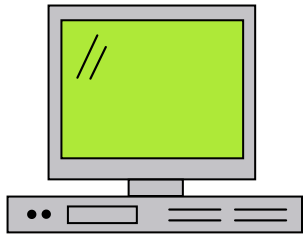
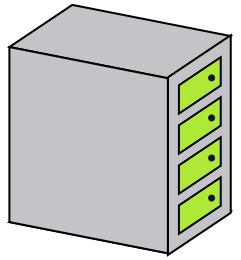
codecademy

Setting up a Server with HTTP

What is HTTP?

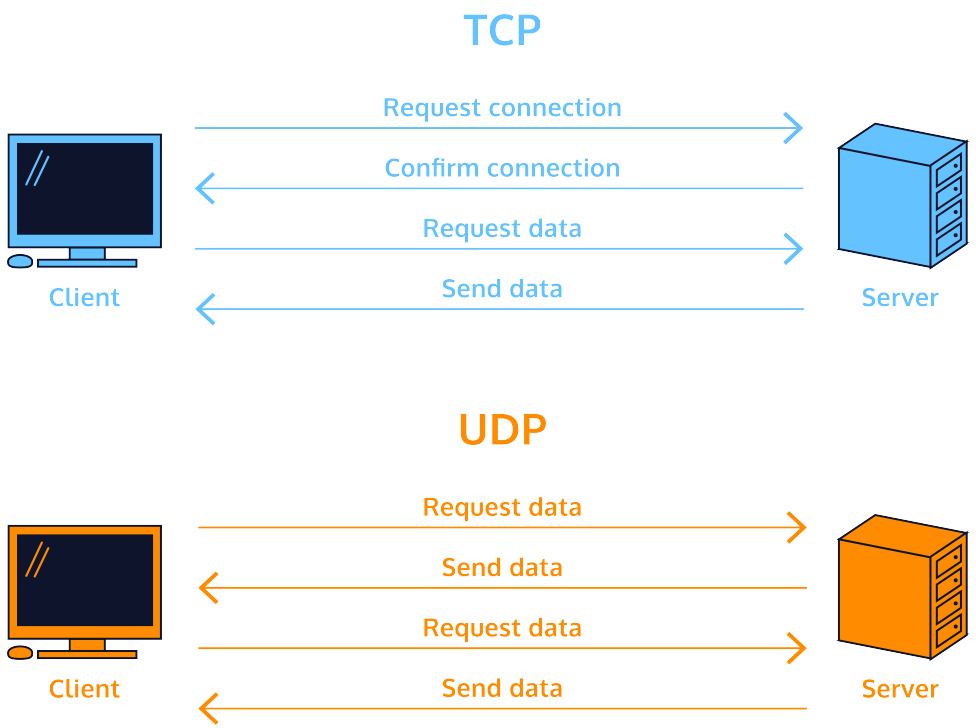
Hypertext Transfer Protocol (HTTP) is a data exchange protocol used to transmit/receive information on the Web. This protocol is widely used and serves as the backbone for communication between applications and servers.





HTTP Transmission

HTTP messages can be sent using various transmission protocols. The two most common protocols used for this are Transmission Control Protocol (TCP) and User Datagram Protocol (UDP).



Securing HTTP

HTTP messages can be protected using Transport Layer Security (TLS), a protocol designed to facilitate secure data transmission via encryption. Using TLS with HTTP will allow you to use HTTPS (Hypertext Transfer Protocol Secure), which helps denote the presence of extra security.

The `.createServer()` Method

The `.createServer()` method from the `http` module is used to create an HTTP server. The `.createServer()` method takes a single argument in the form of a callback function. This callback function has two primary arguments; the request (commonly written as `req`) and the response (commonly written as `res`).

```
const http = require('http');

// Create instance of server
const server = http.createServer((req, res) => {
  res.end('Server is running!');
});

// Start server listening on port 8080
server.listen(8080, () => {
  const { address, port } = server.address();
  console.log(`Server is listening on:
http://${address}:${port}`);
});
```

The Request Object

The `request` object in server request handlers contains information about the incoming HTTP request. Information contained within this `request` object can be used in various ways to handle server requests such as routing and data processing.

