Final Report - UniNot.es: A Collaborative Note-Taking Website

Author: George Nixon
Supervisor: Dr. Alia Abdelmoty
Moderator: Prof. Nick Avis
4/5/2012

Table of Contents

1 Introduction

UniNotes is a website for sharing notes about subjects and their associated events, such as lectures. It is socially enabled, such that after signing up, users can follow organizations, subjects, events and other users that they have an interest in, and read about them on a dashboard with a feed of recent activity. The notes are created and edited using Google Docs, with which the site is tightly knit, using the Google Documents List API. This affords the user the ability to work on a set of notes at the same time as other users, as well as providing other features such as downloading the document as a PDF or Word document, spell checking, &c.

The UniNotes website is written in a variety of technologies, with a core of CakePHP supplemented by some JavaScript and CSS. This report details the development of the project, from design to results. The site itself is currently available at http://uninot.es.

2 Design

2.1 Overview

Fundamentally, UniNotes follows a typical website pattern. It has a Home page and an About page explaining the site to new users. It has Sign Up and Log In pages so users can build a profile and get a customized experience. It uses the structure described in **Section 3.3.1 - Basics** of the Interim Report, i.e. it has collections of Organizations (see Figure 1, Subjects and Events as well as Users, Documents (Notes) and Links, with a complex web of relationships between each. All use cases will be framed around these main entities: a user can view a Subject, add an Event to it, then add a Link to that Event.

The goal of this project wasn't to do anything innovative or unfamiliar to users in terms of UI, but rather to harness their previous experience using websites to provide them with a package of functionality unavailable elsewhere on the web (see **Section 2.1 - Competitive Analysis** of the Interim Report). As such, this report will focus on function over form, and utility over usability. Some effort has been made to provide layout and looks that are pleasing to the user, but it is not the driving force nor the intention of the project to break new ground in UX.

That said, in order to present a quick overview of how users access different actions and data on the site, the main features of the site are illustrated in Figure 1, and some key features are highlighted below. A brief discussion of the reasoning behind such features follows.

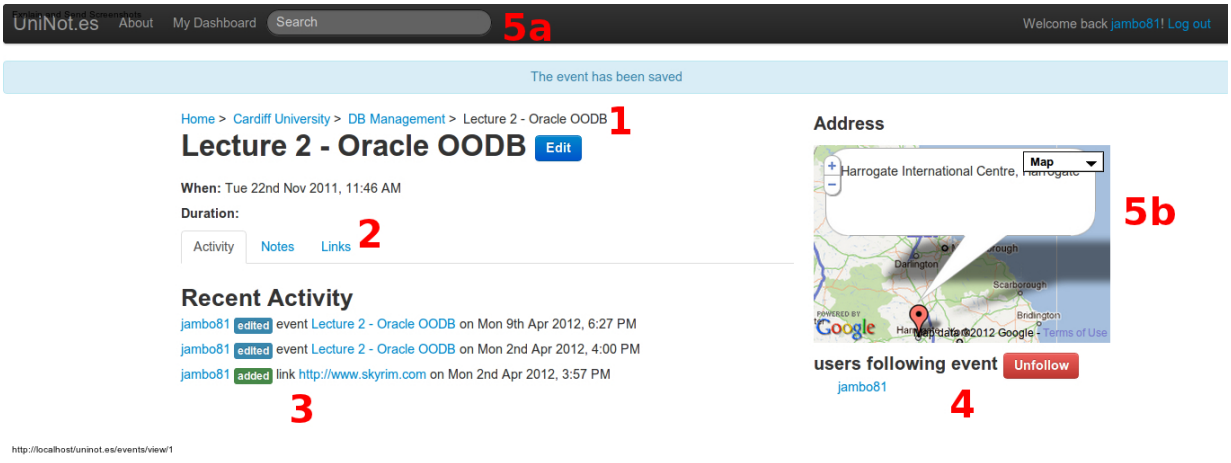

Figure 1: screenshot of a View Organization page in action

1. Basic structure based on educational institutions and conferences, i.e. Organizations, Subjects and Event
2. Subject and Events can be enriched with external weblinks and note sessions - the site

is integrated with Google Docs for advanced note-taking sessions, allowing multiple users to edit the same notes at once
3. Activity feeds for entities and users, to display the latest edits and additions by users to the shared knowledge base
4. Customization on a per-user basis, such that users can “follow” various Organizations, Subjects and Events and find activity updates on their personal Dashboard
5. Place maps, searchability and more small usability additions

2.2 Design Features

2.2.1 Basic Structure

The main actions a user will need to take in order to take advantage of the site’s knowledge sharing features are ordered around a set of structures, namely Organizations, Subjects, and Events. Subjects belong to a particular Organization, and Events belong to a particular Subject. This is to facilitate sharing of information around particular topics or lectures, analagous to the way Wikipedia groups knowledge-sharing around articles based on nouns and sometimes verbs (“Rice”, “UNICEF”, “Canoeing”), which can be educational by the strength of bringing together a collection of relevant information on a single, hyperlinked page.

Wikipedia is less structured than UniNotes, however. For instance, while an article on “Rice” may have a hyperlink to an article on “Staple (food)”, there is not usually a chained hierarchy displayed on the page like “Food > Staple (food) > Rice”. In fact as Wikipedia is an encyclopedia, this topology would be highly problematic, for instance Food could be considered belonging to the category of Fuel, which might be a form of Stored Energy, which might be a form of Mass, and so on, *ad infinitum*. This hierarchy could serve some utility to users, and indeed many articles begin with a sentence such as “Rice is a [staple food]” (where [staple food] is a hyperlink), but this is not necessarily the case across all articles, and would be next to impossible to formalize such an arrangement in such a way. In particular, rice is a food, but it is also a grain, as pertains to botany.

UniNotes exploits the fundamentally more generalizable of the problem space of learning experiences to deliver this hierarchy in its entities. An institution of learning or group hosting a conference has topics it will cover, and learning events will be frequently (though not necessarily) associated with those topics. From these we get our Organizations, Subjects and Events.

2.2.2 Recent Activity

In the Interim report, a suggested feature was to be found in **Section 3.3.3 - User Dashboard** entailing a home screen for a user with a list of recent activity on their followed entities. Later, this was extended so that entities could display their own recent activity irrespective of the logged-in user. For example, on View Organization, we see activity relating to a particular Organization, while a user’s Dashboard will display recent activity based on actions of the user and the entities they “follow” (see Figure 2).

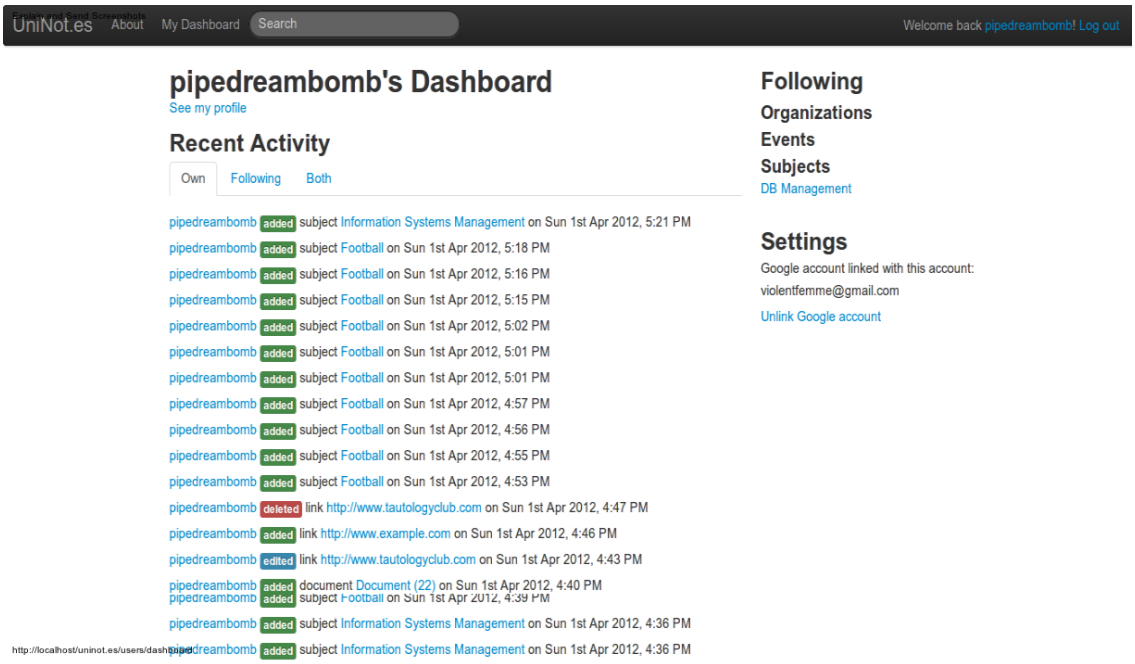

Figure 2: A user's dashboard

This idea behind these lists is to demonstrate to the user what has been happening on the site, particularly with respect to the entity they are viewing. This might engender a feeling of communal participation which encourages the user to take part and contribute. It also naturally highlights the properties and children (e.g. a Subject's related Events) of entities that are most attracting the attention of peer contributors, thus promoting the more useful content as it attracts more mentions in the feed.

The recent activity feed also makes metadata part of an integrated part of the user's experience, i.e. how fresh the data is, how frequently it changes, and so on, each of which is obviously beneficial in its own way.

2.2.3 Dashboard

The User Dashboard page is so called because it allows a logged in user to view activity relevant to them - it is like a home screen that has been personalized for the user, and where she can go to make changes such as to their settings or tracked entities, hence it is analogous to a traditional dashboard on a vehicle where important data and settings are centralized.

The utility of this is that whenever a user returns to the site, she is kept abreast of the changes in the interval between visits. She can see what new content there is on the organizations, subjects and events relevant to her. It would be possible to do this manually, by going around and checking each page previously of interest (having remembered as best she could which those were), but a dashboard is more convenient and reliable for this purpose.

2.2.4 Other Design Features

Maps

As the nature of the site is mostly data driven and heavily textual, it would be less appealing without some visual elements. While a common way of tackling this problem is having images representing different entities, letting users have their own photos and icons, this would mean facilitating the uploading, cropping, resizing and ultimately storage and display of such image files. This is could well be a long and complicated task, so it had to be moved to Future Work.

However, it was anticipated that displaying a map based on the location a user has entered would be a simpler process, as much of the work has already been externalized by services such as Google Maps and Open Street maps. As well as adding a visual element to mostly text-based pages, it could be useful for users who need to know how to get to an event.

Document Previews

While Google Docs provides an excellent interface for creating and editing notes (please refer to **Section 4.1.1 - Google Docs Integration** for analysis of this), particularly in collaboration with other users simultaneously, and despite the fact that users in UniNotes (invisibly, using the Google Documents List API) can create and find notes hosted in Google Docs, they are redirected to the document as an external link, i.e. they are leaving the site in order to see it. By displaying a preview of the document (see Figure 3), embedded in the Go view that they will see before being redirected, this gap is somewhat bridged. The user can see the document while still at Uninotes, which serves as a visual cue to the idea that it is deeply integrated into the workings of the UniNotes site.

