

SWE 432 -Web Application Development

Spring 2023



George Mason
University

Dr. Kevin Moran

Week 7:
Security +
Templates,
Databinding, & HTML





Administrivia

- HW Assignment 2 - Due Thursday at Midnight
- Midterm Exam - In class on Tuesday, March 21st
- Review Video Posted by Tuesday next week



Midterm Exam

- 3 Parts, In-class exam, closed book, 200 points total
 - **Part 1:** Multiple Choice Questions
 - **Part 2:** Short Answer
 - Either provide program output, or answer in a few short sentences
 - **Part 3:** Multi-Part Code Question (*implementing a simple microservice*)
- Covers material from weeks 1-7, from both lectures and readings
- You will have the **entire** class period to complete



Class Overview

- Part 1 - *Security*: What is it, authentication, and important types of attacks
- Part 2 - - *Intro to Frontend*:
HTML, Templates, and Databinding

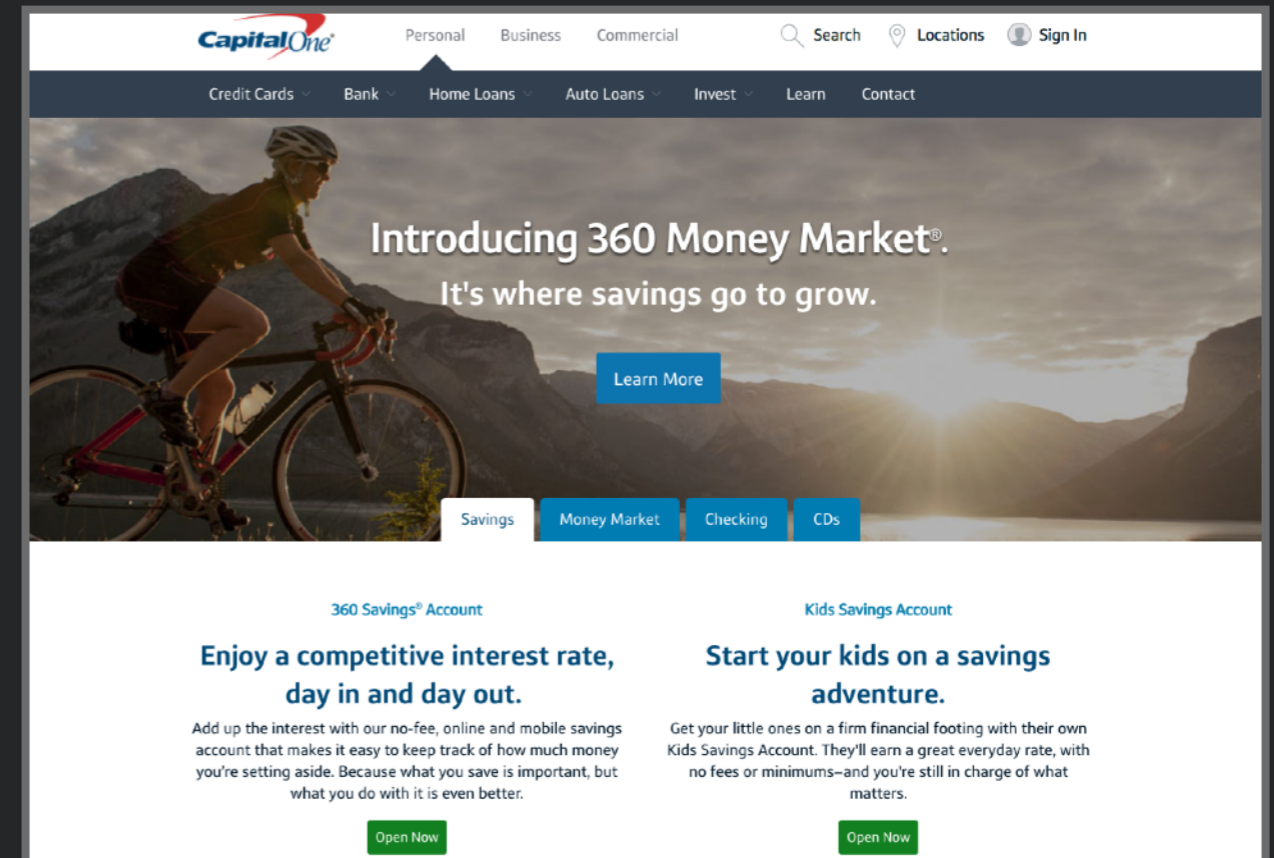
Web Security



Security



- Why is it important?
 - Users' data is on the web
 - Blog comments, FB, Email, Banking, ...
- Can others steal it?
 - or who already has access?
- Can others impersonate the user?
 - e.g., post on FB on the user's behalf





Security Requirements for Web Apps

1. Authentication

- Verify the *identity* of the parties involved
- Who is it?

2. Authorization

- Grant *access* to resources only to allowed users
- Are you allowed?

3. Confidentiality

- Ensure that *information* is given only to authenticated parties
- Can you see it?

4. Integrity

- Ensure that information is *not changed* or tampered with
- Can you change it?

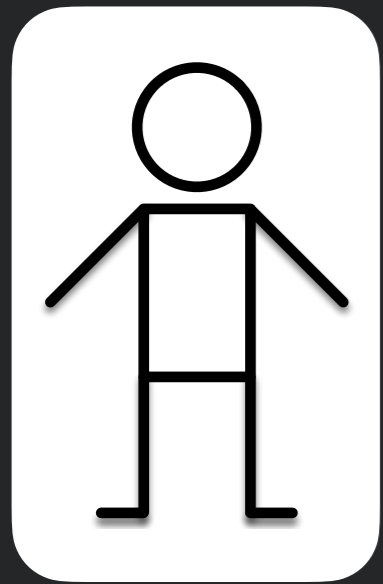


Threat Models

- What is being defended?
 - What resources are important to defend?
 - What malicious actors exist and what attacks might they employ?

- Who do we trust?
 - What entities or parts of system can be considered secure and trusted
 - Have to trust **something!**

Web Threat Models: Big Picture



client page
(the “user”)

HTTP Request

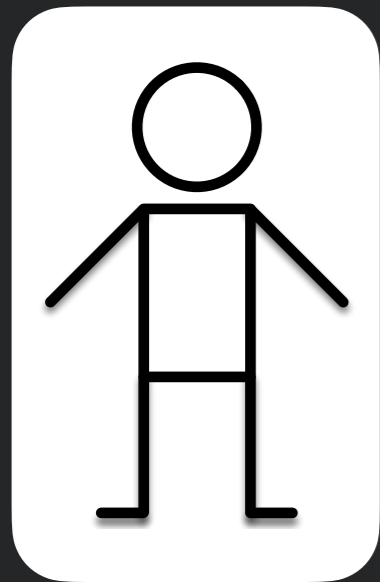


HTTP Response



server

Web Threat Models: Big Picture



client page
(the “user”)

HTTP Request



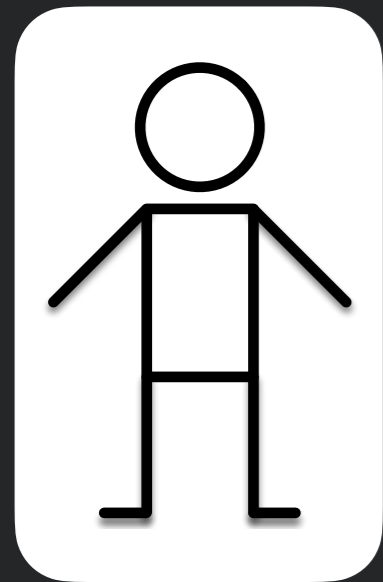
HTTP Response



server

Do I trust that this request *really* came from the user?

Web Threat Models: Big Picture



client page
(the “user”)

HTTP Request



HTTP Response

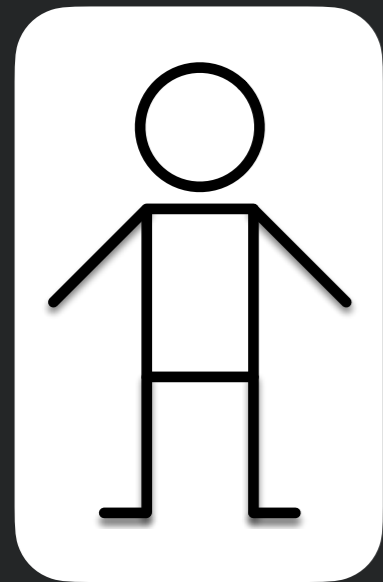


server

**Do I trust that this response
really came from the server?**

**Do I trust that this request *really*
came from the user?**

Web Threat Models: Big Picture



client page
(the “user”)

