Week 6 - API

Dr Kiki Adhinugraha

Lecture 6

- What is Backend?
- Know how routing in express nodejs works
- What is the advantage in having a rest API?

The Three Layers of the Web

- **HTML for Content**
  - There is no download link on this page.

- **CSS for Presentation**
  - .warning { color: red; }

- **JavaScript for Behavior**
  - <script> window.alert(document.getElementsByClassName("warning")[0].innerHTML);
  - </script>
  - React: JavaScript Library for building user interfaces
  - Redux: A new kind of architecture that complements React and the concept of unidirectional data flow

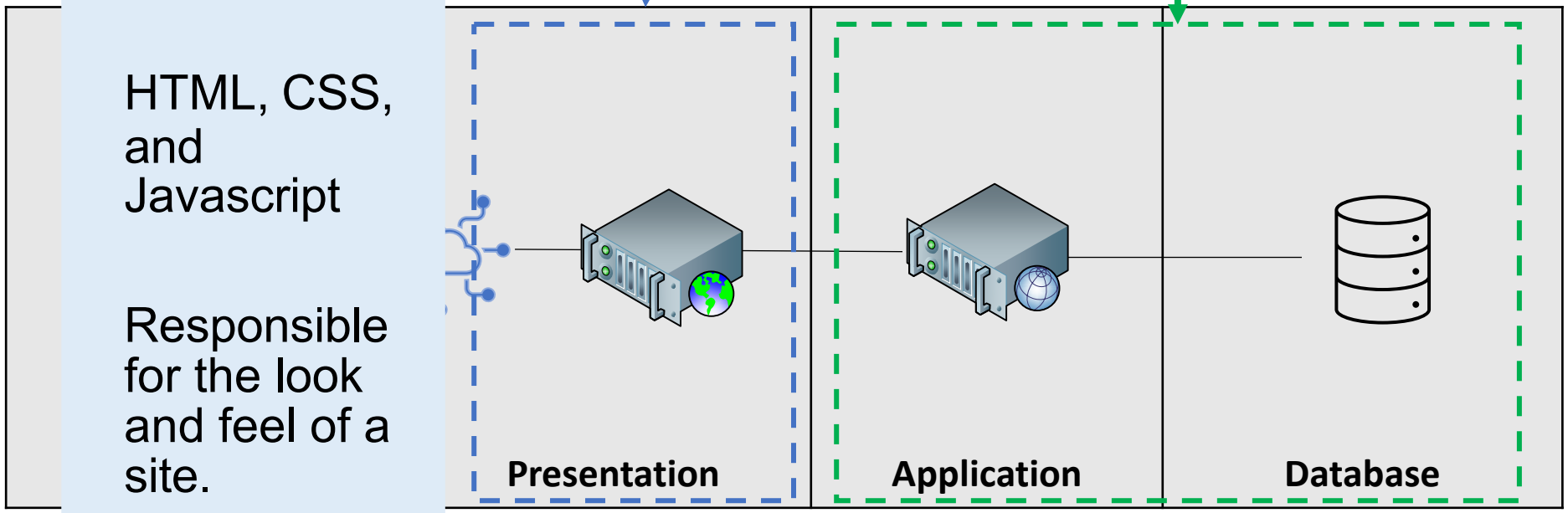
Front End VS Back End

Front End

Manages everything that users visually see first in web sites

HTML, CSS, and Javascript

Responsible for the look and feel of a site.



Back End

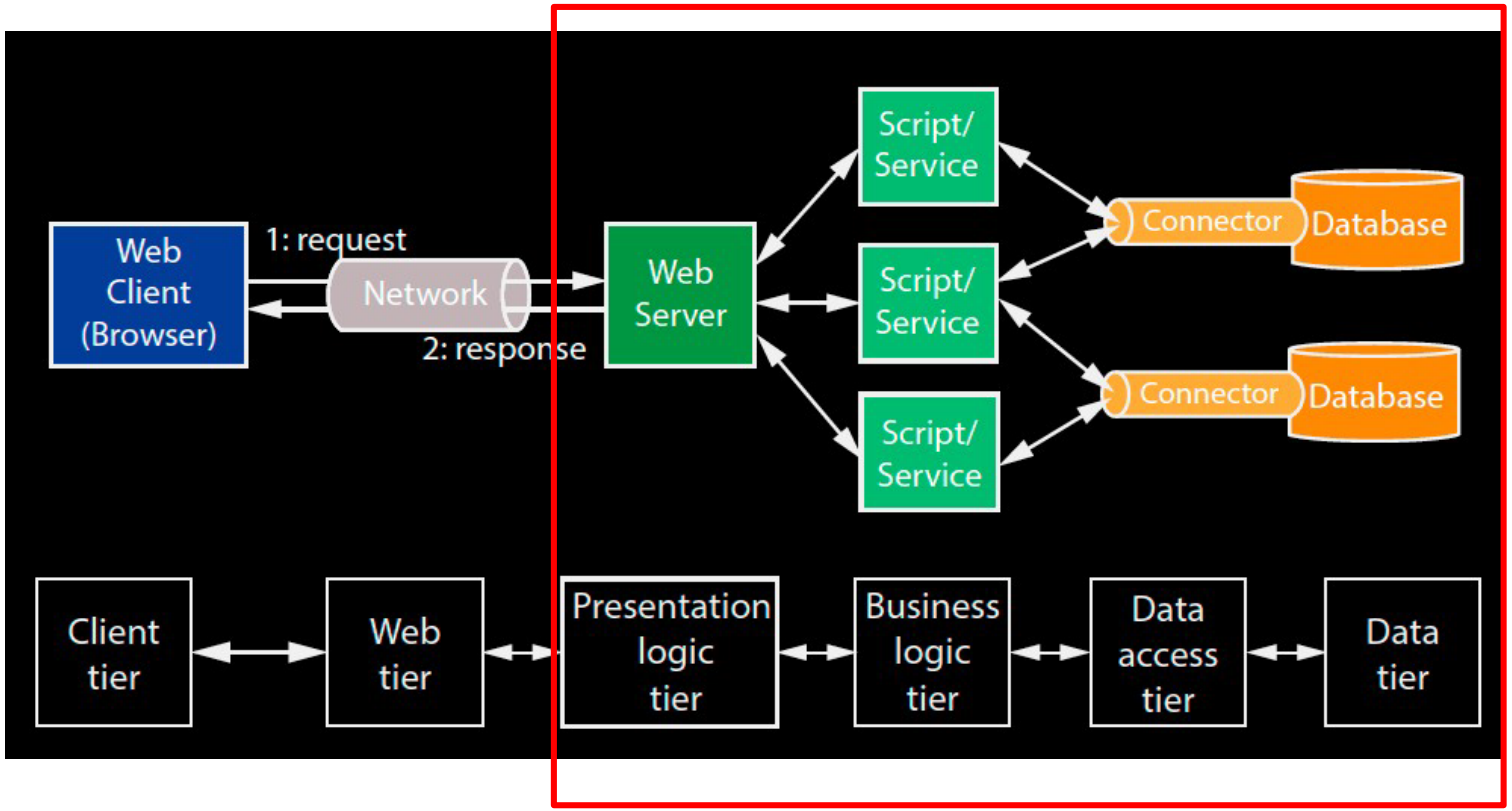
Refers to the server side of a website and everything that communicates between database and the browser.

