

THE DESIGN AND DEVELOPMENT OF A GENERIC LOCATION BASED SOCIAL MEDIA ENGINE

Richard Lawrence Ogden

School of Computing, Science and Engineering
University of Salford, UK

Submitted in Partial Fulfillment of the Requirements of the
Degree of Master of Science, 2014

Contents

Acknowledgements	xi
Abstract	xii
1 Introduction	1
1.1 Research Aim and Objectives	3
1.2 Thesis Overview	4
2 State of the Art Analysis	5
2.1 Location Detection & Smartphones	5
2.1.1 Location Data Formats and Calculations	6
2.2 Evolution of Social Media Applications	7
2.2.1 Blogs	8
2.2.2 Collaborative Projects	9
2.2.3 Content Sharing Sites	10
2.2.4 Social Networks	11
2.2.5 Virtual Social Worlds & Virtual Gaming Worlds	11
2.3 Location Awareness and Social Media	12
2.3.1 Location-Based Social Media Service	12
2.3.2 Wikis	13

2.3.3	Blogs	16
2.3.4	GeoSocial Networks	17
2.3.5	Content Sharing Communities	18
2.3.6	Commonality & Differences	20
2.4	Technology Analysis	21
2.4.1	Compilers & Interpreters	21
2.4.2	Database Servers	22
2.5	Application Design	27
2.5.1	The Model View Controller Pattern	28
2.5.2	Adapter Pattern	28
2.5.3	Registry Pattern	29
2.5.4	Software Development Practices	29
2.6	Rationalisation	31
3	Requirements Analysis	34
3.1	Platform	35
3.1.1	Language	35
3.1.2	Data Storage	35
3.2	Architecture	36
3.2.1	Engine Core	36
3.2.2	Permissions	37
3.2.3	Content	37
3.2.4	Framework	38
3.3	Functional Requirements	39
3.4	Requirements Overview	41
4	Implementation	42
4.1	Modular Post Types	44

4.1.1	Required Classes	45
4.1.2	Configuration File	50
4.1.3	Post Type Controller	51
4.2	Geospatial Library	54
4.2.1	Location_Earth Class	56
4.2.2	Location_Point Class	57
4.2.3	Location_Distance Class	58
4.2.4	Location_Line	60
4.2.5	Location_Mbr	61
4.3	Application Wide Permissions and Configuration	63
4.4	Engine Design	67
4.4.1	User Authentication	68
4.4.2	Geospatial Searching	69
4.4.3	User Commenting	77
5	Testing and Evaluation	79
5.1	Methods of Testing	79
5.1.1	Functional Acceptance Testing	80
5.1.2	Geospatial Library Testing	82
5.2	Testing Environment	83
5.2.1	Separate Implementations	83
5.2.2	Commonality	85
5.3	Functionality Tests	86
5.3.1	Test User Registration	88
5.3.2	Testing Geospatial Searching	89
5.3.3	Testing If A User can Edit another User's Post	90
5.3.4	Moderator Delete Comments	91

5.3.5	Test Results	92
5.4	Spatial Tests	95
5.4.1	Unit Tests on Calculations	95
5.4.2	Bounding Box Tests	102
5.5	Results Evaluation	109
5.5.1	Further Implementation	110
6	Conclusion & Further Work	111
6.1	Technology	112
6.1.1	Database Backend	112
6.1.2	Application Framework	113
6.2	Further Work	114
6.2.1	Missing Features	114
6.2.2	Expansion of the Location Library	116
6.2.3	Front End Design	116
6.3	The Future of location-based social media	117

List of Tables

2.1	Feature Comparison	32
3.1	Requirements Specification	40
5.1	Requirements Specification	81
5.2	Requirements Specification compared to results	94

List of Figures

2.1	Examples of the <code>GeomFromText</code> function containing WKT	23
2.2	The <code>Envelope</code> function	24
2.3	An example of two GeoJSON objects	26
4.1	Overview of Application Structure	43
4.2	Post Type Module Structure	44
4.3	Overview of Application Structure (with the Post Abstracts loca- tion highlighted)	45
4.4	<code>My_PostModule_PostTypeAbstract::getPosts</code> method	48
4.5	<code>My_PostModule_PostTypeAbstract::register</code> method	49
4.6	<code>My_PostModule_Bootstrap::_initRegister</code> method	50
4.7	An example Post Type configuration file	51
4.8	The <code>createAction</code> method	53
4.9	Overview of Application Structure (with highlighted geospatial lib- rary)	55
4.10	Location Library File Structure	56
4.11	<code>Location_Point::getRelativePoint</code> Implementation	58
4.12	Distance calculation implementation	59
4.13	Get Bearing Implementation	61

4.14	MBR Calculation Implementation	62
4.15	Overview of Application Structure with the configuration directory highlighted	64
4.16	lbsm.ini	65
4.17	An example “permissions.ini” configuration	67
4.18	Overview of Application Structure with the core engine directory highlighted	68
4.19	The PHP template followed by the HTML which is generated from the template	70
4.20	Generating the geographic URL	71
4.21	Locations and Locations to Posts Table	72
4.22	The getLocationFromMbr method	74
4.23	Example of generated SQL	75
4.24	Sort the list of posts by distance method	77
5.1	Apache Configuration for the three test environments	84
5.2	List of Selenium test cases	87
5.3	The Register Test Case	89
5.4	Find Post from Geographic Coordinates	90
5.5	A standard user editing another user’s text post.	91
5.6	A moderator has the ability to delete a comment.	92
5.7	Results of the functionality testing.	93
5.8	Location_PointTestL::setUp()	97
5.9	Location_PointTest::testGetRelativePoint	97
5.10	Testing the Distance calculation in kilometres	98
5.11	Overall code coverage of the Location library	99
5.12	Unit test coverage of the Location_Point class	100

5.13	Unit test coverage of the Location_Line class	101
5.14	Unit test coverage of the Location_Distance class	101
5.15	Unit test coverage of the Location_Mbr class	102
5.16	Post Locations	103
5.17	Results from 2.2km search	104
5.18	Results from 2km search	105
5.19	Results from 1.55km search	106
5.20	The 1.55km bounding box	107
5.21	The 0.5km bounding box	108
5.22	Results from 0.5km search	108

Acknowledgements

I am indebted my supervisor Professor Nigel Linge and co-supervisor Dr Adil Al-Yasiri along with the rest of my colleagues in the school of Computing, Science & Engineering at the University of Salford for their guidance and advice throughout this project.

