**Video Parsing Application**

Designed and developed a PHP application to separate video records retrieved from a CSV file into valid and invalid files containing the video IDs based on the following criteria supplied:

A valid video record is one which has:

* Title shorter than 30 characters.
* Over 10 likes.
* Over 200 sales.
* Price must be under 20 Euros or under 25 Canadian Dollars.

**Getting Started**

These instructions will get you a copy of the project up and running on your local machine for development and testing purposes.

**Prerequisites**

Below is the list of softwares to be installed.

1. PHP7
2. Composer
3. PHPUnit

**Installing**

Please install the following for complete setup of the dev environment.

1. Install PHP7 on the machine

Ref: <https://www.sitepoint.com/how-to-install-php-on-windows/>

1. Install Composer

Ref: <https://getcomposer.org/doc/00-intro.md>

1. Install PHPUnit

Ref: <https://phpunit.de/manual/6.5/en/installation.html>

**Running the app**

Start by running the “index.php” file from the command prompt by going to the folder containing the file.

For eg:

cd C:\PHP7\VideoParserApplication

**Note**: for this project I have placed my source code files under VideoParserApplication and the same name is used in composer.json.

Run ‘***php index.php***’

The terminal asks for user input of the file names:

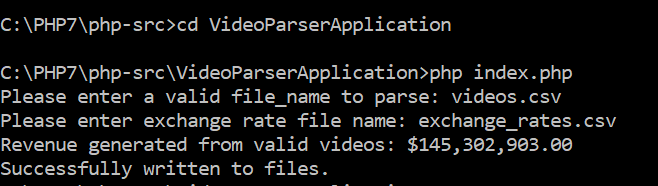
Please enter a valid file\_name to parse: ***videos.csv***

Please enter exchange rate file name: ***exchange\_rates.csv***

Make sure these input files are in the same root folder as the application.

If everything goes well, the terminal displays the total amount of money in USD made by all valid videos. Also, it will produce two csv output files containing valid video IDs and invalid video IDs if any. They are named *‘****valid.csv***’ and ‘***invalid.csv***’ and can be found under the same folder.

**Screenshot:**



**Running the tests**

To run the php unit tests go back one directory:

Run ***cd ..***

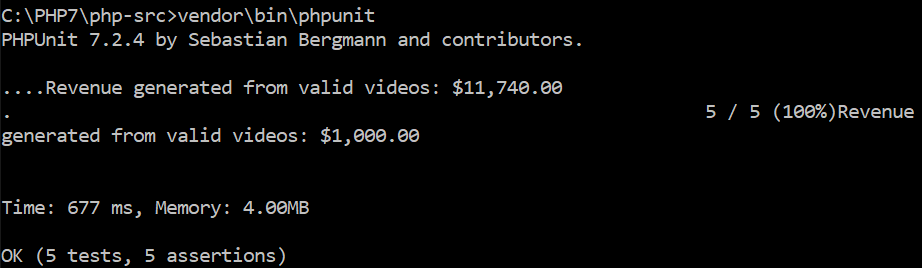
***vendor\bin\phpunit***

as in:

***C:\PHP7\php-src>vendor\bin\phpunit***

This runs the existing unit tests for the files and output the results.

**Screenshot:**



**Test Cases:**

These unit tests are:

**testPreprocessDataTest.php**

* *testIsRequiredColumnsPresent* => checks if the required/ requested columns are present in the given csv file.
* *testConvertCsvToArray* => checks if the csv is converted to the expected array.
* *testGetRequiredColumns* => retrieves the required columns if there are more columns other than the requested ones. Efficient for larger files containing huge data.

**testVideoParserTest.php**

* *testParseCsvFile* => tests if the files have been correctly parsed into valid and invalid records according to the given filters.
* *testDisplayRevenueGenerated* => checks the successful or failure of printing the total revenue made in USD from the valid video record sales.

**Built With**

* PHPStorm

**Questions:**

**1. Was the question/problem clear? Did you feel like something was missing or not explained correctly?**

Yes. The question was absolutely clear. The csv files were correctly formatted.

**2. What makes your solution awesome?**

My solution has been properly separated into two class file and one index file, which clearly modularizes the code and makes it easier for other developers to read and understand the application and its workflow and make future improvements/enhancements if need be. I have handled all necessary empty inputs, correct type conversions for important calculations, validity of file names, its content, correct paths in the system as well as its extension, properly display the error to the user if something unexpected happens, also handled exceptions. This makes it easy to debug the application as well as identify any bugs in the system early. I have also separated out the filter conditions given, as later there can be requirements to further add valid and invalid file parsing rules. By keeping them as a separate class by itself, I was able to achieve maintainability and extensibility of the code.

PHP coding standards were followed as well as proper doc blocks are written to tell other readers of the code their functionality, parameters and return values expected. Also, I have added unit tests as I felt it was really important to properly test each class and its methods. I must admit that the setup of the phpunit took me some time to configure correctly, but it was definitely worth it. In the end, a correct output is what makes everyone happy!

**3. Did you have to make any trade-offs in your design? If so, what?**

I wanted to keep a separate method for calculating the money made from all the valid videos but since I have to keep a separate file for maintaining the ‘total\_purchases’ data besides the valid video ids data, I included it under the same parseCsvFiles method in VideoParser class. Due to time constraints, I plan to refactor this later.

**4.Is there anything else you want to share about your solution or the problem?**

I enjoyed working on this project where I got a chance to work on a solution that is scalable and efficient. By taking into account that files and inputs can be much larger than the sample data, I was able to properly handle duplicates and empty data which allowed faster processing of data and hence faster program execution. There was no way to validate correct video IDs and used it as it is. If time permitted, I would have code sniffed. The problem really simulates a real world problem and was really exciting to work on. Any suggestions to make it better is welcome.

**Authors**

* **DEBIKA DUTT**

**Acknowledgments**

* PHPDocs
* <http://phpunit.readthedocs.io/en/7.1/writing-tests-for-phpunit.html#testing-output>
* <https://www.php-fig.org/psr/psr-2/>