WebDev 4

CS571: Building User Interfaces

Cole Nelson & Yaxin Hu

Today's Warmup

- Clone today's code to your machine.
  - Run the command `npm install` inside of the `starter` and `solution` folders.
- Import the Postman collections.
  - Replace "Enter Your Badger ID" with your Badger ID

Learning Objectives

1. Be able to persist data using complex APIs that go beyond simple `GET` operations.
2. Understand the difference between *controlled* and *uncontrolled* components.
3. Be able to implement *uncontrolled* components via the `useRef` hook.
4. Be able to handle credentials and other sensitive information via cookies 🍪

Working with Complex APIs

Beyond GETting data...

Scenario

You are building a database system. What operations should you allow a developer to perform?

Scenario

You are building a database system. What operations should you allow a developer to perform?

1. **C**reate data.
2. **R**ead data.
3. **U**pdate data.
4. **D**elete data.

CRUD Operations via HTTP

CRUD Operation	HTTP Operation
Create	POST
Read	GET
Update	PUT
Delete	DELETE

HTTP Recap

Data is transmitted by requests and responses that allow us to create (POST), read (GET), update (PUT), and delete (DELETE) data!

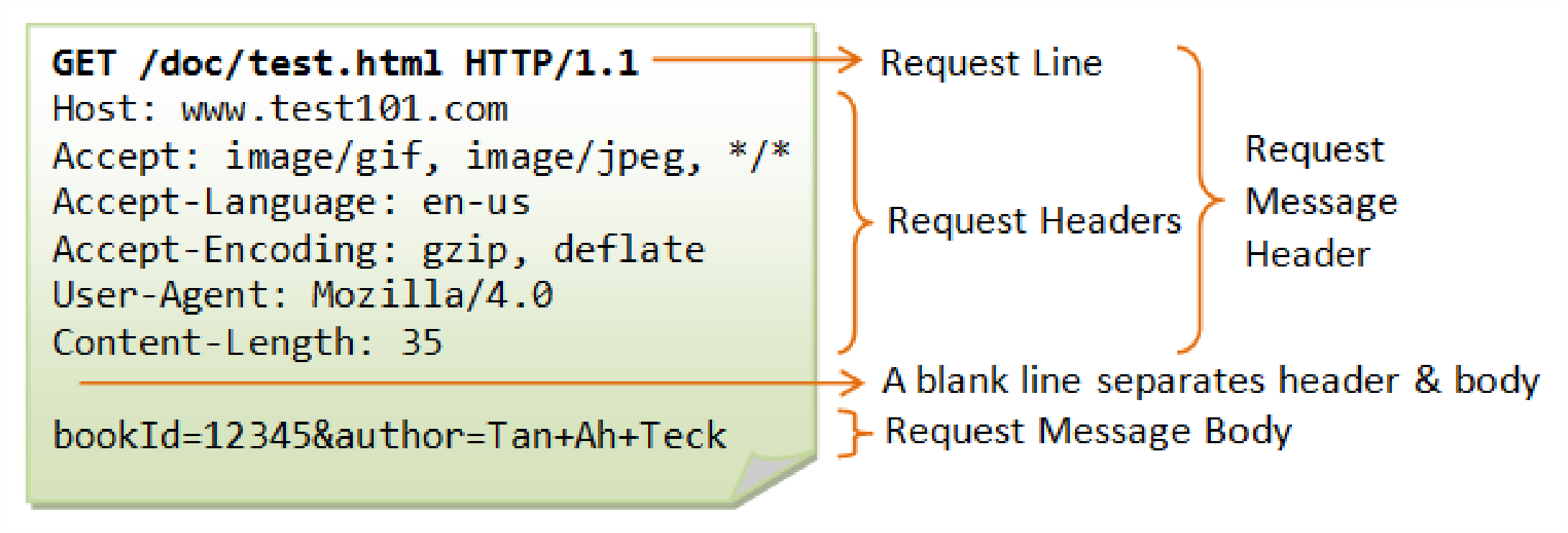


Image Source

```
HTTP/1.1 200 OK
Date: Sun, 08 Feb xxxx 01:11:12 GMT
Server: Apache/1.3.29 (Win32)
Last-Modified: Sat, 07 Feb xxxx
ETag: "0-23-4024c3a5"
Accept-Ranges: bytes
Content-Length: 35
Connection: close
Content-Type: text/html

<h1>My Home page</h1>
```

