Web API using JavaScript
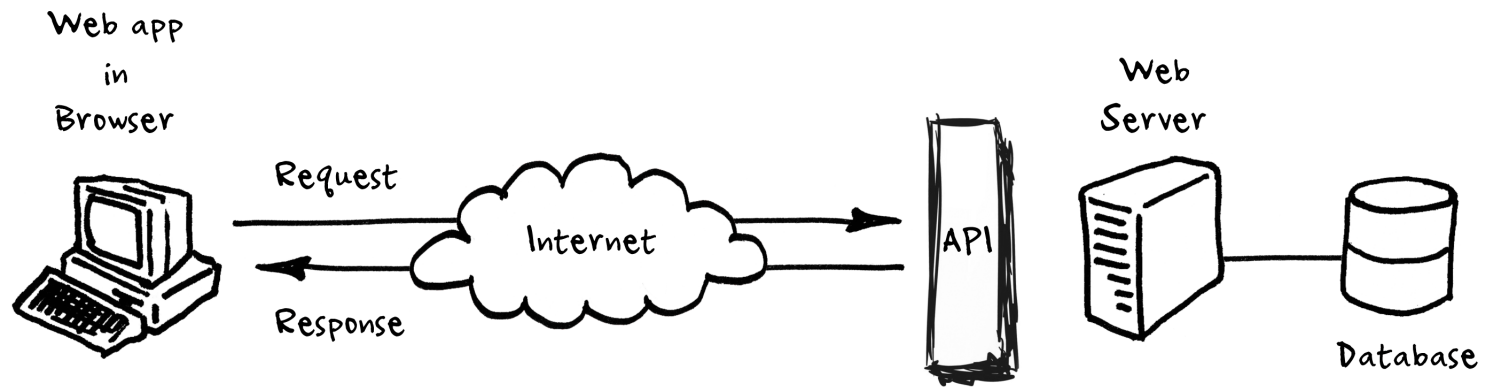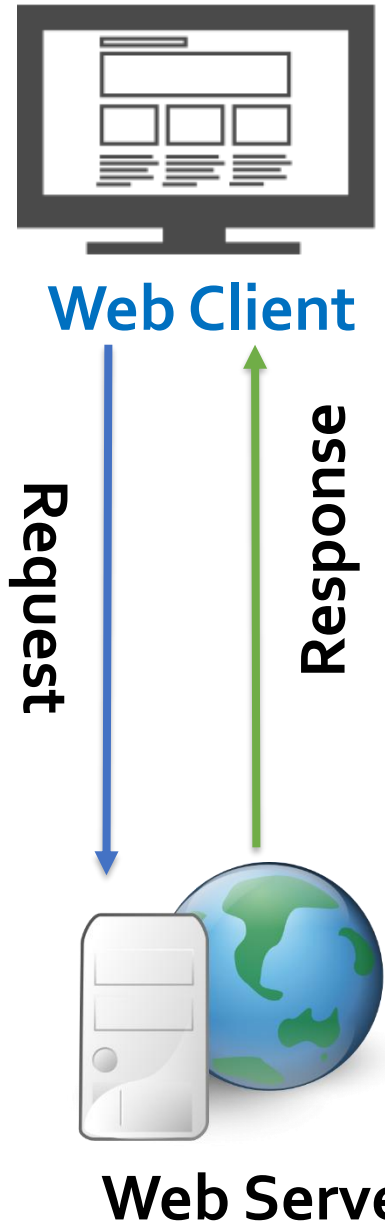
Outline

1. Web and HTTP

2. Web API

3. Web API using Next.js

Course Roadmap

Web Client

Request

Response

Web Server
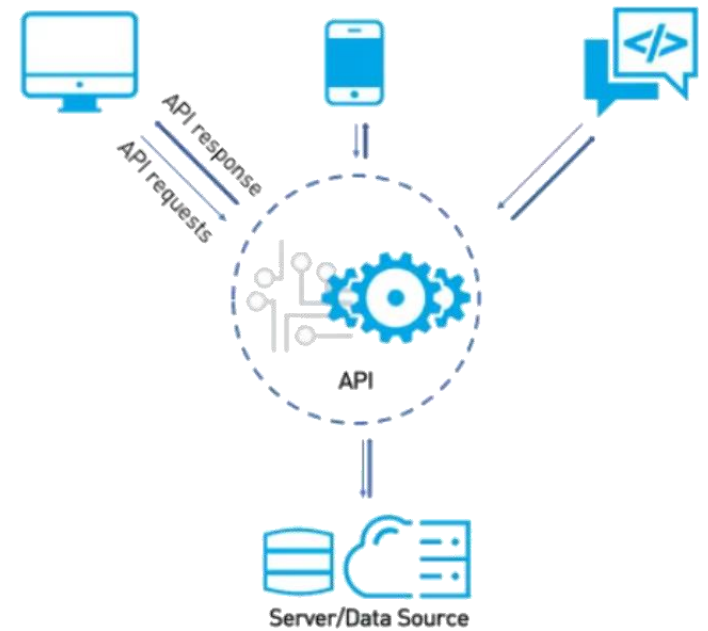
Frontend development

- HTML for page content & structure
- CSS for styling
- JavaScript for interaction

