Learn Git and GitHub without any code!

Using the Hello World guide, you'll start a branch, write comments, and open a pull request.

Read the guide

SEI-CC / **SEIR-11-08-21**

Code | Issues | Pull requests 1 | Projects | Wiki | Insights

main

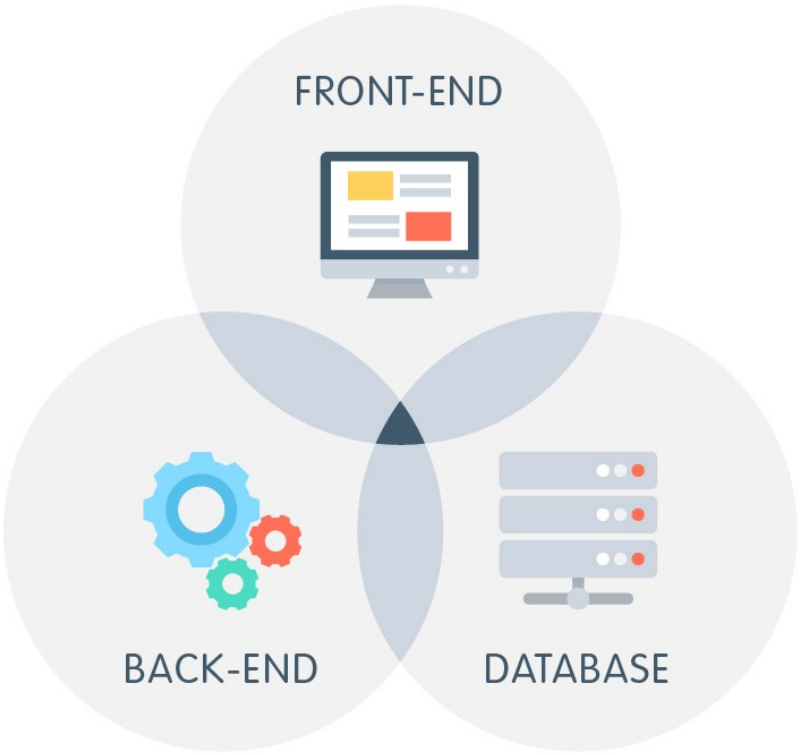
SEIR-11-08-21 / work / w04 / d4 / **01-intro-fullstack-http.md**

cogilvy-ga Update 01-intro-fullstack-http.md — History

1 contributor

Raw | Blame

Executable File 257 lines (150 sloc) 11.3 KB



Intro to Full-Stack Development & HTTP

Learning Objectives

Students Will Be Able To:
Explain Full-Stack Web Development
Describe Client/Server Architecture
Explain the Anatomy of HTTP Requests and Responses
Identify the Two Key Components of an HTTP Request
Explain How HTTP Requests Can be Initiated From a Browser

Students Will Be Able To:
Explain How HTTP Requests Result in Code Running on a Server

Road Map

- Why Learn Full-Stack Web Development?
- What is a Full-Stack Developer?
- SEI Web Technology Stacks
- Client/Server Architecture
- What is HTTP?
- Let's Make an HTTP Request
- Anatomy of an HTTP Request/Response Messages
- The Two Key Components of an HTTP Request
- HTTP Methods
- Ways to Send HTTP Requests From the Browser
- How HTTP Requests Run Code on the Server
- Essential Questions

Full-Stack Web Development

Why Learn Full-Stack Web Development?

- According to U.S. News the second best job in the nation to have is that of a **Software Developer**!
- Further, according to this StackOverflow Developer Survey, **the role of Full-stack (Web) Developer is by far the most popular** as compared to other developer roles.
- The bottom-line is: **Web Developers are in demand**!

What is a Full-Stack Developer?

A full-stack developer:

- Is a developer who is comfortable working with both frontend & back-end technologies.

- Can create full-stack applications by writing code that runs in both a client, such as a browser, and a web server.
- Will often specialize in frontend **or** back-end technologies.
- Is a graduate of SEI!

