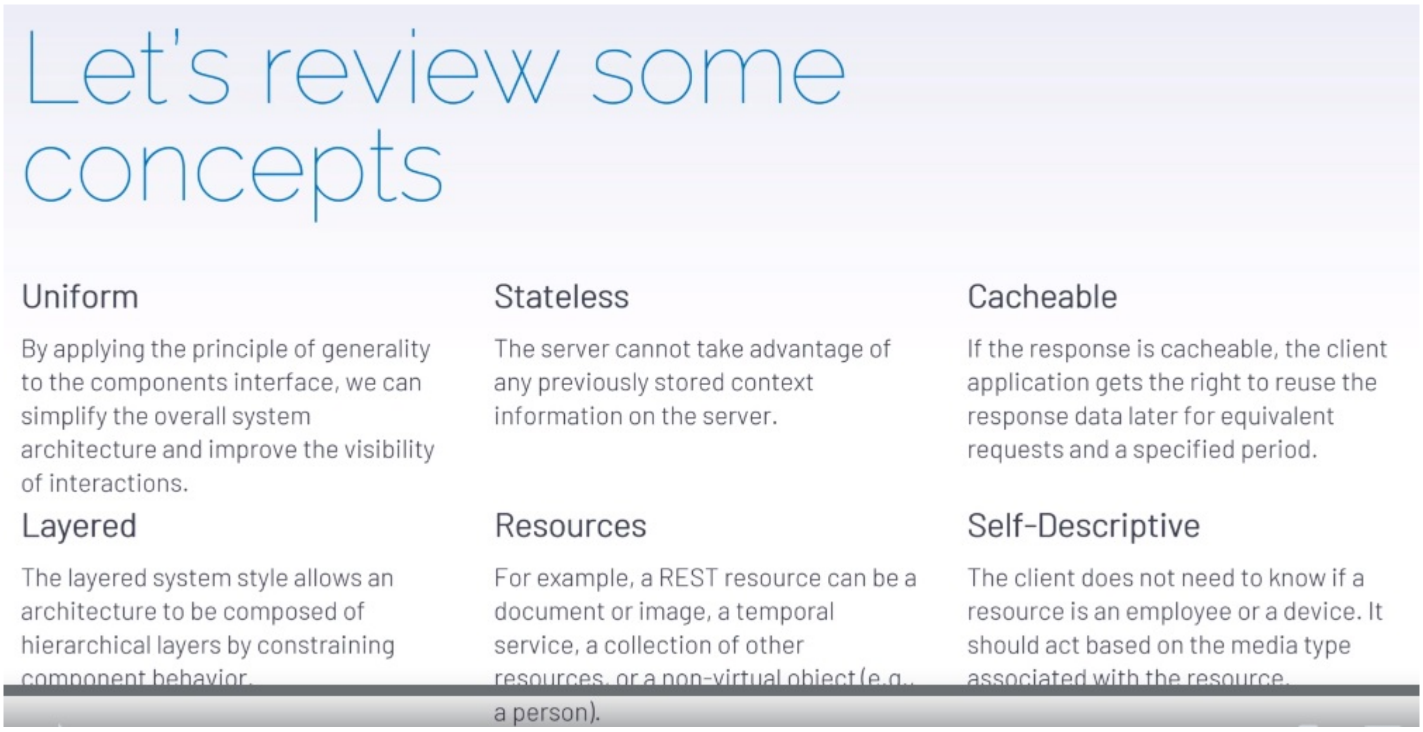
- Understand the RESTful Standards and good practices
- Understand C# and .NET Core 6 Syntax and Workflows
- Understand Repository Pattern, Generics and Entity Framework
- Use JWT to secure API
- Understand Versioning, Caching and Refactoring

Learn Development Tools and Libraries

- Learn How To Use:
  - Git
  - Microsoft Azure
  - Visual Studio 2022
- API Testing with PostMan and Swagger



- REST is an acronym for REpresentational State Transfer
- Architectural style for distributed hypermedia systems
- A Web API (or Web Service) conforming to the REST architectural style is a REST API.



Let's review some concepts

Uniform

By applying the principle of generality to the components interface, we can simplify the overall system architecture and improve the visibility of interactions.

Stateless

The server cannot take advantage of any previously stored context information on the server.

Cacheable

If the response is cacheable, the client application gets the right to reuse the response data later for equivalent requests and a specified period.

Layered

The layered system style allows an architecture to be composed of hierarchical layers by constraining component behavior.

