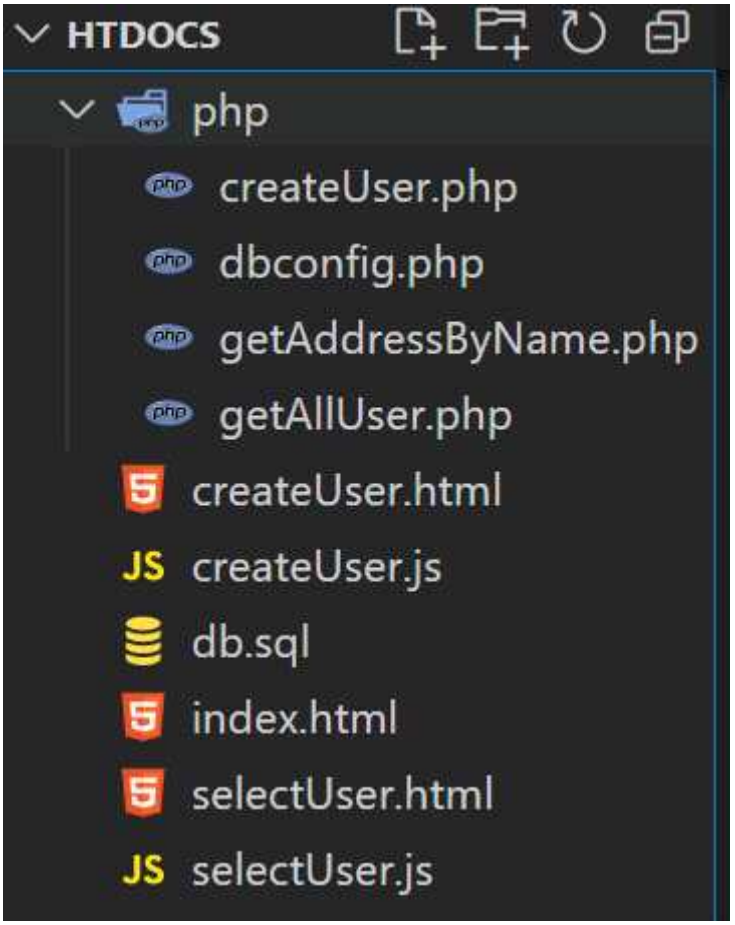
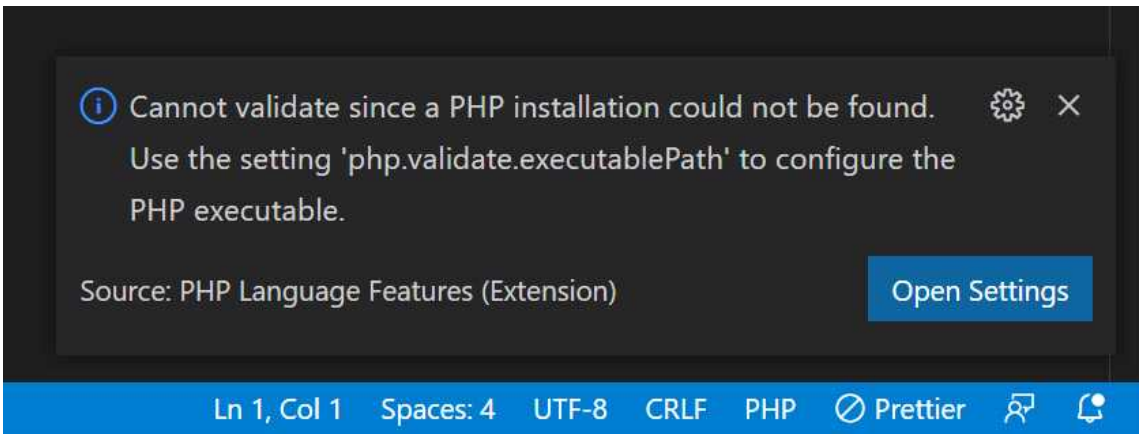


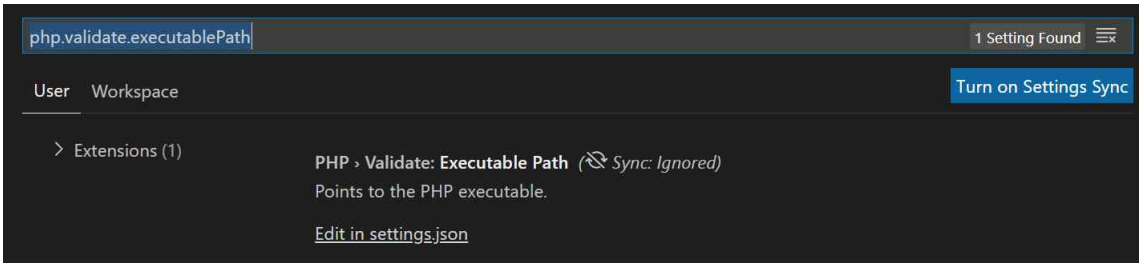
먼저, 프로젝트 디렉터리에 php 디렉터리를 생성하고, 다음 네 개의 php 파일을 만들자.

* 저번시간에 내가 헷갈렸는데, php 파일 이름이 getAddressByID.php 가 아니라, getAddressByName.php 가 맞다. 바꿔서 진행하도록 한다.

저번 시간 마지막에 해결했는지 모르겠는데, 해결 못 한 친구들을 위해 다시 한 번 집고 넘어가겠다.



이런 경고창이 뜰 것이다. 이것은 여러분들의 컴퓨터에서 php 가 설치된 경로를 찾을 수 없기 때문에 발생하는 경고다.
파란색 open settings 버튼을 클릭하자.



