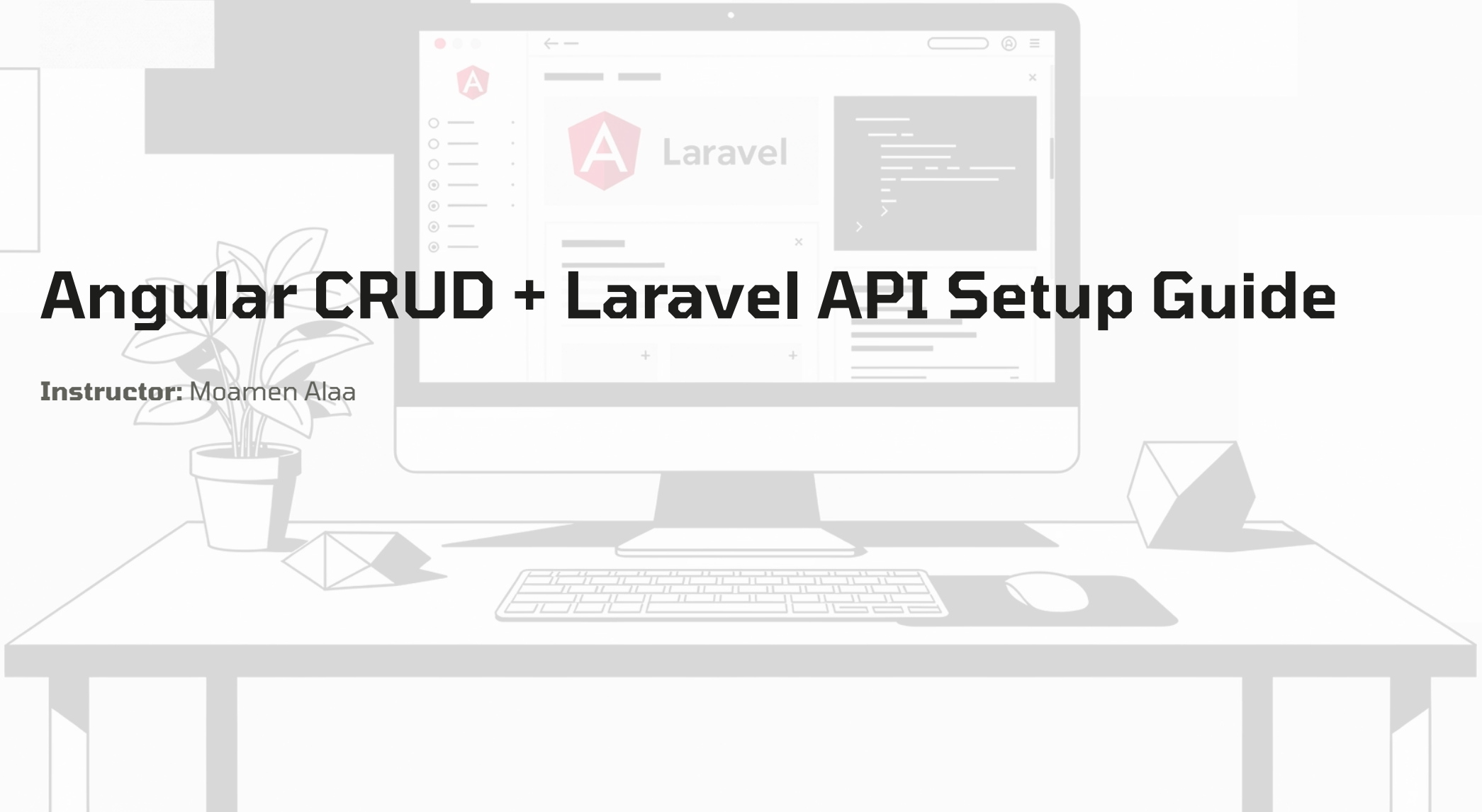


Angular CRUD + Laravel API Setup Guide

Instructor: Moamen Alaa

Prerequisites & Environment Setup

Required Software

Before diving into the installation process, ensure your development machine meets all the necessary requirements. You'll need Node.js version 18 or higher along with npm for managing JavaScript packages. The Angular CLI must be installed globally to scaffold and serve your frontend application. On the backend side, PHP 8.2 or later is required along with Composer for dependency management. Finally, MySQL should be running locally to store your application data.

