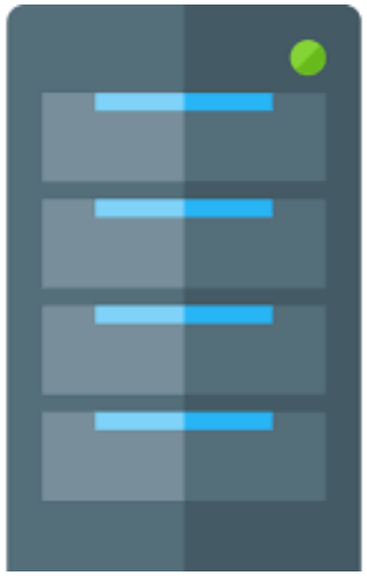
SERVERS

What Why How?

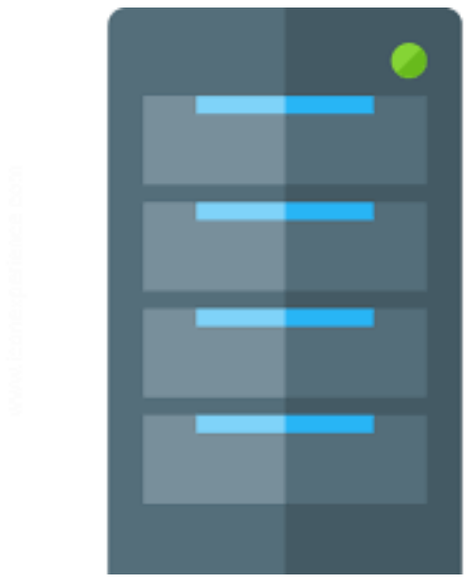
WEB REQUESTS

Client **Server**



WEB REQUESTS

Client **Server**



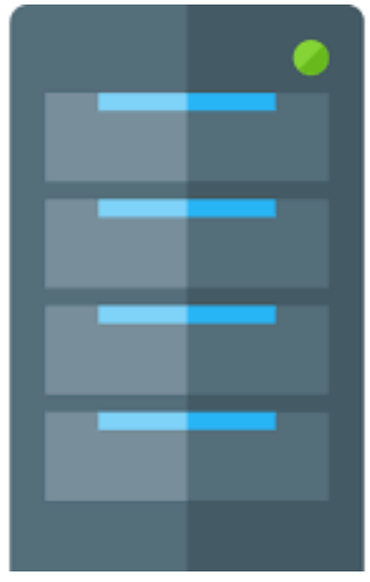
HTML

WEB REQUESTS

Client **Server**

GET http://google.com

HTML



WEB REQUESTS

Client **Server**

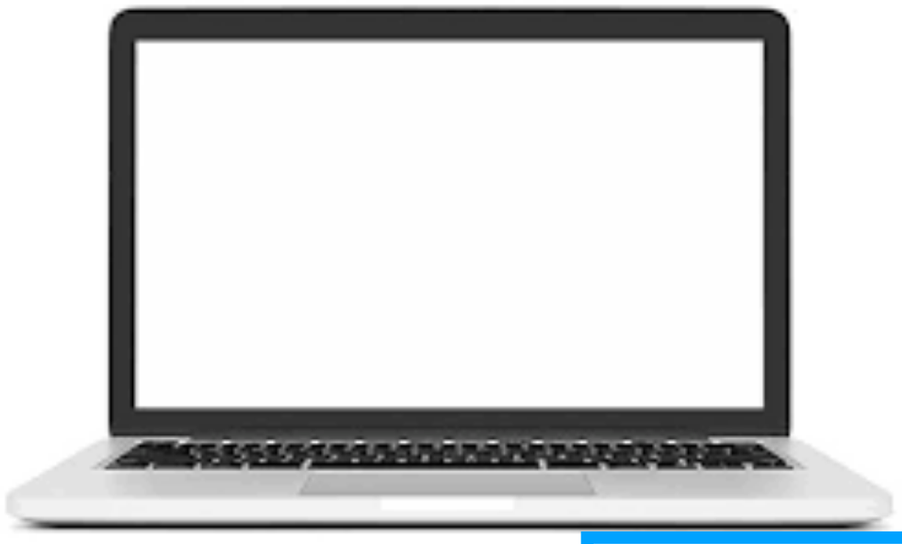
GET
http://google.com

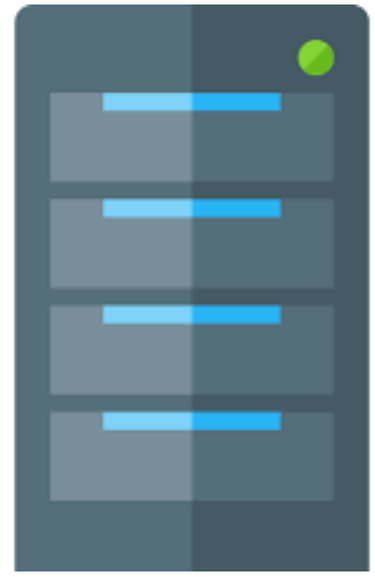
HTML

```
1 <html>
2   <head>
3     <title>Google Search</title>
4   </head>
5   <body>
6     <img src="/logo.png"/>
7     <input type="text"/>
8   </body>
9 </html>
```

