

SOEN 341

Report on Term Project A Classifieds Website

Professor : Dr. Emad Shihab

Date : December 4, 2017

Authors:

Brandon Balala (40029672)

Gneykou Kengne Yvann Monny (40015234)

Han Gao (40020549)

Kevin Camellini (26771009)

Laurie Yung (40008692)

Liyuan Zhang (27201044)

Neqqash Hussin (40016921)

Rami Yahia (26994628)

Important Links:

Github Repo: <https://github.com/classifiedz/classifiedz.github.io>

Website: <http://classifiedz.xyz>

Table of Contents

Table of Contents	1
Table of Figures	1
Introduction	2
Architecture & Design	3
Technologies & Implementation	4
Process	6
Testing	9
Metrics	10
Cyclomatic Complexity (Code)	11
Comment to code ratio(Code)	11
Programmer productivity: LOC/month (Process)	12
Lessons Learned	13

Table of Figures

Figure 1 - Architecture diagram Laravel's MVC Used in The Project	4
Figure 2 - User Story and Task Breakdown	7
Figure 3 - Example of Burndown Chart (made with Waffle.io)	7
Figure 4 - Example of an Acceptance Test	10
Figure 5 - Code Comment Ratio	12
Figure 6 - Code BreakDown	12

Introduction

For our term project we were tasked with creating a classifieds website where users can buy and sell goods and services, similar to kijiji. We created this software product for a customer, whose role was played by our teaching assistant, Mehran Hassani. We followed a Scrum process which is an Agile development method. The process of developing this product involved a total of five sprints, each lasting two weeks, and weekly meetings with our customer. We started with a product that had limited functionality on the first sprint and ended with the final product on the last sprint, which is a fully functional classifieds website. Throughout this report we will outline the architecture, technologies, process, metrics, and lastly the lessons we learned along the way of creating classifiedz.xyz.

Architecture & Design

Architecture and designs define the structure of the project as well as the relations between all of its items and components. The architecture must also reflect the nature of the project and must be adapted to our team's needs and strengths.

Classifiedz was developed using Laravel for the framework and in turn a Model View Controller architecture (MVC). MVC's architecture separates the data manipulation into three components. The **model** represents the data that is stored within a project. All the information from users to products are represented within the model and stored in a database. Manipulation of the data happens within the model. The **controller** on the other hand links the model to the appropriate **view**. The view is what the user gets to see rendered on their screens in HTML. Data accessed through the model is then placed in the appropriate places through the view. Additionally, HTTP requests are handled by **routes**.

Controllers are located in `/app/Http/Controllers/`. All major functions of this website has a controller associated with it, such as WishlistController, UserReviewController and the SearchController. Controller function can be accessed from the view and be placed wherever it is necessary using HTML & CSS or Laravel's templating engine called blade using the `{{ variable }}` syntax, which is the equivalent of `<?php echo " " ?>`.

Similarly, models are listed in the `/app` folder. The specific models extend Model and we then list all of its attributes.