I would also like to thank my parents and my partner Colette for their support and encouragement, particularly during the writing up stages.

Abstract

This thesis examines how location aware applications have evolved alongside the rise in high powered mobile devices that provide a location sensing function. A user's location can now be exploited for the delivery of content that is contextually relevant and for users to tag their own content with location specific information. This in turn has given rise to location-based social media in which common social media applications have now become location aware. However, the growth in location-based social media has resulted in the development of incompatible systems and so this thesis presents the case for the creation of a generic and extensible application engine to facilitate new location-based social media applications.

Social media applications have been categorised into six main types, and the impact of location has been assessed against each one. From this a generic location-based social media engine was developed using open-source technologies with the core features which spanned across these social media types, and was built on a modular system which allowed the differences to be either configured through configuration files or programmed as separate modules. This applicability of this

engine was then demonstrated by implementing it in a cross section of social media types. This was on the whole successful with those limitations that arose being clearly identified and their impact assessed.

The thesis provides an analysis of the state of the art for location-based social media , a rationale with a set of requirements for the developed engine and details on key aspects of the engine's design. This is followed by a description of the tests used, an evaluation of the results obtained and a series of recommendations outlining the suitability of the engine as the basis for the creation of new location-based social media applications.

Chapter 1

Introduction

When conceived in the late 1980s, The World Wide Web was initially developed as a unidirectional medium of content delivery. A web page (usually written in HTML) was uploaded to a web server and a user would then download that data and render it via a web browser. This was described as Web 1.0 and it followed the same concepts as other media which had been around for hundreds of years which were published and distributed in a pre-written, non-customisable format without any ability for users to provide input.

At the end of the 1990s a new type of website started appearing. These were sites that did not necessarily consist of static pages, but started having interactive elements. No longer were websites restricted to a broadcast style of distribution, users started to be able to customise the published content which added the ability for users to contribute back either by commenting on the initial data published

or even in some cases by users creating new content themselves. The term “Web 2.0” was coined to describe these new types of websites (O’Reilly, 2005).

The development of Web 2.0 therefore enabled the provision of user-generated content in which users were able to create the original content with the possibility of others being able to edit or comment on that content. This concept became known as Social Media and has spawned a variety of different forms including Blogs, content sharing sites and Wikis. Users were now the authors as well as consumers.

Today, social media websites have grown so large that they are now some of the largest websites around. Wikipedia, for example, claims that it has over twenty nine million articles with an edit rate of six hundred words per minute (Wikipedia, 2013a) and is the sixth most popular site on the internet (Alexa, 2013). Wikipedia is now fifty eight times larger than the Encyclopædia Britannica (based on printed volume size).

The development of the mobile phone in the mid-1980s in many ways tracked the evolution of the world wide web. By the mid 2000s the smartphone had emerged combining the power of a Personal Data Assistant with the ability to use cellular data networks for both telephone and Internet access. Increasing functionality was added to the smartphone, including a Global Positioning System receiver which consequently opened doors to a whole new set of applications which were able to use the mobile’s physical geographic location to provide data which is even more relevant to the user.

These location aware concepts were very quickly introduced into social media which then gained the ability to associate content published with a geographic location (e.g. Youtube, Flickr and Wikipedia), but it also gave rise to a new type of social media called location-based social media . This type of social media had location as the prime emphasis and gave rise to applications such as Yelp, Urban Spoon and Foursquare.

Every day an increasing number of these location-based social media applications are emerging, but as many of them are developed by companies, each application is written from scratch on a proprietary platform. This means that many hours of development time is wasted due to companies having to “reinvent the wheel” and it is also prohibitive to small companies and volunteers who may want to create an application for non-profit purposes. This problem could be reduced if there was an existing platform and location engine for developers to base future location-based social media applications on.

1.1 Research Aim and Objectives

The aim of the work reported in this thesis is therefore to determine if a truly generic location-based social media engine can be developed to support and provide a basis for the future development of location-based social media applications.

In order to achieve this aim, the following key objectives need to be addressed:

1. To determine what common characteristics location-based social media applications have and to identify the features that differentiate them
2. To produce a design that requires minimum code changes across implementations and can be implemented on a wide range of platforms

1.2 Thesis Overview

The remainder of this thesis is organised as follows. Chapter two provides a review of relevant literature which includes a discussion of the existing major location-based social media applications and the associated technologies which can be used to develop these. Chapter three derives a specification for a generic location-based social media engine for which a design and implementation is presented in chapter four. Chapter five provides the results obtained from testing the generic location based social media engine and its effectiveness in being able to deliver a variety of location based social media applications is analysed and discussed in chapter six. Finally, chapter seven concludes the thesis and identifies areas for further research and development.

Chapter 2

State of the Art Analysis

As described in the previous chapter, the aim of this research is to develop a generic location-based social media engine. In order to realise this aim there is a need to understand location detection in devices, social media applications, open systems and design patterns that can be used in development. Each of these will now be considered in the following sections.

