1. Express 프레임워크

❖ 이전 예제 리뷰

- http + fs 사용 예제
- 서버를 구현할 수는 있지만 뭔가 불편하고 복잡함

  --> 웹 애플리케이션, REST API 등을 손쉽게 작성할 수 있는 프레임워크가 필요함
- 대표적인 Node 기반의 웹애플리케이션 프레임워크
  - Express
  - Restify
  - NestJS

프레임워크(Framework)

"소프트웨어의 구체적인 부분에 해당하는 설계와 구현을 재사용이 가능하게끔 일련의 협업화된 형태로 클래스들을 제공하는 것" 랄프 존슨(Ralph Johnson)의 이 말처럼 프레임워크는 한마디로 미리 필요한 작업을 만들어 놓은 것이라고 생각하면 됩니다. 웹을 만드는 회사들은 대부분 프레임워크를 사용해 웹을 제작합니다.

- 자바스크립트의 클라이언트 측(프론트엔드) 프레임워크 : React.js, Angular.js, Vue.js
- 자바스크립트의 서버 측(백엔드) 프레임워크 : node.js의 express

1. Express 프레임워크

❖ Express 프레임워크란?

- http, connect 모듈 기반으로 작성된 웹애플리케이션 프레임워크
- express 설치 및 nodemon 도구 설치
  - npm install --save express
  - npm install --save-dev nodemon
    - 또는 npm install -D nodemon
- 직접 실행
  - npx nodemon ./server.js
- script를 이용한 실행
  - npm run start-dev

```
//package.json

{
  "name": "test",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.18.2"
  },
  "devDependencies": {
    "nodemon": "^2.0.22"
  }
}
```

1. Express 프레임워크 기본 예제

❖ express_study2.js

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

const app = express();
app.set('port', process.env.PORT || 8080);

app.get('/', (req, res) => {
    res.sendFile(__dirname + '/index.html');
});

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

index.html

```
<!DOCTYPE html>
<html lang="ko">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport"
        content="width=device-width, initial-scale=1.0" />
    <title>express로 웹 만들기</title>
  </head>
  <body>
    <h2>express로 웹 만들기</h2>
    <p>메인 페이지 입니다.</p>
  </body>
</html>
```

