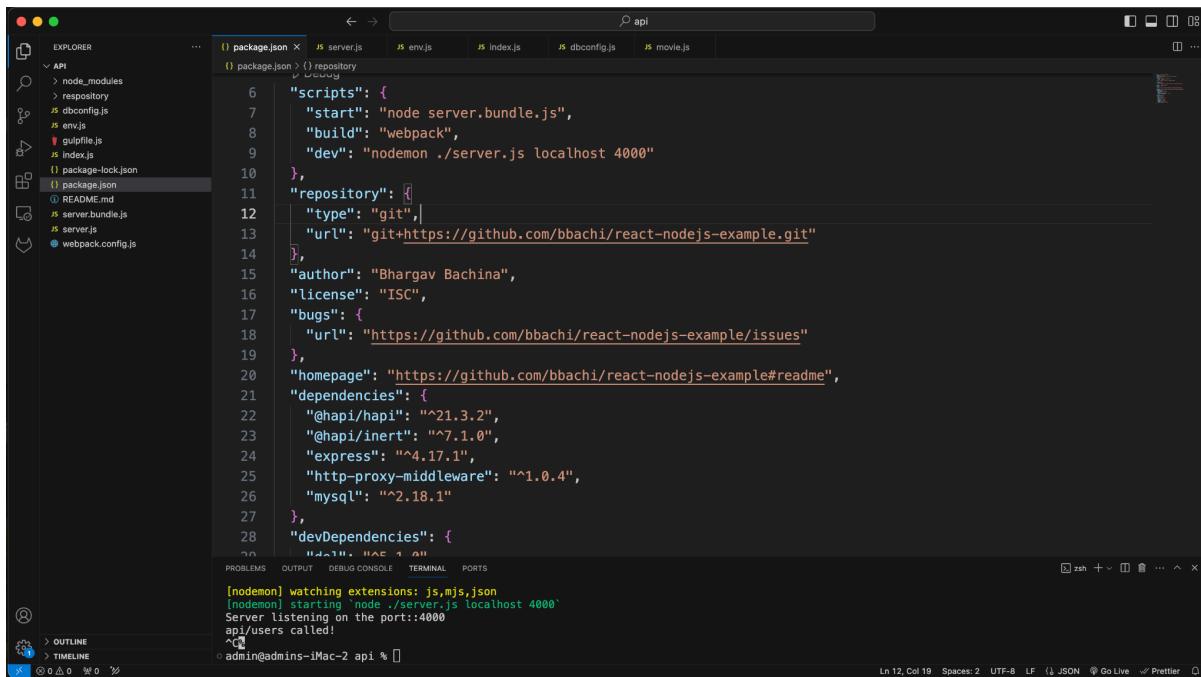


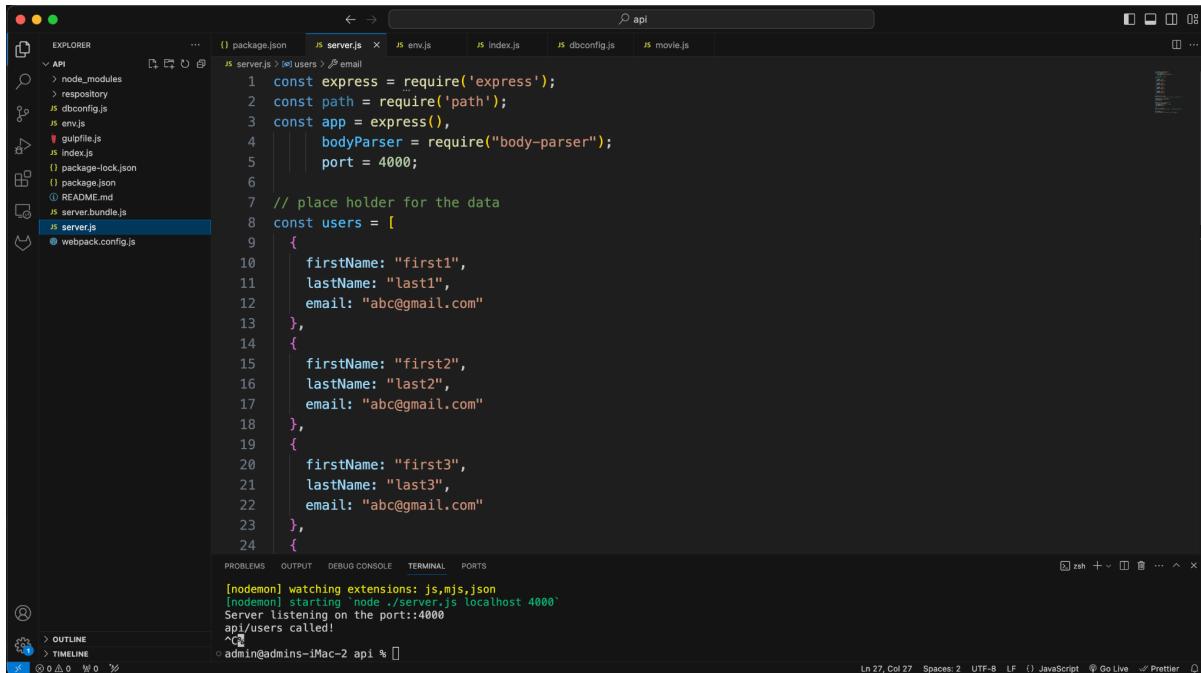
LAB4 ENGSE203

เข้า vs code API เปลี่ยน Port ตรง package.json และ server.js เป็น Port 4000



```
6 "scripts": {
7   "start": "node server.bundle.js",
8   "build": "webpack",
9   "dev": "nodemon ./server.js localhost 4000"
10 },
11 "repository": {
12   "type": "git",
13   "url": "git+https://github.com/bbachi/react-nodejs-example.git"
14 },
15 "author": "Bhargav Bachina",
16 "license": "ISC",
17 "bugs": {
18   "url": "https://github.com/bbachi/react-nodejs-example/issues"
19 },
20 "homepage": "https://github.com/bbachi/react-nodejs-example#readme",
21 "dependencies": {
22   "@hapi/hapi": "^21.3.2",
23   "@hapi/inert": "7.1.0",
24   "express": "4.17.1",
25   "http-proxy-middleware": "^1.0.4",
26   "mysql": "^2.18.1"
27 },
28 "devDependencies": {
29   "nodemon": "1.18.3"
