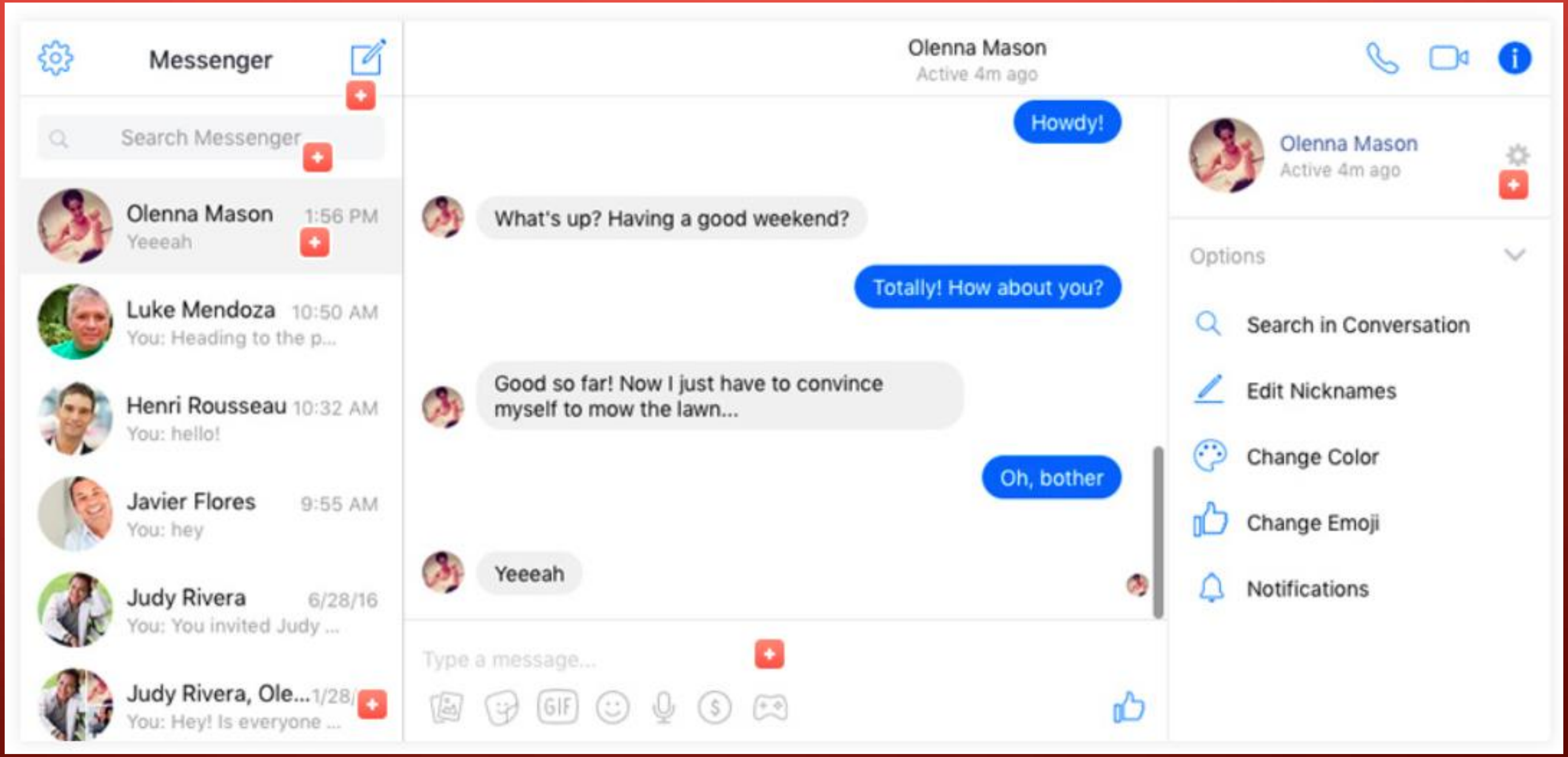
AJAX

WHERE WE STAND SO FAR

- In a Flask application:
  - `render_template('index.html')`



Messenger

Search Messenger

Olenna Mason 1:56 PM
Yeeeah

Luke Mendoza 10:50 AM
You: Heading to the p...