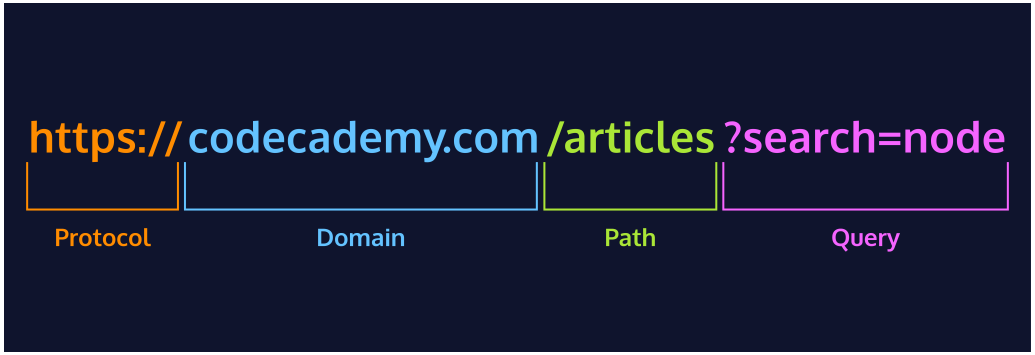
The Response Object

The `response` object in server request handlers contains information about the **outgoing** HTTP response. The `response` object contains various properties and methods that can be used to configure and send the response such as `.statusCode`, `.setHeader()`, and `.end()`.

```
const http = require('http');

const server = http.createServer((req, res) => {

  // Set status and headers
  res.statusCode = 200;
  res.setHeader('Content-Type', 'application/json');

  // Send response
  res.end('Hello World');
});

server.listen(8080);
```

Anatomy of a URL

URLs are made up of multiple parts, such as hostname, path parameters, and query string parameters. Each of these parts provides important information about a request and how it should be processed.



The `url` Module

The `url` module can be used to break down URLs into their constituent parts. It can also be used to construct URLs. Both of these actions are carried out using the `URL` class provided by the `url` module.

```
/* Deconstructing a URL */

// Create an instance of the URL class
const url = new URL('https://www.example.com/p/a/t/h?
query=string');

// Access parts of the URL as properties on the url instance
const host = url.hostname; // example.com
const pathname = url.pathname; // /p/a/t/h
const searchParams = url.searchParams; // {query: 'string'}


/* Constructing a URL */

// Create an instance of the URL class
const createdUrl = new URL('https://www.example.com');

// Assign values to the properties on the url instance
createdUrl.pathname = '/p/a/t/h';
createdUrl.search = '?query=string';

createUrl.toString(); // Creates
https://www.example.com/p/a/t/h?query=string
```

Query String Parameters

Query string parameters are used in conjunction with `GET` requests to submit information used in processing a request. A use case for query string parameters is to provide filter criteria for some requested data. While they are most commonly seen used with `GET` requests, query string parameters can also be used with other types of requests in special cases, though this is not common.

Request Body

Data is commonly provided in the `body` of a request. The `body` is most commonly used in `POST` and `PUT` requests as they usually have information that needs to be processed by the server.

HTTP Headers

HTTP headers can provide metadata that is used by servers to process requests. Frequently, servers will use a variety of standard headers to aid in processing requests. Custom headers can also be used on a per-application basis.

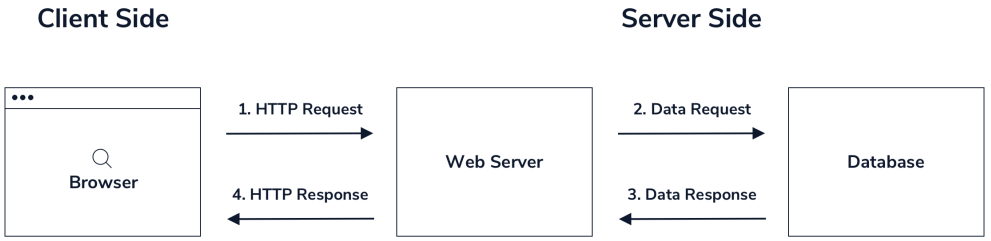
The `querystring` Module

The `querystring` module is used to decode/encode and parse query string parameters into easily usable data structures. This module only operates on query string parameters, requiring isolation of the query string before use. The core methods of the module are `.parse()`, `.stringify()`, `.escape()`, and `.unescape()`.

```
// Parse a querystring into an object with query parameter key/value pairs
const str = 'prop1=value1&prop2=value2';
querystring.parse(str); // Returns { prop1: value1, prop2: value2}

// Build a querystring from an object of query parameters
const props = { "prop1": value1, "prop2": value2 };
querystring.stringify(props); // Returns 'prop1=value1&prop2=value2'

// Percent-encode a querystring
const str = 'prop1=foo[a]&prop2=baz[b]';
querystring.escape(str); // Returns 'prop1%3Dfoo%5Ba%5D%26prop2%3Dbaz%5Bb%5D'

// Decode a percent-encoded querystring
const str = 'prop1%3Dfoo%5Ba%5D%26prop2%3Dbaz%5Bb%5D';
querystring.unescape(str); // Returns 'prop1=foo[a]&prop2=baz[b]'
```

HTTP and Databases

HTTP requests can be used to interact with databases to retrieve remote data. This enables the development of complex applications that can both read and write data to and from remote sources and is the underpinning of modern applications used today.