HTTP Request



HTTP Response

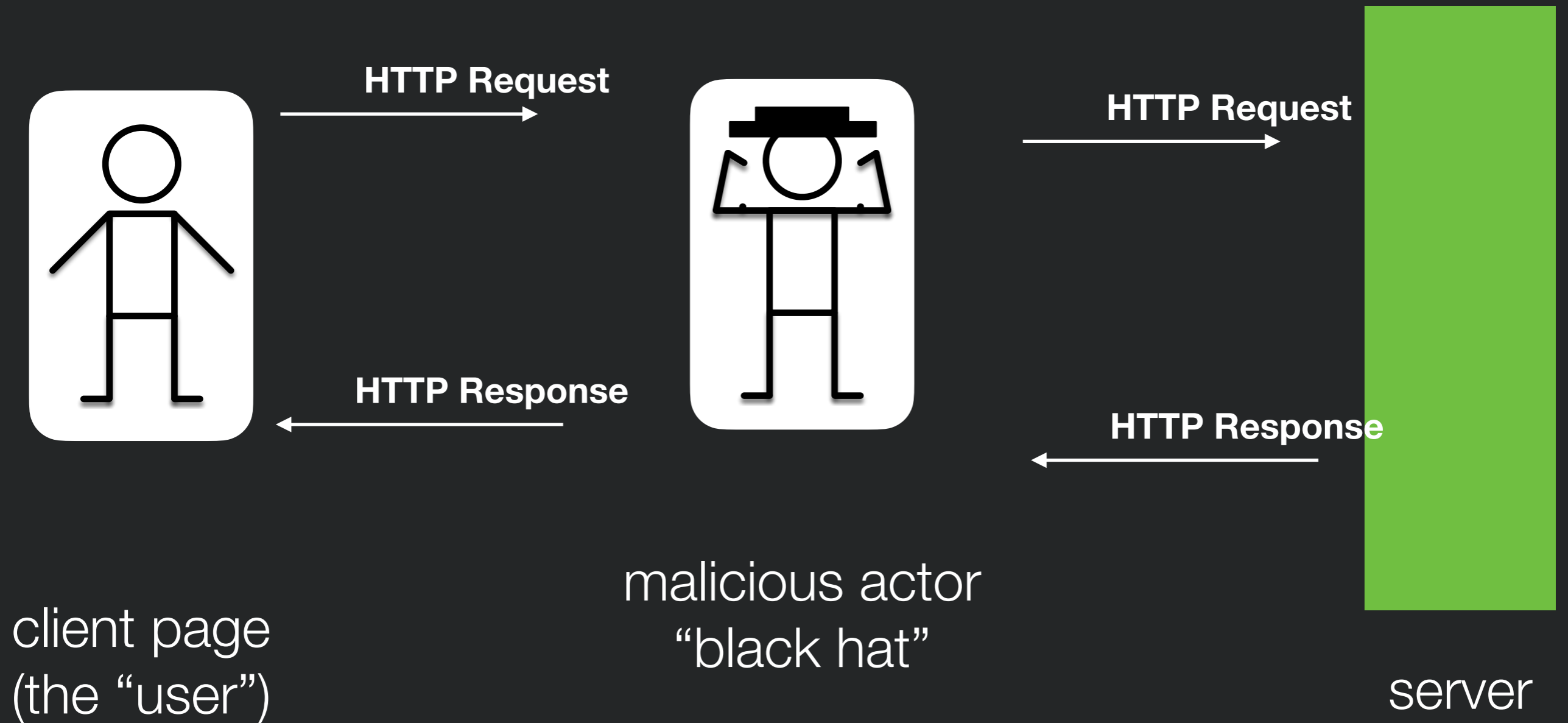


server

**Do I trust that this response
really came from the server?**

**Do I trust that this request *really*
came from the user?**

Web Threat Models: Big Picture

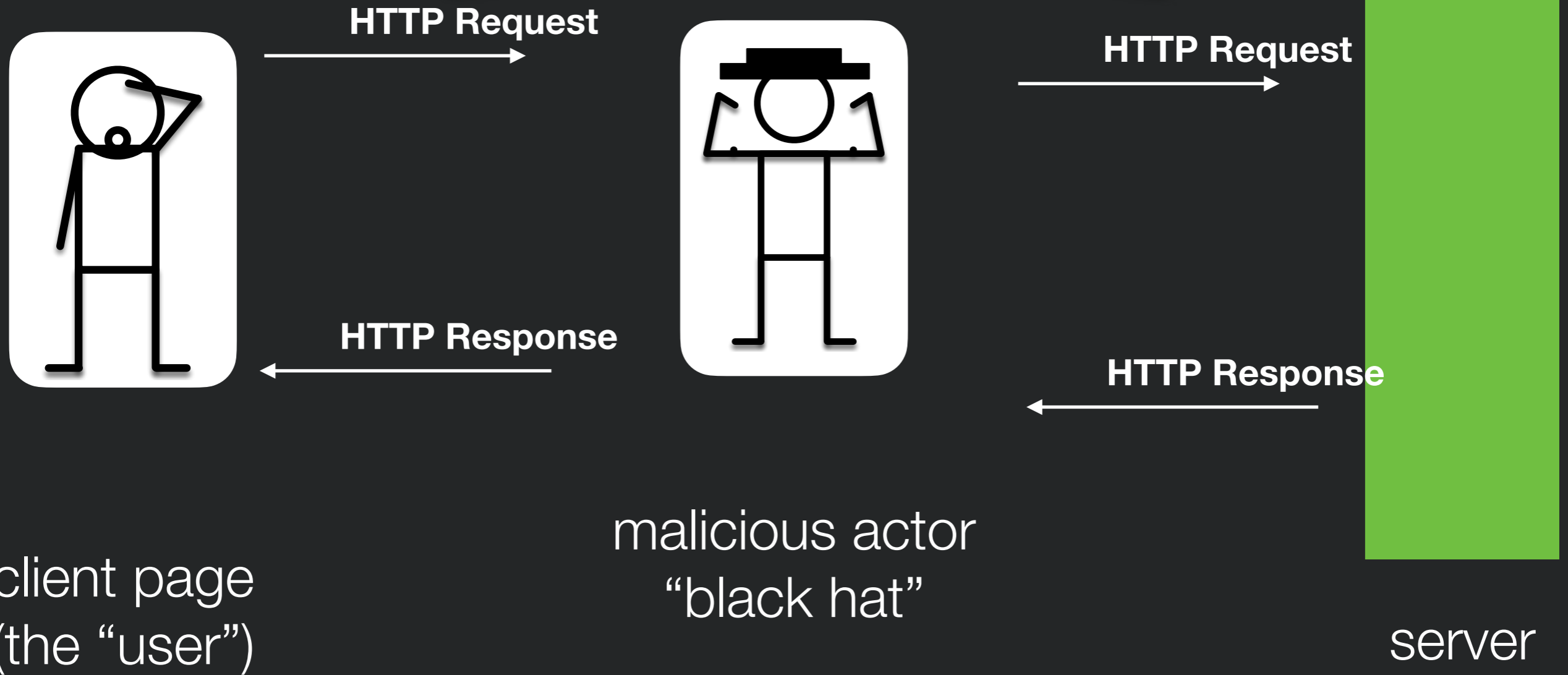


Do I trust that this response really came from the server?

Do I trust that this request *really* came from the user?

Web Threat Mitigation Strategy

Might be “man in the middle” that intercepts requests and impersonates user or server.



Do I trust that this response *really* came from the server?

Do I trust that this request *really* came from the user?



Security Requirements for Web Apps

1. Authentication

- Verify the *identity* of the parties involved
- Threat: Impersonation. A person pretends to be someone they are not.

2. Authorization

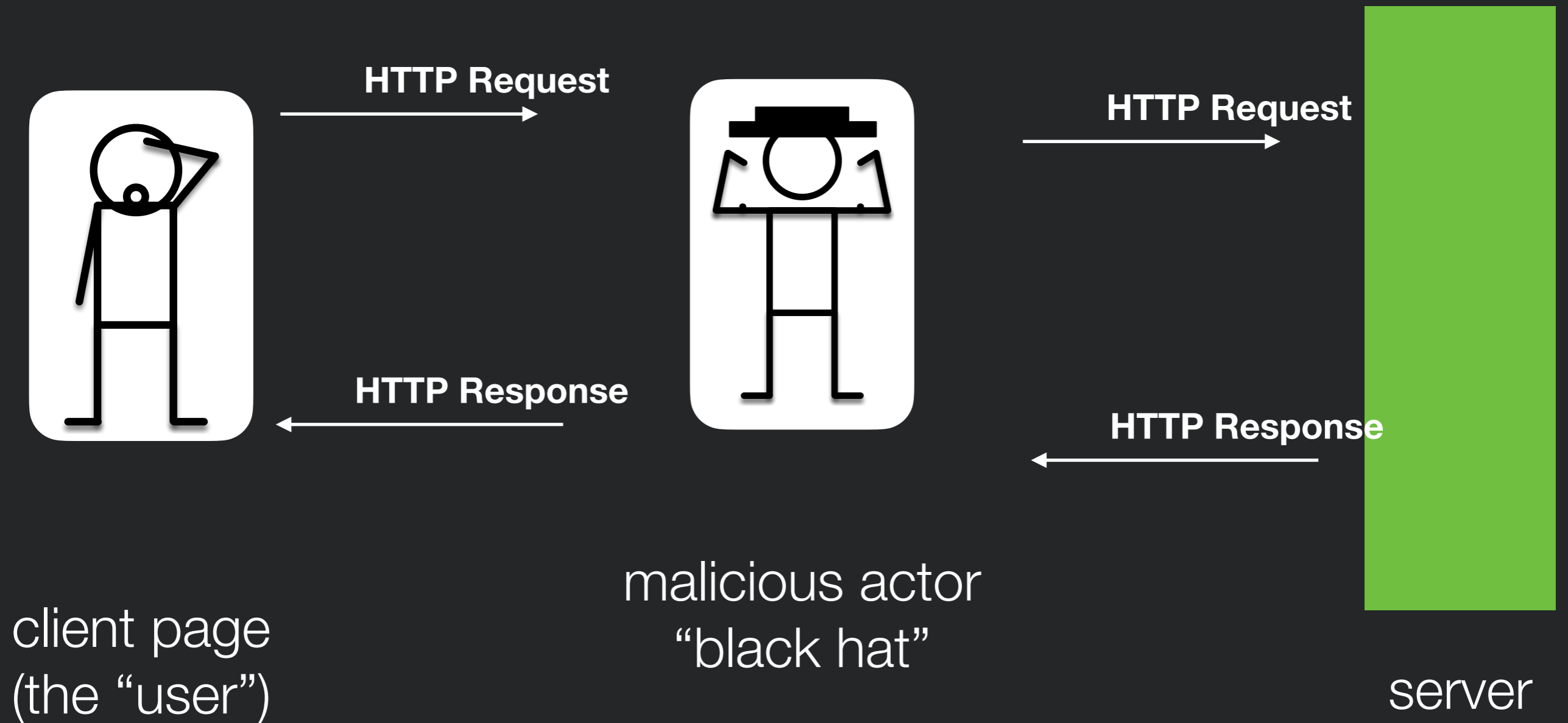
3. Confidentiality

- Ensure that *information* is given only to authenticated parties
- Threat: Eavesdropping. Information leaks to someone that should not have it.

4. Integrity

- Ensure that information is *not changed* or tampered with
- Threat: *Tampering*.

Web Threat Models: Big Picture



What if malicious actor impersonates server?



Man in the Middle

- Requests to server intercepted by man in the middle
 - Requests forwarded
 - But... response containing code edited, inserting malicious code
- Or could
 - Intercept and steal sensitive user data



HTTPS: HTTP over SSL

- Establishes secure connection from client to server
 - Uses SSL to encrypt traffic
- Ensures that others can't impersonate server by establishing certificate authorities that vouch for server.
- Server trusts an HTTPS connection iff
 - The user trusts that the browser software correctly implements HTTPS with correctly pre-installed certificate authorities.
 - The user trusts the certificate authority to vouch only for legitimate websites.
 - The website provides a valid certificate, which means it was signed by a trusted authority.
 - The certificate correctly identifies the website (e.g., certificate received for "https://example.com" is for "example.com" and not other entity).



