CSE 503
Introduction to Computer Science for Non-Majors

Dr. Eric Mikida
epmikida@buffalo.edu
208 Capen Hall

Day 26
AJAX (Part 3)

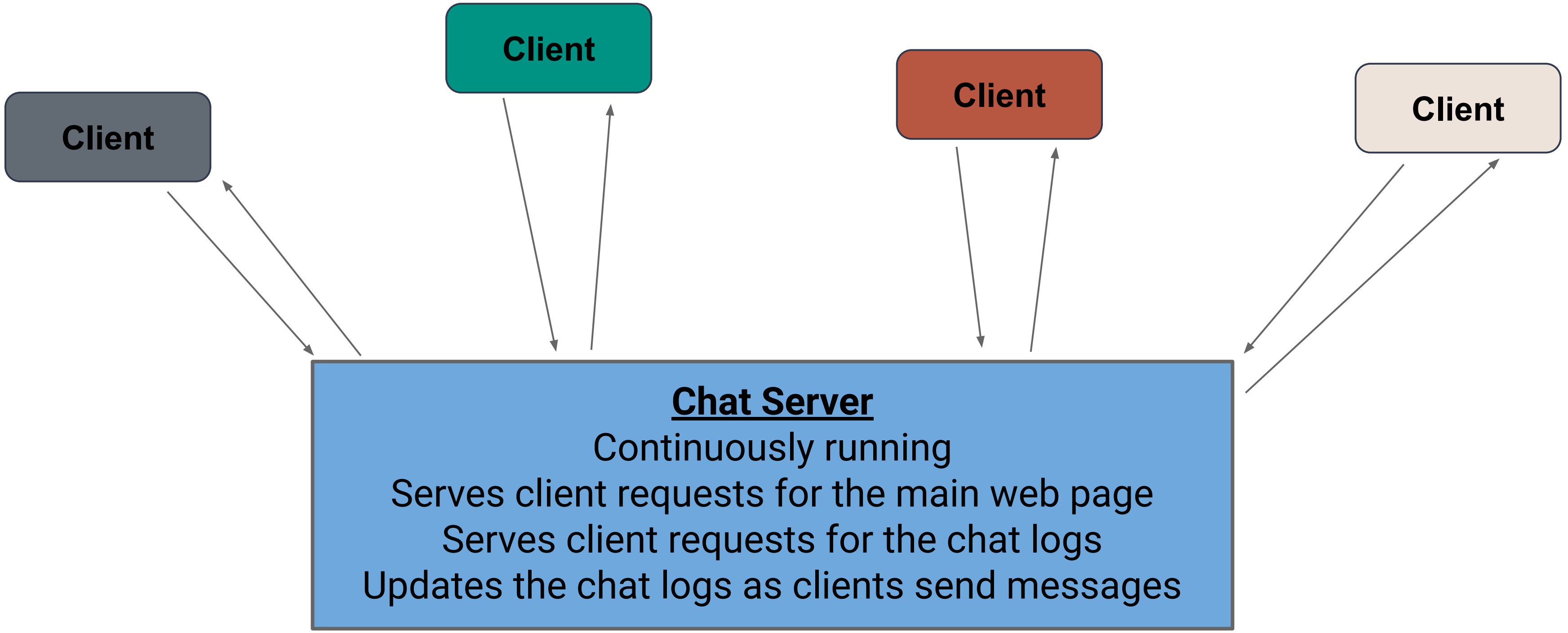
Announcements

- Autolab for Lab 4 should be up by tonight

Recap

- Last time we started making our own chat server…

End Goal



Chat Server

Continuously running

Serves client requests for the main web page

Serves client requests for the chat logs

Updates the chat logs as clients send messages

Chat Server Design

What do we need:

1. A front end web page (with interactive components)
2. Front end JavaScript allowing users to interact with the page
3. Web server code to run the server and handle requests
4. A place to store messages that persists even when server stops
5. A way for the front end and back end to communicate ***even after the page is initially loaded***

Note this is just one possible design!

Chat Server Design

What do we need:

- ✓ A front end web page (with interactive components)
- ✓ Front end JavaScript allowing users to interact with the page
- ✓ Web server code to run the server and handle requests
- ✓ A place to store messages that persists even when server stops

5. A way for the front end and back end to communicate ***even after the page is initially loaded***

Communication Between Client and Server

Now we can set up our communication…

How can our ***JavaScript*** *client and* ***Python*** *web server communicate?*

Communication Between Client and Server

Now we can set up our communication…

*How can our **JavaScript** client and **Python** web server communicate?*

JSON!

JSON in JavaScript and Python

In Python:

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

In JavaScript:

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

JSON in JavaScript and Python

In Python:

```
import json                    ← Loads the JSON library
json.loads(json_string)
json.dumps(python_data)
```

In JavaScript:

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

JSON in JavaScript and Python

In Python:

```
import json                   ← Loads the JSON library
json.loads(json_string)       ← Takes a JSON string and returns Python data
json.dumps(python_data)
```

In JavaScript:

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

JSON in JavaScript and Python

In Python:

```
import json                   ← Loads the JSON library
json.loads(json_string)       ← Takes a JSON string and returns Python data
json.dumps(python_data)       ← Takes Python data and returns a JSON string
```

In JavaScript:

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

JSON in JavaScript and Python

In Python:

```
import json                  ← Loads the JSON library
json.loads(json_string)      ← Takes a JSON string and returns Python data
json.dumps(python_data)      ← Takes Python data and returns a JSON string
```

In JavaScript:

```
JSON.parse(jsonString)       ← Takes a JSON string and returns JavaScript data
JSON.stringify(jsData)
```

JSON in JavaScript and Python

In Python:

```
import json                  ← Loads the JSON library
json.loads(json_string)      ← Takes a JSON string and returns Python data
json.dumps(python_data)      ← Takes Python data and returns a JSON string
```

In JavaScript:

```
JSON.parse(jsonString)       ← Takes a JSON string and returns JavaScript data
JSON.stringify(jsData)       ← Takes JavaScript data and returns a JSON string
```

