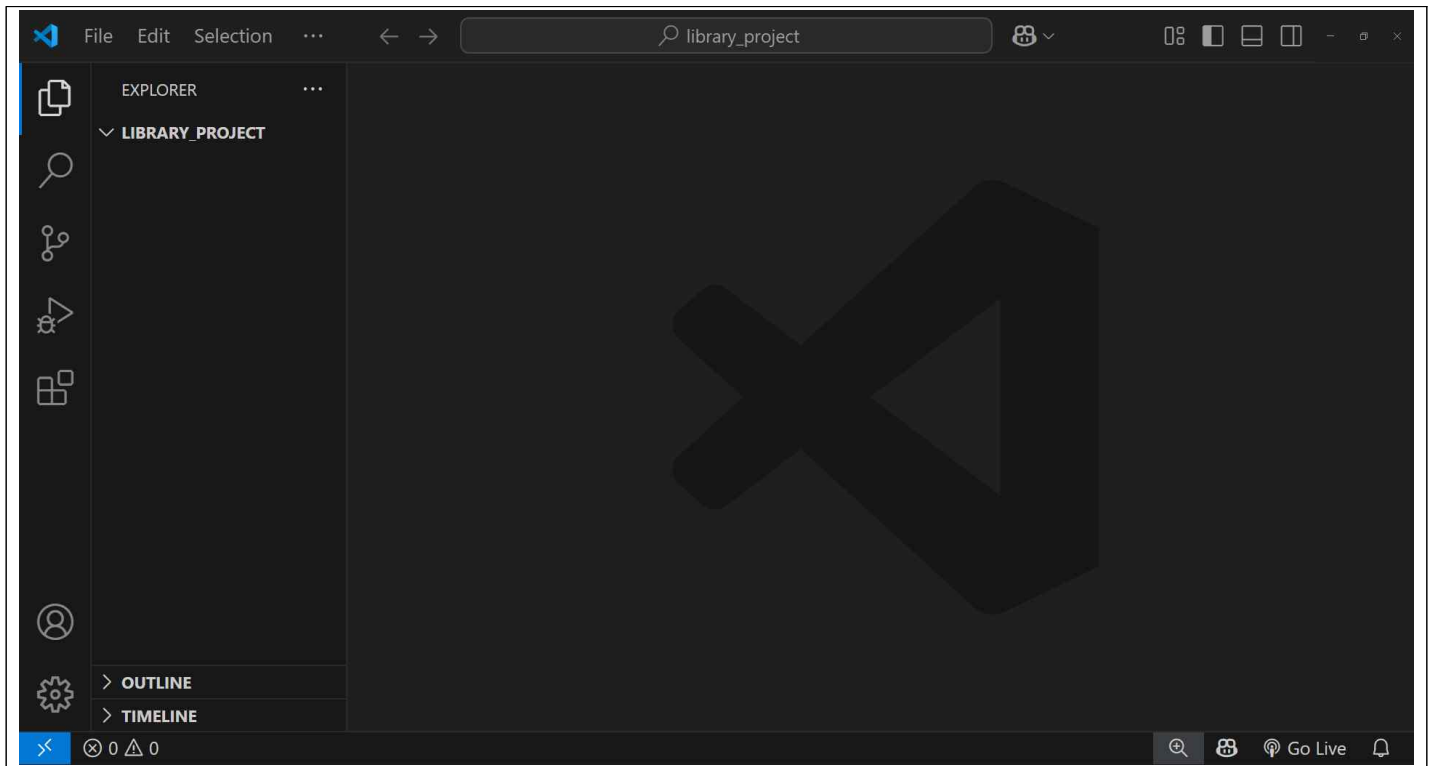


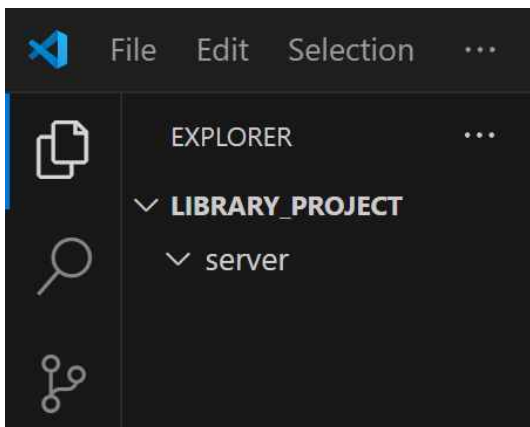
2. Application 구현

- Web Application 폴더 생성 : library_project



- Server 설정 : Node.js + Express.js

- 1) 폴더 생성 : server



2) npm 프로젝트 설정

- NPM(Node Project Manager) : Node 기반으로 생성되는 프로젝트를 관리하는 도구
- package.json : NPM 프로젝트의 설정 정보를 담고 있는 파일로 dependency 정보도 포함.

npm init

```
D:\dev\webworkspace\library_project>cd server
```

```
D:\dev\webworkspace\library_project\server>npm init
```

```
This utility will walk you through creating a package.json file.  
It only covers the most common items, and tries to guess sensible defaults.
```

```
See `npm help init` for definitive documentation on these fields  
and exactly what they do.
```

```
Use `npm install <pkg>` afterwards to install a package and  
save it as a dependency in the package.json file.
```

```
Press ^C at any time to quit.
```

```
package name: (server)
```

```
version: (1.0.0)
```

```
description:
```

```
entry point: (index.js)
```

```
test command:
```

```
git repository:
```

```
keywords:
```

```
author:
```

```
license: (ISC)
```

```
About to write to D:\dev\webworkspace\library_project\server\package.json:
```

```
{  
  "name": "server",  
  "version": "1.0.0",  
  "main": "index.js",  
  "scripts": {  
    "test": "echo \"Error: no test specified\" && exit 1"  
  },  
  "author": "",  
  "license": "ISC",  
  "description": ""  
}
```

```
Is this OK? (yes)
```

```
D:\dev\webworkspace\library_project\server>
```

