- Node.js를 이용한 REST 서버구축
- Express 백앤드(서버) 구축 시 편하게 만들어주는 프레임워크

- 장단점
  - HTML을 직접 다루기에는 불편. => Pug(jade), EJS 라이브러리 이용
  - 서버역할을 하면서 데이터를 다루기 쉽다. => 특히 JSON
  - 웹페이지 그 자체보다는 하나의 REST 서버로서 웹 페이지에서 Ajax 통신을 요청한다거나 안드로이드/아이폰 앱에서 통신을 요청할 때 데이터를 주고받는 데 가장 잘 어울린다.
- 목표
  - 개발 작업은 대부분 반복작업이므로 규격화하고 공통화하면 작업할 애용이 줄어들고 효용성이 달라짐
  - Node.js로 서버를 만들고 REST 형식으로 데이터를 주고받아야 한다

- 프로젝트 생성

```
D:\node_work>mkdir node-project
D:\node_work>cd node-project
D:\node_work\node-project>npm init

package name: (project) node-project  // 패키지 이름
version: (1.0.0)1.0.0                 // 패키지 버전
description:                          // 패키지 설명
entry point: (index.js)               // 가장 먼저 실행되는 main자바스크립트 실행파일
test command:                         // 코드를 테스트할 때 입력할 명령어
git repository:                       // 코드를 저장해둔 깃 저장소 주소
keywords:                             // npm 홈페이지에서 패키지를 검색할 때 제공되는 키워드
author:
license: (ISC)                        // Internet Systems Consortium에서 허용한 자유 SW 라이센스
About to write to D:\node_work\node-project\package.json:

Is this OK? (yes)
```

- package.json 터미널에서 npm install express 입력

```
{
  "name": "node-project",
  "version": "1.0.0",
  "description": "lerning for express",
  "main": "app.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [
    "node",
    "express",
    "mysql"
  ],
  "author": "chichi",
  "license": "ISC"
}
```

너무 길어서 설정해서 test만 쳐도 실행되게함
start랑 dev를 넣어서 사용

- Nodemon 설치 npm install -g nodemon --save-dev 로 할 것

```
> npm install nodemon --save-dev          //development 모드. local에서만 적용
```

- package.json 수정 밑에 app.js 파일 만들고 package.json 수정하기

```
  "scripts": {
    "start": "node app.js",
    "dev": "nodemon app.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "devDependencies": {
    "nodemon": "^2.0.20"
  }
```

- 실행

```
> npm run dev
```

npm start //start는 run생략가능

Express 프레임워크로 프로젝트 생성

- app.js

```
const express = require("express");
const app = express();
const port = 3000;

app.get("/", (req, res) => {
  res.send("hello world!");
});
```

구동하겠다

```
app.listen(port, () => {
  console.log(`server runing http://localhost:${port}`);
});
```

- **라우터 매핑** 기존에는 url과 메소드별로 if조건 걸면서 만들었는데 매우 간단해짐
  - app.get()
  - app.post()
  - app.put()
  - app.delete()
- app.route()
  - get,post, put과 같은 라우트 메소드를 한 곳에서 작성

```
app.route("/customer")
  .get((req, res) =>   { res.send("고객정보조회");  })
  .post((req, res) =>  { res.send("신규고객추가");  })
  .put((req, res) =>   { res.send("고객정보 수정"); })
  .delete((req, res) =>{ res.send("고객정보 삭제"); });
```

- express.Router
  - 라우트 처리를 하나의 파일에서 하는 것이 아니라 여러 개의 파일로 분리해서 구현
  - routes/product.js

```
const express = require("express");
const router = express.Router();

router.get("/", function (req, res) {   res.send("product root"); });
router.post("/insert", function (req, res) {   res.send("insert root");  });
router.put("/update", function (req, res) {   res.send("update root"); });
router.delete("/delete", function (req, res) {   res.send("delete route"); });

module.exports = router;
```

  - app.js

```
const productRoute = require("./routes/product");
app.use("/product", productRoute);
```

- 라우트 핸들러
  - 클라이언트 요청에 따라 라우트가 일치할 때 실행되는 콜백 함수
  - 파라미터
    - req : request
    - res : response
    - next : 다음 미들웨어 함수를 가리키는 오브젝트

