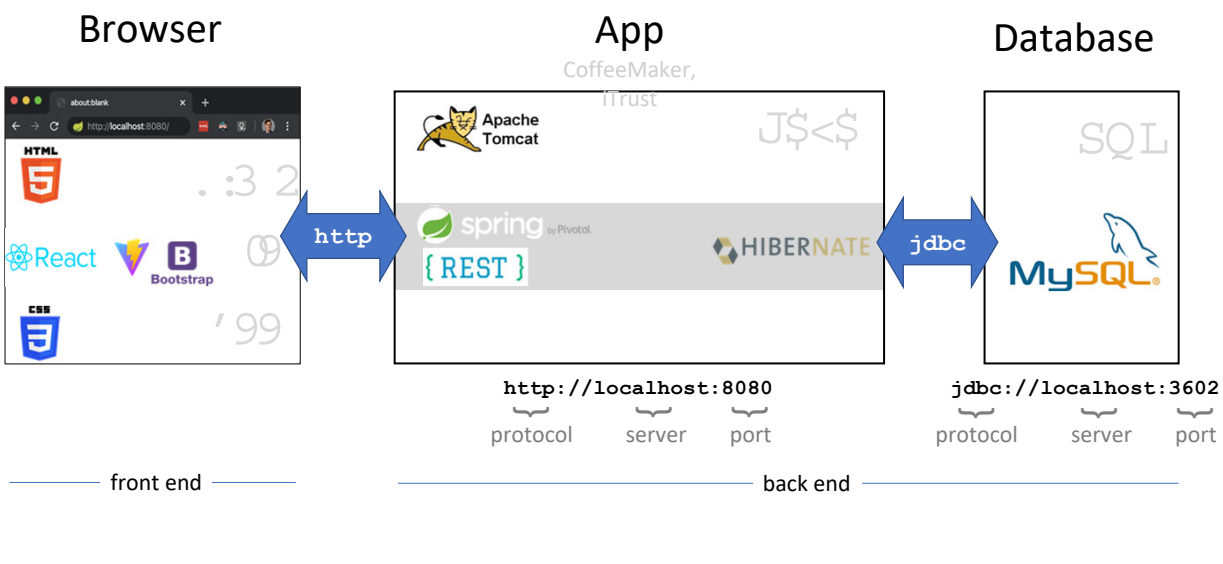
Frontend

CSC 326

© NC State Software Engineering Faculty

1

High Level 326 App Architecture



2

"What is a framework?"

FRAME – Always Inside



SCAFFOLD – Eventually Removed



DEV+

React

Apache Tomcat

{ REST }

spring by Pivotal

HIBERNATE

TEST

cucumber

Selenium

JUnit

BUILD

Maven

© NC State Software Engineering Faculty

HTML

Hypertext Markup Language

Problem: *I want to make a webpage. How do I tell the computer the document is a webpage, and how do I tell it where to put things?*