Server-Side Communication

Let's start by setting up the communication coming from the server:

1. **The server will send the chat to the client**
2. **The server will accept messages from the client (and send chat)**

Importing the Necessary Pieces

In `main.py`:

- Import the **`json`** library and the code we wrote in **`chat.py`**

```
import bottle
import json

import chat
```

Adding Some New Routes

In `main.py`:

- Add a route to handle requests for the chat logs
- Respond with the chat, converted to a JSON string by **`json.dumps()`**

```
@bottle.route('/chat')
def get_chat():
  return json.dumps(chat.get_chat())
```

Adding Some New Routes

In `main.py`:

- Add a route to handle requests for the chat logs
- Respond with the chat, converted to a JSON string by **`json.dumps()`**

```
@bottle.route('/chat')
def get_chat():
  return json.dumps(chat.get_chat())
```

We wrote this function earlier…

Adding Some New Routes

In `main.py`:

- Add a **`bottle.post`** annotation for when a client sends a message
  - Decode the message (turn it into the JSON string and convert to Python)
  - Call our **`add_message`** function to add the message to the chat logs
  - Respond to the client with the full chat

```
@bottle.post('/send')
def do_chat():
  content = bottle.request.body.read().decode()
  content = json.loads(content)
  chat.add_message(content['message'])
  return json.dumps(chat.get_chat())
```

Adding Some New Routes

In `main.py`:

- Add a **`bottle.post`** annotation for when a client sends a message
  - Decode the message (turn it into the JSON string and convert to Python)
  - Call our **`add_message`** function to add the message to the chat logs
  - Respond to the client with the full chat

```
@bottle.post('/send')
def do_chat():
  content = bottle.request.body.read().decode()
  content = json.loads(content)
  chat.add_message(content['message'])
  return json.dumps(chat.get_chat())
```

The **`bottle.post`** annotation lets us handle a POST request (as compared to GET)

Adding Some New Routes

In `main.py`:

- Add a **`bottle.post`** annotation for when a client sends a message
  - Decode the message (turn it into the JSON string and convert to Python)
  - Call our **`add_message`** function to add the message to the chat logs
  - Respond to the client with the full chat

`bottle.request.body` contains the information sent in the request

```
@bottle.post('/send')
def do_chat():
  content = bottle.request.body.read().decode()
  content = json.loads(content)
  chat.add_message(content['message'])
  return json.dumps(chat.get_chat())
```

Adding Some New Routes

In `main.py`:

- Add a **`bottle.post`** annotation for when a client sends a message
  - Decode the message (turn it into the JSON string and convert to Python)
  - Call our **`add_message`** function to add the message to the chat logs
  - Respond to the client with the full chat

```
@bottle.post('/send')
def do_chat():
  content = bottle.request.body.read().decode()
  content = json.loads(content)
  chat.add_message(content['message'])
  return json.dumps(chat.get_chat())
```

These are the functions we wrote earlier

JavaScript and AJAX

Now we need our JavaScript code to communicate with our Python code

We'll do this with **AJAX** (**A**synchronous **Ja**vaScript and **X**ML)

- Allows us to make requests ***after*** the page has been loaded
- Can make HTTP GET requests (to get content from a server)
- Can make HTTP POST requests (to send content to a server)

AJAX GET Request

```
function ajaxGetRequest(path, callback) {
    let request = new XMLHttpRequest();
    request.onreadystatechange = function() {
        if (this.readyState === 4 && this.status === 200) {
            callback(this.response);
        }
    };
    request.open("GET", path);
    request.send();
}
```

AJAX GET Request

```
function ajaxGetRequest(path, callback) {
```

Don't worry too much about the details of this function… feel free to use it as is.

The main thing to know is that it takes a path and a callback as input, and makes a GET request to that path

```
    request.open("GET", path);
    request.send();
}
```

AJAX POST Request

```
function ajaxPostRequest(path, data, callback) {
    let request = new XMLHttpRequest();
    request.onreadystatechange = function() {
        if (this.readyState === 4 && this.status === 200) {
            callback(this.response);
        }
    };
    request.open("POST", path);
    request.send(data);
}
```

AJAX POST Request

```
function ajaxPostRequest(path, data, callback) {
```

**Don't worry too much about the details of this function…
feel free to use it as is.**

**It works the same as the previous, but also requires data
as input, and makes a POST request to the path**

```
    request.open("POST", path);
    request.send(data);
}
```

Using Our New Functions

```
function loadChat() {
    ajaxGetRequest("/chat", displayChat);
}

function displayChat(response) {
    let chat = "";
    for(let data of JSON.parse(response)){
        chat = chat + data.message + "</br>";
    }
    document.getElementById("chat").innerHTML = chat;
}
```

Using Our New Functions

```
function loadChat() {
    ajaxGetRequest("/chat", displayChat);
}

function displayChat(response) {
    let chat = "";
    for(let data of JSON.parse(response)){
        chat = chat + data.message + "</br>";
    }
    document.getElementById("chat").innerHTML = chat;
}
```

To load the chat, we make a GET request to **`"/chat"`**, which will call **`displayChat`** with the response

Using Our New Functions

```
function loadChat() {
    ajaxGetRequest("/chat", displayChat);
}

function displayChat(response) {
    let chat = "";
    for(let data of JSON.parse(response)){
        chat = chat + data.message + "</br>";
    }
    document.getElementById("chat").innerHTML = chat;
}
```

To display the chat, we simply iterate all over all of the messages and add them to a string (remember `</br>` is a newline in HTML)

Using Our New Functions

```
function loadChat() {
    ajaxGetRequest("/chat", displayChat);
}

function displayChat(response) {
    let chat = "";
    for(let data of JSON.parse(response)){
        chat = chat + data.message + "</br>";
    }
    document.getElementById("chat").innerHTML = chat;
}
```

Finally, set the content of our `"chat"` div in the HTML file

Using Our New Functions

```
function sendMessage(){
    let messageElement = document.getElementById("message");

    let message = messageElement.value;
    messageElement.value = "";
    let toSend = JSON.stringify({"message": message});

    ajaxPostRequest("/send", toSend, displayChat);
}
```