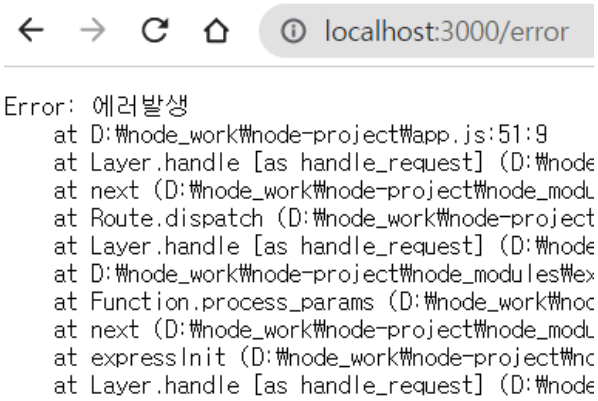
```
app.get("/", (req, res) => {    } );
```

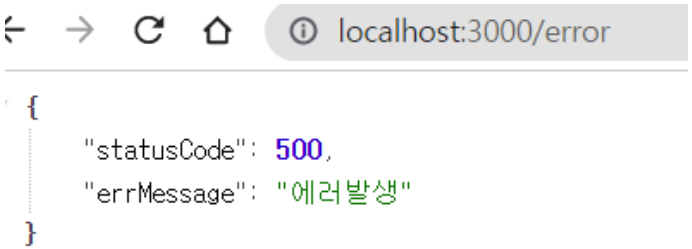
```
app.get(
  "/example",
  (req, res, next) => {
    console.log("첫번째 콜백");
    next();
  },
  (req, res) => {
    res.send("두번째 콜백");
  }
);
```

- 응답 메서드
  - res.download() : 파일을 다운로드하도록 프롬프트
  - res.end() : 응답 프로세스를 종료
  - res.json() : JSON 응답을 전송
  - res.jsonp() : JSPNP 지원을 통해 JSON 응답을 전송
  - res.redirect() : 요청의 경로를 재지정
  - res.render() : 뷰 템플릿을 랜더링
  - res.send() : 다양한 유형의 응답을 전송
  - res.sendfile() : 파일을 octet 스트림으로 전송
  - res.sendStatus() : 응답 상태 코드를 설정한 후 해당 코드를 응답 본문에 담아서 전송

- Express에서 에러 처리
  - 라우트에서 에러가 발생하면 익스프레스가 알아서 이를 잡아서 처리. 클라이언트에게 500 에러코드와 에러 정보를 전달

```
app.use(function (err, req, res, next) {
  res.status(500).json({ statusCode: res.statusCode, errMessage: err.message });
});
```





Express 프레임워크로 프로젝트 생성

form method="post"로 받은 파라미터를 urlencoded라고 함

- body-parse
  - json, raw, text, urlencoded
  - 클라이언트로부터 전달받은 데이터를 처리하는 함수 제공

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

- req.params
  - path variable에 접근 가능

```
app.get("/posts/:id", (req, res) => {
    res.send(req.params.id);
}
```

- req.body
  - json객체에 접근 가능 post데이터는 body로
- req.query
  - query sgtring에 접근 가능

- express.static
  - 이미지, CSS, 자바스크립트 파일과 같은 **정적 파일을 제공** html jpg파일 등

```
app.use(express.static("public"));
```

```
<img src="/images/logo.jpg">
```

  - 가상경로 추가하는 경우

```
app.use('/static',express.static("public"));
```

```
<img src="/static/images/logo.jpg">
```