Using HTTPS

- If using HTTPS, important that all scripts are loaded through HTTPS
 - If mixed script from untrusted source served through HTTP, attacker could still modify this script, defeating benefits of HTTPS
- Example attack:
 - Banking website loads Bootstrap through HTTP rather than HTTPS
 - Attacker intercepts request for Bootstrap script, replaces with malicious script that steals user data or executes malicious action



Authentication

- How can we know the identify of the parties involved
- Want to customize experience based on identity
 - But need to determine identity first!
- Options
 - Ask user to create a new username and password
 - Lots of work to manage (password resets, storing passwords securely, ...)
 - Hard to get right (#2 on the OWASP Top 10 Vulnerability List)
 - User does not really want another password...
 - Use an authentication provider to authenticate user
 - Google, FB, Twitter, Github, ...



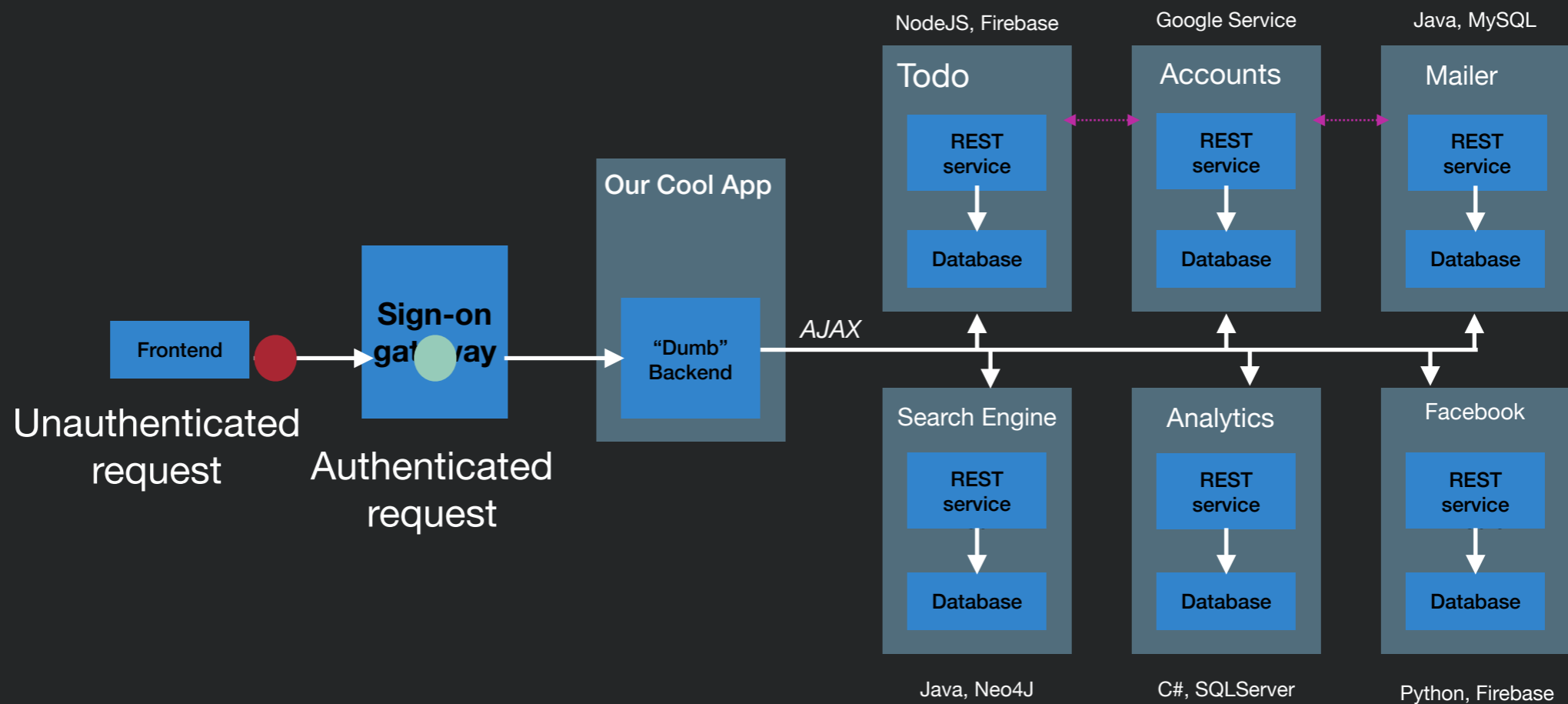
Authentication Provider

- Creates and tracks the identity of the user
- Instead of signing in directly to website, user signs in to authentication provider
 - Authentication provider issues token that uniquely proves identity of user



Sign-on Gateway

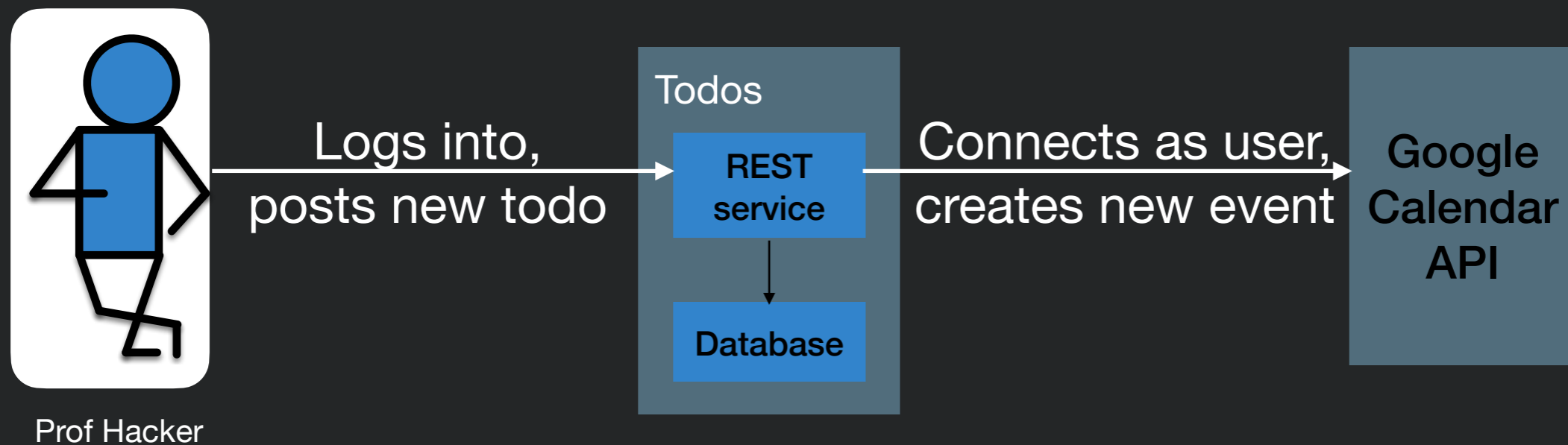
- Can place some magic “sign-on gateway” before our app - whether it's got multiple services or just one



Bigger Picture - Authentication with Multiple Service Providers



- Let's consider updating a Todos app so that it can automatically put calendar events on a Google Calendar





How does Todos tell Google that it's posting something for Prof Hacker?
Should Prof Hacker tell the Todos app her Google password?





We've Got Something for That...

Google user@gmail.com ▾

▾ Example App would like to:

 View your basic profile info ⓘ

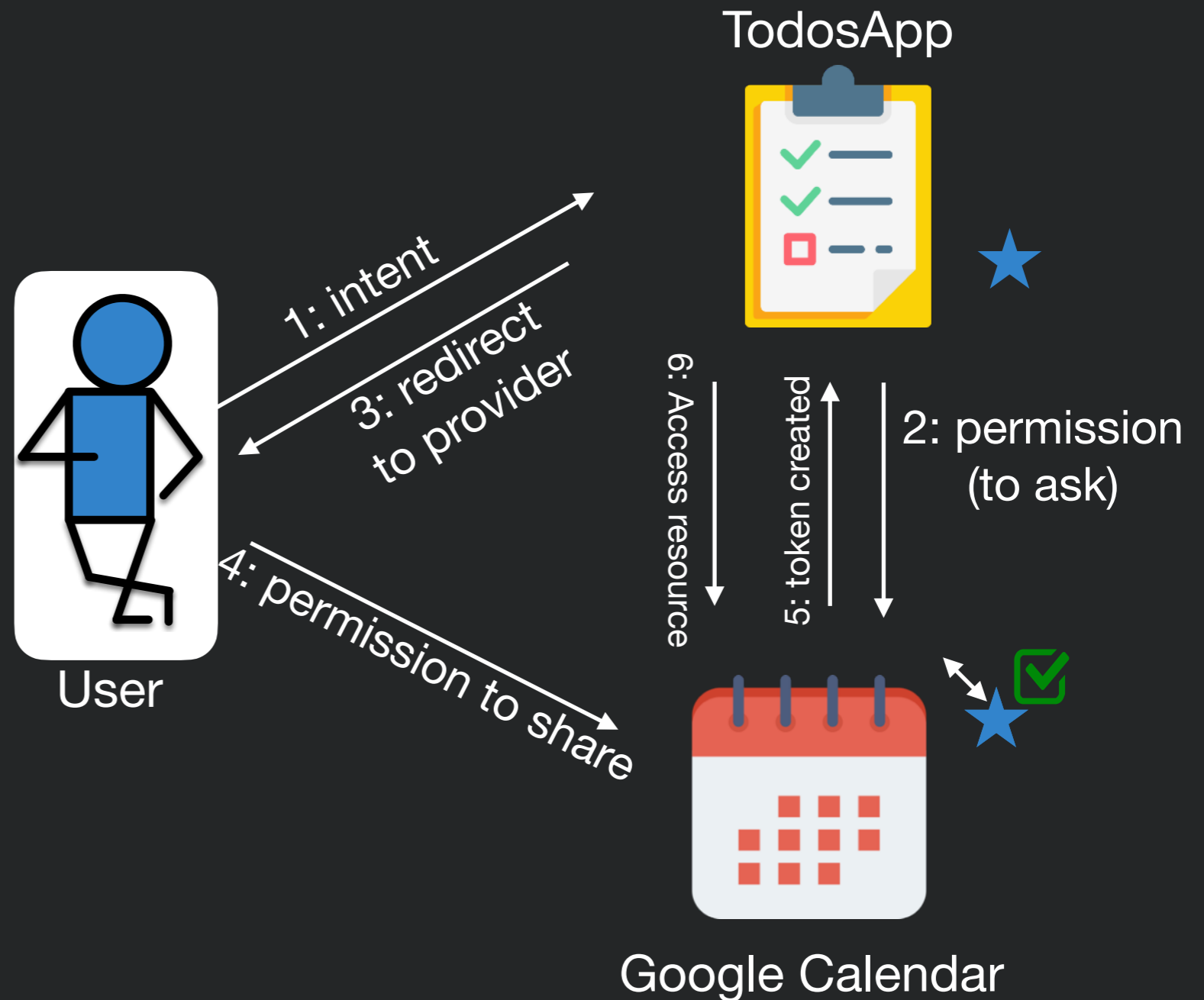
 View your email address ⓘ

By clicking Allow, you allow this app and Google to use your information in accordance with their respective terms of service and privacy policies. You can change this and other [Account Permissions](#) at any time.