30 }
```

[nodemon] watching extensions: js,mjs,json
[nodemon] starting 'node ./server.js localhost 4000'
Server listening on the port::4000
api/users called!



```
1 const express = require('express');
2 const path = require('path');
3 const app = express(),
4   bodyParser = require("body-parser");
5 port = 4000;
6
7 // place holder for the data
8 const users = [
9   {
10     firstName: "first1",
11     lastName: "last1",
12     email: "abc@gmail.com"
13   },
14   {
15     firstName: "first2",
16     lastName: "last2",
17     email: "abc@gmail.com"
18   },
19   {
20     firstName: "first3",
21     lastName: "last3",
22     email: "abc@gmail.com"
23   },
24 ];
```

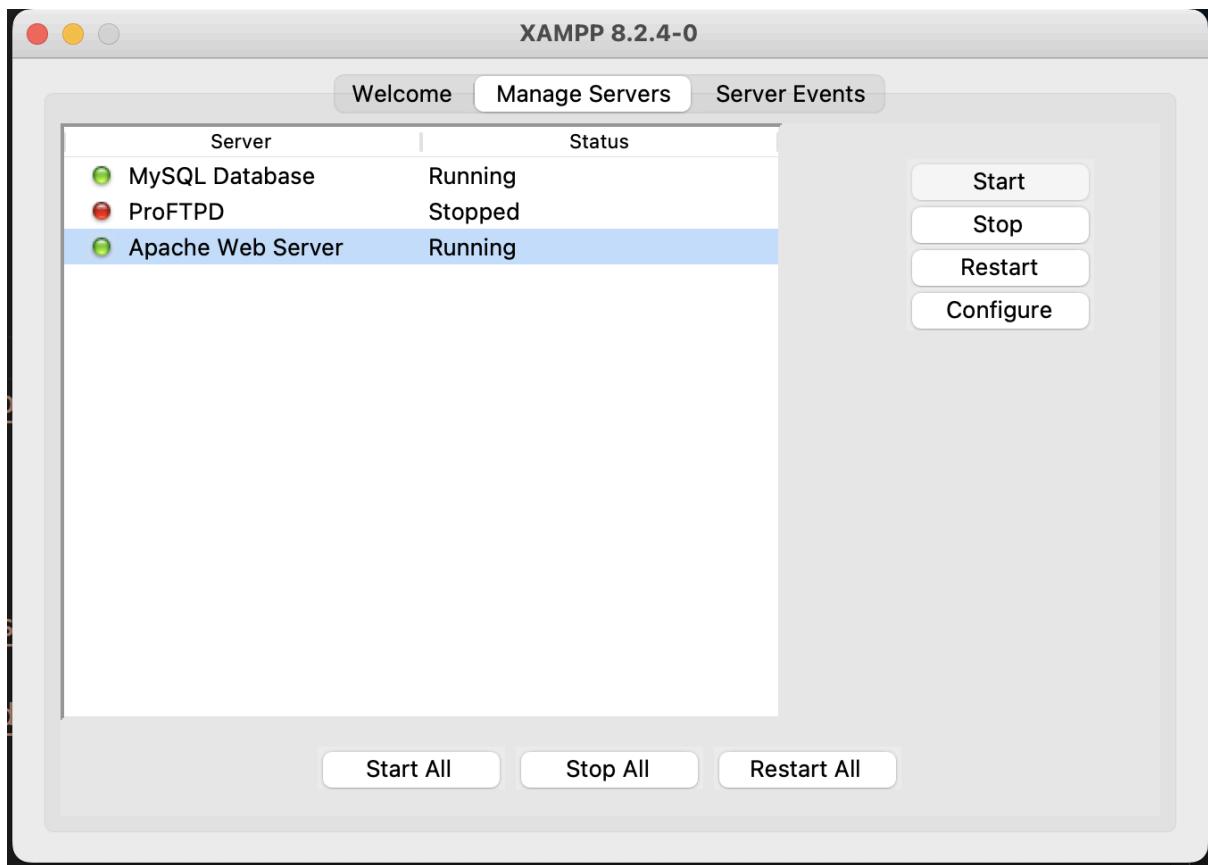
[nodemon] watching extensions: js,mjs,json
[nodemon] starting 'node ./server.js localhost 4000'
Server listening on the port::4000
api/users called!

