


OBS Post-Mortem

LCJJ Post Mortem for Online Booking System (OBS)



Mop Hairdressing

Booking System: Login

Username

Password

Sign in

Register

LCJJ Development Team

LCJJ

Software engineering: Process and tools

Semester 1 2017

- Johnny Huynh (s3604367)
- Craig Barry (s3601725)
- Lachlan Porter (s3537901)
- Jarry Chen (s3600396)

1. Project

1.1 Project Description

Project Name: Online booking system (OBS)

Client: Homy (Tutor in representation of the client)

Project Leader/scrumb master: Johnny Huynh

Start Date: ???????

Completion Date: 21st of May, 2017

1.2 - Project Overview

1.2.1 - Introduction to project scope

Online Booking System (OBS) is a project designed to help businesses move to a digital, faster and more efficient method of storing their business booking information through the web. It enables them to access their system anywhere anytime securely through the web.

1.2.2 - Technical Details

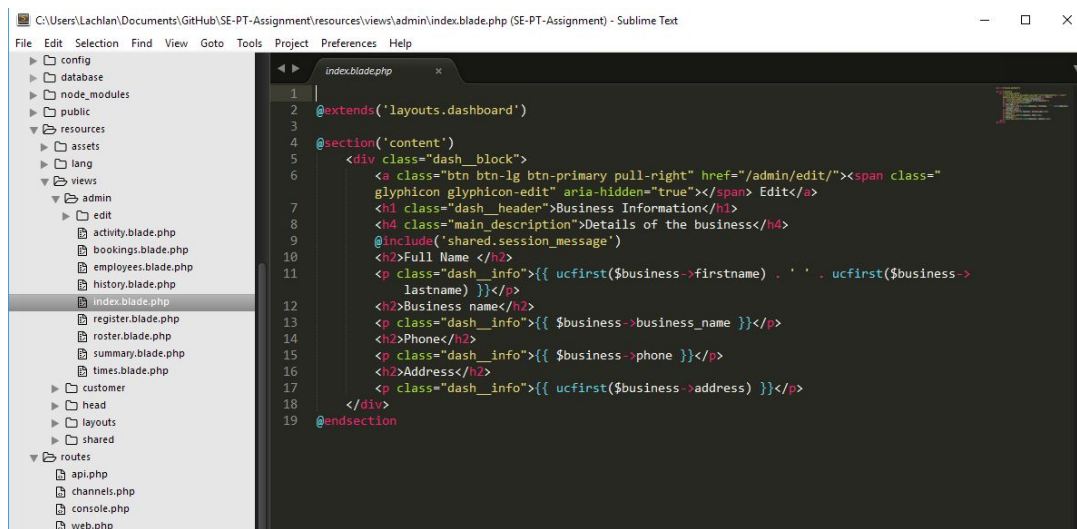
We decided to use PHP and HTML to build the system early on since it made sense to build a system using web technologies. Then we decided to use the PHP framework “laravel” since it provided structure and useful in built features that provided a useful base to start the project with.

Some of these features included PHP unit and Dusk testing which allowed us to do extensive testing on the system to make it more robust by using integration tests with PHP Unit and User acceptance testing using Dusk testing. Furthermore we used a PHP tool called ‘factory-muffin’ which dynamically creates random data in a given format (EG an address or first/last names), this enabled us to perform even more robust testing as we could vary the data for some tests to ensure that the UI was operating as intended. This tool can be found here: <https://github.com/thephpleague/factory-muffin>

The structure that laravel provides encourages good design patterns right from the start of the project. For example MVC (model - view - controller) architecture was used right

from the beginning. Each type of user has their own controller, for example customer has the customerController and the business owner has busniessOwnerController. Then the views use these controllers to dynamically display user information from the database.

Included in laravel is a templating engine called 'Blade' which enabled us to build several different pages without re-using old code. For example each page on the dashboard contains the same layout, only the content section differs page to page. Thus we use blade to set up the layout and pass the content for each page to the template to render each page. An example of this can be seen in the image below (figure 1.1)



```
1 |
2 | @extends('layouts.dashboard')
3 |
4 | @section('content')
5 | <div class="dash_block">
6 |     <a class="btn btn-lg btn-primary pull-right" href="/admin/edit/"><span class="
7 |         glyphicon glyphicon-edit" aria-hidden="true"></span> Edit</a>
8 |     <h1 class="dash_header">Business Information</h1>
9 |     <h4 class="main_description">Details of the business</h4>
10 |    @include('shared.session_message')
11 |    <h2>Full Name </h2>
12 |    <p class="dash_info">{{ ucfirst($business->firstname) . ' ' . ucfirst($business->
13 |        lastname) }}</p>
14 |    <h2>Business name</h2>
15 |    <p class="dash_info">{{ $business->business_name }}</p>
16 |    <h2>Phone</h2>
17 |    <p class="dash_info">{{ $business->phone }}</p>
18 |    <h2>Address</h2>
19 |    <p class="dash_info">{{ ucfirst($business->address) }}</p>
20 | </div>
21 | @endsection
```

Figure 1.1 - use of blade to generate simple views

We also decided to deploy the project to a live domain to more accurately represent the final product that the end user would have. Johnny bought the domain <http://johmodding.com> and hired hosting for it. This website is automatically updated to the main github branch. So whenever a feature is finished we do a pull request from the dev branch onto the main then the the website is automatically updated through the github webhook.