WEB REQUESTS

Client **Server**



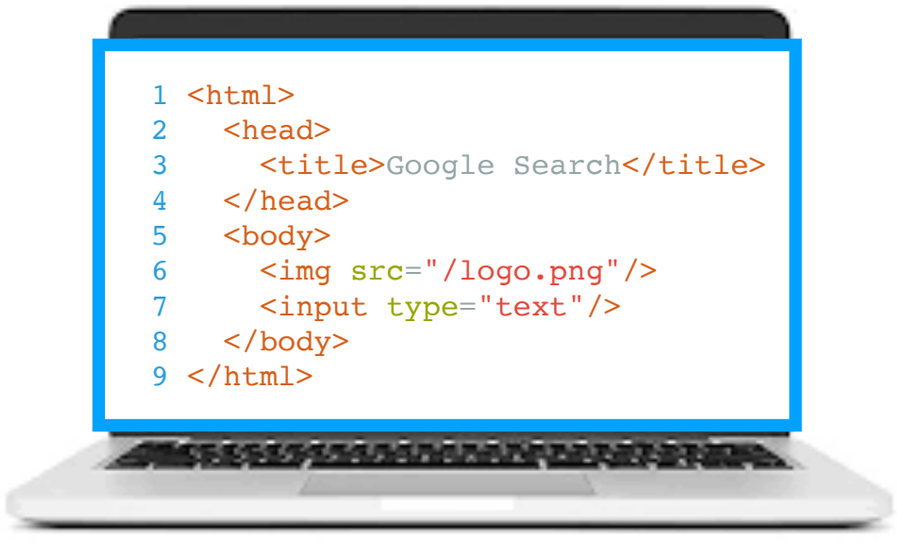


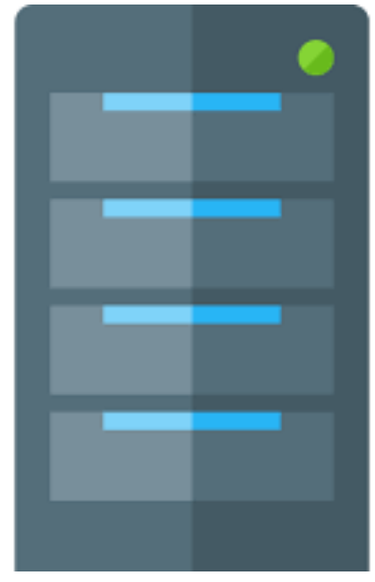
HTML

```
1 <html>
2   <head>
3     <title>Google Search</title>
4   </head>
5   <body>
6     <img src="/logo.png"/>
7     <input type="text"/>
8   </body>
9 </html>
```

WEB REQUESTS

Client **Server**





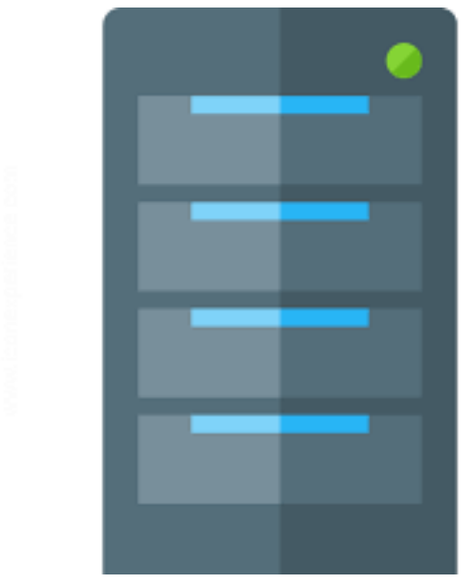
WEB REQUESTS

Client

Server

http://google.com/logo.png

```
1 <html>
2   <head>
3     <title>Google Search</title>
4   </head>
5   <body>
6     <img src="/logo.png"/>
7     <input type="text"/>
8   </body>
9 </html>
```

