HTTP: Hypertext Transfer Protocol

HTTP is a common language that a client and a server can use to communicate. Ex: allows the fetching of resources such as HTML documents from server

Clients and servers can communicate by exchanging individual messages.

Requests - messages sent by the client (usually web browser)

Responses - messages sent by the server

GET

I want to get an HTML file. So I ask to get something and the server responds

POST

I send over some data to Google servers and you add it to your servers or your database.

PUT

I send data to update existing data in Google servers with this new information

DELETE

delete a piece of data on the backend- on the servers or on the database.

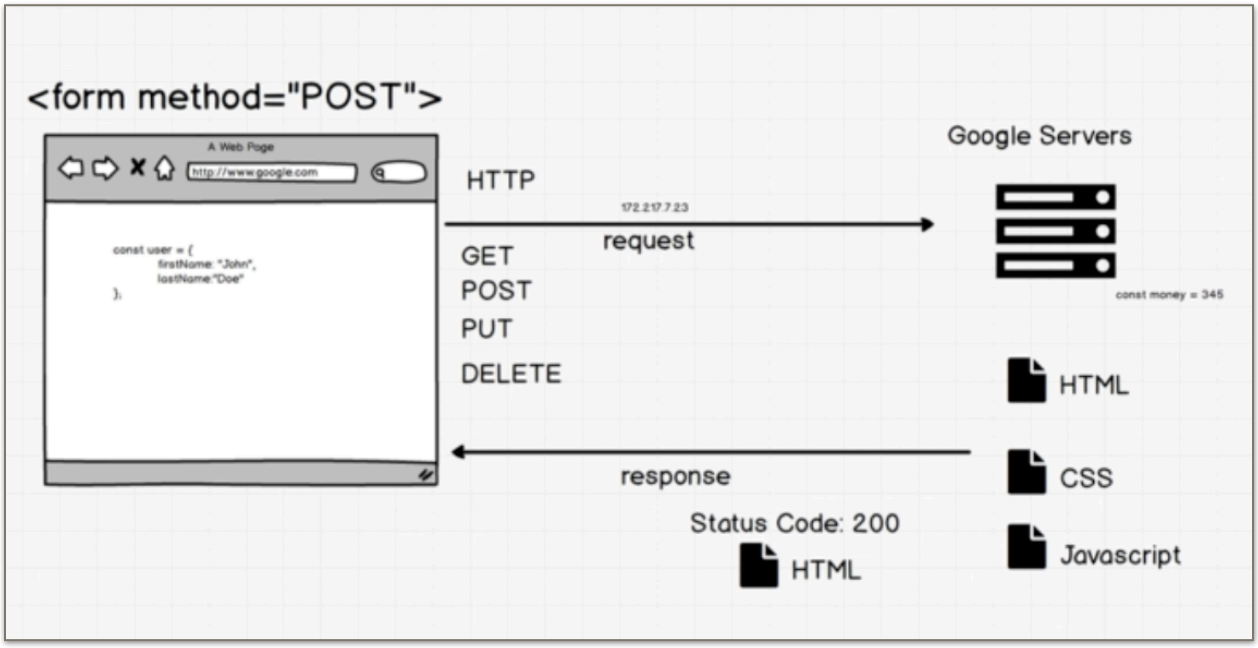
To send information to the server, two ways that you can do it:
- query string parameters using GET
- thru body of request using POST

ex: HTML form

Both of these ways mean you can see passwords, entries, etc…

HTTPS (Hypertext Transfer Protocol Secure)

Encrypted communication between the client and server, uses a technology called transport layer security or its predecessor secure sockets layer or TLS and SSL for short.



Server Response:

- HTTP message, like 200 OK, or 404 Not Found
- data, like HTML

Summary

The browser builds an HTTP request and sends it to communicate with the server.

Now the server can get information back to us. The earliest version we could use

URL parameters such as 'www.google.com/about' and just specifically get HTML.

Then came the form so we can now actually send more data other than just the URL saying we want to go to this HTML file.

Now we could use 'GET' or 'POST' to send data. Either through a body or a query string and a server can take action based on that data and return a new page.

JSON - JavaScript Object Notation

It's a syntax for storing and exchanging data. Text written with javascript object notation.

Can we post anything?

Well not really. When exchanging data between a browser and a server the data can only be text.

So we can't just send a javascript object such as a user with first name John and last name doe.

HTTP won't understand it.

And server can use any type of language like Python

JSON can be read by any language. So now we can convert this into a JSON object, send it over HTTP. And then the Google servers or

whatever it is will change it to their own language understand it and then send a response.