2. HTTP 요청 메서드

❖ HTTP 요청 메서드

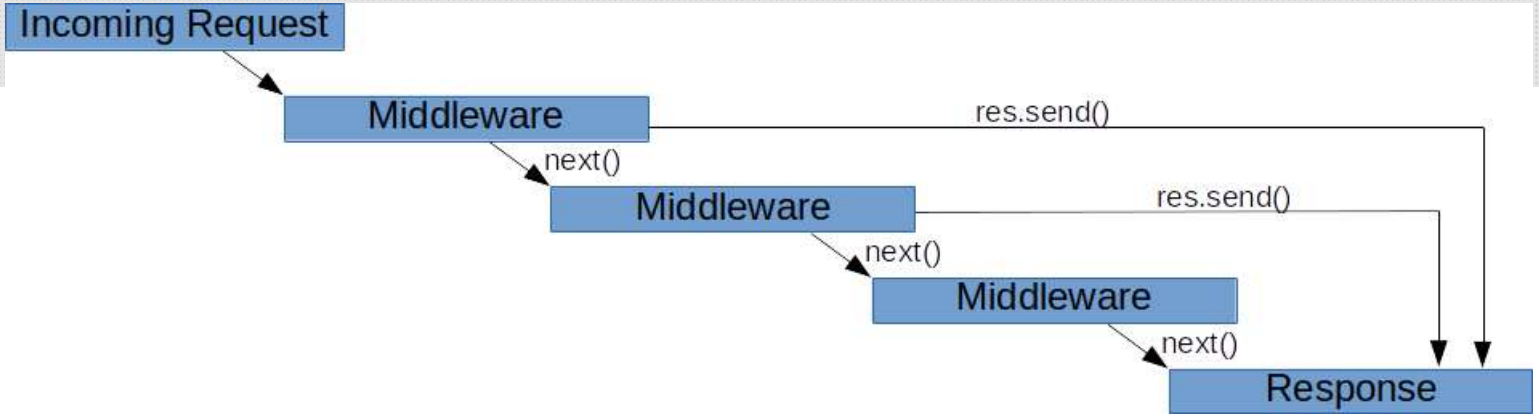
- 클라이언트가 웹서버로 요청 정보를 전달하는 방법

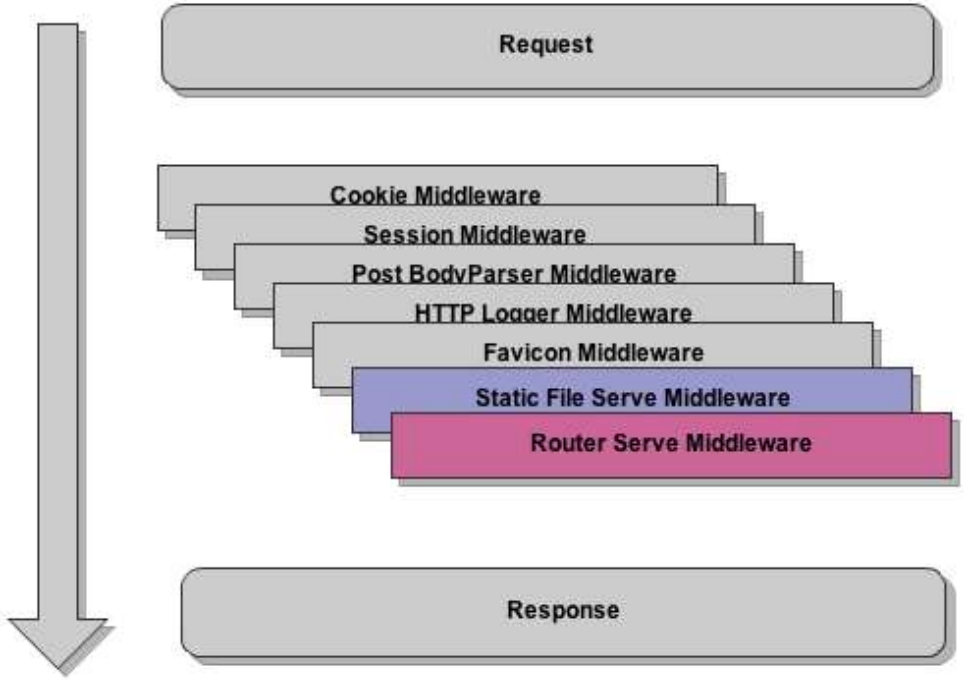
❖ HTTP 요청 메서드의 종류

- GET : 리소스를 얻을 때 사용합니다.
- HEAD : 문서의 정보를 얻을 때 사용합니다.
- POST : 리소스를 전송할 때 사용합니다.
- PUT : 내용 전체를 갱신할 때 사용합니다.
- PATCH : 내용을 부분적으로 갱신할 때 사용합니다.
- DELETE : 파일을 삭제할 때 사용합니다.

3. express와 미들웨어

❖ 미들웨어란?





3. express와 미들웨어

❖ 간단한 미들웨어 적용 예

```
const express = require('express');
const app = express();

const myLogger1 = function (req, res, next) {
    console.log('### logger1');
    next()
};

const myLogger2 = function (req, res, next) {
    console.log('### logger2');
    next()
};

app.use(myLogger1);
app.use(myLogger2);

app.get('/', function (req, res) {
    console.log('### GET /');
    res.send('Hello World!');
});

app.listen(8080);
```

[표 3-5] next()의 종류 및 내용

종류	내용
next()	다음 미들웨어로 가는 역할을 합니다.
next(error)	오류 처리 미들웨어로 가는 역할을 합니다
next('route')	많이 사용하지는 않지만 next()로 같은 라우터에서 분기처리를 할 때 사용합니다.