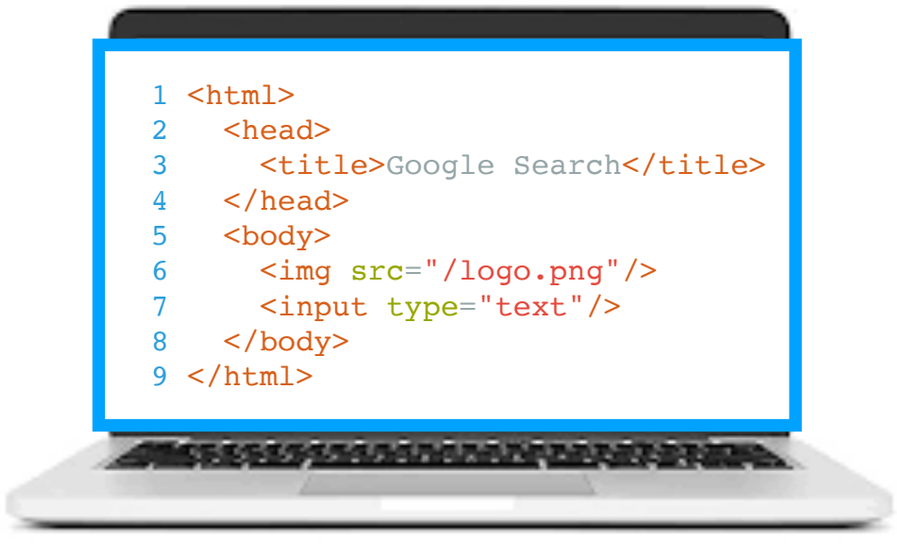


Images

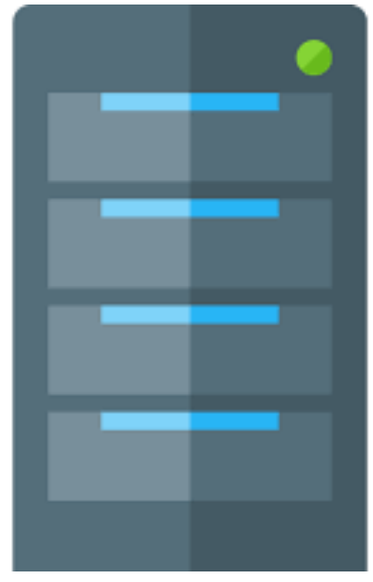
WEB REQUESTS

Client **Server**

GET http://google.com/logo.png



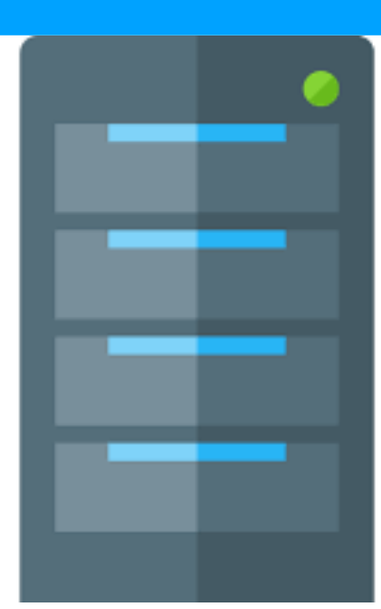
Images

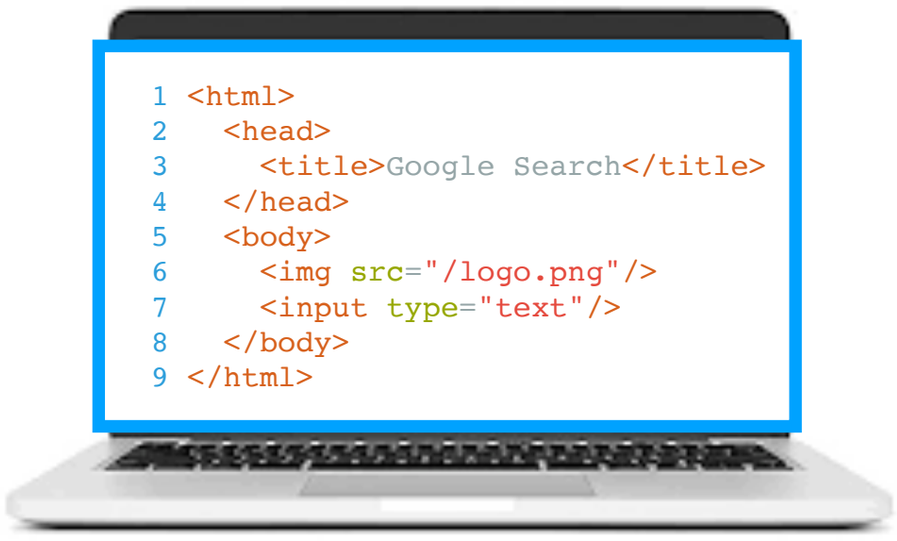


WEB REQUESTS

Client **Server**

GET
http://google.com/logo.png



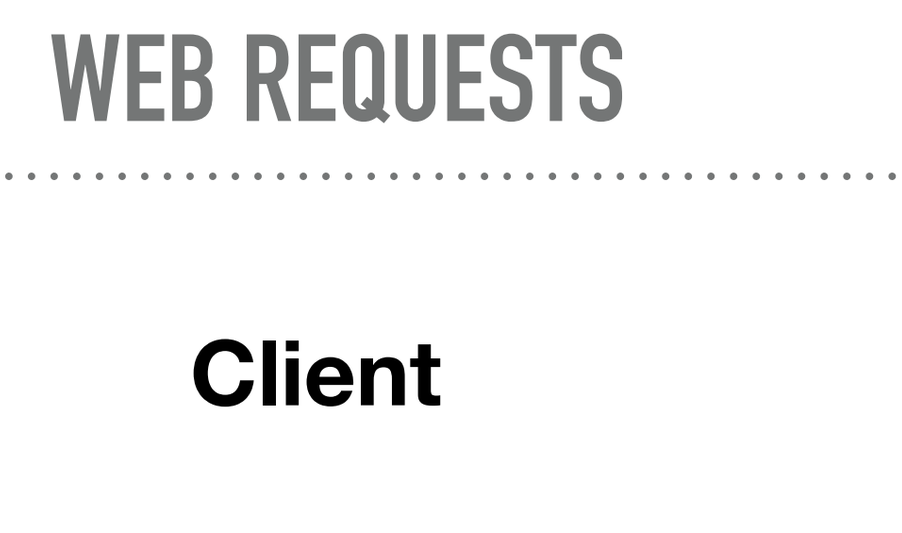


Images

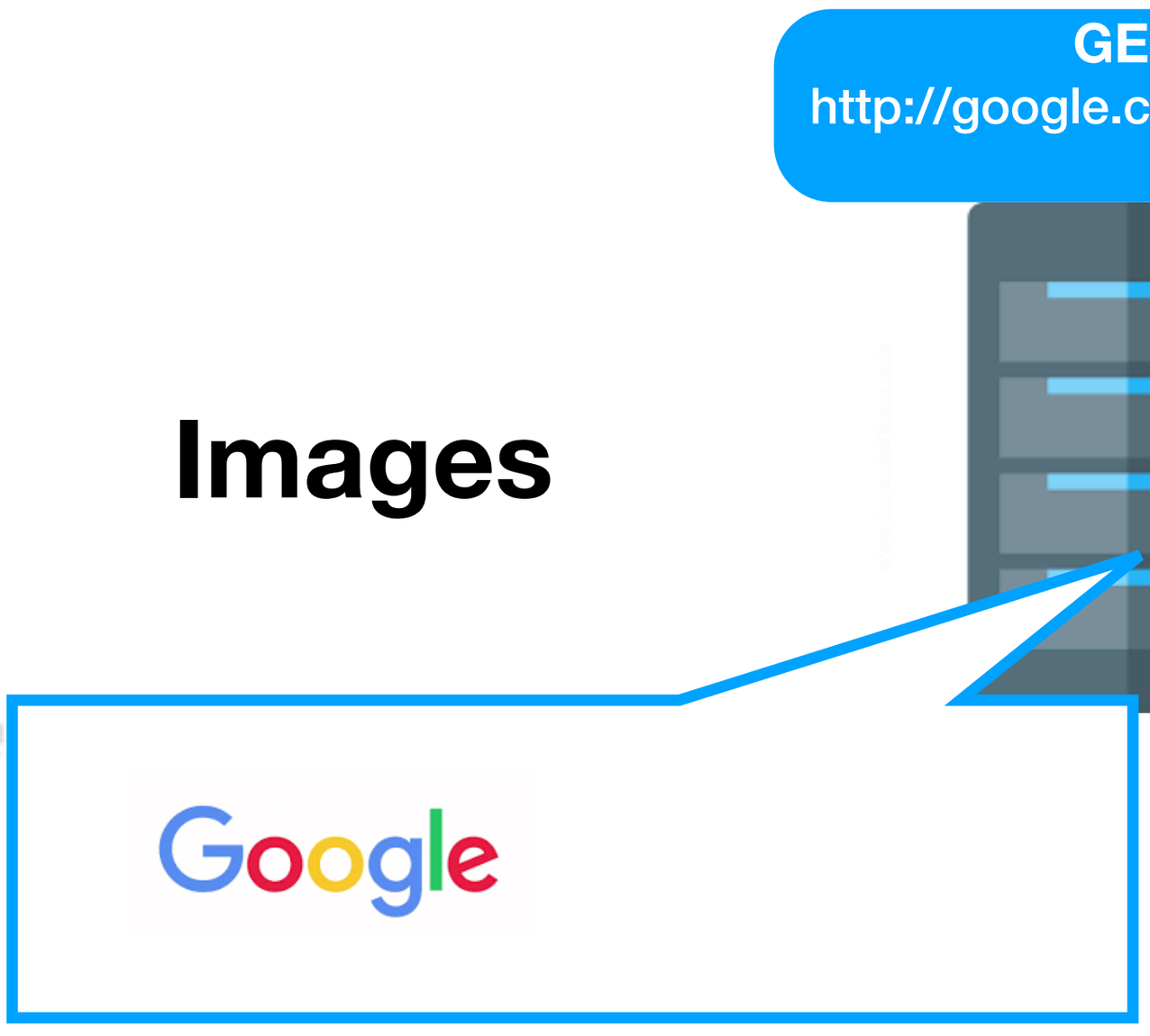
WEB REQUESTS

Client **Server**

GET
http://google.com/logo.png

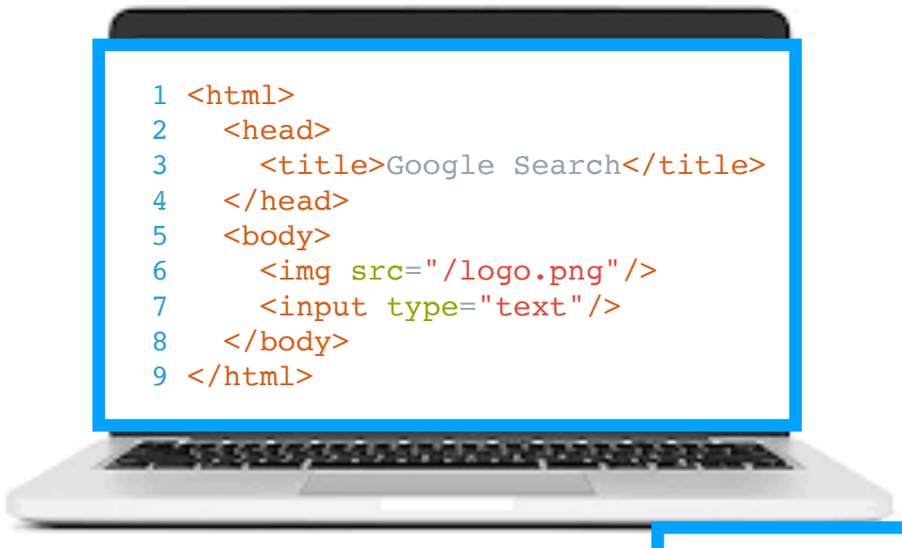


