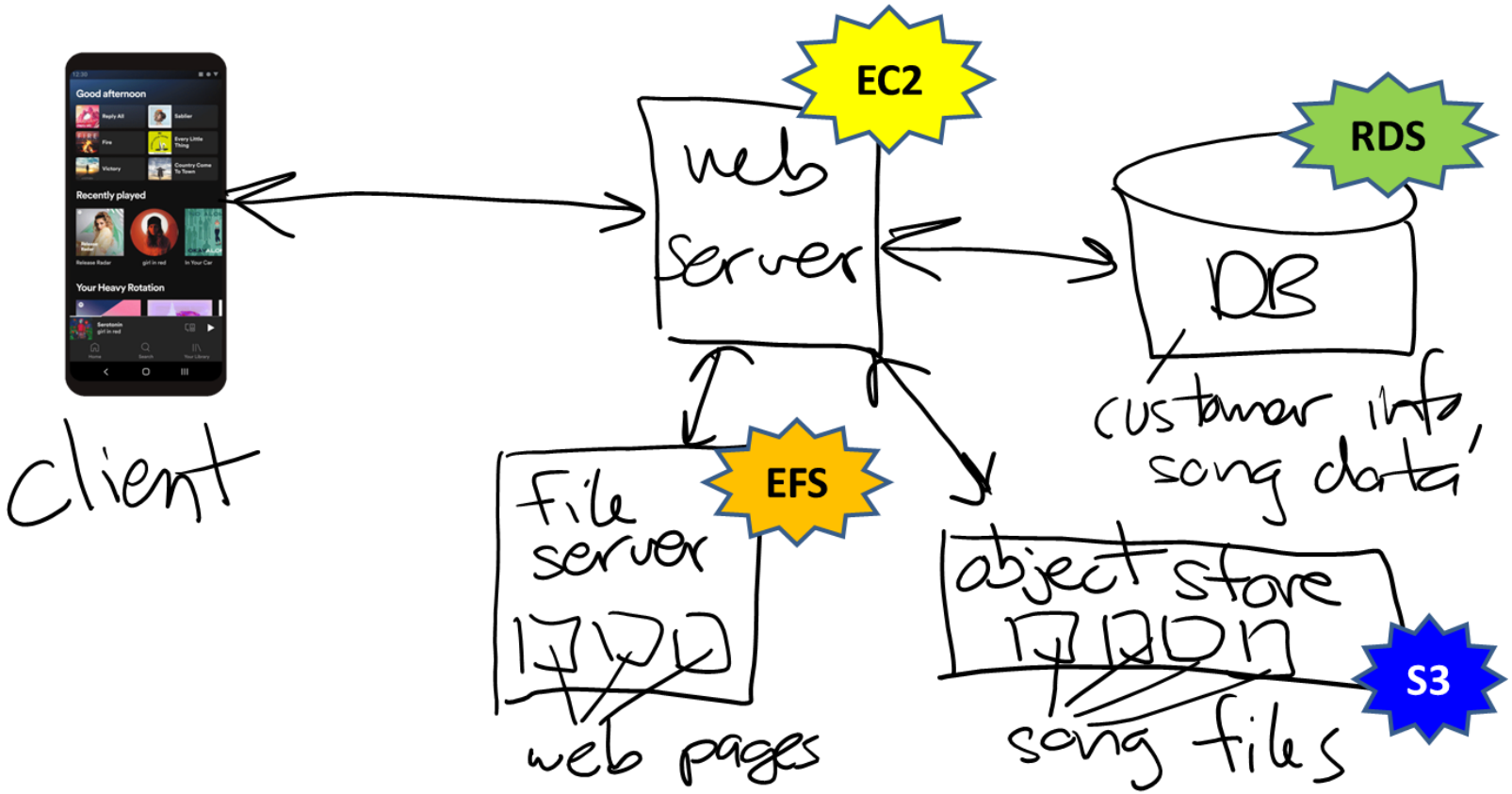
Intro to Web Services

- Service-oriented architectures (SOA)
- Web services
- A simple example in JavaScript

Multi-tier architecture

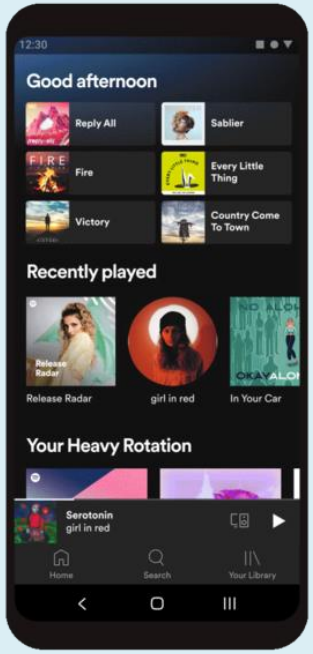
- **Recall that most modern software is multi-tier…**

