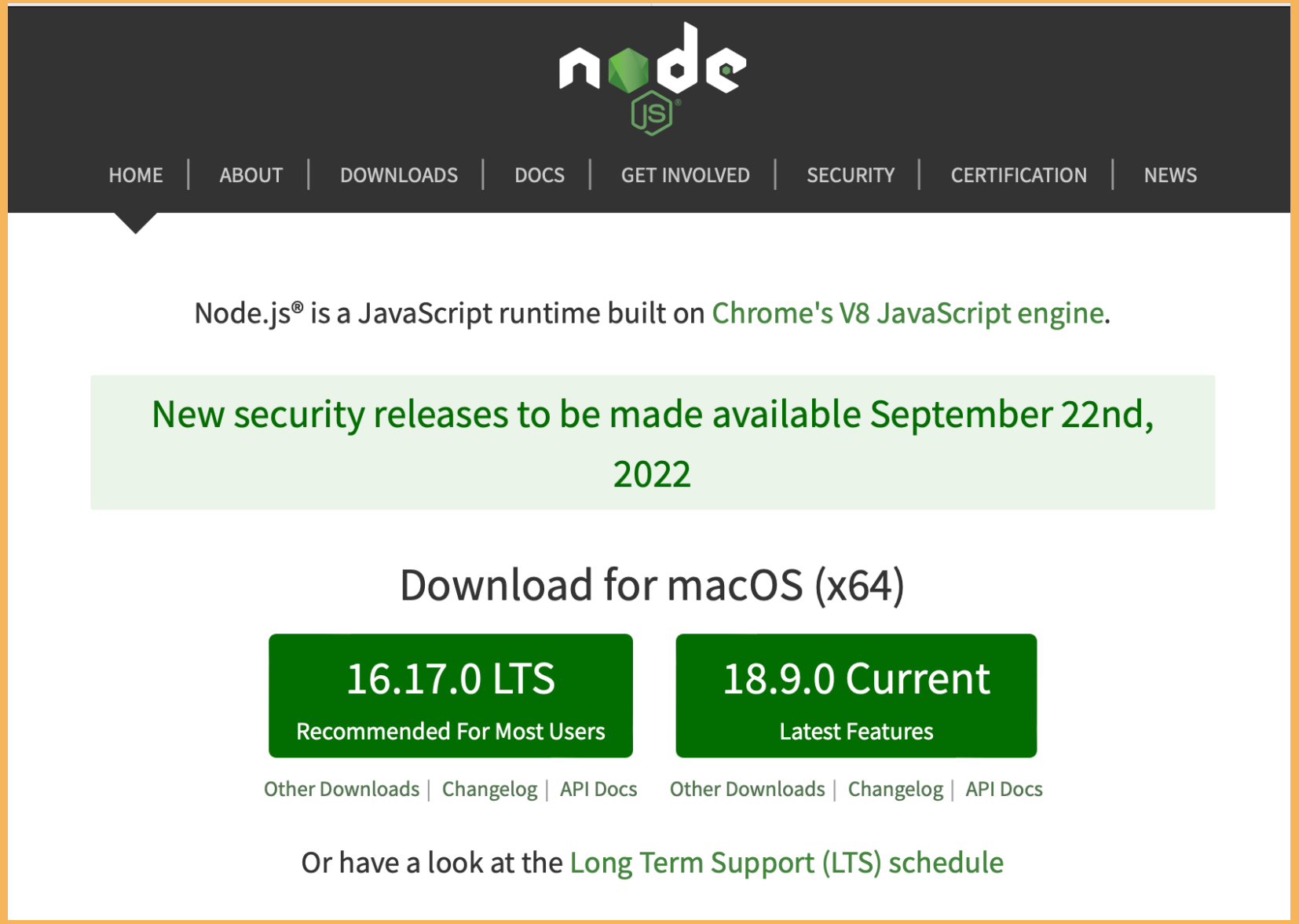
DEVKOR

Backend Study


node
JS
HOME | ABOUT | DOWNLOADS | DOCS | GET INVOLVED | SECURITY | CERTIFICATION | NEWS
Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine.
New security releases to be made available September 22nd, 2022
Download for macOS (x64)
16.17.0 LTS
Recommended For Most Users
18.9.0 Current
Latest Features
Other Downloads | Changelog | API Docs
Other Downloads | Changelog | API Docs
Or have a look at the Long Term Support (LTS) schedule

Node의 특징

About Node.js®

As an asynchronous event-driven JavaScript runtime, Node.js is designed to build scalable network applications. In the following "hello world" example, many connections can be handled concurrently. Upon each connection, the callback is fired, but if there is no work to be done, Node.js will sleep.

