Individual Report

Evan Smith – 113300626 – Project Eve

1. Overview

1.1 Kanban Sensei

1.1.1 Tools

*Taiga.io - **https://taiga.io/***

This was the software we used to help us allocate tasks. It is an online Kanban board, giving us a centralized way of seeing all tasks, both complete and incomplete. Any member is able to create a task and any member can assign any other team member to a task. This task can be in one of many phases, namely; new, ready, in progress, ready for test, done and archived.

1.1.2 Methods

Weekly Standup

Every Monday we had a 15 minute standup meeting. During this time every team member was encouraged to stand up and describe the tasks they tackled in the past week, along with any issues they were having with the project. At the end of the meeting, tasks were distributed or reassigned if necessary. This approach was invaluable due to the fact that it kept every member of the team up to date about the progress of the project. It also allowed any issues in development to be flagged and dealt with early.

My Role as Kanban Sensei

I didn't have any one specific role within the group as I had the most knowledge about the framework we were dealing with. I took on a generalist approach to help others with tasks and complete what was necessary to reach our milestone targets. I gave training to those new to the framework and gave advice/standards for those seeking it. I made sure there was no hierarchy within the group – being kanban sensei didn't mean I was in charge. My role at the standup meetings was to give general direction to the group, discuss problems and help assign/reassign tasks based on how people were getting on.

1.1.3 Obstacles/Surprises

Scheduling and Group Coding

Scheduling was pretty hard to do with everyone's lives/work/academic stuff to plan around. Although we never missed a standup meeting, we regularly missed group coding sessions or other meetings during the week. This meant we all spent a lot less time coding around each other and more time trying to figure problems out on our own which wasn't ideal for those less familiar with how Laravel works.

Commitment and responsibilities

The majority of the team was productive and self-motivated but one member at one time or another was falling behind schedule or not getting tasks finished in a reasonable amount of time.

1.1.4 Workarounds/Alternatives/Solutions

Scheduling and Group Coding

To get around the problems of working from home, we were constantly in communication via Facebook Messenger and Github Issues about what was going wrong, what we were having difficulty with and what we were hoping to implement. Mostly, myself and Colm Cahalane answered queries alternately so that there was always someone available to help at any time someone needed it.

Commitment and responsibilities

One person in particular was regularly falling behind so I dedicated time to sitting with them in the labs and going through code as well as working alongside them. We were often working on separate areas of the application but working together meant that he was focused on the work (and not procrastinating) while it also let him know that I was open to questions about code/functionality. I probably spent a lot more time sitting down with others to overcome this problem than my other team members would have spent together.

1.2 Server/Development Environment Administrator

1.1.1 Tools

Microsoft Azure

To actually show off our application, we needed to run it off a server. We chose to provision a virtual machine from the Microsoft Azure platform as it was really easy to do and we got to choose it's location, domain name and server specifications. It was really easy to deploy and log in. The added security of pre-prepared ubuntu images with security features built in (e.g. disabled root login) and the open-as-needed ports from Azure meant our server was pretty secure just out of the box.

Ubuntu and Linux/Apache/MySQL/PHP

I used a pretty standard LAMP stack to setup our server. Using Ubuntu was a really choice on our part

as there was a lot of support around composer (PHP), gulp (NodeJS) and Apache configs. Not to mention the fact that a lot of the software we wanted to use came in neat packages we could just install and they'd run out of the box.

Fail2Ban

Although not a requirement of our application, I installed Fail2Ban which monitored traffic to our server and their IP addresses. Suspicious traffic, login attempts and generally non-trustworthy behaviour received an automatic ban which meant that we could run without being afraid of someone DDoSing our server on the day of demonstration and ruining our project.

Vagrant

We all had to develop separately and obviously the hardest part of that is unifying the development environment. If someone installs PHP with different settings, that can lead to problems in production when all the software is joined together. To solve this, we all had a singular vagrant box that provided all of the underlying infrastructure for a server right on everyone's laptop. It meant that people had a separate dev environment but one that was identical to what we were running production.