Web Technology Stack - Unit 2

In this unit, we'll learn 3 of the 4 technologies that comprise the MERN-stack:

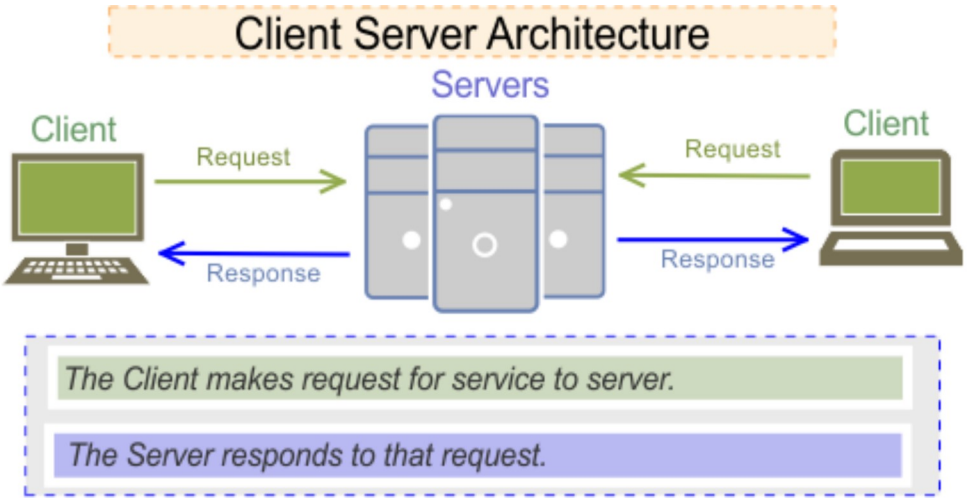
Web Technology Stack - Unit 3

In Unit 3, we'll learn:

Web Technology Stack - Unit 4

In Unit 4, we go full MERN-stack:

Client/Server Architecture

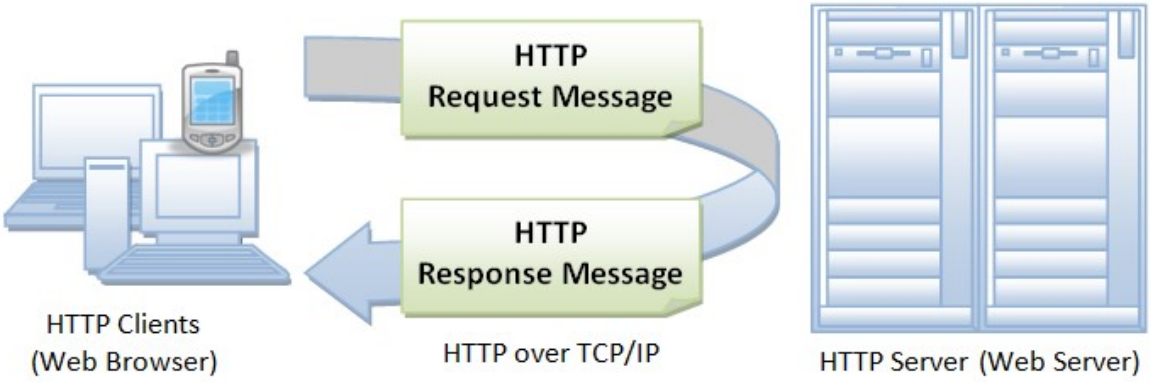


- The terms **client** and **server** can refer to both a **physical device** (computer, tablet, phone, etc.) but can also refer to a **software process**. For example:
    - Database software such as PostgreSQL and web servers like Apache are examples of software processes behaving as servers.
    - Browser software such as Chrome or Firefox are examples of software clients.
- Physical **servers** connected to the Internet are also referred to as **hosts**.
- Web developers usually think of a "web browser" when they hear "client".
- Note that during development, your computer will plays the role of BOTH client and web server.
- The PostgreSQL & MongoDB database servers will also be running on your computer, however, we will move to a cloud-based MongoDB server as soon as it's practical.