Henri Rousseau 10:32 AM
You: hello!

Javier Flores 9:55 AM
You: hey

Judy Rivera 6/28/16
You: You invited Judy ...

Judy Rivera, Ole... 1/28/
You: Hey! Is everyone ...

Olenna Mason
Active 4m ago

Howdy!

What's up? Having a good weekend?

Totally! How about you?

Good so far! Now I just have to convince myself to mow the lawn...

Oh, bother

Yeeeah

Type a message...

Olenna Mason
Active 4m ago

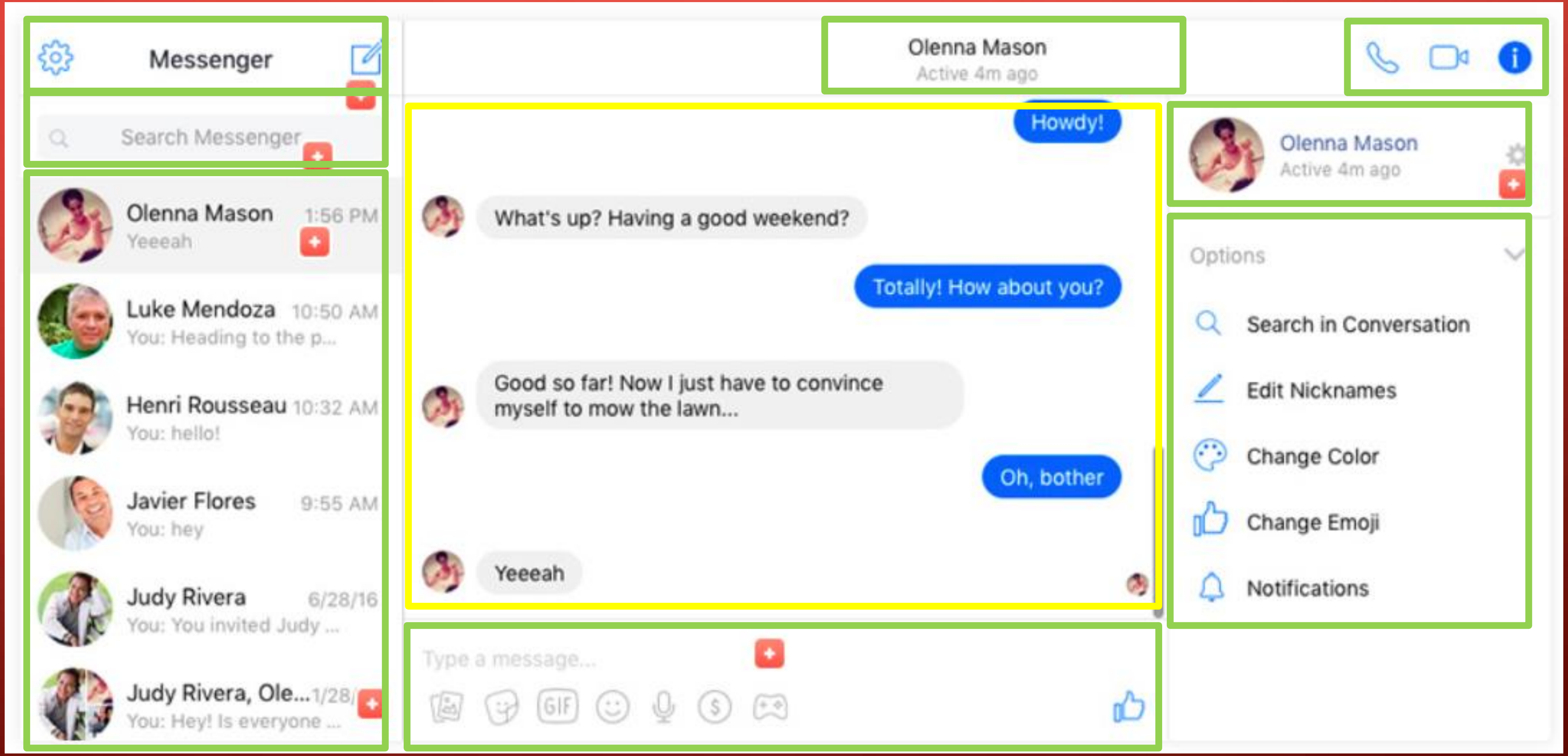
Options

WHERE WE STAND SO FAR

- `render_template('index.html')`
  - The whole page is reloaded when a new message is posted
  - This is not very dynamic: most of the page contents do not change between messages
  - If takes long time to load a whole page, customers will probably give up using this site

WHERE WE STAND SO FAR

- Yellow Rectangle: content changes every time a new message is displayed
- Green Rectangle: content does not change over time



WHAT ARE WE GOING TO FETCH FROM THE SERVER?

- Grabbing an HTML document via JavaScript doesn't seem much better than having the browser do it...
  - What other document format could we get from the server?

Extensible Markup Language (XML)

- Data representation format
  - It uses tags in a very similar manner to HTML
  - It can similarly be traversed using the DOM!

XML EXAMPLE

```xml
<person>
    <name>John Smith</name>
    <age>25</age>
    <address>
        <streetAddress>21 2nd Street</streetAddress>
        <city>New York</city>
        <state>NY</state>
        <postalCode>10021-3100</postalCode>
    </address>
    <phoneNumbers>
        <phoneNumber>
            <type>mobile</type>
            <number>123 456-7890</number>
        </phoneNumber>
    </phoneNumbers>
    <children></children>
    <spouse></spouse>
</person>
```

THOUGHTS ABOUT XML

- Seems a bit unwieldy
  - Very verbose
    - Both to represent data
    - And to parse it with the DOM