- OAuth is a standard protocol for sharing information about users from a “service provider” to a “consumer app” **without** them disclosing their password to the consumer app
- 3 key actors:
 - User, consumer app, service provider app
 - E.x. “Prof Hacker,” “Todos App,” “Google Calendar”
- Service provider issues a **token** on the user’s behalf that the consumer can use
- Consumer holds onto this token on behalf of the user
- Protocol could be considered a conversation...

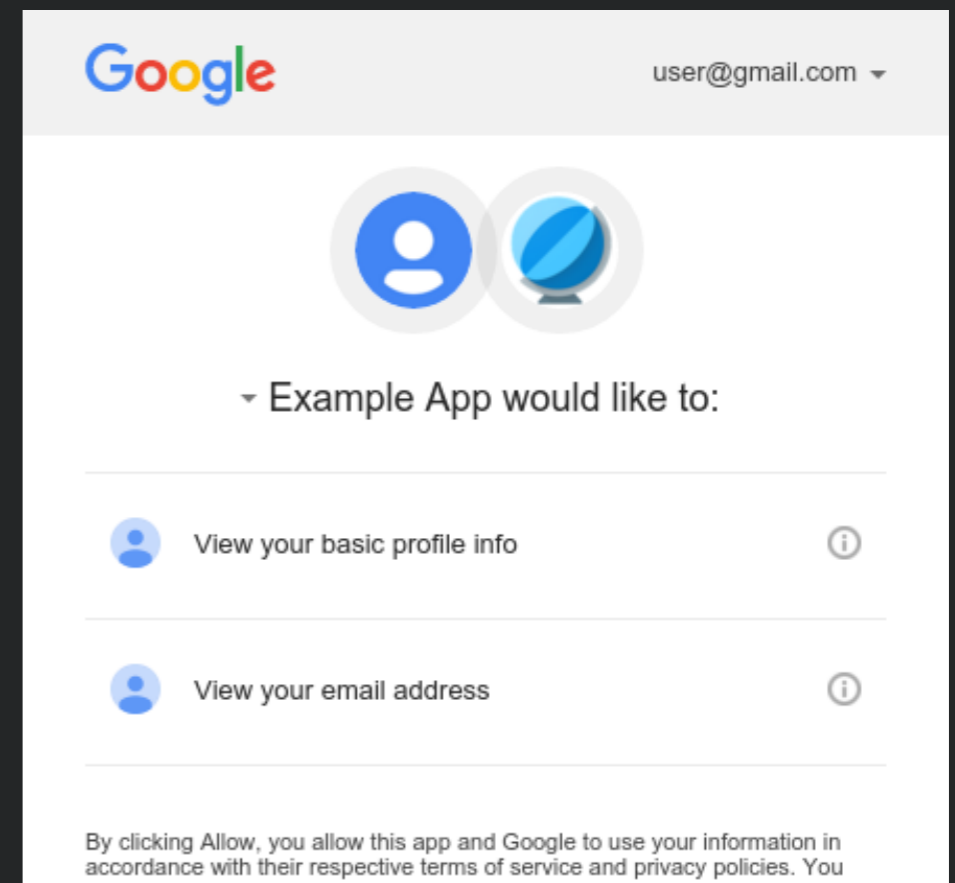
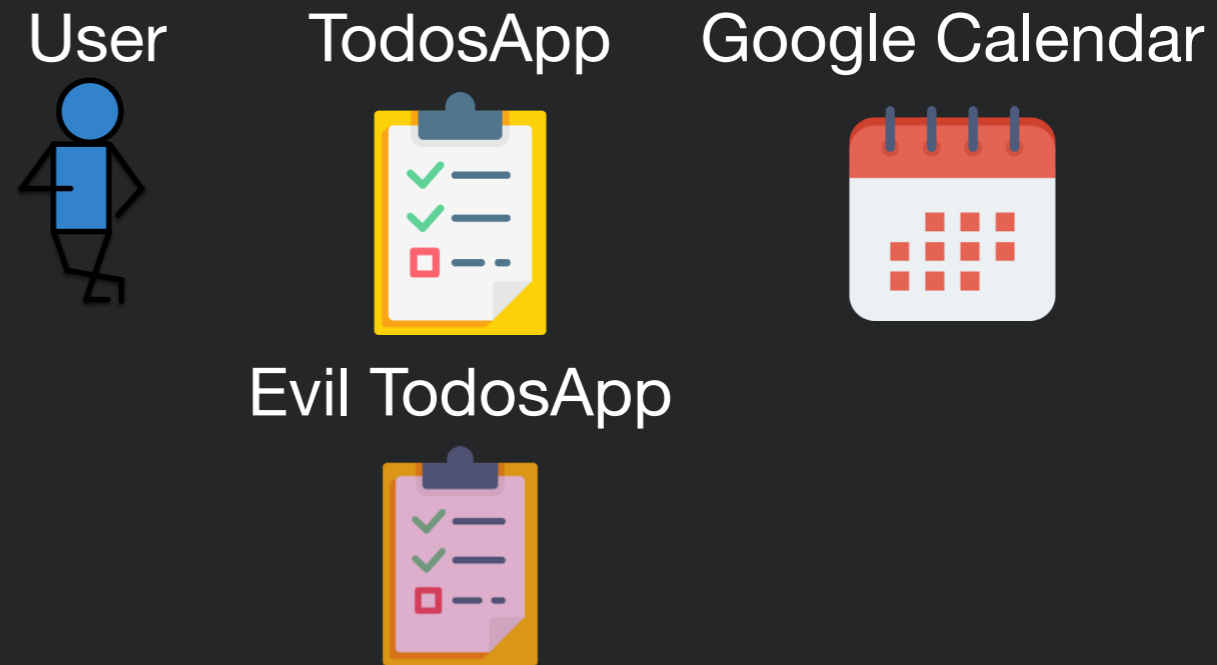
An OAuth Conversation

Goal: TodosApp can post events to User's calendar. TodosApp never finds out User's email or password



Trust in OAuth

- How does the Service provider (Google calendar) know what the TodosApp is?
- Solution: When you set up OAuth for the first time, you must register your consumer app with the service provider
- Let the user decide
 - ... they were the one who clicked the link after all

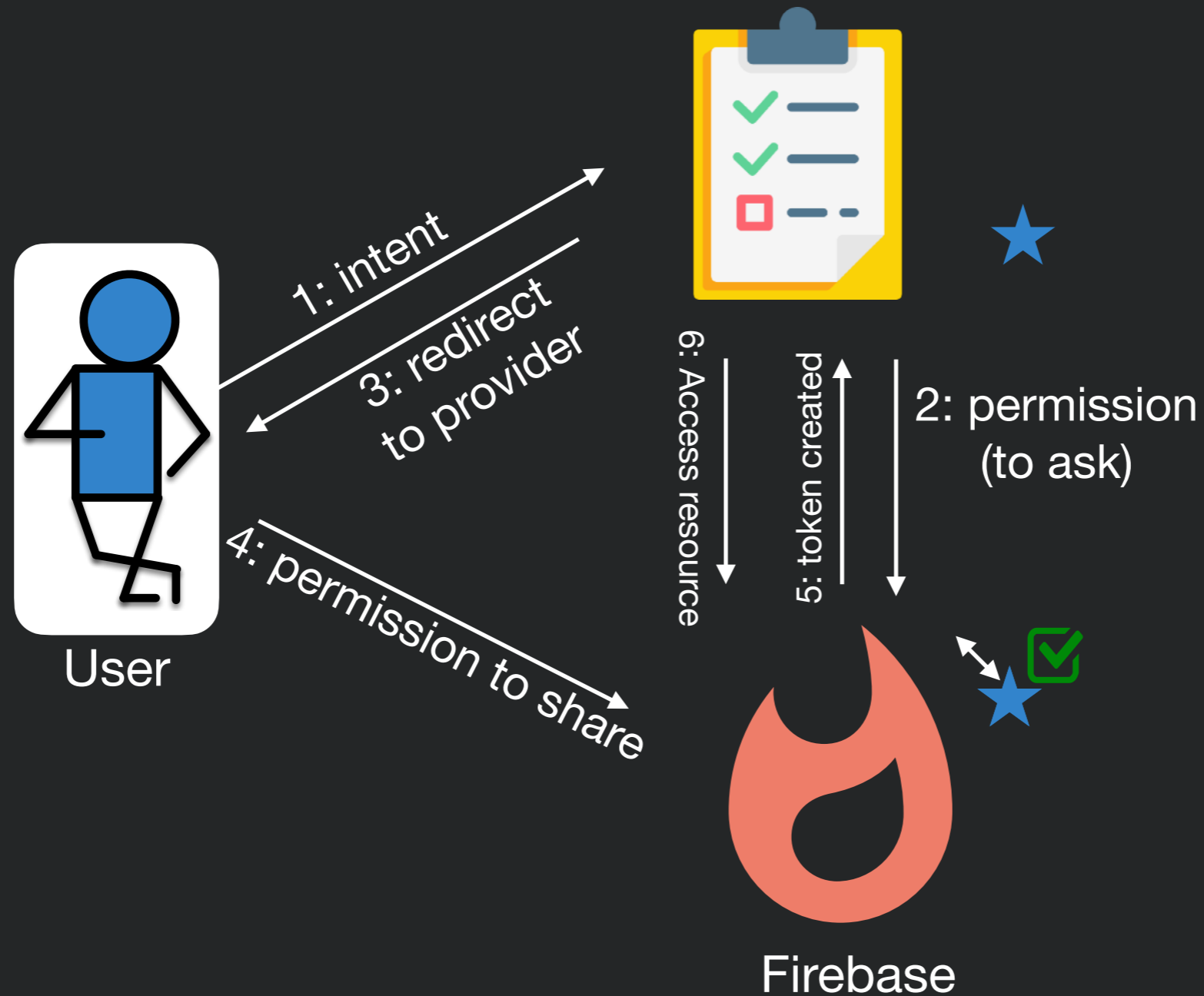




Authentication as a Service

- Whether we are building “microservices” or not, might make sense to farm out our authentication (user registration/logins) to another service
- Why?
 - Security
 - Reliability
 - Convenience
- We can use OAuth for this!