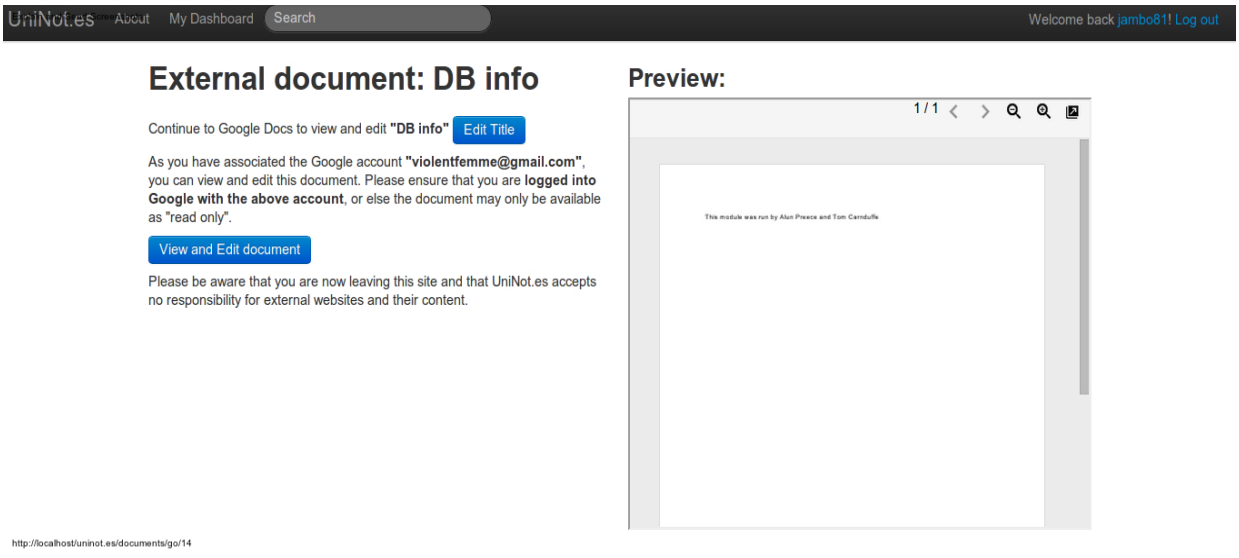

Figure 3: A Document preview

As with maps, while this component serves a need of the user to visualise data and is intended to have a psychological effect on the way he perceives using the site, it also serves a practical purpose by allowing him to preview the contents of the document without having to open a new

page and wait for it to load, thus allowing him to “look before he leaps”.

Tabs

As explored above, there was a danger that the site could become overloaded with data in a way that would make it difficult to navigate and visually overwhelm the user. This would apply most to pages with high data content, meaning that as they got progressively more “useful” in terms of available data, they would conversely become less “usable”. Tabbed content was introduced to solve this problem - each of the main entities (Organizations, Subjects, Events) has a central tabbed pane with currently between two and four tabs.

The advantage of this approach is that the number of tabs, and their related content, can increase without increasing the complexity of the page. For instance, if a page with four tabs had been laid out with each content pane in sequence instead, it would have lead to a long and difficult to parse page for the user to move around.

2.3 Model-View-Controller Architecture

Before discussing design and implementation of the project, it is important to grasp the concept of the Model-View-Controller architecture. Unlike other major software projects, there are no main components to explain and diagram; the code is highly dispersed across *views*, *models*, *controllers* relating to each main entity (Organization, Subject, Event, Link, Document and User), plus *helpers* and *behaviours* which are mixed-in to these as needed.

Each of these types of components are essential to the MVC nature of the codebase, and it is beyond the scope of this report to explain further how they function and interact (see **Section 2.2 - MVC Rapid Development Framework** of the Interim Report for an overview). Instead of a handful of large PHP files that do all the heavy-lifting (business logic, database storage and retrieval, web service API calls), there are perhaps around 50 such smaller CakePHP files (not to mention JavaScript files, images and so on) which have been developed for this project, each with its own set of distinct responsibilities and outcomes. There are therefore no main components to go into in this Design section; rather we can cover the main entities and how they are stored and viewed, before we gain a more holistic understanding of their design in Implementation by adapting the models, views and controllers of these entities to overcome particular problems.

2.4 Database Design

Here, these entities are outlined first by their definitions as data and as stored in the MySQL database. It is by having set out their properties and relationships that we can then explore the visual design of the pages used to view and interact with the entities, at which point the usage and rationale for their design as data objects may become more apparent.

Please note that as CakePHP is an MVC framework, the database and codebase structures are closely coupled, so in general if a table exists called “organizations”, one will need to generate a model called Organization representing it in the PHP realm. As a result, it is both difficult to try to distinctly deal with details in one or the other, as they are naturally interrelated and intertwined. Data structures and relationships between them used in UniNot.es are explained without getting into the specifics of which are defined in the model and which are defined in the database schema.

2.4.1 Basic Models

As put forward in **Section 3.3.1 - Basics of the Interim Report**, some core concepts were required to form a hierarchy common to study and conferences. The top level is the Organization, such as a university or a conference body. Each Organization can have many Subjects, such as educational modules or conference themes. A Subject can have many events, such as lectures over several months, or in the case of a conference, days. The particulars of each are as detailed in the remainder of this subsection. Unique ids, i.e. primary keys, are not explicitly mentioned, but each type in fact has its own “id” field.

Organization

An organization is intended to model an institution or society, so I opted for the following details:

Name	DB Column Type	Form Input	Required
name	varchar(200)	Textbox (single-line)	yes
description	varchar(2000)	Textbox (multi-line)	no
website	special, Link (see below)	Textbox (single-line)	no
location	varchar(1000)	Textbox (multi-line)	no

Subject

A subject is intended to model a theme around which notes could be arranged, such as a module of educational material, e.g. “Database Management”:

Name	DB Column Type	Form Input	Required
name	varchar(200)	Textbox (single-line)	yes
description	varchar(2000)	Textbox (multi-line)	no
organization_id	int, foreign key to Organization	Text box (single-line) displaying name	yes

		of Organization. Disabled to prevent editing.	

Event

An event is a particular occasion of shared learning, such as a lecture. It has the following details:

Name	DB Column Type	Form Input	Required
name	varchar(200)	Textbox (single-line)	**no**
description	varchar(2000)	Textbox (multi-line)	no
subject_id	int, foreign key to Subject	Text box (single-line) displaying name of Subject. Disabled to prevent editing.	yes
datetime	datetime	Timepicker jQuery Plugin (see View)	no
duration	varchar(10)	Select with various possible options	no
address	varchar(1000)	Textbox (multi-line)	no

Owing to the hierarchical nature of the above schema, relationships are created using foreign keys. Each Subject has an “organization_id” representing a foreign key to the Organization table, and each Event has a “subject_id” representing a foreign key to the Subject table. As such in CakePHP, the model Organization “hasMany” Subject, and Subject “belongsTo” and Organization, and similar for Subject and Event.

Some of the optional fields were added later, as this was enough to generate a prototype of the site to work with and it was only later that it became obvious that these extra details would be useful to users and would need adding to forms, displaying and validating. This took considerable effort to add piece by piece, and it did not make much difference in terms of effort whether I did them early or late in the project, so I consider it appropriate that I kept details to a minimum until more ground had been broken on the essential, advanced features. Given more time, I would have liked to add some more fields to these models (see Future Work).

2.4.2 Advanced Models

As explained, CakePHP uses models to represent database objects. There were additional entities required by the site which also were represented by models. The introduction of these

will be discussed in Implementation, however, the design details are as follows:

Link

Represents a hyperlink to a web URL. Has:

Name	DB Column Type	Form Input	Required
url	varchar(2000)	Textbox (single-line)	yes
text	varchar(200)	Textbox (single-line)	no
organization_id	int, foreign key to Organization	Never displayed	yes
address	varchar(1000)	Textbox (multi-line)	no

The organization_id was put in for the special case of Organizations, which can only have one Link, known as its “website”. Subjects and Events can have many Links, and these are associated in HABTM relationships using intermediate tables links_subjects and links_events.

Document

Represents a Google Docs document. Has:

Name	DB Column Type	Form Input	Required
google_doc_id	varchar(300)	Never displayed	yes
text	varchar(200)	Textbox (single-line)	no

Also known as “Notes”, this is a model representing some of the information known about a particular Google Document used to store and edit notes. This model forms the basis of the UniNot.es website. The only field presented to the user is an editable Title, although in the database the document also has a Google Docs id (which Google allocates themselves upon document creation) used for redirecting users to the Google page, and for fetching previews of the document.

User

Name	DB Column Type	Form Input	Required
username	varchar(50)	Textbox (single-line)	yes
password	char(40) (saved as an post-encryption hash, not plaintext)	Textbox (single-line)	yes
active	tinyint	Never displayed	yes
email	varchar(255)	Textbox (single-line)	yes
google_id	varchar(300)	Not displayed in forms, but sometimes printed elsewhere. Is an email address.	yes

When signing up, a user is asked to provide an email address, a username and a password. The two latter fields are used for logging in at a later date. The user has a dashboard, from which she can link her Google account to her UniNot.es account, enabling her to edit UniNot.es' Google Docs. Only this Google link can be altered later, and no public information pertaining to users can be submitted even though users can "view" other users - see Future Work for how this might be desirable and also implemented. "Active" is in theory for disabling/enabling user accounts, but is not yet in use for this (or any other) purpose.

2.5 Layout & Presentation

As explained in **Section 2.2 - MVC Rapid Development Framework** of the Interim Report, CakePHP uses Views to display model data to the user, passed to them by the appropriate Controller in response to a particular request, for example one made by visiting a certain URL in their browser. For instance, visiting "http://uninot.es/events/view/3" could be interpreted (and indeed is in UniNotes) to mean invoking the Events controller, the 'view' action on it in particular, and passing in the extra parameter of 3, meaning we want to 'view' an 'event' with an id of '3'.

One of the powerful features of an MVC framework such as CakePHP and originally Ruby on Rails is "scaffolding" - the idea that if we have already generated the tables in our database and the requisite models, we can automatically generate controllers and views for handling the usual CRUD (Create, Read, Update, Delete) operations. This includes displaying lists of records, paginating long lists and providing buttons to perform actions on them. These scaffolds allow rapid prototyping of basic site structures and behaviours, even if they will eventually be phased out in favour of customized views and controllers giving greater controller to the developer. As such, I concentrated on getting the important functionality working before changing many of the controllers, and left custom views until the latter stages of the project. Hence, as a consequence of the MVC framework and the agile approach I used, the design of the site came largely after the features were in place, as a means of presenting them to the user in an agreeable and user-friendly way.

Please refer to Appendix A for screenshots of the site as presented to the user.

2.5.1 Site-Wide Appearance

Although I am experienced to a degree with CSS and page layout in HTML, I decided to use a style framework that would give a clean and standardized look across the website. I had heard a recommendation of Bootstrap from Twitter by Gina Trapani speaking on This Week In Google. It provides components like buttons, toolbars and tab panes, as well as a layout manager, all accomplished through CSS and some JavaScript. I would need to reformat the divs and links, etc, using the Bootstrap classes.

As some of these were created automatically in CakePHP beyond my control however, such as session flashes displaying errors saving models and so on, I also utilized an open source JavaScript file called Cakephp-Bootstrappifier[1] to make some alterations after the page loads. I made some improvements to it and submitted my changes to the originator of the file using a Pull Request on GitHub, which was merged into the repository on Friday 30th March 2012[2] (see Figure 4 for a screenshot of this taking place).

