

From Web to Web Scraping

Day 3: 3DC Introduction to Programming

Raphael Yee

How does the web work?

How do we retrieve and send information?

What tools do we use to exchange information?

How do these programs communicate?

How do humans accomplish the tasks above?

**What the web pretty much boils down to,
is an exchange of information.**

Choosing what to do with the information is entirely up to us.

First, we need to figure out how to move this information around.

Push or pull?

Are we retrieving information or are we sending information?

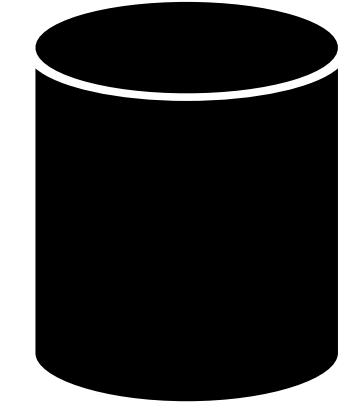


Sending an email



Loading a video

Think: What are the notifications you receive on your mobile phone called? Why?



Before I can watch a video, first I have to ask for the video.

If my **request** for the video succeeds,
then I can display the video.

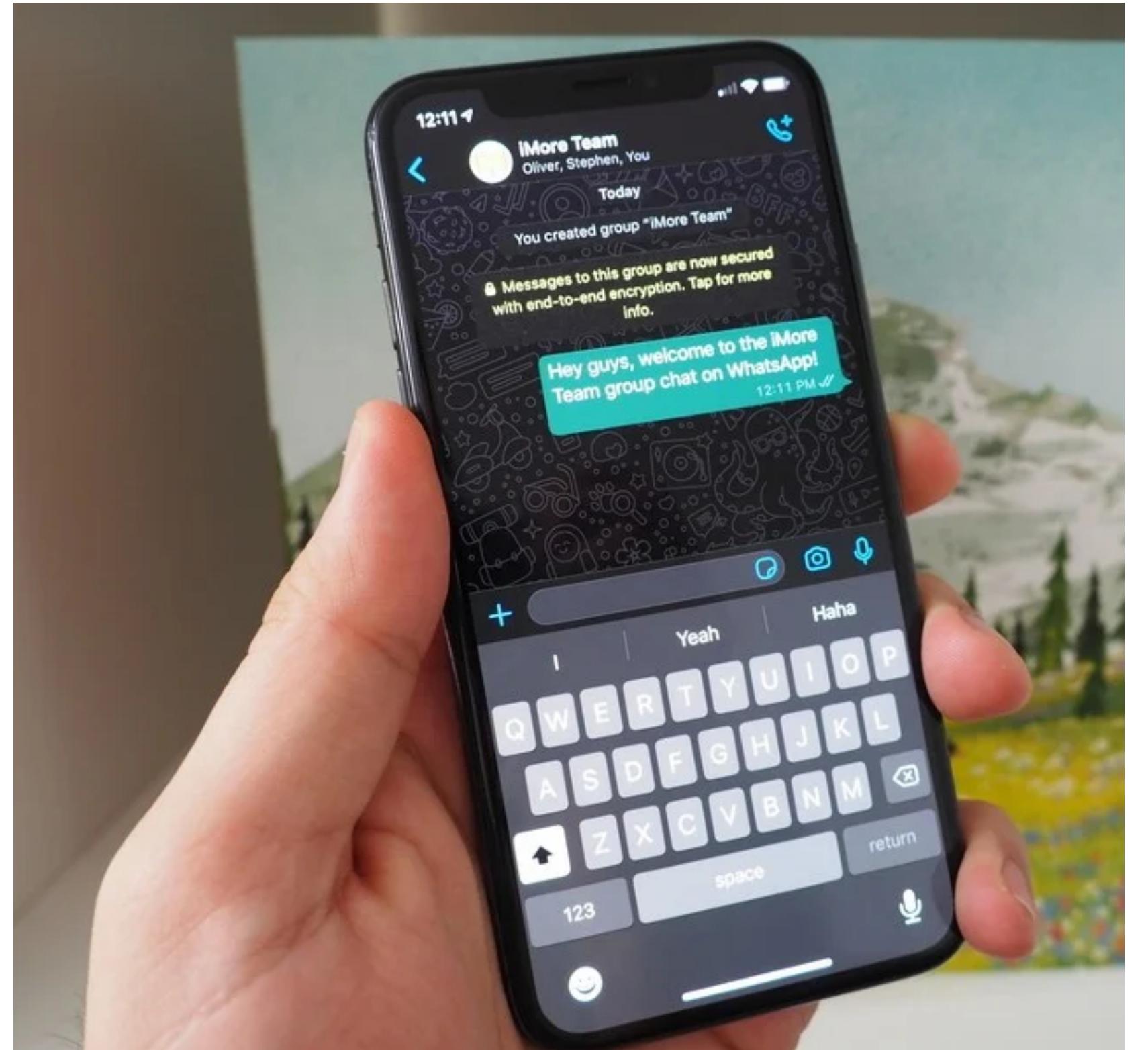
The video displays a series of handwritten calculus integrals solved on a whiteboard:

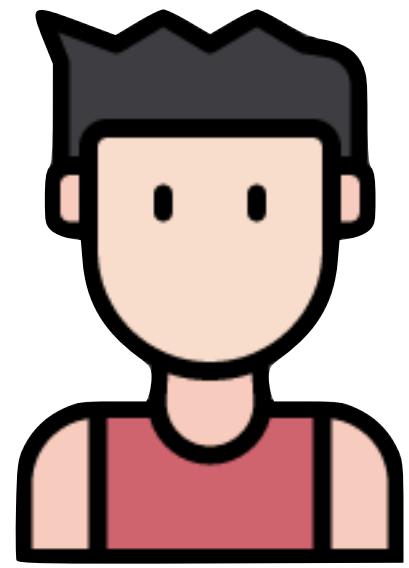
- (2) $\int \frac{\cos(2x)}{\sin x + \csc x} dx$
Simplifies to $\int \frac{(\cos x - \sin x)(\cos x + \sin x)}{\sin x + \csc x} dx$
Further simplifies to $\int (\cos^2 x - \sin^2 x) dx$
Final result: $\int (\cos x + \sin x) dx = \sin x + \cos x + C$
- (3) $\int \frac{x^2+1}{x^4-x^2+1} dx$
Substitution: $u = x - \frac{1}{x}$, $du = (1 + \frac{1}{x^2}) dx$
Simplifies to $\int \frac{1 + \frac{1}{x^2}}{x^2 - 1 + \frac{1}{x^2}} dx = \int \frac{1}{u^2+1} du$
Final result: $\int \frac{x^2-2+\frac{1}{x^2}+1}{(x-\frac{1}{x})^2+1} dx = \tan^{-1}(x-\frac{1}{x}) + C$
- (4) $\int (x+e^x)^2 dx$
Expands to $\int (x^2 + 2xe^x + e^{2x}) dx$
Final result: $\int (\frac{1}{3}x^3 + 2xe^x - 2e^x + \frac{1}{2}e^{2x}) dx$
- (5) $\int \csc^3 x \sec x dx$
Substitution: $u = \sin x$, $du = \cos x dx$
Final result: $\int \frac{\csc^2 x}{\sin^3 x \csc x} dx = \int \frac{\csc x}{\sin^2 x} dx = \int \frac{1}{\sin x} dx = \ln|\sec x| + \ln|\sin x| - \frac{1}{2}$