3. express와 미들웨어

❖ 자주 사용하는 미들웨어

- express.static() : 정적 리소스(ex: HTML 문서, favicon, css 등)의 경로를 지정할 때 사용함

```
app.use(express.static(__dirname + '/public'));
```

- router : 라우터도 일종의 미들웨어임

```
app.use('/경로', 미들웨어);
app.get('/경로', 미들웨어);
app.post('/경로', 미들웨어);
app.put('/경로', 미들웨어);
app.delete('/경로', 미들웨어);
```

```
app.get("/user/:id", function (req, res) {
    res.send("user id: " + req.params.id);
});
```

- express.json, express.urlencoded
  - 요청 정보를 파싱해주는 역할 : json, urlencoded 형식의 요청 정보를 파싱함

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

3. express와 미들웨어

❖ 자주 사용하는 미들웨어(이어서)

- cookie-parser : HTTP Cookie를 손쉽게 사용할수 있도록 도와주는 미들웨어
  - 미들웨어 등록

```
.....
const cookieParser = require('cookie-parser');

const app = express();
app.use(cookieParser());
```

  - 쿠키 생성

```
res.cookie('key1', 'value1',
   { httpOnly: true, path: '/', maxAge: 60*60*1000 });
```

  - 쿠키 정보 획득

```
const name = req.cookies.name;
```

3. express와 미들웨어

❖ 자주 사용하는 미들웨어(이어서)

- cookie-parser(이어서)

```
const express = require('express');
const cookieParser = require('cookie-parser');

const app = express();
app.use(cookieParser('pa$$w0rd'));

app.get('/cookie', (req, res)=> {
    res.cookie('key1', 'value1', { httpOnly: true, path: '/', maxAge: 60*60*1000, signed:true });
    res.send('<h1>cookie 생성</h1>')
})

app.get('/', function (req, res) {
    console.log('### GET /');
    console.log(req.signedCookies.key1);
    res.send('<h1>cookie 확인</h1>')
});

app.listen(8080);
```

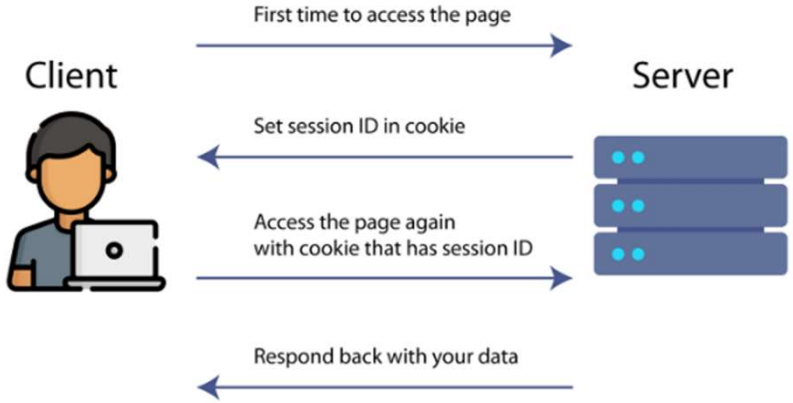
3. express와 미들웨어

❖ 자주 사용하는 미들웨어

- express-session : express 환경에서 session을 손쉽게 사용하기 위한 미들웨어
  - 내부적으로는 cookie 사용

```
const session = require('express-session');
```

```
app.use(session({
    secret: 'secret@1234',     // 암호화
    resave: false,             // 새로운 요청 시 세션에 변동 사항이 없어도 다시 저장할지 설정
    saveUninitialized: true,   // 세션에 저장할 내용이 없어도 저장할지 설정
    cookie: {
    // 세션 쿠키 옵션들 설정 httpOnly, expires, domain, path, secure, sameSite
        httpOnly: true,     // 로그인 구현 시 필수 적용, 자바스크립트로 접근 할 수 없게 하는 기능
    },
    // name: 'connect.sid' // 세션 쿠키의 Name 지정 default가 connect.sid
}));
```



4. 외부 API에 대한 요청을 위한 axios 모듈

❖ axios

- HTTP 요청으로 외부 API 를 호출하고 응답을 받을 수 있도록 하는 모듈
  - Promise 패턴, async/await 모두 사용

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


// GET 요청 전송 (기본 메서드)
axios('/user/12345');