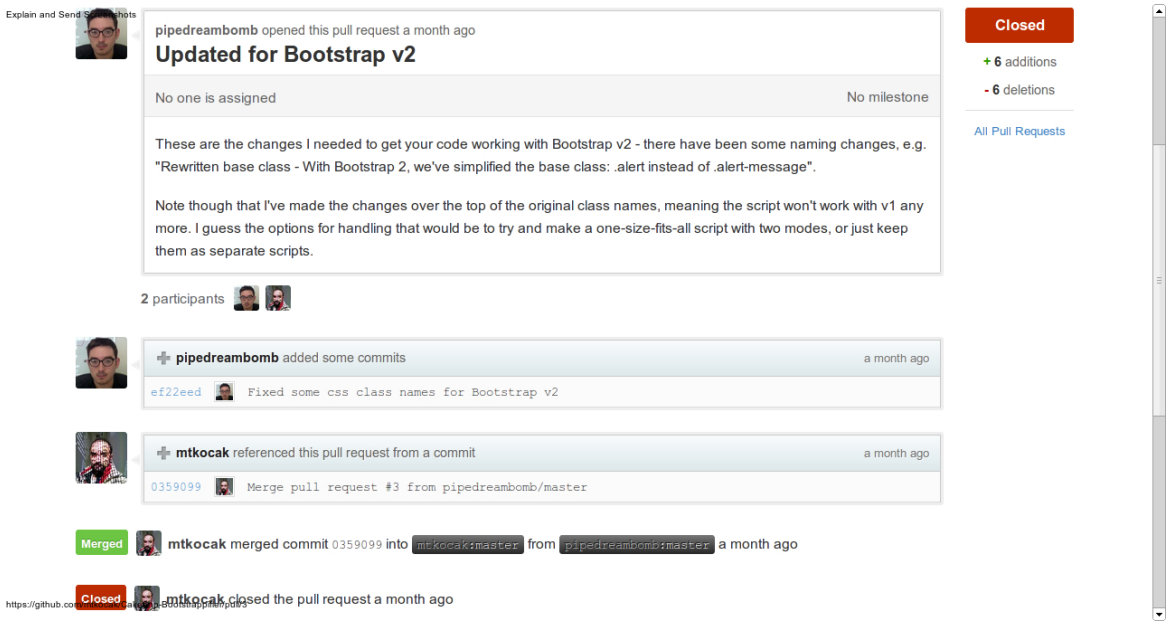

Figure 4: merged pull request on GitHub

The layout scheme, based on the concept of 12 columns in any particular space, such that to

---

[1] https://github.com/mtkocak/Cakephp-Bootstrappifier GitHub - Cakephp-Bootstrappifier project | Mutlu Tevfik Koçak

[2] https://github.com/mtkocak/Cakephp-Bootstrappifier/commit/035909950d8124e1e724ab4f186dd297b41c4cc9 Merge of my pull request by mtkocak (my GitHub name is pipedreambomb)

divide it in three one could use a set of three, four-column blocks, or to divide in two one could use a set of two, six-column blocks and so on. This got me thinking of how I could layout the main pages of the site, in particular with a main pane of 7 blocks and a sidebar of 5 blocks with less significant content. I used this scheme for the View pages of Organizations, Subjects and Events, as well as View User and User Dashboard - predominantly pages with enough data to necessitate spreading it out across the page, rather than presenting a long single column of various lists.

For the colour scheme, I elected to use the default colours that came with Bootstrap, largely with black text, white backgrounds and blue links. I chose custom colours for a few components, for instance the colours of buttons - generally, a blue button implies editing data, and a green button implies adding it, while a red button implies deleting. I kept this same scheme for the colours in activity feeds (e.g. “user123 edited event X” where “edited” has a blue background).

It is worth mentioning here that some elements are common to many pages of the site, at least visually, by making use of “helpers” in CakePHP[3]. These are functions that generate snippets of HTML based on data fed in, and are useful for encapsulating what a list of users looks like, or a set of form buttons looks like, etc. across different pages. The Recent Activity helper (called through `$this->lists->activity()`) that I made is a good example. It prints out activity in the format “[userX] edited event [SpecialEvent] on Mon 9th Apr 2012, 6:27 PM”, where [userX] and [SpecialEvent] are appropriate links to a User Profile and a View Event page respectively. As indicated above, the verb in the sentence is highlighted with a colour to show what sort of activity it is representing.

Figure 5: Top bar

As displayed in Figure 5, the top bar across all web pages on UniNot.es again is based on Bootstrap, this time using the “navbar”[4] component. I simply added some links to certain pages on the left, a search bar, and some user actions on the right (login/out, register and view one’s Profile). For details on how these work, see Implementation.

2.5.2 Views & Pages

Here we explore the design decisions taken when creating the Views, the templates files rendered to display the data from the controller, including any and all HTML components, form inputs, buttons, links and so on. Please see Appendix A for screenshots of these, and where relevant, the original mock designs. These were made with Balsamiq[5], a web app for mocking up applications and websites.

This is not an exhaustive list of all View files; the designs for most Add or Edit views for the various models are not discussed here, as they are mostly simple forms with inputs for text

[3] **http://book.cakephp.org/1.3/en/view/1357/Core-Helpers** CakePHP Core Helpers | CakePHP

[4] **http://twitter.github.com/bootstrap/components.html#navbar** Bootstrap Components - Navbar | Twitter

[5] **http://www.balsamiq.com/products/mockups/mybalsamiq** myBalsamiq Remote User Experience Design | Balsamiq

fields, text boxes and so on.

Organization (View)

It was clear from the database design that the View Organization page would represent a page per organization, such as a university or school, and therefore there would be some basic characteristics to be represented, such as name, description, location etc. I chose to display the location on an embedded Google Map rather than simply as text, although the user can also view the text by clicking on the marker on the map (I chose to hide this by default as it took up too much room on the small sub-frame). I also provided filler text displayed when no location or description is present, for example, "*We don't have an address for this organization yet - why not help out and [add one]?*", with a link to edit the Organization.

The layout of the page common to detailed views across UniNot.es, in that it has a larger left hand column for important information, and a smaller right hand column for lesser details. The left hand column in turn has a tabbed pane in the centre, meaning the same screen real estate can be reused for different purposes depending on which heading the user clicks. I think this helps prevent the user from being overwhelmed with data which might not be relevant to them, as well as reducing the need to scroll up and down the page to find what he is looking for.

In this case, there are only two tabs, one for Recent Activity (explained previously), and one for related Subjects. I chose to make Recent Activity the default pane rather than Subjects because it helps show what users have been doing and will naturally mention where they have added and edited related Subjects, as well as other activities like editing the Organization itself. It was my goal to make the site feel more 'alive' and fresh to have this activity displayed in prime locations on various pages. It would also be easy to change and make Subjects the default pane at a later date if I wanted to put more focus on the Organization > Subject > Event hierarchy.

As can be seen in the Appendix, the finished page is very close to the original design. One aspect missing is the icons for users in the Recent Activity pane - it is common for users on social websites to have small icons next to their name which they can personalize by uploading their own pictures, but unfortunately this would have taken quite a lot of effort (uploading, saving, cropping, resizing, etc) and has been moved to Future Work.

Subject (View)

View Subject shares a lot of common features with Organization; missing is the map, as Subjects do not have a location, but added are two new tabs - Notes and Links. These are simple lists of the Document and Link records related to the Subject in question. Where a Link has an optional Title, this is displayed, but if not the URL is displayed instead. Both are accomplished with helpers and thus are standard across Events as well, and use links to the Go pages for each Link or Document.

Here we also see a breadcrumb trail at the top of the page, in the format "[Home] > [OrganizationX] > SubjectY". These are links to the Home Page and the Subject's parent

Organization respectively. The same is true on the View Event page, only with further detail. Breadcrumbs for other pages like Documents and Links are an item for Future Work.

Event (View)

This view is similar to Subject and Organizations. Of note is that the name of an Event is optional, in which case the heading will be displayed as “Event (SubjectY)”. We can see that not everything in the design made it into the final page - it was easier to implement Following an Event rather than Attending (Yes, No, Maybe). Though the latter is more appropriate to the nature of events, this has been moved into Future Work.

Event (Edit)

This Edit form is special as it has an extra features, a JavaScript “timepicker” allowing users to choose a time and date using a special calendar and sliders. **See Section 4.2.2 - JavaScript** for more on this timepicker.

User (Add)

This page is a form for users to use to sign up for a new user account. As well as the basic inputs for username, email address and password, it also features a CAPTCHA from ReCAPTCHA. This is a device to prove that a user is a human and not a computer, preventing mass signup by scripts intending to add spam content to the site. ReCAPTCHA also has the added benefit of helping digitize scanned books into ebooks, thus benefiting the wider community[6].

I have also added a disclaimer so that new users realize that the site is still at an experimental stage, and that I can’t make any promises about data privacy or integrity at this point. I would like to add a proper, legal disclaimer in Future Work to protect the site from lawsuits and to ensure that users have fair expectations of the reliability and behaviour of UniNot.es.

User (View)

The View User page, also known as User Profile, is the page that displays public information about a user, i.e. their activities on the site and lists of entities that she is following. I had hoped to have time to give room for more information about the user that she wants displayed to other users, such as biographical information and links to their other presences on the web, such as personal blogs or Facebook and Twitter profiles, however this has moved to Future Work.

If a user is viewing their own profile, she is also served a link to visit their Dashboard. This is to serve as a reminder of the separation between Profile and Dashboard and to facilitate navigation between the two.

User (Dashboard)

---

[6] **http://www.google.com/recaptcha/learnmore** What is ReCAPTCHA? | Google

The User Dashboard looks similar to the User profile except for two key distinctions. First, it has a Settings menu on the right hand column, with currently only one setting available, viz. linking and unlinking Google accounts with a user's UniNot.es account. Second, instead of having one Recent Activity feed, it has three in separate tabs: one of actions the user has committed herself, another of actions committed to the entities she is following, and third combining the two former feeds entitled 'Both'. This is to give the user some control over the data she sees, as at various times she may be interested in only her own changes, at others only the changes by others, and yet others ambivalent.

Link (Go)

Rather than put plain links to external websites into pages, I created a "Go" page for Links. This has a couple of advantages: it means that I can warn users that they are about to visit an external link for which UniNot.es is not responsible, and it also means that the site can track outbound traffic when Google Analytics [7]observes them visiting the Go page. It automatically redirects the user to the linked page after 5 seconds, using JavaScript.

Document (Go)

The Go page for Documents (Notes) has little in common with the Link Go page, other than that "View" didn't seem an appropriate verb for a Document which is actually a reference to an external website, namely Google Docs, and is still another click away.

On the left side of the screen is displayed the title of the Document and a button to edit this title. It is then explained that, depending on if a user has associated his UniNot.es account with a Google account, he can view and (given a Google account link) edit the document using the Google website. On the right side is a preview of the document.