- It would be nice to just send JavaScript Objects back and forth from client to server

JAVASCRIPT OBJECTS

- Basically, a group of key/value pairs
- JavaScript Object Notation
  - Also known as JSON
  - Uses human-readable text to transmit objects as key value pairs

JSON EXAMPLE

```
{
    "name": "John Smith",
    "age": 25,
    "address": {
        "streetAddress": "21 2nd Street",
        "city": "New York",
        "state": "NY",
        "postalCode": "10021-3100"
    },
    "phoneNumbers": [
        {
            "type": "mobile",
            "number": "123 456-7890"
        },
        {
            "type": "office",
            "number": "646 555-4567"
        },
    ]
}
```

JSON BASIC DATA TYPES

- Number
  - Signed
  - Can have fractional component
- String
  - Double-quoted
- Boolean
  - `true` or `false`
- Array
  - Enclosed in square brackets
- Objects
  - key: value pairs in curly braces
- null

```
{
    "name": "John Smith",
    "age": 25,
    "address": {
        "streetAddress": "21 2nd Street",
        "city": "New York",
        "state": "NY",
        "postalCode": "10021-3100"
    },
    "phoneNumbers": [
        {
            "type": "mobile",
            "number": "123 456-7890"
        },
        {
            "type": "office",
            "number": "646 555-4567"
        },
    ]
}
```

AJAX

- AJAX: Asynchronous JavaScript and XML (and JSON)
- We use JavaScript to create dynamic client-side applications
  - Edit the DOM
  - Causing the page to be re-rendered
- But how can we use it to fetch new data from the server?
  - Through the use of the `XMLHttpRequest` object
  - The backbone of AJAX

XMLHttpRequest MAIN FUNCTIONS AND ATTRIBUTES

- `XMLHttpRequest.open()`
- `XMLHttpRequest.send()`
- `XMLHttpRequest.readyState`
- `XMLHttpRequest.status`
- `XMLHttpRequest.response`
- `XMLHttpRequest.onreadystatechange`