1.1.2 Obstacles/Surprises

Too low spec

On the day of the demonstration, we realised that the server we'd provisioned (0.25 cores, 2GB RAM) was incredibly underpowered to run quickly. Each request spiked to almost the entire core which meant that as we were demoing to several people at a time, Apache was responding incredibly slowly.

Can't put HTTPS on azure applications

We had intended to put HTTPS on the server via an SSL cert from LetsEncrypt as it's a free service (and one that I use with Netsoc's servers) but unfortunately it's not available to sites on the azure domain (cloudapp.net) so although we had configured our server to serve an SSL cert, we couldn't actually obtain one from LetsEncrypt.

1.1.3 Workarounds/Alternatives

Too low spec

This would have been easily fixed by simply scaling the server on Azure. Azure makes it incredibly easy to scale up and down VM instances at will.

Can't put HTTPS on azure applications

I could simply purchase an SSL cert with something like SSLMate.com to have HTTPS running correctly on the server. LetsEncrypt doesn't offer its services to Azure domains because of the sheer volume of them but any other SSL Cert provider would have worked (we just didn't want to pay).

1.3 Front-end Developer

1.1.1 Tools

MaterializeCSS

MaterializeCSS is a CSS framework, meaning that it is CSS code that we can include directly into the project to provide us with a starting point for our product design. MaterializeCSS includes implementations of Material Design concepts such as responsive typography, colour palettes, components such as cards, galleries, collections, dropdown lists, modal dialog boxes etc. It provides useful interactive Javascript elements, it provides animations on interactions such as button presses and loading-bar elements etc. It provides Material-compliant styling for default HTML components such as form fields. It even provides helper classes for parallax-scrolling elements and vertically-aligned elements.

LESS

LESS is a pre-processor language for CSS, extending the features of CSS by adding variables, macros, includes and the option for writing code in a nested manner. This makes our CSS code easier to write, understand, collect and maintain. The use of global variables in LESS gives us the ability to set style standards throughout the app.

1.1.2 Methods

Gulp

Gulp allowed me to automatically compile LESS and combine/minify javascript as I was editing them. "gulp watch" was an insanely useful command that watched for any changes in my files and then executed a predefined set of commands I outline in my gulpfile.js configuration file. This made it really easy to work with LESS as I didn't have to go and compile it every time I tried even the smallest of changes.

1.4 Back-end Developer

1.1.1 Tools

Sublime Text 3

As an all-purpose editor, Sublime Text 3 was a pretty good standard for us to use. Customising it with light-weight addons made it really easy to emulate the behaviour of an IDE without being tied to a particular language.

Datagrip

Datagrip is one of the tools developed by the JetBrains team. As a student, I had a free license to use all of their software. Datagrip specifically deals with databases and database manipulation. I used it mainly

to inspect and test data as our Laravel migrations handled the actual schema of the tables.

Overall

Overall, my contribution to the project was a lot of core development, API integrations and administration of the staging server.

In terms of core development, I developed the base structure for our project on top of laravel and laid out the main URL routes we'd be using so people could hook them into functionality later. I created the definition middleware between administrators, staff and regular users. As part of this, I created a central admin page for administrators to create events, partners, locations, approve media and add/remove staff members.

When a page requires log in, middleware intercepts the request and checks if a user is logged in or not. If they're not, it redirects them to a login page. Originally, after the user logs in, the application puts them back on the home page just as a default. I wired up a way to save a user's intent between requests and, after they log in successfully, the system automatically redirects them back to the page they intended to go to when they were originally redirected.

In the beginning, I built the initial installation process for site admins so that they could white-label the application with their company info as well as set up the first administration account. The process is done through progressive tabs in a single-page-app so all of the information is uploaded and processed via javascript interacting with API endpoints in our application and executing a callback function to display errors or move to the next section. Because the install process was done heavily in javascript, I needed a way to determine which section a user is at if they reload the page. So, as a user progresses through the process, I push a new URL to their address bar that means if they reload the page, I can easily tell where they left off and move the progress bar appropriately to suit.

