

## LAB 4

เปลี่ยน Port ของ backend เป็น 4000 ที่หน้า package.json

```
package.json > {} scripts > [abc] dev
You, 23 minutes ago | 2 authors (bbachi and others)
1  {
2    "name": "react-nodejs-example",
3    "version": "1.0.0",
4    "description": "example project react with nodejs",
5    "main": "server.js",
6    "scripts": {
7      "start": "node server.bundle.js",
8      "build": "webpack",
9      "dev": "nodemon ./index.js localhost 4000"
10  },
```

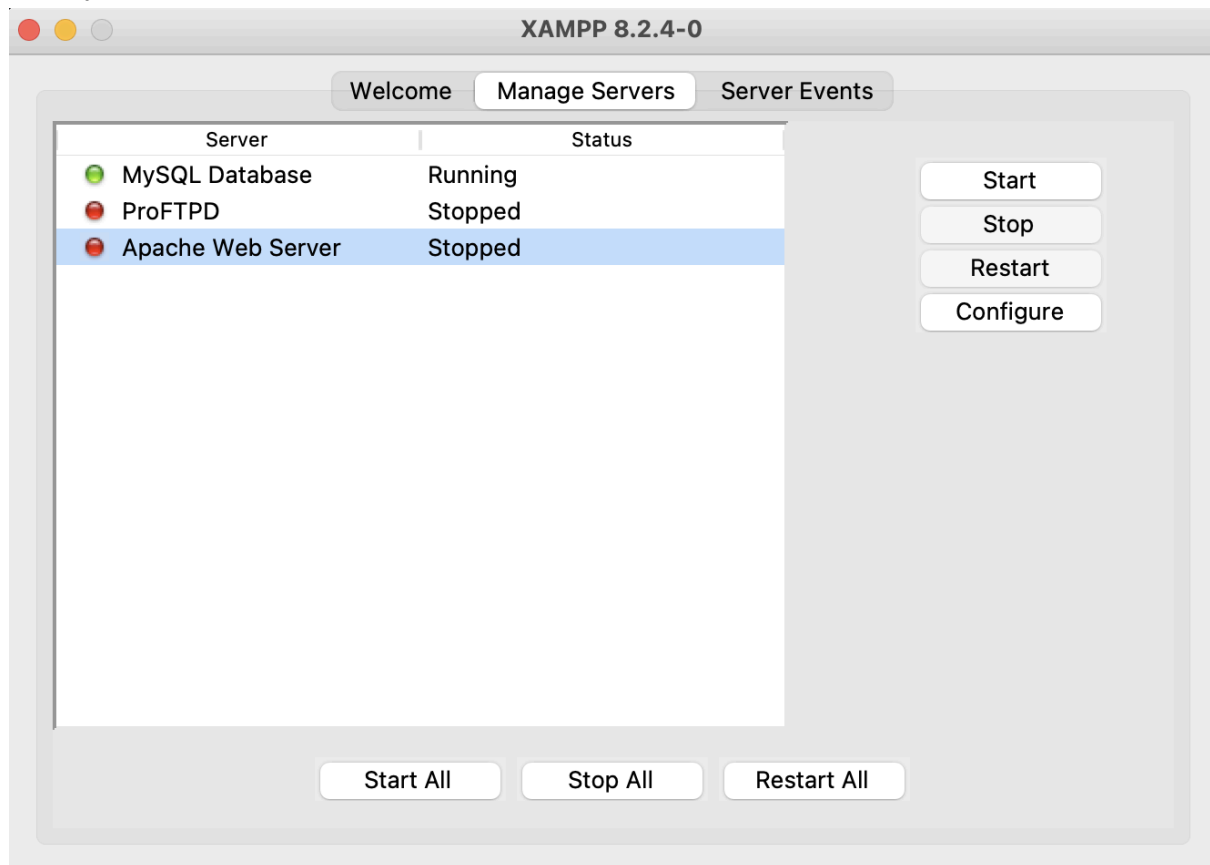
เปลี่ยนเป็น port 4000 ที่หน้า server.js

```
server.js > [?] users
You, 5 hours ago | 2 authors (bbachi and others)
1  const express = require('express');
2  const path = require('path');
3  const app = express(),
4    bodyParser = require("body-parser");
5  port = 4000;
6
```