Status Line

Response Headers

Response Message Header

A blank line separates header & body

Response Message Body

Image Source

HTTP Recap

An HTTP request may have *path* and *query* parameters

```
https://www.example.com/packers/AaronRodgers/stats?all=true&since=2010
```

Here, `AaronRodgers` is a *path* parameter while `all` and `since` are *query* parameters.

Usage depends on the API documentation.

HTTP Recap

HTTP requests (specifically `PUT` and `POST`) may also have a *request body*. This is located below the headers.

```
{
    "title": "Hello world!",
    "content": "abc123",
}
```

For a JSON body, we need an additional HTTP header.

`Content-Type: application/json`

HTTP Status Codes

HTTP Code	Response Type
100s	Informational
200s	Successful
300s	Redirection
400s	Client Error
500s	Server Error

HTTP Specific Status Codes

HTTP Code	Response
200	OK
304	Not Modified
400	Bad Request
401	Unauthorized
404	Not Found
409	Conflict
413	Request Entity Too Large
500	Internal Server Error

What is this "HTTPS" I hear about?

The "secure" version of HTTP.

Same thing as the HTTP protocol with end-to-end encryption. We use HTTPS for our API.

```
{
  "username": "joe_schmoe21",
  "password": "mysecret123!"
}
```

This is only secure because of *HTTP**S***!

ICE WebDev 4 API

Use Postman to explore the API and `POST` a comment. Valid username and password combinations are...

- **bucky** - badger
- **pete608** - gopioneers!
- **gophy77** - boooo

Don't use *real* pins, and don't "hack" either -- *every request* is tied back to your Badger ID! 🦡

Fetching w/ `POST`, `PUT`, and `DELETE`

`fetch` can do a lot more than just retrieving data.

- specify request method
- specify request headers
- specify request body
- inspect response status
- inspect response headers
- inspect response body
- ...and so much more!

```
fetch("https://example.com/create-content", {
    method: "POST",
    headers: {
        "Content-Type": "application/json" // must include this header
    },
    body: JSON.stringify({ // must stringify
        content: "Hello World!"
    })
}).then(res => {
    if (res.status === 409) {
        alert("This content already exists!")
    }
    return res.json();
}).then(json => {
    if (json.msg) {
        alert(json.msg)
    }
});
```

Your Turn!

When the user clicks "Login", `POST` their login information to the API (e.g. "create" a login request). `console.log` the resulting status code.

These are typically documented, like in HW6!

Note: `Options`

The browser will *pre-flight* (i.e. ask permission first) any *complex* request like `POST`; that's why you may see 2 requests in your network log!

This is because of Cross-Origin Resource Sharing (CORS), which can get quite complex!

Detour: Uncontrolled Components

Sometimes, we care about listening to form changes.

Other times, **we don't**. Notice that we didn't care about the changes to *username* and *password* until the user presses the "Login" button.

In cases like these, we can use *uncontrolled components*.

Handling Text Input

We can get user input using the HTML `input` tag or the React-Bootstrap `Form.Control` component.

We can get user input...

- in a *controlled* way using its `value` and tracking `onChange` events
- in an *uncontrolled* manner using `useRef`.

`useRef` Hook

Usually used to "reference" an input element.

```
const inputVal = useRef();
return (
  <div>
    <label htmlFor="myInput">Type something here!</label>
    <input id="myInput" ref={inputVal}></input>
  </div>
);
```

The value of a ref can be retrieved via its `current` property, e.g. `inputVal.current.value`

Controlled vs Uncontrolled Components

`useRef` is used to create a reference to an *uncontrolled* input component.