Java, PHP, Ruby on Rails, Python, and .Net

Responsible for the responsiveness and speed of a site

The Back End

- We will now turn our attention to the backend
- There are lots of choices for backends such as
  - Ruby, PHP, java, express node js, etc.
- In this course we will use a backend based on express node js



https://www.youtube.com/watch?v=4r6WdaY3SOA

What is Node.js?

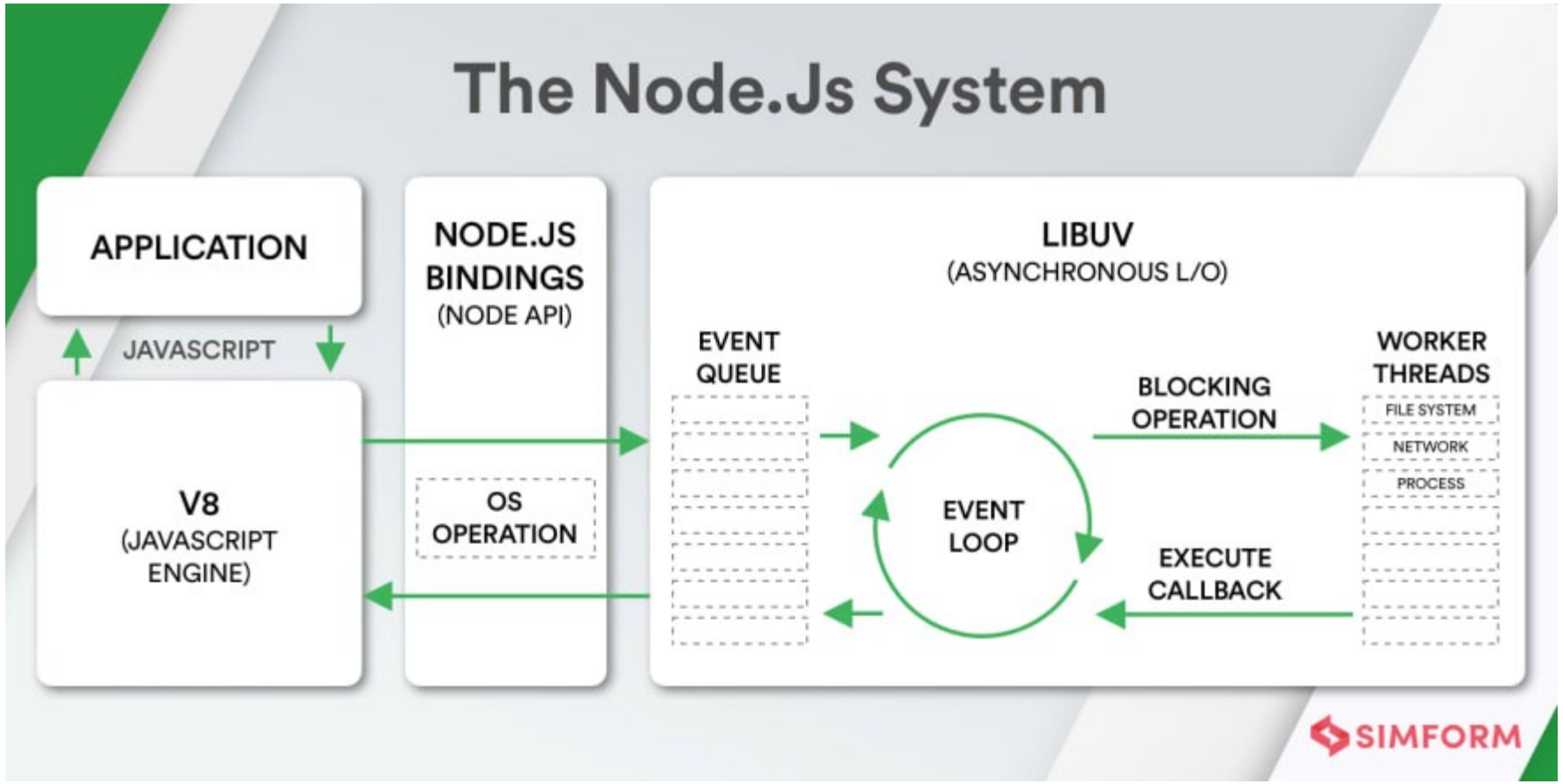
- A server side platform built on the JavaScript engine.
- A cross-platform runtime environment for developing server-side and network applications.
  - So it can be used for any kind of network application
  - does not need to be a web application

- Let's look at this video for quick impression:
- https://www.youtube.com/watch?v=uVwtVBpw7RQ

Important Features of **Node.js**

- **Asynchronous and event driven**
  - Node.js server never waits for an API (application interface) to return data.
  - Server moves to next API call
  - Events are used to get response from previous API calls
- **No Buffering**
  - Node.js simply output the data in chunks
- **Node.js is single threaded**
  - Each node server runs on a single thread
  - To achieve concurrency you just spin up more node servers

How Node.js works?



Node.js vs other server-side scripting

An example how to handle file on the server

ASP / PHP

- Sends the task to the computer's file system.
- Waits while the file system opens and reads the file.
- Returns the content to the client.
- Ready to handle the next request.