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

```
localhost:4000/api/users
[{"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":"nunthiphat","lastName":"sombatwong","email":"abc@gmail.com"}]
```

## เปิด MySQL ใน Xampp



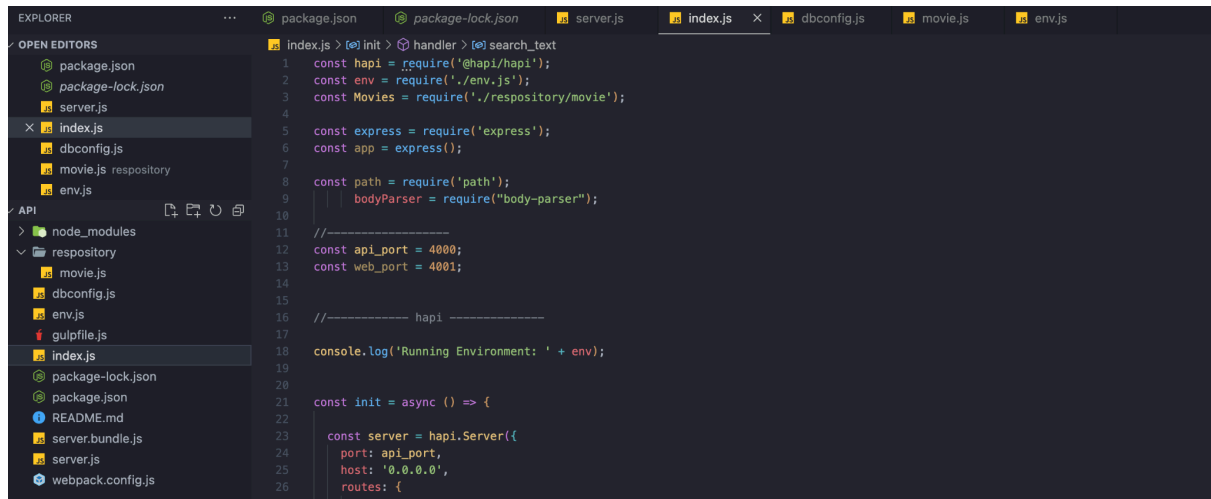
สร้าง database , เพิ่มตารางและเพิ่มข้อมูล

```
1 -- CREATE DATABASE moviedb;
2
3 • USE moviedb;
4
5 -- 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));
6
7 -- INSERT INTO movies VALUE ("Joker", "psychological thriller", "Todd Phillips", 2019);
8
9 • SELECT * FROM movies;
```