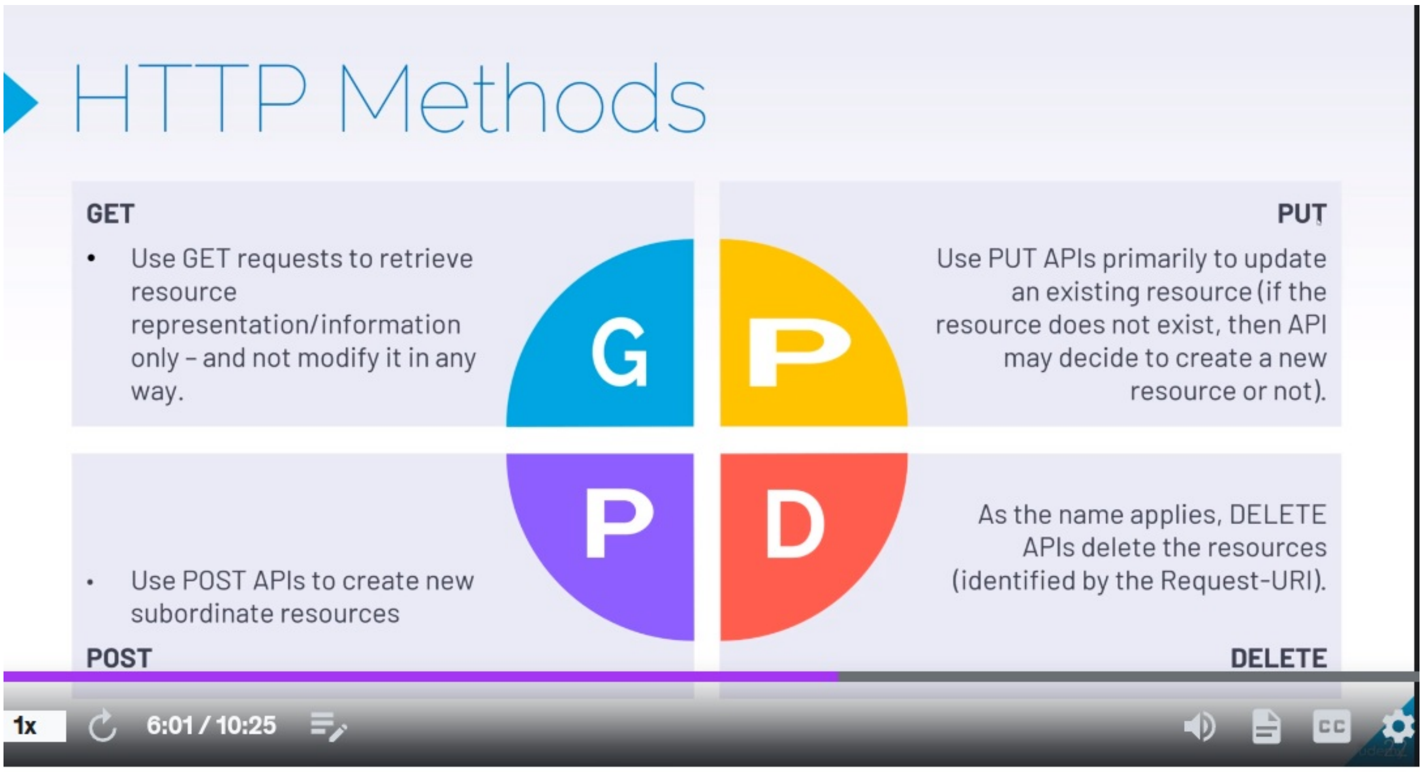
Resources

For example, a REST resource can be a document or image, a temporal service, a collection of other resources, or a non-virtual object (e.g., a person).

Self-Descriptive

The client does not need to know if a resource is an employee or a device. It should act based on the media type associated with the resource.

- Uniform
    - so easy for client to connect
- Stateless
    - no collection of who connected and when
    - we are not keeping cookies and sessions etc.
- Cacheable
    - reuse response of API server
- Layered
    - our System is not one big app in one application
    - we might have web, and mobile application, different kind of interfaces for different users
    - all variations only talk to one API
- Resources
    - anything which can be retrieved/interacted from the API
    - any data, metadata, hypermedia, file etc.
    - HTTP Methods
        - GET, PUT, POST, DELETE
        - the most common
- Self-Descriptive
    -



HTTP Methods

GET

- Use GET requests to retrieve resource representation/information only – and not modify it in any way.

