# Technical Design Document

Prepared for **Square1**
by Germán González Rodríguez

## SUMMARY

Along this document I'll try to explain the technical decisions made for developing the technical assignment.

## INSTALLATION

### Requirements

As part of Laravel 5.6 requirements:

- PHP >= 7.1.3
- OpenSSL PHP Extension
- PDO PHP Extension
- Mbstring PHP Extension
- Tokenizer PHP Extension
- XML PHP Extension
- Ctype PHP Extension
- JSON PHP Extension

For the proposed solution it is also required:

- Mysql
- Composer

As a note, all of these requirements are satisfied by Laravel Homestead, but I didn't use it, instead used my auto deploy docker image that mimics Homestead ( https://github.com/gergonzalez/ubuntu-server ).

## Deployment

For deploying the project there are 2 options:

**Using attached homestead Docker image**. To use it:
1. Install Docker ( https://www.docker.com/community-edition )
2. In a terminal window go to the project /**dockerserver** folder and execute **docker-compose up**
3. Once docker installation and app auto deployment finish (Several minutes depending on internet connection), in a browser window go to *http://localhost:8088* and start testing.

**Using unix based local environment.** In a terminal window:
1. Execute **composer install** in the provided source root directory to install the dependencies.
2. Update **/storage** folder permits **sudo chmod -R 777 storage/**
3. Create a new database in Mysql.
4. Duplicate .env.example file as .env and update the database variables.
5. Set Laravel application key with **php artisan key:generate**
6. Migrate the database. Execute **php artisan migrate**.
7. Seed the database using **php artisan db:seed** command.

## Testing

For a quick evaluation, a beta environment was also set up at:
http://square1.gergonzalez.com

# DEVELOPEMENT

## Coding Style

Standards defined in the PSR-0, PSR-1, PSR-2 and PSR-4 documents were followed.

## Php Framework

Laravel 5.6 was used. The main reasons were that is a skill requirement for the job offer but also due to I have been using it for more than 3 years now. Laravel is very well designed, easy to use and has a big community behind.

## App Directory Structure

The structure used is the Laravel's default. But adding a helpers folder in app directory to store our helpers, in this case **CrawlerHelper**.

## Database

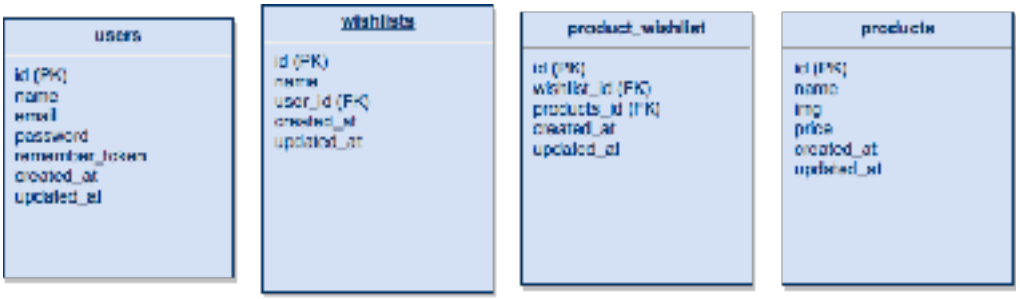
Mysql was used. Due to requirements, 4 tables were added using migrations:

**users.** Stores users data.
**whislists.** Stores wishlists data.
**products.** Stores products data.
**product_wishlist.** Intermediate table used in the N:M relationship to store the relation between wishlists and products.

Two more tables were created by Laravel:
**migrations**. Tracks migrations history.
**password_resets**. Used by Laravel auth to handle password resets.

Also 2 relationships were identified:
A one to one between users and wishlists.
A many to many between wishlists and products.

The simplified EER diagram of the implemented db structure:



## Controllers

Only 3 controllers were implemented. These controllers are in charge of handling the logic of related requests in a single class.
The **HomeController** manages the requests to the home/landing page.
The **WishlistController** manages the requests to the wishlist related pages.
The **ProductController** manages the requests to the quick web hook implemented to sync database values.
**Auth Controllers** were automatically added by Laravel but some were updated.

## Models

Only 3 models needed too, **User, Product and Wishlist**. Each table has one Model and it allows to easily interact with that table.
They implement methods to handle the relationships between the tables and some helpers.

## Views

The views were separated into components to make them easy to update and reuse. A Vue.js component was implemented for the delete button in the product list.

## Crawler

For the crawler, used the PHP package Goutte ( https://github.com/FriendsOfPHP/Goutte ). I tried to don't make a db call for each product, for that reason I first extracted all the data from the origin website and then added at once to the db, even with the problems that this approach could have, like memory limit crashes or not been able to seed the db in case of error or source down for maintenance in the middle of the scrape.

For the data sync I ran out of time. But my approach was, depending of how much access we would have at the other platform, in my opinion, the best solution would be to add a webhook in our system that would be securely called at the source website every time the source website update any product or create a new one.

If the webhook approach wasn't an option, a cron job using Laravel Task would do the trick. But try to don't overload the source website is not an easy task. I'd recommend to do some research about daily traffic times, origin website update routine and frequency, etc. And then decide the periodicity of the task.

As a note, if you are the owner of the scraped website, the best approach would be give access the source db directly.

## Auth
The auth uses out of the box Laravel solution.

## Exceptions
The default Laravel message was updated for a custom, quick, one.

## Routes

| METHOD | URL | Description |
|---|---|---|
| GET | / | Home Screen. Shows the list of products |
| GET | /wishlists | Shows logged user wishlist. |
| GET | /wishlists/{wishlist} | Shows wishlist. Used for sharing. |
| POST | wishlists/{product} | Stores product in the wishlist. |
| POST | wishlists/delete/{product} | Deletes product from the wishlist. |
| GET | products/sync | WIP. Web hook to update products. |

# FINAL CONSIDERATIONS
During the ~14 hours I dedicated to implement, document and test the project, I tried to use the most standard, readable and efficient possible solutions.

I hope this test gives you a glimpse about what my skills and strengths are.

Please, don’t hesitate to contact me if you need any further information.

Thanks so much for giving me the opportunity,
Germán González Rodríguez
April 23th, 2018