여기서, 밑줄 쳐진 Edit in setting.json을 클릭하자.

JSON 이라는 양식에 맞춰줘야한다.
데이터를 추가하고자 하면, 객체의 맨 끝에 콤마(,)를 쓴 후, 위와 같이 작성한다.
아마 여러분들의 bitnami wamp stact 의 버전은 다를 수 있으니, 신경써서 작성하길 바란다.
성공하면 더 이상 에러는 뜨지 않을 것이다. 이제 php 익스텐션을 적용할 준비가 되었다.

dbconfig.php

여러분들이 가장 먼저 작성할 파일은, 여러분의 php를 db 에 연결하는 명령을 담은 php 파일을 작성하는 것이다. 그 파일 이름을 dbconfig.php 라고 하겠다.

왜 이런 파일을 만드냐면, 이 파일이 존재하지 않는다면 php 파일을 하나 만들 때마다 매번 db 연결 코드를 앞에 써야하기 때문이다.
그래서 dbconfig.php 라는 파일을 만들어놓고, 다른 파일에서 이 파일을 맨 첫줄에 불러내 사용할 것이다.

```php
<?php
error_reporting(E_ALL);
ini_set("display_errors", 1);

header("Content-Type:application/json");

$host = 'localhost';
$user = 'root';
$pw = '111111';
$dbName = 'test';
$db = new mysqli($host, $user, $pw, $dbName);

mysqli_set_charset($db, "utf8");
```

line1: php 가 시작된다는 걸 알릴 땐, 다음과 같이 <?php 로 시작한다.
만약, 파일 전체에 php 가 쓰였다면 php 를 닫아줄 필요는 없다.
닫을 때는 ?> 로 닫는다. 그러면 php 문법이 끝났다는 뜻이다.
왜 이런 게 필요하냐면, php 파일에선 html 문법이 사용가능하기 때문이다.
그러나 우린 그 어떤 html 문법도 넣지 않을 것이고,
오로지 서버 통신을 위해서만 php 를 사용할 것이기에
<?php 로 열어만 주지, 닫지는 않는다.
line2, 3: 디버그를 위해 필요한 구문이다.
보안상 매우 위험하다.
개발이 끝나고 실제 배포할때는 이 줄을 주석처리하던지 삭제하던지 하자.
line5: JSON Format 을 사용하기 위해 꼭 필요한 구문이다.
line7-10: 접속정보를 변수에 담는다.
php에서 변수는 $변수이름 이다.

$host	접속하고자하는 mysql db 경로. 우린 localhost 이다.
$user	mysql db 유저이름이다. 우린 root 만 만들었다.
$pw	root 의 비밀번호다. 물론 $pw 가 아니라 $password 로 지어도 잘 동작한다. 변수는 이름일 뿐이다.
$dbname	mysql 의 접속하고자 하는 db 이름. 우린 test 다.

line11: php의 mysqli (mysql 아니고 mysqli) 라는 익스텐션을 실행하는 구문이다.

Bitnami WAMP Stack 엔 기본적으로 깔려있는데, php 와 함께 mysql 을 깔았으니 당연히 mysqli 를 쓸 것이라고 예상하기 때문이다.
앞에 붙은 new 는 객체를 하나 생성한다고 알아두자.
총 네 개의 파라미터가 들어가며, 순서대로 mysql 경로, mysql유저이름, mysql 비밀번호이름, mysql db 이름이다.
리턴값은 $db 라는 변수다. 이 변수로 sql 구문을 실행시키거나, db를 다 쓰고 나서 닫는다.

이것이 dbconfig.php 파일의 전부다. 물론, 실제 db 접속정보를 아무런 보안처리도 없이 이런 파일에 담아두면, 실전에선 큰일난다. php 뿐만아니라 다른 언어들도 마찬가지지만, 서버개발자가 신경써야할 보안기법은 아주 많다. 직접 찾아서 공부해보도록 하자.

getAllUser.php

가장 쉬운 건, 모든 사용자 정보를 가져오는 것이다.
이 php 파일의 목적은, 이 파일이 실행되는 순간 모든 사용자 정보를 클라이언트로 보내는 것이다.
우선 db 접속정보가 필요하다. 맨 앞줄에 다음과 같이 추가한다.

```
1   <?php
2
3   require_once("dbconfig.php"); // 항상 맨 앞줄에 추가
```

require_once() 함수를 사용해 다른 php 파일을 현재의 php 파일로 임포트할수있다.
우린 dbconfig.php 파일 전체를 가져왔으니, 이 파일의 다른 부분 실행 전에 dbconfig.php 의 모든 코드가 실행된 것이며, 당연히 $db 변수도 사용가능하다.

```
5   $sql = "SELECT * FROM person";
```

실행시키고자 하는 sql 구문을 변수에 담는다. sql구문을 쌍따옴표 " " 로 감싸되, 늘상 하던것처럼 세미콜론을 넣지 않았다는점에 유의하자.
이렇게 한다고해서 실행되는건 아니고, $sql 변수 안에 일단 넣어두기만 하는 것이다.

```
7   $data = array();
```

우리가 최종적으로 프론트엔드로 보낼 변수다. 이렇게 쓰면 빈 배열로 선언한 것이다. sql 실행결과에 따라, 이 안에 데이터를 push 할 것이다.

```
9    $res = $db->query($sql);
```

이 구문이 바로 우리가 작성했던 sql 구문, “SELECT * FROM person” 을 실행시키는 구문이다.
$db 객체의 원형은 mysqli 라서, query 라는 함수가 내장되어 있는데, 이 함수의 파라미터로 우리가 작성했던 sql, 즉 $sql 을 넣어서 실행시킨 것이다.
실행결과는 $res 라는 변수에 담았다.