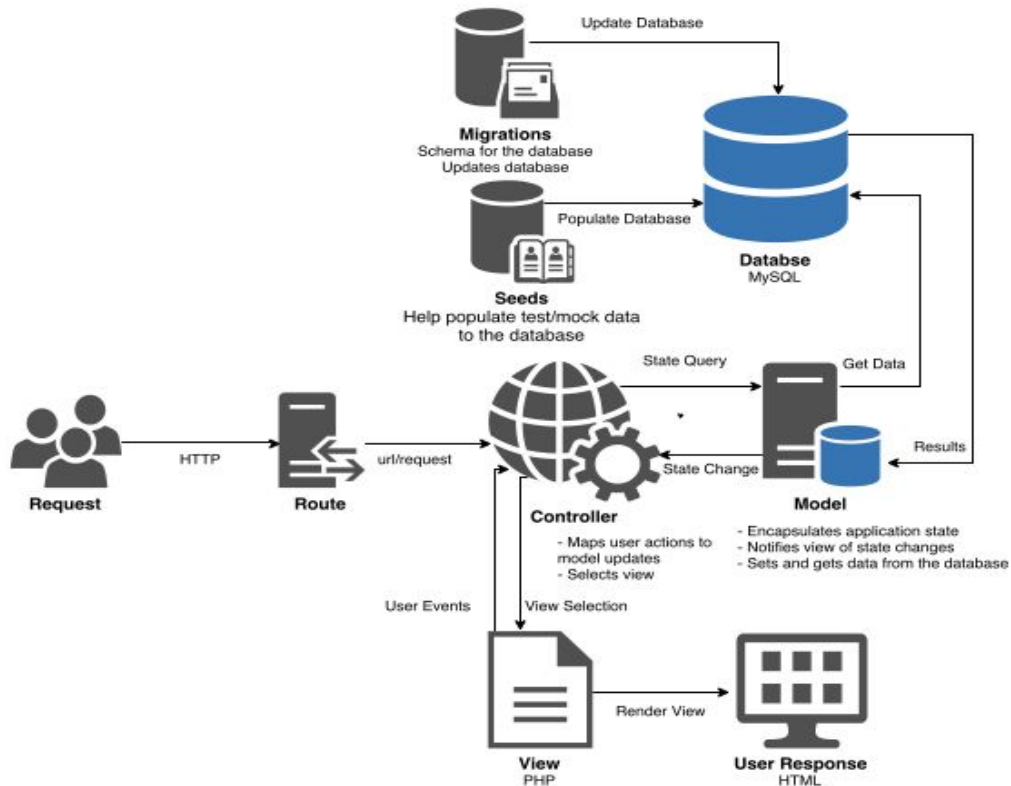


Figure 1 - Architecture Diagram of Laravel's MVC Used

For the purpose of the term project, the MVC architecture allowed us to better split up tasks between team members. Front-end developers could focus and spend more time on the views, while the back-end developers focus on the models, routes, and controllers.

Technologies & Implementation

The following section briefly explains the technologies we used, how they were implemented, and the reason for choosing them.

For our framework, we chose **Laravel** because our backend (Brandon) was already familiar with it, and made an excellent teacher for the rest of the team. In turn we used PHP along

with HTML and CSS because Laravel is built with PHP. The database was implemented by **MySQL** as Laravel has built in functionality with it. **Composer** is used for the package manager, because it is built for PHP (similar to BREW and NPM). **Blade** is the template engine for Laravel which renders views into HTML. **Bootstrap** was used for our front end CSS because Kevin was already familiar with it. **Digital Ocean** is our web hosting server, and we chose it because we can get it free with the Github student package. We used **Laragon** (a LAMP local environment) to host the project locally and **XAMPP** for those on OSX. We used **Algolia** for the searching engine. It was convenient to use Algolia because Laravel has support for it.

Github served as our git version control repository hosting service. Also, It is used for documenting the wiki, discussions, scrum meetings, backlogs, sprints, User Stories, Task breakdown, Milestones etc. **Travis CI** was used together with **PHP Unit** for continuous integration. **Waffle.io** is our project management software (similar to Trello), which is automatically updated from Github as issues get opened, referenced in commits, or closed. For instance, if we closed an issue on Github, it also get closed on Waffle. We also used Waffle.io for our burndown chart. **PHP PSR-1** is the standard we used to enforce coding style. For example, StudlyCaps for class names, snake_case for variable names, and bananaCase for method names. This coding style strongly improves our code understandability and reusability.

Process

The website started off with a very general layout and with little to no functions at all. The team then iteratively enhanced the website and incrementally added functions that the customer required in each sprint.

Each sprint lasted two weeks long over a period of five sprints. Within the sprints, the team had bi-weekly scrum meetings, each lasting 10-15 minutes where each team member commented on their progress, issues, and plans for the near future. The team also met with the customer every Friday face-to-face for 15 minutes. During this time, the customer provided feedback on what was required for the website and whether the progress was at a good pace to meet the deadline. In addition to the bi-weekly scrum meetings, after every meeting with the customer, the team met for 45 minutes to discuss the feedback given by the customer and plans for next sprint in terms of product backlog, user stories, tasks, features, and bugs.

The team decided what user story to work on next based on the priority, risk, and story points, with high priority and low risk taking precedence. User stories were also chosen with velocity taken into consideration of how many points/sprint were being accomplished. Figure 3 below shows the, poor performance, burndown chart which gives an idea of the team's velocity. Figure 2 shows an example of the way user stories were implemented. Story points were assigned according to their relative difficulty and ideal time using a

bucket system using the Fibonacci sequence (1,2,3,5,8,13,21). The team got better at estimating ideal time as the project progressed.

User & Seller Reviews

Risk	Priority	Story Points
Medium	High	8

As a user(buyer) I would like to be able to see a sellers reviews. As a user(seller) I would like to be able to see the a possible buyers reviews before further arrangements are made to sell an item. Reviews should be on a scale of 1-5 and should display as an average on the Store Page [#161](#).