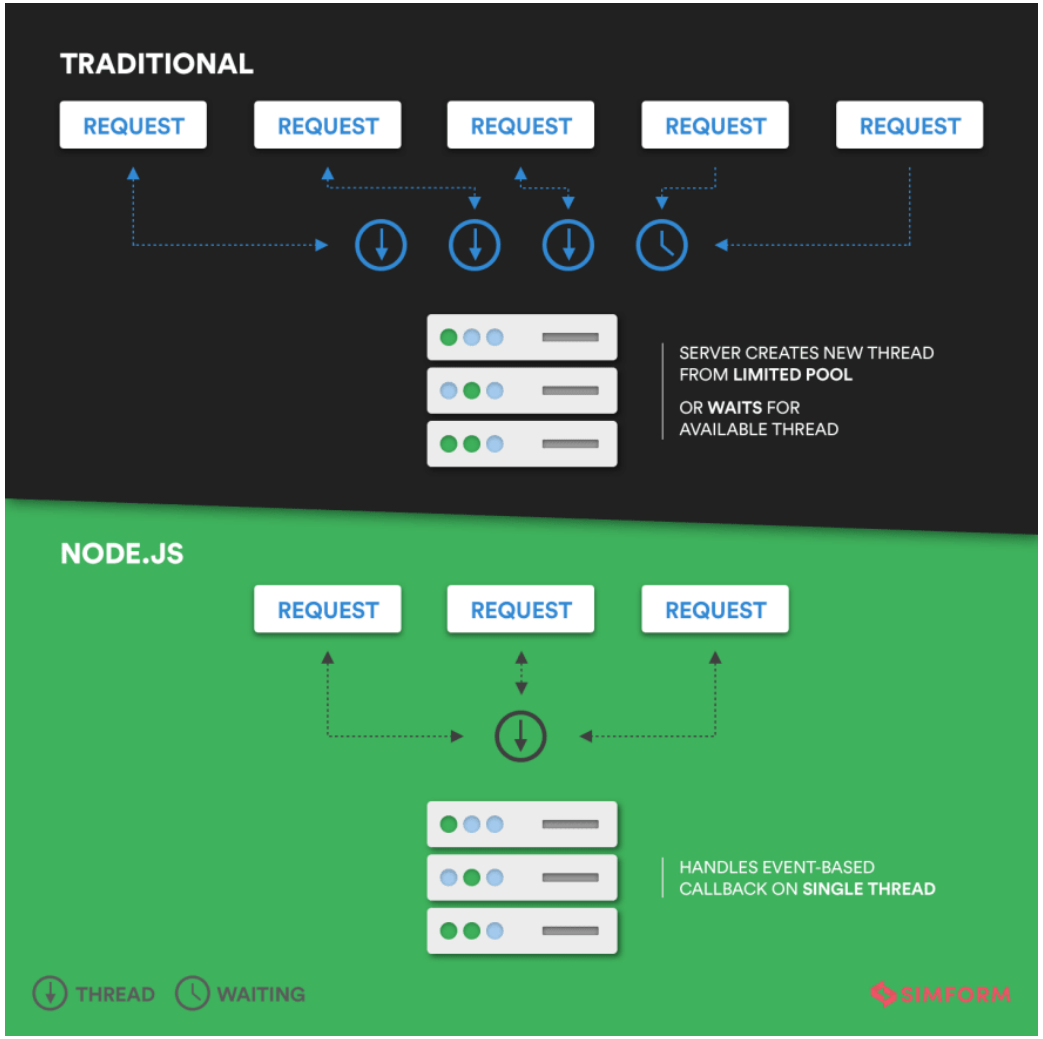
Node.js

- Sends the task to the computer's file system. Ready to handle the next request.
- When the file system has opened and read the file, the server returns the content to the client.

- Node.js eliminates the waiting, and simply continues with the next request.
- Node.js runs single-threaded, non-blocking, asynchronously programming, which is very memory efficient.

Node.js vs other server-side scripting

- Traditional server-side scripting will always wait until the request is fulfilled
- This will cause the service cannot be used while waiting
- Node.js allows multiple requests at a time as the request is handled and managed on a single thread



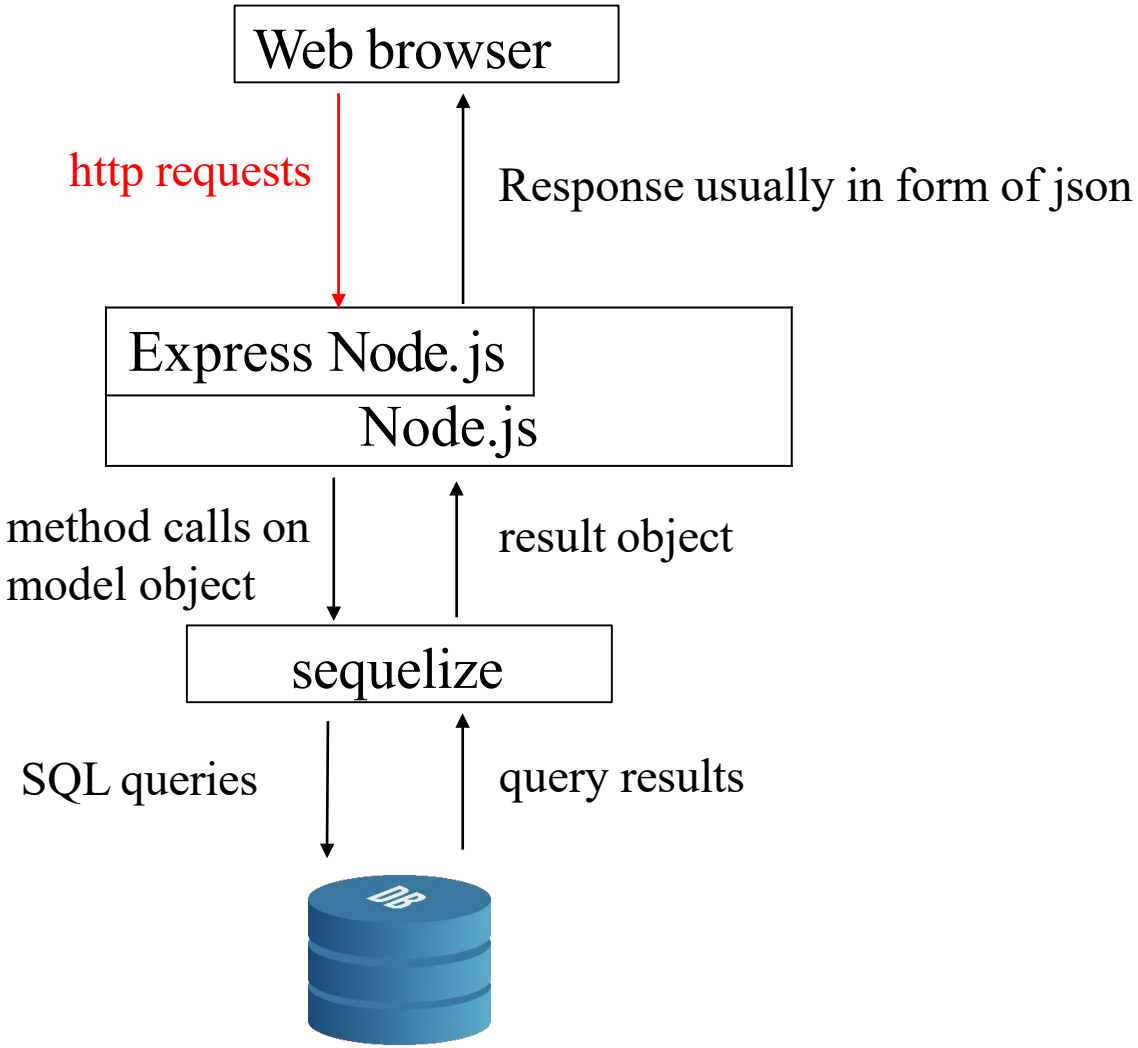
What is Express Node.js?

- Express is a small framework that sits on top of Node.js's web server functionality to simplify its APIs and add helpful new features.
- A minimal and flexible node.js web application framework
  - So it is like a set of library calls that sits on top of node.js
- Minimal
  - Can still do everything you want but via a smaller API
- Flexible
  - Allow you to plug and play different functionality.

Why choose Express node.js for the backend?

- It is all in javascript!
  - Don’t need to learn another programming language
- Becoming very popular
- Relatively simple to use
- Platform independent
  - Microsoft Windows, Linux, OS X, etc.