Images

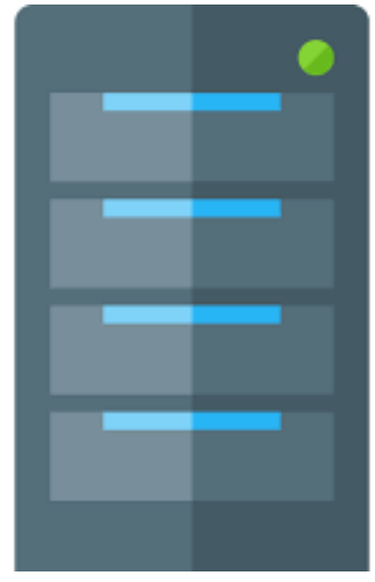


WEB REQUESTS

Client **Server**

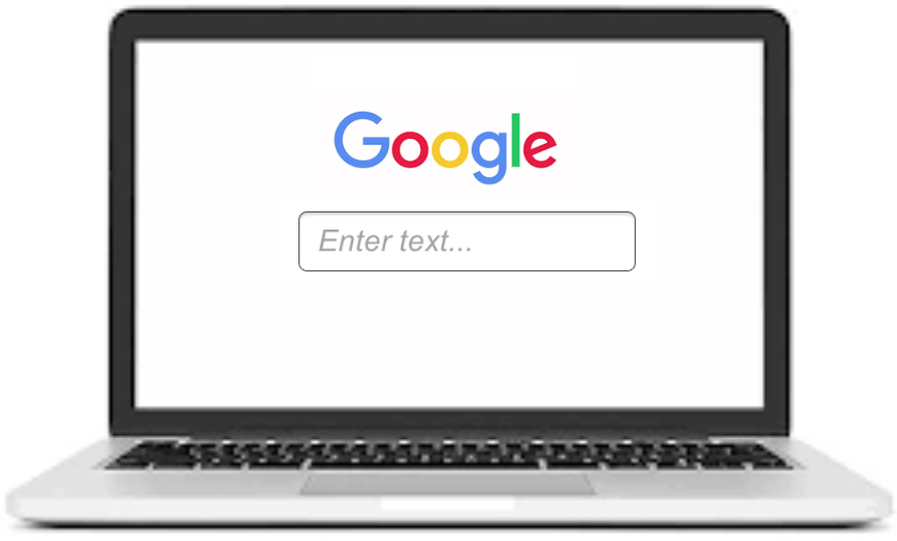


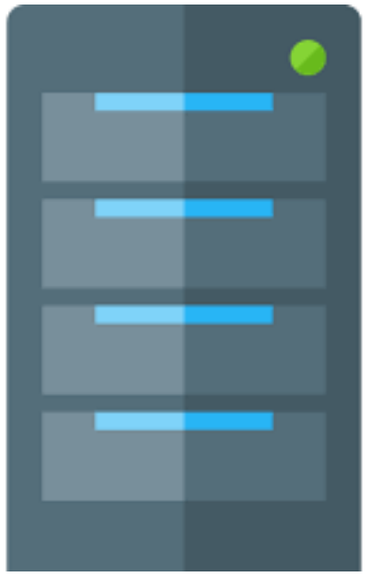
Images



WEB REQUESTS

Client **Server**

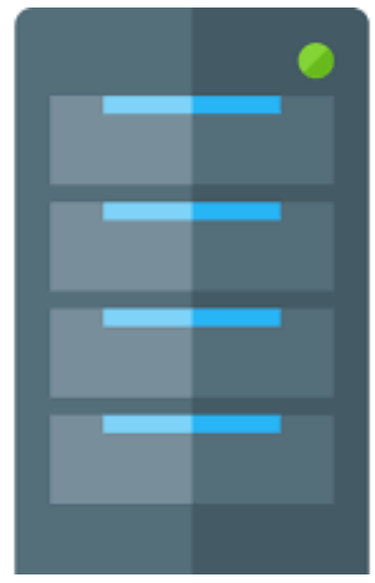




WEB REQUESTS

Client **Server**

Data/JSON

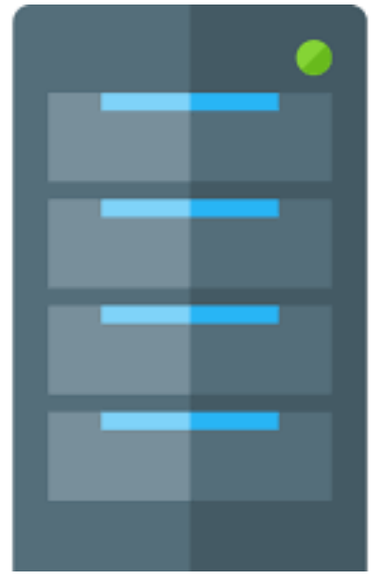


WEB REQUESTS

Client **Server**

GET http://google.com/search?q=pizza

Data/JSON

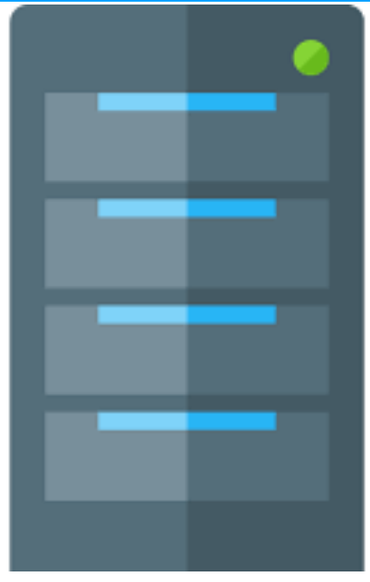


WEB REQUESTS

Client **Server**

GET