- ✓ 3.1 Create new row in the database, under the user table for reviews [#123](#) (2.5 Hours)
- ✓ 3.2 Create HTML and CSS to display the 5 stars on a user profile or seller page [#123](#) (2.5 Hours)
- ✓ 3.3 Get the data from the database and display it [#123](#) (1 Hours)

Total: 6 Hours

Figure 2 - User Story and Task Breakdown

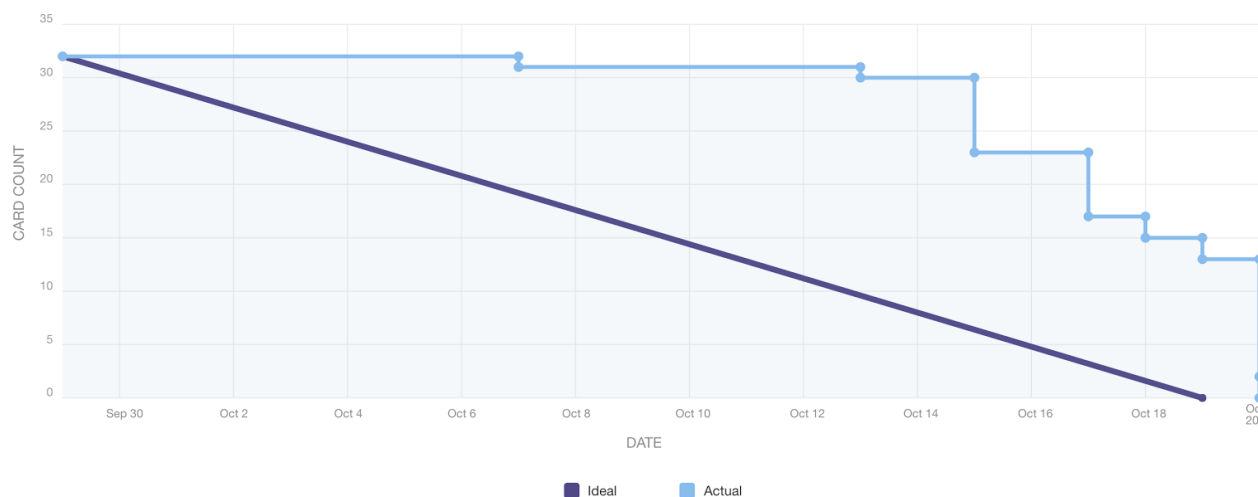


Figure 3 - Example of Burndown Chart (made with Waffle.io)

The Scrum process we underwent can be broken down into three main phases: Initial Phase, Sprint Cycles, and Closure Phase.

Initial phase (Sprint 1): Objective of project was established

- Technologies to use (frameworks, language, database)

- Design of website
- Functions to be added
 - o Unique functions (user ratings, comments under ad's)
 - o Customer requirement functions

Sprint Cycles (*Sprint 2 to 4*): User stories, tasks broken down and assigned to developers

- Meetings (within the team and with customer)
- Velocity
 - o Team tracked progress of work being done (shown by burndown charts)
- For each implementation, validations were performed
 - o Unit testing, acceptance testing, Integration testing
 - o Refactoring

Closure Phase (*Sprint 5*):

- Wrap up the project
 - o Last minute customer design changes
 - o Refactoring
 - o Metrics
 - o Final validation

Although the team used a Scrum process, some practices from Extreme Programming were also useful in the project. Continuous integration, each functionality was tested before it was implemented. Collective ownership, everyone had touched every aspect of the project and code. Lastly the code was constantly refactored to maintain the understandability and efficiency of the product.

Testing

Testing is an essential part of software development. It is used to ensure that implementation lines up correctly with requirements and can be used to reveal the presence of defects. In our case, we performed two types of automated testing: Unit tests and GUI tests. Such tests are ran locally. We are also using Travis CI for continuous integration, which runs all automated tests on every commit and pull requests to ensure that the build is fully functional.

In Laravel, we never deal with raw mySQL statements and queries. Everything is managed through what Laravel calls it: Eloquent ORM. This feature is what maps our models to our database tables. We made simple unit tests, using PHPUnit, to ensure that changes to our models reflects on our actual database. For most models we have, we tested their CRUD methods: create, read, update and delete.