그럼 $res 를 보내버리면 끝일까? 그건 아니다. $res 객체 내에 존재하는 fetch_array(MYSQLI_ASSOC) 함수를 통해 이걸 우리가 쓸만하게 바꾸는 과정이 필요한데, 여기서 php 의 까다로움을 알 수 있다. 표현하자면, 요즘 언어답지 않는 구식 방식을 따른다.

```
11   for ($i = 0; $i < $res->num_rows; $i++) {
12       $row = $res->fetch_array(MYSQLI_ASSOC);
13       array_push($data, $row);
14   }
```

왜 반복문을 쓸까? $res->fetch_array(MYSQLI_ASSOC) 를 실행할 시, 실행결과를 “한 행” 씩 뱉어낸다.

$res->fetch_array(MYSQLI_ASSOC) 처음 시행시 뱉어내는 결과

id	name	address
1	조교행님	경남 진주시
2	전인혁	목포시
3	황승현	마산시

$res->fetch_array(MYSQLI_ASSOC) 두번째 실행 시 뱉어내는 결과

id	name	address
1	조교행님	경남 진주시
2	전인혁	목포시
3	황승현	마산시

$res->fetch_array(MYSQLI_ASSOC) 세번째 실행 시 뱉어내는 결과

id	name	address
1	조교행님	경남 진주시
2	전인혁	목포시
3	황승현	마산시

만약, 실행결과가 다음과 같이 3개의 row 가 아니라, 단 하나밖에 없을 경우엔 굳이 반복문을 사용하지 않아도 될 것이다.

길어졌으니 다시 한 번 보면,

```
11    for ($i = 0; $i < $res->num_rows; $i++) {
12        $row = $res->fetch_array(MYSQLI_ASSOC);
13        array_push($data, $row);
14    }
```

반복문은 0부터 시작하고, $res 의 길이만큼만 반복하는데 길이는 $res 객체의 num_rows 라는 필드에 접근해서 얻는다.
* 필드, 객체 등을 정확하게 알려면 객체지향을 배워야한다. 자바스크립트에서 배운 객체와 php 에서 계속 말하는 객체는 엄밀히 말하면 완전히 다른 것이다. 우리 수업에선 몰라도 무방하기에 생략한다.

그래서 $res->fetch_array(MYSQLI_ASSOC) 한번 실행한 결과를 우리가 미리 만들어놓았던 빈 배열, $data 에 array_push() 함수를 사용해 담는 것이다.

```
16    if ($data != null) {
17        echo json_encode($data, JSON_UNESCAPED_UNICODE | JSON_NUMERIC_CHECK);
18    } else {
19        echo false;
20    }
```

여기서, 변수 $data 에 무슨 값이라도 들어있으면 ($data != null)

echo, 즉 서버에서 “클라이언트” 로 보내는데, JSON Format 의 “객체”로 보낸다.

json_encode() 함수를 사용하며, 첫번째 파라미터에 완성된 $data, 두번째 파라미터에 JSON 규칙같은걸 써주는데,

여러분이 코딩하면서 쓸 일이 있는건 딱 저 두개를 합한 구문,

JSON_UNESCAPED_UNICODE | JSON_NUMERIC_CHECK

밖에 없다고 생각하자.

만약, 아무것도 안 들어있으면 echo 를 사용해 클라이언트로 false 라고 보내는데, 클라이언트에서 찍힐 땐 0 으로 찍힐 것이다.

즉, true 를 보내면 1로, false 를 보내면 0으로 받는다.

true, false 대신 0과 1로 써도 된다.

* JSON 은 JavaScript 의 “객체” 라고 생각하자. 문법이 아주 살짝 다르다.

즉, 클라이언트(프론트엔드)와 서버(백엔드) 가 통신을 할 땐,

JavaScript 객체를 주고받는 것이다.

```
22   mysqli_close($db);
```

마지막으로, db를 썼으면 mysqli_close($db) 를 사용해 닫아준다.

getAllUser.php 파일의 전체 코드는 다음과 같다.

getAllUser.php

```php
<?php

require_once("dbconfig.php"); // 항상 맨 앞줄에 추가

$sql = "SELECT * FROM person";

// 최종결과
$data = array();

// 실행결과는 $res에 저장
$res = $db->query($sql);

// fetch_array는 한번 실행될때마다 한 행씩 뱉어내므로
// 원하는 걸 모두 얻으려면 반복문 사용해야함.
for ($i = 0; $i < $res->num_rows; $i++) {
    $row = $res->fetch_array(MYSQLI_ASSOC);
    array_push($data, $row);
}

if ($data != null) {  // 만약 배열이 존재한다면 통신성공
    echo json_encode($data, JSON_UNESCAPED_UNICODE | JSON_NUMERIC_CHECK);
} else {    // 통신실패: false 반환
    echo false;
}

mysqli_close($db);
```

잘 찍히는지 확인하려면, selectUser.js 로 가서 php 파일 경로를 바꿔주면 될 것이다.