100 INTEGRALS (World Record?)
3,075,425 views • Mar 4, 2019
107K likes 2.3K dislikes SHARE SAVE ...
blackpenredpen © SUBSCRIBE

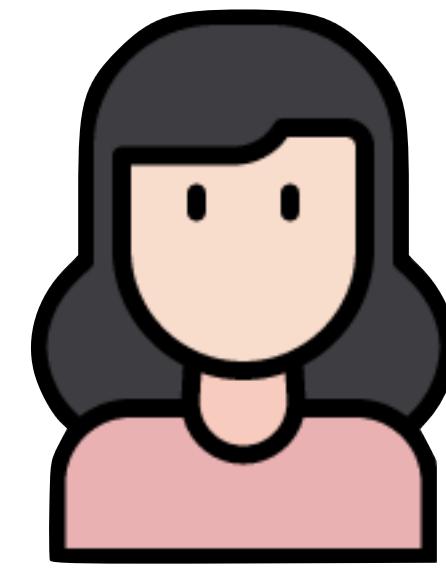
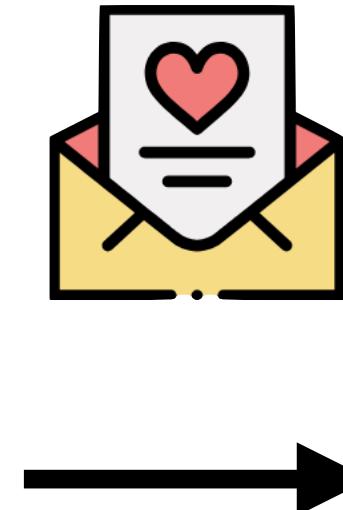
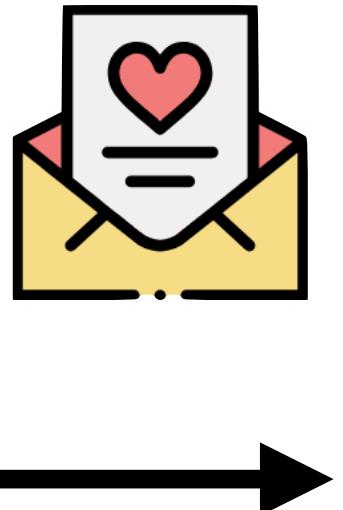
To speak to my friend remotely, I need to **send** a message.

In order to send a message, I need to know **where** to send it and **how** to send it.





Ben



Alyssa

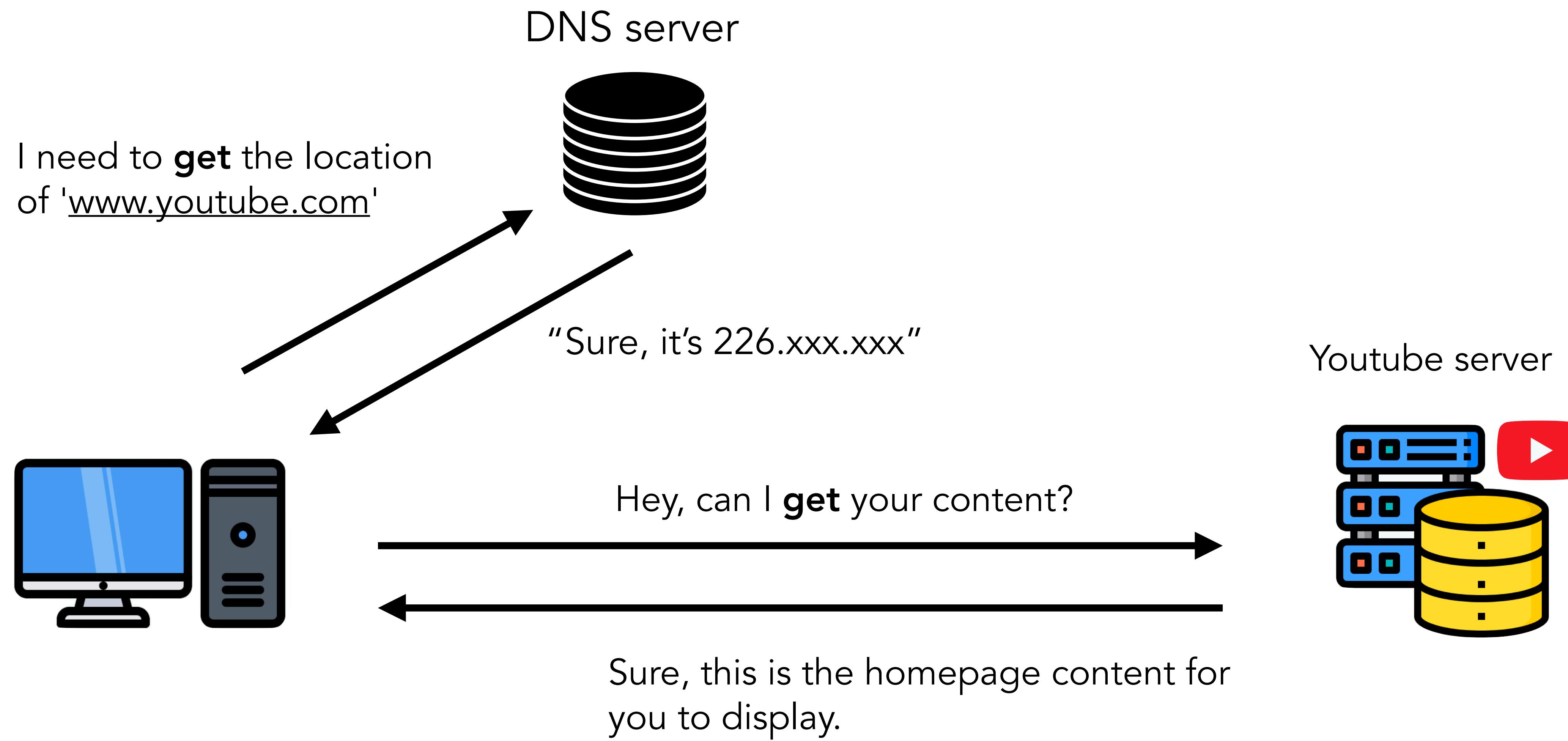
What do we need when sending/receiving mail?

Address...

Transportation...

Security...

Roads...