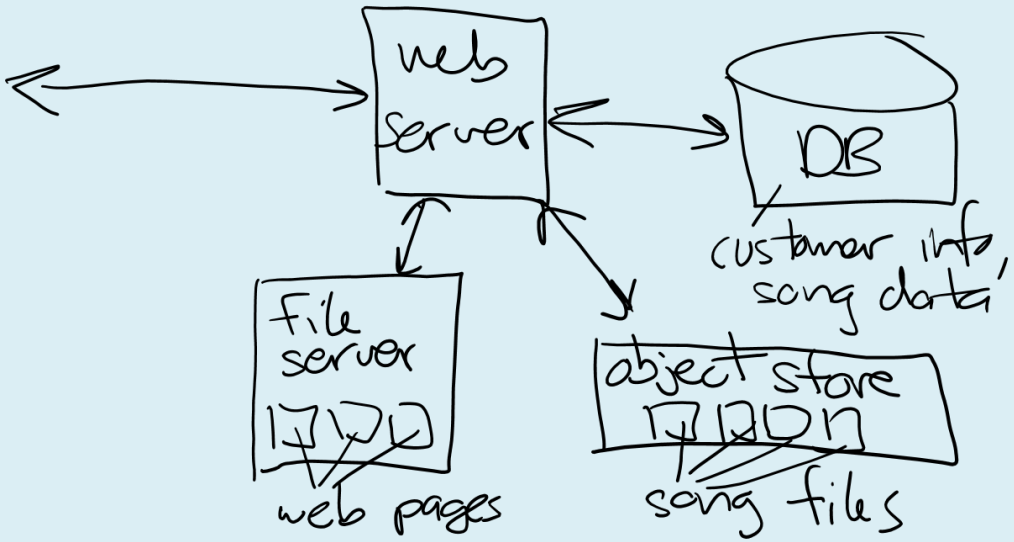


Example

- **Spotify:**
  - *https://developer.spotify.com/documentation/web-api*

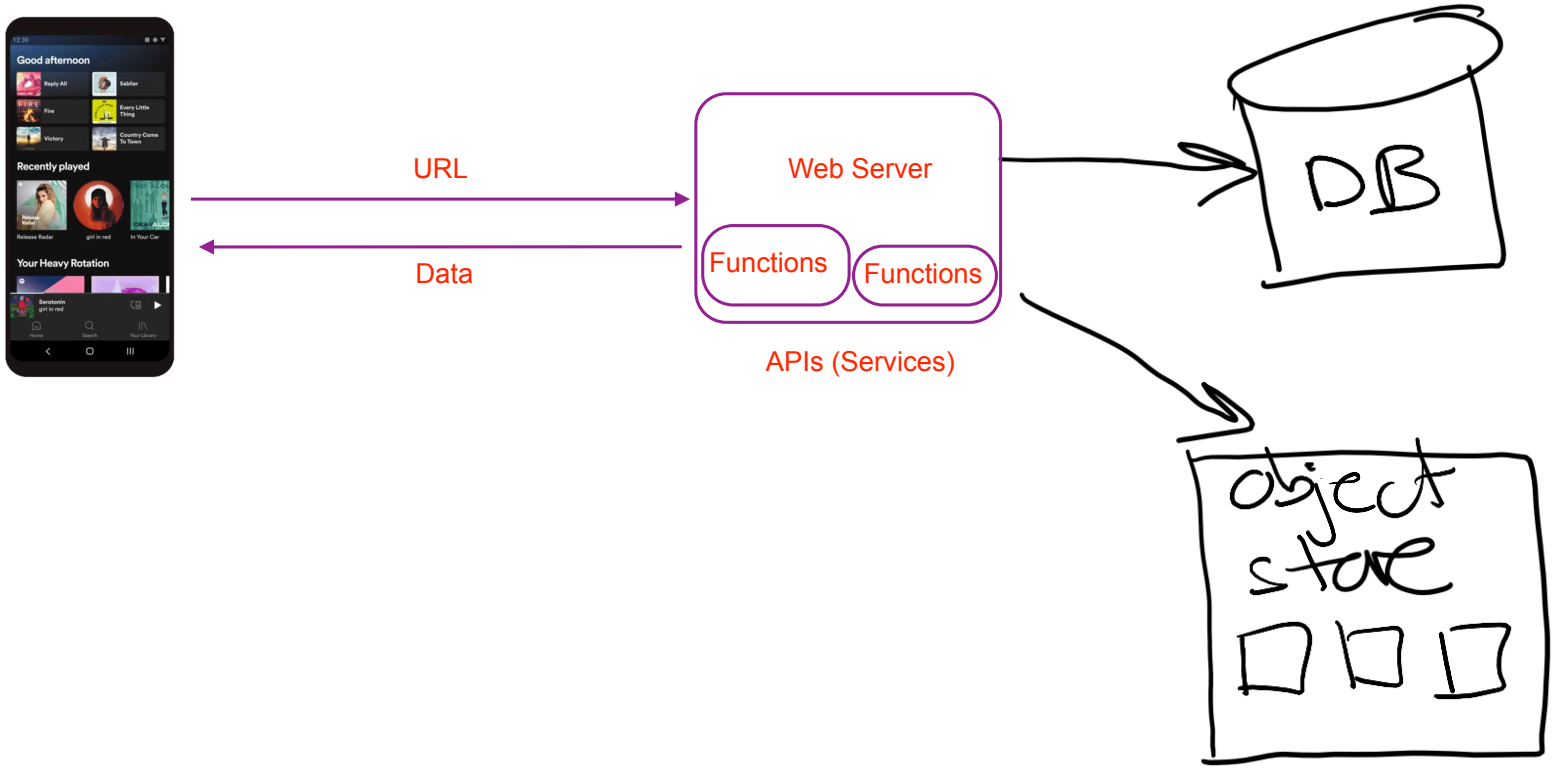
Web server exposes certain functions called APIs. Client interacts with database using these function calls.





