

Welcome Developers!

Monday, December 27, 2021 9:58 AM

*“Any fool can write code that a computer can understand.
Good programmers write code that humans can understand.”
– Martin Fowler*

Overview

Monday, December 27, 2021 10:12 AM

Git Hub Account: <https://github.com/vivekduttamishra/202112-Mobileum>

Participant Information Form: https://docs.google.com/forms/d/e/1FAIpQLSdICorctSNMFnLO961EH-tpWpQkTR5HNOPjEw9RWUQcoRGaYQ/viewform?usp=sf_link

Share view permission to : **vivek@conceptarchitect.in**

References!

Wednesday, December 29, 2021

10:38 AM

Code Sandbox Links

Thursday, January 13, 2022 12:32 PM

State and Parent Child Communication

<https://codesandbox.io/s/elastic-shadow-4lgtg?file=/src/App.js>

A List of books using normal for loop

<https://codesandbox.io/s/book-list-using-loops-839no?file=/src/App.js>

A List of books using array.map (V1)

<https://codesandbox.io/s/book-list-using-map-v1-dyspo?file=/src/App.js>

A List books using array.map and key props

<https://codesandbox.io/s/book-list-using-map-v2-img60?file=/src/App.js>

useState example 1

<https://codesandbox.io/s/state-hook-test-95h0e?file=/src/App.js>

Use State example 2 with Array Destructuring

<https://codesandbox.io/s/state-hook-test-2-wih8v?file=/src/App.js>

Traditional Parent Child Data Flow

<https://codesandbox.io/s/traditional-parent-child-data-flow-v0mf3>

User context with Provider and consumer

<https://codesandbox.io/s/using-context-tg5zx?file=/src/App.js>

User Context with contextHook

<https://codesandbox.io/s/using-context-usecontext-hook-omc3j?file=/src/App.js>

User Context with State

<https://codesandbox.io/s/context-with-state-e9m9f?file=/src/App.js>

Simple Reducer Code

<https://codesandbox.io/s/simple-reducer-demo-n1goj?file=/src/App.js>

Reducer Context Demo V1

<https://codesandbox.io/s/context-with-reducer-v1-97k62?file=/src/user-context.js>

Full Reducer Context Action Creator Example

<https://codesandbox.io/s/context-with-reducer-v2-tzbwn?file=/src/App.js:1955-2001>

<https://codesandbox.io/s/context-with-reducer-v2-tzbwn>

VSCode Snippets

Saturday, January 15, 2022 2:52 PM

Add them to

Files —>settings —> Code Snippet —> Javascript.json

```
"react-function-component":{  
    "prefix":"rfcomp",  
    "body": [  
        "import React from 'react';",  
        "",  
        "const $1=({})=>{",  
        "\t//TODO: Initialize Here",  
        "\t$3",  
        "",  
        "\treturn (",  
        "\t\t<div className='${2:$1}'>",  
        "\t\t\t<h2>Book List</h2>",  
        "\t\t</div>",  
        "\t);",  
        "}",  
        "",  
        "export default $1;"  
    ]  
}
```

```
"Import From":{  
    "prefix":"impf",  
    "body": [  
        "import {$2} from '$1';",  
        "$3"  
    ]  
},  
"Import Default":{  
    "prefix":"impd",  
    "body": [  
        "import $1 from '${2:$1}';",  
        "$3"  
    ]  
}
```

CSS References

Saturday, January 15, 2022 4:50 PM

CSS Selector

<http://flukeout.github.io/>

For Flex Layout

<https://flexboxfroggy.com>

<http://www.flexboxdefense.com>

Grid Layout

<https://cssgridgarden.com/>

Test

<https://www.guess-css.app>

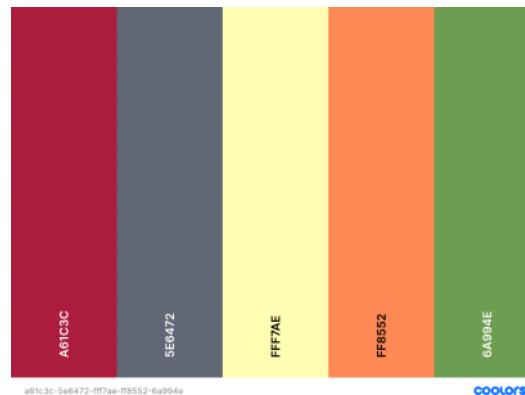
<https://css-challenges.com>

Color Pallettes

Sunday, January 16, 2022 12:37 PM

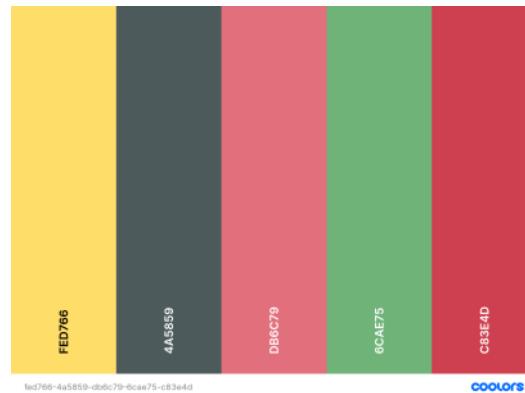
<https://color.co/a61c3c-5e6472-fff7ae-ff8552-6a994e>

```
$vivid-burgundy: #a61c3cff;  
$black-coral: #5e6472ff;  
$green-yellow-crayola: #fff7aeff;  
$coral: #ff8552ff;  
$may-green: #6a994eff;
```



<https://color.co/fed766-4a5859-db6c79-6cae75-c83e4d>

```
$orange-yellow-crayola: #fed766ff;  
$feldgrau: #4a5859ff;  
$candy-pink: #db6c79ff;  
$forest-green-crayola: #6cae75ff;  
$brick-red: #c83e4dff;
```



<https://color.co/bee6ce-bcffdb-8dffcd-68d89b-4f9d69>

<https://color.co/d3c1c3-e2d0be-eee5bf-e8f8c1-d1ffc6>

<https://color.co/91a6ff-ff88dc-faff7f-ffffff-ff5154>

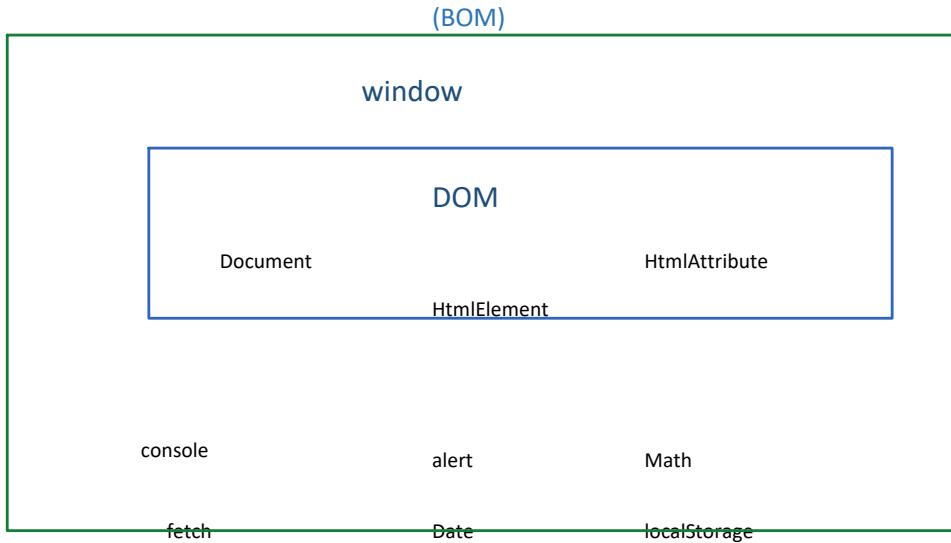
<https://color.co/b89c02-d6be33-c8cbdb-f7e474-918a5e>

<https://color.co/0c79ab-169ddb-0d5f85-acd2e3-5db9e3>

<https://color.co/384d48-08b2e3-acad94-f1c8db-e43f6f>

Javascript Browser Libraries

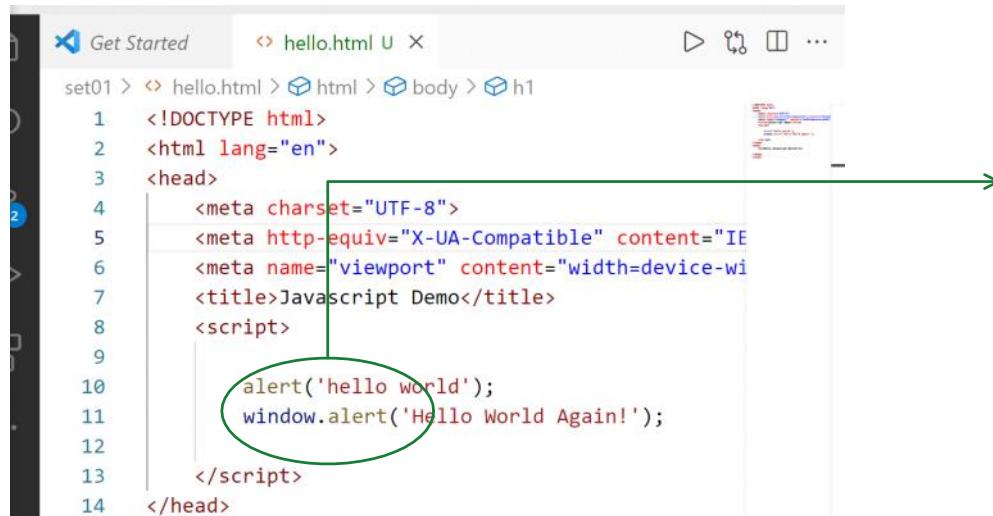
Monday, December 27, 2021 12:21 PM



- Browser of BOM is represented as `window` object
 - You can think of BOM and `window` synonymous.
- Every other object belongs to `window`
- `window` object holds every other object.

Basic Javascript code model

Monday, December 27, 2021 12:35 PM

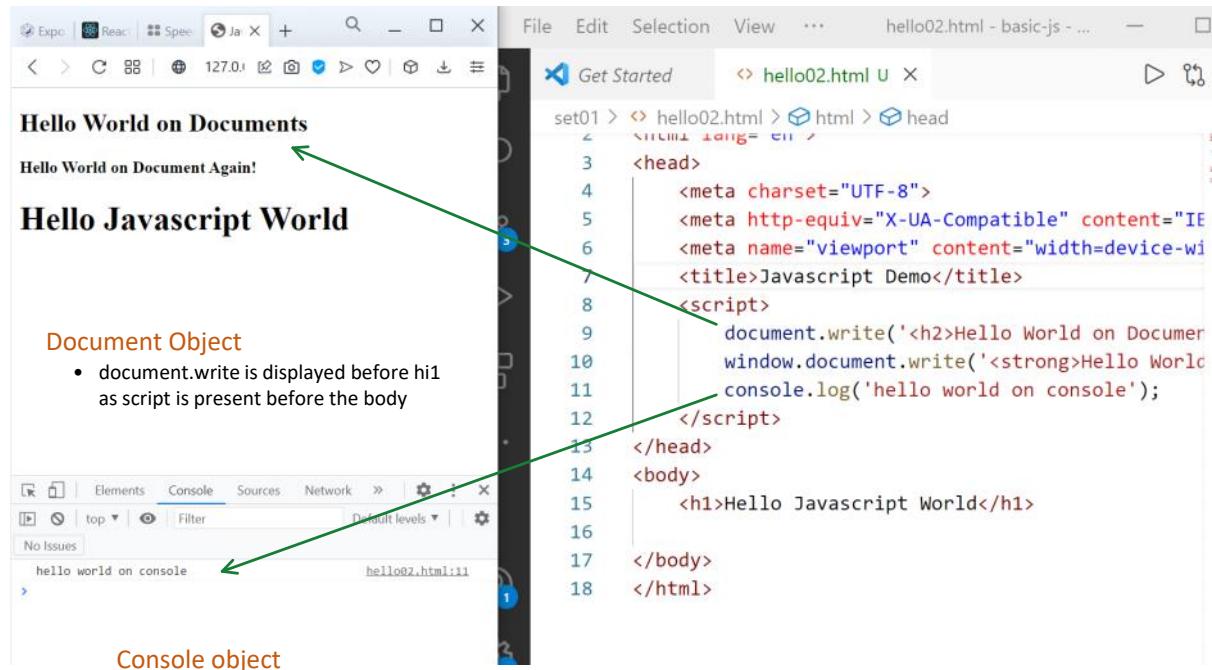


```
Get Started < hello.html U X ...
```

```
set01 > < hello.html > html > body > h1
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta http-equiv="X-UA-Compatible" content="IE">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <title>Javascript Demo</title>
8      <script>
9          alert('hello world');
10         window.alert('Hello World Again!');
11     </script>
12 </head>
```

- alert is basically window.alert()
- Every builtin javascript function and object in browser based application belongs to window object
 - Event our own code and objects

Document and Console



Both document and console belong to window

```
File Edit Selection View ... hello02.html - basic-js - ...
```

```
Get Started < hello02.html U X ...
```

```
set01 > < hello02.html > html > head
1 <html lang="en">
2 <head>
3     <meta charset="UTF-8">
4     <meta http-equiv="X-UA-Compatible" content="IE">
5     <meta name="viewport" content="width=device-width, initial-scale=1.0">
6     <title>Javascript Demo</title>
7     <script>
8         document.write('<h2>Hello World on Document</h2>');
9         window.document.write('<strong>Hello World Again!</strong>');
10        console.log('hello world on console');
11    </script>
12 </head>
13 <body>
14     <h1>Hello Javascript World</h1>
15 </body>
16 </html>
```

Dynamic DOM Model

Monday, December 27, 2021 12:47 PM

- Can be seen in developer tools (F12)
- Check the 'Elements Tab'

The screenshot shows a browser window with developer tools open. The left pane displays the DOM tree under the 'Elements' tab. The right pane shows the source code of 'hello02.html'. Handwritten annotations include:

- A yellow box around the title 'Hello World on Documents' with the word 'Dynamic' written above it.
- A green box around the 'Hello Javascript World' header with the word 'Static Elements' written below it.
- A blue box around the 'Hello World on Document Again!' text.
- Yellow boxes highlighting 'document.write' calls in the script block.
- A green box highlighting the 'Hello Javascript World' header in the source code.

Arrows from these annotations point to the corresponding elements in both the DOM tree and the browser preview.

```
set01 > <!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Javascript Demo</title>
    <script>
      document.write('<h2>Hello World on Document</h2>');
      document.write('<strong>Hello World on Document Again!</strong>');
      console.log('hello world on console');
    </script>
  </head>
  <body>
    <h1>Hello Javascript World</h1>
    <!-- code injected by live-server -->
    <script type="text/javascript">...</script>
  </body>
</html>
```

Order of Execution

Monday, December 27, 2021 1:01 PM

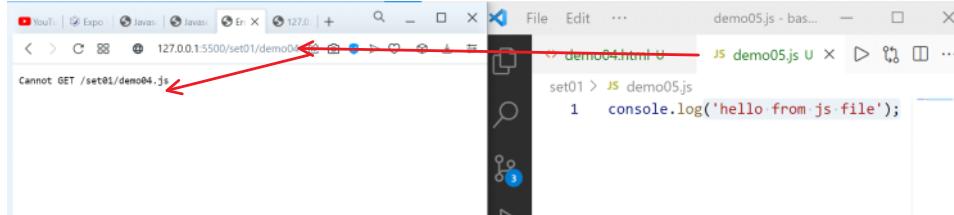
The screenshot shows a browser window with developer tools open. On the left, the 'Elements' tab is selected, displaying the DOM structure. On the right, the 'Sources' tab shows the script file 'hello03.html'. A red box highlights the line of code: 'clientArea.innerHTML = 'Hello World on cli''. A red arrow points from this line to the 'Elements' tab, where a red box surrounds the 'client-area' div. Another red arrow points from the 'client-area' div back to the 'Sources' tab, indicating the flow of execution. A red annotation 'This object doesn't exist yet' is placed near the end of the line of code.

```
set01 > hello03.html > html > body > person
11     console.log( 'Hello World on console' );
12     let clientArea=document
13         .getElementById('client-area'); ←
14
15     clientArea.innerHTML = 'Hello World on cli' ←
16     </script>
17 </head>
18 <body>
19     <h1>Hello Javascript World</h1>
20     <person email="vivek@thelostepic.com">
21         Vivek Dutta Mishra
22     </person>
23     <div id="client-area">
24         If you are seeing this message
25             that means something went wrong!
26     </div>
27 </body>
28 </html>
```

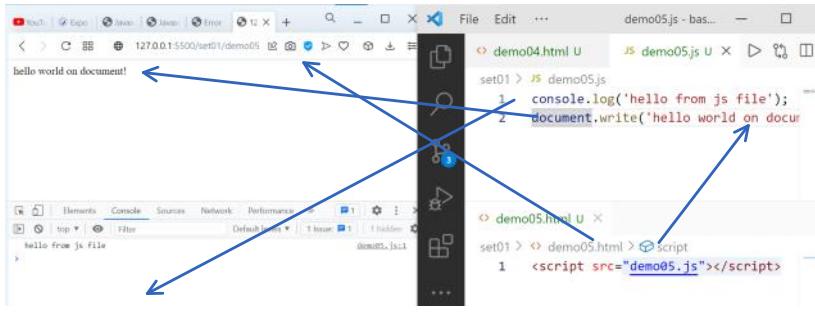
Javascript Execution

Monday, December 27, 2021 2:09 PM

- We can write our javascript code in a separate .js file
- This file can't execute directly in browser
- If we try to access it directly, we will get the file source code



We must include it in some HTML with 'src' attribute



IMPORTANT!

- Script tag must be explicitly closed with end tag even if it contains no code inside
- Don't use self closing script tag

~~<script src="hello.js" />~~

- Always use

`<script src="hello.js" > </script>`

Data types

Monday, December 27, 2021 2:52 PM

- Javascript has 5 key data types
- Data type is associated with value and not with variable
- Variables are typeless. They can refer to anything

Avaialable types

1. number
2. string
3. boolean
4. Object
5. **Function**

Data type belongs to 'value' not 'variable'

```
var x;  
  
x=20; //OK  
  
x='hello world'; //OK
```

- Var is loosely similar to
 - Object of Java
 - Object of C#
- Var is exactly similar to
 - dynamic of C#

```
var a=10;  
var b=20.5;
```

typeof a number
typeof b number


```
var c='hello world';  
var d="hello world";  
var e="x";
```

typeof c string
typeof d string
typeof e string

```

var f= true;
var g=false;                                         typeof f boolean
                                                       typeof g boolean

var h=new Object();
var i={};                                            typeof h object
var j=new Date();                                     typeof i object
var k=[1,2,3,4,5];                                    typeof j object
                                                       typeof k object

var l=null;                                         typeof l object

var m=undefined; //value explicitly undefined ---> {0}
var n;                                                 typeof m undefined
                                                       typeof n undefined
//var o; //variable is undefined

console.log('typeof a',typeof a);
console.log('typeof b',typeof b);
console.log('typeof c',typeof c);
console.log('typeof d',typeof d);
console.log('typeof e',typeof e);
console.log('typeof f',typeof f);
console.log('typeof g',typeof g);
console.log('typeof h',typeof h);
console.log('typeof i',typeof i);
console.log('typeof j',typeof j);
console.log('typeof k',typeof k);
console.log('typeof l',typeof l);
console.log('typeof m',typeof m);
console.log('typeof n',typeof n);
console.log('typeof o',typeof o);                      typeof o undefined

```

Boolean vs Truthies and Falsies

Monday, December 27, 2021 3:08 PM

Trurthies

- Javascript has a specific value "true" to indicate "true case"
- But javascript treats other values also a truth in conditional context
- **Truthies**
 - true
 - All numbers except 0
 - All string except empty string
 - All not-null objects
- Falsies
 - false
 - 0
 - null
 - undefined
 - "

Function

Monday, December 27, 2021 3:29 PM

```
function greet(name){  
    console.log('Hello',name,'Welcome to Javascript');  
}  
  
greet("Prограммеры");  
  
function add(x,y){  
    return x+y;  
}  
  
console.log('add(2,3)',add(2,3));
```

*Identifier for the function

Function may have return value
• If you return nothing you return "undefined"

- Parameter name
 - No explicit type
 - We can't use var keyword here

Function Parameters and Arguments

- There is no direct correlation between number of formal parameters and actual parameters supplied
 - You can supply more or less argument than needed by the function

Passing More Arguments

- If we pass more arguments than formal parameters additional arguments are simply ignored

```
function add(x,y){  
    return x+y;  
}
```

```
console.log('add(2,3)',add(2,3));
```

```
console.log('add(2,3,4,9,8,13)',add(2,3,4,9,8,13));
```

- Not really ignored.
- They are not assigned to formal parameters
- They are present in the 'arguments' parameter
 - (see below)

Passing Fewer Arguments

- Additional parameters are treated undefined.
- Result may be unexpected
 - But no error

```
function add(x,y){  
    return x+y;  
}
```

$5 + \text{undefined} = \text{NaN}$
→ not a number

```
|   return sum,
}
console.log('add(2,3)',add(2,3));
console.log('add(2,3,4,9,8,13)',add(2,3,4,9,8,13));
console.log('add(5)',add(5)); //same as add(5,undefined)|
```

arguments parameter

- Every javascript function apart from getting formal parameters also gets a non-formal parameter called "arguments"
- It is like an array
- It holds all values passed to the function
 - Even the additional value
 - No argument is actually ignored.

Variable Scope

Monday, December 27, 2021

4:10 PM

- Added to global/window context
 - Global context in nodejs
 - Window in browser based application

"add" is a variable created in global/window context

- It refers to another global element "plus"

```
function plus(x,y){  
    return x+y;  
}  
  
let add= plus;
```

```
console.log('typeof plus',typeof plus,plus.name);  
console.log('typeof add',typeof add,add.name);
```

- "subtract" is Directly assigned to "minus"
- Never added to global/window context
- Subtract word will not be accessible directly

"minus" is a variable created in global/window context

- Refers to a function 'subtract' which is never attached to global context.

```
var minus = function subtract(x,y){  
    return x-y;  
};
```

```
console.log('typeof minus', typeof minus,minus.name);  
console.log('typeof subtract', typeof subtract,subtract.name);
```

ReferenceError: subtract is not defined
at Object.<anonymous> (d:\MyWorks\Corporate\202112-mobileum-pern\basic-js\s
0)

Anonymous functions

- Functions that doesn't have explicit name
- They are directly assigned to a variable
- They get the name of the variable to which they are assigned.

```
var minus = function (x,y){  
    return x-y;  
};  
console.log('typeof minus', typeof minus,minus.name);  
  
typeof minus function minus
```

Two ways of writing Function in Javascript

- Classical function syntax

```
function plus(a,b) {  
    return a+b;  
}
```

- Function as Object Syntax/ Function reference syntax

Preferred way of writing function is Javascript

}

- Function as Object Syntax/ Function reference syntax

```
var mins= function () {  
    return a-b;  
}
```



Preferred way of writing function is Javascript

Assignment 01

Monday, December 27, 2021 4:46 PM

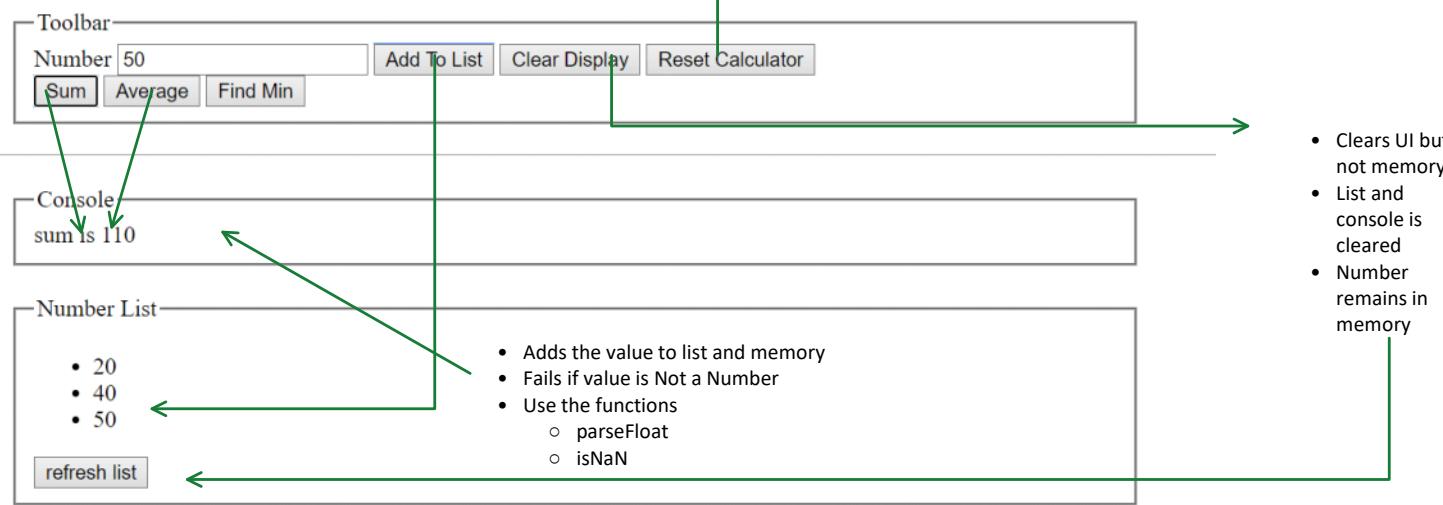


calc

Reset UI and memory

- Implement calculator logic using calc.html file supplied

< > C 88 | file:///D:/MyWorks/Corporate/202112-mobileum-pern/private/calc00.html



Refresh Repopulates list with memory value

Global Name Conflict

Tuesday, December 28, 2021 10:55 AM

The screenshot shows a browser window with developer tools open. The top part displays a simple calculator application with a toolbar, a console input field, and a number list. The bottom part shows the browser's developer tools with the 'Console' tab selected. An error message is visible in the console: "Uncaught TypeError: console.log is not a function" at line 62 of calc.js. A red arrow points from this error message to the user-defined 'console' variable declaration in the script. Another red arrow points from the same declaration to the script code. The script itself defines a local 'console' variable pointing to a div element, which then logs to the user-defined 'console' object.

```
var console=document.getElementById('console');
var reset=function(){
    console.log('clearing the UI');
    clear(); //clear UI
    numbers=[]; //clear memory
}
```

- Overwrites the built-in "console" object
- Now we don't have console.log function

IMPORTANT!

- Any global variable can overwrite another variable declared with the same name
- Our 'console' pointing to 'div' can overwrite 'console' which comes as part of BOM
- This is why we can't have 'function overloading' in javascript.

SOLUTION

- CHOOSE A DIFFERENT NAME FOR 'Console' till we get a better option

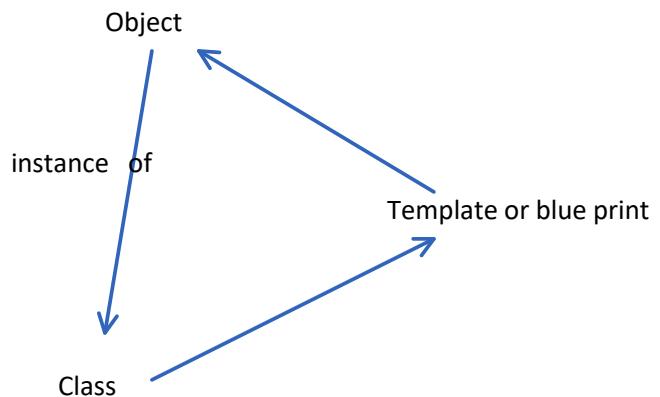
Local vs Global

- A local name is preferred over global name
- If a button calls clear() it will call its built-in clear rather than user defined clear

OOP

Tuesday, December 28, 2021 11:39 AM

anything



```
FresherEmployee arpit=new FresherEmployee();
```

Foooo

Javascript OO Model

Tuesday, December 28, 2021 12:27 PM

```
//we created an object  
var p=new Object();
```

```
//we have an object without  
//any specific property or behavior  
p.name='Shivanshi';  
p.age=15;
```

```
function showPerson(person){  
    console.log('name',person.name);  
    console.log('age',person.age);  
}  
showPerson(p);
```

1. An Object is created

- We don't know the properties or behavior of this object yet.
- It can be attached to the object after it's creation.
- Object can evolve over a period of time.
 - We need not know everything about this object before its creation.
- CLASS ORIENTED DESIGN FORCES YOU TO KNOW EVERYTHING ABOUT AN OBJECT BEFORE ITS CREATION.

2. Associate the required property of the Object

- We can add new properties to the object even after it's creation.
- Allows object to dynamically evolve over a period of time.

3. Interacting with the object

- We can interact with the object and its properties
- Current property set can be modified
- New properties can be added

Different ways to create object in JS

Tuesday, December 28, 2021 12:38 PM

```
function showPerson(person){  
    console.log('name',person.name);  
    console.log('age',person.age);  
    console.log();  
}
```

```
var p1=new Object();  
p1.name='Shivanshi';  
p1.age=15;  
showPerson(p1);
```

```
var p2= {} ; //same as new Object();  
p2.name="Sanjay";  
p2.age=40;  
showPerson(p2);  
var p3 = {  
    name:'Vivek',  
    age:100  
};  
showPerson(p3);
```

```
var p4 = { name:'Prabhat' } ; //JSON Notation  
p4.age=40; //object notation  
  
p4['email']='prabhat@gmail.com'; //dictionary Notation  
showPerson(p4);  
console.log('email',p4.email);  
  
console.log('age', p4["age"]);
```

1. Conventional Object Oriented Semantic

- We use dot(.) notation similar to other languages like java/c++/c#/python

2. {} is a shortcut for new Object

3. JSON Format

- JavaScript Object Notation
- Properties are initialized within {}
- We use ':' instead of '='
- Differs only in creation
- We use the object normally using the dot notation

4. Hybrid Mode

- An object can use all different modes together
 - Few properties can be created using JSON syntax
 - Other can be created using dot syntax or Dictionary Syntax

5. Dictionary Model

- An object is essentially like a dictionary of key-value pairs
- Each property is like a key
- object.something and object["something"] is interchangeable

For-in loop for Object

- Iterates through each property name of an object

```
for( var i in person){  
    console.log(i);  
}  
  
name  
age  
email
```

Iterating through all property-value of an object

```
function showPerson(person){  
    for(var propertyName in person){  
        console.log(propertyName, person[propertyName]);  
    }  
    console.log();  
}
```

For-in vs for-of

Tuesday, December 28, 2021 12:57 PM

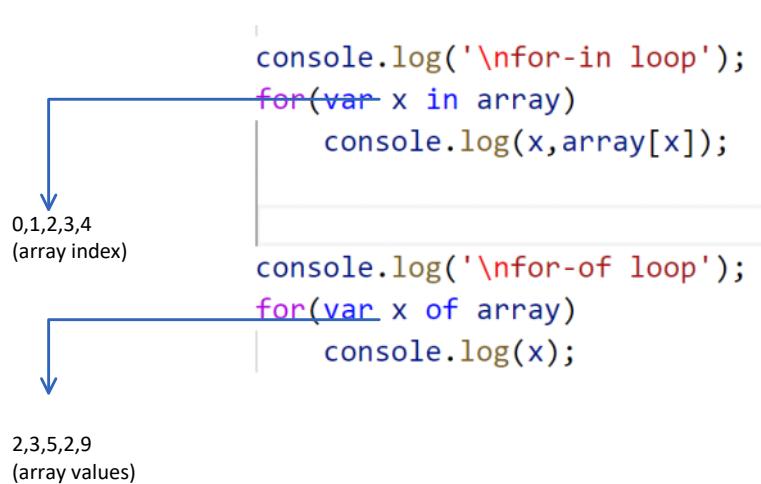
```
var array=[2,3,5,2,9]
```

For-in

- Works for array and object
- For array it returns valid index of an array
- For Object it return all property name of the object

For -of

- Works only for arrays not ordinary objects
- Returns each value one by one
 - Doesn't include index



Recommendation

- Use for-of loop for array
- Use for-in loop for object

Object Behavior

Tuesday, December 28, 2021 1:05 PM

```
function showPerson(person){  
    console.log('name',person.name);  
    console.log('age',person.age);  
}  
  
var p=new Object();  
  
p.name='Shivanshi';  
p.age=15;  
  
p.show=showPerson; ←  
  
//step 3: use object  
p.show(p);
```

- Remember A function is also a type
- It can be assigned to a variable
- It can also be assigned to the property of an object

- Now we can use the function using standard dot notation

Redundant code

- We need to explicitly pass object to the function
- The invoking object has no purpose
- It's just a cosmetic design.

Which object will be shown in the next example

```
var p2= {name:'Prabhat', age:40};  
  
//which object is shown p or p2?  
p.show(p2);
```

- This line shows object 'p2'
 - Should it not show 'p'?
- What is the role of 'p' in this example?

'this' keyword

- 'this' keyword is an implicit keyword in every function
- It refers to the invoking object (if any)

```
12  function showPerson(person){  
13      console.log('name',this.name);  
14      console.log('age',this.age);  
15  }  
16  
17  p.show=showPerson;  
18  //step 3: use object  
19  p.show();  
20  
21
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
c-javascript>node "d:\MyWorks\Corporate\202112-mobileu  
n-tern\basic-javascript\set03-oo\0005.js"  
name Shivanshi  
age 15
```

- An un-needed un-used parameter
- No semantic error
- But we should remove it as it has no purpose.

Calling 'this' for methods without object context.

- If a method is called without any context

- It is treated as
 - 'global' context in node.js
 - 'window' object in browser based application

```

12 function showPerson(person){
13   console.log('name',this.name);
14   console.log('age',this.age);
15 }
16 p.show=showPerson;
17 //step 3: use object
18 p.show();
19
20 showPerson();
21

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
m-pern\basic-js\set03-oo\oo05.js"
name Shivanshi
age 15
name undefined
age undefined
```

- 'this' exists
- 'name' doesn't exist of 'this'

In browser based application

- A function called without context is implicitly called with window as 'this'
- this.name will be same as window.name

Elements Console Sources Network Performance

top Filter Default levels

```

name Shivanshi
age 15
name Mr Global
age undefined

```



set03-oo > JS oo05.js > ...

```

10 //step 2B: object can have behaviors
11 //Remember: even functions are object/type
12 function showPerson(person){
13   console.log('name',this.name);
14   console.log('age',this.age);
15 }
16 p.show=showPerson;
17 //step 3: use object
18 p.show();
19
20 var name='Mr Global';
21
22 showPerson();
23

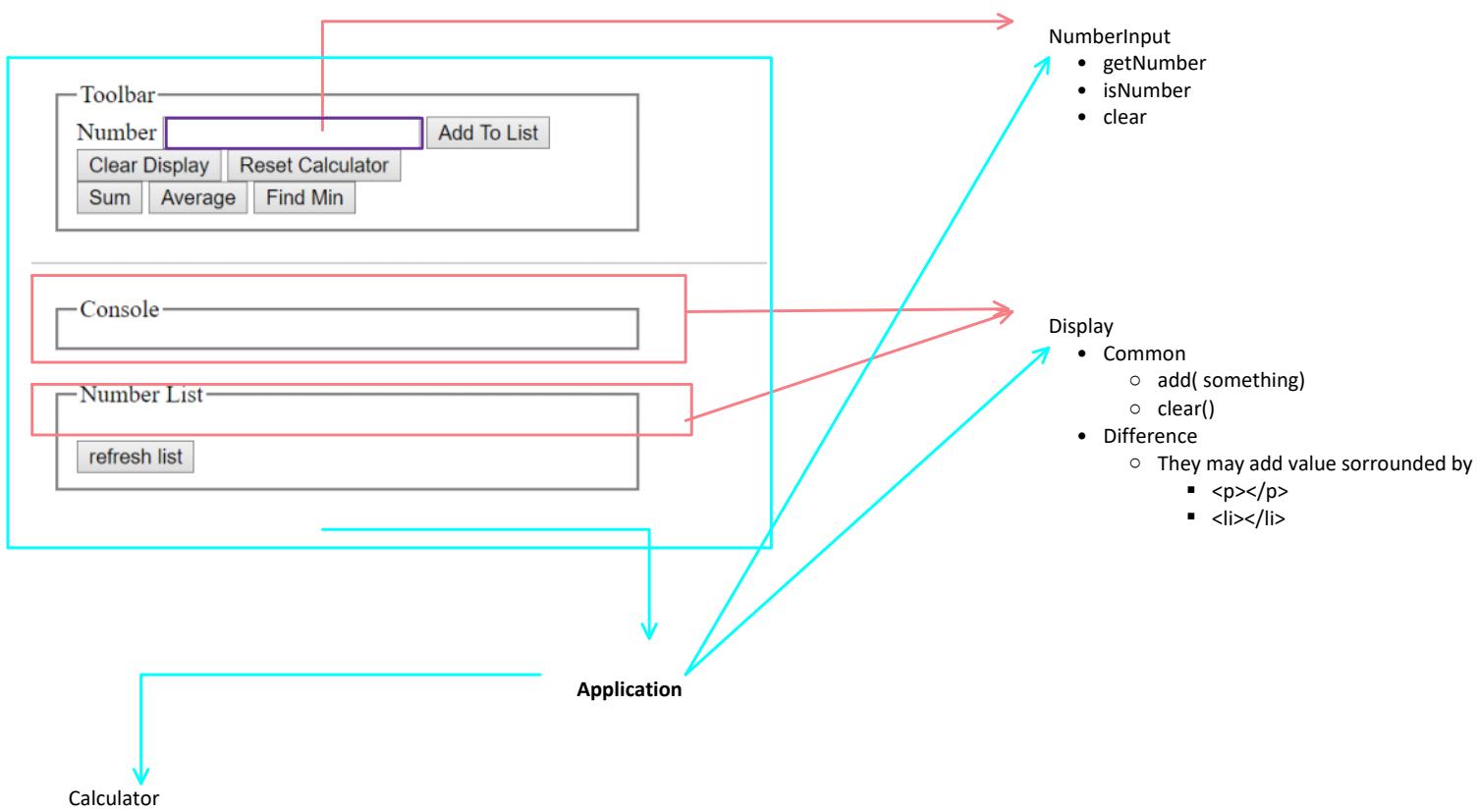
```

Same as
window.showPerson();

Object Model

Tuesday, December 28, 2021

4:14 PM



This

Tuesday, December 28, 2021 4:19 PM

This with global context

The screenshot shows a browser developer tools interface. On the left, the 'Console' tab is active, displaying the following JavaScript session:

```
> createEmployee("Sanjay",100000)
< undefined
> window.name
< "Sanjay"
> window.salary
< 100000
>
```

On the right, the 'Sources' tab shows the source code for 'oo07.js':

```
14 >
15 function createEmployee(name,salary){
16     //object not created here
17     //we are initializing 'this'. What?
18     this.name=name;
19     this.salary=salary;
20     this.show=function(){...}
21     this.work=function(){...}
22     //object not returned
23 }
24
25 }
```

Annotations with arrows point from the highlighted code in the 'Sources' tab to the corresponding variables in the 'Console' tab.

- Uses 'window' object as 'this'
- Sets
 - window.name
 - Window.salary
- Returns undefined

This with 'new'

- When a function is called with 'new'
 1. It creates a new object
 2. 'this' refers to the newly created object within the function
 3. Function implicitly returns the newly created object

The screenshot shows a browser developer tools interface. On the left, the 'Console' tab is active, displaying the following JavaScript session:

```
• new creates a new object
• this refers to that new object

> x=new createEmployee( 'Vivek', 1)
< createEmployee {name: 'Vivek', salary: 1, show: f, work: f}
> x.name
< 'Vivek' * Function returns the new object automatically
> x.salary
< 1
>
```

On the right, the 'Sources' tab shows the source code for 'oo07.js':

```
15
16 function createEmployee(name,salary){
17     //object not created here
18     //we are initializing 'this'. What?
19     this.name=name;
20     this.salary=salary;
21     this.show=function(){...}
22     this.work=function(){...}
23     //object not returned
24
25 }
```

Annotations with arrows point from the highlighted code in the 'Sources' tab to the corresponding variables in the 'Console' tab.

new and create are synonymous

- If we are using 'new'
 - Avoid having prefix in function like
 - createXyz
 - getXyz
 - Generally this function should start with an upper case letter
 - This kind of function is known as **Constructor** function

Constructor vs Normal creator

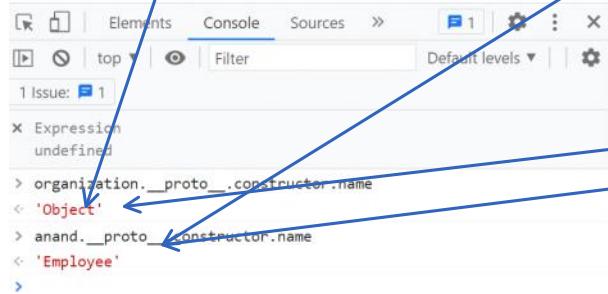
Tuesday, December 28, 2021 5:28 PM

Creator or Factory Function

- Uses Generic 'Object' to create a function
- No special identifier

```
function createOrganization(name){  
    var organization=new Object();  
    var id=0;  
    organization.name=name;  
    organization.employees=[];  
    organization.addEmployee=function(  
        employee.organization=this;  
        employee.id=++id;  
        this.employees.push(employee)  
    }  
  
    return organization;  
};
```

Users generic 'Object' constructor



Constructor Function

- Uses special constructor function
- A constructor function should have names with upper case
- It should assign elements to **this**.
- It shouldn't return a value

```
function Employee(name,salary){  
    //object not created here  
    //we are initializing 'this'.  
    this.name=name;  
    this.salary=salary;  
    this.show=function(){...}  
    this.work=function(){...}  
}  
  
//object not returned
```

```
28 > this.work=function(){...}  
30 }  
31 //object not returned  
32 }  
33  
34  
35 let organization=createOrganization("TechCamp")  
36 let anand=new Employee("Anand",50000)  
37  
38 organization.addEmployee(anand);  
39  
40 anand.work();
```

Object Properties and prototype

Tuesday, December 28, 2021 5:40 PM

```
function Person(name,age){  
    this.name=name;  
    this.age=age;  
    this.eat=function(food){  
        console.log(this.name+' eats  
    }  
    this.move=function(start,end){  
        console.log(this.name+" goes  
    }  
};
```

```
> prabhat  
< ▶ Person {name: 'Prabhat', age: 40, eat: f, move: f}  
> shivanshi  
< ▶ Person {name: 'Shivanshi', age: 15, eat: f, move: f}  
>
```

- Every object gets their own copies of properties including functions
- Both prabhat and shivanshi object have their own copy of
 - name
 - age
 - eat
 - Move
- There are duplicate copies of eat() and move() function in memory
 - We have two eat() function that does the same thing
- In other C++ like language (java/C#) behaviors have single copy
- This may result in memory waste if we have many objects of the same type

Constructor and Prototype

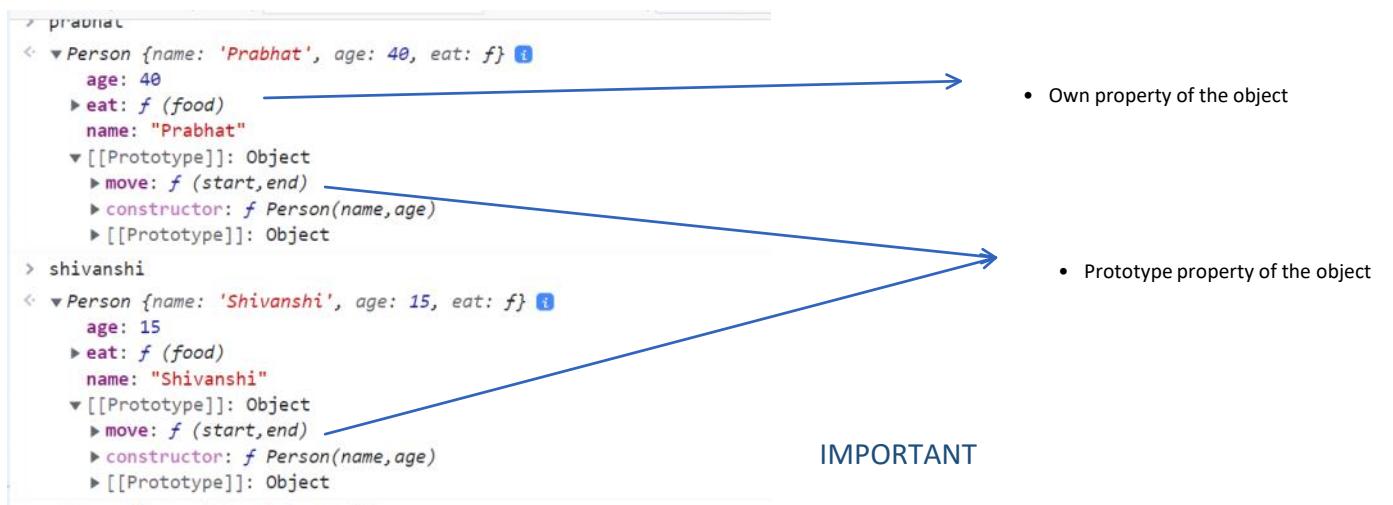
- A constructor function has 'prototype' property.
- A prototype itself is a special object
- Any property or method added to a prototype is automatically available to all objects created using that constructor
 - A newly added method on the prototype will be available even to the older objects created using same constructor
- Advantage
 1. A single copy of the methods shall be available if added to the prototype
 2. If we need add new behavior to prototype it will be available to older objects object

The screenshot shows a browser developer tools interface with two main panes. The left pane is the 'Console' tab, which displays the following JavaScript interactions:

```
> prabhat
< > Person {name: 'Prabhat', age: 40, eat: f}
> shivanshi
< > Person {name: 'Shivanshi', age: 15, eat: f}
> shivanshi.move('home', 'school');
    Shivanshi goes from home to school
< undefined
> prabhat.move('home', 'office');
    Prabhat goes from home to office
< undefined
>
```

The right pane is the 'Sources' tab, showing the code for 'oo08.js'. The code defines a 'Person' constructor function and a 'move' method on its prototype:1
2 function Person(name,age){
3 this.name=name;
4 this.age=age;
5 this.eat=function(food){
6 console.log(this.name+' eats '+food);
7 }
8 };
9
10
11 var prabhat=new Person("Prabhat",40);
12
13 Person.prototype.move=function(start,
14 console.log(this.name+" goes from "+start+" to "+end);
15 }
16 var shivanshi=new Person("Shivanshi",15);

Annotations with blue arrows point from the 'eat' and 'move' method definitions in the code to their respective usages in the console output.



comparision

Wednesday, December 29, 2021 9:53 AM

- Javascript consists of two different variation of comparisoin
 - == vs ===
 - != vs !==
- Most cases we recommend using === and !== to ensure exact match
- Below is the table.
- Note where the comparison actually varies.

x	y	==	===
undefined	undefined	true	true
null	null	true	true
true	true	true	true
false	false	true	true
'foo'	'foo'	true	true
0	0	true	true
+0	-0	true	true
+0	0	true	true
-0	0	true	true
0	false	true	false
""	false	true	false
""	0	true	false
'0'	0	true	false
'17'	17	true	false
[1, 2]	'1,2'	true	false
<code>new String('foo')</code>	'foo'	true	false
null	undefined	true	false

Assignment 3.1

Wednesday, December 29, 2021 10:10 AM

- Expand teach driving to teach a particular type vehicle
- Person need license for each individual vehicle
- Person below 18 years will have no drive method
- A person licensed to drive bike
 - Can drive bike
 - Can't drive truck/car etc

```
var prabhat=new Person("Prabhat", 40);
var shivanshi=new Person("Shivanshi", 15);

teachDriving(prabhat, 'Car'); //prabhat you are now licensed to drive Car
teachDriving(prabhat,'Bike'); //prabhat you are now licensed to drive Bike
teachDriving('shivanshi', 'Bike'); // shivanshi you need to wait for 3 years
to drive

prabhat.drive('car'); //prabhat drives car

prabhat.drive('bus'); //prabhat is not licensed to drive bus -> drive()
method is called an it prints this message

shivanshi.drive('anything'); //Type Error: shivanshi.drive is not a function
```

ES Standards

Wednesday, December 29, 2021 11:38 AM

- ES is like a versioning standard for Javascript
 - Defines the
 - language features such as
 - Keyword
 - Operators
 - Standard library function that should be included
- Browsers/NodeJS/JS Runtime is expected to follow ES standard
 - Exact implementation is dependent on implementer/vendor
 - A vendor/implementer may not be updated to the latest standard
- If you follow the latest standard of ES, chances are older (and few recent browsers) may not have implemented all those features.
 - It may not work everywhere

Common ES Standards

ES5 (Core JS Features)

- Supported by almost every JS runtime created in last one decade.
- You can safely use all the features defined as per ES5 and expect it to have largest compliance.
- So far we have used mostly ES5 features.

ES6 (later renamed to ES2015)

- Recent version of most browsers/runtime (from last 2+ years) supports roughly 96% of features
- IE6 supports only 11% of the features
 - You may be surprised we still have many IE users!
- Can be used if we are sure our clients are using modern browsers
- You should have no problem if you are writing code for NodeJS

ES 2016

ES 2017

Check compatibility List here —> <https://kangax.github.io/compat-table/es6/>

Scope Rules

Wednesday, December 29, 2021 11:54 AM

- ES5 supports 3 scopes for variables
 1. Explicit global
 - Variables have global scope
 - They are defined outside any function
 2. Explicit local
 - Variables have function scope.
 - Defined inside a function.
 3. Implicit global

Global vs Local

```
var global='global variable'; //explicit global

function f1(){
    var filocal= 'f1 local'; //explicit local variable
    console.log('in f1');
    console.log('global',global);
    console.log('filocal',filocal);
    global+= ' modified by f1';
    filocal+= ' modified by f1';
}

> function f2(){...}
>

> function f3(){...}
>

f1();
console.log('in global space');
console.log('global',global);
console.log('filocal',filocal);
```

- Any function can access/modify the global value

- A function local can't be accessed outside the function

```
Corporate\202112-mobileum-pern\basic-js\set04-advancedjs\demo01-scope-rules.js"
in f1
global global variable
filocal f1 local
in global space
global global variable modified by f1
d:\MyWorks\Corporate\202112-mobileum-pern\basic-js\set04-advancedjs\
demo01-scope-rules.js:31
console.log('filocal',filocal);
^

ReferenceError: filocal is not defined
    at Object.<anonymous> (d:\MyWorks\Corporate\202112-mobileum-pern\basic-js\set04-advancedjs\demo01-scope-rules.js:31:23)
```

Implicit Global

```

var global='global variable'; //explicit global

function f1(){
    var filocal= 'f1 local'; //explicit local variable
    console.log('in f1');
    console.log('global',global);
    console.log('filocal',filocal);
    global+= ' modified by f1';
    filocal+= ' modified by f1';
}

function f2(){
    f2local= 'f2 local'; //explicit local variable
    console.log('in f2');
    console.log('global',global);
    console.log('f2local',f2local);
}

function f3(){
    console.log('in f3');
    console.log('global',global);
    console.log('f2local',f2local);
    console.log('filocal',filocal);
}

f1();
f2();
f3();

in f3
global global variable modified by f1
f2local f2 local
d:\MyWorks\Corporate\202112-mobileun-pern\basic-js\set04-advancedjs\demo01-scope-rules.js:24
    console.log('filocal',filocal);
        ^
ReferenceError: filocal is not defined

```

Implicit global variable

- A function can't access the local variables declared within other function

- If we declare (first use rule) a variable without "var" keyword, it is treated as implicitly global even if it is defined (first used) inside some function
- A variable which is never declared but used will get created as a global variable.
- Implicit global is often a cause of bug in our system. Be aware!
- We can declare 'strict mode' to get warning against implicit global

Implicit Global Problem

```

function printer(jobId, times){
    for(i=1; i<=times; i++)
        console.log(jobId, i);
}

function printManager(workerCount, workCount){
    for(i=1; i<=workerCount; i++)
        printer(i, workCount),
}

//printManager(3, 5);

printManager(5, 3);

```

Implicit global

- Both these refers to same variable

1	1
1	2
1	3
5	1
5	2
5	3
5	1
5	2
5	3
5	1

... infinite loop

Basic ES2015 features

Wednesday, December 29, 2021 12:28 PM

1. let and const keyword

- Let/const can be used instead of 'var'
- Const ensures that the variable doesn't change
- Let is similar to var

Let vs var

- Local variable
 - let is block scoped, var is function scoped
 - A var will be available in the enclosing function even if it is defined inside a block like (for loop/switch case)
 - Let will be available only inside that block

```
function simpleFunction(){
```

```
    for(let i=0;i<3;i++){
        var v=i*10;
        let l=i*10;
        console.log(i,v,l);
    }
    console.log('v',v);
    console.log('l',l);
```

```
simpleFunction();
```

- Var is accessible even outside for-loop
 - It is NOT BAD.
 - It is a feature
- Let is not accessible outside the block
 - NOT bad
 - It is a feature
- Note loop counter is declared using 'let' so it will not be available after for-loop is over
 - A common scenario

2. A var declared anywhere inside a function is conceptually declared at the top of the function with undefined value.
 - You can use a 'var' first and declare it later.

```
function simpleFunction(){
    //conceptually v is declared here. I can access it below this point
    //this concept is known as hoisting of values

    for(let i=0;i<3;i++){
        console.log('v in loop',i,v);
        var v=i*10;
        let l=i*10;
        console.log(i,v,l);
    }
    console.log('v',v);
```

- Although 'v' is declared later, conceptually it is declared on the top
 - At this point value may be undefined but variable exists

1. A let is accessible only after it is declared.

```
set05-ES2015 > JS demo02-letjs > simpleFunction
```

```
2  function simpleFunction(){
3      //conceptually v is declared here. I can access it below this point
4      //this concept is known as hoisting of values
5
6      for(let i=0;i<3;i++){
7          console.log('v in loop',i,v);
8          console.log('l in loop',i,l);
9          var v=i*10;
10         let l=i*10;
11         console.log(i,v,l);
12     }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

The screenshot shows a code editor with a terminal tab open. The terminal displays the following error message:

```
ReferenceError: Cannot access 'l' before initialization
  at simpleFunction (d:\MyWorks\Corporate\202112-mobileum-pern\bas
ic-js\set05-ES2015\demo02-let.js:8:35)
```

Global scope

- Var is attached to the 'window' object, let is not attached to the 'window' object
- In Nodejs this difference will not be clear.

The screenshot shows a browser developer console with the following interactions:

```
> var x=20;
< undefined
> let y=20;
< undefined
> console.log(window.x)
20
< undefined
> console.log(window.y)
undefined
< undefined
> |
```

Annotations with arrows show that 'x' is attached to the window object (blue arrow from 'window.x' to '20'), while 'y' is not (red arrow from 'window.y' to 'undefined').

Redeclarations

- A variable can be redeclared using 'var'
- It replaces the old one
- Gets attached to window object
- Let doesn't allow redeclaration of values

The screenshot shows a browser developer console with the following interactions:

```
set05-ES2015 > JS demo03.let.js > ...
1  var x=20;
2  let y=20;
3
4  console.log('x: '+x+ ' y: '+y);
5
6  var x='Hello'; //x can be redeclared with var
7
8  console.log('x: '+x);
9
10 let y='Hello';|
```

Annotations with red arrows point to the second declaration of 'y' and the declaration of 'x' in line 6, highlighting the error. The console output shows:

```
SyntaxError: Identifier 'y' has already been declared
```

String interpolation

- ES2015 supports string interpolation
 - You can inject dynamic variable and expression inside a string
 - They will be automatically expanded.
 - You can do that using back-tick quoted string (` `)

- Back Tick String has two features
 1. String interpolation
 2. Multiple Line
 - You don't need to close it on one line

Interpolation

- Allows us to write expression variable wrapped in `{}$`

```
const checkDrivingEligibilityES5=function(person){
  if(person.age>18)
    console.log("Hey, " + person.name + " you can drive ");
  else
    console.log("Hey, " + person.name+ " you should wait for "+(18-person.age)+" years to drive");
}

const checkDrivingEligibilityES2015=function(person){
  if(person.age>18)
    console.log(`Hey, ${person.name} you can drive `);
  else
    console.log(`Hey, ${person.name} you should wait for ${18-person.age} years to drive`);
```

The diagram shows two annotations for the ES2015 code. A blue bracket labeled "Inject variable" points to the `${person.name}` placeholder in the first `console.log` statement. Another blue bracket labeled "Inject expression" points to the `${18-person.age}` placeholder in the second `console.log` statement.

Function Parameter Defaults

- You can declare a default value for a function parameter
- If user doesn't supply parameter it default to given value rather than undefined

```
var powerES5=function(number,exp){
  if(exp==undefined)
    exp=2;
  return Math.pow(number,exp);
}

var powerES2015=function(number,exp=2){
  return Math.pow(number,exp);
}
```

The diagram shows a blue bracket pointing to the `exp=2` part of the parameter declaration in the ES2015 code, indicating it's a default value.

ES2015 params (aka rest params)

Wednesday, December 29, 2021 2:09 PM

```
function averageES5(){
  var sum=0;
  console.log('arguments',arguments);
  for(var i=0;i<arguments.length;i++)
    sum+=arguments[i];

  return sum/arguments.length;
}

function averageES2015(...numbers){
  let sum=0;
  console.log('numbers',numbers);
  for(let number of numbers) //ES2015 loop
    sum+=number;

  return sum/numbers.length;
}

console.log('averageES5(1,2,3,4)',averageES5(1,2,3,4));
console.log('averageES2015(1,2,3,4)',averageES2015(1,2,3,4));
```

Params specifier

- arguments is not an array. It is array like object
- Can't be used with for-of loop
- Can work with for/for-in

arguments [Arguments] { '0': 1, '1': 2, '2': 3, '3': 4 }
averageES5(1,2,3,4) 2.5
numbers [1, 2, 3, 4]
averageES2015(1,2,3,4) 2.5

Pure array automatically created from supplied parameter.

IMPORTANT!

- params and arguments are functionally interchangeable
 - They are used in a similar scenario
- The two are internally different
- Params is more explicit.

Arguments vs rest params

- arguments get all the arguments supplied
 - Even those arguments that are assigned to formal parameters
- Rest params gets the remaining parameters that are not assigned to a previous formal parameter

```
6
7  function test(x,y, ...params){
8    console.log('x:',x,'y:',y,'params:',params,'arguments:',arguments);
9  }
10
11 test(1,2,3,4,5);
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

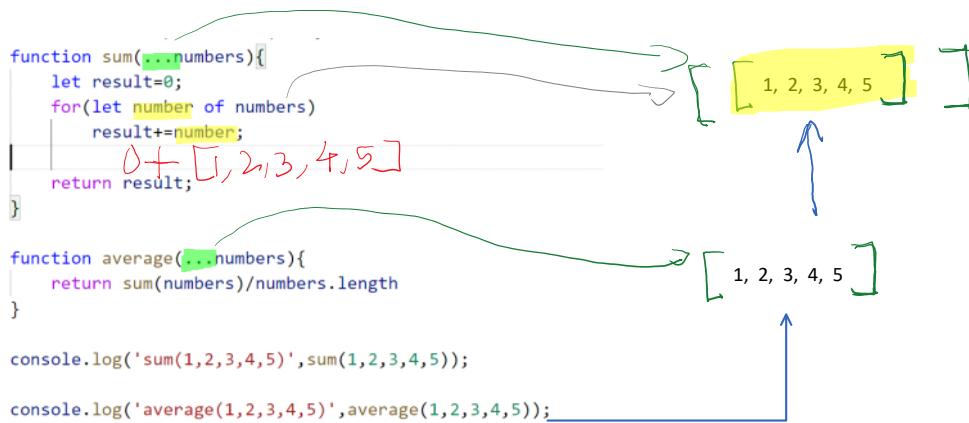
D:\Myworks\Corporate\202112-mobileum-pern\basic-js>node "d:\Myworks\Corporate\202112-mobileum-pern\basic-js\set05-ES2015\demo07-rest-params.js"

x: 1 y: 2 params: [3, 4, 5] arguments: [Arguments] { '0': 1, '1': 2, '2': 3, '3': 4, '4': 5 }

ES2015 spread

Wednesday, December 29, 2021 2:20 PM

Motivation



A screenshot of a code editor showing a file named `demo08-params-and-spread.js`. The code defines a `sum` function that adds up all its arguments. It then uses `console.log` to print the sum of two arrays: `[1, 2, 3, 4, 5]` and `[values]`. A red arrow points from the `values` variable to the `console.log` statement. A green arrow points from the `values` variable to the output window, which shows the result as `sum(values) 01,2,3,4,5`. The output window also shows the command `node "d:\MyWorks\Corporate\202112-mobileum-pern\basic-js\set05\ES2015\demo08-params-and-spread.js"`. A list of notes on the right side of the editor states:

- Wraps comma separated value as an array
- But can't work with an array directly
 - It will rewrap it to create an array of array

```
6  function sum(...numbers){  
7    let result=0;  
8    for(let number of numbers)  
9      result+=number;  
10   return result;  
11 }  
13  
14  
15  console.log('sum(1,2,3,4,5)',sum(1,2,3,4,5));  
16  
17  const values=[1,2,3,4,5];  
18  
19  console.log('sum(values)',sum(values));  
20  
21  
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL  
D:\MyWorks\Corporate\202112-mobileum-pern\basic-js\set05\ES2015\demo08-params-and-spread.js"  
sum(1,2,3,4,5) 15  
sum(values) 01,2,3,4,5
```

Spread Operator

- We have spread operator (opposite of rest parameter) that can unwrap an array as individual value
- Javascript spread operator is (...)
- Remember when you define a function parameter ... represents params everywhere else it is spread.



```

return sum(...numbers)/numbers.length
}
console.log('average(1,2,3,4,5)',average(1,2,3,4,5));

```

Spread Use case

- Spread simply returns individual values from an array

```

function sum(x,y){
  return x+y;
}

var array=[5,9,14];

//how to find sum of first two values of this array?
//ES5
var s1=sum(array[0],array[1]);
console.log('s1',s1);

//ES2015
var s2=sum(...array); //sum(5,9,14) // ---> 14 is ignored.
console.log('s2',s2);

```

- We can spread one array into another

```

//ES5

let set3=[];
for(var i=0;i<set1.length;i++)
  set3.push(set1[i]);
for(var i=0;i<set1.length;i++)
  set3.push(set2[i]);

console.log('set3',set3);

//ES2015

let set4=[...set1,...set2];
console.log('set4',set4);

```

Object Spread

- Spread operator can also work with object properties
- You can copy the values from one object into another object

```

const person={
  name:'Vivek',
  age:100,
  phone:'9939399393',
  email:'vivek@personal.com'
};

```

How do I copy the details from person to employee object

- Employee object will have additional properties
 - Id
 - Salary
- It should overwrite person email with company email

ES5

ES2015

```
const e1={
  id:1, //employee details
  name:person.name,
  age:person.age,
  phone:person.phone,
  //email:person.email,
  salary:1, //employee details
  email: 'vivek@company.com'
};

//ES2015
const e2={
  id:1,
  ...person, //everything from person including unwanted person.email
  salary:1,
  email:'vivek@company.com' //overwrites the earlier email obtained through spread
}
```

Array Destructuring

Wednesday, December 29, 2021 2:50 PM

- We may need to take a few values out of an array

```
const array=[2,9,8,15,4];
```

How to copy first two values from array

ES 5 approach

```
const a=array[0];
const b=array[1];
```

ES2015 approach

```
const [i,j]=array; //i=array[0], j=array[1]
```

//creates two new variable and assigns value to them

ES2015 how to take 0th and 3rd value

- You have to leave space for the two by giving comma

```
const [x,,,y]=array; //x=array[0], y=array[3]
```

ES2015 — How to get 1st and 100th value;

```
const x= array[1];
const y=array[100];
```

IMPORTANT!

- Useful only for smaller array or when you need to access lower indices

Object Destructuring

Wednesday, December 29, 2021 2:57 PM

- Just like array elements, we can also take values from individual properties of an Object

```
const person={  
    name:'Vivek',  
    age:100,  
    phone:'9939399393',  
    email:'vivek@personal.com'  
};  
  
//how to get name and phone of this person  
//ES2015  
  
const {name,phone} = person; //const name=person.name,phone=person.phone  
console.log('name',name);  
console.log('phone',phone);  
  
//you can also change the variable name while extracting  
//how to extract phone number as cellNumber  
  
const {phone:cellNumber} = person; //const cellNumber=person.phone  
console.log('cellNumber',cellNumber);
```

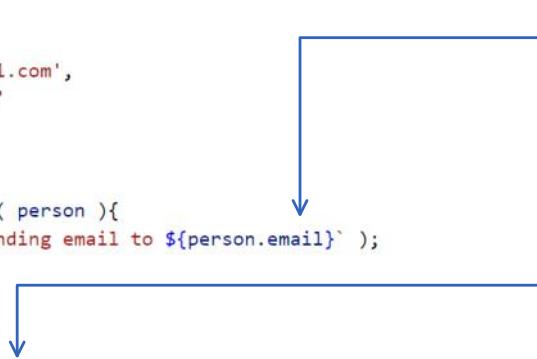
- Note by default we search for a property with same name as variable.

- We may choose a name for variable different than property name
 - We are storing person.phone as cellNumber

Function Parameter Destructuring

- Some time we pass an object to function
- That object needs only a particular property(s) of that object and not the whole object
- We may use destructuring syntax to extract the required property automatically in the function

```
const person={  
    name:'Vivek',  
    age:12,  
    email:'vivek@gmail.com',  
    phone:'9036084835'  
};  
  
function sendEmailES5( person ){  
    console.log(`sending email to ${person.email}`);  
}  
  
sendEmailES5(person);  
  
function sendEmail({email}){  
    console.log(`sending email to ${email}`);  
}  
  
sendEmail(person);
```



- Manual extraction of property
- Automatic destructuring of parameters

ES2015 Arrow Functions

Wednesday, December 29, 2021 3:32 PM

Traditional Function Syntax

```
function plus( x ,y ){  
    return x+y;  
}
```

Anonymous Function as an Object Syntax

```
const minus = function ( x, y ) {  
    return x-y;  
}
```

ES2015 Arrow Function / Lambda Function

```
const multiply = function ( x, y ) => {  
    return x*y;  
}
```

Simplified Lambda Expression

- If your function has a single (return) statement then you can simply it by removing
 - Function block marker {}
 - 'return' keyword
 - Any semicolon at the end of the statement

```
const multiply = ( x, y ) => {  
    return x*y;  
}  
  
const multiply = ( x, y ) => x*y
```

- A lambda function aka Arrow function is an anonymous function
- We don't write the function keyword
- We have to separate parameter list from the body using fat arrow =>

Lambda Function Limitations

There are two limitations of the lambda functions

1. They don't have 'this' reference
 - a. They can refer to the 'this' present in wrapper function if any
2. They don't have 'arguments'
 - a. You can pass ...params

ES2015 OO

Friday, December 31, 2021 11:04 AM

ES2015 object initialization

- Sometimes we initialize an object with some variable
- We want property name and variable name to be same

```
const title=document.getElementById('title').value;
const author=document.getElementById('author').value;
const cover=document.getElementById('cover').value;
```

Now we want to create the book object

ES5 Approach

- Objects are created using key-value pair

```
const book= {
  title:title,
  author:author,
  cover:cover
}
```

ES2015 Approach

- If assigned variable is same as property name we are creating, we may simply list the variables rather than explicit assignment

```
const book= {
  title,
  author,
  cover
}
```

ES2015 class

- ES 2015 introduced the concept of class similar to C++ like language
- We can declare class/constructor/methods etc.

ES 5 Syntax

```
const Book= function (title,author,price,rating){
  this.title=title;
  this.author=author;
  this.price=price;
  this.rating=rating;
}
```

ES 2015 Syntax

```
class Book {
  constructor(title,author,price,rating){
    this.title=title;
    this.author=author;
    this.price=price;
    this.rating=rating;
  }
}
```

IMPORTANT NOTE

- Internally a class is still like ES5 constructor function
- It follows the same notion
- Class methods are added to constructor prototype
- **Javascript Classes are dynamic unlike C++ style classes**
 - You can still add new properties/method to the class or constructor prototype

Note

1. Constructor name is called **constructor**
 - In C++ style languages constructor name is same as class name
2. You can't declare fields directly inside the class
 - It must be initialized in constructor/other functions
 - We must always explicitly use 'this.' to refer to class variable
 - C++ style language makes this optional inside class methods

Classes with method

```
JS book.js U X
es2015-classes > JS book.js > ...
11
12
13  class BookManager{
14
15    constructor(){
16      this.db=[];
17    }
18
19  >   addBooks(...books){...}
20
21  >   getAllBooks(){...}
22
23  >   printBooks(header=''){...}
24
25
26
27  >   printBooks(header=''){...}
28
29
30
31
32
33

JS author.js U X
es2015-classes > JS author.js > ...
9
10  //constructor
11 > function AuthorManager(){...}
12
13  };
14
15  //other methods
16 > AuthorManager.prototype.addAuthors=function(...auth...
17
18  );
19
20  //other methods
21 > AuthorManager.prototype.getAllAuthors=function(){...
22  }
23
24
25 > AuthorManager.prototype.printAuthors=function(head...
26
27
28
29
```

Note

- Class methods don't have
 - Function keyword
 - They can't be arrow function (ES2015)
 - There is a proposed ES2016 class with arrow functions
- All the class functions are added as the constructor prototype methods

We can still add new method to class prototype same as we do for constructor prototype

```
BookManager.prototype.getBookByAuthors= function( a ){
  a=a.toLowerCase();
  return this.db.filter(b=>b.author.toLowerCase().includes(a));
}
```

We can also add new property to objects created using class

```
> bookManager.clear=function(){ this.db=[]; }
```

- This method is not available to other objects of the same class

INHERITANCE

- ES2015 allows class Inheritance
- We don't have an equivalent syntax in ESS
- But we can adopt a certain process to implement inheritance even in ESS
 - Their syntax may look cryptic.

ES2015 Inheritance

```
class Person{
  constructor(name,age){
    this.name=name;
    this.age=age;
  }
  move(start,end){
    console.log(` ${this.name} moves from ${start} to ${end}`);
  }
}

class Employee extends Person{
  constructor(id, name, age, salary){
    super(name,age);
    this.id=id;
    this.salary=salary;
  }
  work(){
    console.log(`${this.name} works as employee#${this.id}`);
  }
}

const emp=new Employee(1,'Vivek',100,100);

emp.move('home','office');
emp.work();
```

Assignment 3.2

Wednesday, December 29, 2021 3:44 PM

- Create a Book Object
- It should have following properties
 - Title
 - Author
 - Price
 - Rating
- Create An array of 5 books (use meaningful names not aaaa,bbbb)
- Create a book manager object with following function
 - addBook
 - getAllBook
 - getBooksByAuthor(author)
 - getBooksInPriceRange(min,max)
 - getBooksWithRatingabove(rating);
- **DON'T USE ANY BUILTIN FUNCTIONS (EXCPT array.push)**

Advanced Function Concepts

Wednesday, December 29, 2021 3:30 PM

- In java script a function is essentially an Object
- It can have it's own properties.
 - Like name
- It can be treated as an object
 - Can be referred by a variable
 - Stored in an array
 - Attached to an object as property
 - **Passed as a parameter to another function**
 - **Can be returned from a function as a value**

Function can be passed as parameter to another function

Callback function

Wednesday, December 29, 2021 3:30 PM

- In java script a function is essentially an Object
- It can have it's own properties.
 - Like name
- It can be treated as an object
 - Can be referred by a variable
 - Stored in an array
 - Attached to an object as property
 - Passed as a parameter to another function
 - Can be returned from a function as a value

Function can be passed as parameter to another function

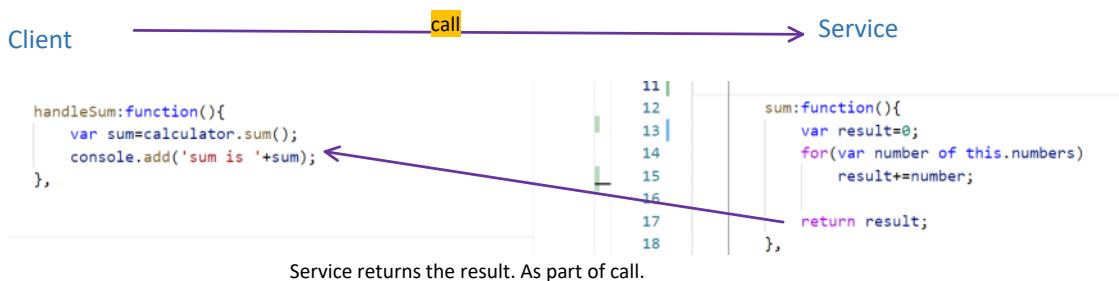
Callback Function

A Normal program flow.