ແລ້ວ Run ດູ

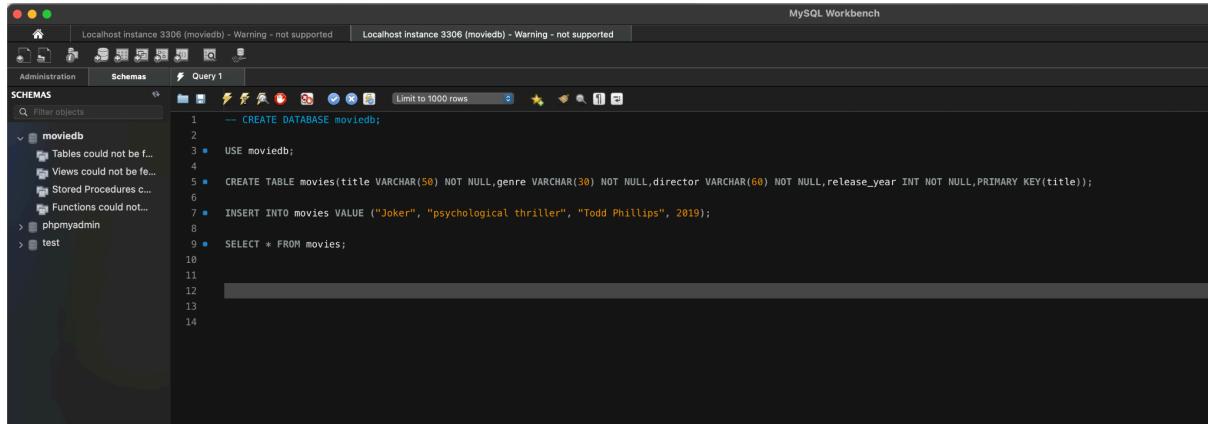


```
{{"firstName": "first1", "lastName": "last1", "email": "abc@gmail.com"}, {"firstName": "first2", "lastName": "last2", "email": "abc@gmail.com"}, {"firstName": "first3", "lastName": "last3", "email": "abc@gmail.com"}, {"firstName": "first4", "lastName": "last4", "email": "abc@gmail.com"}}
```

ເປີດ Xampp

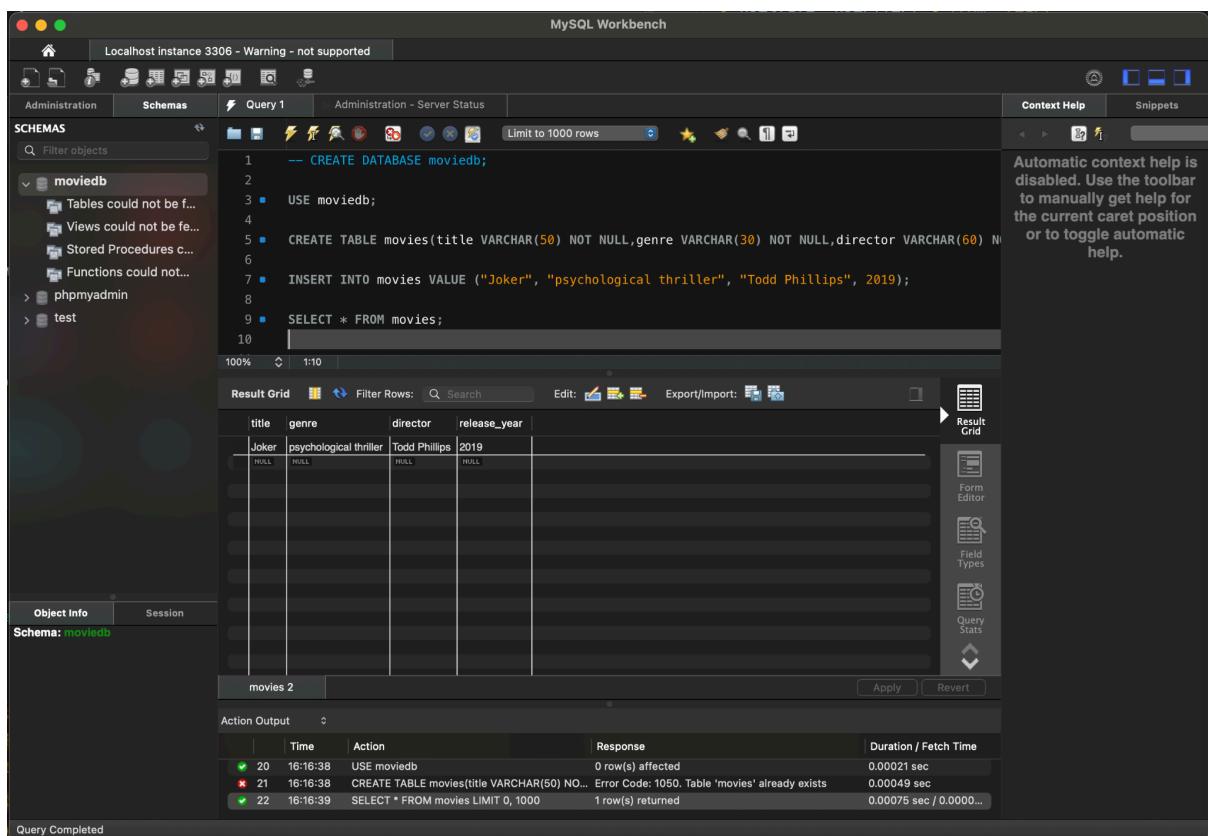


เข้า MySQL Workbench และสร้าง Database (moviedb) และเพิ่มข้อมูลลงไป



```
-- CREATE DATABASE moviedb;
USE moviedb;
CREATE TABLE movies(title VARCHAR(50) NOT NULL,genre VARCHAR(30) NOT NULL,director VARCHAR(60) NOT NULL,release_year INT NOT NULL,PRIMARY KEY(title));
INSERT INTO movies VALUE ("Joker", "psychological thriller", "Todd Phillips", 2019);
SELECT * FROM movies;
```

