Building User Interfaces

Javascript

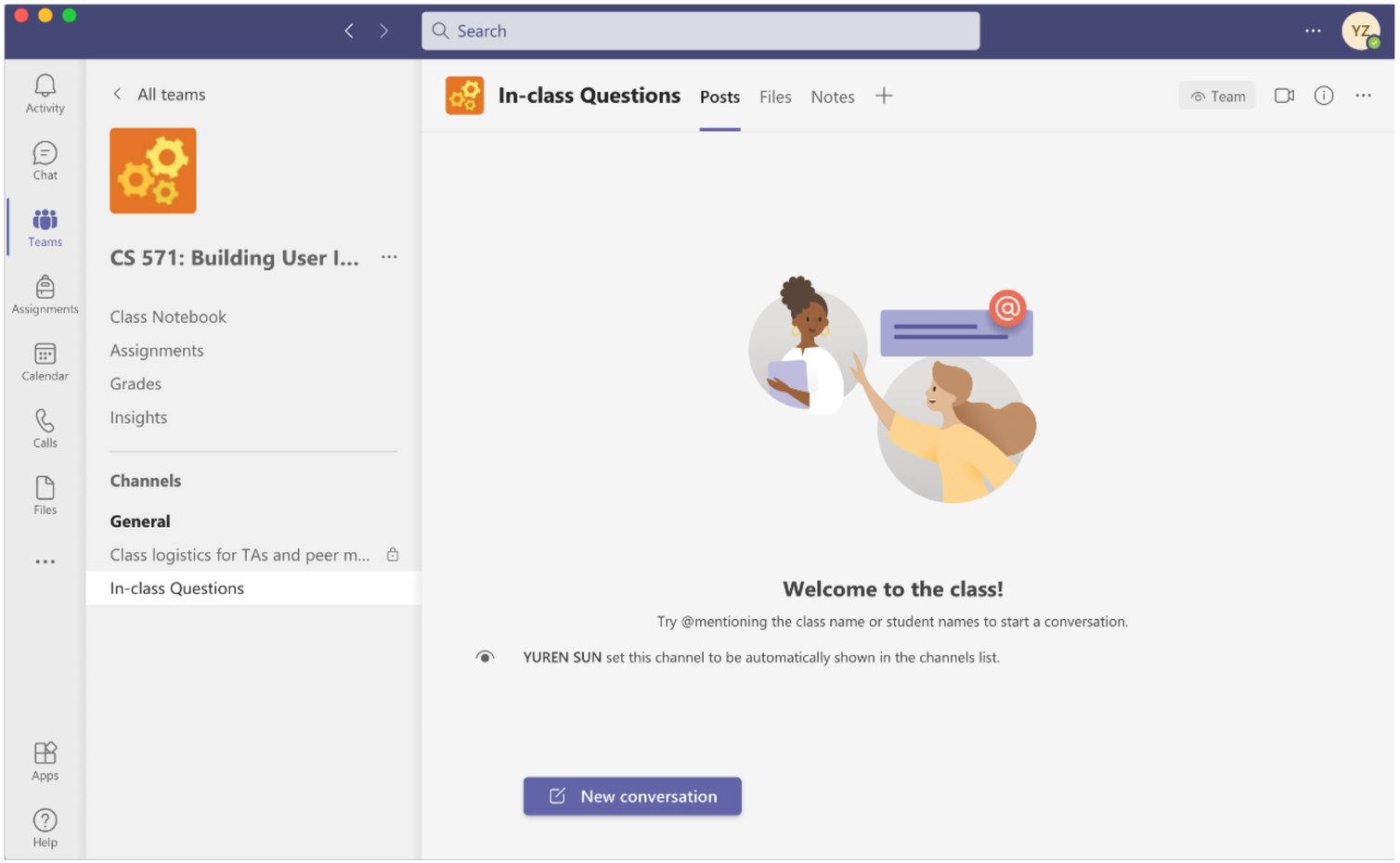
Intermediate Concepts

Professor Yuhang Zhao

What we will learn today?