SELECT \* FROM movies; แสดงข้อมูล

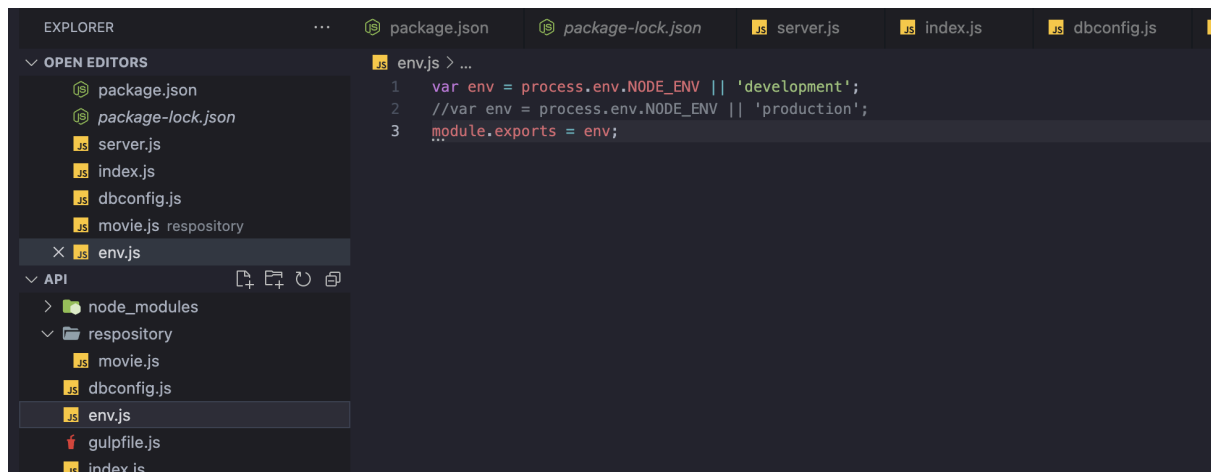
[illegible]

## สร้างหน้า index.js มาใน backend และใส่โค้ด



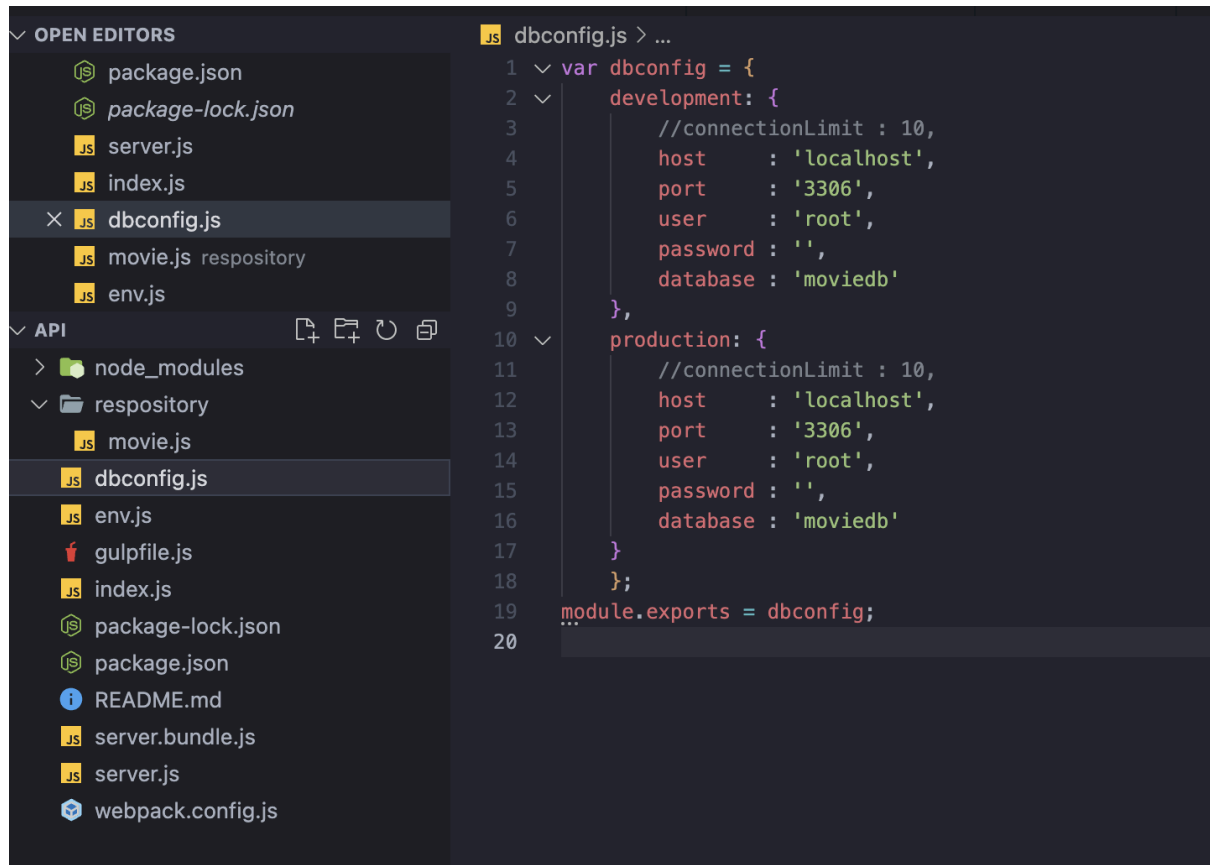
```
1 const hapi = require('@hapi/hapi');
2 const env = require('./env.js');
3 const Movies = require('./respository/movie');
4
5 const express = require('express');
6 const app = express();
7
8 const path = require('path');
9 | | bodyParser = require("body-parser");
10
11 //-----
12 const api_port = 4000;
13 const web_port = 4001;
14
15 //----- hapi -----
16
17 console.log('Running Environment: ' + env);
18
19 const init = async () => {
20
21   const server = hapi.Server({
22     port: api_port,
23     host: '0.0.0.0',
24     routes: {
25       accept: true
26     }
27   });
```

