

ENGSE203 LAB4

เปลี่ยน Port ของ Backend เป็น 4000

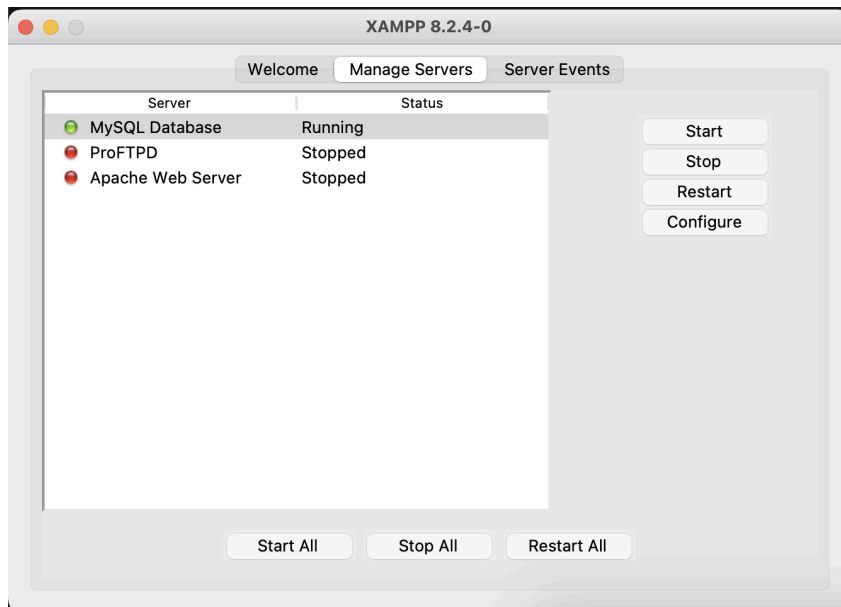
```
JS index.js JS server.js JS movie.js JS dbconfig.js package.json X
package.json > {} scripts > dev
1 {
2   "name": "react-nodejs-example",
3   "version": "1.0.0",
4   "description": "example project react with nodejs",
5   "main": "server.js",
6   "scripts": {
7     "start": "node server.bundle.js",
8     "build": "webpack",
9     "dev": "nodemon ./index.js localhost 4000"
```

```
JS index.js JS server.js X JS movie.js
JS server.js > users
1 const express = require('express');
2 const path = require('path');
3 const app = express(),
4   bodyParser = require("body-parser");
5   port = 4000;
```