— Working with JSON data
— , CSS/No-CSS
— Working with APIs
— Working with component libraries

Live Q&A Reminder



Working with JSON data

What is JSON?

Definition: JavaScript Object Notation (JSON) is a structured way to represent text-based data based on JS object syntax.

JSON can include any JS data type. Do you remember how many types there are?

```
{ string : value, .......}
```

Refresher: JS Objects

Definition: Objects are unordered collection of related data of primitive or reference types.

Object elements are defined using `key: value` statements.

```
var instructor = {
    firstName: "Yuhang",
    lastName: "Zhao",
    gender: "female"
}
instructor;
> {firstName: "Yuhang", lastName: "Zhao", gender: "female"}
```

JSON Objects:

```
{ "firstName": "Yuhang",
  "lastName": "Zhao",
  "role": "instructor",
  "email": "yuhang.zhao@cs.wisc.edu" }
```

JSON Arrays:

```
{ "TAs" : [
    { "Name": "Brandon" , "Year": "First" },
    { "Name": "Sujitha" , "Year": "First" },
    { "Name": "Salman" , "Year": "First" }]}
```

How to use JSON data[1]

```
<p id="TANames"></p>

var text = '{ "TAs": [' +
  '{ "Name":"Brandon Cegelski" , "Year":"First" },' +
  '{ "Name":"Sujitha Perumal" , "Year":"First" },' +
    '{ "Name":"Salman Munaf" , "Year":"First" }]}';

obj = JSON.parse(text);

document.getElementById("TANames").innerHTML =
  "Our TAs are " + obj.TAs[0].Name +
  " and " + obj.TAs[1].Name + ".";
```

[1] See a working example in CodePen

How to request JSON from a server[2]

— Requests can be `synchronous` or `asynchronous`.

— `asynchronous` requests are recommended as they produce a *callback* when the data is received and lets the browser continue its work while the request is made.

[2] More on Synchronous/asynchronous Requests

Slight Detour: Callback Functions[3]

Definition: A *callback function* is passed into another function as an argument, which is then invoked inside the outer function to complete a routine or action.

```
function greeting(name) {
  alert('Hello ' + name);
}

function processUserInput(callback) {
  var name = prompt('Please enter your name.');
  callback(name);
}

processUserInput(greeting);
```

[3] More on callback Functions

Methods for Asynchronous Requests

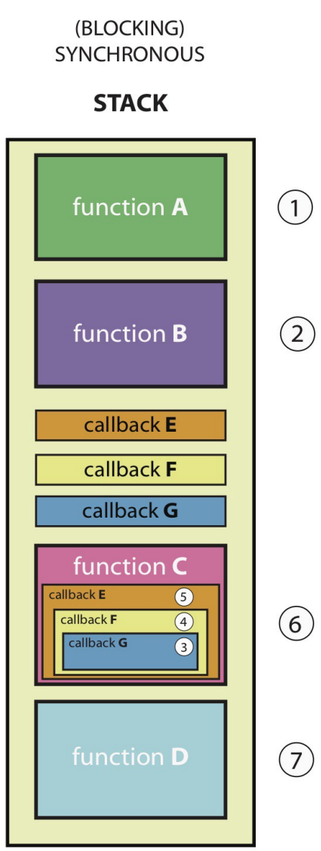
Two key methods: `XMLHttpRequest()` (old) and `fetch()` (new)