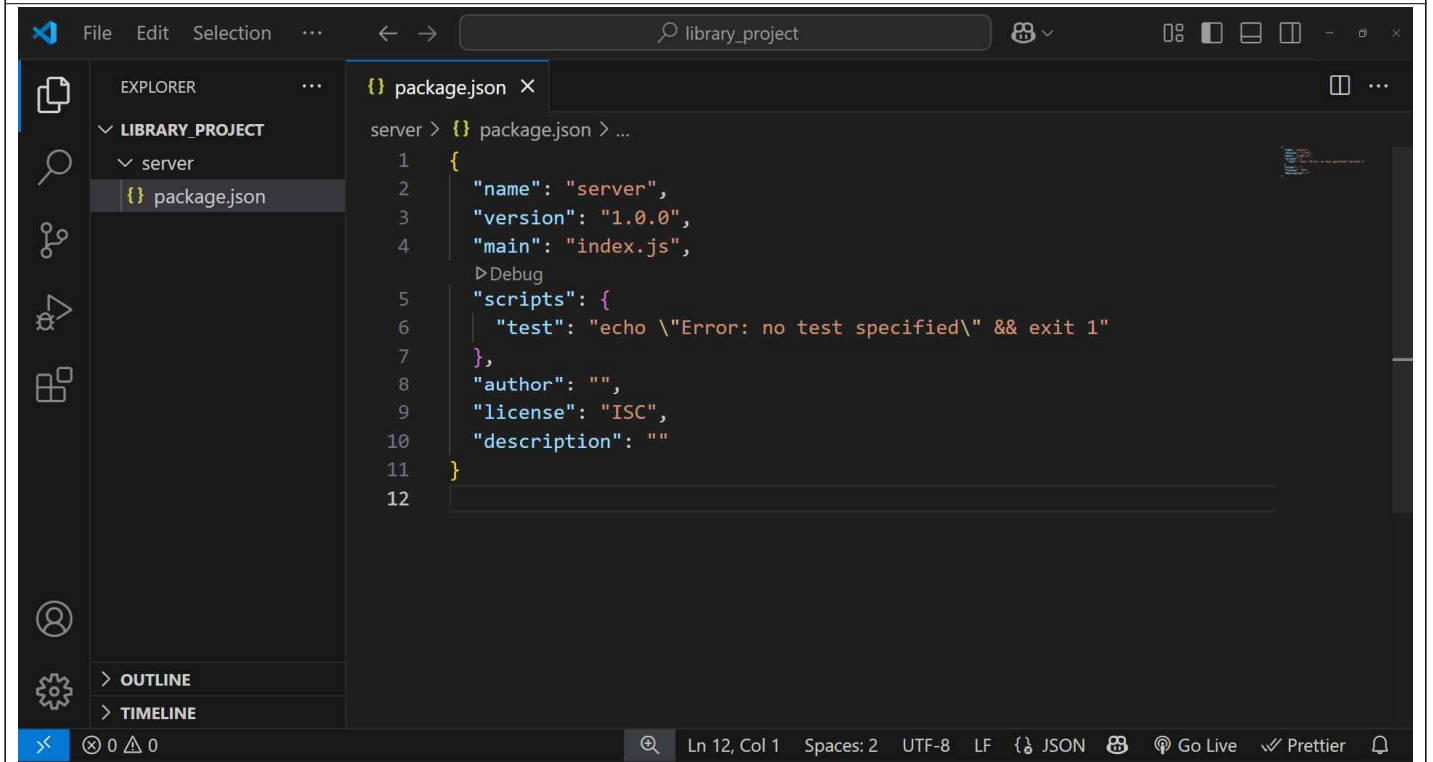
프로젝트 구성 설정으로

기본 설정을 그대로 사용할 경우

각 항목마다 enter을 입력

=> package.json 파일 생성

server/package.json



The screenshot shows the Visual Studio Code interface with a project named 'library_project'. The Explorer sidebar on the left shows the file structure: 'LIBRARY_PROJECT' > 'server' > 'package.json'. The main editor displays the content of 'package.json' for the 'server' directory. The JSON content is as follows:

```
1 {
2   "name": "server",
3   "version": "1.0.0",
4   "main": "index.js",
5   "scripts": {
6     "test": "echo \"Error: no test specified\" && exit 1"
7   },
8   "author": "",
9   "license": "ISC",
10  "description": ""
11 }
```

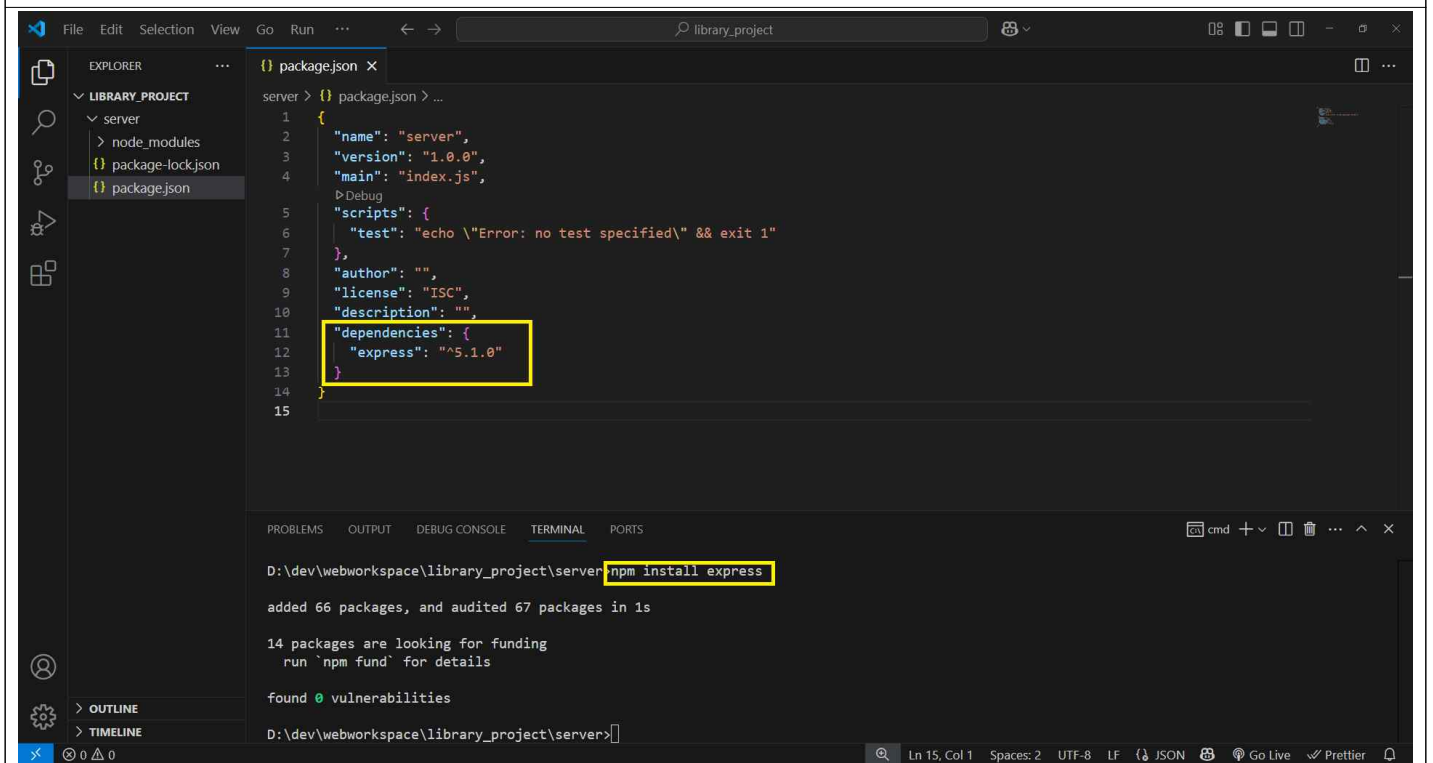
The status bar at the bottom indicates the cursor is at Line 12, Column 1, with 2 spaces, UTF-8 encoding, and LF line endings. It also shows the file is JSON and has Prettier formatting enabled.