JSON alternative is XML. JSON format is a more succinct way which saves bandwidth and improves response times when sending messages back and forth between client and server.

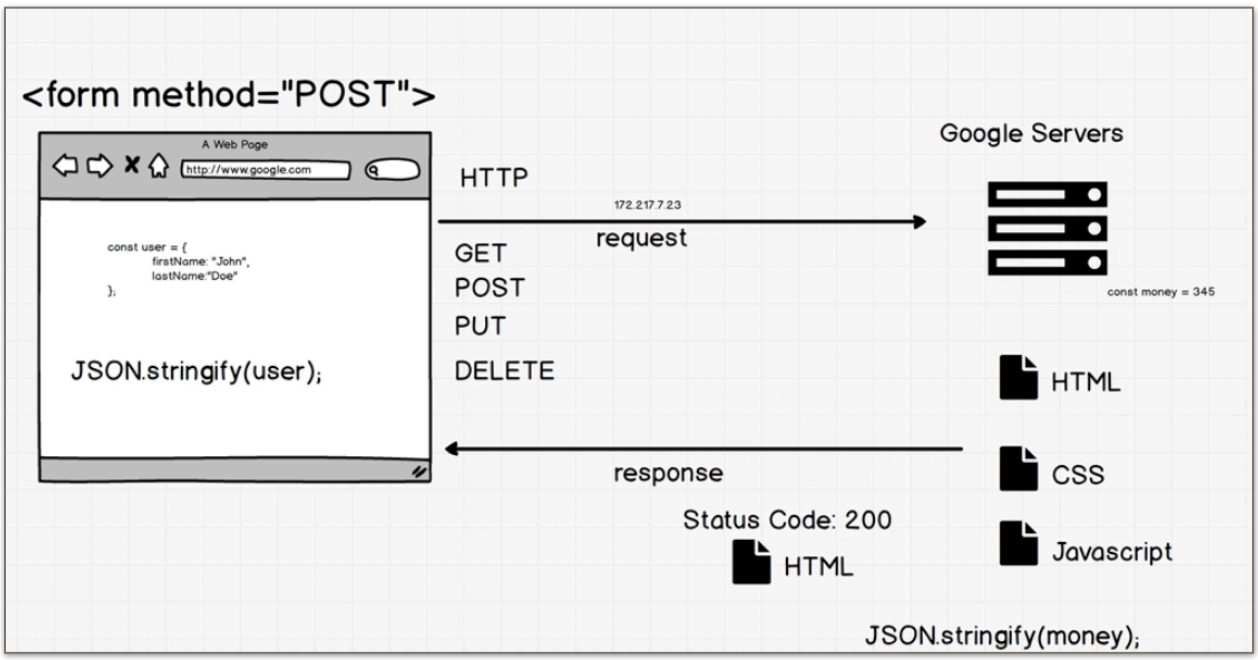
XML uses this HTML like syntax while JSON uses a javascript object like syntax.

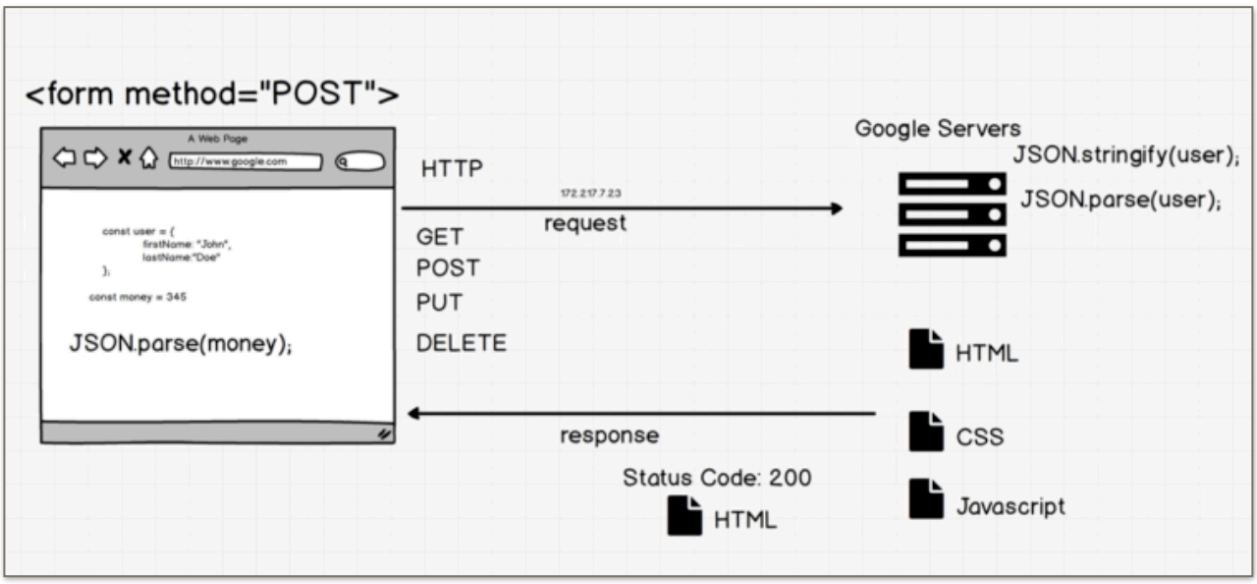
JSON.parse( ) - JSON to Object

```
JSON.parse()

var obj = JSON.parse('{ "name":"John", "age":30, "city":"New York"}');
```