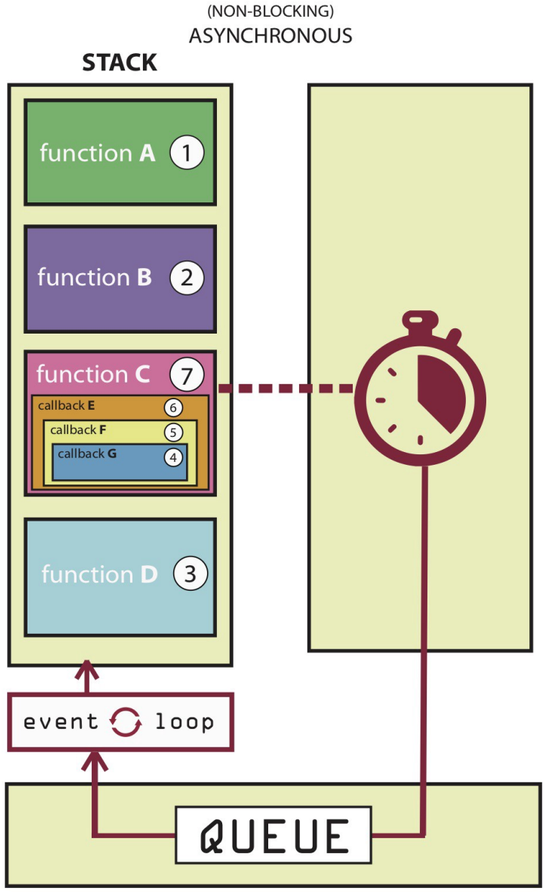
Pro Tip: `fetch()` is a `Promise`-based method.

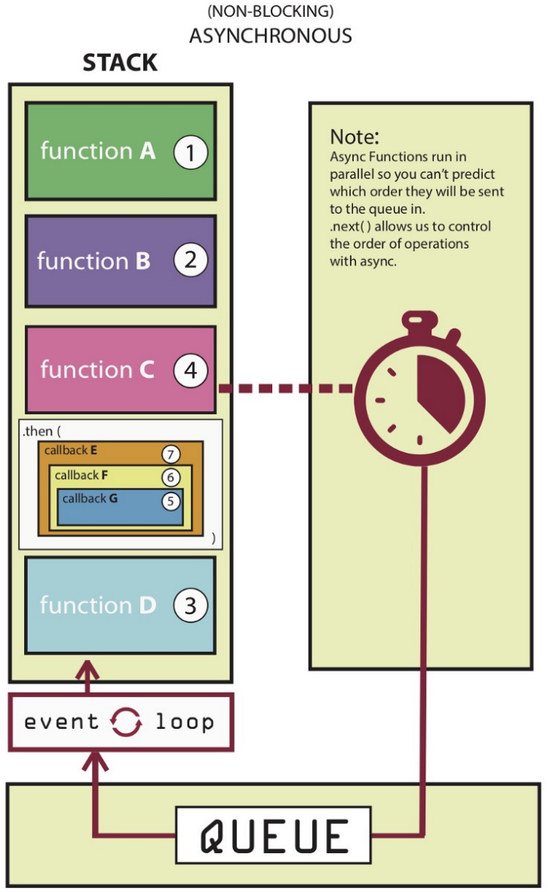
— `Promise` objects represent the eventual completion/failure of an *asynchronous* operation and its resulting value.

— `async` / `await` — keywords to indicate that a function is *asynchronous* 👈 preferred method

— We'll cover these in-depth in React.


(BLOCKING)
SYNCHRONOUS
STACK
function A
1
function B
2
callback E
callback F
callback G
function C
callback E
5
callback F
4
callback G
3
6
function D
7


(NON-BLOCKING)
ASYNCHRONOUS
STACK
function A
1
function B
2
function C
7
callback E
6
callback F
5
callback G
4
function D
3
event loop
QUEUE


(NON-BLOCKING)
ASYNCHRONOUS
STACK
function A
1
function B
2
function C
4
.then (
callback E
7
callback F
6
callback G
5
)
function D
3
event loop
QUEUE
Note:
Async Functions run in parallel so you can't predict which order they will be sent to the queue in.
.next( ) allows us to control the order of operations with async.