http://google.com/search?q=pizza

Data/JSON

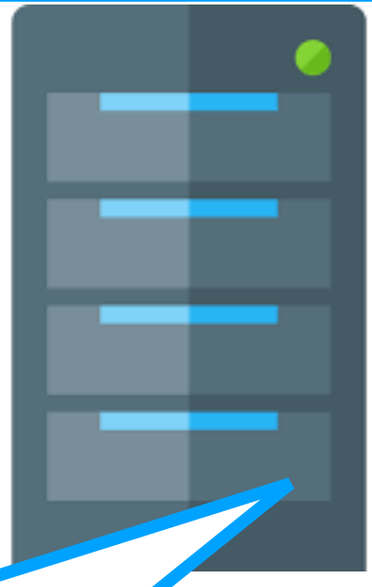


WEB REQUESTS

Client **Server**

GET
http://google.com/search?q=pizza

Data/JSON

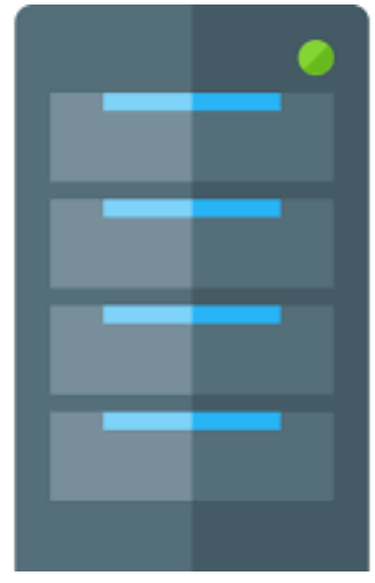


```
{
  "predictions":[
    "pizza hut",
    "pizza by me",
    "pizza factory",
    "pizza delivery",
    "pizza coupons"
  ]
}
```

WEB REQUESTS

Client **Server**

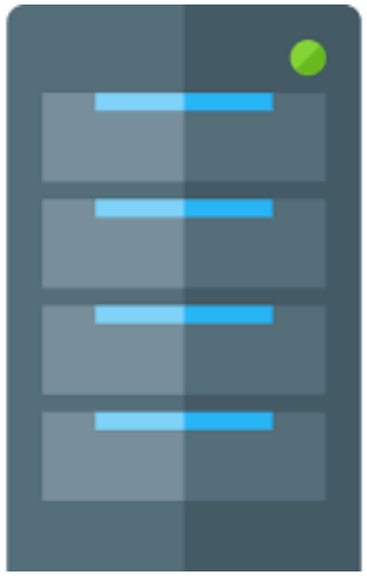
Data/JSON



```
{
  "predictions":[
    "pizza hut",
    "pizza by me",
    "pizza factory",
    "pizza delivery",
    "pizza coupons"
  ]
}
```

WEB REQUESTS

Client **Server**



REST METHODS

- GET: Get Existing Data
- POST: Create New Data
- PUT/PATCH: Update Existing Data
- DELETE: Delete Existing Data

CRUD

- CREATE: Create New Data
- READ: Get Existing Data
- UPDATE: Update Existing Data
- DELETE: Delete Existing Data

NODE

- JavaScript that runs outside of a browser
- Lets us write servers with JavaScript
- Fastest growing server language
- Built for working with SPA (single page applications)