2.1 Location Detection & Smartphones

Smartphones have been key to the evolution of location-based social media . Smartphones are a combination of a number of technologies into a single device:

mobile phone and data connection, the features of a PDA (Personal Digital Assistant) and GPS (Global Positioning System) for positioning (Charlesworth, 2009). The first concept of a Smartphone (which pre-dated the term “Smartphone”) was the Simon which was prototyped in 1992 by a joint venture between IBM and BellSouth Cellular. Simon was a mobile phone which exhibited PDA-like features including having a touch screen and apps. However, it was restricted as web browser and mobile data infrastructure technology was not sufficient for it to reach any meaningful potential (Sagar, 2012). However, it was not until the year 2000 when “selective availability” was discontinued and consumer devices were able to get a much more accurate location from GPS. The first mobile phone to implement inbuilt GPS is the *Benefon Esc!* which could display the user’s location on a map and had the ability to integrate with Yellow Pages short messaging service to bring the user information about services near by (Kaasinen, 2003).

2.1.1 Location Data Formats and Calculations

When a location is determined it needs to be represented in a standard format. These formats are generally represented in terms of coordinates along two axes. There are two major systems used to map coordinates of a location (these are called geodetic systems): OSGB 36 and WGS84. OSGB 36 (Ordnance Survey Great Britain 1936) is a standard grid system developed by the Ordnance Survey. This is the grid system used on Ordnance Survey maps of the UK, but is limited as it is not a global system (Ordnance Survey, 2013).

WGS84 (World Geodetic System 1984) originated from the United States military

during the 1950s when the various departments of the military came together and created the World Geodetic System 1960. This went through various revisions up until 1984 (Burkard, 1984). WGS84 is what is used today by GPS systems across the globe as well as for online location based services such as OpenStreetMap, Google Maps and Wikipedia.

Due to the nature of the surface of Earth being curved, standard Euclidean geometric equations cannot be used. Instead the Haversine formula is used to calculate distance using great circles across the surface, great circles are the shortest path between any two given points on the surface of a spherical object. Veness (2012) has published this and other geospatial formulae along with JavaScript implementations and examples.

2.2 Evolution of Social Media Applications

When social media came into public awareness, it was used to describe various different media forms including: blogs, collaborative projects (such as Wikis), social networking sites, content publishing sites and was even used to describe virtual worlds for either social interaction or game play (Kaplan & Haenlein, 2010). Although very different media types, the commonality between them is that the content is created by the people who consume it.

Social media sites have been analysed and shown to comprise of seven functional blocks: identity, presence, relationships, reputation, groups, conversations and

sharing. Different social media types focus on different functional blocks, so not all blocks are present in all social media types (Kietzmann, Hermkens, McCarthy & Silvestre, 2011).

2.2.1 Blogs

Blogs (short for weblogs) are online diaries where people can periodically (as frequently or infrequently as they like) write posts which are then published on the website. These were one of the first instances of social media and came into existence in the late 1990s when server-side technology enabled users to add information onto a web site without the technical knowledge which was previously required. The first blog platform was launched in 1998 and called Open Diary (Kaplan & Haenlein, 2010). This site offered the ability for users to interact via a comments system.

A blog is a very versatile platform, different people use blogs for different purposes. There are generally five motivations for blogging: documenting of life, where users write about their day to day activities. Commentary blogs where users write about specific issues and current affairs, often opinionated and political. There are cathartic blogs in which users use the blog as an outlet to express their emotions, blogs which users use as an output to aid them in a creative process and finally a blog which can be used as a “community forum” where often multiple blog authors write about a specified subject or subjects (Nardi, Schiano, Gumbrecht & Swartz, 2004).

Although blogs are not specifically covered by Kietzmann et al. (2011), they do state that blogs are strongly based around the “conversation” building block. There can also be relationships through this conversation but they do not have formal relationship connections. Identity also plays a part as a user normally blogs as a specified author (whether or not that author name is real or a pseudonym).

2.2.2 Collaborative Projects

The most common form of collaborative project is a Wiki. Wiki comes from the word “quick” and are websites where the content is contributed and edited by the users. Collaboration is key for a Wiki where users are not only encouraged to create content but also edit content contributed by other people. This allows for correction of erroneous material (which may have been put up there in error or maliciously) and allow material to be kept up to date. This self-righting mechanism in theory should keep the material on a Wiki accurate and current (Dennis, 2013). The most famous implementation of a Wiki is Wikipedia which is powered by the MediaWiki software. This Wiki allows anyone to edit any article (with a few “restricted” articles). With Wikipedia there is no individual ownership of content as often it is written by a number of authors. Other examples of Wikis include documentation for Open Source projects, such as projects hosted on GitHub.

Wikis seem to predominantly revolve around the “sharing” and “conversations” blocks from Kietzmann’s functional blocks. A wiki allows the sharing of content through an article which multiple people can contribute to, which in turn will

enable discussions that can form through the comments people leave on the article. “Identity” is virtually non-existent as no single user owns the article, and some Wikis (such as Wikipedia) allow anonymous accessing (although the user’s IP address is stored) (Wikipedia, 2013b). However, all wiki software still allows user authentication as well.

2.2.3 Content Sharing Sites

A content sharing site is a service which allows users to create content and publish it. Unlike a Wiki, users are generally solely in control of their own content and content can only be edited or removed by the creator or a site administrator.