Think: In which order are these processes executed? Are these pull/push requests? Now do you know why it's called an 'IP address'?

Request/Response

A **request** is a question and a **response** is an answer.

The answer may have useful information or it may not.
Some requests may not get an answer...

...in which case we say that the requested resource
is **not found**.

Note: Responses are **not** “push” requests; you do not *initiate* a response. It is useful to think of it that once a request is sent, it “picks up” the response on its way back to the client.



A **request** is initiated by the client
and a **response** is the answer to the
client's request.

Think: Why do some websites return Error 404?

What makes a request?

There are different kinds of ways to move data around (transport protocols) but for the sake of simplicity we will focus on (RESTful) HTTP requests.

<u>Type of Request</u>	<u>Scheme</u>	<u>URL</u>	<u>Payload</u>
• GET	• HTTP	• Scheme	• The data you want transferred
• POST	• HTTPS	• Domain	
• PUT		... other info	
• DELETE			

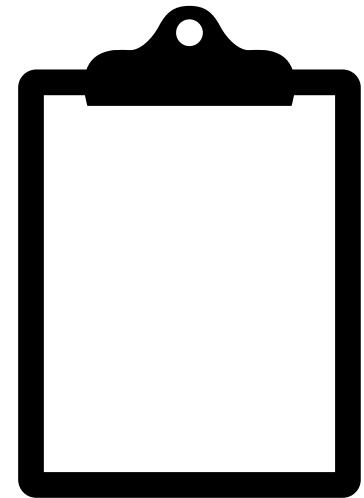
Think: What makes a response?

An analogy

The police wants to transport important packages from one place to another.

URL

<https://www.abc.com>



Okay go to abc address and make sure the package secure

Scheme

HTTPS



Cannot see what's inside

HTTP



Can see what's inside

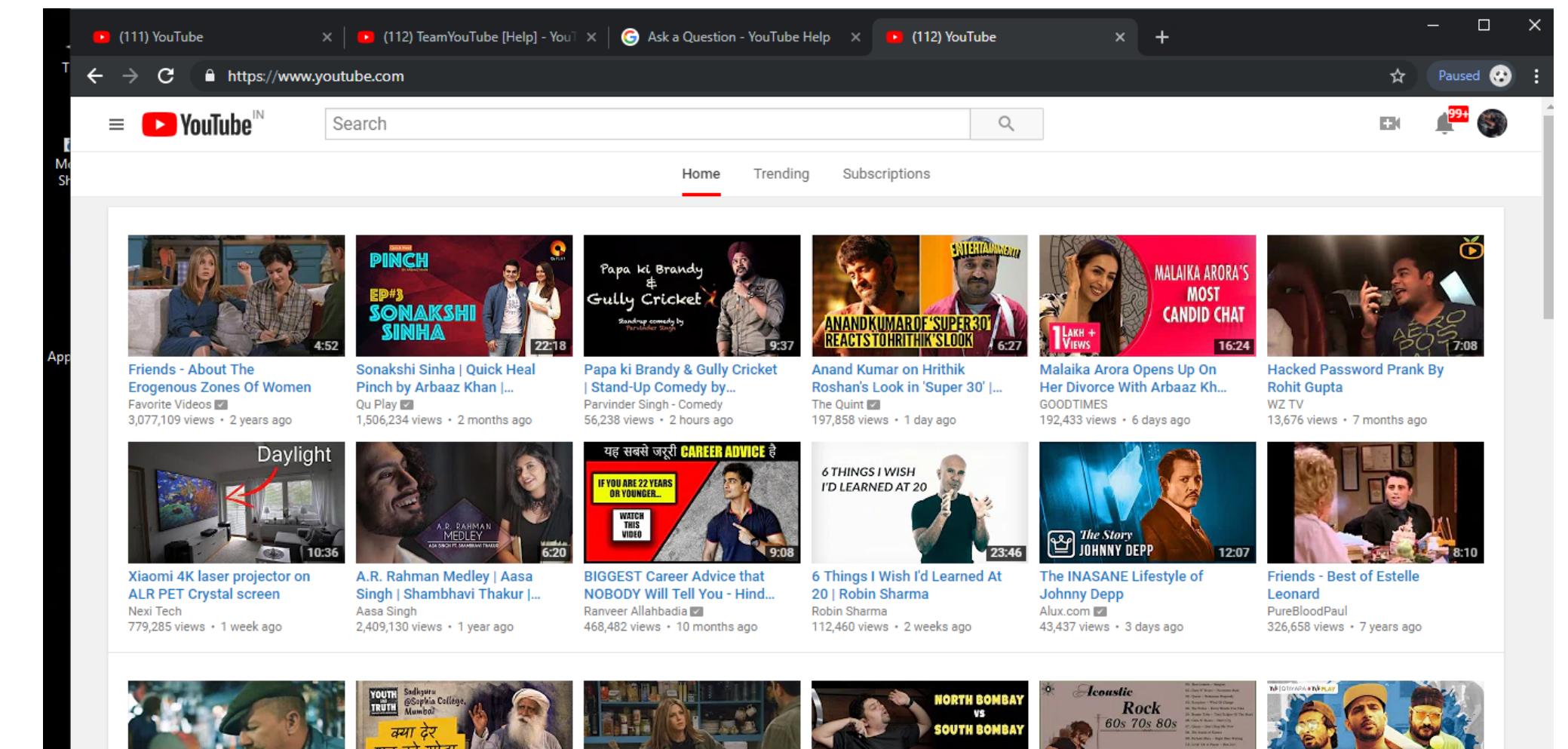
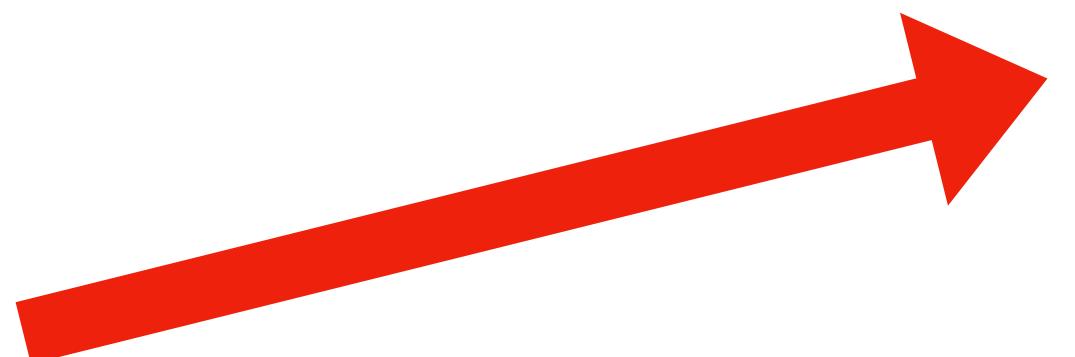
Think: What is the payload?