Using Our New Functions

```
function sendMessage(){
    let messageElement = document.getElementById("message");

    let message = messageElement.value;
    messageElement.value = "";
    let toSend = JSON.stringify({"message": message});

    ajaxPostRequest("/send", toSend, displayChat);
}
```

First, get our textbox element

Using Our New Functions

```
function sendMessage(){
    let messageElement = document.getElementById("message");

    let message = messageElement.value;
    messageElement.value = "";
    let toSend = JSON.stringify({"message": message});

    ajaxPostRequest("/send", toSend, displayChat);
}
```

Then get the text and clear it

Using Our New Functions

```
function sendMessage(){
    let messageElement = document.getElementById("message");

    let message = messageElement.value;
    messageElement.value = "";
    let toSend = JSON.stringify({"message": message});

    ajaxPostRequest("/send", toSend, displayChat);
}
```

Finally, convert it to JSON and send it in a POST request

Chat Server Design

What do we need:

- ✓ A front end web page (with interactive components)
- ✓ Front end JavaScript allowing users to interact with the page
- ✓ Web server code to run the server and handle requests
- ✓ A place to store messages that persists even when server stops

5. A way for the front end and back end to communicate ***even after the page is initially loaded***

Chat Server Design

What do we need:

- ✓ A front end web page (with interactive components)
- ✓ Front end JavaScript allowing users to interact with the page
- ✓ Web server code to run the server and handle requests
- ✓ A place to store messages that persists even when server stops
- ✓ A way for the front end and back end to communicate ***even after the page is initially loaded***

Our Chat Server Diagram

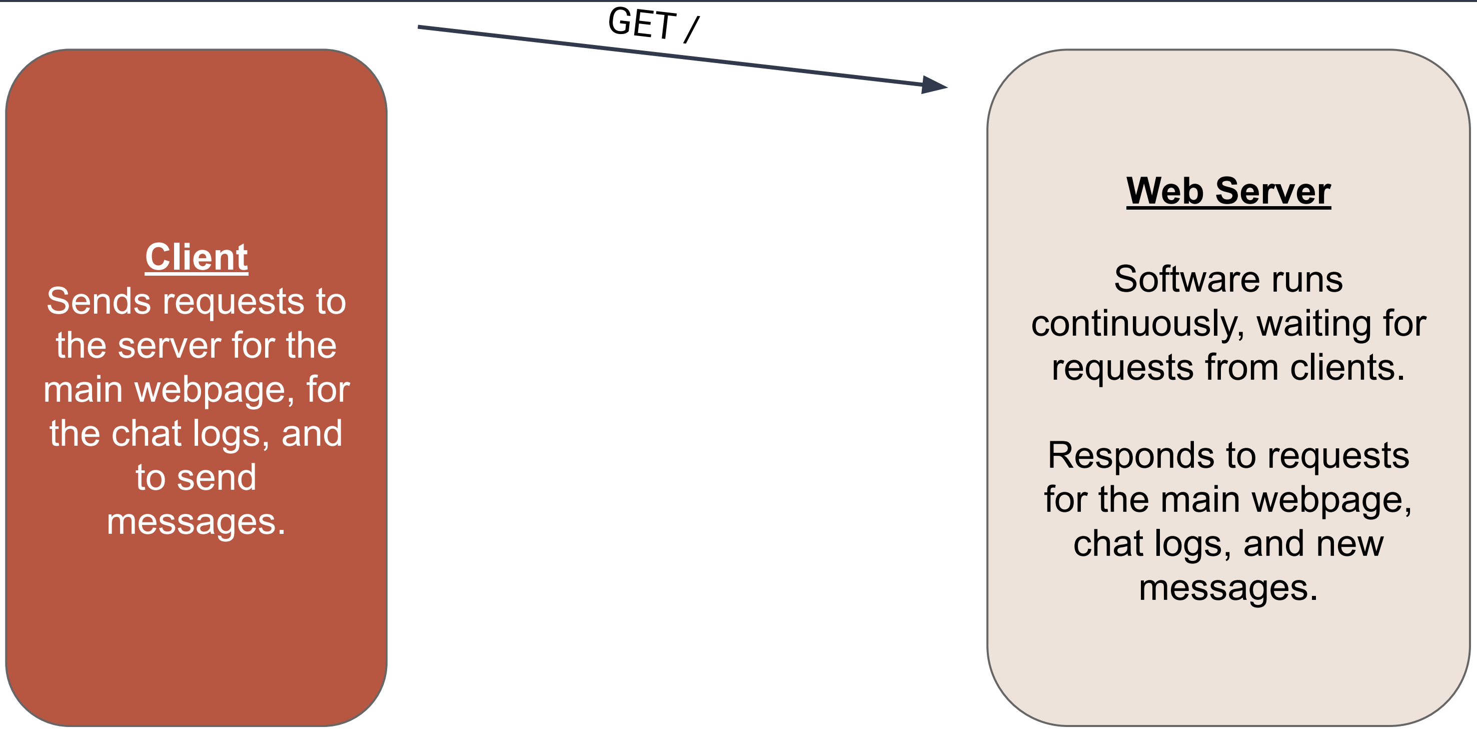
Client
Sends requests to the server for the main webpage, for the chat logs, and to send messages.

Web Server

Software runs continuously, waiting for requests from clients.

Responds to requests for the main webpage, chat logs, and new messages.