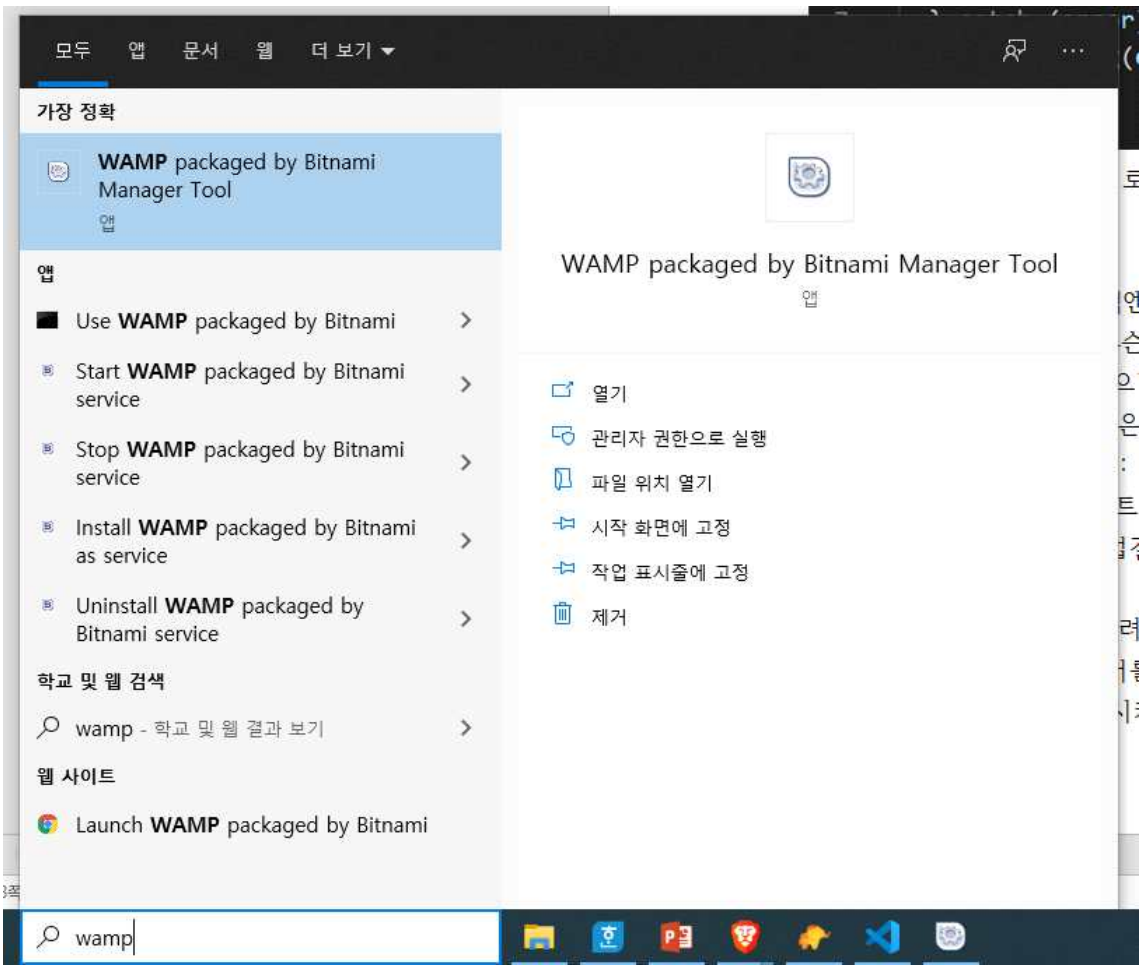
```javascript
const getAllUser = async () => {
  try {
    const response = await axios.get("./php/getAllUser.php");
    if (response.data) {
      console.log(response.data);
    }
  } catch (error) {
    console.log(error);
  }
};
```

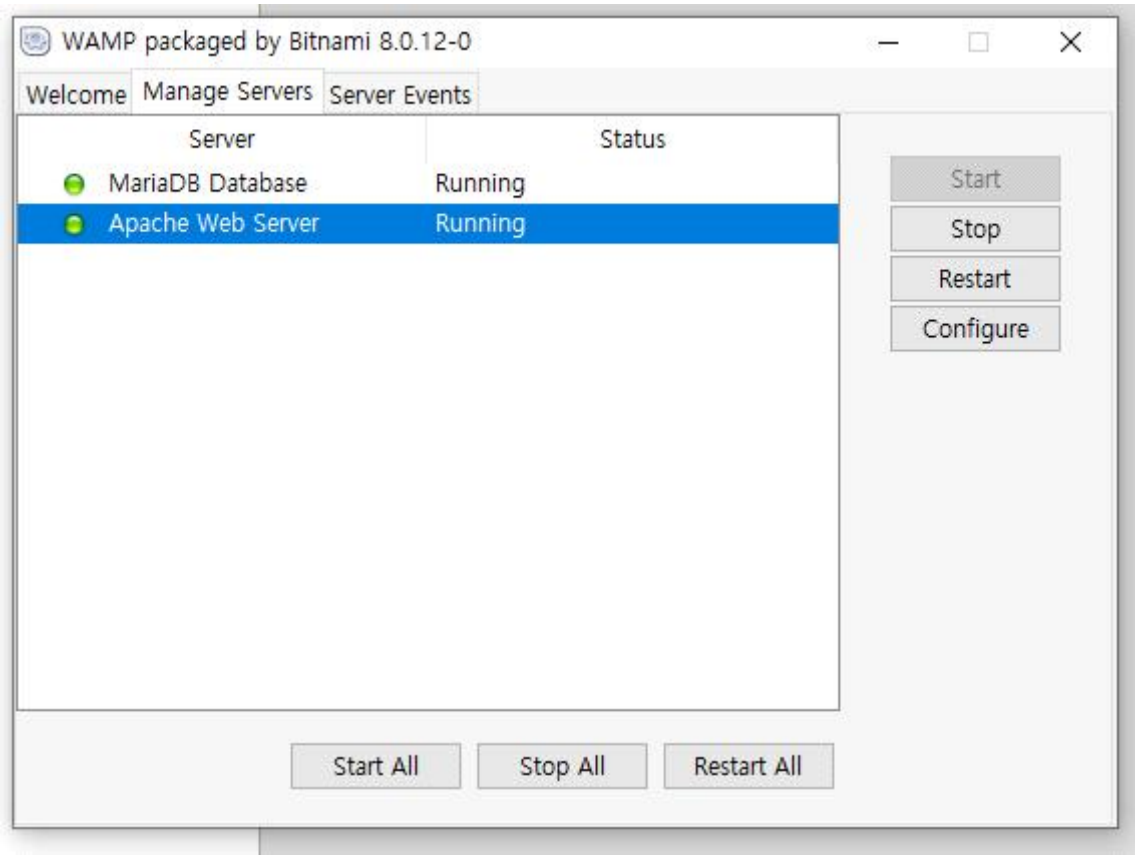
쌍따옴표 “ ” 안에 http 경로를 때버리고, getAllUser.php 파일이 위치한 경로를 입력해주자.