// POST 요청 전송
axios({
    method: 'post',       // 요청 메서드
    url: '/user/12345',   // 요청을 보낼 url
    data: {
      firstName: 'Fred',
      lastName: 'Flintstone'
    }
  });
```

```
axios.get(url[, config])                // GET
axios.post(url[, data[, config]])       // POST
axios.put(url[, data[, config]])        // PUT
axios.patch(url[, data[, config]])      // PATCH
axios.delete(url[, config])             // DELETE
```

5. dotenv

❖ dotenv : 환경 변수를 이용할 수 있도록 하는 모듈

- 설치 : npm install --save dotenv
- 환경변수 설정 : .env 파일에 적절한 키-값 쌍을 저장

```
PORT=8080
SECRET=pa$$w0rd
```

- 이용하기

```
const dotenv = require('dotenv')
dotenv.config()
console.log(process.env.PORT)
console.log(process.env.SECRET)
```

```
const path = require('path')
const dotenv = require('dotenv')
//.env 파일 경로 직접 지정
dotenv.config({ path: path.resolve(__dirname, '../../.env') })
console.log(process.env.PORT)
console.log(process.env.SECRET)
```

6. RESTful API 작성

❖ 예제 RESTful API 작성

- 특정 소유자의 할일 목록 정보를 리턴하는 백엔드 API 서비스
- 제공할 엔드포인트
  - GET / : 서비스 소개 페이지 -> EJS 뷰템플릿 사용
  - GET /todolist/:owner
    - 소유자의 할일 목록 조회
  - GET /todolist/:owner/:id
    - 소유자의 할일 한건 조회
  - POST /todolist/:owner
    - 소유자의 할일 목록에 새로운 할일 추가
  - PUT /todolist/:owner/:id
    - 소유자의 할일 한건 변경
  - DELETE /todolist/:owner/:id
    - 소유자의 할일 한건 삭제
  - PUT /todolist/:owner/:id/completed
    - 소유자의 할일 한건의 완료 여부 필드(completed) 토글

6.1 프로젝트 초기화와 소개페이지 작성

❖ 프로젝트 생성 및 초기화

- todolist-svc 디렉토리 생성 후 디렉토리로 이동하여 npm init 명령어 실행
- 의존성 설치 : 필요한 의존 패키지를 미리 import 함
  - npm install --save bson-objectid cors ejs express lokijs morgan morgan rotating-file-stream
  - npm install --save-dev nodemon
- package.json에 script 등록

```
"scripts": {
  "dev": "nodemon ./server.js",
  "start": "node ./server.js"
},
```

❖ View 템플릿 페이지 등록

- ./views/index.ejs 파일 생성후 강사가 제공하는 코드 작성
  - EJS 뷰템플릿 기반의 템플릿 페이지
- express 애플리케이션 객체에서 View 템플릿 엔진 등록해야 함.

```
.....
<h1><%=title %></h1>
<h3><%=subtitle %></h3><hr />
.....
```

6.1 프로젝트 초기화와 소개페이지 작성

❖ router 객체 작성

- ./router.js 파일 생성후 다음 코드 작성
  - GET /요청시 index.ejs 템플릿을 이용해 title, subtitle 값을 포함하여 렌더링하도록 함

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

router.get("/", (req, res) => {
  console.log("### GET /");
  res.render("index", {
    title: "todolist 서비스 v1.0",
    subtitle: "(node.js + express + lokijs)",
  });
});

module.exports = router;
```