The most famous content sharing service is Youtube which is used for sharing videos. This was launched in 2005 and purchased by Google for \$1.65 billion (Google, 2006). There are other examples of content sharing services such as Instagram and Flickr (for sharing photos which were bought by Facebook and Yahoo! respectively), SoundCloud (for sharing audio) and Slideshare (for sharing presentations). Kietzmann et al. (2011) states that the main functional block of a content sharing site (YouTube in particular) is “sharing”.

2.2.4 Social Networks

Social networks are probably the most prominent example of social media. The content that users put up is generally to do with themselves and their own experiences. This makes social networks quite different from the previous three types of social media as it is not oriented around the content which is published, but around the relationship users have with each other. The content generally consists of a user profile, an ability for the users to post comments and other media and an ability for a user to “befriend” or “follow” other users and their posts.

The biggest social networking site is Facebook with an estimated 750 million unique monthly visitors followed by Twitter which has 250 million as of October 2013 (EBizMBA, 2013). According to Kietzmann et al. (2011), the main functional block for Facebook is “relationships”.

2.2.5 Virtual Social Worlds & Virtual Gaming Worlds

Virtual Social Worlds and Virtual Gaming Worlds are fully immersive worlds where users control avatars which can interact with other users’ avatars as a user would do in the real world. While some of these virtual worlds have spaces modelled on the real world (such as San Francisco in Second Life), it does not relate directly to the physical locations. Therefore it is deemed that these social media are not relevant to this research because of this lack of focus on the real world for which physical location is a key parameter.

2.3 Location Awareness and Social Media

Location-based social media is appearing in a variety of forms, however location awareness is not limited to location-based social media applications. Existing social media are becoming increasingly location aware.

2.3.1 Location-Based Social Media Service

Kim, Lee, Lee and Paik (2010) devised a system where a client retrieves data from a server which is relevant to them based on the user's location and interests. A user would complete a survey to determine their social characteristics. This would then be analysed to allow the system to serve only information which the user would find interesting. This type of system is quite complicated because it requires a certain amount of fuzzy logic to be used to determine what is relevant and what is not. Furthermore, the user's relationship with the author is taken into account, thus meaning this system is also a social network. This could be very useful if the user would like to see what their friend has to say about a location, but there are quite often times that the social relation may not be relevant.

The project is written using .NET technology, which is quite limiting because it will only work on .NET compatible platforms. This means that the server must be a Windows server with the correct version of the .NET libraries. In addition, the client must be a device which has the .NET framework. This limits you to a single platform (Windows) which is far less common than other platforms such

as Android or iPhone. This would cause problems if this product was to be rolled out to consumer devices. By using existing web standard technologies, there may not be a need to have a physical client application installed on the device, and use the application straight through a web browser.

2.3.2 Wikis

WikiWikiWeb was developed in 1995 and is widely regarded as the first Wiki engine developed. The idea was to allow quick collaboration between authors for the Portland Pattern Repository, a project which revolved around pattern languages (C2.com, 2013). Since the original incarnation there have been many Wiki implementations written for a variety of different platforms and in a number of different languages. There is no evidence that WikiWikiWeb contained any geo-location

2.3.2.1 MediaWiki

MediaWiki is the software behind the largest Wiki in the world: Wikipedia. Users can create and edit articles on the whole without moderation (there are some articles which are protected to stop abuse). The application that powers Wikipedia is MediaWiki; an open source wiki system written in PHP.

The MediaWiki database schema is of great interest because the articles are

all created by the users and are continually being updated. Articles may be updated with erroneous information, and therefore require more alterations or to be “rolled back” to a previous version of the article (or page as is the term used by MediaWiki). This means that all versions of the article are kept for future reference. The “revisions” table contains the metadata for each modification of each article, and references the text for that revision (in the “text”: table). This links to the user (the person who edited it), and of course the table of each wiki article (which is the “page” table).

This is a very useful structure but limited to a single content type for articles. It only allows plain text with the standard wiki markup. While this is a good format, it is restrictive and doesn’t allow other types of data (e.g. video, audio or images) to be used directly, only embedded in the content.

MediaWiki also has a complete user management system. In MediaWiki users are members of groups (e.g. Registered-users, moderators or admin). Each group can be given general permissions, (e.g. Only moderators can add or edit pages) and also page specific permissions (e.g. Registered-users can add or edit pages, but on some pages only moderators can edit them). This means that MediaWiki is more flexible, and also allows it to be used in more than just the scenario it was created for (which was to power Wikipedia).

MediaWiki does have the ability to “geotag” an article, however historically it has not been location driven. If a user wanted to geotag an wiki entry they would simply add the coordinates in Wiki markup. Now Wikimedia have released Geodata, a method of storing location information in a separate database for

MediaWiki entries. This enables the use of geospatial searches, which prior to this were not really possible (Semenik, 2013).

MediaWiki is written in PHP, which is an open-source language which runs on a variety of platforms and interacts with MySQL which also runs on a number of platforms. This gives flexibility into the server it can be deployed on.

2.3.2.2 Other Wiki Software

DokuWiki is another popular Wiki engine. It is also written in PHP but does not require a database as it uses text files to store the data. The DokuWiki project is aimed at small-businesses and project documentation, and has the advantage of being very easy to set up as no database configuration is required, however the flat file system does pose limitations.

LocalWiki is a Wiki engine written in Python design around localisation. Articles are written and geographically tagged to the location that is relevant to the article. LocalWiki uses PostgreSQL with the PostGIS extension installed to handle the spatial data queries. The PostGIS extension allows the geospatial calculations to be performed within the database.

2.3.3 Blogs

The most common blog platforms of the modern day are Blogger (owned by Google) and Wordpress. Google do not release the number of blogs which are hosted by Blogger, but it is believed to be tens of millions. In 2008 Google announced that they were experimenting with the geotagging of blog posts in Blogger. This allowed a user to select a location on a map which was then displayed along with the blog post (Google, 2008).