Our Chat Server Diagram



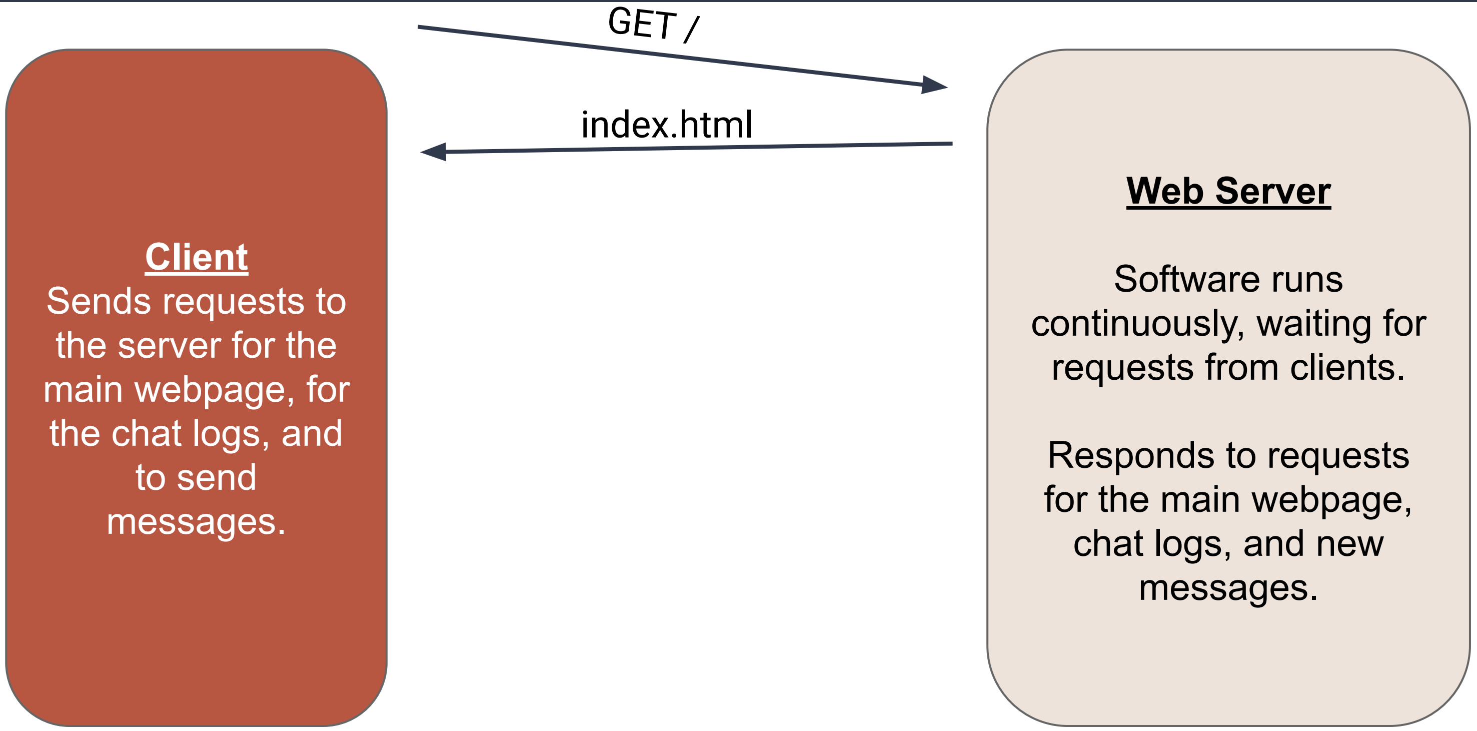
Client
Sends requests to the server for the main webpage, for the chat logs, and to send messages.

Web Server

Software runs continuously, waiting for requests from clients.

Responds to requests for the main webpage, chat logs, and new messages.

Our Chat Server Diagram



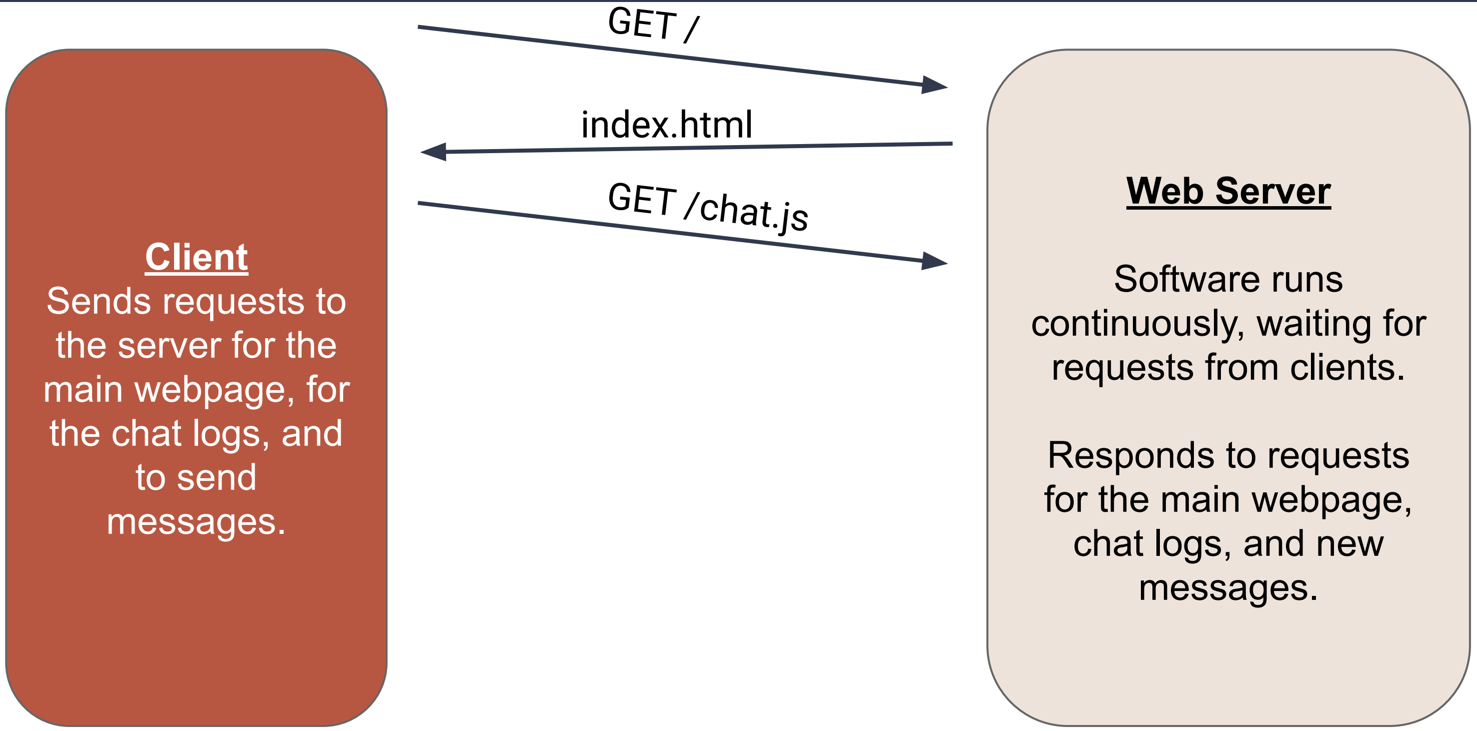
Client
Sends requests to the server for the main webpage, for the chat logs, and to send messages.