6.1 프로젝트 초기화와 소개페이지 작성

❖ server.js 작성

- ./server.js 파일 생성후 다음 코드 작성

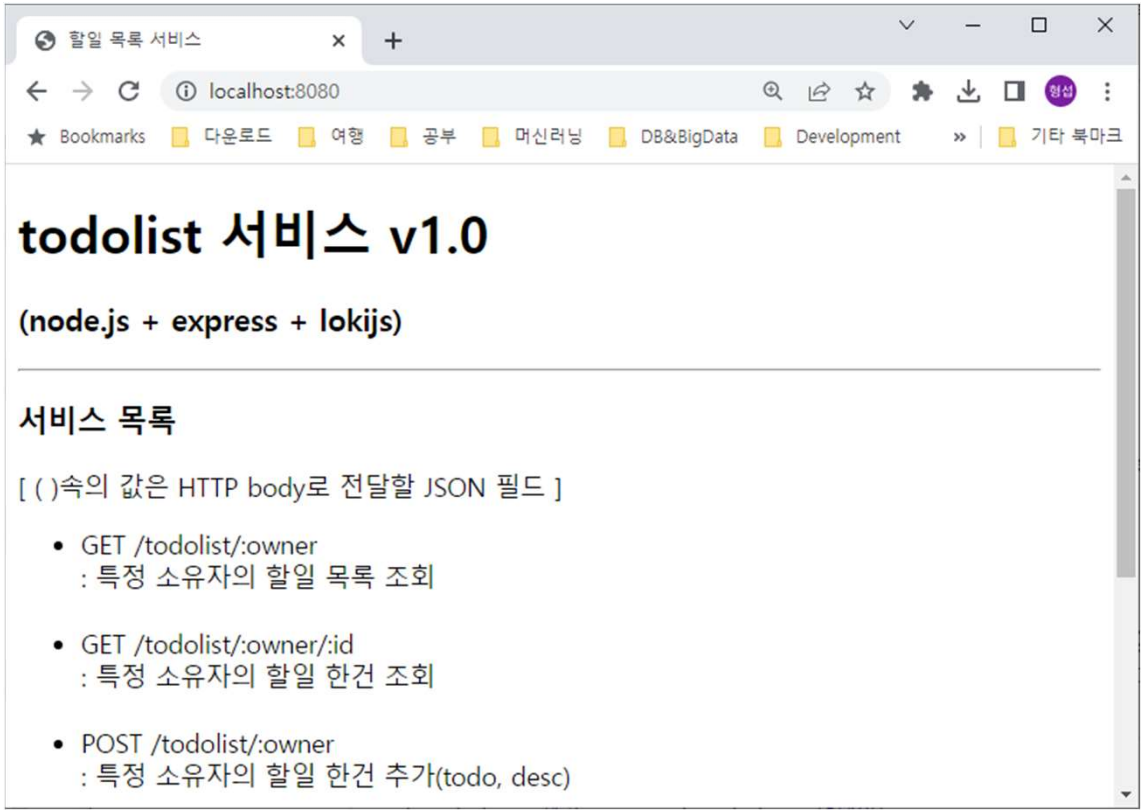
```
const express = require('express');
const path = require('path');

const router = require('./router');

const startServer = async () => {

  const BASEDIR = process.env.BASEDIR || path.resolve('.');
  const PORT = process.env.PORT || 8080;

  const app = express();

  app.use(express.static(BASEDIR + '/public'));
  app.set('views', BASEDIR + '/views');
  app.set('view engine', 'ejs');
  app.engine('html', require('ejs').renderFile);
  app.use(express.json());
  app.use(express.urlencoded({
    extended: true
  }));
```

```
  app.use(router);

  app.listen(PORT, () => {
    console.log(`#### ${PORT} 에서 서버가 시작되었습니다`);
  });

};

startServer();
```

6.1 프로젝트 초기화와 소개페이지 작성

❖ 실행 후 결과 확인

- npm run dev



6.2 DAO 작성과 API Endpoint 추가

❖ DAO 작성

- ./dao/todolistDao.js 파일 생성 후 강사로부터 제공받은 코드를 설정함.
- lokijs라는 로컬 JSON 파일 DB를 사용함
  - 향후 MongoDB 를 이용하는 코드로 변경할 것임
- 제공하는 메서드
  - createOwner({ owner })
  - getTodoList({ owner })
  - getTodoItem({ owner, id })
  - addTodo({ owner, todo, desc })
  - deleteTodo({ owner, id })
  - updateTodo({ owner, id, todo, desc, completed })
  - toggleCompleted({ owner, id })