JSON.stringify( ) - Object to JSON

```
JSON.stringify()

var myJSON = JSON.stringify(obj);
```





JSON Example:

now with this complete User object before I send it with HTTP I will do "JSON.stringify(user);"

So now its converted into a JSON string and sent over HTTP to the Google servers.

It is then going to say JSON.parse this user so it understands it.

Lets say it was running Javascript or Node.

Now it'll understand the user. They'll say - oh he wants the money amount for this user.

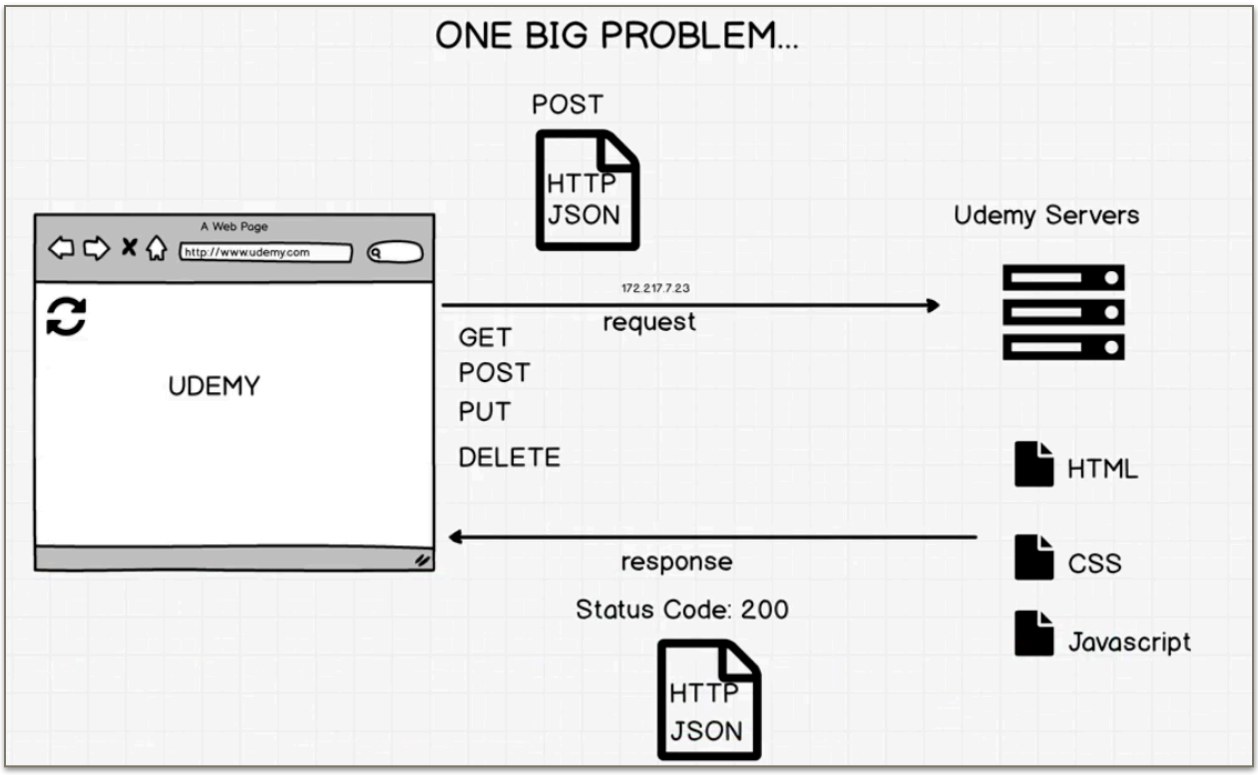
So now it gets money 345 from the account. It stringify's the money variable. Sends it with a status code of 200

And now the web browser can parse this string that it received and see that I have a variable money of $345.