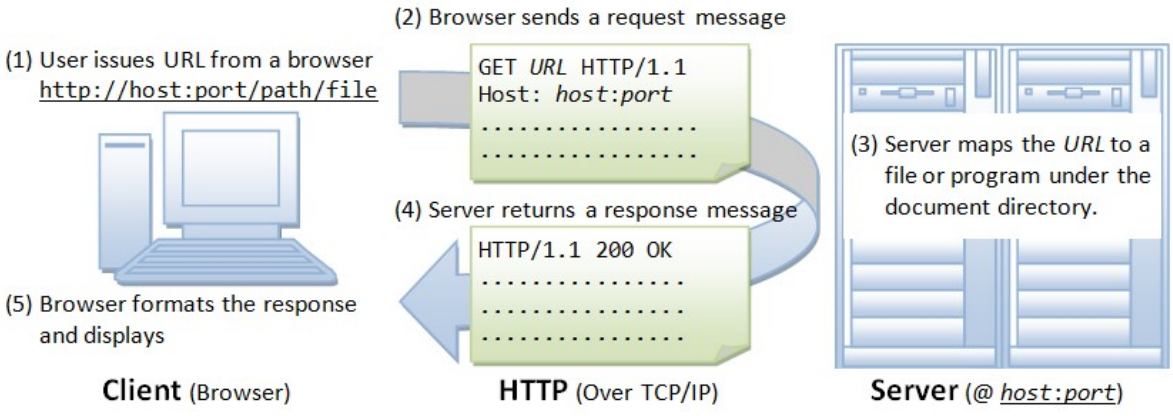
HTTP

What is HTTP?

- **Hypertext Transfer Protocol (HTTP)** is an application-level network protocol that powers the communications across the World Wide Web, more commonly referred to as just **the Web**.
- **HTTP is fundamental to web development** - regardless of which back-end or front-end web technology/framework is used.
- When a user interacts with an amazing **web application** we developed, it's **HTTP** that informs the **web application** what the **browser** wants and it's **HTTP** that delivers the goods from the server back to the browser.
- The process of a client sending a HTTP request, and server responding is known as the **HTTP Request/Response Cycle**:



- When we browse to a website by typing in the address bar, this is what happens:

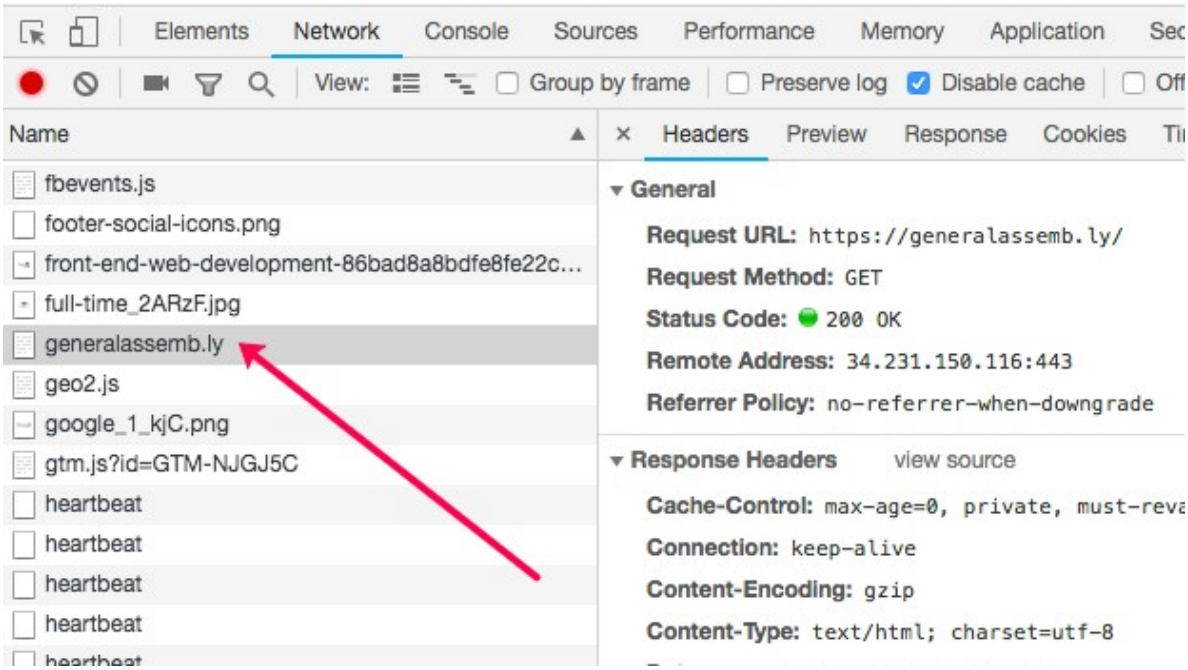


- When the response is received by the client, that request/response cycle has ended and there will be no further HTTP communications unless another request is sent by the client.
-

HTTP itself does not maintain any information regarding previous requests between client and server - this makes HTTP a *stateless* protocol. However, it is possible to remember "state" using cookies or by sending data in the request's body (data payload).