Backend development

We are HERE

- Web API
- Web Pages
- Data Management

HTML5

CSS3

JavaScript
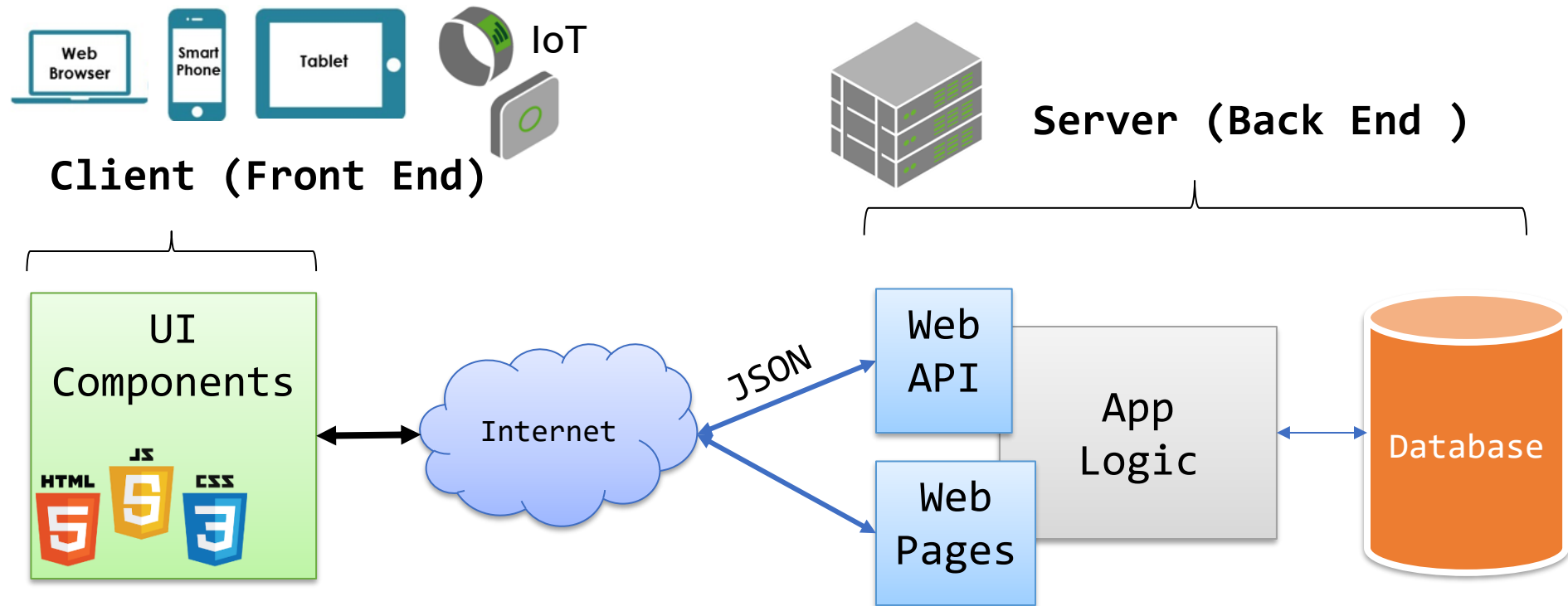
NEXT.js

Prisma

Web App Architecture

- Front-end made-up of **multiple UI components loaded** in response to user actions
- Back-end Web API and Web pages

What is a Web API

- Web API: A set of methods exposed over the web via HTTP to allow **programmatic access to applications**

- Web API are designed for **broad reach**:
  - Can be accessed by a broad range of clients including browsers and mobile devices
  - Can be implemented or consumed in any language

- Uses HTTP as an **application protocol**

Web and HTTP

GET /index.html
HTTP/1.1

HTTP/1.1 200 OK
"Welcome to our
Web site!"

What is Web?

- Web = **global distributed system of interlinked resources accessed over the Internet using the HTTP protocol**

  – Consists of set of **resources** located on different servers:

    HTML pages, images, videos and other resources

  – Resources have unique **URL** (Uniform Resource Locator) address

  – Accessed through standard **HTTP** protocol

- The Web has a Client/Server architecture:

  – **Web browser** (client) requests resources (using HTTP protocol) and displays them

  – **Web server** sends resources in response to requests (using HTTP protocol)

How the Web Works?
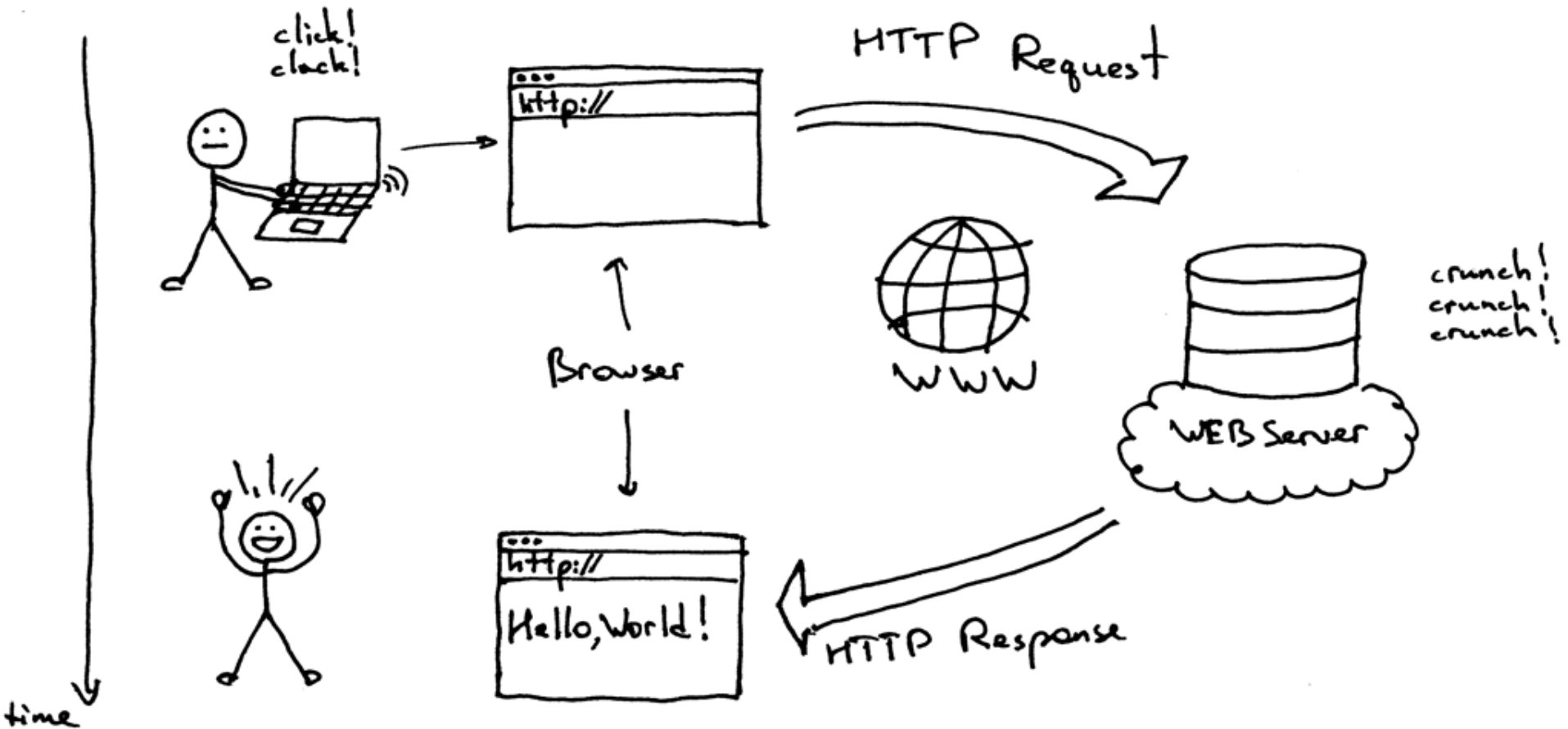
Uniform Resource Locator (URL)

```
http://www.qu.edu.qa:80/cse/logo.gif
```
protocol   host name   Port   Url Path