แล้วแสดงข้อมูล



```
-- CREATE DATABASE moviedb;
USE moviedb;
CREATE TABLE movies(title VARCHAR(50) NOT NULL,genre VARCHAR(30) NOT NULL,director VARCHAR(60) NOT NULL,release_year INT NOT NULL,PRIMARY KEY(title));
INSERT INTO movies VALUE ("Joker", "psychological thriller", "Todd Phillips", 2019);
SELECT * FROM movies;
```

title	genre	director	release_year
Joker	psychological thriller	Todd Phillips	2019
NULL	NULL	NULL	NULL

Object Info Session Schema: moviedb

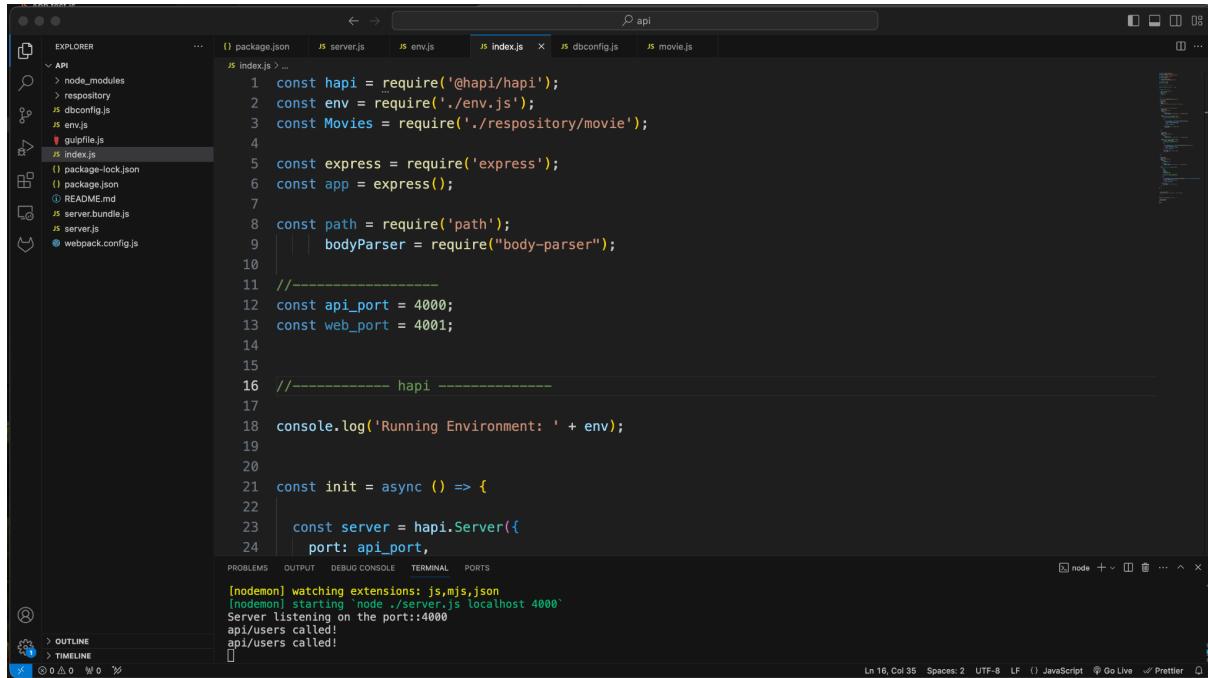
Action Output

Time	Action	Response	Duration / Fetch Time
20 16:16:38	USE moviedb	0 row(s) affected	0.00021 sec
21 16:16:38	CREATE TABLE movies(title VARCHAR(50) NO... Error Code: 1050. Table 'movies' already exists	Error Code: 1050. Table 'movies' already exists	0.00049 sec
22 16:16:39	SELECT * FROM movies LIMIT 0, 1000	1 row(s) returned	0.00075 sec / 0.000...

Query Completed

vscode API

-index.js



A screenshot of the Visual Studio Code interface. The left sidebar shows a project structure with files like package.json, server.js, env.js, index.js, dbconfig.js, movie.js, and others. The main editor tab is on index.js, which contains code for setting up an Express server and a Hapi API. The terminal tab shows output from nodemon starting the server at port 4000. The bottom status bar indicates the file is 16 lines long, 35 columns wide, in UTF-8 encoding, and uses LF line endings.

```
const hapi = require('@hapi/hapi');
const env = require('./env.js');
const Movies = require('../repository/movie');

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

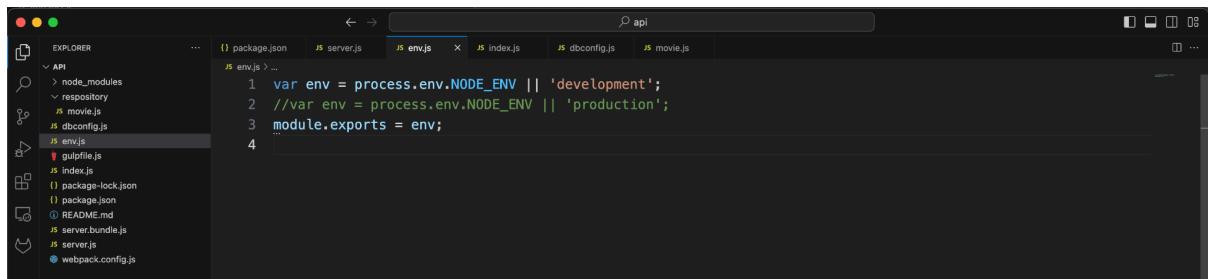
const path = require('path');
const bodyParser = require("body-parser");

//-----
const api_port = 4000;
const web_port = 4001;

//----- hapi -----
console.log('Running Environment: ' + env);

const init = async () => {
  const server = hapi.Server({
    port: api_port,
```