This was part of our advanced workflow method, first we pull any new changes then work locally using 'php artisan serve' to run it locally. Then the changes are pushed to the dev branch so that other members can work off the changes. Finally once a feature is fully complete then it is merged onto the main branch via pull request which then updates the live site.

2. Performance

2.1 - Key Accomplishments

2.1.1 - Learning Laravel Framework

Learning a completely new framework was a great milestone for this group as we were working on a short schedule and no one was really familiar with PHP. So undertaking a PHP project was to meant to push us out of our comfort zones. Despite learning the new framework as we worked we managed to successfully develop the project with minor delays in the sprints.

2.1.2 - Adapting to ever changing requirements

A key reason for the project to be completed in agile is so the project can be flexible with its requirements and adapt when needed, in order to reflect how most projects are in the real world. Homy pushed really hard for this project to mimic the real world by playing a client who is constantly changing his mind about what he would like out of the software and giving design and wireframe requirements to meet what they had in mind. This kept the team on their toes and gave us great insight into how some designs can be more efficient than others and how to refactor code to accommodate the changes.

2.1.3 - Impressive testing PHP unit and Dusk tests

We acquired a lot of knowledge in developing test cases and using them as proof that a feature works in a program. It allowed us to refactor and make small changes without breaking the system.

The latest build of Laravel provided us with acceptance testing through live browser tests. This was similar to integration tests, though using jQuery and a browser to input data and submit like an end-user.

2.2 - Key Problem Areas

2.2.1 - Difficulty setting up dev environment (PHP, composer, laravel, node, etc)

The team has a great deal of problems with setting up all the new software and using the new software for the laravel framework. Throughout the project group members have had issues with their local repos, composer and github. Even after setting up the dev environment the team still had to learn how to use all the new technologies since for many it was their first time partaking in a large team.

2.2.2 - Finding the time to learn new framework while working on project at the same time

The team had struggled with the new framework as it was quite different from normal PHP we learnt in web programming. With limited resources online it was often hard to debug issues that we came across and even harder to learn the new framework on a tight schedule. Often requiring the team to refactor the code as we learnt more about the framework and how to structure it to work proficiently

2.2.3 - Different in team skill levels

Due to the team having varying experiences with development and web applications. It was a struggle for some team members to learn the new framework with weaker understanding on PHP overall. Luckily the team also had very seasoned members in web development that could help and walk the less experienced members through the theories and structuring of the project

2.2.4 - Live server vs Dev environment issues (think the demo GUI customisation issues)

Johnny was in charge of refactoring the “Upload Logo” feature for the Business Owner to customize the GUI in this sense. The feature was rushed and not tested, which then lead to the live server not working with the file uploads.

Craig managed to fix the issue before the demo on a local build of project. Johnny then resolved the issue on the live server since there was an instruction that he missed which didn't show images on the site properly.

2.2.5 - Learn to work as a team on a larger project/ Work allocation

Due to the varying skill levels of the team, it was often difficult to allocate work fairly as working under time constraint didn't give the weaker members much time to learn how to use the new framework. With the help of scrum meetings and an open communication line between members, we were able to help each other out and come to mutual agreements with work allocation. Another issue was with the use of GitHub as some pushes would override the previous pushes and delete code.

2.2.6 - Attempting TDD but having issues with laravel unit testing

Over the initial time of developing the software for the group project, we attempted to use Test-Driven development not knowing how to use test cases properly in the first place.

We felt more at ease going at our own pace with developing features and then creating test cases after the feature was created. We rapidly grew an understanding on how testing works on the project, but we had limited time to change routine. So persisted

3. Key lessons Learned

3.1 - Lessons Learned

3.1.1 - Familiarity vs New Frameworks and technologies

Other than johnny, most of the team had very little PHP experience outside of university core subjects. None of the team had worked with laravel before so we were not familiar with the working environment. Thus not only did we have to complete the project in a timely manner, but we also had to learn a whole new framework to build it. Had we have chosen to do a Java program from the beginning, we would have had a working solution much sooner than we did with PHP and laravel.

3.1.2 - Team experience and skills

There was a large difference in technical skills evident throughout the project. Johnny clearly had the best technical knowledge of web systems. The rest of the team was focused on keeping up and learning how the different systems within the code interacted. Often this was very time consuming and time was wasted trying to understand the system before continuing its construction.

For any newer projects it's clear that an assessment of team experience is critical to the project's success as workload and support can be decided before development begins

3.1.3 - Time management

Later on in the project the main constraint was the time each team member had to work on the system. The workload from other subjects was clearly taking its toll on the project, the team was rushing to get each week's requirements done. Homy mentioned during the project demonstration that Part B and C 'feels a little rushed' which was true.

For future projects, better time management is a skill that will be needed to prevent the later part of the project being rushed to meet the project deadline.

3.2 - Post Project Tasks and future considerations

3.2.1- Live site maintenance

The live website must be maintained and backed up regularly to ensure that the customer always has access to their booking system and data. There needs to be very little down time so that the customer and businesses have access to their system.

3.2.2- Bug fixing and code maintenance

Bugs are an inevitable part of new software, so as bugs are reported they should be fixed so that the system operates as normal. A smaller developer team can be created to handle bugs as this isn't as difficult as creating brand new software.

3.2.3- Business Scalability

As businesses grow using the system they will use more and more data thus the hosting service must be upgraded as the business grows to meet the growing demand created by new customers.