

# WebSockets

Yeah, about that whole “stateless” thing...

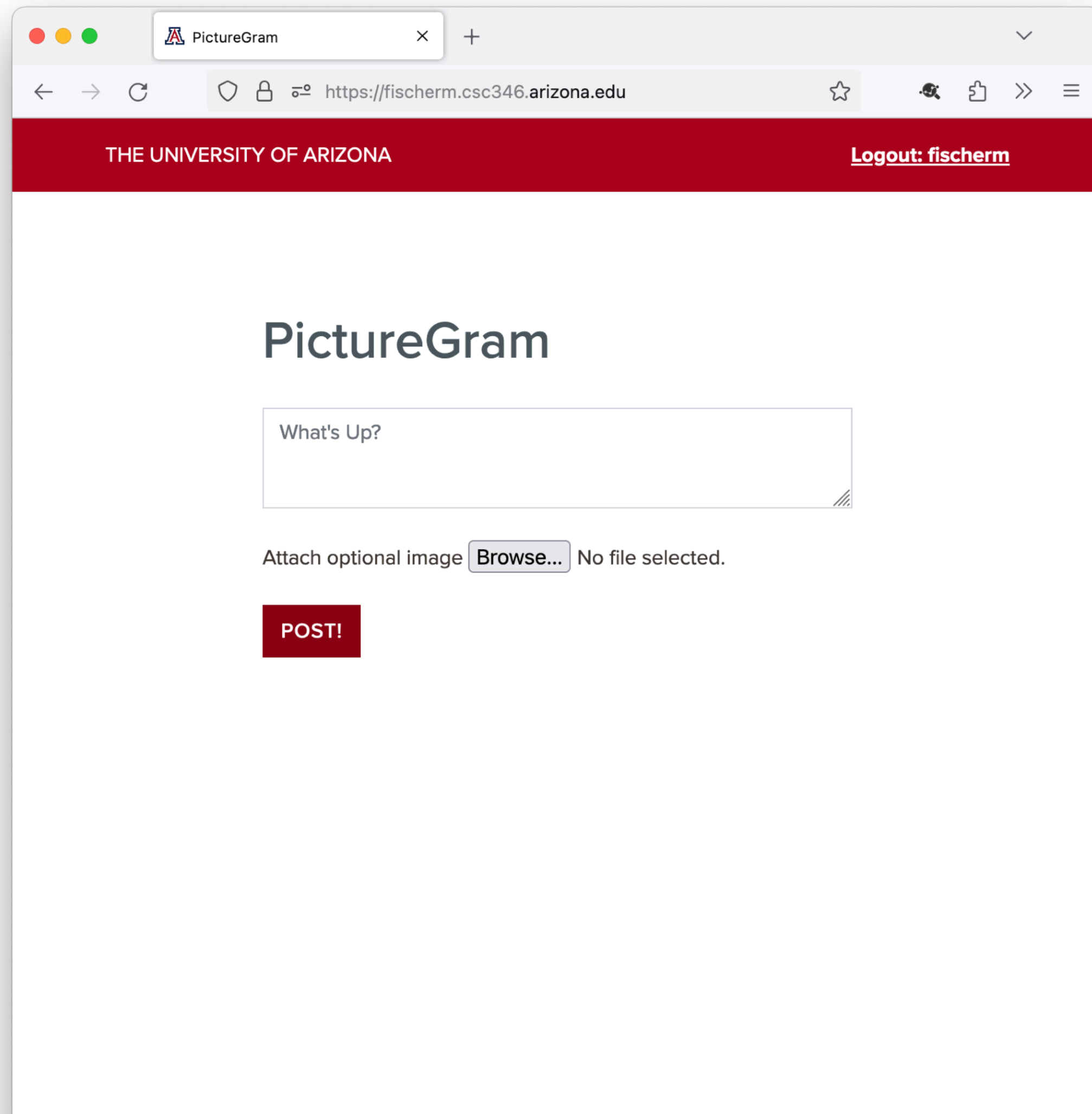
# WebSockets

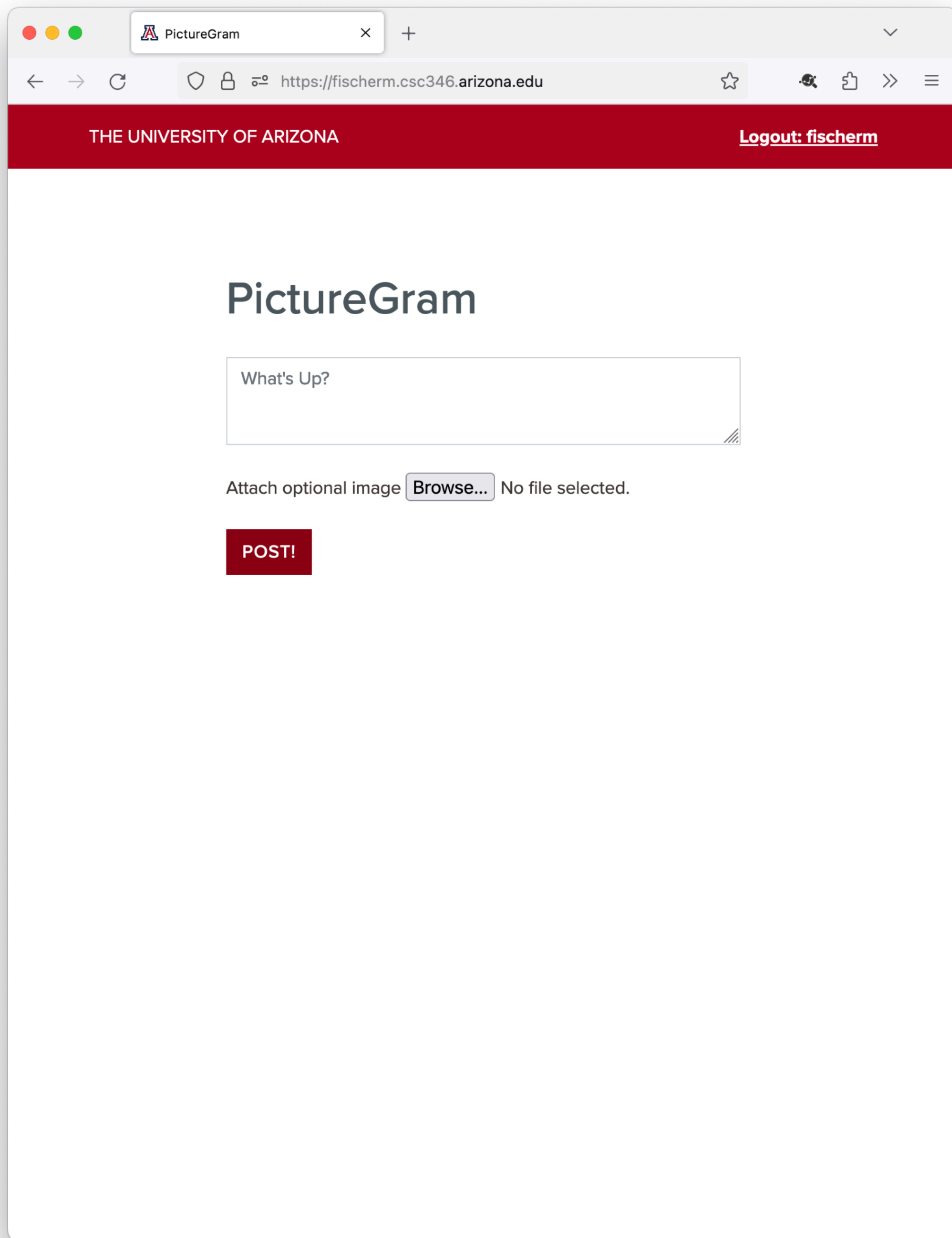
## Sometimes you just need a constant connection

- Recall that the HTTP protocol is stateless.
  - Each HTTP request is separate and isolated from any other ones.
  - We've repeated this more than a few times this semester 🤪
- What are some of the use cases where a stateless network model starts to fail?

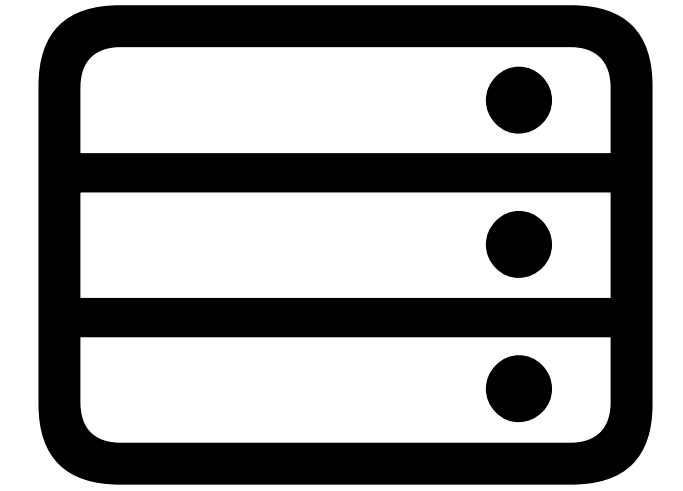
# Chat

## How does our Chat App get new chat messages?

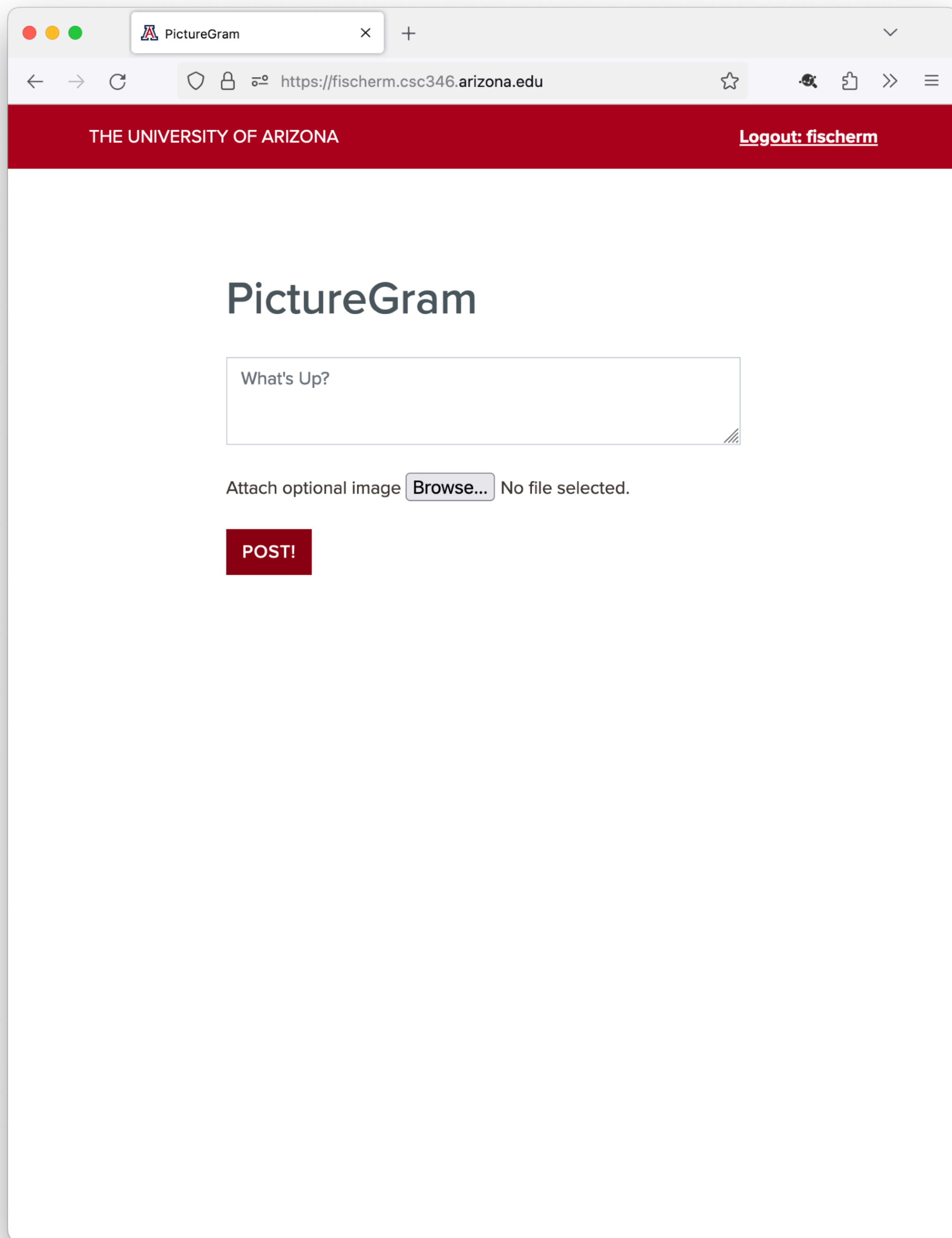




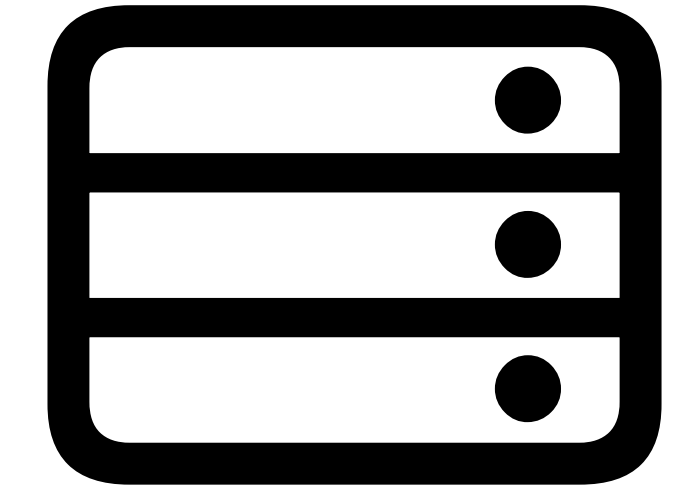
On Page Load:  
Request New Posts Messages