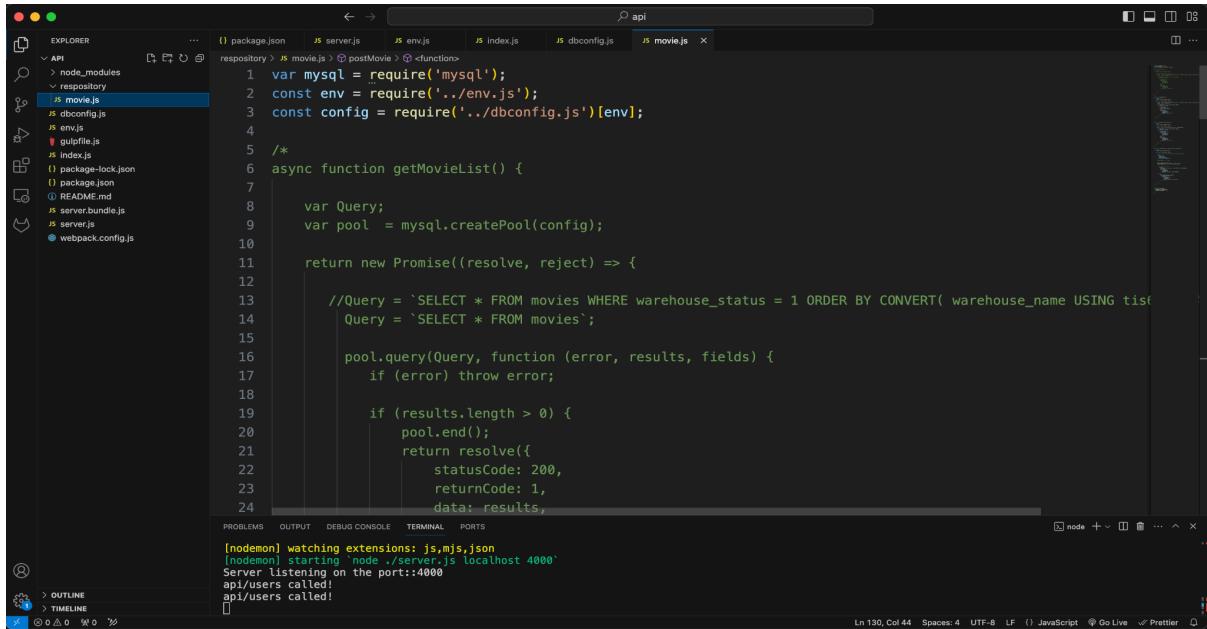
-env.js



A screenshot of the Visual Studio Code interface. The left sidebar shows a project structure with files like package.json, server.js, env.js, index.js, dbconfig.js, movie.js, and others. The main editor tab is on env.js, which sets the NODE_ENV variable to either 'development' or 'production' based on process.env.NODE_ENV. The terminal tab shows the output of nodemon starting the server at port 4000. The bottom status bar indicates the file is 4 lines long.

```
var env = process.env.NODE_ENV || 'development';
//var env = process.env.NODE_ENV || 'production';
module.exports = env;
```

-สร้างไฟล์ movie.js ใน Folder repository



The screenshot shows the VS Code interface with the following details:

