

React 3

CS571: Building User Interfaces

Cole Nelson

Announcements

- Midterm on 10/25 in class (details on next slide).
- Please contact me by 10/18 if you will not be able to make the exam (Canvas/HonorLock alternative).
- HW5 extended to Friday, 10/21 with no penalty (final late day of 10/28). Extra credit available.
- Thanks for your patience! Your feedback is greatly appreciated.

Midterm Details

- Tuesday, 10/25 @ 5:45p-8:15p (2.5 hours)
- 32 questions worth 15 points covering Weeks 1-6.
 - 20 Multiple Choice (0.25 pts each)
 - 10 Short Answer (0.5 pts each)
 - 2 Long Response (2.5 pts each)
 - Half programming, half design
- Closed book, closed internet, closed interpreter.
- You may bring one double-sided standard size notesheet. Handwritten or digital is OK.

What will we learn today?

- `useState` gotcha!
- `useRef` and Controlled vs Uncontrolled Components
- Working with "complex" APIs
- How to keep a secret? 🤔

useState gotcha!

This should have been mentioned earlier...

useState Gotcha

Did you know that the mutator function returned by `useState` happens asynchronously?

```
setBadgermon(["Pikachu", "Squirtle", "Ivysaur"]);  
console.log(badgermon);
```

In the above example, the `console.log` will show an empty array because `setBadgermon` has not been completed yet (it's async)!

useState Gotcha

If we want to `console.log` the *updated* state, we should add a `useEffect` with `badgermon` as its dependency!

```
useEffect(() => {  
  console.log(badgermon)  
}, [badgermon])
```

React Hooks

Final hook(s) that we will be learning!

useReducer Hook

Honorable mention...

```
const [state, dispatch] = useReducer(reducer, initialArg, init);
```

A redux-like way to manage state; change the state based on the `dispatch`.

We won't use this, but you may see it used by others!

[See example.](#)

useRef Hook

Usually used to "reference" an input element.

```
const inputVal = useRef();
return (
  <div>
    <label for="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 normally used to create a reference for an *uncontrolled* input component.

However, we can *control* an input component via its `value` and `onChange` properties.

Example of an uncontrolled input component.

Example of a controlled input component.

Controlled vs Uncontrolled: Pros & Cons

React generally recommends controlled components.

Controlled components can cause many re-renders, however uncontrolled components give you no control over the `onChange` property.

We'll practice using both.

Note: Performance matters with React Native!

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**ppdate data.
4. **D**delete data.

CRUD Operations via HTTP

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

HTTP Intro

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

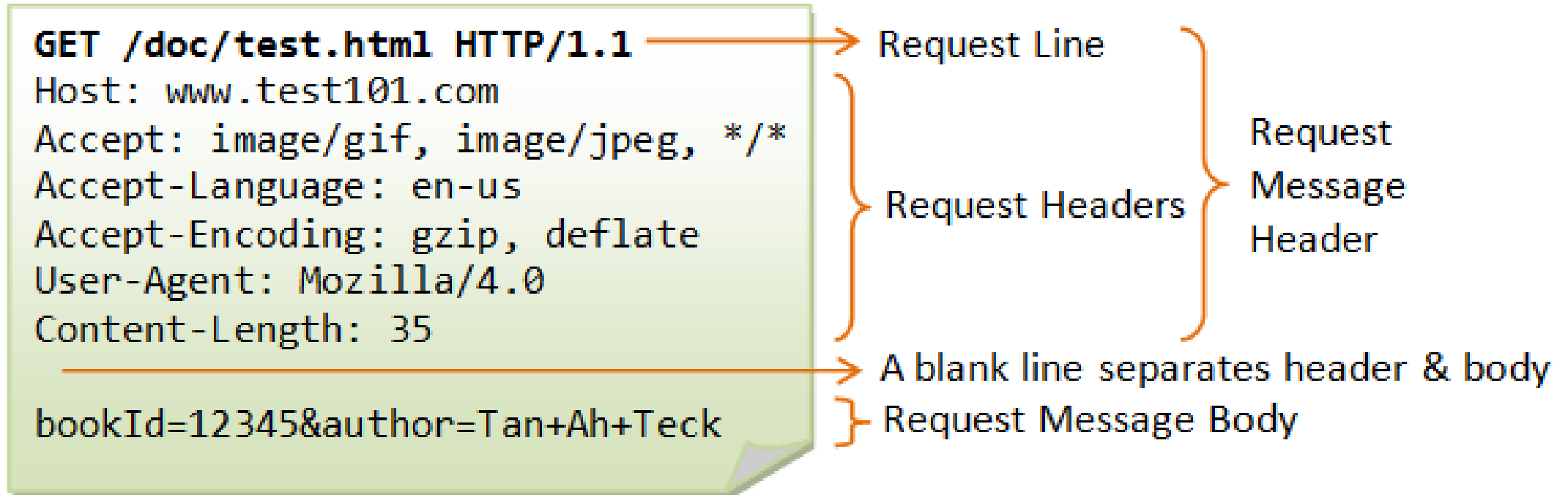


Image Source

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 Intro

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.

HTTP Intro

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