Using an Authentication Service





Firebase Authentication

- Firebase provides an entire suite of authentication services you can use to build into your app
- Can either use “federated” logins (e.g. login with google, facebook, GitHub credentials) or simple email/password logins. Use whichever you want.
- Getting started guide: <https://github.com/firebase/FirebaseUI-Web>
- Firebase handles browser local storage to track that the user is logged in across pages (woo)



Top 3 Web Vulnerabilities

- OWASP collected data on vulnerabilities
 - Surveyed 7 firms specializing in web app security
 - Collected 500,000 vulnerabilities across hundreds of apps and thousands of firms
 - Prioritized by prevalence as well as exploitability, detectability, impact

https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project



#3 - XSS: Cross Site Scripting

- User input that contains a *client-side* script that does not belong
 - A todo item:

```
/><script>alert("LASAGNA FOR PRESIDENT");</script>
```

- Works when user input is used to render DOM elements without being escaped properly
- User input saved to server may be served to other users
 - Enables malicious user to execute code on other's users browser
 - e.g., click 'Buy' button to buy a stock, send password data to third party, ...

#2 - Broken Authentication and Session Management



- Building authentication is hard
 - Logout, password management, timeouts, secrete questions, account updates, ...
- Vulnerability may exist if
 - User authentication credentials aren't protected when stored using hashing or encryption.
 - Credentials can be guessed or overwritten through weak account management functions (e.g., account creation, change password, recover password, weak session IDs).
 - Session IDs are exposed in the URL (e.g., URL rewriting).
 - Session IDs don't timeout, or user sessions or authentication tokens, particularly single sign-on (SSO) tokens, aren't properly invalidated during logout.
 - Session IDs aren't rotated after successful login.
 - Passwords, session IDs, and other credentials are sent over unencrypted connections.



#1 - Injection

- User input that contains *server-side* code that does not belong
- Usually comes up in context of SQL (which we aren't using)
 - e.g.,
 - `String query = "SELECT * FROM accounts WHERE custID='" + request.getParameter("id") + "'";`
- Might come up in JS in context of eval
 - `eval(request.getParameter("code"));`
 - Obvious injection attack - don't do this!



Validating User Input

- Escape Strings that originate from user
- Type of escaping depends on where data will be used
 - HTML - HTML entity encoding
 - URL - URL Escape
 - JSON - Javascript Escape
- Done automatically by some frameworks such as React
- More details: [https://www.owasp.org/index.php/XSS_\(Cross_Site_Scripting\)_Prevention_Cheat_Sheet](https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet)



Authentication: Sharing Data Between Pages

- Browser loads many pages at the same time.
- Might want to share data between pages
 - Popup that wants to show details for data on main page
- Attack: malicious page
 - User visits a malicious page in a second tab
 - Malicious page steals data from page or its data, modifies data, or impersonates user



Solution: Same-Origin Policy

- Browser needs to differentiate pages that are part of same application from unrelated pages
- What makes a page similar to another page?
 - Origin: the **protocol**, **host**, and **port**

<http://www.example.com/dir/page.html>

- Different origins:

<https://www.example.com/dir/page.html>

<http://www.example.com:80/dir/page.html>

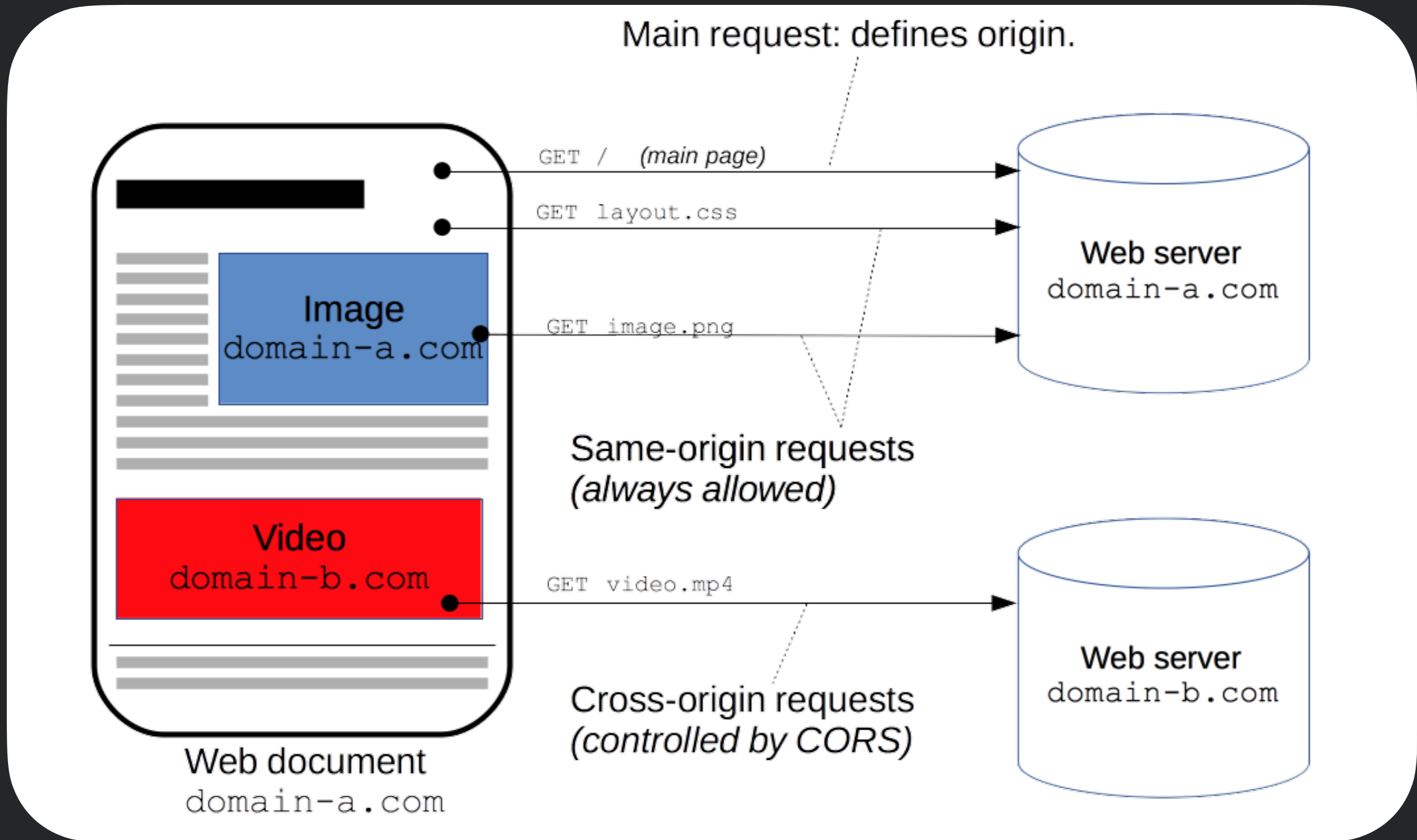
<http://en.example.com:80/dir/page.html>

https://en.wikipedia.org/wiki/Same-origin_policy

Same-Origin Policy

- “Origin” refers to the *page that is executing it*, NOT where the data comes from
 - Example:
 - In one HTML file, I directly include 3 JS scripts, each loaded from a different server
 - -> All have same “origin”
 - Example:
 - One of those scripts makes an AJAX call to yet another server
 - -> AJAX call not allowed
- Scripts contained in a page may access data in a second web page (e.g., its DOM) if they come from the same origin

Cross Origin Requests





CORS: Cross Origin Resource Sharing

- Same-Origin might be safer, but not really usable:
 - How do we make AJAX calls to other servers?
- Solution: Cross Origin Resource Sharing (CORS)
- HTTP header:

```
Access-Control-Allow-Origin: <server or wildcard>
```

- In Express:

```
res.header("Access-Control-Allow-Origin", "*");
```



Takeaways

- Think about all potential threat models
 - Which do you care about
 - Which do you not care about
- What user data are you retaining
 - Who are you sharing it with, and what might they do with it

10 Minute Break



SWE 432 - Web Application Development



George Mason
University

Instructor:
Dr. Kevin Moran

Teaching Assistant:
Oyindamola Oluyemo

Class will start in:
10:00

Templates, Databinding, & HTML



Today



- HTML
- Frontend JavaScript
- Intro to templating and React

HTML: HyperText Markup Language

- Language for describing *structure* of a document
- Denotes hierarchy of elements
- What might be elements in this document?





HTML History

- 1995: HTML 2.0. Published as standard with RFC 1866
- 1997: HTML 4.0 Standardized most modern HTML element w/ W3C recommendation
 - Encouraged use of CSS for styling elements over HTML attributes
- 2000: XHTML 1.0
 - Imposed stricter rules on HTML format
 - e.g., elements needed closing tag, attribute names in lowercase
- 2014: HTML5 published as W3C recommendation
 - New features for capturing more *semantic* information and *declarative* description of behavior
 - e.g., Input constraints
 - e.g., New tags that explain *purpose* of content
 - Important changes to DOM



HTML Elements

`<p lang="en-us">This is a paragraph in English.</p>`

name

value

“Start a paragraph element”

“Set the language to English”

“End a paragraph element”

Opening tag begins an HTML element. Opening tags must have a corresponding closing tag.

HTML attributes are name / value pairs that provide additional information about the contents of an element.

Closing tag ends an HTML element. All content between the tags and the tags themselves compromise an HTML element.