3) 모듈 설치

· 기본 명령어 : `npm install module_name`

3.1. express : Node.js에서 웹 서버를 구축하는 데 사용하는 프레임워크

npm install express



This screenshot shows the same VS Code interface after running the command `npm install express`. The Explorer sidebar now includes a 'node_modules' directory under 'server'. The 'package.json' file has been updated with a new 'dependencies' section, which is highlighted with a yellow box:

```
1 {
2   "name": "server",
3   "version": "1.0.0",
4   "main": "index.js",
5   "scripts": {
6     "test": "echo \"Error: no test specified\" && exit 1"
7   },
8   "author": "",
9   "license": "ISC",
10  "description": "",
11  "dependencies": {
12    "express": "^5.1.0"
13  }
14 }
```

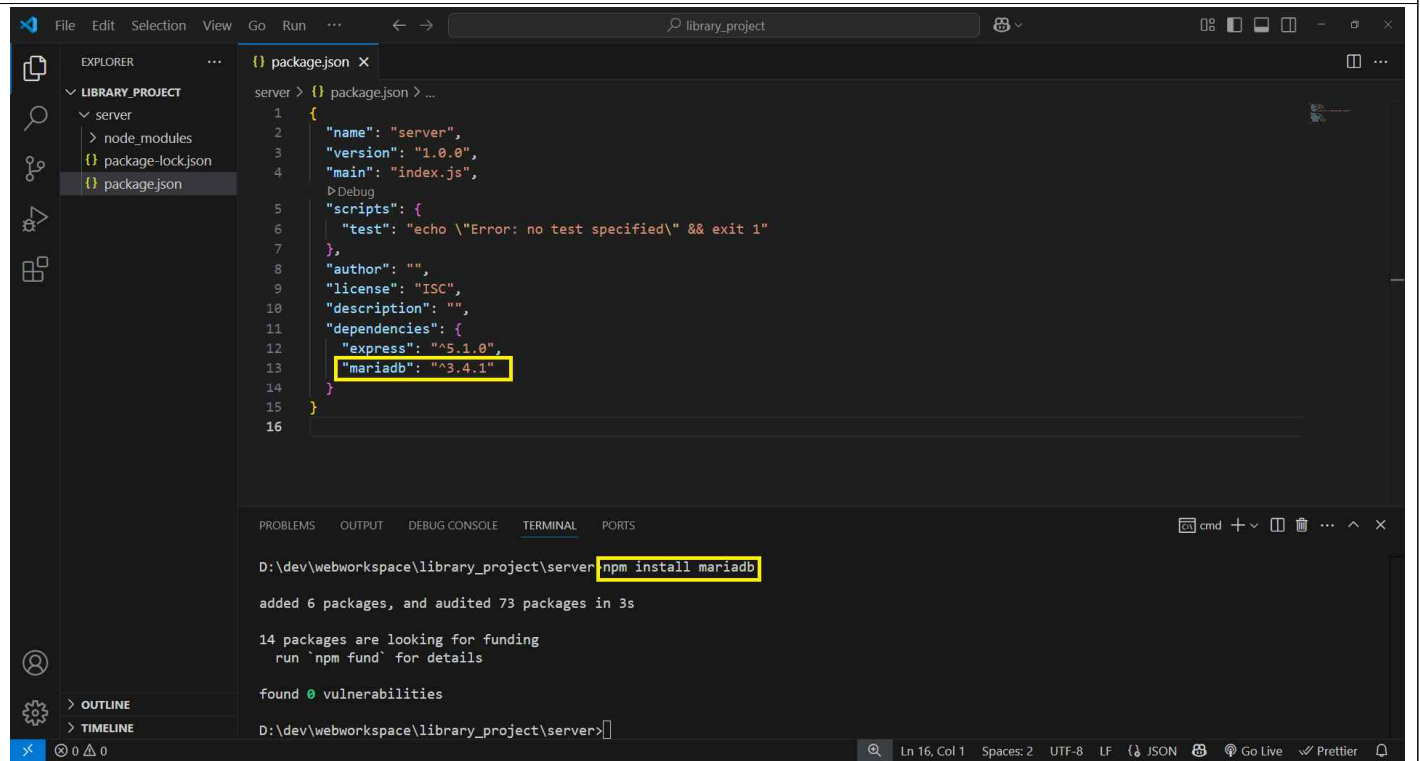
The Terminal at the bottom shows the output of the command:

```
D:\dev\workspace\library_project\server> npm install express
added 66 packages, and audited 67 packages in 1s
14 packages are looking for funding
  run 'npm fund' for details
found 0 vulnerabilities
D:\dev\workspace\library_project\server>
```

The status bar at the bottom now shows the cursor is at Line 15, Column 1.

3.2. mariadb : 관계형 데이터베이스 중 하나인 MariaDB를 활용하기 위한 모듈

npm install mariadb



The screenshot shows the Visual Studio Code interface with a project named 'library_project'. The Explorer panel on the left shows the file structure: 'LIBRARY_PROJECT' > 'server' > 'node_modules' > 'package-lock.json' and 'package.json'. The package.json file is open in the editor, showing the following content:

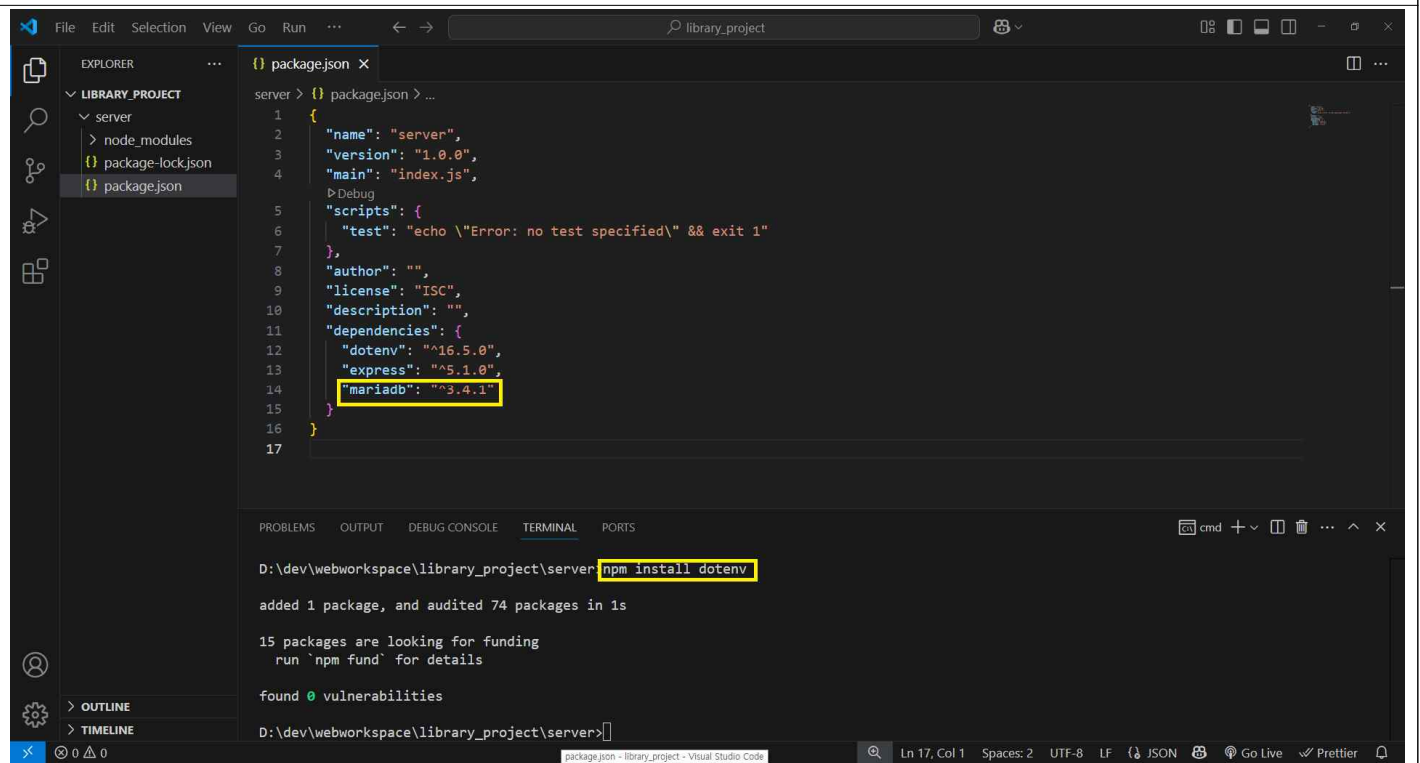
```
1 {
2   "name": "server",
3   "version": "1.0.0",
4   "main": "index.js",
5   "scripts": {
6     "test": "echo \"Error: no test specified\" && exit 1"
7   },
8   "author": "",
9   "license": "ISC",
10  "description": "",
11  "dependencies": {
12    "express": "^5.1.0",
13    "mariadb": "^3.4.1"
14  }
15 }
```