Installation Commands

```
npm install -g @angular/cli
node --version
npm --version
php --version
composer --version
mysql --version
```

Run these commands in your terminal to verify all tools are properly installed and check their versions.

Node.js 18+

JavaScript runtime for Angular development and npm package management

Angular CLI

Command-line interface for Angular project scaffolding and development

PHP 8.2+

Server-side language powering Laravel's robust backend framework

MySQL Database

Relational database system for storing application data securely

Clone the Repository

The first step in setting up your project is cloning the starter repository from GitHub. This repository contains the complete project structure with both the Angular frontend and Laravel backend pre-configured with basic CRUD operations and authentication scaffolding. Navigate to your preferred development directory and execute the clone command. The repository includes essential files like package.json for Angular dependencies, composer.json for Laravel packages, migration files for database schema, and service classes for API communication. Once cloned, you'll have two main directories: **backend-laravel** containing the Laravel API and **angular-crud** housing the Angular application.

```
git clone https://github.com/moemen03/Lab11
cd Lab11
ls -la
```

After cloning, take a moment to explore the directory structure. You'll notice the separation of concerns with distinct folders for frontend and backend, making it easier to manage each part of the stack independently. The repository also includes configuration files, environment examples, and documentation to help you understand the project architecture before diving into setup.

Clone Repository

Download the project files from GitHub

Explore Structure

Familiarize yourself with the project layout

Begin Setup

Proceed with backend configuration

Backend Laravel API Setup

Setting up the Laravel backend involves several critical steps that establish your API foundation. Start by navigating into the backend-laravel directory and installing all PHP dependencies using Composer. This process downloads Laravel's core framework along with additional packages like Sanctum for API authentication. The installation might take a few minutes depending on your internet connection and system performance, but it's essential for getting all the necessary tools in place.

01

Navigate to Backend

Change directory into the Laravel project folder

```
cd backend-laravel
```

02

Install Dependencies

Use Composer to download all required PHP packages

```
composer install
npm install
```

03

Configure Environment

Create and customize your .env configuration file

```
cp .env.example .env
```

04

Generate Application Key

Create a unique encryption key for your Laravel app

```
php artisan key:generate
```

The composer install command reads the composer.json file and downloads all dependencies into the vendor directory. This includes the Laravel framework itself, database drivers, authentication packages, and utility libraries. Make sure you have a stable internet connection during this process, as interruptions can lead to incomplete installations that may cause errors later in development.

Database Configuration

Creating Your Database

Before Laravel can interact with MySQL, you need to create a database manually. Open your MySQL client or use the command line to create a new database. A common convention is to name it something descriptive like **cruddb** or **laravel_crud**. This database will store all your application tables including users, API tokens, and your CRUD entity data.

```
mysql -u root -p
CREATE DATABASE cruddb;
SHOW DATABASES;
EXIT;
```

Environment Variable Configuration

Open the **.env** file you created earlier in your favorite text editor. This file contains all environment-specific settings that shouldn't be committed to version control. Locate the database configuration section and update the following variables to match your MySQL setup. The DB_CONNECTION should remain as mysql, DB_HOST is typically 127.0.0.1 or localhost, DB_PORT defaults to 3306, DB_DATABASE should match the database you just created, and DB_USERNAME and DB_PASSWORD should correspond to your MySQL credentials. Double-check these values as incorrect configuration is the most common source of connection errors.

DB_CONNECTION

mysql

Specifies the database driver

DB_HOST

127.0.0.1

Database server address

DB_DATABASE

cruddb

Name of your database

DB_USERNAME

root

MySQL user account

Laravel Sanctum & Migrations

Laravel Sanctum provides a lightweight authentication system perfect for SPAs (Single Page Applications) and mobile applications. It issues API tokens that your Angular frontend will use to authenticate requests. Before running migrations, you must publish Sanctum's migration files to your project. This step copies the necessary database table schemas into your migrations directory, allowing Laravel to create the tables needed for token-based authentication.