Service-oriented architectures (SOA)

- **Redesign server-side as 1 or more services (APIs)**
- **URL ==> unique "address" for calling function**



Example #2

- **Chicago Transit Authority (CTA) monitoring**
  - *Web service for real-time bus and L tracking*
  - *https://ctabustracker.com/bustime/api/v2/**getpredictions**?key=**KEY**&rt=**ROUTE_ID**&stpid=**STOP_ID**&format=json*

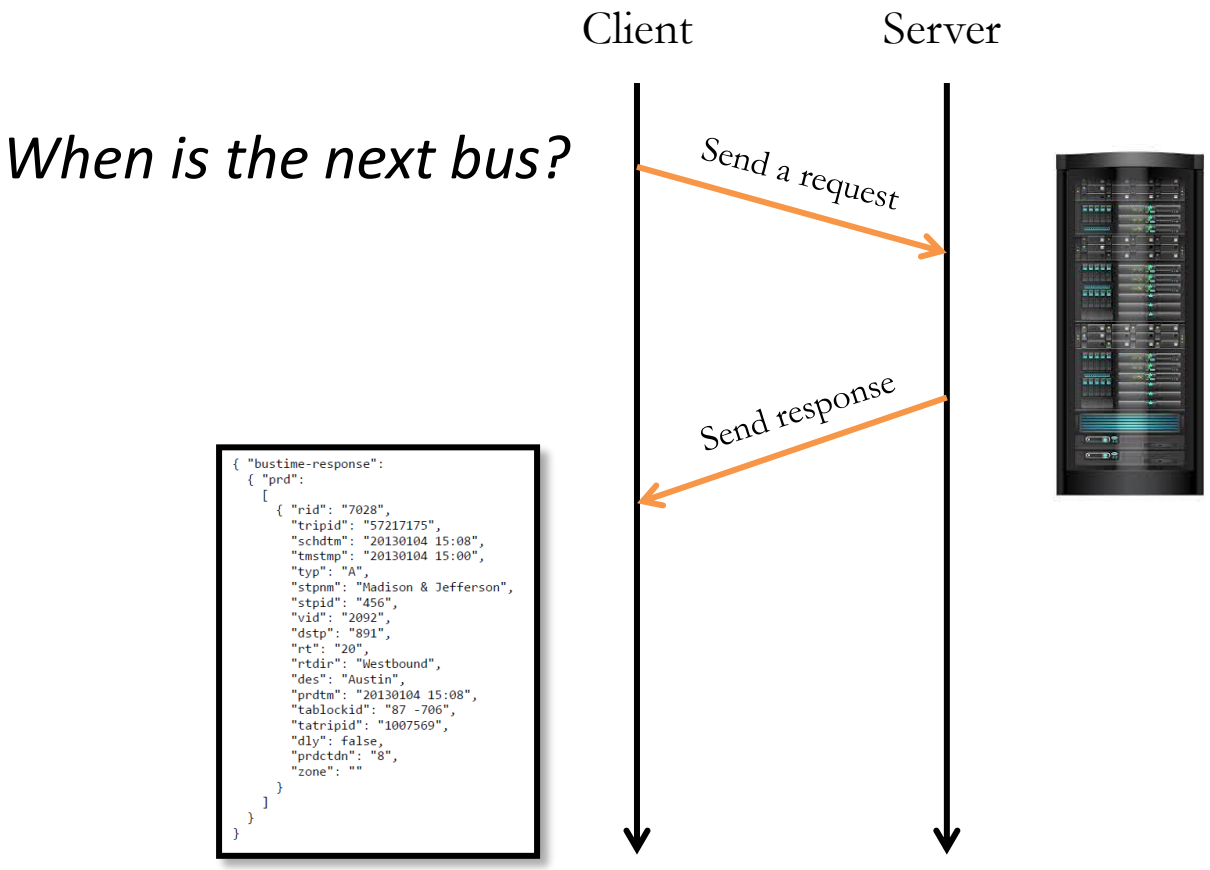
    Function
  - *Example: south-bound stop across from of Tech*


```
{ "bustime-response":
  { "prd":
    [
      { "rid": "7028",
        "tripid": "57217175",
        "schdtm": "20130104 15:08",
        "tmstmp": "20130104 15:00",
        "typ": "A",
        "stpnm": "Madison & Jefferson",
        "stpid": "456",
        "vid": "2092",
        "dstp": "891",
        "rt": "20",
        "rtdir": "Westbound",
        "des": "Austin",
        "prdtm": "20130104 15:08",
        "tablockid": "87 -706",
        "tatripid": "1007569",
        "dly": false,
        "prdctdn": "8",
        "zone": ""
      }
    ]
  }
}
```