HTML Elements

```
<input type="text" />
```

“Begin and end input
element”

Some HTML tags can be self
closing, including a built-in
closing tag.

```
<!-- This is a comment.  
Comments can be multiline. -->
```



A Starter HTML Document

“Use HTML5 standards mode”

“HTML content”

“Header”

Information *about* the page

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Hello World Site</title>
</head>
<body>
  Hello world!
</body>
</html>
```

Hello world!

“Interpret bytes as UTF-8 characters”

“Title”

“Document content”

Used by browser for title bar or tab.

Includes both ASCII & international characters.



HTML Example

```
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" type="text/css" href="style.css">
    <title>Prof Moran's Webpage</title>
  </head>
  <body>
    <img alt="laptop-01.gif" />
    <br>
    <div class="marquee"> This is Prof Moran's ACTUAL homepage from 19991 </div> </p>
    <h2>Welcome, students!</h2>
    <p> <a href="https://www.youtube.com/watch?v=d0w4w9WgXW">See how to make this
page</a> </p>
    <h2> Some funny links </h2>
    <p>
    <ul>
      <li><a href="http://www.homestarrunner.com">Homestar Runner</a></li>
      <li><a href="http://www.wb3w.net/The/6200riginal1420Hamsterdance.htm">Hamster
Dance</a></li> </ul>
    </p> <h3> About Prof Moran </h3>
    <p> Prof Moran's office is at 4442 Engineering Building. His email address is <a
href="mailto:kpmoran@gmu.edu">kpmoran@gmu.edu</a>. </p>
    <p> Last updated: September 28th, 1999 </p> </div> </body>
</html>
```

Use `<h1>`, `<h2>`, ..., `<h5>` for headings

Prof Kevin Moran



This is Prof Moran's ACTUAL homepage from 1991

Welcome, students!

[See how to make this page](https://www.youtube.com/watch?v=d0w4w9WgXW)

Some funny links

- [Homestar Runner](http://www.homestarrunner.com)
- [Hamster Dance](http://www.wb3w.net/The/6200riginal1420Hamsterdance.htm)

About Prof Moran

Prof Moran's office is at 4442 Engineering Building. His email address is kpmoran@gmu.edu.

Last updated: September 28th, 1999



HTML Example

```

<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" type="text/css" href="style.css">
    <title>Prof Moran's Webpage</title>
  </head>
  <body>
    <h1> Prof Kevin Moran </h1>
    <div>
      <p> <img alt="My really cool laptop"
    </p>
    <ul>
      <li><a href="http://www.homestarrunner.com">Homestar Runner</a></li>
      <li><a href="http://www.wb3w.net/The/6200riginal1420Hamsterdance.htm">Hamster
Dance</a></li> </ul>
    </p> <h3> About Prof Moran </h3>
    <p> Prof Moran's office is at 4442 Engineering Building. His email address is <a
href="mailto:kpmoran@gmu.edu">kpmoran@gmu.edu</a>. </p>
    <p> Last updated: September 28th, 1999 </p> </div> </body>
</html>

```

Paragraphs (<p>) consist of related content. By default, each paragraph starts on a new line.

Prof Kevin Moran



This is Prof Moran's ACTUAL homepage from 1991

Welcome, students!

[See how to make this page](#)

Some funny links

- [Homestar Runner](#)
- [Hamster Dance](#)

About Prof Moran

Prof Moran's office is at 4442 Engineering Building. His email address is kpmoran@gmu.edu.

Last updated: September 28th, 1999



HTML Example

```
<!DOCTYPE html>
<html>
<head>
<link rel="stylesheet" type="text/css" href="style.css">
<title>Prof Moran's Webpage</title>
</head>
<body>
<h1> Prof Kevin Moran </h1>
<div>
<p> 
<br>
<div class="marquee"> This is Prof Moran's ACTUAL homepage from 19991 </div> </p>
<h2>Welcome, students!</h2>
<p> <a href="https://www.youtube.com/watch?v=d0w4w9WgXW">See how to make this
page</a> </p>
<h2> Some funny links </h2>
```

Prof Kevin Moran



This is Prof Moran's ACTUAL homepage from 1991

Welcome, students!

[See how to make this page](#)

Some funny links

- [Homestar Runner](#)
- [Hamster Dance](#)

About Prof Moran

Prof Moran's office is at 4442 Engineering Building. His email address is kpmoran@gmu.edu.

Last updated: September 28th, 1999

Unordered lists () consist of list items () that each start on a new line. Lists can be nested arbitrarily deep.

```
<p> <a href="mailto:kpmoran@gmu.edu">kpmoran@gmu.edu</a>. </p>
<p> Last updated: September 28th, 1999 </p> </div> </body>
</html>
```



```
9 <h1>Level 1 Heading</h1>
10 <h2>Level 2 Heading</h2>
11 <h3>Level 3 Heading</h3>
12 <h4>Level 4 Heading</h4>
13 <h5>Level 5 Heading</h5>
14 <h6>Level 5 Heading</h6>
15 Text can be made <b>bold</b> and
16 <i>italic</i>, or <sup>super</sup>
17 and <sub>sub</sub>scripts. White
18 space collapsing removes all
19 sequences of two more more spaces
20 and line breaks, allowing
21 the markup to use tabs
22 and whitespace for
23 organization.
24 Spaces can be added with
25 &nbsp; &nbsp; &nbsp; &nbsp;.
26 <br/>New lines can be added with &lt;
27 ;BR/&gt;.
28
29 <p>A paragraph consists of one or
30 more sentences that form a self-
31 -contained unit of discourse. By
32 default, a browser will show each
33 paragraph on a new line.</p>
34
35 <hr/>
36 Text can also be offest with
37 horizontal rules.
```

Level 1 Heading

Level 2 Heading

Level 3 Heading

Level 4 Heading

Level 5 Heading

Level 5 Heading

Text can be made **bold** and *italic*, or ^{super} and _{sub}scripts.

White space collapsing removes all sequences of two more more spaces and line breaks, allowing the markup to use tabs and whitespace for organization. Spaces can be added with .

New lines can be added with
.

A paragraph consists of one or more sentences that form a self-contained unit of discourse. By default, a browser will show each paragraph on a new line.

Text can also be offest with horizontal rules.

Semantic markup

- Tags that can be used to denote the *meaning* of specific content
- Examples
 - `` - An element that has importance.
 - `<blockquote>` - An element that is a longer quote.
 - `<q>` - A shorter quote inline in paragraph.
 - `<abbr>` - Abbreviation
 - `<cite>` - Reference to a work.
 - `<dfn>` - The definition of a term.
 - `<address>` - Contact information.
 - `<ins>` - Content that was inserted or deleted.
 - `<s>` - Something that is no longer accurate.



Links

```
<a href="http://www.google.com">Absolute link</a><br/>  
<a href="movies.html">Relative URL</a><br/>  
<a href="mailto:tlatoya@gmu.edu">Email Prof. LaToza</a><br/>  
<a href="http://www.google.com" target="_blank">Opens in new  
  window</a><br/>  
<a href="#idName">Navigate to HTML element idName</a>
```

[Absolute link](#)

[Relative URL](#)

[Email Prof. LaToza](#)

[Opens in new window](#)

[Navigate to HTML element idName](#)

Controls



```
<p>Text Input: <input type="text" maxlength="5" /></p>
<p>Password Input: <input type="password" /></p>
<p>Search Input: <input type="search"></p>
<p>Text Area: <textarea>Initial text</textarea></p>
<p>Checkbox:
  <input type="checkbox" checked="checked" /> Checked
  <input type="checkbox" /> Unchecked
</p>
<p>Drop Down List Box:
  <select>
    <option>Option1</option>
    <option selected="selected">Option2</option>
  </select>
</p>
<p>Multiple Select Box:
  <select multiple="multiple">
    <option>Option1</option>
    <option selected="selected">Option2</option>
  </select>
</p>
<p>File Input Box: <input type="file" />
<p>Image Button: <input type="image" src="http://cs.gmu.edu/~tlatzoza
  /images/reachabilityQuestion.jpg" width="50"></p>
<p>Button: <button>Button</button></p>
<p>Range Input: <input type="range" min="0" max="100" step="10"
  value="30" /></p>
```

**Search
input
provides
clear
button**

Text Input:

Password Input:

Search Input:

Text Area:

Checkbox: Checked Unchecked

Drop Down List Box:

Multiple Select Box:

File Input Box: No file chosen

Image Button:

Button:

Range Input:

Block vs. Inline Elements

Block elements

Block elements appear on a new line.
Examples: `<h1>``<p>````<table>``<form>`



```
<h1>Hiroshi Sugimoto</h1>  
<p>The dates for the ORIGIN OF ART exhibition are as follows:</p>  
<ul>  
  <li>Science: 21 Nov- 20 Feb 2010/2011</li>  
  <li>Architecture: 6 Mar - 15 May 2011</li>  
</ul>
```

Hiroshi Sugimoto

The dates for the ORIGIN OF ART exhibition are as follows:

- Science: 21 Nov- 20 Feb 2010/2011
- Architecture: 6 Mar - 15 May 2011

Inline elements

Inline elements appear to continue on the same line.
Examples: `<a>````<input>```



Timed to a single revolution of the planet around the sun at a 23.4 degrees tilt that plays out the rhythm of the seasons, this `Origins of Art` cycle is organized around four themes: `science, architecture, history`, and `religion`.

Timed to a single revolution of the planet around the sun at a 23.4 degrees tilt that plays out the rhythm of the seasons, this *Origins of Art* cycle is organized around four themes: **science, architecture, history, and religion.**



Frontend JavaScript

- Static page
 - Completely described by HTML & CSS
- Dynamic page
 - Adds interactivity, updating HTML based on user interactions
- Adding JS to frontend:

```
<script>  
  console.log("Hello, world!");  
</script>
```

- We try to avoid doing this because:
 - Hard to organize
 - Different browsers support different things



DOM: Document Object Model

- API for interacting with HTML browser
- Contains objects corresponding to every HTML element
- Contains global objects for using other browser features

Reference and tutorials

https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model



Global DOM objects

- *window* - the browser window
 - Has properties for following objects (e.g., window.document)
 - Or can refer to them directly (e.g., document)
- *document* - the current web page
- *history* - the list of pages the user has visited previously
- *location* - URL of current web page
- *navigator* - web browser being used
- *screen* - the area occupied by the browser & page

Working with Popups

- alert, confirm, prompt
 - Create *modal* popups
 - User cannot interact with web

```
> window.confirm('Are you sure you want to  
navigate away from this page and discard the  
document you have been writing for the past  
day?');
```

developer.mozilla.org says:

Are you sure you want to navigate away from this page and discard the document you have been writing for the past day?

Prevent this page from creating additional dialogs.

Cancel

OK

developer.mozilla.org says:

Are you sure you want to navigate away from this page and discard the document you have been writing for the past day?

Prevent this page from creating additional dialogs.

Cancel

Working with location

- Some properties
 - location.href - full URL of current location
 - location.protocol - protocol being used
 - location.host - hostname
 - location.port
 - location.pathname
- Can navigate to new page by updating the current location
 - location.href = '[new URL]';

```
Location {hash: "", search: "", pathname:
  "/~tlatoza/", port: "", hostname:
  "cs.gmu.edu"...} ⓘ
  ▶ ancestorOrigins: DOMStringList
  ▶ assign: function ()
    hash: ""
    host: "cs.gmu.edu"
    hostname: "cs.gmu.edu"
    href: "http://cs.gmu.edu/~tlatoza/"
    origin: "http://cs.gmu.edu"
    pathname: "/~tlatoza/"
    port: ""
    protocol: "http:"
  ▶ reload: function reload()
```

Traveling Through History

- `history.back()`, `history.forward()`, `history.go(delta)`
- What if you have an SPA & user navigates through different views?
 - Want to be able to jump between different views *within* a single URL
- Solution: manipulate history state
 - Add entries to history stack describing past views
 - Store and retrieve object using `history.pushState()` and `history.state`