The 'mariadb' dependency is highlighted with a yellow box. The Terminal panel at the bottom shows the command 'npm install mariadb' being executed, with the output:

```
D:\dev\workspace\library_project\server> npm install mariadb
added 6 packages, and audited 73 packages in 3s
14 packages are looking for funding
run 'npm fund' for details
found 0 vulnerabilities
D:\dev\workspace\library_project\server>
```

3.3. dotenv : 환경변수를 관리하기 위한 모듈로 env파일을 읽어들이고 환경변수로 등록하는 모듈

npm install dotenv



The screenshot shows the Visual Studio Code interface with the same project 'library_project'. The package.json file is open in the editor, showing the following content:

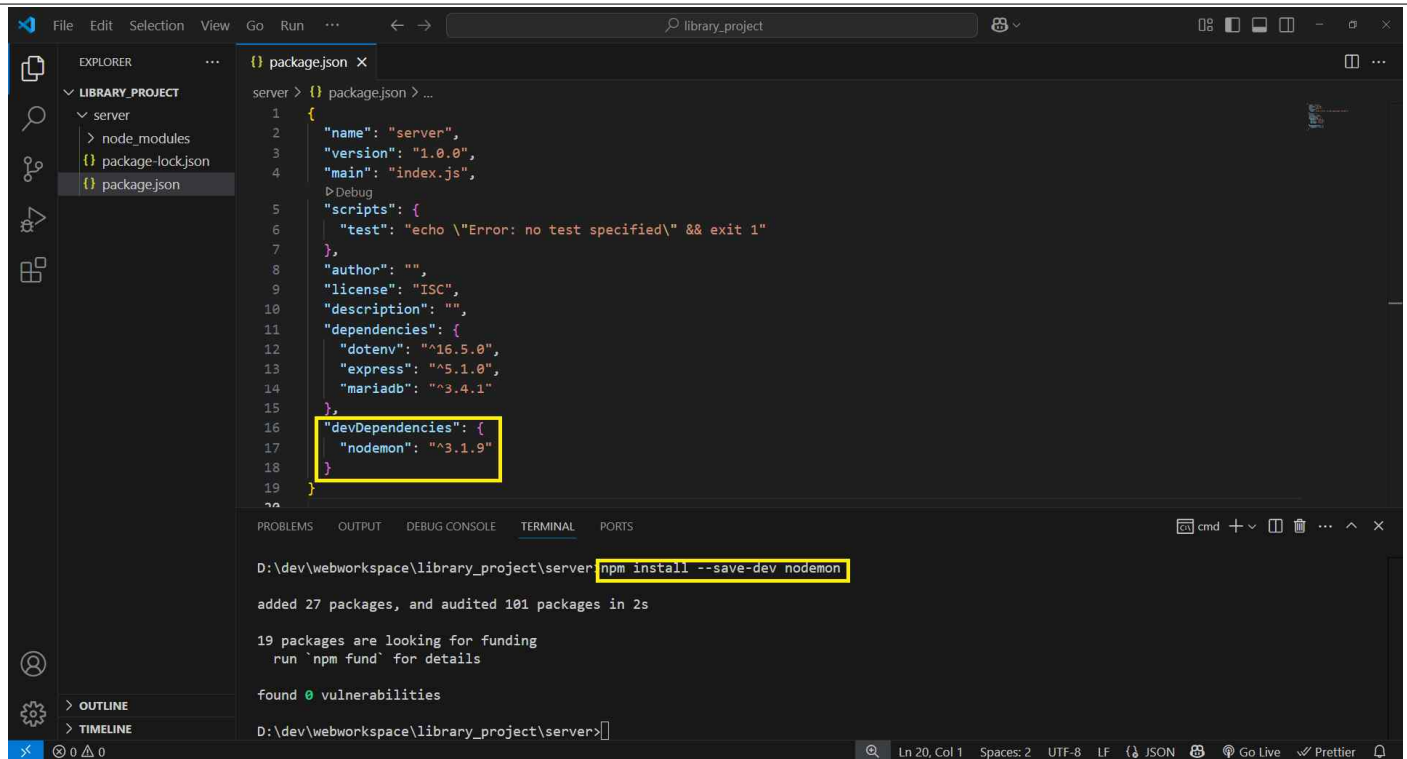
```
1 {
2   "name": "server",
3   "version": "1.0.0",
4   "main": "index.js",
5   "scripts": {
6     "test": "echo \"Error: no test specified\" && exit 1"
7   },
8   "author": "",
9   "license": "ISC",
10  "description": "",
11  "dependencies": {
12    "dotenv": "^16.5.0",
13    "express": "^5.1.0",
14    "mariadb": "^3.4.1"
15  }
16 }
```

The 'dotenv' dependency is highlighted with a yellow box. The Terminal panel at the bottom shows the command 'npm install dotenv' being executed, with the output:

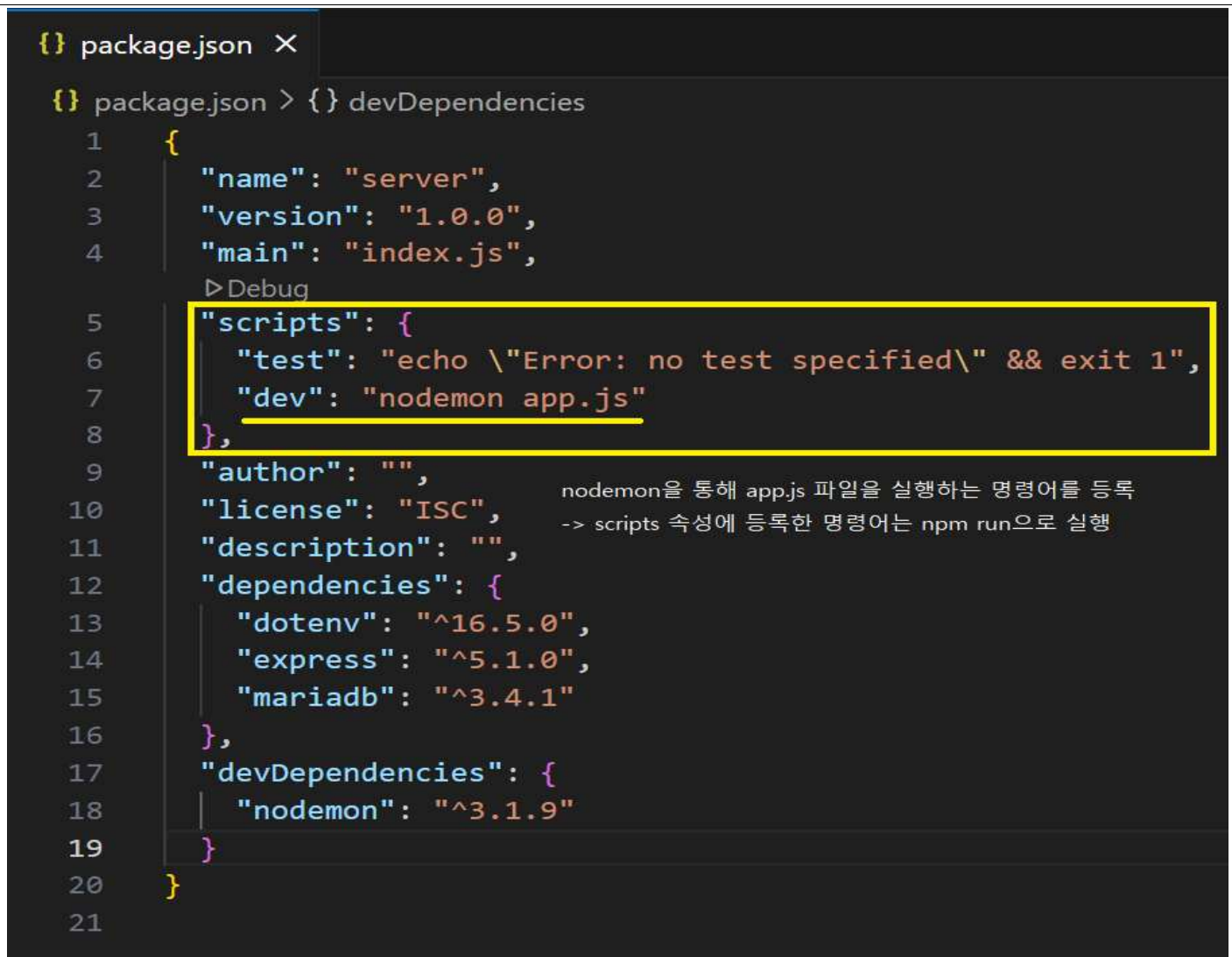
```
D:\dev\workspace\library_project\server> npm install dotenv
added 1 package, and audited 74 packages in 1s
15 packages are looking for funding
run 'npm fund' for details
found 0 vulnerabilities
D:\dev\workspace\library_project\server>
```