- Every functionality that use (function you call) can be treated as a **service**
- One who uses **service** is client

Steps

1. Client calls the service.
2. Service performs some action for client
3. Service returns some value to the client



Example 2

```
const handleSearch= ()=>{
```

```
  const archerBook= bookManager.getBooksByAuthor("archer");
```

```
}
```

client → service

```
this.getBooksByAuthor=function(author){  
  author=author.toLowerCase();  
  function matchAuthor(book){  
    if(book.author.toLowerCase().includes(author))  
      return true;  
    else  
      return false;  
  }  
  return search(this.db, matchAuthor);
```

Call return

- In A client service flow, client calls service and expects a return from it. (sometimes just action no return)
- From the call to action is a single flow

```
const search=(list, match)=>{  
  let result=[];  
  for(let value of list){  
    if(match(value))  
      result.push(value);  
  }  
  return result;
```

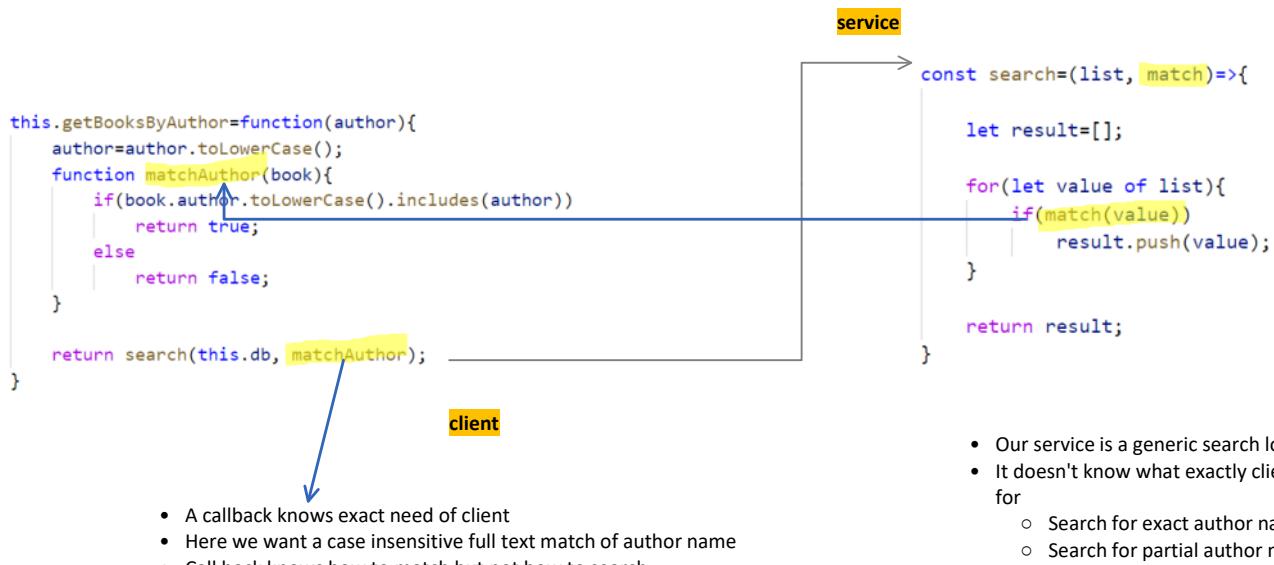
client ← Call return

What is a call back?

- Normally we expect service to perform the entire job required and return a result.
- Sometimes however we may have special situation —
 1. Service may need client's help to complete
 - In search service doesn't know what is match?
 - Service may **call back** client to ask if current value is a match
 - In select service may not know what properties client want to select?
 - Service may **call back** client to ask what they want to select from an item
 - We may want to sort our list of values, sort function doesn't know
 - If we want to sort on title or author or price
 - If we want to sort in ascending or descending order
 - Sorter may **call back** client to ask the ordering
 2. Service may take a long time to finish. In the normal flow, client must wait (without doing anything) till service completes the work.
 - Suppose searching a book requires several minutes.
 - Now client can't do anything till service completes the work.
 3. Now instead of client waiting for the service to finish, service may **call back** the client later when the job is finished.

How a callback is designed

1. When client calls the service, it passes a **call back** function.
 - A call back function is just another function
 - Client is the owner of the function
 - This 'callback' knows specific needs of the client
2. Service is generic
 - **It is designed for a generic solution**
 - Search
 - Sort
 - Process all values
 - It should not be designed for a specific client need
 - Search books by author
 - Search books by price
 - Sort on descending order of rating
 - Print all values on the screen
3. A generic service understands the specific needs of the client by calling the callback which client has passed
 - a. Generic service + client callback → specific service for client
 - Service and callback work together to serve the specific need of the client



How to Search + What to Search => Search
(Generic Service) + (call back) => complete task

Designing a callback

- We must know what type of value a call back will receive and what kind of response call back should returns
- We can use any of these to act as call back
 1. A ready available function that matches call back required
 2. Any user defined named function
 3. Any anonymous function
 4. An arrow function

Arrow/Anonymous Function's Role

- We often design call back using anonymous/arrow functions
- They are short simple
- They are designed locally inside client
- They have easier access to client data due to **closure**

More Use cases of call back

Select/Transform

- We may want to return all elements after transforming them
 - Instead of returning whole book we may want to return just the book titles
 - Instead of returning a list of names I may want to return a list of lengths of those names
 - I may have a list of isbn numbers and I may want to return a list of corresponding books

```
const select=function(source, selector){  
    let result=[];  
    for(let value of source)  
        result.push(selector(value));  
    return result;  
}
```

- Selector callback
 - Takes
 - Single object from original list
 - Returns
 - Another derived object for result list

```
const names= ["india","usa","uk","germany","france" ]  
  
const result= select( names, name => name.length );  
  
console.log(result); // [ 5, 3, 2, 7, 6 ]
```

Process Each Item

- We may want to perform some action on each item
 - Display on screen
 - Add to an html table
 - Send as a email

```
const each=function(list, action){  
    for(let value of list)
```

Action callback

- Take
 - Takes one object from the list

```
const each=function(list, action){
    for(let value of list)
        action(value);
};
```

```
const titles= document.getElementById( 'title');      //returns access to some <ul></ul>

each(books,   book=>  {
    titles.innerHTML+=`<li>${book.title}</li>`;
});
```

Action callback

- Take
 - Takes one object from the list
- Returns
 - nothing

Function returning function

Thursday, December 30, 2021 11:31 AM

- A function may return another function
- The function that is returned may be
 - Global / in depended function
 - Local function

Function Returning global function

```
const plus = function (a,b) {return a+b;};  
const minus= (a,b)=> a-b;  
  
const selectOperation= function (symbol) {  
  
    const multiply=function (a,b) {return a*b; };  
    const divide=(a,b)=>a/b;  
  
    if(symbol==='+')  
        return plus;  
  
    else if (symbol==='-')  
        return minus;  
  
    else if(symbol=='*')  
        return multiply;  
  
    else if(symbol==='/')  
        return divide;  
  
}  
  
//what does this code return?  
const a1 = selectOperation("+"); //it returns the plus function  
  
console.log('typeof a1',typeof a1);  
console.log('a1.name',a1.name);  
console.log('a1==plus',a1==plus);  
  
const a2= selectOperation('+'); //it also returns the plus function  
console.log('a1==a2',a1==a2);  
                                //true
```

const plus = function (a,b) {return a+b;} ← Refers to a global object that is not created everytime you call setOperation
const minus= (a,b)=> a-b;

```
const selectOperation= function (symbol) {  
  
    const multiply=function (a,b) {return a*b; };  
    const divide=(a,b)=>a/b;  
  
    if(symbol==='+')  
        return plus;  
  
    else if (symbol==='-')  
        return minus;  
  
    else if(symbol=='*')  
        return multiply;  
  
    else if(symbol==='/')  
        return divide;  
  
}
```

```
//what does this code return?  
const a1 = selectOperation("*"); //it returns the plus function
```

- This is a local function inside selectOperation
- It is created each time selectOperation is called.
- Two multiply function from two different calls for selectOperation shall be different from each other

```
}
```

```
//what does this code return?  
const a1 = selectOperation("*"); //it returns the plus function  
  
console.log('typeof a1',typeof a1); //function  
console.log('a1.name',a1.name); //multiply
```

```
const a2= selectOperation('*'); //it also returns the plus function  
console.log('a1====a2',a1====a2); //false
```

- Both function
 - do the same job
 - They have same name
 - **They are not same object**
 - They are like two different cars of same model and same spec

Closures

Thursday, December 30, 2021 11:28 AM

A simple function scope

```
function factorial( number ){
    let result=1;
    for(let i=2; i<=number;i++)
        result*=i;
    return result;
}
const fx = factorial(5);
```

- All function parameter and local values are scoped to function
 - number
 - result
 - i
- They are not accessible outside this function
- They are removed from memory (garbage collected) as soon as this function completes
 - number
 - i
 - result
- Will the value of factorial (result) still be available after function call?

When a function returns an inner function

```
function multiplier (number){
    function multiply(value) {
        return number* value;
    }
    return multiply;
}

const m19=multiplier(19); // multiply function
const m23=multiplier(23); // multiply function

console.log('m19(5)',m19(5)); //95
console.log('m23(5)',m23(5)); //115

console.log('m19(7)',m19(7)); //133
console.log('m23(7)',m23(7)); //161
```

- When multiplier(19) is called
 - It returns the multiply function.
 - Multiply function is assigned to m19
 - It will not be removed from memory
 - Normally all local variables are removed from memory
 - Following standard rule number variable should be removed from memory
 - No body is directly referring to it
- m19(5) needs 19 which was stored in 'number' of outer function
 - Traditionally number variable would be removed
 - How will m19(5) work?

Rule of Closure (doesn't follow standard rule)

- As long as a (inner) function is live, all reference available to it as part of its outer function (closure) where it was created shall continue to be available to it.
 - multiply function will keep the 'number' variable live as it needs the access
 - Any value that was available to multiply function because of its closure (multiplier) will remain live and usable as long as multiply function is available.

Why closure is useful?

- How many argument is m19 taking?
 - m19(5)
 - 1. Explicit parameter → 5
 - 2. Closure parameter → 19
- Function can take more parameter than it's official limit.

- We have two instances of multiply function — m19 and m21
- Both are different functions
- Both have their own closures
- **Both will have a different copy of closure scope**
 - number variable

Search revisited

```

function search( list, match) {
  const result=[];
  for( let value of list) {
    if( match ( value ) )
      result.push(value);
  }
  return result;
}

```



- To search a list of books, we pass a match.
- What parameters will match take?
 - A book object

How do I parameterize my match?

- How do I search a book by 'Jeffrey Archer'
- Match doesn't take second parameter
- How do I pass Jefferey Archer?

1. Hard Coded Parameter

- We can hard code second parameter
- But can be parameterize it?

```

function search( list, match) {
  const result=[];
  for( let value of list) {
    if( match ( value ) )
      result.push(value);
  }
  return result;
}

search(books, matchAuthor)

```

```

function matchAuthor( book ) {
  return book.author.toLowerCase().includes("archer");
}

```

Passing a second parameter may not work

```

function search( list, match) {
  const result=[];
  for( let value of list) {
    if( match ( value ) )
      result.push(value);
  }
  return result;
}

```

search(books, matchAuthor)

- Search is not passing the second parameter
- Second parameter will always be undefined.

```

function matchAuthor( book, author ) {
  return book.author.toLowerCase()
    .includes(author.toLowerCase());
}

```

1

```

        return result;
    }

    return book.author.toLowerCase()
        .includes(author.toLowerCase()));

}

```

Solution #1 Closure

```

function search( list, match) {
    const result=[];
    for( let value of list) {
        if( match ( value ) )
            result.push(value);
    }
    return result;
}

search(books, matchAuthor("archer"));

function matchAuthor( author ) {
    return function match( book ) {
        return book.author.toLowerCase()
            .includes(author.toLowerCase());
    }
}

```

- We are not passing outer function to search.
 - We are calling the outer function to get the inner match function
 - We are passing the inner match function to the search
 - Search passes a single parameter 'book' to the match function
 - Match function gets the additional parameter because of the closure.

Search Function

Wednesday, December 29, 2021 5:01 PM

```
this.getBooksByAuthor=function(author){  
    author=author.toLowerCase();  
    let result=[];  
    for(let book of this.db){  
        if(book.author.toLowerCase()===author)  
            result[result.length]=book;  
    }  
  
    return result;  
}  
  
this.getBooksInPriceRange=function(min,max){  
    let result=[];  
    for(let book of this.db){  
  
        if(book.price>=min && book.price<=max)  
            result[result.length]=book;  
    }  
  
    return result;  
}  
  
this.getBooksByRating=function(minRating){  
    let result=[];  
    for(let book of this.db){  
  
        if(book.rating>=minRating)  
            result[result.length]=book;  
    }  
  
    return result;  
}
```

A search Has 5 Steps

1. Create an empty result list
2. Loop through each item in original list
3. Check if current item is a match
4. If yes add it to result list
5. Return result list

How to Search → Search Algorithm

- Steps involved in search

What to Search → Match Algorithm

- What do we want to search
- Note this is not a data, it is a function

```
function search ( list, match){  
    let result=[];  
  
    for( let value of list) {  
        if(match(value)){ ←  
            result.push(value);  
        }  
    }  
  
    return result;  
}
```

- We write common steps of "How To Search" in the search function
- We pass a match function which decided "What to search" as external parameter
- Search function in step 3, calls the match function passed to it.
 - Match function takes one object at a time
 - It return if current object is a match or not
 - True/false
- If match returns true, it includes the current value in the result

Assignment 3.3

Wednesday, December 29, 2021 5:58 PM

- Create A Web Page to display a list of Books
 - Create a Table like this

Search 

Cover	Title	Author	Price	Rating
	The Accursed God	Vivek	399	4.1
	My Experiments with Truth	Mahatma Gandhi	151	4.8

Drop down

Completed
Aman Kumar
Yash
Abhay
Chirag
Rishu
Ashutosh
Karthik
Syed Amir

ISBN
Author
Title

Attaching Function to Constructor prototype

Thursday, December 30, 2021 12:57 PM

- We may have functions related to existing object types such as
 - String
 - Array
- We can attach our functions to prototype of these objects

```
Array.prototype.search=function( match){  
    let result=[];  
    for(let value of this){  
        if(match(value))  
            result.push(value);  
    }  
    return result;  
}  
  
Array.prototype.print=function(header){  
    console.log(header);  
    for(let value of this){  
        console.log(value);  
    }  
}  
  
Array.prototype.each=function(action){  
    for(let value of this)  
        action(value);  
};
```

- Now these functions can be called on any array object using a dot operator
- It is as if they are built-in functions of the array

Some Important Array class built-in methods

Thursday, December 30, 2021 1:00 PM

Method Name	Purpose	Call back parameter	Expected Return from callback	Expected return from function	Our version
forEach	Allow you to perform some operation on all items of the array.	1. Each object one by one 2. Index of the object	Nothing	nothing	each
filter	Returns a new list of matching values	1. Each item one by one 2. Index of current item	Boolean	Array of filtered object matching our callback	search
map	Returns a new list of new objects derived from current object	1. Each item one by one 2. Index of current item	New object that may be of different type than one passed	A list of objects returned by call back	Select
find	Similar to filter but returns only the first matching value	1. Each item one by one 2. Index of current item	Boolean	Returns first matching value or undefined.	--

Assignment 4.1

Thursday, December 30, 2021 1:06 PM

- Take an array of values

```
const array = [2,3,9,11,8,15,4,17];  
  
const primes = _____  
  
console.log(primes) ; // [2,3,11,17]  
  
const alt = _____  
  
console.log(alt) ; // [2,9,8,4]
```

```
function isPrime(number){  
    if(number<2)  
        return false;  
  
    for( let i=2;i<number;i++)  
        if(number%i==0)  
            return false;  
  
    return true;  
}
```

Completed
Aman S
Arpit
Kartik
Aman K
Praveena
Sayed Amir
Sajjan
Aishwarya
Abhay
Rishu
Ashutosh
Burhan
Yash

Unused variable warning

Thursday, December 30, 2021 2:29 PM

```
  console.log(primes) ; // [2,3,11,17]
    'number' is declared but its value is never read. ts(6133)
  //const alt = array.filter  (parameter) number: number
    Quick Fix... (Ctrl+.)
const alt = array.filter ((number,index)=> index%2==0 );
```

- In our application we don't need the first parameter
- But we still need to pass something as a place holder
- We may get an unused-variable-warning
- To avoid this we can simply pass "_" as the parameter name
- It indicates to the linter (code style checker) that this value is not going to be used
- NOTE
 - '_' is a valid variable name
 - It can be used if you need.

```
const alt = array.filter ((_,index)=> index%2==0 );
```



- I don't care about this parameter
- I am not going to use this parameter
- No unused data warning.

Assingment 4.2

Thursday, December 30, 2021 2:33 PM

- Add index feature in our search function
- Rewrite the 4.1 logic to use array.search instead of array.filter
- Push to the git.

Completed
Yash

IIFE

Thursday, December 30, 2021 5:28 PM

IIFE → Immediately invocable Function Expression

Why?

- Avoid global codes

How?

- We put all the logic in some function
- Invoke the function

Shortcut

- We can write a fn as (fn)();
- 1. Wrap the whole function body in a parentheses
- 2. Add another parentheses to invoke it

```
(  
    function () {  
  
        //global logic here  
    }  
  
) (); //invoke it
```

Error Handling in javascript

Monday, January 3, 2022 2:28 PM

- Like most programming language javascript also support try-catch to handle error

Throw

- When a code encounters an error it needs a way to signal its caller that an error has occurred
- Common design may include
 - return
 - Null/undefined
 - False
 - 0
 - 1
 - throw
 - Any javascript object
 - `new Error()`

We prefer throw over return because

- It separates normal information from error signal
 - Return --> information
 - Throw --> error
- Throw is not checked with if, it is checked/handled using catch
- Throw-catch can be present in different functions

Try

- A block of code where you write your normal logic
- Some call from try block may throw an error
- If no error occurs try executes to the end

- Not unlike most languages there can be only one catch after the try.
- Javascript can't specify error

Catch

- Catch must follow a try.
- It receives error which was thrown
- You can perform tasks on catch

- Is always called
 - If no error occurs it is called after try
 - If error occurs it is called after catch
 - If error occurs and not caught it is called after try and before leaving function

finally

Example

```
function doSomething(params){  
    if(!params) ←  
        throw new Error('invalid parameter');  
    ...  
    return params*10;  
}  
  
20  
30  
Error invalid parameter  
  
(we exited the loop because of throw to reach catch)  
  
function client(){  
    let values=[2,3,0,4,8];  
    try{  
        for(value of values){  
            var result= doSomething(value);  
            console.log(result);  
        } →  
        catch(error){  
            console.error('error',error.message);  
        }  
    }  
}
```

How do I ensure other values are also printed.

- We can wrap try-catch inside the loop
- That way even after throw-catch we are still inside the same loop;

```
function client() {
  let values = [2, 3, 0, 4, 8];
  for (let value of values) {
    try {
      const result = doSomething(value);
      //if you reach here then function was a success
      console.log('success', result);
    } catch (error) {
      console.error('error : ', error.message);
    }
  }
}
```

Asynchronous Programming

Friday, December 31, 2021 12:54 PM

- Javascript is a single threaded model
 - It performs the operations on a single thread
 - If one operation is working others must wait
- **Problem Scenario**
 - A Javascript web server may handle only one request at a time
 - If one client is interacting others must wait.
 - Imagine! There is a queue to do a google search and you need to wait for your turn!

Asynchronous Programming

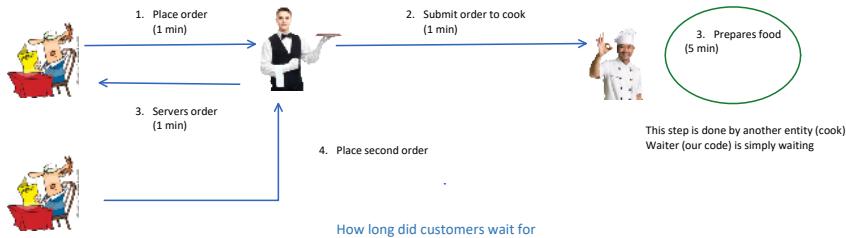
- Javascript follows (recommends) asynchronous programming model
- Most of the time your application needs to interact with external systems
 - Data base server
 - Web server
 - API server
 - OS to read/write files
 - Updating the response on the screen
- All these operations are slow compared to CPU processing of your code
- Your code may have to wait till the external system completes its work
- This wait time can be utilized by performing some other task
 - Example
 - While waiting for database server to provide request for first client we may use the time to take request from the other client.

Food Joint Scenario

- Let us assume we have one waiter and one cook in a food joint
- We may still have multiple customers

Synchronous Scenario

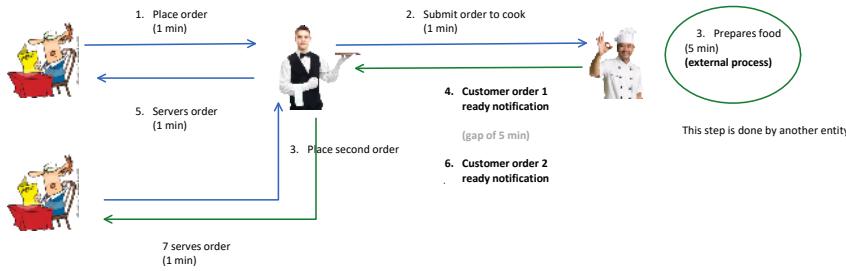
- Assumption
 - taking an order takes 2 min
 - Cooking takes 5 min



How long did customers wait for

Customer	Order Place Wait Time	Order Serve Wait Time
1	0	8 (min required)
2	8	16

Asynchronous Approach



How long did customers wait for

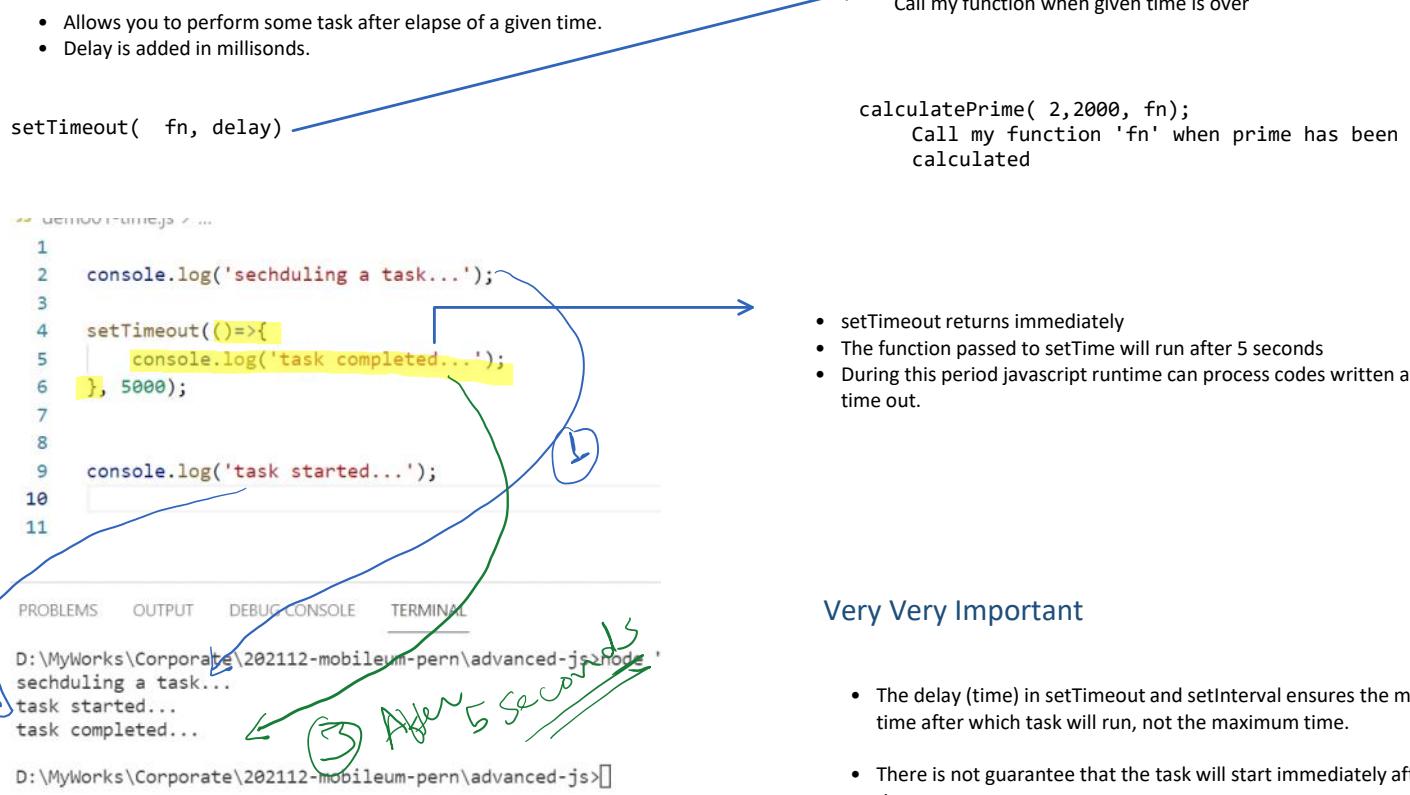
Customer	Order Place Wait Time	Order Serve Wait Time
1	0	8 (min required)
2	2	11

Asynchronous callback

Friday, December 31, 2021 2:32 PM

Primitive Asynchronous operations

setTimeout



- setTimeout returns immediately
- The function passed to setTime will run after 5 seconds
- During this period javascript runtime can process codes written after set time out.

Very Very Important

- The delay (time) in setTimeout and setInterval ensures the minimum time after which task will run, not the maximum time.
- There is not guarantee that the task will start immediately after given time.
- If some other task is running, timed task will run only after they finish.

setInterval

- Set interval executes a task **repeatedly** after a given interval
- It has similar design as setTimeout
 - setTimeout works after given time only once
- setInterval runs repeatedly

setInterval(fn, 1000)

- This functions for the first time after 1 second
- Then the function is repeated after every 1 second

Endless Repetition

- By default setInterval would become endless.
- We can stop the repetition by calling **clearInterval()**

clearInterval

- clearInterval returns an id
- We must pass the id to clearInterval

[clearTimeout](#)

- clearTimeout is like clearInterval
- It can clear a setTimeout call
- [Why do I need to clear a setTimeout?](#)
 - We can clear a timeout before it gets a chance to run.

Asynchronous Use cases

Tuesday, January 4, 2022 9:25 AM

1. Call back my function when
 - a. Service completes the entire work
 - b. Service encounters an error
 - c. Service completes a phase of work!
2. Real world scenario
 - Call my function
 - once we receive data from database
 - We don't wait for internal database processing
 - when we completed reading a file
 - We don't wait while system is getting byte by byte data from file
 - Aside
 - ◆ We may be interested block by block data
 - ◆ Block size may be 8K?
 - When we received data from a web site
 - We may encounter 404 error while getting data from server
 - ◆ Not found
 - Ball by ball cricket score update
 - We don't know when the next ball will be thrown
 - Cricket score update once every five min
 - When user clicks a button
 - We don't know when will user click it.
 - A particular time every day

How does it work?

1. We pass a function to the service.
2. Service calls our function
 - a. On successful completion
 - b. On error
 - c. On some other occasion
 - Milestone reached!

Different styles of passing callbacks

User defined Styles

- When the calculation is over or it encounter an error, the callback is called
- Callback must be able to identify if it is success or failed.
- This can be done by planning

NodeJS style

- Callback should take two parameter
 1. Error
 - Should be null in success case
 2. Result
 - May be error details included here
 - Normally null.

Alternative

- Pass two callback function. First for success and second for failure

```
findPrimes(2,100, (error,result)=>{  
    if(error)  
        handleError(error);  
    else  
        handleSuccess(result);  
});
```

```
findPrimes(2, 100,
```

- normally null.

Alternative

- Pass two callback function. First for success and second for failure

```
findPrimes(min, max, success, failure);
```

```
findPrimes(2, 100,  
success=>{ handleSuccess( success ),  
error=> {handleError(error) }  
});
```

Promise

- Also uses the callback.
- But it passes the callback differently

```
findPrimes( min, max )  
.then(success)  
.catch(failure);
```

- Promise is ES2015 standard
- Better support
- Consistent design in both client and service
- Support for `async await` keyword.
- Library to resolve multiple Promises

Event

- Also uses the callback
- Passes it differently

```
findPrimes( min,max)  
.on("DONE", result=>handleResult(result))  
.on("ERROR", error=>handleResult(error));
```

- NodeJS only features
 - Needs Nodejs 'events' module
- Advantage
 - Can send any user defined event
 - We decide our own event name which could be anything
 - Can send same event multiple times
 - Can be used for continuous communication rather than on finished only
 - We can data in smaller unit
 - Can give progress update
 - Both client and service can 'emit' and listen 'on'
 - Bidirectional communication.

Assignment 5.2

Friday, December 31, 2021 2:49 PM

- Print "Hello JS" 10 times at an interval of 2 second each.
- After the count of 10, it should stop.

Completed	
Aman K	
Yash	
Praveen	
Gulshan	
Aman S	
Sayed	
Karthik	
Ashutosh	
Arpit	
Burhan	
Rishu	
Aishvary	
Mallik	
Abhay	
Arnav	

Solution #1 (NOT WORKING)

```
for(let i=0;i<10;i++){
    setTimeout(()=>{
        console.log('hello world');
    }, 2000);
}
```

- We are scheduling 10 jobs
- Each will print a message 2 second after it is scheduled
- Each job is scheduled immediately
- Each will complete almost together after 2 seconds

Recommendation

- Avoid manual loop
- setInterval itself is like a timed loop.

Solution #2 (working)

```
for(let i=1;i<=10;i++){
    setTimeout(()=>{
        console.log('hello world');
    }, 2000*i);
}
```

- Works
- For-loop is by design synchronous
- Avoid interval scheduling using loops.

Solution #3 (working)

```
let i=0;
const iid=setInterval(()=>{
    i++;
    console.log('hello world');
    if(i==10)
        clearInterval(iid);
}, 2000);
```

- Problem
 - We need separate counter
 - We are doing two independent task in one code
 1. To the main job
 2. Control the interval
 - Violates Single Responsibility
 - They may be required in many cases

Solution#4 (Working if you know total time after which interval should be cleared)

```
const iid=setInterval(()=>{
    console.log('hello world');
}, 2000);

setTimeout(()=>{
    clearInterval(iid);
}, 20100);
```

Asynchronous Function Model

Friday, December 31, 2021 3:39 PM

Normal (Syncrhonous) Functions

```
const findPrimesSync=(min,max)=>{  
    let result=[];  
  
    for(let i=min;i<max;i++){  
        if(isPrimeSync(i))  
            result.push(i);  
    }  
  
    return result;  
}  
  
const result=findPrimesSync(1,100);  
console.log('primes', result);
```

- We start a task
- Task may take a long time. We wait for the task
- We return the result after the work is over.

Alternate way to return result (Callback)

```
const findPrimes = (min, max, cb) =>{  
    let result=[];  
  
    //do some task asynchronously (may be later)  
    //it may take some time  
    //when the work is over  
  
    cb(result);  
  
    //return result;  
    //we may not return anything immediately  
}  
  
findPrimes( 1, 100, result => console.log('primes', result));
```

IMPORTANT: NOT AN ASYNCRHONOUS PROGRAM

- We don't return value using 'return' statement
- We call the supplied callback with the given result
- Client should handle the result passed to the callback

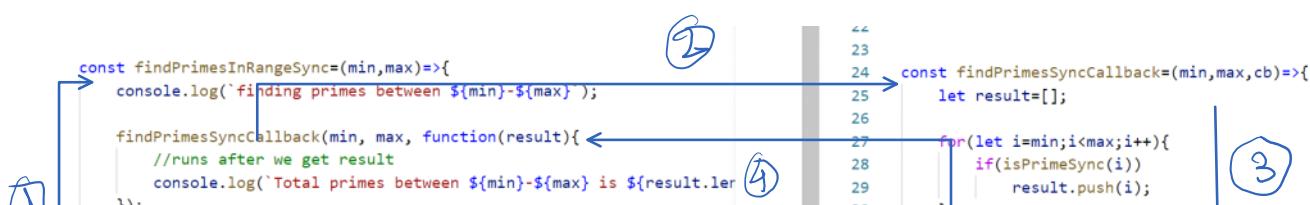
NOTE

- Simply using a callback doesn't make design asynchronous
 - It provides an alternative for direct return.
 - It can be used even in asynchronous code.
- A code will be asynchronous if it returns back before actually completing the job
 - This way application can process statement that comes after function call.
- Asynchronous program should either
 - Call external function
 - Depend on another asynchronous activity like
 - setTimeout
 - setInterval

Here instead of returning the value we are using callback

```
const findPrimesSyncCallback=(min,max,cb)=>{  
    let result=[];  
  
    for(let i=min;i<max;i++){  
        if(isPrimeSync(i))  
            result.push(i);  
    }  
  
    cb(result);  
}  
  
const findPrimesInRangeSync=(min,max)=>{  
    console.log(`finding primes between ${min}-${max}`);  
  
    findPrimesSyncCallback(min, max, function(result){  
        //runs after we get result  
        console.log(`Total primes between ${min}-${max} is ${result.length}`);  
    });  
}
```

Code walkthrough



```

    findPrimesSyncCallback(min, max, function(result){
      //runs after we get result
      console.log(`Total primes between ${min}-${max} is ${result.length}`);
    });
  }

  findPrimesInRangeSync(2,500000);
  findPrimesInRangeSync(2,10000);
  findPrimesInRangeSync(2,10000);
}

26
27   for(let i=min;i<max;i++){
28     if(isPrimeSync(i))
29       result.push(i);
30   }
31
32 cb(result);
33
34
35
36
37
38 module.exports={
39   ...
40   isPrimeSync,
41   findPrimesSync,
42   findPrimesSyncCallback
}

```

Why is the code not asynchronous

- Just by using callback, the code doesn't become asynchronous
- There is nothing that is ensuring 'long-running-task' is not blocking other operation

```

const findPrimesSyncCallback=(min,max,cb)=>{
  let result=[];
  for(let i=min;i<max;i++){
    if(isPrimeSync(i))
      result.push(i);
  }
  cb(result);
}

```

- This function finishes only after the find is completed.
- Asynchronous function should return immediately
 - Its work may finish later.

Simulated Asynchronous Design

- The code is not really an effective programming model
- It is certainly useful to demonstrated how async work may happen.

Code Walkthrough

```

const findPrimesInRangeSync=(min,max)=>{
  console.log(`finding primes between ${min}-${max}`);
  findPrimes(min, max, function(result){
    //runs after we get result
    console.log(`Total primes between ${min}-${max} is ${result.length}`);
  });
}

findPrimesInRangeSync(2,500000);
findPrimesInRangeSync(2,10000);
findPrimesInRangeSync(2,10000);

3+6+9
36
37 const findPrimes=(min,max,cb)=>{
38   let result=[];
39
40   setTimeout(()=>{
41     for(let i=min;i<max;i++){
42       if(isPrimeSync(i))
43         result.push(i);
44     }
45
46     cb(result);
47   },500);
48
49
50
51

```

10. First job finishes and calls cb()

- This will start after 500 ms
- But findPrimes will end immediately
- Now three tasks are scheduled in first 500 ms
- Each job is expected to start after 500 ms

Problem

- The task start is asynchronous.
- We can start all three jobs together.
- But once first task starts to calculate second will not get a chance till first is completed
-

Cooperative Scheduling

Friday, December 31, 2021 4:17 PM

- Make findPrimes 'truly asynchronous' by implementing co-operative scheduling

Cooperative Scheduling

- A cooperative scheduling is a pattern often used in long running task.
- If we know that we will take long time of the CPU we should take few breaks to ensure that other tasks have opportunity to do their work
- We can do that by
 1. Breaking my large work in batches of small work
 2. We should take a small gap between each batch
 - This time can be used by other task
 3. This way no one wait for a very long time.
 - This avoids long starvation

Assignment 5.3

Friday, December 31, 2021 4:17 PM

- Make `findPrimes` 'truly asynchronous' by implementing co-operative scheduling

STEPS

- Start the work after a gap of 10ms
- If we are finding primes in a range <1000 let us do it in one go
- If our range is bigger than 1000 then break it in batches of 1000
 - Batch #1 → all primes between 2-1001
 - Batch #2 → all primes between 1002-2001
 - Batch#3 → all primes between 2001-3000
- We should perform each batch with some gap (setInterval)
 - This gap will help other task to do their work in between
- Return once you have completed.
 - Remember to clear interval before you return.

Cook Demo (Synchronous)

Monday, January 3, 2022 11:35 AM

```
const cook=(clientId, orders, cookingDone)=>{
  const serve={
    clientId,
    items:[]
  };
  console.log(`cook received order from client ${clientId} for ${orders}`);
  for(let order of orders){
    for(let i=0;i<1000000000;i++)
      ; //this is the time taken to prepare the food. represents long running synchronous task
    console.log(`prepared ${order} for client ${clientId}...`);
    serve.items.push(order);
  }
  cookingDone(serve);
}

const consumeFood=(serve)=>{
  console.log(`customer ${serve.clientId} got ${serve.items}`);
}

cook('Vivek', ['chicken soup','chicken tikka', 'butter chiken', 'roti'], consumeFood);
cook('Sanjay', ['sandwitch'], consumeFood);
```

```
cook received order from client Vivek for chicken soup,chicken tikka,butter chiken,roti
prepared chicken soup for client Vivek...
prepared chicken tikka for client Vivek...
prepared butter chiken for client Vivek...
prepared roti for client Vivek...
customer Vivek got chicken soup,chicken tikka,butter chiken,roti
cook received order from client Sanjay for sandwitch
prepared sandwitch for client Sanjay...
customer Sanjay got sandwitch
```

- Represents a long running blocking (synchronous operation)
- Callback is called only after the long running task was over.
- No other operation (asynchronous) can be performed till the work is over.
- Other clients will have to wait till first work is over.

- Note
 - Cook doesn't even receive the order of sanjay, till it completes all the items for vivek

Cook Demo Asynchronous Waiter Syncrhonus Cook

Monday, January 3, 2022 11:35 AM

Synchrhonus Cook

```
const cook=(clientId, orders, cookingDone)=>{
  const serve={
    clientId,
    items:[]
  };
  console.log(`cook received order from client ${clientId} for ${orders}`);
  for(let order of orders){
    for(let i=0;i<1000000000;i++)
      ; //this is the time taken to prepare the food. represents long running synchronous task
    console.log(`prepared ${order} for client ${clientId}...`);
    serve.items.push(order);
  }
  cookingDone(serve);
}
```

- Represents a long running blocking (syncrhonus operation)
- Callback is called only after the long running task was over.
- No other operation (asynchronous) can be performed till the work is over.
- Other clients will have to wait till first work is over.

Asynchronous waiter code

```
const waiter=(clientId, orders, serveDone)=>{
  console.log(`waiter received order from client ${clientId} for : ${orders.join(', ')}`);
  //waiter will send the order after sometime
  setTimeout(()=>{
    //order given to the cook
    cook(clientId, orders, (serve)=>{
      //order received from the cook
      console.log(`waiter serving for ${serve.clientId} : ${serve.items}...`);
      //cook will serve order after 500ms of receiving
      setTimeout(()=>serveDone(serve),500);
    });
    ,500);
  });
}

//Client Logic: interacts with the waiter
const consumeFood=(serve)=>{
  console.log(`customer ${serve.clientId} got ${serve.items}`);
}

waiter('Vivek', ['chicken soup','chicken tikka', 'butter chiken', 'roti'], consumeFood);
waiter('Sanjay', ['sandwitch'], consumeFood);
```

- There are two waits here
 1. Waiter waits before placing order to the cook
 - This allows us to accept other orders before sending everything to the cook
 2. Waiter takes time to serve the food.
 - This is just simulation and not important for our logic

```
waiter received order from client Vivek for : chicken soup, chicken tikka, butter chiken, roti
waiter received order from client Sanjay for : sandwitch
cook received order from client Vivek for chicken soup, chicken tikka, butter chiken, roti
prepared chicken soup for client Vivek...
prepared chicken tikka for client Vivek...
prepared butter chiken for client Vivek...
prepared roti for client Vivek...
waiter serving for Vivek : chicken soup, chicken tikka, butter chiken, roti...
cook received order from client Sanjay for sandwitch
prepared sandwitch for client Sanjay...
waiter serving for Sanjay : sandwitch...
customer Vivek got chicken soup, chicken tikka, butter chiken, roti
customer Sanjay got sandwitch
```

Asynchronous Activity

- Waiter can take the second order immediately after taking the first order.
- Both orders will be submitted in sequence after 500 ms

Synchronous Activity

- Order will be prepared by the cook in sequence and syncrhonously.
- Second customer must wait for their small order till the big order is served.
 - Cook doesn't receive the second order till the first order is fully prepared.

Unexpected side effect (Code only. Not real world)

- Since waiter delayed in serving order (second Timeout in waiter) this time was used by cook to

Unexpected side effect (Code only. Not real world)

- Since waiter delayed in serving order (second Timeout in waiter) this time was used by cook to start the second order synchronously
- In real world cook and waiter are two separate workers
- In javascript there is no separate worker (thread)
 - They share the same space.
 - Only one can really work at a time.
 - In long picture they may appear to be working together
 - One must free CPU before other can work
- COOK doesn't receive the second order till the first order is fully prepared.

Asynchronous cook+waiter

Monday, January 3, 2022 12:06 PM

Asynchronous Waiter

Code unchanged. See previous page.

Asynchronous cook

```
const cook=(clientId, orders, cookingDone)=>{  
  const serve={  
    clientId,  
    items:[]  
  };  
  console.log(`cook received order from client ${clientId} for ${orders}`);  
  
  setTimeout(()=>{  
    for(let order of orders){  
      console.log(`prepared ${order} for client ${clientId}...`);  
      serve.items.push(order);  
    }  
    cookingDone(serve);  
  },2000*orders.length); //cook takes sometime to prepare the order  
}  
}
```

- Note:

- We replaced loop with timeout
 - Loop blocks cpu
 - Timeout frees cpu from this function
- We have simulated a delay by assuming 2sec per order item.
- Larger the order more time it takes to prepare
 - $2000 * \text{order.length}$
- Shorter order will be served earlier

```
waiter received order from client Vivek for : chicken soup, chicken tikka, butter chiken, roti  
waiter received order from client Sanjay for : sandwitch  
cook received order from client Vivek for chicken soup,chicken tikka,butter chiken,roti  
cook received order from client Sanjay for sandwitch  
prepared sandwitch for client Sanjay...  
waiter serving for Sanjay : sandwitch...  
customer Sanjay got sandwitch  
prepared chicken soup for client Vivek...  
prepared chicken tikka for client Vivek...  
prepared butter chiken for client Vivek...  
prepared roti for client Vivek...  
waiter serving for Vivek : chicken soup,chicken tikka,butter chiken,roti...  
customer Vivek got chicken soup,chicken tikka,butter chiken,roti
```

- Cook received both orders.
- It decided
 - Second order will take 2 seconds
 - First order will take 8 seconds
- Second order gets prepared first
- First order will takes it's time

Problem

- Simulation is flawed.
- Food preparation takes no time
- Cook simply delays preparation by 8 seconds in second order
- After second order is taken it is prepared immediately

Assingment 6.1

Monday, January 3, 2022 12:13 PM

- Fix the bug in assignmen **demo03-async-waiter-and-cook.js**
- Each items should be prepared with a gap to 2 seconds

Async cook (NodeJS style)

Monday, January 3, 2022 12:50 PM

```
const cook=(clientId, orders, cookingDone)=>{
  const serve={
    clientId,
    items:[]
  };
  console.log(`cook received order from client ${clientId} for ${orders}`);
  let i=0;
  const iid=setInterval(()=>{
    if(orders.length==0){
      clearInterval(iid);
      return cookingDone(new Error('No order specified...'));
    }

    let order= orders[i];
    console.log(`prepared ${order} for ${clientId}`);
    serve.items.push(order);
    i++;

    if(i==orders.length){
      //all orders are now ready
      clearInterval(iid);
      cookingDone(null, serve); //null indicates no error
    }
  },2000);
}
```

```
const waiter=(clientId, orders, serveDone)=>{
  console.log(`waiter received order from client ${clientId} for : ${orders.join(', ')}`);
  //waiter will send the order after sometime
  setTimeout(()=>{
    //order given to the cook
    cook(clientId, orders, (error,serve)=>{
      if(error){
        serveDone(error,clientId);
      } else {
        //order received from the cook
        console.log(`waiter serving for ${serve.clientId} : ${serve.items}...`);
        //cook will serve order after 500ms of receiving
        setTimeout(()=>serveDone(null,serve),500);
      }
    });
  },500);
  //waiter may submit the order to cook after a while
  //this time can be used for taking other order
}
```

```
const consumeFood=(error,serve)=>{
  if(error)
    console.log(`error in order for ${serve.clientId}`);
  else
    console.log(`customer ${serve.clientId} consuming ${serve.items}`);
}
```

```
66  waiter('Vivek', ['chicken soup','chicken tikka', 'butter chicken', 'roti'], consumeFood);
67  waiter('shivanshi',[],consumeFood);
68  waiter('Sanjay', ['sandwitch','salad'], consumeFood);
69
70
71
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
D:\MyWorks\Corporate\202112-mobileum-pern\advanced-js>node "d:\MyWorks\Corporate\202112-mobileum-pern\waiter-cook-demo\demo04-async-waiter-and-cook.js"
waiter received order from client Vivek for : chicken soup, chicken tikka, butter chicken, roti
waiter received order from client shivanshi for :
waiter received order from client Sanjay for : sandwitch, salad
cook received order from client Vivek for chicken soup,chicken tikka,butter chicken,roti
cook received order from client shivanshi for
cook received order from client Sanjay for sandwitch,salad
prepared chicken soup for Vivek
error in order for shivanshi
prepared sandwitch for Sanjay
prepared chicken tikka for Vivek
prepared salad for Sanjay
waiter serving for Sanjay : sandwitch,salad...
```

Asynchronous Code Design

- We divide long task in small units
- With setInterval we wait between each unit
- Here cook after preparing one of many dishes will wait for sometime
 - This time can be used by other logic asynchronously
 - This is an example of co-operative scheduling
- Cook instead of using a single block of 8 seconds is using 4 blocks of 2 seconds
- Between each block other code may have time to run

Nodejs style callback

- First parameter represents error
 - Null in case of no error
- Second parameter is the correct result.
 - It can also be error details in case of error

Important Note

- this is a common pattern in NodeJS. It is not a feature of NodeJS
- It can also be used with web-based application.

- Note items are prepared asynchronously.
- After first order of vivek it prepares the order of sanjay
- Shorter orders are delivered faster
- Error is handled in case order contains no item

```
error in order for shivanshu
prepared sandwitch for Sanjay
prepared chicken tikka for Vivek
prepared salad for Sanjay
waiter serving for Sanjay : sandwitch,salad...
customer Sanjay consuming sandwitch,salad
prepared butter chiken for Vivek
prepared roti for Vivek
waiter serving for Vivek : chicken soup,chicken tikka,butter chiken,roti...
customer Vivek consuming chicken soup,chicken tikka,butter chiken,roti
```

item

Domain (Buisness) Rule Problem

- Ideally an empty order should be rejected by waiter and it should never reach cook
- Credit: Aman Kumar!

D:\MyWorks\Corporate\202112-mobileum-pern\advanced-js>[]

Assingment5.3 solution

Monday, January 3, 2022 1:31 PM

```
const findPrimeRange=(min,max,cb)=>{
    setTimeout(()=>{
        if(min>=max)
            return cb(new Error(`Invalid range (${min}-${max})`));
        let result={min,max,primes:[]};
        let lo=min;
        let hi=Math.min(max, lo+1000); //max or lo+1000 whichever is greater

        const iid=setInterval(()=>{
            //find primes between lo and hi
            for(let i=lo;i<hi;i++){
                if(isPrimeSync(i))
                    result.primes.push(i);
            }
        });

        if(hi==max){
            //work is done
            clearInterval(iid);
            return cb(null,result); //example {min:0, max:10, primes:[2,3,5,7]}
        }

        //reset the range
        lo=hi;
        hi=Math.min(max,lo+1000);
    },100);
}
```

- We make a small batch of no more than 1000 iteration
- This code runs synchronously
- But it is too small to block for a long time
 - It's synchronous but not long running

- We run multiple small batch with an interval.
- This interval allows other waiting jobs a chance to work
- This is co-operative scheduling

- Batch is recalculated for next interval

IMPORTANT!

- Make sure you know when work is over.
 - Clear interval
 - Call the callback

Promise (ES2015)

Monday, January 3, 2022 2:26 PM

Traditional Async Problems

- A callback based async programming is just a convention not a language feature.
- Different library may use a different style of callbacks
 - Example nodejs expects to pass
 - Error
 - Result
 - Some library may pass
 - Result
 - Error
 - Only result (no error)
 - Single value like { status: error/success value: ... }

ES2015 Promise

- A language support for standardizing asynchronous code.
- Supported out of box in ES2015 runtimes
 - NodeJS
 - Most modern browsers

What is a Promise?

- A future tense.
- A commitment that I will finish some task (and return result) at a future point in time

ES2015 Promise

- A built-in Javascript Object to represent a future return.
- Returned by the asynchronous function immediately
- Client then configure callback that will be called
 - On success
 - On Failure

Why Promise?

- A standard way that all library can use
- Has separate callbacks for success and failure

How to Create Asynchronous Function with Promise

- A Promise takes a callback that takes two callback
 - resolve
 - Call `resolve` to return successful result
 - reject
 - Call `reject` to return error
 - NOTE:
 - We return Promise Immediately
 - This function takes no callback
- ```
function promiseSomething() {
 return new Promise(function(resolve, reject) {
 //do your asynchronous work here
 setTimeout(()=>{
 if(some_error_condition)
 reject(new Error('something went wrong'));
 else
 resolve(result);
 },1000);
 });
}
```

## How to Consume Promise in Client Function

- The client received promise object immediately
  - Client should now configure to callbacks
    - `then`
      - To handle resolved value
    - `catch`
      - To handle rejected value
- ```
const promise = promiseSomething();

//you get promise object immediately
//Now say how do you want to handle success and failure

Promise
.then( result => console.log('result', result)
      .catch( error => console.error('error',error.message));
```

Important!

- We must chain `then` and `catch`
- `Then` returns a different promise which is chained to `catch`.

Promise vs Normal Callback

Normal Callback

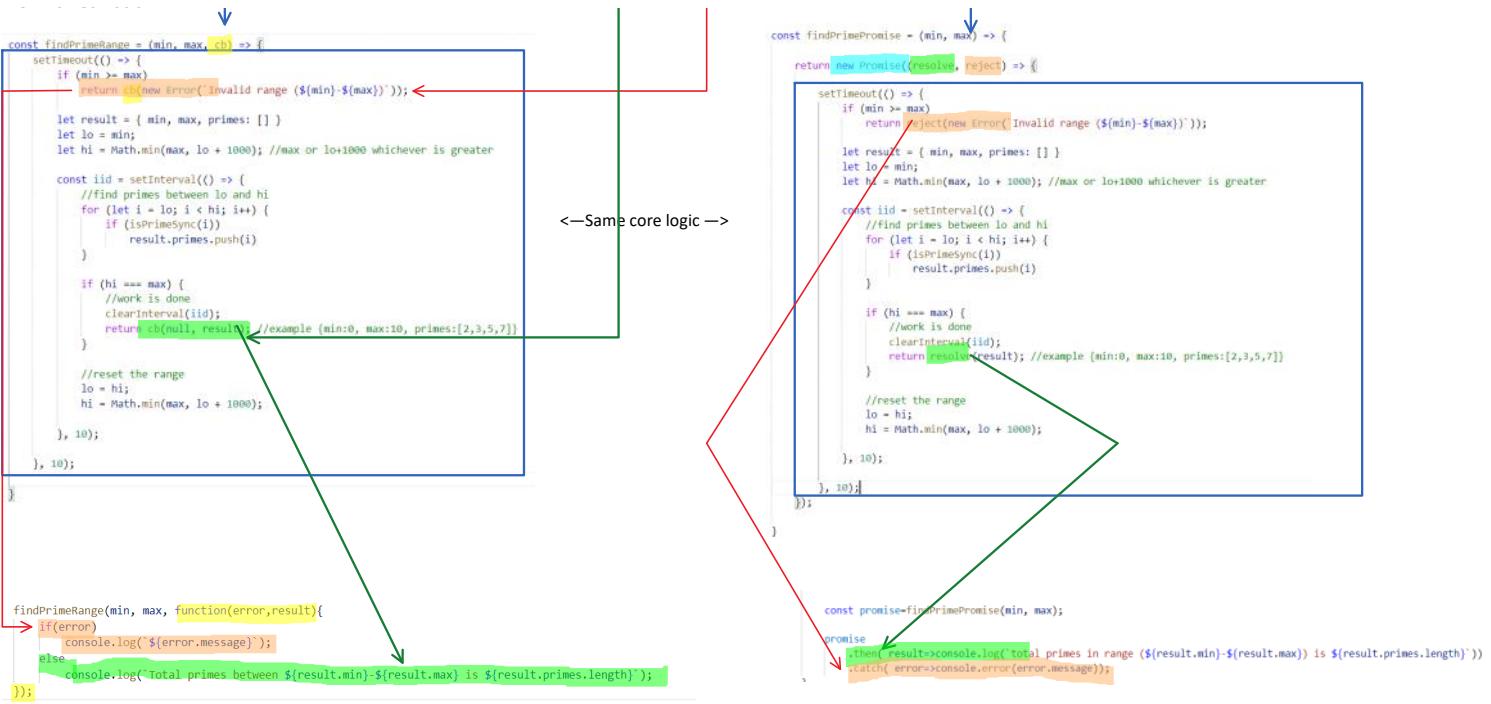
```
const findPrimeRange = (min, max, cb) => {
    setTimeout(() => {
        if( min >= max )
            return cb( new Error('invalid range ${min}-${max}') );
        else
            cb( null, result );
    },1000);
}
```

- Callback is passed as parameter here
 - Same callback is for success and error

Promise

```
const findPrimePromise = (min, max) => {
    return new Promise((resolve, reject) => {
        setTimeout(() => {
            if( min >= max )
                reject( new Error('invalid range ${min}-${max}') );
            else
                resolve( result );
        },1000);
    });
}
```

No callback passed to function. Function returns Promise



async - await keyword

- Javascript provides special keywords **async-await** to handle promise easily.
- It works with Promise
- Also uses try-catch block
- Consider our `findPrimesPromise` code
- Creation of Promise doesn't change
- Client code will change.

Consuming Promise using `then()/catch()`

```

function findPrimesInRange(min,max){
  console.log(`finding primes between ${min}-${max}`);
  const promise=findPrimePromise(min, max);

  promise
    .then(result=>{
      console.log(`total primes in range
      ${result.min}-${result.max}
      is ${result.primes.length}`);
    })
    .catch(error=>console.error(error.message));
}

```

Consuming Promise using `async-await`

- Any function that awaits must be marked `async`
- An `async` function returns a Promise implicitly

```

async function finPrimesInRange (min, max) {
  console.log(`finding primes between ${min}-${max}`);
  try {
    const result = await findPrimePromise(min, max);
    console.log(`total primes in range
      ${result.min}-${result.max}
      is ${result.primes.length}`);
  } catch (error) {
    console.error(error.message);
  }
}

```

- await**
- Waits for promise to resolve. It returns another promise
 - You get back the result but on a future time frame.
 - What follows await it what you would typically write in `then()`
 - If promise is rejected it will be considered as an exception and we handle in `try catch`.
 - `await` can't be written globally. It must be used inside a `async` function only
 - When I get the result (`Promise`) to print it on `console`
 - I am making another promise based on current promise

Important Note

- An `Async` function returns a Promise
- Anything you return from `async` function is like "resolve"
- Any exception thrown from `async` function is like a "reject"

Converting setTimeout to Promise

- setTimeout is by default works like a regular call back
- A regular call back can't be used with async/await
- But we can write our own Promise version of setTimeout

Different Call approach

Monday, January 3, 2022 4:47 PM

Task	Synchrhonous Approach	Async Callback Approach	Promise
What happens when we call it?	<ul style="list-style-type: none">• We wait for the task to complete.• Blocking call	<ul style="list-style-type: none">• We don't wait for function to be over.	<ul style="list-style-type: none">• We don't wait for function to be over.
When we return	<ul style="list-style-type: none">• After the task completes	<ul style="list-style-type: none">• Immediately	<ul style="list-style-type: none">• Immediately
What we return	<ul style="list-style-type: none">• Actual result	<ul style="list-style-type: none">• Nothing	<ul style="list-style-type: none">• Promise
How function return result	<ul style="list-style-type: none">• Return statement	<ul style="list-style-type: none">• Calls our callback with value/error	<ul style="list-style-type: none">• Calls reject/resolve function given to Promise.
How do we get to know result	<ul style="list-style-type: none">• It returns result	<ul style="list-style-type: none">• Calls back my function provided during call	<ul style="list-style-type: none">• Calls callback provided to promise for success (.then)
How do we get to know failure	<ul style="list-style-type: none">• Should throw exception	<ul style="list-style-type: none">• Should include error in callback• Nodejs callback takes error as first parameter• May vary in different library	<ul style="list-style-type: none">• Calls callback provided to promise for failure. (.catch)

Assignment 6.2

Monday, January 3, 2022 3:12 PM

- Create a `findPrimePromise` function that handles Prime finding using Promise
- Define the function and the client to test it.

Assignment 6.3

Monday, January 3, 2022 5:19 PM

- Write a promise version of setTimeout that can work with async/await then/catch
- Following code should print hello and world with a gap of 2 seconds

```
//Client code  
  
const delayedMessage= async ()=>{  
    console.log('Hello');  
    await delay(2000);  
    console.log('World');  
}
```

Solution

```
...  
module.exports= (time)=>{  
    ...  
    return new Promise((resolve, reject)=>{  
        setTimeout( ()=> resolve(), time);  
    });  
};|
```

1. Our function takes the delay time
2. We return a Promise
3. Inside Promise callback we call setTimeout
4. We resolve the promise after set time
 - a. There is no data needed in resolve
5. There is no situation where we will reject it.

Simplified (Or made more complex?)

```
...  
module.exports = time => new Promise(resolve => setTimeout(resolve, time));
```

- Function takes time and returns new Promise
- New promise can only be resolved
 - We are not passing reject parameter
- setTimeout will call my resolve function
 - No need of creating extra arrow function

What are these functions returning

Tuesday, January 4, 2022 11:51 AM

```
const delay=require('../utils/delay');
const sum=(...values)=> {
  let result=0;
  for(const value of values){
    delay(1000);
    result+=value;
  }
  return result;
}

const x=sum(1,2,3,4);
console.log('typeof x',typeof x);
console.log('x',x);
```

```
const delay=require('../utils/delay');
const sum=(...values)=> {
  let result=0;
  for(const value of values){
    delay(1000).then(_=>{result+=value});
  }
  return result;
}

const x=sum(1,2,3,4);
console.log('typeof x',typeof x);
console.log('x',x);
```

```
const delay=require('../utils/delay');

const sum=async (...values)=> {
  let result=0;
  for(const value of values){
    await delay(1000).then(_=>{result+=value});
  }
  return result;
}

const x=sum(1,2,3,4);
console.log('typeof x',typeof x);
console.log('x',x);
```

Promise of future result

what is x?

10

when will we get answer?

Immediately

What happened to the delay?

- Delay will be resolved 4 times after one second
- But we don't care about that delay
 - We are not awaiting
 - Delay is making a promise
 - I am not waiting for its resolution

Note

- Our program will still end after delay is over
- Our code may not wait for promise to resolve, the JS runtime will do so.

What is x?

0

When will we get answer?

Immediately

What happened here

- Values will be summed after the delay
- But some function is not waiting for delay
- It returns immediately
- No sum has happened yet
- You get 0

What is x

Promise

- Any async function always returns a promise
- It may appear that we are returning result
- But we are not returning result immediately
- We are promising that we will return result later.

When will this function return?

Immediately

- With a promise

When will we get the result?

- After 4 seconds

How will we get the result

- Await/then

```
async-test > js demo03-sum.js ...
1  const delay = require('../utils/delay');
2
3  const sum = async (...values) => {
4
5    let result = 0;
6
7    for (const value of values) {
8      await delay(1000).then(_=>{result+=value});
9    }
10   return result;
11 }
12
13 const promise = sum(1, 2, 3, 4);
14
15 promise.then(result => {
16   console.log('promise', promise);
17   console.log('result', result);
18 });
19
20 console.log('promise', promise);
21 console.log('waiting for promise to resolve');
22 |
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
D:\MyWorks\Corporate\202112-mobileum-pern\advanced-js>node "d:\MyWorks\Co
promise Promise { <pending> }
waiting for promise to resolve
promise Promise { 10 }
result 10
D:\MyWorks\Corporate\202112-mobileum-pern\advanced-js>

1. Function returns a promise
2. We set up then for future. It doesn't run immediately
 - We just tell what you should do when promise is resolved in future
3. These two lines of code runs immediately
4. Promise is resolved after 4 seconds

Bound and Unbound Method

Thursday, January 13, 2022 10:51 AM

```
3 const person={  
4   name:'Vivek'  
5 };  
6  
7 const introduce=function(){  
8   console.log(`Hi I am ${this.name}`);  
9 }  
10  
11 person.introduce=introduce;  
12  
13  
14 person.introduce(); //what wil this print? ---> this points to 'vivek'  
15  
16  
17 introduce(); //this doesn't point to person  
18  
19  
20 const i2=person.introduce;  
21  
22 i2(); //does this point to person? what are we assiging?  
23  
24  
25 const onlineForumIntroduction=function(fn){  
26   fn();  
27 };  
28  
29 onlineForumIntroduction(person.introduce);
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
D:\MyWorks\Corporate\202112-mobileum-pern-01\advanced-javascript>node "d:\MyWorks\Corporate\4.js"
Hi I am Vivek
Hi I am undefined
Hi I am undefined
Hi I am undefined

Also applies to class Method

```
2  
3 class Person{  
4   constructor(name){  
5     this.name=name;  
6   }  
7  
8   introduce(){  
9     console.log(`Hi I am ${this.name}`);  
10 }  
11  
12 }  
13  
14 const person=new Person('Vivek');  
15  
16 person.introduce(); //what wil this print? ---> this points to 'vivek'  
17  
18 const i2=person.introduce;  
19  
20 i2(); //does this point to person? what are we assiging?  
21  
22  
23 const onlineForumIntroduction=function(fn){  
24   fn();  
25 };  
26  
27 onlineForumIntroduction(person.introduce);
```

Real world implication (React Code)

```
3 class Cell extends React.Component {  
4  
5   handleClick(){  
6     console.log('cell clicked',this.props.value);  
7   }  
8  
9 }  
10  
11  
12 }
```

Default Function and this relationship

- A normal non-arrow function always has a 'this' context
- But it is a dynamic idea.
 - 'this' depends on how a function called
- 14. function is called with 'person' as invoker
 - a. Function treats 'person' as 'this'
- 17. When a global function is called without invoking context it uses
 - a. Global context in nodejs
 - b. Window in browser based app
- 20. When we assign an object method to other variable only function reference is assigned.
 - It loses the 'person' as 'this'
- 22. i2() is called without 'person.'
 - this doesn't refer to the function
- 29. **MOST IMPORTANT**
 - When we pass 'person.introduce' to another function we are actually passing 'introduce' without 'person'
- 26. Now when the other function call's introduce() it is called without proper 'this'
 - This is not available

Takeaway

- Whenever you pass an object method as a call back, 'this' will not be available when the function is called.
- The same idea is application for
 - Prototype methods
 - Class methods
- 20. i2() is called without 'person' and it doesn't have 'this'
- 27. We are passing 'introduce' not 'person'.
 - It loses 'this'

```

8
9
10
11
12 render(props){
13
14     let value=this.props.value;
15
16
17
18     return (
19         <button className='cell' onClick={this.handleClick}>
20             {value}
21         </button>
22     );
23 }
24
25
26
27
28
29 export default Cell;

```

- We are passing 'handleClick' not 'Cell' to the button click handler
 - When button click calls this function it loses 'this'

Bound Methods

- We have special syntax in ES5+ that can create a bound method
- A bound method is a method which is permanently attached with certain value for
 - 'this'
 - Some of its parameters
- A bound method never loses those association
 - Even when used without dot operator

```

JS demo02.js U JS demo01b-bound.js U X JS demo01.js U
boundmethods > JS demo01b-bound.js > ...
2
3 const person={
4   name:'Vivek'
5 };
6
7 const introduce=function(){
8   console.log(`Hi I am ${this.name}`);
9 }
10
11 person.introduce=introduce.bind(person); //now we 'bind' introduce method's 'this' with 'person'
12
13 person.introduce(); //what will this print? ---> this points to 'vivek'
14
15 console.log('introduce==>person.introduce',introduce==>person.introduce);
16
17 introduce(); //this doesn't point to person
18
19
20
21 const i2=person.introduce;
22
23 i2(); //does this point to person? what are we assigning?
24
25
26 const onTheForumIntroduction=function(fn){
27   fn();

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```

D:\MyWorks\Corporate\202112-mobileum-pern-01\advanced-javascript>node "d:\MyWorks\Corporate\202112-mobileum-pern-01\advanced-javascript\demo01b-bound.js"
Hi I am Vivek
introduce==>person.introduce false
Hi I am undefined
Hi I am Vivek
Hi I am Vivek
Hi I am Vivek
D:\MyWorks\Corporate\202112-mobileum-pern-01\advanced-javascript>

```

- **bind()**
 - Creates a **new version** of this function
 - In this new "introduce" this is permanently binding to 'person'
 - This relationship can't be undone.
- Note bind creates a new copy of function
- It doesn't effect original copy

Bound Function is class Methods

```

2
3 class Person{
4   constructor(name){
5     this.name=name;
6     this.introduce=this.introduce.bind(this);
7   }

```

9. A class method by default is created as unbound method
6. When object is created we take its unbound method

```

3 class Person{
4     constructor(name){
5         this.name=name;
6         this.introduce=this.introduce.bind(this);
7     }
8
9     introduce(){
10        console.log(`Hi I am ${this.name}`);
11    }
12
13 }
14
15 const person=new Person('Vivek');
16
17 person.introduce(); //what wil this print? ---> this points to 'vivek'
18
19 const i2=person.introduce;
20
21 i2(); //does this point to person? what are we assiging?
22
23
24 const onlineForumIntroduction=function(fn){
25     fn();
26 };
27
28 onlineForumIntroduction(person.introduce);

```

9. A class method by default is created as unbound method
6. When object is created we take it's unbound method and bind it.
 - We need to do this bind for every method that may be used as a call back

Style Sheet

Thursday, December 30, 2021 3:31 PM

- Allows us to customize the looks and feel of HTML Element

1. Element based Style

- We can apply same set of styles to all elements of a particular type

Book's Web

Our Book List

Cover	Title	Author	Price	Rating
	The Accursed God	Vivek Dutta Mishra	399	4.2
	Harry Potter and the Philosopher's stone	JK Rowling	109	4.8

```
# app.css
40 /* if there is any <li> inside .nav */
41 .nav li{
42   display:inline;
43   margin-right:10px;
44   font-weight: bold;
45   text-decoration: none;
46   color: #seagreen;
47 }
```

```
index.html
11 <div>
12   <h1 class='siteTitle'>Book's Web</h1>
13   <ul class='nav'>
14     <li>
15       | <a href="/">home</a>
16     </li>
17     <li><a href="/add.html">Add Book</a></li>
18     <li><a href="/authors">Authors</a></li>
19   </ul>
```

2. Class based style

- A Style can be applied on a particular set of items by using class type
- A class is an attribute that can be applied to an html element
- All items with same class can have same style

Book's Web

Our Book List

Cover	Title	Author	Price	Rating
	The Accursed God	Vivek Dutta Mishra	399	4.2
	Harry Potter and the Philosopher's stone	JK Rowling	109	4.8

```
# app.css
24 h1{
25   color: #mediumseagreen;
26   font-family: 'Segoe UI', Tahoma, Geneva, Verdi;
27   font-size: 20px;
28 }
29 .siteTitle{
30   color: #mediumseagreen;
31   font-size: 30px;
32   text-shadow: -2px -2px 2px black;
33 }
34 .nav li{
35   display: inline;
36   margin-right: 10px;
37   font-weight: bold;
38   text-decoration: none;
39   color: #seagreen;
40 }
```

```
index.html
11 <div>
12   <h1 class='siteTitle'>Book's Web</h1>
13   <ul class='nav'>
```

Conflicts and Resolution

Book's Web

Our Book List

Cover	Title	Author	Price	Rating
	The Accursed God	Vivek Dutta Mishra	399	4.2
	Harry Potter and the Philosopher's stone	JK Rowling	109	4.8

```
# app.css
42 .nav li{
43   display: inline;
44   margin-right: 10px;
45   font-weight: bold;
46   text-decoration: none;
47   color: #seagreen;
48 }
```

```
index.html
11 <div>
12   <h1 class='siteTitle'>Book's Web</h1>
13   <ul class='nav'>
14     <li>
15       | <a href="/">home</a>
16     </li>
17     <li><a href="/add.html">Add Book</a></li>
18     <li><a href="/authors">Authors</a></li>
19     <li>Hello</li>
20   </ul>
```

- This style is applicable on a "li" that is present inside another element with class="nav"
- "li" will have the seagreen color
- All children of this li should also follow this style if they don't have their own style
- But "<a>" tag has its own style
 - Blue underlined text
 - A tag overwrites the li style

localStorage

Friday, December 31, 2021 6:05 PM

- It is a BOM object
- Available on the browser
- It allows you to store key value pairs in browsers internal local storage
- The storage is per site per browser
- Data is stored as key value pair
- Value must be a string.
 - It can't be an object
- We have few basic method to work with localStorage

LocalStorage Methods

```
< -> Storage {length: 0} ⓘ
  length: 0
  [[Prototype]]: Storage
    ► clear: f clear()
    ► getItem: f getItem()
    ► key: f key()
    ► length: ...
    ► removeItem: f removeItem()
    ► setItem: f setItem()
    ► constructor: f Storage()
      Symbol(Symbol.toStringTag): "Storage"
    ► get length: f length()
  [[Prototype]]: Object
```

- Clear
 - Removes all item from the localStorage
- getItem
 - Get the value of given key
 - Undefined if key is not found
- setItem
 - Sets a key value pair
 - If key is present value is overwritten
- removeItem
 - Remove the key-value pair based on the key
- key

LocalStorage with objects

- By default localStorage works only with strings
- If we try to store the object it stores only object.toString() calls
 - [object object]
- To store an complex object in local storage we must convert it to string using JSON.stringify
 - It stores the object as json string
- To get the data back you must call JSON.parse() to get the object

```
> var book={title:'The Accursed God', author:'Vivek Dutta Mishra'}
< undefined
> var bookStr=JSON.stringify(book)
< undefined
> bookStr
< '{"title":"The Accursed God","author":"Vivek Dutta Mishra"}'
> localStorage.setItem('book1' , bookStr);
< undefined
> |
```



Get the data back

```
> const bookData= localStorage.getItem('book1')
< undefined
> bookData
< '{"title":"The Accursed God","author":"Vivek Dutta Mishra"}'
> const book= JSON.parse(bookData)
< undefined
> book
< ▶ {title: 'The Accursed God', author: 'Vivek Dutta Mishra'}
> |
```

Assignment 5.4

Friday, December 31, 2021 6:04 PM

- Complete the book-client app with these features

1. Allow adding of new books

- Create a UI to take important fields
 - Title
 - Author
 - Price
 - Cover
 - Rating
 - Description
- Add the data to your book manager
- Book manager should add this data to localStorage

2. Allow user to delete the books by adding a delete button in the book list screen

3. Allow user to edit the book by adding edit button in book list screen

- When user clicks edit it should take me to edit page where it will edit the book

4. Create a book details page where user can see

- Large picture of the book
- Book title
- Other details (like description etc)
- Create a button in book list to reach here

Search Bar

[Add New Book](#)

Title	Author	Price	Actions
The Accursed God	Vivek Dutta Mishra	399	<u>Details</u> <u>Edit</u> <u>Delete</u>

Take to another screen with book details of a single book

Delete the book

Take to book editor to edit this book

Working with Multiple Files

Thursday, December 30, 2021 5:30 PM

How do I split my NodeJS application codes in multiple javascript files?