- URL is a formatted string, consisting of:

  – **Protocol** for communicating with the server (e.g., http, https, …)

  – **Name of the server or IP** address plus port (e.g. qu.edu.qa:80, localhost:8080)

  – **Path of a resource** (e.g. **/ceng/index.html**)

  – **Parameters** aka **Query String** (optional), e.g.

  https://www.google.com/search**?q=qatar%20university**

URL Example
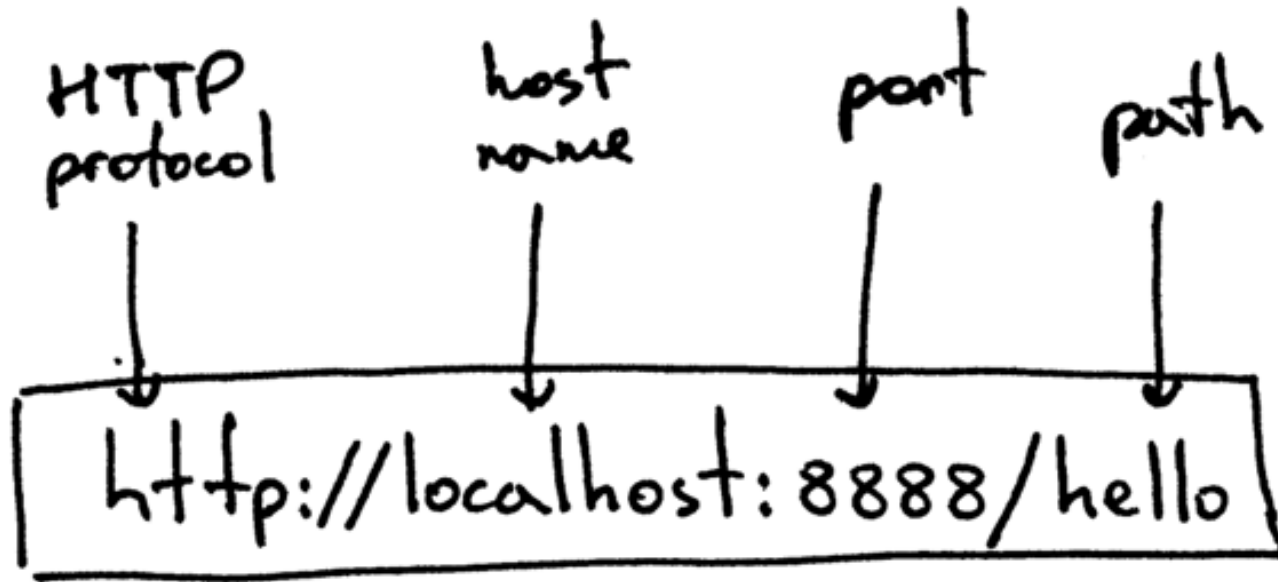
URL Encoding

- According [RFC 1738](#), the characters allowed in URL are alphanumeric [0-9a-zA-Z] and the special characters $-_.+!*'()

- Unsafe characters should be encoded, e.g.,

http://google.com/search?q=qatar%20university

Commonly encoded values:

ASCII Character	URL-encoding
space	%20
!	%21
"	%22
#	%23
$	%24
%	%25
&	%26

Web uses **Request/Response** interaction model
HTTP is the ***message protocol*** of the Web

Client Request: "I need a *resource* (html page, picture, pdf doc, mp3 file…)"

HTTP request

```
GET /resurceUri
    HTTP/1.1
```

Resource access

Web Browser (the client)

HTTP response

Web Server

Server Response: "Here you go!"
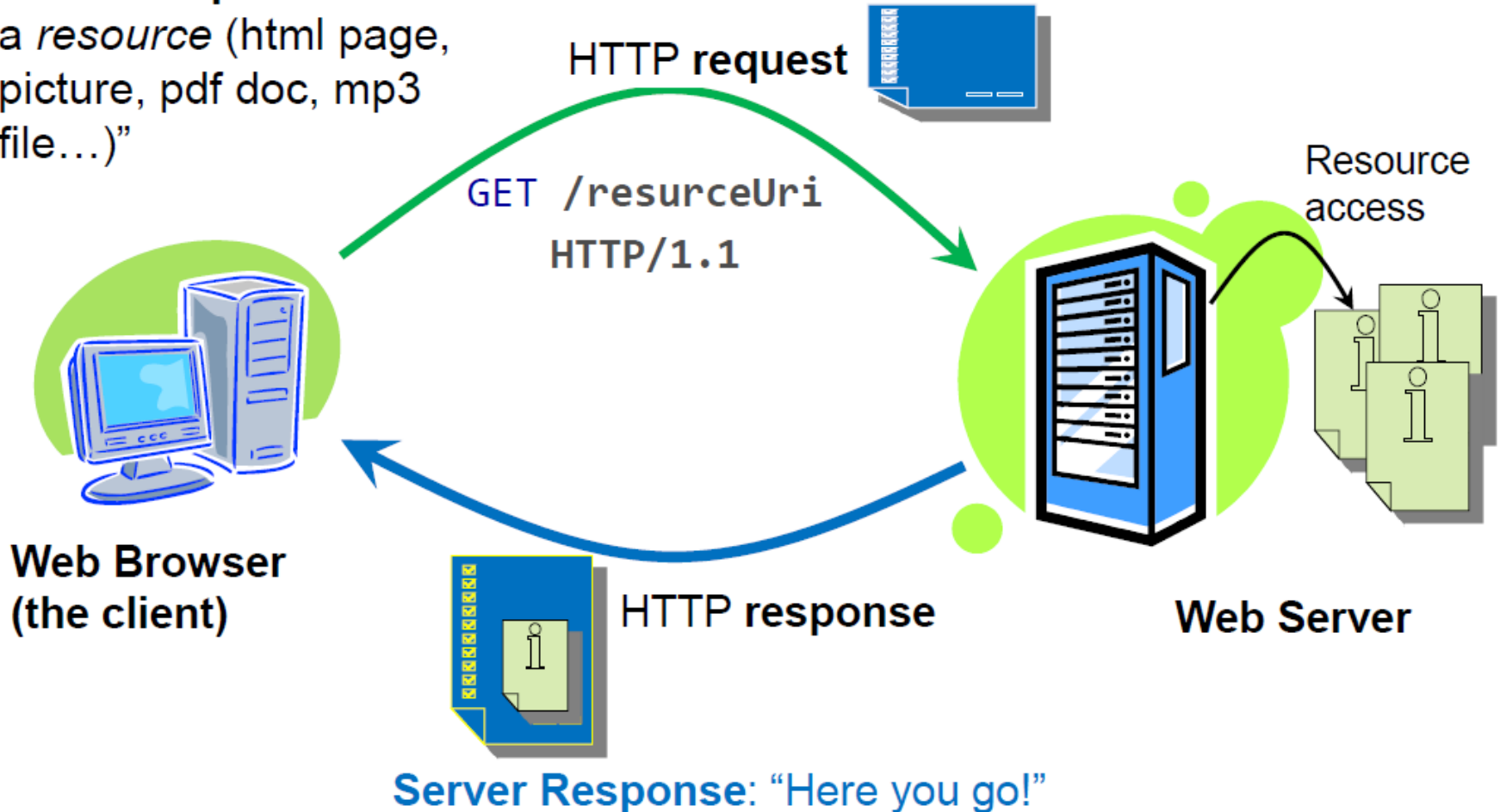
The sequence for retrieving a resource