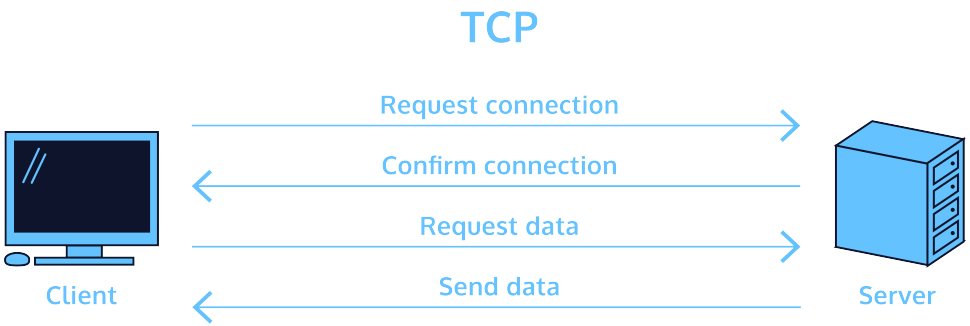
HTTP and External APIs

HTTP requests can be used to interact with other external APIs from within a server. This ability allows for communication between multiple services within the same application and is the underpinning for certain design architectures such as microservice architectures. One common way to make a request to another server is through the `.request()` method on the `http` module.

```
const options = {
  hostname: 'example.com',
  port: 8080,
  path: '/projects',
  method: 'GET',
  headers: {
    'Content-Type': 'application/json'
  }
}

// Make request to external API
const request = http.request(options, res => {
  // Handle response here
});
```

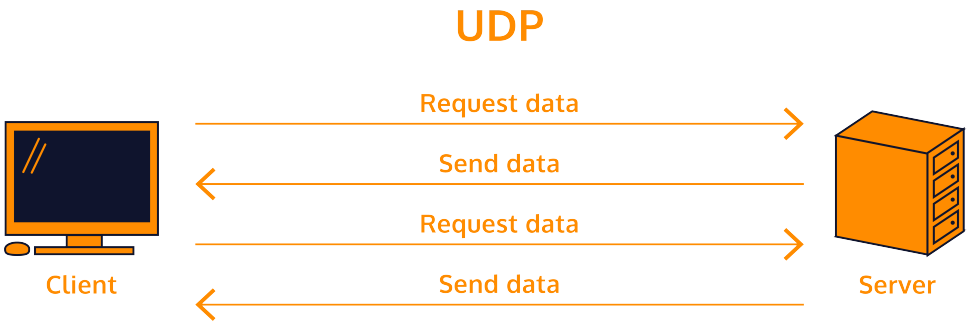
What is TCP?

Transmission Control Protocol (TCP) is a communication protocol that enables applications to exchange messages over a network and ensure the successful delivery of exchanged data packets. Due to its reliability, TCP is the favored protocol for many types of common applications.



What is UDP?

User Datagram Protocol (UDP) is a communication protocol that enables applications to exchange time-sensitive messages over a network but does not ensure successful delivery of all data packets (potential packet loss). UDP is commonly used for applications that can handle some packet loss, such as those that stream audio and video.

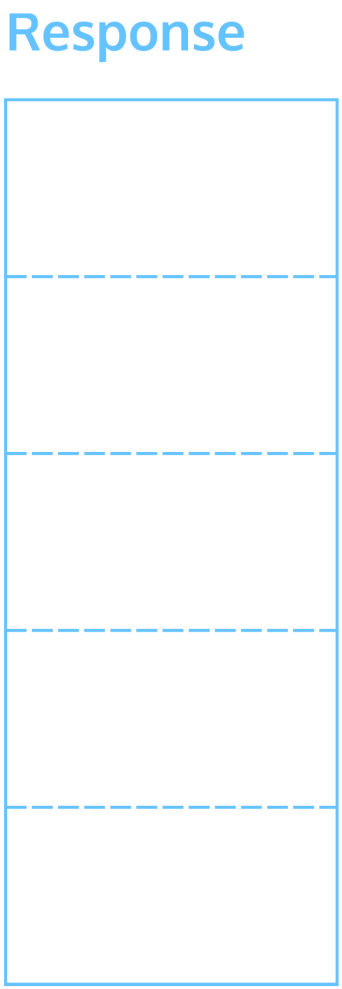


HTTP Requests

HTTP requests provide information to a server and contain HTTP method, path, HTTP protocol version, headers, and body. This information is important in processing requests.

HTTP Responses

HTTP responses contain HTTP protocol version, status code, status message, headers, and body. This information provides a detailed log to the client of what happened during the processing of the request.



HTTP Response Status Codes

HTTP response status codes indicate whether a specific HTTP request has been successfully completed. Each response status code conveys information about what happened during the processing of the request, which in turn helps the client decide how to handle the response and if further action is necessary. Status codes are paired with a short text-based description to help elucidate the meaning of the code.

```
const http = require('http');

// Creates server instance
const server = http.createServer((req, res) => {
  try {
    // Do something here
  } catch(error) {
    res.statusCode = 500; // Sets status code to indicate
server error
        return res.end(JSON.stringify(error.message));
  }
});

// Starts server listening on specified port
server.listen(4001, () => {
  const { address, port } = server.address();
  console.log(`Server is listening on:
http://${address}:${port}`);
});
```