Solution: ***Use HTML, and use file extension*** `.html` ***or*** `.htm`

© NC State Software Engineering Faculty

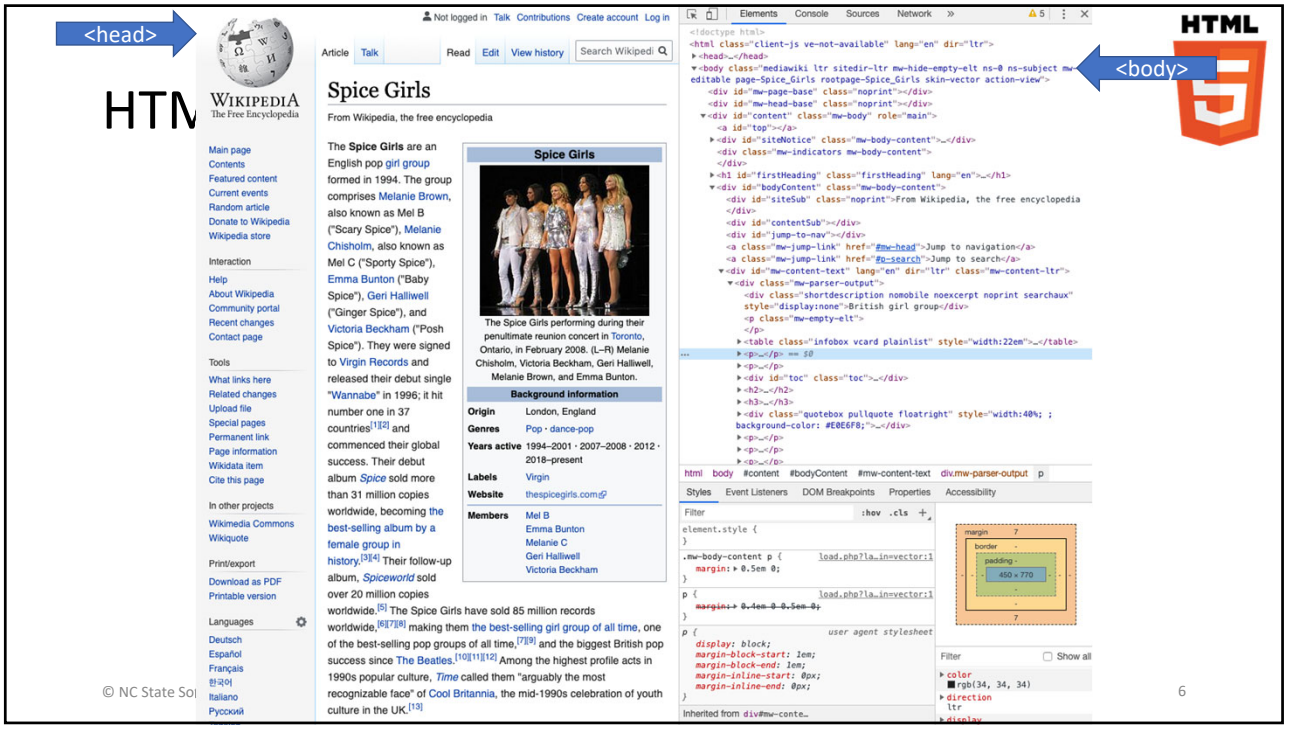
HTML: Hypertext Markup Language

- HTML has a <head>
  - Meta information about the page
  - Things like
    - Page title
    - CSS (we'll see this in a moment)
- And a <body>
  - Page content itself

```
<head>…</head>
<body>
  <table>
   <tbody>
     <tr>
       <td>CSC 216</td>
       <td>Sw. Dev. Fundamentals</td>
     </tr>
     <tr>
       <td>CSC 326</td>
       <td>Software Engineering</td>
     </tr>
   </tbody>
  </table>
</body>
```

© NC State Software Engineering Faculty

5



© NC State So

6



Cascading Style Sheets

Problem: *I want a consistent look and feel for my webpage, but I'm tired of modifying all the html tags for color, fonts, and spacing.*

Solution: ***Use CSS.***

© NC State Software Engineering Faculty 7

7



Cascading Style Sheets (CSS)

- CSS allows separation of content from style.

CSS
```
h1 { color: red; }
```

HTML
```
<h1>Text Colors</h1>
```

- A CSS rule has a **selector** and a **declaration.**

CSS
```
.redtext  { color: red; }
```

HTML
```
<span class="redtext">I am red</span>
```

selector — declaration (property: value pair)

- A selector matches a DOM element using a pattern.
- A declaration sets the element's property.