Wordpress are reporting over sixty five million blogs at the time of writing (Wordpress, 2013c). The reason for the huge adoption of Wordpress seems to be because of the two ways a user can use it. They can use Wordpress.com which is owned by Automattic which allows users to set up their own fully functional blog on the Wordpress.com server for free with adverts, but if a user wants extra customisations or the removal of adverts then they must pay to be upgraded to a “premium” account.

The second way Wordpress can be used is to self host the software. The Wordpress software is released under the GNU General Public License (Wordpress, 2013a) and therefore can be freely downloaded and used. Plugins and themes can be created for Wordpress and installed into the software without overwriting any of the existing code. This, along with the large library of plugins and themes Wordpress hosts for users to download means that the software has expanded beyond the realms of purely a blogging platform and is now often used as a Content Management System for many different types of website. Wordpress has very little in the way of geolocation. Out of the box Wordpress has no concept

of geolocation, but due to the open source nature of the Wordpress project there has been a geolocation plugin which allows users to geotag a blog post.

Wordpress is written in PHP and requires PHP 5.2.4 and MySQL 5.0 or newer to run (Wordpress, 2013b). These are cross platform, open source projects and therefore should enable Wordpress to be run on most web servers and operating systems.

2.3.4 GeoSocial Networks

Gowalla was a location-based social network and was launched in 2007. In 2009 Foursquare, a direct competitor to Gowalla was launched and although many opinions were that Gowalla was better designed, Foursquare ended up dominating due to its larger user base. This was thought to be for a number of reasons such as Gowalla's more complicated user interface. Gowalla was based in Austin whereas Foursquare was based in New York, and as social networks are all about the users and not the content, New York had the population density (which is important for a GeoSocial Network) and therefore the user base (Schonfeld, 2011).

Gowalla ended up being purchased by Facebook who were not interested in the product itself, but the talent for them to create their own geographical addition to their existing social networking site. Facebook users could now "Check In" to locations and tell their friends where they are. As Facebook is the largest social networking site out there with 1.11 billion users as of March 2013 (The Associated Press, 2013), they are able to dominate social networking market due

to social networks being about the identity and relationships between users and less about the content.

Foursquare have retained their user base due to game-like features. Users score points for “checking in” to a location. The user gets more points if they check in to new places or check in to a specific place more times than any other users (this is called being the “mayor”). As well as point rankings against other users, Foursquare offers badges for achievements (e.g. number of check-ins into a specific category of place). All these factors create competition between users and therefore encouraging usage of the service.

Around the same time as the release of Foursquare, Google released their own GeoSocial Network: Google Latitude, which also has failed to gain market share. This could be to do with a number of factors but it definitely is to do with the competition from Foursquare. As with Gowalla, if the users are not using the service, then a social network becomes ultimately pointless. As Foursquare holds this market there seems to be little use in developing a GeoSocial Network, and for that reason it is deemed to be beyond the scope of this project. Furthermore, as all these are proprietary applications, there is little that can be determined about how the locations are stored and queried.

2.3.5 Content Sharing Communities

There are various content sharing communities in existence which predominantly focus around a specific medium (images, audio, video etc). The most famous of

these is probably YouTube. Created in 2005 and purchased by Google in 2006, YouTube is one of the few examples of a successful product despite it being bought out by a larger company (mostly large technology companies purchase smaller ones for the talent as opposed to the product as Facebook did with Gowalla). YouTube allows users to upload videos and other users to discover them through sharing or search. Youtube also allows users to add geographic information to their videos via the “video location” field in the “advanced settings” when editing a video’s information. This can then be retrieved through the API (Application Programming Interface) (Google, 2013) and will also appear on other Google location based services such as Google Earth and Google Maps but cannot be used to search for videos on the YouTube website (Agarwal, 2011). Youtube is not the only application in this market, there are many others including Vimeo and more recently Vine, but neither of these allow geographic tagging of media.

Flickr is similar to YouTube except it is used for sharing images rather than videos. Flickr pre-dates YouTube as it was launched in 2004 and acquired by Yahoo! in 2005. Flickr also allows users to tag the geographic location of images by tagging the image’s location on a map which can then be searched for using the API.

Geotagging in these systems seems to be very much in its infancy, and although the functionality is there, it has little to no effect on how the application performs to a normal user as there are no maps or obvious representation of the locations. The only effect it has is if a developer wants to use the APIs, they can then perform queries on or consume the geographic locations of uploaded content.

2.3.6 Commonality & Differences

Looking at the existing social media applications, it has become apparent that there are three key features which they have in common: identification/authentication, the posting/reading of content and the ability to comment on said content. In order for users to be identified there must be a registration and login feature. This allows users to be identified on the social media platform and prevents users masquerading as other users.

Users (either authenticated or anonymous depending on the application) are able to create media across them all, although the media varies from application to application. For example, users can create articles on Wikipedia, blog posts on Wordpress, upload images on Flickr and add videos to Youtube. Because the media is generally text and easily editable on Wikipedia and Wordpress, users are able to edit and delete existing media, whereas on the more complex media types (such as videos and images), they are generally just removed.

Users should also be able to interact with other users on the subject of the media. This is achieved via comments in Wordpress and Youtube and a talk page in Wikipedia. User interaction is key to social media as it has become apparent that user feedback is as important as the creation of the content.

2.4 Technology Analysis

Web applications generally consist of a number of services running on a web server. These are often referred to as a “server stack” and normally include an operating system, compiler/interpreter, a database management system and a web server. There are some technologies which require specific operating systems or web servers. As this project is to develop a generic engine, this section will focus on cross-platform technologies.