Challenge

- Unlike web, we don't have any html file where you will include all the Javascript files.

How we do it on web?

Index.html

```
<script src="file1.js" ></script>
<script src="file2.js"></script>
<script src="file3.js"></script>
```

- Now
 - file2.js can use everything defined in file1.js
 - File3.js can use everything defined in file1.js and file2.js

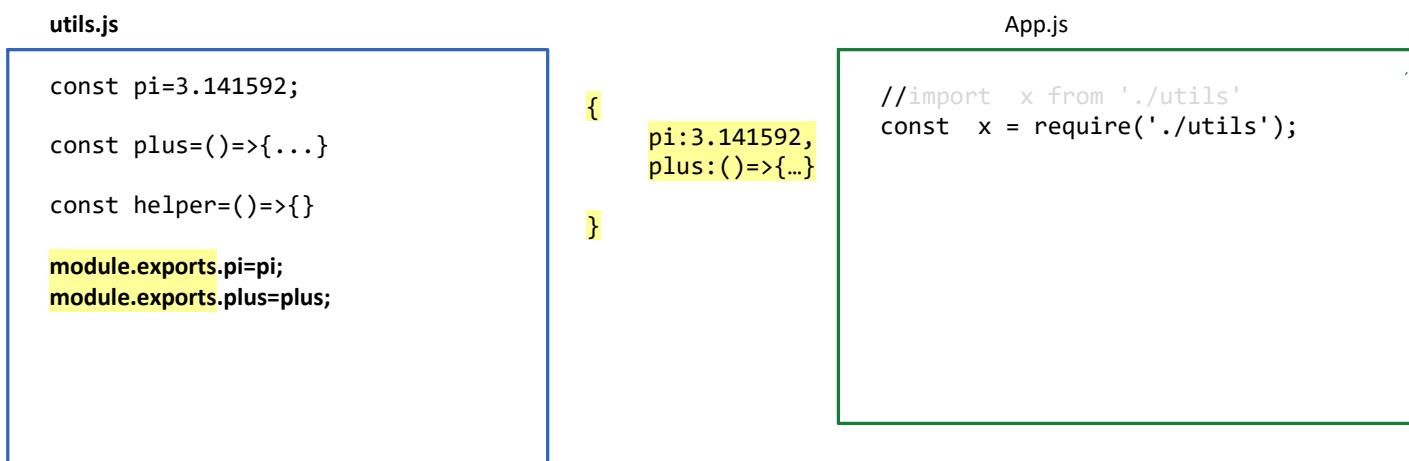
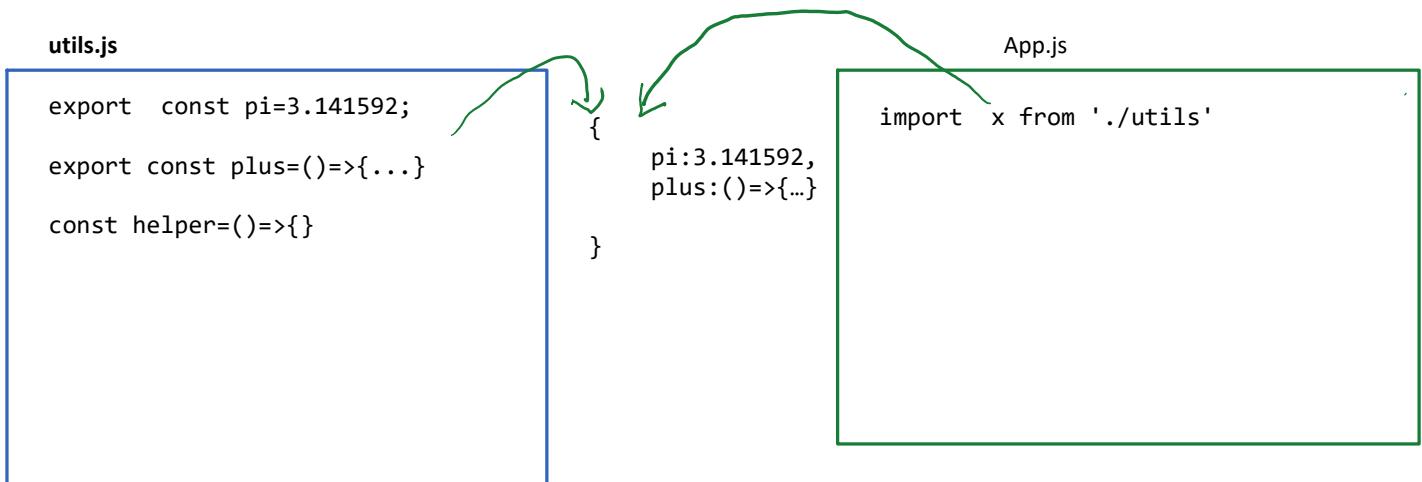
Problems

- We must remember the order in which script should be added.
- All variables from all js file will be added to global scope
 - We can't chose what elements we want other to know (public) and what not (private)

Module Concept

Friday, December 31, 2021 10:07 AM

- We visualize each JS file as a module.
- A module can be considered as a conceptual object we get from the file.
- We export a module from a file
 - A javascript object is created with exported objects
 - This object can be used by other modules that import it.



Module import executes the module script

- When we import a module (ES2015/NodeJS) that module script (JS file) is actually executed
- All code in the file runs (even those that are not exported)
 - Think!
 - If code doesn't run how will functions be created?

Use Case

- Sometimes we have no function to import from a module but we still want to run this script
 1. The module may have some logic that we want to use.
 2. The module may be adding some functionality to another object's prototype.

ES 2015 Syntax

```
import './utils/run-script' ;
```

NodeJS Syntax

```
require('./utils/run-script');
```

ES2015 Module System

Thursday, December 30, 2021 5:38 PM

- ES 2015 introduced the concept of modules
- Every Javascript file will be treated as a module
- A module can
 - represent a set of functionality
 - Decide what they want to **export** outside module
 - Only those elements can be used by other modules
 - A module can **import** functionalities from another module when needed

Example

Named exports and imports

- Consider the below code
- Math.js is a module
- It has three functions
 - plus
 - minus
 - helper
- Two of them are exported
- One is not exported

```
//math.js
export function plus (x,y){return x+y;}
export function minus(x,y){return x-y;}
function helper(){ ...}
```

- Now plus and minus can be used by other modules
- Other modules can't access helper

Importing in another module

```
//client.js
import math from './math.js'

console.log(math); // { plus:function, minus:function}
// we can now use

const result=math.plus(2,3); //works

math.helper(); //fails. There is no helper exported
```

- Note
 - User defined Module name is a relative path
 - It should be included with './' or '..'
 - Now we have a an object called math
 - Math has two functions inside

Popular way to import named exports

- We use object Destructuring to import items

```
//client.js
import {plus, minus} from './math.js'

// we can now use

const result=plus(2,3); //works

helper(); //fails. There is no helper exported
```

2. Default export import

- Sometimes a module has just one "primary" thing to export.
- It can be a default export
- A default export exports object not it's name

```
//services/book-manager.js
```

```
const BookManager=function(){  
  ...  
}
```

```
export default BookManager;
```

Importing a default export

- It can be imported in any name of your choice

```
//client.js
```

```
import BookService from './services/book-manager';
```

- Default export doesn't export name
- Module importing default export can use either the original name or any name it likes

No reference import

- We import a module just to run it.
- Module exports nothing.
- We need to import nothing
- We get empty module object

ES 2015 Syntax

```
import './utils/run-script' ;
```

Module in not supported on any contemporary javascript runtime

- Neither NodeJS nor any browser supports module system out of box.
- We may be able to replicate this idea using third-party-library and frameworks.

NodeJS Module System

Thursday, December 30, 2021 5:38 PM

- NodeJS has its own module system
- It is conceptually similar to ES2015 module system
- It is NOT semantically similar to ES2015 modules system
- Provides the same functionality with a different syntax.
- A module can
 - represent a set of functionality
 - Decide what they want to **export** outside module
 - Only those elements can be used by other modules
 - A module can **import** functionalities from another module when needed
- There is no import/export keyword

Example

Node JS Named exports and imports

- Consider the below code
- Math.js is a module
- It has three functions
 - plus
 - minus
 - helper
- Two of them are exported
- One is not exported

```
//math.js
function plus (x,y){return x+y;}
function minus(x,y){return x-y;}
function helper(){ ...}

module.exports.plus=plus;
module.exports.minus=minus;
```

- Now plus and minus can be used by other modules
- Other modules can't access helper
- **module.exports** is a nodejs feature
- It will not work in browser based application

Importing in another module

```
//client.js

//import math from './math.js'
const math = require('./math');

console.log(math); // { plus:function, minus:function }

//we can now use

const result=math.plus(2,3); //works

math.helper(); //fails. There is no helper exported
```

- Note

- **require** is a nodejs feature
- It is an alternate of import
- It follows same path style
- You don't need to include .js extension

Popular way to import named exports

- We use object Destructuring to import items

```
//client.js

//import {plus, minus} from './math.js'

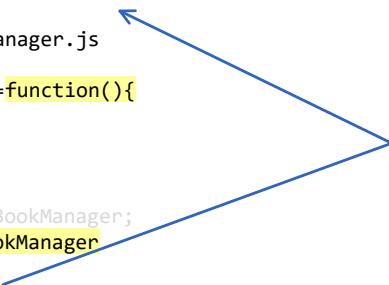
const {plus, minus } = require('./math');
```

```
//we can now use
const result=plus(2,3); //works
helper(); //fails. There is no helper exported
```

2. Default export import

- Sometimes a module has just one "primary" thing to export.
- It can be a default export
- A default export exports object not it's name

```
//services/book-manager.js
const BookManager=function(){
    ...
}
//export default BookManager;
module.exports=BookManager
```



Importing a default export

- Default export doesn't export name
- Module importing default export can use either the original name or any name it likes

- It can be imported in any name of your choice

```
//client.js
//import BookService from './services/book-manager';
const BookService = require('book-manager');
```

Module in not supported on any contemporary javascript runtime

- Neither NodeJS nor any browser supports module system out of box.
- We may be able to replicate this idea using third-party-library and frameworks.

A module actually exports a single Object

- In NodeJS this object is named **module.exports**

If we need to export multiple functions we can do

Approach #1

```
//module.exports -> {}
module.exports.plus=function(){}
module.exports_MINUS=function()
```

- **module.exports** is by default an empty object
- We are adding two functions to this empty object
- We are returning a single object

Approach #2

- We are replacing default **module.exports** object with a new object that contains

Approach #2

```
module.exports= {
  plus: function(){},
  minus:function(){}
}
```

- We are replacing default module.exports object with a new object that contains two function

Approach#3 Named Export

- If we want to return a single object from a module we can replace module.exports with that object

```
function BookManager(){

  this.addBook=...
  ...

};

module.exports.BookManager = BookManager; //--> { BookManager: BookManager }
```

- Module exports an object that contains BookManager function

Approach #4 Default export

```
module.exports = BookManager; //--> BookManager
```

- Module exports the BookManager function directly
 - It is not exporting any other object

No reference import

- We import a module just to run it.
- Module exports nothing.
- We need to import nothing
- We get empty module object

NodeJS Module example

Friday, December 31, 2021 10:26 AM

```
JS book-manager.js U X JS array-utils.js U ...
nodejs-modules > bms > JS book-manager.js > [e] <unknown> ...
31
32 > const BookManager=function(){ ...
67 ...
68
69 module.exports=BookManager; //default export

JS app.js U X
nodejs-modules > JS app.js > ...
1 const bookModule=require("./bms/book");
2 const bookManagerModule=require("./bms/book-manager");
3
4 console.log('bookModule',bookModule);
5 console.log('bookManagerModule',bookManagerModule);
6
7

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
ules\app.js"
bookModule { Book: [Function: Book] } 
bookManagerModule [Function: BookManager]
```

- Note we have imported both named export and default export in the same style
- Both returns the module
- **Named Export**
 - Returns an object that contains my function as property
- **Default Export**
 - My object/function itself is return as module object
 - There is no extra wrapper around it.
 - The object is exported as module.
 - It doesn't retain its name

How to use 'Book' import from named export

Approach #1

- Use module name as prefix

```
nodejs-modules > JS app.js > ...
1 const bookModule=require("./bms/book");
2 const bookManagerModule=require("./bms/book-manager");
3
4 console.log('bookModule',bookModule);
5 console.log('bookManagerModule',bookManagerModule);
6
7 const book1=new bookModule.Book('The Accursed God','Vivek Dutta Mishra', 399, 4.1);
8
9 console.log('book1',book1);
```

Approach#2

- Use destructuring to get methods you need.

```
const {Book}=require("./bms/book");
const bookManagerModule=require("./bms/book-manager");

console.log('bookManagerModule',bookManagerModule);

const book1=new Book('The Accursed God','Vivek Dutta Mishra', 399, 4.1);

console.log('book1',book1);
```

How to use BookManager from default export

- A default export exports a single object and not a module object
- This object has no specific name
- You can use any name at the client side
- **Recommendation**
 - If you know the real name use it.

```
JS book-manager.js U X JS array-utils.js U ...
nodejs-modules > bms > JS book-manager.js > ...
30 ...
31
32 > const BookManager=function(){ ...
67 ...
68
69 module.exports=BookManager; //default export
70
71

JS app.js U X
nodejs-modules > JS app.js > ...
1 const {Book}=require("./bms/book");
2 const BookService=require("./bms/book-manager"); //I can use any variable name here
3
4 const service=new BookService();
5 const book1=new Book('The Accursed God','Vivek Dutta Mishra', 399, 4.1);
6
7 service.addBook(book1);
8 service.printBooks();
```

No Export Module

- We have a module `utils/array-utils`
 - This script adds few methods directly to array prototype
 - All the methods will be now available in all array
 - There is nothing to export from this file.
 - There is nothing to import in the client
 - But this code must run if we want to get these methods in array
 - We can use `require()` without any reference assignment

The screenshot shows a code editor with a file named `app.js`. The code imports utility functions and a Book model, creates a BookService instance, and then runs a script. A blue arrow points from the text "runs the script and adds new methods to Array.prototype" to the line where the service is used. A green arrow points from the same text to the `.select` method call.

```
5 app.js  U X
nodejs-modules > JS app.js > ...
1  require("./utils/array-utils");
2  const {Book}=require("./bms/book");
3  const BookService=require("./bms/book-manager");
4
5  let service=new BookService();
6
7 > service.addBooks(...);
8
9
10 const result=service
11     .getAllBooks()
12     .search(b=>b.price>=300)
13     .search(b=>b.rating>=4)
14     .select(b=> ({title:b.title, price:b
15         .each(console.log);
```

runs the script and adds new methods to `Array.prototype`

Now they can be used in our application with any array

This code will not work in Web Application without external library support

```
nodejs-modules > app.html
  1  <script src="app.js"></script>
JS app.js U X
nodejs-modules > app.js
  1  require('./utils/array-utils');
  2  const {BookService}=require("./bms/entities");
  3  const BookService=require("./bms/book-manager");
  4
  5
```

Aside

- If you need this functionality for the webapplication you may use a third-party library called require.js

<https://requirejs.org/docs/start.html>

NodeJS Module vs ES2015 Module

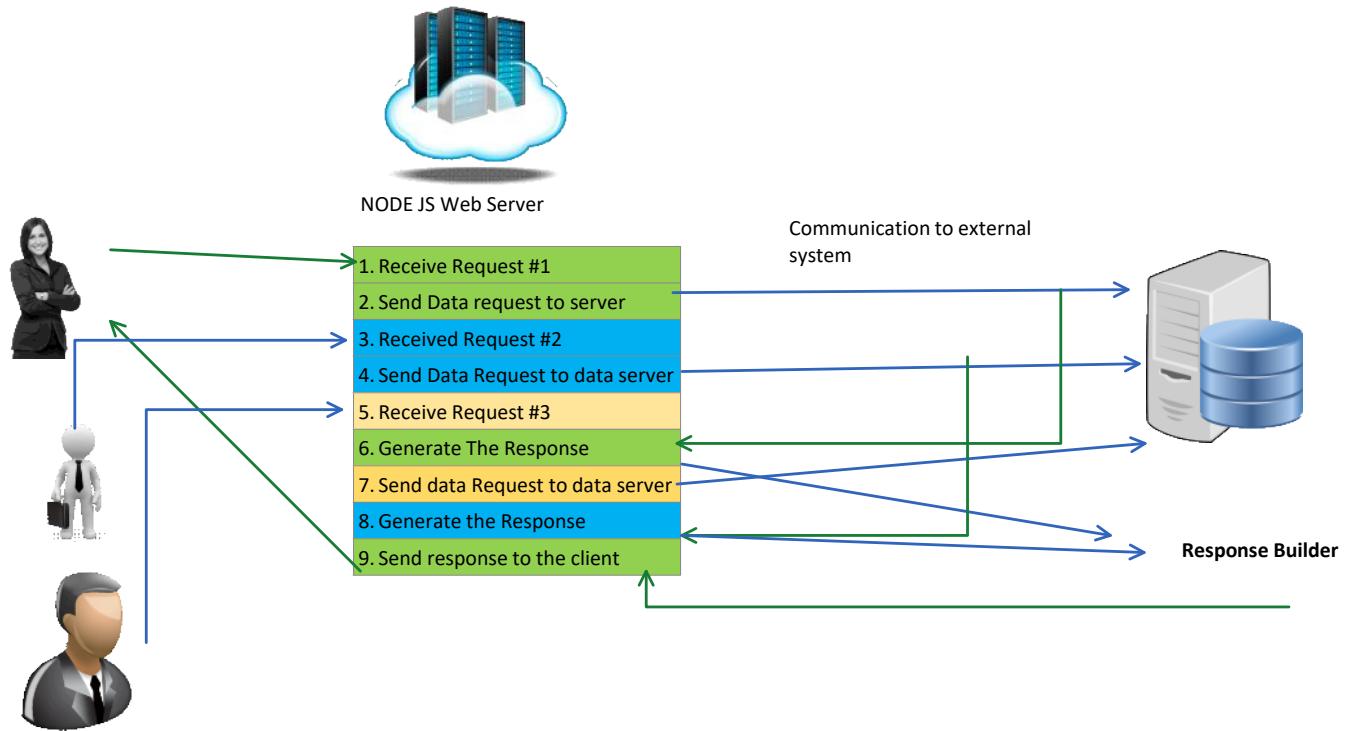
Friday, December 31, 2021 10:51 AM

One Important Difference

- NodeJS allows you to use 'require' anywhere in the code you like
- ES2015 import must be on the top of the page before any other code

NODE JS Asyncronous Pipeline

Friday, December 31, 2021 1:03 PM



Callback — Backbone of the system

- A long running task should not return the value using 'return' statement
- It should return the value by calling a 'callback' function
- Callback function should include what to do next.

Synrhroneous process

```
function handleRequest(request, response){  
  const param= request.getParam();  
  const data = getDataFromDataBase(param);  
  const responseData= createResponse(data);  
  response.send(responseData);  
}
```

Asynchronous Process

```
function handleRequest(request, response){  
  request.getParam( param=>{  
    //after getting param  
    getDataFromDatabase( param, data =>{  
      //after getting data from server  
      createResponse( data, responseData=>{  
        //after building the response  
        response.send(responseData);  
      });  
    });  
}
```

});

}

.

NodeJS is asynchronous

Friday, December 31, 2021 3:29 PM

- NodeJS expects us to write asynchronous API (functions)
- Any function that we write should be asynchronous
- A synchronous function should have the **Sync** suffixed to their names

NodeJS callback pattern

- NodeJS recommends a callback to have a particular signature
- Most NodeJS API will follow the rule
- A nodejs callback function should take two parameters
 1. Error
 - should return null if there is no error
 - Should return the error object in case we have an error
 2. Result
 - The actual result from the asynchronous activity
 - Error should be null to have the correct result
 - In case of error, result should be null/undefined

Example

- If we want to create our findPrimes using node pattern
 - it should return primes as the second parameter
 - We may return error in first parameter
 - Example: invalid range
 - If you pass min>=max

```
function findPrimes( min, max, cb){  
  setInterval( ()=>{  
    //your async code here  
    if( min>=max)  
      cb( new Error("Invalid range") );  
    ...  
    cb( null, result );  
  }, 10);  
}
```

- When you need to return the error
- When you need to return success
 - pass null as first parameter
 - Actual result as second

Calling the Service

IMPORTANT!

```
findPrimes( min, max, (error,result) => {  
  if(error){  
    console.log('error',error);  
  } else{
```

- It is a coding convention NOT compulsion
- Most NodeJS API follow this rule
- You should also follow this design for consistency

```
//handle the result
console.log('total primes', result.length);
}

});
```

NodeJS Predefined Modules

Friday, December 31, 2021 5:15 PM

- NodeJS comes with predefined modules for various tasks
- You can import them by regular **require** statement
- We don't use './' or relative path to import built-in modules
 - We use it directly

'fs' module

- Helps us interact with file system
 - Read file
 - Write file
 - Create directory
 - See directory content
- Most api has a
 - **async versions**
 - No name suffix
 - Will typically take a callback
 - Sync versions are rarely used
 - They have **sync** suffix

```
fs {
  appendFile: [Function: appendFile],
  appendFileSync: [Function: appendFileSync],
  access: [Function: access],
  accessSync: [Function: accessSync],
  chown: [Function: chown],
  chownSync: [Function: chownSync],
  chmod: [Function: chmod],
  chmodSync: [Function: chmodSync],
  close: [Function: close],
  closeSync: [Function: closeSync],
  copyFile: [Function: copyFile],
  copyFileSync: [Function: copyFileSync],
  createReadStream: [Function: createReadStream],
  createWriteStream: [Function: createWriteStream],
  exists: [Function: exists],
  existsSync: [Function: existsSync],
  fchown: [Function: fchown],
  fchownSync: [Function: fchownSync],
  fchmod: [Function: fchmod],
  fchmodSync: [Function: fchmodSync],
  fdatasync: [Function: fdatasync],
  fdatasyncSync: [Function: fdatasyncSync],
  fstat: [Function: fstat],
  fstatSync: [Function: fstatSync],
  fsync: [Function: fsync],
  fsyncSync: [Function: fsyncSync],
  ftruncate: [Function: ftruncate],
  ftruncateSync: [Function: ftruncateSync],
  futimes: [Function: futimes],
  futimesSync: [Function: futimesSync],
  lchown: [Function: lchown],
  lchownSync: [Function: lchownSync],
  lchmod: undefined,
  lchmodSync: undefined,
  link: [Function: link],
  linkSync: [Function: linkSync],
  lstat: [Function: lstat],
  lstatSync: [Function: lstatSync],
  mkdir: [Function: mkdir],
  mkdirSync: [Function: mkdirSync],
  mkdtemp: [Function: mkdtemp],
  mkdtempSync: [Function: mkdtempSync],
  open: [Function: open],
  openSync: [Function: openSync],
  opendir: [Function: opendir],
  opendirSync: [Function: opendirSync],
  readdir: [Function: readdir],
  readdirSync: [Function: readdirSync],
  read: [Function: read],
  readSync: [Function: readSync],
  readv: [Function: readv],
  readvSync: [Function: readvSync],
  readFile: [Function: readFile],
  readFileSync: [Function: readFileSync],
  readlink: [Function: readlink],
  readlinkSync: [Function: readlinkSync],
  realpath: [Function: realpath] { native: [Function] },
  realpathSync: [Function: realpathSync] { native: [Function] },
  rename: [Function: rename],
  renameSync: [Function: renameSync],
```

Synchronous File read

- recommended only when we are sure the file is going to be small (few hundred kbs) no large
- Example: configuration file
- Avoid for large file or if you are not sure if file will be large
- Example: reading a HD movie file that may run in several gbs.

```
const fs=require("fs"); //note: we don't use ./ or ../ prefix here

//console.log('fs',fs);

const buffer= fs.readFileSync('sample.txt'); //get a buffer of data

const data= buffer.toString(); //convert a buffer to plain text

console.log('data',data);
```

Asynchronous File Read

```
const fs=require("fs"); //note: we don't use ./ or ../ prefix here

//console.log('fs',fs);

const buffer= fs.readFile('sample.txt',(error,buffer)=>{
  if(error){
    console.log('error',error.message);
  } else{
    const data= buffer.toString(); //convert a buffer to plain text
    console.log('\n\n',data);
  }
});
```

//you reach here before readFile finishes its work
console.log('please wait while we read the file...');

Call back

Executes first

```
rmdir: [Function: rmdir],
rmdirSync: [Function: rmdirSync],
stat: [Function: stat],
statSync: [Function: statSync],
symlink: [Function: symlink],
symlinkSync: [Function: symlinkSync],
truncate: [Function: truncate],
truncateSync: [Function: truncateSync],
unwatchFile: [Function: unwatchFile],
unlink: [Function: unlink],
unlinkSync: [Function: unlinkSync],
utimes: [Function: utimes],
utimesSync: [Function: utimesSync],
watch: [Function: watch],
watchFile: [Function: watchFile],
writeFile: [Function: writeFile],
writeFileSync: [Function: writeFileSync],
write: [Function: write],
writeSync: [Function: writeSync],
writev: [Function: writev],
writevSync: [Function: writevSync],
Dir: [Function: Dir],
Dirent: [Function: Dirent],
Stats: [Function: Stats],
ReadStream: [Getter/Setter],
WriteStream: [Getter/Setter],
FileReadStream: [Getter/Setter],
FileWriteStream: [Getter/Setter],
_toUnixTimestamp: [Function: toUnixTimestamp],
F_OK: 0,
R_OK: 4,
W_OK: 2,
X_OK: 1,
constants: [Object: null prototype] {
  UV_FS_SYMLINK_DIR: 1,
  UV_FS_SYMLINK_JUNCTION: 2,
  O_RDONLY: 0,
  O_WRONLY: 1,
  O_RDWR: 2,
  UV_DIRENT_UNKNOWN: 0,
  UV_DIRENT_FILE: 1,
  UV_DIRENT_DIR: 2,
  UV_DIRENT_LINK: 3,
  UV_DIRENT_FIFO: 4,
  UV_DIRENT_SOCKET: 5,
  UV_DIRENT_CHAR: 6,
  UV_DIRENT_BLOCK: 7,
  S_IFMT: 61440,
  S_IFREG: 32768,
  S_IFDIR: 16384,
  S_IFCHR: 8192,
  S_IFLNK: 40960,
  O_CREAT: 256,
  O_EXCL: 1024,
  UV_FS_O_FILEMAP: 536870912,
  O_TRUNC: 512,
  O_APPEND: 8,
  F_OK: 0,
  R_OK: 4,
  W_OK: 2,
  X_OK: 1,
  UV_FS_COPYFILE_EXCL: 1,
  COPYFILE_EXCL: 1,
  UV_FS_COPYFILE_FICLONE: 2,
  COPYFILE_FICLONE: 2,
  UV_FS_COPYFILE_FICLONE_FORCE: 4,
  COPYFILE_FICLONE_FORCE: 4
},
promises: [Getter]
}
```

Fs module - Functions

Friday, December 31, 2021 5:15 PM

- NodeJS comes with predefined modules for various tasks
- You can import them by regular **require** statement
- We don't use './' or relative path to import built-in modules
 - We use it directly

'fs' module

- Helps us interact with file system
 - Read file
 - Write file
 - Create directory
 - See directory content
- Most api has a
 - **async versions**
 - No name suffix
 - Will typically take a callback
 - Sync versions are rarely used
 - They have **sync** suffix

```
fs {  
  appendFile: [Function: appendFile],  
  appendFileSync: [Function: appendFileSync],  
  access: [Function: access],  
  accessSync: [Function: accessSync],  
  chown: [Function: chown],  
  chownSync: [Function: chownSync],  
  chmod: [Function: chmod],  
  chmodSync: [Function: chmodSync],  
  close: [Function: close],  
  closeSync: [Function: closeSync],  
  copyFile: [Function: copyFile],  
  copyFileSync: [Function: copyFileSync],  
  createReadStream: [Function: createReadStream],  
  createWriteStream: [Function: createWriteStream],  
  exists: [Function: exists],  
  existsSync: [Function: existsSync],  
  fchown: [Function: fchown],  
  fchownSync: [Function: fchownSync],  
  fchmod: [Function: fchmod],  
  fchmodSync: [Function: fchmodSync],  
  fdatasync: [Function: fdatasync],  
  fdatasyncSync: [Function: fdatasyncSync],  
  fstat: [Function: fstat],  
  fstatSync: [Function: fstatSync],  
  fsync: [Function: fsync],  
  fsyncSync: [Function: fsyncSync],  
  ftruncate: [Function: ftruncate],  
  ftruncateSync: [Function: ftruncateSync],  
  futimes: [Function: futimes],  
  futimesSync: [Function: futimesSync],  
  lchown: [Function: lchown],  
  lchownSync: [Function: lchownSync],  
  lchmod: undefined,  
  lchmodSync: undefined,  
  link: [Function: link],  
  linkSync: [Function: linkSync],  
  lstat: [Function: lstat],  
  lstatSync: [Function: lstatSync],  
  mkdir: [Function: mkdir],  
  mkdirSync: [Function: mkdirSync],  
  mkdtemp: [Function: mkdtemp],  
  mkdtempSync: [Function: mkdtempSync],  
  open: [Function: open],  
  openSync: [Function: openSync],  
  opendir: [Function: opendir],  
  opendirSync: [Function: opendirSync],  
  readdir: [Function: readdir],  
  readdirSync: [Function: readdirSync],  
  read: [Function: read],  
  readSync: [Function: readSync],  
  readv: [Function: readv],  
  readvSync: [Function: readvSync],  
  readFile: [Function: readFile],  
  readFileSync: [Function: readFileSync],  
  readlink: [Function: readlink],  
  readlinkSync: [Function: readlinkSync],  
  realpath: [Function: realpath] { native: [Function] },  
  realpathSync: [Function: realpathSync] { native: [Function] },  
  rename: [Function: rename],  
  renameSync: [Function: renameSync],
```

```
rmdir: [Function: rmdir],
rmdirSync: [Function: rmdirSync],
stat: [Function: stat],
statSync: [Function: statSync],
symlink: [Function: symlink],
symlinkSync: [Function: symlinkSync],
truncate: [Function: truncate],
truncateSync: [Function: truncateSync],
unwatchFile: [Function: unwatchFile],
unlink: [Function: unlink],
unlinkSync: [Function: unlinkSync],
utimes: [Function: utimes],
utimesSync: [Function: utimesSync],
watch: [Function: watch],
watchFile: [Function: watchFile],
writeFile: [Function: writeFile],
writeFileSync: [Function: writeFileSync],
write: [Function: write],
writeSync: [Function: writeSync],
writev: [Function: writev],
writevSync: [Function: writevSync],
Dir: [Function: Dir],
Dirent: [Function: Dirent],
Stats: [Function: Stats],
ReadStream: [Getter/Setter],
WriteStream: [Getter/Setter],
FileReadStream: [Getter/Setter],
FileWriteStream: [Getter/Setter],
_toUnixTimestamp: [Function: toUnixTimestamp],
F_OK: 0,
R_OK: 4,
W_OK: 2,
X_OK: 1,
constants: [Object: null prototype] {
  UV_FS_SYMLINK_DIR: 1,
  UV_FS_SYMLINK_JUNCTION: 2,
  O_RDONLY: 0,
  O_WRONLY: 1,
  O_RDWR: 2,
  UV_DIRENT_UNKNOWN: 0,
  UV_DIRENT_FILE: 1,
  UV_DIRENT_DIR: 2,
  UV_DIRENT_LINK: 3,
  UV_DIRENT_FIFO: 4,
  UV_DIRENT_SOCKET: 5,
  UV_DIRENT_CHAR: 6,
  UV_DIRENT_BLOCK: 7,
  S_IFMT: 61440,
  S_IFREG: 32768,
  S_IFDIR: 16384,
  S_IFCHR: 8192,
  S_IFLNK: 40960,
  O_CREAT: 256,
  O_EXCL: 1024,
  UV_FS_O_FILEMAP: 536870912,
  O_TRUNC: 512,
  O_APPEND: 8,
  F_OK: 0,
  R_OK: 4,
  W_OK: 2,
  X_OK: 1,
  UV_FS_COPYFILE_EXCL: 1,
  COPYFILE_EXCL: 1,
  UV_FS_COPYFILE_FICLONE: 2,
  COPYFILE_FICLONE: 2,
  UV_FS_COPYFILE_FICLONE_FORCE: 4,
  COPYFILE_FICLONE_FORCE: 4
},
promises: [Getter]
}
```

Fs module - Examples

Friday, December 31, 2021 5:15 PM

Synchronous File read

- recommended only when we are sure the file is going to be small (few hundred kbs) no large
- Example: configuration file
- Avoid for large file or if you are not sure if file will be large
- Example: reading a HD movie file that may run in several gbs.

```
const fs=require("fs"); //note: we don't use ./ or ../ prefix here

//console.log('fs',fs);

const buffer= fs.readFileSync('sample.txt'); //get a buffer of data

const data= buffer.toString(); //convert a buffer to plain text

console.log('data',data);
```

With Error Handling

```
const fs=require("fs"); //note: we don't use ./ or ../ prefix here

//console.log('fs',fs);

try{

    const buffer= fs.readFileSync('samplexxx.txt'); //get a buffer of data

    const data= buffer.toString(); //convert a buffer to plain text

    console.log('data',data);

} catch(error){

    console.log('error:',error.message);

    console.log('normal shutdown');

}
```

Asynchronous File Read

```
const fs=require("fs"); //note: we don't use ./ or ../ prefix here

//console.log('fs',fs);

const buffer= fs.readFile('sample.txt',(error,buffer)=>{

    if(error){
        console.log('error',error.message);
    } else{
        const data= buffer.toString(); //convert a buffer to plain text
        console.log('\n\n',data,'');
    }
});

//you reach here before readFile finishes it's work
console.log('please wait while we read the file...');
```

Call back

Executes first

Use Case

- We should create a new folder "temp"
- There we should copy the current file in the new folder
- Display a list of files in the new folder
- Delete the new folder after 10 seconds.

mkdir,
copyFile
rmDir
readDir

```
fs.mkdir(tempDirName, (error,success)=>{

    if(error)
        console.log('error',error.message);
    else
        console.log('success: directory created');

});
```

- Fails if directory already exists

Assignment Day 5

Friday, December 31, 2021 6:03 PM

1. Compp

NodeJS events

Monday, January 3, 2022 5:20 PM

- NodeJS provides yet another mechanism for asynchronous programming
- This is based on a NodeJS library that will not work on web based application
- Events

What is an event

- It is occurrence of some particular situation that user may need to handle
- A web equivalent of event is — when a button is pressed.
- We may use events to represent

Use Case #1 — Prime Number

- A Prime Number is Found
- An Error occurred
- All Primes are found

- Events are some 'incident' happening in future
- We are not sure when they would occur (asynchronous)
- We are not sure which of them would occur and which ones shall not
- We can create mechanism to handle events as they occur

Use Case #2 — Cooking Example

- Order Placed
- Order Prepared
- Order Delivered
- Order Error

How Event Works

- An asynchronous code may return an **EventEmitter** object immediately
 - Just like we return a promise.
- Client should configure event handling by using **event.on()**
 - Just like we handle promise.then() and promise.catch()
- When asynchronous function need to return some answer it can **emit** and event
- The event will be handled in on.
- We can emit different type events
 - Each event will have user defined name
 - Each event can be emitted/handled multiple times

How event differs from promise

- Creating Async function
 - Promise
 - Sends notification using resolve/reject
 - EventEmitter
 - Sends notification using emit
- Consuming the asynchronous message
 - Promise
 - Client need to write then/catch
 - EventEmitter
 - Client need to write .on
- Sending Message
 - Promise
 - Can be resolved/rejected only once
 - It must carry all data at once
 - Event
 - Same event can occur multiple times
 - Each time we can send a small chunk of data rather than sending whole data

Advantage of Event over Promise/Regular callback

1 Event uses custom names

- We can handle event based on our program need
- Each event will have user defined name when you emit it

Eg.

```
event.emit('prime-found', { index:1, prime:2 } )
```

```
event.emit('food-prepared', {food: 'chicken soup'})
```

- It makes code readable

2. Same event can occur multiple times

- A promise can be rejected/resolved only once
- Promise must carry all data together.
- Event doesn't need to return all items together
 - Same event can be emitted multiple times
- We can carry small data frequently rather than wait for large data

3. Two way communication

- Promise/callback are one way flow
- Async function informs your when the job is complete
- Promise is resolved by the service (async function)
 - Client get data when it is resolved
- Client can't send new requests to service
 - A customer can't add new items to their order
 - What if he needs another roti?
- In events service-client share same event object
 - Both can emit
 - Both can receive.
- It allows you to design more features like
 - Notifying the progress bar
 - Allowing user to cancel the running operation
- You don't need to handle every emitted event
 - You may handle what you need

Problem

• Large Data

- Some times we may have really large volume of data
 - Say readFile returns all bytes of the file
 - What if my file is 2gb big?
 - ◆ We will store 2gb of file in memory
 - ◆ Reading of 2gb of data will take long time.
 - ◆ Do I need all those bytes together

• Do we need all those data

- Consider Amazon product search page
 - When you search for an item, how many pages of result do you get?
 - 50-100 pages
 - Do you check all pages?
 - 3-4
 - So is it wise to process 100 pages when we might not check more than 3-4 page?
 - Amazon actually process on 1 page at a time
 - It processes next page on next button click.

• Think Prime

- There are 40K+ primes between 2-500000
- What will I do with all these primes
 - Print them
 - Save them
 - Sum them
 - Average them
- Do you need all the primes together?
- We can't begin to process the prime till we get all of them

• Cooking Example

- In both promise and callback cook must return all items of a customer together
 - Promise is resolved only once
 - Callback is also generally called only once with entire data
- Won't it be better if we can serve food as they are prepared?

Promise vs Events

Monday, January 3, 2022 5:44 PM

#	Promise	Event
ES Feature	ES2015	NO (NodeJS Feature)
Sending the information async	resolve/reject	.emit()
Receiving async infomration	.then()/ .catch() / await/cath	on()
Names	Predefined	User defined names as first parameter to emit and on
Occurance	One time resolve/reject only	Multiple time
Direction	From service to client Service resolves/rejects client receives in then/catch	Bidirection communication Both client and service can use .emit/.on
Support for progress/cancellation	None	YES

Assignment 6.4+5

Monday, January 3, 2022 6:11 PM

6.4 Rewrite the cook example with Promise

- Use async-await in cook and waiter code

6.5 Rewrite cook example with events

- User can request for more food items
- User can request for "BILL" to indicate he will order no more food
- User should inform when he is "DONE" eating
- User can order for food in menu
- Should have error in case ordered item is not in menu
- User should get a "BILL" once he has done eating.

Nodejs fs event

Tuesday, January 4, 2022 10:08 AM

- Reading a very large file using readFile may be problematic
 - We may have to wait for a very long time.
 - We may need a large memory to hold the file data
 - What if file is several gb in size
 - HD movie file
 - Do I need entire movie data in memory to play the movie?
 - We need just few MB to sufficient to display move work say 10-20 seconds
 - And read more data as we display 10-20 seconds
 - We can get data stream.

Event Based File Reading

```
function readBuffer(buffer){ //when file reads some data. called multiple times...  
}  
  
function handleError(error){ ...  
};  
  
function processData(){ ...  
}  
  
events  
.on("data", readBuffer )  
.on("end",processData)  
.on("error", handleError );  
  
  
const fs=require("fs"); //note: we don't use ./ or ../ prefix here  
  
//console.log('fs',fs);  
  
const events=fs.createReadStream('sample-large.txt');  
  
let fileContent='';  
  
events  
.on("data", buffer=>{ //when file reads some data. called multiple times  
    const data=buffer.toString();  
    process.stdout.write(`[${Math.floor(data.length/1024)}K] `);  
    fileContent+=data;  
})  
.on("end",_=>{ //when all data has been read. called only once.  
    process.stdout.write('file read\n-----\n\n');  
    process.stdout.write('total size is '+fileContent.length);  
})  
.on("error", error=> console.error("ERROR READING FILE:",error.message)); //called only once  
  
//you reach here before readFile finishes it's work  
console.log('please wait while we read the file...');
```

- We have defined callback functions that will be called asynchronously when a particular condition is met.
- We are not calling it. Just defining it.
 - Since we don't call the function we don't pass any data
 - We expect that whoever will call this function will pass the data.
- It will be called in response to some activity or situation.

We tell the event

- When 'data' event is emitted call my function readBuffer
 - When readBuffer is called the event also passes the buffer argument.
- We are just defining a function here
- We are not calling it immediately
- We don't know when exactly it will be called
- We are not passing buffer here
- We are expecting whoever calls this function will pass the argument for buffer

IMPORTANT!

- If an emitted event doesn't have a corresponding 'on' (handler) that event will be ignored.
- It is not an error not to handle an emitted event
- There is no default handler for emitted event.

Promisify

Tuesday, January 4, 2022 10:35 AM

- 'fs' object was created before introduction of ES2015 Promises.
 - Most built-in libraries in nodeJS is mostly based on
 - NodeJS style callback
 - NodeJS events
 - Event third party libraries are initially based on these two concepts
-
- In modern NodeJS (and third party library) we started introducing Promise concept
 - Now most of the modern library has 'Promise' introduced for async programming
 - Some may follow only call back.

How to convert a NodeJS callback into a Promise?

How a NodeJS callback looks like?

```
function someFunction(param1, param2, callback) {  
  if(some_error)  
    callback( new Error(...));  
  
  callback(null, result);  
}  
  
function someOtherFunction(param1,param2, param3, callback){  
  if(some_error)  
    callback( new Error(...));  
  
  callback(null, result);  
}
```

Writing a generic Promise Wrapper for Normal callback

```
function promisify ( target ) {  
  return (...params) =>{  
    return new Promise( (resolve,reject) =>{  
      target(...param, (error,data) =>{  
        if(error)  
          reject(error);  
        else  
          resolve(data);  
      });  
    });  
  }  
}
```

1. Wrap the current function into a new function using closure
2. The New Function returns a **Promise**
3. Whenever you call the new function it internally calls the actual function and passes **it's own callback**
4. This callback resolves/rejects promise based on error or data
5. Now we can call our Promise using `.then/.catch`

```
readFilePromise(fileName)
  .then(data=>...)
  .catch(error=>...)
```

6. We can also use try-catch-await

```
const client= async ()=>{
  try{
    const data=await readFilePromise(fileName);
    ...
  }catch(error){
    }
}
```

NodeJS fs Promise Library

Tuesday, January 4, 2022 11:19 AM

- NodeJS also provides a promise based version of fs

```
const fs=require("fs").promises;  
  
console.log(fs);
```

- It contains asynchronous version of fs library

```
access: [AsyncFunction: access],  
copyFile: [AsyncFunction: copyFile],  
open: [AsyncFunction: open],  
opendir: [Function: opendir],  
rename: [AsyncFunction: rename],  
truncate: [AsyncFunction: truncate],  
rmdir: [AsyncFunction: rmdir],  
mkdir: [AsyncFunction: mkdir],  
readdir: [AsyncFunction: readdir],  
readlink: [AsyncFunction: readlink],  
symlink: [AsyncFunction: symlink],  
lstat: [AsyncFunction: lstat],  
stat: [AsyncFunction: stat],  
link: [AsyncFunction: link],  
unlink: [AsyncFunction: unlink],  
chmod: [AsyncFunction: chmod],  
lchmod: [AsyncFunction: lchmod],  
lchown: [AsyncFunction: lchown],  
chown: [AsyncFunction: chown],  
utimes: [AsyncFunction: utimes],  
realpath: [AsyncFunction: realpath],  
mkdtemp: [AsyncFunction: mkdtemp],  
writeFile: [AsyncFunction: writeFile],  
appendFile: [AsyncFunction: appendFile],  
readFile: [AsyncFunction: readFile]
```

How Promise simplifies nested async job

Tuesday, January 4, 2022 11:47 AM

- Sometimes one async job depends on the completion of other async job
- Normal callback can created a nested tree that may be difficult to understand
- Async await makes the call simple to understand
- It will look as simple as normal synchronous job

```
fs.mkdir(tempDirName, (error) => {
  // success or failure, now we have our directory
  fs.copyFile(sourceFileSpecs, tempDirName + "/" + sourceFileSpecs, fs.constants.COPYFILE_EXCL, (error) => {
    if (error) {
      console.log('file copy error:', error.message);
    }

    fs.readdir(tempDirName, (error, files) => {
      if (error) {
        console.error('error reading directory:', error.message);
      } else {
        for(const file of files){
          fs.readFile(`./${tempDirName}/${file}`, (error,data)=>{
            console.log(data.toString());
          });
        }
      }
      setTimeout(()=>{
        fs.rmdir(tempDirName,{ recursive: true, force: true }, error=>{
          if(error){
            console.log('error removing folder:',error.message);
          } else{
            console.log('all done');
          }
        })
      },10000)
    });
  });
});
```

```
const task= async ()=>{
try{
  await fs.mkdir(tempDirName);

  await fs.copyFile(sourceFileSpecs,`${tempDirName}/${sourceFileSpecs}`);

  const files= await fs.readdir(tempDirName);

  for(const file of files){
    console.log('checking out file');
    const fileName= `${tempDirName}/${file}`;
    var data=await fs.readFile(fileName);
    console.log(fileName,data.toString().length);
  }

  console.log('will remove the temp folder after 10 seconds');
  await delay(10000);
  await fs.rmdir(tempDirName,{ recursive: true, force: true });

} catch(error){
  console.log('error:',error.message);
}
```

Streams

Tuesday, January 4, 2022 12:18 PM

What is a stream?

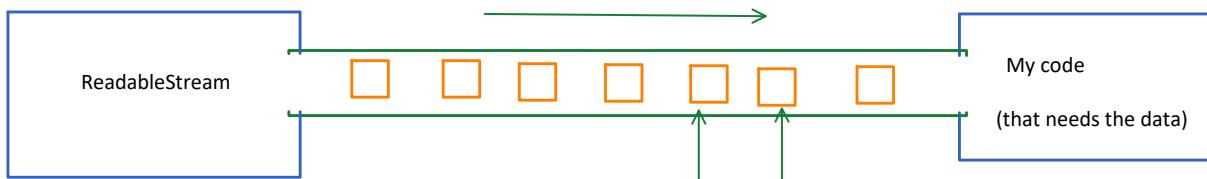
- A continuous flow (of data)

Node JS Streams

- Streams are event based objects
- It is an object which defines a standard set of
 - Methods
 - open
 - close
 - Events
 - error

1. Readable Stream

- An input source data
- Our program reads data from some source like
 - File
 - Network
 - Some programming logic
- We can read the stream as a continuous flow rather than as one single large block



Methods

- `read()`
 - We don't know when next data will be available
- `close()`
 - Close the stream. We don't want any further data.

Events

- "data"
 - Everytime a chunk (buffer) of data is read
- "error"
 - In case you have error reading
 - Example
 - Invalid file name
 - Internet disconnected while reading a large move data
- "end"
 - When we reach the end of stream and we have no more data to read

```
const rs=createReadStream(file)

rs.on("data", buffer=>{})

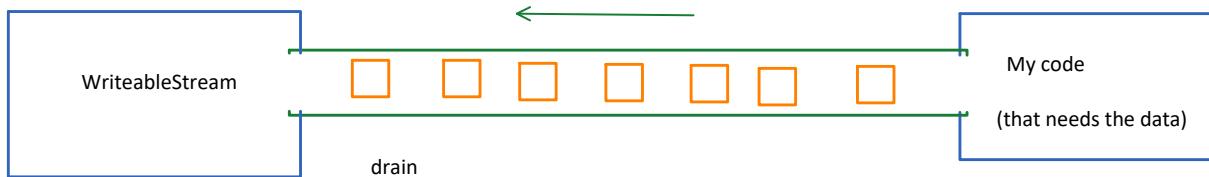
rs.on("error",error=>{})

rs.on("end", _=>{});

rs.close(); //will trigger 'end'
```

Writable Stream

- Allows us to write the data to a stream
 - Write to a file
 - Write to network



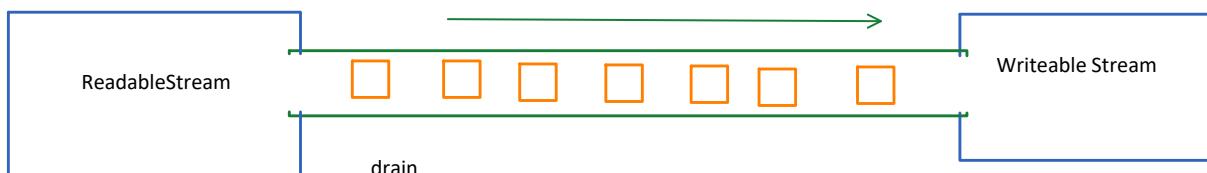
Methods

- `write`
 - Writes to the stream
 - **Important**
 - Just because we called `write` doesn't mean it immediately written
 - It may be a slow device and may take long time to write.
 - It raises a '`drain`' event to inform that all data sent for writing has been consumed ('drained')
 - We are ready for more data
- `Close`

Events

- `Drain`
 - When previous data is written
 - Signal to send more data
- `Error`
- `close`

Copying data from readable to writeable Stream

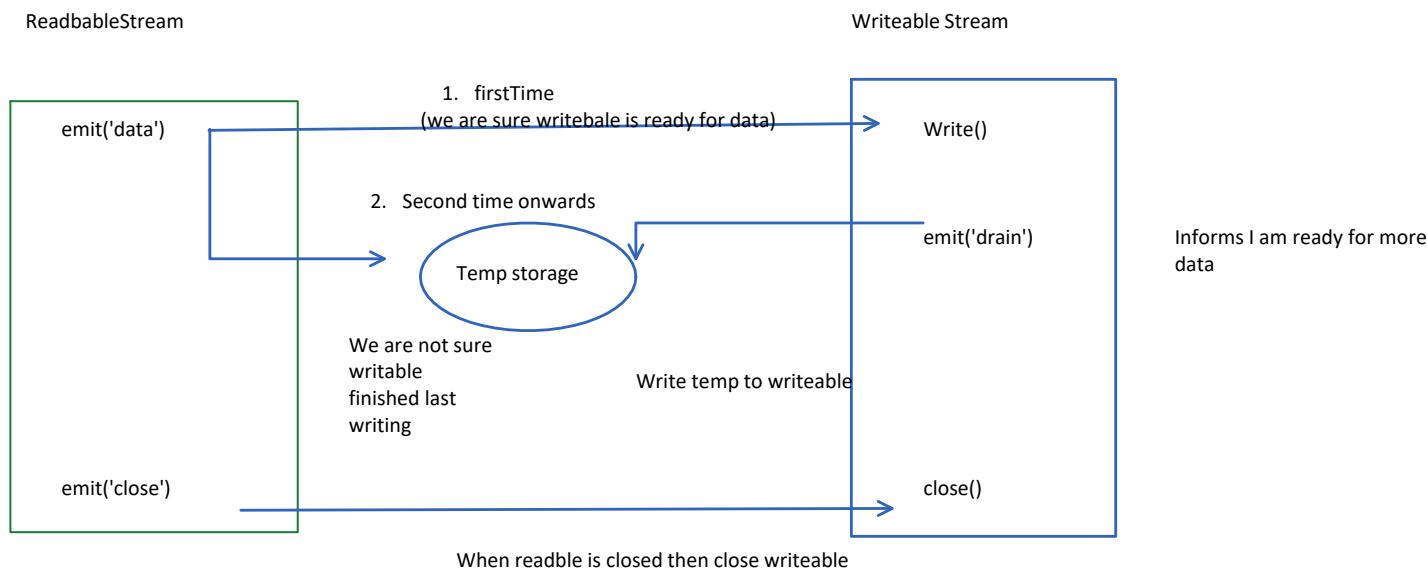


- Here our program doesn't need to be involved.
- Its job is to transfer from a readable stream to a writeable stream

- Use case
 - File copy
 - Copy a local file to network
 - Display the content of a file on the screen

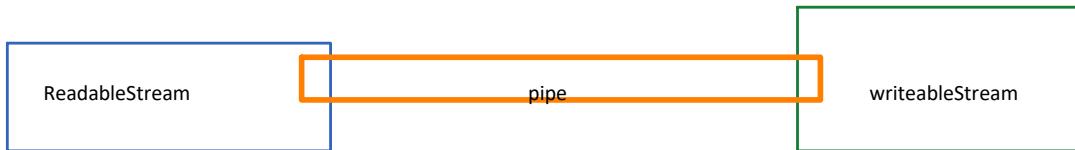
File copy using readable writeable stream

Tuesday, January 4, 2022 12:41 PM



Pipe

- Instead of manually writing the code from copying from one stream to another we can pipe a readable stream to a writeable stream
- **Pipe will take care of the transfer.**



```
const sourceFileName='sample-large.txt';
const targetFileName='sample-large-pipe.txt';

const readable= fs.createReadStream(sourceFileName);
const writeable= fs.createWriteStream(targetFileName);

readable.pipe(writeable);

fs
  .createReadStream('sample.txt')
  .pipe(process.stdout); //stdout is a writeable
```

Console.log vs process.stdout

- `Process.stdout` is a writeable stream
- We can directly pipe to it
- `Console.log` is a normal function and not a writeable

Event Stream Pipe

Tuesday, January 4, 2022 1:16 PM

Event

- A general asynchronous programming concept
- Based on user defined event name

Stream

- An object (interface) that defines a fixed set of
 - Event
 - Methods
- Here event names are predefined

Pipe

- Based one Readable and Writeable stream
- Allows automatic data transfer from readable to writeable
- Internally uses
 - Stream methods and events

NPM

Tuesday, January 4, 2022 2:08 PM

- NPM → Node Package Manager
- Consider is the "PlayStore" or "AppStore" for web development
 - You can publish your component on NPM.
 - You can download the existing components from NPM
- NPM contains
 - Javascript libraries (JS files)
 - CSS libraries
 - Executable Applications
 - To perform various tasks
- It can be considered as one stop market place for all web application needs

Important

- Many developer who use NodeJS use it just because they want to use NPM and not because they want to develop NodeJS application!

Two Parts of NPM

1. Repository

- NPM refers online repository where all various packages are available for download
- <https://npmjs.com>
- More than 6.5 Lacks package as on 2019.
- We have packages for almost every need.
- The repository is an open community where we can publish our own libraries and codes.

2. NPM command line

- NPM is a command line utility that helps you interact with NPM repository and perform some task on your own computer.
- It is like "playstore" or "appstore" app on your mobile device
- NPM command line itself is available on NPM

NPM as a project manager

- NPM also acts as a project manager for web application
- It creates a registry "package.json" that includes
 - Identification of your application
 - All your dependency requirement
 - Some common commands that you run frequently.

1. Check if npm is present

```
c:\> npm --version
```

2. NPM based project structure

1. We create a folder to represent our entire project
2. We create a "package.json" in the folder to represent our "projects property"

```
c:\workspace\myproject> npm init
```

- It will create a package.json file

```
D:\MyWorks\Corporate\202112-mobileum-pern\npm-demo>npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help init` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (npm-demo) my-npm-app →
version: (1.0.0) →
description: A simple npm project →
entry point: (index.js) →
test command:
git repository:
keywords:
author: Vivek Dutta Mishra
license: (ISC)
About to write to D:\MyWorks\Corporate\202112-mobileum-pern\npm-demo\package.json:
{
  "name": "my-npm-app"
}
```

- You can enter the values
- If you hit enter the default value given in parenthesis is taken
- The data is written in package.json file

```

author: Vivek Dutta Mishra
license: (ISC)
About to write to D:\MyWorks\Corporate\202112-mobileum-pern\npm-demo\package.json:

{
  "name": "my-npm-app",
  "version": "1.0.0",
  "description": "A simple npm project",
  "main": "index.js",
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
  },
  "author": "Vivek Dutta Mishra",
  "license": "ISC"
}

Is this OK? (yes)

```

- The data is written in package.json file
 - It can be modified later.

```

{
  "name": "my-npm-app",
  "version": "1.0.0",
  "description": "A simple npm project",
  "main": "index.js",
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
  },
  "author": "Vivek Dutta Mishra",
  "license": "ISC"
}

```

There is a shortcut to create package.json with all defaults

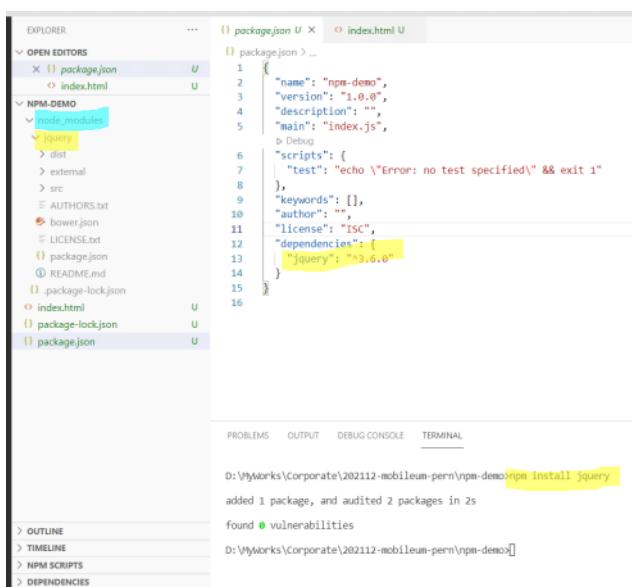
npm init --y

```
D:\MyWorks\Corporate\202112-mobileum-pern\npm-demo>npm init --y
Wrote to D:\MyWorks\Corporate\202112-mobileum-pern\npm-demo\package.json:
```

```
{
  "name": "npm-demo",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

We can install packages from the repository

npm install package-name



- Downloads jquery package into "node_modules" folder inside your project
- Makes an entry for this package in "dependencies" section of your "package.json"
 - Notice we also have version number entry here
 - By default installs the latest version
 - You may specify other version if you need by using

```
npm install jquery@2.3.5
```

How do I know the package name?

- Package name is unique across npm
- It is available on first-come-first-given basis
 - Just like an email account on google.
- For the right package you may need you can search
 - <https://npms.com>
 - Website of the package developer
 - Google!

We can use "i" as shortcut for "install" on npm command line

```

11  "license": "ISC",
12  "dependencies": {
13    "bootstrap": "^4.6.1", ←
14    "jquery": "^3.6.0" ←
15  }

```

```

11 "license": "ISC",
12 "dependencies": {
13   "bootstrap": "4.6.1", ←
14   "jquery": "^3.6.0" ←
15 }
16
17

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
:\\Myworks\\Corporate\\202112-mobileum-perm\\npm-demo>npm i bootstrap@4.6.1
3m [WARN] deprecated popper.js@1.16.1: You can find the new Popper v2 at @popperjs/core, this package is dedicated to the legacy v1
```

We can install multiple packages using a single command line

```
npm i bootstrap@4.6.1 jquery font-awesome
```

Reinstall all packages

- Remember package.json includes all the package we need.
- We can run a simple command that looks up package.json and install them
- Call install without giving any package name.
 - It will reinstall all previous package by looking up in package.json

```
npm install
```

```
npm i
```

Running scripts from package.json

- We may want to run some common operation
- We can run them by defining the scripts section in package.json

Package.json

```

"scripts": {
  "greet": "node app.js",
  "hello": "echo Hello World"
},
"main": "index.js"

```

We can run these scripts using command

```
npm run <script-name>
npm run greet
npm run hello
```

Two special script names

- Npm understand two special script names
 - start
 - Test
- You can run them as

```
npm run start
npm run test
```

Or

```
npm start
npm test
```

Note: You can't run

```
npm greet
```

Installing Executable Utils

Tuesday, January 4, 2022 3:11 PM

- NPM also includes a list of executable utility
- These utility includes
 - Transpilers like
 - Typescript
 - Babel
 - Test Runners like
 - Karma
 - Jest
 - Task manager like
 - Gulp
 - Grunt
 - Utiltiy like
 - Nodemon

Nodemon (monitored node)

- Nodemon is a command like "node"
 - It can run a script
- Unlike "node" nodemon doesn't stop after running script
- It keeps watching for a change in script
- If the script changes it is immediate re-run

Installing executable packages

- We generally do not install executable package as a part of our project
- We install them globally so that it is available every where
- Npm is a global package
- To install global package we use "--global" switch
- Most of these packages are not required to run the application
- They are needed mostly for development

```
npm install --global nodemon
```

- Global pakcages are NOT installed in node_modules of my project
- It is installed in the global node directory

C:\Users\<userusername>\AppData\Roaming\npm\nodemon

Development Libraries

Tuesday, January 4, 2022 3:19 PM

- Sometimes we have a few libraries that are needed during development only
- They are not needed at runtime
- This includes
 - Testing library
 - Transpilers like typescript
 - Bundler like webpack
- During deployment you don't need them

We install them using --save-dev (-d) switch

- They are installed in my project folder 'package.json' like regular package
- They are included in "devDependencies" and not in "dependencies" of package.json

```
{ package.json U X index.html U app.js U utils.js U
  package.json > {} devDependencies
  6   "scripts": {
  7     "start": "node app.js",
  8     "dev": "nodemon app.js",
  9     "hello": "echo Hello World"
 10   },
 11   "keywords": [],
 12   "author": "",
 13   "license": "ISC",
 14   "dependencies": {
 15     "bootstrap": "^4.6.1",
 16     "font-awesome": "^4.7.0",
 17     "jquery": "^3.6.0"
 18   },
 19   "devDependencies": { ←
 20     "jest": "^27.4.5"
 21   }
 22 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
D:\MyWorks\Corporate\202112-mobileum-pern\npm-demo>npm install jest --save-dev
added 325 packages, and audited 330 packages in 40s
27 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities
D:\MyWorks\Corporate\202112-mobileum-pern\npm-demo>
```

- Generally when we deploy our project on production then this library will not be used.
- It will be used only during development.

Unit Testing

Tuesday, January 4, 2022 2:08 PM

- Unit testing ensures that our code is working as per our expectation.
- It is one of the important aspect for modern application design.

Are we not already testing our code?

- We have written several tests and demos to test our application
- We have console.log our application details on screen
- When we can see our code output properly using simple node application and console.log why do we need a new concept?

Problems with the traditional console.log approach

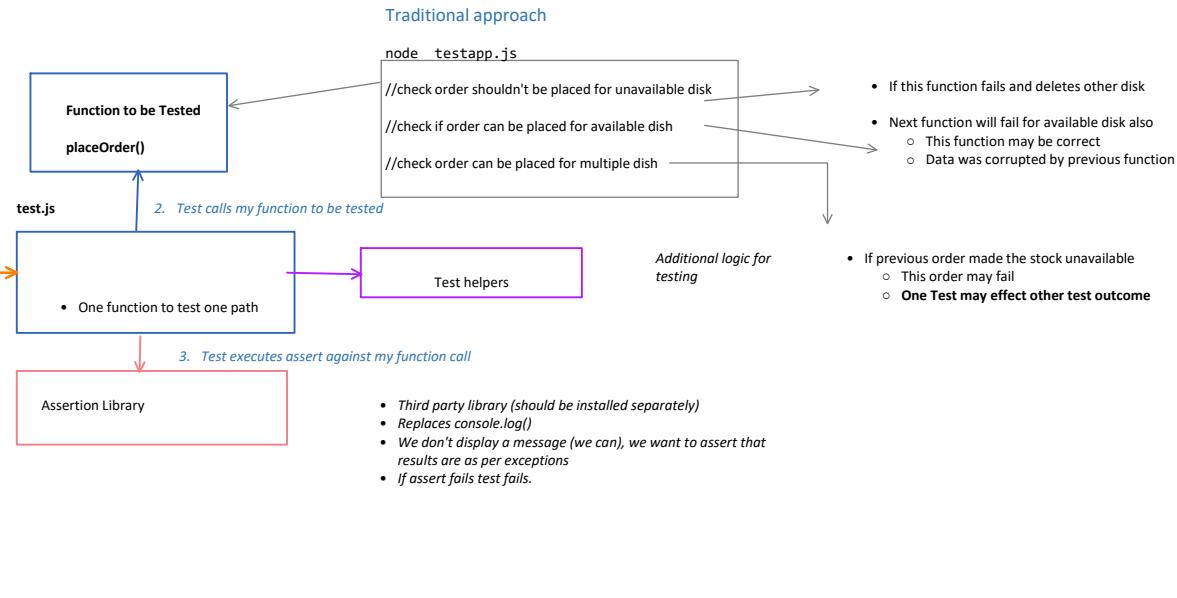
1. When we get an output, we see a message on screen.
 - We are not sure if this is what the message should be

```
isPrime(29) -> false
    - How do we know this output means code is working properly or not?
    - Should we keep calculating the answer for each case?
```
2. How do we know we have tested all aspects of a code?
 - What if some part remains untested?
3. If an exception is thrown does that mean, code is working properly or not?
 - A code is expected to throw exception if you supply invalid input.
 - So we supply and invalid input and code throws an exception that means code is working as expected.
 - Exception is codes way to tell things went wrong.
 - We must ensure that exceptions are thrown when they should be thrown.
4. Are we sure our calls are not dependent on each other

Unit Testing Framework

Tuesday, January 4, 2022 4:14 PM

- Replaces direct use of "node" node test.js



This output can be presented using

- Simple `console.log`
- As an HTML page
- On Some GUI tools

Testing Needs

1. Test Runner
 - Test Reporter
 2. Assert Libraries
 3. Additional Helper Libraries
- These libraries are generally third party libraries that we need to install
- All these functionality may come from different libraries or may come from same library

Purpose	Test Runner	Test Reporter	Assertion Library	Mocking Library
	karma	Karma r	Jasmine	
	mocha	mocha	Chai	
	Jest	jest	Jest	Jest
			shouldjs	

- We can use different combination of libraries for testing
 - Karma+Jasmine
 - Mocha+chai
 - Jest
 - Karma+chai
 - Jest+Jasmine

Jest

- Provides a complete suite
 - Runner
 - Reporter
 - Assertion
- From facebook
- Popular with React developers
- Can be used for testing both front-end and backend code

Test Interference

Friday, January 14, 2022 12:55 PM

```
index.js          AppHeader.js      # App.css      TicToe.js      TicToe.test.js      Cell.js
src > services > _tests_ > TicToe.test.js > describe('TicToe check tests') callback
Debug | Run | Debug
10  it('should identify first row winner', ()=>{
11    cells[0]=cells[1]=cells[2]='X';
12    const result=checkGame(cells);
13    expect(result.over).toBe(true);
14    expect(result.winner).toBe("X");
15    expect(result.winningSequence).toEqual([0,1,2]);
16  });
17
18 });
19
20 });
21
22 });

Run | Debug
describe('Test for running game', ()=>{
  Debug | Run | Debug
  it('should identify first row winner', ()=>{
    cells[0]='X';
    cells[1]='O';
    const result=checkGame(cells);
    // expect(result.over).toBe(false);
    expect(result.winner).toBe(null);
    expect(result.winningSequence).toBe(null);
    expect(result.movesLeft).toBe(7);
  });
});

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
expect(received).toBe(expected) // Object.is equality
Expected: 7
Received: 6
```

- This test sets only 2 out of 9 values.
- Ideally the moves left should be 7.
- But previous test set three values
 - Because of previous test moves left is 6
 - False Negative due to sharing same data

Solution

- Use `beforeEach`

Running the test

Wednesday, January 5, 2022 10:12 AM

- You may use any of the test runner
 - Karma
 - Mocha
 - Jest
 - ...
- There may be a little difference in different test
 - Test setup
 - Test configuration
 - Test code itself
- It is a good idea to provide a standard way to run the test irrespective of test runner

```
$ npm run test
```

```
$ npm test
```

- We should configure "npm run test" in package.json

```
...  
▷ Debug  
"scripts": {  
  "test": "jest --watch"  
},
```

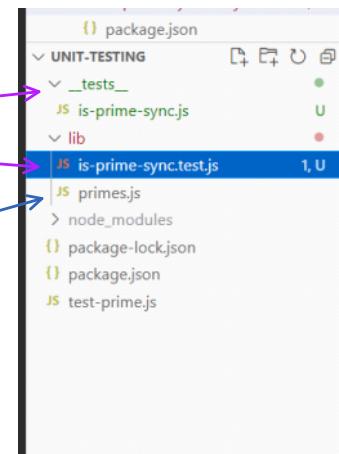
- We are running test using "jest"
- `--watch`
 - Tells jest to keep running test as soon as the any .js file changes
 - Either
 - .test file
 - Any other .js file
- Jest automatically searches files with following patter
 - `__test__/*.js`
 - `*.test.js`

Organizing the test

Wednesday, January 5, 2022 10:18 AM

What files does JEST uses as test file

1. Any file with
 - a. *.test.js
 - b. *.test.ts extension (typescript)
 - c. *.specs.js
 - d. *.specs.ts
2. Any .js or .ts file present in __tests__ folder



Organizing the Test

Approach #1

- Few developer prefer to keep their test file side-by-side to code that is being tested

Advantage

- This allows them to see these files together
- Importing is easier as they are in the same folder

```
const {isPrimeSync} = require('./prime-utils');
```

Disadvantage

- Your production code (actual functions) and test code are mixed up in same folder
- It becomes difficult to separate them during deployment

Approach #2

- Keep your tests separate in a test folder

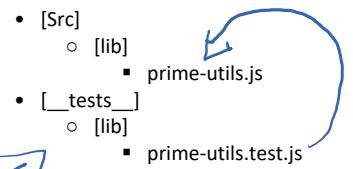
Advantage

- Code is better organized
- Test and production codes are independent
- Developer know where to look for test files

Disadvantage

- You may have to import deep nested imports

```
const {isPrimeSync} = require('../src/lib/prime-utils');
```



Test Organization

```

12
13  Run | Debug
14
15  test('should return true for isPrime(3)', ()=>{
16    expect(isPrimeSync(3)).toBe(true);
17  });
18
19  Debug | Run | Debug
20  it('should return false for isPrime(0)', ()=>{
21    expect(isPrimeSync(0)).toBe(false);
22  });

```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL

```

Expected: false
Received: true

18 |
19 | it('should return false for isPrime(0)', ()=>{
20 |   expect(isPrimeSync(0)).toBe(false);
21 | });

at Object.<anonymous> (_tests__/_is-prime-sync.test.js:20:28)

Test Suites: 1 failed, 1 total
Tests:       1 failed, 2 passed, 3 total
Snapshots:   0 total
Time:        1.614 s, estimated 2 s
Ran all test suites.