Let's Make an HTTP Request

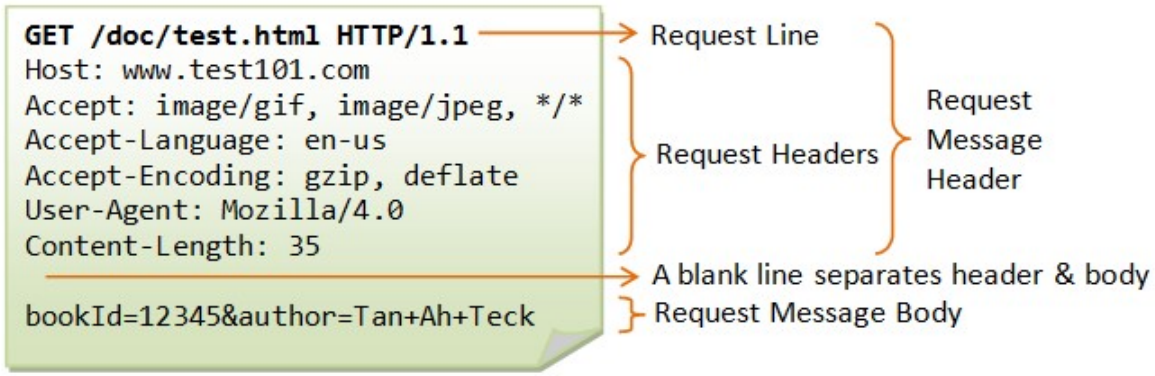
- Let's open a new tab in Chrome, open DevTools, and click on the *Network* tab where we can inspect HTTP requests and responses.
- Now let's browse to General Assembly's site by typing **generalassemb.ly** in the address bar...
- Wow! Each one of those lines represents a separate HTTP request made to a server! Find the line in the left-side pane with **generalassemb.ly** and click on it:



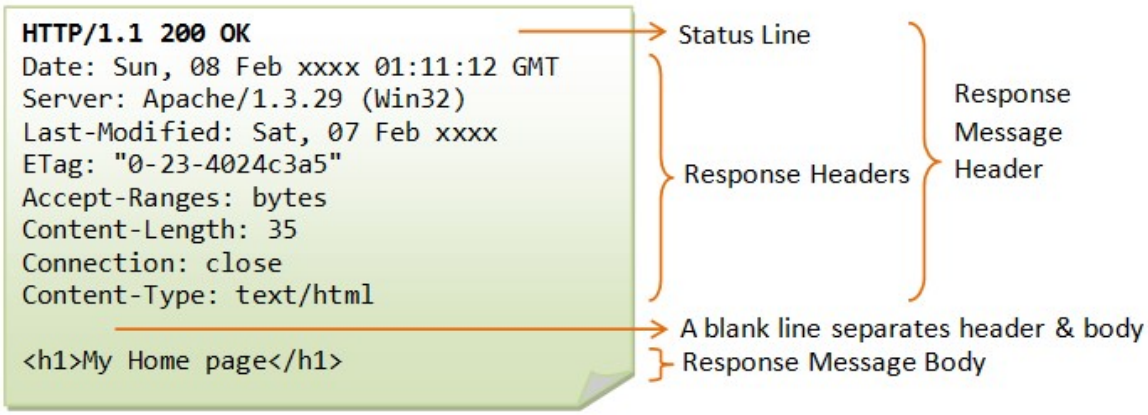
- In the pane on the right you will find all of the information about a particular HTTP request. Select the *Headers* tab and explore!:

Anatomy of HTTP Request/Response Messages

- Use the following diagram to review the **Request Headers** sent to the server when browsing to *generalassemb.ly*:



- The **GET** request we just sent does not contain a *message body* nor a *Content-Length* header because...
- The **Request Body** is optional and present only when the client is sending data to the server.
- An example of when the **body** might contain data is when the contents of an HTML form are being submitted by the user.
- Now use the following diagram to review the **Response Headers** returned by the *generalassemb.ly* server:



You can view the **response body** by clicking the **Response** tab.

- The status code in the first line informs us on how the request/response went. It is always a three-digit number that falls within the following ranges/categories:
    - 1xx Informational
    - 2xx Success
    - 3xx Redirection
    - 4xx Client Error
    - 5xx Server Error
- Most HTTP responses will have a *status code* of `200`, which means *OK*. You also might be familiar with the *status code* of `404` - *Not Found*.
- The **Response Body** is what holds the content of the requested resource.
- The **Content-Type** header is a MIME type and helps the browser determine what to do with the data being sent in the **body** of the HTTP response. For example:
    - **text/html**: The browser will parse the **body** as HTML and, depending on how the HTTP request was initiated, usually **replace** the browser window's content with the newly received HTML.
    - **image/png**
- Although the HTTP protocol is text-based, the content in the body can be binary, for example, images are typically transferred in binary format.