- express-session

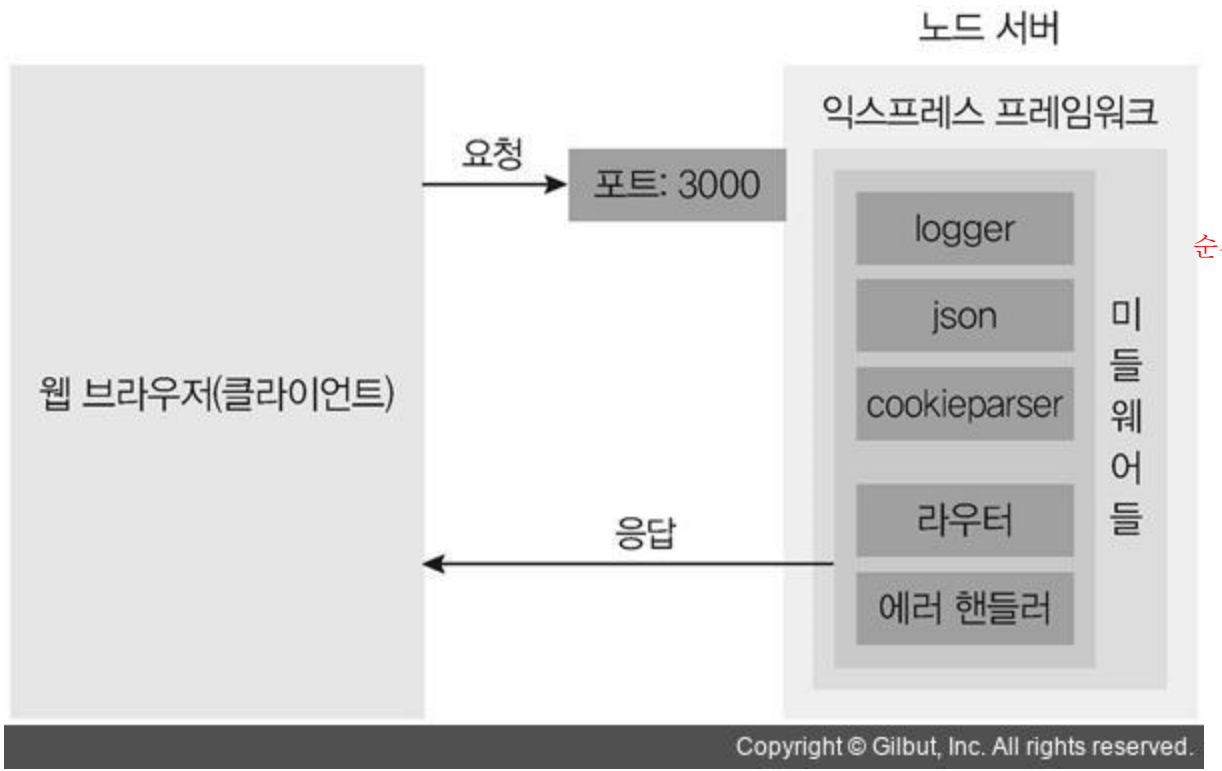
```
D:\node_work\node-project>npm install express-session
```

```
D:\node_work\node-project>npm install session-file-store
```