Pages (Home)

There are three major sections of the Home page, presented whenever the user visits the root of the domain, i.e. "http://uninot.es". The first is a large carousel, displaying depictions of site use, along with textual descriptions highlighting key features of the site, and a link to the About page. This is a common feature for home pages of late[8] for displaying a lot of dynamic, visual content in a small space on the front page, to which the user's eye naturally gravitates due to the motion of the slider. I felt it would be beneficial to UniNotes as it might quickly explain the site to new users while still allowing for actual site data content below demonstrating its output.

Underneath, there are two columns of unequal size. On the left, wider column is a list of the 50 latest activities occuring on the site. This gives the user an impression of what people are using it for, and how actively it's being used. On the right, narrower column is a list of the latest

---

[7] Not really discussed further in this report, Google Analytics is a service for analying site use and traffic data. It was a simple matter of pasting a bit of JavaScript code into the layout so that the reporting is sent to Google when every page loads.

[8]" The most strong and popular web design trend over last couple of years is a sliding horizontal panels also known as Sliders or Carousels. It's a very effective method to increase the web site usability and engage the user." **http://wowslider.com/index.html#overview** | WOW Slider - **NB**: this is a commercial site for a particular carousel plugin and could be seen as partial on the topic of their popularity

Organizations added to the site. This can be a quick way to get started and, again, to see what other people are involved with on UniNotes. A button is also present to add a new Organization.

Pages (About)

The About page is a longer explanation than that offered on the Home page as to what UniNotes is about and how to use it. It has a three column layout for the three (current) main ideas to grasp about the site, i.e. what it's about (sharing notes), some of the features of the site (Google Docs for notes, add links, event times &c.), and a warning that the site is currently work-in-progress in order to set reasonable expectations with the user as to what she can expect her experience with the site to be at this point in its development.

Below this main section is another columned layout, this time two columns wide, explaining who made the site and how to get in touch, what restrictions exist on copying the site's content, and explaining that the source code will be available at a later date.

3 Approach

3.1 Agile Development

As explained in **Section 3.2 - Choice of Methodology** of the Interim report, I decided to use an agile methodology, kanban in particular as I have some experience using it in the past and so it would be quicker and easier to implement than learning a new technique. Kanban is a system based on a collection of “user stories” that are prioritized and added to ass the project progresses. Each story has a description, such as “users can follow other users”, or “Subject, Organization name not optional”, as well as a point value estimating how much time in development it will take to satisfy the story (roughly speaking, 5 points constitutes an average day’s work). Please refer to Appendix B for an aggregation of these stories along with brief summaries of their undertaking.

For the majority of the development of UniNotes, physical 3”x5” cards were written on and stacked up to form the “backlog” of work still needing to be carried out, as exemplified in Figure 6. This stack would sometimes be laid out and re-organized to form a priority queue, i.e. whatever was on the top of the stack was the next card to start work on, following the completion of the previous card.

The approach was useful as it broke the general problem down into manageable tasks and achievable goals. Knowing the time estimates of the goal helped manage time, plan ahead and prioritize some activity over others where more could be achieved in less time. The fact that the cards were stacked, rather than perennially laid out in a grid (as usual in kanban), was due to lack of an effective physical resource on which to do so, i.e. a large board like a whiteboard to which cards could be pinned or otherwise attached at all times. This had the disadvantage of preventing the desired level of oversight, a “3000 foot view” of the project that makes the progress and direction of the project visible to all. Even if a board was available in my main place of work, my residence, it would still have only been observable to myself and not my project supervisor whom I met at her office in the university. That sets it apart from kanban with physical cards as practiced in agile software offices, where the project manager and developer are co-located.

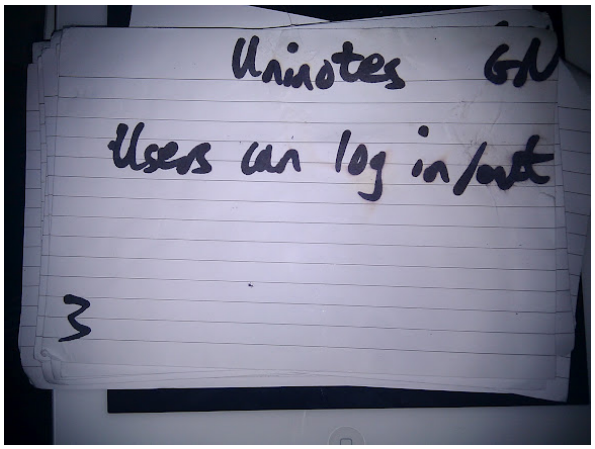

Figure 6 A photograph of a stack of user story cards used in this project

As a result of these limitations, I was largely happy with the method, while being cognizant of some of the drawbacks associated with a limited implementation of kanban. I had surveyed the software implementations of kanban such as Greenhopper[9] from Atlassian (the makers of my source code external repository, see below), but was unable to find a free and simple option that I felt improved on just using cards. In March 2012, however, I came across a web app called Trello, which describes itself as a "collaboration tool that organizes your projects into boards"[10], and decided to try transferring some of my cards to their system. It was a success and I later imported all of my backlog and archived cards into the system.

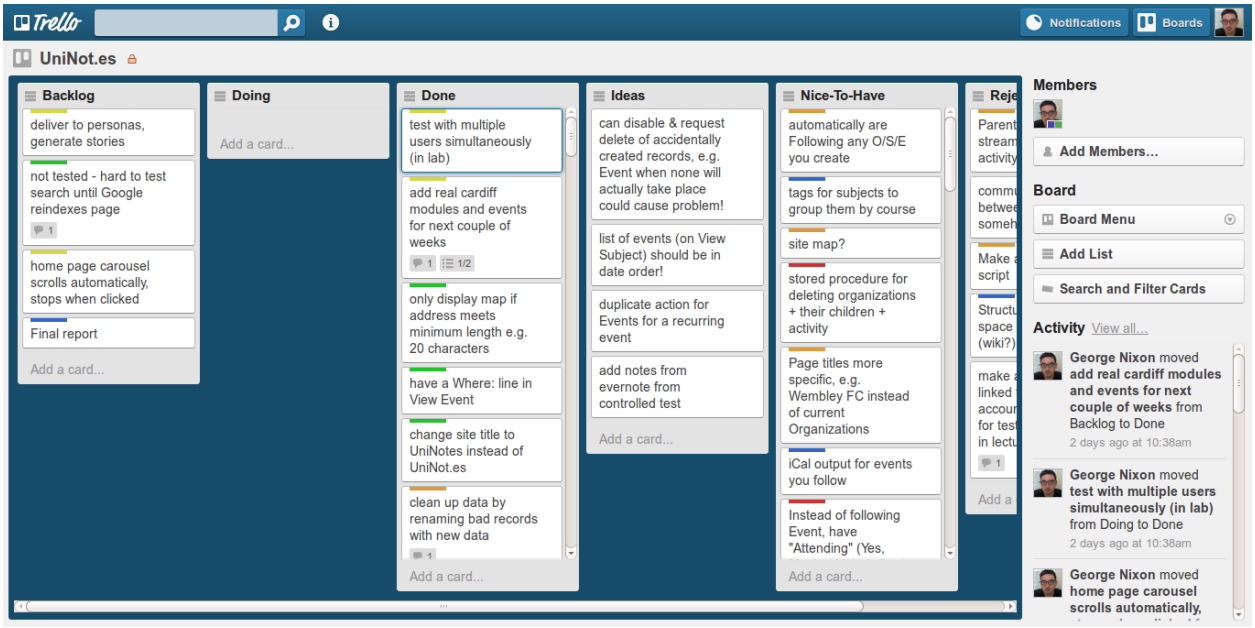

Figure 7 - Screenshot of Trello as used in managing the UniNotes project

Trello is not a kanban system, but is very similar. It is based around the idea of lists of cards, which bear a description (e.g. user story) and can be re-arranged. One might move a card from the "backlog" list to "doing", and then when work on it has completed, move it to "done". See

---

[9] http://www.atlassian.com/software/greenhopper/overview/kanban Greenhopper - Truly Agile Project Management | Atlassian

[10] https://trello.com/about About | Trello

Figure 7 for a screenshot of this in action for the UniNotes project. It brings Trello in line with the overview afforded by keeping cards on a board. It also has the benefit of more space than a physical cards, with the ability to add notes, links and checklists. However, Trello does not have a built-in points system for time estimates. To overcome this discrepancy, I made custom labels for cards (0.5, 1, 2, 3, 5, 8+[11]), each of which had a different colour associated, which made it easy to see at a glance how many points a card carried.

Moving to this new system, even at a late stage in development, paid off quickly. It became easier to create new cards and to prioritize them - I found myself, upon discovering a new bug while developing a user story, to find it easier to quickly write a card, put it in the “ideas” list, and then go back to what I was working on. This was an improvement over my previous working strategy, in which I would sometimes be distracted enough to try and define and then fix the bug, thus straying from my original purpose. Over the course of several hours I could find focus had been lost, and I was not quite sure what range of changes I was committing to source control. With Trello, I had one card in my “doing” pile, and I just had to look at that to regain control.

Given that time was running out within weeks of starting to use Trello, it also became useful in determining which stories still needed to be done as a priority, i.e. which were holding back launch and testing, and which stories could moved to **Section 6 - Future Development** of this report. It could be determined roughly how many days’ work were left before the development was ready to ship, and I could begin testing and writing this report.

3.2 Source Control

While the kanban system was very useful for tracking user stories across the duration of the project, the use of source control provided an even more granular approach, meaning that in writing this report, I had many commits which I could refer to in order to see line-by-line code changes associated with different checkins. It also, of course, provided useful version control, meaning I could revert to previous versions of files where needed and never lose any work performed.

For this, Git was selected as the source control system, which is a popular distributed revision control system. Other systems such as Subversion were options, but I am familiar with Git and there are good web apps like the very popular GitHub (a source of many of the Open Source plugins and libraries used in UniNotes, such as jQuery and CakePHP itself) that enhance it to create a backup repository in the cloud, which they help you navigate through a web interface with advanced markup and version tracking features.

A private UniNotes repository was created on BitBucket.org, which is a website similar to GitHub but with free, private repositories (as opposed to public repositories, often used for Open Source projects where shared code bases are important). All changes were made on my local machine, and then pushed to the Master branch on BitBucket. As such, I was the only person with access to the files, as well as the commits and other historic data.

[11] Trello only permits six labels, so I aggregated points 8 or higher into one category, 8+

4 Implementation

Since the majority of the work on this project has been in CakePHP and some HTML, it would be easy to get into technical detail which would be neither illuminating nor immediately comprehensible to most readers, as the ins-and-outs of CakePHP and the MVC system in general are a large in itself. Full books could be, and indeed have been, written on the subject[12].

I have instead tried in this section to outline my approach to the problem and how I have overcome certain challenges. The source code for the project has been submitted separately and I have tried to add comments in it to explain sections that I felt would not be obvious to someone with an understanding of CakePHP, and PHP in general. See Appendix B for a list of all completed user stories, under “Work Completed".

4.1 Major Features

4.1.1 Google Docs Integration

Since the *raison d’être* of UniNot.es is to provide collaborative note-taking sessions, I decided from an early stage that live editing by multiple users was a necessity. This should take place in the browser rather requiring a special piece of software so that it would be instantly accessible to as many people as possible, thus enhancing the corpus of notes.