- Node JS = Asynchronous, event driven, single thread, Non-blocking I/O
- 특성을 파악하면, 노드 서버 코드의 동작 이해에 도움

Node의 특징

- Process: 프로그램의 인스턴스=현재 실행되는 프로그램.
- Thread: 프로세스보다 작은, 프로세스 내에서의 프로그램 실행 단위 (cpu 점유 단위)

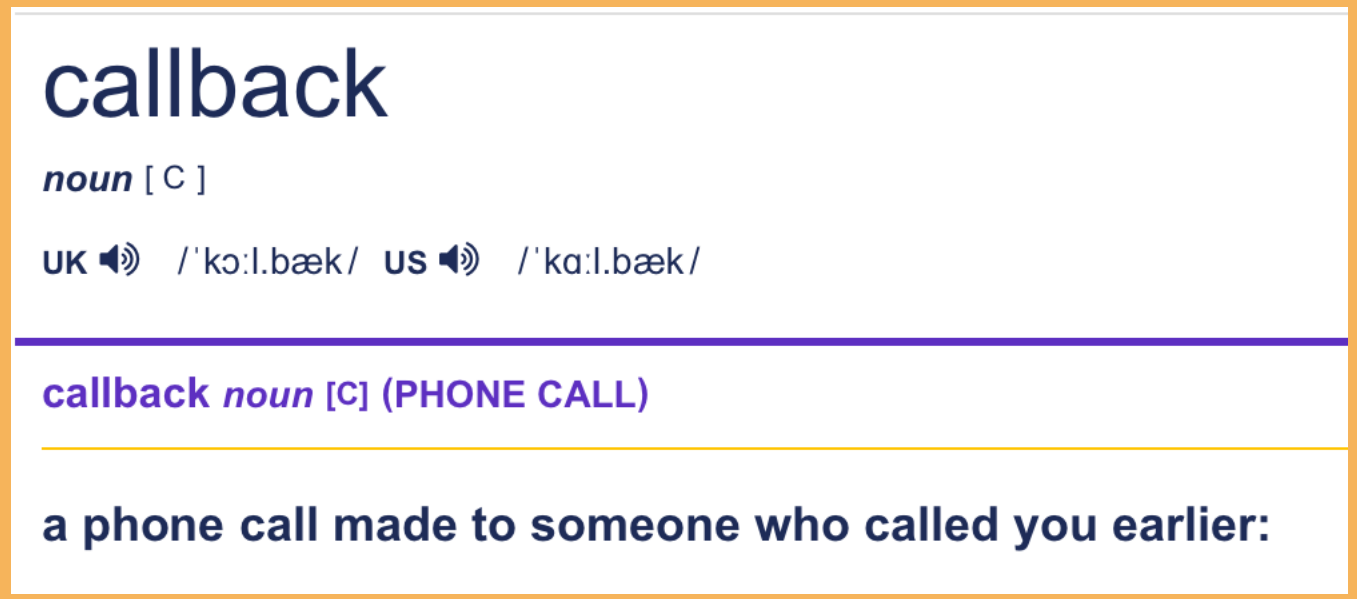
- Single Thread => 하나의 스레드로 서버가 실행됨
- 대신, Input, Output이 논 블로킹
  - 논 블로킹은 ..
  - 이전 작업이 완료되지 않음에도 다음 작업을 실행 가능
  - I/O에 좋은 성능
  - 클러스터링을 통해 멀티 프로세스로 멀티 스레딩을 대신함

```
const longTask = () => {
console.log('Finish');
}

console.log('start');
setTimeout(longTask, 0);
console.log('next');
```

```
❯ node ex.js
start
next
Finish
```