The Two Key Components of an HTTP Request

- We saw earlier that every HTTP request message begins with a request-line like this:

```
GET /sample_page.html HTTP/1.1
```

- **The two key components of any HTTP request are**:
  - The **HTTP method** (`GET` in the example above), and
  - The *request target*, which is usually a **URL** (the above example is more accurately a **URI**)
- The reason these are the key components are because most web frameworks use them to match routes defined on the server (more on routes in a bit).

HTTP Methods

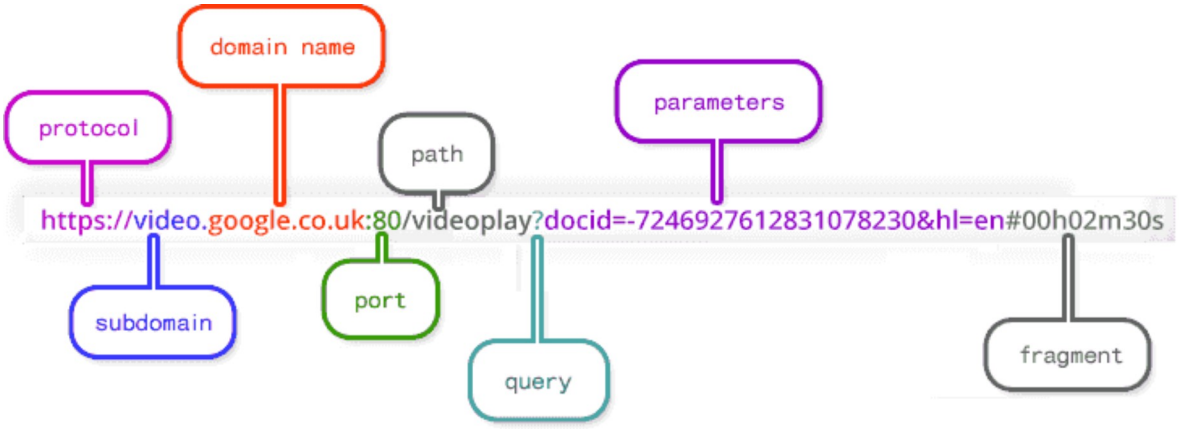
- HTTP methods, are used to indicate the desired **action** to be performed for a given resource (specified by the URL) on the server.
- The fact that they indicate **action** is why they are also at times called **HTTP Verbs**.
- We'll be using the following HTTP Methods when we start defining our application's routes:

HTTP Method (Verb)	Desired Action on Server
GET	The GET method requests a representation of the specified resource (URL). Requests using GET should only retrieve data.
POST	The POST method is used to create a resource on the server.
PUT	The PUT method replaces a resource with the request payload (data in the body).
DELETE	The DELETE method deletes the specified resource.

URL

- **URL** stands for **Uniform Resource Locator**.
- It informs the server of what resource the client wants to GET. create (POST), DELETE, etc.

- Here's the complete anatomy of a URL:



- Developers often refer to just the **path** (incorrectly) as a URL.

Ways to Send HTTP Requests From the Browser

- These are ways that a browser can send an HTTP Request:

Action	HTTP Method(s) Possible
Using the *address bar*	GET
User submits an HTML form	POST, GET (used for searches)
User clicks a link	GET
Using JavaScript	All Methods can be sent using AJAX

Until we get to Unit 4, our apps will rely on the user clicking links and submitting forms to interact with the app - that's it!