3.4. nodemon : 특정 파일이나 폴더의 변경을 감지해서 재시작하도록 도와주는 모듈, 개발에서만 사용하도록 설치

npm install --save-dev nodemon



server/package.json



프로젝트 실행 코드 : npm run dev

- server/app.js 파일 생성 후 실행

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

D:\dev\webworkspace\library_project\server>npm run dev

> server@1.0.0 dev
> nodemon app.js

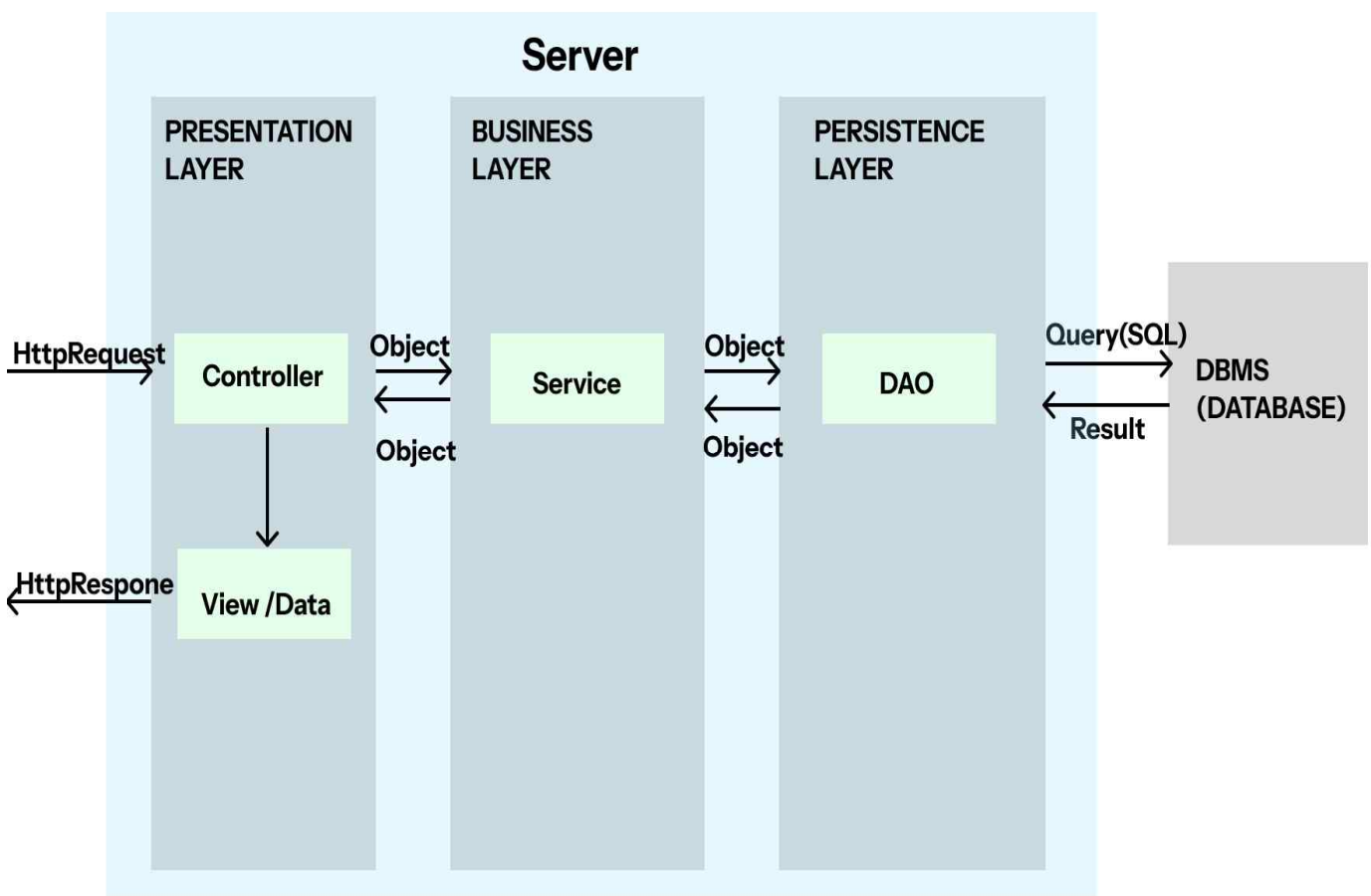
[nodemon] 3.1.9
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node app.js`
Server Start
http://localhost:3000
█
```

4) 프로젝트 구조

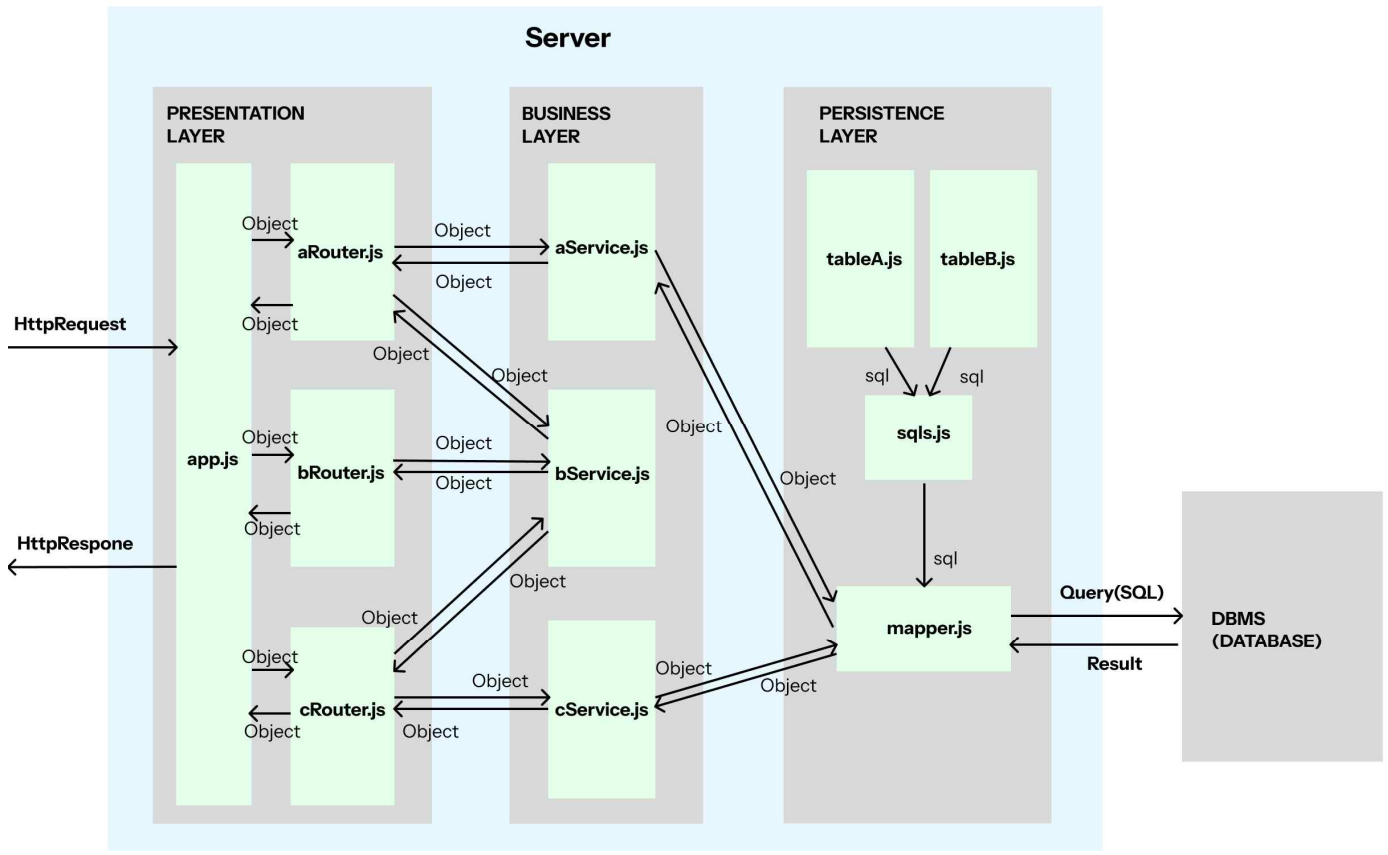
4.1. 3 Tier Architecture (3-tier Layered Architecture)

- Presentation Layer : 사용자와 상호작용하며 요청과 응답을 직접적으로 처리
- Business Layer : 실제 서비스인 비즈니스 로직을 처리
- Persistence Layer : 영속성을 가지는 DB에 접근해 데이터를 조회하거나 등록 등을 담당

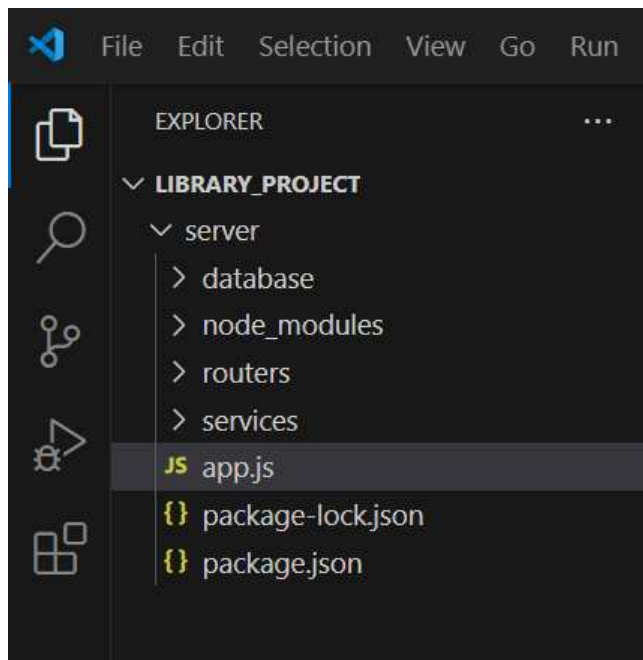
MVC 패턴과 함께 적용, 주로 Java를 기반으로 한 프로젝트에서 많이 사용



현재 프로젝트에 적용한 형태



4.2. 프로젝트 폴더 구조



4.3. 기본 파일 생성

- app.js : express를 기반으로 서버를 설정하고 실행하는 파일

server/app.js

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

// 미들웨어 등록 영역
// 1. body parser
// content-type : application/x-www-form-urlencoded
app.use(express.urlencoded({ extended:false}));
// content-type : application/json
app.use(express.json());

// Server 실행
app.listen(3000, ()=>{
  console.log('Server Start');
  console.log('http://localhost:3000');
})

// 라우팅 등록 영역
const bookRouter =require('./routers/book_router.js');

// 기본 라우팅
app.get('/', (req, res)=>{
  res.send('Welcome!!');
})

// 라우터 모듈 등록
app.use('/', bookRouter);
```

