

# CSCI E-33a (Web50)

## Section 6

*Ref: Lecture 6 (User Interfaces)*






Vlad Popil

*Oct 26, 2022*

# Welcome!

About me:

## Volodymyr “Vlad” Popil

-  Teaching Fellow for E-33a (since 2019)
-  Father of 4 y/o son
-  Master's (ALM) in Software Engineering
-  Software Engineer at Google
-  Born in Ukraine

Email: [vlad@cs50.harvard.edu](mailto:vlad@cs50.harvard.edu)

Sections: Wed 8:30-10:00 pm ET

Office Hours: Fri 7:00-8:30 pm ET

# Agenda

- Logistics
- Lecture review
- Animation/Visualization
- Demo
- Project 3 (recap)
- jshint
- Grading criteria (reminders)
- (maybe) cURL/Postman
- Q&A

# Logistics

# Intro

- Refer to website: <https://cs50.harvard.edu/extension/web/2022/fall>
- Sections and office hours schedule on website sections
- Get comfortable with [command line](#) (cd, ls, cat, python ..., python -m pip ..., rm, touch)
- Text editor is usually sufficient to write code, BUT IDEs (e.g. VSCode) is faster!
- Zoom:
  - Use zoom features like raise hand, chat and other
  - Video presence is STRONGLY encouraged
  - Mute your line when not speaking (enable temporary unmute)
- Six projects:
  - Start early (or even better RIGHT AWAY!!!)
  - Post **and answer** questions on Ed platform
  - Remember: bugs can take time to fix
  - Grade  $\rightarrow 3 \times \text{Correctness (5/5)} + 2 \times \text{Design [code] (5/5)} + 1 \times \text{Style [code] (5/5)}$  (Project 0 is an exception)
    - $15+10+5=30/30$  | e.g. Correctness can be 15, 12, 9, 6, 3, 0
  - Overall weights: projects (90%), section attendance (10%)
  - [Lateness policy](#) - 72 late hours combined for first 5 proj., then 0.1per minute => **16hrs 40 min**
  - Project 3 - Due Sunday, Oct 30th at 11:59pm EDT << **ONLY 5 FULL DAYS LEFT** >>

# Reminders

- Sections/Office Hours:
  - Sections are recorded (published 72hrs), office hours are not
  - Real-time attendance is required of at least one section
  - Video and participation encouraged even more
- Section prep:
  - Watch lecture(s)
  - Review project requirements
- Office hours prep:
  - *~Write down your questions as you go, TODO, etc.~*
  - Come with particular questions

# 10,000 foot overview

- *Section 0 - SKIPPED*
- *Section 1+2 (Git + Python) - Chrome Dev Tools (Inspector), CDT (Network), Project 0, Grading aspects*
- *Section 3 (Django) - Env Config, Markdown, RegEx, IDEs, pycodestyle, Debugging, Project 1*
- *Section 4 (SQL, Models, Migrations) - VSCode, linting, DB modeling, Project 2*
- *Section 5 (JavaScript) - CDT + IDE's Debugging, Project 3*
- *Section 6 (User Interfaces) - Animations, cURL/Postman, jshint*
- *Section 7 (Testing, CI/CD) - Project 4, DB modeling, Pagination, Test Driven Development, DevOps*
- *Section 8 (Scalability and Security) - Final Project, Cryptography, CAs, Attacks, App Deployment (Heroku)*

# Burning Questions?

Please ask questions, or topics to cover today!

Topics:

- If recipient == self, email is already read (expected)
- Final project → release Mon 11/14
- pycodestyle for JS → jshint
- car.model notation is better than car['model']