6.2 DAO 작성과 API Endpoint 추가

❖ API Endpoint 추가

- ./router.js 파일에 다음 코드 추가

```
const express = require('express');
const router = express.Router();
const todolistDao = require('./dao/todolistDao');
.....

router.get("/todolist/:owner/create", (req, res) => {
    console.log("### GET /todolist/:owner/create");
    const { owner } = req.params;
    const result = todolistDao.createNewOwner({ owner });
    res.json(result);
});

router.get("/todolist/:owner", (req, res) => {
    console.log("### GET /todolist/:owner");
    const owner = req.params.owner;
    const todolist = todolistDao.getTodoList({ owner });
    res.json(todolist);
});
```

```
router.get("/todolist/:owner/:id", (req, res) => {
    console.log("### GET /todolist/:owner/:id");
    const { owner, id } = req.params;
    const todoitem = todolistDao.getTodoItem({ owner, id });
    res.json(todoitem);
});

router.post("/todolist/:owner", (req, res) => {
    console.log("### POST /todolist/:owner");
    const { owner } = req.params;
    let { todo, desc } = req.body;
    const result = todolistDao.addTodo({ owner, todo, desc });
    res.json(result);
});

router.put("/todolist/:owner/:id", (req, res) => {
    console.log("### PUT /todolist/:owner/:id");
    const { owner, id } = req.params;
    let { todo, completed, desc } = req.body;
    const result = todolistDao.updateTodo({ owner, id, todo, desc, completed });
    res.json(result);
});
```

6.2 DAO 작성과 API Endpoint 추가

❖ API Endpoint 추가

- ./router.js 파일에 다음 코드 추가(이어서)

```
router.put("/todolist/:owner/:id/completed", (req, res) => {
    console.log("### PUT /todolist/:owner/:id/completed");
    const { owner, id } = req.params;
    const result = todolistDao.toggleCompleted({ owner, id });
    res.json(result);
});

router.delete("/todolist/:owner/:id", (req, res) => {
    console.log("### DELETE /todolist/:owner/:id");
    const { owner, id } = req.params;
    const result = todolistDao.deleteTodo({ owner, id });
    res.json(result);
});

//----에러 처리 시작
router.get("*", (req, res, next) => {
    const err = new Error();
    err.status = 404;
    next(err);
});
```

```
router.use((err, req, res, next) => {
    console.log("### ERROR!!");
    if (err.status === 404) {
        res.status(404).json({ status: 404, message: "잘못된 URI 요청" });
    } else if (err.status === 500) {
        res.status(500).json({ status: 500, message: "내부 서버 오류" });
    } else {
        res.status(err.status).jsonp({ status: "fail", message: err.message });
    }
});

.....
```

6.3 미들웨어 추가

❖ express App 객체에 미들웨어 추가

- ./server.js 변경

```
const express = require('express');
const cors = require('cors');
const morgan = require('morgan');
const path = require('path');
const fs = require('fs');
const rfs = require('rotating-file-stream');

const router = require('./router');

const startServer = async () => {

  const BASEDIR = process.env.BASEDIR || path.resolve('.');
  const LOGDIR = process.env.LOGDIR|| path.join(BASEDIR, '/log')
  const PORT = process.env.PORT || 8080;

  const app = express();

  //cors 설정
  app.use(cors());
```

```
  //로깅
  fs.existsSync(LOGDIR) || fs.mkdirSync(LOGDIR)
  const accessLogStream = rfs.createStream('access.log', {
    interval: '1d', // 매일 매일 로그 파일 생성
    path: LOGDIR
  })
  app.use(morgan('combined', {stream: accessLogStream}))

  app.use(express.static(BASEDIR + '/public'));
  app.set('views', BASEDIR + '/views');
  app.set('view engine', 'ejs');
  app.engine('html', require('ejs').renderFile);
  app.use(express.json());
  app.use(express.urlencoded({
    extended: true
  }));

  app.use(function (req, res, next) {
      res.header('Cache-Control',
            'private, no-cache, no-store, must-revalidate');
      res.header('Expires', '-1');
      res.header('Pragma', 'no-cache');
      next()
  });
```