- Explorer View:** Shows the project structure with a folder named "repository" containing "node_modules", "movie.js", "dbconfig.js", "env.js", "index.js", "package-lock.json", "README.md", "server.bundle.js", "server.js", and "webpack.config.js".
- Editor View:** The "movie.js" file is open, displaying code for interacting with a MySQL database to get movie lists.
- Terminal View:** Shows the command "node ./server.js" running successfully on port 4000, with the message "Server listening on the port:4000" and "api/users called!" appearing twice.
- Status Bar:** Shows the file is 130 lines long, has 4 spaces per tab, and is in UTF-8 encoding.

```
var mysql = require('mysql');
const env = require('../env.js');
const config = require('../dbconfig.js')[env];

/*
async function getMovieList() {

    var Query;
    var pool = mysql.createPool(config);

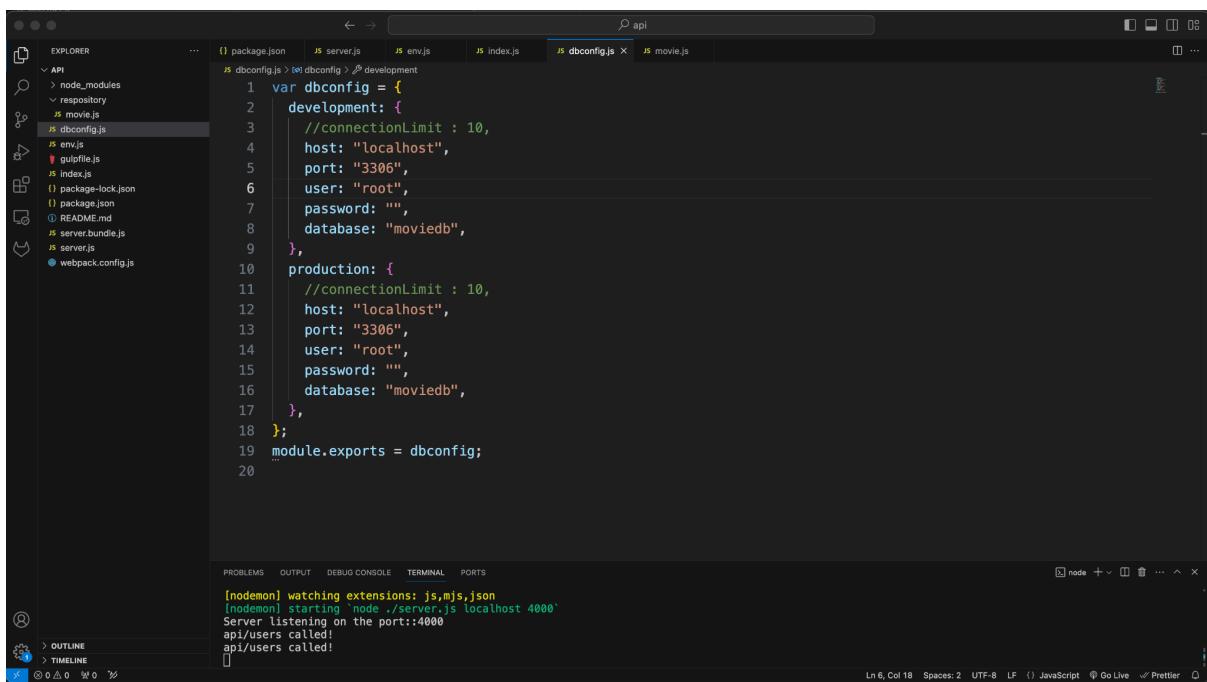
    return new Promise((resolve, reject) => {

        //Query = `SELECT * FROM movies WHERE warehouse_status = 1 ORDER BY CONVERT(warehouse_name USING tifinn)`; 
        Query = `SELECT * FROM movies`;

        pool.query(Query, function (error, results, fields) {
            if (error) throw error;

            if (results.length > 0) {
                pool.end();
                return resolve({
                    statusCode: 200,
                    returnCode: 1,
                    data: results,
                });
            }
        });
    });
}
```

dbconfig.js



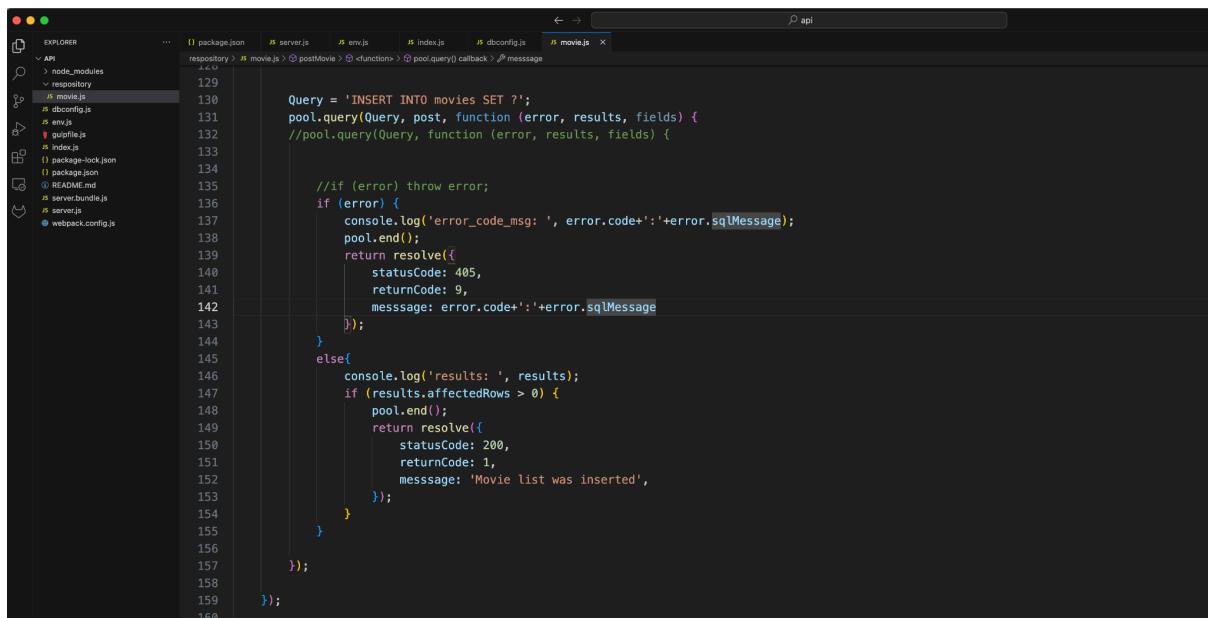
The screenshot shows the VS Code interface with the following details:

- Explorer View:** Shows the project structure with a folder named "repository" containing "node_modules", "movie.js", "dbconfig.js", "env.js", "index.js", "package-lock.json", "README.md", "server.bundle.js", "server.js", and "webpack.config.js".
- Editor View:** The "dbconfig.js" file is open, defining a "dbconfig" object with "development" and "production" environments.
- Terminal View:** Shows the command "node ./server.js" running successfully on port 4000, with the message "Server listening on the port:4000" and "api/users called!" appearing twice.
- Status Bar:** Shows the file is 6 lines long, has 2 spaces per tab, and is in UTF-8 encoding.

```
var dbconfig = {
    development: {
        connectionLimit : 10,
        host: "localhost",
        port: "3306",
        user: "root",
        password: "",
        database: "moviedb",
    },
    production: {
        connectionLimit : 10,
        host: "localhost",
        port: "3306",
        user: "root",
        password: "",
        database: "moviedb",
    },
};

module.exports = dbconfig;
```

แก้ไขไฟล์ movie.js



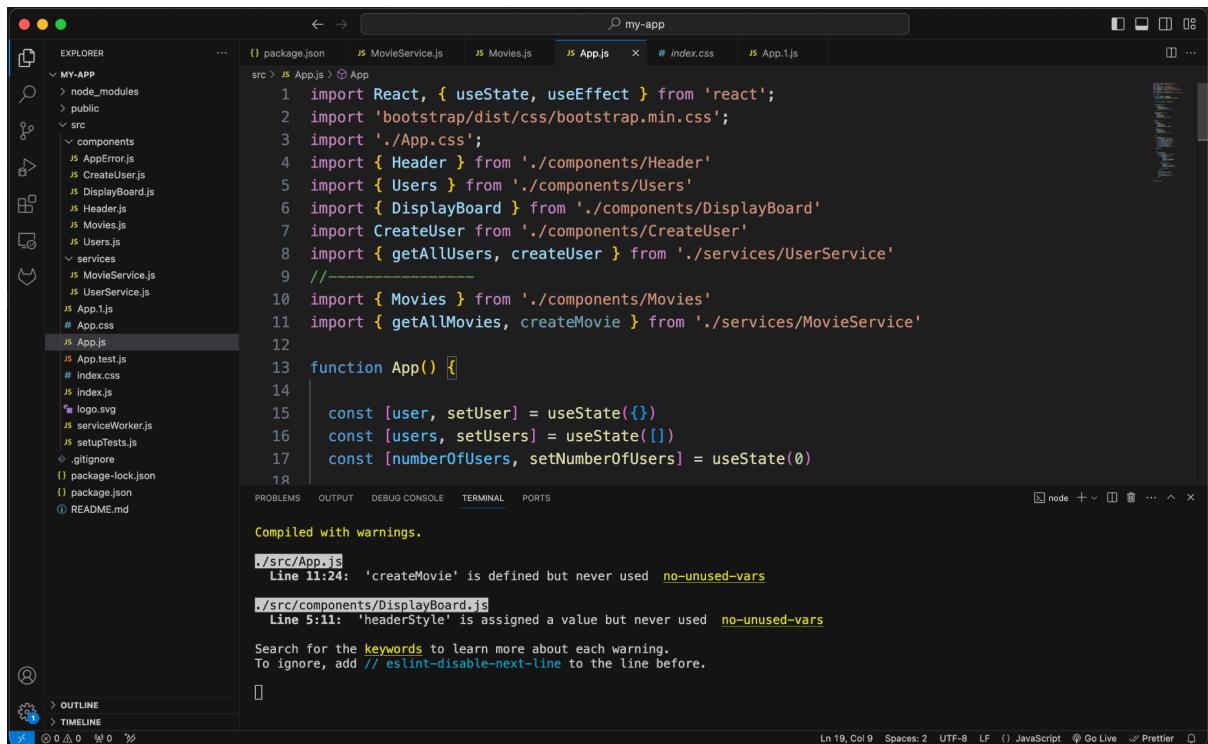
The screenshot shows the VS Code interface with the file `movie.js` open in the editor. The code is a Node.js script for inserting data into a database. It includes imports for `pool.query`, `postMovie`, and `<function>`. The logic handles errors and returns appropriate status codes and messages.

```
Query = 'INSERT INTO movies SET ?';
pool.query(Query, post, function (error, results, fields) {
  //pool.query(Query, function (error, results, fields) {

    //if (error) throw error;
    if (error) {
      console.log('error_code_msg: ', error.code+' '+error.sqlMessage);
      pool.end();
      return resolve({
        statusCode: 405,
        returnCode: 9,
        message: error.code+' '+error.sqlMessage
      });
    }
    else{
      console.log('results: ', results);
      if (results.affectedRows > 0) {
        pool.end();
        return resolve({
          statusCode: 200,
          returnCode: 1,
          message: 'Movie list was inserted',
        });
      }
    }
  });
});
```

vscode My-app

App.js



The screenshot shows the VS Code interface with the file `App.js` open in the editor. The code is a React component that uses hooks like `useState` and `useEffect` to manage state. It imports components from `./components` and services from `./services`. The code includes several imports for `React`, `bootstrap`, and various component files. There are also imports for `MovieService` and `UserService`.

```
import React, { useState, useEffect } from 'react';
import 'bootstrap/dist/css/bootstrap.min.css';
import './App.css';
import { Header } from './components/Header'
import { Users } from './components/Users'
import { DisplayBoard } from './components/DisplayBoard'
import CreateUser from './components/CreateUser'
import { getAllUsers, createUser } from './services/UserService'
//-----
import { Movies } from './components/Movies'
import { getAllMovies, createMovie } from './services/MovieService'

function App() {
  const [user, setUser] = useState({})
  const [users, setUsers] = useState([])
  const [numberOfUsers, setNumberOfUsers] = useState(0)
```

Compiled with warnings.

```
Line 11:24: 'createMovie' is defined but never used no-unused-vars
Line 5:11: 'headerStyle' is assigned a value but never used no-unused-vars
```

Search for the [keywords](#) to learn more about each warning.
To ignore, add `// eslint-disable-next-line` to the line before.