PUT

Use PUT APIs primarily to update an existing resource (if the resource does not exist, then API may decide to create a new resource or not).

- Use POST APIs to create new subordinate resources

POST

As the name applies, DELETE APIs delete the resources (identified by the Request-URI).

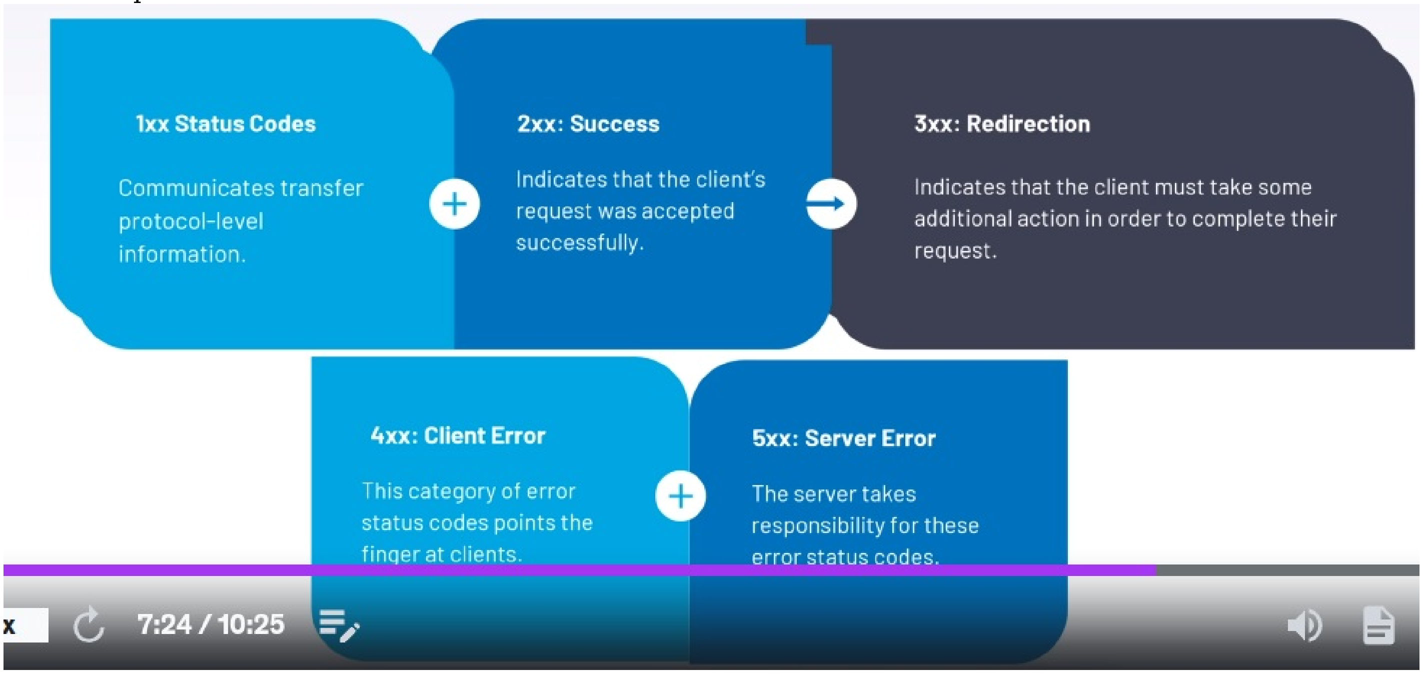
DELETE

PUT:

- recommended for update operation for example
- resources already there, replace existing with new data
- easy to be messed up; but otherwise works beautifully
- let the context determine your action

POST
DELETE

HTTP Responses



1xx Status Codes

Communicates transfer protocol-level information.

2xx: Success

Indicates that the client's request was accepted successfully.

3xx: Redirection

Indicates that the client must take some additional action in order to complete their request.

4xx: Client Error

This category of error status codes points the finger at clients.

5xx: Server Error

The server takes responsibility for these error status codes.

Postman

- create private workspace
- on plus sign button add requests
- free API service
  - e.g. apipheny.io

    - url: https://api.publicapis.org/entries
    - or with query string: https://api.nationalize.io?name=nathaniel
- insert as GET request
    - you will get a response as JSON output
    - we will producing such an JSON from our API later on
    - quite commonly used in development staffs
    - an API developed by .NET can be consumed by other technologies
- some APIs allow you to pass in a parameter of them