Node의 특징

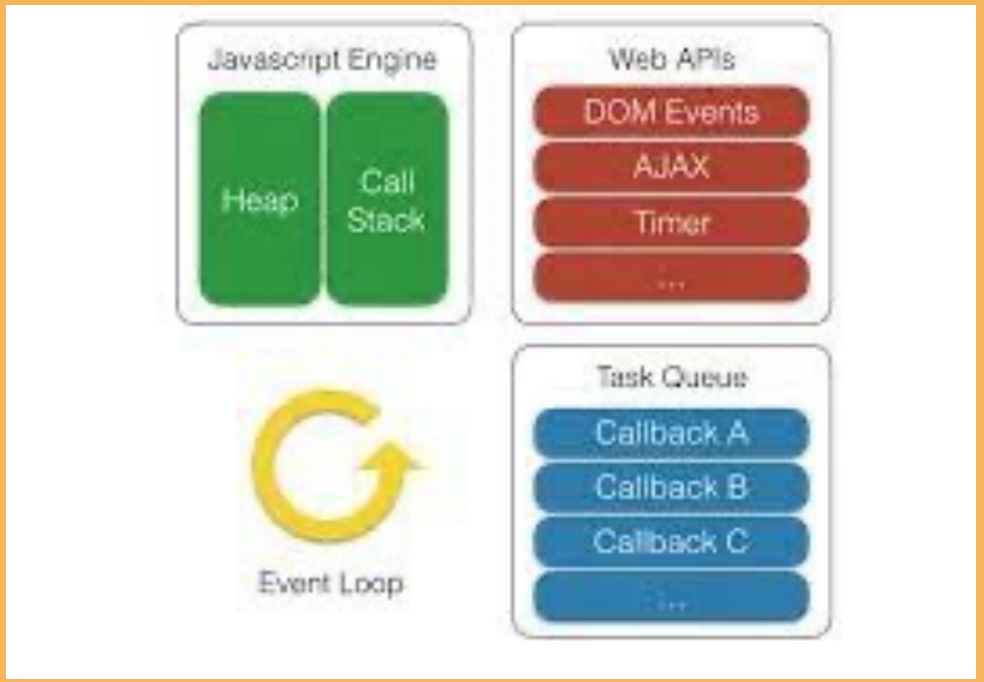
- Event-Driven
- 이벤트가 발생하면, 지정해둔 작업을 수행하는 방식
- Event Listner에 등록된 Call Back 함수를 실행하는 방식으로 동작함



- JS 공부를 하다 보면 많이 보게 될 call back
- 이벤트의 뒷처리 함수라고 생각하자

Node의 특징

- Event Loop
- Event Driven -> Node는 Event Loop이다
- 메시지 큐, 콜 스택, 이벤트 루프



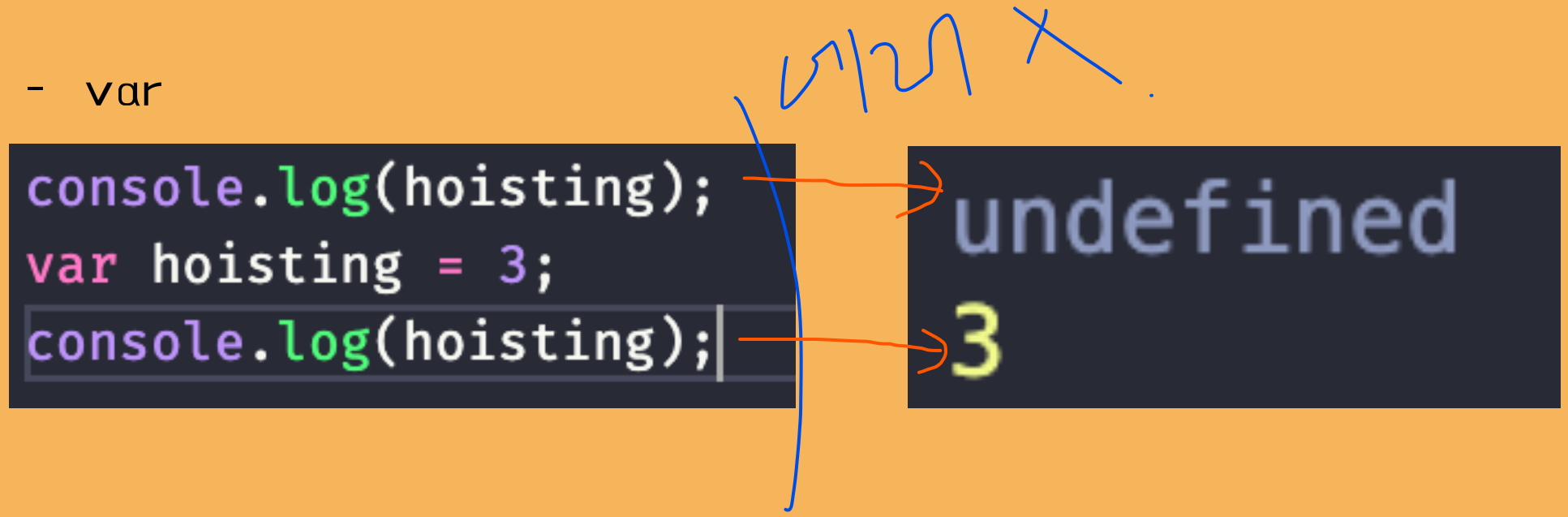
- 할 일 목록, 실제 호출중인 함수가 쌓이는 스택, 큐와 스택을 관리하는 루프

