익스프레스 웹 서버 만들기

익스프레스: 서버제작과정의 불편함을 해소하고 편의 기능을 추가한 대표 웹 서버 프레임워크.

; koa, hapi 등의 웹 서버 프레임워크가 있다.

■ 익스프레스 프로젝트 시작하기

1. learn-express 폴더 생성
2. npm init
3. npm i express
4. npm i –D nodemon //nodemon: 코드에 수정이 생길 때 마다 서버를 자동으로 재시작

   //nodemon은 개발용으로만 사용할 것을 권고 함.
5. 서버의 역할을 할 app.js 작성

```
const express = require('express');
//express 내부에 http 모듈이 내장되어 있음.
const path = require('path');
const app = express();
app.set('port', process.env.PORT||3000);//http://localhost:3000

app.get('/', (req, res)=>{
    //res.send('Hello, Express');
    res.sendFile(path.join(__dirname, '/index.html'));//index.html 화면 출력 됨.
});

app.listen(app.get('port'),()=>{
    console.log(app.get('port'),'번 포트에서 대기 중!');
});
```

- app.set('port', 포트) //서버가 실행될 포트를 설정 함.
- app.set(키, 값) //데이터 저장. app.get(키)로 값을 가져올 수 있다.
- app.get(주소, 라우터) //GET요청 시 실행할 동작을 적는 부분

  // res.write, res.end, res.send 를 사용 함.
-

6. $ npm start 실행 후 → http://localhost:3000 실행으로 app.js 실행 됨.

   ← package.json 내용에서 script 부분에 start 속성을 삽입함. -> nodemon app 하면 app.js 를 nodemon으로 실행한다는 의미 이다.

{

```
"name": "learn-express",
"version": "1.0.0",
"description": "express webServer testProject",
"main": "app.js",
"scripts": {
  "start": "nodemon app"
},
"author": "InjooJang",
"license": "MIT",
"dependencies": {
  "express": "^4.17.1"
},
"devDependencies": {
  "nodemon": "^2.0.12"
}
  }
```

■ 자주 사용하는 미들웨어

: 미들웨어는 express의 핵심이다. 요청과 응답을 조작하여 기능을 추가, 나쁜 요청을 걸러 내기도 한다.

: 미들웨어는 app.use와 함께 사용됨. 형식: app.use (미들웨어)

1. app.use 의 매개변수 : req, res, next
2. 미들웨어가 실행되는 경우
   i. app.use (미들웨어) // 모든 요청에서 미들웨어 실행
   ii. app.use ('/abc', 미들웨어) //abc로 시작하는 요청에서 미들웨어 실행.
   iii. app.post('/abc', 미들웨어) //abc로 시작하는 POST 요청에서 미들웨어 실행.
3. 에러 처리 미들웨어의 매개 변수 : err, req, res, next
4. ex.)

```
app.use((req, res, next)=>{
    console.log('모든 요청에 다 실행됩니다.');
    next();
});
app.get('/',(req, res, next)=>{}, (req, res)=>{});
app.use((err, req, res, next)=>{});
```

5. 자주 사용되는 패키지 설치 후 → app.js 수정

   $ npm I morgan cookie-parser express-session dotenv

   ** dotenv: process.env 를 관리하기 위해 설치

6. morgan

GET / 500 1.649 ms – 48 로그는 morgan 미들웨어에 의해 나옴.

morgan 미들웨어의 사용: app.use(morgan('dev'));

//인수는 dev이외에 combined, common, short, tiny 등 사용.

combined사용 후 ➔ `::1 - - [18/Jul/2021:06:31:39 +0000] "GET / HTTP/1.1" 500 48 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.164 Safari/537.36"`

common 사용 후 ➔ `::1 - - [18/Jul/2021:06:34:07 +0000] "GET / HTTP/1.1" 500 48`

dev 사용 후 ➔ GET / 500 1.649 ms – 48

tiny 사용 후 ➔ `GET / 500 48 - 11.715 ms`

7. **static**

: static 미들웨어는 정적인 파일들을 제공하는 라우터 역할. 기본적으로 express 객체 내에서 제공 됨.

app.use('요청경로', express.ststic('실제경로'));

예) `app.use('/', express.static(path.join(__dirname,'public')));`

→현재 실제 경로 = public

→의미: public/stylesheets/style.css는 http://localhost:3000/stylecheets/stlyle.cee로 접근 가능.

→서버의 실제 경로 와 요청경로가 다름 → 외부인의 서버 구조 파악을 어렵게 함. 보안기능.

→ 정적 파일 자동 제공으로 fs.readFile 을 수행할 필요가 없다.

8. **body-parser**

: 요청의 본문에 있는 데이터를 해석해서 req.body 객체로 만들어주는 미들웨어.

: 보통 폼 데이터나 AJAX 요청의 데이터를 처리 한다.

: 멀티파트(이미지, 동영상, 파일) 데이터는 처리하지 못함 → multer 모듈을 사용.

: 내부적으로 스트림을 처리해서 req.body에 추가 함으로, req.on('data'), req.on('end') 필요 없음.

app.use(express.json()); //익스프레스 4.16.0 이전버전에서 body-parser 별도 설치 필요 함

// $ npm I body-parser

// const bodyParser = require('body-parser');

// app.use(bodyParser.raw());

// app.use(bodyParser.text());

app.use(express.urlencoded({extended: false}));.

// {extended: false} → querystring 모듈을 사용하여 쿼리스트링을 해석.

// {extended: true} → qs모듈을 사용하여 쿼리스트링을 해석.

9. **cookie-parser**

: 요청에 동봉된 쿠키를 해석해 req.cookies 객체로 만든다.

app.use(cookieParser(비밀키));

: 해석된 쿠키는 req.cookies 객체에 들어간다.