Web Server

Software runs continuously, waiting for requests from clients.

Responds to requests for the main webpage, chat logs, and new messages.

Our Chat Server Diagram



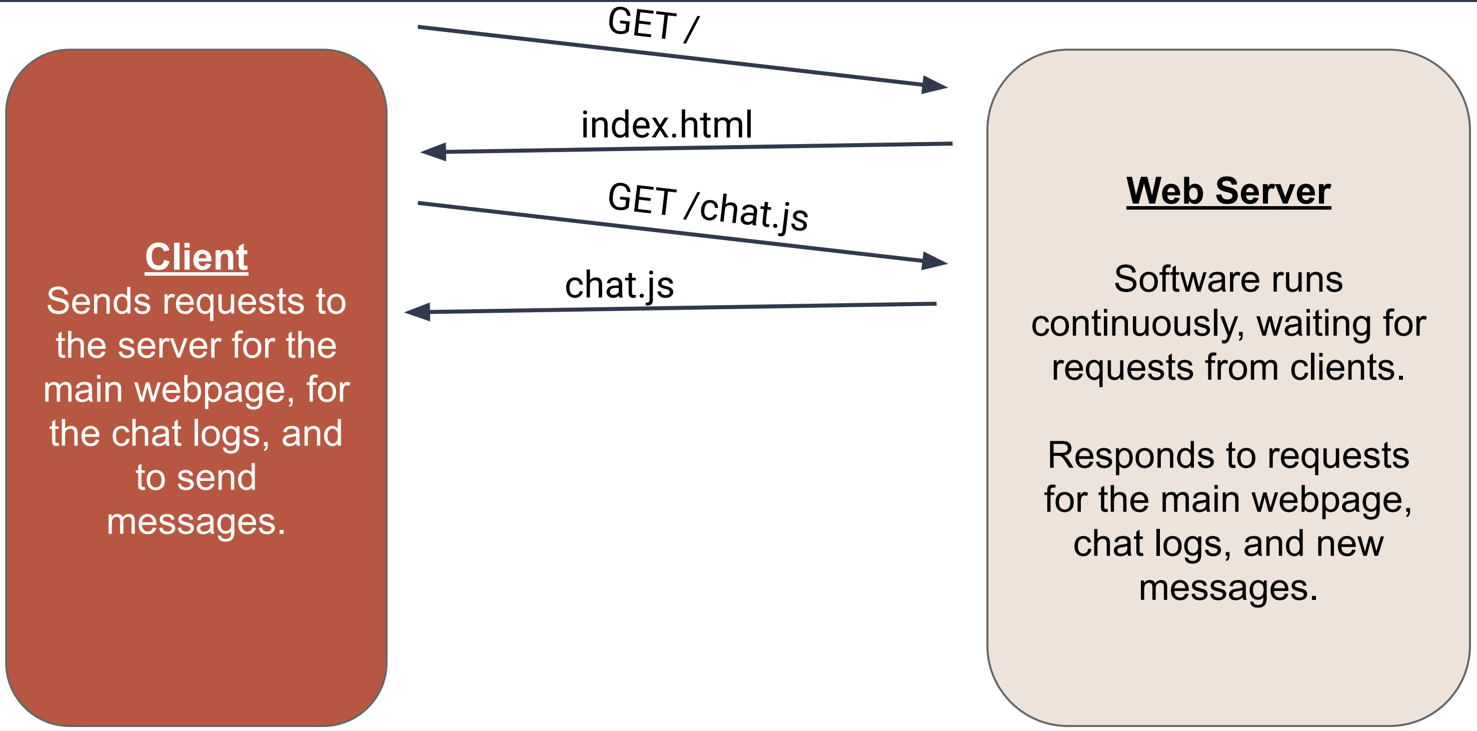
Client
Sends requests to the server for the main webpage, for the chat logs, and to send messages.

Web Server

Software runs continuously, waiting for requests from clients.

Responds to requests for the main webpage, chat logs, and new messages.