JSON vs Form Data

Originally, the only way to submit some form data to a server was through the <form> tag in HTML. As we have learned, it can do a POST or a GET request.

With JSON you can now grab the contents of the <input> in a form and submit those with JSON instead of as a form data. You can now submit to the server whenever you want without it necessarily being a <form>, through AJAX.. What is AJAX you might say?



AJAX - so entire page doesn't have to be refreshed

Asynchronous JavaScript and XML

allow web pages to request small chunks of data such as HTML, XML, plain text, JSON and display them only when needed.

Updates a web page without reloading the page. Sends data in the background while the user is interacting with the Website.

The Old Way: XHR

```
var request = new XMLHttpRequest();
request.open('GET', '/my/url', true);

request.onload = function() {
  if (request.status >= 200 && request.status < 400) {
    // Success!
    var data = JSON.parse(request.responseText);
  } else {
    // We reached our target server, but it returned an er

  }
};

request.onerror = function() {
  // There was a connection error of some sort
};

request.send();
```

The New Old Way: jQuery

```
$.getJSON('/my/url', function(data) {

});
```

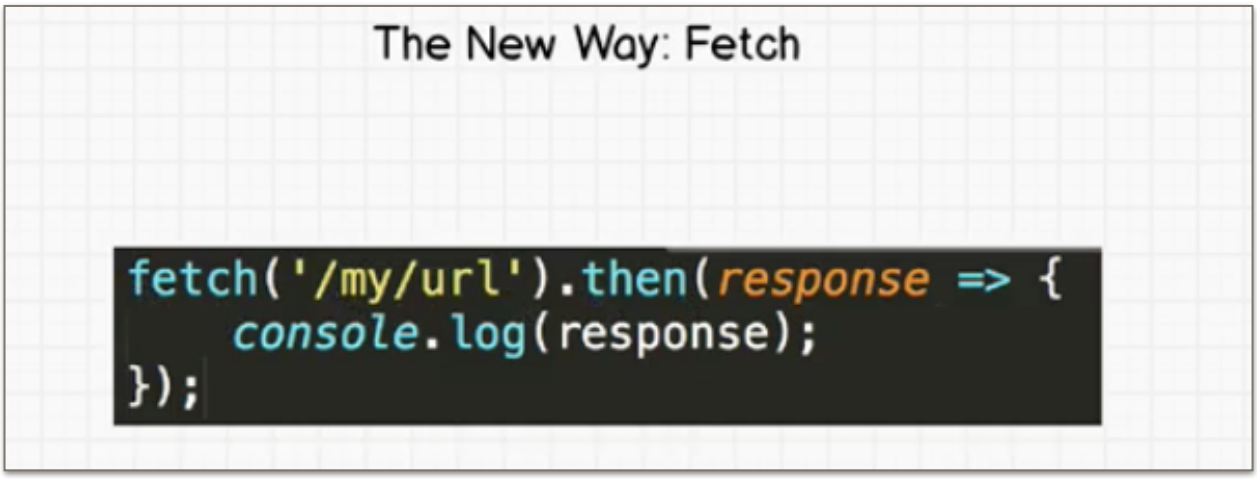
So let's review again what happens with Ajax. An event occurs on a web page such as logging in and I click sign in. XML HTTP request object (something that web browsers have implemented) is created using javascript.

XML HTTP request object sends a request to the web server the server processes the request and then the server sends a response back to the web page.

The response is read by javascript and the user is able to login. At the same time only updating a small portion of the window.

That is what a single page application is load a base an almost empty page and build the content on the fly based on the data fetched from the server.

These applications nearly never do a full reload, they destroy the previous content (all or a part of it) and rebuild it based on new data



Make AJAX request using Fetch API + JSON

Fetch is part of the window object: 'window.fetch'

fetch is a function that we can use and this URL if you remember returns for us- a JSON object.