Watch Usage: Press w to show more. []

```

- A test code can be wrapped in either
 - `test()`
 - `it()`
- It takes two parameter
 - A descriptive text defining the purpose of test
 - A function that actually runs the test

Why testing function is called "it()"?

- "it" forms a nicer sentence with description that generally starts with should
- You may use something like
 - `test("that isPrime(3) returns true")`
 - `it('should return false for isPrime(30)');`

Test Suite

- A Test suite is a collection of Tests
- Generally a single .js file represents a test suite
- Different .js file is different suits
- We can also explicitly define suites by wrapping our tests in describe block.

Skipping a test

- Some times we want to skip a test without actually deleting it
- We may do this because
 1. We are not ready to the test yet
 2. We want to make sure that we will run this test at somepoint
 3. We want to keep showing it as 'skipped' test
- We may use `xit()` to skip a single test or `xdescribe()` to skip entire group of test

```
43
44  Run | Debug
45  describe('isPrimeSync sad path tests', () => {
46    xit('should throw error if no argument is supplied', ()=>{
47      const result=isPrimeSync();
48      //we will never reach here as it will throw an error
49      //expect(...)  

50    });
51  });
52
53
54
55
56
57
58
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
isPrimeSync tests
  isPrimeSync true value tests
    ✓ should return true for isPrime(13) (3 ms)
    ✓ should return true for isPrime(2) (1 ms)
  isPrimeSync false value tests
    ✓ should return false for isPrime(20) (1 ms)
    ✓ should return false for isPrime(0) (1 ms)
    ✓ should return false for isPrime(-5)
  isPrimeSync sad path tests
    o skipped should throw error if no argument is supplied
```

```
Test Suites: 1 passed, 1 total
Tests:       1 skipped, 5 passed, 6 total
Snapshots:   0 total
Time:        4.096 s
Ran all test suites.
```

Assertions

Wednesday, January 5, 2022 10:27 AM

- Assertions are the heart of any test
- A Test follows a simple rule known as AAA

A-A-A

- A test follows the rule to build the test
- It includes 3 A

1. A → Arrange

- We need to prepare for test
- Preparation may include
 - Creating the object whose functions are being tested
 - Adding some data that we may use for the test purpose
- This step is common between
 - Testing and actual execution
 - Testing using unit testing and traditional app based test

```
//Arrange
const manager=new BookManager();
manager.addBooks( new Book(...), new Book(...), new Book(...));
```

2. A → Act

- Run the function whose result you want to test
- This step is something you would run in production code also
 - Here we want to ensure it will not fail in production
- Say we want to test if my function getBookByIsbn is working or not.

```
//Act
```

```
const book1 = manager.getBookByIsbn( validIsbn)
const book2 = manager.getBookByIsbn(invalidIsbn);
```

3. A → Assert

- Assert is informing the test runner what is my expectation from running the code
 - Should be get the book?
 - Should be get null?
 - Should be get undefined?
 - Should be get some exception thrown?
- We do this assertions using third party assert libraries
 - Some basic asserts is present in NodeJS code library also
 - Some assert comes with test runner
 - Jest has rich set of asserts available
 - We may also install other assert libraries to make my design even better.
- We will use jest asserts here

```
//Assert
expect(book1).notBeNull();
expect(book1.title).toBe("The Accursed God");
expect(book2).toBeUndefined();
```

IMPORTANT!

- Above code is just to give hint of A-A-A process
- You shouldn't test different paths or features in a single test
- We should have a separate test for
 - Valid isbn
 - Invalid isbn
- We should NEVER merge different code path test in a single test
- Isolation is the key for unit testing

NodeJS Assert

- Nodejs comes with built-in assert library
- We can use these assert with any test runner
- We have many interesting assert options available in third party libraries

For details

<https://nodejs.org/api/assert.html>

```
tests_ > js is-prime-sync.test.js > ...
1
2 const {isPrimeSync}=require( '../lib/primes' );
3 const assert = require('assert');
4
5
6
7 Debug | Run | Debug
test('should return true for isPrime(3)', ()=>{
```

```
_tests_ > js is-prime-sync.test.js >_
1
2  const {isPrimeSync}=require('../lib/primes');
3  const Assert = require('assert');
4
5
6
7  test('should return true for isPrime(3)', ()=>{
8    const result=isPrimeSync(20);
9    Assert.deepEqual(result,true);
10 });

```

PROBLEMS DEBUG CONSOLE TERMINAL

Expected value to deeply equal to:
true
Received:
false

```
7 | test('should return true for isPrime(3)', ()=>{
8 |   const result=isPrimeSync(20);
> 9 |   Assert.deepEqual(result,true);
10 | });

```

Why test failed.

What does Assert/Expect do Internally

Wednesday, January 5, 2022 2:45 PM

- An assert or expect checks for given condition
- If expectations are met, it does nothing simply returns
- If expectations are not met **it throws an error**
 - **An error proves that expectations are not met**
 - **On getting exceptions thrown the test runner reports failure**

Jest Asserts

Wednesday, January 5, 2022 10:46 AM

Official Site

<https://jestjs.io>
<https://jestjs.io/docs/using-matchers>

Simple JEST Asserts (Happy Path Asserts)

- Asserts where your code is not likely to fail
 - Not likely to throw exception
 - Number may/may not be prime.
 - Both are expected values
 - Neither represents some kind of error

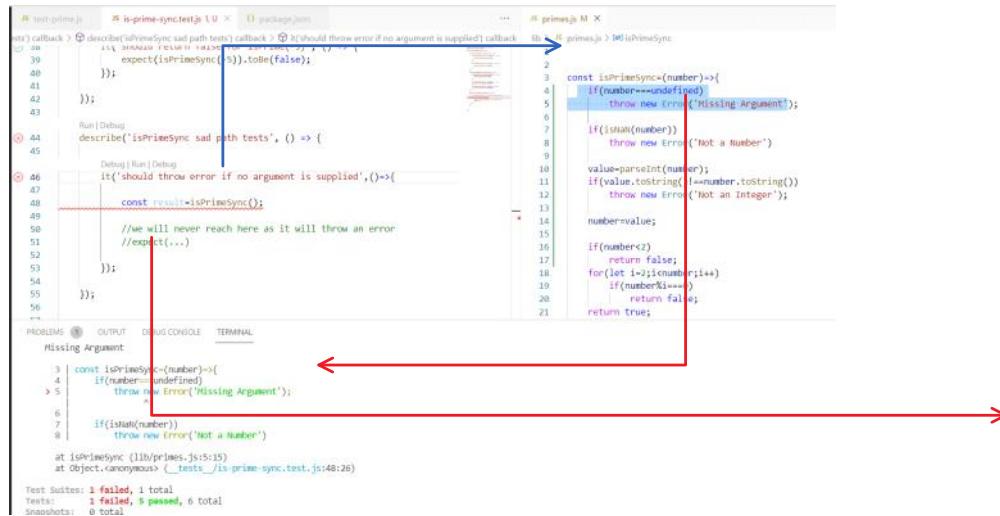
```
expect( actual_value ) .toBe( expected_value )
expect( actual_value ).notToBe(expected_value)
expect( actual_value ).toBeNull();
expect(actual_value).notBeNull();
```

Sad Path Assert

- Sometimes a code is expected to throw exception
- Example
 - We want to throw exception
 - if invalid isbn number is supplied to getBookisbn()
 - If isPrimeSync is passed a non-numeric value

Challenge

- When an exception is thrown the next expect can't be executed.
- You can't expect exception was thrown using normal approach



The screenshot shows a code editor with two files: `is-prime-sync.test.js` and `primes.js`. In `is-prime-sync.test.js`, there is a test for the `isPrimeSync` function that expects a missing argument error. In `primes.js`, the `isPrimeSync` function is defined to throw an error if a non-numeric value is passed. The browser's developer tools show a red arrow pointing from the error message in the console back to the line of code in `primes.js` where the check for a non-numeric value is located.

- You don't reach here to do any expect

Approach#1

1. Write a try-catch block
2. Call your function in try

3. If you get an exception
 - a. The test passed
4. If you don't get exception test failed.

```

54
55     Debug | Run | Debug
56     it('should throw error if no argument is supplied', ()=>{
57
58         try{
59             isPrimeSync();
60             //if we reach here exception was not thrown and test failed.
61             throw new Error('expected exception not thrown');
62         } catch(e){
63
64             expect(e.message).toBe('Missing Argument');
65         }
66     });
67
68 });
69

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

- isPrimeSync tests > isPrimeSync sad path tests > should throw error if no argument is supplied


```
expect(received).toBe(expected) // Object.is equality
      Expected: "Missing Argument"
      Received: "expected exception not thrown"
```

```

62     } catch(e){
63
64         expect(e.message).toBe('Missing Argument');
65
66     }
67

```

- If we reach here, that means `isPrimeSync` didn't throw exception it was supposed to.
- Now we are throwing our own exception to inform that my assertion failed.

Approach #2 (use special jest expect for exception)

test-prime.js

```

68
69
70
71     });
72
73     Run | Debug
74     it('should throw error if non-numeric value is supplied', ()=>{
75
76         expect(()=>isPrimeSync('hello'))
77             .toThrow(); //to throw what?
78     });
79
80     Run | Debug
81     it('should throw error if argument is not integer', ()=>{
82
83         expect(()=>isPrimeSync(20.5))
84             .toThrow('Integer');
85     });
86

```

primes.js

```

3 const isPrimeSync=(number)=>{
4     if(number==undefined)
5         throw new Error('Missing Argument');
6
7     if(isNaN(number))
8         throw new Error('Not a Number');
9
10    value=parseInt(number);
11    if(value.toString()!==number.toString())
12        throw new Error('Not an Integer');
13
14    number=value;
15
16    if(number<2)
17        return false;
18    for(let i=2;i<number;i++)
19        if(number%i==0)
20            return false;
21
22    return true;
23
24

```

- We must call our function inside another wrapper function
 - Test runner will call the outer function which may throw
- Matches message as substring not exact
- Works only if you throw `new Error()`

Throws some exception
• We don't care about exact details

isPrimeSync false value tests

- ✓ should return false for isPrime(20)
- ✓ should return false for isPrime(0) (1 ms)
- ✓ should return false for isPrime(-5) (1 ms)

 isPrimeSync sad path tests

- ✓ should throw error if no argument is supplied
- ✓ should throw error if non-numeric value is supplied (11 ms)
- ✓ should throw error if argument is not integer (1 ms)
- o skipped should throw error if no argument is supplied

Fixing Bug with Test

Wednesday, January 5, 2022 11:13 AM

The screenshot shows a Visual Studio Code interface with two files open:

- test-prime.js**: A Jest test file containing several test cases for the `isPrimeSync` function. Some tests pass (green checkmarks) and some fail (red X's). The failing tests are for `isPrime(0)` and `isPrime(-5)`.
- prime.js**: The source code for the `isPrimeSync` function. It contains a logic error where it returns true for non-prime numbers less than or equal to 1.

A red arrow points from the failing test in `test-prime.js` to the problematic line of code in `prime.js`. Another red box highlights the failing test results in the `test-prime.js` terminal output.

Test helps you detect your problems

A test may fail if

- There is error
 - You are importing wrong file
 - Your function name is in correct
 - There is a syntax error
 - Your Arrange is not correct
 - You forgot to add books to book manager before testing
- Your expectations are wrong
 - You think 2 shouldn't be prime
 - You must check what is expected output
- Your code is incorrect
 - Fixing this is the core goal of Unit Testing

TDD/TFD

Wednesday, January 5, 2022 12:34 PM

Test Driven Development / Test First Development

- Test's have become such an important component in development ecosystem that now we introduce test at a very early stage development.
- In fact we promote the idea of "Test First Code Later"
- Test is not just a mechanism to test if your code is working correctly
 - *It has become a technique to express how your code should be working*
- In this way a test document becomes a **specification document**
 - We specify what we expect from a test to do.
 - Specification can look like
 - It should return a valid book for valid isbn
 - It should throw exception for invalid isbn
 - It should throw error while trying to add book with duplicate isbn
 - It should throw error while adding book with missing required details
 - We can name our file as
 - BookManager.specs.js
 - .specs.js is a valid extension recognized by most test runners including JEST

Red - Green - Refactor cycle

- TDD has three phases

1. Red

- Start with a failing.
- A test that fails
- **Why should I start with a failing test?**
 - Remember it is test first
 - We don't have the working code yet
 - How can test pass?
 - If it passes it breaks TFD
- This phase defines the system specifications
 - What we need
 - Not what we have

2. Green

- Write minimum code to make test pass
 - This code may not accurate or final code
 - Final code may get data from database
 - At this point we may return dummy data
- Goal is to make test pass somehow
- **Why we want to pass the test without correct code?**
 - It is acknowledging the requirement

- It says developer understand what is required.
- This enables quick deployment for testing purpose
- Later we can replace the dummy code with exact code
- Even if the service layer is not giving right data at least UI design can start

3. Refactor

- Change the code from dummy to exact
- Keep running test as you change your code
- The test result will assure your are not deviating from original specs
 - If original specs says invalid isbn should throw error and you are returning null, the test will fail and remind you the real goal

Test Problem

Wednesday, January 5, 2022 12:57 PM

False Positive/False Negative

False Positive

- Some times a test is passing an expectation despite of error in the code
- It may be passing for a different reason.
- This is false positive
- Example
 - Test should return null because we don't have a matching book
 - It returns null because we don't have any book
 - You got expected result but because of a wrong reason
 - You expected it to throw "Invalid Argument" it actually throws because function doesn't exist
 - Your test passes
 - Because you didn't test it properly

False Negative

- Test appears to be failing even if the code is correct.
- Another test may have added some value which your test don't expect

A --> Arrange

Wednesday, January 5, 2022 1:15 PM

```
describe('BookManager getAllBooks tests',()=>{  
  it('should return all books',()=>{  
    bookManager.books=[  
      {},{},{}  
    ];  
    const result=bookManager.getAllBooks();  
    expect(result.length).toBe(3);  
  });  
});  
  
> describe('BookManager addBook tests',()=>{  
});  
  
describe("BookManager getBookByIsbn tests",()=>[  
  it('should return valid book with valid isbn',()=>{  
    const validIsbn='1234';  
    var book=bookManager.getBookByIsbn(validIsbn);  
  
    expect(book).not.toBeNull();  
    expect(book.isbn).toBe(validIsbn);  
  })]  
);
```

- Many of our test will need some data to test against
 - getBookByIsbn
 - getAllBook
 - addBook to test duplicate isbn
- We don't want to add same set of dummy books again and again in all the tests
- We also want to make sure that each test gets a fresh set of value which is not modified by other test

Test Lifecycle methods

1. beforeEach



Most Important

- This function is called before running each test in the current file or describe block
 - If it is called globally it will run before each test in the file
 - If it is called within some describe block it is called before each test in
 - That describe block
 - Any child describe block
- Useful for initializing dummy data
- It is arrange stage of AAA

2. afterEach

- Just like before each
- Called after each test has completed
- May be used for cleaning data

3. beforeEach

- Runs only once before running first test of the block and not before each function

4. afterEach

- Runs to clean up after last test in the block has executed.

Testing function with callback

Wednesday, January 5, 2022 2:26 PM

- If a function doesn't return result directly but via callback we must put our expect in the callback

The screenshot shows two code files side-by-side in a code editor.

Test File (left):

```
Run | Debug
it('should return 4 primes under 10', ()=>{
  const result=findPrimesSyncCallback(0,10,(result)=>{
    expect(result.primes.length).toBe(4);
  });
})
```

Source File (right):

```
24
25
26 const findPrimesSyncCallback = (min, max, cb) => {
27   let result = [];
28
29   for (let i = min; i < max; i++) {
30     if (isPrimeSync(i))
31       result.push(i);
32   }
33
34   cb({primes:result,min,max});
35
36
37 }
```

Testing No Promise async functions

Wednesday, January 5, 2022 2:27 PM

- Most of our codes are going to be asynchronous.
- Results will be delayed and come later.
- How do we test those asynchronous functions
- We may need to test for
 - NodeJS style callback
 - Any other callback
 - Events
 - Even Promises.

1. We must write our expect inside the callback

- Refer to previous page where we tested callback

2. How do we ensure test waits for task to finish.

- Test runner doesn't know when our test will be over
- It finishes after waiting for a given time say (5 seconds)
- If call back is not called within the given time frame it exits assuming test didn't fail
 - A test fails only when expect/assert fails or exception is thrown
 - If there is no exception and no assert failure test will be considered passed
- This is false positive
 - As there are certainly more primes under 10K than 1

The screenshot shows the VS Code interface with two files open:

- findPrimeRange.test.js**:
 - Contains Jest test cases for the `findPrimeRange` function.
 - One test case (line 11) has its assertions circled in red: `expect(error).toBeNull();` and `expect(result.primes.length).toBe(1);`.
 - A red arrow points from this circled code to the implementation file.
 - The status bar at the bottom shows: "Test Suites: 2 passed, 2 total" and "Tests: 3 passed, 3 total".
 - A warning message in the terminal says: "A worker process has failed to exit gracefully and has been force exited. This is likely caused by tests leaking due to improper teardown. Try running with --detectOpenHandles to find leaks. Active timers can also cause this, ensure that .unref() was called on them."
- primes-utils.js**:
 - Contains the implementation of the prime number generator.
 - A red arrow points from the implementation code to the test results.
 - The code uses a `setInterval` loop to find primes between `lo` and `hi`.
 - The implementation continues indefinitely, as indicated by the final line: `const findPrimeRange = (min, max) => {`.

- False positive as test runner existed before it could run `expect()`
- Test ran for a time longer than test framework was willing to wait.

How Jest handles asynchronous (not only promises) calls

1. You should pass a callback to your test function
2. You should call this function after your asynchronous task is over and you have made your assertions
3. This call back informs Jest that function actually finished.

- 4. If you are taking the callback and not using it, JEST will cause a timeout

```
it('should wait till my long running task is over and we call our callback', (cb) => {
  findPrimeRange(1,10000, (error, result) => {
    expect(result.primes.length).toBe(1229);
    //now tell jest you are done
    cb();
  });
});
```

```
Run | Debug
it('should return 1229 primes under 10000', (cb) =>{
  findPrimeRange(0,10000, (error,result)=>{
    expect(error).toBeNull();
    expect(result.primes.length).toBe(1229);
    cb();
  });
});
```

The above code will fail with timeout if expectation fails

- If expectation fails the above code will fail with a time out

```
Run | Debug
describe('findPrimeRange tests', () => {
  Debug | Run | Debug
  it('should return 1229 primes under 10000', (cb) =>{
    findPrimeRange(0,10000, (error,result)=>{
      expect(error).toBeNull();
      expect(result.primes.length).toBe(1);
      cb();
    });
  });
});
```

- If a expectation fails, it throws an exception
- If this line throws an exception that means next line will not be executed

```
findPrimeRange tests > should return 1229 primes under 10000
thrown: "Exceeded timeout of 5000 ms for a test.
Use jest.setTimeout(newTimeout) to increase the timeout value, if this is a long-running test."
4 |   describe('findPrimeRange tests', () =>{
5 |
> 6 |     it('should return 1229 primes under 10000', (cb) =>{
7 |       ^
8 |     );
```

- We never called the callback cb()
- Jest times out**

- It is not an answer to my expectation
- It doesn't say that '1' is invalid
- It says we couldn't test it in time
- Even if this tests finished in time limit

A proper test with the callback

```
Run | Debug
it('should return 4 primes under 10', (cb) =>{
  findPrimeRange(0,10, (error,result)=>{
    try{
      expect(error).toBeNull();
      expect(result.primes.length).toBe(1); //throws exception
    }
    cb(); //test passed
  } catch(ex){
    cb(ex); // test failed with given reason
  });
});
```

- Always do you expect inside a try block
 - Always means while testing for asynchronous task inside call back
- Call empty cb() in case of success
- Call cb with exception in case things go wrong

How to Avoid Redundant code?

- Every callback test will
 - Define try-catch
 - Return empty callback on success
 - Return cb(ex) on catch
- Can we create reusable function so that we don't write this again and again

Create a Closure Function

```
1
2
3 function asyncAssert(jestCallback, userFunction){
4
5   return function(...params){
6     try{
7       userFunction(...params);
8       jestCallback();
9     } catch(ex){
10      jestCallback(ex);
11    }
12  }
13
14 }
15
16
30
31 });
32 Run | Debug
33 it('should return 25 primes under 100', cb=>{
34   findPrimeRange(0,100, asyncAssert(cb, (error,result)=>{
35     expect(result.primes.length).toBe(25);
36   }));
37 });
38
39
40
41
42
43
44
```

Input Parameter

1. **jestCallback**
 - The "cb" parameter we pass to jest function
 - It is called to tell jest function completed.
2. **userFunction**
 - This is the function user actually wants to write without try-catch and jest callback

Returns

- A new closure function
- This closure function is passed as async callback to my api
- Whatever my api passes is passed to this function
- This function calls userFunction in a try block with same parameter
- It handles try-catch callback

Testing events

Wednesday, January 5, 2022 3:25 PM

- Events work in the same way as regular callback
- We can use the same approach to handle the even example

```
Run | Debug
it('should return 15 primes under 50',(cb)=>{

    var count=0;
    findPrimesEvent(0,50)
        .on("PRIME",()=>count++)
        .on("DONE", asyncAssert(cb,_=>{
            expect(count).toBe(15);
        }));
})
```

```
Run | Debug
> it('should return 10 primes between 50 and 100',(cb)=>{ ...
});
```

```
Run | Debug
it('should return error for invalid range',cb=>{

    findPrimesEvent(100,50)
        .on("ERROR",asyncAssert(cb,error=>{
            expect(error.message).toContain("Invalid range");
        }));
});
```

Testing Promise

Wednesday, January 5, 2022 3:37 PM

There are several different ways we can test a Promise

1. Approach #1 Test it just like any other call back test using "cb" parameter

- You can expect inside
 - .then()
 - .catch()

```
Run | Debug
describe('using regular callback approach', ()=>{
  Run | Debug
  it('should have 4 primes under 10', (cb)=>{
    findPrimePromise(0,10)
      .then(asyncAssert(cb,result=>[
        expect(result.primes.length).toBe(4);
        expect(result.primes).toEqual(expect.arrayContaining([3,7]));
      ]));
  });

  Run | Debug
  it('should reject invalid range', cb=>{
    findPrimePromise(100,1)
      .catch(asyncAssert(cb,error=>{
        expect(error.message).toContain('Invalid range');
      }));
  });
});
```

Standard callback design

Promises are special because they are standard

Approach #2

- If our function returns a Promise then jest runtime will automatically wait for promise to resolve/reject
- We don't need "cb" parameter
- We just handle promise the standard way.
- Remember to return a Promise at the end

IMPORTANT!!!!

- Don't merge Approach#1 and Approach#2 together
- If you are returning a promise then don't pass "cb" or using asyncAssert function

```
Run | Debug
describe('using Promise return approach', ()=>{
  Run | Debug
  it('should return 4 primes under 10', ()=>{
    return findPrimePromise(2,10)
      .then(data=>{
        expect(data.primes).toEqual([2,3,5,7]);
      });
  });
});
```

- No callback is passed or used

```

    return findPrimePromise(2,10)
      .then(data=>{
        expect(data.primes).toEqual([2,3,5,7]);
      });
  });

  You must return a promise
  • What promise are we returning?
    ○ .then()
    ○ .catch()

Run | Debug
it('should reject primes with invalid range',()=>{
  return findPrimePromise(100,1)
    .catch(error=>{
      expect(error.message).toContain('Invalid range');
    });
});

});

```

Approach #3

- ES2015 supports **async-await keyword** for Promise
- Jest allows you to define test function as **async**
- Then you can use
 - your standard await to test success
 - Code is simpler without visible call back

```

Run | Debug
it('should return 8 primes under 20',async()=>{
  const result=await findPrimePromise(2,20);
  expect(result.primes).toEqual([2,3,5,7,11,13,17,19]);
});

```

Handling Rejection using **async-try-catch**

```

Debug | Run | Debug
it('should reject invalid range', async()=>{
  try{
    await findPrimePromise(20,2);
  }catch(error){
    expect(error.message).toContain('Invalid range');
  }
});

```

Apporach #4 jest special matchers for promises

```

Run | Debug
describe('using jest matcher for promises',()=>{

Run | Debug
it('should return 4 primes under 10',async()=>{
  await expect(findPrimePromise(0,10))
    .resolves
    .toEqual({min:0,max:10,primes:[2,3,5,7]});
});

Run | Debug
it('should reject invalid range',()=>{

```

- Waits for promise to resolve
 - You may either
 1. await the call in **async function**
 2. Return the promise that is created here
- Remember!
- When you throw an exception from a **async function**, it is not an exception, it is a rejection

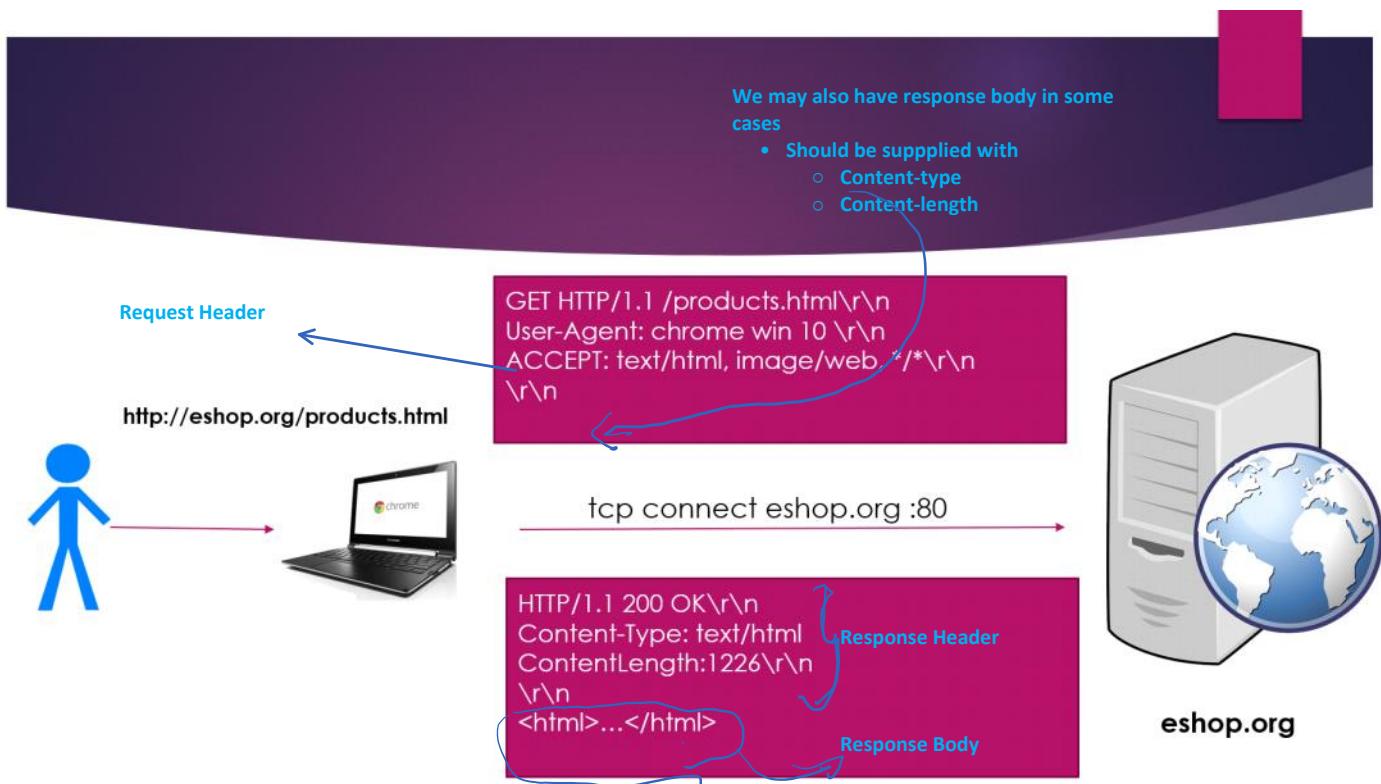
```
Run | Debug
it('should reject invalid range', ()=>{
    return expect(findPrimePromise(10, 2))
        .rejects
        .toThrow(/Invalid range/);
})
});
```

Remember!

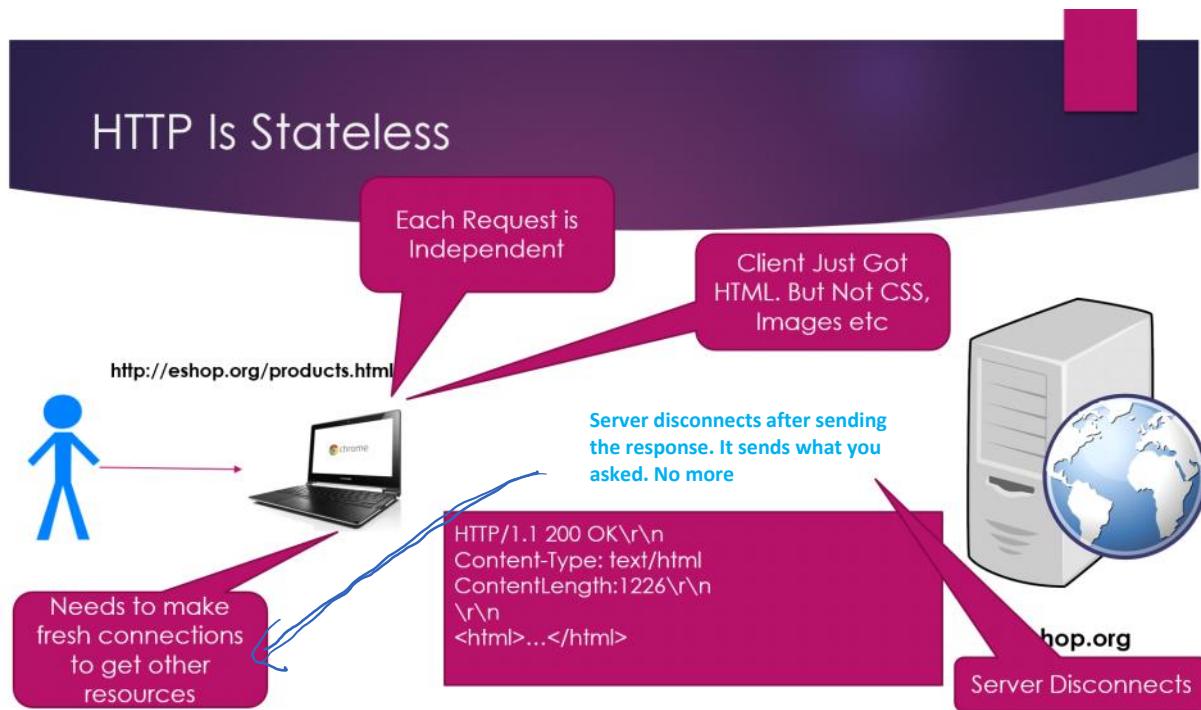
- When you throw an exception from a async function, it is not an exception it is a rejection
- You can't directly check for throws
- You must check for 'rejects' to throw

HTTP Protocol

Wednesday, January 5, 2022 5:10 PM



HTTP Is Stateless



Request For Embedded Resource



GET Request

- ▶ Request to **Retrieve** Resource from the server
- ▶ Is Not Expected to modify the server
- ▶ Can take limited data (query string) to parameterize retrieval
 - ▶ get product from category mobile
 - ▶ <http://esho.com/products?cat=mobile>
- ▶ Default HTTP Method
 - ▶ Every time you type a url its GET
 - ▶ Every time you click a normal hyperlink its GET
- ▶ Only Method that can be
 - ▶ bookmarked
 - ▶ cached
- ▶ On Success should return **200**

Often violated

How else your
browsing history
keeps following
you on every
other website?

POST Request

- ▶ Request to **CREATE** a New Resource on the Server
 - ▶ Also creates Associated **URL**
 - ▶ Returns URL as **LOCATION** Header
- ▶ Can take unlimited data as body of request
- ▶ Can't be
 - ▶ bookmarked
 - ▶ cached
- ▶ On Success should return **201**

Often used for
CREATE
UPDATE
DELETE

Often server
returns 200

PUT/PATCH Request

- ▶ Request to **UPDATE** an existing Resource on the Server
 - ▶ Resource should be identified by the **URL**
- ▶ Can take unlimited data as body of request
- ▶ PUT → Complete Update
- ▶ PATCH → Partial Update
- ▶ On Success should return **202**

PUT is non-
functional in
web
application

DELETE Request

- ▶ Request to **DELETE** an URL and associated Resource from the Server
 - ▶ Resource should be identified by the **URL**
- ▶ Can NO data
 - ▶ URL itself is the identifier of what to delete
- ▶ On Success should return **204**

DELETE is non-
functional in
web
application

BROWSER BEHAVIORS

- ▶ Browser supports only GET and POST
- ▶ All Non-Post requests are converted to GET
- ▶ GET is a bad substitute for PUT/PATCH/DELETE
 - ▶ GET is not supposed to modify the server
 - ▶ PUT/PATCH may need unlimited data; GET can carry limited DATA

WEB APP STRATEGY

- ▶ DEVELOPERS MANUALLY SUBSTITUTE POST FOR
 - ▶ PUT/PATCH
 - ▶ DELETE
- ▶ IN A WEB APP
 - ▶ GET → RETRIEVE
 - ▶ POST → UNIVERSAL MODIFIER

- ▶ 200 OK
 - ▶ Success status for a GET request
 - ▶ Generally used as universal success case
 - ▶ Even PUT responds 200 in a web app
 - ▶ Why?
- ▶ 201 Created
 - ▶ Success Status for a POST request
 - ▶ returns URL or newly created resource in location header
 - ▶ ~~PUT~~ generally retruns 200
 - ▶ More useful for REST Services

~~PUT~~
POST

- ▶ 202 ACCEPTED
 - ▶ Request for modification accepted
 - ▶ Generally used as success for PUT/PATCH calls
 - ▶ ~~PUT~~ is not used in web apps
 - ▶ Why?
 - ▶ More useful in REST services
- ▶ 204 No Content
 - ▶ Request Success, No content to show
 - ▶ Success status for ~~DELETE REQUEST~~
 - ▶ ~~DELETE~~ is not used in web application
 - ▶ More useful for REST Services

- ▶ 301 MOVED (PERMANENT REDIRECTION)
 - ▶ Resource has moved to new location permanently
 - ▶ Location header includes URL of new location
 - ▶ May be due to change in domain or organization structure
- ▶ 302 FOUND (TEMPORARY REDIRECTION)
 - ▶ Resource should be accessed by visiting another URL in current context
 - ▶ Location header includes URL
 - ▶ This is a contextual diversion and not a permanent change
 - ▶ Use Case: Redirection to Login page if trying to access a secured page
- ▶ 304 – Not Modified
 - ▶ Content Not Modified since you last accessed it
 - ▶ Request contains header called **If-Modified-Since**
 - ▶ Redirected to your own cache
 - ▶ No Content returned
 - ▶ No Location header is added to response

- ▶ 400 Bad Request
 - ▶ Invalid Data in Form
 - ▶ Bad Query String
 - ▶ Validation Failure etc
- ▶ 401 Unauthorized
 - ▶ You are not authorized to access this resource
 - ▶ Forgot to login?
 - ▶ Often replaced by a pair of 302
- ▶ 403 Not Permitted
 - ▶ Not Permitted to access the resource
 - ▶ User may even be Authorized
 - ▶ May Not have sufficient credentials
 - ▶ Business Logic May not Allow
 - ▶ ADMIN may not be allowed to log-in from remote terminal
 - ▶ You may not be allowed to break the workflow
- ▶ 404
 - ▶ ~~Page Not Found~~
 - ▶ Resource Not Found

A Simple Http Server

Wednesday, January 5, 2022 5:35 PM

1. Server should listen to a port on the machine

- a. Default port is 80
- b. For development we may choose any other port
 - Example: 4000
 - Current machine can always be referred using
 - Name of the machine: **shivoham**
 - Common loop back name: **localhost**
 - Ip address: **127.0.0.1**

Your server address is

<http://servername:<port-number>>

- Port number is optional to provide if it is 80

For development you server may become

<http://localhost:4000>

- Remember vscode live server runs on port: 5500

2. Now server can receive all the request on assigned address

3. Server should respond back to client request as they are made

NodeJS Http Server

Wednesday, January 5, 2022 5:38 PM

```
const http=require('http');

function requestHandler(request,response){
    //request contains all request header parameters
    //console.log('request',request);

    //response is a writeable stream that sends information to the server
    response.write(`${request.method} ${request.url} Welcomes You!`);
    response.end();
}

const server = http.createServer( requestHandler );

const port=4000;

server.on('error', error=>console.error('error:',error.message));

server.listen(port, success=>{
    |   console.log(`server started: http://localhost:\${port}`);
});
```

NOTE

- Our server doesn't know anything we haven't coded for
- It can't serve regular pages like
 - Html
 - Css
 - Javascript etc

1. To create a server we need "http" module
 - It is built-in module with NodeJS
 - No "npm install" required
2. This step comes after server has started and user receives the request
 - We get two objects
 - Request
 - A readable stream
 - Contains request header
 - Body
 - Query string
 - Response
 - A writeable stream
 - User for
 - Setting status code
 - Sending response to client
3. We create a Server and pass a request handler function that will handle all the request that will come in future
4. We listen for user request on a given **port**
 - It can be any value between 0-65535
 - Recommended to use 1024+ value for development server
 - Generally we use
 - HTTP --> 80
 - HTTPS -->443
5. Server emits "error" event in case server can't start
 - You can't start multiple servers on same port

Sending a post request

- You can't type a post request in browser.
- You can create an HTML with a form with method "POST"
- It can send the POST request

The screenshot shows a developer's workspace with three main components:

- Code Editor:** A Visual Studio Code window titled "index.html" displays the following HTML code:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <h1>Magic Login!</h1>
    <form method="post" action="http://localhost:4000/magic/login">
        <button type="submit">Magic Login</button>
    </form>
</body>
</html>
```

- Terminal:** A terminal window titled "server.js" shows the command "node server.js" being run.
- Browser:** A Microsoft Edge browser window titled "Document" shows the rendered HTML with the heading "Magic Login!" and a "Magic Login" button.

Sending Static Content

Thursday, January 6, 2022

12:41 PM

```
function requestHandler(request, response){
    //request contains all request header parameters
    //console.log('request',request);

    if(handleStaticUrl(request, response))
        return; //if it is static file it is already sent.

    console.log('processing dynamic url');

    //handle dynamic url here.
    let message=`${request.method} ${request.url} Welcomes You!`;
    if(request.url.startsWith('/time'))
        message=JSON.stringify({time:new Date().toLocaleTimeString()});
    else if(request.url.startsWith("/date"))
        message=JSON.stringify({date:new Date().toLocaleDateString()});

    //response is a writeable stream that sends information to the server

    response.write(message);
    response.end();
}
```

- Our Request handler receives all requests
 - It is its job to handle all requests
- Here we delegate request response to another function to check if current request is for a static file like
 - Html
 - Css
 - Js
 - Json
 - Images
- If it is a static file, then handleStaticUrl will handle that url
- Else
 - We will write our own logic here!
 - Generally to send dynamic data

```
function handleStaticUrl(request, response){
    //Take the URL
    let url=request.url; // --> /details.html?isbn=1234

    if(url=="/" || url=="")
        url="/index.html"; //default file

    //Remove query string
    const [urlPath]=url.split("?");
    const localPath=path.join(__dirname,'public',urlPath);
    console.log('localPath',localPath);

    //checking path is very short. so let us use sync
    if(!fs.existsSync(localPath)){
        let ext=localPath.split('.').pop(); //take the last path of the file
        if(ext.length){
            //file should be a static file but it doesn't exist
            response.statusCode=404;
            response.end(`File NOT Found: ${url}`);
            return true;
        }
        return false; //this is not a static file, so I will do nothing
    }

    fs.createReadStream(localPath).pipe(response);
    return true;
}
```

Static Resource (URL) Handler

1. We check the current url
 - If url is empty or "/" we assume user wants index.html
 - In practice this file can be any of
 - Index.html
 - Default.html
 - Index.html
 - Default.htm
 - Start.html
 - Start.html
 - ...
 - We will assume only index.html
2. We remove any query string from the url to get actual physical file name
3. We create a local path
 - We are assuming all our static files will be present in a folder "public" in current directory
 - We build a path with three components
 - Current directory path
 - Public
 - Path from the url
4. Now we check if this url actually exists
 - Since checking if file exists is a very quick job (we are not reading file content etc) we are using sync method here
5. If we couldn't find this file, that means url is dynamic and we will not process it.
 - 5.1 If url has an extension
 - Return statusCode(404)
 - Send message "NOT FOUND"
 - Return true
 - ◆ We have handled the problem
 - 5.2 If url has no extension
 - Return false
 - Now main request handler will do whatever it wants
6. Now we need to send this file to the end user.
 - response is a Writeable Stream
 - We can pipe any readable to it
 - We can create a readableStream for the file user wants
 - We pipe it to response
 - User gets the response
7. We return true to tell main requestHandler that response is sent

and you don't need to do anything.

What a typical web server will do

Thursday, January 6, 2022 1:00 PM

A typical web server may do the followings

1. Handle request for static data
2. Handle request for dynamic data
 1. Build an html on server and send to the client
 - Known as server side rendering
 2. Return the data as json/xml so that client can use it whatever way they like
 - Known as REST API

1. Handling Static data

- A server will have typically static contents that need to be sent to client as it is
- No server side processing required
- The files are stored on the server and sent to client if they need
- Example
 - Html
 - Css
 - Javascript (is Javascript static)?
 - For server client side javascript is static file
 - It is a program that will run on the client
 - Pdf
 - Zip
 - ...

Consideration

- Isn't this a common activity in all application?
- Should we really write this logic again and again
- Can't there be a third party package that can give me this feature out of box.
- This is not really a business logic but something that should be part of http or similar package

2. Working with Dynamic Data

- Our actual data is stored in some database like
 - MySql
 - Oracle
 - **Postgres**
 - Mongodb
- We should write our logic to get the data from database
- We may need different type of data on different requested url
- Let us consider some example
- Examples
 - Get A Page with all books
 - <http://books.org/book/list>
 - Add a new book
 - <http://books.org/book/add>
 - Get book by isbn
 - <http://books.org/book/info/1234>
 - Edit a book
 - <http://books.org/book/edit/1234>
 - Delete a book
 - <http://books.org/book/delete/1234>
 - User login
 - <http://books.org/user/login>
- These URLs are dynamic as they need to get data from database and do the work
- Fetching data from database can be done only on server.
 - Browser is not allowed to make such request
 - It may expose your database credential
 - There may not be browser side api available for it.

2.1 Rendering HTML pages

- In response to the above url we can return an html page
- This html page has static data
 - Html
 - Head
 - Body
 - Title
 - Tr
- It has data from the database
 - Book title
 - Book author
- This page can't be processed on the browser
 - Browser don't have data
- We need to generate these pages on server
 - Server should run the logic to merge html tag and dynamic data

2.2 REST Service

- We don't build HTML format response on server
 - Building such response is time taking
 - We may need to build different page based on mobile/desktop
 - We may be developing mobile application and not web pages
 - Html will be useless.
- Rest API is again a simple URL processing system
- It also fetches the data from the database or other source
- Instead of returning html it may return raw data in formats like
 - JSON
 - XML
 - Plain Text
 - SOAP
- We have different type of client
 - Browser that wants to display html
 - Android app that displays same information in android widget
 - IOS App that displays same information in ios UI system
- They all can take simple data and display however they like.

Important!

- This is the most popular job on a web server today
- Accept User request as an url
- Generate/process the data on server
- Returns JSON or other format
 - JSON today is the most popular format
 - Other formats include
 - ATOM
 - XML

REST API Url formats

Thursday, January 6, 2022 2:30 PM

- Generally we use a common path for our REST Api starting with **/api/**
- We then define a sub path for each feature
 - /books
 - /authors
 - /reviews
 - /users
- We then define actions for each feature as a combination of URL and HTTP Method

Example

- Books
 - **/api/books**
 - GET —> returns all books
 - POST —> adds a new book
 - **/api/books/:isbn** <— ':isbn' represents a variable
 - GET —> return book by isbn
 - PUT —> modify book by isbn
 - DELETE —> delete book by isbn
 - **/api/books/:isbn/reviews**
 - GET —> show all reviews for book with given isbn
 - POST —> Add a new review for the book with given isbn
 - **/api/books/:isbn/review/:id**
 - GET —> find one review about with with :isbn where review id is :id
 - **/api/books?q=term**
 - GET —> search book based on term
- Author
 - **/api/authors**
 - GET —> get a list of all authors
 - POST —> add new author
 - **/api/authors/:id**
 - GET —> get details about a single author with given id
 - PUT —> update author with given id
 - DELETE —>delete author with given id
 - **/api/author/:id/books**
 - GET —> get a list of all books by given author
 - **/api/authors/search?q=term**
 - GET —> Search all authors matching the given term

- **Users**

- **/api/users**
 - **GET** —> get a list of all users
- **/api/users/register**
 - **POST** —> register a new user
- **/api/users/login**
 - **POST** —> login an existing user
- **/api/users/:email**
 - **GET** —> get user info
 - **PUT** —> update user info
 - **PATCH** —> update info like email/password
- **/api/users/:email/favorites**
 - **GET** —> get a list of favorite book of this user
 - **POST** —> add a new favorite book for this user
- **/api/users/:email/favorites/:isbn**
 - **DELETE** —> delete a favorite book for this user

How 'http' module works

Thursday, January 6, 2022 2:30 PM

What a typical request processing look like on a server?

```
function requestHandler(request,response){  
    //request contains all request header parameters  
    //console.log('request',request);  
  
    if(handleStaticUrl(request,response))  
        return ; //if it is static file it is already sent.  
  
    console.log('processing dynamic url');  
  
    let responseData='';  
    let statusCode=200;  
    //we need a url to response mapper  
    console.log('request.url',request.url);  
  
  
    if(request.url=='/api/books'){  
        responseData=JSON.stringify(books,null,4);  
    }  
  
    else if(request.url.includes('/api/books/')){  
        const isbn=request.url.split('/').pop();  
        let book= books.find(b=>b.isbn==isbn);  
        console.log('book',book);  
        if(book){  
            responseData=JSON.stringify(book,null,4);  
        } else{  
            responseData=JSON.stringify({error:"Book Not Found",isbn:isbn});  
            statusCode=404;  
        }  
    }  
  
    console.log('responseData',responseData);  
    response.statusCode=statusCode;  
  
    response.end(responseData);  
}
```

Challenges?

- Your entire application will be crammed in a single large request handler
 - Solution: we can call external functions for each url
 - We will still have dozens or more url handled here
- Your request may not just a URL but also depend on
 - Method
 - For example
 - GET /api/books/1234
 - GET details of the book
 - PUT /api/books/1234
 - UPDATE detail of the book
 - DELETE /api/books/1234
 - Delete the book 1234
 - Now How do we handle only based on URL?
- A URL may have a variable inside
 - 1234 is a variable in the current url
 - We may have different variables here like
 - GET /api/books/1234
 - GET /api/books/3333
- We need to separate variables from the URL
- We need to convert each object into JSON string before sending
- We need to write redundant codes such as
 - Handling static data
 - Converting object to JSON before sending
 - Converting string to object before using
 - Handling Query String
 - Handling dynamic path
- If we don't define the event handler or event handler doesn't send a response browser waits for ever

express

Thursday, January 6, 2022 2:33 PM

- Express is a addon module for developing large scale complicated web server
- You can think of express as a wrapper around regular 'http' module
- It has a tons of features to make our code more manageable

Some common feature

Basic Features

- Always returns a response even if there is no handler
 - Default response is a 404 with simple not found message
- Has built-in logic for serving static content
- Has separate functions to handle different types of URL
- Can handle URL with variables
- Has a simple method to set
 - Content type
 - Status
 - Actual body

Advanced Feature

- Routers
 - Have router to handle different application features separately
- Middlewares
 - Concept of middleware to separate common jobs such as
 - Checking for authentication
 - Adding logging
 - Adding caching
 - Filtering client api

Getting Started

```
npm install express
```

Starting with express project

Thursday, January 6, 2022 2:51 PM

1. Create a new folder for the workspace
2. Create empty app.js file
 - This will be the starting point of our application
3. Create package.json

```
$ npm init --y
```

4. Set up script section of package.json

```
✓ default
"scripts": {
  "test": "echo \"Error: no test specified\" && exit 1",
  "start": "node app.js",
  "dev": "nodemon app.js"
},
```

5. Install express

```
$ npm install express
```

```
  license : 'ISC',
  "dependencies": {
    "express": "^4.17.2"
  }
```

6. Run your application

```
$ npm run dev
```

Creating the basic express server

- Process is similar to working with http

1. Get express module

```
const express= require('express');
```

2. Create an app by calling express()

```
//creates an express app instance
```

```
const app = express();
```

```
//start listening to a port
```

```
const port=4000;
```

3. Create a server by calling app.listen()

```
const server=app.listen(port, _=>{
  | console.log(`server started: http://localhost:${port}`);
});
```

4. Handle 'error' event on server

```
server.on('error',error=>console.error('error:',error.message))
```

Handling request for a particular URL

- Express app has functions matching the names of http verbs
 - get
 - post
 - put
 - delete
 - patch
 - ...
- We call these functions by passing two parameter
 - Url
 - Function to handle url

- Takes
 - request
 - response
 - Next
 - ◆ We will talk about this optional parameter later

```
app.get('/api/books',(request,response)=>{
  response.send('getting a list of all books');
});

app.post('/api/books',(request,response)=>{
  response.send('adding a new book on the server');
});

app.get('/api/books/:isbn',(request,response)=>{
  const isbn=request.params.isbn; //we automatically get
  let book={isbn, title:'The Accursed God', author:'Vivek Dutta Mishra'}
  response.send(book);
});
```

- Method name is associated with http methods
 - Get() will be called for get request
 - Post() will be called for post request
- URL can have variables inside it
 - Variables are automatically added as request.params
- No need to convert your objects to "string" using Json.stringify
 - It is automatically converted by the send function

Assignment 9.1

Thursday, January 6, 2022 3:12 PM

- Create a simple web server with following end points
 - GET /api/books
 - POST /api/books
 - GET /api/books/:isbn
- Take our "books5.json" file from 'resources'
 - DO NOT COPY THE CONTENT IN YOUR JS FILE
- When program starts, use 'fs' module to read the books5.json
 - Read the file content to in **memory array**
- Display the data in the given end point
 - GET /api/books
 - GET /api/books/:isbn

Phase 2

- Implement POST /api/books
- Add the data to your in-memory array
- Save all books back to books5.json using 'fs' module

Now you will have file based persistent database

Calling our REST API Manually

Thursday, January 6, 2022 4:43 PM

- Browser is not a right tool to call your REST API for manual testing
- Browser can support only
 - GET request
- Browser does not support
 - POST request by typing URL
 - You may send POST request by creating an HTML forms
 - Creating form everytime is painful.
- Browser/Form can't send
 - PUT
 - DELETE
 - PATCH
 - ...
- You can't send data by typing URL in browser
 - You may have to send
 - Body of request
 - Eg. Book to add
 - Extra header
 - Eg. Authentication

Solution

- We can use specialized tools for handling such problems
- Common tools

1. Fiddler

- A powerful HTTP inspector and request composer
- You have total control on
 - HTTP header
 - HTTP method
 - Inspect exact data transmitted

2. Postman (Very Popular)

- Web based on downloadable app
- You can send all types of request
- It maintains a history of your older requests
 - You may send them again
- Many programmable features

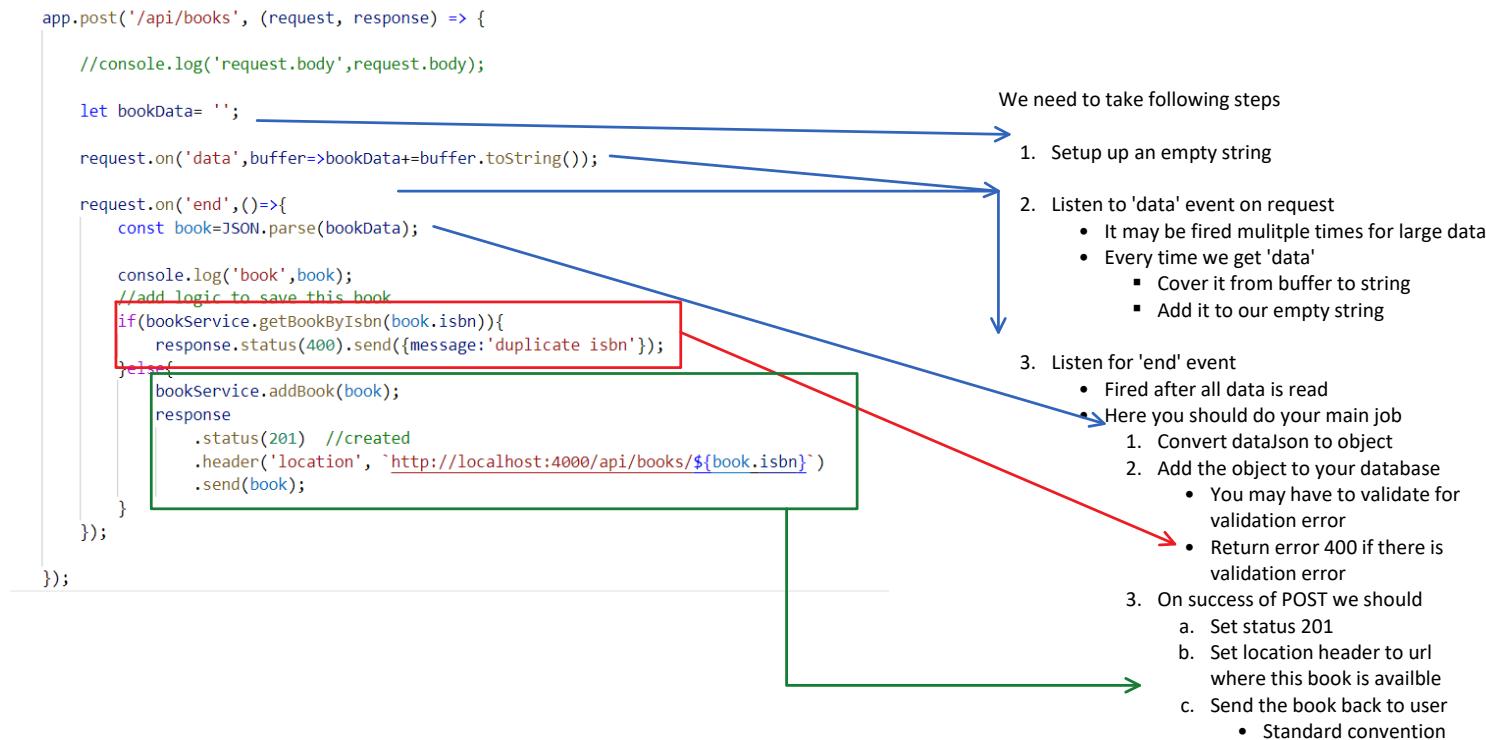
3. Thunderclient (VS Code Extension)

- Integrated into VSCode
- You never leave your VS Code window
- Similar features as POST man
- You can save your requests

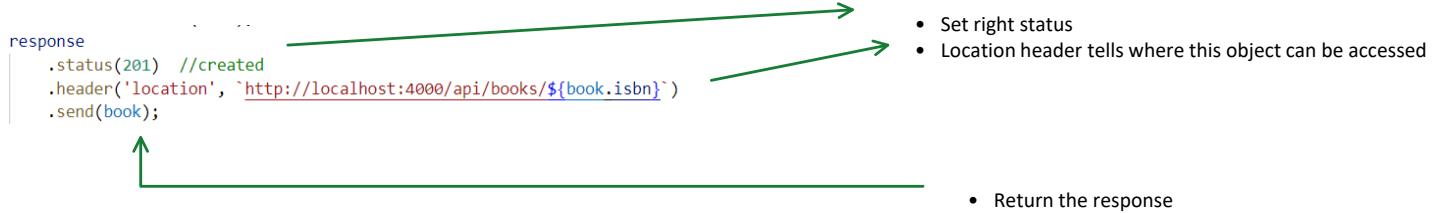
Sending a post request

Thursday, January 6, 2022 5:23 PM

- Post request requires to send the content as the body of request
- It should be added in the body section of your POST request client
- The data is received via request object
- But you will not find any "request.body" property on your object.
- Request is received as a stream of data
 - Request is a ReadableStream
- We will need to extract the values from stream by listening to events like
 - 'data'
 - 'end'



Important Element in response



```

request.on('end', ()=>{
  const book=JSON.parse(bookData);

  console.log('book',book);
  //add logic to save this book
  if(bookService.getBookByIsbn(book.isbn)){
    response.status(400).send({message:'duplicate isbn'});
  }else{
    bookService.addBook(book);
    response
      .status(201) //created
      .header('location', 'http://localhost:4000/api/books/${book.isbn}')
      .send(book);
  }
});

});
```

Response Headers	
Header	Value
x-powered-by	Express
location	http://localhost:4000/api/books/123456789
content-type	application/json; charset=utf-8
content-length	194
etag	W/"c2-sk8t9jj4R4DQTF8eAbq9UCigj0Q"

Request Body Handling

Thursday, January 6, 2022 5:49 PM

- Request body is a Readable Stream
- To get the data you need to create a complex setup
- This will be required in every request done via
 - POST
 - PUT
 - PATCH
- You may end up writing a similar code again and again

Solution#1 → can we convert ReadableStream to a Promise?

Steps1.

- Return a promise
 - Collect the data internally in "data" event
 - Resolve the promise in "end" event
 - Reject the promise in "error" event

```
const promisedReadableStream=(stream)=>{  
  return new Promise((resolve, reject)=>{  
    let data='';  
    stream  
      .on('data', buffer=> data+=buffer.toString())  
      .on('end', ()=>resolve(data))  
      .on('error', error=>reject(error));  
  });  
}
```

Simplifying POST method

Thursday, January 6, 2022 6:00 PM

Original Code

```
app.post('/xapi/books/', (request, response) => {
  //console.log('request.body',request.body);

  let bookData= '';
  request.on('data',buffer=>bookData+=buffer.toString());
  request.on('end',()=>{
    const book=JSON.parse(bookData);

    console.log('book',book);
    //add logic to save this book
    if(bookService.getBookByIsbn(book.isbn)){
      response.status(400).send({message:'duplicate isbn'});
    }else{
      bookService.addBook(book);
      response
        .status(201)  //created
        .header('location', `http://localhost:4000/api/books/${book.isbn}`)
        .send(book);
    }
  });
});
```

Simplified code

```
app.post('/api/books',async (request,response)=>{
  let bookData= await promisedReadableStream(request);
  if(bookService.getBookByIsbn(book.isbn)){
    response.status(400).send({message:'duplicate isbn'});
  } else{
    bookService.addBook(book);
    response
      .status(201)  //created
      .header('location', `http://localhost:4000/api/books/${book.isbn}`)
      .send(book);
  }
});
```

Assingment 9.2

Thursday, January 6, 2022 6:04 PM

- Add the logic for
 - Updating a book details
 - Deleting a book from the server
- Add few new records

Why a separate service layer

Friday, January 7, 2022 11:04 AM

Where should we write our business logic?

Option A (In a separate service File)

```
JS book-service.js M • JS book-routes.js M • JS utils.js ...
JS book-service.js > (e) updateBook
30 const addBook=async (book)=>{
31   books.push(book);
32   await saveBooks();
33 }
34 const updateBook = async(book)=>{
35   let index = books.findIndex(b=>{
36     return b.isbn == isbn ;
37   });
38   if(index >=0){
39     books[index].author = book.author;
40     books[index].title = book.title;
41     books[index].cover = book.cover;
42     books[index].price = book.price;
43     books[index].rating = book.rating;
44     saveBooks();
45   }
46   else{
47     throw new Error('Book Not Found');
48   }
49 }
50 module.exports={
51   getAllBooks,
```

Option B (Directly in the express route)

```
JS book-routes.js > <unknown> > exports > app.put('/api/books/:isbn') callback
79   response.status(204).send({ message: 'Book Not found' });
80 }
81 );
82
83 app.put('/api/books/:isbn',async (request,response)=>{
84
85   let updatedBook = await promisedReadableStream(request);
86   const {isbn}=request.params;
87   let books = bookService.getAllBooks();
88   let index = books.findIndex(b=>{
89     return b.isbn == isbn ;
90   });
91
92   if(index >=0){
93     books[index].author = updatedBook.author;
94     books[index].title = updatedBook.title;
95     books[index].cover = updatedBook.cover;
96     books[index].price = updatedBook.price;
97     books[index].rating = updatedBook.rating;
98     bookService.saveBooks();
99     response.status(202).send(`Book ${books[index].isbn}`)
```

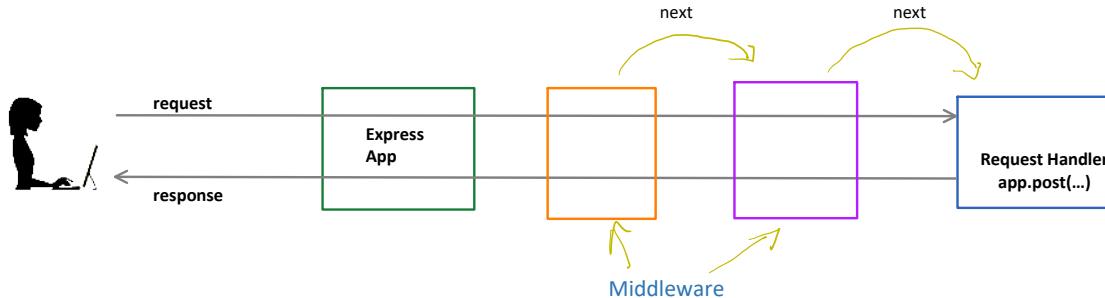
- We have an independent function
- It can be used anywhere
 - Console application
 - Web application
 - Frontend
 - Unit Testing
- Unit testing is important.
 - Your code must be verified that it works properly
- This code doesn't anything about web application
 - Request
 - Response
 - Status code

- This function has no independent existence
- It will be called on a http put request only.
- It can't be called directly
- It can't be reused
- It can't be tested outside an express request using unit testing like jest

Express Middlewares

Friday, January 7, 2022 11:10 AM

- Before your request is handled by express, it can pass through several middleware
- A middleware is function that sits between incoming request and request handler



A simple middleware design

```
const myMiddleware=(request,response,next) =>{  
    //do whatever you want to do with request  
    //object here  
  
    if(condition){  
        response.send(`send output directly  
without passing it forward`);  
    } else{  
        next(); //passes to next function in  
        //pipeline  
    }  
}
```

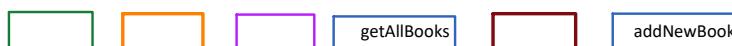
- Middleware sits in the middle of express app and the request handler
- A middleware gets my request before it reaches my handler
- A middleware gets the response from my handler before it reaches express app or client
- We can configure multiple middlewares
- A Middleware can
 - Inspect the request/response
 - Modify the request response.
 - Record/Log my request
- They can choose to
 - Pass incoming request forward toward the handler
 - Reject incoming request by responding back without letting it reach the request handler
- Middleware may decide to send the response itself
- It will not pass the request forward rather send response back to client on its own

How to configure a middleware

- Assume we have two middleware functions
 - orangeMiddleware
 - purpleMiddleware
 - brownMiddleware
- We can configure them in pipeline
- Pipeline (sequence) is decided by the order in which they are written

```
app.use(orangeMiddleware);  
  
app.use(greenMiddleware);  
  
app.get( '/books', getAllBooks );  
app.use(brownMiddleware);  
app.post('/books', addNewBook );
```

Middlewares will follow this sequence



Even request handlers are middlewares

• Note

- Each function passes through all middlewares as long as middlewares call next()
- Any middleware that doesn't call next() breaks the chain and response is sent without going to the next
- Brown middleware doesn't effect "getAllBooks"
 - It is configured after getAllBooks
 - It affects only addNewBook

Middlewares will follow this sequence

Even request handlers are middlewares

```
app.get('/books', getAllBooks)
```

- This function handles request only if
 - It is a GET request
 - `request.url === '/books'`
- What does it do otherwise?
 - It passes to next()

Suppose we made a request for "GET /api/books/1234"

1. It reaches `app.get('/api/books', ...)`
 - Url doesn't match,
 - it calls next()
2. It reaches `app.post('/xapi/books/...', ...)`
 - Method and URL doen't match.
 - It calls next()
3. It reaches `app.post('/api/books', ...)`
 - Method/url doesn't match
 - It calls next()
4. It reaches `app.get('/api/books/:isbn', handler);`
 - Method matches
 - Url pattern matches
 - It calls the handler
 - **It doesn't call next()**
5. Call doesn't reach `app.delete()`

- ... , middleware that doesn't return a value and response is sent without going to the next
- Brown middleware doesn't effect "getAllBooks"
 - It is configured after `getAllBooks`
 - It affects only `addNewBook`

```
app.get('/api/books', (request, response) => { ... });
app.post('/xapi/books/', (request, response) => { ... });
app.post('/api/books',async (request,response)=>{ ... });

app.get('/api/books/:isbn', (request, response) => { ... });
app.delete('/api/books/:isbn',(request,response)=>{ ... });
...
```

Why do we use a middleware

- Sometimes we have a few common task that we may want to perform for multiple requests
- Instead of repeating those codes in every function, we can write them in a middleware
- The middleware will perform the common task
- It will pass the request to the next item

Common Use Cases

1. Logging

- We may want to log our incoming requests everytime
- This can be done by a middleware
- Middleware can then pass the request to next item

```
app.use((request,response,next)=>[
  console.log(` ${request.method} ${request.url}`);
  next(); //to pass to next middleware in chain
]);
```

Advanced Use Case

Build Popular List

- We may store all request for `/api/books/:isbn` to a database
- This way we will build a database that can tell which is the most popularly requested book on our server
- We may have another route to return top 5 books

Log 404 urls

- We may log 404 urls to find out what user is searching that we don't have
- As a product company we may want to bring that item on our website
- For example some one searched for an isbn 92929933
 - We don't have any book with this isbn on our server
 - We returned 404
 - Now our backend should check if this is a valid isbn which is not in my database
 - We may add it later.

2. Blacklisting IP Address

- We may have blacklisted range of ip address.
- If request comes from a blacklisted ip address we may return a response
 - 403 --> Forbidden
- Request from other IP address will served normally

3. Authentication checker middleware

- Some action needs user to be logged in
- A middleware can check if user is logged in or not
- It may reject the request if user is not logged in.
- It may pass to next item if user is logged in.

4. Request body generator

- All post/put/patch middleware will need a body.
- We need to handle ReadableStream to extract the body
- We need to do it again and again in every function that needs the body.
- Can we shift this job to a middleware?

5. Serve Static files

- Remember our serveStatic file was like a middleware
- It checked if the request is for static resource
 - If yes, it returns static file
 - If no, it goes to handle dynamic urls
- We can create a middleware to handle static files

We don't need to create every middleware

- Express comes with a lot of middleware to plugin into our application
- There are many other NPM package for different middleware

Express Middlewares

1. Static file handler

```
app.use(express.static('public'))
```

2. Body parser

- Earlier there was a third-party middleware for this job

```
npm install body-parser
```

- It is still there. But now express has this feature buildi

- Express comes with its own build in body parser
- Body parser can convert request stream to JSON()
- Incoming data can be
 - JSON
 - Form Input

```
//if request contains json data
app.use(express.json());
```

```
//if you are submitted an html form!
app.use(express.urlencoded({extended:true}));
```

- express.json() converts incoming data of json format in to request.body property

("title":"The Accursed God", "author":"Vivek", price:"290")

- express.urlEncoded converts incoming data which is in urlEncoded Format

title=The Accursed God&author=Vivek&price=290

```
POST http://shivoham:4000/api/books/ HTTP/1.1
user-agent: Thunder Client (https://www.thunderclient.io)
accept: "*/*"
content-type: application/x-www-form-urlencoded
content-length: 103
accept-encoding: gzip, deflate, br
Host: shivoham:4000
Connection: close

title=The Mahabharata Project&author=Vivek Dutta Mishra&price=200&rating=4&isbn=22222224&cover=tmp.jpg
```

Conditional middleware

Conditional Middleware

- Sometimes we want to execute a middleware conditionally.
- It doesn't apply to all the routes, but only to a few routes

1. Use Case — Authentication checker middleware

- Some action needs user to be logged in
 - A middleware can check if user is logged in or not
 - It may reject the request if user is not logged in.
 - It may pass to next item if user is logged in.
- Note
 - Not all url will require authentication
 - BookAdd should be authenticated
 - BookList may not need authentication
 - Login doesn't need authentication
 - Remember only non-authenticated users need to login
 - How do I configure such middleware that effect only selected routes.

2. Logging failed search

- We may want to loggin only selected failed search such as
 - Book by isbn
 - Author by id
- We don't want to apply this middleware to all case

How to configure conditional middleware

Option #1 Order them as per requirement

- All actions which should not be effected are configured above middleware
- All actions affected by it should be configured later

```
//unauthenticated actions
app.get('/api/books', getAllBooks);
app.get('/api/books:isbn', getBookByIsbn);
app.post('/user/register', registerUser);
app.post('/user/login' ,loginUser);

//authenticated actions
app.use( aunethentactionChecker);

app.post('/api/books',addBook);
app.put('/api/books/:isbn',updateBook);
app.delete('/api/books/:isbn',deleteBook);
```

Option #2 Create Middleware to Accept what method and URL it should handle

- Our middleware can take urls or methods on which it should be applied

```
const applyOnUrl = ( middleware, ...urls)=>{
  return (request,response,next) =>{
    if(urls.include(request.url))
      return middleware(request,response,next);
```

```

        else
            return next();
    }

}

```

Option #3 Apply middleware to a particular action

- We can pass multiple functions as arguments to app.get(), app.post()
 - All these functions act as middleware

```
app.get( '/api/books/:isbn', log404, getBookByIsbn) ;
```

- Log404 is applied before calling getBookByIsbn

```
app.get('/api/books/:isbn', requestLogger, (request, response) => {
    const isbn = request.params.isbn; //we automatically get
    // console.log('get called for isbn',isbn);
    const book = bookService.getBookByIsbn(isbn);
    if (book)
        response.send(book);
    else
        response.status(404).send({ message: 'Book Not found', isbn });
});
```

A middleware applied before actual task

What do we do with REST Data?

Friday, January 7, 2022 2:42 PM

- A REST service returns JSON (or XML) type of raw data
- This data is not visually interesting to human clients
- As a human we prefer
 - HTML output
 - Mobile device output
- Our web page/ mobile app can make HTTP request for our web server
- The can pull the data

Browser based application

- Browser has a builtin function called "fetch" to fetch data from another url using HTTP
- Fetch can make request to the service and get the data
- Once we get the data we can use standard methods to update the page
 - document.getElementById().innerHTML
- Fetch is asynchronous function