Q. 이러면 프론트엔드와 백엔드가 분리 안되는거 아닌가요? 같은 프로젝트 디렉터리에 있는데 이게 무슨 서버코드죠?
A. 완전한 분리를 하고 싶으면 포트까지 다르게 작업해서 서비스해야한다. 이러면 무지하게 어려워진다! 지금은 위치는 같되, 역할이 다르다고 생각하자.

JavaScript(프론트): 서버와 데이터통신을 해서, 브라우저를 조작한다.
PHP(백): 클라이언트로부터 요청받으면, MySQL 에 접근해 적당한 작업을 한 후, 작업결과를 클라이언트로 넘겨준다.

그리고 php 파일을 적용하려면 우리의 Bitnami 서버가 느리기 때문에 굉장히 귀찮은 일을 해줘야한다. 바로 서버를 껐다 키는 것이다.
WAMP Manager 를 실행시켜주자.

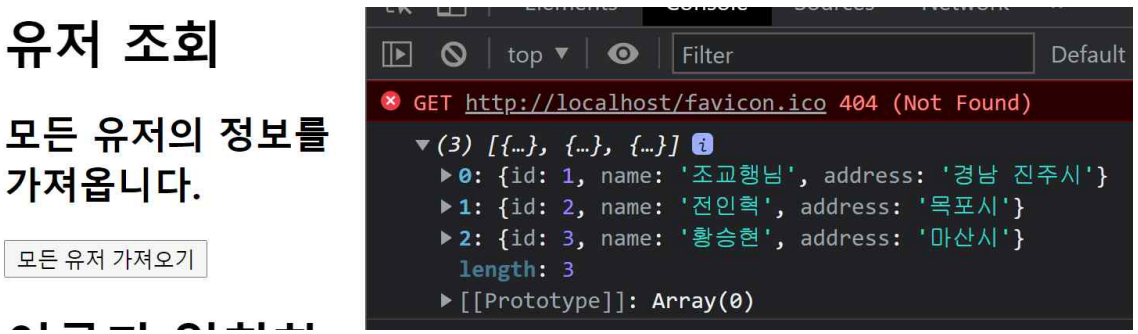




MariaDB Database, Apache Web Server 둘 다 리스타트하는게 속 편하다. 즉, VSCode 에서 작업한 모든 코드의 내용을 저장한 후, Restart 해주는 것이다. PHP 파일을 새로 만들거나, 수정하거나, 삭제할 땐 반드시 이 작업을 거쳐주자.

브라우저에선 언제나 그랫듯이, 캐시 비우고 강력 새로고침 해주고 버튼을 클릭해보자.

결과는 바로 나오지 않는다. 내 컴퓨터 기준 3초 정도 기다려야했다.



잘 받아온 것을 볼 수 있다.
여기서 비동기(Asynchronous) 의 존재이유를 알 수 있다. Windows 를 쓰는 경우 Bitnami WAMP Stack 은, 진지하게 서버를 운영하는 용도가 아니라 테스트용이기 때문에 심각하게 느린데다가, Restart 까지 해줘야했다.
지금 받아오는데만 3초정도 걸렸는데, 이런 식으로 서버에서 언제 가져올지 모르는 데이터를 한도끝도없이 기다리기만 하면 사용자만 지칠 것이다. 로딩이 10초 이상 길어지면 "서버에 문제가 있습니다" 라는 경고창이라도 띄워줘야 할 것이고, 다른 실행해야 할 작업들을 미리 실행하는 센스도 있어야한다. 이것이 비동기의 힘이다.

다른 php 파일도 마저 완성해보자.

getAddressByName.php

이 php 파일은
클라이언트로부터 이름을 "받아서"
해당 이름의 "집 주소" 를 DB로부터 가져온 다음
집 주소를 클라이언트로 "보내주는" 것이 목적이다.

먼저, selectUser.js 의, 해당 php 파일을 call 하는 getAddressByName() 함수부터 작성하자.