Request-response protocol

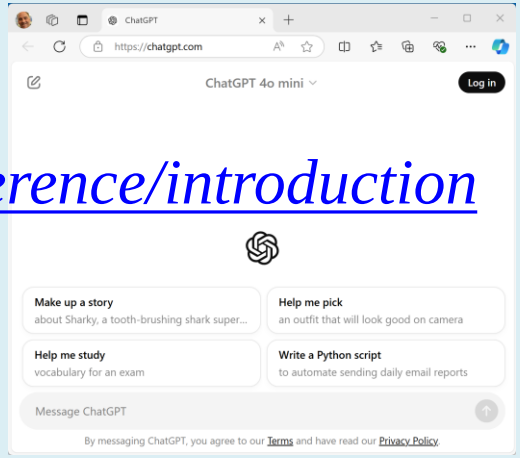
- **Services adhere to request-response protocol**

https://ctabustracker.com/bustime/api/v2/**getpredictions**?key=**KEY**&rt=**ROUTE_ID**&stpid=**STOP_ID**&format=json



Other API examples…

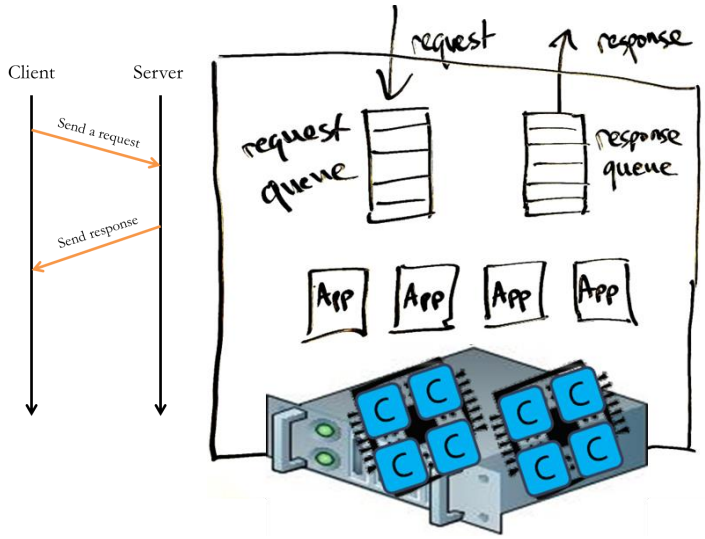
- **ChatGPT:**
  - *https://platform.openai.com/docs/api-reference/introduction*