```
> history.pushState( { activePane: 'main' }, ""  
< undefined  
> history.state  
< ► Object {activePane: "main"}  
> history.back();  
< undefined  
> history.state  
< null
```



DOM Manipulation

- We can also manipulate the DOM directly
- For this class, we will *not* focus on doing this, but will use React instead
- This is how React works though - it manipulates the DOM



DOM Manipulation

Multiply two numbers

* = 6

```
<h3>Multiply two numbers</h3>
<div>
  <input id="num1" type="number" /> *
  <input id="num2" type="number" /> =
  <span id="product"></span>
  <br/><br/>
  <button id="compute">Multiply</button>
</div>
```

“Get compute element”

```
document.getElementById('compute')
  .addEventListener("click", multiply);
function multiply()
{
  var x = document.getElementById('num1').value;
  var y = document.getElementById('num2').value;
  var productElem = document.getElementById('product');
  productElem.innerHTML = x * y;
}
```

“When compute is clicked, call multiply”

May choose any event that the compute element produces. May pass the name of a function or define an anonymous function inline.



DOM Manipulation

Multiply two numbers

* = 6

```
<h3>Multiply two numbers</h3>
<div>
  <input id="num1" type="number" /> *
  <input id="num2" type="number" /> =
  <span id="product"></span>
  <br/><br/>
  <button id="compute">Multiply</button>
</div>
```

```
document.getElementById('compute')
  .addEventListener("click", multiply);
function multiply()
{
  var x = document.getElementById('num1').value;
  var y = document.getElementById('num2').value;
  var productElem = document.getElementById('product');
  productElem.innerHTML = x * y;
}
```

“Get the current value of the num1 element”

“Set the HTML between the tags of productElem to the value of x * y”

Manipulates the DOM by programmatically updating the value of the HTML content. DOM offers accessors for updating all of the DOM state.



DOM Manipulation Pattern

- Wait for some event
 - click, hover, focus, keypress, ...
- Do some computation
 - Read data from event, controls, and/or previous application state
 - Update application state based on what happened
- Update the DOM
 - Generate HTML based on new application state
- Also: JQuery



Problems with Direct DOM Manipulation

- Managing state becomes difficult for complex applications
- Directly Manipulating the DOM can be very slow
- Reasoning about the many different states in code can become difficult
- Working in a team trying to reason about many different states in code is even more difficult
- Working directly with the DOM is possible, but requires discipline and great documentation.
- Modern web frameworks like Vue.js and React.js make this much easier.

Examples of events

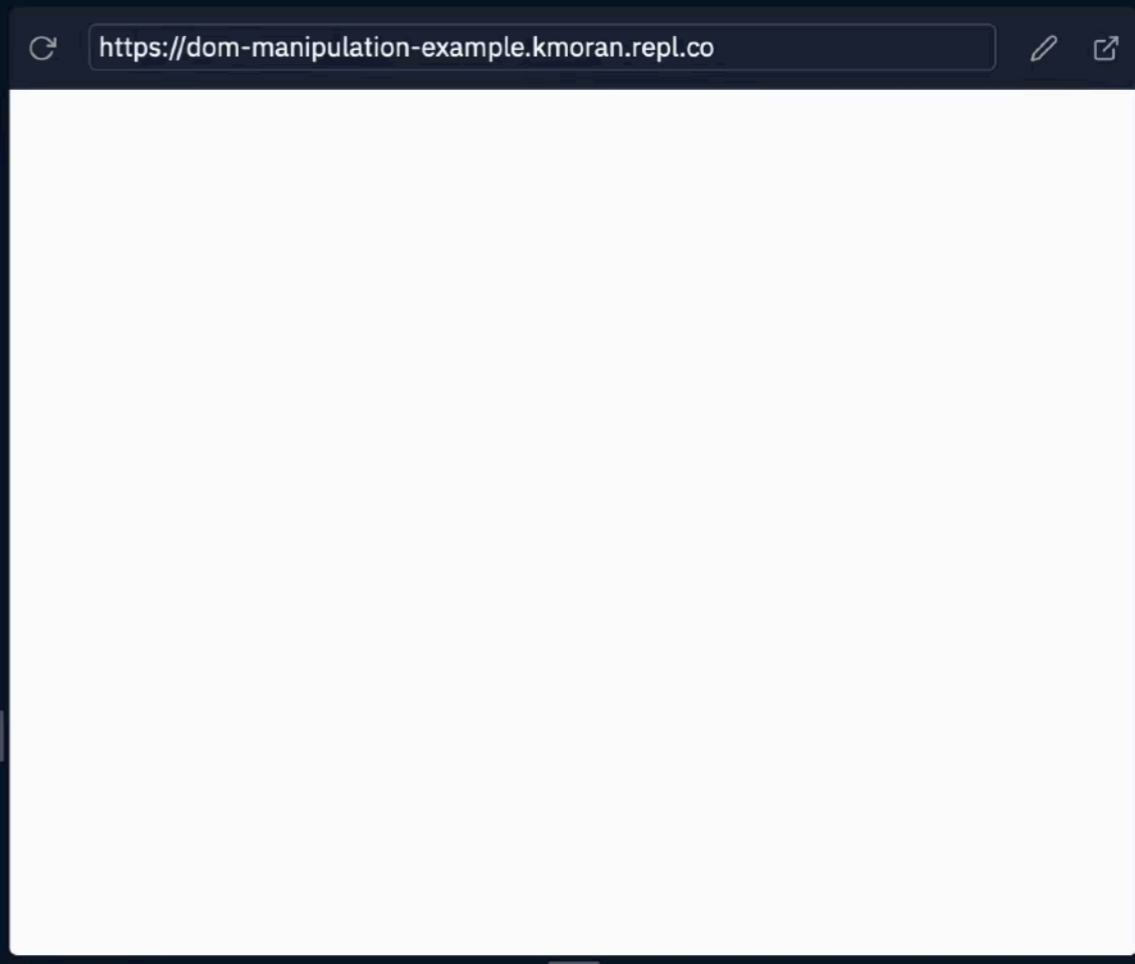
- **Form element events**
 - change, focus, blur
- **Network events**
 - online, offline
- **View events**
 - resize, scroll
- **Clipboard events**
 - cut, copy, paste
- **Keyboard events**
 - keydown, keypress, keyup
- **Mouse events**
 - mouseenter, mouseleave, mousemove, mousedown, mouseup, click, dblclick, select

DOM Manipulation Example



- index.html
- script.js
- style.css

```
1 <!DOCTYPE html>
2 <html>
3
4 <head>
5   <title>Dom Manipulation Example</title>
6 </head>
7
8 <body>
9   <div id="todoItems"></div>
10  <button id="new">New item</button>
11 </body>
12 </html>
13 <script src="script.js"></script>
```



Console Shell

Loading Pages

- What is the output of the following?

```
<script>
```

```
document.getElementById( 'elem' ).innerHTML =  
'New content';
```

```
</script>
```

```
<div id="elem">Original content</div>
```

- **Answer:** cannot set property innerHTML of undefined
- **Solution:** Put your script in after the rest of the page is loaded Or, perhaps better solution: don't do DOM manipulation



Anatomy of a Non-Trivial Web App

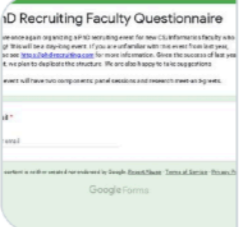
Twitter navigation menu:

- Home
- Explore
- Notifications
- Messages
- Bookmarks
- Lists
- Profile
- More

Profile Header:

Kevin Moran
714 Tweets
Assistant Professor @GMUCompSci, @holy_cross & @williamandmary alum, Software Engineering Researcher, Director of @SageSELab
Fairfax, VA | kpmoran.com | Joined April 2011
622 Following | 482 Followers

Tweets:

You Retweeted
Emma Tosch, not a thought leader. @ToschEmma · Aug 19
Are you a new-ish faculty member in a CS-ish discipline who is recruiting graduate students for the 2022-23 academic year? If yes, consider participating in the second annual phd-recruiting.com event! Interest form here:

PhD Recruiting Faculty Questionnaire
We are once again organizing a PhD recruiting event for new CS/Informatics faculty who are ...
docs.google.com

You Retweeted
Brittany Johnson-Matthews, PhD @DrBrittJay · Aug 17

User profile widget

Menu Bar Widget

Feed widget

Feed item widget



Typical Properties of Web App UIs

- Each widget has both visual **presentation** & **logic**
 - e.g., clicking on follow button executes some logic related to the containing widget
 - Logic and presentation of individual widget strongly related, loosely related to other widgets
- Some widgets occur **more than once**
 - e.g., Follow widget occurs multiple times in Who to Follow Widget
 - Need to generate a copy of widget based on data
- Changes to **data** should cause changes to **widget**
 - e.g., following person should update UI to show that the person is followed. Should work even if person becomes followed through other UI
- Widgets are **hierarchical**, with parent and child
 - Seen this already with container elements in HTML...



Idea 1: Templates

```
document.getElementById('todoItems').innerHTML +=  
    '<div class="todoItem" data-index="' + key  
    + '"><input type="text" onchange="itemChanged(this)" value="'  
+ value + '"><button onclick="deleteItem(this.parentElement)">&#x2716;</button></div>';
```

- Templates describe ***repeated*** HTML through a single ***common*** representation
 - May have ***variables*** that describe variations in the template
 - May have ***logic*** that describes what values are used or when to instantiate template
 - Template may be ***instantiated*** by binding variables to values, creating HTML that can be used to update DOM



Templates with Template Literals

```
document.getElementById('todoItems').innerHTML +=  
  `    <input type="text" onchange="itemChanged(this)" value="${value}">  
    <button onclick="deleteItem(this.parentElement)">✖</button>  
  </div>` ;
```

- Template literals reduce confusion of nested strings

Server Side vs. Client Side

- Where should template be instantiated?
- *Server-side* frameworks: Template instantiated on **server**
 - Examples: JSP, ColdFusion, PHP, ASP.NET
 - Logic executes on server, generating HTML that is served to browser
- *Front-end* framework: Template runs in web **browser**
 - Examples: React, Angular, Meteor, Ember, Aurelia, ...
 - Server passes template to browser, browser generates HTML on demand