Reference:
https://www.w3schools.com/cssref/css_selectors.asp

8

8



JavaScript

a lightweight, interpreted, or just-in-time compiled programming language

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

Problem: *I want to display more than just static content on my webpage.*

Solution: ***Use JavaScript.***

© NC State Software Engineering Faculty

9



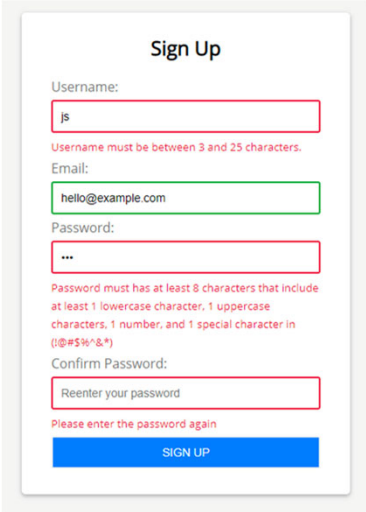
JavaScript Client-side Basics

- Store variables
- Perform operations on variables (e.g., add, concatenate)
- Run code in response to certain events (e.g., a click)

© NC State Software Engineering Faculty

10

Example - Javascript Form Validation



Validate some data at client-side, reduce the burden of server and much faster.

```
const checkUsername = () => {

    let valid = false;
    const min = 3, max = 25;
    const username = usernameEl.value.trim();

    if (!isRequired(username)) {
        showError(usernameEl, 'Username cannot be blank.');
    } else if (!isBetween(username.length, min, max)) {
        showError(usernameEl, `Username must be between ${min} and ${max} characters.`)
    } else {
        showSuccess(usernameEl);
        valid = true;
    }
    return valid;
}
```

https://www.javascripttutorial.net/javascript-dom/javascript-form-validation/ 11

11



When HTML, CSS, and JavaScript meet together…

© NC State Software Engineering Faculty 12

12

Putting it all together



© NC State Software Engineering Faculty

Adapted from: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript

13

Putting it all together



© NC State Software Engineering Faculty

Adapted from: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript

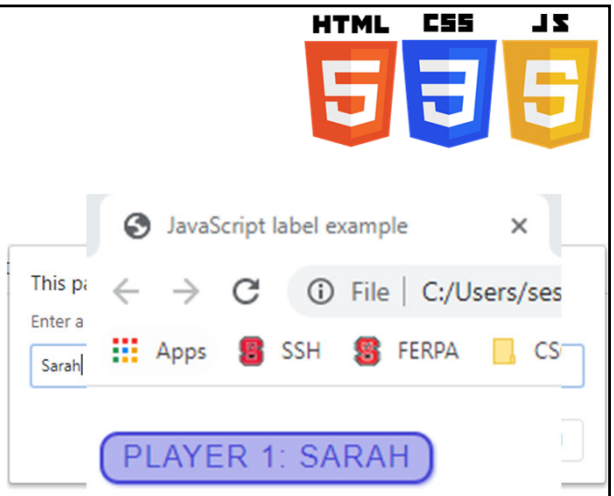
14

```
<!-- https://developer.mozilla.org/en-US/docs/Learn/JavaScript
<!DOCTYPE html>
<html lang="en-US">
  <head>
    <meta charset="utf-8">
    <title>JavaScript label example</title>
    <style>
      p {
        font-family: 'helvetica neue', helvetica, sans-serif;
        letter-spacing: 1px;
        text-transform: uppercase;
        text-align: center;
        border: 2px solid rgba(0,0,200,0.6);
        background: rgba(0,0,200,0.3);
        color: rgba(0,0,200,0.6);
        box-shadow: 1px 1px 2px rgba(0,0,200,0.4);
        border-radius: 10px;
        padding: 3px 10px;
        display: inline-block;
        cursor: pointer;
      }
    </style>
  </head>
  <body>
    <p>Player 1: Laura</p>

    <script>
      const para = document.querySelector('p');

      para.addEventListener('click', updateName);

      function updateName() {
        let name = prompt('Enter a new name');
        para.textContent = 'Player 1: ' + name;
      }
    </script>
  </body>
</html>
```