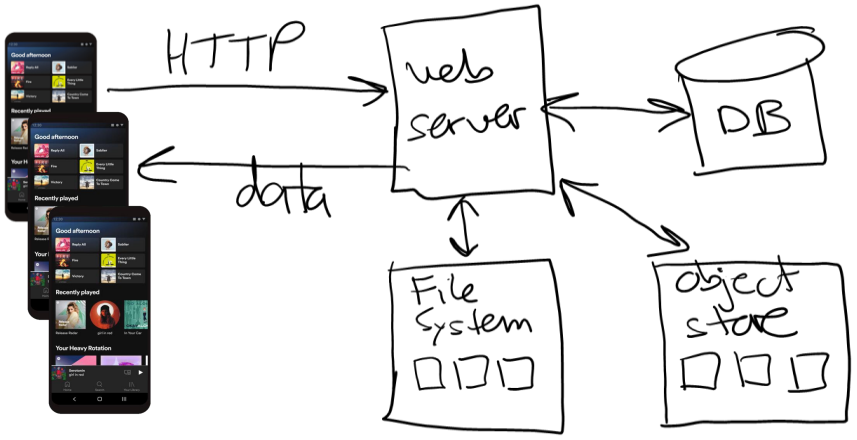
Why use Web Services for building applications?

Apache is being used as the web server technology. It is most stable as welll as free.

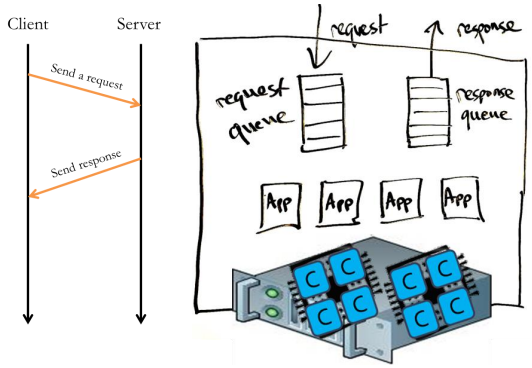
- Web community has already solved the problems designers face:
  - Support for concurrent users
  - Client-side configuration
  - Security / encryption
  - Supports most programming languages
  - Lots of frameworks (here are just a few):
    - **Spring** (Java)
    - **Django** (Python)
    - **Node.js** (JavaScript)



- Main disadvantages?
  - Messages as human-readable strings are less efficient
  - Limited API design options



Hyper Text Transport Protocol (HTTP)

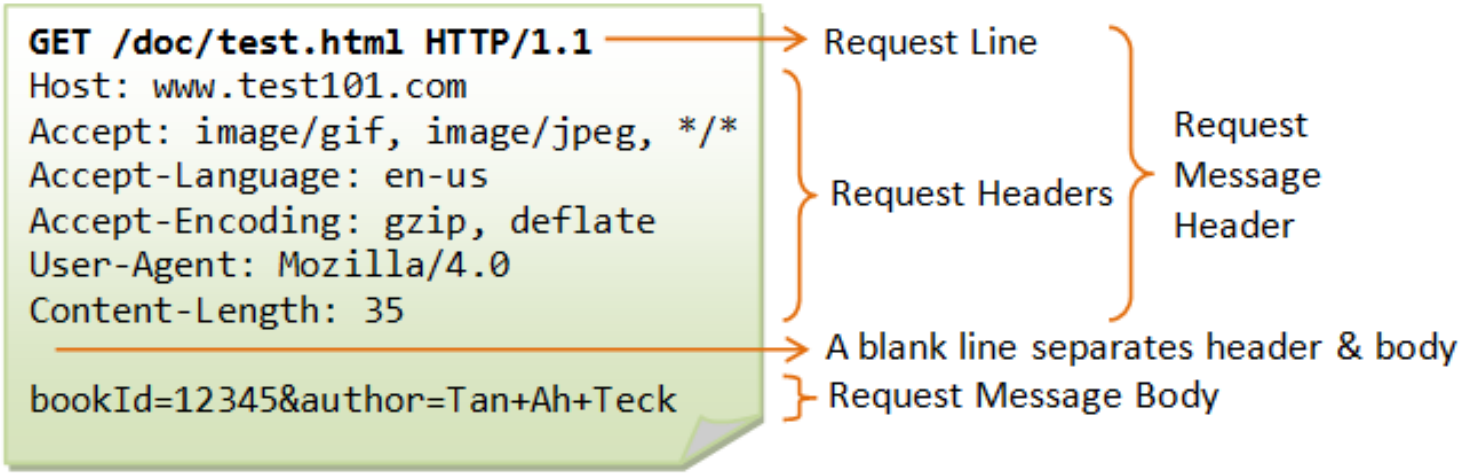


- **HTTP** is a client-server data exchange protocol
- Invented for web browsers to fetch pages from web servers