XMLHttpRequest.open()

- `open(method, url, async)`
  - `method` is an HTTP method (GET, POST, DELETE…)
  - `url` is the location of the server
  - `async` is a boolean to determine if the transfer is to be done asynchronously or not
    - Defaults to `true`

`XMLHttpRequest.send()`

- `send(data)`
  - Issues the specified HTTP request to the server
  - `data` is the (optional) information to be sent to the server
    - Can be formatted in various ways, with different encodings
      - E.g., `var=value` pair query string
    - If data is sent to the server, the **content type** must be set in the request header
      - E.g., for a query string:
      - `req.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded’);`
        - Where `req` is an `XMLHttpRequest` object

GET HTTP REQUEST EXAMPLE

- GET HTTP request:

```
var xmlHttp = new XMLHttpRequest();
xmlHttp.open("GET", “www.myapi.com”, false); // false for synchronous request
xmlHttp.send(null);
return JSON.parse(xmlHttp.responseText);
```

POST HTTP REQUEST EXAMPLE

- A POST HTTP Request

```
var xmlHttp = new XMLHttpRequest();
xmlHttp.open("POST", theUrl, false); // false for synchronous request
xmlHttp.setRequestHeader('Content-type', 'application/json');
xmlHttp.send(JSON.stringify(newBlog));  // newBlog is a JSON object
```

PUT HTTP REQUEST EXAMPLE

- A PUT HTTP Request

```
var xmlHttp = new XMLHttpRequest();
xmlHttp.open("PUT", theUrl, false); // false for synchronous request
xmlHttp.setRequestHeader('Content-type', 'application/json');
xmlHttp.send(JSON.stringify(updatedBlog));  // updatedBlog is a JSON object
```

`XMLHttpRequest.readyState`

- Attribute that stores the current state of the `XMLHttpRequest` object
  - `readyState changes throughout the execution:`
    - `0 → XMLHttpRequest.UNSENT`
    - `1 → XMLHttpRequest.OPENED`
    - `2 → XMLHttpRequest.HEADERS_RECEIVED`
    - `3 → XMLHttpRequest.LOADING`
    - `4 → XMLHttpRequest.DONE`

`XMLHttpRequest.status`

- Stores the HTTP status code of the response to the request
  - `200`
  - `404`
  - `500`
  - etc.
- Before the request completes, will have a value of `0`

EXAMPLE

```
function logResponse(xhr) {
    console.log(`readyState: ${xhr.readyState}`);
    if (xhr.readyState === XMLHttpRequest.DONE) {
        console.log(`status: ${xhr.status}`);
        if (xhr.status === 200) {
            console.log("Value sent to server successfully!");
        } else {
            console.log("There was a problem with the request.");
        }
    }
}
```

XMLHttpRequest.response

- So, the readystatus is DONE and the request status is 200… We've got a response! Let's retrieve it from the XMLHttpRequest object!

- `XMLHttpRequest.response` holds the data returned from the server
  - Type is determined via `XMLHttpRequest.responseType`
  - Response data can also be accessed via:
    - `XMLHttpRequest.responseText`
    - `XMLHttpRequest.responseURL`
    - `XMLHttpRequest.responseXML`

EXAMPLE

```
function logResponse(xhr) {
	console.log(`readyState: ${xhr.readyState}`);
	if (xhr.readyState === XMLHttpRequest.DONE) {
		console.log(`status: ${xhr.status}`);
		if (xhr.status === 200) {
			console.log(“server response“ + xhr.response);
		} else {
			console.log("There was a problem with the request.");
		}
	}
}
```

`XMLHttpRequest.onreadystatechange`

- Attribute to which we can assign an event handler
  - This will associate the function with the occurrence of the `readystatechange` event
- This event fires in several places throughout the the execution (each time the state changes)
- We can check the `XMLHttpRequest.readyState` to see what, if anything, we will do to handle the event
- Note that this attribute should be set before starting the request