```
{  
  "username": "testuser1234",  
  "password": "pass123",  
  "refCode": "bid_999988887777"  
}
```

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

Reading the Docs

Go over the [HW5 API documentation](#).

Secrets! Secrets!

Is there anything **special** about requests for logging in? Not really!

It varies from system to system, but typically we POST a username and password in the request body to an authentication endpoint.

e.g.

POST /register or

POST /login

Secrets! Secrets!

These endpoints return a session for the user, in HW5 this is in the form of a [JavaScript Web Token \(JWT\)](#).

This is a temporary, all-access token for authenticating with the API. It is used in lieu of a username and password. Why might we do this?

Secrets! Secrets!

We typically store these in an http-only cookie.

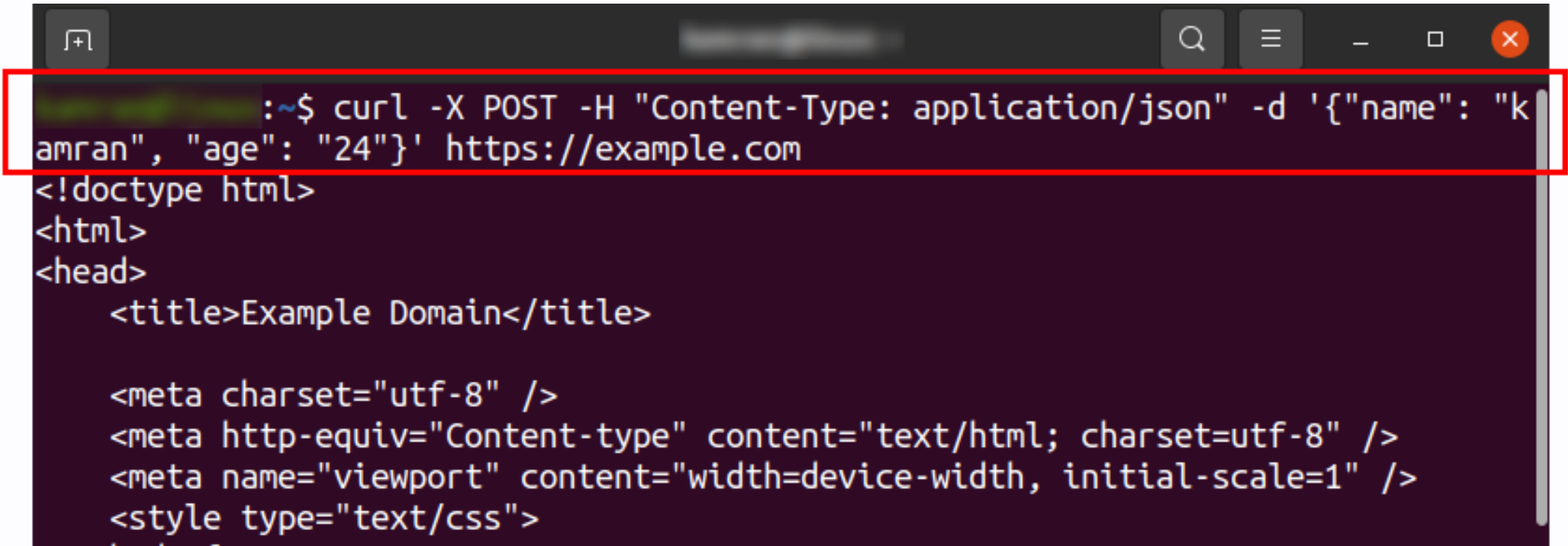
However, for HW5, we store it in a context provider and pass it as an Authorization header, e.g.

```
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwidXNlcm5
```

How can we do this?

cURL, Postman, and fetch!

cURL

A terminal window with a dark background. The command prompt shows a user at a shell running a cURL command. The command is highlighted with a red rectangle. The output is an HTML document structure, also partially highlighted with a red rectangle. The terminal window has standard macOS window controls (red, yellow, green buttons) and a search icon in the top right.

```
:~$ curl -X POST -H "Content-Type: application/json" -d '{"name": "kamran", "age": "24"}' https://example.com
<!doctype html>
<html>
<head>
  <title>Example Domain</title>

  <meta charset="utf-8" />
  <meta http-equiv="Content-type" content="text/html; charset=utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  <style type="text/css">
```

Image Source

Postman

www.postman.com

Image Source

**HOW I THINK
OF POSTMAN**

**HOW
POSTMAN THINKS
OF POSTMAN**



Postman Demo

Using the HW5 API

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"
  },
  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)
  }
});
```

`fetch` Demo

Same thing as in Postman, but with `fetch`

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.

What did we learn today?

- `useState` gotcha!
- `useRef` and Controlled vs Uncontrolled Components
- Working with "complex" APIs
- How to keep a secret? 🤫

On to Interaction Design!