Our Chat Server Diagram



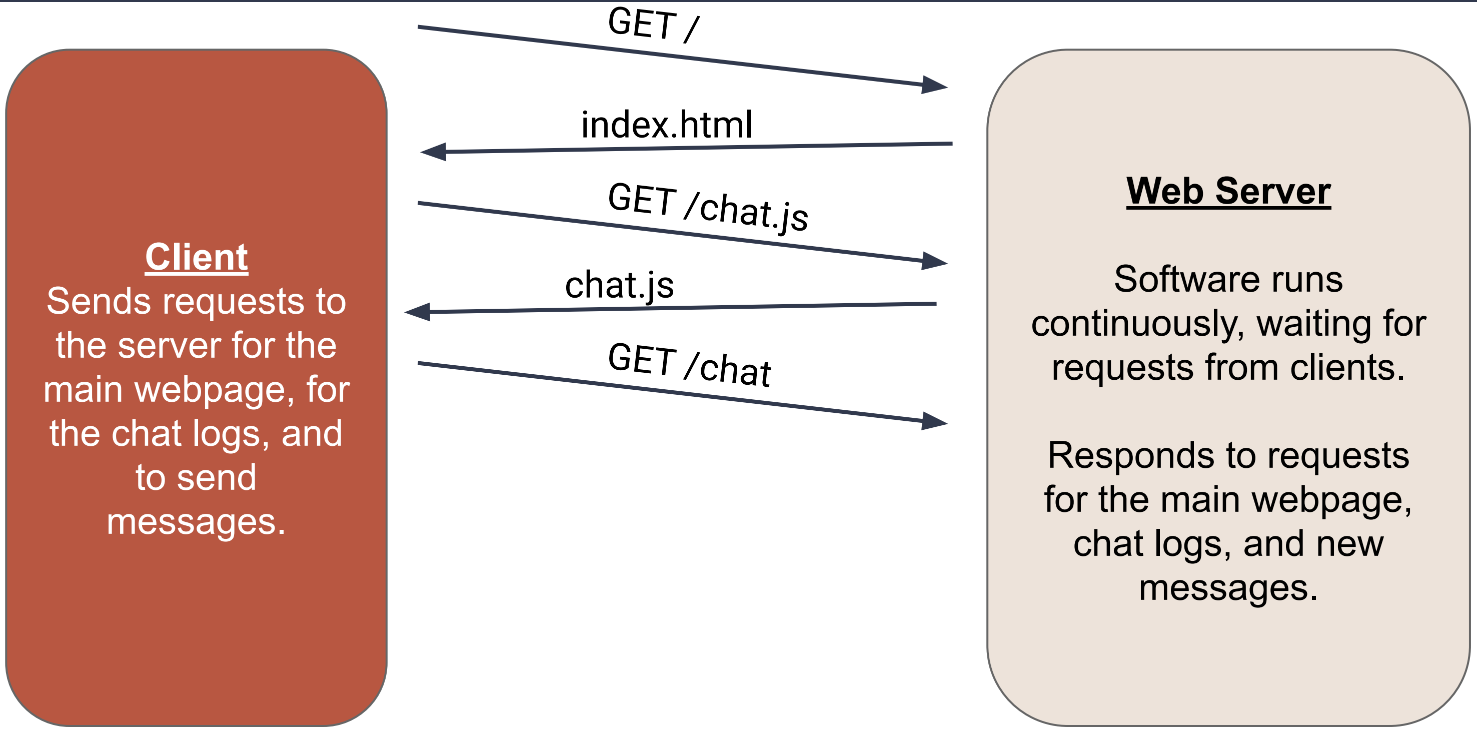
Client
Sends requests to the server for the main webpage, for the chat logs, and to send messages.

Web Server

Software runs continuously, waiting for requests from clients.

Responds to requests for the main webpage, chat logs, and new messages.

Our Chat Server Diagram



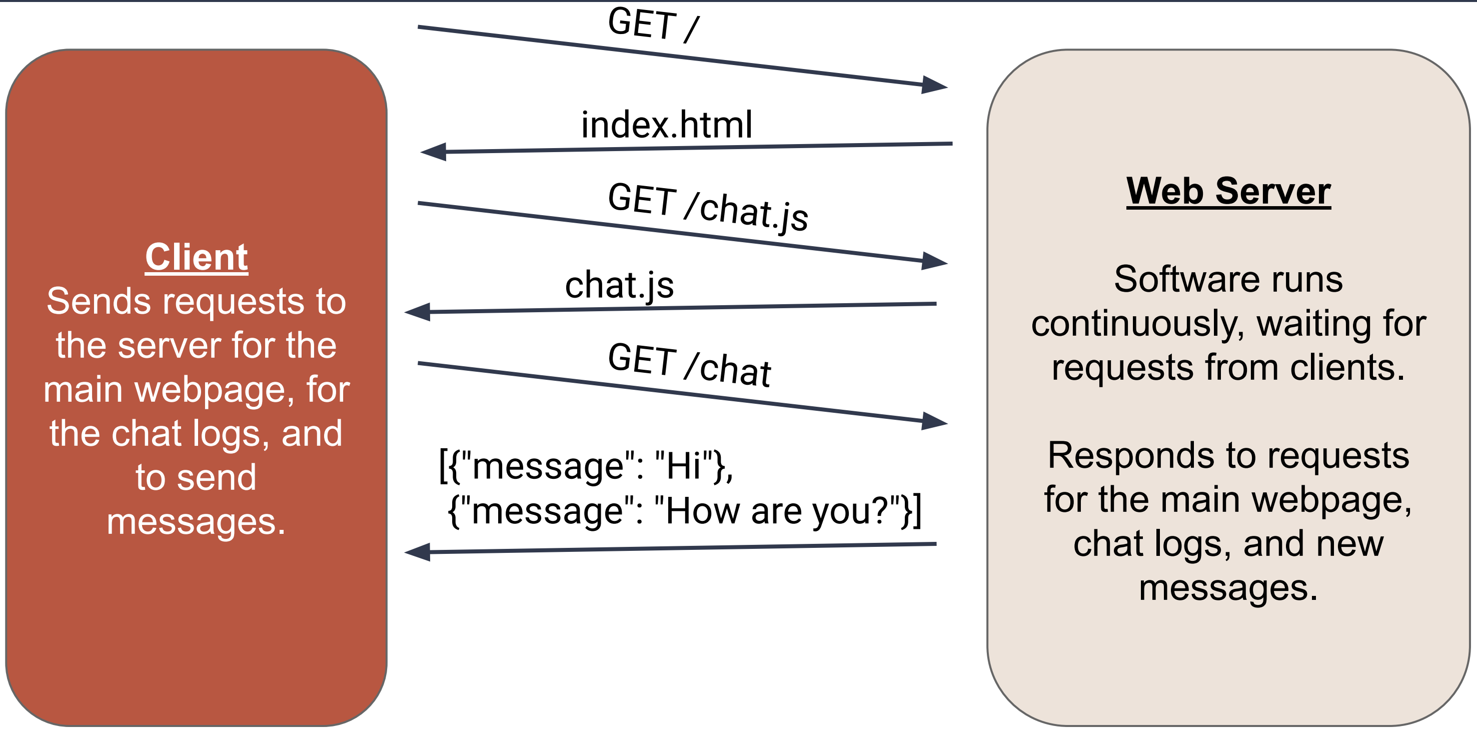
Client
Sends requests to the server for the main webpage, for the chat logs, and to send messages.

