# API Authentication



## # Introduction

By default, Laravel ships with a simple solution to API authentication via a random token assigned to each user of your application. In your `config/auth.php` configuration file, an `api` guard is already defined and utilizes a `token` driver. This driver is responsible for inspecting the API token on the incoming request and verifying that it matches the user's assigned token in the database.

> **Note:** While Laravel ships with a simple, token based authentication guard, we strongly recommend you consider using Laravel Passport for robust, production applications that offer API authentication.

## # Configuration

### # Database Preparation

Before using the `token` driver, you will need to create a migration which adds an `api_token` column to your `users` table:

```
Schema::table('users', function ($table) {
    $table->string('api_token', 80)->after('password')
                          ->unique()
                          ->nullable()
                          ->default(null);
});
```

Once the migration has been created, run the `migrate` Artisan command.

If you choose to use a different column name, be sure to update your API's `storage_key` configuration option within the `config/auth.php` configuration file.

## # Generating Tokens

Once the `api_token` column has been added to your `users` table, you are ready to assign random API tokens to each user that registers with your application. You should assign these tokens when a `User` model is created for the user during registration. When using the authentication scaffolding provided by the `laravel/ui` Composer package, this may be done in the `create` method of the `RegisterController`:

```
use Illuminate\Support\Facades\Hash;
use Illuminate\Support\Str;

/**
 * Create a new user instance after a valid registration.
 *
 * @param  array  $data
 * @return \App\User
 */
protected function create(array $data)
{
    return User::forceCreate([
        'name' => $data['name'],
        'email' => $data['email'],
        'password' => Hash::make($data['password']),
        'api_token' => Str::random(80),
    ]);
}
```

### # Hashing Tokens

In the examples above, API tokens are stored in your database as plain-text. If you would like to hash your API tokens using SHA-256 hashing, you may set the `hash` option of your `api` guard configuration to `true`. The `api` guard is defined in your `config/auth.php` configuration file:

```
'api' => [
    'driver' => 'token',
    'provider' => 'users',
    'hash' => true,
],
```

#### Generating Hashed Tokens

When using hashed API tokens, you should not generate your API tokens during user registration. Instead, you will need to implement your own API token management page within your application. This page should allow users to initialize and refresh their API token. When a user makes a request to initialize or refresh their token, you should store a hashed copy of the token in the database, and return the plain-text copy of token to the view / frontend client for one-time display.

For example, a controller method that initializes / refreshes the token for a given user and returns the plain-text token as a JSON response might look like the following:

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use Illuminate\Support\Str;

class ApiTokenController extends Controller
{
    /**
     * Update the authenticated user's API token.
     *
     * @param  \Illuminate\Http\Request  $request
     * @return array
     */
    public function update(Request $request)
    {
        $token = Str::random(80);

        $request->user()->forceFill([
            'api_token' => hash('sha256', $token),
        ])->save();

        return ['token' => $token];
    }
}
```

Since the API tokens in the example above have sufficient entropy, it is impractical to create "rainbow tables" to lookup the original value of the hashed token. Therefore, slow hashing methods such as `bcrypt` are unnecessary.

## # Protecting Routes

Laravel includes an authentication guard that will automatically validate API tokens on incoming requests. You only need to specify the `auth:api` middleware on any route that requires a valid access token:

```
use Illuminate\Http\Request;

Route::middleware('auth:api')->get('/user', function(Request $request) {
    return $request->user();
});
```

## # Passing Tokens In Requests

There are several ways of passing the API token to your application. We'll discuss each of these approaches while using the Guzzle HTTP library to demonstrate their usage. You may choose any of these approaches based on the needs of your application.

#### Query String

Your application's API consumers may specify their token as an `api_token` query string value:

```
$response = $client->request('GET', '/api/user?api_token='.$token);
```

#### Request Payload

Your application's API consumers may include their API token in the request's form parameters as an `api_token`:

```
$response = $client->request('POST', '/api/user', [
    'headers' => [
        'Accept' => 'application/json',
    ],
    'form_params' => [
        'api_token' => $token,
    ],
]);
```

#### Bearer Token

Your application's API consumers may provide their API token as a `Bearer` token in the `Authorization` header of the request:

```
$response = $client->request('POST', '/api/user', [
    'headers' => [
        'Authorization' => 'Bearer '.$token,
        'Accept' => 'application/json',
    ],
]);
```

# Become a Laravel Partner

Laravel Partners are elite shops providing top-notch Laravel development and consulting. Each of our partners can help you craft a beautiful, well-architected project.

Our Partners

Laravel

**Highlights**

- Release Notes
- Getting Started
- Routing
- Blade Templates
- Authentication
- Authorization
- Artisan Console
- Database
- Eloquent ORM
- Testing

**Resources**

- Laracasts
- Laravel News
- Laracon
- Laracon EU
- Laracon AU
- Jobs
- Certification
- Forums

**Partners**

- Vehikl
- Tighten Co.
- Kirschbaum
- Byte 5
- 64Robots
- Cubet
- DevSquad
- Ideil
- Cyber-Duck
- ABOUT YOU
- Become A Partner

**Ecosystem**

- Vapor
- Forge
- Envoyer
- Horizon
- Lumen
- Nova
- Echo
- Valet
- Mix
- Spark
- Cashier
- Homestead
- Dusk
- Passport
- Scout
- Socialite

Laravel is a web application framework with expressive, elegant syntax. We believe development must be an enjoyable and creative experience to be truly fulfilling. Laravel attempts to take the pain out of development by easing common tasks used in most web projects.