api.csc346.arizona.edu



API Responds with  
new post JSON data



api.csc346.arizona.edu

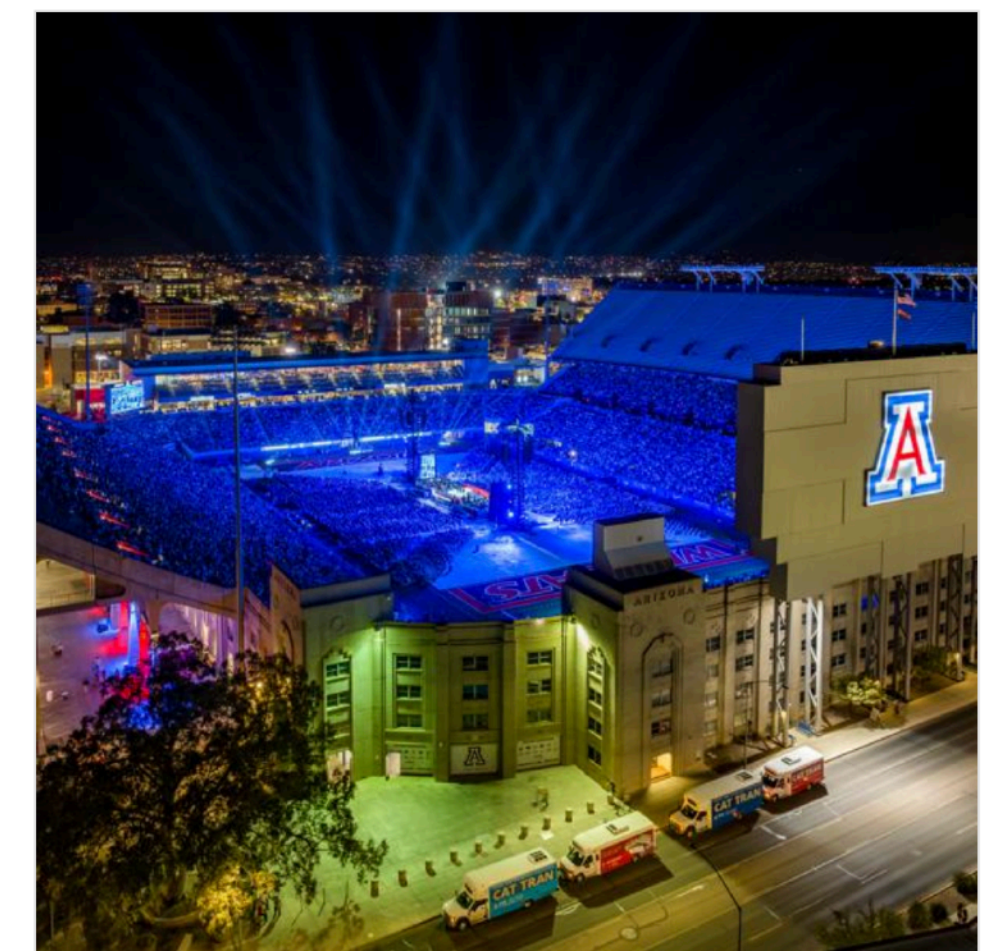


Image Post Test 52945207-df8d-447f-9d4c-c041794fc8a5  
@fischerm\_student (4/15/2024)

PictureGram

← → ↻ 🔒 🔍 📄 >> ☰

https://fischerm.csc346.arizona.edu

THE UNIVERSITY OF ARIZONA [Logout: fischerm](#)

# PictureGram

What's Up?

Attach optional image  No file selected.

**POST!**

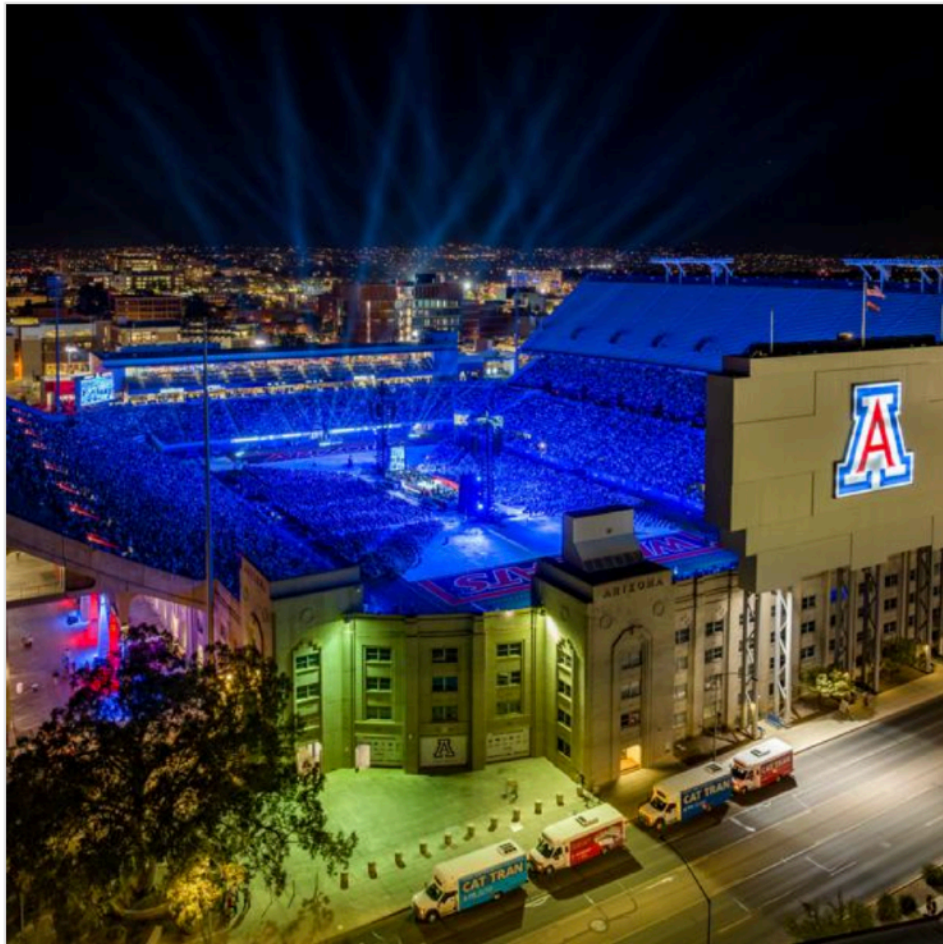
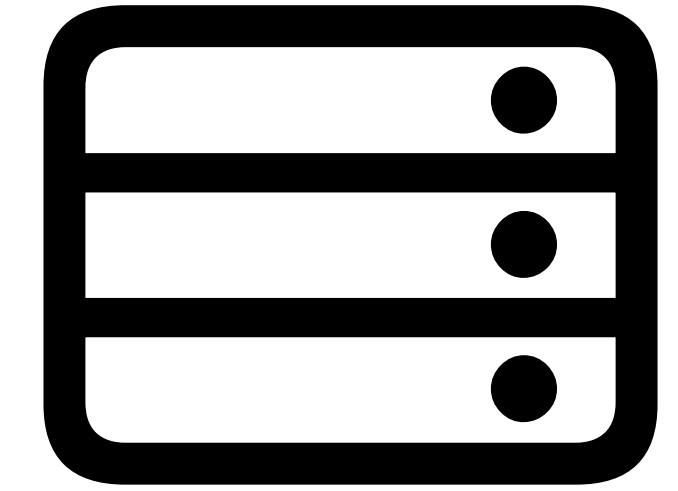


Image Post Test 52945207-df8d-447f-9d4c-c041794fc8a5  
@fischerm\_student (4/15/2024)



api.csc346.arizona.edu

PictureGram

← → ↻ 🔒 🔍 📄 ⌵

https://fischerm.csc346.arizona.edu

THE UNIVERSITY OF ARIZONA [Logout: fischerm](#)

# PictureGram

What's Up?

Attach optional image [Browse...](#) No file selected.

**POST!**

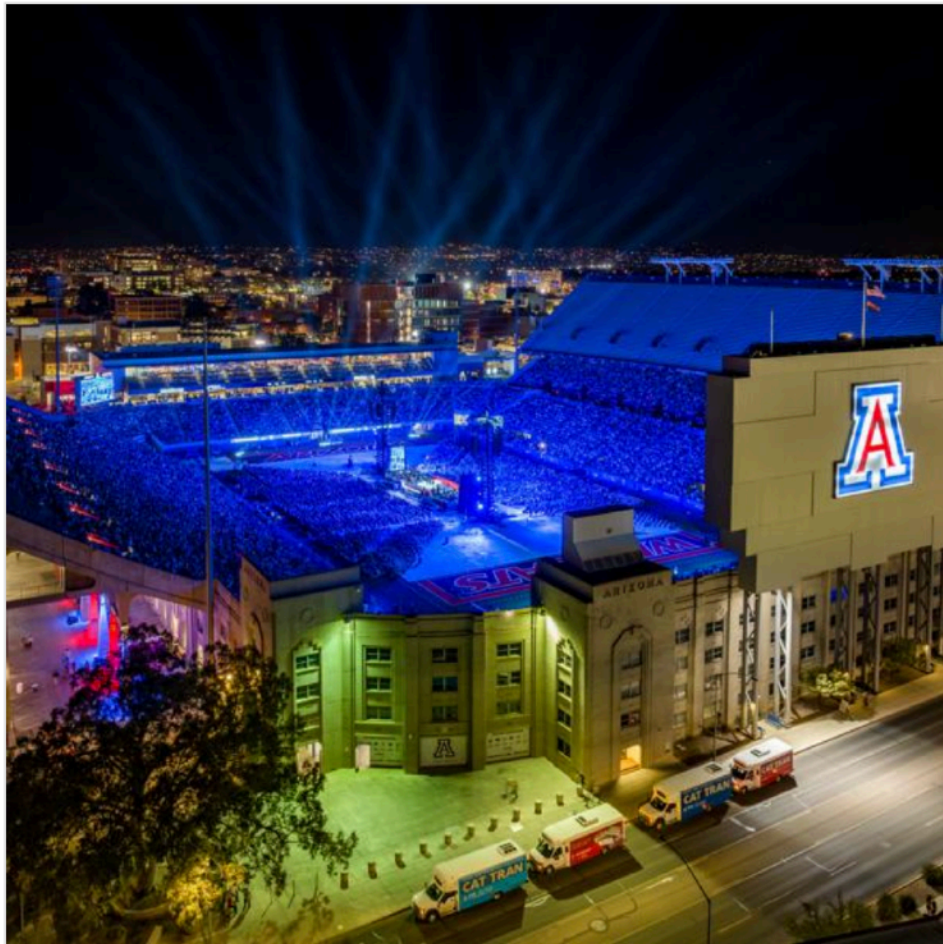
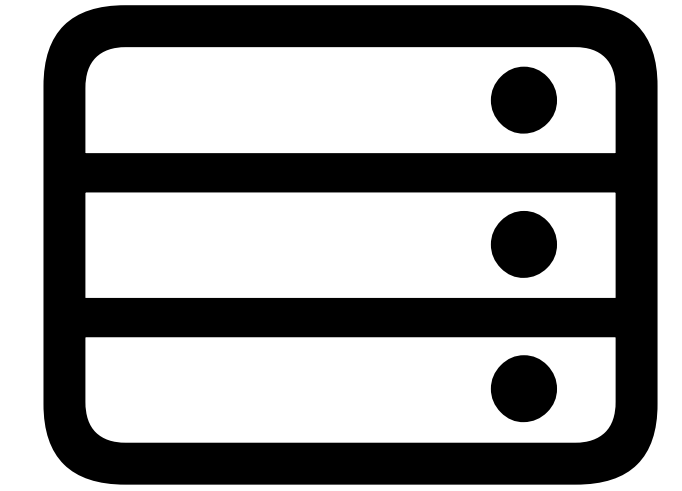
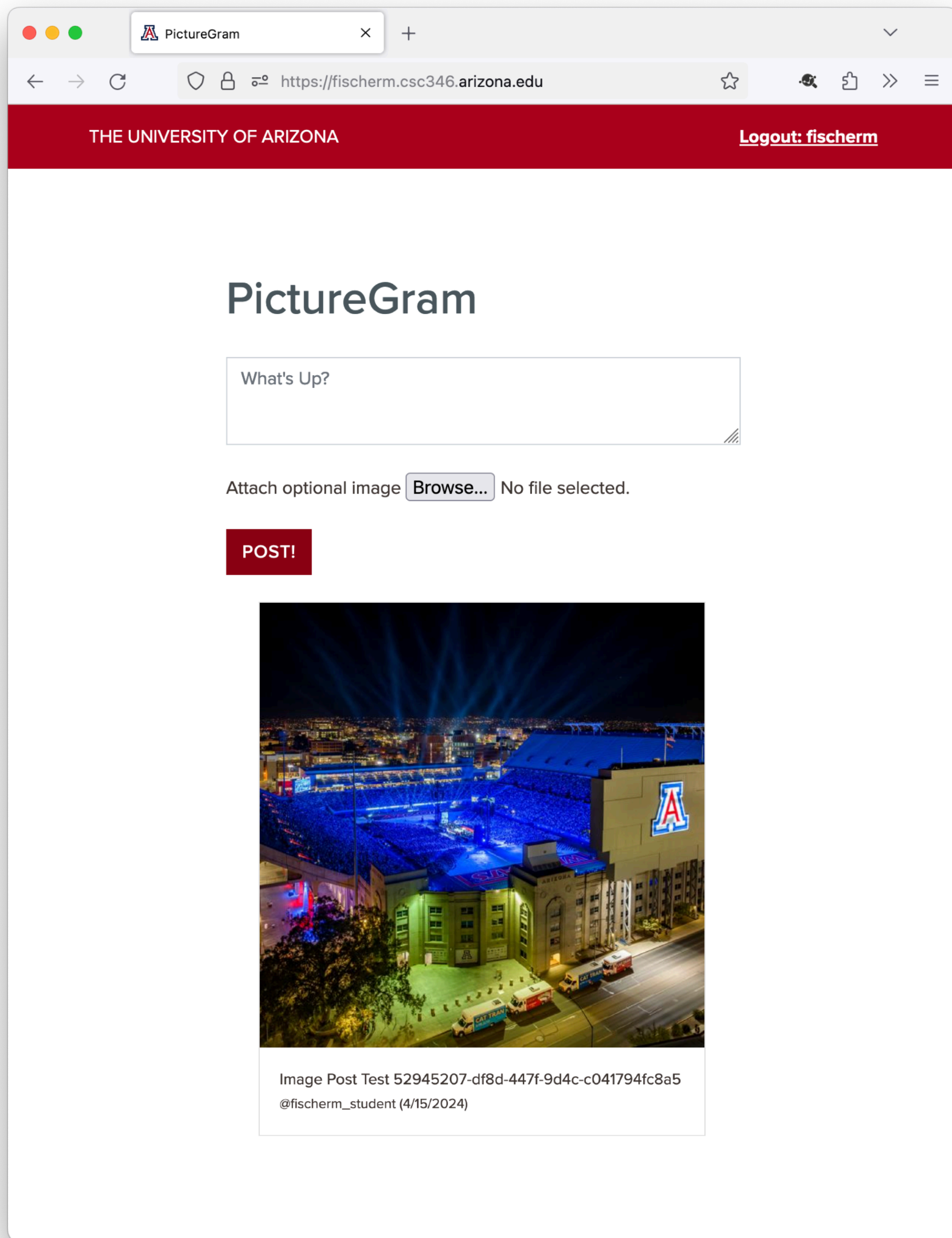