Web Server

Software runs continuously, waiting for requests from clients.

Responds to requests for the main webpage, chat logs, and new messages.

Our Chat Server Diagram



[{"message": "Hi"},
{"message": "How are you?"}]

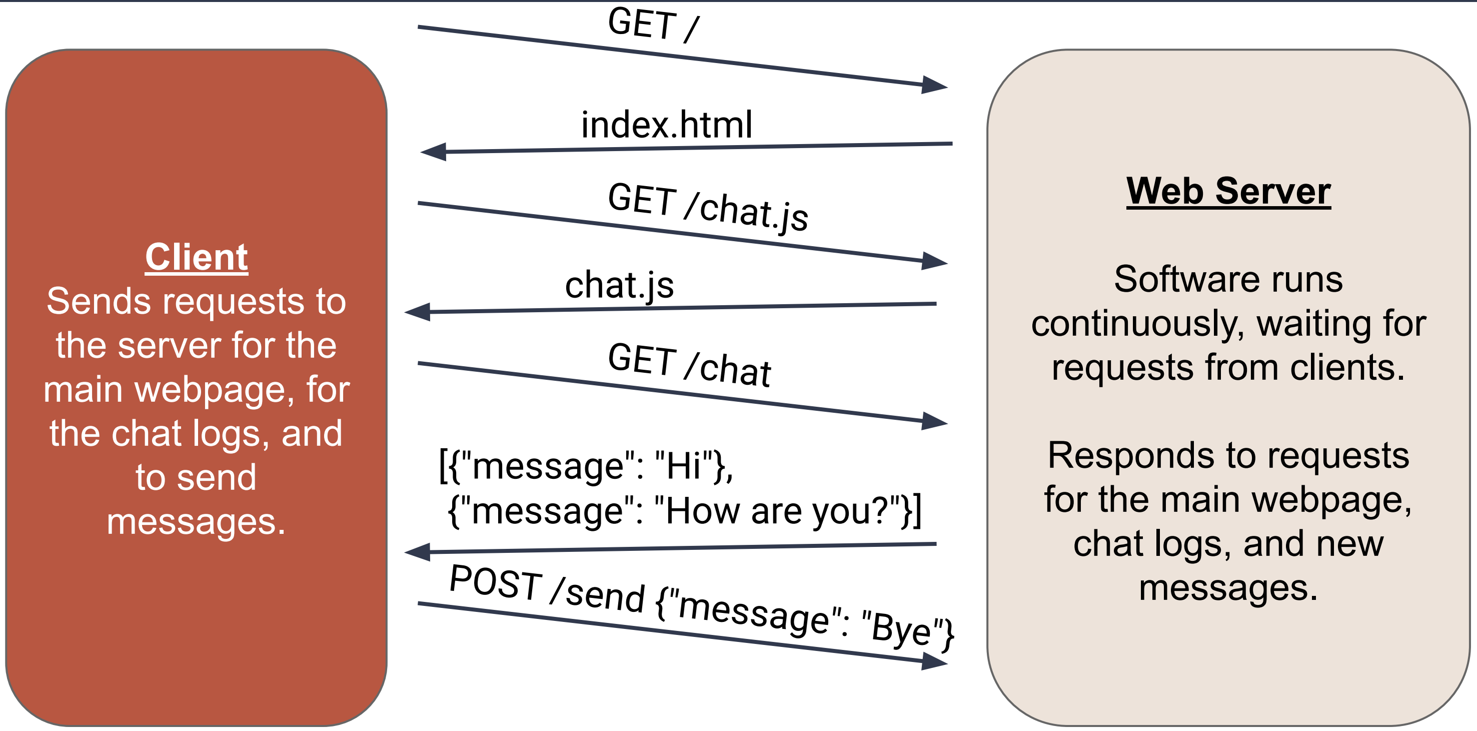
Client
Sends requests to the server for the main webpage, for the chat logs, and to send messages.