- router.js : 각 서비스를 제공하는 라우팅을 설정하는 파일

server/routers/book_router.js

```
const express =require('express');
// Express의 Router 모듈을 사용해서 라우팅 등록, 라우팅을 별도 파일로 관리
const router =express.Router();

// 해당 라우터를 통해 제공할 서비스를 가져옴
const bookService =require('../services/book_service.js');

// 라우팅 = 사용자의 요청(URL+METHOD) + Service + 응답형태(View or Data)

// 실제 라우팅 등록 영역

// 해당 javascript 파일의 마지막 코드, 모듈화
// 위에 선언한 기능(변수, 함수 등)들 중 외부로 노출할 대상을 설정
// => 다른 파일에서 require()을 통해 가져옴
module.exports =router
```


- service.js : 실제 제공하는 서비스를 정의하는 파일

server/services/book_service.js

```
// Service에서 필요하면 DB에 접속할 수 있도록 mapper를 가져옴
const mariadb=require("../database/mapper.js");

// 실제 제공할 서비스 등록 영역

module.exports={
  // 해당 객체에 등록해야지 외부로 노출
};
```

- mapper.js : DB에 접근해 데이터를 조회하거나 조작(등록, 수정, 삭제)를 담당하는 파일

server/database/mapper.js

```
// MariaDB에 접속할 모듈
const mariadb=require('mariadb/callback');
// DB에서 실행할 SQL문을 별도 파일로 작성
const sqlList=require('./sqlList.js');

// ConnectionPool 생성
const connectionPool=mariadb.createPool({
  // DB에 접속하는 정보
  host:localhost,
  port:3306,
  user:dev01,
  password:1234,
  database:dev,
  connectionLimit:10,

  // Object의 필드정보(Entiry)를 Query문에 있는 '?'에 자동변환 설정
  permitSetMultiParamEntries:true,
  // DML(insert, update, delete)를 실행할 경우
  // 반환되는 Object의 insertId 속성을 Number 타입으로 자동 변환
  insertIdAsNumber:true,
  // MariaDB의 데이터 타입 중 bigInt 타입을 Javascript의 Number 타입으로 자동 변환
  // 해당 타입을 Javascript에선 자동으로 변환하지 못함
  bigIntAsNumber:true,
  // logger 등록
  logger:{
    // 실제 실행되는 SQL문이 console.log로 출력되도록 설정
    query:console.log,
    // error 발생 시 처리함수
    error:console.log,
  }
});

// MariaDB에 SQL문을 보내고 결과를 받아올 함수 설정
// -> 실제로 동작하는 mariadb의 query 함수를 또 하나의 함수로 감싸는 방식으로
// 반복적인 작업을 효율적으로 처리하도록 함.
const query=(alias, values)=>{
  // alias : 각 테이블 별로 실행할 SQL문을 가지고 있는 변수
  // values : SQL문 안에 선언된 '?'들을 대체할 값의 집합
  // MaraiDB 모듈을 통해 설정한 ConnectionPool을 기반으로 SQL문 실행
  // -> 비동기 작업, Promise 방식
  // 비동기 작업) 작업의 요청과 결과가 동시에 일어나지 않으므로
```

```
//          요청한 작업의 결과를 언제 돌려받을지 알 수 없음
let conn = null;
try{
    conn = await connectionPool.getConnection();
    // SQL문 선택
    let executeSql = sqlList[alias];
    // SQL문을 실행할 결과를 처리
    let result = await conn.query(executeSql, values);
    return result;
}finally{
    if(conn) conn.release(); // Release to pool
}
};

module.exports = {
    query,
}
```