[4] Source

XMLHttpRequest()[5]

```
var requestURL = 'tas.json';
var request = new XMLHttpRequest();
request.open('GET', requestURL, true); // true for asynchronous
request.responseType = 'json';

request.onload = function() {
    // do something with the response.
}
request.send();
```

[5] See a working example in CodePen

fetch()[6]

```
fetch(url)
  .then(response => response.json())
  .then(data => {
    // Do something with the data
  })
  .catch(error => console.error(error)) // Print errors
```

[6] See a working example in CodePen

Back to JSON: `parse` and `stringify`

`parse()` takes a JSON string and returns JS objects.

```
var tas = JSON.parse(request.response);
```

`stringify()` takes a JS object and returns JSON string.

```
var tas = { "name": "Chris", "age": "38" };
var tasJSON = JSON.stringify(tas);
```

Accessing JS objects from JSON data

```
{ "firstName": "Brandon", "lastName": "Cegelski",
"role": "TA", "email":"bmcegelski@wisc.edu" }
```

```
var myTA = JSON.parse(request.response);
console.log(myTA.firstName);
console.log(myTA["firstName"]);
```

Using JS to render content

DOM Container

Definition: `<div>` defines a "division" or a section in an HTML document. You can place `<div>`s anywhere on the page and as many as you like. They will serve as canvases to manipulate using JS/React.

Prototype declaration:

```
<div id="name"></div>
```

CSS[7]

Consider the following button:

```
<button id="button">Submit</button>
```

We can use CSS to style it:

```
button {
  background-color: #008CBA;
  border: none;
  color: white;
  padding: 15px 32px;
  font-size: 16px; }
```

[7] See live at CodePen

No CSS[8]

Consider the same button:

```
<button id="button">Submit</button>
```

We can also style it using JS:

```
document.getElementById("button").style.color = "white";
document.getElementById("button").style.padding = "15px 32px";
document.getElementById("button").style.border = "none";
document.getElementById("button").style["background-color"] = "#008CBA";
document.getElementById("button").style["font-size"] = "16px";
```

[8] See live at CodePen

Working with APIs

What are APIs for Web Development?

Definition: Application Programming Interfaces (APIs) are constructs that facilitate the programming of complex functionality.

APIs abstract away the low-level implementation of tools and services and provide the programmer with easier syntax.

How do APIs work?

Browser APIs (e.g., fullscreen API, screen orientation API, vibration API), vs. **third-party APIs** (e.g., Google Maps API, Twitter API).

JS interacts with APIs over JS objects.

An Example [9]

Play an `mp3` file using the *Audio API*:

1. Create the audio and control elements — HTML
2. Create an *audio context* — JS
3. Create an audio element — JS
4. Control the element — JS

[9] See live at CodePen

Step 1: Create elements

```
<audio src="Haydn_Adagio.mp3" type="audio/mpeg"></audio>
<button data-playing="false" role="switch" aria-checked="true">
    <span>Play | Pause</span>
</button>
```

Step 2: Create an audio context

```
const audioContext = new AudioContext();
```

Step 3: Create an audio element

```
const audioElement = document.querySelector('audio');

const track = audioContext.createMediaElementSource(audioElement);

track.connect(audioContext.destination);
```

Step 4: Control the element

```
playButton.addEventListener('click', function() {
    if (audioContext.state === 'suspended') { audioContext.resume();}
    if (this.dataset.playing === 'false') {
        audioElement.play();
        this.dataset.playing = 'true';
        console.log("Playing...");
    } else if (this.dataset.playing === 'true') {
        audioElement.pause();
        this.dataset.playing = 'false';
        console.log("Stopped..."); }
}, false);

audioElement.addEventListener('ended', () => {
    playButton.dataset.playing = 'false';
}, false);
```

Working with Component Libraries

What are Component Libraries?[11]

Definition: Software libraries that abstract away the low-level CSS implementation of user-facing elements.

Some popular libraries:
* Bootstrap
* Foundation
* Semantic UI
* Pure
* UIkit

[11] A comparison of the frameworks

Bootstrap

— Download for offline development

```
$ npm install bootstrap
```

— BootstrapCDN (Content Delivery Network)

```
<link
    rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"
    integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T"
    crossorigin="anonymous">
<script
    src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"
    integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM"
    crossorigin="anonymous">
</script>
```

How Bootstrap Works

Main categories of HTML specification:
* Layouts
* Content
* Components
* Utilities

There is much more!

Bootstrap Categories: Layouts

— **Containers** are the most basic element of layouts.