Putting it all together



Putting it all together

With the information the Youtube server provides us,
the browser does some magic...



What is HTML?

According to [W3C](#):

HTML (the Hypertext Markup Language) and CSS (Cascading Style Sheets) are two of the core technologies for building Web pages. HTML provides the structure of the page, CSS the (visual and aural) layout, for a variety of devices. Along with graphics and scripting, HTML and CSS are the basis of building Web pages and Web Applications.

Simply put, HTML contains the instructions on how to structure a web page and CSS provides information on styling (eg. colour, width, height...).

What is HTML?

<body>

<p>

Hello this is a piece of text

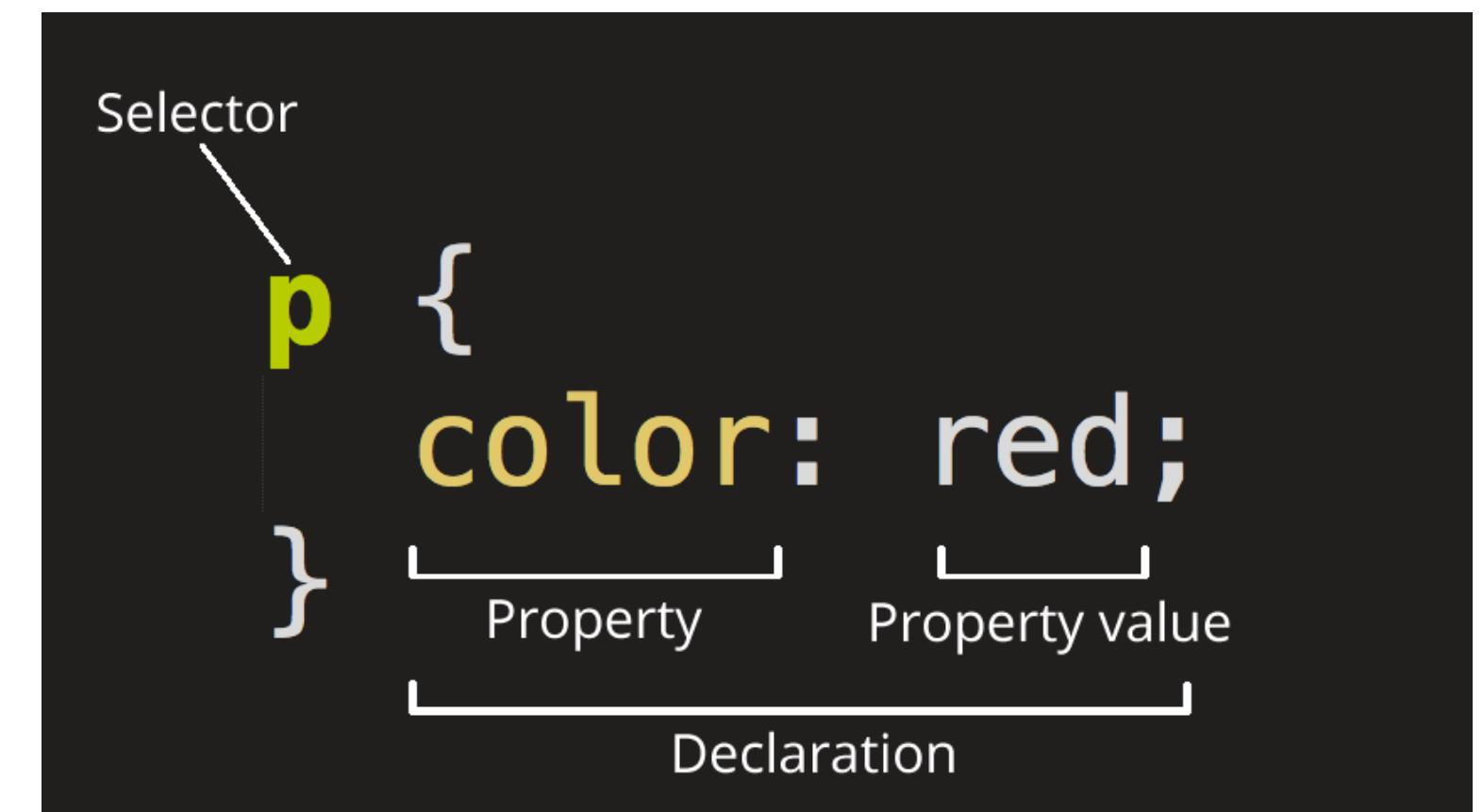
</p>

</body>

Hello this is a piece of text

CSS examples

Hello this is a piece of red text



Let's play with requests

Activity #1

What is Postman?

Postman is an API client that provides a GUI interface to make server requests.

The screenshot shows the Postman application interface. The top navigation bar includes 'New', 'Import', 'Runner', 'Filter', 'Education Program', 'Invite', and 'Upgrade'. The main area displays a collection named 'Assignments' with five requests: 'Get assignments', 'Get submissions', 'Post assessment', 'Update assessment', and 'Remove award'. The 'Remove award' request is selected, showing a 'DELETE' method and the URL 'postman-echo.com/delete?id={{award_id}}'. The 'Params' tab is active, showing a table with a single row: 'id' with value '{{award_id}}'. The 'Body' tab shows a JSON response with various headers and parameters. The bottom status bar indicates 'Status: 200 OK', 'Time: 598 ms', and 'Size: 786 B'. The bottom right corner features icons for 'Bootcamp', 'Build', 'Browse', and other application functions.

Get Postman here:

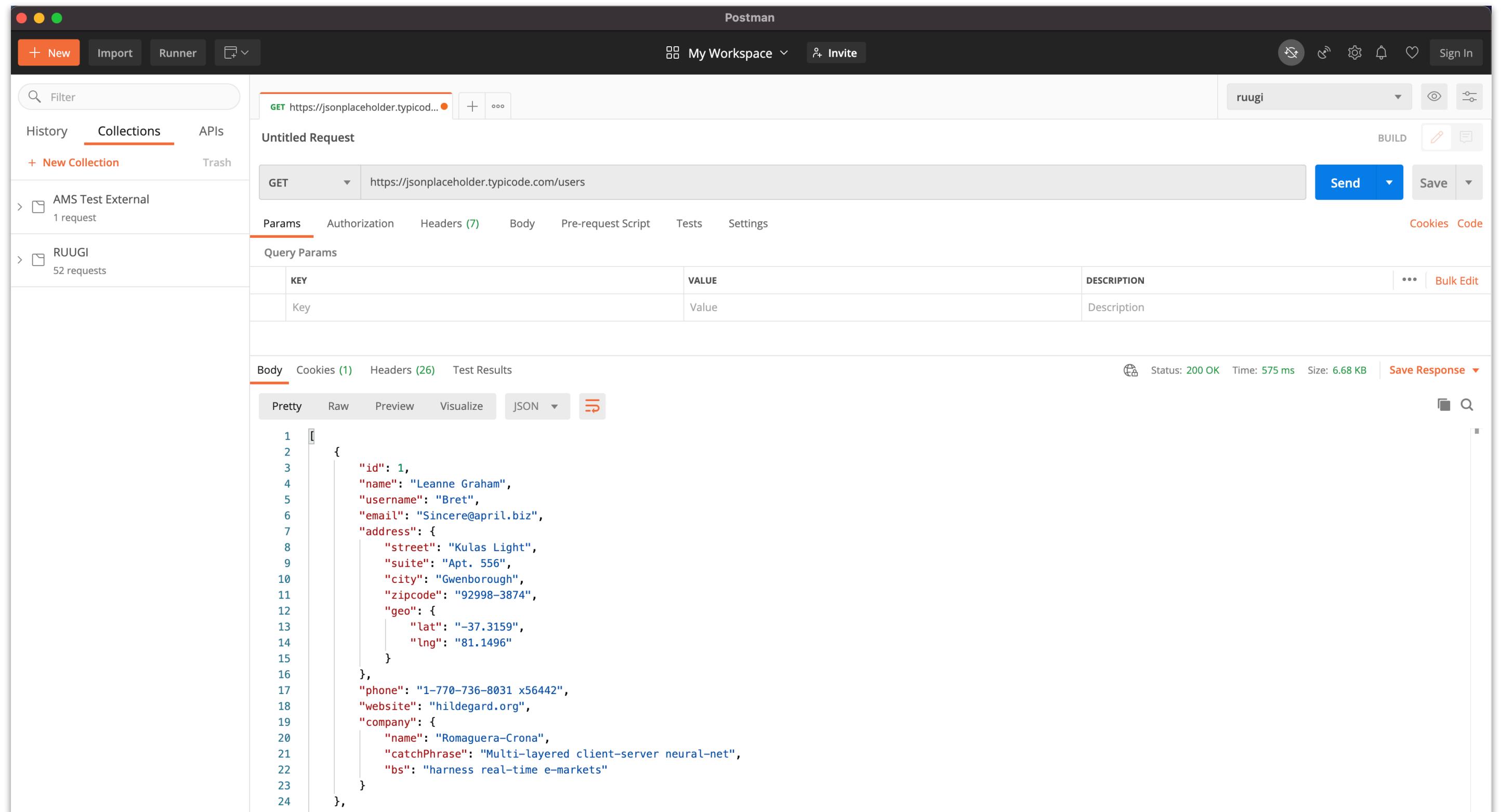
<https://www.postman.com/downloads/>

Make a GET request

Let's get some user information.

GET

<https://jsonplaceholder.typicode.com/users>



The screenshot shows the Postman application interface. In the top bar, there are buttons for '+ New', 'Import', 'Runner', and a search bar containing 'ruugi'. The title bar says 'Postman' and 'My Workspace'. Below the title bar, there are tabs for 'History', 'Collections' (which is selected), and 'APIs'. A sidebar on the left shows a collection named 'RUUGI' with 52 requests. The main area is titled 'Untitled Request' with a 'GET' method and the URL 'https://jsonplaceholder.typicode.com/users'. Under the 'Params' tab, there is a table with one row: 'Key' (empty) and 'Value' (empty). The 'Body' tab is selected, showing a JSON response with line numbers from 1 to 24. The response is:

```
1  [
2   {
3     "id": 1,
4     "name": "Leanne Graham",
5     "username": "Bret",
6     "email": "Sincere@april.biz",
7     "address": {
8       "street": "Kulas Light",
9       "suite": "Apt. 556",
10      "city": "Gwenborough",
11      "zipcode": "92998-3874",
12      "geo": {
13        "lat": "-37.3159",
14        "lng": "81.1496"
15      }
16    },
17    "phone": "1-770-736-8031 x56442",
18    "website": "hildegard.org",
19    "company": {
20      "name": "Romaguera-Crona",
21      "catchPhrase": "Multi-layered client-server neural-net",
22      "bs": "harness real-time e-markets"
23    }
24  },
```

In the bottom right corner of the main area, there is status information: 'Status: 200 OK', 'Time: 575 ms', 'Size: 6.68 KB', and a 'Save Response' button.

Breaking down the response

GET https://jsonplaceholder.typicode.com/users

The screenshot shows the Postman application interface. At the top, there's a header bar with a red circle icon, a '+' button, and an 'ooo' button. To the right is a user profile 'ruugi' with dropdown and settings icons. Below the header is a toolbar with 'BUILD' and various icons. The main area is titled 'Untitled Request'. A search bar shows 'GET https://jsonplaceholder.typicode.com/users'. On the right are 'Send' and 'Save' buttons. Below the search bar are tabs for 'Params', 'Authorization', 'Headers (7)', 'Body', 'Pre-request Script', 'Tests', and 'Settings'. The 'Params' tab is selected. Under 'Query Params', there's a table with columns 'KEY', 'VALUE', and 'DESCRIPTION'. A 'Bulk Edit' button is at the bottom right. Below this is a section titled 'Response Data Type' with tabs for 'Body', 'Cookies (1)', 'Headers (26)', and 'Test Results'. The 'Body' tab is selected. It has buttons for 'Pretty', 'Raw', 'Preview', 'Visualize', and a 'JSON' dropdown menu with a red circle around it. The 'Pretty' view shows a JSON object with numbered lines from 1 to 16. The first few lines are: 1 { 2 "id": 1, 3 "name": "Leanne Graham", 4 "username": "Bret", 5 "email": "Sincere@april.biz", 6 "address": { 7 "street": "Kulas Light", 8 "suite": "Apt. 556", 9 "city": "Gwenborough", 10 "zipcode": "92998-3874", 11 "geo": { 12 "lat": "-37.3159", 13 "lng": "81.1496" 14 } 15 } 16 }. To the right of the body preview is a status summary: a globe icon, 'Status: 200 OK', 'Time: 575 ms', 'Size: 6.68 KB', and a 'Save Response' button with a red circle around it. Below the body preview, the text 'Response Data' is circled in red.

Response Data Type

Body Cookies (1) Headers (26) Test Results

Pretty Raw Preview Visualize JSON

```
1 {  
2   "id": 1,  
3   "name": "Leanne Graham",  
4   "username": "Bret",  
5   "email": "Sincere@april.biz",  
6   "address": {  
7     "street": "Kulas Light",  
8     "suite": "Apt. 556",  
9     "city": "Gwenborough",  
10    "zipcode": "92998-3874",  
11    "geo": {  
12      "lat": "-37.3159",  
13      "lng": "81.1496"  
14    }  
15  }  
16 }
```

Response Data

Status: Semantic info about request
Time: How long for a response to be returned
Size: Size of response

What are HTTP status codes?

Status codes allow us to gather semantic information about the state of a request/response.

Assuming the server provides accurate status codes, we can use this in our code logic to decide whether or not to take a certain action, retry the request or abort the operation.