```
const baseUrl='http://localhost:4000/api/books';

const getAllBooks=async()=>{
    //get the book by calling some REST service

    const response=await fetch(baseUrl);
    const books=await response.json();
    // console.log('books',books);

    return books;
}
```

What about updating on the server

- Fetch is a unsuitable name for the function
- The function is really a "request" function
- It doesn't only fetch ("GET")
- It can send all HTTP requests including
 - POST
 - PUT
 - DELETE

IMPORTANT

- While browser can't send PUT/DELETE request using forms, fetch can send PUT/DELETE requests also

Express Router

Friday, January 7, 2022 2:52 PM

- Normally an application has multiple facets
- Consider our book management app
- We need following features

Book Related

- Get a list of all books
- Search books
- Get book by isbn
- Add book
- Edit Book
- Delete book

Authors

- Get a list of all authors
- Get a list of all author names
- Add new author
- Edit author details
- Delete author
- Search author
- List of all books by this author

User

- Login
- Registration
- Profile
- Favorites

- This could lead a large number of URLs (Request Handlers)
- We can think of them as broad functions with sub functions
- Each core aspect will have a core service that they will use
- Each core aspect may have a common url section
 - Each sub functionality will have its url under main functionality
- We can better organize the project by using express router

- Book : (/api/books)
 - Service: BookService
 - Routes:
 - /api/books/
 - GET —> get books
 - POST —> add book
 - /api/books/:isbn
 - GET --> get book by isbn
 - PUT --> update book by isbn
 - DELETE —> delete book by isbn
- Author (/api/authors)
 - Service: Author Service
 - Routes
 - /api/authors/
 - GET —> get authors
 - POST —> add authors
 - /api/authors/:id
 - GET --> get author by id
 - PUT --> update author by id
 - DELETE —> delete author by id

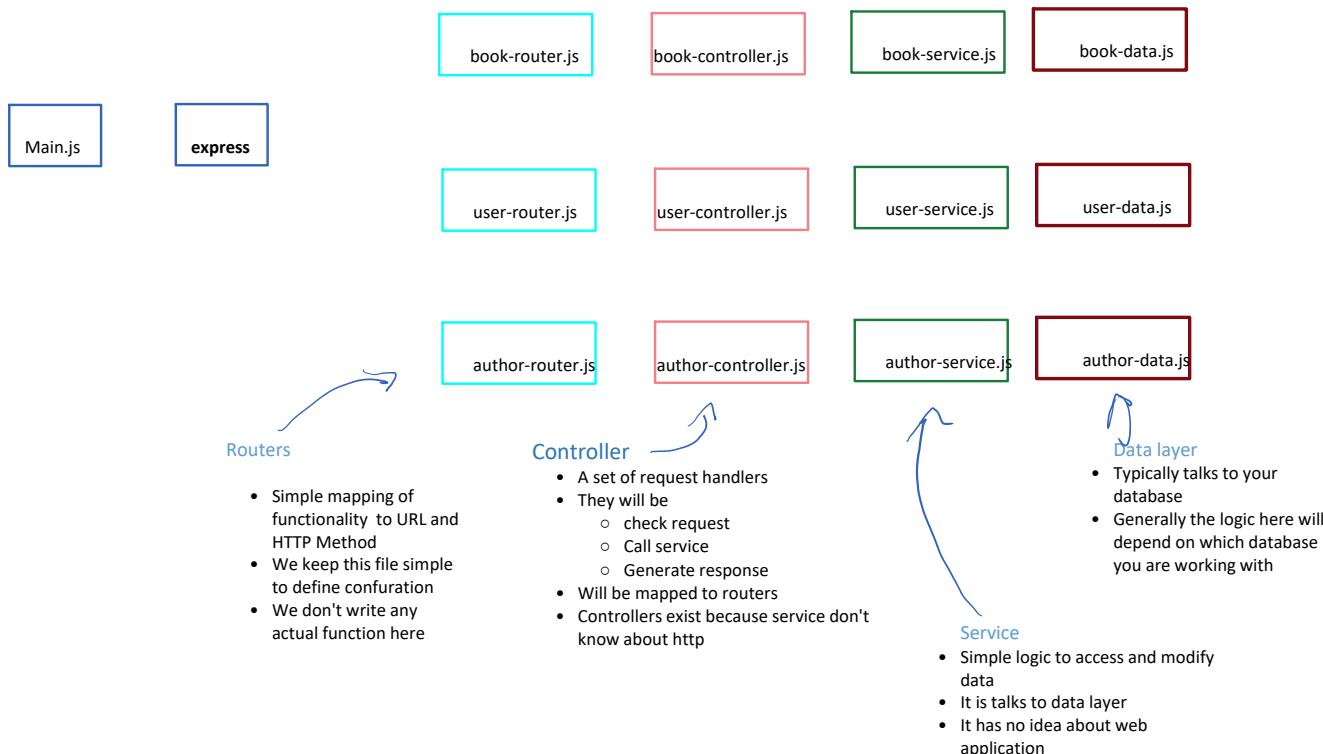
Express Router

- Router is a component in express which allows us to organize related url and functionality
- We can create router for each different section
 - Book router
 - Author router
 - UserRouter
- Each router will have a top level url
 - bookRouter—> /api/books
 - authorRouter--> /api/authors
 - userRouter —> /api/users
- Each router can define sub functionality under it
 - bookRouter.route('/:isbn')
 - Means a url '/api/books/:isbn'
 - Router can handle the routes they need using desired method.
- A router is finally attached to app using "use" function like a middleware

Organizing An Express App

Friday, January 7, 2022 3:05 PM

And express App will have following main components



Project Directory Structure

```
<project root>
  [node_modules] ---> your node packages
  [public] ---> your static client side files
  package.json --> your package file
  .env      ---> environment configuration file
  [src]     ---> your server side code
  [data]    --> all data related items
  [services] --> all service components
  [controllers] --> controller files
  [routes]   --> route configuration
  app.js    --> express configuration
```

main.js —> calls the express configuration

A simple express router

Friday, January 7, 2022 3:39 PM

```
const express= require('express');

module.exports=()=>{
  const router=express.Router();

  router.get('/',(request,response)=>{
    response.send([{title:"book1"},{title:"book2"},{title:"book3"}])
  });

  router.get('/:isbn',(request,response)=>{
    const {isbn}=request.params;

    response.send({isbn,title:'Dummy Title'});
  });

  return router;
};
```

```
const configureRoutes=async(app)=>{
  app.use('/api/books', getBookRouter());
};
```

- Router routes are children of main route mentioned in use

• /api/books/:isbn

A More structured route option

```
module.exports=()=>{
  const router=express.Router();

  router
    .route("/")
    .get((request,response)=>{
      response.send([{title:"book1"},{title:"book2"},{title:"book3"}])
    })
    .post((request,response)=>{
      response.send({isbn:'new-isbn',title:'Dummy Title',status:'created'});
    });

  router
    .route("/:isbn")
    .get((request,response)=>{
      const {isbn}=request.params;

      response.send({isbn,title:'Dummy Title'});
    })
    .put((request,response)=>{
      const {isbn}=request.params;

      response.send({isbn,title:'Dummy Title',status:'updated'});
    })
    .delete((request,response)=>{
      const {isbn}=request.params;

      response.send({isbn,title:'Dummy Title',status:'deleted'});
    });

  return router;
};
```

All actions for /api/books

All Actions for /api/books/:isbn

controller

Friday, January 7, 2022 3:52 PM

```
router
  .route("/")
    .get((request, response)=>{
      response.send([{title:"book1"}, {title:"book2"}, {title:"book3"}])
    })
    .post((request, response)=>{
      response.send({isbn:'new-isbn', title:'Dummy Title', status:'created'});
    });
  
```



- A logic directly written in .get() and .post() can't be tested or reused
- We should send this logic to a separate module
- This logic often includes large code like try-catch etc
- Controller
 - A module that will contain request handlers

Job of a controller

1. Check the request
2. Call the service
3. Send the response
4. Acts as a middle man between service and http

```
const getAllBooks=async (request,response)=>{
  try{
    const books=await service.getAllBooks();
    response.json(books);
  }catch(error){
    response.status(400).json({message:error.message});
  }
};

const getBookByIsbn=async(request,response)=>{
  try{
    const book=await service.getBookByIsbn(request.params.isbn);
    response.json(book);
  }catch(error){
    response.status(404).json({message:error.message});
  }
};

const addBook=async (request, response, next) => {
  try{
    const book=await service.addBook(request.body);
    response.status(201).send(book);
  } catch(error){
    response.status(400).json({message: error.message});
  }
};
```

- 
1. Takes isbn from request
 2. Asks service to give book for given isbn
 3. Sends book to user using response
 4. May send error if book is not found

services

Friday, January 7, 2022 4:09 PM

- Services are plain methods
- They don't interact with http
- They don't know request or response
- Controller gets data from service
- Controller handles request response

```
const router=express.Router();

router
  .route("/")
  .get(getAllBooks)
  .post(addBook);

router
  .route("/:isbn")
  .get(getBookByIsbn)
  //..put()
  //..delete()

;
```

```
JS books-service.js U X
src > services > JS books-service.js > [e] getAllBooks
1
2  const getAllBooks=async()=>{
3
4    return await [{title:'book1'}, {title:'book2'}, {title:'book3'}];
5
6  }
7
8  const getBookByIsbn=async(isbn)=>{
9
10   if(!isbn || isNaN(isbn))
11     throw new Error("invalid isbn");
12   return await {isbn, title:"Dummy Title"};
13 }
14
15 const addBook=async(book)=>{
16
17   book.isbn='new-isbn';
18   return await book;
19 }
20
21 module.exports={
22   getAllBooks,
23   getBookByIsbn,
24   addBook
25 }

JS router.js U
src > controllers > JS books-controller.js > [e] addBook
4
5  const getAllBooks=async (request, response)=>{
6    try{
7      const books=await service.getAllBooks();
8      response.json(books);
9    }catch(error){
10      response.status(400).json({message:error.message});
11    }
12  };
13
14 const getBookByIsbn=async(request, response)=>{
15   try{
16     const book=await service.getBookByIsbn(request.params.i
17     response.json(book);
18   }catch(error){
19     response.status(404).json({message:error.message});
20   }
21
22 const addBook=async (request, response, next) => {
23   try{
24     const book=await service.addBook(request.body);
25     response.status(201).send(book);
26   } catch(error){
27     response.status(400).json({message: error.message});
28   }
29 }

JS book-router-old1.js U
src > controllers > JS books-controller.js > [e] addBook
4
5  const getAllBooks=async (request, response)=>{
6    try{
7      const books=await service.getAllBooks();
8      response.json(books);
9    }catch(error){
10      response.status(400).json({message:error.message});
11    }
12  };
13
14 const getBookByIsbn=async(request, response)=>{
15   try{
16     const book=await service.getBookByIsbn(request.params.i
17     response.json(book);
18   }catch(error){
19     response.status(404).json({message:error.message});
20   }
21
22 const addBook=async (request, response, next) => {
23   try{
24     const book=await service.addBook(request.body);
25     response.status(201).send(book);
26   } catch(error){
27     response.status(400).json({message: error.message});
28   }
29 }
```

Get book from the database...

Database

Friday, January 7, 2022 4:13 PM

- Normally every application need to store persistent data
- The data is stored on server so that it can be accessed from anywhere
 - Only a small limited info is generally stored in client side
 - localStorage
 - Cookies
- We may store the data in various ways include
 - Simple files like json

Problem

- Simple json files can't handle very big data efficiently.
- We may have a problem if more than one client is trying to add the data to the file.
- Here we are loading entire json file in one go.
 - If we have millions of records this can be highly inefficient.
- We generally prefer database servers

Data Base servers

- They are independent server application
 - Think of them like webserver but following different protocol
- They can be present on
 - The same computer
 - Running on different port
 - Different Cloud computer
- Our application can send request to them in specific database format
- It can get response from them.
- Generally each different database type will provide its own npm package to interact with it.

Popular Choices

RDBMS System

- Generally store information as
 - Database
 - Tables
 - Rows
 - Columns
 - Relationship

- Accessed using SQL Queries

MySql

- owned by oracle
- Has free version
- Most commonly available on shared web hosting servers

Oracle

- Commercial and popular data base service

Sql Server

- Microsoft RDBMS server
- Popular with Microsoft technology stack

Sqlite

- A light weight version of RDBMS Server
- Generally used for development and testing only
- Not for production
- Is also available on mobile devices
- Mobile apps generally use them for storing data locally
- Popular choice for mobile applications

Postgres

- Yet another RDBMS server like mysql and sql server
- Available for free downloads
- Cloud servers available as part
 - Azure
 - Google Cloud
 - AWS
- **We are going to use this database.**

NO SQL Database system

- Most confusing acronym
- NO SQL ——> Not Only Sql
- Doesn't store data as table, row, column or relationship
- Generally each nosql system attempts to solve a different problem
- Common Design
 - Object graph
 - JSON
 - Key value pair
- They have different objectives

- Very fast access
- Replication
- Single Key access
- Handling Non-structured dynamic data

MongoDB

- One of the prominent non-sql database
- **Most popular choice in NodeJS application Stack**
- Stores data as Collection of Documents
 - A Document is a JSON object

Redis

- Stores information key value pair
- Very fast access
- Not meant for a very complex data structure
- Useful for storing
 - Session data
 - Configuration

Postgres

Friday, January 7, 2022 4:54 PM

RDBMS System

- Core concepts

1. Database

- Store house for all your information.
- A large block of information generally storing every aspect of your application

2. Tables

- A table is generally used for storing information related to a single entity
 - Eg.
 - BookTable
 - AuthorsTable
 - UsersTable
- A table has row and column
- Rows
 - Each row represent different entity / object
 - Each book will have it's own row in the table
- Columns
 - Each column represents one field/property about the entity/object
 - Isbn
 - Title
 - Name

3. Relations

- Generally entities are related to each other
 - Book and Author are related
 - Book and reviews are related
 - Book and User are related.
- They are implemented using primary key and foreign key concept
- Primary key
 - A unique identifier or a row or entity
 - Can be
 - Autogenerated
 - User provided
 - Must be unique
- Foreign key
 - Primary key of some other entity to relate the two entities together

Example

A master table

- One without dependency on other

- Authors

columns

AuthorID [PK]	Name	Biography	photo
1	Vivek Dutta Mishra		
2	Jeffrey Archer		
3	JK Rowling		

rows

Books

- A book will have an author
- An author may have authored many books

ISBN [PK]	Title	AuthorID	Price	Rating
1234	The Accursed God	1	399	4.3
2222	Harry Potter 1	3		
3333	Kane and Abel	2		
4444	Harry Potter 2	3		
5555	Harry Potter 3	3		
6666	Sons of Fortune	2		

Create Database and Tables

- Postgres has command line and a UI to manage the data without needing an app
- You may administer the database
 - Data base access rights
 - Creating database
 - Creating table
- We need to have database and tables ready before we connect to express.

NodeJS —> postgres

Friday, January 7, 2022 5:08 PM

There two common approach

1. Official node package for postgres

- We need all the knowledge of database access
- If we know how to write our query and create our tables this is our approach

2. Using an ORM solution like Sequelize

- An ORM solution hides the database implementation details
- It allows you to access data in a more object oriented approach
- It generates the necessary queries to interact with database on fly
- It is a more preferred solution
- We will use this most of the time.

Plain vanilla postgres

Sequelize

Friday, January 7, 2022 5:11 PM

- An ORM solution for NODEJS
- Works with databases like
 - Mysql
 - Sqllite
 - Postgres
- Gives an object oriented design for defining our table
- Instead of manually using sql query we work by calling javascript functions

Official Documentation

- <https://sequelize.org/v7/>

0. Installation

- You need to install at least two packages
 1. Sequelize
 2. Official library of underlying data base.

```
$ npm install sequelize pg
```

1. Connecting to the database

- There are several options here

```
const { Sequelize } = require('sequelize');

// Option 1: Passing a connection URI
const sequelize = new Sequelize('sqlite::memory:') // Example for sqlite
const sequelize = new Sequelize('postgres://user:pass@example.com:5432/dbname') // Example for postgres

// Option 2: Passing parameters separately (sqlite)
const sequelize = new Sequelize({
  dialect: 'sqlite',
  storage: 'path/to/database.sqlite'
});

// Option 3: Passing parameters separately (other dialects)
const sequelize = new Sequelize('database', 'username', 'password', {
  host: 'localhost',
  dialect: /* one of 'mysql' | 'mariadb' | 'postgres' | 'mssql' */
});
```

3. Create the Model

- A model is the representation of an object that will be stored in the database
- Each model will generally correspond to one of the tables
- In javascript world it will represent an object
- In database world it will represent a table

```

const { Sequelize, DataTypes } = require('sequelize');
const { sequelize } = require('../connection'); //model will be created in this connection

const Book = Sequelize.define(
  "Book", //Name of this model
  {
    //properties of his model
    isbn: {
      type: DataTypes.STRING,
      primaryKey: true
    },
    title: DataTypes.STRING,
    author: DataTypes.STRING,
    price: DataTypes.INTEGER,
    cover: DataTypes.STRING,
    rating: DataTypes.DOUBLE,
    details: DataTypes.STRING(5000)
  }
);
module.exports={Book};

```

Connecting to database

`sequelize.sync();`

- Syncs my model with the database.
- If tables doesn't exist it create a new table
- It defines all the primary and foreign key relationship

Options:

- `{force:true}` deletes all the tables and recreates them.
- This is useful if you changed your model
- DO NOT USE THIS OPTION ON PRODUCTION DATABASE

Assignment 10

Friday, January 7, 2022 6:44 PM

Add Following Models

- Author
 - Name
 - Id [Primary Key]
 - Biography
 - Photograph
- User
 - UserName
 - Email [PK]
 - Password
 - Photograph

Create Following End Points by Implementing

- Service
- Controller
- Router

EndPoints

- /api/books
 - GET
 - POST
- /api/books/:isbn
 - GET
 - PUT
 - DELETE
- /api/books/pricerange/:min/:max
 - GET
- /api/books/ratingrange/:min/:max
- /api/authors
 - GET
 - POST
- /api/authors/:id
 - GET
 - PUT
 - DELETE
- /api/authors/:id/books
 - GET
 - Get all book by a given author

- /api/user
 - POST —> register user
 - GET —> get a list of all user
- /api/user/login
 - POST —> login user
- /api/user/:email
 - GET —> profile of user
 - PUT —> update user info
 - PATCH —> update password

Environment and Privacy

Monday, January 10, 2022 10:22 AM

Sensitive data in source code

```
const username='postgres';
const password='tr@1n1ng';
const dbName='mbooks';
const host='localhost';
const dialect='postgres'; //which data base you are using. query will be generated accordingly
```

- We often need sensitive private information like passwords and API keys in our source code
- The source code is always committed to version controls like git (private/public)
- There your private data is available for everyone to see.
- People can easily breach your database
 - They can connect and steal/destroy your data using this password.

Very Very IMPORTANT!!!

Never include sensitive private information in your source code.

How do I store the sensitive information?

- We use system environment variables to store these information
- They are not committed to GIT
- They must be setup at the system level
 - The exact processes may differ from one OS to another.
- Refer to setting up variable page below.

How to access the variable in NodeJS

- Node JS contains a special API to access environment variable

`process.env`

- We can access all environment variable as properties of this object

`process.env.DB_PASSWORD`

Or

`process.env["DB_PASSWORD"]`

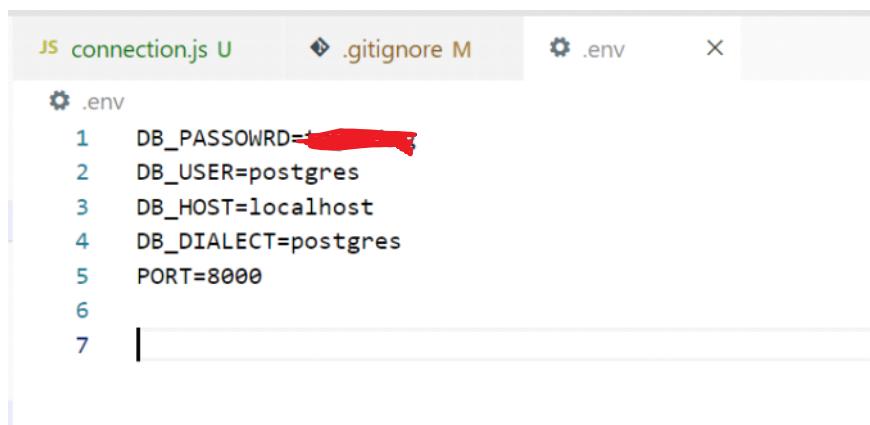
dotenv package

- There is a third-party package that allows us an easy access to environment variable
- Not only it allows access of environment variable, it also allows us to create a speical called ".env" where we can store all the environment variable for our project for development time
- Normally you should ensure that this file is not committed to version control.
- For example you can add this file to ".gitignore" to ensure this file is not uploaded to version control

1. Installation

```
$npm install dotenv
```

2. Creating .env file



The screenshot shows a code editor window with a tab bar at the top. The tabs are labeled 'connection.js U', '.gitignore M', '.env', and a close button 'X'. The '.env' tab is active. Below the tabs is a file content area containing the following text:

```
1 DB_PASSWORD=[REDACTED]
2 DB_USER=postgres
3 DB_HOST=localhost
4 DB_DIALECT=postgres
5 PORT=8000
6
7
```

NOTE:

- Must be on your root level folder (next to package.json)
- No quotes required around variables.
- No blank space after =
 - It will be considered as part of the string

3. Make Sure you don't version commit this file.

- You may include this file in version control exceptions
- If you are using git, you may add it to .gitignore



```
JS connection.js U      .gitignore M X
D: > MyWorks > Corporate > 202112-mobileum-pern > .gitignore
1  private/
2  private/
3  *.tmp
4  node_modules/
5  .env
6
```

4. Make configure environment variable table to read it from .env file also

```
require('dotenv').config();
```

- This will over write any variable read from OS environment table with our own copy.

Special Node Environment Variable.

- In Nodejs we use a special environment variable called

NODE_ENV

- This variable should store the current NODE_ENV which generally can tell why we are running the program
- Popular values for this variable is
 - development
 - production
 - testing
- This variable is important as it can tell the mode of running the application
- We may be using different database for
 - Production
 - Cloud based database
 - We don't want to modify this production time database with dummy data
 - Testing
 - We may be using in memory data base like sqlite rather than postgres
 - We may be using {force:true} to remove data before every start
 - Faster to clear the data in "beforeStart"
 - Development
 - We may be using local database
 - We may want {force:true} to update tables to latest changes
-

Where should we define the NODE_ENV variable

- Option#1 → command line
- Option#2 → system environment variable

Option #3 → package.json

- We can set an environment while running our script from package.json

```

JS main.js M      JS book.js M      JS connection.js M      {} package.json M X
{} package.json > {} scripts > test
1  {
2    "name": "book-api-server-v3",
3    "version": "1.0.0",
4    "description": "",
5    "main": "main.js",
6    > Debug
7    "scripts": {
8      "start": "set NODE_ENV=production& node main.js",
9      "dev": "set NODE_ENV=development& nodemon main.js",
10     "test": "set NODE_ENV=test& jest -w"
11   },
12
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```

```
D:\MyWorks\Corporate\202112-mobileum-pern\book-api-server-v5>npm run dev
> book-api-server-v3@1.0.0 dev
> set NODE_ENV=development& nodemon main.js

[nodemon] 2.0.15
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node main.js`
current environment: development
[nodemon] clean exit - waiting for changes before restart
Terminate batch job (Y/N)? y
```

```
D:\MyWorks\Corporate\202112-mobileum-pern\book-api-server-v5>npm start
> book-api-server-v3@1.0.0 start
> set NODE_ENV=production& node main.js

current environment: production
```

Should we keep this value in package.json?

- Package.json is always committed to git
- Is it safe to add this value package.json?
- YES
 - It is not a sensitive information

```

JS main.js M      JS book.js M      JS connection.js M      {} package.json M      JS er
__tests__ > JS environment.test.js > it('should be running in test environment') callback
1
2   Run | Debug
3   it('should be running in test environment', ()=>{
4     expect(process.env.NODE_ENV).toBe('test');
5   });
6

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Test Suites: 1 passed, 1 total
  __tests__/environment.test.js
    ✓ should be running in test environment (2 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:  0 total
Time:        0.525 s, estimated 1 s
Ran all test suites related to changed files.

```

Windows Environment Variable

Monday, January 10, 2022 10:22 AM

Option #1 (Windows) Setting Temporary data

- You can create variables on the command line.

```
D:\MyWorks\Corporate\202112-mobileum-pern\book-api-server-v5>set PASSWORD=theRedacted
```

```
D:\MyWorks\Corporate\202112-mobileum-pern\book-api-server-v5>echo %PASSWORD%
```

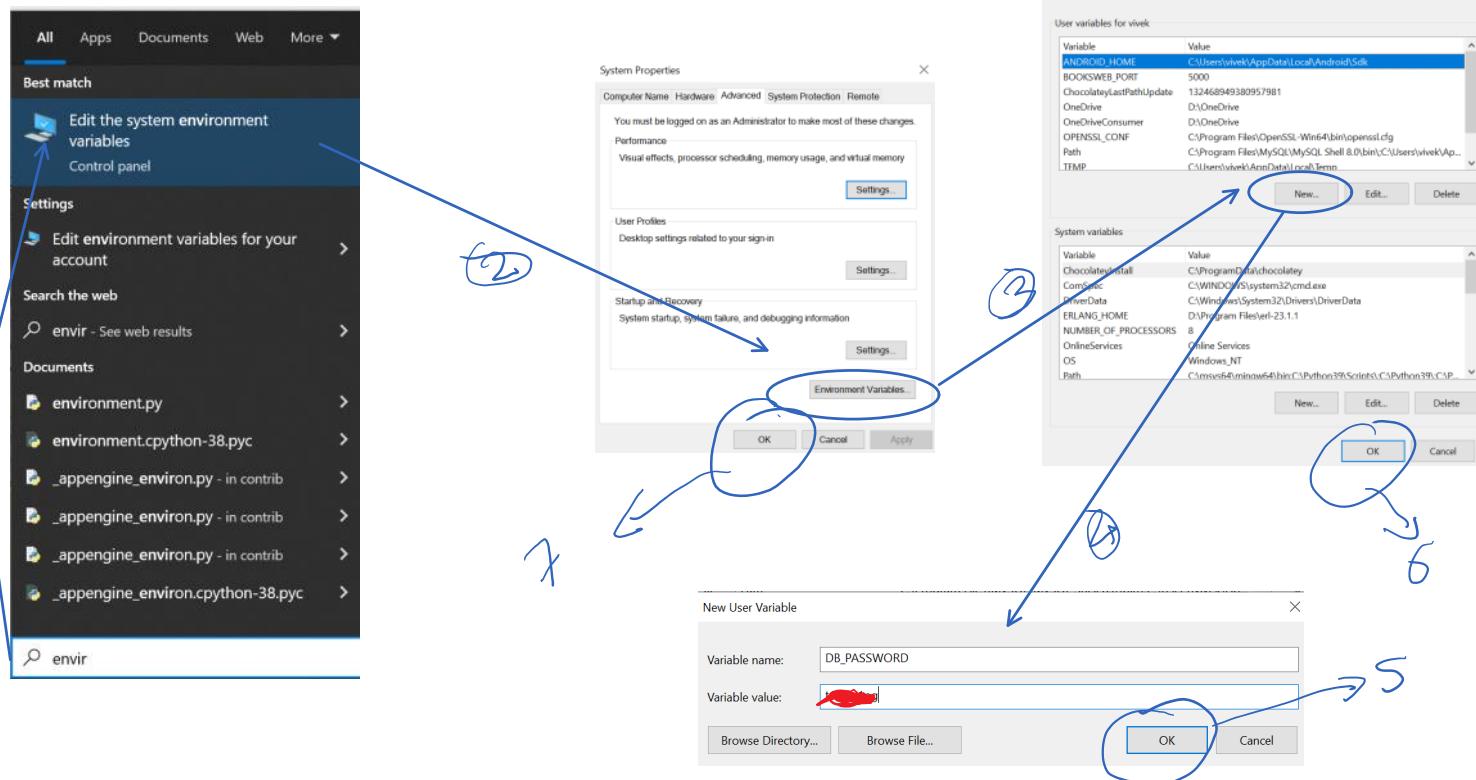
- This is a temporary data.
- It is lost as soon as you close the current terminal

- You can see this variable using
 - Echo command
 - Referring variable in % symbol

Option #2 (Windows) Setting data in a durable way

- You can set it in system environment variable table

- Start Menu → search for Environment Variable



NOTE

- Changes here doesn't effect current open programs or command window.
- Only newly started programs can access this value.
- Close your command window/ VS code etc and restart

Assignment 11.1

Monday, January 10, 2022 12:15 PM

Consider Environment variables like this

The screenshot shows a code editor window with tabs for '.env', 'main.js', 'book.js', and 'connection.js'. The '.env' tab is active, displaying the following content:

```
1 DB_PASSWORD=*****  
2 DB_USER=postgres  
3 DB_HOST=localhost  
4 DB_DIALECT=postgres  
5 DB_NAME_DEVELOPMENT=mbooks  
6 DB_NAME_PRODUCTION=mbooks-production  
7 PORT=8000  
8  
9
```

Blue arrows point from the right side of the slide to specific lines in the .env file: one arrow points to line 2 (DB_USER), another to line 5 (DB_NAME_DEVELOPMENT), and a third to line 6 (DB_NAME_PRODUCTION).

- We can have environment specific names for variables
- We can have environment neutral names which may be common for all.

Update getEnv() function so that it gives your right value following below algorithm

- Assume default environment to be "development"
- When we search for a key such as "DB_NAME"
 - It should search for names for current environment DB_NAME_DEVELOPMENT
 - If present should return the right value
 - If no environment specific names is found, it should search for neutral names like DB_NAME
 - If present return the right value
 - If neither is present return undefined or some default value

getEnv Refactored

Monday, January 10, 2022 1:23 PM

```
module.exports={

    getEnv:(key,defaultValue)=>{

        //always check for key_environment first
        const envKey= `${key}_${process.env.NODE_ENV.toUpperCase()}`; //DB_HOST_DEVELOPMENT
        const value= process.env[envKey] || process.env[key]; //first DB_HOST_DEVELOPMENT DB_HOST
        return value || defaultValue;
    }
};

};
```

- Searches for a key using following workflow
- Assume you are searching for a key "PORT" and your NODE_ENV is "development"
- We will call the function as

```
const port=getEnv("PORT", 5000);
```

1. Search for key "PORT_DEVELOPMENT"
 - If present return it's value
2. Search for key "PORT"
 - If present return it's value
3. If neither of the two key is present
 - a. Return defaultValue 5000

Sequelize Connect

Monday, January 10, 2022 1:26 PM

- There are multiple ways to connect to a database

```
const { Sequelize } = require('sequelize');

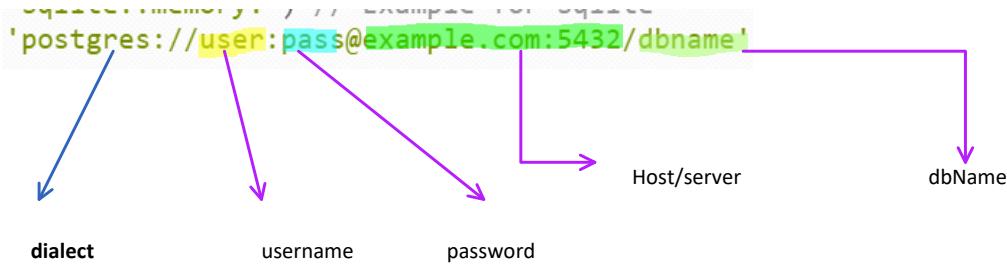
// Option 1: Passing a connection URI
const sequelize = new Sequelize('sqlite::memory:') // Example for sqlite
const sequelize = new Sequelize('postgres://user:pass@example.com:5432/dbname') // Example for postgres

// Option 2: Passing parameters separately (sqlite)
const sequelize = new Sequelize({
  dialect: 'sqlite',
  storage: 'path/to/database.sqlite'
});

// Option 3: Passing parameters separately (other dialects)
const sequelize = new Sequelize('database', 'username', 'password', {
  host: 'localhost',
  dialect: /* one of 'mysql' | 'mariadb' | 'postgres' | 'mssql' */
});
```

Option #1 Pass Connection URI

- A connection URI contains all the required information



Option #2

- Pass all these options separately

```
// Option 3: Passing parameters separately (other dialects)
const sequelize = new Sequelize('database', 'username', 'password', {
  host: 'localhost',
  dialect: /* one of 'mysql' | 'mariadb' | 'postgres' | 'mssql' */
});
```

Connecting to development and production Environment

- We are using elephantSql free tier for production
- ElephantSql provides a uri to connect
- We are using parameters to connect to our local db during development mode

//create a connection

```

//create a connection
const url = getEnv("DB_URL");
|
let sequelize = null;
//if url exists it gets preference from dbName, dbHost
if (url) {
  sequelize = new Sequelize(url);
}
else {
  const username = getEnv('DB_USER');
  const password = getEnv('DB_PASSWORD');
  const dbName = getEnv('DB_NAME');
  const host = getEnv('DB_HOST');
  const dialect = getEnv('DB_DIALECT');
  sequelize = new Sequelize(dbName, username, password, {
    host,
    dialect,
  });
}

}

```

- Check if an URL is present
 - Use the URL.
 - NO need for individual parameters

- If URL is not present then look for other parameters like
 - DB_USER
 - DB_PASSWORD
 - DB_NAME
 - DB_HOST
 - DB_DIALECT

Data Seeding

Monday, January 10, 2022 3:03 PM

- In development system, we may be using dummy data
- Everytime we make change in the model, it purges all the table and the data
- And we are forced to add manually using REST Clients like
 - Postman
 - Thunderclient
- This may be a slow process.
- We can speed up this process by adding seeder logic
 - Seeder can be a function that adds a set of dummy data everytime we update our tables.

```
11 const setup=async()=>{
12
13     //automatically create the tables if not exists
14     //force true will drop table and recreate
15     await sequelize.sync({
16         //force:true, //this will drop current tables and reac
17         force
18     });
19
20     if(force)
21         await seedData();
22 }
23
```



```
> const _books=[ ...
]
module.exports={

    seedData: async()=>{
        console.log('seeding...');
        const {Book}=sequelize.models;
        for(let book of _books){
            await Book.create(book);
        }
    }
};
```

Add a set of dummy data every time we use

`sequelize.sync({force:true})`

- Ensure you do this step only on development/test database and never on production database.
- On production database, it may be done one time if you have the real data with you.

A

Storing a simple array

Monday, January 10, 2022 4:06 PM

- Some time one of the column needs to store an array of simple value.
- If the value is simple and small like int, string we will use array style
- If the value is large like array of book, it must be a different table.

Example of Simple Array

- Book Tag
 - A book may have array of tags like
 - Fiction
 - Best-seller
 - Mythology
 - Each book can have more than one tag
 - We may want supply an array of tags
- User Roles
 - A user may have one or more of multiple roles
 - Admin
 - Editor

How to store the array

- Postgres doesn't support array type
- We have two choices

1. Store the items as a single string separated by some separator

- a. Write a getter/setter to convert value from array to string and string to array

```
roles:{  
    type: DataTypes.STRING,  
    get:function(){  
        this.getDataValue('roles').split(',');  
    },  
    set:function(value){  
        this.setDataValue('roles', value.join(',') );  
    }  
}
```

1. User is actually a STRING type
2. But we can set an array to this field
 - Set method will convert the past array to a string
3. When you get the value next time
 - System will read the String from database
 - It will convert the string to array
 - It will return the value

2. We may store the data as JSON

- Now the array will be stored as JSON string

```
sequelize.define(  
    "Book", //Name of this model  
    {  
        //properties of his model  
        isbn: {  
            type: DataTypes.STRING,  
            primaryKey: true  
        },  
        title: DataTypes.STRING,  
        author: DataTypes.STRING,  
        price: DataTypes.INTEGER,  
        cover: DataTypes.STRING,  
        rating: DataTypes.DOUBLE,  
        details: DataTypes.STRING(5000),  
        tags: DataTypes.TEXT  
    }  
)
```

```
        cover: DataTypes.STRING,  
        rating: DataTypes.DOUBLE,  
        details: DataTypes.STRING(5000),  
        tags:DataTypes.JSON,  
    }  
}
```

tags	json
	[{"fiction","detective","hercule poirot","suspense","murder"]
	[{"harry potter","fiction","fantasy","best-seller"]
	[{"epic","indian","mahabharata","bhishma","history"]
	[{"harry potter","fiction","fantasy","best-seller"]
	[{"harry potter","fiction","fantasy","best-seller"]

Automating our Model defining procedure

Monday, January 10, 2022 3:20 PM

- Normally we keep all our model in a models folder
 - Over a period of time we may add new models.
 - Each model need to be setup in the following way
-
- We define a model
 - Our model definition is wrapped in a default export function that takes 'sequelize' connection
 -

```
module.exports=(sequelize) =>{
  //console.log('building the model Book');
  sequelize.define(
    "Book", //Name of this model
    {
      //properties of his model
      isbn: {
        type: DataTypes.STRING,
        primaryKey: true
      },
      title: DataTypes.STRING,
      author: DataTypes.STRING,
      price: DataTypes.INTEGER,
      cover: DataTypes.STRING,
      rating: DataTypes.DOUBLE,
      details: DataTypes.STRING(5000)
    }
  )
}
```

- We need to add each model in our setup file in the following manner

```
5
6   //write a require for each your model like below
7   require('./models/book')(sequelize);
8 |
```

- This must be done everytime you are adding a new model
- If we forget this step, model will not be defined.

Add all models automatically

1. Check all files in models folder using 'fs' module

Add all models automatically

1. Check all files in models folder using 'fs' module
2. Go through each file name
3. Load them using require
4. Pass sequelize as a parameter.

```
const attachModels= async(baseDir)=>{  
    let modelDirectory=path.join(baseDir, 'src', 'data', 'models');  
    let modelFiles= await fs.readdir(modelDirectory);  
    console.log('modelFiles',modelFiles);  
    for(let modelFile of modelFiles) {  
        let modelFileFullPath=path.join(modelDirectory,modelFile);  
        console.log(modelFileFullPath);  
        require(modelFileFullPath)(sequelize);  
    }  
}
```

Associations (Relation)

Monday, January 10, 2022 3:32 PM

- Often tables are associated to each other
- There are different type of association between tables

One to One Relationship [Not very popular]

- One row of a table is connected to another row in a table
- This can be used for master details kind of relationship
- A short summary of details can be present in one table
- Additional information may be present in another table

Example

- Book Summary and Book Details table

BookSummary

- Sufficient information for book list page

ISBN	Title	Author	Price	Cover Thumbnail
111	The Accursed God	Vivek Dutta Mishra	399	Tag-small.jpg
222	Kane and Abel	Jeffrey Archer	450	k-a-small.jpg

Book Details

- Additional information about the same book
- The two are connected with ISBN

ISBN	CoverLargeURL	Votes	Rating	Details	Languages
111					
222					

One to Many Relationship

- One object of Table A can be associated with multiple objects of Table B
- Example
 - An Author may have written many books
 - A book may have multiple reviews
- The owner object (Author) is considered as Master Table
- The owned object (Book) will have the primary key of the master table as its foreign key

Authors columns

AuthorID [PK]	Name	Biography	photo
1	Vivek Dutta Mishra		
2	Jeffrey Archer		
3	JK Rowling		

rows

Books

- A book will have an author
- An author may have authored many books

ISBN [PK]	Title	AuthorID	Price	Rating
1234	The Accursed God	1	399	4.3
2222	Harry Potter 1	3		
3333	Kane and Abel	2		

4444	Harry Potter 2	3		
5555	Harry Potter 3	3		
6666	Sons of Forutne	2		

Many to Many Relationship

- Two table may have many to many relationship
- Example
 - Book and Reviewer
 - A Book may have many reviewer
 - A Reviewer may have reviewed many Books
- Example
 - Movie and Actor
 - A movie has many actor
 - An Actor has many movies (where he acted)
- Implementation
 - Generally we need a third table (called Auxiliary table) to hold the primary key of both entities

Actors

Actor id	Name
1	Amitabh Bhachchan
2	Dharmendra
3	Sanjeev Kumar
4	Hema Malini

Movies

MovieID	MovieName
1	Sholey
2	Sita or Gita
3	Ram Balram
4	Faraar

MovieActors

MovieID	ActorID
1 (sholey)	1 (amitabh)
1(sholey)	2(dharmendra)
1(sholey)	3(sanjeev)
1(sholey)	4 (hema)
2(sitagita)	2 (dharmendra)
2 (sitagita)	3 (sanjeev)
2(sitagita)	4 (hema)
3(ram)	1(amitabh)
3(ram)	2(dharmendra)
4(faraar)	1(amitabh)
4(faraar)	3(sanjeev)

How to create Association

Monday, January 10, 2022 3:47 PM

<https://sequelize.org/v7/manual/assocs.html>

Author - Book

- One To Many Relationship

- An Author has written Many Book
 - A Book has a single Author

Note

- We are ignoring the possibility for multiple authors for a single book

Steps

1. Define Book Model
 2. Define Author Model
 3. Defines Association between them
 - a. Association may add new columns to the models

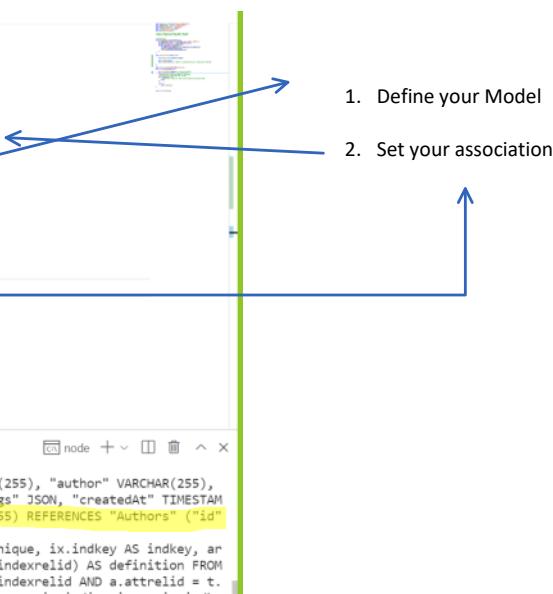
The screenshot shows a code editor with a Node.js file named `setup.js`. The code defines a function `setAssociations` that sets up associations between `Book` and `Author` models. It also contains a `setup` function that attaches models, sets associations, and syncs the database. The code is annotated with blue arrows and lines:

- An arrow points from the `getEnv` call to the `'SYNC_FORCE'=='true'` condition.
- An arrow points from the `attachModels` call to the `//create the models` comment.
- An arrow points from the `setAssociations` call to the `//set association` comment.
- An arrow points from the `sync` call to the `//force true will drop table and recreate` comment.
- A horizontal line spans across the code from the `//force true` comment to the `force` parameter of the `sync` call.

```
const setAssociations=async ()=>{
    const {Book,Author}=sequelize.models;
    AuthorhasMany(Book);
    Book.belongsTo(Author); //Book will automatically get a foreign key "AuthorID"
};

const force= getEnv('SYNC_FORCE')=='true';
const setup=async(baseDir)=>{
    await attachModels(baseDir); //create the models
    await setAssociations(); //set association
    //automatically create the tables if not exists
    //force true will drop table and recreate
    await sequelize.sync({
        //force:true, //this will drop current tables and reacreate them
        force
    });
};


```



By default the relationship is optional

- It is ok if book doesn't have an Author

AuthorId
character varying (255)
[null]

- But in reality it is not ok that book doesn't have an author
 - Book must have an author

Mandatory Association

```
const {Book, Author} = require('sequelize');

Author.hasMany(Book, {foreignKey:{ allowNull:false}});
Book.belongsTo(Author, {
  foreignKey:{ 
    allowNull:false
  }
}); //Book will automatically get a foreign key "AuthorID"
```

We also have a different name for our foreign key

```
const setAssociations=async ()=>{
  const {Book,Author}=sequelize.models;

  Author.hasMany(Book,{foreignKey:{ allowNull:false,name:"authorId"}});
  Book.belongsTo(Author,{ 
    foreignKey:{ 
      name:'authorId',
      allowNull:false
    }
  }); //Book will automatically get a foreign key "AuthorID"
};

};
```

Auto generated Primary Key

```
const {DataTypes}=require('sequelize');
module.exports=(sequelize)=>{
  sequelize.define("Review",
  {
    //no primary key given
    reviewer:DataTypes.STRING,
    rating:DataTypes.DOUBLE,
    title:DataTypes.STRING,
    review: DataTypes.TEXT(2000)
  }
);
```

• Serial indicates "auto generated" serially

```
ray_agg(a.attnum) as column_indexes, array_agg(a.attname) AS column_names, pg_get_indexdef(ix.indexrelid) AS definition FROM pg_class t, pg_class i, pg_index ix, pg_attribute a WHERE t.id = ix.indexrelid AND i.oid = ix.indexrelid AND a.attrelid = t.oid AND t.reloid = 'r' AND i.relname = 'Books' GROUP BY i.relname, ix.indexrelid, ix.indisprimary, ix.indisunique, ix.indkey ORDER BY i.relname;
Executing (default): DROP TABLE IF EXISTS "Reviews" CASCADE;
Executing (default): CREATE TABLE IF NOT EXISTS "Reviews" ("id" SERIAL, "reviewer" VARCHAR(255), "rating" DOUBLE PRECISION, "title" VARCHAR(255), "review" VARCHAR(2000), "createdAt" TIMESTAMP WITH TIME ZONE NOT NULL, "updatedAt" TIMESTAMP WITH TIME ZONE NOT NULL, PRIMARY KEY ("id"));
Executing (default): SELECT i.relname AS name, ix.indisprimary AS primary, ix.indisunique AS unique, ix.indkey AS indkey, array_agg(a.attnum) as column_indexes, array_agg(a.attname) AS column_names, pg_get_indexdef(ix.indexrelid) AS definition FROM
```

Setting up the Association

```
Book.belongsTo(Author,{ 
  foreignKey:{ 
    name:'authorId',
    allowNull:false
  }
}); //Book will automatically get a foreign key "AuthorID"

Review.belongsTo(Book);
```

```
32     |         |     allowNull:false
33     |         |   }
34     |         |}); //Book will automatically get a foreign key "AuthorID"
35
36     | Review.belongsTo(Book),
37     | BookhasMany(Review);
38
39   };
40
41   const force= getEnv('SYNC_FORCE')=='true';
42   const setup=async(baseDir)=>{
43
44     | await attachModels(baseDir); //create the models

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
Executing (default): DROP TABLE IF EXISTS "Reviews" CASCADE;
Executing (default): CREATE TABLE IF NOT EXISTS "Reviews" ("id" SERIAL , "reviewer" VARCHAR(255), "rating" DOUBLE PRECISION, "title" VARCHAR(255), "review" VARCHAR(2000), "createdAt" TIMESTAMP WITH TIME ZONE NOT NULL, "updatedAt" TIMESTAMP WITH TIME ZONE NOT NULL, "BookIsbn" VARCHAR(255) REFERENCES "Books" ("isbn") ON DELETE SET NULL ON UPDATE CASCADE, PRIMARY KEY ("id"));
```

Adding Review and associating it with some book

```
for(let book of _books){

  let dbBook=await Book.create(book);

  for(let i=0;i<5;i++){
    let review= await Review.create(createReview(book.title));
    //update to include bookIsbn in review
    await dbBook.addReview(review); //add this review to the current book
  }
}
```

Assignment 11.2

Monday, January 10, 2022 6:16 PM

- Update your services to include the following end points
- Get books by author
 - /api/authors/vivek-dutta-mishra/books
- Get reviews for a particular book
 - GET /api/book/22293939393/reviews
 - POST /api/books//22293939393/reviews
 - PUT
- Get a single review
 - GET /api/books/22939393/reviews/5
 - PUT /api/books/2233939393/reviews/7
 - DELETE /api/books/399393939/reviews/9

Create a closure to eliminate duplicate controller codes

Eliminating controller

Monday, January 10, 2022 5:37 PM

- Controllers have almost redundant code

```
const anyController= async(request,response)=>{
    try{
        const result= callService( someData);
        response
            .status(some2xxStatus)
            .send(result);
    }catch(error){
        response
            .status(someErrorStatusCode)
            .send(error.message);
    }
}
```

Challenges

- Almost Redundant code
 - We write a similar type of controller for each service action
- 'someData' here may be
 - request.params...
 - request.body
- Success Status code will generally depend on HTTP Method
 - GET —> 200
 - POST —> 201
 - PUT/PATCH —>202
 - DELETE —> 204
- We are not sure about what should the status code for failure?
 - 400 —> validation failure/bad request
 - 401 —> unauthorized
 - 403 —> not permitted
 - 404 —> not found

Our own Http API

1. We will define our own error type that will also include
 - a. Status code
 - b. Details of error
2. We will define our own response type that will also include
 - a. Status code
3. We will handle controllers redundant logic using a closure function

1. Creating a custom error

```
class ResponseError extends Error{
    constructor(message, statusCode=400, info={}){
        super(message);
        this.info={
            message,
            statusCode,
            ...info
        }
    }

    //convert other errors into ResponseError
    static fromError(error,statusCode=400){
        if(error instanceof ResponseError)
            return error;
        else
            return new ResponseError(error.message,statusCode,{cause:error});
    }

    send(response){
        response.status(this.info.statusCode).send(this.info);
    }
}
```

2. Response Message

- We create a subclass of Error
 - I can throw even a non-error object
 - It is easy to test Errors if they are subclass of Error
- "Info" represents the information that is passed as response
 - It will at least include
 - Message
 - Status code
 - We may add extra element such as
 - Missing id
 - Invalid field info
 - Actual failure reason rather than just "invalid data"
- fromError
 - Converts an error that is not ResponseError into ResponseError
 - Thus we add all feature that we need to ordinary error
 - The underlying actual error is supplied as "cause" into "info"
 - If current error is already a ResponseError we return it without change
- Send
 - It uses response to send error after setting the right status code

- Used for sending success responses to user

2. Response Message

```

class ResponseMessage{
  constructor(data, statusCode, headers={}){
    this.data=data;
    this.statusCode=statusCode;
    this.headers=headers;
  }

  static fromData(data){
    if(data instanceof ResponseMessage)
      return data;
    else
      return new ResponseMessage(data);
  }

  send(request,response){
    const responseMap={
      "get":200,
      "post":201,
      "put":202,
      "patch":202,
      "delete":204
    };

    if(!this.statusCode)
      this.statusCode= responseMap[request.method.toLowerCase()] || 200;

    for(let key in this.headers){
      response.set(key, this.headers[key]);
    }

    response.status(this.statusCode).send(this.data);
  }
}

```

- Used for sending success responses to user
- The message includes
 - Actual data to be sent
 - Status code to be sent
 - Leave it undefined if you want default status code based on method type
 - Header to be included
- forData
 - Wraps normal object as ResponseMessage
 - Ensures you have send() function
 - If you pass a ResponseMessage it is used unchanged.
- Send
 - Automatically uses request.method to decide what should be success status
 - If you include any additional header it is added to response
 - Response is send with appropriate
 - Status
 - Header
 - Data

Important!

- If you want to use default status and no header you don't need to create an object of this type

3. Avoided Redundant Controller Logic

```

const controller=async (request, response, next) => {
  try{
    const result=await serviceMethod(request.body);

    ResponseMessage.fromData(result).send(request,response);
  } catch(error){
    console.log('error',error);

    ResponseError.fromError(error).send(response);
  }
}

```

Challenges #1

- How to pass a serviceMethod
 - Controller logic doesn't take additional parameters like serviceMethod

Solution to Challenge#1

- Pass the additional function to controller using a closure
 - An outer function that takes the parameter
 - Returns the controller

```

const controllerCreator = (serviceMethod)=>{
  const controller=async (request, response, next)=> {
    try{
      const result=await serviceMethod(request.body);

      ResponseMessage.fromData(result).send(request,response);
    } catch(error){
      console.log('error',error);

      ResponseError.fromError(error).send(response);
    }
  }
}

```

- Notice controllerCreator is NOT an async function
- It returns immediately
- We don't pass this function to route
 - We call this function while setting route

```
router.get("/", controllerCreator( getAllBooks ) )
```

```

        ResponseMessage.fromData(result).send(request,response);
    } catch(error){
        console.log('error',error);
    }
    ResponseError.fromError(error).send(response);
}

return controller;
}

```

router.get("/", controllerCreator(getAllBooks))

- What you are actually adding to the route is the "controller" function which is returned by controllerCreator

Challenge #2

- What parameter should service method take?
 - Sometimes it needs
 - Request.body
 - Request.params
 - Request.query
 - On rare occasion it may also need
 - Response
 - next

Solution #1

- We may pass the request object directly to service

Problem

- By definition service shouldn't know about http request/response
- By passing request we force service to know about it.

```

const controller=async (request, response, next) => {
  try{
    const result=await serviceMethod(request.body);

    ResponseMessage.fromData(result).send(request,response);
  } catch(error){
    console.log('error',error);

    ResponseError.fromError(error).send(response);
  }
}

```

Final Solution

```

const requestHandler= (serviceMethod)=>{

  const controller=async (request, response, next) => {
    try{
      const argument={
        model: request.body,
        ...request.params,
        ...request.query,
        request,
        response,
        next
      };

      const result=await serviceMethod(argument);
      ResponseMessage.fromData(result).send(request,response);
    } catch(error){
      console.log('error',error);

      ResponseError.fromError(error).send(response);
    }
  }

  return controller;
}

```

- Here we pass a single object that contains
 - request.body as "model"
 - All values in "params"
 - All values in query String
 - For rare cases
 - Request
 - Response
 - Next
- We pass everything as an object but service may take what it really needs

Managing Password

Tuesday, January 11, 2022 11:45 AM

- Password shouldn't be stored in plain text format
- Even site admin/ db admin shouldn't know the customer's password
- Password must be saved in encrypted format so that
 - It can't be decrypted.
- To compare the password
 - We have to encrypt new password and compare with the old one

Bcrypt package

```
npm install bcrypt
```

From <<https://www.npmjs.com/package/bcrypt>>

To encrypt (hash a password) before storing

```
bcrypt.hash(myPlaintextPassword,saltRounds)
  .then(function(hash){// Store hash in your password DB.});
```

- The greater value for saltRound
 - more difficult is to break encryption
 - Longer time it takes to salt and compare

To compare a salted password with plain password

- Internally plain password is again hashed and then compared

```
// Load hash from your password DB.
bcrypt.compare(myPlaintextPassword,hash).then(function(result){// result ==
true});
```

From <<https://www.npmjs.com/package/bcrypt>>

Encrypt

Encrypt some text. The result shown will be a Bcrypt encrypted hash.

\$2a\$12\$scaSvsThtVYjBlqTlYzdauoHqgBucX1267k///8xJsQJjd7eng.tO

My complex Password

Encrypt

Rounds

- 12 +

Decrypt

Test your Bcrypt hash against some plaintext, to see if they match.

Not a match!

\$2a\$12\$scaSvsThtVYjBlqTlYzdauoHqgBucX1267k///8xJsQJjd7enç

My Complex Password

Check

<https://bcrypt-generator.com>

Authentication

Tuesday, January 11, 2022 12:05 PM

How authentication works with REST Service

- We don't use sessions in micro services
- Authentication server may be different from data server
 - You may authenticate elephantsql with github
 - Elephantsql is data server
 - Github is authentication provider
 - This server tells the data server if authentication is valid or not

The Flow

1. We authenticate ourselves with the credentials on Authentication server
 - If our login is successful
 - Authentication server generates a "token"
 - Token includes
 - ◆ Any encrypted data you like
 - ◆ This data can be decrypted
 - A validation time frame
 - ◆ For how long token will be valid
2. We need to pass this token as a part of "Authorization" in request Header for operations that need authorization
3. We can use a middleware to decrypt the authorization header and get the token value out.
 - a. This can add a user object into by request.
4. We can have another middleware to check if valid user info is present in request

Package needed

- Json web token

```
npm i jsonwebtoken
```

From <<https://www.npmjs.com/package/jsonwebtoken>>

2. Generating a webtoken

- It needs multiple information
- Most important if you are creating authentication for your own site (not for others)
 - Data to be encrypt
 - A secret key for encryption / decryption
 - Keep this safe in environment variable
 - Treat information as sensitive
 - Expiry of the token

```
jwt.sign({  
  exp: Math.floor(Date.now() / 1000) + (60 * 60),  
  data: 'foobar'  
}, 'secret');
```

From <<https://www.npmjs.com/package/jsonwebtoken>>

Middleware to convert token into user

- Token should be passed as authorization header in request in below format

```
Authorization: BEARER token
```

A Middleware to validate the token and extract data from it

- This middleware should work for all route
- It doesn't need to reject anyone

```
tokenChecker:(request,response,next)=>{
  console.log(request.headers);
  const authorization= request.headers["authorization"];

  if(authorization){
    const token= authorization.split(" ").pop(); //remove BEARER word

    jwt.verify(token,getEnv("JWT_SECRET"),(error,user)=>{
      if(error){
        request.authenticationError=error;
        console.log('invalid token', error.message);
      } else{
        request.user=user;
        console.log('adding user to request', request.user);
      }
    });
  } else{
    console.log('no token found in the request');
  }
  next();
}
```

- Check if authorization header exists
- If it exists take out the token
- Verify the token
- On success
 - Add data to request
- On failure
 - Add error to request
 - Failure may be
 - Invalid token
 - Expired token

How to authorize a particular route

- We expect few routes available only to logged in user
 - Anyone can see the list of books
 - Only logged in user should be able to add a new book

We create another middleware to check if user is authorized.

```
authorized:async(request,response,next)=>{
  //this middleware will stop any unauthorized request.

  if(!request.user){
    // user is not authorized
    const data={
      message:"You are unauthorized to carry out this action",
      cause: request.authenticationError || "No authorization header found"
    }

    response.status(401).send(data);

  } else {
    next(); //OK! you can move to next action.
  }
}
```

* previous middleware must have added a user in request if token was valid

- Here we check if user is present
 - User is authorized
 - Next action should be allowed
- If user is not found, next action is not allowed
 - Return 401 without calling next()

How to apply middleware

```
router
  .route("/")
  .get(requestHandler(service.getAllAuthors))
  .post(authorize, requestHandler(service.addAuthor));
```

- We are passing to middleware to post
- If first one doesn't call "next()" second one will not be called

```
1 * [
2   "message": "You are unauthorized to carry out this action",
3   "cause": {
4     "name": "TokenExpiredError",
5     "message": "jwt expired",
6     "expiredAt": "2022-01-11T07:32:14.000Z"
7   }
8 ]
```

Assignment 12.1

Tuesday, January 11, 2022 1:04 PM

- Some action requires you to have specific role and not just being authorized
- For example
 - Any one can see books
 - Any logged in user can
 - add review
 - Add a favorite
 - Only moderator/admin can
 - Add
 - Modify
 - Only 'admin' can
 - Delete
 - Get a list of users
- Create a middleware authorize that can check if user is in required role
- It should return 403 if role doesn't match

```
userRouter
  .route("/")
  .get( authorizeFor("admin"), requestHandler(service.getAllUsers))
  .post( requestHanlder(service.register));
```

```
bookRouter
  .route("/")
  .get( requestHandler(service.getAllBooks))
  .post(authorizeFor("admin","moderator"),
  requestHandler(service.addBook))
```

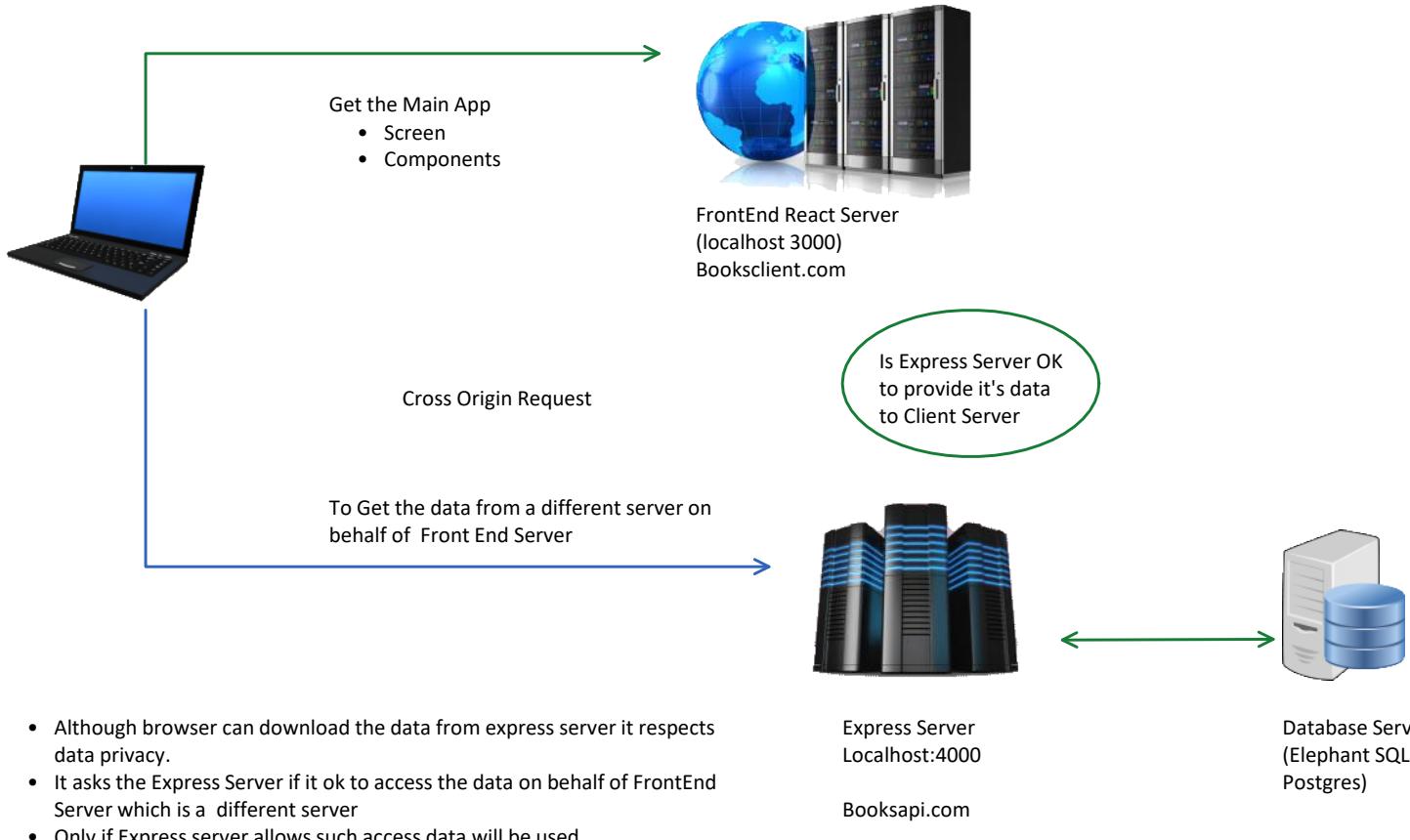
```
bookRouter
  .route("/:isbn")
  .get( requestHandler(service.getBookByIsbn))
  .put(authorizeFor("admin","moderator"),
  requestHandler(service.updateBook))
  .delete(authorizeFor('admin'), requestHandler(service.deleteBook));
```

CORS

Tuesday, January 18, 2022 5:22 PM

Cross Origin Request Security

- Often our Front end server may fetch data from some other API server
 - Use Case
 - BookMyShow is fetching data from different servers like
 - PVR
 - Inox
 - By default browser doesn't allow a cross-origin request unless it is approved by API server



Note

- There is no problem in
 - Same Origin data
 - If requester and data server is same
 - Direct access
 - Using browser
 - Postman
 - Direct access from one server to other
 - localhost 3000 can directly access data from localhost 4000 in one of their routes
- The issue is when
 - A browser based application from Domain P (Presentation) tries to access from domain A (API)
 - Domain "A" must allow such transaction

Enabling CORS on Express server

Step #1

- Install a package for cors

```
npm install cors
```

Step #2

- Configure cors as a middleware

Traditional Web Flow

Tuesday, January 11, 2022 2:15 PM

Challenges

1. Redundant Code

- Duplicate codes across the different web pages

```
details.html
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="UTF-8">
5     <meta http-equiv="X-UA-Compatible" content="IE=edge">
6     <meta name="viewport" content="width=device-width, initial-scale=1.0">
7     <link rel="stylesheet" href="app.css">
8     <title>Document</title>
9   </head>
10  <body>
11    <header>
12      <h1 class='siteTitle'>Book's Web</h1>
13      <ul class='nav'>
14        <li>
15          <a href="/">home</a>
16        </li>
17        <li><a href="/add.html">Add Book</a></li>
18        <li><a href="/authors">Authors</a></li>
19      </ul>
20    </header>
21  </body>

index.html
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="UTF-8">
5     <meta http-equiv="X-UA-Compatible" content="IE=edge">
6     <meta name="viewport" content="width=device-width, initial-scale=1.0">
7     <link rel="stylesheet" href="app.css">
8     <title>Document</title>
9   </head>
10  <body>
11    <header>
12      <h1 class='siteTitle'>Book's Web</h1>
13      <ul class='nav'>
14        <li>
15          <a href="/">home</a>
16        </li>
17        <li><a href="/add.html">Add Book</a></li>
18        <li><a href="/authors">Authors</a></li>
19      </ul>
20    </header>
21  </body>
```

- Suppose we need to add more menu items in the header such as
 - Login
 - Register
 - ...
- We have to update each existing page
- Suppose we need to changes styles
 - We need to update each existing page

Important Concerns

Concerns

- ▶ Various different device sizes access the Internet.
 - ▶ Desktop
 - ▶ Mobile
- ▶ Server is over worked
 - ▶ It needs to create the web page
 - ▶ The webpage should be suitable for different devices

- It needs to create the Web page
 - ▶ The webpage should be suitable for different devices
 - ▶ Client is under utilized
 - ▶ Modern Client has great processing power.
 - ▶ User need to wait between pages
 - ▶ Handling slow internet or disconnection makes the design difficult
 - ▶ Not all clients use HTML

Need of Third party libraries

Tuesday, January 11, 2022 3:03 PM

Need of Libraries & Transpilers

- ▶ **Browser implements DOM/BOM/AJAX differently**
 - ▶ We need a cross browser supports
 - ▶ Replace Redundant DOM/BOM/AJAX
- ▶ **Not all browser supports ES2015+**
 - ▶ We need more structured language feature for large scale application
 - ▶ Typescript or Bable
- ▶ **No features for Application management**
 - ▶ Routing between pages
 - ▶ Structured way of managing the data
 - ▶ Interacting with server
 - ▶ Updating the pages

What is a component

Tuesday, January 11, 2022 3:05 PM

A component is

- Like an User Defined HTML element like
 - <html>
 - Instead of
 - <div>list of books</div>
 - We want to use
 - <BookList/>

What is a component

- ▶ **A component is a piece of UI**
- ▶ **It consists of**
 - ▶ **HTML Templates**
 - ▶ **Styling**
 - ▶ **Javascript**
 - ▶ **Contains Data**
 - ▶ **Event Handling Logic**

HTML 5 Web Component

- HTML 5 offered an official way to create a user defined component
- Thus theoretically we no more need a third party library for creating custom component
- Angular/React came at a time when HTML had no such official standard.
- HTML 5 Web component are like later arrival
- Third party library provides many powerful feature that we continue to use them

HTML 5 WebComponent

- ▶ **HTML 5 allows creation of custom user defined Elements**
- ▶ **They can be simple element like**
 - ▶ <StarRating rating="5" />
- ▶ **Or complex reusable UI like**
 - ▶ <UserLogin post="/register" />
- ▶ **Can contain custom attributes, data**
- ▶ **Not too popular due to compatibility and late arrival**

A Typical Component Driven Design

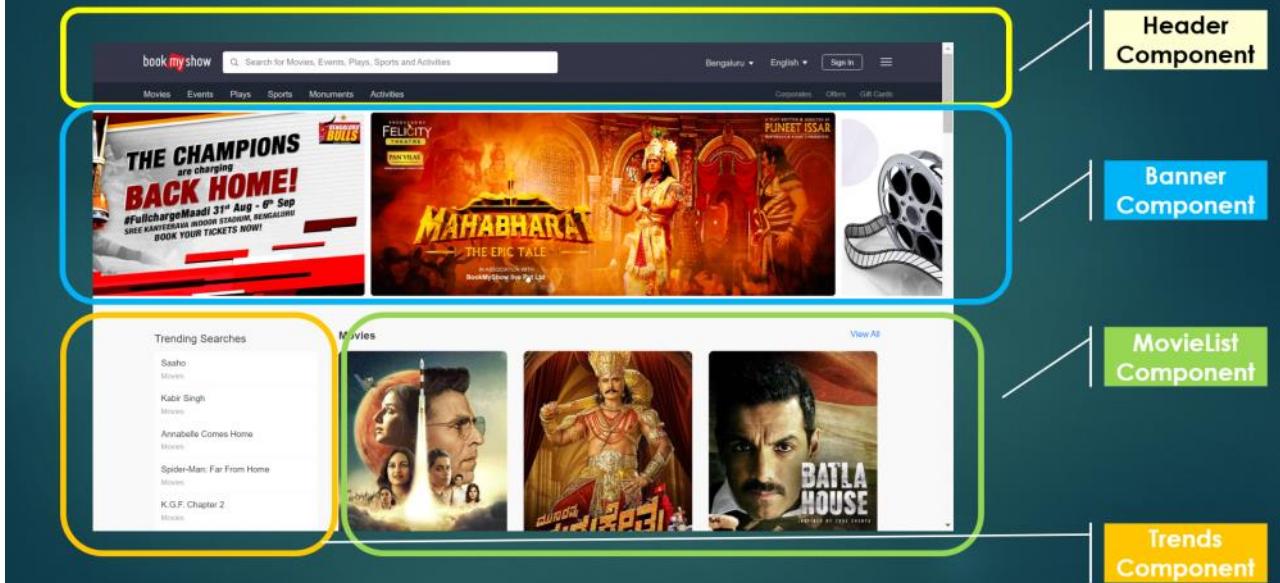
```
<html>
  ...
<body>
  <div class='header'>
  ...
  </div>
  <ul class='movie-list'>
    <li onClick='selectMovie()'>
      ...
    </li>
  </ul>
</body>
</html>
```

```
<html>
  ...
<body>
  <MovieHeader/>
  <MovieList
    movies={movies}
    selected={movie} />
</body>
</html>
```

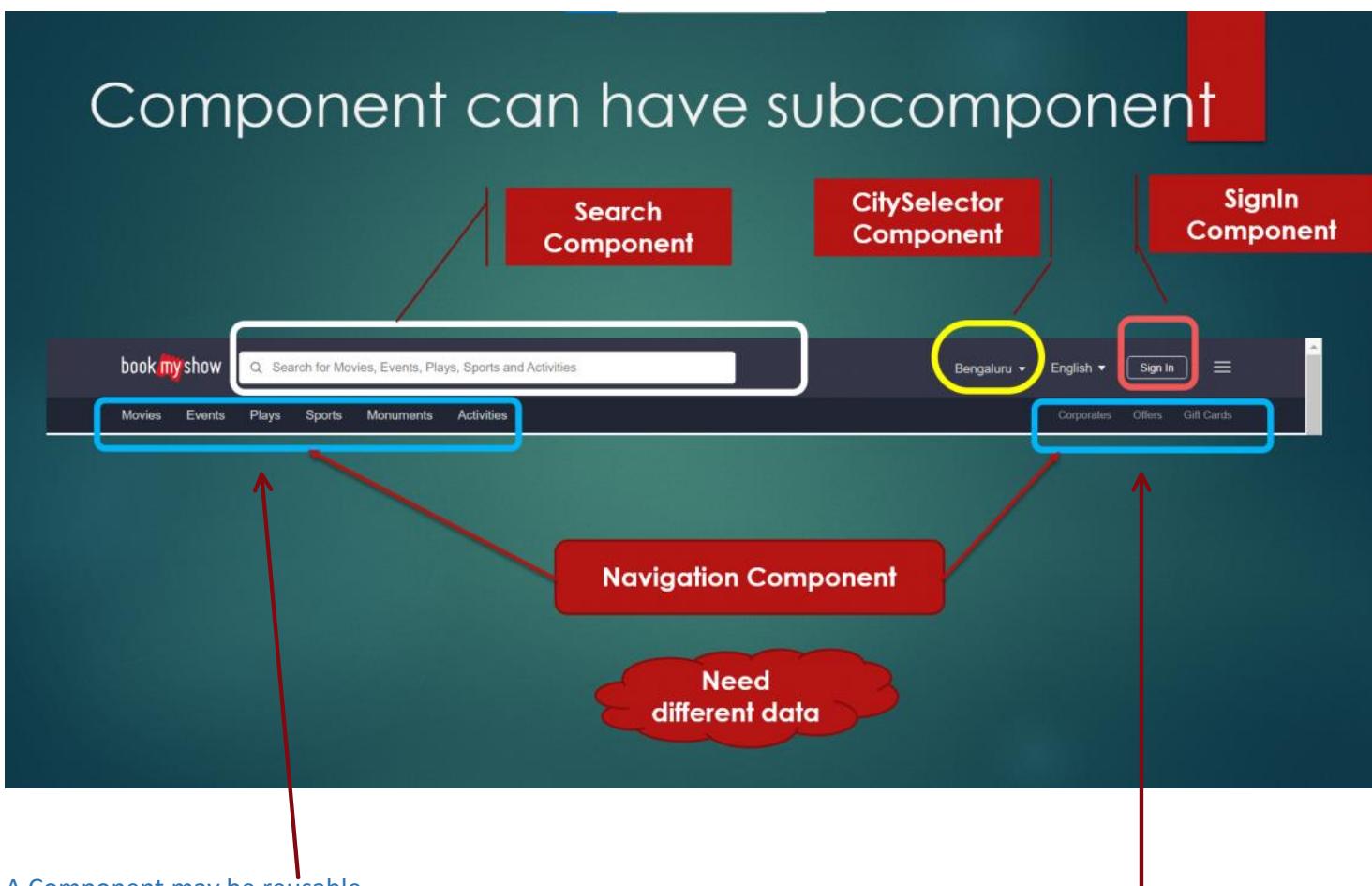
We can visualize our entire screen as composition of multiple components

- The screen itself can be visualized as a component

What is a Component



A component can have child components



A Component may be reusable

- It may use different data but same logic

SPA (Single Page Application)

3:18 PM

Traditional Web Flow

- A traditional web site has multiple pages displaying different information
 - Index.html
 - Details.html
 - Login.html
 - Registration.html
- We need a way to navigate between the pages
 - Clicking hyperlinks
 - Calling windows.navigation.href=...