Such systems exist, but their complexity requires advanced AJAX skills and a lot of resources to build from scratch - even Google bought AppJet for their EtherPad project, rather than develop one from scratch[13], open-sourcing it and discontinuing active development. I considered using the now open source EtherPad set up on the UniNot.es server to host locally held note-taking. In the end I decided using Google Docs as it has a published API while ecosystem is maintained by Google. The API would be simpler and easier to get started with than EtherPad, and the fact it was hosted off-site would give me more time to develop other features of the site suggested in the previous reports.

Google Docs is a suite of web apps aimed at being the online equivalent of desktop office software such as Microsoft Office. Users can create documents, edit them, upload existing ones in other formats for conversion, and download them in a variety of formats. They can also benefit from the implied networked status of the web app by sharing and collaborating on documents hosted in the Cloud. The API allows applications to perform these tasks too, which is

---

12 For a selection of literature available on MVC as it applies to different web languages, see: **Beginning CakePHP: From Novice to Professional**; David Golding; Apress Media LLC**;** July 25, 2008**;** 978-1-4302-0977-5

Pro ASP.NET MVC 3 Framework (3rd edition); Adam Freeman, Steven Sanderson; Apress Media LLC; June 27, 2011; 978-1430234043

Agile Web Development with Rails (4th edition); Sam Ruby, Dave Thomas, David Heinemeier Hansson; April 14, I 2011; Pragmatic Bookshelf; 978-1934356548

[13] **http://etherpad.com/** Google Etherpad | Google

perfect for websites like UniNot.es which need to automate tasks in order to execute the desired actions of its users, for instance to create a new document attached to an Event in UniNot.es for the purposes of making notes about the lecture.

The first step in getting this working was to be able to track documents on the Google servers using their Google Doc IDs. Since none of the existing models had this property, I created a new model called Document that stored it. Relationships were set up such that a HABTM relationship existed between a Document and either an Organizations, a Subject or an Event. I was able to put together a link to view the document based on the ID and Google's URL conventions, and thus a Go page that could redirect the user to the document viewing/editing screen on docs.google.com.

Now that documents could be tracked, it was necessary to use the Docs API to create new documents. Using a pool of available, unused documents might work for a time, but this would be eventually exhausted given enough users. Hence I downloaded the Zend framework PHP implementation of the Google APIs library and integrated it into the project, as recommended by Google[14] (over writing one's own SOAP client for CakePHP, for instance). The Document model could now create documents over on Google Docs when a new Document was added to UniNot.es.

It became clear that permissions for documents created by UniNot.es could present a number of challenges. The permissions model used by Google Docs allows for several options: a document can be publicly available, available only to those with the "link" (url), or shared with only selected individuals. Each of these options applies separately to viewing and editing - for example, a person may be allowed to view a document but not to edit it. While the former, more liberal options would be easier to implement, and while in theory all content on UniNot.es should be available to everyone, I was concerned that it gave no control over who could edit documents, or rather, who could not. Without this control, users' content would be vulnerable to graffiti and abuse. Hence I decided that all documents should be publicly viewable, but to maintain a list of eligible editors privately, and programmatically.

By creating all documents in the same "group" of documents, permissions only needed to be changed for the group each time a user was added or removed from the list of permitted contributors, rather than needing to go through every document changing the permissions individually. I discovered that I could use OAuth to create a link between a user's account on UniNot.es and a Google account.

Users are redirected to the Google OAuth page, and upon logging in with their Google details, they would be presented with a dialogue requesting permission to pass on their email address (used as a unique identifier) to UniNotes. If the process was carried out to the end and permission granted, the user would be redirected to UniNot.es at a specific URL, the request for which would carry the email address, which could then be saved to the UniNot.es user account, and the person added to the list of permitted editors. This meant sending a second

---

[14] **http://code.google.com/apis/gdata/docs/client-libraries.html** Google Data Protocol | Google

request to Google to add her to the aforementioned group. Upon success of this, the user's dashboard and the Document Go page would display the email address linked with the user's account, along with an admonition on the latter explaining that she must check she is logged into Google with that particular address in order to edit Google Docs. She is also presented with the option of unlinking the account, which would reverse the process by sending a request to Google removing her from the group, and removing her linked email address from the UniNot.es database.

It is worth mentioning here the Preview pane for the Document Go page, as this is not a documented feature of Google Docs, and required some extra work. While Google Docs can have embeddable previews, for example for posting on a blog for others to see, this is achieved by a series of clicks (File -> "Publish to Web..." followed by a dialog prompt) rather than programmatically and in a way which scales to the needs of UniNot.es, i.e. across all created documents. I discovered that Google also publish a separate embeddable document viewer called Google Docs Viewer[15], designed to allow people to publish their documents, such as Word documents or other, non-HTML formats, in a form readable on the Web. I managed to put together a URL which when visited would download a given Google Docs document as a PDF. I then entered this into the Google Docs Viewer source URL, such that the Viewer would display this PDF version, embedded in my Go page.

4.1.2 Following Entities

As indicated in the Interim report, I felt it important that the site would be social, and that activities carried out by users would be tracked and reported such that users could see what had been happening to the entities like Universities (Organizations) or modules (Subjects) they were particularly interested in. I borrowed the concept of "following" as a verb for engaging with and watching from Twitter, where people have "followers" indicating a asymmetric relationship between an observer and an observee. In Twitter this relationship can also be reciprocated, such that the observee can also "follow" the observer, but in UniNot.es entities do not follow users. In fact, users cannot follow users, although this was planned and has unfortunately moved to Future Work because of time constraints.

In order for "following" entities to be possible, HABTM relationships had to be possible between a User and either an Organization, a Subject or an Event. This required creating the necessary tables and model relationships. It also presented a challenge, because as only the former model class was fixed and the latter could vary between the three possibilities, it became non-trivial to allow adding the relationship in a context-sensitive way. That is to say, in CakePHP, it made sense to create the Follow action in one place, namely the Users controller. Simply passing in an id for the followed entity would be insufficient however, as the id could be referring to a record in any one of the three associated tables. This was solved by adding a second parameter to the Add action naming the class to be followed, e.g. "users/follow/event/3", meaning that an Event with the ID 3 was to be followed by the logged-in user. This made for some complex business logic in the User model for creating the relationship and for listing Users following an entity, as well as the Users controller when redirecting the user back to the entity they were viewing after the relationship had been created.

---

[15] **https://docs.google.com/viewer** Google Docs Viewer | Google

4.1.3 Recent Activity

Having decided users needed Recent Activity streams for their own dashboards, as I built this feature, I realised that each entity should also be able to display its own user stream in a similar way. If a common method for logging and retrieving actions was developed, it would be possible to display them in both modes.

The first part of this was logging actions. I was afraid that this could be quite a complicated endeavour, but I found a Behaviour[16] called Logable[17] on the Cake Bakery (a place for sharing CakePHP add-ons and user code). This was imported into the UniNot.es project, and out of the box was able to log most of the necessary components needed to display activity, e.g. the ID of the user who performed a CRUD action, the ID of the model, the type of the model and so on. I created a Helper (`app/views/helpers/list.php`) to display lists of Activities, taking the ID and type of Model as parameters, but in doing so it became apparent that I would need to add functionality to the Logable behaviour as it was not returning all the data I wanted to display in the Helper.

Logable has a useful function called `findLog()` which return a log for a particular model, however, it does not return the activities for linked models. An example of this would be a Subject, where `findLog()` finds CRUD actions such as editing the Subject's descriptions, but not adding relationships such as when a Link is added to the Subject. For this, I wrote a function called `findLinkedLog()`, which takes an array of extra model names, e.g. `array("Link", "Document")`, and for each matching model linked to the base model, it adds logs about those as well. Before returning this collection of activities, it merges them into one array and sorts by date descending, such that all activities relating to the model are displayed, most recent first.

I also needed to customize the Logable Behaviour so that when retrieving logs, the name of the User is returned in addition to the default ID, such that it could be displayed in the helper as "userX edited...". It is better to do this in the Model realm rather than in the Controller (Skinny Controller, Fat Model principle of MVC[18]) and certainly than in the View which should have no interaction with the database, so I added it to the Behaviour, which is also neater because it applies across any Model that `actsAs` Logable.

The Behaviour was now set up to retrieve logs regarding the main models, specifically Organization, Subject and Event. It still needed to work for a couple of Actions on the Users Model - View and Dashboard (see Design). View meant displaying a user's activities on their profile page, i.e. retrieving all the logs with the user ID specified, which was fairly trivial.

---

[16] This is a CakePHP feature that allows consistent "behaviour" across several Models. One create a Behaviour file in the Models/Behaviours folder, and in each Model one sets an option "actsAs" => "xBehaviourName".

[17] **http://bakery.cakephp.org/articles/alkemann/2008/10/21/logablebehavior** LogableBehavior | alkemann

[18] http://weblog.jamisbuck.org/2006/10/18/skinny-controller-fat-model Skinny Controller, Fat Model | Jamis Buck

Dashboard, on the other hand, was a little more complicated, as it meant fetching logs for every Organization, Subject and Event (and their related entities like Links) that the user is following, and merging them into one stream.

I also decided to give the user the option seeing the stream with or without their own actions added to the mix by having several tabs to choose activity relevant to "Following", "Own" or "Both" (again, see Design), for which I had to write the function `findDashboardActivity()` which returned an array consisting of one activity list of each type. Finally, there were kinks to iron out, such as what to display when an Event's name is empty and so on, and I made a way to fetch the log of all activities for the home page.

4.2 Minor Feature Examples

This section highlights a few of the smaller obstacles overcome in the course of building UniNot.es. Inevitably, many more small changes were committed to the repository but are not documented in this report, being too numerous and unilluminating to warrant enumeration.

4.2.1 Maps

I decided during the design phase to add maps showing an Organization's location to its View page. Google Maps was an obvious candidate as it is commonly used across the web, but with Apple and Foursquare among others[19] moving away from Google Maps towards Open Street Maps, a free and open alternative, I first researched other possibilities. In particular, I had heard about a tool called Mapstraction[20], which allows one to write a map-based widget with a layer of abstraction between the widget and the underlying map API. This sounded ideal as in theory I would be able to write the widget once, and change the map source at any time, for instance from Google to Open Street Maps or vice versa. In practice however, I found switching wasn't as straightforward as it sounded, and in fact I couldn't get OSM to work with it at that point. It seemed the project wasn't being maintained well and that I couldn't rely on it going forward, so I decided to simply use Google Maps, as I anticipated use of the site would easily come within the limits of the free map usage limits (25,000 map loads per day), and that maps were non-crucial and if necessary the site could simply not show maps until a new, perhaps OSM, widget was written to replace the Google version.

On entities where a map is desired (currently Organization and Event), a piece of code is required in the PHP of the View to create a JavaScript call that passes on the "address" field from the database. This can be achieved in a single line such as this:

```
echo $this->map->locationMap($address);
```

This calls the helper Map (`app/helpers/map.php`) and its method `locationMap()`, which

---

[19] http://www.appleinsider.com/articles/12/04/05/wikipedia_joins_apple_in_migrating_from_google_maps_to_openstreetmaps.html Wikipedia joins Apple in migrating from Google Maps to OpenStreetMaps | Daniel Eran Dilger

[20] http://mapstraction.com/ Javascript mapping abstraction library | Mapstraction

takes the address as a parameter. The Map helper checks that the location is longer than 20 characters[21], and if it is, it adds the necessary JavaScript files to the page, i.e. the Google library hosted at maps.google.com, and map.js (`app/webroot/js/map.js`), another file written for this project, although in this case much of it was adapted from example Google code.