2.4.1 Compilers & Interpreters

There are a wide variety programming languages across the web, although there is one major language which powers far more websites than any other; PHP. PHP stands for “PHP: Hypertext Processing” and was developed in the mid-nineties by Rasmus Lerdorf. It is an open-source cross platform language which can be run on a number of different web servers and platforms and it is used to power many of the websites mentioned above including Facebook, Wordpress and Wikipedia. PHP is the widest used interpreter on the web with 80% of web sites running it and has a large amount of development tools and frameworks (such as Zend Framework, Symfony, CakePHP, CodeIgniter) which can aid a developer in creating a large application structure and keeping it flexible. ASP.NET is the next most popular language with a 19% market share (W3Techs, 2013). ASP.NET is a language developed by Microsoft as a successor of ASP (Active Server Pages) and normally runs on the IIS web server on Windows.

Python is another open-source and cross platform language. When developing a web application it is often used in conjunction with the Django web framework. Like the PHP frameworks mentioned above Django provides an application structure and pre-written code which can be used and reused throughout an application. Ruby with the Rails framework is also another language with an associated framework. These both have relatively low usage across the web.

2.4.2 Database Servers

Databases are used to power most social media applications on the web. Each database system has its own specialities, this section will outline the some of the most common databases in use on the web which have geometric capabilities. These databases need to be able to handle geospatial data as this is a core requirement of the engine.

2.4.2.1 PostgreSQL & PostGIS

PostgreSQL is an open source database server which is often used for geospatial data. This is due to it having the specialist GIS extension PostGIS which adds support for geometric and geographic data.

PostGIS implements the Simple Features from the Open Geospatial Consortium

Inc. (2010). The Simple Features specification is an ISO standard for the implementation of geospatial data on a two dimensional plane. The format of Simple Features can be in one of two formats: a text format named Well-Known Text (WKT) or a binary format named Well-Known Binary (WKB). The information contained in these formats revolve around a set of geometric classes including the **Point** class, which is a single dimensionless location on a two dimensional grid (e.g. X and Y or Longitude and Latitude). Other specified classes include **Line** and **Polygon**. These are used to specify a set of points to create a line or path and a set of points which create a two dimensional shape respectively.

In PostGIS, geometry can be used in an SQL query using the `ST_GeomFromText()` function with the parameters containing WKT as a string as illustrated in figure 2.1.

```
ST_GeomFromText('POINT(1 2)')  
ST_GeomFromText('LINESTRING(2 3, 6 9)')
```

Figure 2.1: Examples of the `GeomFromText` function containing WKT

PostGIS also implements many other spatial functions, many are not applicable to this project. One functionality, however, that is relevant is `ST_Distance` which is used for distance calculations. This calculation defaults to using a spherical surface which means that it is suitable for calculating distances across the planet.

Another applicable feature is the `~` and `&&` operators which are used to test if one geometry contains or intersects another respectively. This is useful with the

`ST_Envelope` function which is supplied with minimum and maximum coordinates for each axis to create a bounding box as shown in figure 2.2.

```
ST_Envelope('POLYGON(0 0, 0 6, 3 6, 3 0, 0 0)') &&  
GeomFromText('POINT(3 5)')
```

Figure 2.2: The Envelope function

2.4.2.2 MySQL

MySQL (like PostgreSQL) is an open source DBMS and forms part of the common LAMP stack (Linux, Apache, MySQL, PHP/Perl/Python) and is the most commonly used database for web projects (Oracle, 2008). The social media projects Wordpress and MediaWiki both use MySQL by default. MySQL has a spatial extension which adds support for Simple Features including WKT and WKB however has its limitations.

Spatial indexing is not supported by the InnoDB storage engine which as of MySQL 5.5 is the default storage engine used by MySQL (Oracle Corporation, 2013). In order to use spatial indexing the MyISAM storage engine must be used. MyISAM has its drawbacks as it does not support features like foreign keys or row level locking (which should increase reliability).

MySQL's spatial classes and functions conform to the naming standard, however, unlike PostGIS they drop the optional `ST_` prefix. This means that `ST_GeomFromText()` becomes simply `GeomFromText()`. Other than this the creation of geometries is

the same as PostGIS.

Operators are also different. Instead of using the `&&` or `~` operators, MySQL implements its own functions `MBRIntersects(geom1, geom2)` and `MBRContains(geom1, geom2)`.

Another drawback of MySQL is it does not implement any distance calculations. This means that the calculation must either be implemented manually in the query (which may be very slow as it is not optimised) or calculated post-retrieval in the application layer.

2.4.2.3 MongoDB

Unlike the previous two DBMSs MongoDB is a NoSQL document-oriented database. This means there are no predefined schemas and the data is stored in documents. MongoDB uses its own form of JSON (JavaScript Object Notation) for data representation called BSON (standing for Binary JSON) (MongoDB Inc., 2013). This means that the database would be unfamiliar to many developers who want to implement the engine into their application. Additionally, it has considerably lower usage than the others so is less likely to be installed on a server.

MongoDB has geospatial functionality built in. As of version 2.4 MongoDB uses the GeoJSON format for storing and representing geometry. This GeoJSON standard contains similar data to the Simple Features specification, however, it is represented differently. Generally, it is a JSON object containing the two fields

“type”, which contains the data about what type of geometry it is representing and “coordinates” which contains arrays of coordinates which the geometry comprises of. This is illustrated in figure 2.3.

```
{
  "type" : "Point",
  "coordinates" : [0, 4]
}
{
  "type" : "LineString",
  "coordinates" : [[2, 0], [3, 6]]
}
```

Figure 2.3: An example of two GeoJSON objects

Geospatial searching in MongoDB is very efficient as it is possible to perform a “near” search, which will return the n closest results to a location in distance order (nearest to furthest). This is achieved by creating a grid containing all entries and searching further and further away around the location for until it reaches its limit (Rethans, 2014). This is the most efficient way of searching if it is required to find the closest entries without limiting the distance.