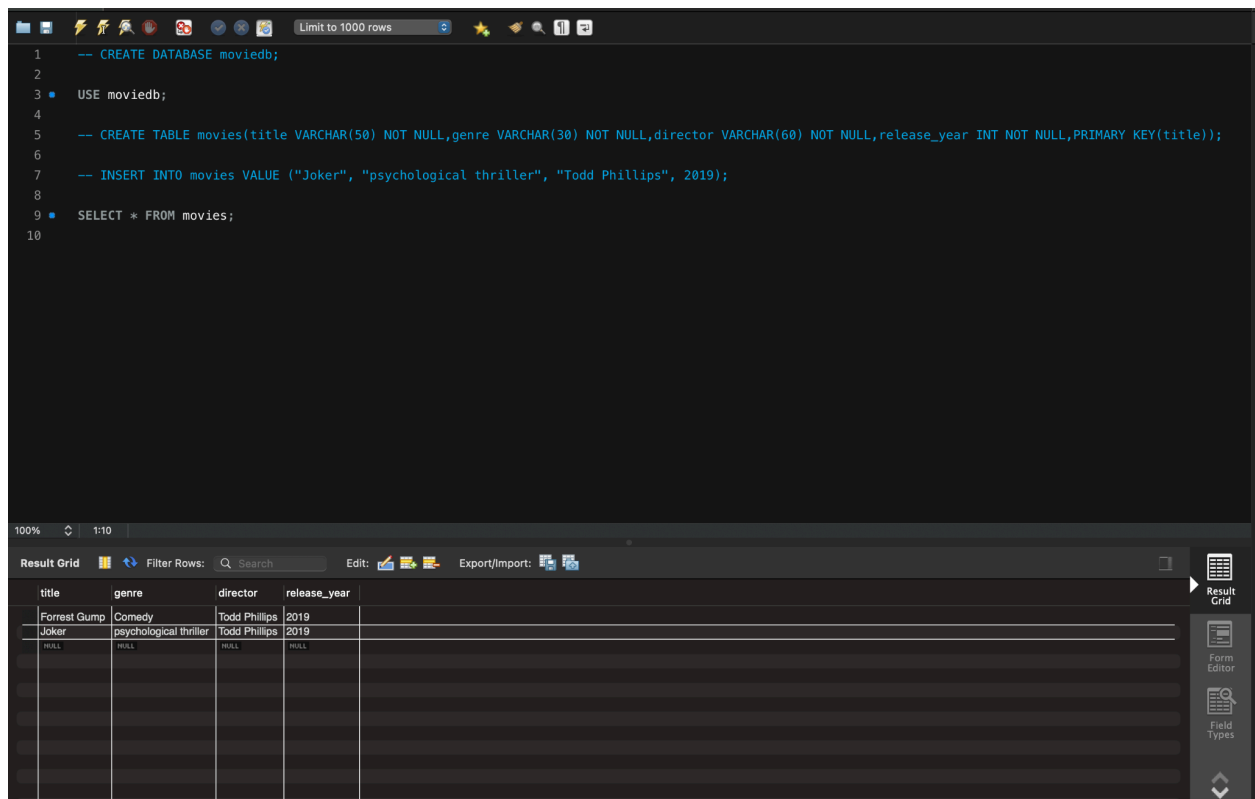
ทดสอบ รัน Port 4000

```
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":"Surapat","lastName":"Supap","email":"abc@gmail.com"}]
```

เปิด MySQL ใน Xampp



สร้าง database , เพิ่มตารางและเพิ่มข้อมูลลงในตาราง



สร้าง index.js

```
index.js > [e] init
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
16  //----- hapi -----
17
18  console.log('Running Environment: ' + env);
19
20
21  const init = async () => {
22
23      const server = hapi.Server({
24          port: api_port,
25          host: '0.0.0.0',
26          routes: {
27              cors: true
28          }
29      });
30
31      //-----
32
33      await server.register(require('@hapi/inert'));
34
35      server.route({
36          method: "GET",
37          path: "/",
38          handler: () => {
39              return '<h3> Welcome to API Back-end Ver. 1.0.0</h3>';
40          }
41      });
42
43
44      //API: http://localhost:3001/api/movie/all
45      server.route({
46          method: 'GET',
47          path: '/api/movie/all',
48          config: {
49              cors: {
50                  origin: ['*'],
51                  additionalHeaders: ['cache-control', 'x-requested-width']
```

```

102 });
103
104
105 server.route({
106   method: 'POST',
107   path: '/api/movie/insert',
108   config: {
109     payload: {
110       multipart: true,
111     },
112     cors: {
113       origin: ['*'],
114       additionalHeaders: ['cache-control', 'x-requested-width']
115     }
116   },
117   handler: async function (request, reply) {
118
119     const {
120       title,
121       genre,
122       director,
123       release_year
124     } = request.payload;
125
126     //const title = request.payload.title;
127     //const genre = request.payload.genre;
128
129     try {
130
131       const responsedata = await Movies.MovieRepo.postMovie(title, genre, director, release_year);
132       if (responsedata.error) {
133         return responsedata.errMessage;
134       } else {
135         return responsedata;
136       }
137     } catch (err) {
138       server.log(["error", "home"], err);
139       return err;
140     }
141   }
142 });
143
144
145
146
147
148 await server.start();
149 console.log('API Server running on %s', server.info.uri);
150
151 //-----
152 };