EXPRESS

- Framework for a server
- Makes it easy to receive requests
- Makes it easy to send responses back
- Makes it easy to add 3rd party packages to extend server

DATABASE

- Let us store information
- Structures data for consistency
- They are designed for speed and reliability
- Sql - Document - Graph
- Postgres SQL / MassiveJS

NPM

- Node Package Manager
- Lets us use other people’s code
- Version management of code
- Can publish our own modules for others to use

PACKAGE.JSON

- Describes the project we are working on
- Name, Description, Authors, Version
- Dependencies that our code needs to run
- Created by npm init OR create-react-app

.gitignore Files

- Specify files/folders we don’t want git to keep track of
- Ignore node_modules folder
- Ignore system files (.DS_Store, .log, .Spotlight)
- Ignore secret information
- Ignore configuration that might be different between deployed and development versions

EXPRESS EXAMPLE CODE

```
1  const express = require('express');
2  const app = express();
3  const bodyParser = require('body-parser');
4  const port = 8080;
5
6  let books = ["Harry Potter", "Mistborn", "War
7               and Peace", "Pride Prejudice and
8               Zombies"];
9
10 app.use(bodyParser.json());
11
12 app.get('/api/books', function(req, res){
13   res.send(books);
14 })
15
16 app.post('/api/books', function(req, res){
17   books.push(req.body.title);
18   res.send(`${req.body.title} added to the list
19             of books`);
20 })
21
22 app.listen(port, ()=>{
23   console.log(`Listening on port ${port}`);
24 })
```

EXPRESS EXAMPLE CODE

1) Node uses require to import packages Here we bring in express and save it to a variable

2) We name our express application app. We create a new application by invoking express

3) We bring in the package body-parser that we will use to convert the incoming request's body to a js object

```
1 const express = require('express');
2 const app = express();
3 const bodyParser = require('body-parser');
4 const port = 8080;
5
6 let books = ["Harry Potter", "Mistborn", "War
7              and Peace", "Pride Prejudice and
8              Zombies"];
```

4) We define the port that our server will run on. We will use it a bit later in our server

6) We are building an array of book titles this will work as our fake database until we learn how to do real databases next week

EXPRESS EXAMPLE CODE

Our application we made earlier

Use tells our app that it will use this on every request coming into our server

bodyParser is the package we brought in earlier.
It will parse the bodies of any requests that come into our server

```
10 app.use(bodyParser.json());
```

This is our first example of using middleware. Middlewares are functions that we can run on multiple endpoints, and runs before our own code.

The type of conversion we are doing. We will only use the json conversion type. There are others for other data types (raw, text, url encoded)

EXPRESS EXAMPLE CODE

Name of our app

Almost universally called app

REST method this endpoint handles

get, post, put, patch, delete

Endpoint url

Commonly starts with /api/

req and res variables

Can be called anything but req and res standard

```
12 app.get('/api/books',(req, res)=>{
13   res.send(books);
14 })
```

Response object

controls everything about our response

Defining what to format to send information back

send, json, sendFile, sendStatus

Defining what information to send back

can be anything from a string, to an object or an array

EXPRESS EXAMPLE CODE

Method that this endpoint will listen for.

URL this endpoint will listen for

Callback function that will fire when a request matches this endpoint

req and res variables that we will use to handle the request and send response

```
16 app.post('/api/books', function(req, res){
17   books.push(req.body.title);
18   res.send(`${req.body.title} added to the list
19             of books`);
20 })
```