Image Post Test 52945207-df8d-447f-9d4c-c041794fc8a5  
@fischerm\_student (4/15/2024)

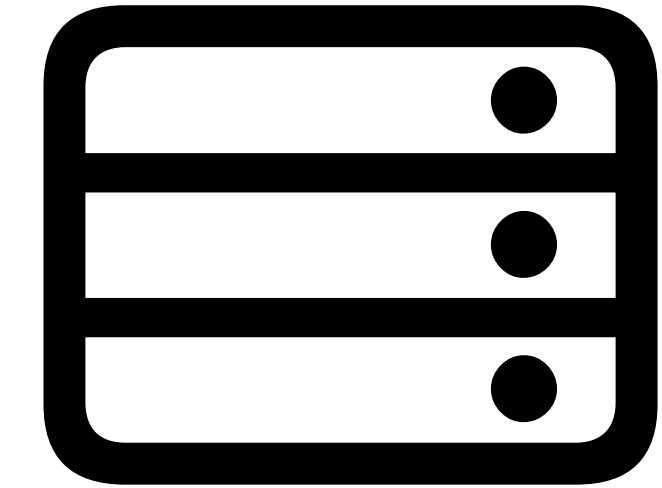


api.csc346.arizona.edu

- Now What?
- If new messages are posted by someone else, how does this browser get them?
- Currently you have to reload the page



# Polling



api.csc346.arizona.edu

- A common approach is known as polling
- The browser checks with the API on a timer and asks for new chat messages



PictureGram

https://fischerm.csc346.arizona.edu

THE UNIVERSITY OF ARIZONA [Logout: fischerm](#)

# PictureGram

What's Up?

Attach optional image [Browse...](#) No file selected.

**POST!**

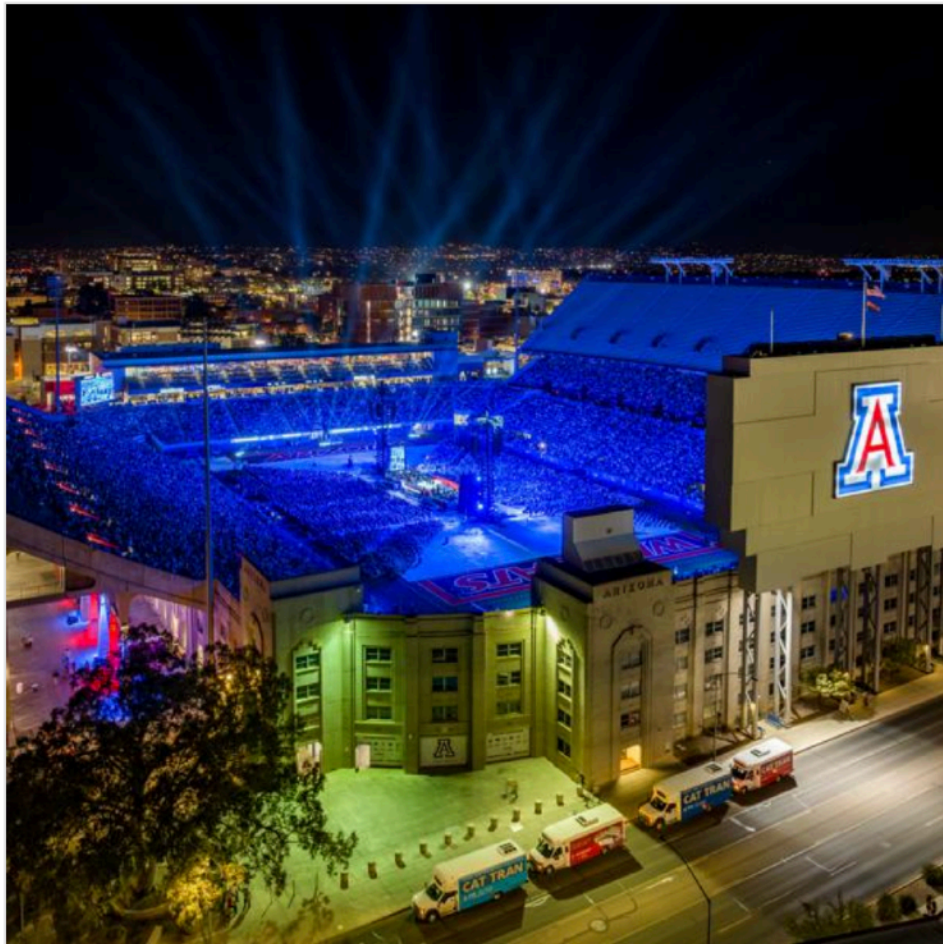
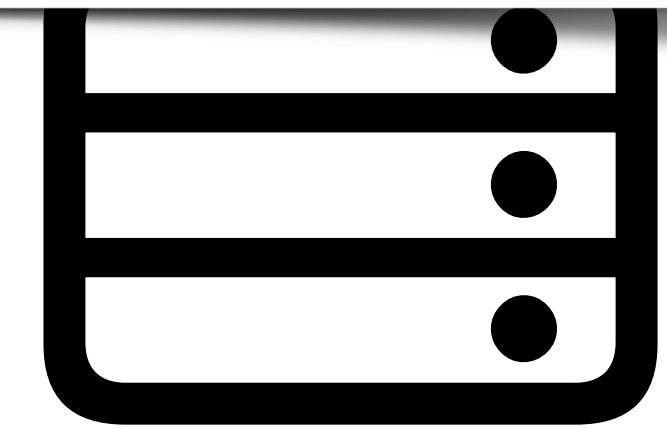


Image Post Test 52945207-df8d-447f-9d4c-c041794fc8a5  
@fischerm\_student (4/15/2024)

```
setTimeout( () => {  
  loadPosts(newestPostTimestamp, null)  
}, 10000)
```



api.csc346.arizona.edu

- The `setTimeout()` function will call the first argument at an interval specified in the second argument
- Here the second argument is `10000ms`, so every 10 seconds the `loadChats(...)` function is called

PictureGram

https://fischerm.csc346.arizona.edu

THE UNIVERSITY OF ARIZONA Logout: fischerm

# PictureGram

What's Up?

Attach optional image  No file selected.

**POST!**

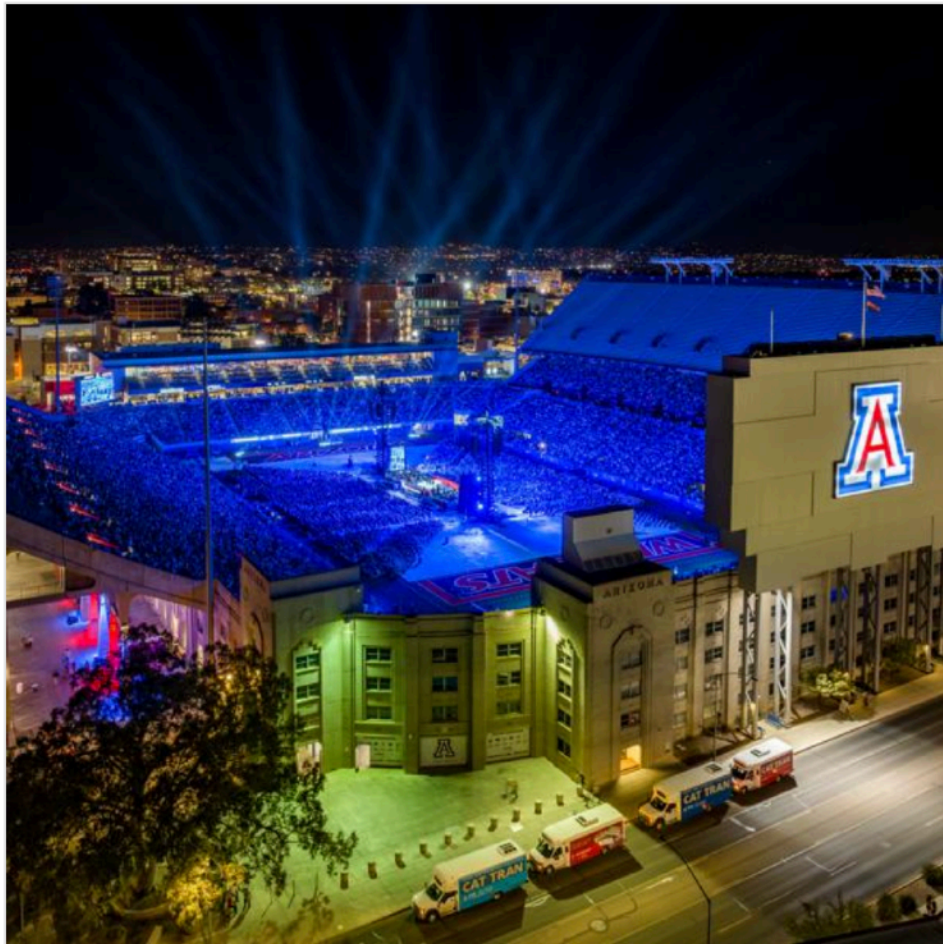
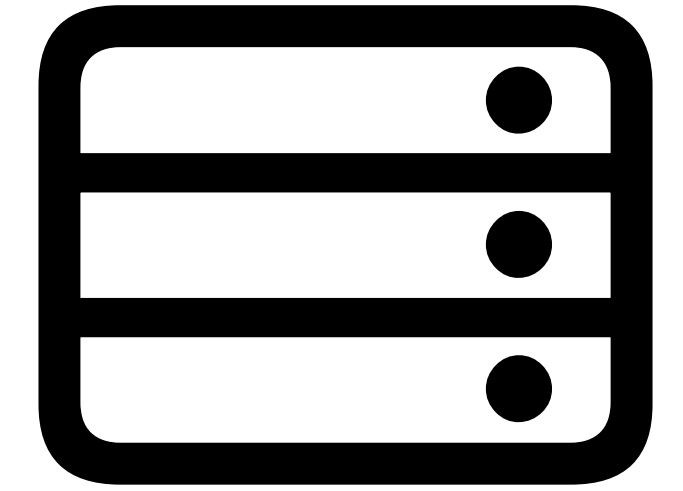


Image Post Test 52945207-df8d-447f-9d4c-c041794fc8a5  
@fischerm\_student (4/15/2024)

"Got any new posts?"



api.csc346.arizona.edu

PictureGram

https://fischerm.csc346.arizona.edu

THE UNIVERSITY OF ARIZONA [Logout: fischerm](#)

# PictureGram

What's Up?

Attach optional image [Browse...](#) No file selected.