```

```

52     }
53   },
54   handler: async function (request, reply) {
55     //var param = request.query;
56     //const category_code = param.category_code;
57
58     try {
59
60       const respondedata = await Movies.MovieRepo.getMovieList();
61       if (respondedata.error) {
62         return respondedata.errMessage;
63       } else {
64         return respondedata;
65       }
66     } catch (err) {
67       server.log(["error", "home"], err);
68       return err;
69     }
70   }
71 }
72 });
73
74 server.route({
75   method: 'GET',
76   path: '/api/movie/search',
77   config: {
78     cors: {
79       origin: ['*'],
80       additionalHeaders: ['cache-control', 'x-requested-width']
81     }
82   },
83   handler: async function (request, reply) {
84     var param = request.query;
85     const search_text = param.search_text;
86     //const title = param.title;
87
88     try {
89
90       const respondedata = await Movies.MovieRepo.getMovieSearch(search_text);
91       if (respondedata.error) {
92         return respondedata.errMessage;
93       } else {
94         return respondedata;
95       }
96     } catch (err) {
97       server.log(["error", "home"], err);
98       return err;
99     }
100   }
101 }
102 });

```

```

153
154
155 process.on('unhandledRejection', (err) => {
156
157   console.log(err);
158   process.exit(1);
159 });
160
161 init();
162

```

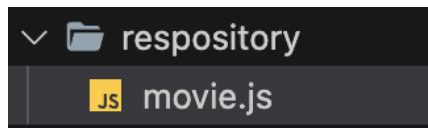
สร้าง env.js

```
index.js  env.js  movie.js  dbconfig.js  package.json
env.js > ...
1  var env = process.env.NODE_ENV || 'development';
2  //var env = process.env.NODE_ENV || 'production';
3  module.exports = env;
4
```

สร้าง dbconfig.js

```
index.js  env.js  movie.js  dbconfig.js  package.json
dbconfig.js > ...
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



```
respository > JS movie.js > postMovie > <function>
1  var mysql = require('mysql');
2  const env = require('../env.js');
3  const config = require('../dbconfig.js')[env];
4
5  /*
6  async function getMovieList() {
7
8      var Query;
9      var pool = mysql.createPool(config);
10
11      return new Promise((resolve, reject) => {
12
13          //Query = `SELECT * FROM movies WHERE warehouse_status = 1 ORDER BY CONVERT( warehouse_name USING tis620
14          Query = `SELECT * FROM movies`;
15
16          pool.query(Query, function (error, results, fields) {
17              if (error) throw error;
18
19              if (results.length > 0) {
20                  pool.end();
21                  return resolve({
22                      statusCode: 200,
23                      returnCode: 1,
24                      data: results,
25                  });
26              } else {
27                  pool.end();
28                  return resolve({
29                      statusCode: 404,
30                      returnCode: 11,
31                      message: 'No movie found',
32                  });
33              }
34          });
35      });
36  });
37
38  }
39
40  */
41
42  async function getMovieList() {
43
44      var Query;
45      var pool = mysql.createPool(config);
46
47      return new Promise((resolve, reject) => {
48
49          //Query = `SELECT * FROM movies WHERE warehouse_status = 1 ORDER BY CONVERT( warehouse_name USING tis620
50          Query = `SELECT * FROM movies`;
```

```

52
53     pool.query(Query, function (error, results, fields) {
54         if (error) throw error;
55
56         if (results.length > 0) {
57             pool.end();
58             return resolve(results);
59         } else {
60             pool.end();
61             return resolve({
62                 statusCode: 404,
63                 returnCode: 11,
64                 message: 'No movie found',
65             });
66         }
67     });
68
69 });
70
71
72
73 }
74
75
76 async function getMovieSearch(search_text) {
77
78     var Query;
79     var pool = mysql.createPool(config);
80
81     return new Promise((resolve, reject) => {
82
83         Query = `SELECT * FROM movies WHERE title LIKE '%${search_text}%'`;
84
85         pool.query(Query, function (error, results, fields) {
86             if (error) throw error;
87
88             if (results.length > 0) {
89                 pool.end();
90                 return resolve({
91                     statusCode: 200,
92                     returnCode: 1,
93                     data: results,
94                 });
95             } else {
96                 pool.end();
97                 return resolve({
98                     statusCode: 404,
99                     returnCode: 11,
100                     message: 'No movie found',
101                 });
102             }
103         });
104     });
105 }

```