: 서명이 붙은 쿠키는 req:signedCookies 객체에 들어간다.

: 쿠키의 생성과 삭제: res.cookie(키, 값, 옵션) / res.clearCookie(키, 값, 옵션)

: 옵션 → domain, expires, httpOnly, maxAge, path, secure 등.

: 서명을 위한 비밀키는 cookieParser 미들웨어에 인수로 넣은 process.env.COOKIE_SECRET

10. express-session

: 세션 관리용 미들웨어

: 로그인 등의 이유로 세션을 구현하거나 특정 사용자를 위한 데이터를 임시적으로 저장해 둘 때 매우 유용하다.

: 세션은 사용자별로 req.session 객체 안에 유지 된다.

: express-session 1.5버전 이전에는 cookie-parser 미들웨어 뒤에 나와야 한다.

: express-session은 인수로 세션에 대한 설정을 받는다.

```
app.use(session({
    resave: false,  // resave:요청이 올 때, 수정사항이 없어도 세션 재 저장.
    saveUninitialized: false, //저장할 내용이 없어도 처음부터 세션 저장 설정.
    secret: process.env.COOKIE_SECRET,
    cookie: {   //쿠키 옵션: maxAge, domain, path, expires, sameSite,
                //         : httpOnly, secure
        httpOnly: true,  // 클라이언트에서 쿠키 확인을 못하도록 함.
        secure: false,   //https 환경이 아니어도 사용 가능.
    },
    name: 'session-cookie',
}));
```

: express-session 로 만들어진 req.session 객체에 값을 대입, 삭제하여 세션 변경이 가능 함.

req.session.name = 'injoo'; //세션 등록

req.sessionID; //세션 ID 확인

req.session.deatroy(); // 세션 모두 제거

req.session.save(); // 세션 강제 저장.

: express-session 에서 서명한 쿠키 앞에는 s가 붙는다.

11. 미들웨어의 특성

: 미들웨어는 req, res, next를 매개변수(에러인 경우 만 err 추가)로 가지는 함수.

```
app.use((req, res, next)=>{
    console.log('모든 요청에 다 실행됩니다.');
    next();
});
```

: app.use 나 app.get, app.post 등으로 장착 됨.

: 예) 여러 개의 미들웨어 동시에 장착 가능.

```
app.use(
      morgan('tiny')
    express.static('/', path.join(__dirname,'public')),
    express.json(),
    express.urlencoded({extended: false}),
    cookieParser(process.env.COOKIE_SECRET)
);
```

→ 다음 미들웨어로 넘어가기 위해서는 next 함수 호출(내부적인 next 함수 호출시 생략.).

→ 내부적인 next 함수 호출이 없는 경우, res.send 또는 res.sendFile 등의 메서드로 응답 보냄.

→ 미들웨어 장착 순서에 따라 미들웨어 실행에 영향을 줌.

→next();//다음 미들웨어로

→next('route'); //다음 라우터로

→next(error); //에러 핸들러로ex) next(err)→(err, req, res, next)={. . .}

현재의 요청이 처리되는 동안 req.data를 통해 미들웨어 간에 데이터를 공유할 수 있다.

```
app.use((req, res, next)=>{
    req.data = '데이터 넣기';
    next();
}, (req, res, next)=>{
    console.log(req.data);  //데이터 받기
    next();
});
```

: 새로운 요청이 들어오면 req.data는 초기화 된다.

: app.set은 전역적으로 사용 됨 → 개개인의 값을 넣기에는 부적절.

: 미들웨어 안에 미들웨어를 넣는 방식으로 기존의 미들웨어의 기능을 확장 사용할 수 있다.

12. multer

: 이미지, 동영상 등을 비롯한 여러 가지 파일들을 멀티파트 형식으로 업로드 할 때 사용함.

: 멀티파티 형식: enctype = “multipart/form-data”

: $ npm i multer

: 예)

```
//====== multer 미들웨어 장착 >================================
const multer = require('multer');
const fs = require('fs');  //폴더 존재 여부 확인 및 생성을 위한 파일 관리모듈

try{
    fs.readdirSync('uploads');
} catch (error){
    console.error('uploads 폴더가 없어 생성합니다.');
    fs.mkdirSync('uploads');
```

```
}
const upload = multer({
    storage: multer.diskStorage({
            //어디에 어떤 이름으로 저장되는지에 관한 정보.
        destination(req, file, done) {
            done(null, 'uploads/');
            //req 나 file의 데이터를 가공해서 done 으로 넘기는 형식
            //req 매개변수 = 요청에 관한 정보.
            //file 매개변수 = 업로드한 파일에 관한 정보.
            //첫번째 인수: 에러가 있다면 에러
            //두번째 인수: 실제 경로나 파일 이름.
        },
        filename(req, file, done){
            const ext = path.extname(file.originalname);
            done(null, path.basename(file.originalname, ext)+Date.now()+e
xt);
        },
    }),
    limits: {fileSize: 5*1024*1024},//업로드에 대한 제한 사항
});
/*<form id="form" action="/upload" method="POST" enctype="multipart/form-
data" >
<input type="file" name="image1" />
<input type="file" name="image2" />
<input type="text" name="title" />
<button type="submit" >업로드</button>
</form>
*/
//==>localhost:3000/upload 에 접속하면 <form~ 수행된다.
app.get('/upload', (req, res) => {
    res.sendFile(path.join(__dirname, 'multipart.html'));
});
app.post('/upload',
    upload.fields([{name: 'image1'}, {name: 'image2'}]),
    (req, res)=>{
        console.log(req.files, req.body);
        res.send('ok');
    },
);
```