## สร้างหน้า env.js เพื่อเก็บ config



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

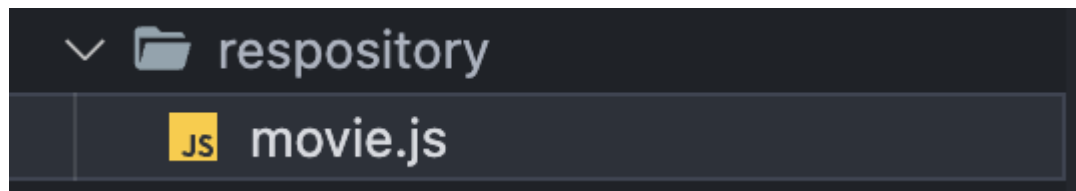
## สร้างหน้า dbconfig.js



The screenshot shows the VS Code interface. On the left, the 'OPEN EDITORS' panel lists several files, with 'dbconfig.js' selected. Below it, the 'API' panel shows a file tree with 'dbconfig.js' highlighted. The main editor area displays the content of 'dbconfig.js' with the following code:

```
1 var dbconfig = {
2   development: {
3     //connectionLimit : 10,
4     host      : 'localhost',
5     port      : '3306',
6     user      : 'root',
7     password   : '',
8     database   : 'moviedb'
9   },
10  production: {
11    //connectionLimit : 10,
12    host      : 'localhost',
13    port      : '3306',
14    user      : 'root',
15    password   : '',
16    database   : 'moviedb'
17  }
18 };
19 module.exports = dbconfig;
20
```

## สร้างโฟลเดอร์ repository และสร้างไฟล์ movie.js



## Code

```
var mysql = require('mysql');
const env = require('../env.js');
const { error } = require('fancy-log');
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 tis620 ) ASC `;

        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,
                });
            } else {
                pool.end();
                return resolve({
                    statusCode: 404,
                    returnCode: 11,
                    message: 'No movie found',
                });
            }

        });

    });

}
*/

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 tis620 ) ASC `;

        Query = `SELECT * FROM movies`;
        pool.query(Query, function (error, results, fields) {
            if (error) throw error;

            if (results.length > 0) {
                pool.end();
                return resolve(results);
            } else {
                pool.end();
                return resolve({
                    statusCode: 404,
                    returnCode: 11,
                    message: 'No movie found',
                });
            }

        });

    });

});

}

async function getMovieSearch(search_text) {

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

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

        Query = `SELECT * FROM movies WHERE title LIKE
'${search_text}%`;

        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,
            });
        } else {
            pool.end();
            return resolve({
                statusCode: 404,
                returnCode: 11,
                message: 'No movie found',
            });
        }
    });

});

});

}

async function
postMovie(p_title,p_genre,p_director,p_release_year) {

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

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

        //Query = `SELECT * FROM movies WHERE title LIKE
        '%${search_text}%'`;

        var post = {
            title: p_title,