6.3 미들웨어 추가

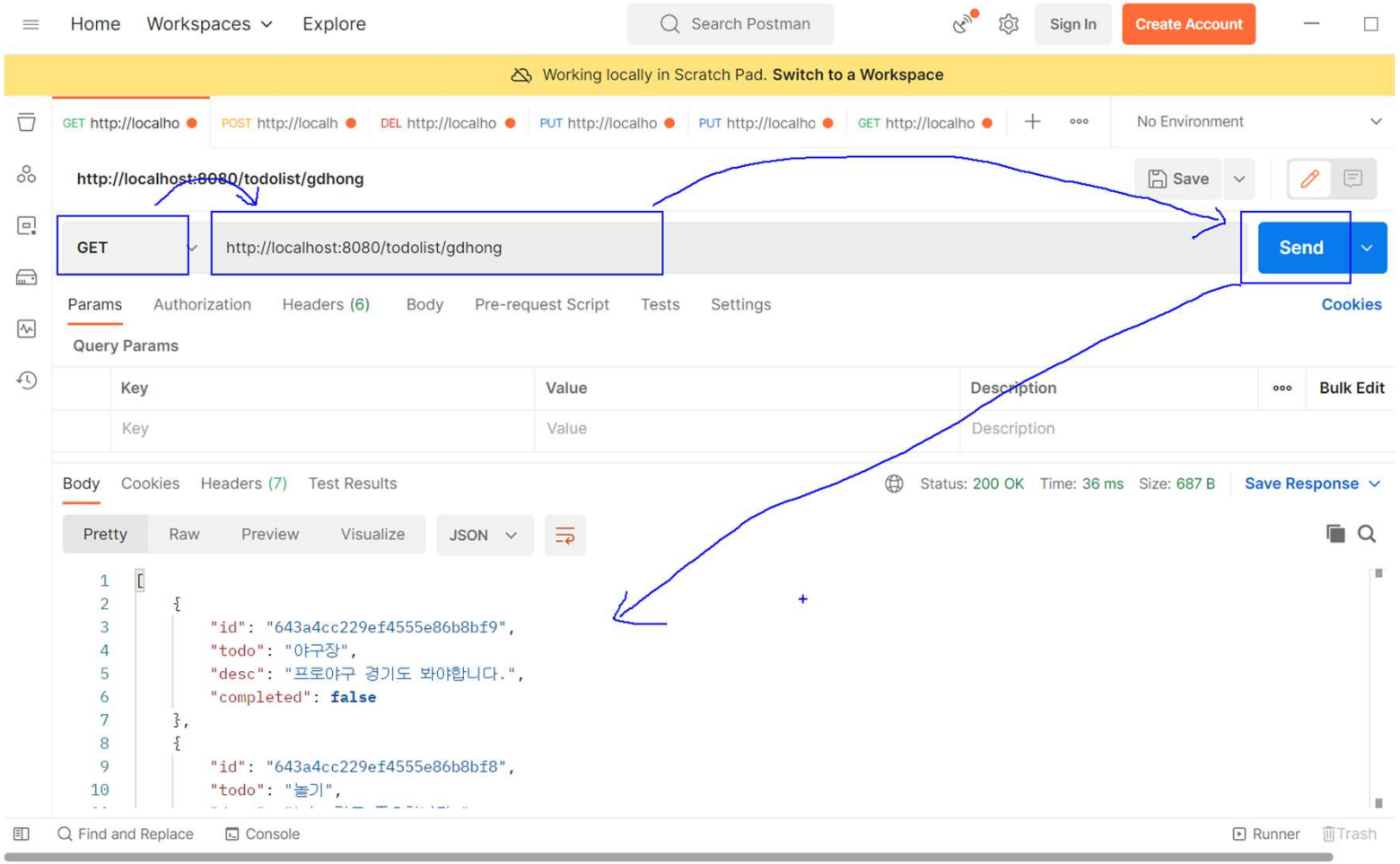
❖ express App 객체에 미들웨어 추가(이어서)

- ./server.js 변경

```
  app.use(router);

  app.listen(PORT, () => {
    console.log(`#### ${PORT} 에서 서버가 시작되었습니다`);
  });

};

startServer();
```