- message queue(Task queue + Job Queue)에서 실행시킬 일 들을 콜 스택으로 불러와 실행
- setTimeOut 등 event처리가 필요한 코드를 만나면,
- 해당 callback을 queue에 넣어두고, 나머지 코드를 실행
- 이것을 Node, Browser 환경이 관리 (Background, 3초를 기다린다던지.)

https://velog.io/@woohm402/javascript-async-and-event-loop

변수

- var



https://velog.io/@woohm402/javascript-async-and-event-loop

변수

- var의 scope 문제

```
var a =3;
if(true) {
  var a = 5;
}
console.log(a);
```

- Block scope가 아닌 함수 scope.

- 변수 이름이 겹치거나 할 때, 문제가 발생할 수 있음
- 함수 외부에서 선언 시 무조건 전역 변수가 됨

변수

- let, const

```
let be = 'study'
console.log(be) // output: study

// let name = 'howdy' // output: Uncaught SyntaxError: Identifier 'name' has already been declared

be = 'letsgo'
console.log(be) // output: letsgo
```

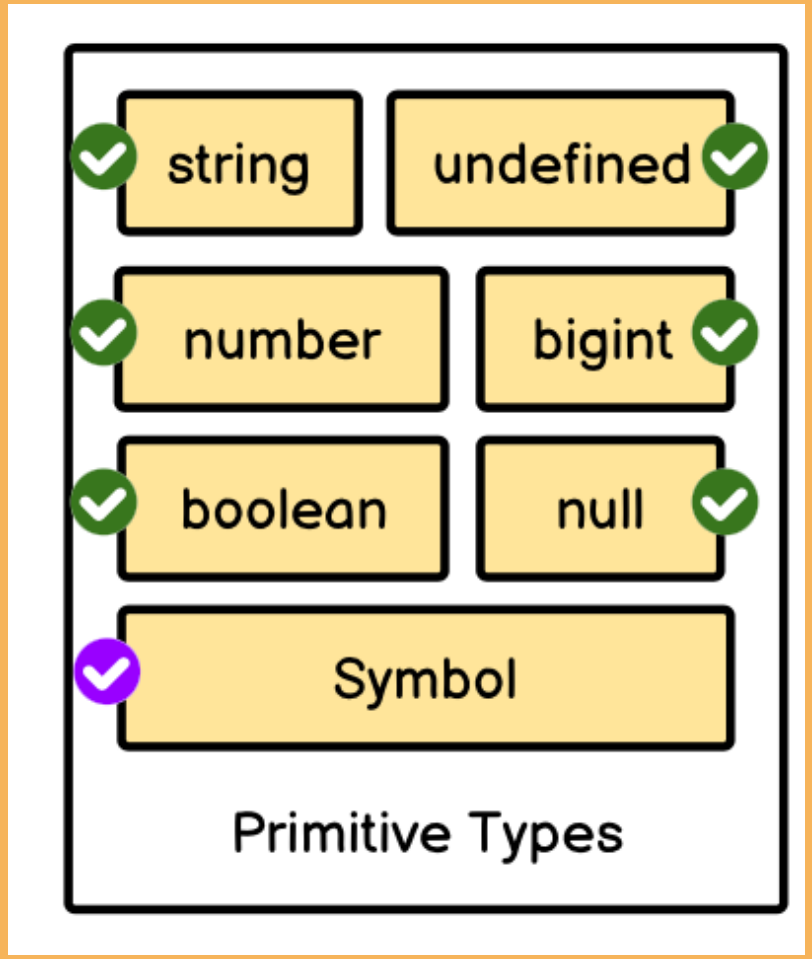
- Block scope - if, for, 함수 등 모든 곳에서 만들어지는 중괄호 {} 기준.
- 중복 선언이 불가능하다

- 값의 변경 가능 여부.