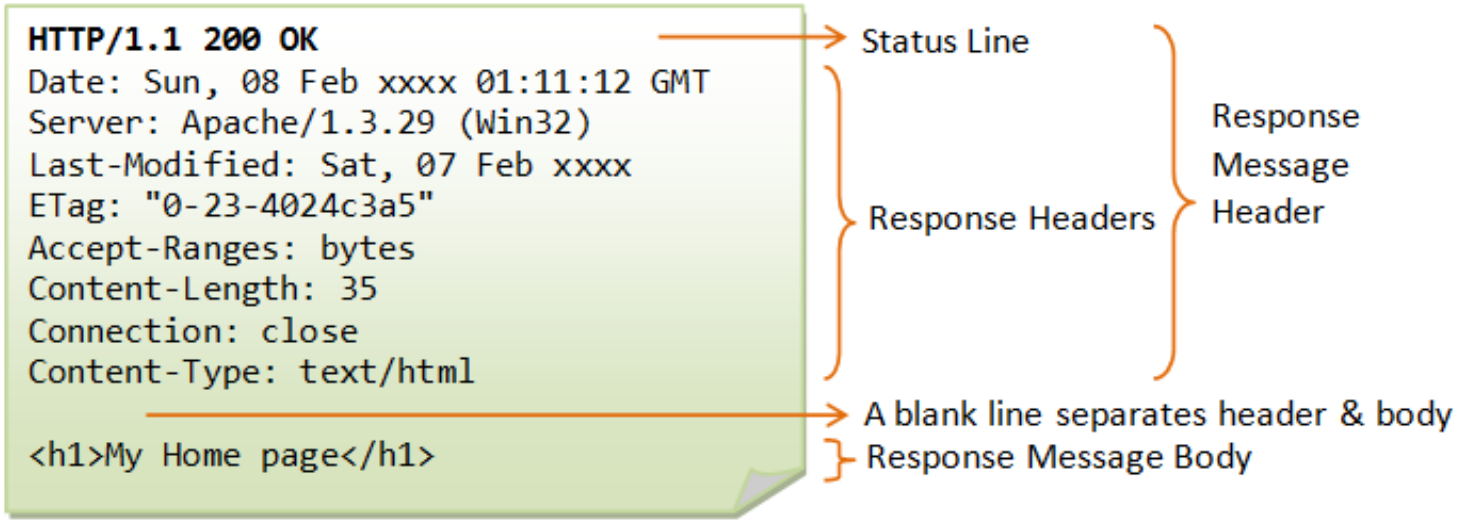
- **Request** specifies:
  - A human-readable header with: *URL*, *method* (plus some optional headers)
  - An optional *body* storing raw data (bytes)

- **Response** includes:
  - A human-readable header with *response code* (plus some optional headers)
  - An optional *body*

Request:



Response:

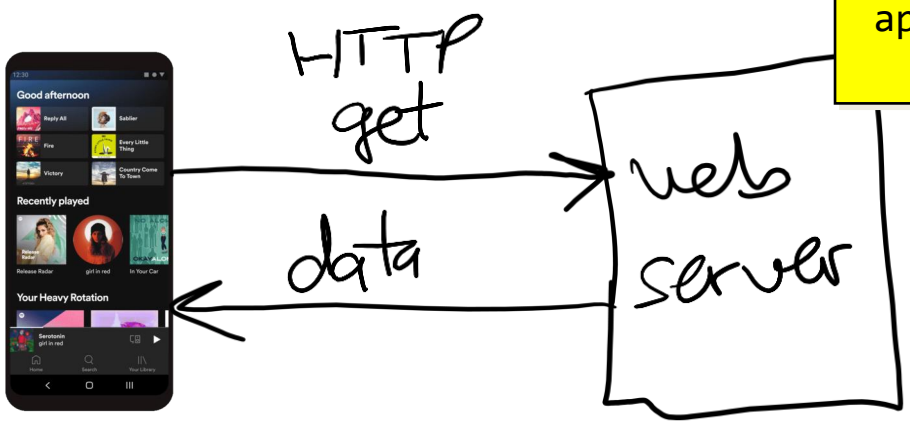



EXAMPLE

Simple SOA example

- **Let's build a simple calculator web service…**

```
// increment x:
app.get('/incr/:x', (req, res) => {…});

// add x and y:
app.get('/add/:x/:y', (req, res) => {…});

// raise x to the exponent e:
app.get('/pow/:x/:e', (req, res) => {…});
```