User clicks a link in the browser.

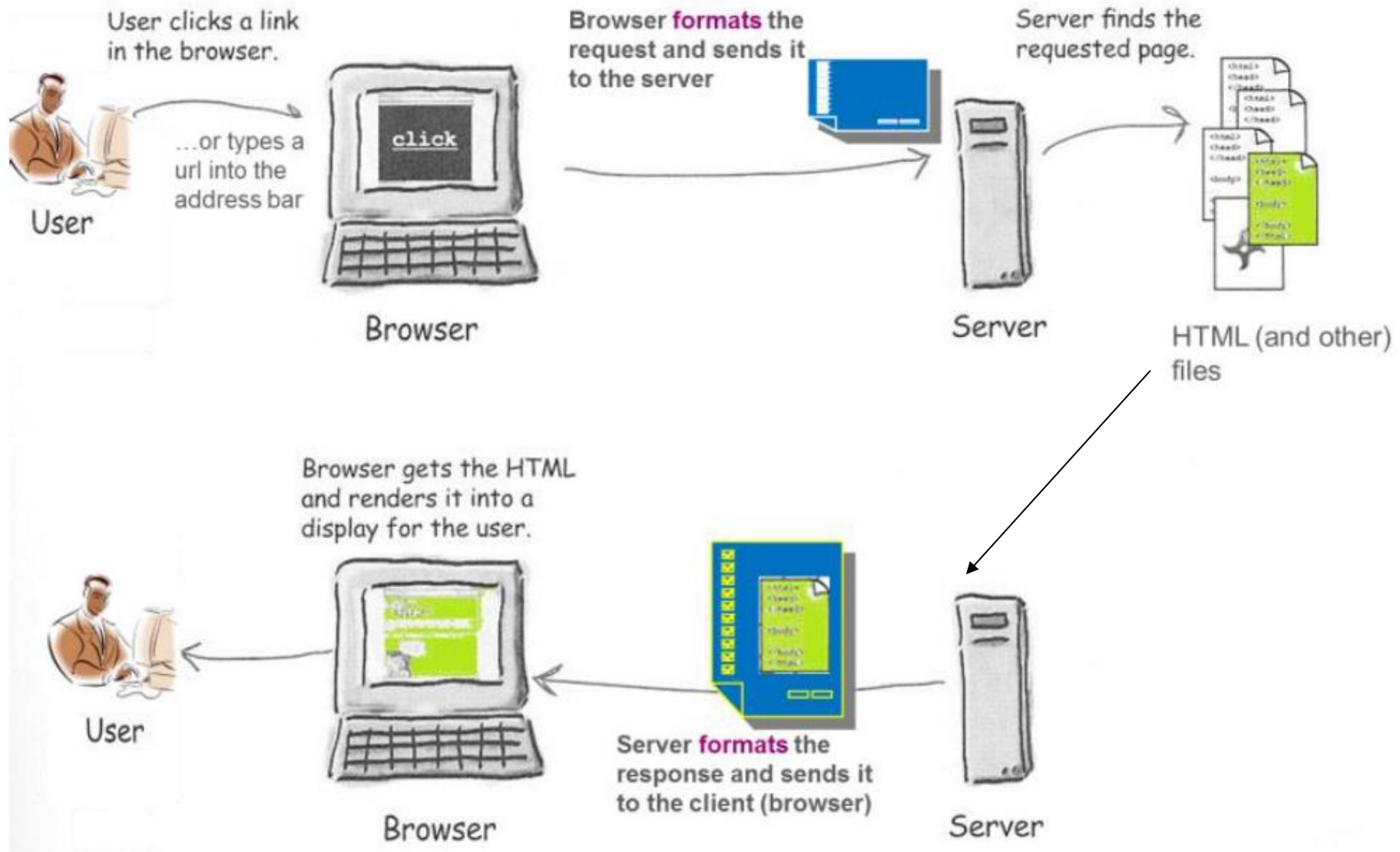...or types a url into the address bar

User

Browser

Browser **formats** the request and sends it to the server

Server finds the requested page.

Server

HTML (and other) files

Browser gets the HTML and renders it into a display for the user.

User

Browser

Server **formats** the response and sends it to the client (browser)

Server

Request and Response Examples

◆ HTTP request:

request line
(GET, POST,
HEAD commands)

header
lines

```
GET /index.html HTTP/1.1
Host: localhost:800
User-Agent: Mozilla/5.0
<CRLF>
```

The empty line denotes the end of the request header

◆ HTTP response:

```
HTTP/1.1 200 OK
Content-Length: 54
<CRLF>
<html><title>Hello</title>
Welcome to our site</html>
```

The empty line denotes the end of the response header

HTTP Request Message

- Request message sent by a client consists of

– **Request line** – request method (GET, POST, HEAD, …), resource URI, and protocol version

– **Request headers** – additional parameters

– **Body** – optional data

- e.g. posted form data, files, etc.

```
<request method> <URI> <HTTP version>
<headers>
<empty line>
<body>
```

HTTP Request Methods

- **GET**

  – **Retrieve a resource** (could be static resource such as an image or a dynamically generated resource)