Problem

- Once you click a link, it is browser's job to go to next page
- Your application is no longer in control
- If there is some error you will get that error from browser
 - Page not found
 - Internet disconnected
- Screen may get blank or unresponsive till the next page is loaded
 - Slower connection

Single Page Application

- Website will have single page containing
 - Blank html (no data)
 - Javascript to get data
 - CSS required
- Once page is loaded it will load UI as User defined Components
- A Component has
 - Own html fragment
 - Data
 - Style
- A Component is loaded using Ajax (internal request)
- Each user request may
 - Update existing component
 - Move from one "screen component" to another
- User will still have an experience of "Multi Page Application" But internally it will be a SPA
 - User will see different content
 - Even the URL on the bar will be changed by our framework giving a feeling of moving to another page.
 - But all will be done by a library on the browser

Advantage

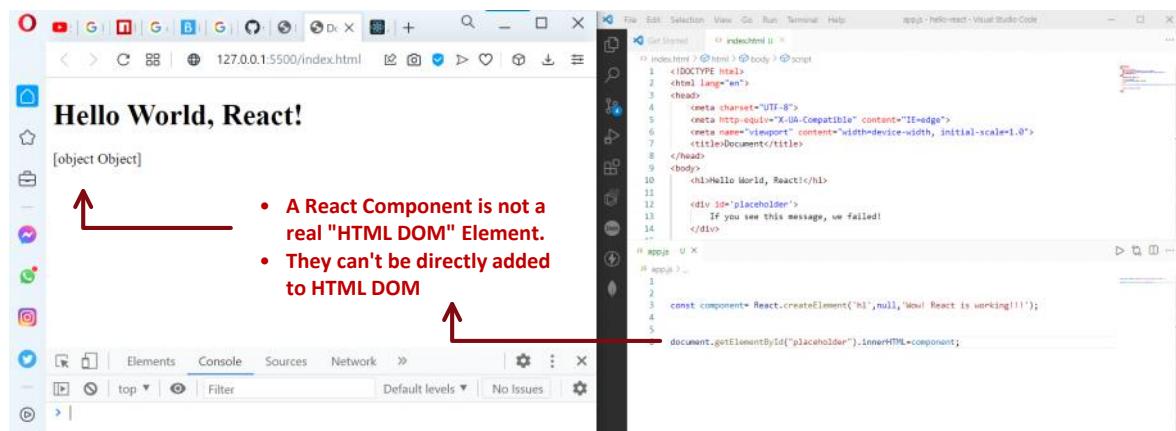
- Common Components like Header and footer may not be rendered again.
 - No unnecessary data redownloaded.
- Application is in control even when we move from one screen to another
- If there is an error in navigation our application can present whatever it wants to display
- User will have a better experience.
- Better information sharing
 - We are not going to different but just a different component sitting on the same page.

1min React

Tuesday, January 11, 2022 3:34 PM

<https://reactjs.org/docs/add-react-to-a-website.html>

```
<script src="https://unpkg.com/react@17/umd/react.development.js" crossorigin>
</script>
<script src="https://unpkg.com/react-dom@17/umd/react-dom.development.js"
crossorigin></script>
```

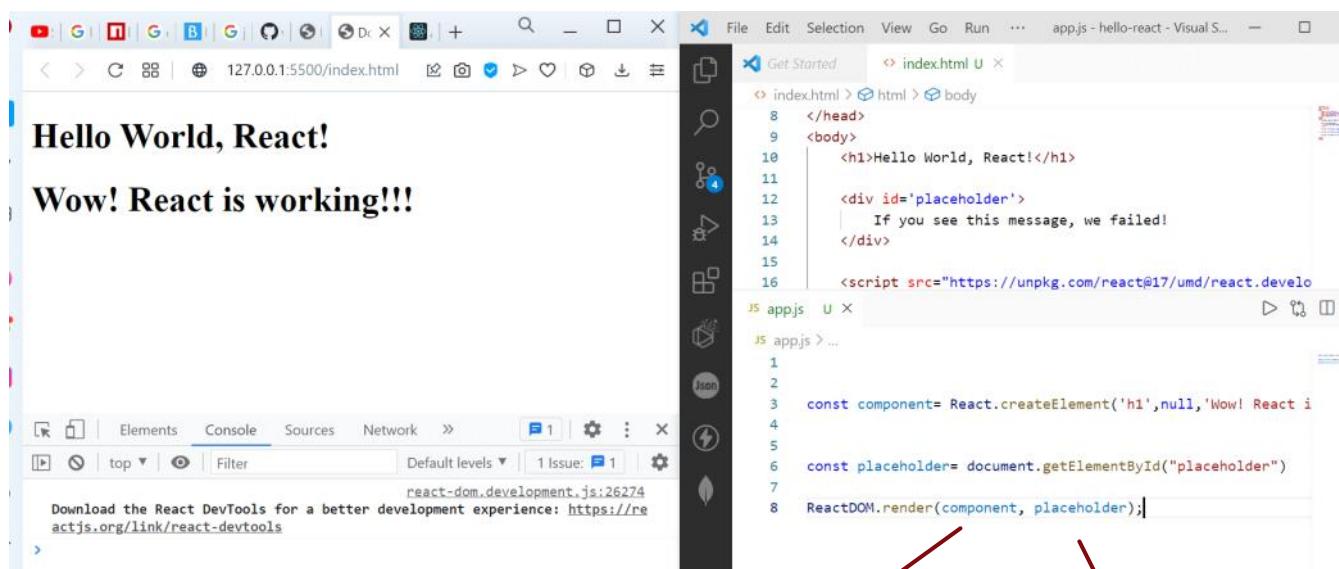


- A component is created
- It can't be displayed on UI directly
- We need someone who knows how to convert a React Object into an HTML DOM element

React DOM

- It is a library that can render a React Element (React Element Tree) into HTML DOM
- React Components are part of a virtual (in-memory) DOM objects
 - They are not real HTML Elements
- The job of translation is done by a separate library
- This library checks and updates only those HTML elements which have changed since the last update

Include ReactDOM on our page



```
react-dom.development.js:26274
Download the React DevTools for a better development experience: https://reactjs.org/link/react-devtools
>
```



```
7
8 ReactDOM.render(component, placeholder);
```

IMPORTANT!

- We use this library only once in our application
- Just a render() to render root element of our element

A React Element

DOM element where React Element shall we rendered

React + React DOM

Tuesday, January 11, 2022 4:10 PM

React

- A component creation library
- Creates a virtual component
 - NOT HTML
- Manages a virtual DOM tree
- Manages component life cycle

React DOM

- Translates React Elements to HTML Elements
- Compares React Virtual to HTML DOM
 - Updates portion of HTML DOM based on changes in ReactDOM

ReactNative

- Another library that can translate a ReactComponent to a Mobile UI
- Can help us develop mobile applications for
 - Android
 - IOS
- We use same 'React' Library
- We use 'react-native' instead of 'react-dom'

React Component Examples

Tuesday, January 11, 2022 4:15 PM

HTML	React
 	▶ const break= React.createElement("br")

HTML	React
<h1> Hello World </h1>	const header= React.createElement("h1", null, "Hello World")

- Null indicates no attributes added to the element
- Last item is enclosed child item

Elements with attributes and children

HTML	React
 Home </h1>	const link= React.createElement("a", {href:"/"}, "Hello World")

Important Consideration

The screenshot shows a browser window displaying a "Hello World, React!" application. Below the browser is the Chrome DevTools interface. In the DevTools' Elements tab, there is a warning message: "Warning: Invalid DOM property 'class'. Did you mean 'className'? at a". A red box highlights this warning. Another red box highlights a portion of the React component code where the word "class" is used as an attribute name. To the right of the screenshot, a section titled "Style" contains a list of rules:

- A react component doesn't use "string" style
- Style itself is javascript object
- We can use hyphen in style properties
 - Hyphen is not allowed in javascript variable names
 - We use camel convention by changing the second word start to upper case
- Values of style must be quoted
 - In css file none is not quoted
 - None is not known to javascript

We can't use "class" as attribute name

- class is a keyword in javascript
- Shouldn't be used as variable name
- React offers an alternative "className"
- Using class is a warning
 - It still works

HTML vs React Component

The diagram compares the same navigation bar structure in both HTML and React. It highlights differences in styling, variable naming, and class usage.

HTML:

```
<a href="/" style="text-decoration:none; color:white" class="nav"> Home </a>
```

Note: variables can't have hyphen. So we use camel convention

React:

```
const link= React.createElement("h1", { href:"/", style:{ color:"white", textDecoration:"none" }, className:"nav" }, "Home")
```

Note: style is a nested java script object

Note: class is a keyword, so react uses className instead

Nested React Element

The diagram compares a header component with nested children in both HTML and React. It highlights the structure of nested elements and their corresponding React.createElement calls.

HTML:

```
<div class="header">
  <h1 >ReactWeb</h1>
  <p class="slogan">Welcome to React</p>
</div>
```

Note: Nested Child

React:

```
const header= React.createElement("div", { className:"header" },
  React.createElement("h1",null, "ReactWeb"),
  React.createElement("p",{
    className:"slogan"
  },
  "Welcome to React"
));
```

Transpilation

Tuesday, January 11, 2022 4:54 PM

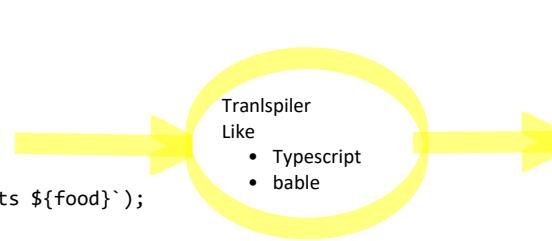
- Transpilation → Translation + Compilation
- A code written in one language (or standard) is translated to another language (or standard)
- The source code may not be well understood by the execution environment
- We use a tool that can convert my code to a well known standard
- The translated with work of Browser

Use Case

1. ES2015 to ES5 translation

- We may write a code following new features of ES2015
- This code can be translated to ES5 keywords and features
- As a developer we can use modern features
- Even older client can use newer features

```
//ES2015
class Person{
    constructor(name){
        this.name=name;
    }
    eat(food){
        console.log(` ${this.name} eats ${food}`);
    }
}
const person=new Person('Vivek');
person.eat("Lunch");
```



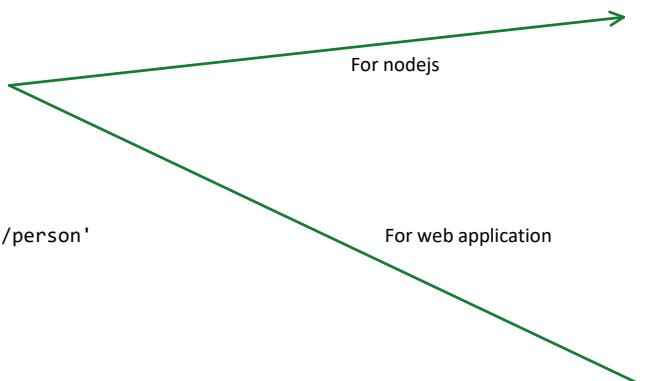
```
//ES5
var Person=function(){
    var Person=function name(){
        this.name=name;
    }
    Person.prototype.eat=function(food){
        console.log(this.name+" eats "+this.food);
    }
    return Person;
}
var person=new Person('Vivek');
person.eat("Lunch");
```

Use Case 2 — Adding features that are not supported in any browser/runtime

- Typescript and bable can offer us features not supported by any browser
 - Example
 - Import/export

```
//person.js
export class Person{
}

//client.js
import {Person} from './person'
...
```



```
//person.js
class Person{...}
module.exports={
    Person
}

//client.js
const {Person}=require('./person');
```

Translated code involving additional library like require.js



Translated code involving additional library like require.js

Use Case 3 — Adding features not even planned for any version of ES

- Strong type checking
- Interfaces
- Generics

Typescript

- Can add new languages features
- You write typescript code
- It gets converted to javascript code

Babel

- Doesn't support this

Use Case 4 — Adding JSX

- JSX is Javascript XML
- A syntax that looks similar to HTML
- Directly written inside Javascript
- Translated to React.createElement
- Supported by
 - Babel
 - Typescript

```
const component=<h1 className="heading">Hello World</h1>;
```

```
const component=React.createElement("h1",  
  {className:"heading"},  
  "Hello World");
```

Note

- Html like code here is NOT HTML. It is JSX
 - We are using className not class
- It is not a String
 - We have not wrapped it in quotes.

You can't run your own code in browser

- Browser doesn't know Javascript
- It must be translated to normal javascript before sending server
- We can run babel in watch mode which can translate my code to javascript on the fly

JSX => React Component => HTML

React.createElement

```
const header= React
    .createElement("div",
      {
        className:"header",
        title: "header"
      },
      "Home"
    )
  
```

By ReactDOM

JSX

```
<div className="header" title="header">
  Home
</div>
```

By Babel/typescript

HTML

```
<div class="header" title="header" >
  Home
</div>
```

JSX => React Component => HTML

React.createElement

```
const myStyle={ 
  backgroundColor: "blue",
  height: window.innerHeight/2;
};

const header= React
  .createElement("div",
    {
      style: myStyle
    },
    "Home"
  )
```

JSX

```
<div style={myStyle} >
  Home
</div>
```

Javascript
expression
injected in JSX
using {}

HTML

```
<div >
  Home
</div>
```

Shall be inserted
using Javascript
at runtime.
**No fragment
here**

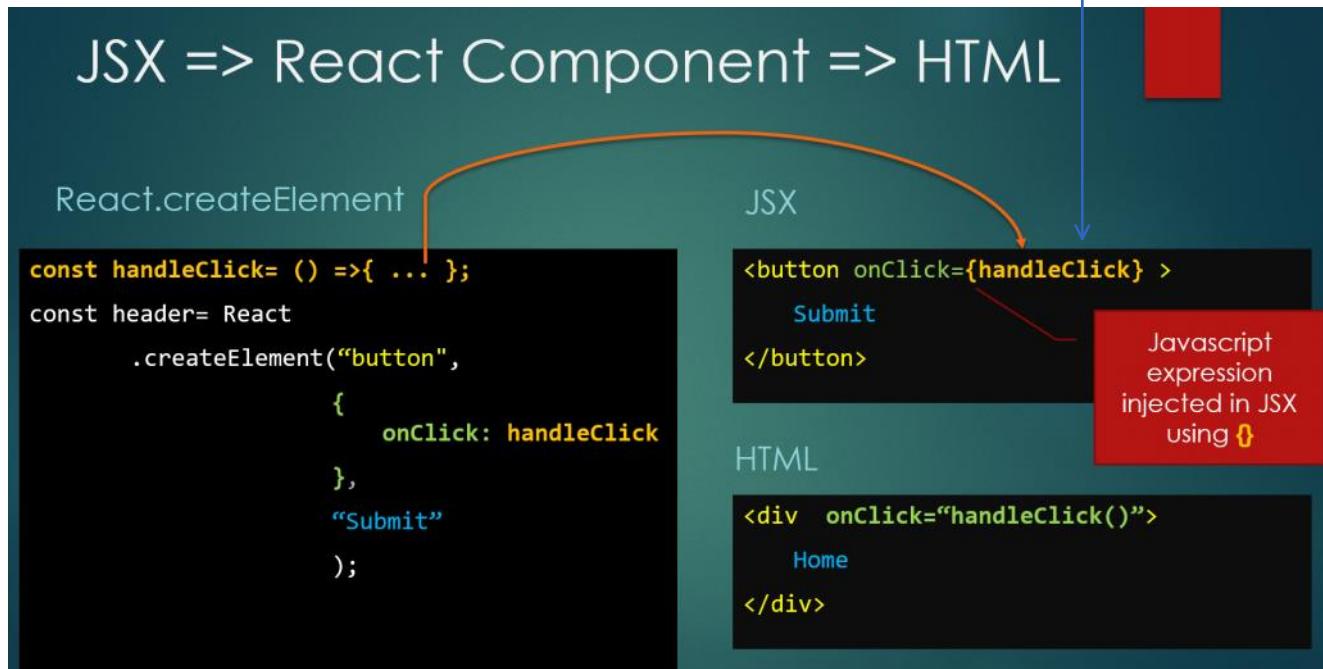
Interpolation

- Javascript expression injected into JSX by wrapping them in {}

Ineterpolation

- Javascript expression injected into JSX by wrapping them in {}
- You can add
 - Expression
 - Variable
 - Call function
- You can't write other statements like
 - Loop
 - if

Handling the events



A typical JSX to bable code

A Typical jsx file

JSX is internally React.createElement
We should include import

```
import React from "react";
const myStyle={
  backgroundColor: "blue",
};
const handleClick=()=>alert("hello");

const submit=<button
  style={myStyle}
  onClick={handleClick}
>Submit</button>
```

```
import React from "react";
const myStyle={
  backgroundColor: "blue",
};
const handleClick=()=>alert("hello");

const submit = React.createElement(
  "button",
  {
    style: myStyle,
    onClick: handleClick
  },
  "Submit"
);
```

Babel/TypeScript

IMPORT!

- Since a JSX is essentially React.createElement, we must import React in our file
- This has become optional latest react application template
- It is still compulsory in ReactNative projects
- Better to use it

Starting with React

Tuesday, January 11, 2022 5:26 PM

What do we Need?

- ▶ Node JS
 - ▶ Entire eco system of React (and most **JavaScript** based projects) is based on NodeJS echo system
 - ▶ Needed to install components related to React
 - ▶ Hooks into entire process
 - ▶ Install
 - ▶ Development run
 - ▶ Build/deploy
- ▶ VS Code (or any text editor)
 - ▶ Provides and IDE to develop the application

IMPORTANT!

- We don't need NODEJS, We need NPM
 - NPM is part of NODEJ
- We will not be creating a web server here
- Even front end app needs many tools like
 - JEST
 - Typescript/babel
- We need to convert and pack our enhanced Javascript codes into browser understandable codes
- We must conver import/exprort to something that browser will understand

Create React App

- We have a npm package called "create-react-app"
- This package can help us create a react project quickly
- We install this package globally so that we don't have to install everytime
- You may also use it without explicit install using npx command.

Install

- Install it globa

```
npm install --global create-react-app
```

React Component

Tuesday, January 11, 2022 6:04 PM

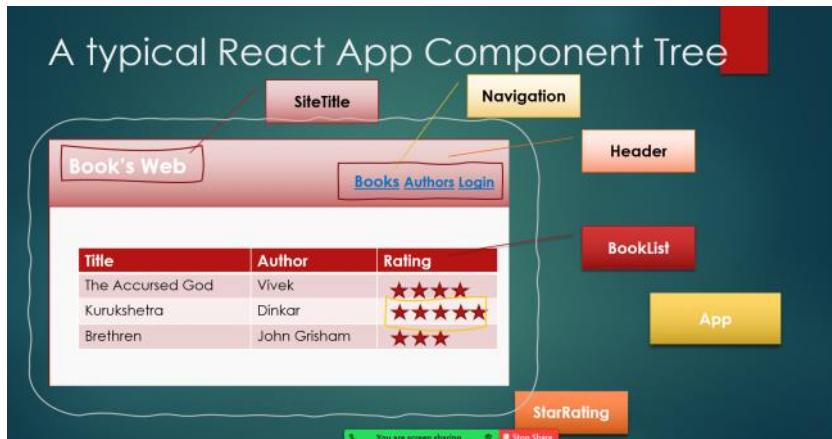
```
2 import ReactDOM from 'react-dom';
3
4 //const component=React.createElement('h1',null,'Hello World');
5 const headerStyle={
6   color:"blue"
7 }
8
9 const component=<div style={headerStyle}>
10   <h1>Hello World</h1>
11   <p>Welcome to React</p>
12   <p>2+2 = {2+2}</p>
13   <p>Date is {new Date().toLocaleDateString()}</p>
14   <p>Time is {new Date().toLocaleTimeString()}</p>
15   <p>{for(let i=0;i<5;i++)}</p>
16 </div>
17
18
19 const root=document.getElementById("root");
20
21 //root.innerHTML=component;
22
23 ReactDOM.render(component,root);
```

Interpolation

- Allows
 - Normal variables
 - Expression
 - Functions calls
- Doesn't Allows
 - Statements like
 - If
 - For
 - Creating a function
 - You are allowed to create arrow function

A Typical React App

Wednesday, January 12, 2022 10:14 AM



IMPORTANT!

- A React app consists of multiple components
- A React app itself is a component
- In a React
 - A component would be made up of many smaller components
 - Each smaller component may be made up of more components
- A component may be reusable

A Component Tree



Building complex UI

Wednesday, January 12, 2022 10:33 AM

```
//const component=React.createElement('h1',null,'Hello World');
const titleStyle = {
  color: "blue"
}

const title = <h1 style={titleStyle}>Tic Tac Toe</h1>;
const header = <div><title></div>;
const scoreBoard = <p>score board</p>

const game = <div>
  <h2>Game</h2>
  {scoreBoard}
</div>

const appStyle = {

};

const app = <div style={appStyle}>
  {header}
  {game}
</div>
```

- A JSX is internally a React Object created in JavaScript using `React.createElement()`

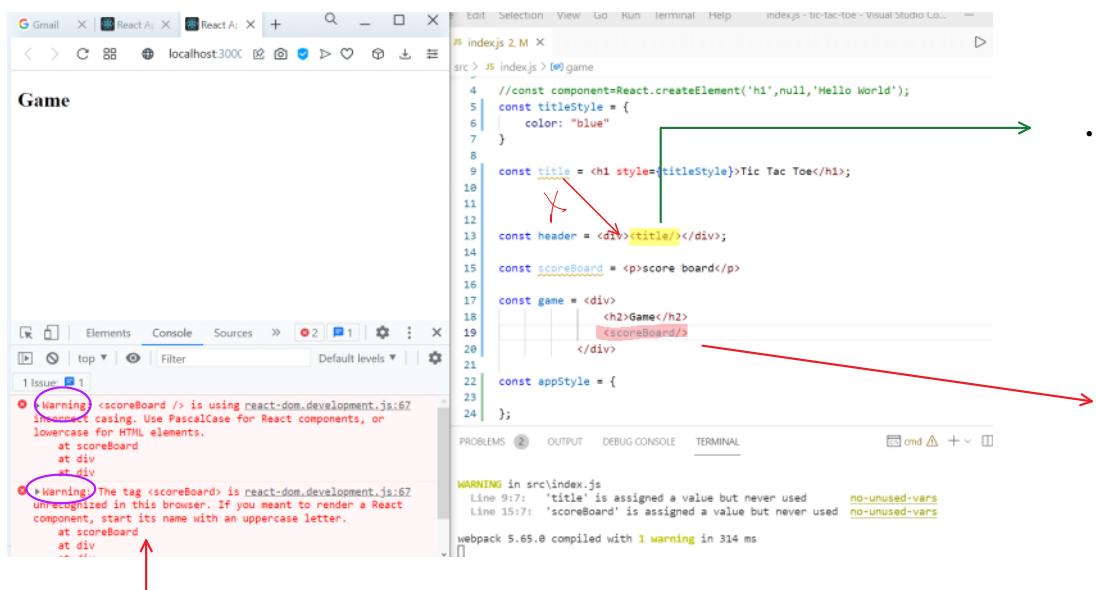
- We can inject any javascript into JSX using interpolation syntax

Warning!

- An interpolation syntax is NOT A COMPONENT
- A Component looks like an html tag
- Interpolation
 - `{title}`
- Component
 - `<title></title>`

Using JSX as a Tag

- React treats any tag that is lower case or starts with lower case a builtin html component
- A React custom component must follow pascal convention
 - Start with upper case
 - Each word join upper case
 - Examples
 - `game` -> Game
 - `scoreBoard` -> ScoreBoard
- You may get warning for unrecognized component in developer console
- Application will not crash.



- Title we are using here is standard HTML title
 - Actually React provides a component mapping for every HTML builtin elements
 - They are all lower case
 - You can use in react
 - `<H1>Hello World</H1>`
 - It works in plain HTML
- Here React treats lower cased `<title>` as well-known html title tag
 - It ignores our title variable
- "scoreBoard" doesn't start with upper, so React doesn't consider it as a Custom Component and Ignores it
 - You get no error or crash
- React knows that "scoreBoard" is not built-in HTML component, so it is issuing a friend warning.

Takeaway

1. Always name your custom component in pascal convention
 - Start with upper case letter
 - If there is a word join that should be upper case letter
 2. Always refer standard html elements in lower case only
 - Browser doesn't care if you use upper case or lower case
 - React always uses lower case
 3. Always close your tags with JSX
 - JSX is XML
- We are not writing HTML
 - We are writing XML

- Xml needs tag closing
- If there is nothing inside do a self close
- Examples
 - `<p>hello world` → `<p>hello world</p>`
 - `<hr>` → `<hr/>`
 - `<input type='text'` → `<input type="text" />`

React Component Basics

Wednesday, January 12, 2022 10:57 AM

A Plain JSX is not a Component Type

- It is a component instance
 - One object of unknown type
- <Title/>
 - It tries to create a new instance of "Title" type

```
JS index.js M ...
src > JS index.js > ...
5 |   const titleStyle = {
6 |     color: "blue"
7 |
8 |
9 |   const Title = <h1 style={titleStyle}>Tic Tac Toe</h1>;
10 |
11 |   const header = <div><Title/></div>;
12 |
13 |   const ScoreBoard = <p>score board</p>
14 |
15 |   const game = <div>
16 |     <h2>Game</h2>
17 |     KScoreBoard/>
18 |   </div>
19 |
20 |   const appstyle = {
21 |
22 |   };
23 |
24 |   const app = <div style={appstyle}>
25 |     {header}
26 |     {game}
27 |   </div>
28 |
29 |   const root = document.getElementById("root");
30 | 
```

- Element type (tag) must be either
 - String
 - For standard html elements
 - React Component Type
 - Function
 - Class

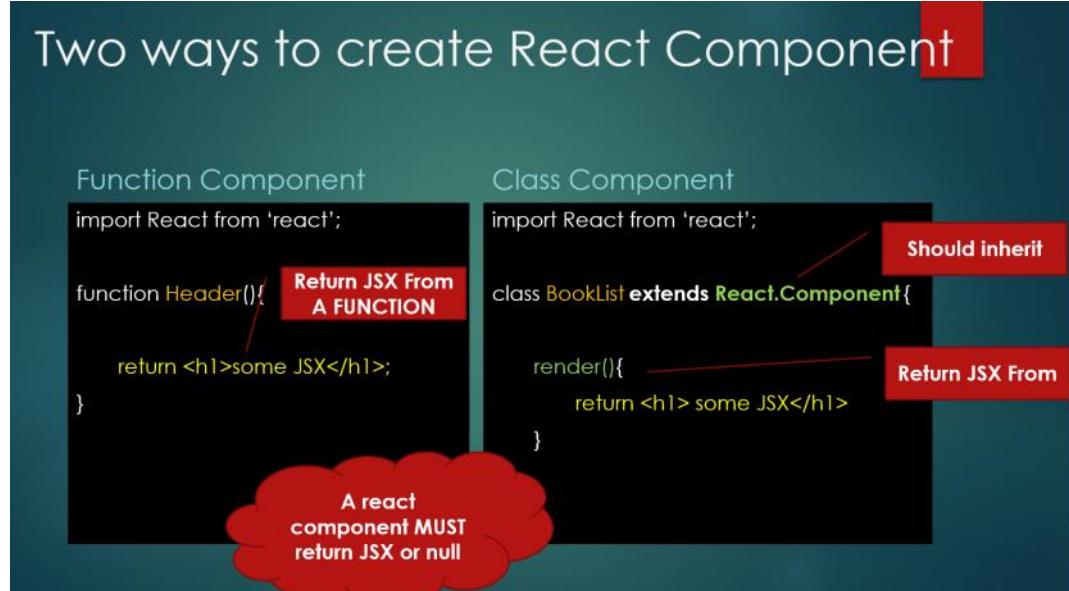
Takeway

- A tag is an object creator not an object
- It can be a
 - Class
 - That has a render method
 - Returns JSX
 - Function
 - That returns JSX

Why a Custom Component Must be a class/function

- A Custom component isn't just a bunch UI Tags
- It should also have Styles
- It should also have its own logic
 - To get the required data
 - Display the required data
 - Handler User interaction
- JSX can only replace HTML part
- JSX can't replace Javascript Logic

Two ways to create React Component



Simplest JSX Component

```
const Title = ()=> <h1 style={titleStyle}>Tic Tac Toe</h1>;
```

A blue bracket underlines the entire code block. A blue arrow points to the left from the start of the code, with the text "• A lambda function
○ Return from the lambda".

Small Problem

- In a function/class component JSX must start on same line as `return`
- If we start on a different it babel fails to connect the dots

```
class App extends React.Component {

  render(){
    return <div>
      <Header/>
      <Game/>
    </div>
  }
}
```

A red arrow points from the text "If we start on a different it babel fails to connect the dots" to the line `return <div>`.

Solution

- We may wrap JSX in parentheses
- Start parentheses on same line as `return`
- Write rest of the JSX whatever way you like

```
return (
  <div>
    <Header/>
    <Game/>
  </div>
);
```

Passing Information

Wednesday, January 12, 2022 11:41 AM

- We often need to pass information from a component to its child
- Who will decide the title of the application?
 - TitleComponent?
 - HeaderComponent?
 - AppComponent?

What happens when we pass an attribute to our component?

```
const Title =()=> <h1 style={titleStyle}>Dummy Title</h1>;  
  
function Header(){  
  return <div><Title color="red" text="Tic Tac Toe" /></div>;  
}
```



A typical JSX Component with input properties

```
const Title =(arg)=>{  
  //write your logic before returning JSX  
  console.log('arg',arg);  
  const titleStyle={  
    color: arg.color || "black"  
  };  
  
  return <h1 style={titleStyle}>{arg.text}</h1>;  
}  
  
function Header(){  
  return <div><Title color='brown' text="Tic Tac Toe" /></div>;  
}
```



Component with Children

- A component can run its own logic before returning the JSX
- JSX can include data supplied via attribute or children

```

const Title =(arg)=>{
  //write your logic before returning JSX
  console.log('arg',arg);
  const titleStyle={
    color: arg.color || "black"
  };

  return <h1 style={titleStyle}>{arg.children}</h1>;
}

function Header(){
  return <div><Title color='brown'>Tic Tac Toe</Title></div>;
}

```

General Convention

- The argument we pass to our component we call it **"props"**

Design Limitation

```

const SiteTitle= (props)=> <h1>{props.children}</h1> ;

const Header= (props)=> <div>
  <SiteTitle>
    Book's Web
  </SiteTitle>
  <Navigation/>
</div>;

const App= (props)=><div>
  <Header>
    <BookList/>
  </Header>
</div>;

```

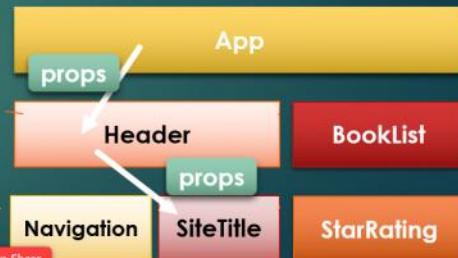
Option#3 App

- Can't Pass props to Grand Child "Site Title"
- Can only pass props to direct children

How to pass props from grand parent to grand child

```
const SiteTitle= (props)=><h1>{props.children}</h1> ;  
  
const Header= (props)=> <div>  
    <SiteTitle>  
        {props.title}  
    </SiteTitle>  
    <Navigation/>  
</div>;  
  
const App= (props)=><div>  
    <Header title="Book's Web"/>  
    <BookList/>  
</div>;
```

1. Pass title from App to Header as props
2. Header should pass it down to Site Title as Props



How to pass props to class based component

- In class based component 'prop' is passed to the constructor
 - The constructor must pass the 'prop' to super
 - Props must be accessed using 'this.props'
-
- Not constructor is optional if
 - All it does is to pass props to super class
 - Define constructor only if
 - Constructor is doing some other initialization also
 - We will discuss this later.

```
class Header extends React.Component {  
    constructor(props){  
        super(props)  
    }  
    render(){  
        return <div><Title color='brown'>{this.props.title}</Title></div>;  
    }  
}
```

Organizing Our Code

Wednesday, January 12, 2022 12:13 PM

- We generally write all our components in separate JS files
- All components (except App) are generally placed in a component folder inside src
 - App is generally placed directly in the source
- We use import /export to get the details

- **Folder structure**

- We can import a css file
 - Now we can use this through our all our child elements
- We can also import css file in particular components.

The screenshot shows a code editor with an Explorer sidebar on the left and a main editor area on the right. The Explorer sidebar displays a file tree for a project named 'TIC-TAC-TOE'. The 'src' directory contains 'components' (with 'AppHeader.js', 'Game.js', and '#App.css'), 'App.js', and several 'index' files ('index.js', 'indexV0.js', 'indexV1.js'). The '.gitignore' file is also listed. The main editor area shows a file named 'index.js' with the following content:

```
import React from 'react';
import ReactDOM from 'react-dom';
import './App.css';
import App from './App';

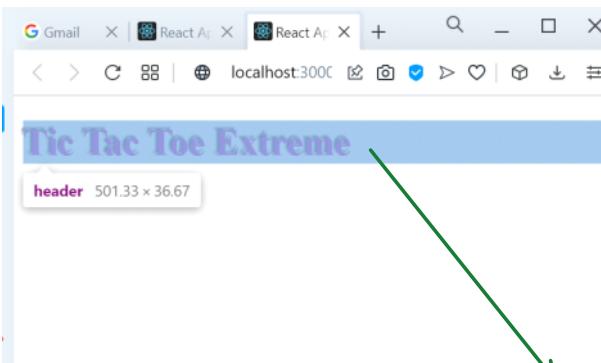
ReactDOM.render(<App/>, document.getElementById('root'));
```

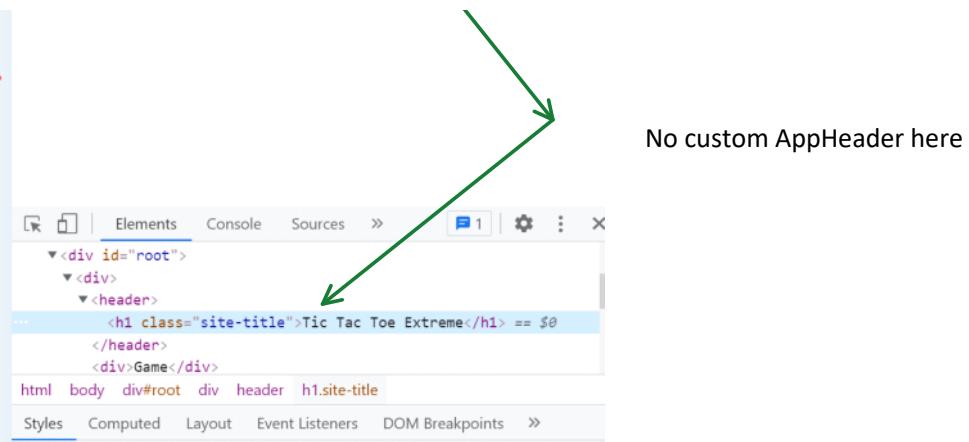
Why can't we write specific CSS for our custom component

Example

```
SiteHeader{
    background-color:gray;
}
```

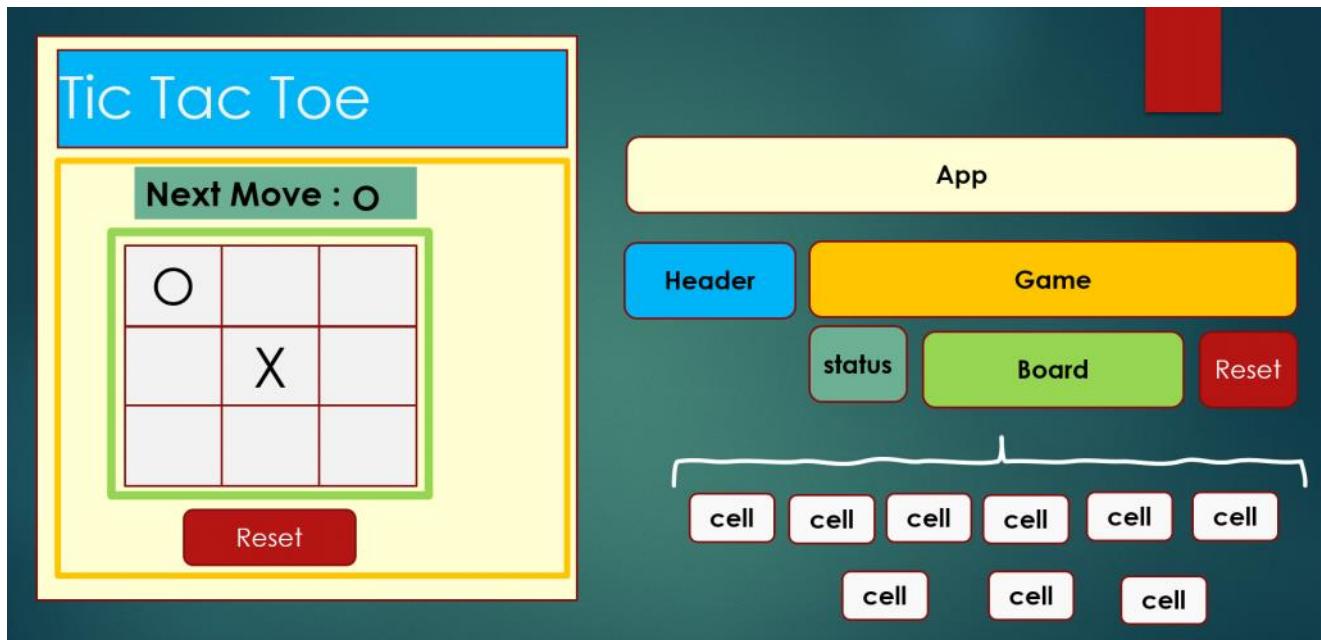
- ReactDOM converts our custom components to plain HTML it is rendering
- There no actual custom component in our DOM Tree





Tic Tac Toe Game

Wednesday, January 12, 2022 12:33 PM



Handling User Interaction

Wednesday, January 12, 2022 12:59 PM

Event Handler

- We can create our own event handlers
 - They can be any function
 - Generally we use a naming convention
 - To handle event onXYZ we write function handleXYZ
 - If there are many buttons we can name based purpose
 - <button onClick={handleReset}>reset</button>
 - <button onClick={handleSave}>save</button>

```
const Cell=(props)=>{
  let value=props.value;
  const handleClick=>{
    console.log('cell clicked');
    value="0";
    console.log('value changed to ',value);
  }
  return (
    <button className='cell' onClick={handleClick}>
      {value}
    </button>
  );
}
```

handles button click

Updating UI based on User Interaction

- Consider the below situation

```
1. Button click is handled successfully  
2. The value has changed  
3. But UI is not updated
```

```
cell clicked  
value changed to 0
```

```
const Cell=(props)=>{
  let value=props.value;
  const handleClick=>{
    console.log('cell clicked');
    value="0";
    console.log('value changed to ',value);
  }
  return (
    <button className='cell' onClick={handleClick}>
      {value}
    </button>
  );
}
```

The diagram illustrates a Tic Tac Toe application interface and its corresponding React component code. The UI shows a title 'Tic Tac Toe', a message 'Next Move : O', a 3x3 grid with empty cells, and a 'Reset' button. A red arrow points from the text 'UI Not Changed' to the grid. A callout bubble says 'Cell design'. A red cloud bubble says 'React doesn't know that 'value' changed'. The code defines a 'Cell' component with a state variable 'value' initialized to '_'. It has a 'handleCellClick' function that changes 'value' to 'O' and logs the change. It returns a button element with the current 'value' prop.

```
function Cell () {
  let value = "_";

  function handleCellClick(){
    value = "O";
    console.log("value changed to ",value);
  }
  return <button onClick={handleCellClick}>{value}</button>;
}
```

Value changed to O

- UI Not Changed
- Cell design
- React doesn't know that 'value' changed
- Event called.
- Value Changed

Assignment Day 13.1

Wednesday, January 12, 2022 1:06 PM

- Create Component with dummy text and **proper design** to represent
- A score board
 - Data can be passed to the score board
 - Total games
 - Wins
 - Draws
- Update the UI to make it look better
- Write a logic to find out how we decide if a game is won?
 - Take an array of 9 cells
 - Tell if current value means
 - Win
 - Game over without a win
 - All cells are used and no winner

Props design

Thursday, January 13, 2022 10:32 AM

```
JS index.js      JS AppHeader.js    # App.css      JS Board.js  X  JS Game.js
src > components > JS Board.js > [e] Board
  8  return (
  9    <div className='board'>
10      <div className='cells'>
11        <Cell value=" " />
12        <Cell value=" " />
13        <Cell value=" " />
14
15        <Cell value=" " />
16        <Cell value=" " />
17        <Cell value=" " />
JS Cell.js  X
src > components > JS Cell.js > [e] Cell
  1  import React from 'react';
  2
  3  const Cell=(props)=>[
  4
  5    let value=props.value;
  6
  7    const handleClick=()=>{
  8      console.log('cell clicked');
  9      value= value==="0"? "X": "O";
10      console.log('value changed to ',value);
```

- A component can't modify 'props' passed to it.
 - It is considered immutable
- Changing the props or local variables **will not update the UI**
- UI is updated when React knows associated value has changed
 - React Doesn't local value has changed
 - React knows props can't change
- Parent who is passing the prop can change it.
- When passed 'prop' is changed at the parent, React will re-render the component with the updated value.

Event Handling using class method

Thursday, January 13, 2022 11:22 AM

- By default a class method is unbound
- If we try to handle events using a class method then we get a problem of missing 'this'

```
class Cell extends React.Component {
  constructor(props) {
    super(props);
    this.value=props.value;
  }
  handleClick(){
    console.log('cell clicked',this.props.value);
    this.value=this.value==='X'? 'O': 'X';
    console.log('value changed to ',this.value);
  }
  render(props){
    return (
      <button className='cell' onClick={this.handleClick}>
        {this.value}
      </button>
    );
  }
}
```

For detailed discussion refer to [BoundUnBoundMethod](#) in Javascript 03

- Unbound method when used as call back will lose this context

Solution

There are 4 solution to this problem

1. Don't use class method

- Write a method directly in render().
 - Same way as you do for function component
- Because of closure it will always get this and any other value available
- **Disadvantage**
 - Your render will be too large and complex.

2. Use bind syntax to bind the method

- Use ES5+ bind method syntax that will bind the method's this context
- **Disadvantage**
 - We need to write bind for every event handler

3. Call the unbound event handler method using an arrow function

- Arrow functions don't have its own 'this'
- It uses this from its closure
 - Closure remembers everything.
- Now we don't need to write bind
- **Disadvantage**
 - We create an additional function to call my function

```
class Cell extends React.Component {
  constructor(props){
    super(props);
    this.value=props.value;
    this.handleClick = this.handleClick.bind(this);
  }
  handleClick(){
    console.log('cell clicked',this.props.value);
    this.value=this.value==='X'? 'O': 'X';
    console.log('value changed to ',this.value);
  }
  render(props){
    return (
      <button className='cell' onClick={this.handleClick}>
        {this.value}
      </button>
    );
  }
}
```

```
class Cell extends React.Component {
  constructor(props){
    super(props);
    this.value=props.value;
    //this.handleClick = this.handleClick.bind(this);
  }
}
```

24. Note we are not passing this.handleClick to event
- We are passing an arrow function that calls my class method

```

6    super(props);
7    this.value=this.props.value;
8    //this.handleClick = this.handleClick.bind(this);
9 }
10
11 handleClick(){
12   console.log('cell clicked',this.props.value);
13   this.value='X'?'O':'X';
14   console.log('value changed to ',this.value);
15 }
16
17 }
18
19 render(props){
20
21   return (
22     <button className='cell' onClick={()>this.handleClick()}>
23       {this.value}
24     </button>
25   );
26 }
27
28 }
29
30

```

4. ES2016 class member syntax

- ES2016 allows us to create members directly in the class
- Method takes the form of an arrow function
- **Disadvantage**
 - This syntax is still not supported on most browser and NodeJS environments
 - NodeJS recently introduced it
 - This is an experimental syntax
- **How do we use this feature?**
 - Babel supports it.

24. Note we are not passing this.handleClick to event

- We are passing an arrow function that calls my class method
- Arrow functions don't have their own 'this'
 - So the function will refer to 'this' of its closure "render"

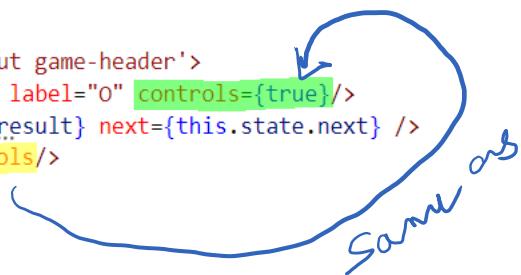
Assorted React Concepts

Saturday, January 15, 2022 10:52 AM

JSX Attribute without value

A JSX Attribute without value means the value supplied is true

```
<div className='column-layout game-header'>
<StopWatch fraction={false} label="0" controls={true}/>
<Status result={this.state.result} next={this.state.next} />
<StopWatch label="X" controls/>
```

A handwritten note "Same as" is written in blue ink, with a blue arrow pointing from the word "controls" in the first StopWatch component to the word "controls" in the third StopWatch component.

State

Thursday, January 13, 2022 11:57 AM

- A State is a mutable data member
- By default available only in class based component
- Changes to state (through proper channel) can update the UI

A screenshot of the Visual Studio Code interface. The code editor shows a file named Cell.js. A tooltip is displayed over the line of code `this.state.value = this.state.value === 'X' ? 'O' : 'X';` with the message "Do not mutate state directly. Use setState(). react/no-direct-mutation-state". A red arrow points from this tooltip towards the explanatory text below.

```
src > components > JS Cell.js > Cell > constructor
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
288
289
289
290
291
292
293
294
295
296
297
298
299
299
300
301
302
303
304
305
306
307
308
309
309
310
311
312
313
314
315
316
317
318
319
319
320
321
322
323
324
325
326
327
327
328
329
329
330
331
332
333
334
335
336
337
338
339
339
340
341
342
343
344
345
346
347
348
349
349
350
351
352
353
354
355
356
357
358
359
359
360
361
362
363
364
365
366
367
368
369
369
370
371
372
373
374
375
376
377
378
379
379
380
381
382
383
384
385
386
387
388
389
389
390
391
392
393
394
395
396
397
398
399
399
400
401
402
403
404
405
406
407
408
409
409
410
411
412
413
414
415
416
417
418
419
419
420
421
422
423
424
425
426
427
428
429
429
430
431
432
433
434
435
436
437
438
439
439
440
441
442
443
444
445
446
447
448
449
449
450
451
452
453
454
455
456
457
458
459
459
460
461
462
463
464
465
466
467
468
469
469
470
471
472
473
474
475
476
477
478
479
479
480
481
482
483
484
485
486
487
488
489
489
490
491
492
493
494
495
496
497
498
499
499
500
501
502
503
504
505
506
507
508
509
509
510
511
512
513
514
515
516
517
518
519
519
520
521
522
523
524
525
526
527
528
529
529
530
531
532
533
534
535
536
537
538
539
539
540
541
542
543
544
545
546
547
548
549
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
687
688
689
689
690
691
692
693
694
695
696
697
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
778
779
779
780
781
782
783
784
785
786
787
787
788
789
789
790
791
792
793
794
795
796
797
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
878
879
879
880
881
882
883
884
885
886
887
887
888
889
889
890
891
892
893
894
895
896
896
897
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
917
918
918
919
919
920
921
922
923
924
925
926
927
927
928
928
929
929
930
931
932
933
934
935
936
937
937
938
938
939
939
940
941
942
943
944
945
946
946
947
947
948
948
949
949
950
951
952
953
954
955
956
957
957
958
958
959
959
960
961
962
963
964
965
966
966
967
967
968
968
969
969
970
971
972
973
974
975
976
977
977
978
978
979
979
980
981
982
983
984
985
986
986
987
987
988
988
989
989
990
991
992
993
994
995
996
997
998
999
999
1000
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1017
1018
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1027
1028
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1037
1038
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1051
1052
1053
1054
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1071
1072
1073
1074
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1081
1082
1083
1084
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1091
1092
1093
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1101
1102
1103
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1111
1112
1113
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1121
1122
1123
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1131
1132
1133
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1141
1142
1143
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1151
1152
1153
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1161
1162
1163
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1171
1172
1173
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1181
1182
1183
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1191
1192
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1201
1202
1203
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1211
1212
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1221
1222
1223
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1231
1232
1233
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1241
1242
1243
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1251
1252
1253
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1261
1262
1263
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1271
1272
1273
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1281
1282
1283
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1291
1292
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1301
1302
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1311
1312
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1321
1322
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1331
1332
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1341
1342
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1351
1352
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1361
1362
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1371
1372
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1381
1382
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1391
1392
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1401
1402
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1411
1412
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1421
1422
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1431
1432
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1441
1442
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1451
1452
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1461
1462
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1471
1472
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1481
1482
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1491
1492
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1501
1502
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1511
1512
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1521
1522
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1531
1532
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1541
1542
1543
1544
1544
1545
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1551
1552
1553
1554
1554
1555
1555
1556
1556
1557
1557
1558
1558
1559
1559
1560
1561
1562
1563
1564
1564
1565
1565
1566
1566
1567
1567
1568
1568
1569
1569
1570
1571
1572
1573
1574
1574
1575
1575
1576
1576
1577
1577
1578
1578
1579
1579
1580
1581
1582
1583
1584
1584
1585
1585
1586
1586
1587
1587
1588
1588
1589
1589
1590
1591
1592
1593
1594
1594
1595
1595
1596
1596
1597
1597
1598
1598
1599
1599
1600
1601
1602
1603
1604
1604
1605
1605
1606
1606
1607
1607
1608
1608
1609
1609
1610
1611
1612
1613
1614
1614
1615
1615
1616
1616
1617
1617
1618
1618
1619
1619
1620
1621
1622
1623
1624
1624
1625
1625
1626
1626
1627
1627
1628
1628
1629
1629
1630
1631
1632
1633
1634
1634
1635
1635
1636
1636
1637
1637
1638
1638
1639
1639
1640
1641
1642
1643
1644
1644
1645
1645
1646
1646
1647
1647
1648
1648
1649
1649
1650
1651
1652
1653
1654
1654
1655
1655
1656
1656
1657
1657
1658
1658
1659
1659
1660
1661
1662
1663
1664
1664
1665
1665
1666
1666
1667
1667
1668
1668
1669
1669
1670
1671
1672
1673
1674
1674
1675
1675
1676
1676
1677
1677
1678
1678
1679
1679
1680
1681
1682
1683
1684
1684
1685
1685
1686
1686
1687
1687
1688
1688
1689
1689
1690
1691
1692
1693
1694
1694
1695
1695
1696
1696
1697
1697
1698
1698
1699
1699
1700
1701
1702
1703
1704
1704
1705
1705
1706
1706
1707
1707
1708
1708
1709
1709
1710
1711
1712
1713
1714
1714
1715
1715
1716
1716
1717
1717
1718
1718
1719
1719
1720
1721
1722
1723
1724
1724
1725
1725
1726
1726
1727
1727
1728
1728
1729
1729
1730
1731
1732
1733
1734
1734
1735
1735
1736
1736
1737
1737
1738
1738
1739
1739
1740
1741
1742
1743
1744
1744
1745
1745
1746
1746
1747
1747
1748
1748
1749
1749
1750
1751
1752
1753
1754
1754
1755
1755
1756
1756
1757
1757
1758
1758
1759
1759
1760
1761
1762
1763
1764
1764
1765
1765
1766
1766
1767
1767
1768
1768
1769
1769
1770
1771
1772
1773
1774
1774
1775
1775
1776
1776
1777
1777
1778
1778
1779
1779
1780
1781
1782
1783
1784
1784
1785
1785
1786
1786
1787
1787
1788
1788
1789
1789
1790
1791
1792
1793
1794
1794
1795
1795
1796
1796
1797
1797
1798
1798
1799
1799
1800
1801
1802
1803
1804
1804
1805
1805
1806
1806
1807
1807
1808
1808
1809
1809
1810
1811
1812
1813
1814
1814
1815
1815
1816
1816
1817
1817
1818
1818
1819
1819
1820
1821
1822
1823
1824
1824
1825
1825
1826
1826
1827
1827
1828
1828
1829
1829
1830
1831
1832
1833
1834
1834
1835
1835
1836
1836
1837
1837
1838
1838
1839
1839
1840
1841
1842
1843
1844
1844
1845
1845
1846
1846
1847
1847
1848
1848
1849
1849
1850
1851
1852
1853
1854
1854
1855
1855
1856
1856
1857
1857
1858
1858
1859
1859
1860
1861
1862
1863
1864
1864
1865
1865
1866
1866
1867
1867
1868
1868
1869
1869
1870
1871
1872
18
```

- Causes re-render of component
- Render method is called for a second time()
- All child components are also re-rendered
- If we have function component as child, the component function is called again
- Updated value will be available in next render cycle
- **setState doesn't update value realtime.**
 - It often batches update calls
- Value on the console is still the old and stale one

setState behavior a closer look

- A typical state is an object that can have many properties
- setState() call can update one or more property in the state
 - It doesn't need to update all property everytime
 - The new properties are merged to the existing state
 - It doesn't replace the existing state
- Multiple calls of setState may be batched together before next render

```

handleCellClick=(id)=>{
  //console.log('cell',id,'clicked');
  //never change original value directly
  //always work on a duplicate
  const cells= [...this.state.cells];
  if(cells[id]!=='_')
    return ; //this value had earlier.

  cells[id]=this.state.next;
  const result= checkGame(cells);
  this.setState({result});

  const next=this.state.next==='0'? 'X': 'O';

  //update the state
  this.setState({cells,next});
  console.log('cell clicked', id);
}

getInitialState=(id)=>{
  const s={
    cells:[[' ',' ',' ',' ',' '],
           [' ',' ',' ',' ',' '],
           [' ',' ',' ',' ',' '],
           [' ',' ',' ',' ',' '],
           [' ',' ',' ',' ',' ']],
    next:'O'
  }
  s.result=checkGame(s.cells); //default result {over:false, movesLeft:9, winner:null, winningSequence:null}
}

```

1. Our state contains multiple properties
 - a. cells
 - b. next
 - c. Result
2. Here we are updating only the result of state
 - This will add the latest result to the state
 - Old result will be overwritten
 - It will not effect cells and next value
 - They will continue to be what they were.
3. Here we have made two separate setState call.
 - They don't execute realtime.
 - The state may be merged and all may get executed together before next render

Props and state

Thursday, January 13, 2022 12:13 PM

Props vs State

Props

- ▶ Special object containing values passed from parent to child
- ▶ Prop received from parent is immutable
 - ▶ component can't change props passed to it.
- ▶ Present in both class and function components
- ▶ Readonly for component that received it.

State

- ▶ Special Object to handle internal values of a component
- ▶ Can be changed to bring changes in the UI
 - ▶ Done using special setState method
- ▶ Present only in class component
- ▶ Can be changed.
 - 2. When Parent State changes it updates itself and passes updated props to each child. Thus children are also updated.

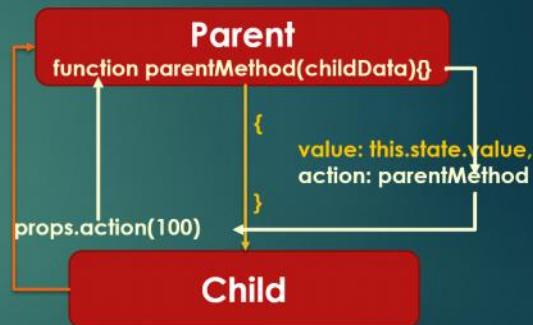
1. A component may pass its state as a prop to the child component

Parent-Child Communication

Thursday, January 13, 2022 12:17 PM

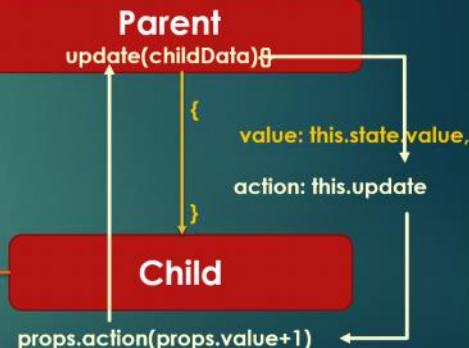
Communication between Components

- ▶ Parent Passes Value to Child as props
 - ▶ Generally it passes its state
- ▶ Child can communicate with parent by calling a method of the parent and passing information as parameter.
- ▶ Parent must first pass Child the method that child should call as part of props.
- ▶ Parent may update its state on receiving the value.



Communication between Components

```
class Parent extends React.Component{  
  constructor(){  
    this.state={ value:0 }  
  }  
  update=(value) { this.setState({value}) }  
  render(){  
    return <Child value={this.state.value} action={this.update}>  
  }  
}  
  
function Child ( props ) {  
  return <button  
    onClick={()=>props.action(props.value+1)}  
    value={props.value}>  
  </button>  
}
```



```
export default class Parent extends React.Component {  
  constructor(props) {  
    super(props);  
    this.state = { count: 0 };  
  }  
  increment = (delta) => this.setState({ count: this.state.count + delta })  
  render = () => {  
    return (  
      <div className="App">  
        <h1>Hello CodeSandbox</h1>  
        <PresenterChild value={this.state.count} />  
        <ActionChild />  
      </div>  
    );  
  }  
}
```

Hello CodeSandbox

Parent pass value to child

112

112 + 1 | 112 + 10 | 112 + 100

Child passing value to parent

```
<div className="App">
  <h1>Hello CodeSandbox</h1>
  <PresenterChild value={this.state.count} />
  <ActionChild
    delta={1}
    count={this.state.count}
    onUpdate={this.increment}>
  />
```

1. Parent passes value to child. It includes
 - a. Value that child can
 - b. Method that child can call to pass information to parent
2. Child display the UI
 - a. On click child calls parent method with desired value
3. Parent method updates the state by calling setState
4. Component and all it's childs are re-rendered with latest value

```
const ActionChild = (props) => {
  return (
    <button onClick={() => props.onUpdate(props.delta)}>
      {props.count} + {props.delta}
    </button>
  );
}
```

Takeaway

- Indirectly ActionChild informed a new value to PresenterChild
 - Communication between siblings

Communication between siblings

Thursday, January 13, 2022 12:42 PM

- No direct communication
- Each child must inform the Parent
- Parent must broadcast the information to each child

Back to Tic Tac Toe

Thursday, January 13, 2022 12:10 PM

- The value that a cell would display depend on value in the other cell
- If a cell value has changed once, it shouldn't change again.
- How will one cell know what value was entered in the previous cell.

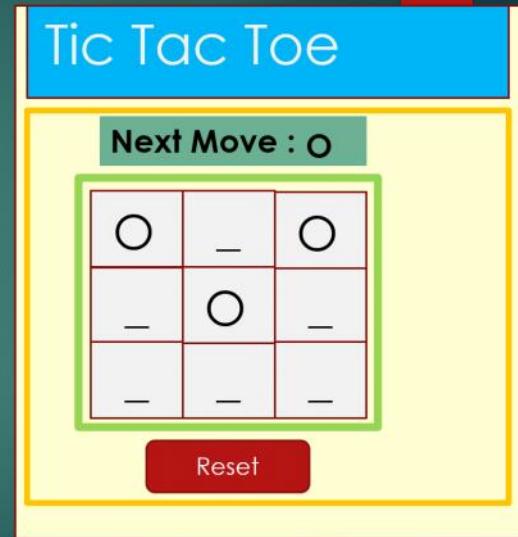
Cell Problem

- ▶ Each Cell works independently and changes from empty to "O"
- ▶ They don't know each other
- ▶ In the game The value of next cell depends on Previous.
- ▶ A React component can't interact with it's siblings.
- ▶ They can communicate via parent
- ▶ It is the job of the board to know which cell should display what value

Board

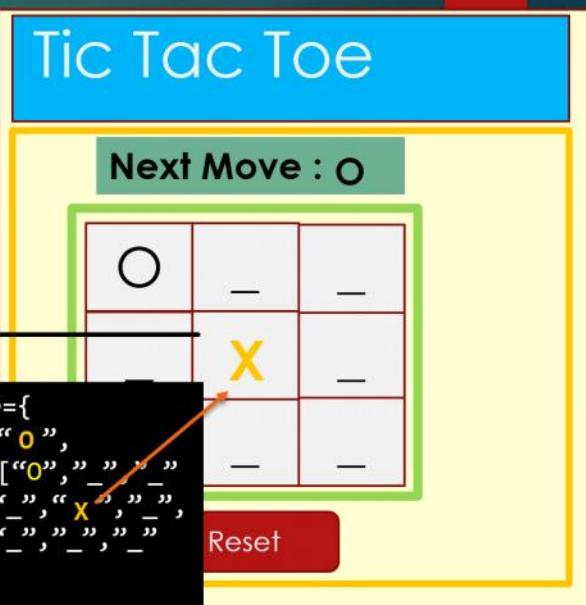
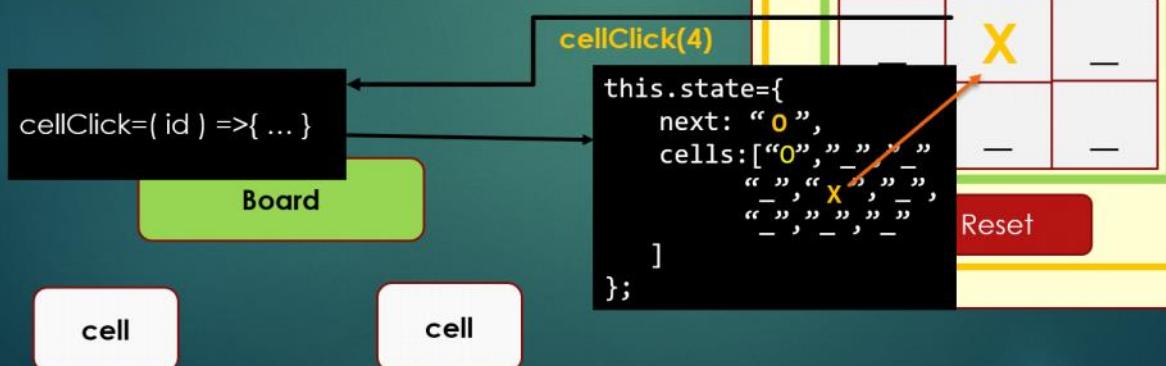
cell

cell



Board Cell Interaction

- Board should maintain state for
 - All cells
 - Value for the next cell
- Each cell should inform its click to Board
- Board updates its state
- Each cell is re-rendered with latest value



Calling the Parent Method

The screenshot shows the **Board.js** file in Visual Studio Code. It contains the following code:

```
21 handleCellClick=(id)=>{  
22   console.log('cell',id,'clicked');  
23 }  
24  
25  
26  
27 render=>{  
28   return (  
29     <div className='board'>  
30       <div className='cells'>  
31         <Cell value={this.state.cells[0]}  
32           onCellClick={this.handleCellClick} id={0} />  
33         <Cell value={this.state.cells[1]} onCellClick={this:  
34           handleCellClick} id={1} />  
35         <Cell value={this.state.cells[2]} onCellClick={this:  
36           handleCellClick} id={2} />
```

Two blue arrows point from the **onCellClick** props in the **Cell.js** component to the **handleCellClick** method in the **Board.js** component. Handwritten notes indicate:

- Value Handler**: Points to the **onCellClick** prop in the **Cell.js** component.
- variable (event)**: Points to the **onCellClick** prop in the **Cell.js** component.

The **Cell.js** file is also shown in the screenshot, containing the following code:

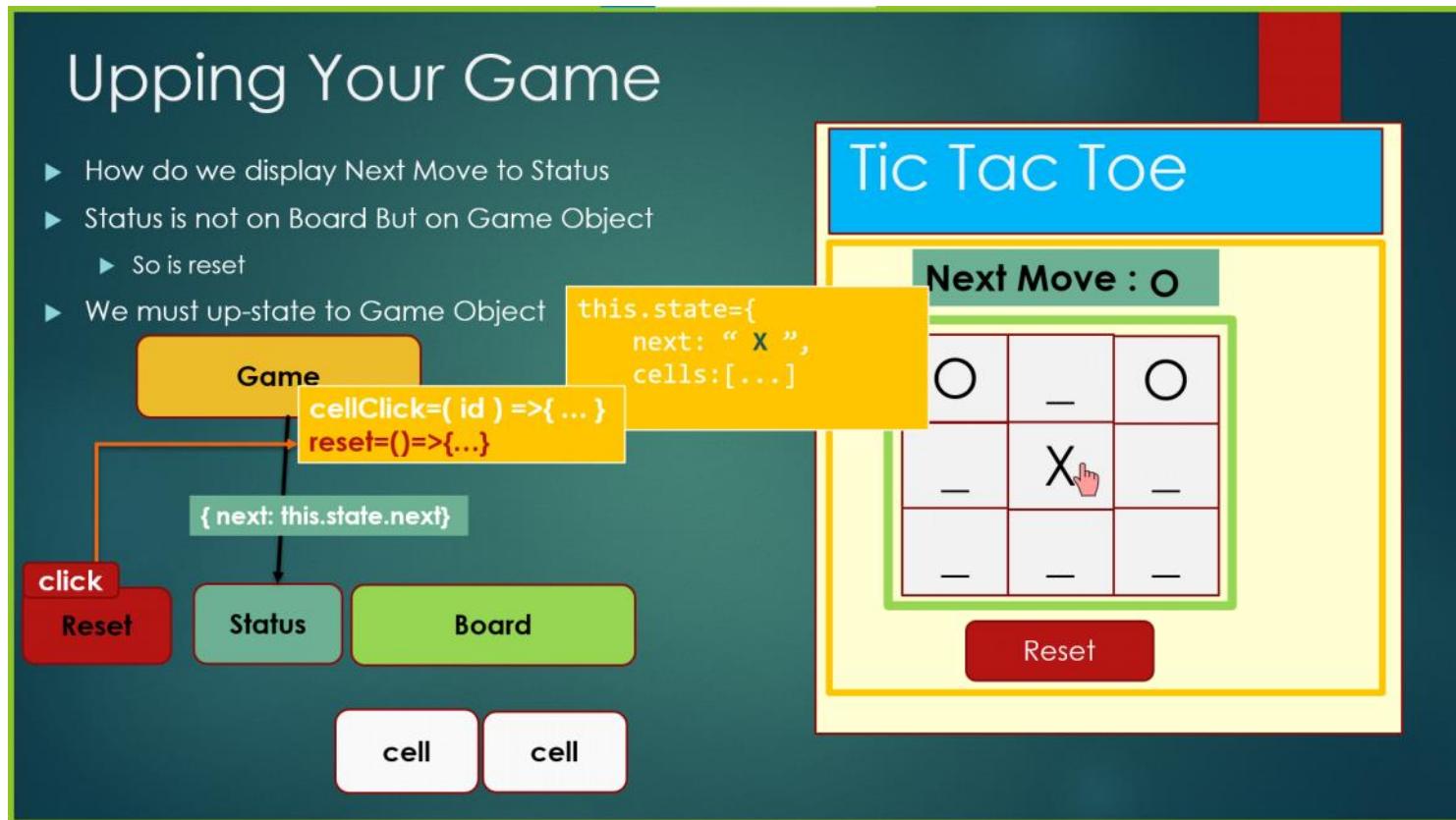
```
23  
24  
25  
26   return (  
27     <button className='cell'  
28       onClick={()=>this.props.onCellClick(this.props.id)}>  
29       {this.state.value}  
30     </button>  
31   );  
32  
33
```


Uplifting the state

Thursday, January 13, 2022 1:05 PM

- A state typically should belong to a parent component that can share the state to all child that need it.
- We often need to refactor our code to uplift the states to a higher parent component to make it usable
- In such cases we have downgrade a class component back to function component if it doesn't need its own state

1. We created state in the cell component
 - It was a bad call as
 - A Cell can't share its state to its sibling
 - A Board can share its state to all its children
2. Now we realize that even board is not the rightful owner of this information
 - This information should be shared to
 - Status
 - Reset button



Assignment 13.2

Thursday, January 13, 2022 1:21 PM

- Complete the Game
- Status message should tell
 - Next Move: O
 - Winner: 'O'
 - Stalemate
 - *If no winner!*
- Update your score board once a game is over
- If you press reset button after game is over
 - It doesn't change the score board
- If you press reset button while in the game it should not work

Interpolation

Friday, January 14, 2022 9:48 AM

Working with list challenges

- ▶ JSX interpolation syntax {} can work with
 - ▶ Simple Expressions
 - ▶ Function calls
- ▶ JSX interpolation syntax {} can't include statements like
 - ▶ For loop
 - ▶ If Statements

```
<div> { 2 + 2 } </div>

<div> { getDate() } </div>

<div> {
  for( let x : values)
    if(x)
      doSomething()
}
</div>
```



Working with a list/collection

Friday, January 14, 2022 9:48 AM

Working With Lists

- ▶ One of the most common requirement in an application is to display a list of values
 - ▶ A List of products on a e shopping website
 - ▶ A List of movies showing in nearest theatres
 - ▶ A List of Orders placed by a customer
 - ▶ A List of reviews given for a product
- ▶ Generally Data will be present in an array/list in javascript
- ▶ We need to display each item in some component like
 - ▶ table
 - ▶ div
 - ▶ li

A typical scenario when we have a large number of data present in some array and we want to display them using react components

Loop Alternative Array Functions

• **Array.forEach**

- Loops through each item
- You can do whatever you like.

```
books.forEach( book => console.log('book'));
```

• Can we use it with JSX?

- We can
- But we don't want to perform some action on JSX
- We want to return a bunch of JSX for current set of books.

• **Array.filter**

- Loops through each item
- Returns the item (unchanged) if it matches given condition

```
book.forEach(book => book.author.includes('Vivek'));
```

• May be useful for filtering a list

- But currently we need to display all items
- We are not interested in the item itself but in JSX needed to display that item.