* 저번시간에 getAddressByID 로 JavaScript 함수테스트를 했기 때문에, selectUser.html 의 이벤트핸들러에서도 이름을 바꿔줘야 한다.

```
12  const getAddressByName = async () => {
13    const nameInput = document.querySelector(".nameInput").value;
14    if (nameInput) {
15      try {
16        const response = await axios.post("./php/getAddressByName.php", {
17          nameInput: nameInput,
18        });
19        if (response.data) {
20          console.log(response.data);
21        } else {
22          console.log("그런 이름 없습니다.");
23        }
24      } catch (error) {
25        console.log(error);
26      }
27    }
28  };
```

line12: 역시 async 로 시작한다. 비동기가 포함되어있기 때문이다.

line13: 이름은 어디서 가져오나? 입력창이다.
html 의 nameInput 클래스를 querySelector() 로 가져온 다음,
사용자 입력을 가져오려면 value 에 접근한다.
그걸 nameInput 변수에 담았다.
line14: 만약, 사용자가 뭐라도 입력했으면,
line15: 역시 try, catch 가 쓰인다.
line16: response 로 받고, 비동기니깐 await 를 쓴다.
이 경우, get 이 아니라 post 다.
REST API 에서 get 과 post 로 왠만한 건 다 할 수 있다.
get 데이터를 가져올 때 (CRUD 의 R)
post 데이터를 변경할 때 (CRUD 의 CRD)
그러나 세상 편하게 코딩하려면, 내가 서버에 보낼 데이터가 있을 땐
POST 를 쓴다고 생각하자. 여기선, nameInput 변수를 php 로
보내버릴 것이기에 POST 를 쓴 것이다.
line17: nameInput 을 딸려보낼 것이다. 보낼때도 객체로 보낸다.
즉, 서버와 통신을 한다는건 객체를 주고받는다는 뜻이다.
line19: 만약 받아온 게 뭐라도 들어있으면 출력하라.
line21: false(0) 을 받으면 보내준 게 없는 것이다. 즉, db 조회 시 없는
이름을 조회해서 실패했을 시 false 를 보내도록 php 로 코딩할
것이다. 이 경우, "그런 이름 없습니다" 라고 찍어내자.
line24: 통신 실패 시 error 를 출력한다.

이제 getAddressByName.php 파일을 작성해보자. 아까 전, getAllUser.php 시 설명했던 부분은 일부 생략한다.

```php
<?php
require_once("dbconfig.php");
$_POST = JSON_DECODE(file_get_contents("php://input"), true);
$nameInput = $_POST["nameInput"];
$sql = "SELECT address FROM person WHERE name = '$nameInput'";
$res = $db->query($sql);
$row = $res->fetch_array(MYSQLI_ASSOC);
if ($row) {
    echo json_encode($row,JSON_UNESCAPED_UNICODE|JSON_NUMERIC_CHECK);
} else {
    echo false;
}
mysqli_close($db);
```

line3: POST 통신 시 관습적으로 쓰는 코드다.
$_POST 변수에 클라이언트에서 넘겨준 JSON 을 DECODE 한다.

JSON_DECODE(file_get_contents("php://input"), true) 라고 써주자.

line4: 우린 딱 하나, nameInput 만 서버로 넘겨줬지만 상황에 따라 여러개를 넘겨줄 수도 있다. 넘겨준것은 각각의 php 변수로 담아 php 에서 쓸 수 있게 만드는데, 접근하려면 $_POST["nameInput"] 으로 접근한다.

line5: sql 구문. 여기서 name 에 php 변수 $nameInput 이 들어갔는데, 작은따옴표 ‘ ’ 를 사용했다. 큰따옴표를 쓰면 안되는데, sql 구문 전체를 큰따옴표로 감쌌기 때문이다.

line6: sql 실행

line7: for loop를 쓸 필요가 없는데, 결과값은 “한 줄” 이기 때문이다.

line8: 만약, 결과값이 있다면 if($row)

결과값이 있다는 뜻은 무엇인가? sql을 실행시켜봤더니, 해당 아이디가 존재하고, 그 아이디의 address 를 받아왔다는 뜻이다.

그걸 json_encode() 로 json 으로 만든다음 클라이언트로 보내버린다.

만약, 결과값이 없다면

결과값이 없다는 뜻은? 입력한 이름과 일치하는 이름이 없다는 뜻이므로, false 를 보낸다.

클라이언트에선 0 으로 받을 것이고, “그런 이름 없습니다” 가 출력될 것이다.

line13: db는 썼으면 닫아준다.

결과:

유저 조회

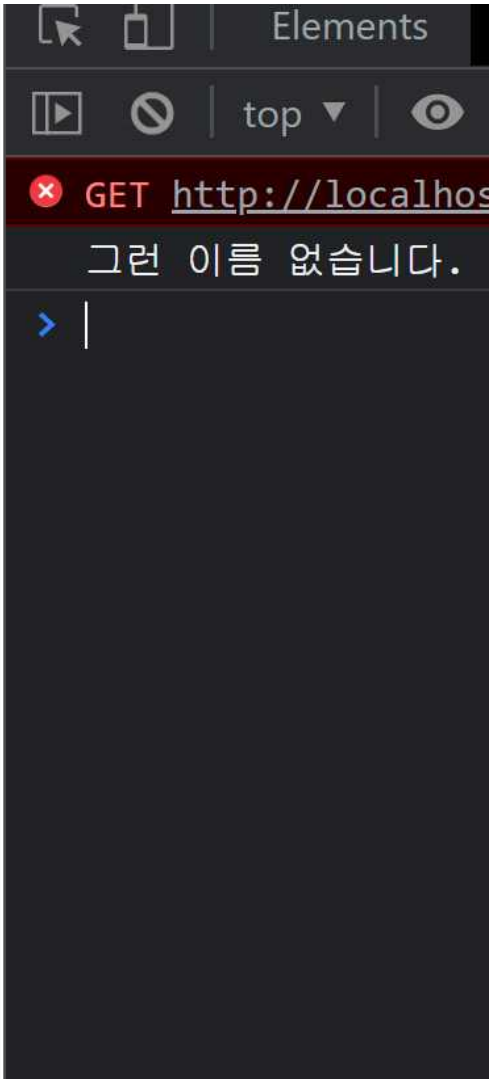
모든 유저의 정보를 가져옵니다.