API function format

- **This example is using the Express framework**
- **Express is built on top of JavaScript and Node.js**

app.get is a function triggered by the user, and its gets triggered when the path matches. req is the input object and res is the output response object. => is the callback function which triggers the code in parenthesis when the API gets triggered.

```
//
// increment the value of x and return it:
//
app.get('/incr/:x', (req, res) => {


  }
);
```

callback function to execute the call:
req is request object
res is response object

object representing express application

HTTP method

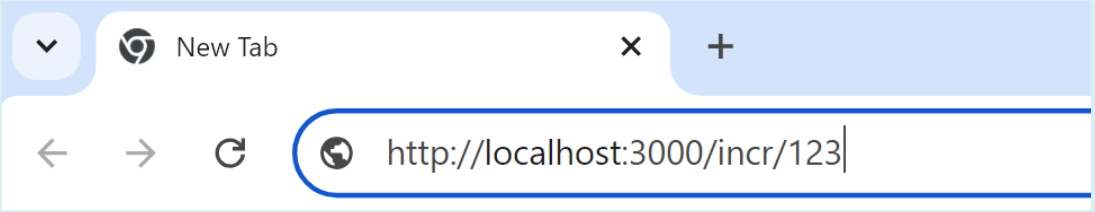
Path (or route)

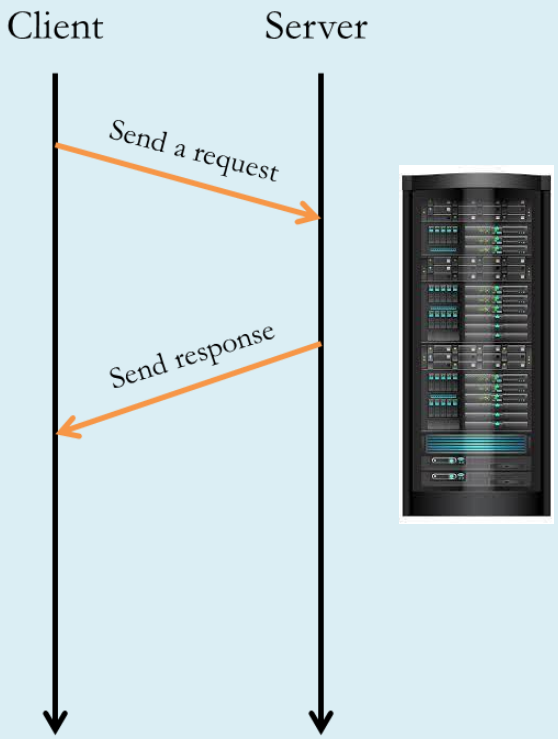
URL parameter

This is different from the query parameter, which is written like ?x=123

Demo : client-side

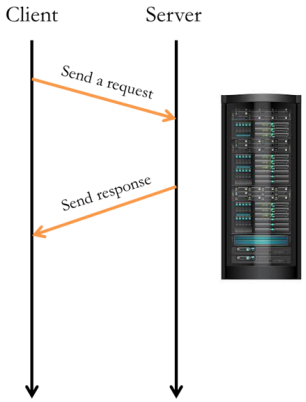
- It's a web service, our client can be a web browser!





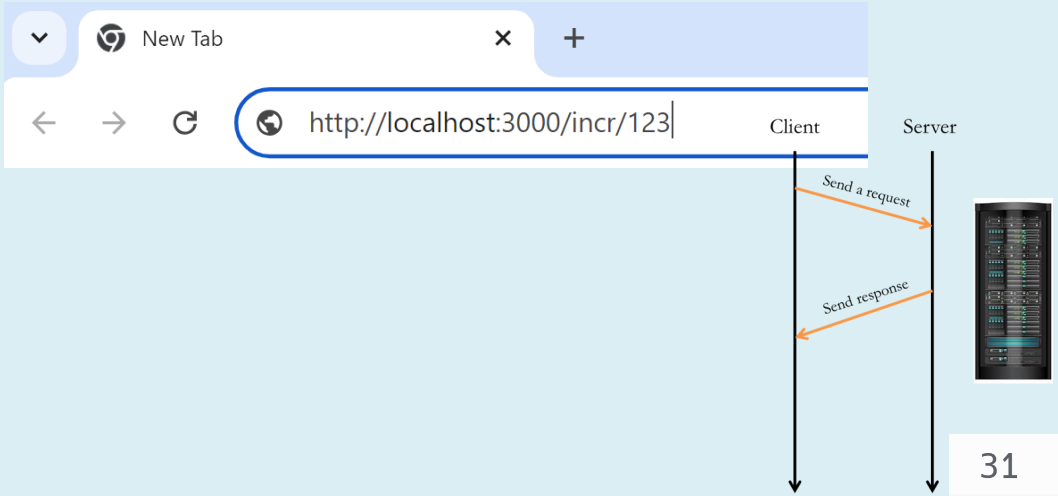
Demo : server-side

- **Server starts up and listens for requests…**




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

// main():
app.listen(3000, () => {
  console.log('**SERVER: web service running, listening on port 3000...');
});