ແລ້ວຮັນໂດມົນ

http://localhost:3001

The screenshot shows a web application interface. At the top, there's a red header bar with the text "React With NodeJS". Below it, on the left, is a "Create User" form with fields for First Name (test), Last Name (test), and Email (test@demo.com). A "Create" button is at the bottom of the form. On the right, there's a green box titled "Users Created" containing the number "5" and a "Get all Users" button. Below these, there's a table titled "Users" with columns: User Id, Firstname, Lastname, and Email. The table contains five rows of data.

User Id	Firstname	Lastname	Email
1	first1	last1	abc@gmail.com
2	first2	last2	abc@gmail.com
3	first3	last3	abc@gmail.com
4	jirapat	tanuwoharn	abc@gmail.com
5	test	test	test@demo.com

POSTman

GET
<http://localhost:4000/api/users>

The screenshot shows the Postman application interface. On the left, the "My Workspace" sidebar lists collections like ENGSE203 and ENGSE203, and environments. In the center, a "getUsers" request is selected under the "LAB4" collection. The request URL is "http://localhost:4000/api/users". The "Body" tab of the response panel shows a JSON array of user objects:

```
[{"firstName": "first1", "lastName": "last1", "email": "abc@gmail.com"}, {"firstName": "first2", "lastName": "last2", "email": "abc@gmail.com"}, {"firstName": "first3", "lastName": "last3", "email": "abc@gmail.com"}, {"firstName": "first4", "lastName": "last4", "email": "abc@gmail.com"}]
```

POST

<http://localhost:4000/api/user>

POST <http://localhost:4000/api/user>

```
1 {
2   "user": {
3     "firstName": "jirapat",
4     "lastName": "tanuwoharn",
5     "email": "abc@gmail.com"
6   }
}
```

Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize JSON

1 "user added"

GET

<http://localhost:4000/>

GET <http://localhost:4000/>

Params Authorization Headers (9) Body Pre-request Script Tests Settings

Query Params

Key	Value	Description	...	Bulk Edit
Key	Value	Description		

Body Cookies Headers (9) Test Results

Pretty Raw Preview Visualize HTML

1 <h3> Welcome to API Back-end Ver. 1.0.0</h3>

GET

<http://localhost:4000/api/movie/all>

The screenshot shows the Postman application interface. On the left, the 'My Workspace' sidebar lists collections, environments, and history. Under the 'ENGSE203 / LAB4' collection, the 'getMovie' endpoint is selected. The main panel displays the request details: method 'GET', URL 'http://localhost:4000/api/movie/all', and various tabs like Params, Authorization, Headers, Body, Pre-request Script, Tests, and Settings. Below the request, the response section shows a status of 200 OK with a response time of 10 ms and a body size of 400 B. The response body is displayed in Pretty JSON format:

```
1 [  
2 {  
3   "title": "Joker",  
4   "genre": "psychological thriller",  
5   "director": "Todd Phillips",  
6   "release_year": 2019  
7 }  
8 ]
```

GET

http://localhost:4000/api/movie/search?search_text

The screenshot shows the Postman application interface. On the left, the 'My Workspace' sidebar lists collections, environments, and history. Under the 'ENGSE203 / LAB4' collection, the 'searchMovie' endpoint is selected. The main panel displays the request details: method 'GET', URL 'http://localhost:4000/api/movie/search?search_text', and various tabs like Params, Authorization, Headers, Body, Pre-request Script, Tests, and Settings. Below the request, the response section shows a status of 200 OK with a response time of 68 ms and a body size of 442 B. The response body is displayed in Pretty JSON format:

```
1 {  
2   "statusCode": 200,  
3   "returnCode": 1,  
4   "data": [  
5     {  
6       "title": "Joker",  
7       "genre": "psychological thriller",  
8       "director": "Todd Phillips",  
9       "release_year": 2019  
10     }  
11   ]  
12 }
```

POST

<http://localhost:4000/api/moive/insert>

The screenshot shows the Postman application interface. On the left, the 'My Workspace' sidebar lists collections, environments, and history. In the center, a specific collection named 'ENGSE203 / LAB4' is selected, and a sub-collection 'LAB4' is expanded, showing several API endpoints: 'getUsers', 'addUser', 'hello', 'getMovie', 'searchMovie', and 'insertMovie'. The 'insertMovie' endpoint is currently selected and highlighted with a blue border. The main workspace displays a POST request to 'http://localhost:4000/api/moive/insert'. The 'Body' tab is active, showing a JSON payload:

```
1 {  
2   "title" : "test2",  
3   "genre" : "t",  
4   "director" : "t",  
5   "release_year" : 2024  
6 }
```

Below the request, the response details are shown: status 200 OK, duration 87 ms, and size 349 B. The response body is also displayed in JSON format:

```
1 {  
2   "statusCode": 200,  
3   "returnCode": 1,  
4   "message": "Movie list was inserted"  
5 }
```

66543210006-3 ຈິරກັທຣ ທນໂວຫາຣ