CSE 503

Introduction to Computer Science for Non-Majors

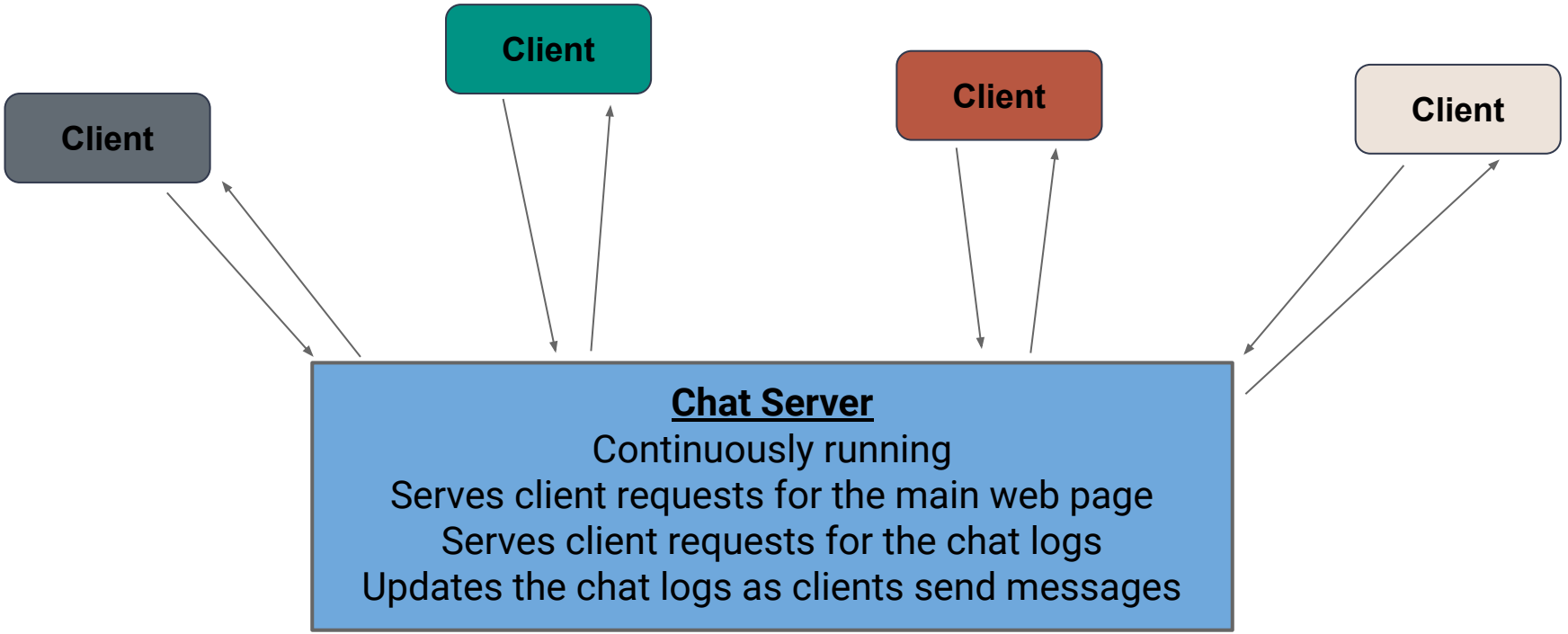
Dr. Eric Mikida
epmikida@buffalo.edu
208 Capen Hall

Day 25
AJAX (Part 2)

Recap

- Last time we started making our own chat server…

End Goal


Client
Client
Client
Client
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
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***

Storing Chat Logs (chat.txt)

Now we can create a place on the server to store the chat logs…

Storing Chat Logs (chat.txt)

Now we can create a place on the server to store the chat logs…

In this case we can just store them in a text file (let's call it `chat.txt`)

Reading and Writing Chat Logs (chat.py)

Now that we have a place to store our chat, we need to be able to read and write from the chat logs.

We'll write this in `chat.py` to keep it separate from server code.

Reading and Writing Chat Logs (chat.py)

```
filename = "chat.txt"

def get_chat():
  full_chat = []
  with open(filename) as file:
    for line in file:
      full_chat.append({"message": line.rstrip("\n")})
  return full_chat

def add_message(message):
  with open(filename, "a") as file:
    file.write(message + "\n")
```

Reading and Writing Chat Logs (chat.py)

```
filename = "chat.txt"

def get_chat():
  full_chat = []
  with open(filename) as file:
    for line in file:
      full_chat.append({"message": line.rstrip("\n")})
  return full_chat

def add_message(message):
  with open(filename, "a") as file:
    file.write(message + "\n")
```

Create a variable with the filename so we can refer to it throughout the rest of the code

Reading and Writing Chat Logs (chat.py)

```
filename = "chat.txt"

def get_chat():
  full_chat = []
  with open(filename) as file:
    for line in file:
      full_chat.append({"message": line.rstrip("\n")})
  return full_chat

def add_message(message):
  with open(filename, "a") as file:
    file.write(message + "\n")
```

Read from the chat file, and return a list of messages. We've put the messages in a dictionary...more on that later.

Reading and Writing Chat Logs (chat.py)

```
filename = "chat.txt"

def get_chat():
  full_chat = []
  with open(filename) as file:
    for line in file:
      full_chat.append({"message": line.rstrip("\n")})
  return full_chat

def add_message(message):
  with open(filename, "a") as file:
    file.write(message + "\n")
```

Write a function to add a new message to the chat file. Note the file mode: **"a"**. This means append.

Reading and Writing Chat Logs (chat.py)

Note that `chat.py` does not have any server code...it just reads and writes files, and we can test it just like any other Python code.

Reading and Writing Chat Logs (chat.py)

Note that `chat.py` does not have any server code...it just reads and writes files, and we can test it just like any other Python code.

When building applications from smaller pieces, make sure to test the pieces individually, let's do that now with `chat.py`.

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
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***

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:

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

← Loads the JSON library

In JavaScript:

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

JSON in JavaScript and Python

In Python:

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

← Loads the JSON library

← Takes a JSON string and returns 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.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())
```

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

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

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 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 the names *and* messages (in `main.py`)**
5. **We must display the names *and* messages (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`)**
4. **The server must send the names *and* messages (in `main.py`)**
5. **We must display the names *and* messages (in `chat.js`)**