17) Saving the new book's title to the array of books

18) We tell the server to send back a response saying a book was added

EXPRESS EXAMPLE CODE

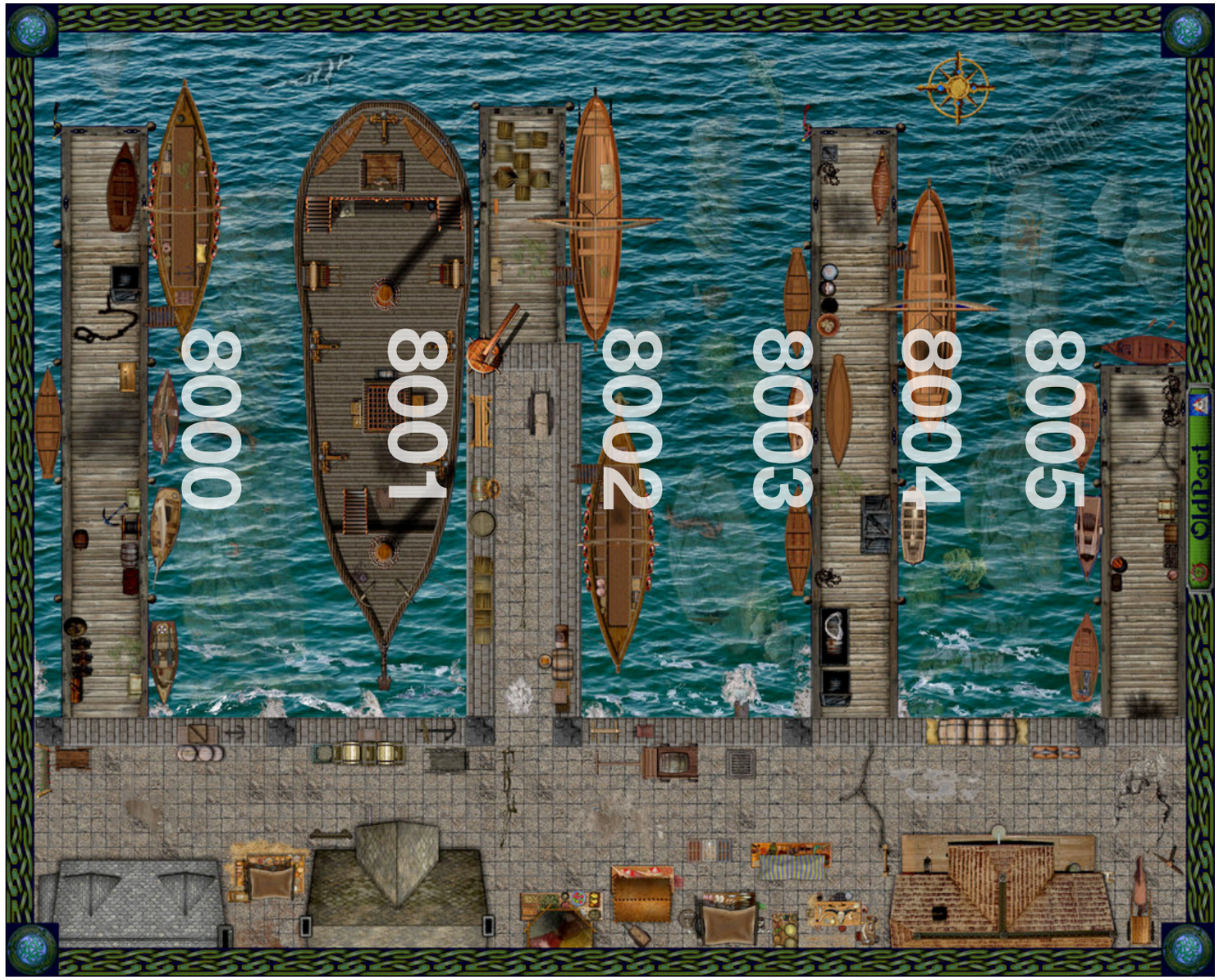
Tell our server to start handling any requests that come into a specific port on our server. The console log will be display when the server starts listening on the port. It is normal practice to have the server say the port it is running on. This makes working with it easier when you go to deploy the project

```
22 app.listen(port, ()=>{
23   console.log(`Listening on port ${port}`);
24 })
```

PORTS

On a computer ports are places that requests (ships) can go to find information. Each port is numbered. We can tell our code that it needs to watch a particular port, then when a request (ship) comes in. It will handle the request, and send the ship out with it's requested information.

Some ports are reserved already on your computer. To avoid conflicting with existing ports, you can run on ports over 3001



ENDPOINTS ARE FUNCTIONS

- Endpoints are functions that run on a separate computer
- They have a return value specified by the res.send()
- They have 3 ways to accept incoming information
    - Params
    - Query
    - Body

PARAMS

- Defined when making the endpoints url
- Must be present for the endpoint to match
- Great for any information the needs to be there for the endpoint to work
- Available as req.params
- ‘/api/books/:id’
- ‘/api/geolocation/:lat/:long’
- ‘/api/events/:company/:date’

QUERY

- Passed in when making the API call
- Optional, will match endpoint without them
- Great for any information that refines an existing endpoint

- Available as req.query
- ‘/api/books?name=Harry’
- ‘/api/recipes?ingredient=chicken&category=mexican’
- ‘/api/movies?genre=action’

BODY

- Passed in when making the API call
- Optional, endpoint will match without checking it
- Great for sending entire objects or large bits of data that don’t fit in a URL
- Available as req.body (if you are using body-parser)
- Use on POST, PUT, PATCH, DELETE endpoints.

PARAMS VS QUERY VS BODY

```
app.delete('/api/params/books/:index', (req, res)=>{
  books.splice(req.params.index, 1);
  res.send(req.params.index + ' was deleted');
})

app.delete('/api/query/books', (req, res)=>{
  if (req.query.index){
    books.splice(req.query.index, 1);
    res.send(req.query.index + ' was deleted');
  }else{
    res.status(400).send('Must provide a query
                param of index to delete');
  }
})

app.delete('/api/body/books', (req, res)=>{
  if (req.body.index){
    books.splice(req.body.index, 1);
    res.send(req.body.index + ' was deleted');
  }else{
    res.status(400).send('Must supply a body
                with an index parameter to delete');
  }
})
```

STATUS CODES

- We can use status codes to let the client know how our server treated their request
- 200’s Success
- 300’s Redirections
- 400’s Client Errors
- 500’s Server Errors
- https://en.wikipedia.org/wiki/List_of_HTTP_status_codes
- https://http.cat/