HTTP STATUS CODES

2xx Success

200 Success / OK

3xx Redirection

- 301** Permanent Redirect
- 302** Temporary Redirect
- 304** Not Modified

4xx Client Error

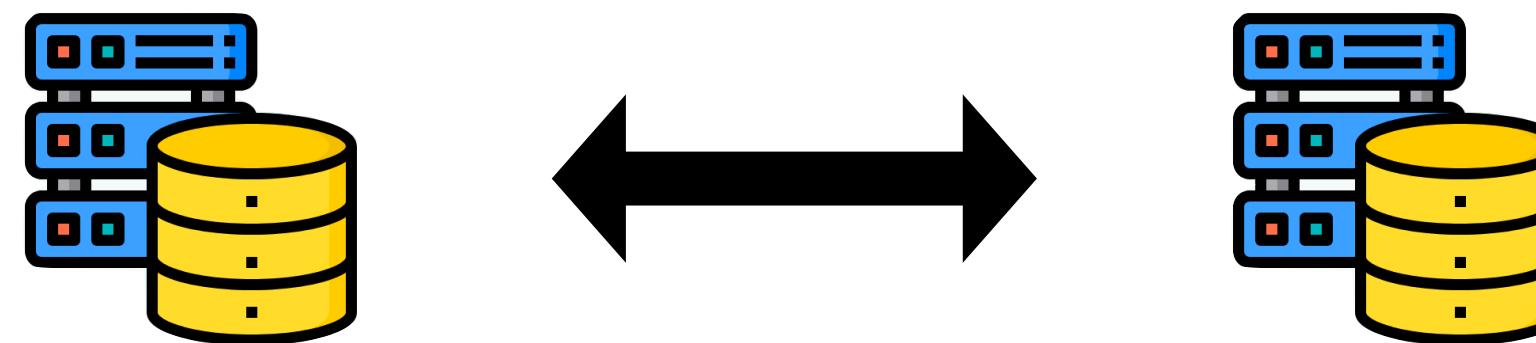
- 401** Unauthorized Error
- 403** Forbidden
- 404** Not Found
- 405** Method Not Allowed

5xx Server Error

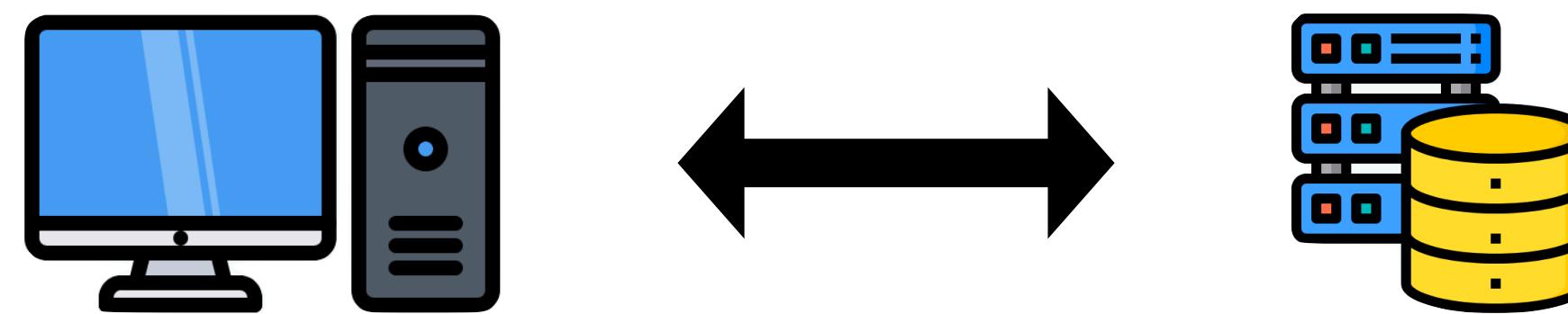
- 501** Not Implemented
- 502** Bad Gateway
- 503** Service Unavailable
- 504** Gateway Timeout

Response Data Formats

Different data types will result in different client-side behaviour.



Some APIs are intended to allow server-server communication and do not need instructions for how to *display* data. In such cases, JSON is a popular data format.

