1. How to do config database in Laravel

 Install **Laravel (if you haven't already):** Ensure you have Composer installed, then run:

composer create-project --prefer-dist laravel/laravel project\_name

 Create **a Database** Create a database in your preferred DBMS (MySQL, PostgreSQL, SQLite, SQL Server, etc.).

 Configure **Environment Variables:** Open the .env file in the root directory of your Laravel project and set your database connection details. For example, for a MySQL database:

DB\_CONNECTION=mysql

DB\_HOST=127.0.0.1

DB\_PORT=3306

DB\_DATABASE=your\_database\_name

DB\_USERNAME=your\_username

DB\_PASSWORD=your\_password

 Update **config/database.php:** Laravel uses the config/database.php file to manage database configurations. The default settings are usually sufficient, but you can customize them as needed. For example, you can set the default connection to MySQL:

'default' => env('DB\_CONNECTION', 'mysql'),

'connections' => [

'mysql' => [

'driver' => 'mysql',

'host' => env('DB\_HOST', '127.0.0.1'),

'port' => env('DB\_PORT', '3306'),

'database' => env('DB\_DATABASE', 'forge'),

'username' => env('DB\_USERNAME', 'forge'),

'password' => env('DB\_PASSWORD', ''),

'unix\_socket' => env('DB\_SOCKET', ''),

'charset' => 'utf8mb4',

'collation' => 'utf8mb4\_unicode\_ci',

'prefix' => '',

'strict' => true,

'engine' => null,

],

],

 **Run Migrations:** To create the necessary tables in your database, you can run Laravel migrations. Ensure your database is set up and accessible, then run:

php artisan migrate

 **Test the Connection:** You can test if your Laravel application is connected to the database by running a simple query. For example, you can use Tinker:

php artisan tinker

Then, within Tinker, try to fetch some data:

DB::table('users')->get();

1. Call MySQLi Store Procedure from Laravel.

 **Step 1:** Install the MySQLi extension if you haven’t already.

sudo apt-get install php-mysqli

**Step 2:** Create a custom class or service to handle MySQLi connections and procedure calls. For example, you can create a new class in the app directory named MySQLiService.php:

<?php

namespace App\Services;

class MySQLiService

{

protected $connection;

public function \_\_construct()

{

$this->connection = new \mysqli(

env('DB\_HOST'),

env('DB\_USERNAME'),

env('DB\_PASSWORD'),

env('DB\_DATABASE')

);

if ($this->connection->connect\_error) {

die("Connection failed: " . $this->connection->connect\_error);

}

}

public function callProcedure($procedureName, $params = [])

{

$paramStr = implode(',', array\_fill(0, count($params), '?'));

$stmt = $this->connection->prepare("CALL $procedureName($paramStr)");

if ($stmt === false) {

die("Prepare failed: " . $this->connection->error);

} if (!empty($params)) {

$types = str\_repeat('s', count($params));

$stmt->bind\_param($types, ...$params);

}

$stmt->execute();

$result = $stmt->get\_result();

$data = [];

while ($row = $result->fetch\_assoc()) {

$data[] = $row;

}

$stmt->close();

$this->connection->close();

return $data;

}

}

**Step 3:** Use the service in your controller or wherever you need to call the stored procedure:

namespace App\Http\Controllers;

use App\Services\MySQLiService;

class UserController extends Controller

{

public function index()

{

$mysqliService = new MySQLiService();

$users = $mysqliService->callProcedure('GetUsers');

return response()->json($users);

}

}

 **Route Configuration:**

Ensure you have a route set up to access the controller method:

use App\Http\Controllers\UserController;

Route::get('/users', [UserController::class, 'index']);

 **Test the Endpoint:**

Finally, test your endpoint by visiting http://your-domain/users in your browser or using a tool like Postman to ensure it returns the expected data from your stored procedure.