This is opposed to *controlling* an input component via its `value` and `onChange` properties.

Example of an uncontrolled input component.

Example of a controlled input component.

Input Best Practices

In either case, each `input` should have an `id` associated with the `htmlFor` of a `label`.

If you are using react-bootstrap components, be sure each `Form.Control` has an `id` associated with the `htmlFor` of a `Form.Label`.

Read more here.

Your Turn!

Turn your *controlled* components into *uncontrolled* components.

Secrets! Secrets! 🤫

Handling user credentials with cookies! 🍪

Secrets! Secrets!

Is there anything **special** about credentialed requests? As developers, we do **not** like to handle credentials. We delegate this to the browser with **cookies** 🍪

Cookies hold a small amount of data. When set as `HTTPOnly`, they hold that data securely. For us, that data is a JWT exchanged for the user's credentials.

You offer...

```
{
  "username": "bucky",
  "password": "badger"
}
```

... and you receive ...


```
eyJhbGciOiJIUzI1NiIsInR5cCI6
IkpXVCJ9eyJ1c2VybmFtZSI6ImJ1
Y2t5IiwiaWF0IjoxNzA5NTg3MjQz
LCJleHAiOjE3MDk1OTA4NDN9.-ap
P_1KeJXZzN3DqgVz0m1YqZScAJoM
cg-JueyrxjTI
```

JSON Web Tokens (JWTs)

A cryptographically-signed access token issued by a server for a set period of time (typically short). Used in lieu of the username and password directly.

Why? We don't want to be caught holding the user's password (especially in XSS attacks).

Encoded PASTE A TOKEN HERE

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6ImJ1Y2t5IiwiaWF0IjoxNzA5NTg3MjQzLCJleHAiOjE3MDk1OTA4NDN9.-apP_1KeJXZzN3DqgVz0m1YqZScAJoMcg-JueyrxjTI

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYLOAD: DATA

```
{
  "username": "bucky",
  "iat": 1709587243,
  "exp": 1709590843
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
) ☐ secret base64 encoded
```

Cookies 🍪

A cookie, specifically an `HTTP-Only` cookie, is just a container for this sensitive data. **It is managed by the browser**, your JavaScript code *cannot* access it!

Secrets! Secrets!

For a successful request with credentials, the server will send back a `Set-Cookie` response header to your browser. This includes your newly-issued JWT!

You should be able to see this in your browser by visiting F12 > Application > Cookies.

We are only concerned about `icewebdev4_auth` and `badgerchat_auth` cookies! Ignore the others.

Name	Headers	Payload	Preview	Response	Initiator	Timing	Cookies
login							
login							

Content-Length:	73
Content-Type:	application/json; charset=utf-8
Date:	Mon, 04 Mar 2024 22:02:24 GMT
Etag:	W/"49-q4JDPViDFv1yeIRpsDyDmlerXww"
Keep-Alive:	timeout=5, max=100
Server:	Apache/2.4.54 (Ubuntu)
Set-Cookie:	icewebdev4_auth=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2Vybm FtZSI6ImJ1Y2t5IiwiaWF0IjoxNzA5NTg5NzQ0LCJleHAiOjE3MDk1OTMzN DR9.PjWCZrx62_eU06ATIk_9hPZ95JlyXkMuK8GzaFSXYh8; Max-Age=3600; Domain=cs571.org; Path=/; Expires=Mon, 04 Mar 2024 23:02:24 GMT; HttpOnly; Secure; Partitioned; SameSite=None

Elements | Console | Sources | Network | Performance | Memory | Application | Security | 5 | 5

Application
- Manifest
- Service w
- Storage

Storage

Filter | Only show cookies with an issue

Name	Value	Domain	Path	Expires / Max-Age	Size	H..	S...	S..	Partition Key
[redacted]									
[redacted]									
icewebdev4_auth	eyJhbGciOiJI...	.cs571.org	/	2024-03-04T23:02:2...	168	✓	✓	N...	http://localho
[redacted]									

The Catch?

