Client-Server Communication

Activity

1. Write a web page that uses AJAX to load content after the page loads

2. Implement the same functionality using WebSockets (You can use socket.io)

You may choose any functionality you'd like as long as it's requested and rendered after the page loads

Reviews - sockittoem.net

- We'll add a form to let users review items

- Live demo using only concepts we've already seen
  - Analyze where this falls short

Reviews - sockittoem.net

Live demo

HTTP Requests

- Every time a client sends an HTTP request to the server, the page reloads
- This includes submitting forms
- But what if we don't want to reload the page after every request?

JSON - Reminder

```
JSON.stringify(jsData)
JSON.parse(jsonString)
```

JS

```
import json

json.dumps(python_data)
json.loads(json_string)
```

- json.loads to convert from a JSON string to python type
- json.dumps to convert Python types to JSON strings
- JSON.stringify to convert JavaScript types to a JSON string
- JSON.parse to convert a JSON string to JavaScript type

Libraries exist for all major languages

- More involved in strongly typed languages

AJAX

Asynchronous JavaScript [And XML]

A way to make HTTP requests from JavaScript *after* the page loads

Can make HTTP GET and POST requests

AJAX - HTTP GET Request

```
var request = new XMLHttpRequest();
request.onreadystatechange = function(){
    if (this.readyState === 4 && this.status === 200){
        console.log(this.response);
        // Do something with the response
    }
};
request.open("GET", "/path");
request.send();
```

There are many different ways to setup an AJAX call

We setup the call in a function that takes

- The path where we want to send the requests (matches the paths in the annotations of the bottle server)
- A callback function. This function will be called when the server responds to our requests with the response as an argument

JS

AJAX - HTTP POST Request

```
var request = new XMLHttpRequest();
request.onreadystatechange = function(){
    if (this.readyState === 4 && this.status === 200){
        console.log(this.response);
        // Do something with the response
    }
};
request.open("POST", "/path");
let data = {'username': "Jesse", 'message': "Hello"}
request.send(JSON.stringify(data));
```

To make a POST request most of the code is the same

The major difference is that we have a third parameter named data which must be a string containing the data that will be in the body of the request

JS

Sock App Continued

To do this we will make an AJAX get request after the page loads to get and display the items and reviews

- Faster page loads
- Use JavaScript to render the data as HTML

Then we will make an AJAX POST request each time the user adds a review

- Allows for Single Page Apps (SPAs)

sockittoem API

- Building the review API

- GET /allItems
  - Returns a JSON string containing all the items and reviews
- POST /addReview
  - Send ItemID / Rating / Review to the server

- **Live Demo**

Making it Live

- What if someone adds a review after you load the page?

- Polling
  - Keep sending AJAX requests at fixed intervals to refresh the data

- Long-Polling
  - Same as polling but the server waits for data to be available before responding to the request

WebSockets

- Full two-way communication between client and server
- Enables server pushes
  - Server can send data to a client without responding to a request

socket.io

- A library adding functionality to WebSockets
- Maintains the connection and reconnecting
- Based on message types

- Sockets "emit" data to send it to the other side of the connection
- Sockets respond to messages using "on" syntax

Flask - Web Sockets

- Use flask_socketio library
- Install with pip, or a requirements.txt file

requirements.txt

```
flask
flask-socketio
eventlet
```

```python
from flask_socketio import SocketIO

app = Flask(__name__)
socket_server = SocketIO(app)

socket_server.run(app, port=8080)
```

Flask - Web Sockets

- Send/receive messages with types over web sockets
  - Similar to Scala actor messages
- Annotate methods to run them when certain message types are received

```
app = Flask(__name__)
socket_server = SocketIO(app)

@socket_server.on('clickGold')
def click_gold():
    pass
    # received a "clickGold" message over the socket


@socket_server.on('buy')
def buy_equipment(equipmentID):
    pass
    # received a "buy" message over the socket with a parameter equipmentID

socket_server.run(app, port=8080)
```

Flask - Web Sockets

- Send message using emit
- Specify the message type and any parameters needed

```
app = Flask(__name__)
socket_server = SocketIO(app)

@socket_server.on('clickGold')
def click_gold():
    # received a "clickGold" message over the socket
    emit('goldClicked', 'I see you clicking that gold')


socket_server.run(app, port=8080)
```

Flask - Web Sockets

- To send a message to a client later
  - Remember their socket id (sid)
- Access request.sid when a user connects
  - Associate this sid with their username (or other identifier)
- When you want to send this user a message
  - Get their sid and send a message to their "room"

```
usernameToSid = {}

@socket_server.on('register')
def got_message(username):
    usernameToSid[username] = request.sid


# Later

user_socket = usernameToSid.get(username)
socket_server.emit('message', data, room=user_socket)
```

Browser - Web Sockets

- Connect to the web server via sockets
- Import the library in the head of your HTML

```
<script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/socket.io/2.2.0/socket.io.js"></script>
```

- Connect, send, receive in JavaScript
  - Use callback methods when a message is received
  - Use emit to send a message with a type

```
var socket = io.connect({transports: ['websocket']});

socket.on('connect', function (event) {
    // connected to server
});

socket.on('message', function (event) {
    // received a message from the server
    console.log(event);
    socket.emit("ack", "thanks for the message!");
});

socket.emit("clickGold");
socket.emit("buy", "shovel");
```

Activity

1. Write a web page that uses AJAX to load content after the page loads

2. Implement the same functionality using WebSockets (You can use socket.io)

You may choose any functionality you'd like as long as it's requested and rendered after the page loads