**POST!**

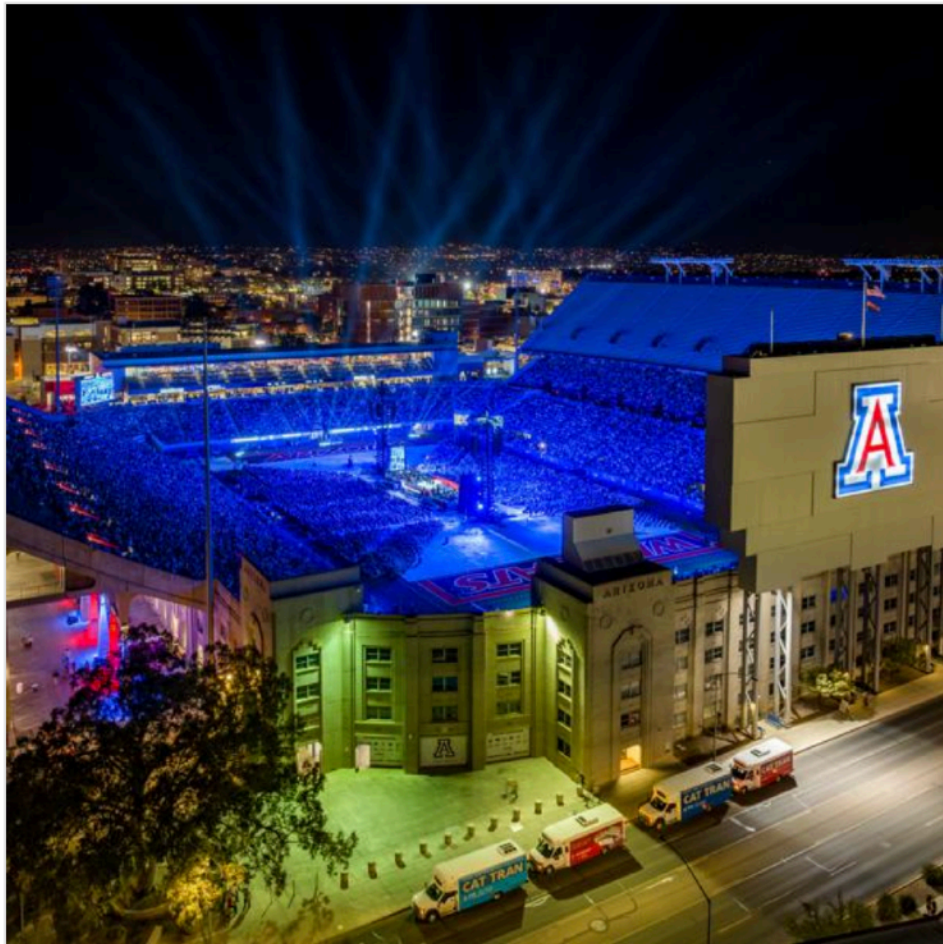


Image Post Test 52945207-df8d-447f-9d4c-c041794fc8a5  
@fischerm\_student (4/15/2024)





10



seconds



Later



PictureGram

https://fischerm.csc346.arizona.edu

THE UNIVERSITY OF ARIZONA Logout: fischerm

# PictureGram

What's Up?

Attach optional image  No file selected.

**POST!**

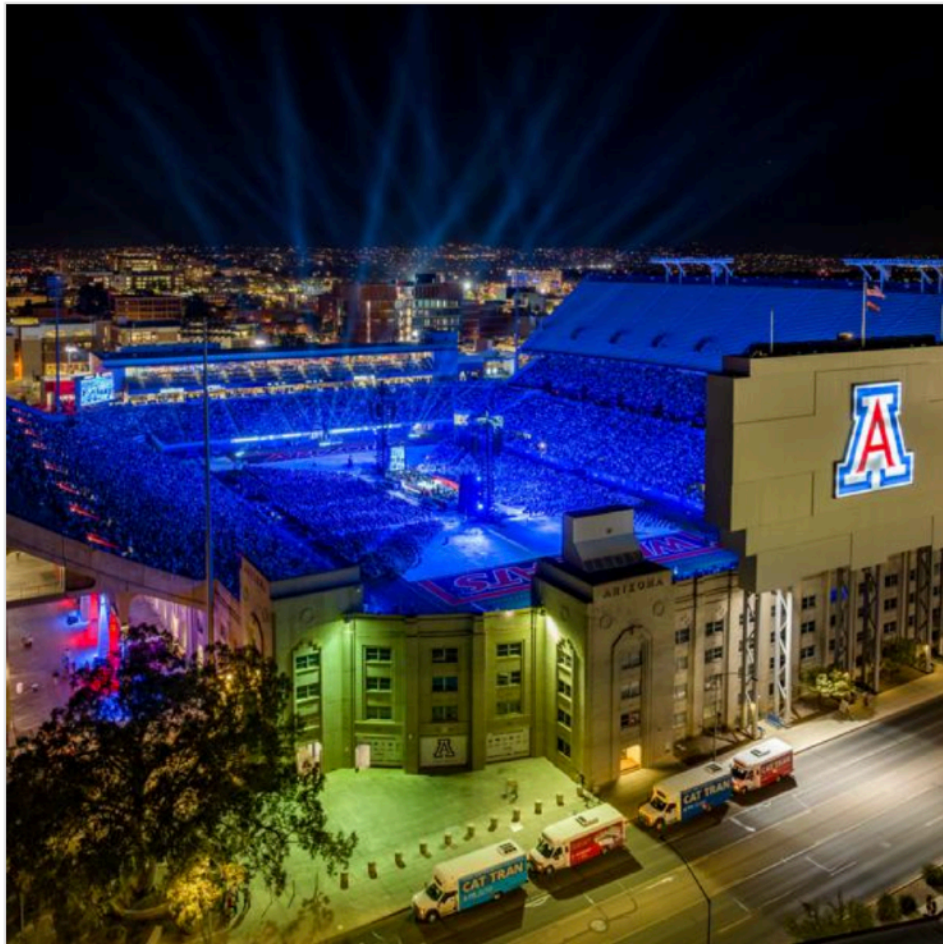
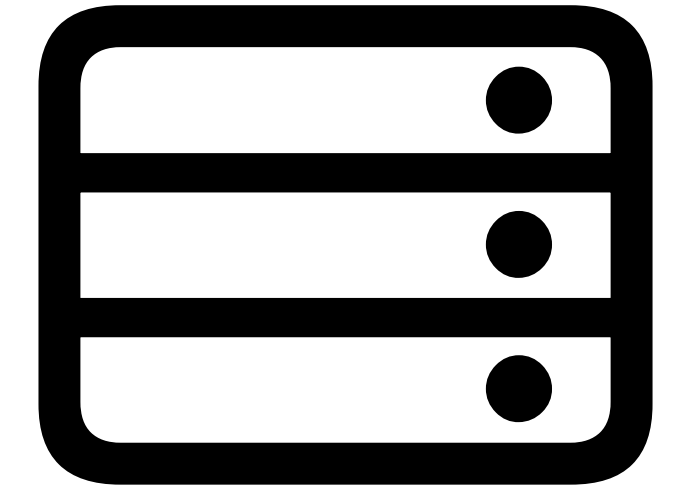


Image Post Test 52945207-df8d-447f-9d4c-c041794fc8a5  
@fischerm\_student (4/15/2024)

"Got any new posts?"



api.csc346.arizona.edu

PictureGram

https://fischerm.csc346.arizona.edu

THE UNIVERSITY OF ARIZONA Logout: fischerm

# PictureGram

What's Up?

Attach optional image  No file selected.

**POST!**

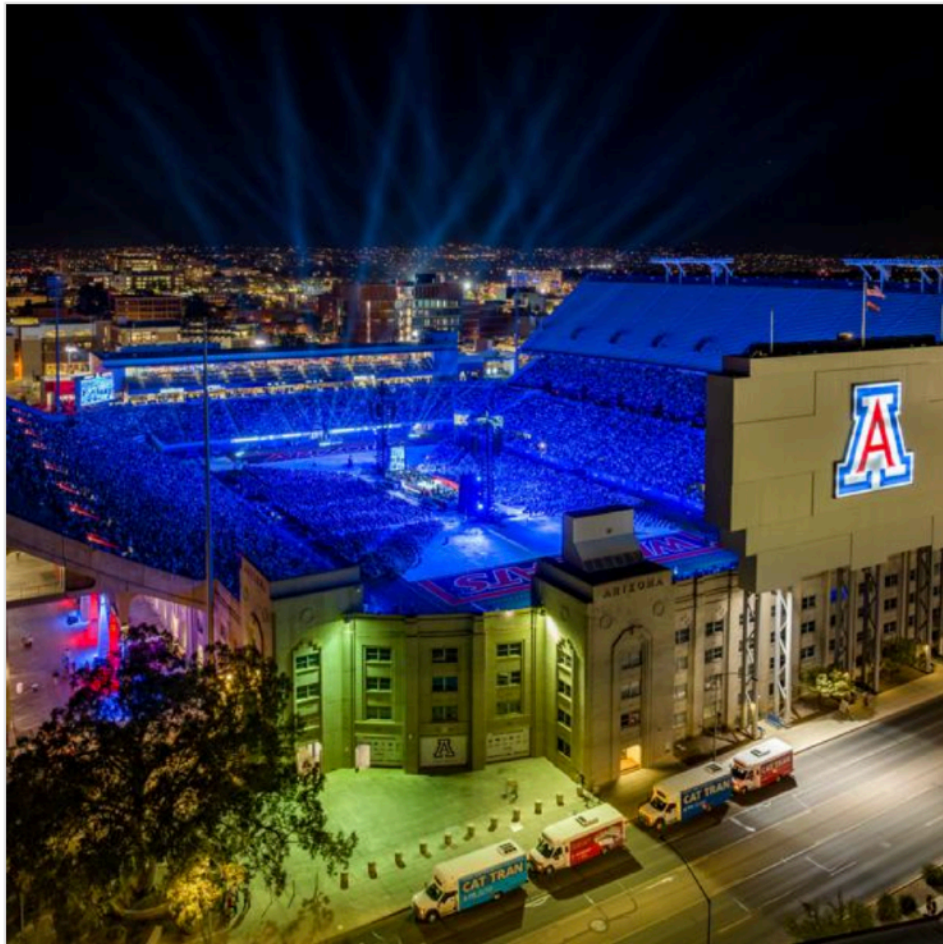


Image Post Test 52945207-df8d-447f-9d4c-c041794fc8a5  
@fischerm\_student (4/15/2024)





10



SECONDS



LATER



PictureGram

https://fischerm.csc346.arizona.edu

THE UNIVERSITY OF ARIZONA Logout: fischerm

# PictureGram

What's Up?

Attach optional image  No file selected.

**POST!**

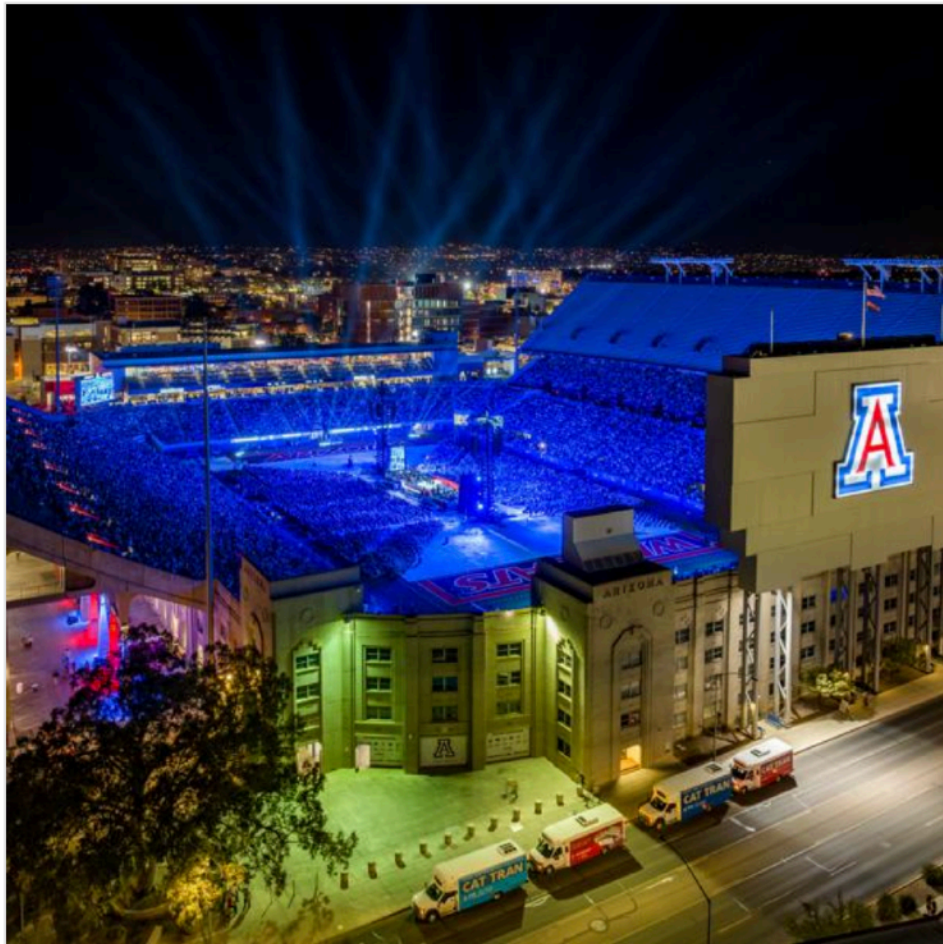
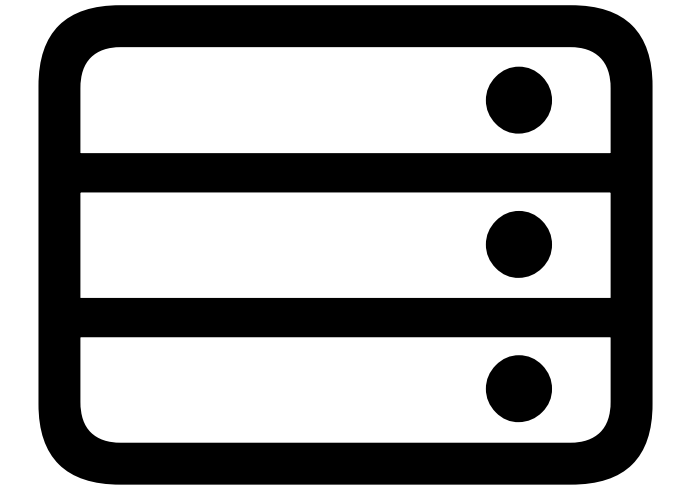


Image Post Test 52945207-df8d-447f-9d4c-c041794fc8a5  
@fischerm\_student (4/15/2024)

"Got any new posts?"



api.csc346.arizona.edu



PictureGram

https://fischerm.csc346.arizona.edu

THE UNIVERSITY OF ARIZONA Logout: fischerm

# PictureGram

What's Up?

Attach optional image  No file selected.