  — *Responsive, fixed-width, fluid-width.*

```
<div class="container">
  ...
</div>

<div class="container-fluid">
  ...
</div>
```

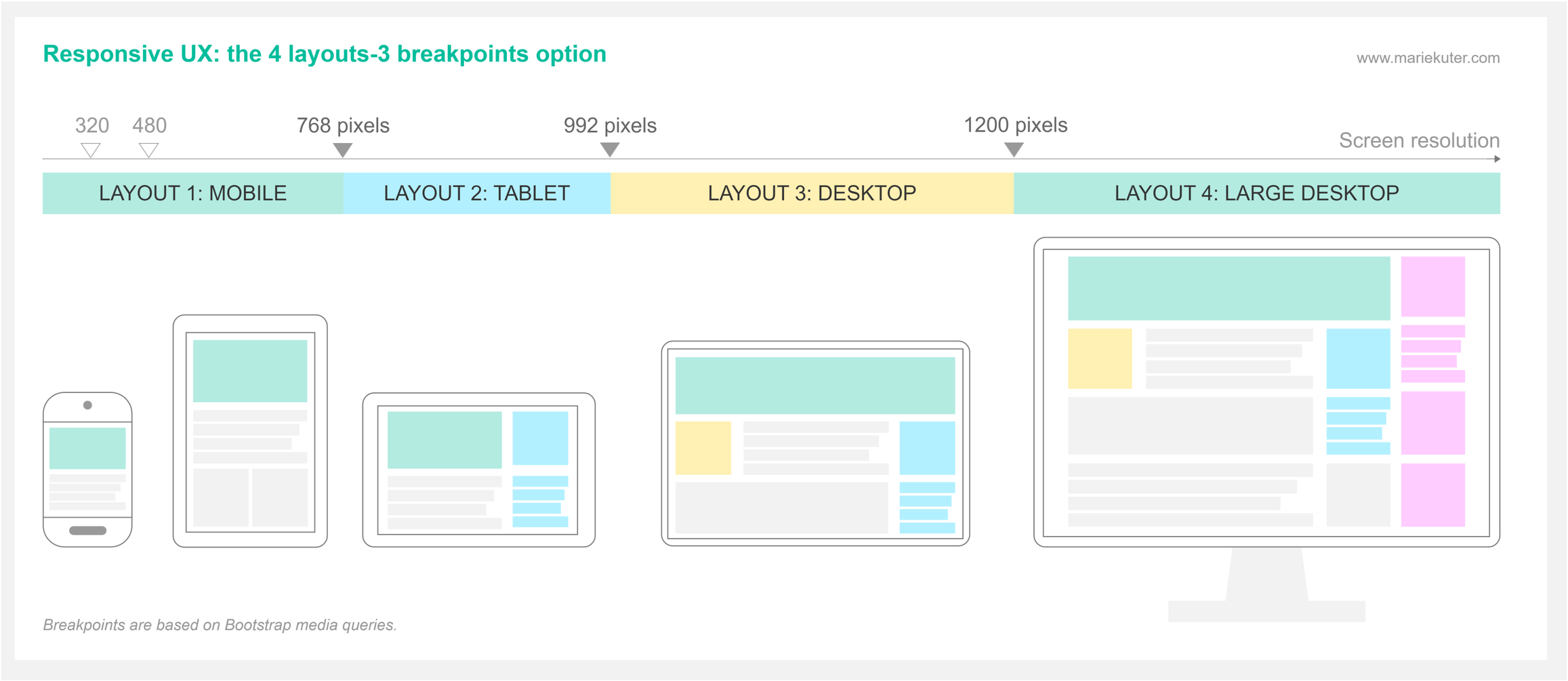
Layouts: Responsive Design[12]

Definition: Responsive web design (RWD) is an approach that adapts web content to a variety of devices and window or screen sizes.[13]

Width breakpoints determine whether the design will scale or be reorganized.