• **Array.map**

- We can use it to return JSX data

- Loops through each item
- Can return an array of items
- Returned items are somehow related to original item
- The returned item may be different from the original item

```
book.forEach( book=> `<h2>${book.title}</h2>`);
```

Array.JSX to return a JSX List of books

```
book.map( book=> <h2>{book.title}</h2>)
```

```

App.js
1 import "./styles.css";
2 const books = [...];
3;
4 export default function App() {
5;
6;
7 return (
8     <div className="App">
9         <h1>My Books</h1>
10        Total Book: {books.length}
11        <ul>
12        {
13            books.map(book=><li>
14                <img alt={book.title} size={book.cover}/>
15                {book.title}</li>
16            )
17        }
18        </ul>
19    </div>
20);
21;

```

My Books

Total Book: 5

- The Accursed God
- Harry Potter and the Philosopher's Stone
- Five Little Pigs
- Harry Potter and the Half-Blood Prince

Console Problems React DevTools Filter

Console was cleared

Warning: Each child in a list should have a unique "key" prop.

Check the render method of `App`. See <https://reactjs.org/link/warning-keys> for more information.

at li
at App

"key" Props

- A Key uniquely identifies a component on the UI.
- With key changes we identify component uniquely
- Key is specially needed while displaying a list item
- It can identify each item uniquely

Why is this important!

- For a list whose item will not change, this is not important
 - It is important For a list whose item may get
 - Modified
 - deleted
 - Say you have a list of 1000 items and you deleted one item
 - Without key prop
 - ReactDOM doesn't know which item is removed
 - It will re-render the entire list
 - It will remove all items and add them again.
 - With key prop
 - ReactDOM can update only one item directly in the DOM.
 - This will avoid unwanted large updates
 - Improves the performance
- It is just a warning not an error.
 - Always a good idea to provide a key
 - If array doesn't change you can use array index as the key
 - If array changes think of some key
 - Generally all business objects will have some natural key like id/primary keys
 - Make sure your key is always unique

Console Problems React DevTools Filter

Check the render method of `App`. See <https://reactjs.org/link/warning-keys> for more information.

at li
at App

Warning: Encountered two children with the same key, 'JK Rowling'. Keys should be unique so that components maintain their identity across updates. Non-unique keys may cause children to be duplicated and/or omitted – the behavior is unsupported and could change in a future version.

at ul
at div
at App

Assignment Day 14.1

Friday, January 14, 2022 10:25 AM

Add Following Elements to the Tic Tac Toe

Games	O Wins	X Wins	Draw
4	2	1	1

Also

NEXT : O

X		X
	O	

Moves

Moves	Player	Position
1	X	1
2	O	5
3	X	3

- Regenerate cells in the board using
 - array.map

[Reset](#)

Destructuring props

Friday, January 14, 2022 1:23 PM

- User may pass multiple props
- Everytime component uses a prop key we need to write prop.key
- We can use param restructuring to simple take parameters we need.

```
const Status = ({next, result}) => {  
  let message = `Next : ${next}`; //default message  
  if(result.over){  
    if(result.winner){  
      message = `${result.winner} Wins`;  
    } else  
      message = "Stalemate";  
  }  
  
  return (  
    <div className="status">  
      {message}  
    </div>  
  );  
}
```

Displaying UI conditionally

Friday, January 14, 2022 1:26 PM

- Sometime particular UI element should be rendered conditionally
- Consider the reset button
 - It should reset to new game only when current game finished
 - It should not reset a running game



JSX doesn't support if statement

- We can use conditional operators to handle similar situation

1. Ternary Operator ?:

- We can use ternary operator if we need to display a UI conditionally

```
{ state.result.over ? <button>reset</button> ? null }
```

- Null means render nothing
- I may render different component in if and else.

2. Conditional operator && and ||

- Conditional operators have a short circuit in nature in most programming language

2.1 && operator

- And checks if both condition is true

```
condition1 && condition2
```

- If condition1 is false, then overall value will be false.
- Is there any point in solving condition2?
 - Most language doesn't solve second condition of && if first is false
 - Second is solved only if first is true.

2.1 || operator

- Or checks if either of the two condition is true

```
condition1 || condition2
```

- If condition1 is true, then value will be true
- Is there any point in solving condition2?
 - Most language will not evaluate condition2 of || if first is already ready true

We can emit conditional JSX using boolean operator

Display a JSX only if condition is true

```
{condition && <SomeJSX/>}
```

- The JSX will be rendered only if condition is true
 - If condition is false second part of if is ignored.

Display a JSX only if condition is false

```
{condition || <SomeJSX/>}
```

3. Create a custom component --> If

Conditional styling and event

Friday, January 14, 2022 2:55 PM

```
const cell = ({value, onCellClick, id}) => [
  let handleCellClick = () => onCellClick(id);

  let style = {};

  if(value!=='_'){
    style.cursor='not-allowed';
    handleCellClick=null; //no event will be triggered.
  };

  //console.log('style: ',id, style);
  return (
    <button className='cell' style={style}
      onClick={handleCellClick}>
      {value}
    </button>
  );
}
```

- We can use javascript object as a style object here
- We can conditionally change the style on our object

- If we set and event handler to null that even will not be fired.
- This also can be set dynamically.

Assignment 14.2

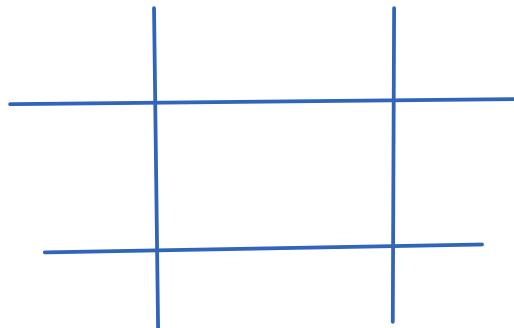
Friday, January 14, 2022 2:56 PM

1. Hide the "_" display on the cell board
2. Implement following Ideas when the game is over

- User shouldn't be able to click the remain empty cells.
- If there is a winner the winning cells should be highlighted with light green background
- If it is a draw all the cells should be displayed dimmed.

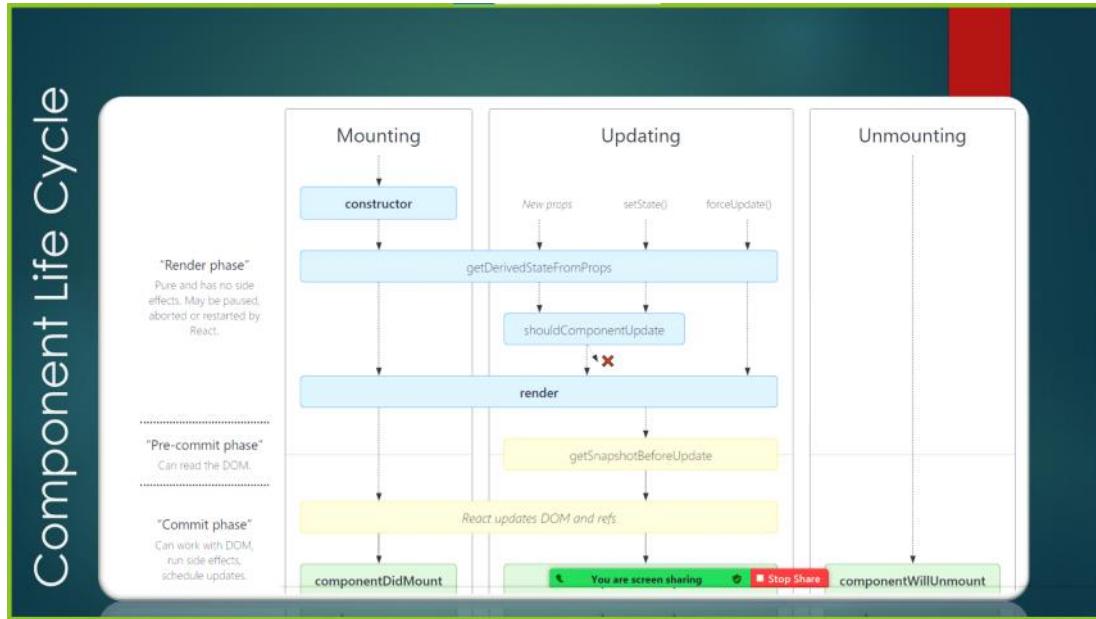
3. Currently our board looks like 9 cubes. Let it look like a tick-tac-toe board

Hint: play with the border property of the cell.



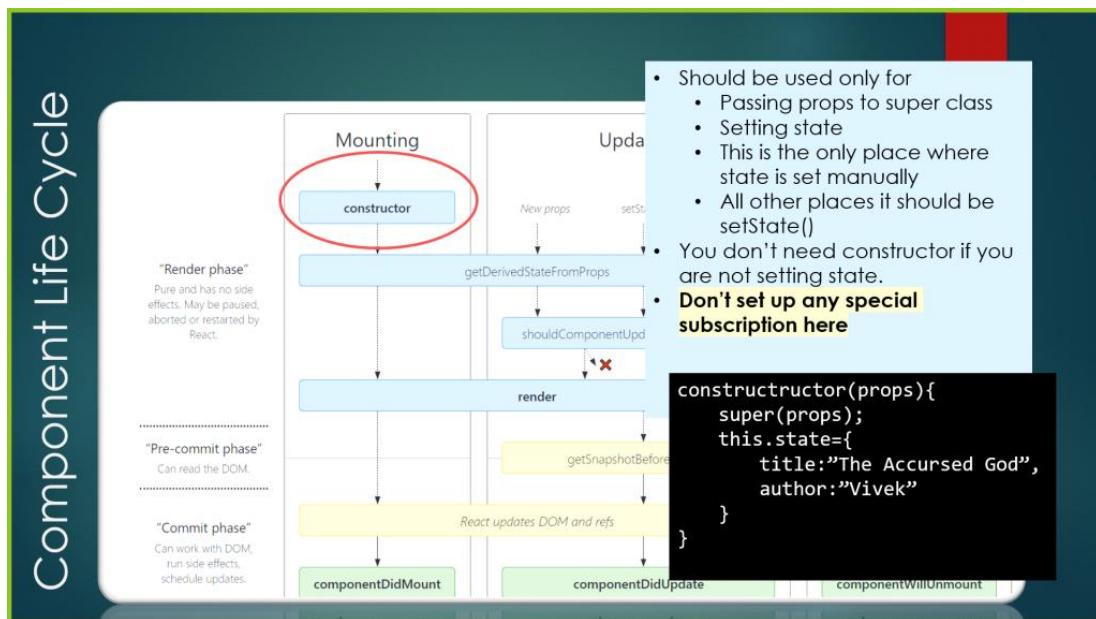
Component Life Cycle

Friday, January 14, 2022 4:23 PM



Step 1 Constructor

- Not yet displayed on the UI

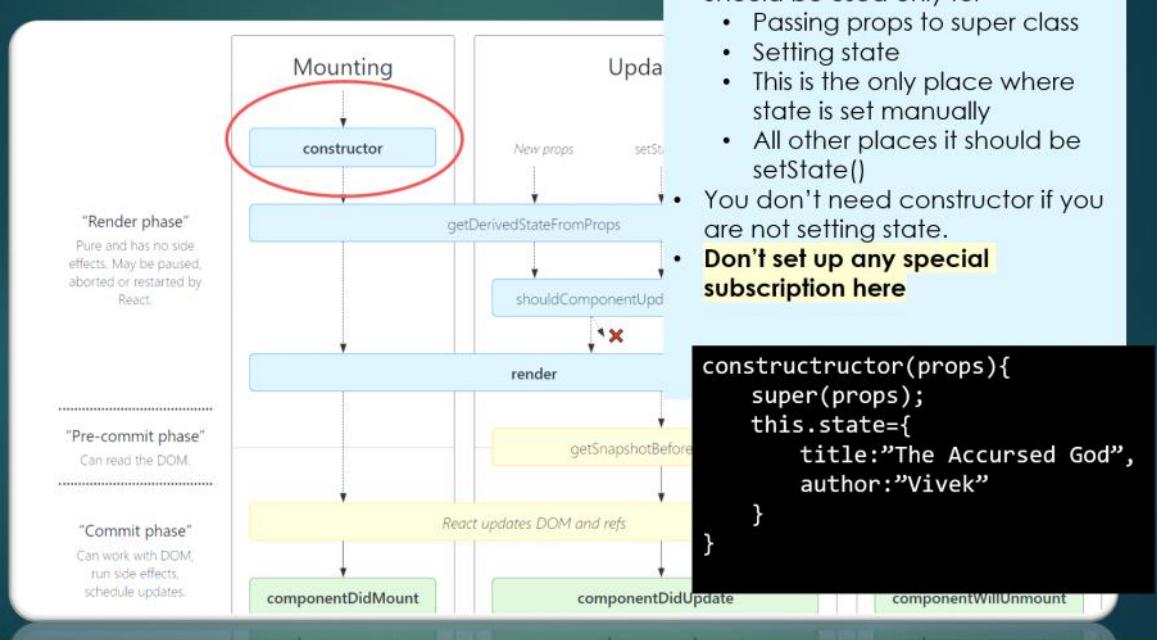


Step 2. Render

- Creates virtual DOM
- Should return JSX
- No subscription
- No Async operation

- No setState

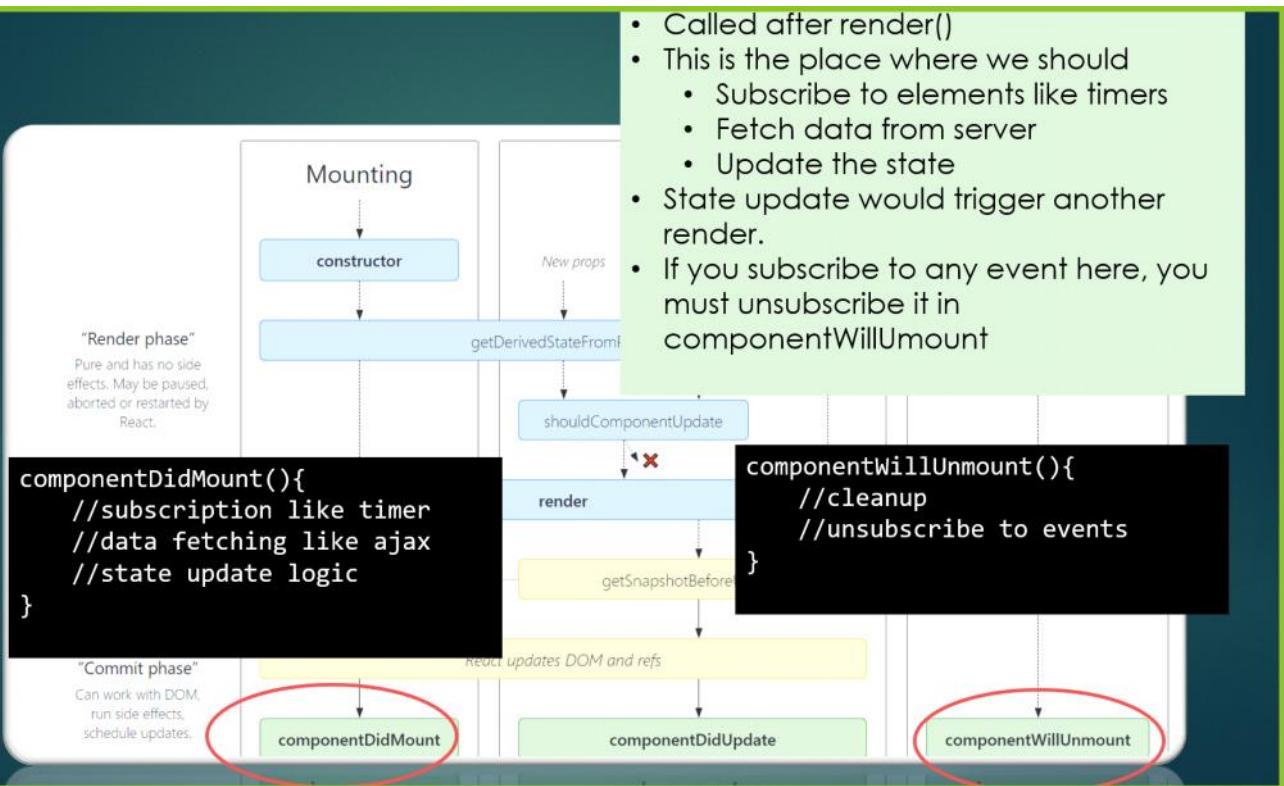
Component Life Cycle



Step 3 componentDidMount

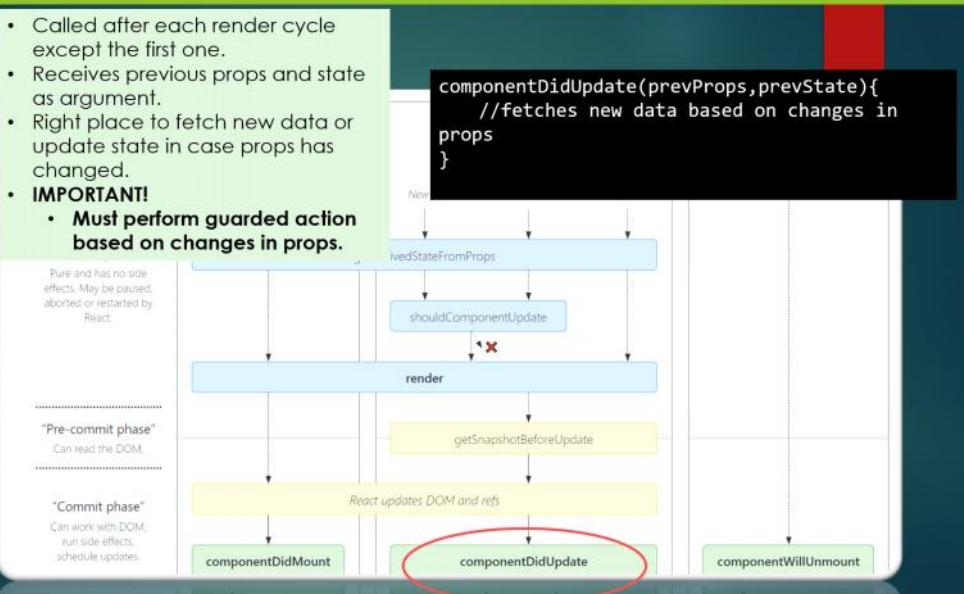
- Called after component has been rendered and displayed on the UI
- Here we can take our next action
 - Async function call
 - Request for data to external server
 - Subscriptions like
 - `setTimeout/setInterval`
 - `setUpdate`
- Occurs Only once in component life cycle
- Should be paired with `componentDidUnMount` to cancel any
 - Subscription
 - Timer
- Component didMount is not needed if you performed a fetch task and that is over.

Component Life Cycle



Step 4 — componentDidUpdate

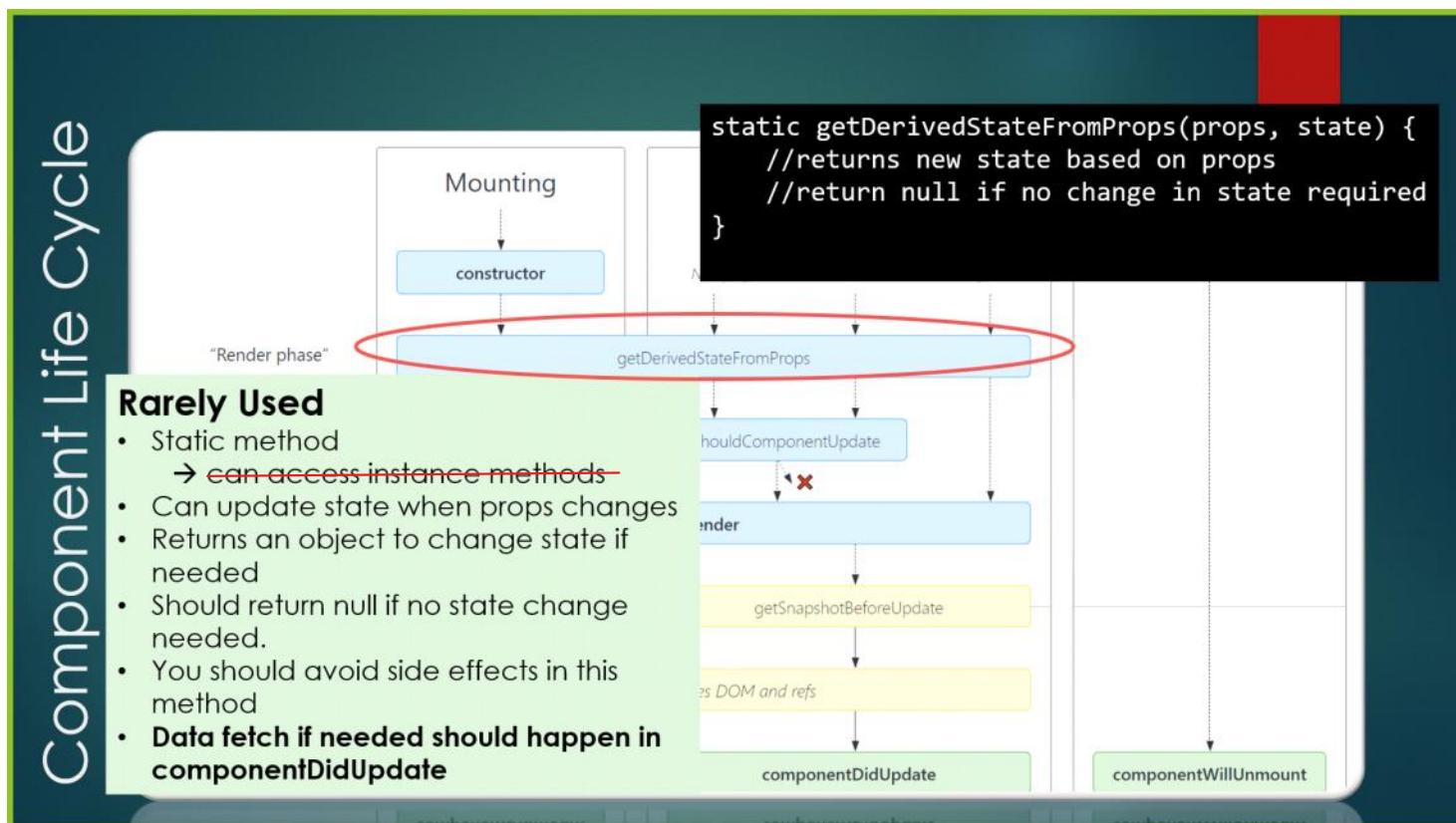
- Called after every render
- If a state is changed
 - Render is called
 - Redner again calls componentDidUpdate
- You may change state in this function
 - But that may lead to infinite loop
 - Be careful
 - Always update state after a some if check**
 - To prevent infinite loops



- Takes two parameters
 - previousProps
 - previousState
- Can be compared with this.props and this.state
- Check if some prop changed from outside that should update my UI

Rare Step — getDerivedStateFromProps

- Is there a state in component that depends on the outside prop?
- If prop changes, you may want to change the state
- Returns new state based on the prop
- Return null if no change required
- It can't access component methods
- Actual data fetching should be done in componentDidMount/componentDidUpdate



Assignment 14.3

Friday, January 14, 2022 4:57 PM

- Add Two Timers on for each player.
- Using Digital Number Font
- Timer of current player should be counting
- When plays makes move the timer should pause itself
- Timer of next player should start counting

Score Board			
Games	O Wins	X Wins	Draw
0	0	0	0

Next : O

-	-	-
-	-	-
-	-	-

#	Player	Position
---	--------	----------

O's Timer

00:23

X's Timer

00:11

- Timers should automatically pause and play based on player's turn.
- In case of a draw, player with smaller value in timer will be considered a winner.

How do I control the internal state of a component

Saturday, January 15, 2022 10:56 AM

A Stopwatch

- It maintains an internal state.
- It can be controlled by a button inside it

Solution

1. Call this method from outside
2. Change the state from outside

```
togglestate=()=>{
  console.log('toggling state...'); 
  this.setState({running: !this.state.running})
}

render(){
  const {cs,seconds,min}=this.getTime();
  return (
    <div className='stop-watch'>
      <p>{this.props.label}</p>
      <p className='stop-watch-time'>
        <span className='stop-watch-min'>...
        </span>
        <span condition={this.props.fraction}>
          <span className='stop-watch-cs'>
            .{cs}
          </span>
        </span>
      </p>
      <If condition={this.props.controls}>
        <button onClick={this.togglestate}>
          {this.state.running?"PAUSE":"RESUME"}
        </button>
      </If>
    </div> );
}
```

Updates internally

1. How to call a child components function from the parent?

Refs

Saturday, January 15, 2022 11:01 AM

- In React we are not expected to use
 - `document.getElementById()`
- Remember a ReactComponent is not a real DOM element
- It is not attached to the DOM directly
- You may not be able to access the component properly.

Ref

- Can be considered React Equivalent of `document.getElementById`
- It is a three step process.

Step 1. create a ref

- A ref is created using `React.createRef()`
- It is an empty pointer to start with
- It should be attached to a React component before use

```
this.oTimerRef=React.createRef();
this.xTimerRef=React.createRef();
```

Step 2. Make a ref point to some component in JSX

```
<Timer ref={this.oTimerRef} />
<Timer ref={this.xTimerRef} />
```

Step 3. Once a ref points to a ReactComponent it is like variable pointing to object

- Now you can access any method/field of that object using **current reference**

```
this.oTimerRef.current.toggleState();
```

Ref type

```
class Timer{  
    state={ ms: 0 }  
  
    start=()=> { ... }  
    stop= () => { ... }  
  
    render=()=>{  
        return <div> {ms} </div>;  
    }  
  
}  
  
const TimerClient = () => {  
  
    const timerRef = React.createRef();  
  
    const handleStart =() => { ref.current.start(); }  
    const handleStop =() => { ref.current.stop(); }  
  
    return <Timer  
        ref={timerRef}  
    />  
}
```

How to Access components methods from Client

1. Create a ref object

2. Attach 'ref' to Timer Component

3. Invoke the methods

Warning

- 'ref' goes against the declarative programming model of React
 - In react we shouldn't be 'controlling'
 - React should be rendering based on data changes
- We should separate data flow from UI rendering
- We control our data
 - Our data controls UI
- We shouldn't manage 'control flow' but 'data flow'

Use Ref only rarely when you are out of all other options

Assignment 15.1

Saturday, January 15, 2022 12:31 PM

- Implement in declarative model without 'ref'
 1. Stop the watches when the game is over
 2. Reset the watches when game starts
 3. We should not be able play the game without start
 - Remove Alert box
 - Provide a better solution.
 4. Add feature of winning by time

Books Client

Saturday, January 15, 2022 12:53 PM

Reference Implementation: <https://ca-booksapi.herokuapp.com/>

Navigation Bar
• Common for all pages

Site Title

Links

Search Bar

A dropdown set of Links

Home Page

- Shows a list of quotes at random
- The quotes changes every few seconds
- We may refresh it manually

Our Recommendations

- Clicking link takes you to a different page
 - The URL of this page is different
 - Looks like Multi-Page Application

If there is a delay in loading content you get a loading animation

Dynamic url with isbn

Book's Web Books Add Book Authors Search Guest

The Accursed God
by Vivek Dutta Mishra

THE ACCURSED GOD THE LOST EPIC BOOK 1

HARRY POTTER and the PHILOSOPHER'S STONE J.K. ROWLING

Agatha Christie THE QUEEN OF MYSTERY FIVE LITTLE PIGS A Hercule Poirot Mystery

J.K. ROWLING HARRY POTTER and the Half-Blood Prince

Price: 179 ★★★★☆

Synopsis

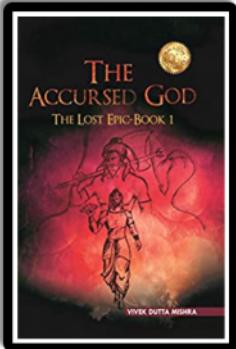
THE LOST EPIC The story of the epic battle of Kurukshetra has been told and retold for ages. Millennia of dust, fables, imaginations — and the epic itself is lost. What remained is the story of a family feud and ambition of Kauravas and Pandavas. But why, then, was this an epic war? Why entire Aryavart plunged into this first real world-war? Why the echo of this ancient war still resonates after all those centuries? Rediscover the lost epic whose origin lies in the birth of the Kurukshetra that had tasted its first blood over a hundred years before the final Mahabharata war. Discover the complete saga of Mahabharata which goes far and beyond just Kauravas and Pandavas and their ambitions. THE ACCURSED GOD ===== Long before the epic battle, long before even the birth

< > C 88 | ca-booksapi.herokuapp.com/book/details/9781393495574

Book's Web Books Add Book Authors Search Search Guest ▾

The Accursed God

by Vivek Dutta Mishra



• Price: 179
★★★★★

Synopsis

THE LOST EPIC The story of the epic battle of Kurukshetra has been told and retold for ages. Millennia of dust, fables, imaginations — and the epic itself is lost. What remained is the story of a family feud and ambition of Kauravas and Pandavas. But why, then, was this an epic war? Why entire Aryavart plunged into this first real world-war? Why the echo of this ancient war still resonates after all those centuries? Rediscover the lost epic whose origin lies in the birth of the Kurukshetra that had tasted its first blood over a hundred years before the final Mahabharata war. Discover the complete saga of Mahabharata which goes far and beyond just Kauravas and Pandavas and their ambitions. THE ACCURSED GOD ===== Long before the epic battle, long before even the birth of Kurukshetra, a man swore on his father's pyre to avenge against the mightiest empire, any civilization had ever seen. Between his might and near-certain destruction of the Empire, stood a warrior, who rose like a phoenix from the ashes of his seven dead brothers — taking the mantle of a fabled Accursed God. In the clash that followed, Aryavart heard several more oaths by the side of more burning pyres, until, a young king decided that no price is too high for peace. Little did he know that the price would be a war engulfing the entire Aryavart, where few would

For Invalid URL

< > C 88 | ca-booksapi.herokuapp.com/book/details/978139349557477

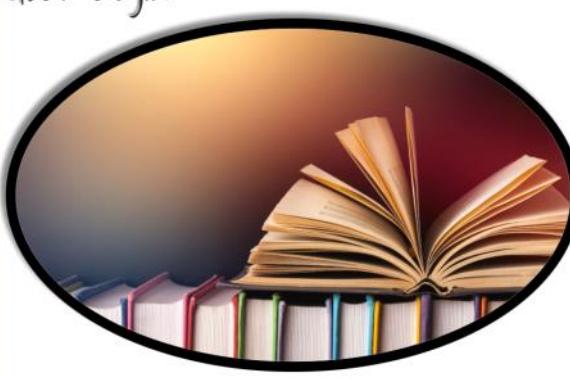
Book's Web Books Add Book Authors S

Not Found

Sorry no book with isbn: 978139349557477 present in our record

Book's Web Books Add Book Authors Search Search Guest ▾

User Login



@ email address

password

Login

Add Book

Book's Web Books Add Book Authors

Search Guest ▾

Add New Book

ISBN

TITLE

AUTHOR

PRICE

COVER

DESCRIPTION

Author Manage Page

< > 🔍 🔒 ca-booksapi.herokuapp.com/author/manage

Book's Web Books Add Book Authors

Search Guest ▾

Author Manager

Mohan Das Karmchand Mahatma Gandhi	<input type="button" value="New Author"/>
Vivek Dutta Mishra	
Ramdhari Singh Dinkar	
Jeffrey Archer	
John Grisham	
Alexandre Dumas	
JK Rowling	
Agatha Christie	

New Author



Organization

Saturday, January 15, 2022 2:20 PM

- Our application is broadly divided in
 - Screens
 - One full view that you see in the browser
 - BookListScreen
 - UserLoginScreen
 - Components
 - Other smaller UI elements that will appear within the screen
 - StarRatingComponent
 - Custom Button
 - SearchBar
 - Service
 - Interacts with business layer
 - May get data from outside
 - Any other Business data

CSS Libraries

Saturday, January 15, 2022 2:29 PM

- We will use a few third party libraries to make our application stand out

Bootstrap

- One of the most popular css library with a lot of built-in styling for different kind of design
- <https://getbootstrap.com/docs/4.0/getting-started/introduction/>

FontAwesom

- A css library that provides icons to display on our page

Installation

Option#1 — index.html

- Include your library url from CDN (Content delivery Network) in the header of your Index.html
- You should include
 - .css file in he head section
 - ```
<link rel="stylesheet"
 href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css"
 integrity="sha384-Gn5F0JiVvfXWfA058RXPxPg6fy4IWvTNh0E263XmFcJ1SAwiGgFAW/dAiS6JXm" crossorigin="anonymous">
```
    - .js file at the bottom before the closer of the body tag.
      - ```
<script src="https://code.jquery.com/jquery-3.2.1.slim.min.js" integrity="sha384-KJ3o2DKtkYIK3UENmM7KCKR/rE9/Qpg6aAZGJwFDMVNA/GpGFF93hXpG5KkN"
  crossorigin="anonymous"></script>
<script src="https://cdn.jsdelivr.net/npm/popper.js@1.12.9/umd/popper.min.js"
  integrity="sha384-ApNbgh8BqxDuQIvvAICfF8Fvj不舍
  crossorigin="anonymous"></script>
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/bootstrap.min.js"
  integrity="sha384-JZR6Spjh4U02d8Ot6LEHfe/JQGiRRSQQxSFWpi1MquVdAyUar5+76PVCmYI" crossorigin="anonymous"></script>
```

From <<https://getbootstrap.com/docs/4.0/getting-started/introduction/>>

Option#2 — Install as npm package

- Install the required npm package
- Import them in index.js

Step #1 get bootstrap from npm

```
npm i bootstrap@4.6.0
```

From <<https://www.npmjs.com/package/bootstrap/v/4.6.0>>

Step #2 Get jquery and popper libraries

Styling

Saturday, January 15, 2022 3:13 PM

- Css doesn't support variables or functions to dynamically work
- We may need to introduce variables that can be changed later
 - Toolbar color (primary color)
 - Secondary color
 - Height of a component

SASS/SCSS

- A new styling standard
- Supports variable
- Support extending of one style by another
- Supports function creation in style document
- Supports inclusion of one document into another

SASS/SCSS is NOT supported by the browser

- Just like browser don't support
 - Typescript
 - JSX
 - Many new ES syntax.
- We can use Transpiler that can translate SCSS → CSS on the fly.

How to use SCSS in our React Project

Step #1

- Install SCSSS transpiler in your react project

Step #2

- Rename your .css file as .scss
 - Every .css feature works the same way in scss
 - Now we can use new variables here

Step

- Import .scss file in you index.js
- DO NOT INCLUDE IT DIRECTLY IN index.html

Assignment 15.2

Saturday, January 15, 2022 4:44 PM

1. Clicking the book in the side list should show the details of those book on the right pane

The screenshot shows a web application interface titled "World Wide Books". On the left, there is a sidebar with a list of books: "The Accursed God", "Harry Potter and the Philosopher's Stone", "Five Little Pigs", "Harry Potter and the Half Blood Prince", and "Harry Potter and the Order of the Phoenix". The first item, "The Accursed God", is highlighted with a blue arrow pointing to its details page on the right. The details page for "The Accursed God" by Vivek Dutta Mishra includes a "Meta" section with ISBN, price, and rating information, and a "Details" section with a summary of the epic story. To the right of the details is a thumbnail image of the book cover for "The ACCURSED GOD: THE LOST EPIC-BOOK 1".

- It should show the selected book in the list as highlighted using primary_color
2. Create empty (non-functioning) screens with dummy data based on the reference implementation

<https://ca-booksapi.herokuapp.com/>

Reusable Parent to Add feature

Monday, January 17, 2022 10:09 AM

If is a parent/container component

- It doesn't have its own UI
- It adds some feature around other components
 - Conditional visibility

```
<If condition={this.state.screen==='HOME'}>  
  <HomeScreen/>  
</If>  
  
<If conditon={this.state.screen==='BOOKS'}>  
  <BookListScreen/>  
</If>  
  
<If condition={this.state.screen==='AUTHORS'}>  
  <AuthorListScreen/>  
</If>
```

Advantage

- We have single component to conditionally render components
- It can be used everywhere just by wrapping components with an If block.

Disadvantage

- Who will press the button?
- You need to write If statement every time.

What I want ?

```
<HomeScreen visible={this.state.screen==='HOME' /}>  
<BookListScreen visible={this.state.screen==='BOOKS' /}>  
<AuthorListScreen visible={this.state.screen==='AUTHORS' /}>
```

Problem

- We need to write repetitive code for same Logic across all components

```
const HomeScreen( props ) =>{  
  //check if we shouldn't be visible  
  if(!props.visible)  
    return null;  
  
  // component specific logic  
  ...  
}  
  
const BookListScreen( props ) =>{  
  //check if we shouldn't be visible  
  if(!props.visible)  
    return null;  
  
  // component specific logic  
  ...  
}  
  
const AuthorListScreen( props ) =>{  
  //check if we shouldn't be visible  
  if(!props.visible)  
    return null;  
  
  // component specific logic  
  ...  
}
```

Higher Order Component

Monday, January 17, 2022 10:17 AM

How to Reuse Elements across different components without re-writing

- It is an automatic wrapping of a component in a re-usable parent

HOC vs Parent Component

- ▶ HOC is designed as re-usable container component
- ▶ It doesn't know actual Child Component
- ▶ Once composition is done, user treats HOC+Target as Target
- ▶ User of component may not know an HOC is applied.
- ▶ User doesn't do manual composition of HOC and Parent
- ▶ It is meant to enhance the functionality of the Target Component
- ▶ It behaves as Enhanced Target Component

UserComponent + HOC = SmarterUserComponent

- We apply HOC using a Closure style

- A function that takes a component
- It returns another enhanced component.
 - Enhanced component is
 - HOC <----- parent
 - Component <----- child

Most Basic Nothing doing HOC Model

```
//useless-hoc.js
export const addUselessHOC = ( Component ) =>{

  function EnhancedComponent( props ){
    return <Component {...props} />
  }

  return EnhancedComponent;
}

//my-component.js
import {addUselessHOC} from 'useless-hoc';
const HelloWorldComponent=(message)=>{

  return <h1>{message}</h1>;
}

export default addUselessHOC(HelloWorldComponent);

//app.js
import HelloWorldComponent from 'my-component';

const App =()=>{
  return <HelloWorldComponent message="Hey HOC" />
}
```

HOC Design

- A function that takes a React Component
- It creates another react component called EnhancedComponent
- EnhancedComponent is like a Parent
 - It includes my OriginalComponent
- User will use EnhancedComponent instead of OriginalComponent
- props will be passed to EnhancedComponent
- EnhancedComponent must pass them to OriginalComponent

How do we compose HOC + TargetComponent

- We export the component returned by HOC creator closure function
- We pass our original component
- Client always gets the enhanced component

What is HelloWorldComponent?

}

Coding Convention

- HOC closure function is written with **"with"** prefix
 - `WithTitle`
 - `WithVisibility`
 - `WithDate`

HOC Examples

1. Common JSX like Screen Title

- Consider the below code

```

1 const BookListScreen=()=>{
2   //TODO: Initialize Here
3   const books=[...]
4
5   return (
6     <div>
7       <h1>Book List</h1>
8     <div className='BookListScreen'>...
9     </div>
10    </div>
11  );
12}
13
14 export default BookListScreen;

```

```

1 const BookDetailsScreen = (...args) => {
2   //TODO: Initialize Here
3   const book = {...}
4
5   return (
6     <div>
7       <h1>{book.title}</h1>
8     <div className='BookDetailsScreen'>...
9     </div>
10    </div>
11  );
12}
13
14 export default BookDetailsScreen;

```

- Each component has its own logic (shown in blue and green color)
- But they all will need a screen title.
 - That is a common reusable logic.
- We will have a dozen of Screens.
- We will need to write the same code again and again.

```

JS BookListScreen.js 1, M X JS BookDetailsScreen.js 1, M X JS with-title.js U X
src > screens > JS BookListScreen.js > ...
1 import React from 'react';
2
3 const BookListScreen=()=>{
4   //TODO: Initialize Here
5   const books=[...]
6
7   return (
8     <div>
9       <h1>Book List</h1>
10      <div className='BookListScreen'>...
11      </div>
12    </div>
13  );
14
15 export default BookListScreen;
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142

```

```

1 export const withTitle=(TargetComponent)=>{
2
3   const TitledComponent=(props)=>{
4     let title='Dummy';
5     return (
6       <div>
7         <h1>{title}</h1>
8         <TargetComponent {...props} />
9       </div>
10      )
11    }
12
13    return TitledComponent;
14  };
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
39
40
41
42
43
44
45
46
47
48
49
49
50
51
52
53
54
55
56
57
58
59
59
60
61
62
63
64
65
66
67
68
69
69
70
71
72
73
74
75
76
77
78
79
79
80
81
82
83
84
85
86
87
88
89
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
109
110
111
112
113
114
115
116
117
118
119
119
120
121
122
123
124
125
126
127
128
129
129
130
131
132
133
134
135
136
137
138
139
139
140
141
142
143
144
145
146
147
148
149
149
150
151
152
153
154
155
156
157
158
159
159
160
161
162
163
164
165
166
167
167
168
169
169
170
171
172
173
174
175
176
176
177
178
179
179
180
181
182
183
184
185
185
186
187
187
188
189
189
190
191
192
193
193
194
195
195
196
196
197
197
198
198
199
199
200
200
201
201
202
202
203
203
204
204
205
205
206
206
207
207
208
208
209
209
210
210
211
211
212
212
213
213
214
214
215
215
216
216
217
217
218
218
219
219
220
220
221
221
222
222
223
223
224
224
225
225
226
226
227
227
228
228
229
229
230
230
231
231
232
232
233
233
234
234
235
235
236
236
237
237
238
238
239
239
240
240
241
241
242
242
243
243
244
244
245
245
246
246
247
247
248
248
249
249
250
250
251
251
252
252
253
253
254
254
255
255
256
256
257
257
258
258
259
259
260
260
261
261
262
262
263
263
264
264
265
265
266
266
267
267
268
268
269
269
270
270
271
271
272
272
273
273
274
274
275
275
276
276
277
277
278
278
279
279
280
280
281
281
282
282
283
283
284
284
285
285
286
286
287
287
288
288
289
289
290
290
291
291
292
292
293
293
294
294
295
295
296
296
297
297
298
298
299
299
300
300
301
301
302
302
303
303
304
304
305
305
306
306
307
307
308
308
309
309
310
310
311
311
312
312
313
313
314
314
315
315
316
316
317
317
318
318
319
319
320
320
321
321
322
322
323
323
324
324
325
325
326
326
327
327
328
328
329
329
330
330
331
331
332
332
333
333
334
334
335
335
336
336
337
337
338
338
339
339
340
340
341
341
342
342
343
343
344
344
345
345
346
346
347
347
348
348
349
349
350
350
351
351
352
352
353
353
354
354
355
355
356
356
357
357
358
358
359
359
360
360
361
361
362
362
363
363
364
364
365
365
366
366
367
367
368
368
369
369
370
370
371
371
372
372
373
373
374
374
375
375
376
376
377
377
378
378
379
379
380
380
381
381
382
382
383
383
384
384
385
385
386
386
387
387
388
388
389
389
390
390
391
391
392
392
393
393
394
394
395
395
396
396
397
397
398
398
399
399
400
400
401
401
402
402
403
403
404
404
405
405
406
406
407
407
408
408
409
409
410
410
411
411
412
412
413
413
414
414
415
415
416
416
417
417
418
418
419
419
420
420
421
421
422
422
423
423
424
424
425
425
426
426
427
427
428
428
429
429
430
430
431
431
432
432
433
433
434
434
435
435
436
436
437
437
438
438
439
439
440
440
441
441
442
442
443
443
444
444
445
445
446
446
447
447
448
448
449
449
450
450
451
451
452
452
453
453
454
454
455
455
456
456
457
457
458
458
459
459
460
460
461
461
462
462
463
463
464
464
465
465
466
466
467
467
468
468
469
469
470
470
471
471
472
472
473
473
474
474
475
475
476
476
477
477
478
478
479
479
480
480
481
481
482
482
483
483
484
484
485
485
486
486
487
487
488
488
489
489
490
490
491
491
492
492
493
493
494
494
495
495
496
496
497
497
498
498
499
499
500
500
501
501
502
502
503
503
504
504
505
505
506
506
507
507
508
508
509
509
510
510
511
511
512
512
513
513
514
514
515
515
516
516
517
517
518
518
519
519
520
520
521
521
522
522
523
523
524
524
525
525
526
526
527
527
528
528
529
529
530
530
531
531
532
532
533
533
534
534
535
535
536
536
537
537
538
538
539
539
540
540
541
541
542
542
543
543
544
544
545
545
546
546
547
547
548
548
549
549
550
550
551
551
552
552
553
553
554
554
555
555
556
556
557
557
558
558
559
559
560
560
561
561
562
562
563
563
564
564
565
565
566
566
567
567
568
568
569
569
570
570
571
571
572
572
573
573
574
574
575
575
576
576
577
577
578
578
579
579
580
580
581
581
582
582
583
583
584
584
585
585
586
586
587
587
588
588
589
589
590
590
591
591
592
592
593
593
594
594
595
595
596
596
597
597
598
598
599
599
600
600
601
601
602
602
603
603
604
604
605
605
606
606
607
607
608
608
609
609
610
610
611
611
612
612
613
613
614
614
615
615
616
616
617
617
618
618
619
619
620
620
621
621
622
622
623
623
624
624
625
625
626
626
627
627
628
628
629
629
630
630
631
631
632
632
633
633
634
634
635
635
636
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
699
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
799
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
899
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
```

```

JS App.js 3, M ×
src > JS App.js > [0] App
7 const App=({})=>{
8   return <div className="App">
9     <AppHeader title="World Wide Books" />
10    <div className="Screen">
11      <BookListScreen screenTitle="Books" />
12    </div>
13    <AppFooter />
14  ;
15}

JS withTitle.js 1, M ×
src > hoc > JS with-title.js U > JS withVisibility.js U × JS If.js 1, U ...
3 const withTitle = ( TargetComponent, defaultTitle='true' ) => {
4
5   //HOC component
6   return (props)=>{
7     let title=defaultTitle;
8     // console.log('using withTitle...'); 
9     if(props.title==undefined){
10       console.log('applying props.title',props.title);
11       title=props.title;
12     }
13
14     return (
15       <TargetComponent {...props} />
16     );
17   };
18
19 }

JS withVisibility.js 1, M ×
src > hoc > JS with-title.js U > JS withVisibility.js U × JS If.js 1, U ...
3 const withVisibility = ( TargetComponent, defaultVisibility=true ) => {
4
5   //HOC component
6   return (props)=>{
7     let visibility=defaultVisibility;
8     // console.log('using withVisibility...'); 
9     if(props.visibility==undefined){
10       console.log('applying props.visibility',props.visibility);
11       visibility=props.visibility;
12     }
13
14     return (
15       <TargetComponent {...props} />
16     );
17   };
18
19 }

JS BookListScreen.js 1, M ×
src > screens > JS BookListScreen.js > ...
2 import {withTitle} from '../hoc/with-title';
3 import withVisibility from '../hoc/with-visibility';
4
5 const BookListScreen=({})=> {
6   //TODO: Initialize Here
7   const books=[...];
8
9   return (
10     <div className='BookListScreen'>...
11     </div>
12   );
13
14 }
15
16 //export default BookListScreen;
17 export default withTitle( withVisibility(BookListScreen,"Book List Screen") );
18
19
20
21
22
23

```

Our HOC is adding reusable title component

HOC also uses a special prop "screenTitle" which is not known or handled by actual target component

Visibility HOC for a component

- Handles a prop called 'visibility'
- If visibility is false component is not displayed
- It is like our if component but works internally with HOC

```

JS BookDetailsScreen.js 1, M ×
src > hoc > JS with-title.js U > JS withVisibility.js U × JS If.js 1, U ...
3 const withVisibility = ( TargetComponent, defaultVisibility=true ) => {
4
5   //HOC component
6   return (props)=>{
7     let visibility=defaultVisibility;
8     // console.log('using withVisibility...'); 
9     if(props.visibility==undefined){
10       console.log('applying props.visibility',props.visibility);
11       visibility=props.visibility;
12     }
13
14     return (
15       <TargetComponent {...props} />
16     );
17   };
18
19 }

JS withVisibility.js 1, M ×
src > screens > JS BookListScreen.js > ...
2 import {withTitle} from '../hoc/with-title';
3 import withVisibility from '../hoc/with-visibility';
4
5 const BookListScreen=({})=> {
6   //TODO: Initialize Here
7   const books=[...];
8
9   return (
10     <div className='BookListScreen'>...
11     </div>
12   );
13
14 }
15
16 //export default BookListScreen;
17 export default withVisibility( BookListScreen,"Book List Screen" );
18
19
20
21
22
23

```

```

JS App.js 3, M ×
src > JS App.js > [0] App
7 const App=({})=>{
8   return <div className="App">
9     <AppHeader title="World Wide Books" />
10    <div className="Screen">
11      <BookListScreen visibility=[true] screenTitle="Books" />
12    </div>
13    <AppFooter />
14  ;
15}

```

- We can apply multiple HOC on our component to add different features

- Toggles visibility based on "visibility" props
- If 'visibility' is not supplied defaults to true
 - Makes sure you can use your component without any problem
- Target component doesn't know or handle 'visibility' prop

Quiz

- What if we change the order of HOC here?

```
export default withTitle( withVisibility(BookListComponent),"Book List" );
```

Answer

- It will work but title will always be visible
- withVisibility is toggling the visibility of BookListComponent that doesn't have title
- Title is being applied to component that supports visibility
 - It is a child component to titled component

Updating state via HOC

Monday, January 17, 2022 11:45 AM

Consider Multiple components in our application needs current date/time

- Date should be updated every second
- We will need to write the logic for updating date

The screenshot shows a browser developer tools window with two tabs: 'Elements' and 'Console'. The 'Console' tab displays the following error message:

```
Uncaught TypeError: Cannot read properties of undefined (reading 'toLocaleTimeString')
at BookListScreen (BookListScreen.js:125)
at renderNodeHook (react-dom.development.js:14985)
at mountIndeterminateProps (react-dom.development.js:17831)
at beginWork (react-dom.development.js:19849)
at Object.invokeOnMainCallback (react-dom.development.js:3945)
at Object.invokeOnMainCallback (react-dom.development.js:3994)
at invokeGuardedCallback (react-dom.development.js:4066)
at beginWork$1 (react-dom.development.js:33964)
at performUnitOfWork (react-dom.development.js:22775)
```

The code in BookListScreen.js is as follows:

```
const BookListScreen = ({ now }) => {
  //TODO: Initialize Here
  const books = [...]
  //let date=new Date();
  return (
    <div className='BookListScreen'>
      <h3>{now.toLocaleTimeString()}</h3>
      <ul>
        {books.map(book => (
          <li key={book.isbn}>
            <img src={book.cover} alt={book.title} />
            {book.title}
          </li>
        ))}
      </ul>
    </div>
  );
}

export default BookListScreen;
```

- Parent is not passing the date object**
 - Since the date should be updated every second, this logic must be written in either
 - Parent
 - Child
- Can be create a design where it need not be written in either parent or child

HOC sits between Parent and Child

- Parent actually uses HOC
- HOC using Actual target component

- This new auto updated property is available to my component, thanks to HOC

```
const BookListScreen = ({ now }) => {
  //TODO: Initialize Here
  const books = [...]
  //let date=new Date();
  return (
    <div className='BookListScreen'>
      <h3>{now.toLocaleTimeString()}</h3>
      <ul>
        {books.map(book => (
          <li key={book.isbn}>
            <img src={book.cover} alt={book.title} />
            {book.title}
          </li>
        ))}
      </ul>
    </div>
  );
}

//export default BookListScreen;
export default withVisibility(
  withTimer(
    BookListScreen,
    "Book List Screen"
  )
);
```

- BookListScreen is HOC enhanced component

- Timer injects a new property which keeps updating every second

The screenshot shows a browser developer tools window with three tabs: 'Components', 'Timer.js', and 'render'. The 'Components' tab shows the 'BookListScreen' component. The 'Timer.js' tab contains the following code:

```
const TargetComponent=this.props.component;
//console.log('TargetComponent',TargetComponent);

return <TargetComponent {...this.props} now={this.state.time} />
```

The 'render' tab shows the rendering logic:

```
const TargetComponent=this.props.component;
//console.log('TargetComponent',TargetComponent);
//HOC component
let TimerComponent= (props)=>{
  return <Timer {...props} component={TargetComponent} />
}
```

The 'App.js' tab shows the final rendered component:

```
return <div className="App">
  <AppHeader title="World Wide Books" />
  <div className="Screen">
    <BookListScreen visibility={false} screenTitle="Books" />
  </div>
  <AppFooter />
</div>
```

- We are Not passing "now" component to our BookListScreen
 - We are not managing setInterval either in
 - App
 - BookListScreen

[Full Source Code](#)

Full Source Code

```
import {Component} from 'react';

export class Timer extends Component {
  state = [
    time: new Date()
  ]

  componentDidMount() {
    console.log('timer based component mounted');
    this.timer = setInterval(() => {
      this.setState({time: new Date()});
    }, 1000);
  }

  componentWillUnmount() {
    console.log('timer based component unmounted');
    if(this.timer) {
      clearInterval(this.timer);
    }
  }

  render() {
    const TargetComponent = this.props.component;
    return <TargetComponent {...this.props} now={this.state.time} />
  }
}

export const withTimer = (TargetComponent) => {
  //HOC component
  let TimerComponent = (props) => {
    return <Timer {...props} component={TargetComponent} />
  }

  return TimerComponent;
};
```

- All the properties actually supplied for the TargetComponent

- Additional property supplied.

Take away

- HOC can supply new information as 'prop' to my component
- This information can be completely different from what client is passing
- We just get additional information
- Those information may also get updated behind the scene
 - HOC may be using some state

Same Logic Different UI

Monday, January 17, 2022 1:02 PM

Unoptimized Approach

```
class BookListAsTable {  
  state={books:[]}  
  componentDidMount(){  
    this.setState({books: await  
      fetchBooks()})  
  }  
  
  render(){  
    return <BookTable  
      books={this.state.books}  
    />  
  }  
}
```

```
class BookListAsGrid {  
  state={books:[]}  
  componentDidMount(){  
    this.setState({books: await  
      fetchBooks()})  
  }  
  
  render(){  
    return <BookGrid  
      books={this.state.books}  
    />  
  }  
}  
  
What differs here is  
what to do when we  
have the book
```

- Sometimes we need to display same information differently.
- Example
 - Books as a Table
 - Books as a Grid
- Both Component will have same
 - Lifecycle method to fetch data from the server
 - Both need to work with same data that is fetched from the server
- Both will display same set of data differently

Problem — Redundant Logic

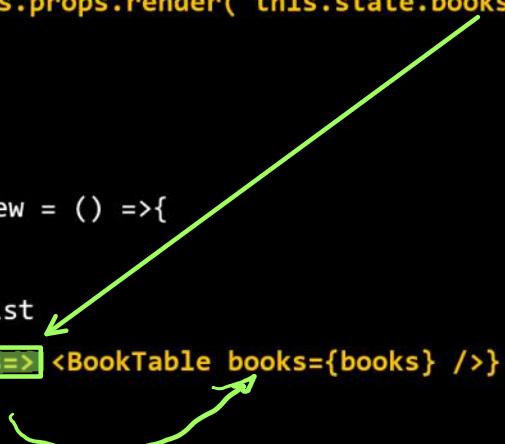
Render Prop Pattern

Monday, January 17, 2022 1:09 PM

Phase #1 Create Logic Component

- Component should contain the logic to fetch the data
- This is a redundant logic.
- Component shouldn't know how to render the UI
- Render Logic will be Passed as **function() in prop**

```
class BookList {  
    componentDidMount(){  
        fetchBooks()  
    }  
    render(){  
        return this.props.render( this.state.books);  
    }  
}  
  
const BookTableView = () =>{  
  
    return <BookList  
        render={ books=> <BookTable books={books} />}  
    />  
}
```



Phase 1

1. Create a single BookList component
 - ▶ Should contain the common business logic for data fetching
2. Calls the render function passed as props to it

Phase 2

1. Create component
2. Wrap BookList component
3. Define the render function

• Phase 2

1. Client will call my BookList component
 - They will Pass the Logic to render the function
 - Here we get **data as function parameter** and not as props

HOC vs RenderProp

Monday, January 17, 2022 1:23 PM

- Both help us separate redundant logic from presentation logic
 - HOC can inject new prop to my component
 - Render Prop injects a UI logic in a generic component that knows data logic

```
<Timer>  
    <h3>{now.toLocaleTimeString()}</h3>  
</Timer>
```

- HOC hides the Timer component
 - It injects the 'now' as 'prop'

- Render Prop takes UI element
 - It takes the 'now' as parameter
 - Timer component is visible.

React 16.8 Hooks

Monday, January 17, 2022 2:19 PM

- Applies only to function based components
- Can add new functionality to your Function Components

React Hooks

- ▶ React 16.8 Introduced the concept of hooks to add new functionalities to a React component.
- ▶ Adds functionality to function components
- ▶ Conventional naming `useXXX()`
- ▶ Example of Features
 - ▶ State Management
 - ▶ Life cycle management
- ▶ Now you have lesser reason to create class based components
- ▶ Class based component is not getting deprecated
 - ▶ Just little less useful.

useState

```
import {useState} from 'react';
```

```
export default App =()=>{  
  
  const counter= useState(0);  
  console.log('counter',counter);  
  
  return (  
    <div className="App">  
  
      <h2>Counter App </h2>  
  
      {/* <DeltaButton onChange={change} value={this.state.  
        counter}> */}  
    </div>  
  );  
};
```

- useState takes the initial value for a state
- It returns an array of two items
 - 0 → the state value that may change over time
 - 1 → the function that can change the state
 - It replace this.setState

```
Console 0 Problems 2 React DevTools 0  
Console was cleared  
counter ▶ (2) [0, f bound dispatchAction()]
```

Full Example

```

1 import React, { useState } from "react";
2 import "./styles.css";
3
4 export default App = () => {
5   const counter = useState(0);
6   console.log("counter", counter); 1. The value of the state
7   const value = counter[0];
8   const setValue = counter[1];      2. Function that changes the state
9
10  return (
11    <div className="App">
12      <h2>Counter App </h2>
13
14      <h3>value: {value}</h3>
15
16      <button onClick={() => counter[1](value + 1)}>+</button>
17      <button onClick={()=>setValue(0)}>0</button>
18      <button onClick={() => setValue(value - 1)}>-</button>
19
20    </div>

```

Destructuring

Without Destrcturing	With Array Destructuring
<pre>const xState = useState(init_value); const value=xState[0]; const setValue=xState[1]</pre>	<pre>const [value, setValue] = useState(init_value);</pre>

State Management — Class vs Function

Features	Class Based	Function Based
Availability	From beginning	React 16.8+
Access	As builtin part of Class component this.state/this.setState	Accessed using useState() hooks
Creating State	A single complex state know as this.state <pre>this.state={ next:'0', cells:[null,null,...], winner:null; }</pre>	Multiple simpler state variables. <pre>const [next, setNext] = useState('0'); const [cells, setCells] = useState([null, null, ...]); const [winner, setWinner] = useState(null);</pre> <ul style="list-style-type: none"> Technically we can create a single complex state object even here But that would be actually 'complex'
Changing the state	Function names is fixed <pre>this.setState({...})</pre>	Function name can be anything of your choice. <pre>setNext("X");</pre>

CLASS STATE VS USE STATE

Class based state

```
class Board{
  state={
    next:"0",
    cells:[null,null,null]
  }

  const update=()=>{
    this.setState( { next: "X" } );
    this.setState( {cells:["0",null,null]} );
  }
  render () => <p> {this.state.next} </p>
}
```

useState

```
const Board = () => {
  const { next, setNext } = useState("0");
  const {cells, setCells } =
    useState([null,null,null] )

  const update=()=>{
    setNext( "X" );
    setCells( ["0",null,null]);
  }

  return <p>{next}</p>
}
```

useEffect

Monday, January 17, 2022 3:12 PM

- A hook to provide life cycle methods inside a React Function Component
- Can provide option similar to
 - componentDidMount
 - Called only once
 - componentDidUpdate
 - After every render
 - componentWillUnmount
- useEffect runs after first render of the component
- useEffect is a synchronous function
 - It can perform async job but can't be async itself.

Step 1

componentDidUpdate

```
const MyComponent = (props) => {  
  useEffect( () =>{  
    //write the logic for componentDidMount  
    console.log('component is mounted');  
  });  
  console.log('component is getting rendered...');  
  return <h2>Component Loaded</h2>;  
}  
}
```

1. Direct code inside the function will execute
 - a. console.log('component is getting rendered...');
2. Component will be rendered.
3. useEffect will be called after rendering
 - console.log('component is mounted');
4. If this component changes some state component will be re-rendered.
5. After every render, this useEffect will be called.
 - It works like componentDidUpdate

IMPORTANT!!!

- You should not do anything unconditional in this useEffect

Advanced useEffect design

```
useEffect ( () =>{  
  //run this code if anything in dependency list changes  
}, [ //my dependency list  
]);
```

- We can pass an array of values as second parameter to useEffect
- This use Effect will be called only if any of those dependencies changed after the last update.
- If they don't change this useEffect() will not be called for a second time.

IMPORTANT!!!

- Always use this version
- This is like a guarded componentDidUpdate

componetDidMount

- You can write useEffect with a empty dependency list
- That way useEffect() will be called once
- Since there is a empty dependency list it will not be called again!

```
useEffect ( () =>{
    //componentDidMount
}, [ ]);
```

When It is called	Syntax	Class Lifecycle Equivalent	Remarks
After every render	useEffect(()=>{})	componentDidUpdate	<ul style="list-style-type: none"> • NO dependency list supplied • NOT ADVISABLE
Only if a particular props/state changed	useEffect (()=>{}, [book.isbn]);	componentDidUpdate with condition	<ul style="list-style-type: none"> • If same component is called for a different book • You should use it to reuse component with different data
Only once at the beginning	useEffect(()=>{}, []);	componentDidMount	<ul style="list-style-type: none"> • Called after first render • Not called again till component is unmounted and remounted again

Cleanup function

- Each useEffect can return a function
- If useEffect returns a function it will be called to clean up previous call of useEffect

componentDidUnmount

```
useEffect( ()=>{
    //component did mount logic here
    return () =>{
        //component did unmount logic here
    }
}, [ ]);
```

- 
- This function will be called when component is unmounted.

Cleanup can be associated even with componentDidUpdate

```
useEffect( () => {
    //do the logic you would do when you get a new book with isbn
    return {
        //cleanup logic for previous book that was displayed.
    }
}, [book.isbn])
```

```
useEffect(()=>{
  console.log('book details is updated');
  setTimeout(()=>{
    setTitle(book.title);
  },5000);

  return ()=>{
    setTitle("Loading...");
    console.log('removing book ',book.title);
  }
},[book.isbn]);

console.log('book details is rendering');
```

- This code will run
 - After first render
 - Everytime when book.isbn changes
- This code will run
 - When book.isbn changes
 - Before running the next call of useEffect

Timer

Monday, January 17, 2022 3:51 PM

```
import {useEffect, useState} from 'react';

export const Timer = ({render})=>{
  const [time, updateTime] = useState(new Date());
  useEffect(()=>{
    //componentDidMount -->[] --> no dependency
    const id= setInterval(()=>{
      updateTime(new Date());
    },1000);

    //componentDidUnmount
    return ()=>{
      clearInterval(id);
    }
  },[1]);
  return render(time);
}

export const withTimer= (TargetComponent)=>{
  //HOC component
  let TimerComponent= (props)=>{
    return <Timer render={ now=>(<TargetComponent {...props} now={now}/>) } />
  };
  return TimerComponent;
};
```

- useState adds state management in our own component

- useEffect adds life cycle to function based component

- Render() helps us use the Timer directly in our code and display the actual time in our desired format

- HOC helps us inject 'now' prop to any component

User Input

Monday, January 17, 2022 4:33 PM

- Input contains two important properties
 - Value
 - onValueChange
- We must wire them up with
 - useState

The screenshot shows a React application running on localhost:3000. The UI has fields for 'TITLE' (containing 'The Accursed God'), 'AUTHOR' (containing 'AUTHOR'), and 'COVER'. A tooltip message 'When user types a value it is reflected in the event' with the note 'But not on UI' is displayed over the 'AUTHOR' field. The developer tools (Elements and Console) are open, showing the DOM structure and the console log output for the 'handleValueChange' event. The code for 'LabeledInput.js' is shown on the right, highlighting the 'value' prop and the event handler.

```
const LabeledInput = ({ id, value, label, help, placeholder, onChange, type="text" }) => {
  //TODO: Initialize Here

  if(!label) label =id;
  if(!placeholder) placeholder =label;
  if(!help) help=label;

  const handleValueChange=(e)=>{
    console.log(e.target.id, e.target.value);
  }

  return (
    <div className="form-group">
      <label htmlFor={id}>{label}</label>
      <input
        type={type}
        className="form-control" aria-describedby="emailHelp"
        id={id}
        value={value}
        onChange={handleValueChange}
        placeholder={placeholder} />
      <small id={`${id}_help`} className="form-text text-muted">{help}</small>
    </div>
  );
}

export default LabeledInput;
```

• Textbox's value is bound to 'value' prop
• 'prop' can't be changed.

• Solution

- Bind your input value to some state
- Fire an event to let outsider know value has changed.

Designing Input Element

The code editor shows the 'LabeledInput.js' file with several annotations:

- An annotation points to the 'value' prop in the component props, stating: "User supplied value is the initial value only" and "It will be stored in the state".
- An annotation points to the 'useState' hook call, stating: "Input box value is bound to the state" and "When state will change display will change".

```
import React,{useState} from 'react';

const LabeledInput = ({ id,value,label,help,placeholder,onChange,type="text" }) => {
  //TODO: Initialize Here

  const [value,updateValue]= useState(value);

  if(!label) label =id;
  if(!placeholder) placeholder =label;
  if(!help) help=label;

  const handleValueChange=(e)=>{
    console.log(e.target.id, e.target.value);
    updateValue(e.target.value);
    onChange(id,e.target.value);
  }

}
```

```

        console.log(e.target.id, e.target.value);
        updateValue(e.target.value);
        onChange(id,e.target.value);
    }

    return (
        <div className="form-group">
            <label htmlFor="{id}">{label}</label>
            <input
                type={type}
                className="form-control" aria-describedby="emailHelp"
                id={id}
                value={value}
                onChange={handleValuechange}
                placeholder={placeholder} />
            <small id={`${id}_help`} className="form-text text-muted">{help}</small>
        </div>
    );
}

export default LabeledInput;

```

change

- onChange should update the internal state
 - This way UI will be updated with latest value
- We can call user callback to inform user outside my component that value has changed.
 - They can save this value for future use.

Assignment 16.1

Monday, January 17, 2022 5:25 PM

1. Complete the BookAdd Page with following fields
 - a. Isbn
 - b. Title
 - c. Author
 - d. Cover
 - e. Price
 - f. Rating
 - g. Description
 - h. Tags (comma separated)
 - i. Votes
2. Save the book details in local storage and access them using a service
3. Create UI and service for
 - a. User login
 - b. User registration
4. In App header Membership area should display different value based on user status
 - a. No Logged User
 - i. Shows current menu (without logout)
 - ii. Change the work memebrship to Guest
 - b. Logged in user
 - i. Show user Name instead of Memebr
 - ii. Show following links
 - 1) Profile
 - 2) Favorites
 - 3) Logout
 - c. Just create the links and empty screens.
 - i. No logic neeeded

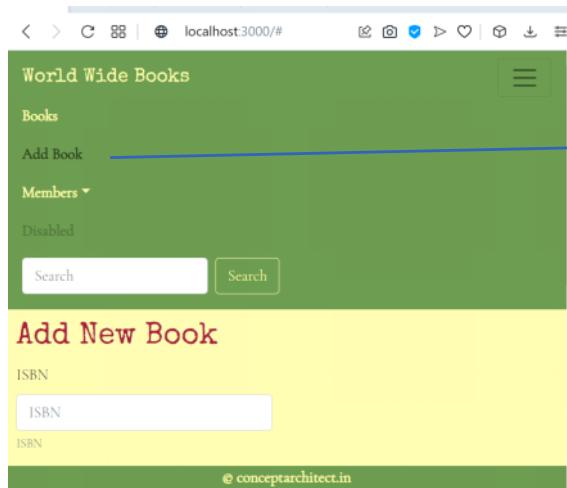
HINT

CSS Challenges

- If you read the header component carefully there is an answer to both questions

1. Move the Membership dropdown to the right hand side like reference implementation
2. In collapse mode if we select any menu item from drop down, it should collapse again. Currently it doesn't

1. Search bard is currently right aligned.
 - a. How?
2. The menu collapses when we select menu button.
 - a. How?



- If I select something it should auto-collapse
- Currently not happening

Hooks Summary

Tuesday, January 18, 2022 9:30 AM

- Hooks typically have **use** prefix
 - useState
 - useEffect
 - useXyz
- They can be called only from
 - A function component only
 - Function name must be upper case
 - Another hook
- They must be called unconditionally.
 - There shouldn't be any if statement or other statement that may not make the hook run on some condition.
 - You can't create a hook inside
 - use effect
 - nested function
 - If statements

How have we Reacted?

Tuesday, January 18, 2022 9:26 AM

Problem #1

- We are managing transition (navigation) between our screens using App 'state'
- The entire mechanism is required in most of the application.
- But we still need to create this design for every application
- Since the navigation is based on 'state' if we refresh on any screen we will return to default screen
- URL doesn't change on the screen

The screenshot shows a web browser window with a dark blue header bar. The title bar says "A Typical Single Page Application". Below the header, the URL is "localhost:3000/book/details/9781393495574". The main content area has a dark background with a red sidebar on the right. At the top left, there's a navigation bar with links: "Book's Web", "Books", "Add Book", and "Authors". A search bar with the placeholder "Search" and a user profile "Vivek Dutta Mishra" are also present. The main content displays a book cover for "The Accursed God" by Vivek Dutta Mishra. The cover art features a figure in a dynamic pose against a red and orange background. To the right of the cover, a yellow button labeled "Price:" contains a five-star rating icon. A red callout box labeled "Custom Rating component" points to this button. Below the book cover, the word "Synopsis" is followed by a detailed paragraph about the book's plot and history. At the bottom of the page, there's a social sharing section with a green button that says "You are screen sharing" and a "Stop Share" button. A green arrow points from the text "URL doesn't change on the screen" to this sharing section.

- Typically if user types this url on a new browser, they should be taken to this screen
 - Use may bookmark their favorite books by their URL

IMPORTANT!

- These URL's are not present on our server.
- Only React knows about them.

Problem #2

- All our application data is present in "App"
 - Books
 - Selected book
 - User
- App component doesn't need any of those data
 - It holds it for other components
- We can consider App to be a data store
 - Is it the right usage of App?

React Router

Tuesday, January 18, 2022 9:39 AM

- An external third party library downloadable for NPM
- Not part of core React setup.
- There are many third party libraries that are present and can be used
- We will use "react-router-dom" which is the most popular of them.

Official Documentation

<https://reactrouter.com/docs/en/v6/getting-started/installation>

Installation

\$ npm install react-router-dom@6

From <<https://reactrouter.com/docs/en/v6/getting-started/installation>>

IMPORTANT!

- THERE ARE OFTEN BREAKING CHANGES IN MOST WEB API TODAY
- THEY CHANGE FREQUENTLY
- YOUR CODE MAY NOT WORK WITH THE LATEST UPDATE
 - YOU MAY STICK TO A OLD STABLE VERSION THAT WORKS FOR YOU
 - YOU MAY SWITCH TO NEW VERSION AND CHECK THE DOCUMENTATION

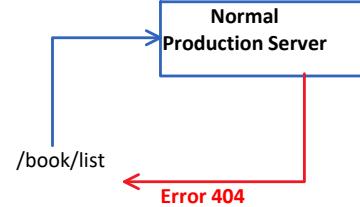
Routing and Server Side support

Tuesday, January 18, 2022 10:18 AM

- React Routing is based on "URL"
 - It displays different screens based on different URLs
- React Routing is client side feature
 - React internally changes the URL making user believe that we are on a different screen
 - It also changes the history of browser so that back button can work

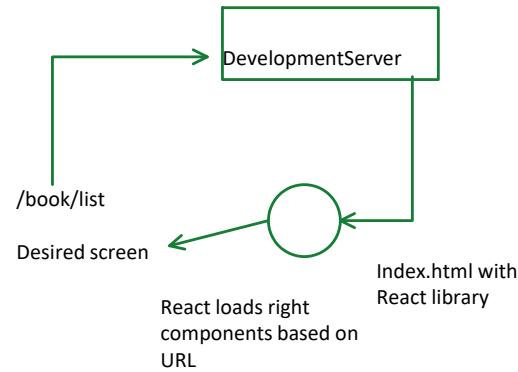
Problem

- If we directly type the URL on a new browser window/tab it will go to server directly
- Server doesn't know about these URLs
 - So server will naturally return 404 and not the desired screen.



How is our React Development server is returning the right page!

- We are using a special development server for React Development
- This server is configured to return "index.html" for all URL's not found on the server.
- Index.html loads React library
- React Router checks the incoming URL and displays the right screen



How will React work on production Server

1 Option#1 — configure production server

- We can configure a production server to return "index.html" as a catch all route
- Different server has different way to configure which is available in server manual
- A good start may be to google search
 - "configure IIS for SPA"
 - "configure apache for SPA"
 - "configure IIS for React routes"
- We will learn NodeJS setup later.

<http://localhost:3000/book/list>

2. Option#2 — configure HashRouter on React side

- Hash router explicitly makes include "index.html"
- No server side configuration is needed
- The URL using hash to separate client side paths.