To properly handle user uploads as well as white-labelling the application in the install phase, I built a lot of functionality for media. As well as creating the API endpoints for javascript to interact with the approval or denial of user-uploaded media to appear on the front page, I also created a generic function to handle all image uploads for whatever section of the site they were needed. This upload function appears in the installation process, on all of our events pages and in most administration pages such as creating an event or updating a partner. Anything that involved uploading an image goes through this function. For our design, we needed a white version of the company logo so instead of creating one manually, we process the company logo (if it's transparent) when it's uploaded in the installation phase and process it with the Image Intervention package to overlay a full white colour onto the image.

I also created the news section as we felt it's a necessary part for any company website nowadays. For this, I created the functionality to create, show, edit and delete news items. Each news item had a What You See is What You Get editor for the content so it could be presented in rich HTML and not rely on pre-defined styling by developers. Each news item also had an image that was automatically “smart”-cropped to fit the proportions of the design we had laid out. Smart cropping is a feature of Image Intervention which simply tries to scale the image naturally down to the specified width or height

(whichever is closest to the image's) and then it punches out the centre of the image with the specified dimensions which saves disk space (because we have exactly the size we need) and gives us the focus of each image. As another note here, I used encryption (as well as Laravel's built in CSRF protection) to prevent someone from maliciously/inadvertantly deleting news items, events, partners or any other model in our application.

A very small piece of the admin page I'm particularly proud of is that as an admin scrolls down the page, the menu on the left intelligently knows which section their currently viewing and highlights the option in the menu. As well as this, I implemented a “smooth scroll” javascript function so that when they click on a menu option, it scrolls down to the section naturally (instead of jumping to it immediately).

When a user is logged in and has a ticket to an event, they can visit the event page and click “upload media” which will show a modal pop-up window with a drag-and-drop interface to upload new images and manage their current images. This utilises the Dropzone.js library and hooks into more API endpoints handled by Laravel. Everything to do with event media is handled via javascript because it often has to be handled in bulk and reloading the page for each request would have been tedious so I chose to use javascript to make asynchronous actions.

For internationalisation, we couldn't exactly afford to hire anyone to translate our site into other languages so I integrated a composer package and wrapped all of our strings in the translation function. As part of the automatic translation, I created middleware to detect the chosen language of a user whether logged in or not so that we could allow anyone of any language to use the website. The package uses Google Translate to process translations and the result is cached for later. I ended up contributing to the open source package (with a pull request to the Github repo) we used by improving how the main translation function handled caching and increased the overall efficiency.

Ordering a ticket sends an email to the user with their ticket and QR code. To properly scale with something like that, I made sending the email an asynchronous action using Laravel's queue functionality. So when payment for a ticket completes, the application dispatches an email job to the queue and the system can process it in its own time as well as dedicating other threads/cores to several different jobs at one time. It doesn't make much of a noticeable difference with only one or two people ordering at one time but we ran a test with 100 people ordering a ticket and the result was that load times weren't affected for the user and all the emails were sent successfully within a couple of minutes.

In terms of design, I designed and wired up the home page and the main events page. For the home page, we chose to focus very heavily on a lot of imagery and use the very popular google card style to display everything. On the events page, I focused heavily on the details of the event on desktop but if you have a ticket for an event when you visit it on mobile, the ticket is placed at the very top of the page with your QR code front and centre for you to scan walking into the event. Although we also have a ticket emailed to the user, the ability to have a user simply visit the event page and already have their ticket was a really nice feature and I really like the design of it.

During group coding sessions, we all used a virtualised server on our laptops except for Darragh.

Unfortunately, Darragh didn't have a laptop he could use in college so I cloned our staging server on the Azure platform and gave him login details so that he could work via SSH in the labs while we worked locally. It worked really well for the sessions we spent together but he ultimately ended up spending more time at home where he had access to a dev environment and an IDE (whereas SSH required him to use vim).