4.2.2 JavaScript

Maps, as outlined above, are a good example of utilizing JavaScript and XMLHttpRequests (collectively termed AJAX[22]) to bring dynamic content to pages on the client side, updating the page without having to reload the whole document, linked content, &c. This was the only use of XHR, while JavaScript was used in various places to bring pages to life. Sections of JavaScript written for the project fell into two categories: some called existing code downloaded from the web, e.g. tried and tested UI components, while others were original features of the site.

Note that the following code samples are written using the jQuery library, which is acts as a subset of language and features to JavaScript and allows the user to write code in a more versatile and powerful way while performing largely the same operations as a traditional JavaScript program[23]. As a result, some knowledge of jQuery may be required to make sense of the code samples, for instance the phrase `$("#myID")` selects the DOM element with id myID, which is equivalent to `document.getElementById("myId")`[24].

Original Features

Tabs

I made a simple script (`app/webroot/js/tabs.php`) to display different content in the same pane, depending on which tab was clicked by the user. It is displayed below in full, and is documented using comments.

```
// JavaScript for handling tabbed content panes
$(document).ready(function(){

    $("#javascript_warning").hide();

    // When a tab heading is clicked
```

[21] It is assumed that addresses shorter than 20 characters are going to lead to very poor maps, as it is generally too little information to geocode successfully.

[22] **http://www.w3schools.com/ajax/default.asp** AJAX Tuturial | W3Schools

[23] **http://jquery.com** jQuery: The Write Less, Do More, JavaScript Library | jQuery

[24] The jQuery version will return an object whose prototype has more functions available than the normal JavaScript version, e.g. .hide() to make it invisible, or .html() or .html(htmlString) to get or set the HTML content of the element

```
    $(".nav-tabs li").click(function(clickEvent) {
        // Make it look like the active tab
        $(".nav-tabs li").removeClass("active");
        $(this).addClass("active");
        // Make the related content visible
        $(".tabbed_content").hide();
        var id = $(this).text().trim().toLowerCase();
        $("#" + id).show();
     });

    // On doc load, show first tabbed pane
    $(".nav-tabs li a").first().trigger("click");
});
```

All content is loaded in the same pane (in the View's HTML), but is hidden using CSS until the script reveals it. When the document has loaded, there is a click event handler attached to each tab heading. This handler will hide all contents, and then show only the relevant content by making it visible using CSS. It also makes the heading clicked appear active, so the user knows which tab he is currently on. Whenever the page loads from scratch, this handler is called immediately by imitating a click on the first heading (`trigger("click")`).

Go Page Redirect

A Go Page was made for Link entities, as it seemed prudent to show a warning telling the user that they were being redirected to external content for which UniNotes would not be responsible (see Design for more on this). However, to reduce the inconvenience caused by this extra step (as opposed to most hyperlinks on the web which take you to the linked page in one click with no intermediate page), an automatic redirect was created using JavaScript. A countdown is displayed, starting at 5 and counting down to 0, at which point the location of the user's browser changes to the new URL.

The following PHP code creates the JavaScript for this dynamically and puts it into the page's JavaScript buffer, which is then printed at the bottom of the page as the layout file (`app/views/layouts/default.ctp`) dictates. It is dynamic, rather than placed in a static `.js` file, because the information from the database as to the Link's URL needs to be added to the script in order to perform the redirect.

```
<?
$countdownScript = "$(function(){
          var count = 5;
            countdown = setInterval(function(){
                        $('#countdown').html(count.toString());
                            if (count == 0) {
                                        window.location = '" .
$link['Link']['url'] . "';
                                                }
              count--;
```

```
            }, 1000);
        });
    ";
$this->Js->buffer($countdownScript);
?>
```

Here, the function has a variable called count which is initialized at 5. This is displayed in a DOM element with id countdown. With every passing second, the count is decremented by 1, and redisplayed on the page, and if equal to zero, the user's browser is redirected to the new page (`window.location = '" . $link['Link']['url'] . "';`).

Libraries & External Code

There are several examples of small pieces of JavaScript written to integrate existing JavaScript libraries and components into UniNotes. Cakephp-Bootstrappifier has already been discussed in **Section 2.5.1 - Site-Wide Appearance**. Bootstrap also comes with some built-in jQuery plugins, and I used one for the carousel on the Home Page, entitled 'Carousel".

To set dates and times for events, rather than use a text input and convert this from a string to a datetime somehow, it made sense to use a JavaScript component. A plain HTML component would not be dynamic enough, e.g. it might not load enough months on screen, or else would take up the whole page. This appeared to be unusual, as there are many date pickers, including one built into jQuery UI, a set of jQuery-compatible components by the makers of jQuery, but few with time dimensions. A satisfactory solution was found in Trent Richardson's extension of said jQuery UI component with added hour and minute sliders[25].

5 Results & Evaluation

5.1 Testing

Validation of entities was added to the project as work progressed. The website in its final iteration then went through several stages of testing. It was necessary to test that the site not only worked as expected from my point of view, but also that it worked in a 'live' context, with multiple users in real time. This meant putting the website online where people could access it. Regression testing was used to check that the user stories covered previously were still in place, once the source code had been uploaded to my web host at http://uninot.es, and that no issues were outstanding regarding the transition from test environment to the live server. User testing was then performed in a controlled lab context with selected users, and then broadcast the url to students so that many people could test the site to elicit bug reports and feedback.

---

[25] **http://trentrichardson.com/examples/timepicker/** Adding a Timepicker to jQuery UI Datepicker | Trent Richardson

5.1.1 Regression Testing

Method

Putting the site 'live' by transferring the source code to my web host using FTP can cause new bugs to appear, so once this had been conducted, it was necessary to conduct some testing to ensure that the developed features still worked. This was ascertained by using regression testing - testing that all the past features (generally tested at the time and worked on until they worked in most cases), which in my chosen kanban methodology was simple as I had a big stack of user story cards to go through, one by one.

Results

Many of the user stories were working, following upload to http://uninot.es on April 19th, 2012. Please refer to Table 1 for the results.

User Story	Issue found	Resolved by
User can link account with Google.	Google redirects to localhost not uninot.es.	Changing the address passed to Google in the Document model setting up the url to redirect the user to after the authentication process is completed.
Registering users are people, not 'bots.	After registering, displaying dashboard without activity gets error.	Code issue around merging arrays when an array is empty.
Bug - short names in edit Event don't get saved.	Can't edit Events.	Removed a line that output some debug information. Didn't cause a problem on my system but PHP must be set up slightly differently on the host.
Go screen for Links opens in new tab/window.	Links open in same window from activity feed.	Activity feeds were added after this user story was complete, so a change was needed to the activity feed Helper to also open Links in a new tab/window.
Google Docs user can create new document.	Get errors creating Google Docs document.	Small code change, again probably due to a different PHP configuration.

Table 1: Testing issues

Some features' testing was held up by this issues, so I had to revisit them after the relevant blocking issue was resolved. I also came across a number of minor issues or improvements that I was aware were now broadcast online and which I wanted to clear up as quickly as possible. For example, I found the 404 Page Not Found page needed styling to look like the other pages since the introduction of Bootstrap, I thought that the Events page should have a "Where:" line in addition to "When:" and "Duration:" despite the existence of the map already on the page, and I found maps were being displayed for even short addresses like room numbers, which would probably make sense to relevant users but made for a useless and confusing map based on arbitrary geographic point with similar naming conventions, so I set a minimum length on location fields to be checked before attempting to show them on a map in the Map Helper.

5.1.2 User Testing

At this point, I was confident that the various features of the site worked for me, but I wanted to check that one of the core features, simultaneous editing of notes by users, worked by testing it with some test subjects in a controlled setting.

Method

The test plan was simple: they would be asked to sign up for a UniNotes account, link it with a Google account, and then edit a Google document together. For this, I used two friends, Subjects A and B, as well as myself as a third participant. A and B are males in their early twenties and Computer Science students - their selection was not intended to be representative of UniNotes' intended audience, but were selected to help test the technical aspects of concurrent editing of Notes.

Results

All participants were able to sign up to UniNotes successfully. B did not have a Google account to link with his new UniNotes account, so I provided him with my own account by logging into Google on his machine. They were then told them to edit a particular document "New Notes" on the Subject "Individual Project" belonging to Organization "Cardiff University". Participants found that they able to edit the document simultaneously and use the chat facility to discuss changes, thus validating the utility of UniNotes for live collaboration. See Appendix A, Google Docs Editing and Google Docs Chat for screenshots of this process.

5.2 Critical Appraisal

5.2.1 MVC Framework

There were a number of ways development on the project benefitted from the decision to use an MVC framework, as opposed to writing a PHP and MySQL application from scratch. In point of fact, that the site was made using MySQL has rarely been mentioned in this report, the reason

being that using CakePHP meant writing virtually no SQL code whatsoever[26]. As a platform, it is DBMS agnostic, so with only small adjustments, it would have run on a PostgreSQL or Oracle database instead.

CakePHP generates models based on the database tables using a command-line interface called Cake Bake, and without instruction from the developer, it knows how to read and write to the tables based on their metadata. It can also handle pagination, performs joins automatically, and so on, thus eliminating a lot of coding for querying and updating the database in order to perform actions and show pages. By reducing this coding workload, we make a double saving - both writing the code in the first place, and then by negating the need to maintain it as the project progresses and requirements change.

It is this iteration-friendly quality that made using an MVC framework like CakePHP particularly effective in this project. Whether or not agility was important will be addressed shortly, but the fact is that even from the Initial Plan, iteration was favoured over specification. Scaffolding is another core concept in MVC that really helped this approach. By starting with the Models, we can assume a lot about the controllers and views, which means we can Bake pages and actions for Creating, Reading, Updating and Deleting (CRUD) automatically. This includes forms for adding/updating entities, lists of entities (with pagination) and buttons for deleting, etc.

Please refer to Figure 8 for a sample of how UniNotes looked while using scaffolding, noting that little code had to be explicitly to render these views and perform the generic CRUD actions. Note the way the SQL queries are output at the bottom of each page for illustrative purposes - all of these queries were generated automatically by CakePHP.