모든 유저 가져오기

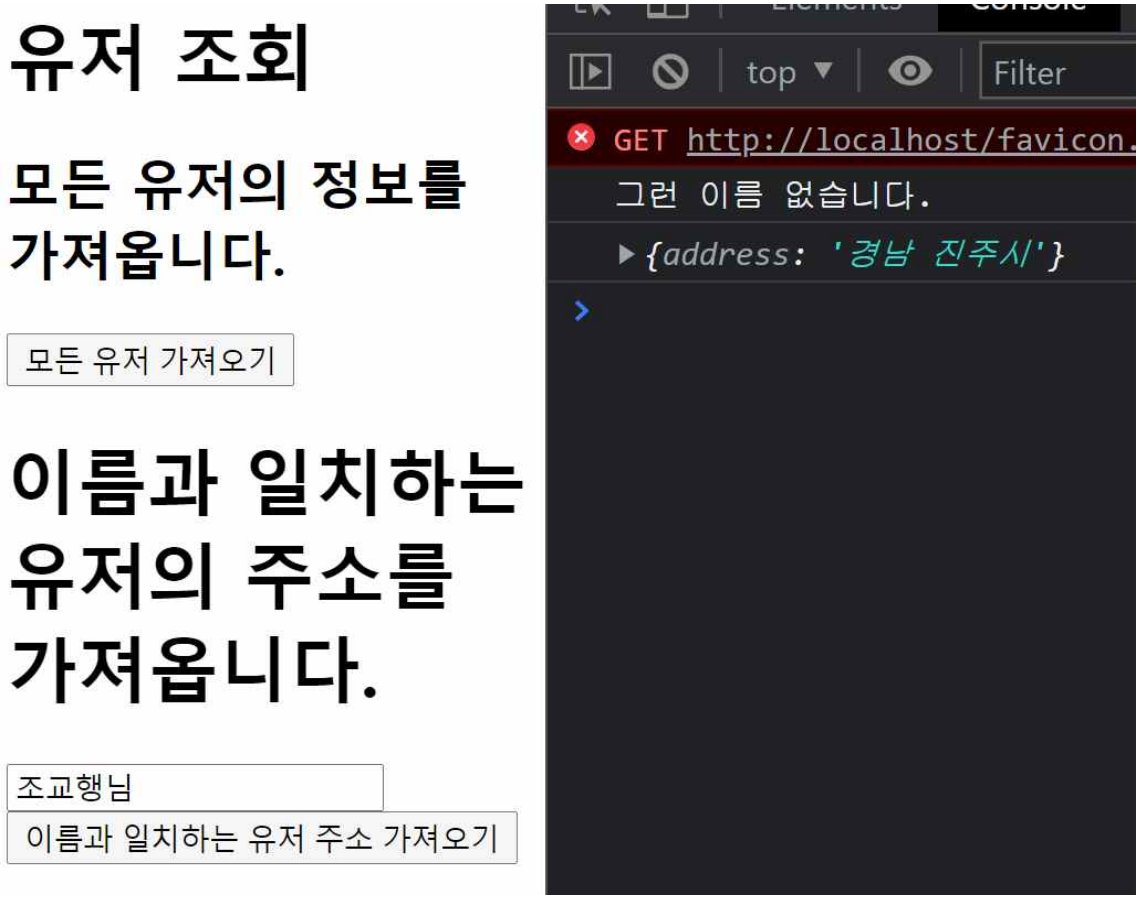
이름과 일치하는 유저의 주소를 가져옵니다.

조교누나

이름과 일치하는 유저 주소 가져오기



조교누나라는 이름은 db에 존재하지 않기 때문에 “그런 이름 없습니다”가 뜬 것이다.
만약 조교행님이라고 치면?



조교행님의 주소인 “경남 진주시” 를 성공적으로 가져왔다.

createUser.php

마지막, user 생성이다. 현재 db에서 user를 늘려보겠다.
먼저, 이벤트핸들러부터 달아야한다. createUser.html 로 가서 button 에 달아주자.

```
<button onclick="createUser()">유저 생성하기</button>
```

또 하나 신경써야할 것이 있는데, createUser.php 파일에도 axios cdn 을 포함시켜야한다는것이다.

createUser.html > html

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-sca
    <script src="./createUser.js"></script>

    <!-- axios -->
    <script src="https://cdn.jsdelivr.net/npm/axios@0.24.0/dist/ax

    <title>유저 생성</title>
  </head>
```

이제 createUser.js 에 가서, 다음과 같이 작성하자.

JS createUser.js > ...