  – Input is appended to the request URL E.g.,

  http://google.com/?q=Qatar

- **POST**

  – **Create or Update a resource**

  – Web pages often include form input. Input is submitted to server in the **message body.** E.g.,

```
POST /calc HTTP/1.1
Host: localhost
Content-Type: application/x-www-form-urlencoded
Content-Length: 27
num1=20&operation=*&num2=10
```

HTTP Response Message

- Response message sent by the server

– **Status line** – protocol version, status code, status phrase

– **Response headers** – provide metadata such as the Content-Type

– **Body** – the contents of the response (i.e., the requested resource)

```
<HTTP version> <status code> <status text>
<headers>
<empty line>
<response body>
```

HTTP Response – Example

status line
(protocol
status code
status text)

Try it out and see HTTP
in action using **HttpFox**

```
HTTP/1.1 200 OK
Content-Type: text/html
Server: QU Web Server
Content-Length: 131
<CRLF>
<html>
   <head><title>Calculator</title></head>
   <body>20 * 10 = 200
       <br><br>
       <a href='/calc'>Calculator</a>
   </body>
</html>
```

HTTP response headers

The empty line denotes the end of the response header

Response body. e.g., requested HTML file

Common Internet Media Types

- The **Content-Type** header describes the media type contained in the body of HTTP message
- **Full list @** http://en.wikipedia.org/wiki/MIME_type
- Commonly used media types (**type/**subtype):

Type/Subtype	Description
application/json	JSON data
image/gif	GIF image
image/png	PNG image
video/mp4	MP4 video
text/xml	XML
text/html	HTML
text/plain	Just text

HTTP Response Status Codes

- Status code appears in 1st line in the response message

- HTTP response code classes
  - 2xx: success (e.g., "200 OK")
  - 3xx: redirection (e.g., "302 Found")

  "302 Found" is used for redirecting the Web browser to another URL

  - 4xx: client error (e.g., "404 Not Found")
  - 5xx: server error (e.g., "503 Service Unavailable")

Popular Status Codes

Code	Reason	Description
200	OK	Success!
301	Moved Permanently	Resource moved, don't check here again
302	Moved Temporarily	Resource moved, but check here again
304	Not Modified	Resource hasn't changed since last retrieval
400	Bad Request	Bad syntax?
401	Unauthorized	Client might need to authenticate
403	Forbidden	Refused access
404	Not found	Resource doesn't exist
500	Internal Server Error	Something went wrong during processing
503	Service Unavailable	Server will not service the request

Browser Redirection

- HTTP browser redirection example

  – HTTP GET requesting a moved URL:

(Request-Line)	GET /qu HTTP/1.1
Host	localhost:800
User-Agent	Mozilla/5.0 (Windows NT 6.3; WOW64; rv:27.0) Gecko/20100101 Firefox/27.0
Accept	text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8

  – The HTTP response says that the browser should request another URL:

|	
(Status-Line)	HTTP/1.1 301 Moved Permanently
Location	http://qu.edu.qa

Web API
(aka REST Services)

What is a Web API?

- Web API = Web accessible Application Programming Interface. Also known as REST Services.

- Web API is a web service that accepts requests and returns **structured data** (JSON in most cases)

  - o Programmatically accessible at a particular URL

  - o You can think of it as a Web page returning JSON instead of HTML

- Major goal = **interoperability between heterogeneous systems**

REST Principles
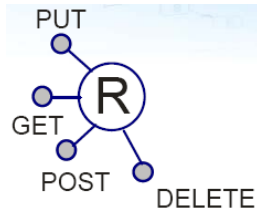
- **Resources have unique address** (**nouns**) i.e., a URI

e.g., http://example.com/customers/123

- **Can use a Uniform Interface** (**verbs**) to access them:

  - HTTP verbs: GET, POST, PUT, and DELETE

- **Resource has representation(s)** (**data format**)

  - A resource can be in a variety of data formats: JSON, XML, RSS..
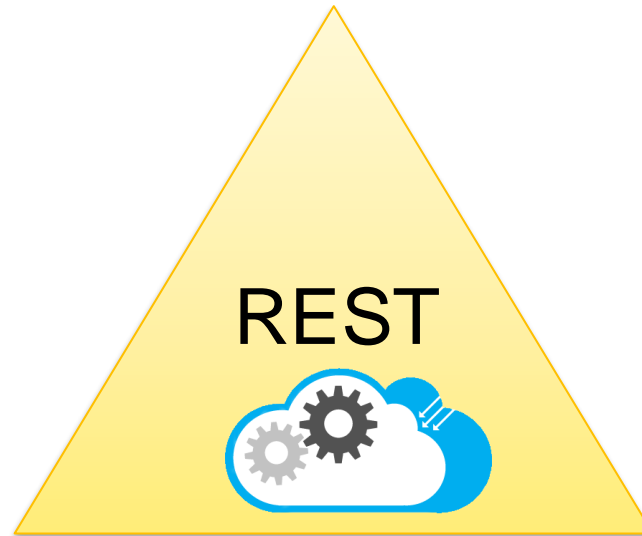
Resources

- The key abstraction in REST is a **resource**

- A resource is a conceptual mapping to a set of entities

  - o Any **information that can be named can be a resource**: a document or image, a temporal service (e.g. "today's weather in Doha"), a collection of books and their authors, and so on

REST Services Main Concepts

Nouns **(Resources)**
e.g., http://example.com/employees/12345

REST

Verbs
e.g., GET, POST

Representations
e.g., XML, JSON

Naming Resources

- REST uses URL to identify resources

> Dedicated **api** path is recommended for better organization

  - o http://localhost/api/books/
  - o http://localhost/api/books/ISBN-0011
  - o http://localhost/api/books/ISBN-0011/authors

  - o http://localhost/api/classes
  - o http://localhost/api/classes/cmps350
  - o http://localhost/api/classes/cmps350/students

- As you traverse the **path** from more generic to more specific, you are navigating the data