1

Publish Sanctum Migrations

Copy Sanctum's table schemas to your project

```
php artisan vendor:publish --provider="Laravel\Sanctum\SanctumServiceProvider" --tag=sanctum-migrations
```

2

Run All Migrations

Create database tables from migration files

```
php artisan migrate
```

3

Verify Tables

Check that all tables were created successfully

```
php artisan migrate:status
```

The migration process creates several essential tables including **users** for storing account information, **personal_access_tokens** for managing API tokens, **password_resets** for handling password recovery, and any custom tables defined in your project migrations. Each migration file is timestamped and tracked by Laravel, allowing you to roll back changes if needed. After running migrations, verify their success by checking your database directly or using the migrate:status command to see which migrations have been executed.

Starting the Laravel Development Server

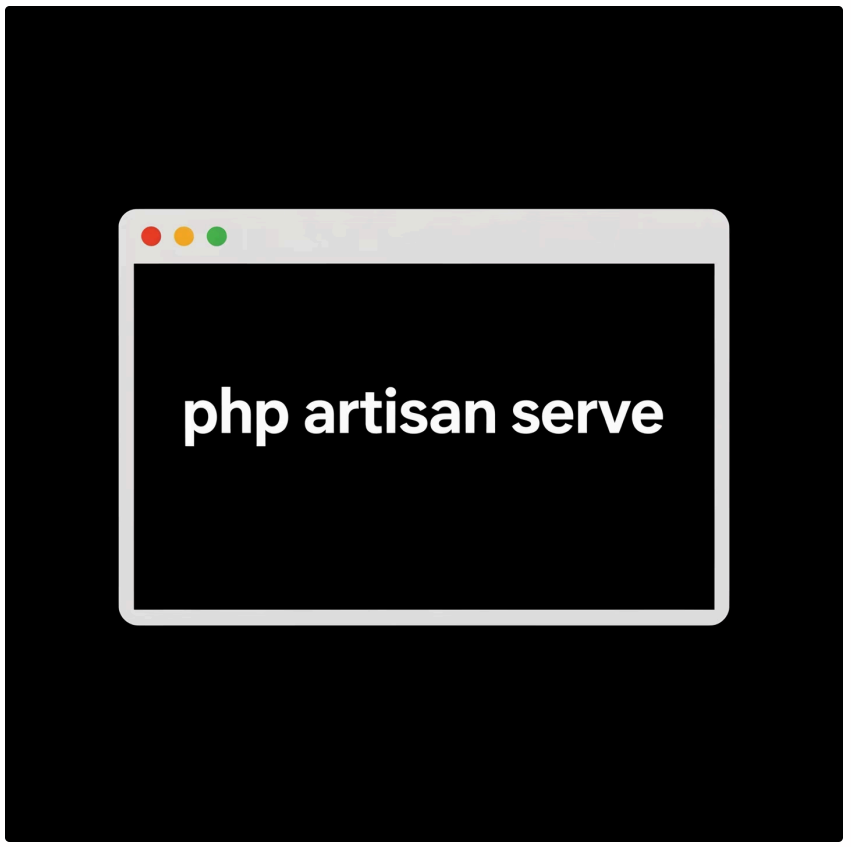
Launch Your API

With your database configured and migrations complete, you're ready to start the Laravel development server. The **php artisan serve** command launches a built-in PHP development server that listens for incoming HTTP requests. By default, it runs on **http://127.0.0.1:8000**, making your API accessible locally. This server is suitable for development but should never be used in production environments.

```
php artisan serve

Starting Laravel development server:
http://127.0.0.1:8000
[Ctrl+C to quit]
```

Keep this terminal window open while developing. The server will log all incoming requests, making it easy to debug API calls from your Angular frontend.



API Endpoint Structure

Your Laravel API endpoints are available under the **/api** prefix, meaning all routes are accessible at **http://127.0.0.1:8000/api**. This project includes authentication endpoints for user registration and login, as well as CRUD operations for managing resources. The API follows RESTful conventions with GET requests for retrieving data, POST for creating new records, PUT or PATCH for updates, and DELETE for removing records. Each endpoint returns JSON responses that your Angular application will consume and display in the user interface.