JS Data Types

https://javascript.info/types

https://javascript.info/object
- Array 등의 data type 역시 객체로 이루어짐



함수

- this 값이 달라진다는 것 외엔
- 명확하고 간편한 표기법

- function의 this,
- arrow function의 this

```
function foo() {
  return true;
}
const foo2 = function {
  return true;
}

const bar  = () ⇒ true;
const mult2 = x ⇒ x*2;
const pow3 = (a,b) ⇒ {
  let result = 1;
  for(let i = 0; i<b;++i) result *=a;
  return a;
}
```

https://codingapple.com/unit/es6-3-arrow-function-why/

클래스

```
class Rectangle {
  constructor(height, width) {
    this.height = height;
    this.width = width;
  }
  // Getter
  get area() {
    return this.calcArea();
  }
  // 메서드
  calcArea() {
    return this.height * this.width;
  }
}

const square = new Rectangle(10, 10);

console.log(square.area); // 100
```

```
class thisEx {
  array=  [1,2,3];
  getArray(){
    this.array.forEach(function ( element)  {
      console.log(this); // undefined
    })

    this.array.forEach((element) =>{
      console.log(this); // thisEx instance
    })
  }
}
const ex = new thisEx();
ex.getArray();
```

https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Classes

클래스

```
class Rectangle {
  constructor(height, width) {
    this.height = height;
    this.width = width;
  }
  // Getter
  get area() {
    return this.calcArea();
  }
  // 메서드
  calcArea() {
    return this.height * this.width;
  }
}

const square = new Rectangle(10, 10);

console.log(square.area); // 100
```

```
class thisEx {
  array=  [1,2,3];
  getArray(){
    this.array.forEach(function ( element)  {
      console.log(this); // undefined
    })

    this.array.forEach((element) =>{
      console.log(this); // thisEx instance
    })
  }
}
const ex = new thisEx();
ex.getArray();
```

https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Classes

비동기 프로그래밍 이용하기

```
function generatePrimes(quota) {

  function isPrime(n) {
    for (let c = 2; c <= Math.sqrt(n); ++c) {
      if (n % c === 0) {
          return false;
       }
    }
    return true;
  }

  const primes = [];
  const maximum = 1000000;

  while (primes.length < quota) {
    const candidate = Math.floor(Math.random() * (maximum + 1));
    if (isPrime(candidate)) {
      primes.push(candidate);
    }
  }

  return primes;
}
```

https://developer.mozilla.org/ko/docs/Learn/JavaScript/Asynchronous/Introducing

Promise

```
const fs = require('fs');

function readDirPromise() {
  return new Promise((resolve, reject) => {
    fs.readdir('.', function (err, filenames) {
      err ? reject(err) : resolve(filenames);
    });
  });
}

readDirPromise()
  .then(function (filenames) {
    console.log('filenames - ', filenames);
  })
  .catch(function (error) {
    console.log('error - ', error);
  });
```

비동기 실행은
순서가 꼬이기 쉬운데,
비동기 작업 순서가 정확히 작동하도록 하는 방법

callback보다 알아보기 쉬움.
Promise 객체를 반환
추후에 결과가 반환됨을 약속하는 Object 객체

resolve -> then
reject -> catch

https://developer.mozilla.org/ko/docs/Learn/JavaScript/Asynchronous/Promises

Promise

```
chooseToppings()
.then(toppings =>
  placeOrder(toppings)
)
.then(order =>
  collectOrder(order)
)
.then(pizza =>
  eatPizza(pizza)
)
.catch(failureCallback);
```

https://developer.mozilla.org/ko/docs/Learn/JavaScript/Asynchronous/Promises

async / await

비동기 함수 만들기

```
async function foo() {
  try{
    const promiseResult =  await bar();
    return promiseResult;
  } catch (err) {
    return err;
  }
}
const arrowFoo = async () => {
  try{
    const promiseResult =  await bar();
    return promiseResult;
  } catch (err) {
    return err;
  }
}
```

bar의 peomise가 resolve 된 후
promiseResult로 해당 데이터가 반환됨

reject -> catch로 간다

https://developer.mozilla.org/ko/docs/Learn/JavaScript/Asynchronous/Promises

JS module

함수를 쓰는 이유 중 하나
재 사용성!