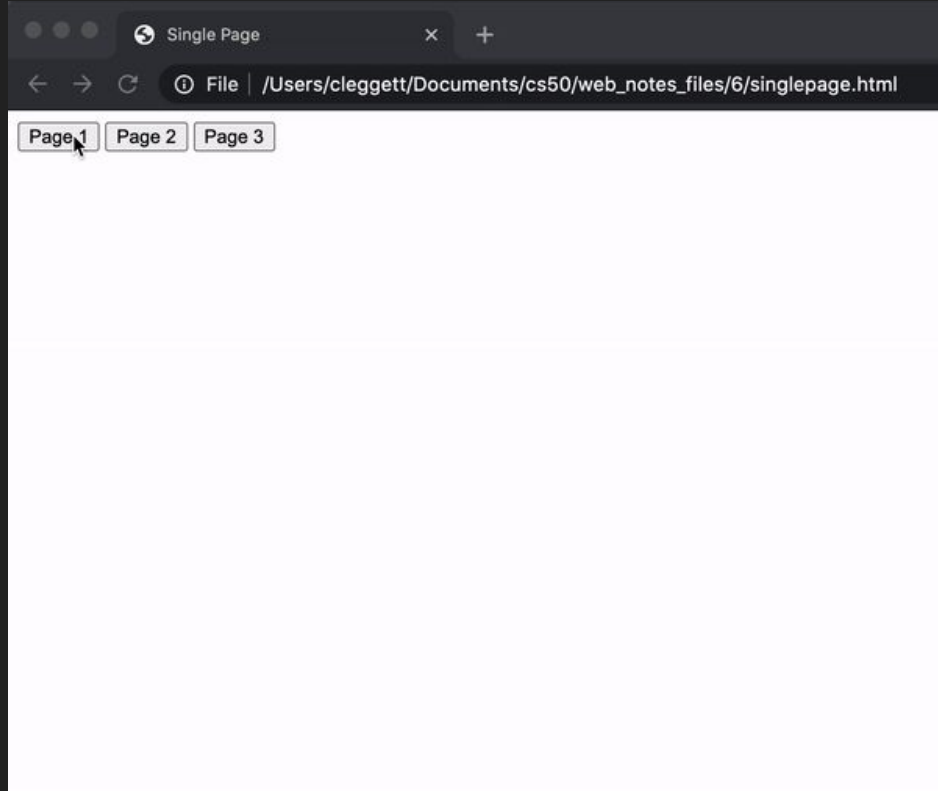
# Lecture Recap

5 min

# User Interfaces

- We want the User Experience to be as good as possible on our websites.
- There are many methods we can use to improve our interfaces:
  - Visually appealing pages (CSS)
  - Single-Page applications (Javascript)
    - React is one way of doing this
  - Animation (CSS)

# Single-Page Applications



# Single-Page Applications

## Advantages:

- Only need to re-render the parts of the page that are changing.
- Often much faster than switching pages
- Debugging in Google Chrome

## Disadvantages:

- A bit more difficult to manage the URL and history if you wish to do so.
- Have to be more careful about security.
- Initial Download could be slower

# Using APIs

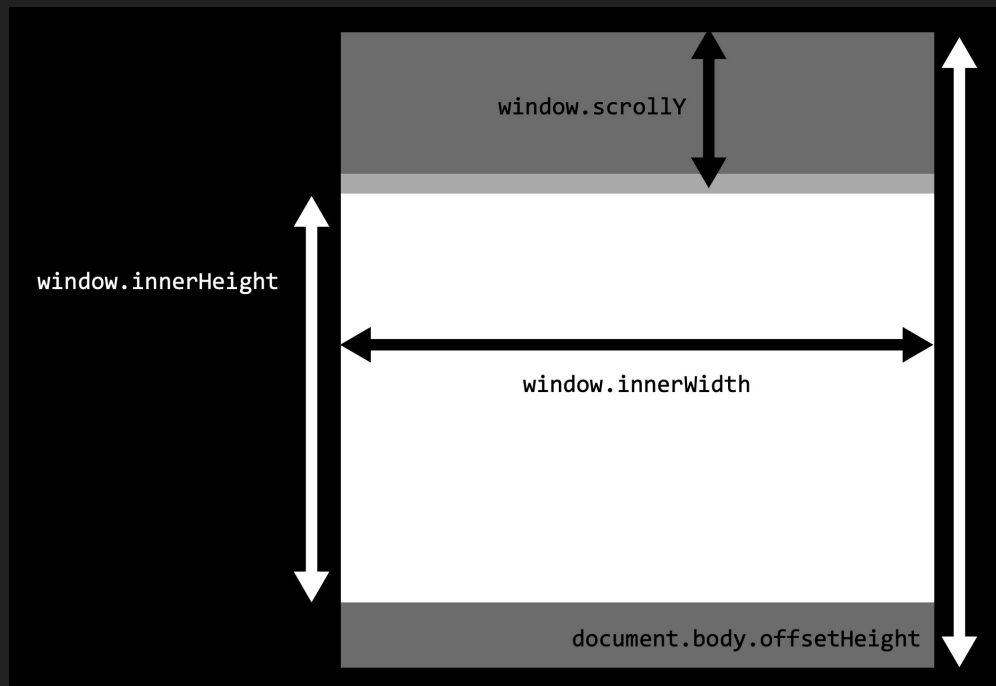
- We can use APIs to update the data associated with our site in JavaScript
- Sometimes, we'll have to write these APIs ourselves