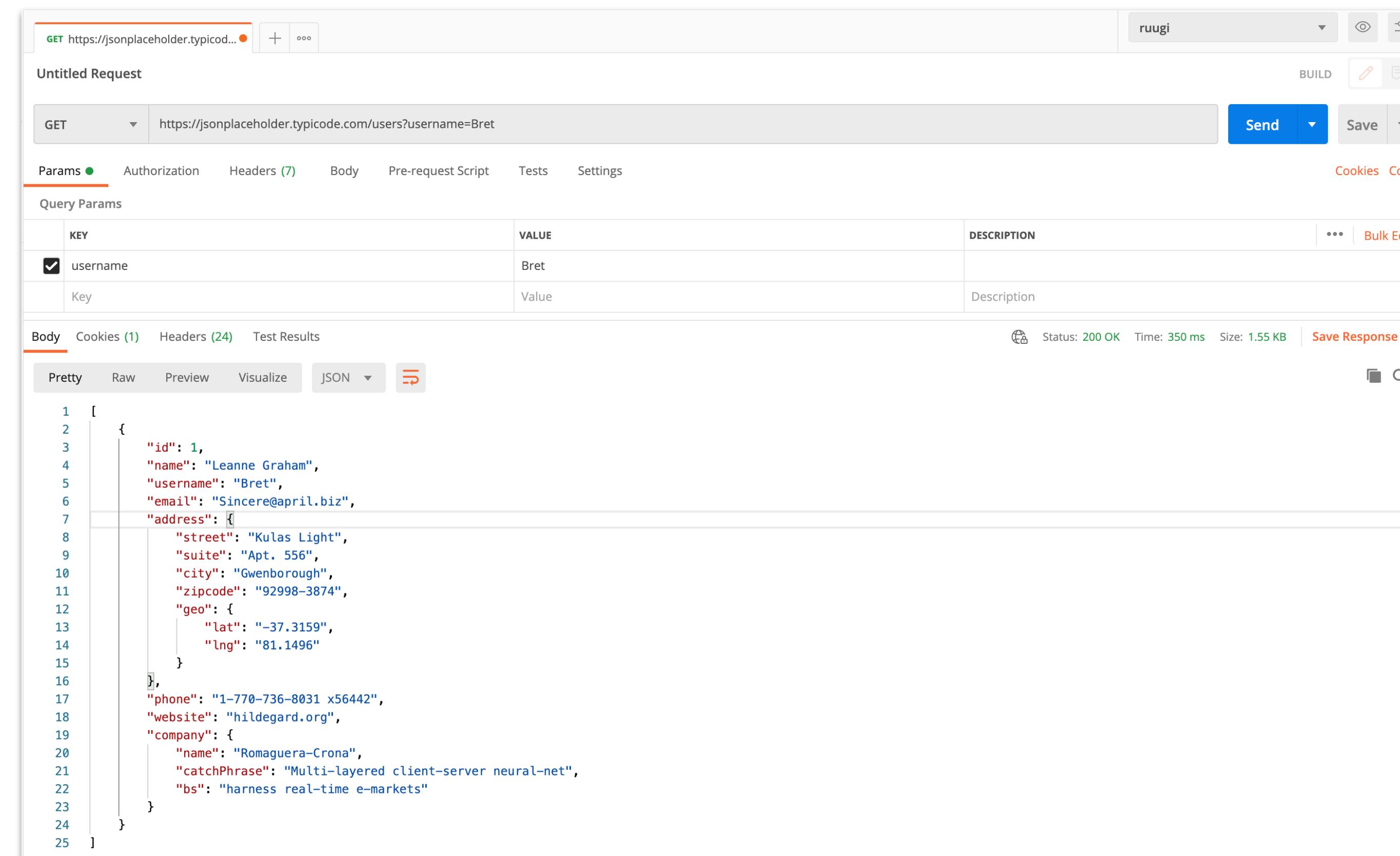


In the case of web frontend APIs, since the purpose of the browser is to display information for humans, GET requests for web pages almost always return HTML.

GET specific information

Let's get a specific user

GET <https://jsonplaceholder.typicode.com/users?username=Bret>



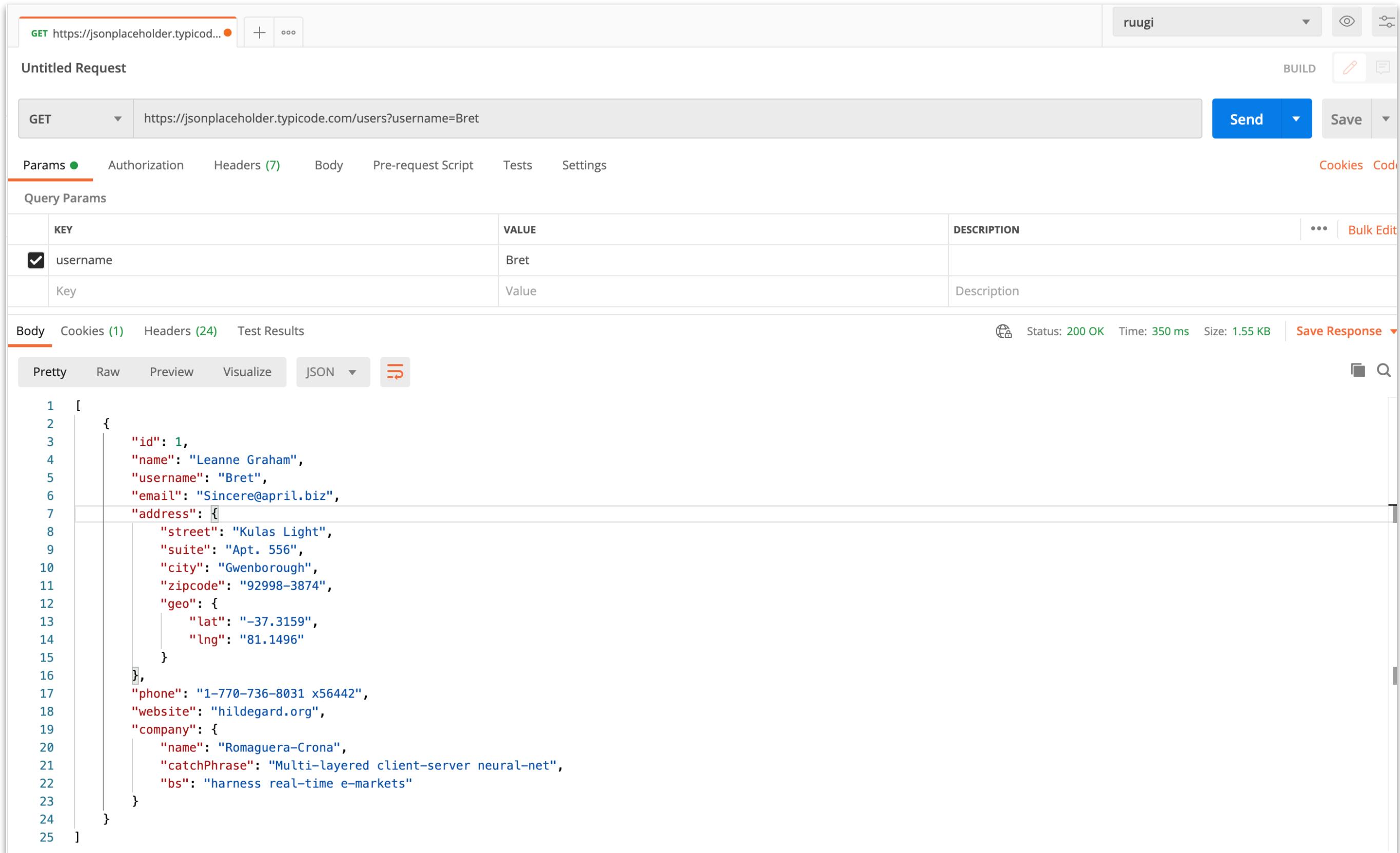
The screenshot shows the Postman application interface with the following details:

- Request URL:** GET https://jsonplaceholder.typicode.com/users?username=Bret
- Params Tab:** Shows a single query parameter `username` with value `Bret`.
- Body Tab:** Displays the JSON response in pretty-printed format.
- Response Headers:** Status: 200 OK, Time: 350 ms, Size: 1.55 KB.
- Response Body (Pretty):**

```
1 [  
2 {  
3   "id": 1,  
4   "name": "Leanne Graham",  
5   "username": "Bret",  
6   "email": "Sincere@april.biz",  
7   "address": {  
8     "street": "Kulas Light",  
9     "suite": "Apt. 556",  
10    "city": "Gwenborough",  
11    "zipcode": "92998-3874",  
12    "geo": {  
13      "lat": "-37.3159",  
14      "lng": "81.1496"  
15    },  
16    "phone": "1-770-736-8031 x56442",  
17    "website": "hildegard.org",  
18    "company": {  
19      "name": "Romaguera-Crona",  
20      "catchPhrase": "Multi-layered client-server neural-net",  
21      "bs": "harness real-time e-markets"  
22    }  
23  }  
24}  
25 ]
```