```
import {HashRouter as Router, Routes, Route } from 'react-router-dom';
```

<http://localhost:3000/index.html#/book/add>

- Server doesn't handle the # part of the URL
- It reads URL before the hash and return the corresponding page
- Now for all URL the page is index.html
- Any server without any configuration can return this page.

Problem

- This URL looks ugly. I don't like it.
- It breaks the magic of SPA
 - SPA is

- SPA for developer
- For client is like a regular MPA app

Avoiding unnecessary Server trips

Tuesday, January 18, 2022 10:41 AM

"A" tag problem

- Normally the "a" tag is handled by the browser
 - Browser sends the request to the server
 - Server again sends the request to the same page
 - React displays the page
- We can realize that it causes an additional network request which was un-needed
- We are already in react app
- React app could have resolved url without going to the server
- But normal 'a' tag is not in 'React's' control

React Router "Link" Component

- We should be React router "Link" component to replace 'a' tag in our code
- Link works in the same way as a
 - It uses 'to' prop rather than 'href' prop.
 - It is internally translated to 'a' that is powered by react scripts

```
<a className="navbar-brand" href="/">{title}</a>
```

```
<Link className="navbar-brand" to="/">{title}</Link>
```

IMPORTANT ERROR

- Link or any other React feature must be included as child of "Router"
- It can't work outside "Router"

```
③ ► Uncaught Error: useHref() may be used only in the context of a <Router> component.  
    at invariant (index.tsx:19)  
    at useHref (index.tsx:353)  
    at LinkWithRef (index.tsx:256)  
    at renderWithHooks (react-dom.development.js:14985)  
    at updateForwardRef (react-dom.development.js:17044)  
    at beginWork (react-dom.development.js:19098)  
    at HTMLUnknownElement.callCallback (react-dom.development.js:3945)  
    at Object.invokeGuardedCallbackDev (react-dom.development.js:3994)  
    at invokeGuardedCallback (react-dom.development.js:4056)  
    at beginWork$1 (react-dom.development.js:23964)  
  
④ ► The above error occurred in the <Link> component:  
  
    at LinkWithRef (http://localhost:3000/static/js/bundle.js:57480:5)
```

SOLUTION

- Wrap all your components inside Router
 - Even the fixed components like Header and Footer
 - They should be inside Router but not inside Routes

```
<Router>
  ... fixed component needing route support like Link

  <Routes>
    ... dynamic routed component here
  </Routes>

  ... fixed components needing route supports like Link
</Router>
```

Will there be a zero network traffic if we use Link tag?

- NO
- There will be network request for
 - Components that are not yet requested
 - React need to use Ajax to bring the component related data for the first time
 - Once a component/screen is requested it is cached in memory and we will not make a second network request for the same screen
 - Data that is to be displayed on the component
 - It may be requested via an api
 - API call will generate network request
 - But that request will be for the data and not the UI

Routing Info

Tuesday, January 18, 2022 11:38 AM

- React Router Includes a few api to control the navation
 1. history
 - Allows you to programmatically move to another route
 - You can use
 - `history.push(url)`
 - `history.pop()`
 - ◆ To move the previous page
 2. match
 - Includes dynamic 'params' passed to the route like
 - `isbn`
 3. location
 - Includes other information like
 - `Url`
 - `Query string`

How to get these information in our component?

Option #1 — hoc

- React route provides it's own hoc to inject these props to our Component

```
//STEP 1
import {withRoute} from 'react-router-dom';

const MyComponent=(props) =>{

  //STEP 3 -> can access route information in it's prop after applying hoc
  ...
}

//STEP 2
export default withRoute(MyComponent);
```

IMPORTANT!!!

- Deprecated (removed) from version 6
- For class based component we may need to create our own HOC.
 - That is beyond scope

Accessing Route Info within our component

Higher Order Component

- Traditional Router mechanism
- Router provides a HOC
 - `withRouter`
- Injects three properties to 'props'

```
import {withRouter} from 'react-router-dom';

const BookDetails = (props) => {
  ...
  return (<div>
    <h1>ISBN: {isbn} </h1>;
  </div>);
}

export default withRouter( BookDetails );
```

Option #2 React hooks

- works on both version 5 and 6 (may not work on older releases) prior of React 16.8
- We have three hooks to access the three core elements
 - `useHistory`
 - `useLocation`

```

    ○ useParams

//Step #1
import {useMatch} from 'react-router-dom';

const MyComponent=(props) =>{

    //Step #2 get the match object
    const match=useMatch();

    //now use the match feature here

}

```

```

import React from 'react';
//import {withRouter} from 'react-router-dom'; //removed in react-router-dom@6
import {useParams,useLocation} from 'react-router-dom'; //rem

import {Timer} from '../components/Timer';
import withVisibility from '../hoc/with-visibility';

const BookDetailsScreen = (props) => [
    //TODO: Initialize Here
    const params=useParams();
    const location=useLocation();
    console.log('params',params);
    console.log('location',location);
]

```

<http://localhost:3000/book/info/12345?info=compact&format=json>

```

params ► {isbn: '12345'}
location ► {pathname: '/book/info/12345', search: '?info=compact&format=json', hash: '', state: null, key: 'default'}

```

Data Management in React

Tuesday, January 18, 2022 12:14 PM

Problem

- All our application data is present in "App"
 - Books
 - Selected book
 - User
- App component doesn't need any of those data
 - It holds it for other components
- We can consider App to be a data store
 - Is it the right usage of App?

Ideal Scenario

- Data should be present in service layer
- All data should be requested via a service object within our application

A Simple Book Service Code

```
class BookService{  
  constructor(){  
    this._load();  
  }  
  
>  _load=()=>{ ...  
  }  
  
  _save=()=> {  
    localStorage.setItem('books', JSON.stringify(this.books));  
  }  
  
  addBook=async (book)=>{  
    await delay(2000);  
    this.books.push(book);  
    this._save();  
  }  
  
  getAllBooks=async ()=>[  
    await delay(5000);  
    return this.books;  
  ]  
  
  getBookByIsbn=async (isbn)=>{  
    await delay(5000);  
    return this.books.find(book=>book.isbn==isbn);  
  }  
}
```

- Note my service would generally be asynchronous
 - You will get the data from some server
- We don't know how long it will take to get the data
 - But we know it will not be immediate
- In the simple service we are simulating the delay
 - In real application delay will be natural.

Problem

- We can't call the service directly in the component body or render function
 - It will work for synchronous function
 - Services are rarely synchronous
- Async functions can't be called directly in the body
- React components are not Async components

Solution

- You should call the asynchronous function componentDidMount or equivalent useEffect
- useEffect can also not be asynchronous
 - You should use .then() promise and update some state

Problem in consuming Service

Tuesday, January 18, 2022 2:54 PM

- Every component need to interact with the service they need
 - What if the service object changes tomorrow?
- Every component still need to maintain some state.
 - Sometimes multiple component use the same component
 - They may be loaded so they don't know when service has updated an object.

Consider the current login system

- Member Menu loads the current user status when the app is loaded in memory
 - That time user was not logged in
- When user logs in, User service creates a 'user' key in the localStorage
 - But localStorage is not a React 'state'
 - How will the MemberMenu know that login has changed?
- We don't have a common Parent who can maintain state.

The screenshot shows a browser window with a 'User Login Screen'. The URL is 'localhost:3000/user/login'. The page has fields for 'email' (vivek@conceptarchitect.in) and 'password' (****). A 'Login' button is highlighted with a green box. To the right, there's a sidebar with 'Members' dropdown, 'Login', and 'Register' buttons. A red arrow points from the 'Members' dropdown to the 'Members' section in the developer tools. The developer tools are open, specifically the Application tab under Storage. It shows Local Storage with a key 'users' containing the value: [{"name": "Vivek Dutta Mishra", "email": "vivek@conceptarchitect.in", "password": "p@ssw..."}]. A green box highlights the 'Storage' tab and the 'users' entry in the list.

- A successful login updates the local storage
- Change in the local storage is not like a state change
 - It doesn't update / re-render other components

Can we maintain a state on a common parent like App?

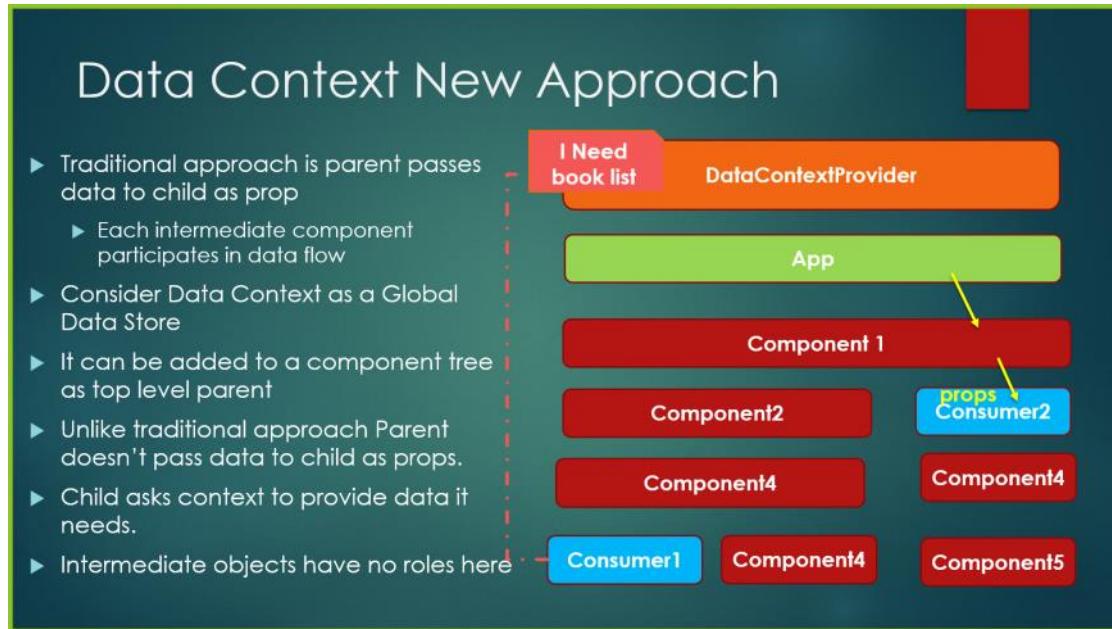
- App need to pass the state to its child
- But Login Screen is not a direct child of App

- It is a grand child
 - Immediate parent of login screen is Route
- Hierarchy
- **App**
 - **BrowserRouter**
 - **AppHeader**
 - MemberMenur
 - **Router**
 - **Route**
 - ◆ **UserLoginScreen**
 - These components will not pass props down.

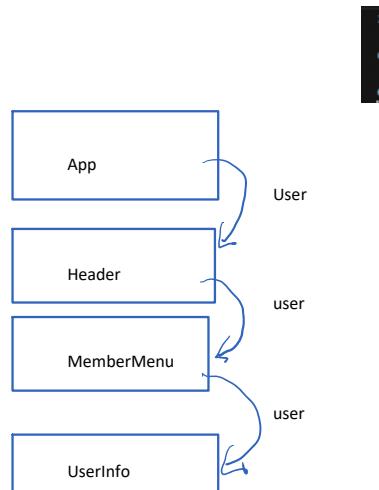
Solution — Global (Stateful) Data Management

React Context

Tuesday, January 18, 2022 3:32 PM

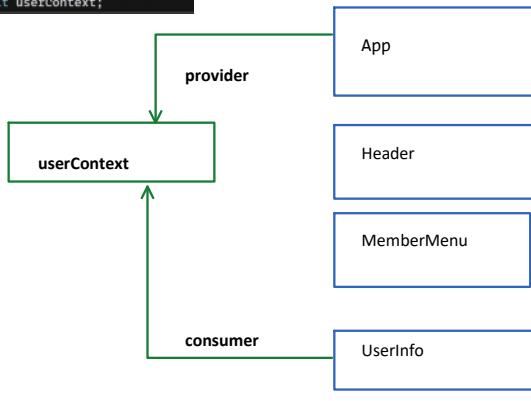


Traditional Flow



```
import { createContext } from "react";
const userContext = createContext();
export default userContext;
```

Context Based Design



```
<userContext.Provider value={user} >
  All components here
</userContext.Provider>
```

```
<userContext.Consumer>
  {(user)}=>{
    return JSX
  }
</userContext.Consumer>
```

Consuming the data

1. Use context.Consumer
2. Use useContext hook

useContext hook

context.Consumer

```
import {useContext} from 'react';
import userContext from "./user-context";

const UserInfo = () => {

  const user=useContext(userContext);

  if(user.loggedIn)
    return <h2>Hello {user.name}</h2>;
  else
    return <h2> Hey Guest.Please Login</h2>;
};
```

```
import userContext from "./user-context";

const UserInfo = () => {
  return (
    <UserContext.Consumer>
      {(user) => {
        if (user.loggedIn) return <h2>Welcome {user.name}</h2>;
        else return <h2>Welcome Visitior</h2>;
      }}
    </UserContext.Consumer>
  );
};
```

User Context is not State

Tuesday, January 18, 2022 3:57 PM

- A context is just a way to get the global data
- It is not a state
- Changes to global data will not automatically render/refresh UI

```
export default function App() {
  let user = { name: "Guest", loggedIn: false };

  const login = (name) => {
    user.name = name;
    user.loggedIn = true;
    console.log(user);
  };

  const logout = () => {
    user.loggedIn = false;
  };

  return (
    <userContext.Provider value={user}>
      <div className="App">
        <Header />
        <button onClick={() => login("Mansi")}>Login User 1</button>
    
```

Change in context value is
not automatically
propagated

We can use context with State

- If value attached to context is a stateful value, changes to the state will update the context
- Remember userContext is a child of App
- If App state changes userContext.Provider will re-rendered
- Remember <userContext.Provider value={user} />
 - value is a prop of Provider

```
export default function App() {
  let [user, setUser] = useState({ name: "Guest", loggedIn: false });

  const login = (name) => {
    setUser({ name, loggedIn: true });
  };
  const logout = () => {
    setUser({ loggedIn: false });
  };

  return (
    <userContext.Provider value={user}>
      <div className="App">
```

Fetching data from Server

Tuesday, January 18, 2022 4:49 PM

- React is a browser application.
 - It has access to
 - **XmlHttp object**
 - **fetch**
 - Both are builtin
 - We have used fetch earlier
- There are third party libraries which provide a lot of features to work with http request

Axios

- Axios provides with several features that makes working with API easier
- We will use axios

Installation

```
npm install axios
```

Assignment 17.1

Tuesday, January 18, 2022 6:00 PM

- Integrate our React App with the End Points of Api Server
- Add Functionality for
 - Add Book
 - Delete Book
 - Add Review
 - Show Review
 - Login
 - Registration
 - Author List
 - Author Add

Challenge

- We need to store authorization token locally
- We must pass authorization token to every request that require authorization such as
 - Add book
 - Add author
 - Delete author
 - Delete book
- How do we pass authorization header?
 - Research Topic
 - Search for : Passing Authorization token in axios request

Data List

Friday, January 21, 2022 8:52 PM

```
<input list="browsers" name="browser">
<datalist id="browsers">
  <option value="Internet Explorer">
  <option value="Firefox">
  <option value="Chrome">
  <option value="Opera">
  <option value="Safari">
</datalist>
```

Reducer Pattern

Saturday, January 22, 2022 11:35 AM

- Reducer is a generic design pattern for maintaining stateful data
- IT IS NOT A PART OF REACT FRAMEWORK
- This idea can work with any framework
- React had no built-in support for Reducer framework before React 16.8
- One of the most popular framework to implement this idea is Redux

Redux

- Redux is an independent library not connected with React
- It can be used with any framework
- It is extensively used for React Application for state management
- There are other thirdparty libraries like
 - Mobx

Important

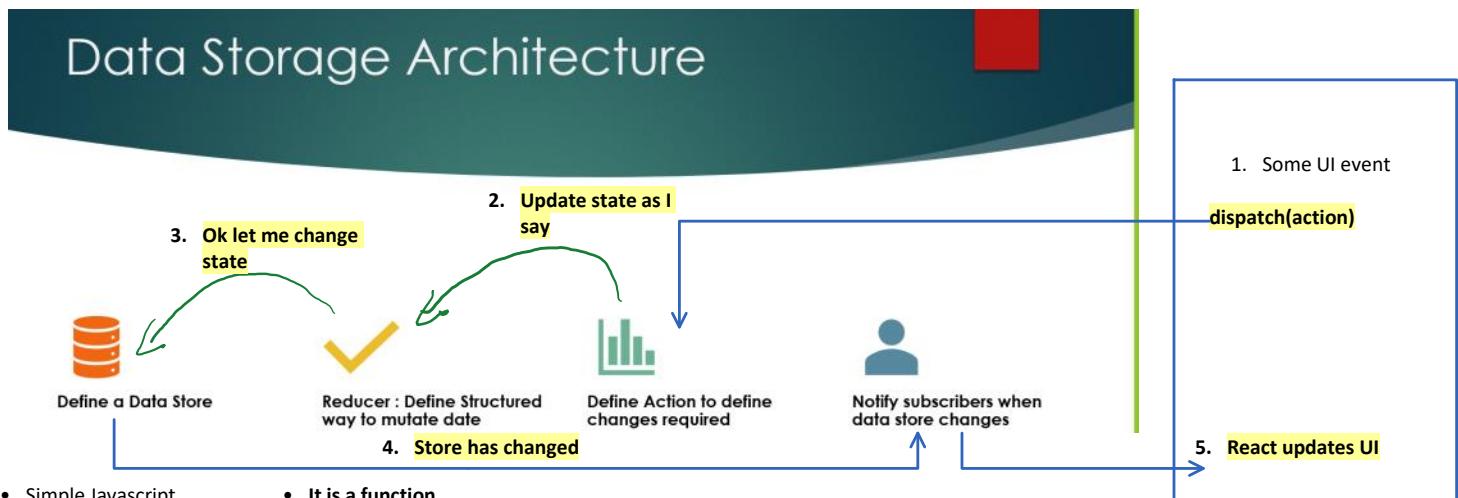
- Redux is still very popular and has lot of features over and above React 16.8 Reducer
- Many large project continue to use Redux even in modern application because of its popularity and large number of features

React 16.8 Reducer

- React introduced the idea of Reducer inspired by Redux library
- Now we don't really need a separate library like Redux in our project
-

Reducer Architecture

Saturday, January 22, 2022 10:39 AM



- Simple Javascript Object
- Immutable in itself

- It is a function
- Changes stored data as per requirement
- Store changes are brought as new object that replaces the old one
- It should be called intention not action
 - Person opinion
- Is an object that includes
 - type: What kind of changes we are looking
 - Any data needed for this purpose
- Action simply tell reducer what is to be done.
 - It doesn't do anything on its own

REACT

Data Storage

- ▶ Defines as a Javascript Object
- ▶ This Object is expected to change over a period of time.
- ▶ Changes should always create a new object.
- ▶ NEVER MUTATE AN EXISTING PROPERTY.
- ▶ Change to this object should be notified to the subscriber.

```
const store={  
  books: [...],  
  selectedBook: [...],  
  user:{name:vivek}  
}
```

- We have a simple in memory data
- Part of this data can be displayed in different UI component.

We may pass our intention

Actions

- ▶ Can also be considered as intent
- ▶ Generally includes a String label

```
const store={  
  books: [...],  
  selectedBook: [...],  
  user: {name:vivek}  
}
```

```
{ type: "USER_LOGOUT" }
```

I want user to logout

Actions

- ▶ Can also be considered as intent
- ▶ Generally includes a String label
- ▶ Defines how data would change.

```
const store={  
  books: [...],  
  selectedBook: [...],  
  user: null <--
```

```
}
```

```
{ type: "USER_LOGOUT" }
```

Action may include information

Actions

- ▶ Can also be considered as intent
- ▶ Generally includes a String label
- ▶ Defines how data would change.
- ▶ May need to supply arguments required to bring about the change.

```
const store={  
  books: [...],  
  selectedBook: [...],  
  user: null
```

```
}
```

```
{type: "USER_LOGIN", arg:{name: "admin" }}
```

- Hey go and login user "admin"

Actions

- ▶ Can also be considered as intent
- ▶ Generally includes a String label
- ▶ Defines how data would change.
- ▶ May need to supply arguments required to bring about the change.

```
const store={  
  books: [...],  
  selectedBook: [...],  
  user: {name: "admin" } <--
```

```
}
```

```
{type: "USER_LOGIN", arg:{name: "admin" }}
```

IMPORTANT!

- Action doesn't make this happen.
- It is like a command from a manager
 - Do logout user
- Actually someone else will do the job of changing store data
 - Reducer

Reducer

Reducers

- ▶ Function That Modifies Store.
- ▶ Receives two parameters
 - ▶ Current Store value
 - ▶ Action
- ▶ Generally handles actions using switch-case
- ▶ Should return current value if no change is required
- ▶ Should change state based on valid action type

```
const store={
  books: [...],
  selectedBook: [...],
  user: null
}

{type: "USER_LOGIN", arg:{name: "admin" } }

const reducer = ( state, action ) =>{
  switch( action.type ){
    case "USER_LOGIN": ...

    case "USER_LOGOUT": ...

    default:
      return state;
  }
}
```

- If user passes an unknown action and you don't want to change the store you must return the default state which you received
- If you want to handle the action you may do it in
 - Switch case
- You must always return a new object that will replace the state completely
 - You have to change all values not only what you are changing
 - This is unlike setState

Reducers

- ▶ Return same object to return no change done
- ▶ Never modify existing state
- ▶ Always create new object if changes are needed.

```
const store={
  books: [...],
  selectedBook: [...],
  user: null
}

{type: "USER_LOGIN", arg:{name: "admin" } }

const reducer = ( state, action ) =>{
  switch( action.type ){
    case "USER_LOGIN": ...
      state.user=action.arg
      return { ...state,
        user:action.arg
      };
    case "USER_LOGOUT": ...

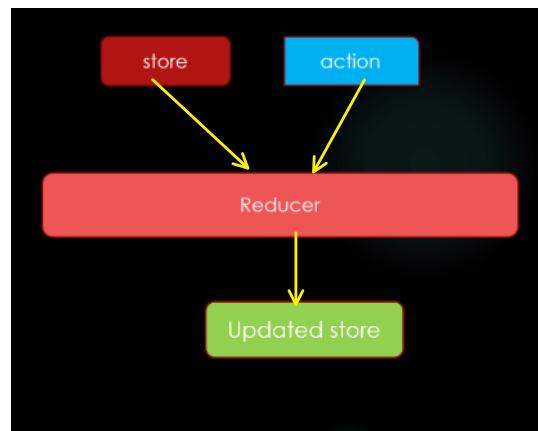
    default:
      return state;
  }
}
```

- destructure current state
- Modify what you need to
- Return the new value that will completely replace old value

What Reducer Does

- It takes
 1. Current store object
 2. Action to tell how to change it
- It returns new State object

- ▶ Reducer should be pure.
 - ▶ It should modify only on the basis of two parameters passed.
- ▶ It Should never talk to external entities like
 - ▶ localStorage
 - ▶ Http Request
- ▶ It should always perform synchronous operation.



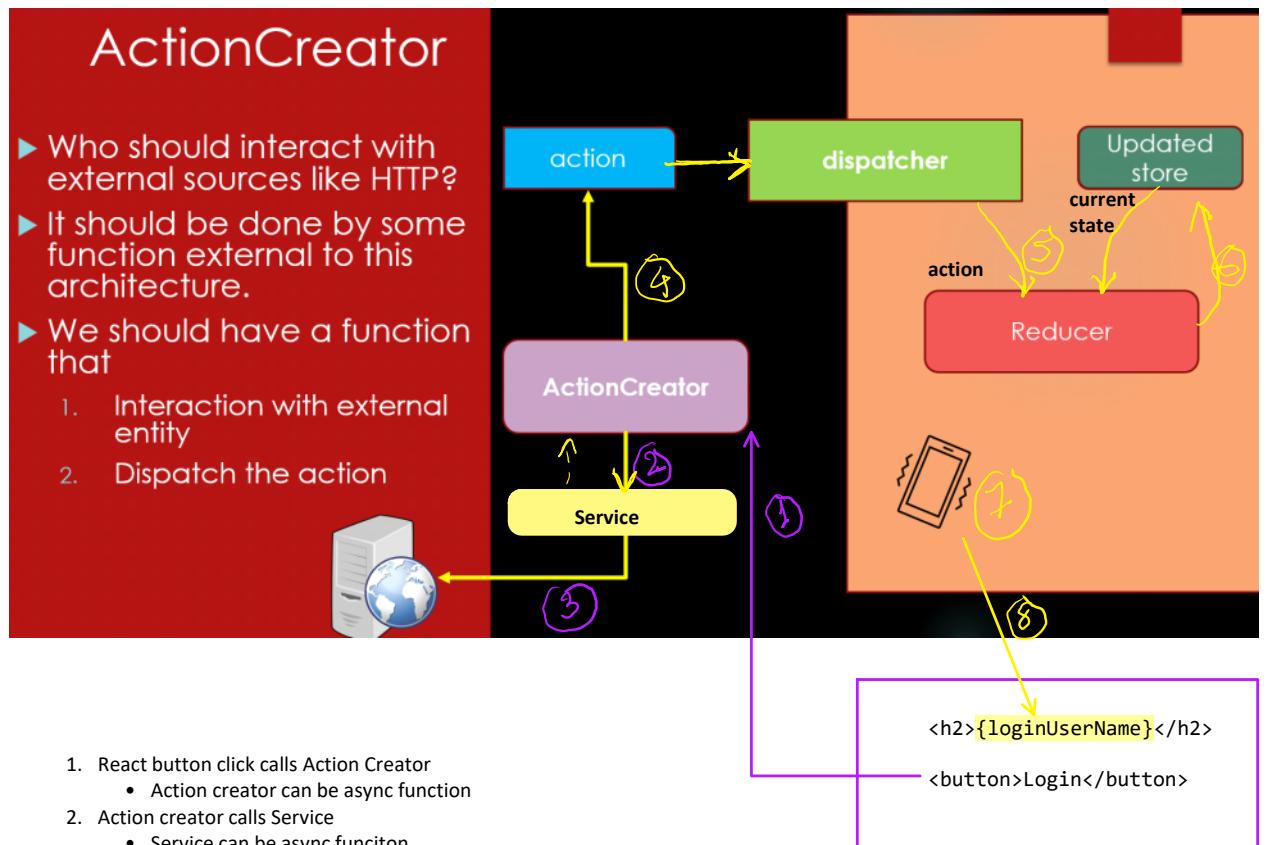
Dispatcher

Dispatcher

- ▶ Dispatcher is the main interface to a store
- ▶ We never call reducer explicitly
- ▶ We dispatch an action
- ▶ Dispatcher calls reducer
- ▶ Reducer updates store
- ▶ Changes in store is notified to subscribers
 - ▶ React component may be a subscriber.

- Think of dispatcher as **an errand boy**.
- Manager wants store to be changed
 - It creates an "action memo"
 - A instruction what is to be done
 - It asks the dispatcher to take this action to reducer
- Think of **Reducer as worker**
 - Reducer receives
 - the original store
 - Action
 - Instruction from the manager
 - It updates the store
 - It notifies everyone that data has been updated

Entire Reducer Pattern Flow



6. Reducer returns a new state
 - This step is synchronous
7. Once the state is updated a notification is sent
 - This is an internal step
8. Your component is update just the same way as state change re-renders.

Action Design common convention

- It is a common convention to design your action with these two keys
- **type:**
 - Type can also be considered as "intent" or "command" or "action name"
 - It describes the broad idea of what we want to do
 - Example
 - "SELECT BOOK"
 - "LOGOUT USER"
 - "LOGIN USER"
- **payload**
 - Many action needs arguments to change the data
 - Remember, reducer can't get the data from external source
 - It must get it from action

Examples of actions

```
{ type: "USER_LOGOUT" } // no extra info is needed

{type: "SET_CURRENT_DATE" } //no extra info is needed
//need to tell who logged in
{type:"USER_LOGIN", user:{ name:"vivek", role:["admin","editor"]} }

//need to tell which book is selected
{type:"BOOK_SELECT", book:{ title:"The Accursed God",...} };
```

IMPORTANT!

- The argument we pass to action can be called anything like
 - user
 - book
 - date
- It is a common convention to call this argument as "payload"
- IT IS HIGHLY RECOMMENDED
- May Redux feature assume that you have named it "payload"

```
{ type: "USER_LOGOUT" } // no extra info is needed

{type: "SET_CURRENT_DATE" } //no extra info is needed
//need to tell who logged in
{type:"USER_LOGIN", payload:{ name:"vivek", role:["admin","editor"]} }

//need to tell which book is selected
{type:"BOOK_SELECT", payload:{ title:"The Accursed God",...} };
```

React 16.8 Reducer

Saturday, January 22, 2022 12:00 PM

- It is based on hooks.

useReducer()

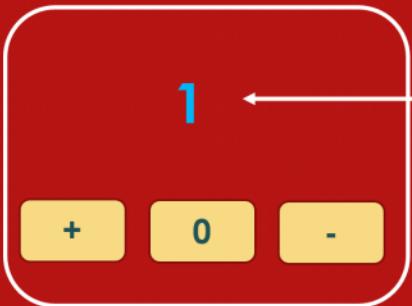
- It takes two parameter
 1. Reducer function
 2. Initial state
 - 3.
- It returns an array of two items
 0. Store value
 - Which automatically gets updated on change
 - Just like state
 1. dispatch function
 - You need to use this function to dispatch actions.

```
const reducerFunction=(store, action) =>{  
  switch(action.type){  
  
    default: return store;  
  }  
}  
  
const MyComponent=()=>{  
  const [ value, dispatch ] = useState( ReducerFunciton, 0);  
  
  ...  
}
```

Reducer in Action

1. How Reducer handles a known/desired action type

Reducer in Action

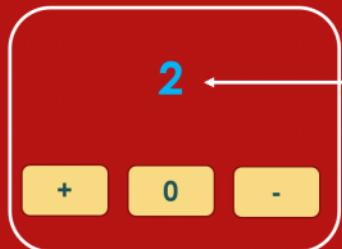


```
import {useReducer} from 'react';
const reducer= (state,action) =>{
  switch( action.type){
    case "INC":
      return state+1; //incremented
    case "RESET":
      return 0; //state set to 0
    default:
      return state; //no change
  }
}

const Counter = () =>{
  const [count,dispatch]=useReducer( reducer, 0);
  return <div><h2>{count}</h2>
    <button onClick={()=>dispatch({type:"INC"})}>+</button>
    <button onClick={()=>dispatch({type:"RESET"})}>0</button>
    <button onClick={()=>dispatch({type:"DEC"})}>-</button>
  </div>
}
```

2. How do we handle unknown actions

Reducer in Action



```
import {useReducer} from 'react';
const reducer= (state,action) =>{
  switch( action.type){
    case "INC":
      return state+1; //incremented
    case "RESET":
      return 0; //state set to 0
    default:
      return state; //no change
  }
}

const Counter = () =>{
  const [count,dispatch]=useReducer( reducer, 0);
  return <div><h2>{count}</h2>
    <button onClick={()=>dispatch({type:"INC"})}>+</button>
    <button onClick={()=>dispatch({type:"RESET"})}>0</button>
    <button onClick={()=>dispatch({type:"DEC"})}>-</button>
  </div>
}
```

Action can pass parameters

- Sometimes (most of the time) we need to pass some information that will help reducer decide the next value.
- This value can be passed inside action object

```
<button onClick={() => dispatch({ intent: "SET", argument: 10 })}>
  Set 10
</button>
<button onClick={() => dispatch({ intent: "SET", argument: 0 })}>
  Set 0
</button>
```

Set 0

</button>

- This value is used by the reducer to handle the task

```
function counterReducer(value, action) {  
  switch (action.intent) {  
    case "INCREMENT":  
      return value + 1;  
    case "DECREMENT":  
      return value - 1;  
    case "SET":  
      return action.argument;  
    default:  
      return value;  
  }  
}
```

How to share Reducer in different component

Saturday, January 22, 2022 12:31 PM

How do we share the reducer state to different components?

- Reducer doesn't have any answer to this question.
- Reducer is a more elaborate 'state' mechanism
 - It doesn't know how to propagate itself to various components
- You may use it by
 1. Passing as props to child component
 - Just like you did to state earlier
 2. Use it along with React Context API

Context vs Reducer

Context

- You can think of a context as better replacement of 'props'
- You don't have to pass it to deeply nested hierarchy.

Reducer

- You can think of reducer as a better replacement of "state"

State + Prop Combination

- Remember we always used state and prop as combination
- App contained 'state'
- It passed the state as 'props' to its children

Reducer+Context combination

- We create reducer and pass it as a value to the context
- Any one can access the reducer values from the context.

Creating Sharable Reducer using context

1. Define your Reducer
2. Define your Context
3. Create special Provider component to connect Reducer, Context and React Component as Children
 - Initializes Reducer
 - Creates Context Provider
 - Passes Children
 - Passes state and dispatch as value to Provider
4. Create a hook to use our context

```
import { useReducer } from 'react';
import { createContext } from 'react';
import { useContext } from 'react';

const userReducer = (state,action)=>{
  ...
};

export const UserContext= createContext();

export const UserContextProvider = ( {children} ) =>{
  const [user, dispatch] = useReducer(userReducer,null);

  return <UserContext.Provider value={ {user,dispatch} }>
    {children}
  </UserContext.Provider>;
}

export const useUserContext= () =>{
  return useContext(UserContext);
}
```

1. Create Reducer Function
2. Create Context
 - At this point context and Reducer have no relationship
3. Create a special component
 - This component has four Key jobs
4. Set up reducer using useReducer
 - Remember you can call a hook only in
 - A React component
 - Other hook
2. Wrap and Replace UserContext.Provider
 - My component creates UserContext.Provider
 - So React App will use my component as Provider.
3. My component as a provider will wrap my React application

Provider.

3. My component as a provider will wrap my React application
 - Now every component can get whatever value we are passing
4. We can pass
 - reducer state
 - dispatch
 - Anything else we like to pass.

User Context v1

Saturday, January 22, 2022 1:12 PM

Step 1 Create User Provider

- Create the UserContextProvider
- It includes
 - userContext
 - userReducer
 - Store
 - Dispatch
- We are creating A UserContextProvider component
- It is including two things from the reducer
 - 'user' state
 - 'dispatch' function

```
export const UserContextProvider = ({ children }) => {
  const [user, dispatch] = useReducer(userReducer, null);

  return (
    <UserContext.Provider value={{user, dispatch}}>
      {children}
    </UserContext.Provider>;
};
```

Step #2 Wrap our Application in User Provider

```
export default function App() {

  return (
    <UserContextProvider>
      <div className="App">
        <Header />
        <Login />
        <Logout />
      </div>
    </UserContextProvider>
  );
}
```

Step 3 Anyone who needs use the value from the context can use it

- We need to call useContext function and pass the userContext Object
- useContext will return
 - user
 - dispatch
- We can destructure this object to get what we need.

```

const UserInfo = () => {
  const {user} = useContext(userContext); // {user, dispatch}
  if(!user)
    return <h2>Welcome Guest</h2>
  else
    return <h2>Welcome Back {user.name}</h2>
};

```

Step 4 Updating Reducer data

- The context includes "dispatch" function
- We can use this to dispatch new data

```

const Login = () => {
  const [username, setUsername] = useState("");
  const [password, setPassword] = useState("");
  const [error, setError] = useState('');
  const {dispatch} = useContext(userContext); // {user, dispatch}

  const handleLogin = () => {
    if(username && username === password) {
      const user = {name: username};
      dispatch({type: UserActions.LOGIN, payload: user})
      setUsername('');
      setPassword('');
      setError('');
    } else {
      setError('Invalid Login!!!');
    }
  }
}

```

```

const Logout = () => {
  const {dispatch} = useContext(userContext);

  const handleLogout = () => {
    dispatch({type: UserActions.LOGOUT});
  }

  return <button onClick={handleLogout}>Logout</button>;
}

```

UserContext V2

Saturday, January 22, 2022 1:23 PM

Motivations

- We have redundant code

```
const { user } = useContext(userContext); // {user, dispatch}
```

- We have to call
 - useContext
 - Pass
 - userContext

Solution

- We can create our own hook to simplify this code

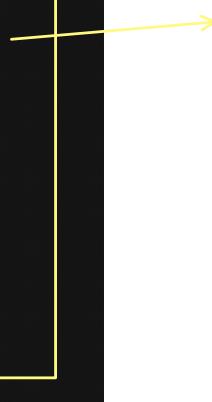
```
export const useUserContext = () => useContext(userContext);
```

Main Problem

- My component still need to
 - Call the business logic
 - Dispatch the action
- UI shouldn't have too much of logic.

```
/*
const { dispatch } = useContext(userContext); // {user, dispatch}

const handleLogin = () => {
  if (username && username === password) {
    const user = { name: username };
    dispatch({ type: UserActions.LOGIN, payload: user });
    setUserName("");
    setPassword("");
    setError("");
  } else {
    setError("Invalid Login!!!");
  }
};
```



- It is a business logic or action creator calling business logic
- It shouldn't be present in our component
- A component shouldn't dispatch
- An action creator should dispatch.

Solution

- We can write separate action creators
- We can pass those action creators using context

Introducing Status

- We can introduce status to our reducer
- It can handle async functions status like
 - Waiting
 - Success
 - Error
- Our store now looks like

```
{
  user:{ name:'Vivek', roles:['admin','manager'],
  status:{type:'success' }
}

{
  user:null,
  status:{type:'error',message:'invalid credentials' }
}
```

```
// structure of mv.state is {user:{}|status:"waiting|success|error"}
const userReducer: (state: any, action: any) => any
export const userReducer = (state, action) => {
  switch (action.type) {
    case UserActions.LOGIN:
      return {...state, user:action.payload};
    case UserActions.LOGOUT:
      return {...state, user:null};
    case UserActions.STATUS:
      return {...state, status:action.payload};
    default:
      return state;
  }
};
```

- Login changes 'user' key only
- Logout changes 'user' key only
- Status changes 'status' key only
- All of them use spread operator to keep other values unchanged.

Designing Action Creators

- An action creator should be an async function
- It should return a action at the end of successful call
- It should throw an error incase of failure
- The idea here is that all logic should be out of the component
- Action creator will call the service and return action
- We will dispatch the action using a closure

```
export const login=async(username, password)=>{
  let user=await UserService.login(username,password);
  return {type:UserActions.LOGIN, payload:user};
}

export const logout=async()=>{
  await UserService.logout();
  return {type:UserActions.LOGOUT};
}

export const checkLogin=async()=>{
  let user=await UserService.checkLogin();
  return {type:UserActions.LOGIN,payload:user}
}
```

- Why do we need two separate functions?
 - Service
 - Action Creator
- Service is a plain javascript object.
 - It doesn't know React
 - It doesn't know Reducer
- Action Creator is a function
 - That returns action object
 - It knows the idea of reducer
 - Action Creator exists because services don't know the actions

How will action Creator Work?

Step#1 --> call action creator function

- It should return an action

Step #2 --> dispatch action returned by action creator

- we need a dispatcher

```
const handleLogin=async(){
  dispatch({type:"STATUS", payload:{ type:"WAITING"}});
  try{
    const action=await login(username,password);
    dispatch(action);
    dispatch({type:"STATUS", payload:{ type:"SUCCESS"}});
  }catch(error){
    dispatch({type:"STATUS", payload:{ type:"ERROR",
      message:error.message}});
  }
}
```

- We will need to write a similar logic for every actions

Solution

Take all actions in an action object

```
//create a list of actions
const actions = {
  login,
  logout,
  checkLogin
};
```

- These are the function which are calling our service and returning an action
- These function don't handle
 - Try-catch error
 - Waiting-success status

Apply Closoure to make them dispatchable

```
const dispatchable = {};
for (let actionName in actions) {
  let actualAction=actions[actionName];
  let dispatchableAction= (...params)=>{
    return async ()=>{
      dispatch({type:UserActions.STATUS,payload:{type:"WAITING"}});
      try{
        let action=await actualAction(...params);
        dispatch({type:UserActions.STATUS,payload:{type:"SUCCESS"}});
      }catch(error){
        dispatch({type:UserActions.STATUS,payload:{type:"ERROR",
          message:error.message}});
      }
    }
  dispatchable[actionName]=dispatchableAction;
}
```

- We create a new set of function under dispatchable

```
dispatchable:{
  login: closureFunctionCallingOriginalLogin,
  logout:closureFunctionCallingOriginalLogout,
  checkStatus:closureFunctionCallingOriginalCheck
  Status
}
```

How

1. Create a closure called dispatchable function
2. It return an async function that does the following work
 1. It dispatches waiting status
 2. Calls our original function in a try-catch block
 - We pass all parameter closure received to actual function
 3. If original function is success
 - i. It dispatches the action returned
 - ii. It dispatches success status
 4. If original function throws error
 - i. It dispatches 'error' status with message

Step 3 Add all the dispatchable actions to context

```
return (
  <UserContext.Provider value={if user dispatch [ dispatchable ]}>
```

- Now we will not be needing dispatch function directly
- But in case someone wants to they can

```

return (
  <userContext.Provider value={{ user, dispatch, ...dispatchable }}>
    {children}
  </userContext.Provider>
);

```

- Now we will not be needing dispatch function directly
- But in case someone wants to they can

Step 4 call the dispatchable function from React

Requirement #1 Modification

- Create your own hook to return user context
- A hook is any function that
 1. Begins with use...
 2. Calls any other hook

```

export const useUserContext=()=>{
  return useContext(userContext);
};

```

Back to React

- Now We can access and print user name and roles

```

import { useUserContext } from "./user-context";

const UserInfo = () => {
  //const { user } = useContext(userContext); // {user, dispatch}
  const { user } = useUserContext();
  console.log("user", user);
  if (!user) return <h2>Welcome Guest</h2>;
  else
    return (
      <div>
        <h2>Welcome Back {user.username}</h2>
        {user.roles.map((role) => (
          <span key={role} className="role">
            {role}
          </span>
        ))}
      </div>
    );
};

```

- We can access "user" object from the context

We can access status object from the context

```

const Status = () => {
  const { status } = useUserContext();

  const style = {
    color:
      status.type === "ERROR"
        ? "red"
        : "green"
  };

```

- Status object

```

const style = {
  color:
    status.type === "ERROR"
      ? "red"
      : status.type === "WAITING"
      ? "blue"
      : "green"
};

return (
  <p style={style}>
    {status.type} {status.message}
  </p>
);
};

```

- Status object

We can call the dispatchable action

```

const Logout = () => {
  const { logout, user } = useUserContext();
  if (!user) return null;
  return <button onClick={logout}>Logout</button>;
};

```

- We can call the dispatchable actions here

```

const Login = () => {
  const [username, setUsername] = useState("");
  const [password, setPassword] = useState("");

  const { login, user } = useUserContext(); // [user, dispatch]
  if (user) return null;
  //console.log('x', x);

  return (
    <div>
      <input
        type="text"
        placeholder="user"
        value={username}
        onChange={(e) => setUsername(e.target.value)}
      />
      <input
        type="password"
        placeholder="password"
        value={password}
        onChange={(e) => setPassword(e.target.value)}
      />
      <button onClick={() => login(username, password)}>Login</button>
    </div>
  );
};

```

IMPORTANT

- We can call some initializer Function at the startup to trigger initialization of store based on external data

```
export default function App() {
  const { user, checkUser } = useUserContext();
  useEffect(() => {
    checkUser(); ←
  }, []);
  return (

```

Book App Reducers and Constants

Saturday, January 22, 2022 4:28 PM

```
//handles a list of books
export const booksReducer=(books=[], action)=>{

  switch(action.type){

    case BookActions.BOOK_LIST_SELECT: //{type: BookActions.BOOK_LIST_SELECT, payload:[book1,book2,book3]}
      return action.payload;

    case BookActions.BOOK_ADD: //{type: BookActions.BOOK_ADD, payload:book}
      return [...books,action.payload];

    case BookActions.BOOK_DELETE: //{type: BookActions.BOOK_DELETE, payload:isbn}
      return books.filter( b=> b.isbn!==action.payload);

    default: return books;
  }
}
```

Book App Reducer and Constants

Saturday, January 22, 2022 4:28 PM

```
//handles a list of books
export const booksReducer=(books=[], action)=>{
  switch(action.type){
    case BookActions.BOOK_LIST_SELECT: //{type: BookActions.BOOK_LIST_SELECT, payload:[book1,book2,book3]}
      return action.payload;

    case BookActions.BOOK_ADD: //{type: BookActions.BOOK_ADD, payload:book}
      return [...books,action.payload];

    case BookActions.BOOK_DELETE: //{type: BookActions.BOOK_DELETE, payload:isbn}
      return books.filter( b=> b.isbn!==action.payload);

    default: return books;
  }
}
```

- Manages an array of all books
1. Book_LIST_SELECT
 - Stores the list of all books
 - Will be called at the start of application
 - May be to replace all book with the search list
 - Gets a list of book as the payload
 2. BOOK_ADD
 - Adds a new book to the existing list of books
 - Gets book as the payload
 3. BOOK_REMOVE
 - Removes a single book from the list of books
 - Gets isbn number as the payload

Create Action with dispatch

Saturday, January 22, 2022 5:06 PM

```
export const getAllBooks= ()=> {
  return async dispatch=>{
    //actual logic here
  }
}
```

- getAllBooks is a closure function.
- It is a synchronous function that immediately returns an inner function
- The inner function is asynchronous and takes dispatch parameter
- It can dispatch any message whenever it likes

```
export const getBookByIsbn=(isbn)=>async dispatch=>{
}
```

- getBookByIsbn is a closure function
- Outer function takes business parameter (isbn) and return another function immediately
 - Inner function takes dispatch
 - It can dispatch asynchronously

How do we call the function

```
const BookListComponent=()=>{
  const {dispatch} = useBookContext();
  useEffect(()=>{
    getAllBooks()(dispatch);
  },[]);
}

const BookDetailsComponent=()=>{
  const {dispatch} = useBookContext();
  const {isbn} =useParams();
  useEffect(()=>{
    getBooksByIsbn(isbn)(dispatch);
  },[isbn]);
}
```

Why do I need two functions

- Outer function takes actual parameter needed by the logic
- Inner function takes dispatch
- We want to keep it separate

```
export const getAllBooks= ()=> {
  return async dispatch=>{
    //actual logic here
    dispatch({type:Status.WAITING});
    try{
      let books=await bookService.getAllBooks();
      (parameter) dispatch: any BOOK_LIST_SELECT,payload:books);
      dispatch({type:Status.SUCCESS});
    }catch(error){
      dispatch({type:Status.ERROR, payload:error});
    }
  }
}
```

Redux

Saturday, January 22, 2022 5:15 PM

- Redux is a third party library that we can use with React
- We need to install the library first

```
npm install redux
```

- Redux has all the concept same as Reducer patterns including

- Store
 - Reducer
 - Action
 - Dispatch
 - Action creator
- It adds extra functionality such as
 - combineReducers
 - Middlewares

combineReducer

- It allows us to create many small reducer
- Each reducer handling a single piece of data
- Then we can combine them into a single store object

```
store.js
import {booksReducer, selectedBookReducer} from './book-reducers';
import {userReducer} from './user-reducer';
import {authorsReducer, selectedAuthorReducer} from './author-reducer';
import {statusReducer} from './status-reducer';
import {combineReducers, createStore} from 'redux';

const reducers=combineReducers({
  books:booksReducer,
  selectedBook:selectedBookReducer,
  authors:authorsReducer,
  selectedAuthor:selectedAuthorReducer,
  user:userReducer,
  status:statusReducer
});

export default createStore(reducers);
```

- Here we have a store that will look like

```
{
  books:[],
  selectedBook:null,
  authors:[],
  selectedAuthor:null,
  status:{status:success, message:null}
}
```

- We will have one store in memory
- Each property of this store will be handled by a different reducer named here

If I use React 16.8 for multiple reducer

Options#1

- Create a single reducer function to include all action handling
 - It will make it too big

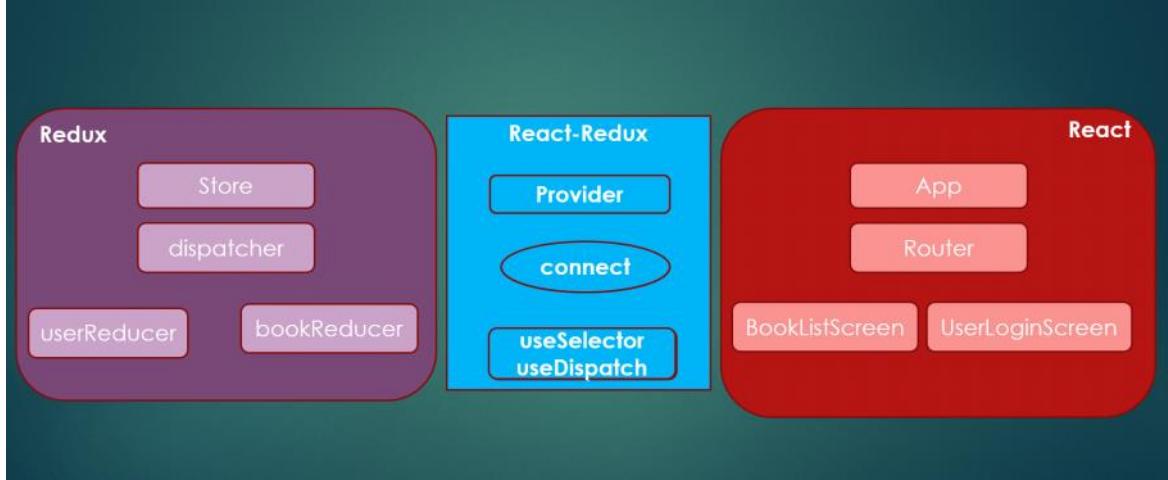
Option#2

- Create different Reducers,
- Create different context for each reducer
- Apply multiple context around our app

Redux and React don't know each other

- The two are independent libraries.
 - They don't know how to work with each other.
 - You need someone to act as bridge between them

```
npm install react-redux
```



- This package provides connectors between React and Redux
 - It provides a Context so we don't need to create our own Context
 - It provides hooks to access dispatch and data
 - It also provides and HOC to connect component with Redux data

Adding Redux Data to our app

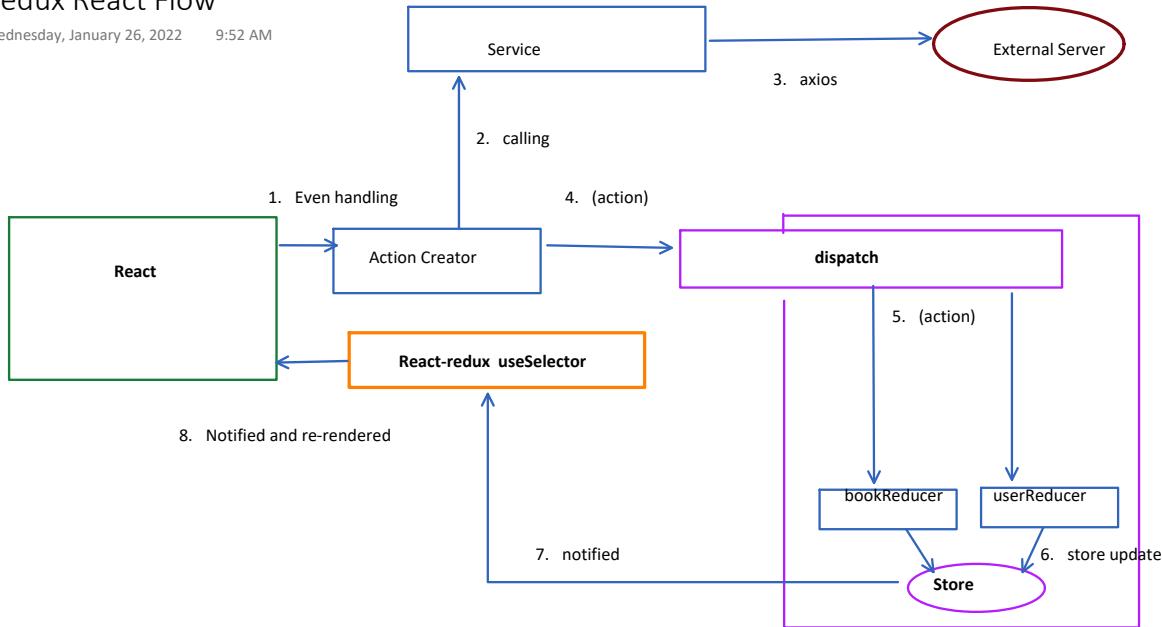
- We can use "Provider" from our app to add Redux Store to ReactApp

```
3 import store from './store/store';
4 import {Provider} from 'react-redux';
5
6
7 > const App = () => {
8   ;
9
10  const ReduxApp=()=>{
11
12    return (
13      <Provider store={store}>
14        <App/>
15      </Provider>
16    );
17  }
18
19
20  export default ReduxApp;
```

- Redux Store using
 - Reducer
 - Action
 - Dispatch
 - Connected to React App
 - With the help of React-Redux Provider

Redux React Flow

Wednesday, January 26, 2022 9:52 AM

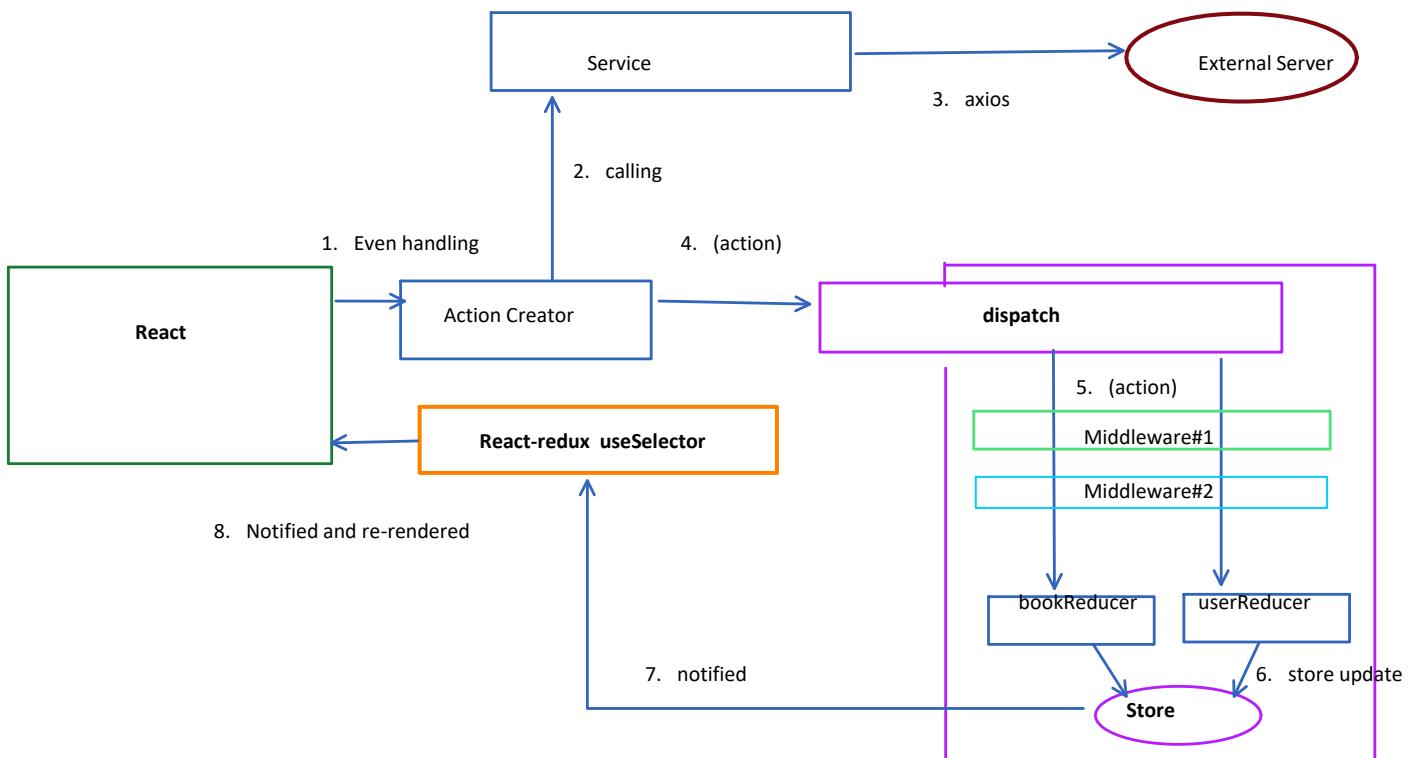


- Redux doesn't know which action is meant for which reducer
- So all actions are sent to all reducer
 - USER_LOGIN will also be sent to bookReducer
- That is why we have default case

Redux Middlewares

Wednesday, January 26, 2022 10:19 AM

- Redux Middleware is a special function that can sit between dispatch and Reducer
- Before an action reaches a reducer, a middleware can
 - Inspect it
 - May be log it
 - Modify it
 - May be change the action name or payload
 - Stop it from reaching the reducer
- We can configure multiple middleware
 - Each of them is called before it reaches reducer
- Consider reducer as the last middleware in a chain of middleware's



Middleware

- When an action is dispatched to redux redux dispatches the action to each reducer
- Before the action reaches a reducer, it is passed through various middleware
- Each middleware can perform an action to that "action"
- Each middleware must explicitly call "next()" to pass it on to
 - Next middleware or
 - Reducer
- If a middleware doesn't call next
 - The action will be lost as if never sent.
 - It will not reach reducer

How to create a Middleware

- A middleware uses the closure concept with three depths

1. Middleware Creator

- We define a function that can be called middleware

How to create a Middleware

- A middleware uses the closure concept with three depths

```
function middlewareCreator ( store ) {  
  
  return middleware( next ){  
  
    return middlewareAction(action){  
  
      //actual middleware job here  
      next(action); //to pass on  
    }  
  }  
}
```

3. Middleware action

- This function is called on every dispatch
- This is where we write our main logic
- Due to closure syntax, this function has an access to
 - store
 - next
 - action
- The middleware can
 1. Check current store data
 2. Use a dispatch from the store
 3. Can check the current action type and payload
 4. Can pass the control to "next"

1. Middleware Creator

- We define a function that can be called middleware creator
- This function gets the redux store object containing
 - state
 - Dispatch
- This function should return another function that can be considered the middleware
- **This function called exactly once while configuring redux**

2. Middleware

- This is the function that is used to set our middleware in the redux pipe line
- This function gets "next" which is a reference to
 - next middleware or
 - reducer
 - If this is the last middleware
- **This function is also called only once while building the redux middleware pipeline**
- **This function returns another function that will do the actual job given to middleware**

Simplifying the Middleware

- Note
 - the outer two functions have a single return statement
 - They are called only once during initial setup and not on every dispatch

```
const myMiddleware =  store => next => action =>{  
  
  //your middleware logic here  
}  
}
```

- You can visualize it as a simple function with three parameters separated by "`=>`" instead of ","

Simpler Do Nothing Middleware will look like

```
const myMiddleware =  store => next => action =>{  
  
  //at least make the next action work  
  next(action);  
}
```

Configuring the middleware in the store

```
//store.js
```

```
import { combineReducers, createStore, compose, applyMiddleware } from 'redux';
```

```

...
const reducers=combineReducers({
  books: booksReducer,
  ...
});
export default createStore( reducers,
  compose(
    applyMiddleware(myMiddleware1, myMiddleware2)
  )
)

```

Middleware with Arguments

- Sometimes we may need to pass our own argument to middleware

1. We can wrap the above middleware structure in yet another function and pass our argument

```

const myMiddleware = myArg => store => next => action =>{
  //middleware logic using myArg
}

```

2. Call your function to unwrap middleware creator before applyMiddleware

```
createStore( reducers, compose(applyMiddleware( myMiddleware(arg) )));
```

Important Use Case

Wednesday, January 26, 2022 11:02 AM

- Consider our action creators
- They have always a lot of redundant code
- Most Action creator looks like

```
const actionCreator => (action_params) => async dispatch =>{  
  
    dispatch({ type: Status.WAITING } );  
  
    try{  
        let result=await service.task( action_params);  
  
        dispatch({type:Status.SUCCESS} )  
        dispatch({type: SOME_TYPE, payload:result});  
    }catch(error){  
  
        dispatch({type:Status.ERROR, payload:error} );  
    }  
}
```

- Each action
 - sends three common status
 - Creates try -catch block
 - Calls our service
 - Dispatches the result of the service with a given action type
- Can we make our actions simpler

Ideal Action Creator

```
const actionCreator => async( action_params) =>{  
  
    let result=await service.task( action_params);  
  
    return {type: SOME_TYPE, payload:result };  
}
```

- I want all the benefit of above code without writing them
- You see I dispatch nothing.
- I don't use try-catch
- I don't set status dispatches

Problem

- You can't dispatch and async function
- You must have to wait for function to resolve and then dispatch the plain object as action
 - Then whoever is waiting must set the dispatches.

The screenshot shows a Visual Studio Code interface with several tabs open. On the left, a browser window displays an error message: "Uncaught Error: Actions must be plain objects. Instead, redux.js:275 the actual type was 'Promise'. You may need to add middleware to your store setup to handle dispatching other values, such as 'redux-thunk' to handle dispatching functions. See https://redux.js.org/tutorials/fundamentals/part-4-store#middleware and https://redux.js.org/tutorials/fundamentals/part-6-async-logic#using-the-redux-thunk-middleware for examples." A red arrow points from this error message to the word "Promise" in the code snippet above. A green arrow points from the "Recommendation" section below to the "useEffect" code in the middle of the screen.

```

author-actions.js U
export const getAllAuthors = async () => {
  const authors = await authorService.getAllAuthors();
  return { type: AuthorActions.LIST, payload: authors };
}

AuthorList.js 4. M X
src > components > AuthorList.js > AuthorList > useEffect() callback
// TODO: Initialize Here
const { authors, selectedAuthor, author, status } = useSelector(s => s);
const dispatch = useDispatch();
useEffect(() => {
  if (!authors || !authors.length) {
    dispatch(getAllAuthors());
  }
}, [status]);

```

We are dispatching a Promise

Recommendation

- We can write our own middleware to handle Promise

handlePromiseMiddleware

Wednesday, January 26, 2022 11:23 AM

- If action is a promise
 - Display waiting status
 - Let promise execute
 - On Resolve
 - dispatch success
 - Dispatch result
 - On Reject
 - Dispatch error
 - If not a promise
 - Pass to the next one

```
✓ export const handlePromiseWithStatus= store =>next =>action =>{  
  ✓ if(action instanceof Promise){  
    //I should handle it  
    store.dispatch({type:Status.WAITING});  
  
    action  
    | .then( actualAction=>{ //on success  
    | | store.dispatch({type:Status.SUCCESS});  
    | | store.dispatch(actualAction);  
    | })  
    | .catch(error=>{  
    | | store.dispatch({type:Status.ERROR,payload:error});  
    |});  
  }  
  ✓ } else {  
    // let the next guy handle it  
    next(action);  
  }  
}
```

Deployment

Saturday, January 22, 2022 6:01 PM

- React application is currently designed to run on development server
- It is not ready for production
- We can make it production ready

Step #1

- Build the project using the command `npm run build`
- Before you build make sure your code is using right api end point

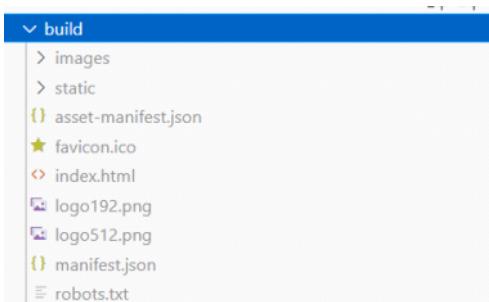
```
npm run build
```

```
npm i -D --save-exact mini-css-extract-plugin@2.4.5
```

From <<https://stackoverflow.com/questions/70715794/typeerror-minicsextractplugin-is-not-a-constructor/70716720>>

What build does

- It optimizes all for javascript file
 - It bundles all of them in a single js file
 - It converts JSX to javascript
 - It generates code understandable by browser
 - It creates a new folder containing files that can run on their own
-
- This file can run on any server that knows how to handle SPA



Hosting React App on a NodeJS Server

Step #1 copy our build folder on React Server

- Copy the content of the 'build' folder in express 'public' folder

Step #2 Tell express to serve this folder as static files

The screenshot shows a code editor interface with several tabs at the top: 'EXPLORER', '... hor-service.js', '.env', 'authors.json', 'books.json', 'book.js', and 'JS seeder.js'. The 'EXPLORER' tab is active, displaying a file tree for a 'BOOK-API-SERVER-V5' project. The 'public' folder is highlighted with a blue background. Inside 'public', there are subfolders 'images', 'static', and files like 'asset-manifest.json', 'favicon.ico', 'index.html', 'logo192.png', 'logo512.png', 'manifest.json', and 'robots.txt'. A yellow callout box with the text 'My React distribution code is here' points to the 'index.html' file. The main code editor area shows 'app.js' with the following code:

```

src > JS app.js > [e] configureMiddlewares
1 const express= require('express');
2 const path=require('path');
3 const db=require('./data/setup'); //must setup data before setting routes.
4 const getBookRouter=require('./routers/books-router');
5 const getAuthorRouter=require('./routers/author-router');
6 const getUserRouter=require('./routers/user-router');
7 const {tokenChecker} =require('./services/user-service');
8 const cors=require('cors');

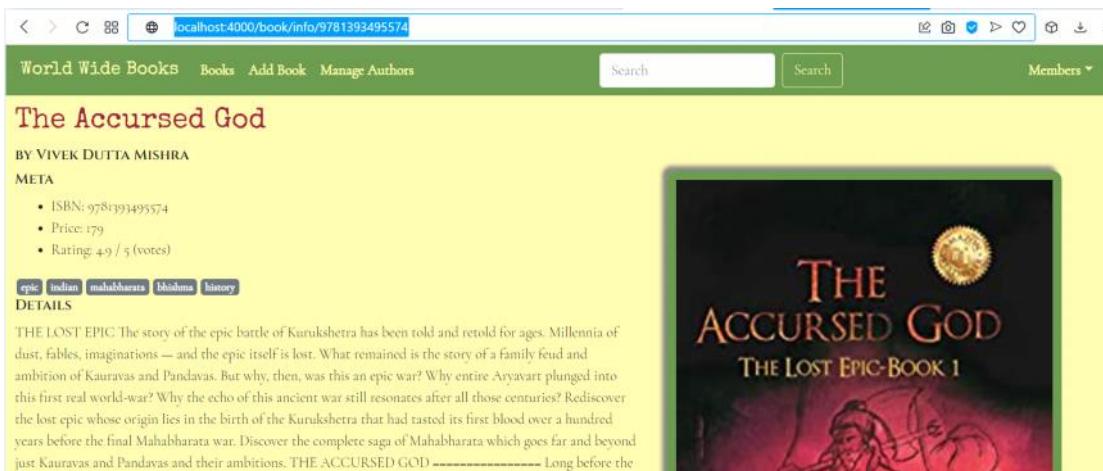
9
10
11 const configureMiddlewares=async(app,baseDir)=>{
12   app.use(express.static(path.join(baseDir, 'public')));
13   app.use(express.json());
14   app.use(cors());
15   app.use(express.urlencoded({ extended:true}));
16   app.use(tokenChecker);
17 };
18
19
20 const configureRoutes=async(app)=>{
21
22   app.use('/api/books', getBookRouter());
23   app.use('/api/authors', getAuthorRouter());
24   app.use('/api/users',getUserRouter());
25
26 };

```

Below the code editor are tabs for 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', and 'TERMINAL'. A status bar at the bottom says 'Installing C# dependencies...'. A small note in the bottom right corner says 'There's'.

Now My Server is ready to serve the home URL of React

- If you try for HOME URL you will get it
- You can navigate to other urls also from the application



Typing a nested React URL for a Browser Route will be rejected

Cannot GET /book/info/9781393495574

- This problem will not come if we use HashRouter
- But we want to use Browser route

Configure express to return Index.html for all unknown Requests

Add this route as the last route on express app

```
const configureSPARoute=(app, baseDir)=>{
  app.get("*", (request,response)=>{
    //for every other request send the index.html to the client
    const indexFile=path.join(baseDir,"public","index.html");
    fs.createReadStream(indexFile).pipe(response);
  });
}
```

Redux Dev Tools

Wednesday, January 26, 2022 9:19 AM

```
const store = createStore(  
  reducer, /* preloadedState, */  
+window.__REDUX_DEVTOOLS_EXTENSION__ && window.  
__REDUX_DEVTOOLS_EXTENSION__());
```

From <<https://github.com/zalmoxisus/redux-devtools-extension>>

```
const composeEnhancers=(typeof window!=='undefined'&&window.  
__REDUX_DEVTOOLS_EXTENSION_COMPOSE__)||compose;
```

From <<https://github.com/zalmoxisus/redux-devtools-extension>>

```
import { createStore, applyMiddleware, compose } from 'redux';  
  
+ const composeEnhancers = window.__REDUX_DEVTOOLS_EXTENSION_COMPOSE__ || compose;  
+ const store = createStore(reducer, /* preloadedState, */ composeEnhancers(  
- const store = createStore(reducer, /* preloadedState, */ compose(  
  applyMiddleware(...middleware)  
));
```

Topics for the Day

Wednesday, January 26, 2022 9:49 AM

1. Redux Middleware

2. Authentication Action

3. Dropdown List

4. React Tools

5. Redux with React Tools

6. Deployment to a Remote Server

Deployment

Wednesday, January 26, 2022 12:28 PM

<https://mobileum-books.herokuapp.com/>

1. On react app
`npm run build`
2. Copy content of 'build' from 'react' app to 'public' folder of 'express' server app
3. Copy all the files from express app to a new folder outside current repo
4. Push the deployment folder to a new github rep main branch
5. Create a heroku account
6. Create an app on heroku
7. Choose git deployment
8. Authorize heroku to read you git
9. Tell which repo you want to push
10. Chose deploy
11. Goto settings
12. Add your config values from .env to heroku manually

You are done.

Third Party Component Libraries for React

Wednesday, January 26, 2022 12:34 PM

- We have some high quality pre created react components available to use in our project
- Some important ones are

1. Material UI

<https://mui.com>

2. Ant Design

<https://ant.design/components/overview/>

3. Bootstrap

<https://react-bootstrap.github.io>