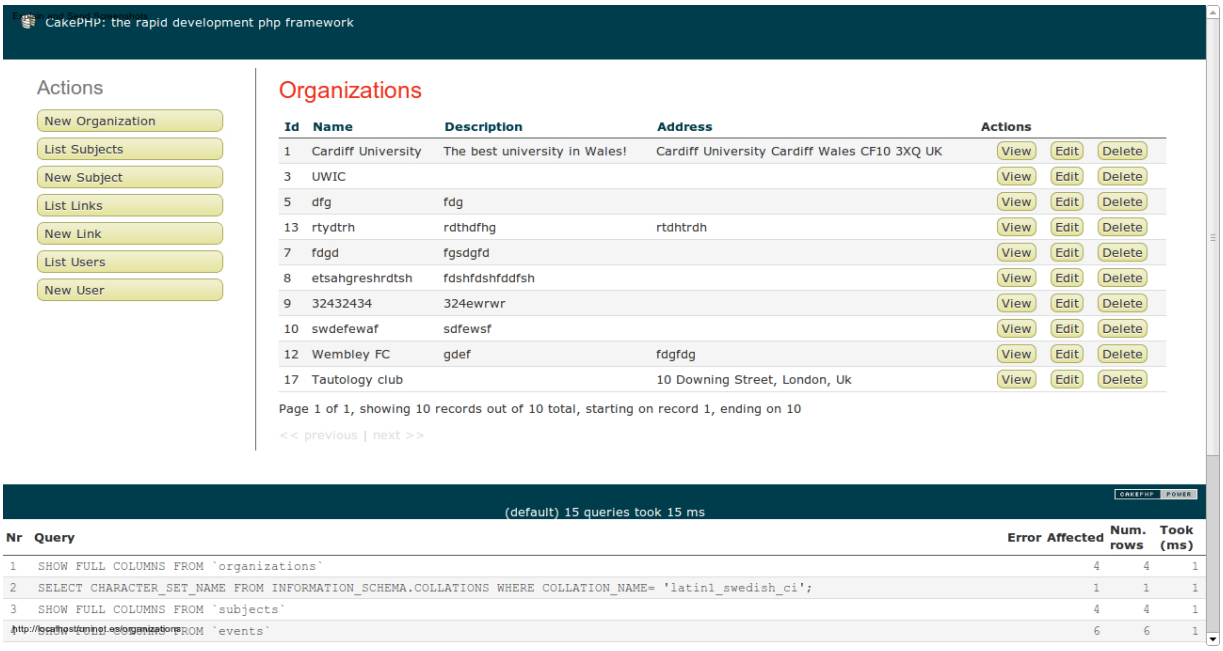

Figure 8: UniNotes while still using scaffolding

[26] A couple of lines created “virtual fields” for some models. These are fields returned as part of a Model as if there were columns present in the database underpinning them, however as the name “virtual” implies, they are in fact not stored and are generally calculated on-the-fly using the “real” fields.

It was easy to get on with creating entities, their relationships, navigation and how to interact with them piece-by-piece, writing user stories for the next logical progression or advantageous feature. I gradually phased out scaffolds and made views that were more tailored as part of this process, but it was not until quite late in the project that a full design of how the site would look was really required. It let me focus on the core functionality while delaying presentation designs until I knew more firmly what it was I would be presenting.

5.2.2 Agile approach

The primary motivation of choosing an agile approach to software development, as opposed to the traditional waterfall method, is the anticipation of changing requirements. While it might be said that of all such projects, an academic one such without commercial or other third-party demands is least subject to change, I would still argue that many eventualities were unknown at the outset. In particular, the amount of time I could devote to the project, and how long each feature would take to develop, were difficult to predict. The dependence implied by relying on many third party tools, from CakePHP to jQuery to external APIs such as the Google Documents List API, accelerated development and provided useful services that would take huge amounts of time to develop internally, but it also introduced many more variables over which I had no control. In particular, the massive lack of documentation for the Zend PHP library for accessing Google Documents List APIs would have been difficult to allow for when creating a full specification upfront.

I feel that by adopting an agile approach, I was able to adapt to these external factors and work on the most achievable goals at any particular stage, one step at a time. Even the fact that there was no upfront interface design paid dividends, as I was able to make mock designs with Balsamiq, bring in Bootstrap and refine the views after the core functionality was already in place, which is an improvement over initial designs that rapidly go out of date and need to be continually synchronized with the way the site now functions.

5.2.3 Feature Set

If we look at the Interim Report, we see that there were five proposed features to be developed if possible, namely Basics (basic structure with notes), Collaboration, User Dashboard, Policing and Discoverability. The first three were indeed achieved. Policing is in Future Work under Flagging Inappropriate Use, and the Discoverability is also in Future Work under Geolocation and Social Contacts. These could have been worked on, but other work such as making the site presentable took priority. As the report stipulates, these were only suggested features and it was part of the agile approach to adapt to the needs of the site as it grew.

We can also look at the User Personas in the Interim Report to judge how well users have been served by the development process.

Rory has most of his needs met: he can create organizations and other entities himself, rather than waiting for his university to get on board, he can follow subjects and see updates on them

from his home screen. UniNotes does not currently provide a way for him to make private notes however; this was seen as being a low priority as he probably already has many means of doing this at his disposal.

Marissa can indeed use UniNotes to post links and create notes, and no distinction is currently made between teachers and students as regards user status, so being both is not an issue. As stated above, geolocation is in Future Work for discovering events near her, so this need is not currently met.

Elana can create notes and links to share content with her students, including links to her own website where her lecture slides are hosted. She is partially able to protect her intellectual property rights over these slides by emailing UniNotes to request that they are taken down, were they to be disseminated in note form in a way that she feels is in violation of her rights, and this process is to be improved in Future Work (see Flagging Inappropriate Use).

Finally, Steve is also benefitted that anyone can set up entities representing unofficial channels for his material, as he does not have time to manage this himself. If he, or someone else working at his organization, found time however, it might be beneficial to his need to “get his message across to a large number of people” by being able to brand his content with logos and images relating to his company. See Uploading & Displaying Pictures in Future Work for more on this.

5.2.4 Specific Flaws

There were a few areas of the site as delivered that are not ideal, and given more time might be improved. A tricky example of this was the Google Docs integration, and I am still not sure what a better approach would be, but the one downside to the permissions model adopted is that when a user logs into Google Docs for their own, usual purposes, their feed of documents is flooded with UniNot.es documents. This situation would only increase as the site gained more users, and thus more notes. This is really due to the way Google has designed Docs - a user can still click a category called “Owned By Me” to see purely their own documents, but their Home screen is composed of every document by them or shared with them. Some users will undoubtedly be unaware of this view distinction and find it difficult to work with their documents, however I have been unable to find a way to stop UniNotes documents from appearing in this Home feed.

I am also disappointed that the site is not as intuitive as it might be - participants in the user testing looked at the site but also asked me what it was about. Some effort was put into creating a Home page and an About page explaining UniNotes, but I’m not convinced that it worked. It is very difficult to measure or evaluate how new users will experience the site, or how it might be improved, and these are not my areas of speciality. Perhaps fuller user testing with more feedback solicited might have proven useful, and perhaps users would have had suggestions on how they might be better informed as to what the site is for, and how to use it. Fundamentally,

I suppose, the driving force behind the project was a need to create a work of academic quality with the desired functionality, and since there were no real end users, it became less of a priority making the site intuitive. Perhaps in another context, UniNotes would have been more usable as a result of iteration according to user preference, rather than theoretical gains in functional capacity.

6 Future Work

As with any project, given unlimited time and resources, there was still a lot left that could be achieved after the deadline. As laid out in **Section 5.2 - Critical Appraisal**, not everything planned was implemented, and so this section comprises a number of enhancements which would be advantageous to the system. It explores some features that might have taken a day or two each, and through to larger and more complex dimensions that could be added to the site with significant effort.

Just like with **Section 4 - Implementation** in this report, only a few examples are outlined here for illustrative purposes, and the full lists of user stories for each category of future work is available in Appendix B.

6.1 New Features

The following are examples of new features, i.e. tasks delivering significant new functionality in return for around 1-3 days new work, which would provide benefit to UniNotes.

6.1.1 Uploading Documents

Currently, the only way to create a set of notes in UniNotes is to create a blank document, with the option of giving it a title at the same time which will then become the title in Google Docs as part of the API call. However, the Google Documents List API also affords its users to programmatically upload existing documents[27]. Various filetypes such as PDFs and Microsoft Word formatted documents are supported, and the file will be converted into a regular Google Doc by default. This conversion can be disabled as part of the service request as not every document will be a good candidate for conversion (particularly highly graphical documents).

For our purposes, we could ask users when uploading documents if they want to enable conversion, which would mean they could both upload docs that would be editable as notes, thus providing a base perhaps from their existing own personal notes collection which others might find a good starting point, and also to add static content such as glossy lecture handouts (assuming they have the rights to do so).

To add this functionality would like require altering the view for the Add action on the Document model to include elements for file choosers and other relevant inputs. The controller would need to change to accept document uploads and store them in a temporary folder on the server. The model would then need to pass the document, if present, to Google as part of the creation service call. On confirmation that the model had been created without error from Google, the

[27] https://developers.google.com/google-apps/documents-list/#uploading_a_new_document_or_file_with_both_metadata_and_content Uploading a new document or file with both metadata and content | Google Developers

controller could then delete the temporary file, at which point the user could be redirected to view the new document's Go page, along with a session flash message confirming that the document had been uploaded successfully.

6.1.2 Sign Up With Google Account

Section 4.1.1 - Google Docs Integration explains the permissions model of UniNotes as pertaining to the editing of Google Docs documents, viz. the site maintains a list of users who have linked their Google account to their UniNotes account, and all documents are editable if and only if the user is logged into Google with the appropriate account when viewing one. The linking of accounts is performed after signing up to UniNotes, from the user's Dashboard, and is necessary before they can edit documents. Participants in user testing (see **Section 5.1.2 - User Testing**) found this confusing, and it could perhaps be better eloquised by presenting a post-signup screen explaining and offering a way to link their accounts immediately. There is a second way however which might cut the steps needed for the process for some users.

Google has a single sign-on method, meaning that its users can quickly and easily register with many sites without having to fill in more forms (such as the one on the UniNotes signup page) by using their Google credentials instead. Facebook, Yahoo and others[28] have similar schemes, all of which are part of the multilateral OpenID[29] system. If UniNotes implemented this system, it would mean users could sign up and link their accounts at the same time, saving time, effort and possibly some confusion. The system works similar to the way accounts are linked already in UniNotes, i.e. by seeking permission using OAuth and storing the information Google returns, should the user accept the request after being told what data UniNotes would be requesting to see.

Implementing this feature would not totally replace the existing signup procedure, as some users do not have and possibly do not want Google accounts, so if their intention is to use UniNotes without ever editing documents themselves, they should continue to be able to do so.

6.1.3 Uploading & Displaying Pictures

In **Section 2.2 - Design Features**, we discuss the relative paucity of visual richness and variety currently present in UniNotes. It would be beneficial if we allowed adding pictures and photos to represent entities such as universities or user graphically. There exist plugins for PHP or JavaScript libraries for uploading and editing photos in the browser, or alternately a basic uploader could be written from scratch. We would need to consider making sure files are of the right size and possibly ratio, as well as being able to be reported and taken down quickly, as this aspect of the system would be particularly vulnerable to abuse. Storing the files may become a problem as the site grows, as images generally take up much more space than plain text in a database. None of these are insurmountable challenges however, and the site may

---

[28] **http://mashable.com/2009/05/18/facebook-openid-2/** Facebook Embraces OpenID; Login With Gmail | Mashable

[29] http://openid.net/add-openid/ Add OpenId To Your Site | OpenID

become more pleasant to look at, users would be able to personalize their profiles, and they might also feel a stronger connection between, say an Organization in UniNotes and its real-life counterpart if there was an image of it present.

6.1.4 Flagging Inappropriate Use

UniNotes relies upon user-generated content, and as such is open to abuse from marketers, vandals and so on. The CAPTCHA on signup may help prevent some bots from using the site, but users still need to be able to report misuse of the site from people who have successfully registered and then go on to add inappropriate content. Currently, they would be able to do that by email, but it would be better to have buttons or links to report activities that fall into this category. It would be useful also to have a published policy on what constitutes inappropriate content such that users will know what they can and cannot do on the site, and offenders can be referred to it and told how they are breaching it. At first, such reporting actions could perhaps trigger an email, but if the problem worsened, content could immediately be pulled down and offending users have their accounts blocked, pending review.