```javascript
const createUser = async () => {
  const nameInput = document.querySelector(".nameInput").value;
  const addressInput = document.querySelector(".addressInput").value;
  if (nameInput && addressInput) {
    try {
      const response = await axios.post("./php/createUser.php", {
        nameInput: nameInput,
        addressInput: addressInput,
      });
      if (response.data) {
        console.log(response.data);
      } else {
        console.log("입력 실패");
      }
    } catch (error) {
      console.log(error);
    }
  }
};
```

line1: 역시 비동기 포함이므로 async

line2, 3: 여기선 입력을 두 개 받아야한다.
유저를 생성하기위해 사용자 이름과 주소가 필요하다.

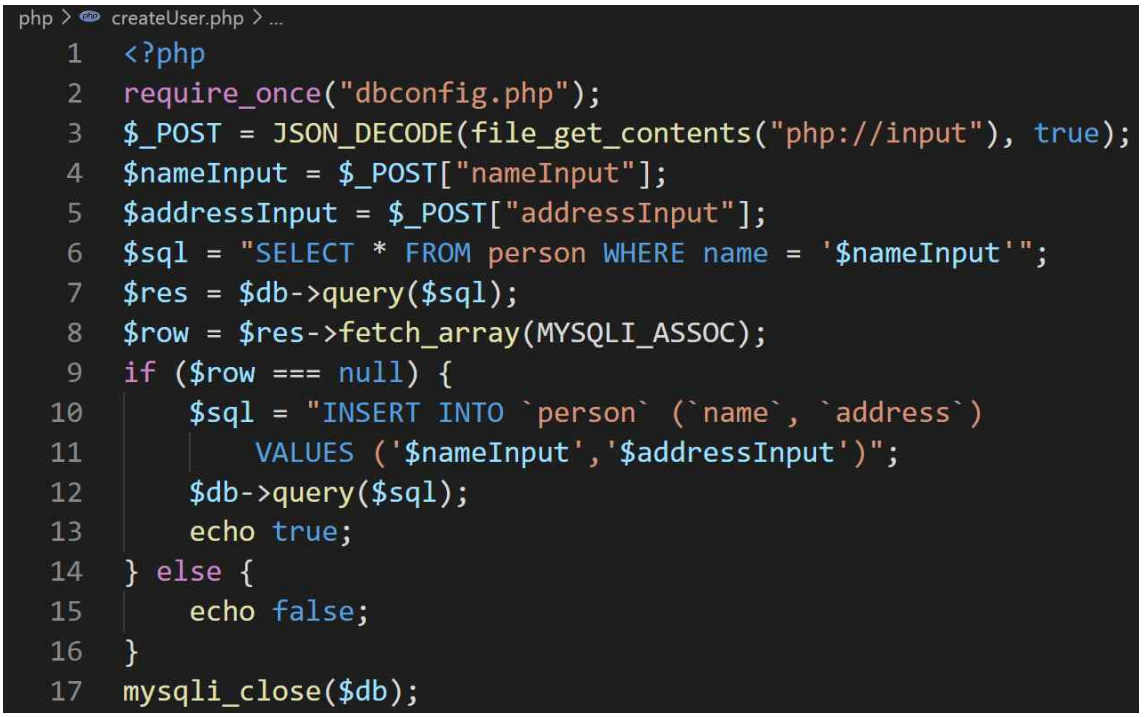
line4: 만약 사용자가 둘 다 입력했으면,

line6: createUser.php 파일로 이번엔 두 개를 보낸다.
nameInput, addressInput

line10: 성공 시 true, JavaScript 입장에선 1을 리턴받도록 했다.

성공하면 1이 찍힐 것이다.

line12: 유저 생성 실패 시 “입력 실패” 가 뜨도록 설정했다.
이 경우, 해당 사용자 이름이 이미 있는 경우다.



```php
<?php
require_once("dbconfig.php");
$_POST = JSON_DECODE(file_get_contents("php://input"), true);
$nameInput = $_POST["nameInput"];
$addressInput = $_POST["addressInput"];
$sql = "SELECT * FROM person WHERE name = '$nameInput'";
$res = $db->query($sql);
$row = $res->fetch_array(MYSQLI_ASSOC);
if ($row === null) {
    $sql = "INSERT INTO `person` (`name`, `address`)
        VALUES ('$nameInput','$addressInput')";
    $db->query($sql);
    echo true;
} else {
    echo false;
}
mysqli_close($db);
```

line4,5 : 이번엔 두 개를 받았기에 변수도 두 개가 필요하다.
line6: 일단, 내가 추가하고싶은 이름이 이미 db에 있을수도 있다.
그래서 SELECT 구문으로 먼저 확인하는 것이다.
line9: if($row === null) 은, sql 실행해봤더니 그런 아이디가 없다는 뜻이다.
즉, 동일한 아이디가 없으니 INSERT 를 통해 유저를 추가해도 된다.
line10: $sql 변수는 재활용 가능하다. INSERT 실행
line11: 클라이언트로 true 를 보낸다. 즉, 데이터 추가가 성공했다는 뜻이다.
JavaScript 에서 받을 땐 true 가 아니라 1이다.
line12: 동일한 아이디가 있으므로, 데이터 추가를 진행할 수 없다.
클라이언트에 false 를 보낸다. 즉, 데이터 추가가 실패했다는 뜻이다.
JavaScript 에서 받을 땐 false 가 아니라 0이다.
line17: DB를 닫아준다.

결과:

유저 생성

조교행님ㅇㅇ

ㅇㅇㅇ

유저 생성하기

첫번째 시도엔 조교행님으로 진행해서, 동일한 아이디가 있으므로 입력 실패.
두번째 시도엔 "조교행님ㅇㅇ" 이므로 동일한 아이디가 없는데다가 주소까지 의미없는 데이터를 넣었다. 별다른 조치를 하지 않았으므로 1이 출력된 것이 확인 가능하다.

TablePlus 로 확인해보면

id	name	address
1	조교행님	경남 진주시
2	전인혁	목포시
3	황승현	마산시
4	조교행님ㅇㅇ	ㅇㅇㅇ

성공적으로 추가되었음을 알 수 있다.

php 는 여기까지다. 이제 여러분은 실전을 위한 모든 걸 갖추었다.

다음 영상부터 조교도시락을 만들러 떠나볼것인데, 조교도시락의 모든 걸 구현하진 않고, 도전해볼만한 기능들만 만들어볼 것이다. 나머지는 여러분들에게 맡기도록 하겠다.