A Collection with Resources

Resource Model

A Collection with Resources

A Singleton Resource

Sub-collections and Sub-resources
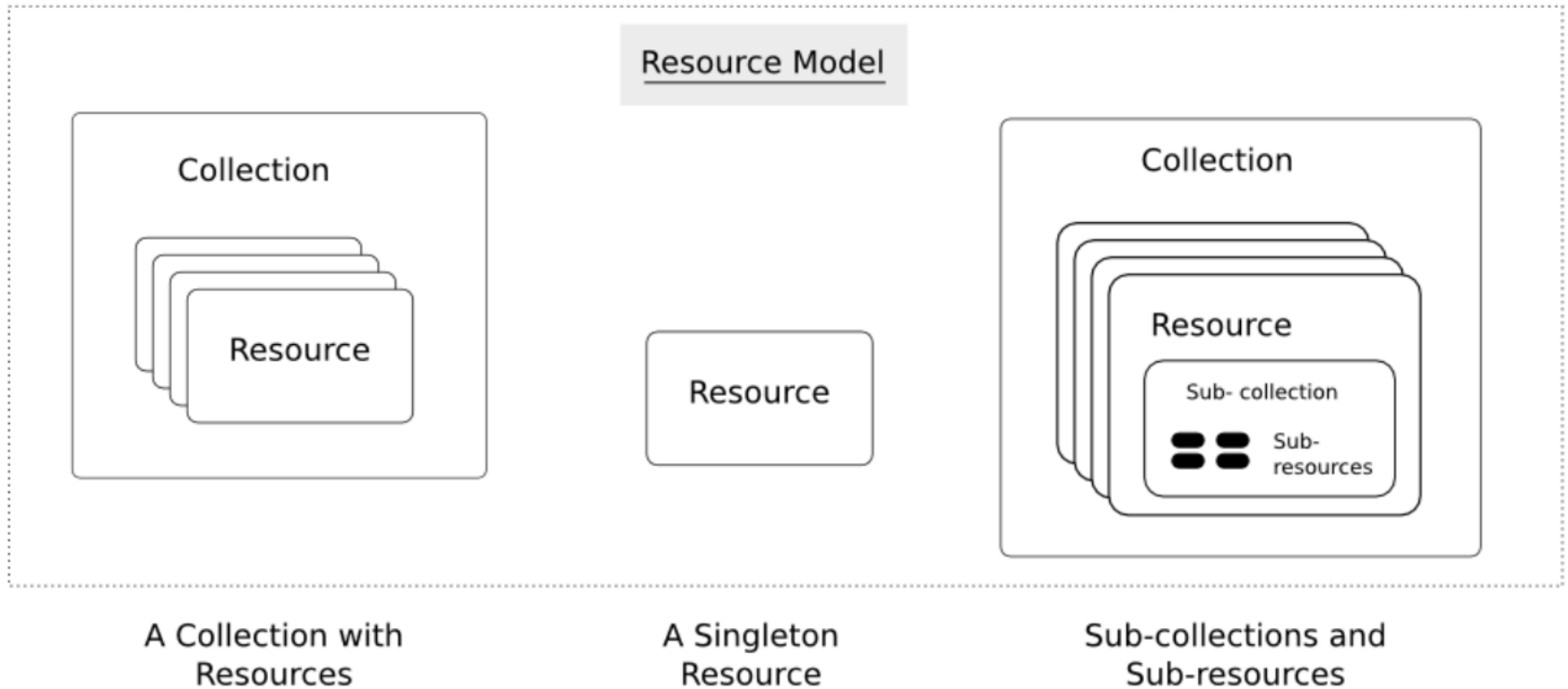
Example CRUD (Create, Read, Update and Delete) API that manages books

- Create a new book
  - **POST** /books

- Retrieve all books
  - **GET** /books

- Retrieve a particular book
  - **GET** /books/:id

- Replace a book
  - **PUT** /books/:id

- Update a book
  - **PATCH** /books/:id

- Delete a book
  - **DELETE** /books/:id

Representations

Two main formats:

- **JSON**

```
{
    code: 'cmp123',
    name: 'Web Development'
}
```

- **XML**

```
<course>
    <code>cmp123</code>
    <name>Web Development</name>
</course>
```
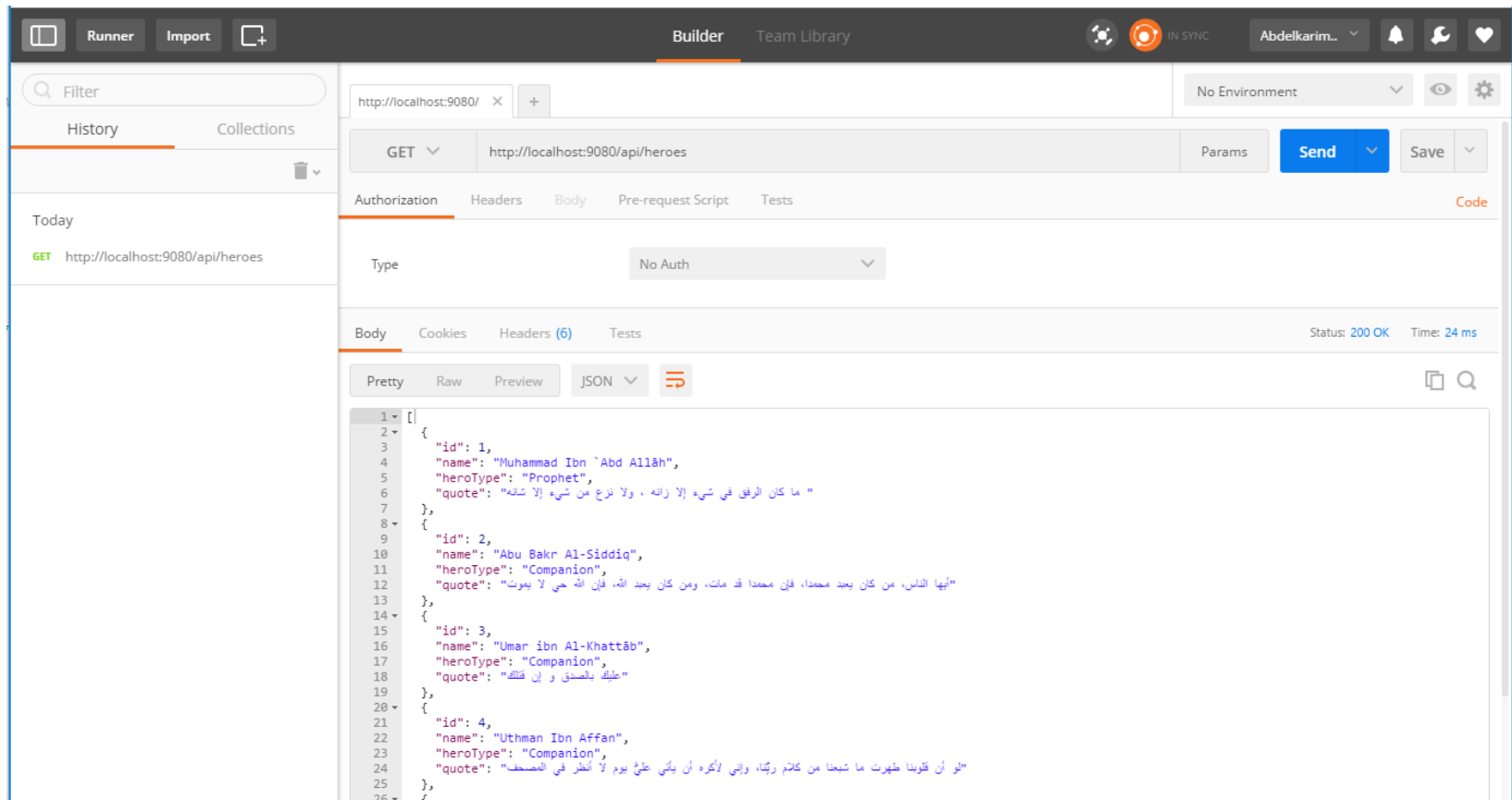
HTTP Verbs