[12] Wikipedia: Responsive Web Design

[13] Image Source: InVision



[14] Image Source: Marie Kuter

How does Bootstrap do this?[15]

```
// Extra small devices (portrait phones, less than 576px)
// No media query for `xs` since this is the default in Bootstrap

// Small devices (landscape phones, 576px and up)
@media (min-width: 576px) { ... }

// Medium devices (tablets, 768px and up)
@media (min-width: 768px) { ... }

// Large devices (desktops, 992px and up)
@media (min-width: 992px) { ... }

// Extra large devices (large desktops, 1200px and up)
@media (min-width: 1200px) { ... }
```

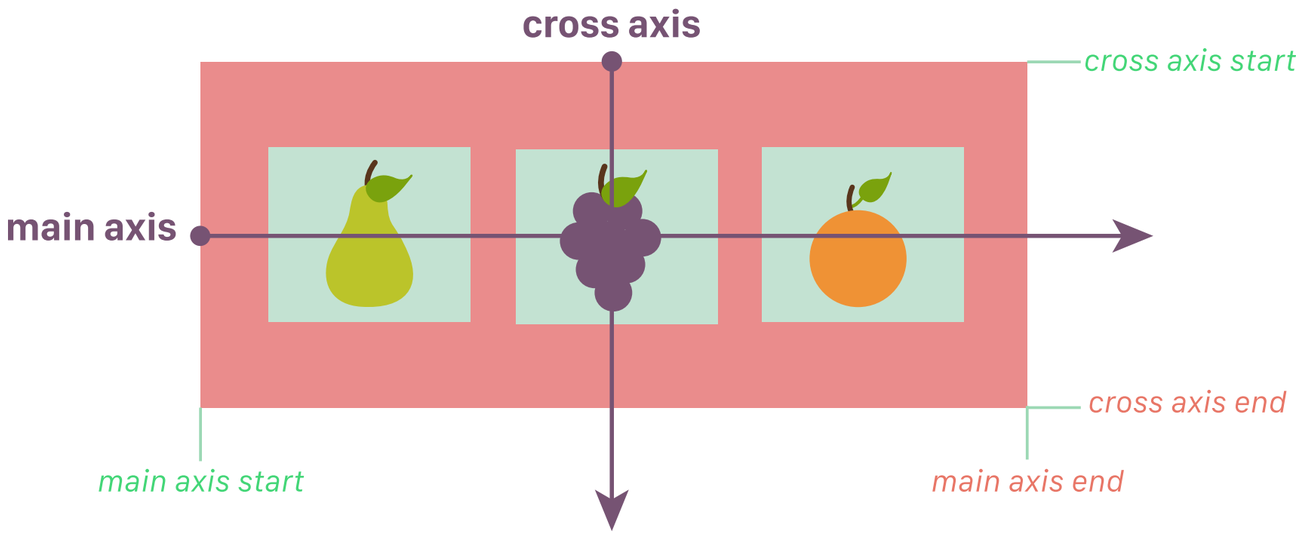
[15] Bootstrap Layout Overview

Detour: Responsive Layouts using CSS Flexbox

Definition: A CSS layout mode for responsive content.[16] [17] [18]

```
.flex-container {
    display: flex;
}
```

```
<div class="flex-container">
  <div>Content A</div>
  <div>Content B</div>
  <div>Content C</div>
</div>
```



[16] Excellent Flexbox Cheatsheet

[17] See example on CodePen

[18] Image source

Layouts: Grids

Basic usage:

```
<div class="row">
  <div class="col-*-*"></div>
  <div class="col-*-*"></div>
</div>
```

Where the first * is *grid class*.

The Bootstrap grid system classes:[19]

	Extra small <576px	Small ≥576px	Medium ≥768px	Large ≥992px	Extra large ≥1200px
Max container width	None (auto)	540px	720px	960px	1140px
Class prefix	`.col-`	`.col-sm-`	`.col-md-`	`.col-lg-`	`.col-xl-`
# of columns	12				
Gutter width	30px (15px on each side of a column)				
Nestable	Yes				
Column ordering	Yes				

[19] Bootsrap grid

Second * is the number of grid columns (max = 12). [20] [21]


```
<div class="row">
  <div class="col-sm-4">.col-sm-4</div>
  <div class="col-sm-4">.col-sm-4</div>
  <div class="col-sm-4">.col-sm-4</div>
</div>
```

[20] W3 Schools: Bootstrap

[21] See in CodePen

Bootstrap Categories: Content

Content styling includes basic HTML elements, typography, code, images, tables, figures.