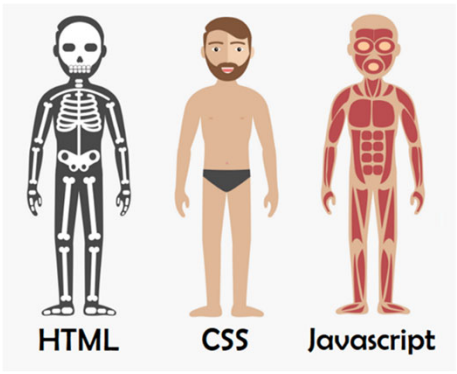


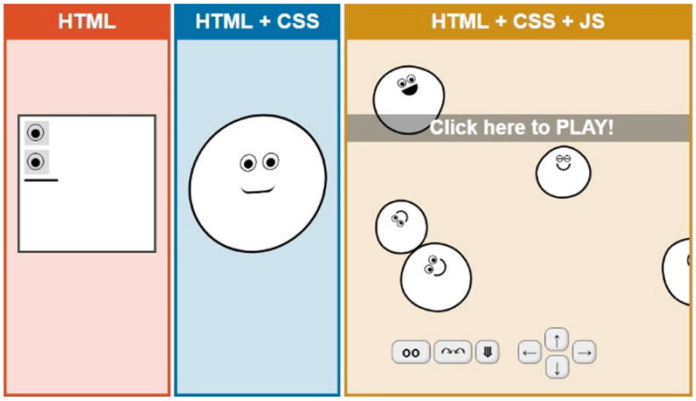
15

ted from: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript

15

Relationship between HTML, CSS, JS





16

16

Bootstrap

noun. a technique of loading a program into a computer by means of a few initial instructions which enable the introduction of the rest of the program from an input device. *Google Dictionary Definition Jan 10, 2020*

Problem: *I want to make a website that's pretty on my giant monitor and on my phone. I'm pretty savvy with if-statements, but that sounds inelegant.*

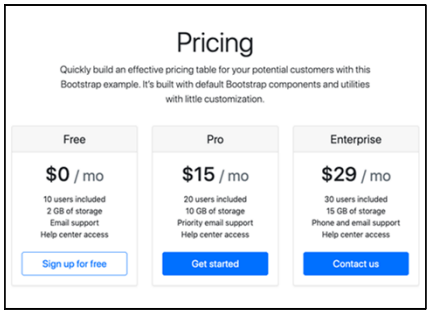
Solution: ***Use Bootstrap.***

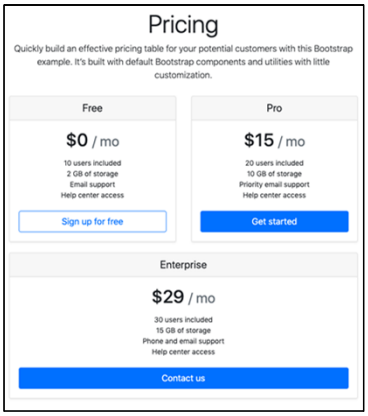
© NC State Software Engineering Faculty

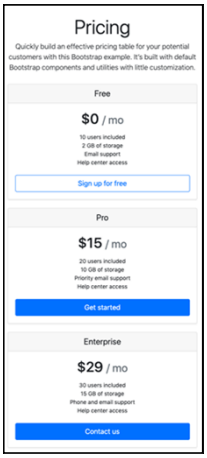
17

Bootstrap

- A library that supports responsive design and other operations
- Columns flow down when no room on screen