- Represent the actions to be performed on resources

- Retrieve a representation of a resource: **GET**

- Create a new resource:
  - Use **POST** when the server decides the new resource URI
    - Post is not repeatable
  - Use **PUT** when the client decides the new resource URI
    - Put is repeatable

- **PUT** is typically used for update

- Delete an existing resource: **DELETE**

- Get metadata about an existing resource: **HEAD**

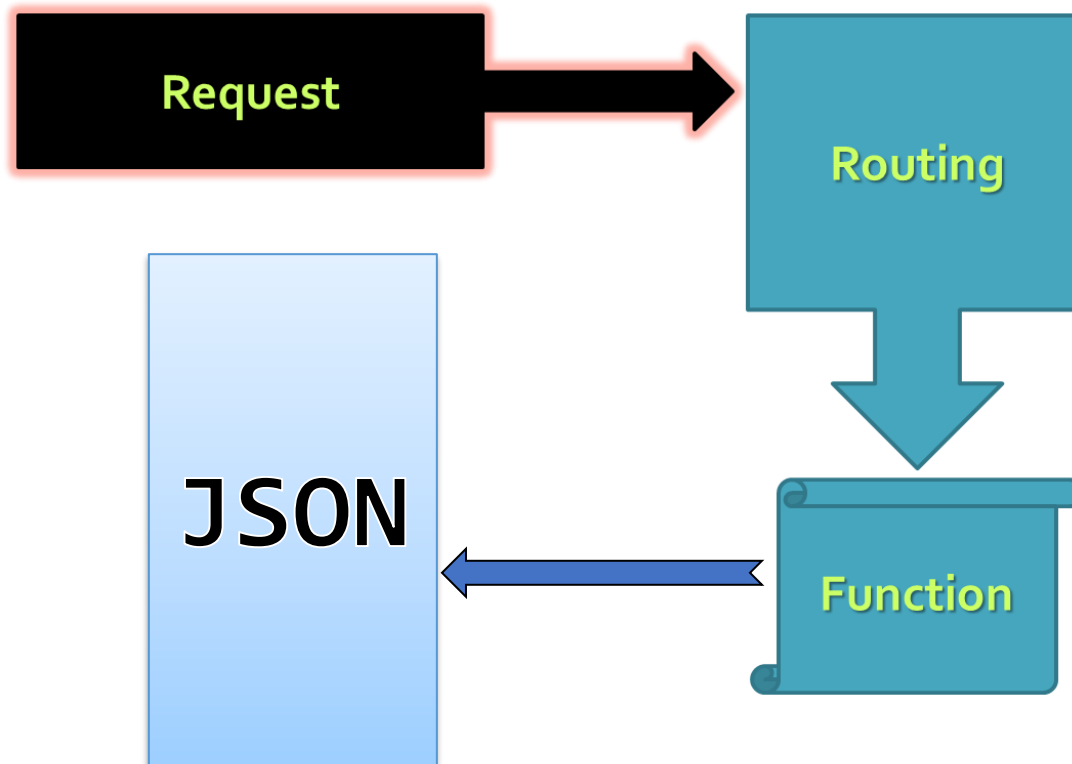- Get which of the verbs the resource understands: **OPTIONS**

Testing REST Services

- Using Postman to test Web API

https://www.getpostman.com/postman

Web API using NEXT.js

Request → Routing

Routing → Function

Function → JSON

Getting started

- Create an empty folder (with no space in the name use **dash -** instead)

- Create next.js app (select **No** for all questions)

  `npx create-next-app@latest .`

```
√ Would you like to use TypeScript with this project? ... No / Yes
√ Would you like to use ESLint with this project? ... No / Yes
√ Would you like to use `src/` directory with this project? ... No / Yes
√ What import alias would you like configured? ... @/*
```

- Creates a new **Next.js** project and downloads all the required packages

- Run the app in dev mode: **npm run dev**

Next.js Routing
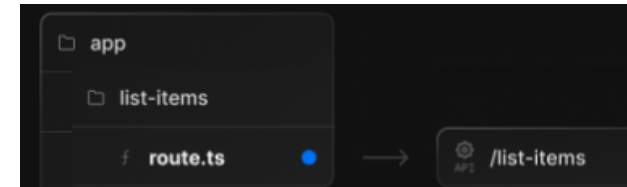
- Next.js has a **file-system based App Router**:
  - Folders inside the **app** directory are used to define routes
    - A route is a single path of nested folders, from the root folder down to a leaf folder
  - Files are used to create Web pages (`page.js`) or Web API (`route.js`)