2.4.2.4 Database Summary

Overall, any of these databases could be used for a location-based social media as they all handle geospatial data types and each have their benefits. MongoDB has the best geospatial searching techniques, but has the lowest adoption which will limit the engine’s deployment ability and therefore makes the engine less

generic. PostgreSQL is an SQL database which has higher adoption than MongoDB and also has some quite advanced geospatial handling. Finally MySQL has considerably higher adoption than either of the previous two databases, and is very common in the LAMP (Linux, Apache, MySQL, PHP/Perl/Python) server stack. Although it does handle geospatial data, it does not, however, have the level of functionality that the other two databases have.

2.5 Application Design

When designing a generic engine, it is important to have a project structure which is easily editable and maintained by others. This can be maximised by following standard well known programming paradigms to aid the future development, expansion and maintainability of the engine. These are known as design patterns.

A design pattern is a standard approach which can be used to solve a variety of problems in software engineering. For a generic engine which can be used (and potentially developed by) by many users, it is important that these are used to allow ease of customisation and understanding of the software. This section includes an overview of the most applicable design patterns for this research project.

2.5.1 The Model View Controller Pattern

The design pattern Model, View, Controller (or MVC) is one which organises your program into three different sections. The model contains the “business logic”. This means that it contains the bulk of the decision making code which makes the application work. This code is organised into classes to keep code self contained and easily maintainable. The view is the interface that a user (or another external program) interacts with. This could be in the form of a web page (for a user to interact with through a web browser), or an API for another application to interact with (e.g. A smartphone application). The controller takes the inputted data (by the user or other application) and passes it to the model to allow it to be processed (Leff & Rayfield, 2001).

The separation of the business logic from the display and the handling allows for better project structure, easier maintenance and more scalable applications. This is incredibly important when a developer is building an application around the core location-based social media engine.

2.5.2 Adapter Pattern

The Adapter pattern is a structural pattern. The adapter pattern gives the ability to change how an object is interacted with. This is useful if the object has features which are not expected by the rest of the application. It means that you can transform an individual class into something that conforms with the rest

of the application without actually having to change the code or functionality of this new class (Noble, 1998). This could be very useful when working with multiple methods of retrieving a location or data which is contained, posts/articles which contain different types of content and added functionality by an application developer using the location-based social media engine.

2.5.3 Registry Pattern

The registry pattern allows a single, central place for data or instances of objects to be stored. A registry is often a static class which has store and retrieve methods (or equivalent of). This is often used to overcome the need for global variables, which are considered bad practice, and allows a more organised storage of data which needs to be accessed throughout the application. Within a location-based social media engine it could be useful to store all the different post types in a registry, to allow them to be added and retrieved easily, along with the application configuration and permissions system.

2.5.4 Software Development Practices

Along with the design patterns, there are further software development techniques (Perks, 2006) which aid development and ensure software meets the requirements of the specification.

Frameworks are used in software development as they provide pre-written code which often performs common functions and contains a predefined organised project structure. This predefined structure aids consistency within and between projects as well as encouraging good development practices. Furthermore, the pre-written code decreases the length of time required to write software therefore increasing productivity and efficiency of a development team (Mnkandla, 2009).

The software needs to be tested once it has been written. Unit tests are a method of automatically testing code; they work on the principle of assertions which check that the outputs of methods in objects are consistent with expected values. If the outputted values are consistent with the assertions, then the code is deemed to have passed the unit test. Traditionally tests are written after the code, however, there is also an agile software development methodology called “Test Driven Development”. With Test Driven Development (or TDD), development starts with the writing of the tests and then code is written which should pass all the tests. The advantages of this are that it encourages developers to write testable code and the tests themselves also act as a design specification for the functionality of the code (Maximilien & Williams, 2003).

Unit tests are useful for testing individual classes, but less useful for testing the overall functionality of web applications. Historically this was achieved through user testing. However, Selenium WebDriver and IDE are tools which allow a developer to automate this. Selenium effectively takes control of a web browser, and follows a set of instructions (which can include the clicking of links and putting text in text boxes) and assertions can be made from page content to make sure the website has met the functional requirements. By controlling the

web browser, it also tests the browser engine's ability to render the page and execute any client side code, which is becoming increasingly important in modern web applications (Holmes & Kellogg, 2006).

2.6 Rationalisation

With the exception of LocalWiki, these existing open source social media platforms are becoming increasingly location aware but are not location orientated. The rest of the location-based social media services are proprietary systems built for the specific applications. The consequence of this is that there is little available in terms of open source social media platforms. With this being the case, there is a need for a generic location based platform which will allow a cross-section of social media applications to be developed on top of.

This platform should have the common features across the social media platforms built in, and allow a developer to customise the differences. The core features of social media seem to be: identity (through authentication), the ability to create content and the ability to feedback on the content. There are differences, however, around these from different social media types. These are compared in table 2.1, which focuses on the identity (which is common across all platforms), the content which is created, and the ability for social interaction around the content.