```
<html>
<head><title>First JSP</title></head>
<body>
  <%
    double num = Math.random();
    if (num > 0.95) {
  %>
    <h2>You'll have a luck day!</h2><p><%= num %></p>
  <%
    } else {
  %>
    <h2>Well, life goes on ... </h2><p><%= num %></p>
  <%
    }
  %>
```

```
class HelloMessage extends React.Component {
  render() {
    return (
      <div>
        Hello {this.props.name}
      </div>
    );
  }
}

ReactDOM.render(
  <HelloMessage name="John" />,
  mountNode
);
```



Server Side vs. Client Side

- Server side
 - Oldest solution.
 - True when “real” code ran on server, Javascript
- Client side
 - Enables presentation logic to exist entirely in browser
 - e.g., can make call to remote web service, no need for server to be involved
 - (What we are looking at in this course).



- Templates require combining logic with HTML
 - Conditionals - only display presentation if some expression is true
 - Loops - repeat this template once for every item in collection
- How should this be expressed?
 - Embed code in HTML (ColdFusion, JSP, Angular)
 - Embed HTML in code (React)

Embed Code in HTML

```
<cfcomponent name = "PersonalChef">
  <cffunction name = "makeToast" returnType = "component">
    <cfargument name = "color" required="yes">

    <cfset this.makeToast = "Making your toast #arguments.color#!" />
    <cfreturn this />
  </cffunction>
</cfcomponent>
```

```
<html>
<head><title>First JSP</title></head>
<body>
  <%
    double num = Math.random();
    if (num > 0.95) {
  %>
    <h2>You'll have a luck day!</h2><p>(<%= num %>)</p>
  <%
    } else {
  %>
    <h2>Well, life goes on ... </h2><p>(<%= num %>)</p>
  <%
    }
  %>
```

- Template takes the form of an HTML file, with extensions
 - Custom tags (e.g., <% %>) enable logic to be embedded in HTML
 - Uses another language (e.g., Java, C) or custom language to express logic
 - Found in frameworks such as PHP, Angular, ColdFusion, ASP, ...



Embed HTML in Code

- Template takes the form of an HTML fragment, embedded in a code file
 - HTML instantiated as part of an expression, becomes a value that can be stored to variables
 - Uses another language (e.g., Javascript) to express logic
 - This course: *React*

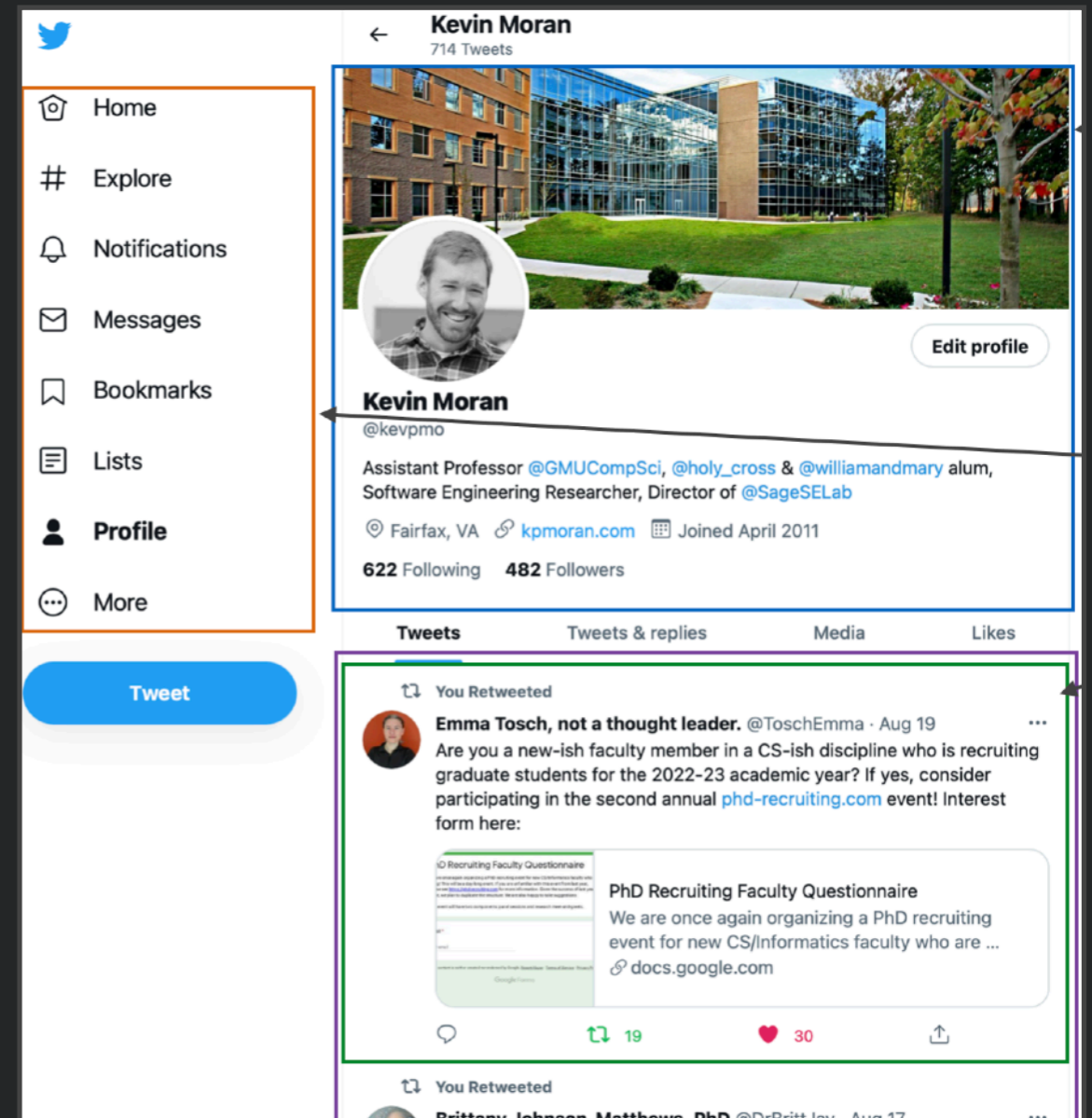


Templates Enable HTML to be Rendered Multiple Times

- Rendering takes a template, instantiates the template, outputs HTML
- Logic determines which part(s) of templates are rendered
- Expressions are evaluated to instantiate values
 - e.g., { this.props.name }
 - Different variable values ==> different HTML output

Idea 2: Components

- Web pages are complex, with lots of logic and presentation
- How can we organize web page to maximize modularity?
- Solution: Components
 - Templates that correspond to a specific widget
 - Encapsulates related logic & presentation using language construct (e.g., class)





Components

- Organize related logic and presentation into a single unit
 - Includes necessary *state* and the logic for updating this state
 - Includes presentation for *rendering* this state into HTML
 - Outside world *must* interact with state through accessors, enabling access to be controlled
- Synchronizes state and visual presentation
 - Whenever state changes, HTML should be rendered again
- Components instantiated through custom HTML tag



React: Front End Framework for Components

React

A JavaScript library for building user interfaces

- Originally built by Facebook
- Open-source frontend framework
- Powerful abstractions for describing frontend UI components
- Official documentation & tutorials
 - <https://reactjs.org/>



Example

```
class HelloMessage extends React.Component {  
  render() {  
    return (  
      <div>  
        Hello world!  
      </div>  
    );  
  }  
}  
  
ReactDOM.render(  
  <HelloMessage/>, mountNode  
);
```

“Declare a HelloMessage component”

Declares a new component with the provided functions.

“Return the following HTML whenever the component is rendered”

Render generates the HTML for the component. The HTML is dynamically generated by the library.

“Render HelloMessage and insert in mountNode”

Instantiates component, replaces mountNode innerHTML with rendered HTML. Second parameter should always be a DOM element.



Example - Properties

```
class HelloMessage extends React.Component {  
  render() {  
    return (  
      <div>  
        Hello {this.props.name}  
      </div>  
    );  
  }  
}  
  
ReactDOM.render(  
  <HelloMessage name="John" />,  
  mountNode  
);
```

“Read `this.props.name` and
output the value”

Evaluates the expression to a value.

“Set the name property of
HelloMessage to John”

Components have a `this.props` collection that
contains a set of properties instantiated for
each component.



Embedding HTML in Javascript

```
return <div>Hello {this.props.name}</div>;
```

- HTML embedded in JavaScript
 - HTML can be used as an expression
 - HTML is checked for correct syntax
- Can use { expr } to evaluate an expression and return a value
 - e.g., { 5 + 2 }, { foo() }
- Output of expression is HTML

- How do you embed HTML in JavaScript and get syntax checking??
- Idea: extend the language: JSX
 - Javascript language, with additional feature that expressions may be HTML
 - Can be used with ES6 or traditional JS (ES5)
- It's a new(ish) language
 - Browsers *do not* natively run JSX
 - If you include a JSX file as source, you will get an error

React

Docs Tutorial Blog Community Search v17.0.2 Languages GitHub

React

A JavaScript library for building user interfaces

[Get Started](#) [Take the Tutorial >](#)

Declarative

React makes it painless to create interactive UIs. Design simple views for each state in your application, and React will efficiently update and render just the right components when your data changes.

Component-Based

Build encapsulated components that manage their own state, then compose them to make complex UIs.

Since component logic is written in JavaScript

Learn Once, Write Anywhere

We don't make assumptions about the rest of your technology stack, so you can develop new features in React without rewriting existing code.

React can also render on the server using Node

The screenshot shows a Replit workspace for a React application. The code in `src/App.jsx` is as follows:

```
1 import React from 'react';
2 import './App.css';
3
4 function App() {
5   return (
6     <main>
7       React + Vite + Replit
8     </main>
9   );
10 }
11
12 export default App;
```

The README.md preview on the right contains the following text:

Running React on Replit.it

React is a popular JavaScript library for building user interfaces.

Vite is a blazing fast frontend build tool that includes features like Hot Module Reloading (HMR), optimized builds, and TypeScript support out of the box.

Using the two in conjunction is one of the fastest ways to build a web app.

Getting Started

- Hit run
- Edit `App.jsx` and watch it live update!

By default, Replit runs the `dev` script, but you can configure it by changing the `run` field in the `.replit` file.

- Pastebin sites such as Replit work with React
- Just need to include React first



Create React App



```
λ npx create-react-app my-app  
npx: installed 114 in 4.308s
```





Acknowledgements

Slides adapted from Dr. Thomas LaToza's
SWE 432 course