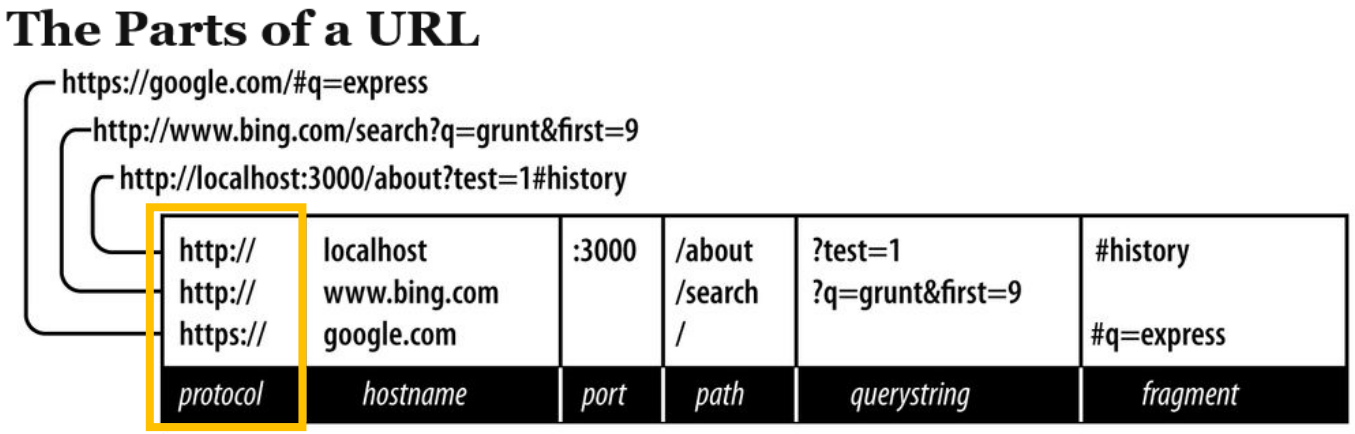
So what happens when a request comes from the web browser?



- Express Node.js routes the request
  - It basically looks at the URL and then figures out what method to call to handle the request

How Express node.js routes the HTTP requests?

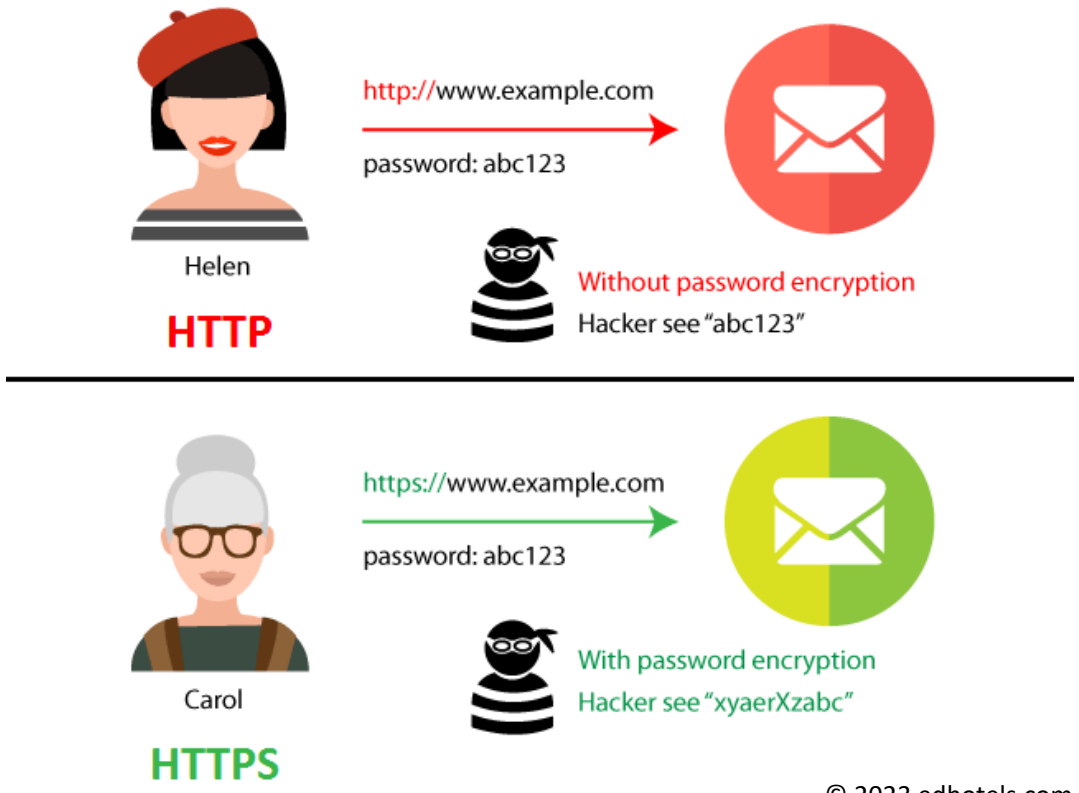
Routing (parts of URL)



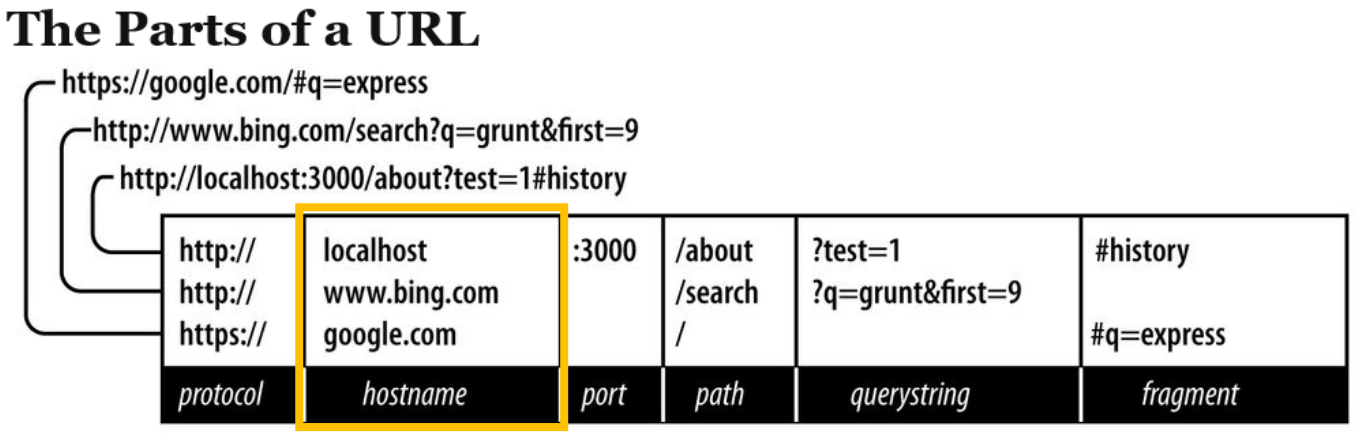
- Protocol
  - The protocol determines how the request will be transmitted.
  - HTTP is a protocol that enables data transmission via the world wide web
  - HTTPS is essentially a more secure version. HTTPS uses SSL/TLS to encrypt connections between web browsers and servers.

Why HTTPS?