For requests that *either* set *or* use cookies, we must include credentials.

```
fetch("https://example.com/create-content", {
  method: "POST",
  credentials: "include", // <---- must do this for cookies!
  // ...
```

This needs to be done for any requests related to logging in, logging out, or creating a post!

```javascript
fetch("https://example.com/create-content", {
    method: "POST",
    credentials: "include", // add this to requests related to cookies!
    headers: {
        "Content-Type": "application/json"
    },
    body: JSON.stringify({
        content: "Hello World!"
    })
}).then(res => {
    if (res.status === 409) {
        alert("This content already exists!")
    }
    return res.json();
}).then(json => {
    if (json.msg) {
        alert(json.msg)
    }
});
```

Secrets! Secrets!

What's the benefit? The browser handles all things authentication! 🥳🎉

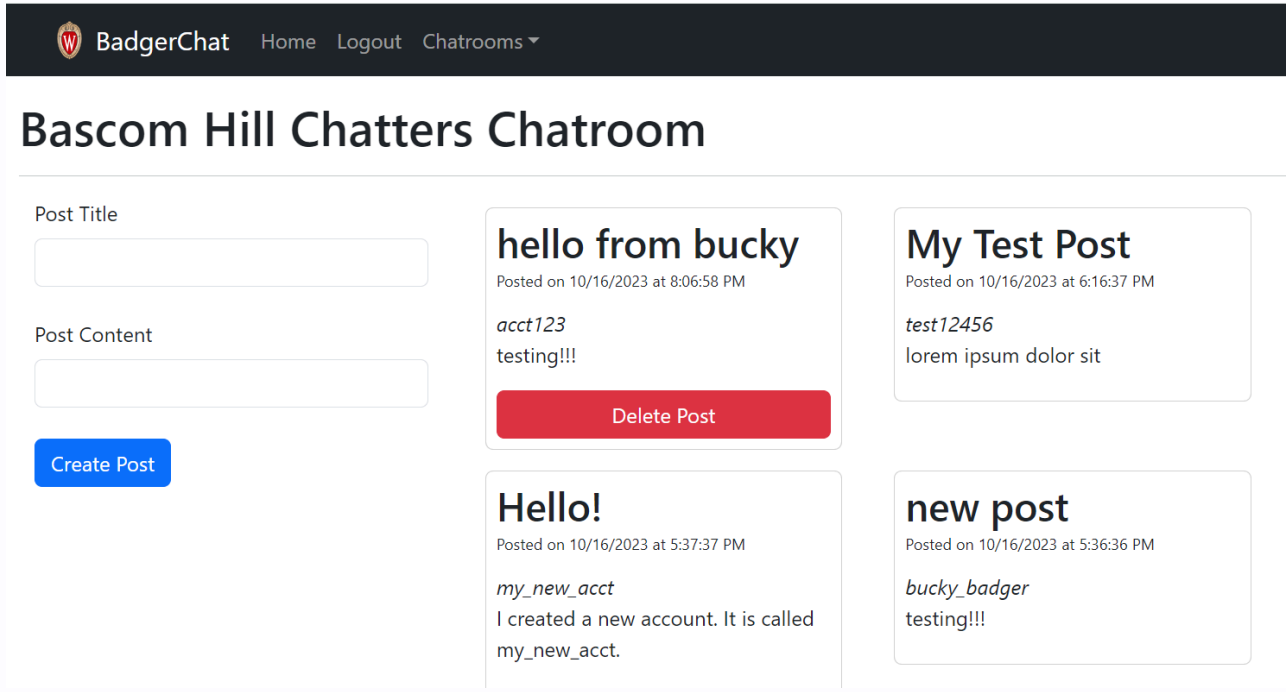
Your Turn!

Finish the implementation of BadgerChat Mini!

HW6 Demo

BadgerChat! 🦡🗣️

Go over the HW6 API documentation.



Questions?