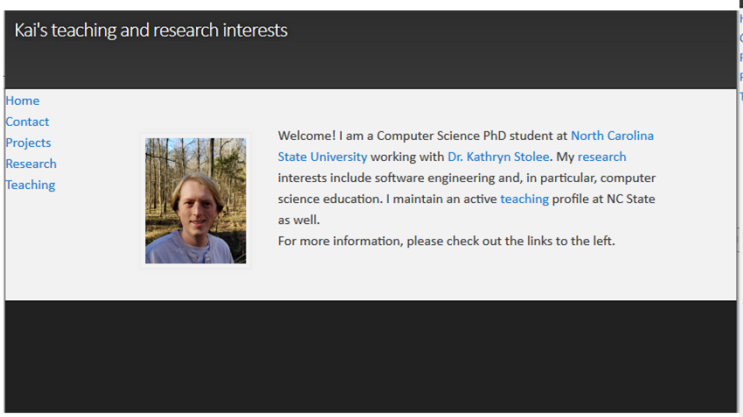




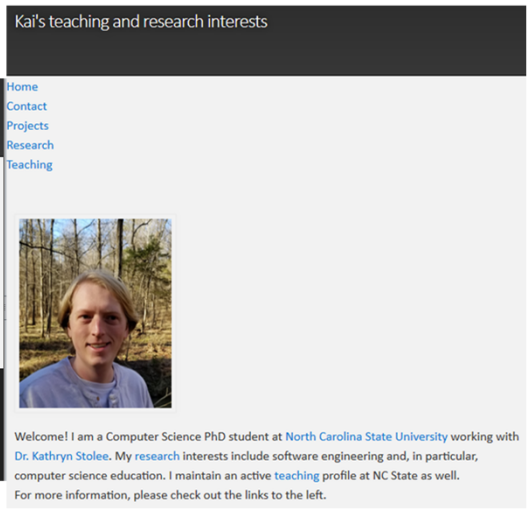
© NC State Software Engineering Faculty

18

Bootstrap in Action



Large-ish display



Smaller display

© NC State Software Engineering Faculty

React

Problem: *I want my webpage to have interactive and dynamic content*

Solution: ***Use React.***

© NC State Software Engineering Faculty

React



- Toolset for building dynamic web applications
- Works with JavaScript, HTML and CSS

- Combines standard form elements into ***components***
- Components are JavaScript functions - can include markup (like JSX as a stricter version of HTML)
  - Export components from one file for use in another file
  - Include JavaScript in JSX in {}
- Connections to the backend via ***services***

© NC State Software Engineering Faculty

Frontend in CoffeeMaker

Focus on Add Recipe Use Case

© NC State Software Engineering Faculty

Component Structure

index.html

```
<body>
  <div id="root"></div>
  <script type="module" src="/src/main.jsx"></script>
</body>
```

main.jsx

```
createRoot(document.getElementById('root')).render(
 <StrictMode>
   <App />
 </StrictMode>,
)
```

App.jsx

```
function App() {
   return (
   <>
   <BrowserRouter>
     <HeaderComponent />
     <Routes>
       <Route path='/' element = { <ListRecipesComponent /> }></Route>
       <Route path='/recipes' element = { <ListRecipesComponent /> }></Route>
       <Route path='/add-recipe' element = { <RecipeComponent /> }></Route>
       <Route path='/inventory' element = { <InventoryComponent /> }></Route>
       <Route path='/make-recipe' element = { <MakeRecipeComponent /> }></Route>
     </Routes>
     <FooterComponent />
   </BrowserRouter>
   </>
 )
}
export default App
```

23

23

```
import { useState } from 'react'
import { createRecipe } from '../services/RecipesService'
import { useNavigate } from 'react-router-dom'
/** Form to create a new recipe. */
const RecipeComponent = () => {
   const [name, setName] = useState("")
   const [price, setPrice] = useState("")
   const [coffee, setCoffee] = useState("")
   const [milk, setMilk] = useState("")
   const [sugar, setSugar] = useState("")
   const [chocolate, setChocolate] = useState("")
   const navigator = useNavigate()
   const [errors, setErrors] = useState({
       general: "",
       name: "",
       price: "",
       coffee: "",
       milk: "",
       sugar: "",
       chocolate: "",
   })

   [functions go here]

   return ( JSX goes here )

}

export default RecipeComponent
```

Imports
- **useState** from **react** libraries that let's us initialize state for the component
- **createRecipe** from **RecipesService** that connects to the backend
- **useNavigate** from **react-router-dom** library for moving between components

Defines the **RecipeComponent** function using a lambda expression. This **RecipeComponent** function returns the JSX page that the browser renders.