It was really easy to get things up and running with Microsoft Azure and all the guides on the internet. We'd also been shown how to install and secure PHP and MySQL properly in first year which meant I was already experienced with locking down the server properly. We realised after the demo that our server was incredibly underpowered for running multiple users on the application but scaling the server was really easy on Azure and took about 5 minutes total to go from 1 core and 3GB RAM to 4 cores and 7GB RAM.

2. Lessons Learned

2.1 Own Project

Working with 4 other people can be hard

It's pretty difficult to keep track of 4 other people and it's just as difficult to get a sense of what everyone else has done (or is doing). The standup meetings at the start of each week helped greatly to resolve this and give me a sense of where the project was at. It was incredibly reassuring to have everyone go through exactly what they'd done and explain their thought process.

A mixture of talents/skillsets is both good and bad

It was nice to work with people who had different focuses and talents within the project but overall, we all contributed heavily to back and front ends of the application. It was difficult to start from a position where some people could move very quickly whereas others had to take more time to learn the framework and comfortably developing in it. We made the mistake of moving very quickly from the outset with one or two people. Unfortunately, this meant that the rest of the group got left behind and experienced a great deal of demotivation because of it. We spent a good deal of time after this sitting down all together and explaining how to do things as well as pointing out useful resources. Starting from the start again taught me a lot of things I hadn't noticed about the framework when I was first learning it and it built a lot of trust and comfortability within the group.

The best thing I did in the early stages was admit I didn't know everything and was more interested in other people's approaches rather than trying to commandeer the whole thing and drive it in one particular direction. The other team members said it made them more comfortable sharing their ideas and making suggestions with other people's code. Although I often helped people with bugs and general problems, I was definitely on the same level as the others and we worked better because of it.

Frameworks make things a lot easier

Developing in Laravel made our lives a whole lot easier. Right off the bat, we got user authentication and an easy way to plan out and model our database. Because it obeyed correct object orientation (something a lot of normal PHP work lacks), it was very easy to come from our java background and begin working within the framework. It took a bit of time to explain the how the Model View Controller layout worked together but the object orientation and encapsulation made it much more familiar to us.

When it came to things like google maps and translation, it was incredibly easy to add packages to integrate features that would've taken extensive knowledge of APIs and data formats. Translation was implemented in two hours with very little effort to get the entirety of our site translated into any language Google Translate could handle. It also employed a smart caching mechanism which eased the load on our server and the loading times of our web pages. With very little effort, we have an immensely powerful feature for our application.

2.2 Appraised Project (Drop Table Groups)

Impressive/Good Ideas

- The rainbow top and bottom borders on the forms are really pretty
- The embedded youtube videos on event pages are a very nice way of implementing the media – letting youtube handle the heavy-lifting
- Event locations link directly to a Google map of the location which makes it really easy to sort out directions
- The homepage sliding images are really nice and add a lot of colour to the website

Maybe not such a good idea

- The agree to the terms and conditions button on the register page is purely a placebo, there's no check to see if it's been clicked
- Creating an alert popup via javascript for every error and confirmation message on the site is pretty annoying and makes it seem like every message is an error

3. Suggestions

3.1 Clearer requirements for the presentations

There was a lot of ambiguity around what was required for each presentation and that made it difficult and stressful to create them and rehearse them. We often felt very lost about what we were presenting and that we weren't really given enough direction with what things we should highlight. We had a very good working application early on but spoke a lot more about our management and overall progress as

we thought those were what was being asked but the mark we got made us feel like we'd gotten that fact wrong. I think a clearer definition of what should be in each presentation and some hints on what things you should highlight if you have them.

3.2 Earlier knowledge of the project

With only 8 weeks to plan and develop the application, it was pretty tight all around. It would have helped if we were introduced to the project in the first semester and could plan out some of the things ourselves with our groups so it wasn't as rushed. I feel that if we had more time to prepare, it would have been a lot easier to get further in the project and do more complex things. It would have also made it easier to introduce people to the framework and our development workflow before christmas