**POST!**

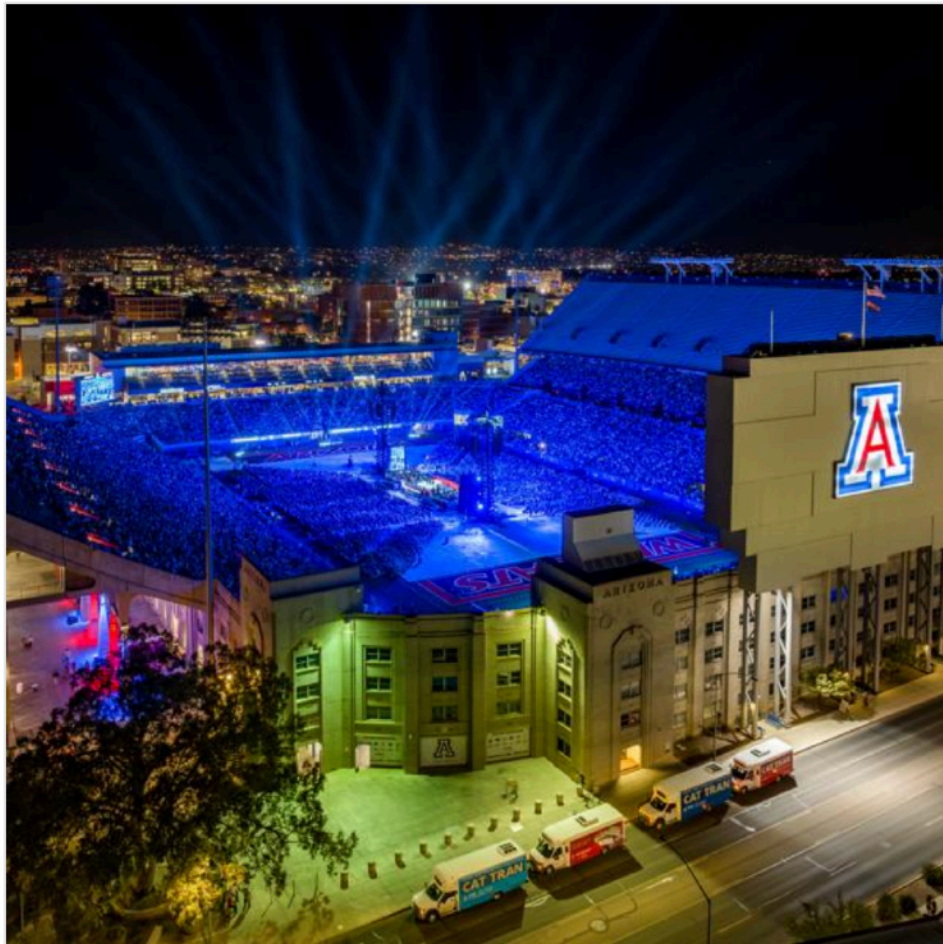
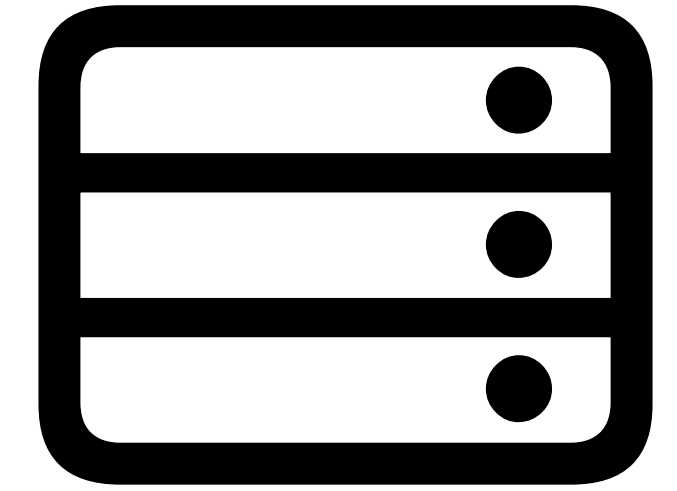


Image Post Test 52945207-df8d-447f-9d4c-c041794fc8a5  
@fischerm\_student (4/15/2024)

← "Hey I do! Here you go"



api.csc346.arizona.edu



Image Post Test  
da834b5e-1039-47c7-9e79-260ef8aafd28  
@fischerm\_student (4/15/2024)

PictureGram

← → ↻ 🔒 📄 🗑️ ⌵

https://fischerm.csc346.arizona.edu

THE UNIVERSITY OF ARIZONA [Logout: fischerm](#)

# PictureGram

What's Up?

Attach optional image  No file selected.

**POST!**


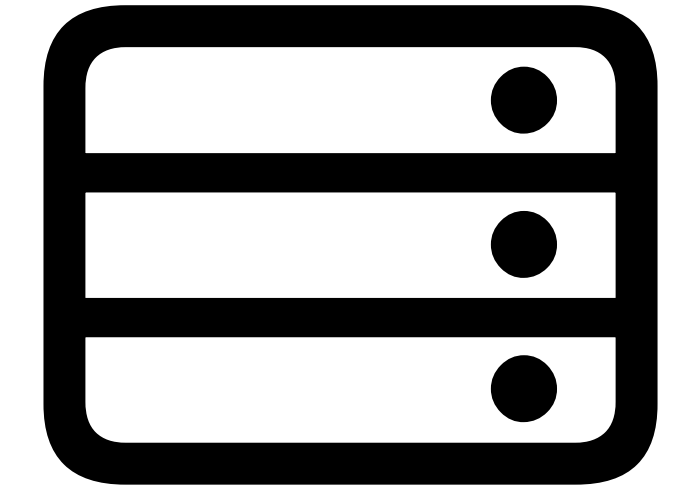
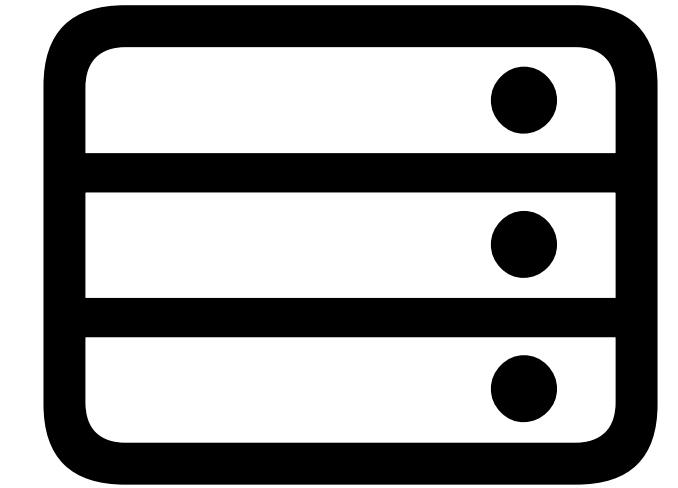


Image Post Test  
da834b5e-1039-47c7-9e79-260ef8aafd28  
@fischerm\_student (4/15/2024)



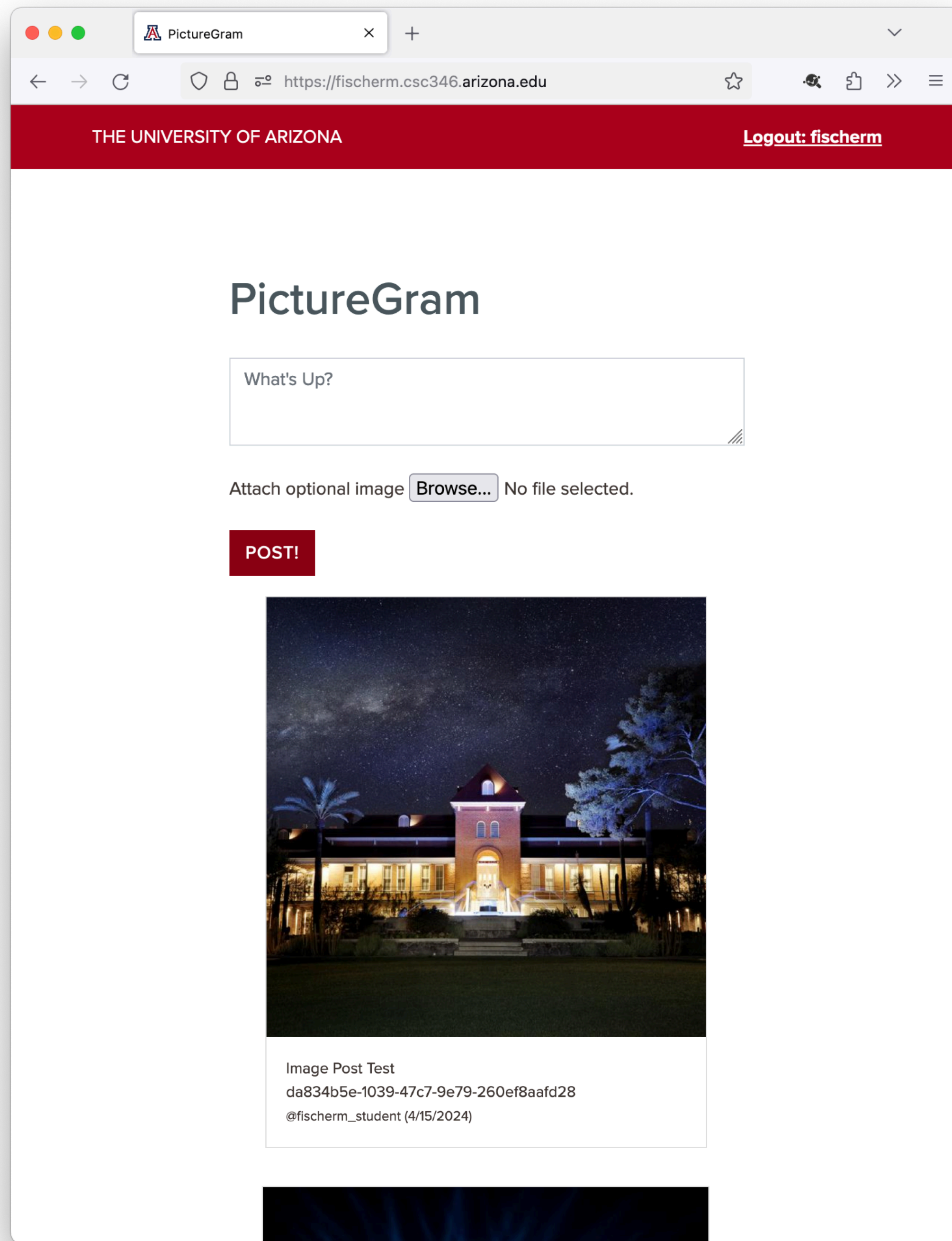
api.csc346.arizona.edu

# Polling



api.csc346.arizona.edu

- This works OK for small numbers of infrequent polling
- What happens when there are many clients?



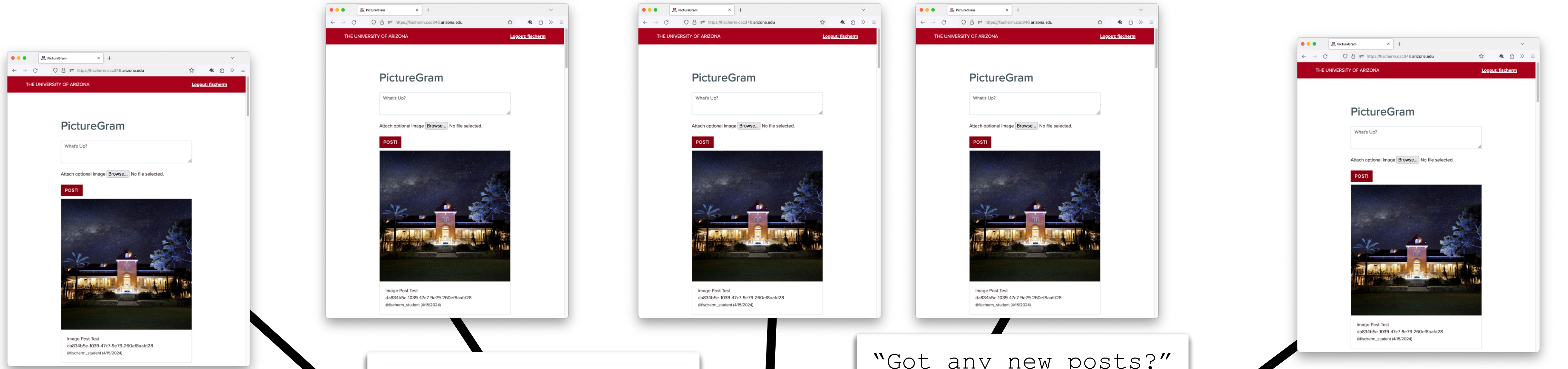
PictureGram

What's Up?

Attach optional image [Browse...](#) No file selected.

**POST!**

Image Post Test  
da834b5e-1039-47c7-9e79-260ef8aafd28  
@fischerm\_student (4/15/2024)



"Got any new posts?"

"Got any new posts?"

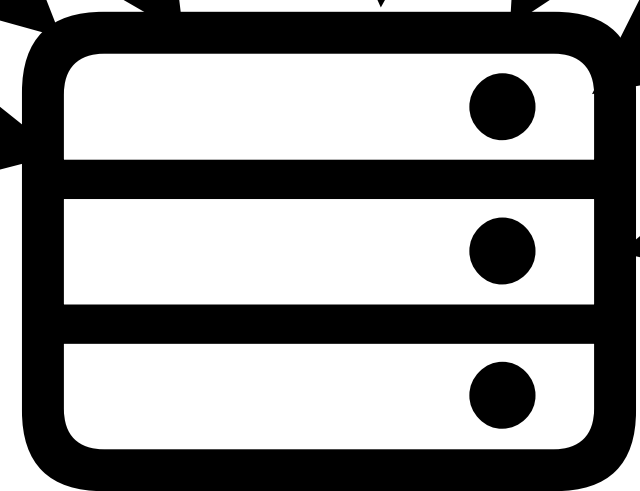
"Got any new posts?"

"Got any new posts?"

"Got any new posts?"

"Got any new posts?"

