CS 572 Modern Web Applications

Najeeb Najeeb, PhD (<u>najeeb@miu.edu</u>)

Copyright © 2021 Maharishi International University. All Rights Reserved. V1.0.0



JavaScriptFullStack Development



- MongoDB
 - NoSQL database (document store)
 - Stores JSON documents
- Express
 - JavaScript web framework
 - On top of Node
- Angular
 - JavaScript UI framework
 - Single Page Applications
- Node
 - JavaScript server-side platform
 - Single threaded, fast and scalable

Roadmap and Outcomes

- Node.js: write asynchronous (non-blocking) code. Understand node platform to start a project.
- Express: setup express and get requests and send back responses. REST API.
- MongoDB: what NoSQL DB looks like. Full API interacting with DB.
- AngularJS: Investigate AngularJS and architect it. A single page application.
- MEAN application: Learn by example. We will create a MEAN Games application.



Callbacks

Callback Usage Async



Run piece of code after events:

- click (link, button)
- data access db (read, write)
- other code finishing.

It is an anonymous function (nameless, passed to fun).

Example:

const timeoutHolder= setTimeout(callback, delayMillisec);

clearTimeout(timeoutHolder);

Callback Usage Async



Callbacks are asynchronous. Run when required not based on order on code. They are based on functions being able to invoke other questions.

```
const myCallback= function (number) {
  console.log("The number "+number+" is Odd");
const myFunction= function (number, callback) {
  console.log("This function may call another function if the
number is Odd");
  if (number % 2) {
    callback(number);
const randomNumber= Math.round(Math.rand());
myFunction(randomNumber, myCallback);
```

Callback Usage Async Scope

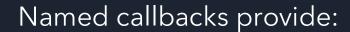


Callback function's scope is the scope in which it was defined not invoked.

```
const useVariablesHereNext;
const myCallback() {
  const useVariablesHereFirst;
  console.log("The vlaue of number is "+number);
Another module (where the callback is invoked).
const variableNotUsedInCallback;
const myFunction(myFunction) {
  const varaibleCannotBeUserInCallback;
  myCallback();
Best practice to pass varaibles as parameters to callback
const myFunction(myFunction) {
  const varaiblePassedToCallback;
  myCallback(varaiblePassedToCallback);
```

Callback

Usage Async Scope Named



- 1- Esiler to follow.
- 2- Easier to maintaine.
- 3- Unit test.



Callback

Usage
Async
Scope
Named
Node



Node needs callbacks.

- Main Node Process is single threaded.

Node Callbacks Callback Nested

Insert money into account, then get balance.

account= insertInAccount(amount, account);

const balance= getBalance(account);

return balance;

What is the problem?



Node Callbacks Callback Nested



```
Insert money into account, then get balance.
insertInAccount(amount, account, function(err, account) {
});
getBalance(account, function(err, balance) {
    return balance;
});
What is the problem?
```

Node Callbacks Callback Nested



```
Insert money into account, then get balance.
insertInAccount(amount, account, function(err, account) {
  getBalance(account, function(err, balance) {
    return balance;
  });
});
What is the problem?
Callback hell.
```

Node Callbacks Callback Nested Named



```
Insert money into account, then get balance.
insertInAccount(amount, account, onInsert);
const onInsert= function(err, account) {
  getBalance(account, onBalanceCheck);
const onBalanceCheck= function(err, balance) {
  return balance;
```



Promises

PromisesDefinition States



Promise is the result of asynchronous operations. When the result is available run a certain function otherwise run another.



PromisesDefinition States

Pending: initial state of Promise.

Fulfilled: Asychronous operation successfully resolved.

Rejected: Asychronous operation not successfully

resolved.

Fulfilled and Rejected promisses are immutable.



Promises Definition States Declaration



Promise is a function that takes two callback functions. One that is executed on success and one that is executed on failure.

The callbacks are fired by the promise execution.

Execution of callbacks in then() are executed (chained) on success.

Execution of callback in catch() is executed on failure.

```
const myPromise= new Promise((resolve, reject)=> {
    let num= Math.random();
    setTimeout(()=> {
        if (num > 0.5) {
            resolve(num);
        } else {
            reject(num);
        }
    }, 3000);
});
```

Promises Definition States Declaration all/race



Promise.race return the first one that finishes.

Promise.all([promise1,promise2,promise3]).then(success). catch(failuer);

Promise.race([promise1,promise2,promise3]).catch(succes s).catch(failure);





TypeScript

TypeScript install S

Promise.all success only if all succeed (array) or single err.

Promise.race return the first one that finishes.

npm i -g typescript

Promise.race([promise1,promise2,promise3]).catch(succes s).catch(failure);



TypeScript install Decorator



```
Helps with DI.

Class decorator

@myDecorator

class MyClass{ ...}

function myDecorator(constructor: Function) {
    Object.freeze(constructor);
    Object.freeze(constructor.prototype);
```

TypeScript install Decorator



Property Decorator

```
class MyClass{
  @myDecoratorFactory()
  property= "value";
function myDecoratorFactory() {
  return function(target: Obejct, key: string | sybmol) {
    let val= target[key];
    const getter= () => { return val; };
    const setter= (next) => {
      console.log("updating property");
      val= "! ${next} !";
    Object.defineProperty(target, key, {get:getter,
set:setter, enumerable:true, configurable: true,});
```

TypeScript install Decorator



Method Decorator

```
class MyClass{
  @addTax(0.07)
  get value() {
function addTax(percent: number) {
  return function(target: Obejct, key: string, descriptor:
PropertyDescriptor) {
    //modify the descriptor get
    const original= descriptor.get;
    descriptor.get= function() {
       const result= original.apply(this);
      return (result * (1+rate)).toFixed(2);
  return descriptor;
```

TypeScriptinstall Decorator



Introducing Hocks, a function that retuns getter and setter

```
class MyClass{
  value= 15.95;
  @addTax(0.07)
  get value() {
    return value;
  }
}
function addTax(percent: number) {
}
```