Defines the state, associated setter method, and initializes the values. Note that for **errors**, the state structure is JSON. We also create a **navigator** reference to work with moving between components.

We export the component for use in other components (like the App component, which handles routing to this page).

24

24

```
function saveRecipe(e) {
    e.preventDefault();
    if (validateForm()) {
        const recipe = {name, price, coffee, milk, sugar, chocolate}
        console.log(recipe)
        createRecipe(recipe).then((response) => {
            console.log(response.data)
            navigator("/recipes")
        }).catch(error => {
            console.error(error)
            const errorsCopy = {... errors}
            if (error.response.status == 507) {
                errorsCopy.general = "Recipe list is at capacity."
            }
            if (error.response.status == 409) {
                errorsCopy.general = "Duplicate recipe name."
            }
            setErrors(errorsCopy)
        })
    }
}
function validateForm() {
    let valid = true

    const errorsCopy = {... errors}
    // Cut for brevity, but it's really weird there's only error checking on name!
    setErrors(errorsCopy)
    return valid
}
function getGeneralErrors() {
    if (errors.general) {
        return <div className="p-3 mb-2 bg-danger text-white">{errors.general}</div>
    }
}
```

Prevents a browser reload/refresh

If everything looks good in the form…

Call the service (which calls the backend)

Navigate to the recipes list page

Catch any errors from the backend

Update error messages for the user

Checks the form elements for errors

Starts with a copy of the errors variable

Updates the errors variables with any info

Helper function to add error messages to UI

25

25

```
return (
    <div className="container">
        <br /><br />
        <div className="row">
            <div className="card col-md-6 offset-md-3">
                <h2 className="text-center">Add Recipe</h2>
                <div className="card-body">
                    { getGeneralErrors() }
                    <form>
                        <div className="form-group mb-2">
                            <label className="form-label">Recipe Name</label>
                            <input
                                type="text"
                                name="recipeName"
                                placeholder="Enter Recipe Name"
                                value={name}
                                onChange={(e) => setName(e.target.value)}
                                className={`form-control ${errors.name ? "is-invalid":""}`}
                            >
                            </input>
                            {errors.name && <div className="invalid-feedback">{errors.name}</div>}
                        </div>
                        <!-- Leaving out other items for brevity →


                        <button className="btn btn-success" onClick={(e) => saveRecipe(e)}>Submit</button>
                    </form>
                </div>
            </div>
        </div>
    </div>
)
```

Group elements by **divs**. Use **className** to add CSS style

Bootstrap provides lots of default options for layout

Calling the **getGeneralErrors()** function

Creating a form.

Sets up the **label** and text field (**input**) for the form.

Fills the form with the value in the variable **name**

Calls the **setName()** function with the value from the form (**e**)

Handles errors for the form

Displays any errors

Connects the button click to the appropriate function (**saveRecipe(e)**)

26

26

```
import axios from "axios"

/** Base URL for the Recipe API - Correspond to methods in Backend's Recipe Controller. */
const REST_API_BASE_URL = "http://localhost:8080/api/recipes"

/** GET Recipes - lists all recipes */
export const listRecipes = () => axios.get(REST_API_BASE_URL)

/** POST Recipe - creates a new recipe */
export const createRecipe = (recipe) => axios.post(REST_API_BASE_URL, recipe)

/** GET Recipe - gets a single recipe by id */
export const getRecipe = (id) => axios.get(REST_API_BASE_URL + "/" + id)

/** DELETE Recipe - deletes the recipe with the given id */
export const deleteRecipe = (id) => axios.delete(REST_API_BASE_URL + "/" + id)
```

Using axios libraries to make REST API calls to backend

Uses HTTP verbs as methods

Body information is lambda parameter that is passed to method

Path variables are lambda parameters concatenated to the URL

NOTE: The service function name cannot be the same as function names in the component! Weird shadowing occurs!

Component: saveRecipe()
Service: createRecipe()

27

27