```

104     });
105
106     });
107
108
109 }
110
111 async function postMovie(p_title,p_genre,p_director,p_release_year) {
112
113     var Query;
114     var pool = mysql.createPool(config);
115
116     return new Promise((resolve, reject) => {
117
118         //Query = `SELECT * FROM movies WHERE title LIKE '%${search_text}%'`;
119
120         var post = {
121             title: p_title,
122             genre: p_genre,
123             director: p_director,
124             release_year: p_release_year
125         };
126
127         console.log('post is: ', post);
128
129
130         Query = 'INSERT INTO movies SET ?';
131         pool.query(Query, post, function (error, results, fields) {
132             //pool.query(Query, function (error, results, fields) {
133
134                 console.log('error_code_msg: ',error.code+'-'+error.sqlMessage);
135
136                 if (error) {
137                     pool.end();
138                     return resolve({
139                         statusCode:405,
140                         return:9,
141                         message: error.code+'-'+error.sqlMessage
142                     });
143                 }
144                 else{
145                     console.log('resukts: ',results);
146                     if (results.affectedRows > 0) {
147                         pool.end();
148                         return resolve({
149                             statusCode: 200,
150                             returnCode: 1,
151                             message: 'Movie list was inserted',
152                         });
153                     }
154                 }
155             }
156         });
157
158     });
159 }
160
161
162 }
163
164 module.exports.MovieRepo = {
165     getMovieList: getMovieList,
166     getMovieSearch: getMovieSearch,
167     postMovie: postMovie,
168
169 };

```

```

155     }
156     });
157
158     });
159
160
161 }
162
163
164 module.exports.MovieRepo = {
165     getMovieList: getMovieList,
166     getMovieSearch: getMovieSearch,
167     postMovie: postMovie,
168
169 };

```

```
130 Query = 'INSERT INTO movies SET ?';
131 pool.query(Query, post, function (error, results, fields) {
132 //pool.query(Query, function (error, results, fields) {
133
134     console.log('error_code_msg: ',error.code+':'+error.sqlMessage);
135
136     if (error) {
137         pool.end();
138         return resolve({
139             statusCode:405,
140             return:9,
141             message: error.code+':'+error.sqlMessage
142         });
143     }
144     else{
145         console.log('resukts: ',results);
146         if (results.affectedRows > 0) {
147             pool.end();
148             return resolve({
149                 statusCode: 200,
150                 returnCode: 1,
151                 message: 'Movie list was inserted',
152             });
153         }
154     }
155 }
156 });|
```

แก้ไข การทำงาน ในบรรทัด 132 ถึง 142

Before

```
if (error) throw error;

    if (results.length > 0) {
        pool.end();

        return resolve({
            statusCode: 200,
            returnCode: 1,
            message: 'Movie list was inserted',
        });
    }

});

});

}
```

After

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

        console.log('error_code_msg: ',error.code+':'+error.sqlMessage);

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

        }

    });
});
```