- Without HTTPS, any data passed is insecure.
- Without encryption, all packages exchanged in a web application will be visible to public
- Users can identify whether a site uses HTTPS protocol by the web address

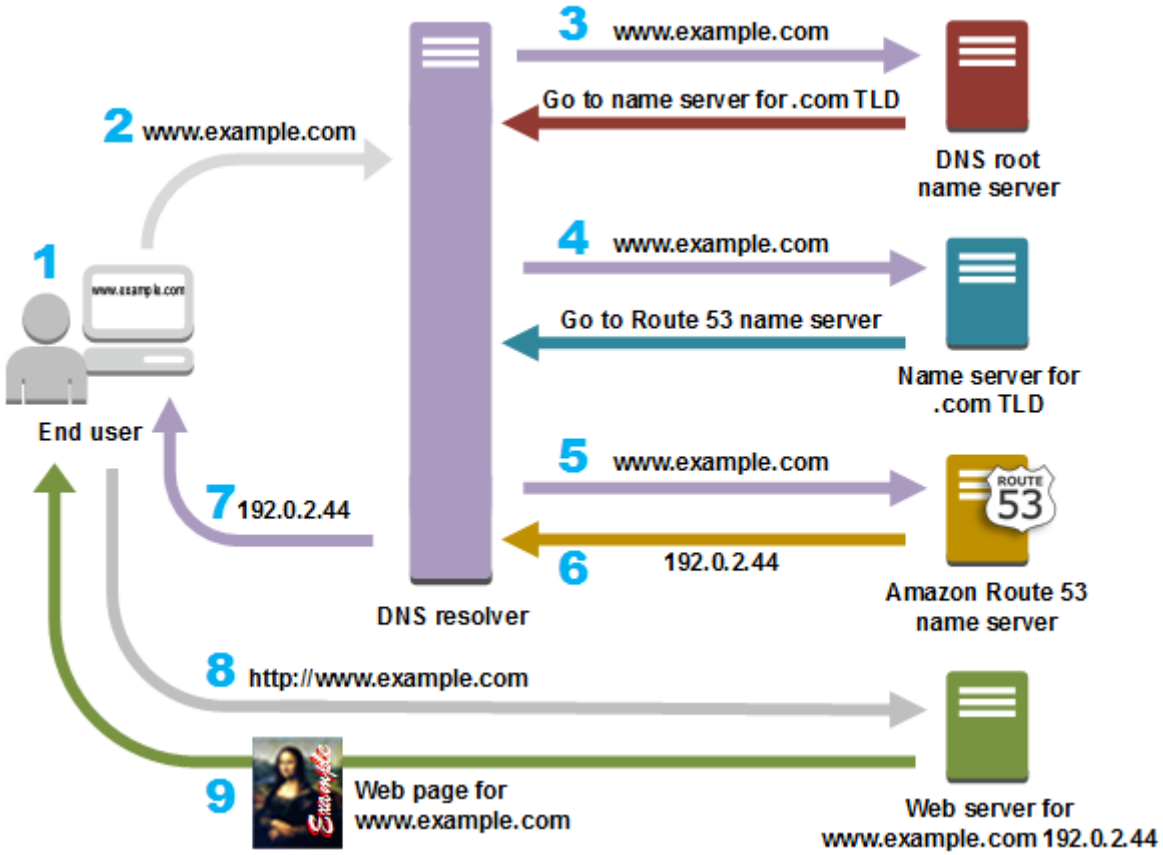


Routing (parts of URL)

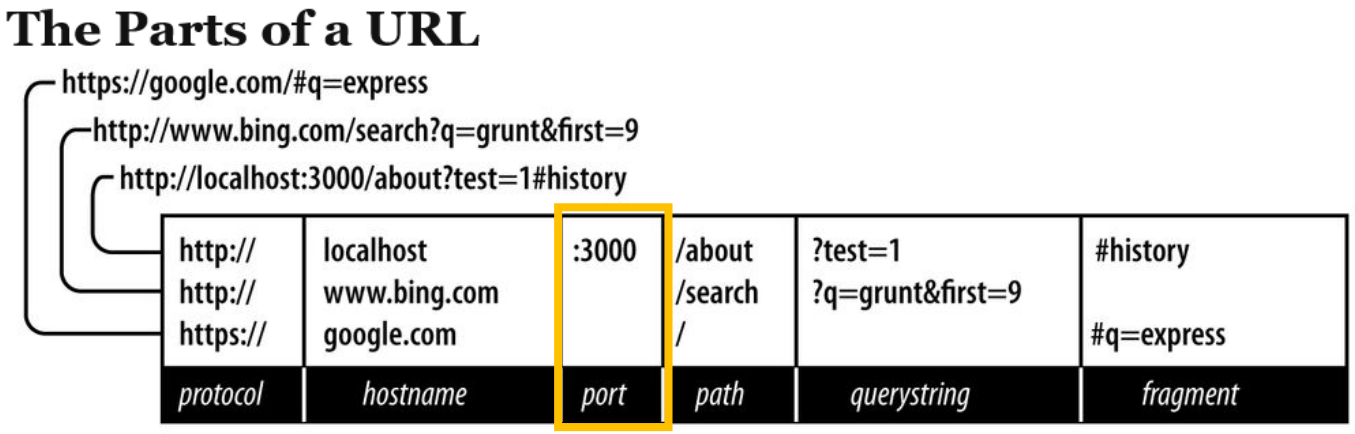


- Host
  - The host identifies the server.
  - E.g. localhost (your computer), single word, can also be numeric IP address
  - The servername is resolved by DNS (Domain Name System)
  - DNS translates human readable domain names (for example, www.amazon.com) to machine readable IP addresses (for example, 192.0.2.44)

How DNS works in AWS platform

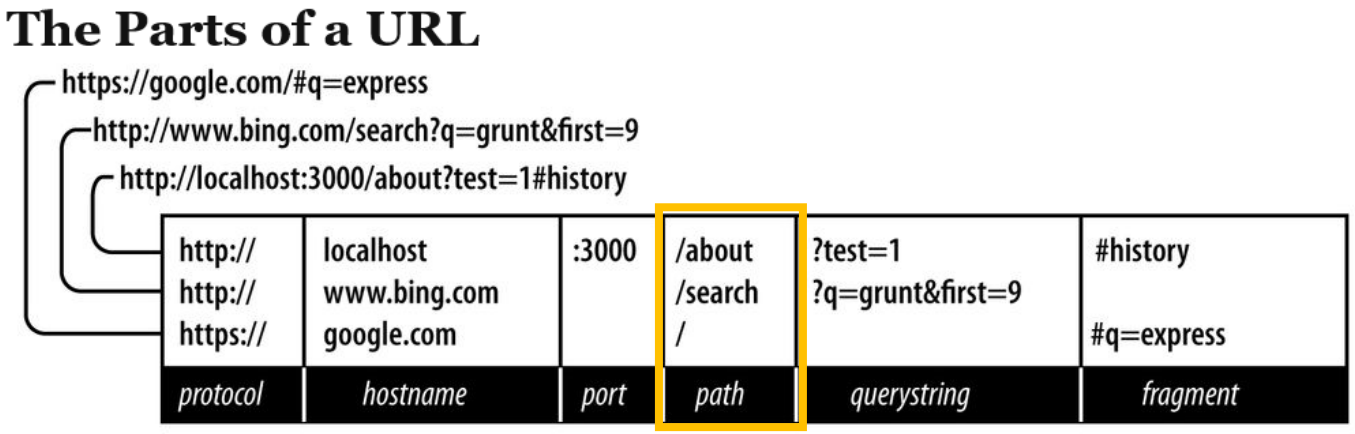


Routing (parts of URL)