EXAMPLE OF ONREADYSTATECHANGE

```
xhr.onreadystatechange = () => logResponse(xhr);
xhr.addEventListener("readystatechange", logResponse(xhr))

function logResponse(xhr) {
    console.log(`readyState: ${xhr.readyState}`);
    if (xhr.readyState === XMLHttpRequest.DONE) {
        console.log(`status: ${xhr.status}`);
        if (xhr.status === 200) {
            // debugger;
            console.log("Value sent to server!");
        } else {
            console.log("There was a problem with the request.");
        }
    }
}
```

PARSING TEXT RESPONSE TO JSON OBJECTS

- Remember that `XMLHttpRequest.response` maybe in text format
- Recall that a JSON object is just formatted text
  - To obtain JSON object, just parse the response
    - xhr.response.json()
    - JSON.parse(xhr.response)

```
function logResponse(xhr) {
        console.log(`readyState: ${xhr.readyState}`);
        if (xhr.readyState === XMLHttpRequest.DONE) {
            console.log(`status: ${xhr.status}`);
            if (xhr.status === 200) {
                console.log(“server response“ + xhr.response.json());
            } else {
                console.log("There was a problem with the request.");
            }
        }
}
```

THE FETCH API

- Would you like to have your JavaScript application wait as long as needed for a request response to be sent from the server before making your page responsive again?
  - Your page could be frozen for a while, making your customers to leave it
- The Fetch API allows you to asynchronously request for a resource.
- Use the fetch() method to return a promise that resolves into a Response object.
- To get the actual data, you call one of the methods of the Response object e.g., text() or json().
- These methods resolve into the actual data.