- sqls.js(sqlList.js) : 각 테이블별로 분리해서 관리하는 파일들을 하나로 합치는 역할

server/database/sqlList.js

```
// 각 테이블 별로 실행한 SQL문을 별도 파일로 작성
const books = require('./sqls/books.js');

module.exports = {
    // 펼침연산자(spread operator, ...)을 활용해 객체의 필드를 다른 객체로 쉽게 복사
    ...books,
}
```

- tablesNames.js : 각 테이블별로 실행할 SQL문을 관리하는 파일

server/databse/sqls/books.js

```
// Table : t_book_01

// 각 변수별로 SQL문을 등록할 때 백틱(`)을 사용하는 이유는 줄바꿈 허용을 허용하기 때문.
// ( 따옴표는 줄을 바꿀 경우 값이 깨지면서 에러발생 )

// 조건없이 전체조회
const selectBookList =
``;

// PRIMARY KEY를 활용한 단건조회
const selectBookOne =
``;

// 등록
const bookInsert =
``;

// 수정
const bookUpdate =
``;

// PRIMARY KEY를 활용한 삭제
const bookDelete =
``;
```

```
module.exports = {
  selectBookList,
  selectBookOne,
  bookInsert,
  bookUpdate,
  bookDelete,
}
```

· dbConfig.env : DB에 접속하는 정보를 별도로 관리하는 파일

server/database/configs/dbConfig.env

```
# env 파일은 단순히 값을 저장하는 파일로
# 일반적으로 보안과 관련된 정보나 환경에 따라 변경되는 정보를 따로 관리하기 위해 사용
# javascript 파일이 아니기 때문에 dotenv 모듈을 사용해서 env파일을 읽어들이는 코드가 반드시 필요!

# 컬렉션 중 하나인 Map처럼 Key와 value을 한쌍으로 저장

# 작성시 주의사항
# 1) 한쌍(Key와 Value)을 줄단위로 인식하므로 한 줄에 한쌍을 작성할 수 있도록 함
# 2) 어떤 경우에도 공백을 포함하지 않도록 주의함 (컴퓨터는 공백도 하나의 값으로 인식)
```

```
DB_HOST=localhost
DB_PORT=3306
DB_USER=dev01
DB_PWD=1234
DB_DB=dev
DB_LIMIT=10
```

- env 파일 추가 후 변경사항

server/app.js

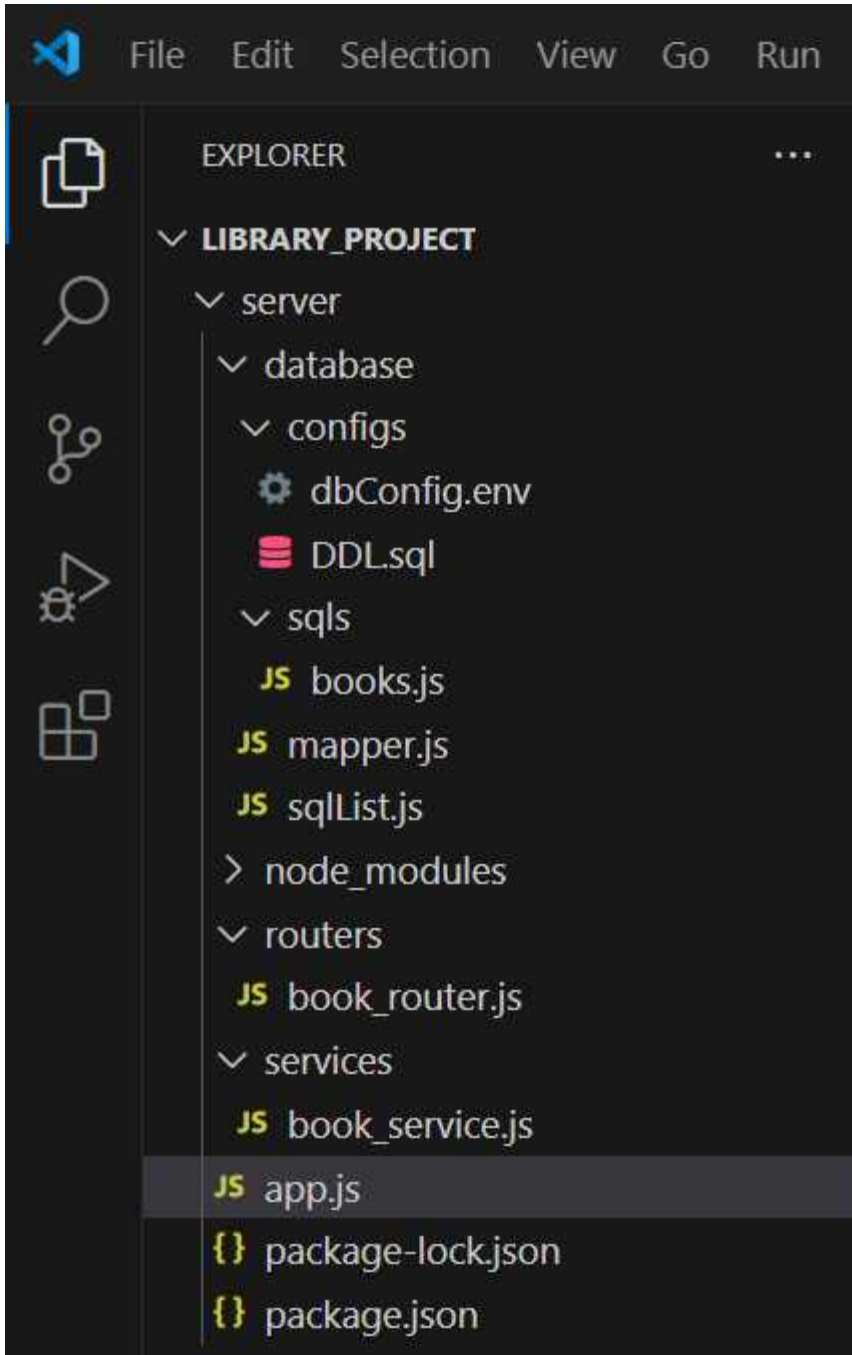
```
const express = require('express');
const app = express();

// env 파일을 읽어들이는 코드 => 가능한 가장 첫번째 줄에 작성
require('dotenv').config({path: './database/configs/dbConfig.env'});
const express = require('express');
const app = express();
```

server/database/mapper.js

<pre>// DB에 접속하는 정보 host:localhost, port:3306, user:dev01, password:1234, database:dev, connectionLimit:10,</pre>	<pre>// DV에 접속하는 정보를 별도 env파일로 처리 // -> 내장모듈(별도 설치가 필요없는 모듈)인 // process 모듈의 env 속성으로 접근 host:process.env.DB_HOST, port:process.env.DB_PORT, user:process.env.DB_USER, password:process.env.DB_PWD, database:process.env.DB_DB, connectionLimit:process.env.DB_LIMIT,</pre>
---	--

4.4. 생성된 프로젝트 및 파일 구조



5) 기능구현

5.1. HTTP 란?

- 참고사이트 : [HTTP 정리](#)

· HTTP

1. HyperText Transfer Protocol의 약자
2. 클라이언트와 서버 사이 일어나는 텍스트 기반의 데이터 교환에 대한 프로토콜(통신 규약, 약속)
3. 통신 시 Request와 Response 객체를 사용한다.