6.2 New Dimensions

The following are larger goals that likely require a greater investment of time, perhaps 1-2 weeks of effort, but in return would afford new categories of utility to the project.

6.2.1 Geolocation

HTML5 and GPS-enabled devices afford unprecedented abilities for web apps to be enhanced by exact or approximate knowledge of a user's location, should the user opt into the appropriate browser preference. It was discussed in the interim report that this would be useful for finding live collaboration sessions occurring near the user in which they might wish to partake, with a particular view to helping find the right session for a lecture or talk that they are currently attending. This could increase uptake in the live, multi-user aspect of the site, and lead to better collaboration and reduced redundancy, especially if previous to this new functionality, users would sometimes start separate sessions for the same event because they did not realise another was already in progress.

Enhancing the site with geolocation would be a multifarious endeavour, as the site would need to be aware of which events were currently being edited, it would need to request permission to use and publish geolocation data from the appropriate users, and it would need to implement some kind of search for the user trying to find nearby events in progress.

6.2.2 Social Contacts

As explained in Sign Up With Google Account above, social networks often provide ways to log into third party sites using a user's existing login details with them, for instance by using Facebook Connect or authorizing login with his Twitter account. While this reduces the time

and effort required of the user in order to provide access to sites, it has a second benefit that the site in question can also access a list of their friends and their details. This can happen not only at sign up, but also at a later stage, for instance to import one's Gmail contacts into the site might be achieved from a user settings screen. The benefit to both user and the site, assuming they want to do so, is that the site can then help them find their friends in its databases, thus recreating the social links he has on Facebook or in his email account on this new site, without having to manually discover people for himself.

In the context of this project, we would map the idea of "friends" on Facebook, or people we are "following" on Twitter, to other users who the user might want to follow in UniNotes. Presently, users cannot follow other users on UniNotes, so this feature would need implementing first, followed by one or more ways to import contacts. Google Contacts from Gmail, Google+, &c. would be the easiest to implement, as we already track users with permission to edit notes by their Google id. For something like Facebook Connect to work, users would have to link their Facebook account to their UniNotes account in the same way they can already link their Google account, after which their friends on signing up will see that they are also members of UniNotes and will have the opportunity to follow one another.

6.2.3 Historical Data

Returning to the idea of damage control viz. Flagging Inappropriate Use (above), it makes sense not only to allow reporting of bad users or bad usage, but also to be able to repair any damage done. Vandalism is a problem in Wikipedia, and sometimes articles are locked in repeated cases, but in general it is dealt with as part of their revision control system. Any version of an article can be compared to any previous or later version, and one can also revert the whole article to a previous state. Thus when improper text is added to the article, perhaps as a replacement to some carefully crafted and informative description, it is quick and easy for anyone to switch back to the undamaged version, in addition to reporting the bad user. Indeed, a change does not need to be malicious in order to roll it back, but simply not up to Wikipedia's standards for article writing.

The same benefits of a complete history of data objects would be advantageous to the UniNotes project also. Logged-in users are currently able to edit many things about entities like Organizations, Links and so on, and there are no provisions for being able to undo such changes. Perhaps the closest resemblance is the Log table, which will contains the record of a change taking place, but not always the associate content change (it logs the "display field"[30] of the model in question, but not other fields). However, it should be noted that this could be a very complicated feature to develop. It would mean views for revising, comparing, rolling back. It would also require the storage of separate versions of entities, either as deltas (only storing the precise changes between the new and old versions after a change) or complete copies, which means more work on the model and/or controller, and more room for error.

6.2.4 Google Docs Metadata Synchronization

Currently, users are able to create new Google Docs documents, and edit existing ones. The

[30] A "display field" is a chosen field, set in the model, which generally speaking gives that model its title. For instance, in Organization, the "display field" is Organization.Name. It is in some sense an arbitrary distinction that does not necessarily apply well across all models.

Google Documents List API also allows metadata to be read, meaning we can interrogate it to find out when it was last changed and by whom, what its current title is, and so on. This data could be used to enhance the current activity feeds, which currently are unaware of changes to the contents of notes (and hence also some of the activities of users making those changes), and whose titles can become out of sync with the documents.

6.2.5 New Document Types

In **Section 2.1.2.2 - Siloed Collaborative Sites** of the Interim Report, we looked at some siloed collaboration web apps that provide useful and varying ways to share knowledge. One of the examples was a site called MindMeister, which provides excellent live, multi-user mindmapping technology. Interestingly, the recently released Google Drive[31] builds on the Google Docs platform in several ways, most notably allowing users to sync their normal files and Google Docs in a folder on their local machine to Google (and back out to their other devices), and it incorporates many previously siloed content creation services, including MindMeister[32]. Another service compatible with Google Drive is Balsamiq, a design mocking tool that was used to create the site designs for UniNotes as presented in Appendix A.

It is conceivable that in the future, sites like UniNotes will be able to use the Google Documents List API to create other formats of document, such as MindMeister mindmaps, and that this functionality could be incorporated into UniNotes to afford users a variety of formats for their notes. This technology is still nascent (Google Drive was announced roughly a week before this report was due), but once the APIs are in place, it could open new avenues for projects such as UniNotes.

---

[31] **https://drive.google.com/start** Google Drive. Keep everything. Share anything. | Google Drive

[32] **http://www.mindmeister.com/blog/2012/04/24/were-on-google-drive/** We're on Google Drive! | MindMeister Blog

7 Reflections

This section of this report will deal with my own reflections as a student about the course that the project took from a non-technical standpoint, how I felt about it and what learning outcomes I gained.

In retrospect, while the presented in **Section 2.1 - Competitive Analysis** of the Interim Report was useful in generating ideas and analysing different approaches to a the problem of sharing notes around a topic, I think it probably set unrealistic expectations of what I could realistically achieve within the timeframe of the project. A university year sounds like a long time to develop a website, but in the face of competing time factors like lectures, coursework, revision for exams and so on, it really translates to a few weeks of solid development time. I had an embryonic desire to create a site approaching the usefulness of GitHub or Wikipedia, but I realised belatedly that those sites had required a great deal more than one part-time programmer to get to where they are today.

I satisfied my main goal for the project, namely Google Docs integration for rich note-taking which could be edited by multiple people simultaneously, all within a framework based around structured learning. However, it was unclear what I could realistically achieve with the rest of the time remaining, and I found that while I managed to accomplish some other big goals like activity feeds, a large proportion of my time was spent re-implementing the interface with Bootstrap and also solving a huge number of small issues, from handling validation through to redirecting users after each of the perhaps 30 actions they might take, bugs, and so on.

My intention with this mountain of small, presentation-level user stories was to create a smooth user experience with as few flaws as possible, however upon writing this report, I found that very little of the weeks of work I'd put into this effort translated into good material for a write-up. I don't know much theory about user interface, I just fixed what I saw as problematic, little-by-little, trying to make the site a clean and pleasant experience. I realise now that it would have been better to continue to build function rather than form, because I would at least be able to elucidate the details of my thought process, obstacles I overcame, etc.

The result of all this is a website I can be proud of, but with fewer features than I would have liked, and with many days' worth of work that probably are impossible to reward with good grades. If I can gain some reflection from this, I suppose it would be that I am more convinced than ever that I prefer practical projects with real outcomes to academic endeavours which my personality and approach to work does not favour.

Finally, I made the project public from the outset, releasing several versions of the site over the course of the year, but at no point did anyone but myself really use them. I think my perfectionism in trying to get the site ready for public consumption meant that time ran out, and I had still not strongly considered how to explain the site to new users. It would have been great to get public feedback by perhaps inducing my fellow students to trial the website for its intended purpose, viz. creating shared notes together about their areas of study. I did not feel

confident to try this, however, and I still am not sure how it would stand up to public scrutiny. I was also not convinced that, although I thought UniNotes was a good idea, or at least good enough to try to develop as an individual project subject, that anyone else would actually want to use it, or find it useful. I intend to maintain the site and to release the source code as an open source project after graduation, and perhaps at some point I will get an affirmative answer to this question.

8 Conclusions

In the course of this project, a site was successfully created that delivered a space for sharing of notes around an educational structure, with some social attributes. There is room for future work, including a number of improvements and possible new dimensions, and in particular there could be enhancements to user experience and intuitiveness to new users. On the whole, however, UniNotes has achieved a lot of its ambitions and overcome many challenges to provide methods for collaboration in learning environments.

Glossary

AJAX - stands for Asynchronous Javascript And XML. Is shorthand for updating parts of pages without refreshing the page or changing the browser's current location

API - stands for Application Programming Interface. Computer programs use APIs as ways of communicating with each other.

Behaviour - an MVC term for shared features written in one location or file that more than one model can adopt.

CakePHP - an Open Source MVC framework written in PHP.

CAPTCHA - a method for testing that the "user" is a human being and not a computer script masquerading as a person. Most CAPTCHAs work by exploiting the visual pattern recognition capabilities of humans which machines are currently unable to match, such as by presenting a graphic of a word with some visual noise that would confuse a machine but not most people.

Cloud - a general, broad term for distributed computing or storage external to one's own computer, often in large data centres.

CSS - stands for Cascading Style Sheet. CSS is used to separate styling rules from the HTML, such that different CSS files can be switched to get different appearances for the same page. It is easier to maintain and edit than styles "hard-coded" into HTML documents.

FTP - stands for File Transfer Protocol. Used to transfer files between machines.

Git - a popular, modern VCS that uses distributed repositories.

GitHub - a website for sharing code using the Git VCS, particularly popular among the Open Source community as it affords unlimited public repositories.

HABTM - stands for Has And Belongs To Many, a type of relationship between objects. An example would be a person Has Many family members And Belongs To the members as a member of *their* family.

Helper - an MVC feature for creating pieces of HTML to display in a view. Can encapsulate chunks of HTML code which would otherwise need replicating many times in various views, thus improving maintainability.

HTML - stands for HyperText Markup Language. HTML is the language that web pages are written in.

JavaScript - a language primarily used in web browsers (all major web browsers support it) to allow programming and changing of content using the user's computer as the agent of change (rather than on the server side, before or after the page is sent).

MVC - stands for Model-View-Controller. See **Section 2.2 - MVC Rapid Development Framework** of the Interim Report for an explanation of this concept.

MySQL - can refer to a database management system (DBMS), as well as its query language

OAuth - a way for sites to authenticate with other sites and seek the user's permission to access their data on the external site.

Open Source - computer software or other development projects which divulge their source code or relevant workings as well as their user-ready output can be said to be Open Source.

PHP - stands for PHP Hypertext Preprocessor, a recursive acronym. PHP is a web language for writing dynamic HTML pages on the server side (i.e. formulating them before they are sent to the user).

Repository - a place where a VCS stores its files, versions and metadata. Some repositories are publicly accessible to allow code sharing.

URL - stands for Unique Resource Location. Is a way of saying a web or filesystem address.

varchar(x) - a database datatype meaning a variably-sized text string, with a limit of x characters.

VCS - stands for Version Control System. Used to store and track multiple versions of files over time.