# Using History

- Use `history.pushState` function to add to the browser history, and then set the `window.onpopstate` attribute to change behavior when back arrow pressed.
- `history.pushState(data, title, urlExtension);`
  - `data`: A JavaScript object with any information you would like to associate with the current state
  - `title`: Title of the state, ignored by most web browsers
  - `urlExtension`: What should be displayed in the url
- `window.onpopstate = myFunction(event);`
  - Access data using `event.state.parameterName`
  - This function will be run whenever the back arrow is pressed

# Infinite Scroll

- You can choose to load new data only when the user gets to a certain part of the page.



# Animation

- We can use some simple CSS to animate our page!
- First use @keyframes to create an animation
- Then apply animation in CSS

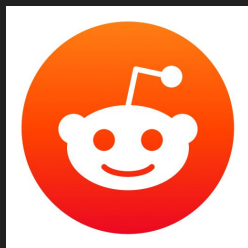
```
@keyframes animation_name {  
  0% {  
    /* Some styling for the start */  
  }  
  
  75% {  
    /* Some styling after 3/4 of animation */  
  }  
  
  100% {  
    /* Some styling for the end */  
  }  
}
```

```
h1 {  
  animation-name: grow;  
  animation-duration: 2s;  
  animation-fill-mode: forwards;  
}
```



# React

- Javascript Framework
- Uses Declarative Programming: stating the logic of what we wish to display without worrying about the precise content.
- Uses JSX, which combines HTML and JavaScript
- Some sites built with React:



# Including React

- Include within HTML:

- Link to React, React-DOM, and Babel in HTML page

```
<script src="https://unpkg.com/react@16/umd/react.development.js" crossorigin></script>  
<script src="https://unpkg.com/react-dom@16/umd/react-dom.development.js" crossorigin></script>  
<script src="https://unpkg.com/babel-standalone@6/babel.min.js"></script>
```

- Write your components in a script tag.
- Render component:

```
ReactDOM.render(<App />, document.querySelector("#app"));
```

- A bit slow, as we need to load these libraries each time

- create-react-app

- Created by Facebook for React Developers
- Automatically generates baseline react app

# Components

- Individual parts of a website like a nav-bar, a post, or even a page.
- Can be passed props (a bit like arguments) when rendered
- They can also store information in their state
  - When you wish to do this, you should include a constructor method to be run when the component is first created.
- Must include a render function that returns some HTML to render

# Components: Syntax

```
class App extends React.Component {  
  constructor(props) {  
    super(props);  
    // Initialize the state  
    this.state = {key: value; ...}  
  }  
  render() {  
    // You should only return one thing  
    return ( <div> {this.state.key} </div> );  
  }  
}
```

# Components: Changing State

```
class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {key: "value"};
  }

  render() {
    return ( <input onChange={this.updateKey} value={this.state.key}> </input> );
  }

  updateKey = (event) => {
    this.setState({
      key: event.target.value
    });
  }
}
```

# Using createreactapp

1. Install [Node.js](#).
2. In your terminal, run `npm create-react-app [app_name]`
3. cd into new app name folder
4. Edit App.js to change what is displayed
5. Run `npm start` in the terminal to run your application

Questions?

Demo



# Animation quick review

A few examples...

# Animation with D3.js

A few examples...

# Project

# Project 3

- Start early!!!
- Make a checklist of requirement and check all before submission
- Make sure there's no bugs (**especially those that don't happen all time**)
- Focus on functionality (NOT PRETTINESS)!!!