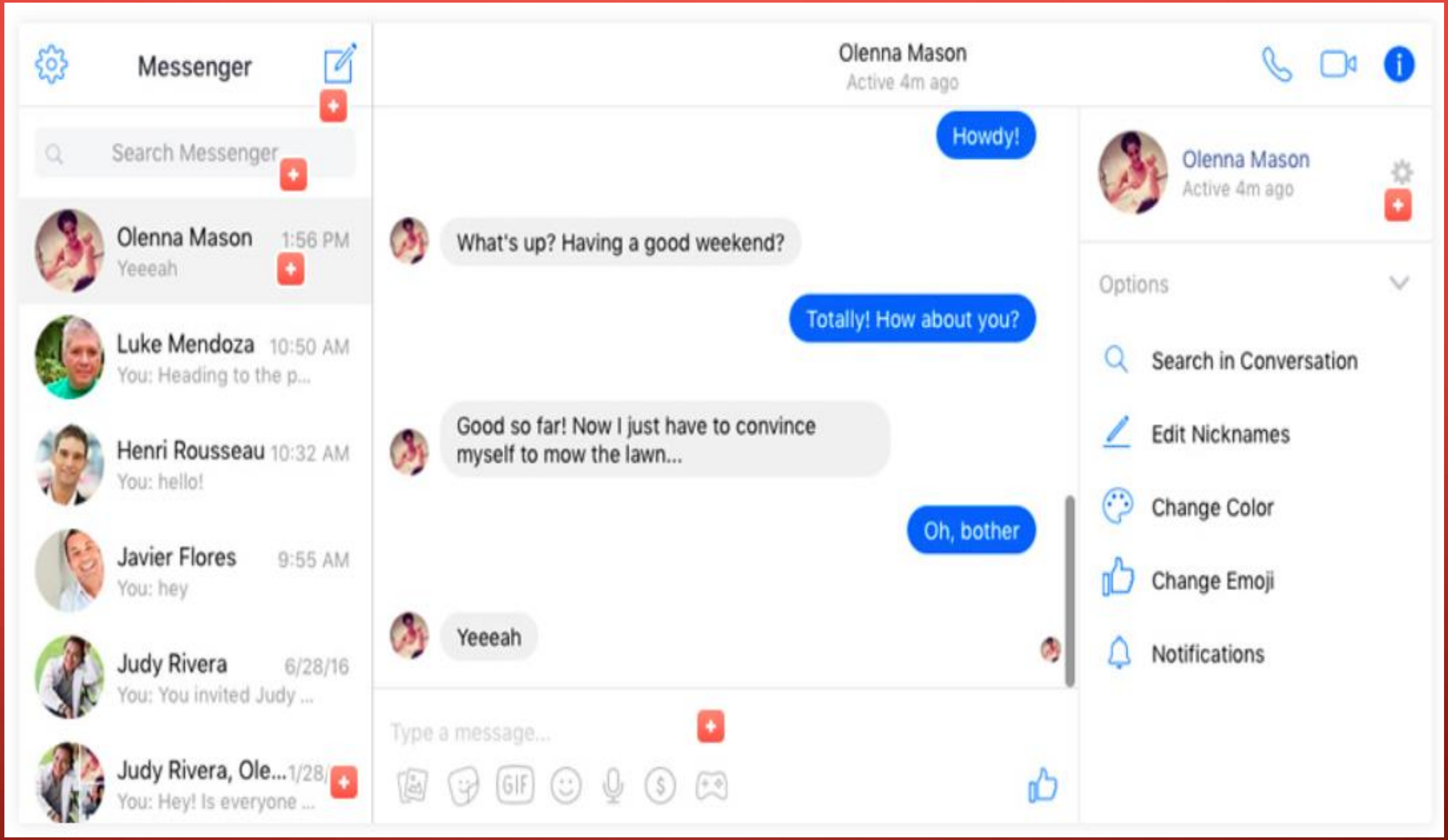
EXAMPLE OF PROMISES AND FETCH

```
fetch("http://example.com/movies.json")  🡐 a promise
   .then(function(response) {  🡐 when the promise is paid up
       return response.json(); 🡐 a new promise
   })
   .then(function(myJSON) {    🡐 when the promise is paid up
       console.log(myJSON);
});
```

EXAMPLE OF PROMISES AND FETCH

```
return fetch(url, {
   method: "POST",
   credentials: "same-origin",
   headers: {
      "Content-Type": "application/x-www-form-urlencoded“
   },
   body: "key1=val1&key2=val2“
})
   .then(response => response.json())
   .catch(error => console.error("Fetch Error =\n", error));
```

POOLING DATA



Messenger

Search Messenger

Olenna Mason 1:56 PM
Yeeeah

Luke Mendoza 10:50 AM
You: Heading to the p...

Henri Rousseau 10:32 AM
You: hello!

Javier Flores 9:55 AM
You: hey

Judy Rivera 6/28/16
You: You invited Judy ...

Judy Rivera, Ole... 1/28/
You: Hey! Is everyone ...

Olenna Mason
Active 4m ago

Howdy!

What's up? Having a good weekend?

Totally! How about you?

Good so far! Now I just have to convince myself to mow the lawn...

Oh, bother

Yeeeah

Type a message...

Olenna Mason
Active 4m ago

Options

Search in Conversation

Edit Nicknames

Change Color

Change Emoji

Notifications

POOLING

- So we can have the page update itself
  - In response to user actions
  - New information is available on the server
  - Periodically request updates from the server (pooling)

- How to accomplish pooling ➔ JavaScript Timers:
  - `window.setTimeout()`
  - `window.setInterval()`
  - `window.clearTimeout()`
  - `window.clearInterval()`

EXAMPLE OF POOLING

```
let timeoutID;
let timeout = 15000;

function setup() {
    document.getElementById("theButton").addEventListener("click", makePost);
    timeoutID = window.setTimeout(fetchNewDataFunction, timeout);
}

timeoutID = window.setTimeout(fetchNewDataFunction, timeout);
```