As for UI testing, it ensures the proper functionality of the graphical user interface of our project. Such tests involve physical interaction with the website. This was done using Laravel Dusk. In the background, before the tests starts, it opens an instance of a Google Chrome browser. Dusk can be seen as a bot, as it simulates the actions of a human being. You give it a destination or URL to visit, which contains the feature you can to test. Then you specify actions for it to perform such as: typing data in input field, choosing an option in a dropdown, checking a checkbox, clicking or pressing anywhere on the screen. Lastly, just like any other testing framework, you would perform assertions. We performed UI tests on

new features implemented during Sprint 5 (rating system, editing ad) and on all our authentication features (register, login, logout).

Lastly, we performed acceptance tests to ensure that what was implemented lines up with the user story scenarios tied to it. Figure 4 below is an example of how we formatted our acceptance tests. We give user stories and tasks a description, pre-conditions, scenarios and expected results. We would then manually perform such scenarios and grade it a pass or fail based on whether actual results match the expected result. Only once everything passes do we merge new features into our production build.



User Story 17: Prompt Before Deleting Ads				Tested On: 16/11/2017
Description: Users will be asked to confirm their choice of deleting their ad				
Pre-Condition: Must be logged in with at least one ad posted.				
Test Case ID	Scenario	Test Data	Expected Result	Pass/Fail
17.1_195	Go to "Your Store" and press the delete ad button - confirm		Ad should be successfully deleted and no longer appear anywhere on the site	
17.2_195	Go to "Your Store" and press the delete ad button - cancel		Ad should not be deleted and should still appear anywhere on the site	

Figure 4: Example of an Acceptance Test

Metrics

We used **Understand** to get the metrics from our source code. We choose to analyze:

- Cyclomatic complexity
- Comment to code ratio
- Programmer productivity

Cyclomatic Complexity (Code)

Cyclomatic complexity can be thought of as being similar to big O complexity. The cyclomatic complexity report gives us the **max cyclomatic (52)**, **average cyclomatic(2)** and **max nesting (10)**. The reason that the average cyclomatic and max nesting is low is that most of the functions like the UI are simple, and don't do much "heavy lifting". However, the max cyclomatic can be much higher than the average as some of the backend function are much more complex than front end ones. Additionally, the five most complex files are all Controller files.

Comment to code ratio(Code)

The comment to code ratio indicates the amount of source code that is dedicated to comments. As we can see from the following report, the comment to code ratio is **0.26**, which means that roughly every three lines of code will have a comment to go with it. That ratio indicates that our development process follows the code convention, that is, the code is not so overburdened with code that it is hard to understand, nor too little either.

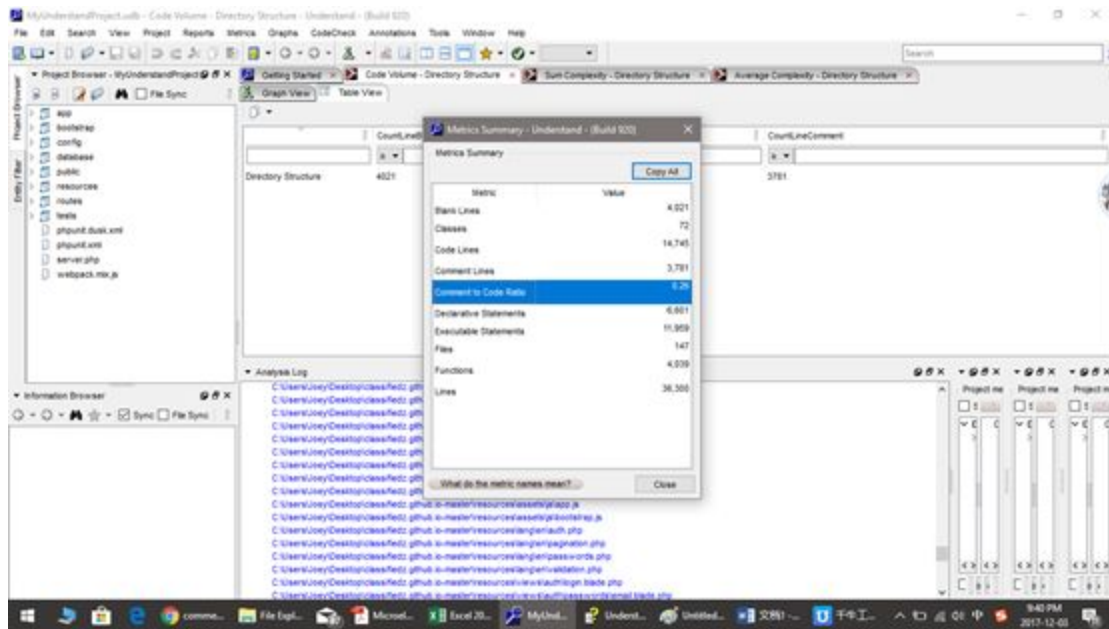


Figure 5 - Code Comment Ratio

Programmer productivity: LOC/month (Process)

From the report we saw that there are a total of 14745 LOC and our work months are 3.5 months. Thus our programmer productivity is 4212.85 LOC per month. Compared to professionals, this is very low, but for the scope of the class, it is an efficient pace.