- Each folder in the subtree represents a route segment in a URL path

- E.g., create `/dashboard/settings` route by nesting two subfolders in the app directory
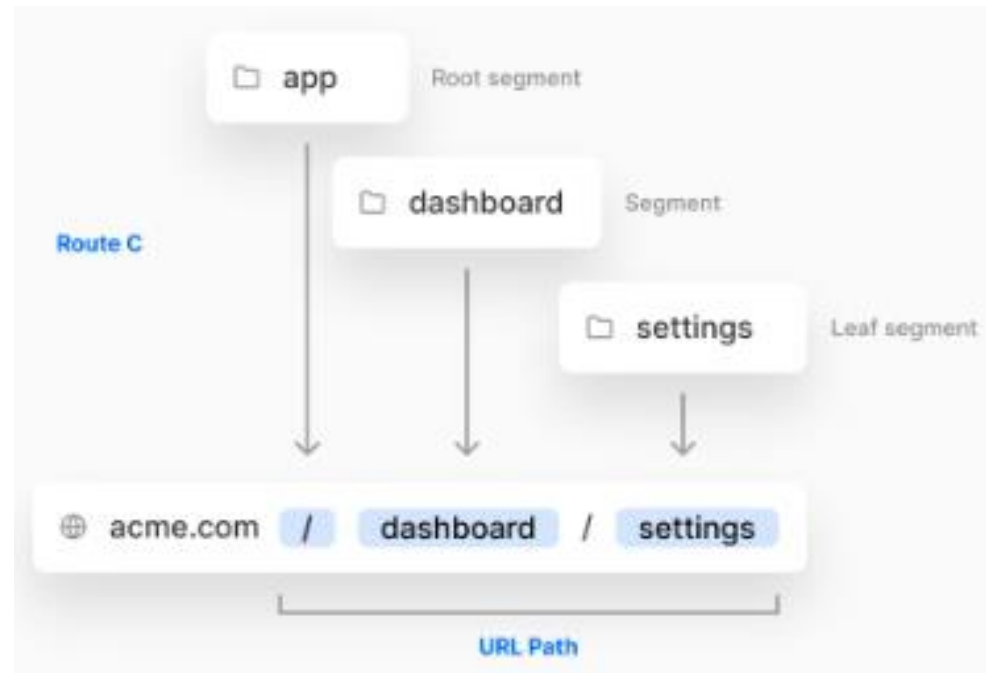
API Routes

- Add a **route.js** file under the **app** folder or within subfolders to define API routes

- A route.js file can export async functions named after HTTP methods (GET, HEAD, OPTIONS, POST, PUT, DELETE, PATCH) to handle requests

- Any subfolder within app containing a route.js file is treated as a Web API endpoint (e.g., app/api/hello/route.js).

```
export async function GET(request) {
    return new Response('Hello, Next.js!');
}
```

- Visiting http://localhost:3000/api/hello will return Hello, Next.js!

Routing in Next.js

- Requests can be routed based on:
  - ➢ **HTTP Verbs**: GET, POST, PUT, DELETE
  - ➢ **URL Paths**: e.g., /users

- An App Route maps an HTTP Verb (e.g., GET, POST) and a URL path (e.g., /users/123) to a **route handler function**
  - ➢ The handler function receives a **request** object and returns a **response** object
  - ➢ The **request** object allows extracting data, such as the request body
  - ➢ The **response** object represents the HTTP response and is used to send the generated output

Dynamic API Routes

- To create a dynamic route (having named path parameters) simply wrap the folder's name in square brackets **[folderName]**

  o Allows adding **path parameters** to the URL path. E.g., **/blogs/123**

Route	Example URL	params
app/blogs/[id]/route.js	/blog/123	{ id: '123' }
app/blogs/[id]/route.js	/blog/234	{ id: '234' }

- Dynamic segments are passed as **params** argument to the handler functions

  o E.g., if you have the path **/blogs/[id]**, then the "id" property is available as **(await params).id**

```js
// app/blogs/[id]/route.js
export async function GET(request, { params }) {
  const id = (await params).id;
  return new Response(`Blog id# ${id}`)
}
```

Catch-all dynamic routes

- **catch-all dynamic routes**: allows a dynamic route to catch all paths by simply adding ellipsis(...) inside the brackets [...folderName]

  - e.g., The catch all page in **app/blogs/[...filterBy]** will match any path underneath **/blogs** such as: /blogs/2025, /blogs/2025/3/10, and so on

  - Matched parameters array can be access using the **params**, so the path /blogs/2025/3/10 will have the following **params** object ["2025", "3", "10"]

Route	Example URL	params
app/blogs/[...filterBy]/route.js	/blogs/2023	{ filterBy: ['2023'] }
app/blogs/[...filterBy]/route.js	/blogs/2023/3	{filterBy: ['2003', '3']}

Optional Catch-all Segments

- Catch-all Segments can be made optional by including the parameter in double square brackets: **`[[...folderName]]`**

- For example, **`app/shop/[[...slug]]/route.js`** will also match `/shop`, in addition to `/shop/clothes`, `/shop/clothes/tops`, `/shop/clothes/tops/t-shirts`

- The difference between catch-all and optional catch-all segments is that with optional, the route without the parameter is also matched (`/shop` in the example above).

Query Parameters

- Named **query parameters** can be added to the URL path after a **?** E.g., **/products?sortBy=price**

- Query parameters are often used for **optional** parameters (e.g., optionally specifying the property to be used to sort of results)

- **request.nextUrl.searchParams** is an object containing a property for each query parameter in the URL path

  - If you have the path /products**?sortBy**=**price**, then the "**sortBy**" property can accessed as shown below:

```
export async function GET(request) {
    const { searchParams } = request.nextUrl;
    const sortBy = searchParams.get("sortBy") || "default";
    const res = await fetch(`https://api.com/products?sortBy=${sortBy}`);
    const products = await res.json();
    return new Response(JSON.stringify(products), { status: 200 });
}
```

Working with a Request Body

- The request body can be retrieved using one of the following request methods:

  .json() , **.text()** or **.formData()**

```
export async function POST(request) {
    let hero = await request.json()
    hero = await addHero(hero)
    return Response.json(hero, { status: 201 })
}
```

Headers

- You can read http headers with the **headers** library from next/headers package

- You can also return a new Response with new headers

```
import { headers } from "next/headers";

export async function GET(request) {
  const headersList = await headers();
  const apiKey = headersList.get("apiKey");

  return new Response("Hello, Next.js!", {
    status: 200,
    headers: { apiKey: apiKey || "No API Key" },
  });
}
```

Redirect

- Sends a redirect response to another Url

```
import { redirect } from 'next/navigation'

export async function GET(request) {
  redirect('https://nextjs.org/')
}
```

Summary

- Next.js = React-based full stack web framework that allows creating server-side rendered pages, and Web API

- Next.js has a **file-system based router**: when a folder is added to the **app** directory, it's automatically available as a route

  - In Next.js you can add brackets to the folder name to create a dynamic route

- Add Web API Route Handlers inside the app/ directory (e.g., app/api/users/route.js)

Summary (continued…)

- Build a public API if it need

  o to be shared by web, mobile, or third-party clients to consume your data/functionality

  o to proxy a backend /external service and include secret authentication headers

- Export HTTP methods (GET, POST, PUT, DELETE, etc.) in the route.js file

- Use Web Standard APIs to interact with the [Request](#) object and return a [Response](#)

- Fetch the API routes from the client with fetch('/api/…')

Resources

- Learn Next.js

https://nextjs.org/docs

- Next.js blog

https://nextjs.org/blog

- Next.js Web API

https://nextjs.org/blog/building-apis-with-nextjs