"Got any new posts?"



api.csc346.arizona.edu

# Polling

## Has its downsides

- Polling requires each client to constantly ask the API for new data
- Short polling intervals can overwhelm the API host with incoming requests for updates
- Long polling intervals can result in significant delay getting new data out to clients
  - The Host may know there's a new message, but it has to wait for a client to ask for it

# WebSockets

## All that is old is new again

- What if we could establish a long-lived network connection between the client and the host?
- This is what WebSockets does

# WebSockets

- So are WebSockets just regular TCP Sockets?
- Spoiler, No
- Conceptually, WebSockets and TCP Sockets have similar goals
  - Support Long-Lived Connections
  - Two-Way Communication
  - Not Request Based
- However they are not related technologically
  - WebSockets are an extension to the HTTP Protocol that runs on top of a TCP Socket

# WebSockets

## Challenges

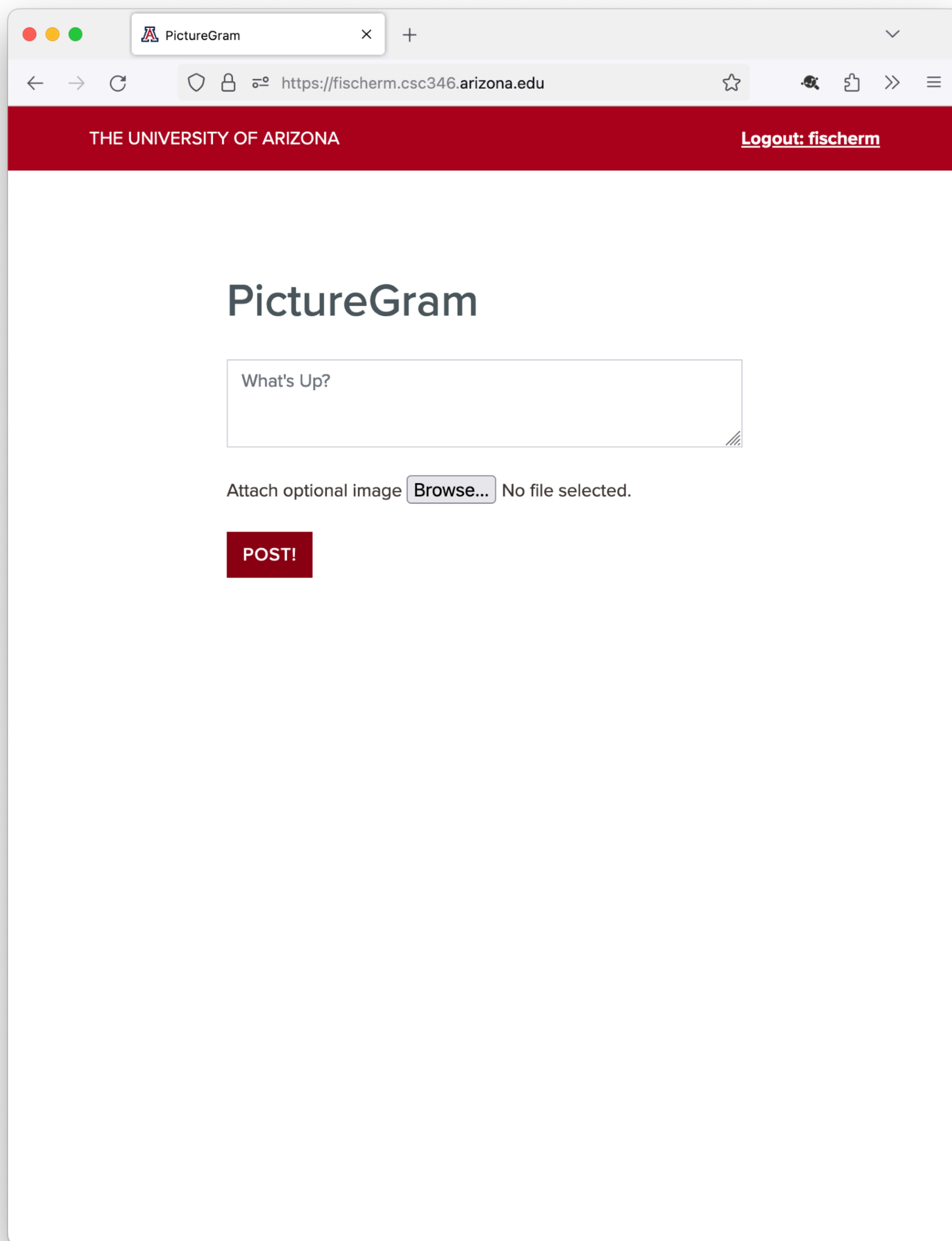
- Low-level socket programming is hard
- Many network situations only permit “web” traffic over ports 80 or 443
- Session and state information about web application logins are already using Cookies, we don't want a new way of handling state
- Security and encryption are already established for HTTPS communications, developing an additional model would be annoying



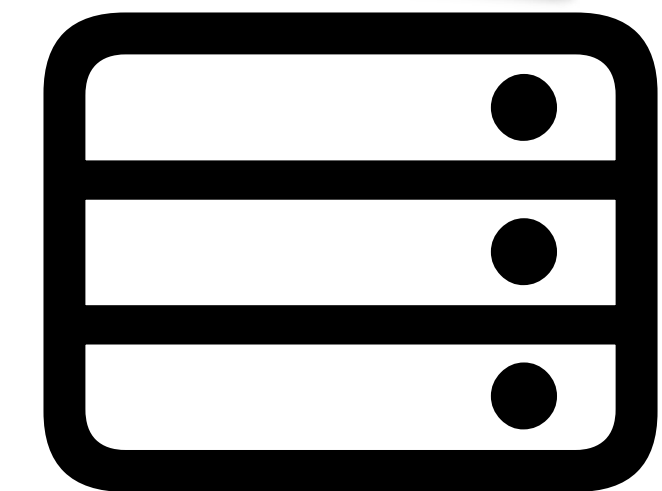
# WebSockets

## Solutions

- Implement a new type of HTTP request
- New request creates a “socket” inside an HTTP request
- Can stay open forever
- Bi-directional comm (not request/response)
- Relatively inexpensive (server memory, network)
- Uses standard HTTP mechanisms for encryption, cookies, etc.
  - Uses standard HTTP/HTTPS ports

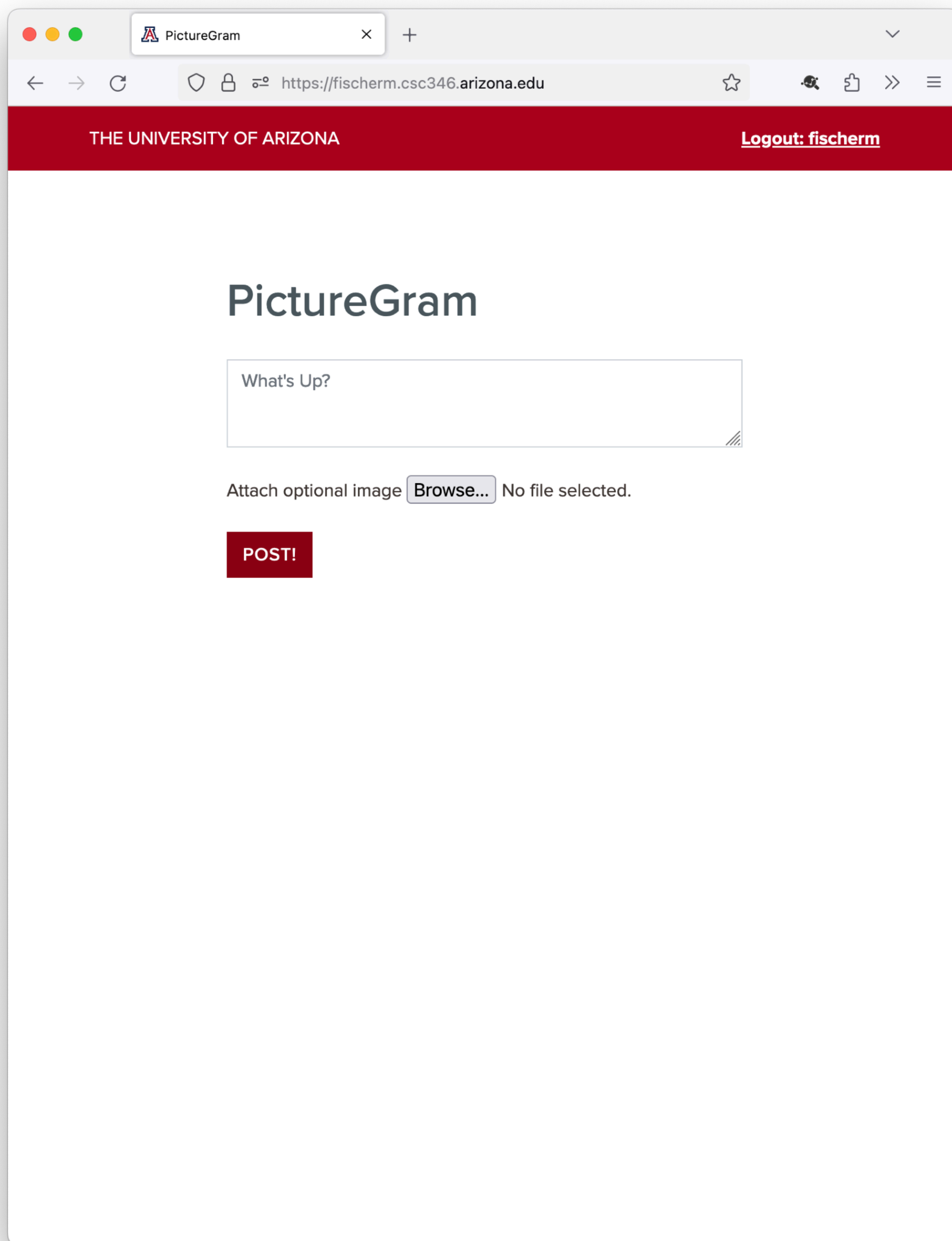


```
GET /chat HTTP/1.1
Host: chat-api.csc346.arizona.edu
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGh1IHhnbXBsZSBub25jZQ==
Sec-WebSocket-Version: 13
```

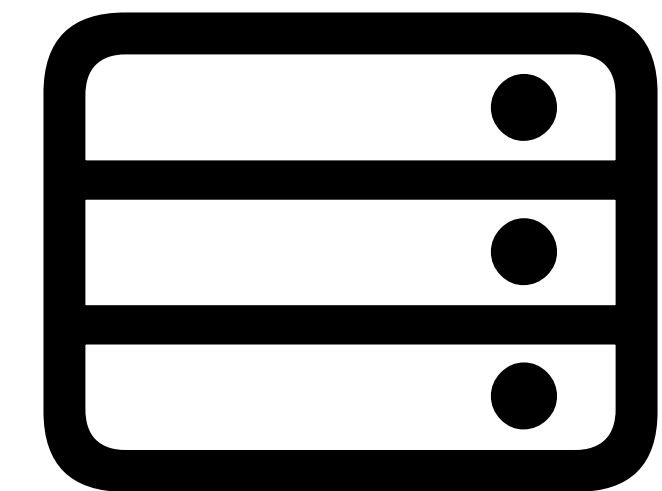


api.csc346.arizona.edu

- A regular HTTP request initiates the WebSocket handshake
- Additional headers are sent, telling the host that the client would like to upgrade this connection to a WebSocket
- Passes along a client key
  - This is just an identifier, not a cryptographic key

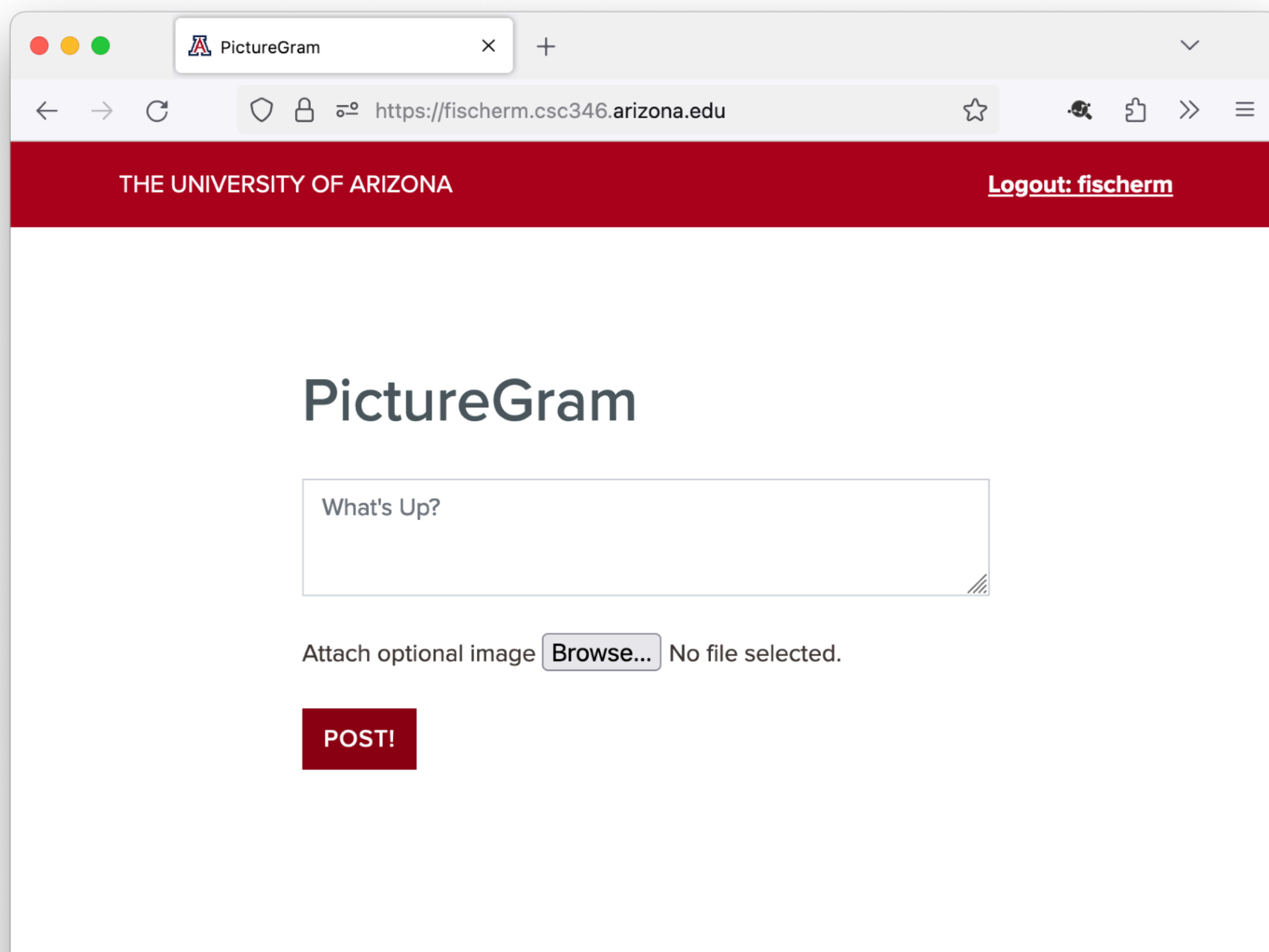


```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
```