POST /api/register

Create new user accounts with email and password

POST /api/login

Authenticate users and receive API token

GET /api/items

Retrieve all CRUD items from database

Angular Frontend Setup

Now that your backend API is running, it's time to set up the Angular frontend that will communicate with it. Angular provides a powerful framework for building dynamic single-page applications with a component-based architecture. The setup process involves installing Node.js dependencies and starting the development server that compiles your TypeScript code and serves the application with hot-reload capabilities.

1

Navigate to Frontend

Move into the Angular project directory

```
cd angular-crud
```

2

Install Dependencies

Download all npm packages specified in package.json

```
npm install
```

3

Start Dev Server

Launch the Angular development server with live reload

```
npm start
```

The **npm install** command reads the package.json file and downloads all dependencies into the node_modules directory. This includes Angular core libraries, routing modules, HTTP client for API calls, RxJS for reactive programming, and development tools like the TypeScript compiler. The installation creates a node_modules folder that can be quite large, often containing thousands of files—this is normal for modern JavaScript projects. Once complete, **npm start** compiles your TypeScript code, bundles assets, and launches a development server typically on **http://localhost:4200**. The terminal will display compilation progress and any errors that need attention. When you see "Compiled successfully," open your browser and navigate to localhost:4200 to view your application.

Configuration & Authentication Flow

API Endpoint Configuration

The Angular application is pre-configured to communicate with your Laravel backend through the auth.service.ts file located in the src/app directory. This service contains the base URL for API calls, which should point to **http://127.0.0.1:8000/api**. All HTTP requests for authentication and CRUD operations flow through this service, which handles request headers, token management, and response processing. If your Laravel server runs on a different address or port, update the baseUrl constant in this file accordingly.

Registration Process

When users register through the Angular interface, the application sends a POST request to **/api/register** with email and password fields. The Laravel backend validates the input, creates a new user record in the database, and returns the user object as JSON. Angular then typically redirects to the login page or automatically logs in the new user. Error handling is built in to display validation messages if registration fails due to duplicate emails or weak passwords.

Login & Token Storage

The login flow sends credentials to **/api/login**, and upon successful authentication, Laravel Sanctum generates an API token. Angular receives this token and stores it in the browser's localStorage, making it persistent across page refreshes. Subsequent API requests include this token in the Authorization header as a Bearer token, allowing Laravel to identify and authorize the authenticated user for protected routes.

1 User Submits Form

Angular captures login credentials from input fields

2 API Request Sent

HTTP POST to Laravel with email and password

3 Token Generated

Laravel Sanctum creates personal access token

4 Token Stored

Angular saves token in localStorage for future requests

Troubleshooting Common Issues

Personal Access Tokens Table Error

One of the most frequent issues developers encounter is the "personal_access_tokens table not found" error during login or registration. This occurs when Laravel Sanctum's migration files weren't properly published or executed. The solution is straightforward: re-publish the Sanctum migrations and run them again. First, execute the vendor:publish command with Sanctum's provider and tag parameters to copy the migration files into your project. Then run php artisan migrate to create the missing table. If you've already run migrations before, Laravel is smart enough to skip the ones that already exist and only create the missing tables.

CORS Errors

If Angular can't connect to Laravel, check that CORS middleware is properly configured in Laravel's cors.php config file. Add your Angular URL to allowed origins.

Port Conflicts

If port 8000 or 4200 is already in use, specify a different port: **php artisan serve --port=8001** or **ng serve --port=4201** and update your configuration accordingly.

Database Connection Failed

Verify your .env database credentials are correct and that MySQL is running. Test the connection with: **php artisan tinker** then **DB::connection()->getPdo();**

npm Install Errors

Clear npm cache with **npm cache clean --force** and delete node_modules folder, then run npm install again. Ensure you're using a compatible Node.js version.

Additional Tips

Always run both servers simultaneously during development—the Laravel API server in one terminal and the Angular dev server in another. Watch the terminal outputs for errors and warnings that provide valuable debugging information. If authentication suddenly stops working, check that your API token hasn't expired or been cleared from localStorage. Use browser developer tools to inspect network requests and verify that requests are reaching the correct endpoints with proper headers. Finally, remember that Laravel caches configuration files, so after changing .env values, run **php artisan config:clear** to ensure Laravel picks up the new settings.