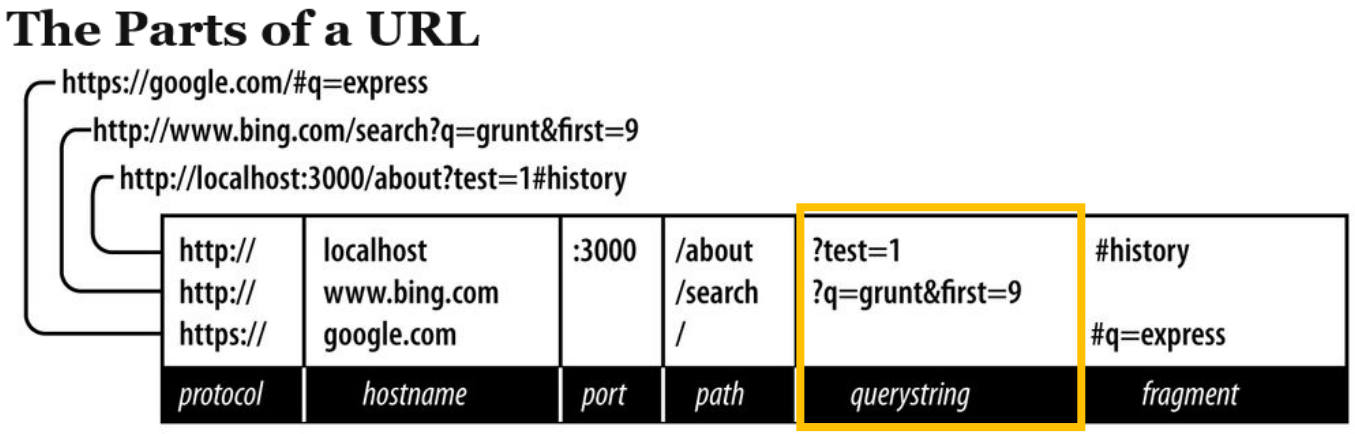
- Port
  - Ports are software-based and managed by a computer's operating system.
  - Each server has a collection of numbered ports.
  - Each port is associated with a specific process or service.
  - Ports allow computers to easily differentiate between different kinds of traffic: emails go to a different port than webpages, for instance, even though both reach a computer over the same Internet connection.
  - By default the port 80 is used for http requests.
- You can imagine the port as the door apartment or unit number, where hostname is the street number

Routing (parts of URL)



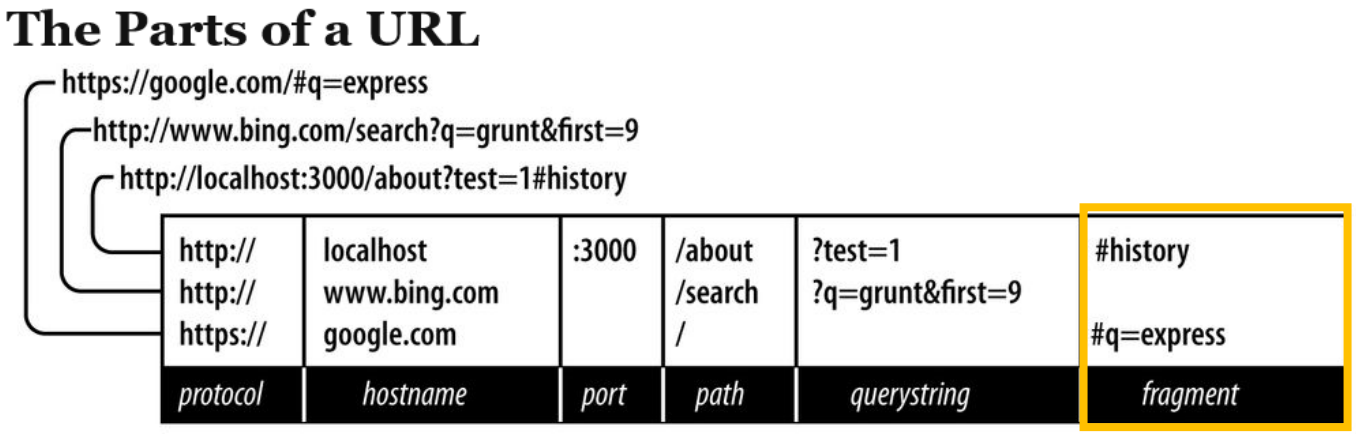
- Path
  - The path should be used to uniquely identify pages or other resources in your app
  - The path is similar to the folder in your drive.
  - / indicates the root of the host or server

Routing (parts of URL)



- Querystring
  - Optional collection of name/value pairs
  - Querystring starts with a question mark (?)
  - Name/value pairs are separated by ampersands (&)

Routing (parts of URL)



- Fragment
  - Not passed to the server at all
  - Used only by the browser
  - Used for single-page applications or AJAX heavy applications
  - Used as an anchor tag in order to display a specific part of the document

HTTP Request Method

- HTTP defines a set of request methods to indicate the desired action to be performed for a given resource.
- List of HTTP Methods:
  - GET
  - POST
  - PUT
  - DELETE
  - HEAD
  - PATCH
  - OPTIONS
  - CONNECT
  - TRACE

The GET Method

- GET is used to request data from a specified resource.
- Note that the query string (name/value pairs) is sent in the URL of a GET request:

```
/test/demo_form.php?name1=value1&name2=value2
```

- Some notes on GET requests:
  - GET requests can be cached
  - GET requests remain in the browser history
  - GET requests can be bookmarked
  - GET requests should never be used when dealing with sensitive data
  - GET requests have length restrictions
  - GET requests are only used to request data (not modify)

The POST Method

- POST is used to send data to a server to create/update a resource.
- The data sent to the server with POST is stored in the request body of the HTTP request:

```
POST /test/demo_form.php HTTP/1.1
Host: w3schools.com

name1=value1&name2=value2
```

- Some notes on POST requests:
  - POST requests are never cached
  - POST requests do not remain in the browser history
  - POST requests cannot be bookmarked
  - POST requests have no restrictions on data length

The PUT Method

- PUT is used to send data to a server to create/update a resource.
- The difference between POST and PUT is that PUT requests are idempotent. That is, calling the same PUT request multiple times will always produce the same result.
- In contrast, calling a POST request repeatedly have side effects of creating the same resource multiple times.