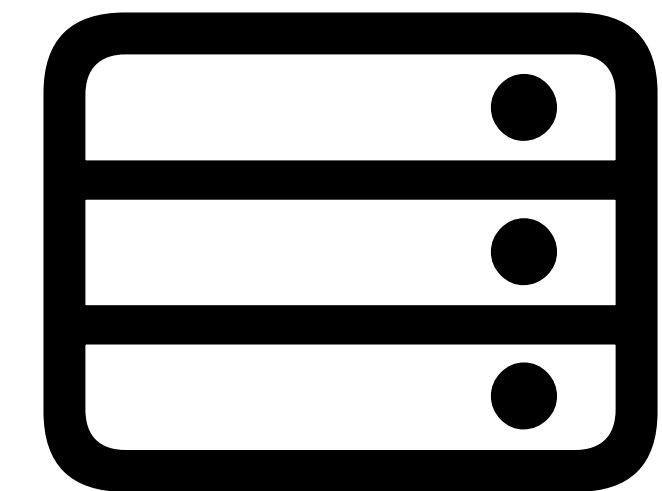


api.csc346.arizona.edu

- If the server supports WebSockets, it responds with the correct headers
- The Sec-WebSocket-Accept response header is calculated in a seemingly overcomplicated way, but exists so that it's obvious to the client whether the server supports WebSockets

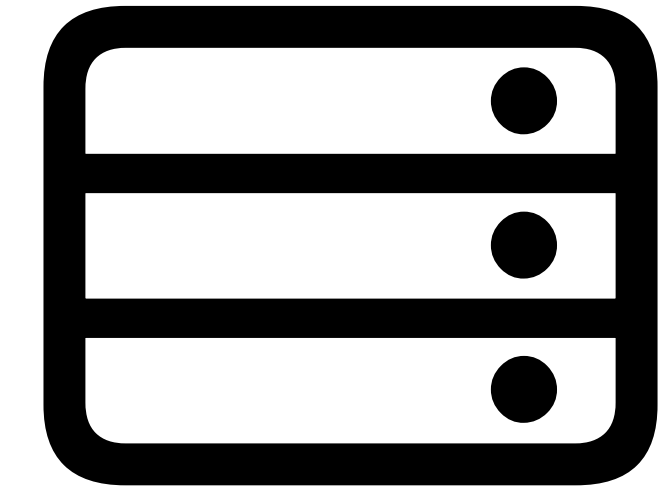
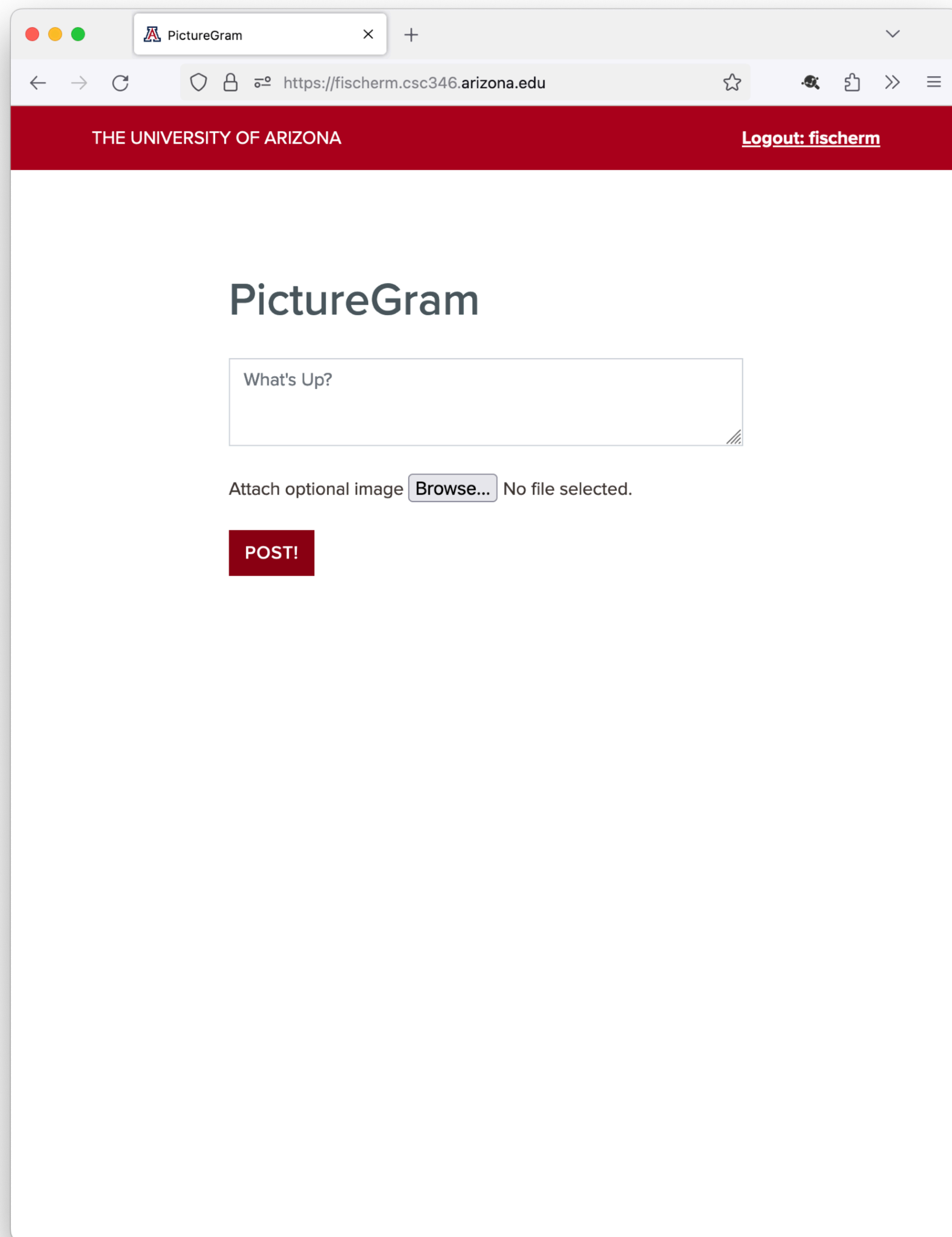


```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+x0o=
```



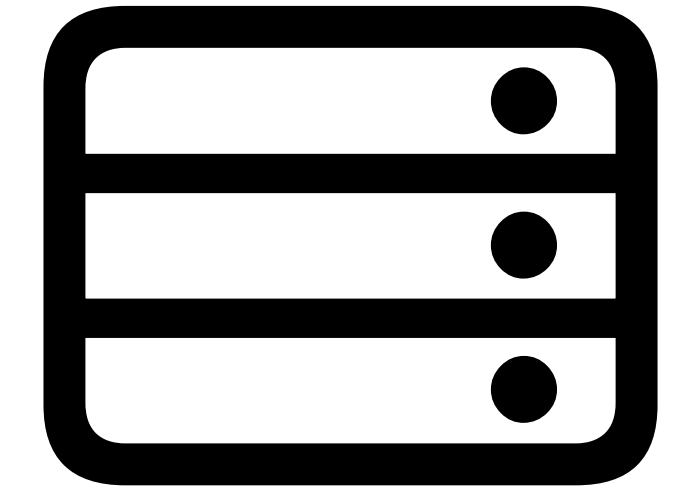
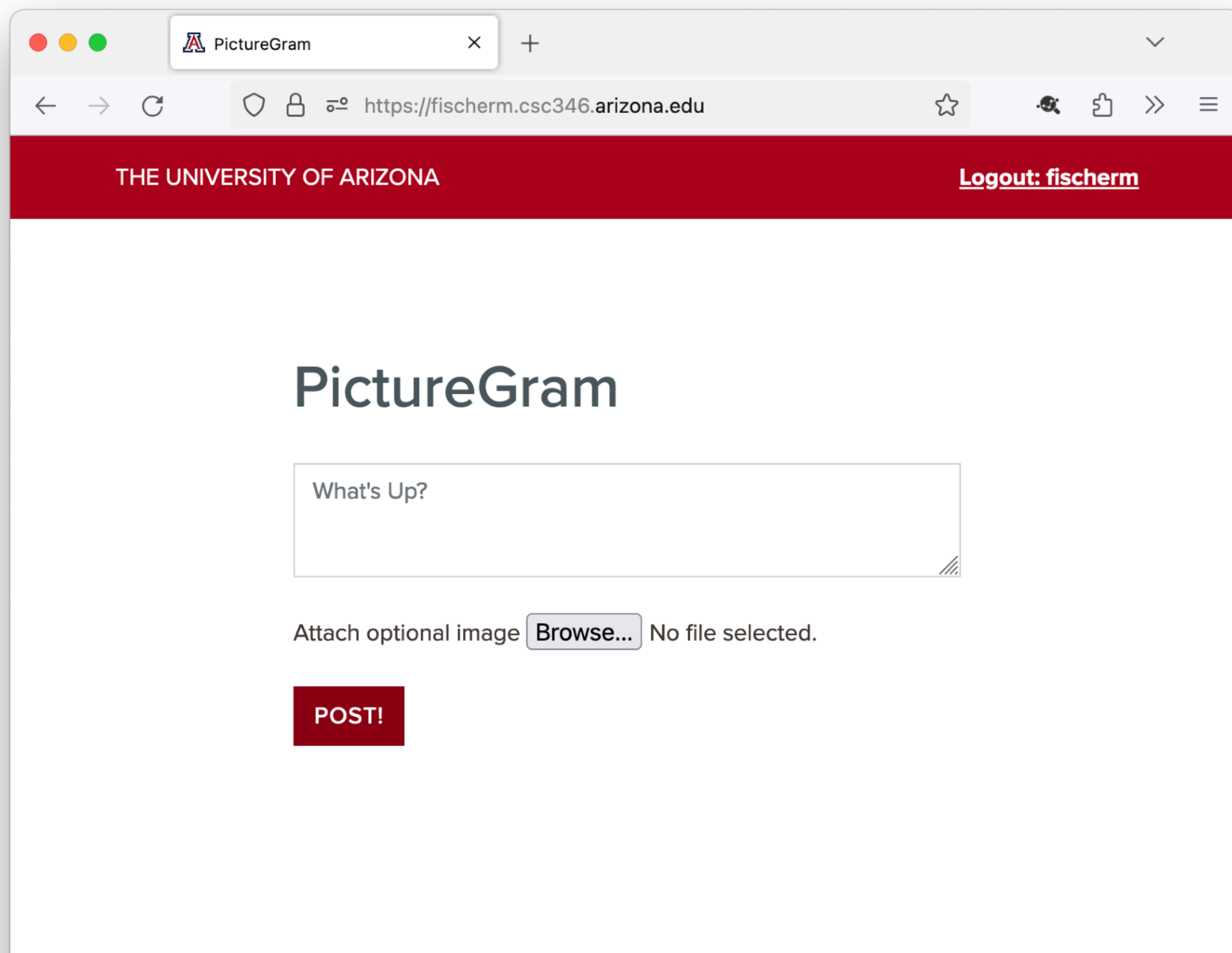
api.csc346.arizona.edu

- The `Sec-WebSocket-Accept` header is important in that the server must derive it from the `Sec-WebSocket-Key` that the client sent to it.
- To get it, concatenate the client's `Sec-WebSocket-Key` and the string `"258EAF5-E914-47DA-95CA-C5AB0DC85B11"` together, take the SHA-1 hash of the result, and return the base64 encoding of that hash.
- You likely will never have to do this unless you want to implement a WebSockets compliant HTTP server. Still useful to know that it's part of the handshake.



api.csc346.arizona.edu

- From that point on, there is a persistent connection between the client and host
- Connection remains open until one side or the other explicitly closes it
- Data can be sent and initiation in either direction by either the client or the host at any time
- Data transfer is now a binary format



api.csc346.arizona.edu

[https://developer.mozilla.org/en-US/docs/Web/API/WebSockets\\_API/Writing\\_WebSocket\\_servers](https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API/Writing_WebSocket_servers)

# WebSockets

## Using with JavaScript

- Handshake details are handled by the browser
- Presents a JavaScript interface to us: **new WebSocket (...)**

```
const apiHost = "wss://chat-api.csc346.arizona.edu/chats"  
const exampleSocket = new WebSocket(apiHost)
```

# WebSockets

## Using with JavaScript

- New Protocol prefix: `ws://` and `wss://`
  - `ws://` kicks off a handshake over `http://`
  - `wss://` kicks off the handshake over `https://`

```
const apiHost = "wss://chat-api.csc346.arizona.edu/chats"  
const exampleSocket = new WebSocket(apiHost)
```



