Step 1:

At first we have to create the project by running the following command in CLI:

```
composer create-project laravel/laravel my_pos_project
```

Step 2:

Now we have to open the .env file in the root directory from the project and update the database connection settings as our database credentials.

Step 3:

Now we have to install the tymon/jwt-auth package using Composer by running the following command:

```
composer require tymon/jwt-auth
```

After the package installation, have to run the following command to publish the JWT configuration file:

```
php artisan vendor:publish --provider="Tymon\JWTAuth\Providers\LaravelServiceProvider"
```

This will create a config/jwt.php file. Now we have to update this file by following:

```
'guards' => [

    'api' => [

        'driver' => 'jwt',

        'provider' => 'users',

    ],

],


'providers' => [

    'users' => [

        'driver' => 'eloquent',

        'model' => App\Models\User::class,

    ],

],
```

Step 4:

Now we have to generate a JWT secret key by running the following command:

```
php artisan jwt:secret
```

This will update .env file with a JWT_SECRET value.

Step 5:

Now we have to create an authentication controller by the following command:

```
php artisan make:controller AuthController
```

Now we have to add the necessary methods for registration, login, and token retrieval in AuthController file by following:

```
<?php

namespace App\Http\Controllers;


use App\Models\User;

use Illuminate\Http\Request;

use Illuminate\Support\Facades\Auth;

use Illuminate\Support\Facades\Validator;

use Tymon\JWTAuth\Facades\JWTAuth;

use Tymon\JWTAuth\Exceptions\TokenExpiredException;

use Tymon\JWTAuth\Exceptions\TokenInvalidException;

use Tymon\JWTAuth\Exceptions\JWTException;

use Symfony\Component\HttpKernel\Exception\HttpExceptionInterface;



class AuthController extends Controller

{

    public function register(Request $request)
```

```
{
    $validator = Validator::make($request->all(), [
        'name' => 'required',
        'email' => 'required|email|unique:users',
        'password' => 'required|min:6',
    ]);

    if ($validator->fails()) {
        return response()->json(['error' => $validator->errors()], 400);
    }
    $user = new User();
    $user->name = $request->name;
    $user->email = $request->email;
    $user->password = bcrypt($request->password);
    $user->save();
    return response()->json(['message' => 'Registration successful'], 201);
}
public function login(Request $request)
{
    $credentials = $request->only('email', 'password');

    if (!$token = JWTAuth::attempt($credentials)) {
        return response()->json(['error' => 'Invalid credentials'], 401);
    }
    return response()->json(['token' => $token], 200);
    }

    public function getAuthenticatedUser()

    {

        try {

            if (!$user = JWTAuth::parseToken()->authenticate()) {

                return response()->json(['error' => 'User not found'], 404);

            }

        } catch (TokenExpiredException $e) {

            return response()->json(['error' => 'Token expired'], $e->getStatusCode());

        } catch (TokenInvalidException $e) {

            return response()->json(['error' => 'Invalid token'], $e->getStatusCode());

        } catch (JWTException $e) {

            return response()->json(['error' => 'Token absent'], 500);

        } catch (HttpExceptionInterface $e) {

            return response()->json(['error' => $e->getMessage()], $e->getStatusCode());

        }

        return response()->json(['user' => $user], 200);

    }

}
```

Step 6:

Now we have to define the necessary routes for registration, login, and retrieving the authenticated user as following:

```
<?php

use Illuminate\Http\Request;

use Illuminate\Support\Facades\Route;

use App\Http\Controllers\AuthController;
```

```
/*
|--------------------------------------------------------------------------
| API Routes
|--------------------------------------------------------------------------
|
| Here is where you can register API routes for your application. These
| routes are loaded by the RouteServiceProvider and all of them will
| be assigned to the "api" middleware group. Make something great!
|
*/
Route::post('register', [AuthController::class, 'register']);
Route::post('login', [AuthController::class, 'login']);
Route::get('user', [AuthController::class, 'getAuthenticatedUser'])->middleware('jwt.auth');
```

Step 7:

Anyone can now test the authentication system using a tool like Postman. Can send a POST request to /api/register with the required user information to register a new user. Then, can send a POST request to /api/login with the user's credentials to retrieve a JWT token. Finally, can send a GET request to /api/user with the token in the Authorization header to retrieve the authenticated user's information