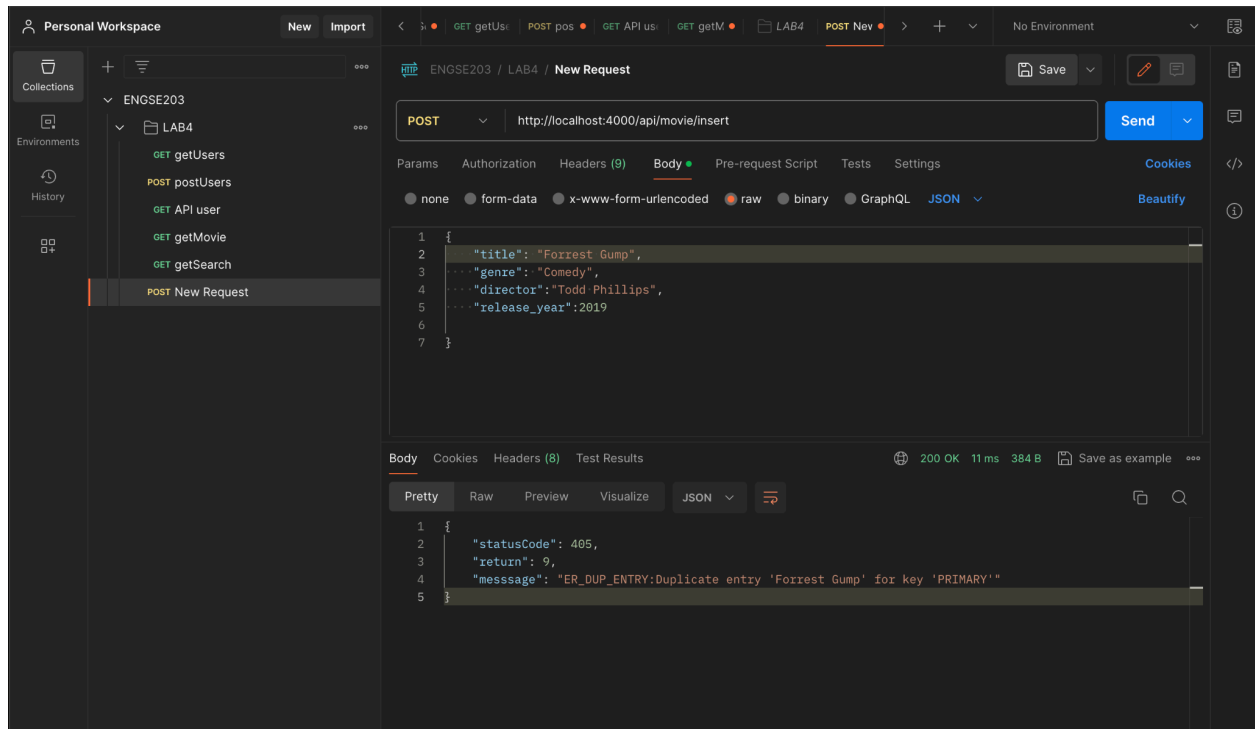
ทดสอบการทำงานโดยใช้ Postman

The screenshot shows the Postman interface with a collection named 'ENGSE203' containing a folder 'LAB4'. Inside 'LAB4', the request 'GET getMovie' is selected. The request is a GET method to the URL 'http://localhost:4000'. The response is a 200 OK status with a 3 ms response time and 338 B of data. The response body is displayed in HTML format, showing a single line: `<h3> Welcome to API Back-end Ver. 1.0.0</h3>`.

Key	Value	Description
Key	Value	Description

The screenshot shows the Postman interface with the same collection and folder structure. The request 'GET getSearch' is selected. The request is a GET method to the URL 'http://localhost:4000/api/movie/all'. The response is a 200 OK status with an 8 ms response time and 490 B of data. The response body is displayed in JSON format, showing an array of two movie objects:

```
1 {
2   {
3     "title": "Forrest Gump",
4     "genre": "Comedy",
5     "director": "Todd Phillips",
6     "release_year": 2019
7   },
8   {
9     "title": "Joker",
10    "genre": "psychological thriller",
11    "director": "Todd Phillips",
12    "release_year": 2019
13  }
```



```
API Server running on http://0.0.0.0:4000
post is: {
  title: 'Forrest Gump',
  genre: 'Comedy',
  director: 'Todd Phillips',
  release_year: 2019
}
```

กรณีข้อมูลซ้ำกัน

```
error_code_msg: ER_DUP_ENTRY: Duplicate entry 'Forrest Gump' for key 'PRIMARY'
post is: {
  title: 'Forrest Gump',
  genre: 'Comedy',
  director: 'Todd Phillips',
  release_year: 2019
}
error_code_msg: ER_DUP_ENTRY: Duplicate entry 'Forrest Gump' for key 'PRIMARY'
```