1. PROJECT DETAILS

* 1. Design Patterns:

MVC

Repository

It is designed with the Repository Pattern, which allows a Controller to pass data to the Repository which in turn interfaces with the Model.

This is achieved by injecting a repository Dependency in the Controller, so that the repository implements its interface with the model, starting in the Controller.

OOP (Interfaces, Namespaces, Encapsulation, Information Hiding, properties and methods Accessibility, Static, Dynamic, etc.).

All this ensures the code is:

Testable (injection are mocked)

Maintainable (OOP not procedural)

Reduced code (reusable), and

Less bugs (info hiding, accessibility, encapsulation, etc)

Note: The specifics for this DEVELOPMENT ENVIRONMENT follows:

Virtual Box, Vagrant, Centos 7, PHP 5.6, MySQL 14.14, PHPMyAdmin, git, composer, artisan, pear, PHPStorm 2017 with XDebug, Code Sniffer (phpcs), GitHub);

1.2 Web Server requirements:

PHP >= 5.6.4

OpenSSL PHP Extension

PDO PHP Extension

Mbstring PHP Extension

Tokenizer PHP Extension

XML PHP Extension

2. FRONT-END TECHNICAL DETAILS

INPUT:

The blade template receives the the associative array() containing the config.XML data

2.2 PROCESS:

The blade engine iterates, displays and attributes (custom 'data-\*' attributes) this associative array() data

JQuery stores these 'data-\*' into variables

A JavaScript Object (class) initializes JQuery, binds/fires the button click event, and processes server response

JQuery asynchronously post() the data to '/create'

OUTPUT:

Client is notified when sending/saving ('button is disabled'), and on saved ('on success button is enabled')

3. BACK-END TECHNICAL DETAILS

3.1 INPUT:

a. The server accepts two (get/post) requests in the routes, namely 'receive' and 'create'

b. Request handling logic are mapped to 2 Controller class methods respectively

3.2 PROCESS:

a. An Utility class parses the config.XML file, using the native 'simplexml\_load\_file()' PHP function

b. 'simplexml\_load\_file(config.XML)' returns an associative array() containing the config.XML data (ex: 'title', 'description' ...)

c. These data is persisted in the database during the click event process in front-end

Note: This project uses the SimpleXML - PHP5’s new API for accessing the contents of XML documents.

3.3 OUTPUT:

a. The Controller returns the package view with this associative array()

b. When the data is to be persisted to the db, the Controller responds to the asynchronous call appropriately

2. FRAMEWORK

2.1 Laravel 5.4 (PHP v5.6)

Q: Why Laravel?

A: Choices! Everybody is using it and it works well. I just like the brand, though some online resources ought to be free, as is most community support

4. USAGE

4.1 Basic Setup:

a. Copy this 'dario\_coding-test' folder to destination web server's folder

b. Run the following command inside this app's root directory

composer install

c. Create a database called 'eon' or another name

Note: MySQL driver is used. If different driver then change this '.env' file so it matches the destination's server environment:

Ex: DB\_CONNECTION='destination db connection'

DB\_HOST='destination IP address'

DB\_PORT='destination DBMS connection port'

DB\_DATABASE='destination DBMS db name'

DB\_USERNAME='destination DBMS user name'

DB\_PASSWORD='destination DBMS user password'

d. Install 'output' table and migrations: Run the following command inside the app's root directory

php artisan migrate:refresh --seed

Note: The command above will install Laravel's 'users' and other table migrations

5. ASSUMPTIONS

Given the spec, I assumed the following:

a. Routes:

'receive' and 'create'

b. The config.XML File:

Because it is a configuration file, I assumed that the structure will not change often ('from time to time').

So the code is strongly coupled around the given config file, and will choke with a different XML file structure.

6. Plugins and 3rd Party Libraries:

All functionality required to fulfill most of this project's requirements was implemented with the Laravel ecosystem and PHP native functions.

7. Coding Standards:

PSR-1/2/3/4

8. SCALING CONSIDERATIONS

a. This app design is Decoupled with separate concerns for Presentation and Business logic, allowing for scalability of this package

Decoupling also increases the lifetime since it is not tied to any technology or concrete implementations

b. Laravel's cache design is optimal as well as the management of pool resources such as database connections and termination

c. No Form here means no need to store flashed old input Session Data in the event of errors

d. Packages folder structure enables future growth, such as Models, Assets and other sub-folder structures

8. PERFORMANCE CONSIDERATIONS

a. Data is posted asynchronously so the UI (User Interface) is not blocked or stalled, improving performance

b. Laravel's cache design is optimal as well as the management of pool resources such as database connections and termination

c. No Form here means no need to store flashed old input Session Data in the event of errors

d. For PHPUnit testing performance this app does not access the model directly. So it is possible to bypass database, otherwise, would be expensive to make database calls