Web Server

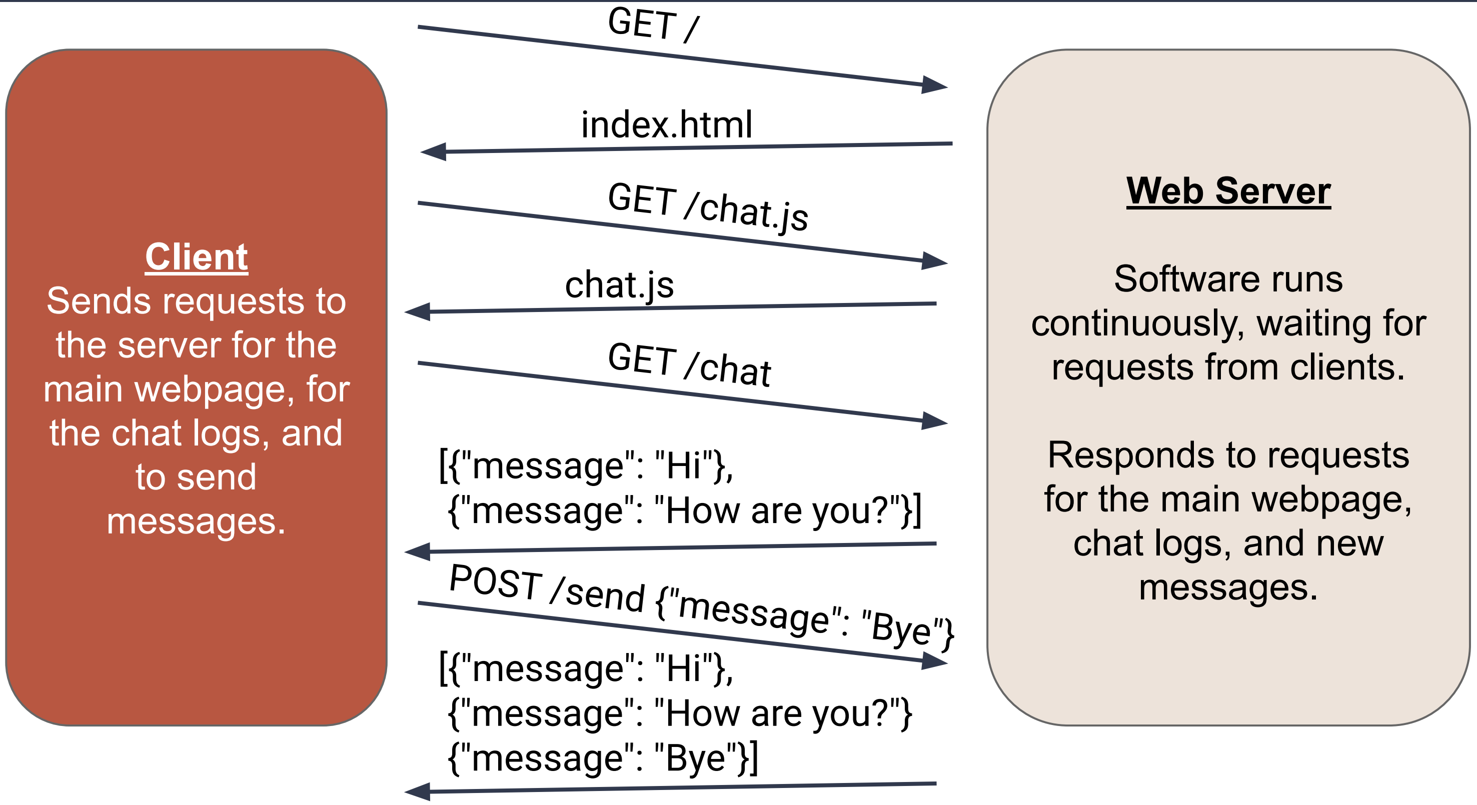
Software runs continuously, waiting for requests from clients.

Responds to requests for the main webpage, chat logs, and new messages.

Our Chat Server Diagram


Client
Sends requests to the server for the main webpage, for the chat logs, and to send messages.
GET /
index.html
GET /chat.js
chat.js
GET /chat
[{"message": "Hi"},
{"message": "How are you?"}]
POST /send {"message": "Bye"}
Web Server
Software runs continuously, waiting for requests from clients.
Responds to requests for the main webpage, chat logs, and new messages.

Our Chat Server Diagram



Client
Sends requests to the server for the main webpage, for the chat logs, and to send messages.

Web Server

Software runs continuously, waiting for requests from clients.

Responds to requests for the main webpage, chat logs, and new messages.

Extending Our Example

What if we wanted to include our name with each message?

What would we need to add/change in our code?

Extending Our Example

What if we wanted to include our name with each message?

What would we need to add/change in our code?

1. **We must have a way to input our name (in `index.html`)**

Extending Our Example

What if we wanted to include our name with each message?

What would we need to add/change in our code?

1. **We must have a way to input our name (in `index.html`)**
2. **We must send the name *and* message to the server (in `chat.js`)**

Extending Our Example

What if we wanted to include our name with each message?

What would we need to add/change in our code?

1. **We must have a way to input our name (in `index.html`)**
2. **We must send the name *and* message to the server (in `chat.js`)**
3. **The server must store the name *and* message (in `chat.py`)**

Extending Our Example

What if we wanted to include our name with each message?

What would we need to add/change in our code?

1. **We must have a way to input our name (in `index.html`)**
2. **We must send the name *and* message to the server (in `chat.js`)**
3. **The server must store the name *and* message (in `chat.py`)**
4. **The server must send/receive the names *and* messages (in `main.py`)**

Extending Our Example

What if we wanted to include our name with each message?

What would we need to add/change in our code?

1. **We must have a way to input our name (in `index.html`)**
2. **We must send the name *and* message to the server (in `chat.js`)**
3. **The server must store the name *and* message (in `chat.py`)**
4. **The server must send/receive the names *and* messages (in `main.py`)**
5. **We must display the names *and* messages (in `chat.js`)**

Inputting a Name (index.html)

We can add a text box to our webpage for name, and give it an id that the JavaScript can use to access the value:

```
Name: <input type="text" id="name"><br/>
```

Sending the Message (chat.js)

Instead of just sending { "message": "..." } as our JSON string, we can send { "name": "...", "message": "..." }. We can get the values for name and message from the text boxes:

```
let nameElement = document.getElementById("name");
let name = nameElement.value;
let toSend = JSON.stringify({"name": name, "message": message});
```

Storing the Message (chat.py)

How can we store more than just the message in our chat log?

JSON strings are just strings…we can write strings to text files…

When adding a message, dump it to a JSON string and write:

```
file.write(json.dumps(message) + "\n")
```

When reading the logs, read the line (JSON string) and convert to data:

```
full_chat.append(json.loads(line.rstrip("\n")))
```

Sending/Receiving the Message (main.py)

The only small change to the server is that when receiving a message, it should just add the entire dictionary to the chat log, rather than just the message text:

```
chat.add_message(content['message']) → chat.add_message(content)
```

Displaying the Message

Now the message data received from the server has a name and a message, so display both in the chat:

```
chat = chat + data.name + ": " + data.message + "</br>";
```

Project Checklist

Front-End Requirements:

- HTML
- AJAX
- Callback functions

Back-End Requirements:

- Bottle routes
- Data retrieval (HTTP requests)
- Data cleaning and processing
- Local data caching

Project Checklist

Front-End Requirements:

- ✓ HTML
- ✓ AJAX
- ✓ Callback functions

Back-End Requirements:

- ✓ Bottle routes
- ✓ Data retrieval (HTTP requests)
- ✓ Data cleaning and processing
- ● Local data caching