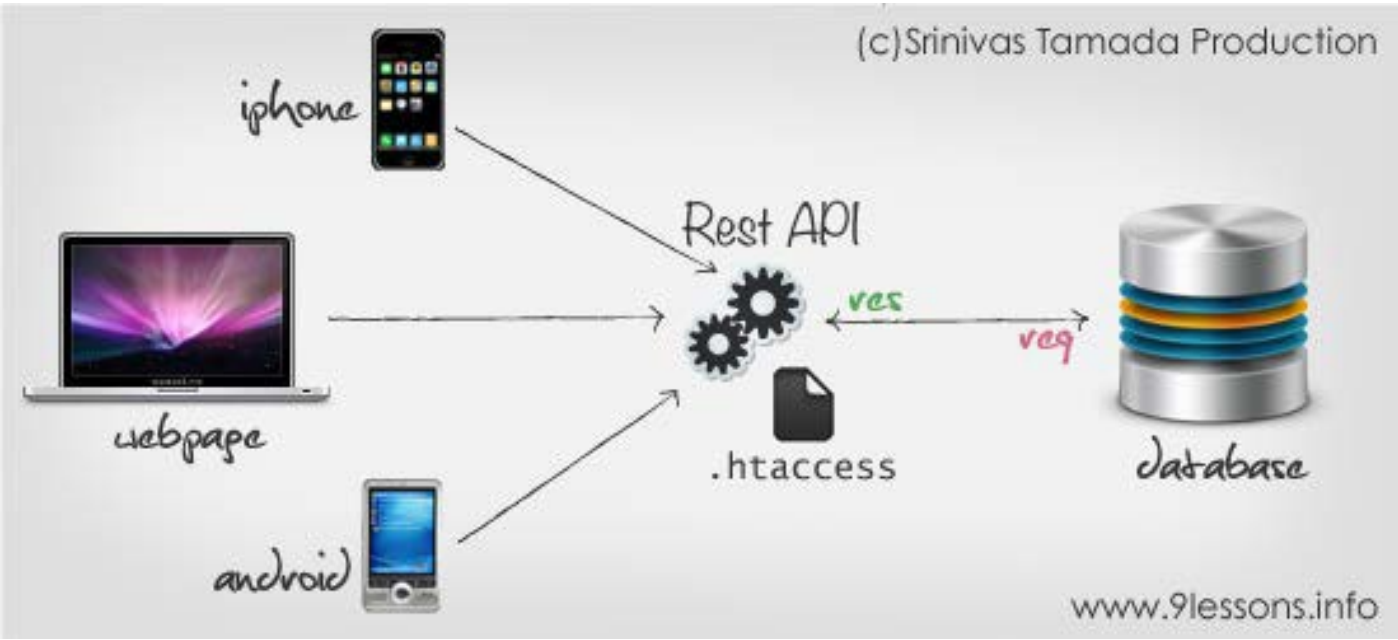
The DELETE Method

- The DELETE method deletes the specified resource.
- DELETE will not throw exception/error if the object cannot be found

Representational state transfer (REST)

- A software architectural style that was created to guide the design and development of the architecture for the World Wide Web.
- REST defines a set of constraints for how the Web, should behave.
- Architectural Constraints
  - Client–server architecture
  - Statelessness
  - Cacheability
  - Layered system
  - Code on demand
  - Uniform interface

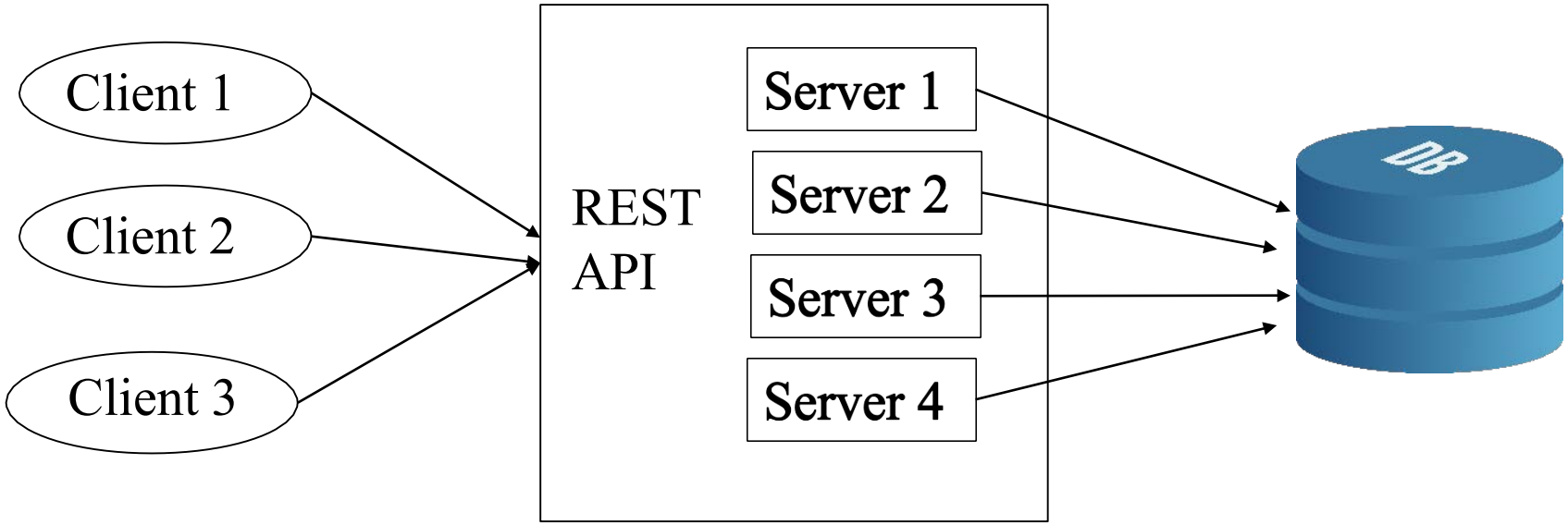
REST API



- A RESTful API is an application program interface (API) that uses HTTP requests to GET, PUT, POST and DELETE data.
- Above is a typical example of a REST API
  - It shows a set of requests from the browser that can be used to display and edit a list of tasks.
  - It makes extensive use of the HTTP methods, such as GET, POST, PUT and DELETE.
- REST API has become very popular because it separates backend and front services
- This helps decouple the application
- This allows multiple clients e.g. Web Site, IOS app and Android app to use the same API

https://www.youtube.com/watch?v=6sUbt-Qp6Pg

REST API is Stateless



- No Client state at server.
  - The benefit is
    - The client can use a different server on every request
    - Very scalable
- Any state is maintained at the client side
- Each request has all the information to process request

REST API HTTP Request Methods and Arguments

HTTP method	Path	Name	Description
GET	/posts	Index	List posts
POST	/posts	Create	Create a new posts
GET	/posts/:id	Show	Show single post
DELETE	/posts/:id	Destroy	Destroy existing post
PUT	/posts/:id	Update	Update existing post

- You will use these methods in your labs and assignments
- :id is the number of existing post

HTTPie

- To test the functionality of API, we will use HTTPie (https://httpie.io/)
- HTTPie (pronounced aitch-tee-tee-pie) is a command-line HTTP client. Its goal is to make CLI interaction with web services as human-friendly as possible.

```
wdc@wdc-VirtualBox:~/Documents/lab 07/try/lab07-blog$ http GET localhost:3001/posts/6
HTTP/1.1 200 OK
Connection: keep-alive
Content-Length: 133
Content-Type: application/json; charset=utf-8
Date: Sat, 27 Aug 2022 02:56:36 GMT
ETag: W/"85-KREobMu0ylXZL0jwDnJOpA8X908"
X-Powered-By: Express

{
    "content": "World domination cat",
    "createdAt": "2022-08-27T02:49:08.000Z",
    "id": 6,
    "title": "WDC",
    "updatedAt": "2022-08-27T02:49:08.000Z"
}
```

HTTP Request Methods

http GET localhost:3000/posts/1

http request method

http POST localhost:3000/posts/1 title="Rings" author ="Sauron"

- The above example is using the HTTPie tool hence it does not look exactly like an URL but you can see all the important elements of the request in it.
- When you type URL into a web browser the browser issues an HTTP GET request
- POST requests are usually reserved for sending information back to the server and used in forms
  - <form action="localhost:3001/myaction" method="post">

HTTP Request Methods

http GET localhost:3000/posts/1

http request method

http POST localhost:3000/posts/1 title="Rings" author="Sauron"

- GET requests:
  - GET is used to request data from a specified resource.
  - GET is one of the most common HTTP methods.
- POST requests:
  - POST is used to send data to a server to create/update a resource.
  - The data sent to the server with POST is stored in the request
  - body of the HTTP request.