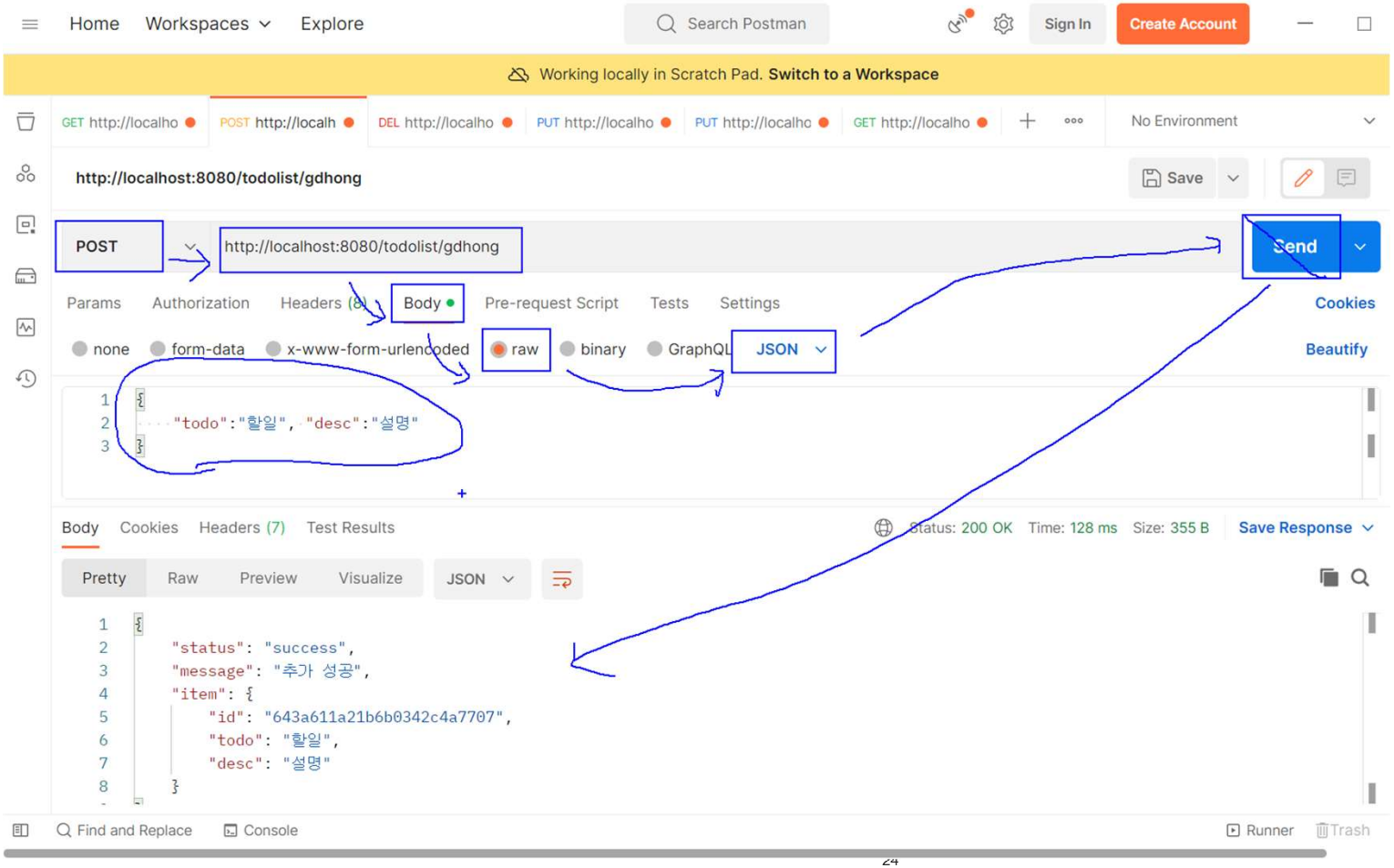
6.4 Postman 도구로 테스트하기

❖ GET /todolist/gdhong



6.4 Postman 도구로 테스트하기

❖ POST /todolist/gdhong



6.4 Postman 도구로 테스트하기

❖ PUT /todolist/gdhong/:id -> id는 실제 데이터의 것을 활용