비슷하게, 하나의 기능을 가진 상수, 함수 등을 하나의 모듈로 만든다면.
ex-달력 관련 기능.. 등

코드를 알아보기 쉽게, 반복된 코드의 작성을 피하게 해준다

```
const odd = '홀수';
const even = '짝수';

const checker = (num) ⇒ {
  if(num % 2 === 1) return odd;
  else return even;
}

module.exports={odd, even,checker};
```

```
const nums = require('./ex.js');
console.log(nums.odd, nums.even, nums.checker(3));
```

JS module

```
export default {odd, even, checker};

import nums from './ex.js';
import {odd, even, checker} from './ex.js';
```

```
const member= object.member ;
const {member} = object;
```

JS module

```
export odd;
export even
export checker;

import * as nums from './ex.js';
import {odd, even, checker} from './ex.js'
```

npm

- javascript만 사용해서 내가 모든 기능을 만든다면..
- C의 #include, python의 pip3 install --- 처럼,
- 다른 개발자들이 먼저 만들어놓은 모듈들의 집합, 패키지가 있다!

- npm node package manager
- node.js를 설치할 때 자동으로 따라옴

- 한 프로젝트의 package 설치, 의존성을 관리해준다

- $ npm init
- ++ github repository 만들기

https://www.npmjs.com

json

- JavaScript Object Notation

- 키 - 값 쌍으로 이루어진배열 자료형
- 자바스크립트에서 파생되어 많이 쓰이는 데이터 형식
- 객체와 비슷!

```
{
        “key”: “value”,
        ”object” : {
                ”member”: “value”
        }
}
```

express

- yarn
- $ npm install --global yarn

- $ npm install express
- $ yarn add express

express

- listen
  - 서버로 오는 HTTP request를 Listen
- middleware
  - express는 Request를 next function으로 넘기는 방식으로..
- routing
  - get, post, delete, put
- event-driven
  - request에 맞춰 해당 이벤트 리스너를 호출해줌

express

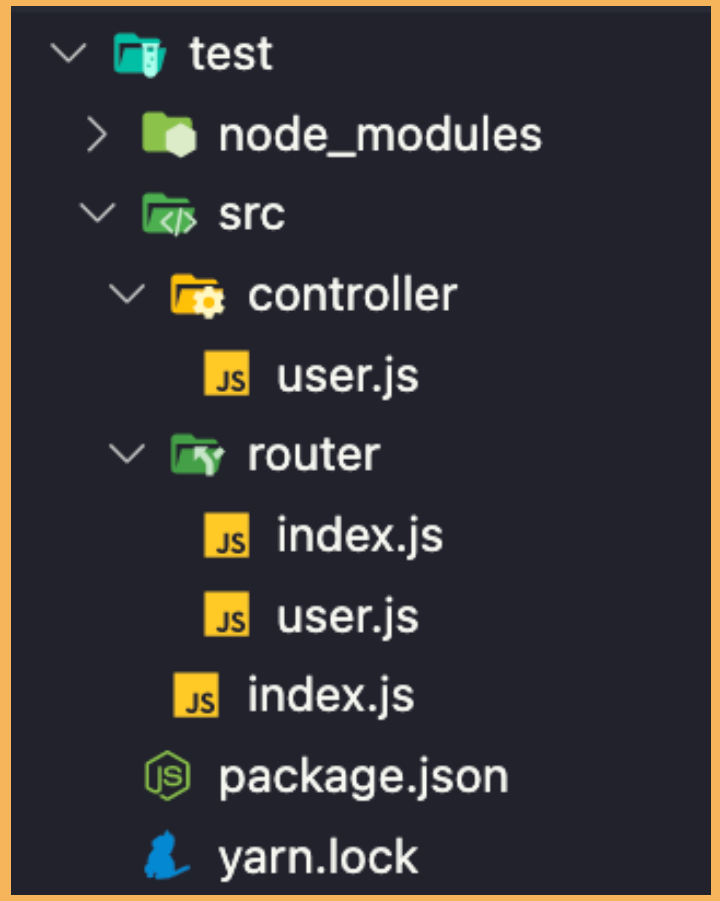
```
import express from 'express';
const app = express();
const port = 3000;

const loggingMiddleware = (req, res, next) => {
  console.log(req);
  next();
}
app.use(loggingMiddleware);

app.get('/', (req, res) => {
  res.send('Hello World!');
})

app.listen(port);
```

http://localhost:3000

https://expressjs.com

express

디렉토리 구조 분리하기



감사합니다!