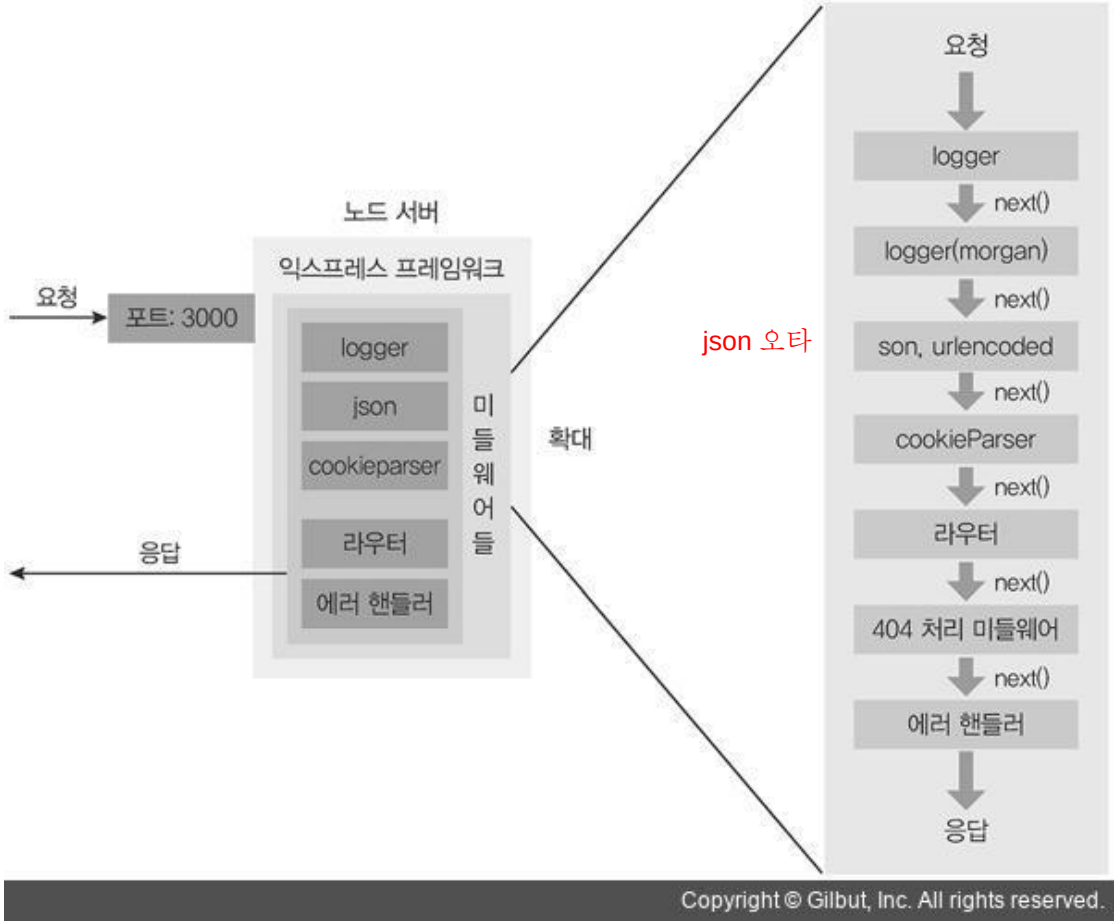
미들웨어 모듈

- 미들웨어 모듈
  - body-parser : HTTP 요청 body를 해석 app.use(json)
  - compression : HTTP 요청을 압축
  - connect-rid : 고유한 요청 ID를 생성
  - cookie-parser : 쿠키 헤더를 파싱하고 req.cookies에 할당
  - cors : 쿠키기반의 세션을 생성
  - csrf : CSRF 취약점을 방어
  - errorhandler : 개발 중에 발생하는 에러를 핸들링하고 디버깅
  - method-override : 헤더를 이용해 HTTP 메소드를 덮어씀
  - morgan : HTTP 요청 로그를 남김
  - multer : multi-part 폼 데이터를 처리
  - response-time : 응답시간을 기록
  - serve-favicon : 파비콘을 제공
  - serve-index : 주어진 경로의 디렉토리 리스트를 제공
  - server-static : 정적파일을 제공
  - express-session : 서버 기반의 세션을 생성
  - connect-timeout : HTTP 요청 처리를 위해 timeout을 생성
  - vhost : 가상 도메인을 생성

요청이 들어와서 응답하기 전까지 중간과정
app.use(session), app.use(router), app.use(cookies) app.use(json)



순서 잘 맞춰야 됨



json 오타

타

Express-generator로 프로젝트 생성 https://expressjs.com/en/starter/installing.html

웹앱 구성을 자동으로 해줌

- 웹서버 프레임워크
  - 서버 제작 시 불편함을 해소하고 편의 기능을 추가
- express 모듈 설치

```
> npm install express                    //POSt 처리 등 복잡한 작업을 대신 처리
```

```
> npm install express-generator –g       //express 앱의 기본골격을 빠르게 생성
```

- 프로젝트 생성

```
> express MyApp
```

- 의존성 있는 패키지 설치

```
> cd MyApp  && npm install
```

- 앱 시작

```
> port=80 npm start                      //포트를 생략하면 3000번으로 지정됨
```

set port=80 입력 후 npm start 할 것

Express 폴더 구성

```
.
├── app.js
├── bin
│   └── www
├── package.json
├── public
│   ├── images
│   ├── javascripts
│   └── stylesheets
│       └── style.css
├── routes
│   ├── index.js
│   └── users.js
└── views
    ├── error.pug
    ├── index.pug
    └── layout.pug
```

.

- http 모듈에 express 모듈을 연결하고 포트를 지정
- 프로그램 이름, 버전, 필요한 모듈 등 노드 프로그램의 정보를 기술
- 정적파일

- 라우팅을 위한 폴더. 라우팅 리소스별로 모듈 분리(컨트롤러 역할)

  routes는 필요한 대로 js파일 추가

- EJS 또는 Pub 템플릿

conf : DB 연결정보, API 키값 등 환경변수 정보 저장
web.js : express 설정파일

- Cross-)rigin Resources Sharing
  - 자신이 속하지 않은 다른 도메인, 다른 포트에 있는 리소스를 요청하는 방식. 설정을 하지 않으면 CORS 에러가 발생
- CORS 패키지 설치

```
> npm I cors
```

- 모두 허용

```
const cors = require('cors')
app.use(cors());
```

- 특정 도메인만 허용

```
const cors = require('cors')
let corsOption = {
  origin: 'http://www.sample.com',
  credentials: true
}
app.use(cors(corsOption));
```

- Pug, Mustache, and EJS.

https://www.npmjs.com/package/ejs
https://github.com/mde/ejs