Read Me

This document covers the deployment and evaluation of the Prime Tables exercise.

# How to Run

## Prerequisites

A server with the following features installed:

PHP 7.1

Composer 1.3.1 or higher

Apache 2.4 (or whichever HTTP server software you are comfortable with).

**Note**: In this guide I am using Centos 7 with Apache 2.4, and am using vhosts.conf to manage my configurations, your setup may differ slightly when it comes to file paths.

## Setup

1. Clone the git repository into the relevant directory on your web server. In my case this is **/var/www/primetables**.
2. If you are required by your web server to specify a **document root**, make sure this is pointed to the public folder inside the primetables directory (**/var/www/primetables/public**).
3. Ensure you are inside the primetables directory itself (/var/www/primetables) and type following command to run composer and install the relevant framework components: ***composer install***, then press **Enter**.
4. The next step is to set up the environment file for Laravel. Type the following command to make a copy of the template env file and use it as the base of our new one: ***cp .env.example .env*** and press **Enter**.
5. Next we need to generate a key. At the command line, type ***php artisan key:generate*** and press **Enter**. You should see a message pop up confirming an application key has been set successfully.
6. Finally, to avoid issues with permissions for things like logging, recursively set the permissions on the primetables directory to allow your web service to manage it. In my case this is apache, and is managed by using the command ***sudo chown -R apache:apache /var/www/primetables***.
7. Open your servers IP (or domain name depending on your configuration) in a browser and you should be greeted with the Prime Table Exercise page.

## Unit Testing

I have designed some basic unit tests using PHPUnit. In order to run these, from the primetables directory run the command **vendor/bin/phpunit** and hit **Enter**.

## Troubleshooting

* **Something went wrong error text on a white screen** – Something in Laravel has not been setup as it should have. Ensure you have a .env file copied from the template, have generated a key in it, and have run an update from composer. If all of these have been done correctly, review **/var/www/primetables/storage/logs/laravel.log** for more information on the issue.
* **/Storage/Logs could not be accessed** – This is usually a permissions issue. Check that your web service can read/write to the storage/logs directory in Laravel. In a worst case scenario you can just **chmod -r 777** the entire **/var/www/primetables** directory, though this is not advised in a live environment.

# Evaluation

## Positive Items

I am pleased the speed and memory usage of the program, especially as I encountered an issue early on that wouldn't let me go above the first 100 primes without hitting PHP’s 120 second timeout, whereas after refactoring some of the code I was able to get 1000 primes out without much issue, and could even pull out 2000 before the time limit started coming back again.

Another area I am happy with is the implementation of unit testing. While I am aware of the concept of unit testing, I have never had chance to try it myself. It was a nice challenge to sink my teeth into, and I now feel I understand a bit more about the concept of it, and where it can be useful, especially in automatic builds.

Lastly, I felt that adding a touch of bootstrap into the page output was worth doing, just to help keep things neat.

## Further Improvements

I can already see a possible way to improve how many primes can be shown by using pagination and limiting each page to around 1000 primes, though this would make navigation to a specific area of the page difficult, and would require two different pagination systems, one for vertical direction, and one for horizontal.

I am also not sure that I cover all ground with the unit testing component, so more research into that and into what other tests would normally be expected on an application like this would prove useful.

Lastly, in a time-is-no-issue situation, I could have probably written this better in a smaller more compact language or framework (i.e. C# or Python) and gone for a console or application based visual feedback, rather than web-based. However as Unit Testing was also required, and I had no real experience of implementing that in any language, I went with the PHP/Laravel combo due to my familiarity with it from daily use, which in turn made adding the Unit Testing a bit more straight forward.