The `fetch()` method returns a `Promise` that resolves the `Response` from the `Request` to show the status (successful or not). If you ever get this message `promise {}` in your console log screen, don't panic — it basically means that the `Promise` works, but is waiting to be resolved. So in order to resolve it we need the `.then()` handler (callback) to access the content.
So in short, we first define the path (**Fetch**), secondly request data from the server (**Request**), thirdly define the content type (**Body**) and last but not least, we access the data (**Response**).

```
> fetch('https://jsonplaceholder.typicode.com/users')
<· ▶ Promise {<pending>}
```

```
> fetch('https://jsonplaceholder.typicode.com/users')
  .then(response => console.log(response));
<· ▶ Promise {<pending>}

                                              console.js:35
  Response {type: "cors", url: "https://jsonplaceho
▶ lder.typicode.com/users", redirected: false, stat
  us: 200, ok: true, …}
```

And for this JSON object, fetch allows us to do something called response - 'response.json()'.

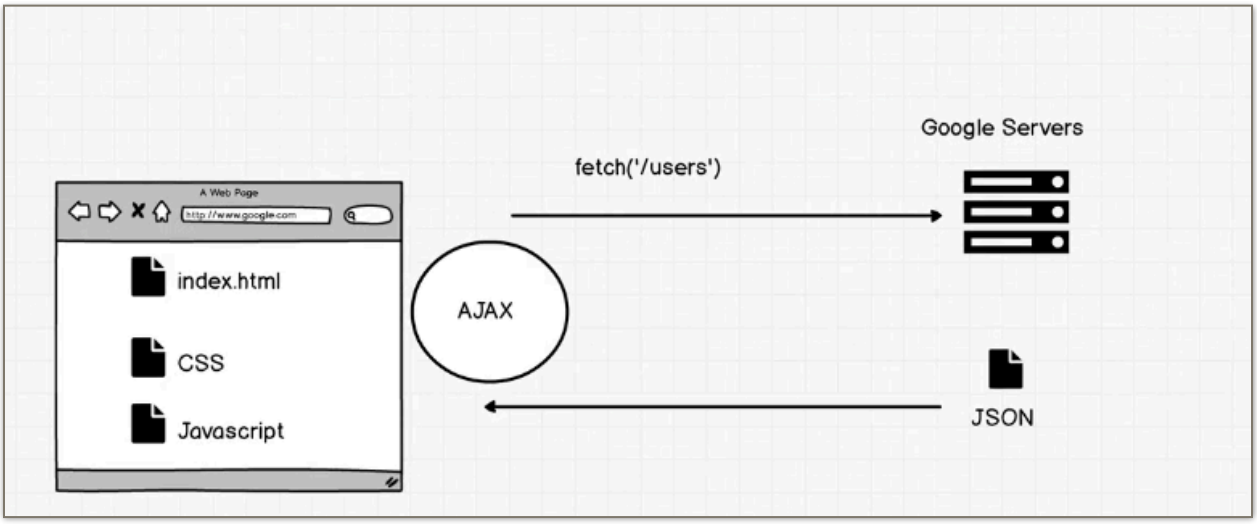
```
> fetch('https://jsonplaceholder.typicode.com/users')
  .then(response => response.json());
<· ▶ Promise {<pending>}
```

```
> fetch('https://jsonplaceholder.typicode.com/users')
  .then(response => response.json()).then(data =>
  console.log(data));
<- ▶ Promise {<pending>}
                                                  console.js:35
▼ (10) [{…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}
  , {…}] i
  ▶ 0: {id: 1, name: "Leanne Graham", username: "Br…
  ▶ 1: {id: 2, name: "Ervin Howell", username: "Ant…
  ▶ 2: {id: 3, name: "Clementine Bauch", username: …
  ▶ 3: {id: 4, name: "Patricia Lebsack", username: …
  ▶ 4: {id: 5, name: "Chelsey Dietrich", username: …
  ▶ 5: {id: 6, name: "Mrs. Dennis Schulist", userna…
  ▶ 6: {id: 7, name: "Kurtis Weissnat", username: "…
  ▶ 7: {id: 8, name: "Nicholas Runolfsdottir V", us…
  ▶ 8: {id: 9, name: "Glenna Reichert", username: "…
  ▶ 9: {id: 10, name: "Clementina DuBuque", usernam…
    length: 10
  ▶ __proto__: Array(0)
```

If I copy and paste this fetch I get something called a promise

Promise is saying "hey I'm making a request to somewhere over the Internet and I promise to let you know when I have this value returned."

So the way you access promise is - so you have this: 'once this is returned



```
// Basic blueprint
fetch(url)
.then(response.something) // Define response type (JSON, Headers, Status codes)
.then(data) // get the response type

// Practical example
fetch('https://jsonplaceholder.typicode.com/todos')
.then(response => response.json())
.then(data => console.log(JSON.stringify(data)))
```