Breaking it down

GET https://jsonplaceholder.typicode.com/users?username=Bret



The screenshot shows the Postman application interface. At the top, there's a header bar with the URL "https://jsonplaceholder.typicode.com/" and a status indicator. Below the header is a toolbar with "Untitled Request", "BUILD" button, and "Send" button. The main area is divided into sections: "Params" (selected), "Authorization", "Headers (7)", "Body", "Pre-request Script", "Tests", and "Settings". Under "Params", there's a table for "Query Params" with one entry: "username" set to "Bret". Below this are tabs for "Body", "Cookies (1)", "Headers (24)", and "Test Results". The "Body" tab is selected, showing a JSON response with line numbers from 1 to 25. The response content is:

```
1 [  
2 {  
3   "id": 1,  
4   "name": "Leanne Graham",  
5   "username": "Bret",  
6   "email": "Sincere@april.biz",  
7   "address": [  
8     {"street": "Kulas Light",  
9      "suite": "Apt. 556",  
10     "city": "Gwenborough",  
11     "zipcode": "92998-3874",  
12     "geo": {  
13       "lat": "-37.3159",  
14       "lng": "81.1496"  
15     }  
16   },  
17   "phone": "1-770-736-8031 x56442",  
18   "website": "hildegard.org",  
19   "company": {  
20     "name": "Romaguera-Crona",  
21     "catchPhrase": "Multi-layered client-server neural-net",  
22     "bs": "harness real-time e-markets"  
23   }  
24 }  
25 ]
```

At the bottom right of the body panel, there are status details: "Status: 200 OK", "Time: 350 ms", "Size: 1.55 KB", and a "Save Response" button. Below the body panel are buttons for "Pretty", "Raw", "Preview", "Visualize", and "JSON".

Again, we have the same response properties as before but we have a new set of data.

Breaking it down

GET https://jsonplaceholder.typicode.com/users?username=Bret

The screenshot shows the Postman interface with an 'Untitled Request' titled 'GET https://jsonplaceholder.typicode.com/users?username=Bret'. A red oval highlights the URL bar. Below it, the 'Params' tab is selected, showing a table with one row: 'username' (KEY) with a checked checkbox and 'Bret' (VALUE). Other tabs include 'Authorization', 'Headers (7)', 'Body', 'Pre-request Script', 'Tests', and 'Settings'.

KEY	VALUE
username	Bret
Key	Value

Try: Can you set multiple query params?

Query Parameters

They allow us to specify certain conditions we want in the final set of data.

GET /users?username=Bret

GET users **where** username equals Bret

Note: This feature does not exist for all APIs, it depends on how the developer has designed the server to interact. More on this in Day 4.

Let's make a web scraper!

Activity #2

Getting started

We will be using a few external libraries for this activity that you should have installed.



```
pip install requests
```

The BeautifulSoup logo consists of the word "Beautiful" in a bold, black, sans-serif font followed by "soup" in a stylized, thin black font where the 'o' has a long vertical stroke curving down and to the right.

```
pip install beautifulsoup4
```



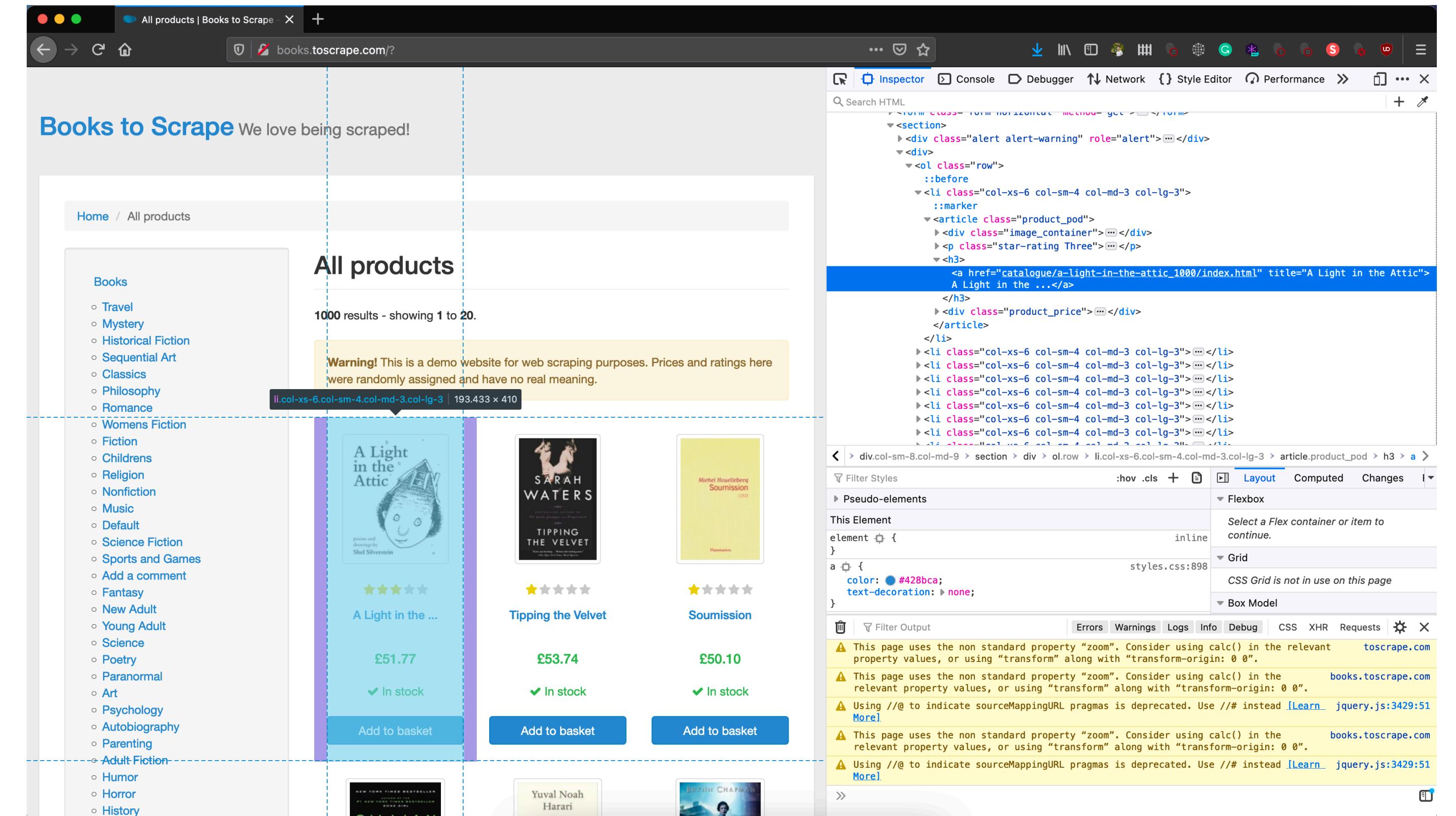
```
pip install jupyter
```

What is the Inspector Tool?

Inspecting a web page allows us to view the underlying HTML.

Think of the HTML code as the frame of the web page. It tells the browser what to render in what place; it is the skeleton of the page.

This is useful when we want to see how information is organised on a page.



It's a wrap!

What have we learnt today?

- A high-level understanding of how the web works
- How information is passed around over the network
- What are HTTP Requests
- Practical use cases + light web scraping!