PHP Laravel Directory Structure Best Practices

Laravel is a popular PHP framework that comes with a well-defined directory structure in itself. Here are some best practices for structuring PHP Laravel projects:

1. Follow the Default Structure: Laravel provides a default directory structure that aligns with industry best practices. Stick to this structure as it's widely recognized and understood by the Laravel community. There is no point in reinventing the wheel!

2. Division of Code by Type:
- Routes: Define routes in the **routes** directory, keeping them organized by functionality. For example, create a file `web.php` file for web routes and `api.php` for API routes.
- Models: Models are meant to be used for database-related operations and hence all queries should be driven from models.
- Controllers: Controllers are the bridge connecting the data and the views. Hence, all exchange between the database (i.e. models) and views should be done using Controllers.
- Views: Views are supposed to be the **dumbest** part of the application in the sense that there should be absolutely no logic placed in them. They should just render whatever data the controller gives them. Keep views in the `resources/views` directory.
- Middleware: Store custom middleware classes in the `app/Http/Middleware` directory. These classes are very useful to apply a system wide security and authentication mechanism.

4. Configuration Files: Keep configuration files in the `config` directory. You can publish vendor configurations using `php artisan vendor:publish`. Laravel comes with prebuilt support for .env files and hence all environment specific configuration values should always be kept in the env file.

5. Database Migrations and Seeders: Database migrations and seeders are very important to setup database for the application in the event of a fresh deployment. Storing database migration files in the `database/migrations` directory and seeders in `database/seeders` directory can allow us to easily setup the necessary database structure and initialisation data that is required by the application.

6. Localisation and Translations: Most applications these days are supporting multiple languages i.e. internationalisation, to easily show different strings based on the preferred language of the user, store language files in the `resources/lang` directory.

7. Assets (CSS, JavaScript, Images): Place your assets in the `public` directory organised by type (css, js, images).

8. Jobs and Queues: Laravel comes with inbuilt capability for executing cron jobs and background processes unlike most other PHP frameworks. You can store your cron job classes in the `app/Jobs` directory and organise them by functionality. You can also use queues for background processing.

9. Dependency Management: Laravel uses Composer to manage project dependencies. Keep the `composer.json` file up-to-date and version-controlled. Ensure no files are ever directly changed/manipulated in the `vendor` directory.

10. Artisan Commands: Although the default artisan commands are enough to perform almost all necessary operations, if there is a need for you to creating custom Artisan commands, place them in the `app/Console/Commands` directory.

11. Documentation: It's very important to have a simple documentation file such as a `README.md` to explain the purpose of different directories, how to set up the project on local, and any specific guidelines to deploy and get the project running on a server.

12. Version Control: Always use a version control system like Git to manage your project. Make sure to include a `.gitignore` file to exclude unnecessary files from version control. As a rule of thumb, all user generated content like uploaded files etc. should be placed in gitignore as well as all external dependencies (node_modules, vendor, etc.) that we can easily download with package managers should be excluded from the git repo as well.

Remember, the key is to maintain consistency and make it easy for yourself and other developers to understand the structure of your project. As your project evolves, periodically review and refactor your file structure to ensure it remains organised and scalable. Also, spend some time every month to clean up unnecessary code, files and assets from the project so the technical debt never gets overwhelming as the project grows.

Revision #1
Created 21 August 2023 04:30:22 by Nilesh Rathour
Updated 21 August 2023 04:45:51 by Nilesh Rathour