```

```

        genre: p_genre,
        director: p_director,
        release_year: p_release_year
    };

    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'
                    }
                );
            }
        }
    });

```



```

    });
}

module.exports.MovieRepo = {
  getMovieList: getMovieList,
  getMovieSearch: getMovieSearch,
  postMovie: postMovie,
};

```

แก้ไขบรรทัดที่ 135-155 ด้วยโค้ดด้านล่าง

```

    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'
          });
      }
    }
  }
}

```

ทดสอบ

<http://localhost:4000/>

HTTP ENGSE203 / LAB4 / RUN ENV

Save

Send

GET

http://localhost:4000

Params

Authorization

Headers (7)

Body

Pre-request Script

Tests

Settings

Cookies

Query Params

	Key	Value	Description	...	Bulk Edit
	Key	Value	Description		

Body

Cookies

Headers (9)

Test Results

200 OK

7 ms

338 B

Save as example

...

Pretty

Raw

Preview

Visualize

HTML

1

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

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

GET

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

Send

Params

Authorization

Headers (7)

Body

Pre-request Script

Tests

Settings

Cookies

Query Params

	Key	Value	Description	...	Bulk Edit
	Key	Value	Description		

Body

Cookies

Headers (9)

Test Results

200 OK

62 ms

500 B

Save as example

...

Pretty

Raw

Preview

Visualize

JSON

1

[

2

{

3

"title": "Joker",

4

"genre": "psychological thriller",

5

"director": "Todd Phillips",

6

"release\_year": 2019

7

},

8

{

9

"title": "Joker1",

10

"genre": "psychological thriller",

11

"director": "Todd Phillips",

12

"release\_year": 2019

13

}

14

]

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

POST

http://localhost:4000/api/movie/insert

Send

Params

Authorization

Headers (9)

Body

Pre-request Script

Tests

Settings

Cookies

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON

Beautify

1

2

3

4

5

6

{

"title": "Joker3",

"genre": "psychological thriller",

"director": "Todd Phillips",

"release\_year": 2019

}

	title	genre	director	release_year	
	Joker	psychological thriller	Todd Phillips	2019	
	Joker1	psychological thriller	Todd Phillips	2019	
	Joker2	psychological thriller	Todd Phillips	2019	
	Joker3	psychological thriller	Todd Phillips	2019	
	NULL	NULL	NULL	NULL	

กรณี insert ข้อมูลซ้ำ <http://localhost:4000/api/movie/insert>

The screenshot shows a REST client interface with a POST request to `http://localhost:4000/api/movie/insert`. The request body is a JSON object: `{ "title": "Joker1", "genre": "psychological thriller", "director": "Todd Phillips", "release_year": 2019 }`. The response status is `200 OK` with a response time of `8 ms` and a body size of `383 B`. The response body is a JSON object: `{ "statusCode": 405, "returnCode": 9, "message": "ER_DUP_ENTRY: Duplicate entry 'Joker1' for key 'PRIMARY'" }`. The interface includes tabs for Params, Authorization, Headers (9), Body, Pre-request Script, Tests, and Settings. The Body tab is active, showing the request and response JSON. The response is displayed in the 'Pretty' view.

```
1 {
2   ...
3   ... "title": "Joker1",
4   ... "genre": "psychological thriller",
5   ... "director": "Todd Phillips",
6   ... "release_year": 2019
7   ...
8 }
```

Body Cookies Headers (8) Test Results 200 OK 8 ms 383 B Save as example

```
1 {
2   "statusCode": 405,
3   "returnCode": 9,
4   "message": "ER_DUP_ENTRY: Duplicate entry 'Joker1' for key 'PRIMARY'"
5 }
```

นาย นันทิพัฒน์ สมบัติวงศ์ 66543210018-8