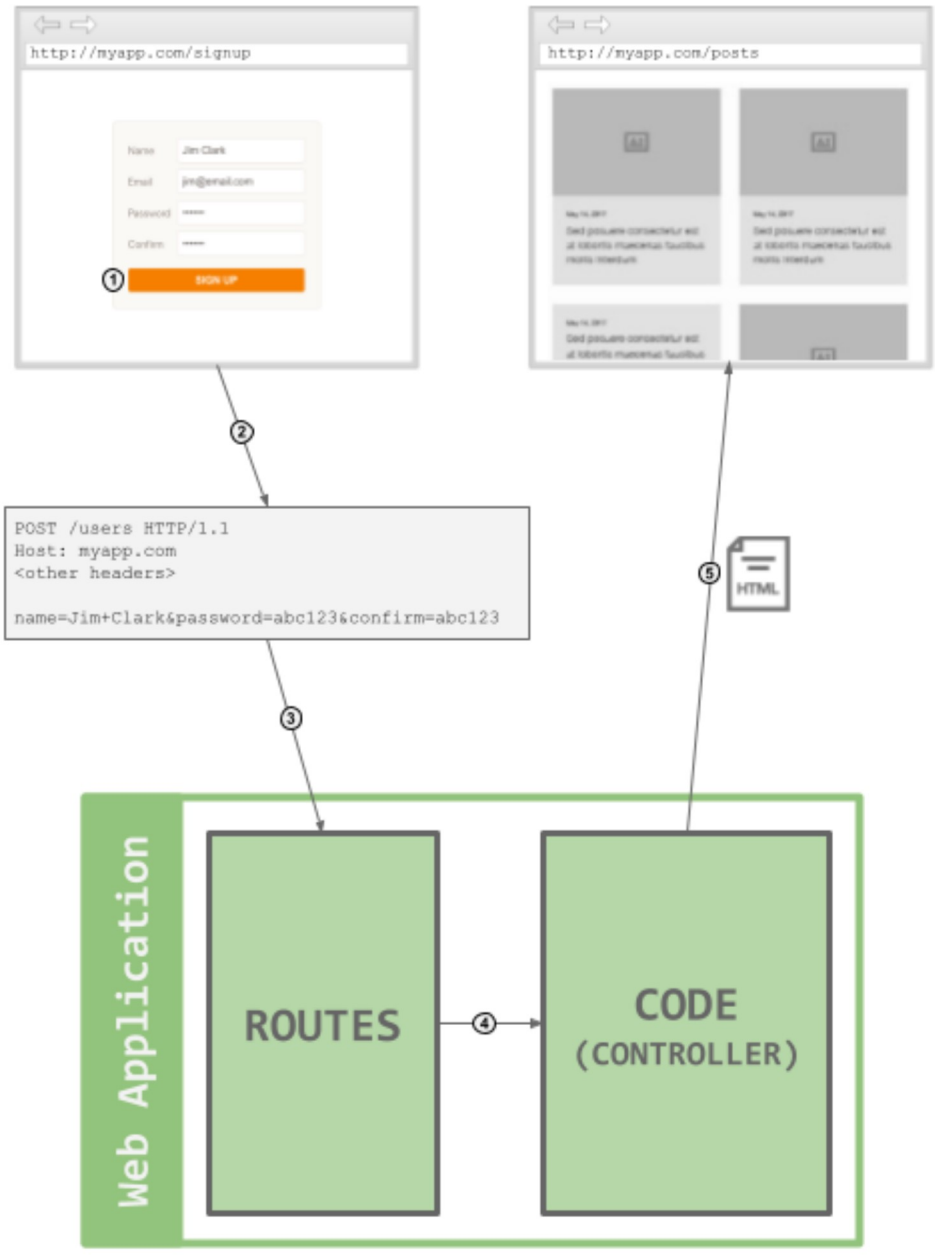
Ways to Send HTTP Requests Without Using the Browser

- There are plenty of developer tools that enable us to issue HTTP requests to servers without using a browser.
- One tool we use later in the course is Postman.
- If you prefer to use a command-line tool, our computers already have a tool called cURL.

How HTTP Requests Run Code on the Server

- As full-stack web developers, we'll create front-ends that display in the browser.

- As a user interacts with the app by clicking links and submitting forms, they will cause HTTP requests to be sent to our back-end web application.
- In our web app on the back-end, we will create **routes** that listen for these requests from the front-end, and...
- Those **routes** will map HTTP requests to our back-end **code**!
- As an example, you want your app to add a new user to your database when they sign up...



1. The user submits the sign up form.
2.

An HTTP request message with the form's data in the request body leaves the browser.

3. The HTTP request is received by the web app's routing and, in this case, would match the route defined to match a `POST /users` (**HTTP Method** & **Path**) HTTP request.
4. The purpose of a defined route is to map an HTTP request to code which is typically a function defined in the "controller" module. The controller function would perform any necessary data operations, etc.
5. The controller function ultimately responds with an HTTP Response which can either:
    - Containing an HTML page in the message body (in the case of a GET request)
    - Or, with a Status Code of 302 (Redirect), make the browser issue a new GET request.

❓ Essential Questions

1. What is the network protocol of the web?
2. What is the **body** of an HTTP request/response used for?
3. Name three HTTP methods/verbs.
4. What are the two key components of an HTTP Request that are typically used to match a route on the server?