// requests for default page /:
app.get('/', (req, res) => {
  console.log('**SERVER: call to /');
  res.send('<HTML><body>Home page is empty, we are a calculator service!</body></HTML>');
});

// API functions:
// increment x:
app.get('/incr/:x', (req, res) => {…});

// add x and y:
app.get('/add/:x/:y', (req, res) => {…});

// raise x to the exponent e:
app.get('/pow/:x/:e', (req, res) => {…});
```

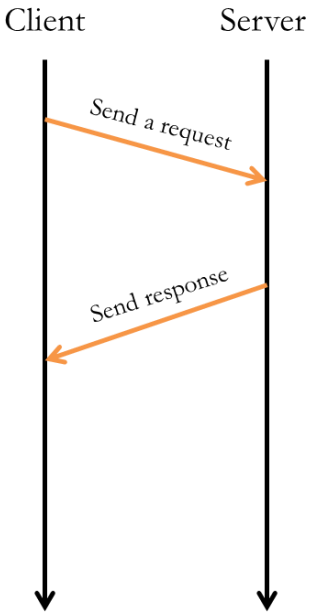
incr(x)




```
31  //
32  // incr(x)
33  //
34  app.get('/incr/:x', (req, res) => {
35
36    console.log("incr: call")
37
38    let x = parseInt(req.params.x);
39
40    let y = x + 1;
41
42    res.send(y.toString());
43    return;
44  });
45
```

Exception handling

- **Exception handling is critical on the server to ensure the client receives a response --- they are waiting!**




```
//
// increment the value of x and return it:
//
app.get('/incr/:x', (req, res) => {

   try {

      · if (isNaN(x))
      ·    throw Error("x is not a number")
      ·
      res.send(…);    // by default sends code 200 => success

   }
   catch(err) {
      res.status(400).send(err.message);   // 400 => bad request
   }

});
```

Frameworks

- **Web service running Node.js + Express frameworks**
  - *https://nodejs.org/*
  - *https://expressjs.com/*
  - *Node.js provides JS execution + basic web server support*
  - *Express provides additional support for web services*

add(x, y) and pow(x, e)

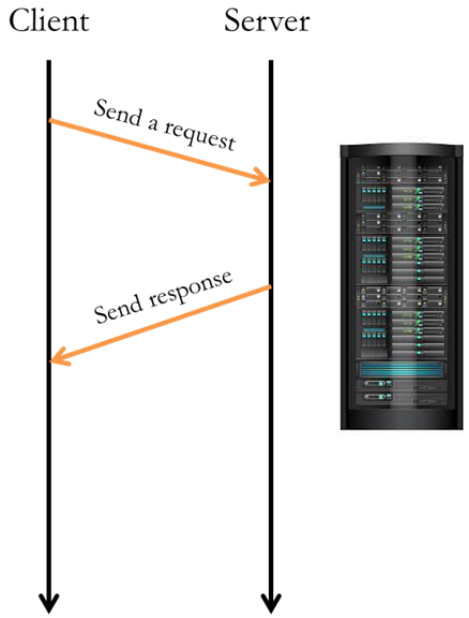
```
45  //
46  // add(x, y)
47  //
48  app.get('/add/:x/:y', (req, res) => {
49
50    //
51    // TODO: assume integers x and y (use parseInt)
52    //
53
54  });
55
```

```
56  //
57  // pow(x, e)
58  //
59  app.get('/pow/:x/:e', (req, res) => {
60
61    //
62    // TODO: assume x and e can be floating-point (use parseFloat)
63    //
64
65  });
66
```

Demo : client-side from Python

- **Use Python's requests networking module**


```
import requests

baseurl = 'http://localhost:3000/'

x = input('Enter integer x> ')

# build URL:
url = baseurl + 'incr/' + x

# call the service:
response = requests.get(url)

# print the result:
print('status code:', response.status_code)
print('result:', response.text)
```



That's it, thank you!