# Project 3

- Refactor functions to:
  - add email to the mailbox (render email: create element and append)
  - archive current email (and go back to inbox) - takes T/F
  - compose
  - reply email -> compose email and plug in some values
  - view single email
  - send email
- Styling the inbox entry:
  - CSS classes
  - `<span>` for timestamp
  - CSS child selector

# Project 3

- Reading mailbox (inbox/sent/archive)
  - on POST email add **`event.preventDefault();`**
  - Style your 'read' emails, e.g.:

```
if (email.read) {  
    row.classList.add('email-read');  
}
```

- Add event listeners to each email:

```
email.addEventListener('click', function() {  
    view_email(email.id);  
});
```

- When loading mailbox make sure to hide the single view as well

# Project 3

- Showing single email:
  - You can display button depending on which mailbox you are observing (or by fetching info)
  - Changing visibility →  
`myArchiveElement.style.display = if globalMailbox === 'inbox' ? 'inline-block' : 'none';`
  - When showing single email, flush all values before fetching, in case no data or partial data returned
  - Make sure to mark email as read when landing on the page
  - Breaking ``n`` with `<br>`: OR? `<pre></pre>`
    - `email.body.split("\n").forEach(line => {`
      - 1) insert line
      - 2) insert `<br>`
    - }

# Project 3

- Showing single email HTML:
  - separate entries for : From/To/Subject/Timestamp
  - buttons: reply/(archive|unarchive)
  - Also the div for body



# Project 3

- Archiving
  - Simple method that calls backend:
    - No global value you need to pass T/F and email id
- Reply:
  - If `subject.slice(0, 4) !== 'Re: '`  
    `subject += 'Re: '`
  - `.....element.value= `<br><br><br>----- <br> On ${email.timestamp}  
${email.sender} wrote:\n${curEmail.body}`;`
    - OR `\n\n\n`
  - My reply

-----

On \_\_\_\_ wrote: "Hello"

# Design

What can be considered (not exclusively):

- Proper refactoring (copy-paste is usually a no-no)
- Use of constants/vars:
  - 1. const
  - 2. let
  - ~~00000. var~~
- Proper use of functions
- More reasonable solution
- Code/file structure
- `fetch('/email/' + id)` -> `fetch(`/email/${id}`)`

# Design (continued)

What can be considered (not exclusively):

- Repetitive use of `querySelector`?
- Proper data structures
- `==` vs `===` ?
  - `const x = 5`
  - `const y = '5'`
  - `x == y -> T`
  - `x === y -> F`
- Code repetition

# Style

What can be considered (not exclusively):

- *jshint* (*indentations, line breaks, long lines*)
- COMMENTS!
- Naming for variable, function, files, etc.:
  - getemailbyid -> get\_email\_by\_id (Python convention)
  - getEmailById (JS convention)
- Consistency is the key!

# Style (continued)

What can be considered (not exclusively):

- ‘ vs “ consistency
- camelCase(c\*, Javascript, Java) vs snake\_case (Python)
- == vs ===

# jshint

- UI:
  - <https://jshint.com/>
- CLI:
  - brew update
  - brew doctor
  - brew install node
  - npm install -g jshint
  - In `~/.jshintrc` add:
    - {
    - "esversion": 6
    - }

# pycodestyle (formerly pep8)

- `python -m pip install pycodestyle`
- `pycodestyle app.py --max-line-length=120`

# pylint (checks beyond style, but including)

- `python -m pip install pylint`
- `pip install pylint-django`
- `pylint auction/views.py --load-plugins pylint_django`



# Chrome Developer Tools (Network)

In Chrome:

1. Right click
2. Inspect
3. → Demo

Extremely powerful! Let's try Javascript...

# cURL / Postman

Allows to call API endpoints directly.

Demo...

# Random Tips

- Video Speed Controller (Chrome Extension)
- Spotify + Hulu + Showtime => \$5
- GitHub Education Pack

- Windows licence (<https://harvard.onthehub.com>)
- Chrome Tab
- DSA
  - [Video by CS50: https://www.youtube.com/watch?v=QDQ1Ik5cQJk](https://www.youtube.com/watch?v=QDQ1Ik5cQJk) - channel CS50
  - LeetCode / AlgoExpert / Etc.
  - Stanford Algorithms Specialization (EdX link / Coursera) - more theory (time consuming)
  - e22 seems good!
  - $e20 + e124$  (combo) - HARD!
- System Design:
  - Grokking System Design
  - Alex Xu - System Design

# Resources

- <https://github.com/vpopil/e33a-sections-fall-2022>

# CSCI E-33a (Web50)

## Section 6

*Ref: Lecture 6 (User Interfaces)*

Vlad Popil

*Oct 26, 2022*