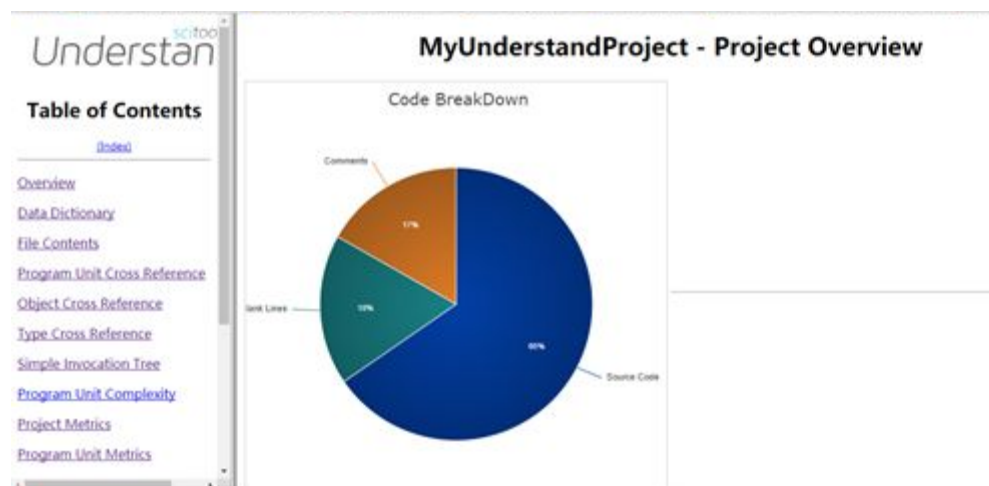


Figure 6 - Code BreakDown

Lessons Learned

Over the course of the project, many things were learned and improved upon when given the opportunity. Different aspects of the project brought along with it, different obstacles. These different aspects include the overall project management, communication between team members, and working with the customer.

During the first sprint, we focused all of our effort on setting up a working product. While this does comply with agile methods, it tunnel-visioned our team and prevented us from thinking past this stage. With our focus locked on setting up the foundation, we failed to come up with a concrete plan of where we wanted to take our classifieds website. With the realization of this oversight, we were able to come together to brainstorm and come up with ideas of what features we wanted to implement.

The previous mistake made during the first sprint was brought to our attention by the TA, who took on the role of a customer that gave us free rein on the features to implement but gave us direction on the process. Over the following weeks, we made similar missteps and the majority of the times it was due to the lack of communication between us and the customer. While meetings between the two parties were held every week, we didn't ask for enough clarification or we misunderstood requirements. These highlighted the necessity to iron out details and requirements placed by the customer through proper communication.

In addition to communication with the customer, strong communication between the team was equally as important. As not everyone was familiar with the framework being used, help was required from some of the more experienced team members. There were occasions when tasks required more support than others and when communication was lacking, it often lead to a scramble for time.

Overall, there is always room for improvement. Our experiences during this project exemplified the importance of properly planning out a project and looking towards the future of it, refining the requirements of the customer and clearing up any confusions, and communication between teammates whether it's to request help for a task or expressing an opinion on current or future features.

Contributions

Liyuan Zhang	Profile page, edit profile page, wish list page, edit ad page, your store page
Rami Yahia	User rating, 404 page, legal pages.
Kevin Camellini	Team leader, project and repo management, masonry layout, search bar, ad cards layout, general front end review/maintenance.
Brandon Balala	All backend related work, ad chat, browse ads, server deployment/maintenance, tests
Laurie Yung	Individual ad page, post ad form, ad cards layout, project and repo management, general front end maintenance.
Neqqash Hussin	Scrum Master, login/register page.
Gneykou Kengne Yvann Monny	Delete ad, navbar dropdown, classifiedz logo
Han Gao	Profile page, edit ad page.