HTTP Request Methods

http GET localhost:3000/posts/1

http request method

http POST localhost:3000/posts/1 title="Rings" author="Sauron"

- You can still pass information to a server via a get request although this is not preferred when using the REST API (we will talk about this soon).
  - <form action="localhost:3001/myaction" method="get">
    - When using GET, the data that you pass to the server will be visible

      localhost:3001/myaction?firstname=Mickey&lastname=Mouse
- Using POST the submitted data is not visible on the page address

Routing

- Routing is the mechanism by which requests (as specified by a URL and HTTP method) are routed to the code that handles them.
- Here is an example where for any path other than /test the server will respond with a 404 Not Found.

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

// Handles requests on path /test and sends the Hello World! message to the server
app.get('/test', (req, res) => {
  res.send('Hello World!');
});

// The server listens on port 3000 for connections
app.listen(3000, function() {
   console.log('example app listening on port 3000!');
}
```

Routing(How does Express Node.js handle a request?)

http GET localhost:3000/test

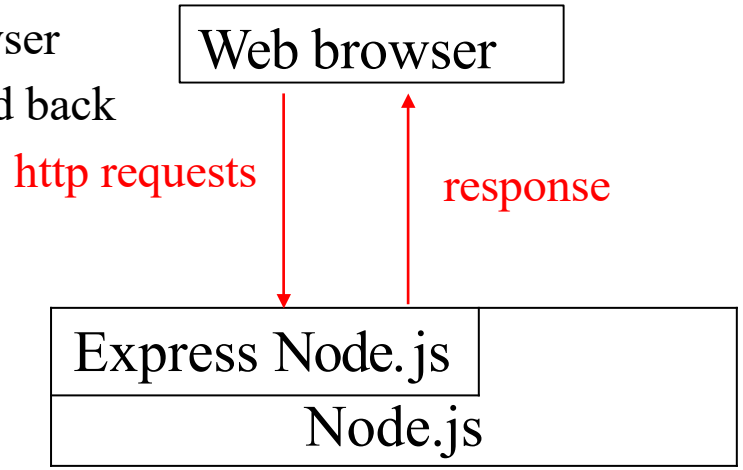
request object received from browser

response object send back to the client

```
app.get('/test', (req, res) => {
    res.send('Hello World!');
});
```

Output on browser is:

Hello World!



- The above method will be invoked when the server receives a GET request on the path /test.
- It sends back the Hello World message as a string to be displayed by the browser.

How do we get a parameter passed in by the user?

http GET localhost:3000/user/1

```
app.get('/user/:id', (req, res) => {
    res.send('user id is ' + req.params.id);
});
```

Output on browser is:

user id is 1

- Reads a parameter from the path of the request.
- In the above example the methods extracts the user id parameter and sends it back to the browser via a string.

What is JSON?

- JSON stands for **J**ava **S**cript **O**bject **N**otation.
- JSON example

```
{"employees":[
    {"firstName":"John", "lastName":"Doe"},
    {"firstName":"Anna", "lastName":"Smith"},
    {"firstName":"Peter", "lastName":"Jones"}
]}
```

- It is a very convenient way of passing information from browser to server and vice versa
- The above example encapsulates an array of employees in a JSON object
  - Each employee has a firstName and lastName attribute

How do we get a JSON parameter passed in by the user in a post parameter?

```
$ http POST localhost:3000/user/1 title=Javascript  author=Peter
```

```
const myParser = require("body-parser");
app.use(myParser.urlencoded({extended : true})
app.post('/user/:id', (req, res) => {
   res.send('user id is ' + req.params.id + ', title is' + req.body.title
             + ', author is '+ req.body.author);
});
```

Output on browser is:
user id is 1, title is Javascript, author is Peter

- This takes the json formatted data from the browser that is send via the POST method to node.js and then creates a string containing the json information and sends it back to the browser.

Other Request Object Methods

- **req.query**
  - An object containing querystring parameters (sometimes called GET parameters) as name/value pairs
    - localhost:3000/user/1?name=peter
- **req.headers**
  - The request headers received from the client
  - HTTP headers are the core part of these HTTP requests and responses, and they carry information about the client browser, the requested page, the server and more.
- **req.ip**
  - The IP address of the client
- **req.path**
  - The request path (without protocol, host, port or querystring)
- **req.host**
  - Hostname reported by the client
- Etc.

Req.headers

Headers | Response | Cache | Cookies

Response Headers

Date	Mon, 30 Nov 2009 01:30:37 GMT
Server	LiteSpeed
X-Powered-By	W3 Total Cache/0.8
Pragma	public
Expires	Mon, 30 Nov 2009 02:30:37 GMT
Etag	"pub1259544156;gz"
Cache-Control	max-age=3600, public
Content-Type	text/html; charset=UTF-8
Last-Modified	Mon, 30 Nov 2009 01:22:36 GMT
X-Pingback	http://net.tutsplus.com/xmlrpc.php
Content-Encoding	gzip
Vary	Accept-Encoding, User-Agent

Request Headers

Host	net.tutsplus.com
User-Agent	Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US; rv:1.9.1.5) .5.30729)
Accept	text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q
Accept-Language	en-us,en;q=0.5
Accept-Encoding	gzip,deflate
Accept-Charset	ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive	300
Connection	keep-alive
Cookie	_bsau=12595308253191404821; __utma=112694043.1672996432.125 ; __utmz=112694043.1259530826.1.1.utmccn=(direct)|utmcsr=(d ; __utmb=112694043
If-Modified-Since	Mon, 30 Nov 2009 01:22:36 GMT
If-None-Match	"pub1259544156;gz"
Cache-Control	max-age=0

Lets look at the Response Object

```
$ http POST localhost:3000/user/1 title=Javascript author=Peter
```

```
app.post('/user/:id', (req, res)
  => { res.json(req.params.body);
});
```

- The response object is used to return the response back to the browser.
- In this case we just return the json object that we receive from the browser back to it.

Other Response Object Methods

- **res.status(code)**
  - Sets the HTTP status code
  - Default is 200 (OK)
  - 404 (Not Found)
  - 500 (Server Error)
- **res.redirect([status], url)**
  - Redirects the browser
  - Optional status, defaults to 302 (found)
- **res.send(body), res.send(status, body)**
  - Sends a response to the client, with an optional status code.
  - Defaults to type text/html
  - If you want text/plain then you need to use
    - res.set(‘Content-Type’, ‘text/plain’) before calling res.send
  - If body is an object or array then a JSON will be returned.
    - Better to use res.json instead

Other Response Object Methods

- res.json(json), res.json(status, json)
  - Sends JSON to the client with an optional status code
- res.attachment([Filename]), res.download(path, [filename], [callback])
  - Both of these methods set a response header called Content-
  - Disposition to attachment.
- Etc.

Lab Activity

- In the lab, you will design and create REST API for the web application
- Follow the tutorial and complete the tasks
- Your assignment will have REST API component

Conclusion

- Express node.js makes developing back ends easy and allows you to do it using JavaScript
- REST APIs allow people to use your web site for multiple purposes