(Request(요청) : 클라이언트가 서버에 전달하는 메시지 / Response(응답) : 서버가 클라이언트에 반환하는 메시지)

· Request/Response 객체

1. Header와 Body라는 공통된 구조를 가지고 있다.
2. Header(헤더)
 - 통신에 대한 전반적인 내용을 가지고 있다. (URL, method, status, origin, content-type 등)
 - METHOD : GET, POST, PUT, DELETE 등이 있으며 이 중 POST와 PUT만 Body를 가진다.
 - ORIGIN : URL 중 프로토콜:IP:PORT, ORIGIN을 기준으로 SOP, CORS가 적용된다.

- CONTENT-TYPE : 전송하는 데이터 포맷을 의미하며 주로 Body를 사용하는 메소드에 적용된다.

application/x-www-form-urlencoded	<ul style="list-style-type: none"> - QueryString(질의문자열) : key=value&key=value - METHOD : GET, POST, PUT, DELETE
application/json	<ul style="list-style-type: none"> - JSON(Javascript Object Notation) : { "key" : "value" , "key" : 'value' , ... } or [] - METHOD : POST, PUT
multipart/form-data	<ul style="list-style-type: none"> - 멀티미디어 파일 - METHOD : POST, PUT

- STATUS : HTTP 응답 상태 코드, 각 상황에 따라 다른 코드 값을 가짐

주로 발생하는 에러		발생 순서	원인
400	Bad Request	4	요청한 데이터가 아닌 경우 - 필수 값이 없거나 데이터 타입이 맞지 않는 경우 - 날짜를 다루는 양식이 맞지 않는 경우
403	Forbidden	-	접근 권한이 없는 경우
404	Not Found	1	요청한 URL이 서버에 없는 경우
405	Method Not Allowed	2	URL에 지원되지 않는 Method인 경우
415	Unsupported Media Type	3	지원되지 않는 Content-type인 경우
500	Internal Server Error	5	정상적으로 요청했지만 서버 내부에서 처리 중 에러 발생

4. Body : 특정 메서드에서 데이터를 전달하기 위해 사용하는 영역

7) 구현결과

· Boomerang - SOAP & REST Client

1. 크롬의 확장프로그램으로 View(화면)없이 서버에 요청과 응답을 할 수 있음.
2. 각 Request에 대해 왼쪽이 Request를, 오른쪽이 Response를 의미.
3. METHOD(GET, POST, PUT, DELETE)에 따라 Body 영역의 사용을 다르게 설정.

전체조회

The screenshot shows the Boomerang interface with a GET request to `http://localhost:3000/books`. The response body is displayed in the 'Raw' tab, showing a JSON array of three book objects. The first object has `no: 100`, `name: "아침 그리고 저녁"`, `writer: "온 포세"`, `publisher: "문학동네"`, `publication_date: "2019-09-25T15:00:00.000Z"`, and `info: "연젠가는 사라져 존재하지 않겠지만 그래 여기 머물러라"`. The second object has `no: 101`, `name: "그대들 어떻게 살 것인가"`, `writer: "요시노 겐자부로"`, `publisher: "양철북"`, `publication_date: "2012-06-27T15:00:00.000Z"`, and `info: "100년 가까이 사랑받아 온 청소년 인생론의 고전"`. The third object has `no: 102`, `name: "모순"`, `writer: "양귀자"`, `publisher: "쓰다"`, `publication_date: "2013-03-31T15:00:00.000Z"`, and `info: "인생은 탐구하면서 살아가는 것이 아니라, 살아가면서 탐구하는 것이"`.

단건조회

The screenshot shows the Boomerang interface with a GET request to `http://localhost:3000/books/100`. The response body is displayed in the 'Raw' tab, showing a single book object with `no: 100`, `name: "아침 그리고 저녁"`, `writer: "온 포세"`, `publisher: "문학동네"`, `publication_date: "2019-09-25T15:00:00.000Z"`, and `info: "연젠가는 사라져 존재하지 않겠지만 그래 여기 머물러라"`.

등록

library_project

book 전제조회 단건조회 등록 수정 삭제

SERVICES COLLECTION POST http://localhost:3000/books SEND

New Folder New Request

book

- GET 전제조회
- GET 단건조회
- POST 등록
- PUT 수정
- DEL 삭제

BODY HEADERS PARAMS AUTH SCRIPTS

Raw Form Multipart Beautify JSON

```
1 {
2   "name": "홍학의 자리",
3   "writer": "정해연",
4   "publisher": "엘릭시르",
5   "publication_date": "2025-04-21",
6   "info": "이 행록이 영원할 거라고 생각한 적은 없었다. 그러나 이런
7   끝을 상상한 적도 없었다."
8 }
```

BODY HEADERS TESTS 200 21 ms 42 bytes

Raw Preview

```
1 {
2   "isSucceeded": true,
3   "bookNo": 110
4 }
```

수정

library_project

book 전제조회 단건조회 등록 수정 삭제

SERVICES COLLECTION PUT http://localhost:3000/books/110 SEND

New Folder New Request

book

- GET 전제조회
- GET 단건조회
- POST 등록
- PUT 수정
- DEL 삭제

BODY HEADERS PARAMS AUTH SCRIPTS

Raw Form Multipart Beautify JSON

```
1 {
2   "name": "홍학의 자리",
3   "writer": "정해연",
4   "publisher": "엘릭시르 출판사",
5   "publication_date": "2021-07-25",
6   "info": "이 행록이 영원할 거라고 생각한 적은 없었다. 그러나 이런
7   끝을 상상한 적도 없었다."
8 }
```

BODY HEADERS TESTS 200 13 ms 233 bytes

Raw Preview

```
1 {
2   "isUpdated": true,
3   "bookInfo": {
4     "name": "홍학의 자리",
5     "writer": "정해연",
6     "publisher": "엘릭시르 출판사",
7     "publication_date": "2021-07-25",
8     "info": "이 행록이 영원할 거라고 생각한 적은 없었다. 그러나 이
9     런 끝을 상상한 적도 없었다.",
10    "no": "110"
11  }
12 }
```

삭제

Boomerang

Boomerang - SOAP & REST Client

chrome-extension://eipdrjedkpcnimmddfdkgfpljanehloah/index.html#/

library_project

book전제조회단건조회등록수정삭제

No Environment

SEND

SERVICESCOLLECTION

New FolderNew Request

book

- GET 전제조회
- GET 단건조회
- POST 등록
- PUT 수정
- DEL 삭제

DELETEhttp://localhost:3000/books/110

BODYHEADERSPARAMSAUTHSCRIPTS

RawFormMultipart

1

BODYHEADERSTESTS

2009 ms62 bytes

RawPreview

```
1 {
2   "affectedRows": 1,
3   "insertId": 0,
4   "warningStatus": 0
5 }
```