DEVELOPER BEST FRIEND

1 Purpose

This document outlines the strategy for deploying the Developer Best Friend web application leveraged by the Laravel framework. The Developer Best Friend web application functions as a central point of access to core functional tools allowing developers to retrieve readily available data to expedite the release of web applications.

2 Scope

The initial deployment effort applies exclusively to the following components:

- Lorem Ipsum Generator
- Profile Maker
- Chmod Cruncher
- XKCD Password Generator

3 Tools and Resources

- Composer
- Git and GitHub
- XAMPP v3.2.1 or greater - development
- Digital Ocean (droplet) – production
- Laravel 5.1 or greater
- Bootstrap v3.3.5 - getbootstrap.com
- Depositphotos.com
- FontAwesome V3.2.1 - fortawesome.github.io
- Badcow/lorem-ipsum

4 Requirements and Risk Scenarios

MVP Minimum Viable Product (core features for go live)

Non-MVP (Non-Minimum Viable Product)

	Requirement	Description	Category (MVP)
ID	**Home Page**		
ID-1	Intro (text/html)	Describe the intent of the web app	MVP
ID-2	Background (image/jpg)	Background image	MVP
ID-3	Nav. Icons (FontAwesome)	Need navigation icons for the following: • **Dashboard (MVP)** • **Lorem Ipsum Generator (MVP)** • **Profile Maker (MVP)**	MVP

	Requirement	Description	Category (MVP)
		• Chmod Cruncher • XKCD Password Generator	
ID-4	Dashboard	Return user to the main (home) page	MVP
ID-5	LIG (lorem Ipsum Gnerator) Form	Form must contain the following: Label => “How many paragraphs do you want?” Text => text box to enter number of paragraphs Submit => submit button	MVP
ID-6	LIG Landing page	Landing page to generate the paragraphs returned by the number entered by the user	MVP

ID-7	PRM (Profile Maker) – Form	Form must contain the following:
		Label => “How many users?”
		Text => text box to enter number of users
		Checkbox => to generate birthday
		Checkbox => to generate profile information
		Submit => submit button
			MVP

ID-8	PRM Landing page User name (text/html) User main photo (image/jpg) User about (text/html) User email	Landing page to generate the user information: User name Profile image About the user User email	MVP

ID-9	Chmod Cruncher (CHC)
	Form
		Form must contain the following:
		Unix Permissions
		
		Label => Owner permissions
		Checkbox => Read
		Checkbox => Write
		Checkbox => Execute
		Label => Group permissions
		Checkbox => Read
		Checkbox => Write
		Checkbox => Execute
		Label => Public permissions
		Checkbox => Read
		Checkbox => Write
		Checkbox => Execute
		Submit => submit button
			NON-MVP

ID-10	CHC Landing Page	Display permissions	NON-MVP
ID-11	XKCD Password Generator (XKCD) Integration	Integrate the XKCD Password Generator from P2.	NON-MVP

5 Technical Specification

5.1 Route Plan

Purpose	Method	URI
Homepage	GET	/
Show Dashboard	GET	/
Show Lorem Ipsum Generator - LIG	ANY	/loremipsum
Show Profile Maker – PRM	GET	/profile
Process form - PRM	POST	/profile
Show Chmod Cruncher – CHC	GET	/chmod
Process form – CHC	POST	/chmod
Show XKCD Password Generator – XKCD	ANY	/xkcd
How to video – demo of the tool to the public	GET	/{start?}/{end?} **Note:** start and end are variables to pick up a portion of the youtube video (e.g. clicking the Lorem Ipsum Generator link at the dashboard under the LOADED COMPONENTS.

5.2 View Plan

\resources\views\layout

master.blade.php –

\resources\views

home.blade.php - **dashboard**
lorem.blade.php – **lorem ipsum text generator**
profile.blade.php – **profile maker**
chmod.blade.php – **chmod cruncher**
xkcd.blade.php – **xkcd password generator**

5.3 Controllers

Requirement	Description
GeneratorController	
showGenerator(Request $request)	Shows the generator view, takes the user input, assigns to paragraph variable and sends paragraph to the view
ProfileController	

const LISTS = 30;
const MAXLIST = 30;
	The following constants set the max limit of profiles being generated. To increase this limit, additional profiles images will have to be created.

public $profile = array();	This array holds the randomly generate associative array of profiles. Instead of creating an array of profiles upfront, this array is actually created at the time the user enters the n input to generate profile. If the user input is 4, this array will be built up to 4 items. But where do all the elements come from?
autoCreateList()	Under the __construct, there's a call to a method: Digging further to the **autoCreateList()** method, it is where the array gets built. Notice that **$list_name** and **$list_email** are two items that I had to create lists for, which are defined under **\app\includes\list.php**, so the logic can pick any of the names and emails available when building the profile array.
doRandom()	All elements of the array are generated with the **doRandom()** method. When an element is created, for instance, $this->doRandom(0, $this->name, $this->track_name); The **doRandom** method will pick a name from the **$this->name** and add to the **$track_name** array. The **$track_name** array just holds a list of names to supply to the profile array later. Same applies when **doRandom**() runs to the other items, it puts the right data in the corresponding track array. So down to the code where **$array** is defined, notice that all the track values are being requested. Once the codes loops for the first time, it then pushes the associative data to the **$profile** array. So Id, name, birthday, etc... is created as the item 0 in array $profile.
salt()	Generates the salt, and it is called in **autoCreateList()**
xkcd()	Uses the P2 class, and then gets called **autoCreateList()** to provide the password.
showRandom($param)	Shows the item in the view

display()
displayJson()
displayCsv()
	Display the items in the specific format in view

Requirement	Description
ChmodController	

public $read;
public $write;
public $execute;
public $noread;
public $nowrite;
public $noexecute;
	These variables are used for displaying the permission in view

public $userPermission;
public $groupPermission;
public $otherPermission;
public $specialPermission;
	These variables take the sum of the permission

showChmod(Request $request)	It creates default values to be sent to the view. Also, used so the code can get back to in case of failed validation.
postChmod(Request $request)	The first few lines of the code are related to validation. Then **$text** and **$spl** are created. The **$spl** just splits the input 0000, into 0, 0, 0, 0 so they can be processed as unique entities.
$post	Reads all the inputs sent by the user.

setSum($request)
getSum()
	setSum($request) works with **getSum()** to perform calculations. First getSum() reads the inputs from checkbox and sums them. Then **setSum()** adds them to the **$userPermission, $groupPermission, $otherPermission, $specialPermission** variables.

setHtmltags($request)
getHtmltags()
	Works along with **getHtmltags()** method, so the allowed and denied permissions are explicitly printed in green and red in the view.

$postSum	Reads everything (all variables) and sends to the view for displaying. The controller is also send to the view so public methods can be executed when required.

mapChmod()
selectPermission()
	It performs the general logic of all the possible permissions. It is a map, so if the user checks User Reads, "ur", the entire array contains values related to read 'r', or possible octal values corresponding to read such as 4,5,6,7. Please note that the map array, it's there to simplify the displaying of all possible scenarios. Also, keep in mind that the controller is performing the logic for both input scenarios - octal input and checkbox, all in one view.

XkcdApiController	
showXkcd(Request $request)	Instantiate the class Xkcd to run the Xkcd password generator app integrated. The class itself is in app\lib\Xkcd.php

-

6 Revision History

Version-No.:	Change Reason
001	First revision has been created.
002	Update documentation, include the profile previous/next buttons to allow user to traverse profile views, update readme.