Basic HTML examples:

```
<h1></h1>
<ul></ul>
<input></input>
<button></button>
```

Note the possibility of using, e.g., `<h1>` and `class="h1"`.

Styling of other elements

```
<img src="..." class="img-fluid">

<table class="table">
  <thead class="thead-dark">
    <tr>
      <th scope="col">...</th>
      ...

<div class="table-responsive-sm">
  <table class="table">
  ...
```

Bootstrap Categories: Components

Components include all other visual/interactive elements that make up the design, e.g., buttons, forms, navbar, tooltips, etc.

```
<button type="button" class="btn btn-primary">Fill button</button>

<button type="button" class="btn btn-outline-primary">Outline button</button>

<div class="btn-group-toggle" data-toggle="buttons">
  <label class="btn btn-secondary active">
    <input type="checkbox" checked autocomplete="off"> Switch
  </label>
</div>
```

Bootstrap Categories: Utilities

Utilities are not elements themselves, but they modify/control other elements, e.g., adding rounded corners to an image.

```
<img src="..." class="rounded">
```

```
<div class="shadow p-3 mb-5 bg-white rounded">Shadow</div>
```

Example HomePage[22]

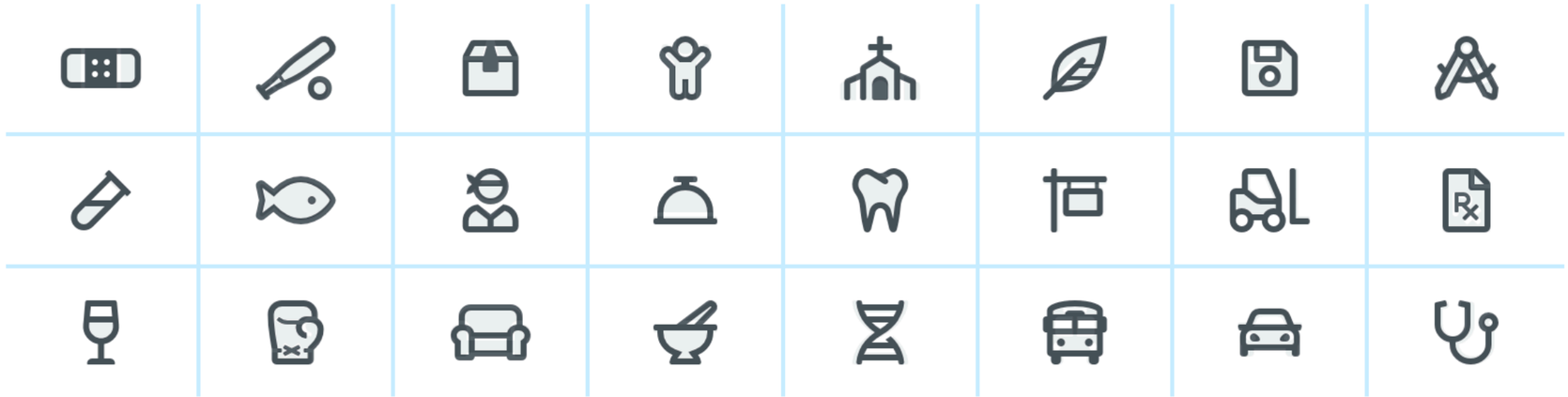
[22] See in CodePen

Additional Resources

— Bootstrap documentation
— Tutorial Republic
— W3 Schools

Assets

Asset libraries, e.g., icons, are usually used in conjunction with frameworks such as Bootstrap.[23] [24]



[23] Icon libraries

[24] Image source

What we learned today

— Working with JSON data
— , CSS/No-CSS
— Working with APIs
— Working with component libraries

Assignment

Javascript α released — due next week, Friday

— Implement the functionality supporting **Badger Shop**

— In Javascript β, to be released next Monday, we will improve on the visual design