# WebSockets

## Sending messages to the server

```
const apiHost = "wss://chat-api.csc346.arizona.edu/chats"
const exampleSocket = new WebSocket(apiHost)

exampleSocket.send("Message to the server")

data = {
  "type": "newchat",
  "message": "Here's a new chat message",
  "user": "fischer"
}

exampleSocket.send(data)
```

# WebSockets

## Listening for incoming messages

```
const apiHost = "wss://chat-api.csc346.arizona.edu/chats"  
const exampleSocket = new WebSocket(apiHost)  
  
exampleSocket.addEventListener('message', (event) => {  
    console.log('Message from server ', event.data);  
});
```

# WebSockets

**MTG Card Demo**

# **WebSockets**

**From the Server's Side**

# WebSockets

## Server Responsibilities

- The server side has a few duties
  - Accept HTTP Connections and look for the `Upgrade: websocket` and `Connection: Upgrade` headers
  - Calculate the correct `Sec-WebSocket-Accept` response value
  - Keep the WebSocket open
  - Keep track of all open WebSockets, and allow an API to send messages to specific clients

# WebSockets

## AWS API Gateway

- Most Cloud Providers have a managed service for WebSockets
- AWS API Gateway supports multiple API specifications
  - REST
  - Basic HTTP
  - WebSockets

<https://docs.aws.amazon.com/apigateway/latest/developerguide/apigateway-websocket-api-overview.html>

# WebSockets

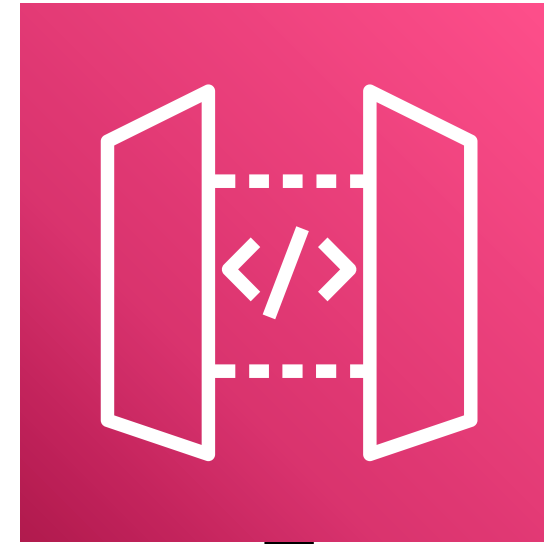
## AWS API Gateway

- API Gateway takes care of all the protocol level work associated with WebSockets
  - Accepts and Upgrades WebSocket connections
  - Calculates `Sec-WebSocket-Accept` responses
  - Keeps Socket connections open
  - Assigns Connection IDs to each open WebSocket and tracks activity
  - Sends activity to a backend processor, ie Lambda

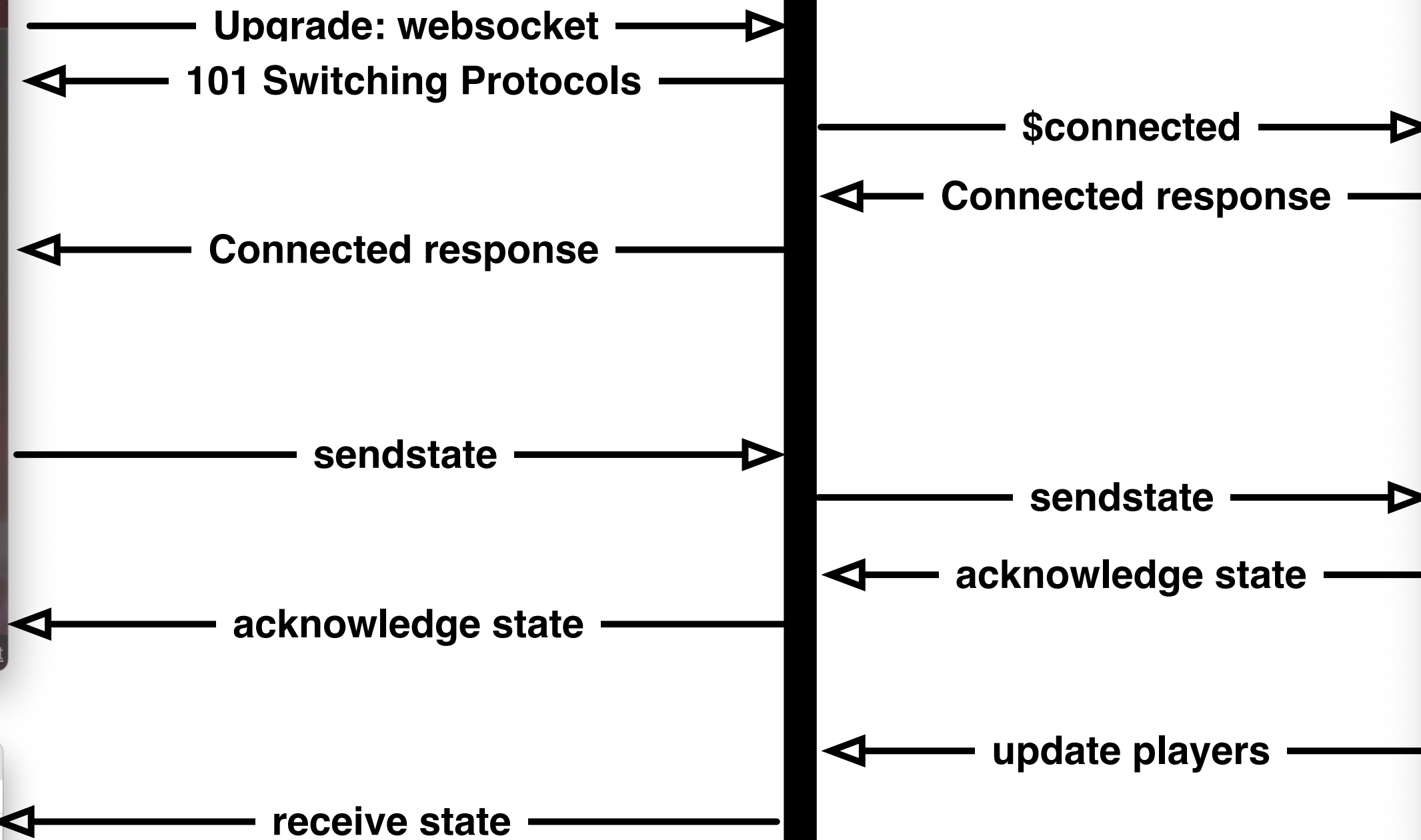
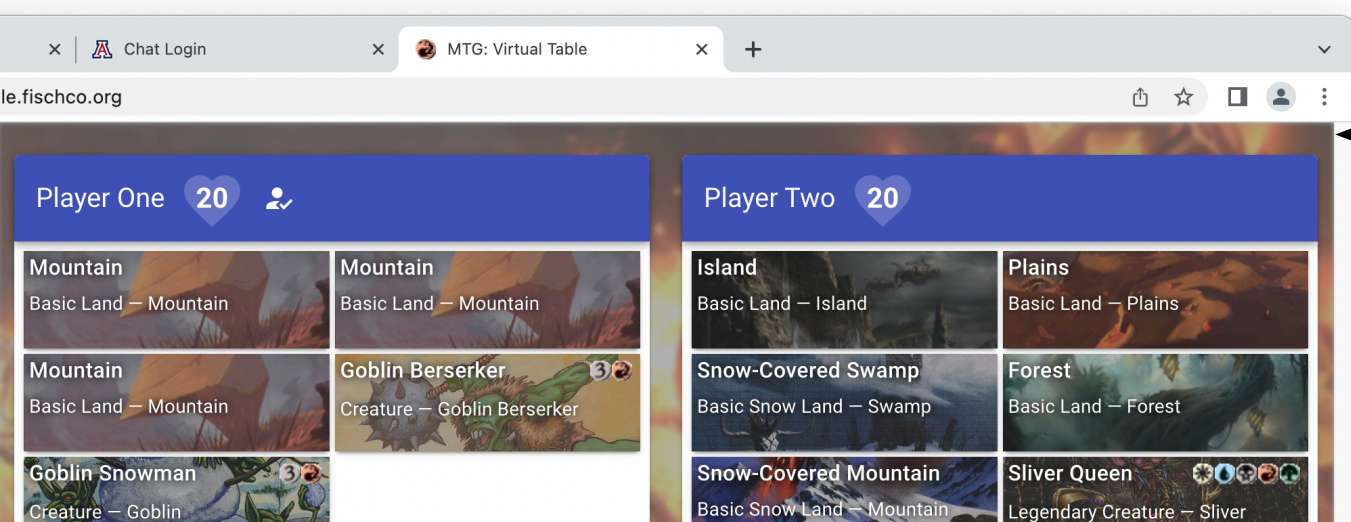
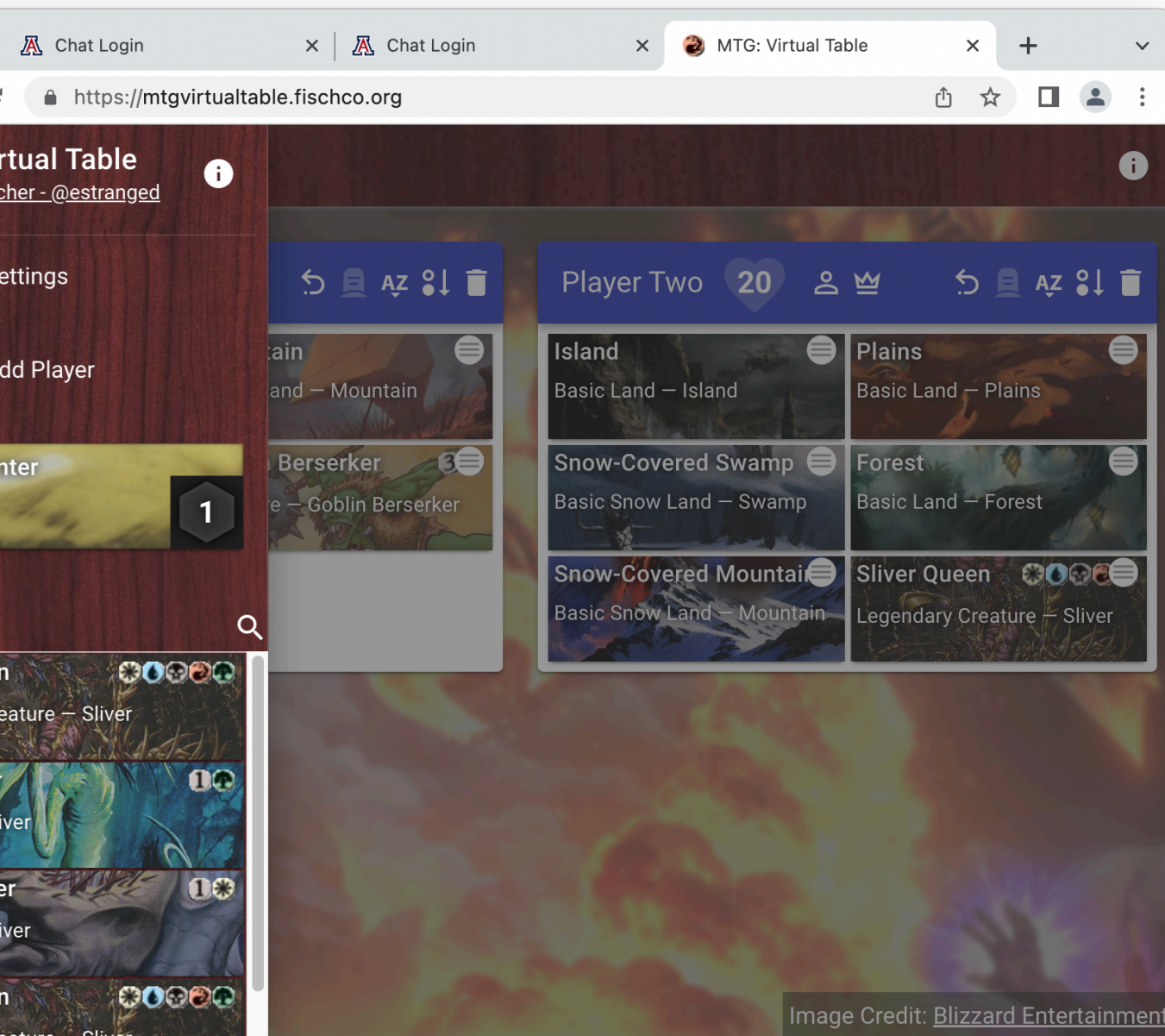
# WebSockets

## AWS API Gateway

API Gateway



Lambda



```
connection.py x
serverless > connection.py >
239 def handler(event, context):
240
241     logger.info(json.dumps(event))
242
243     action = event.get('event', None)
244     request_context = event.get('requestContext', None)
245     if not request_context:
246         return _format_response(f'Missing requestContext', 400)
247
248     route_key = request_context.get('routeKey', None)
249     if not route_key:
250         return _format_response(f'Missing routeKey', 400)
251
252     connection_id = request_context.get('connectionId', None)
253     if not connection_id:
254         return _format_response(f'Missing connectionId', 400)
255
256     domain_name = request_context.get('domainName', None)
257     if not domain_name:
258         return _format_response(f'Missing domainName', 400)
259
260     body = event.get('body', None)
261     if body:
262         body = json.loads(body)
263
264     if route_key == "$connect":
265         return _format_response(f'Connected', 200)
266
267     if route_key == "$disconnect":
268         return _format_response(f'Bye', 200)
269
270     if route_key == "$default":
271         resp = {}
272
273         if not body:
274             return _format_response(f'Missing body', 400)
275
276         action = body.get('action', None)
277
278     if not action:
```



# WebSockets

## Server Code Demo