Feature	Blog	Wiki	Content-Sharing Site
Identity	All social media types have a similar way of identifying users. The authentication method is by allowing users to register and then login with a username and password to identify themselves. The identity of the user and the role they have states the permissions for what they can and cannot do.		
Content	A blog post normally consists of text and HTML. These posts can generally only be created and edited by users who have the permissions (normally the blog owners).	A wiki article is made up of text and a special non-standardised markup called “wiki markup”. These articles can generally be created and edited by anyone, sometimes even those who are not registered.	A content sharing site normally allows any registered user to create content and edit their own content. The content itself is generally a specific type of media such as image, video or audio.
Social Interaction	A blog often allows other users (can be either registered users or anonymous users) to post comments about the blog post. This creates a threaded conversation about the subject matter of the blog post.	A wiki site often has a “talk” section which allows people to discuss potential edits and content of the main article.	A content sharing site has an ability for people to comment on the content created.

Table 2.1: Feature Comparison

The requirements for a generic location-based social media engine will be based upon the aforementioned findings and will be discussed in chapter 3.

Chapter 3

Requirements Analysis

In order to realise the goal of developing a generic engine for location-based social media applications as defined in chapter 1 which can realise the application types identified in chapter 2, the engine must clearly be able to power a location based blog, wiki or content sharing site, and run on the widest possible number of platforms. Meanwhile the engine itself should be well structured and maintainable. This chapter will discuss the decisions which need to be made regarding the platform, architectural design and the feature requirements.

3.1 Platform

As stated in section 2.4, there are a number of platforms which could be used for developing this engine. The platform should be chosen to allow this engine to be deployed on the widest possible platforms. The platforms discussed are the language the engine is written in (and consequently the interpreter required on the server) and the data storage.

3.1.1 Language

The language chosen is PHP due to its ability to run all common Operating Systems and web servers. PHP is also the most common web-programming language so it should be familiar to the widest possible number of developers who will be using, deploying and modifying the application which facilitates the widespread adoption amongst developers.

3.1.2 Data Storage

The database is a key aspect, and as all databases handle queries differently from each other, a decision needs to be made regarding which database this application is going to use. As discussed in chapter two, MySQL is the most common database for a web server, but has limited geospatial functionality. On the other-hand PostgreSQL with PostGIS or MongoDB have very good geospatial

functionality, but are not as widely available. For this reason, MySQL is chosen to enable this engine to stay as “generic” as possible by enabling it to be deployed onto the widest number of web servers available.

This has an impact on the application design, as it means the geospatial calculations must be done application side to increase efficiency rather than in the database queries.

3.2 Architecture

The architecture of the engine is key to aid its implementation into applications and future development of the engine. This section will cover how the engine should be structured to enable it to be as flexible and maintainable as possible.

3.2.1 Engine Core

All location and distance data should be stored and calculated within the application. This could be handled by a third party service (such as Google Maps APIs) which would ease the development of the engine. This will allow the application to be standalone and reduce the requirement for third party subscriptions which, if the engine was implemented in a popular enough application, would require a subscription. The content should be searched for based on a geographic location and returned in the order of closest to furthest. Finally, as identity is key to all

social media types, the application core should also handle user registration and authentication.

The engine core should not need to be altered or edited by a developer who is using it to implement their own application. As this will need to be maintained and developed however, it should still be organised in a well structured manner.

3.2.2 Permissions

It makes sense that the design allows the permissions to be configured on a per-application basis. This means that there should be a simple way an administrator can alter the permissions. This could be using web interface, but could also be a configuration file which contains the permissions in a simple, easily readable, understandable and editable format.

3.2.3 Content

The content can be anything from a simple block of text, to a mixture of multiple media types (including text, images, videos, external content etc.) and any combinations of the above. For example, a image-sharing site may just want a title and an uploaded image, whereas a restaurant review site will probably want information such as; name, review, image and rating. On top of this, some applications may want multiple types of content within the same application. For

this reason, how the content is inserted, validated and retrieved will be different depending on the application.

To enable this, the engine will be designed in a modular system where modules (called “post types”) can be plugged in to the engine to allow different types of content to be created. Modules should register themselves with the application so no extra configuration is required. This will allow non-technical administrators to simply drop in a post type module from a repository of pre-created post types. It will also allow developers to create their own post types with minimal amount of coding. This could be enhanced by providing abstract classes which a developer extends to create their own post types.

3.2.4 Framework

To allow these variations, the engine needs to be developed in a consistent, well structured manner. To allow for ease of future development the engine should use an industry standard framework. This means that any other developer would be able to modify the engine with relative ease and give a pre-defined structure for the application to be built around. When this research began the most common framework for PHP is Zend Framework. This provides a consistent structure and aids the ability for this to be cross platform. It requires a minimum of PHP version 5.2 and can run on Linux, Windows and Mac OS X.

3.3 Functional Requirements

This engine should be able to power a blog, wiki or content sharing site which is location-oriented. This means that the way articles are searched and sorted are based on a geographic location and distance from that location rather than by key phrase searching or date order. There needs to be the ability to create, edit and delete content together with the ability to create and delete comments relating to the content. Who is able to do this varies between the social media types as discussed in table 2.1. The three levels of user (known as “roles”) will be “anonymous” (an unregistered or unauthenticated user), “user” (an authenticated member of the site) and “moderator” (a user that has special moderation privileges). On top of this there should also be an “owner” privilege, which is the specific user who created the content. Table 3.1 lists the abilities of each of these levels for each social media type.

Function	Blog	Wiki	Content-Sharing
User Management			
Registration	✓	✓	✓
Authentication	✓	✓	✓
Content Searching			
Search around Location	✓	✓	✓
Post Actions			
Anonymous Read	✓	✓	✓
Anonymous Create	X	✓	X
Anonymous Edit	X	✓	X
Anonymous Delete	X	✓	X
User Read	✓	✓	✓
User Create	✓	✓	✓
User Edit	X	✓	X
User Delete	X	✓	X
Owner Read	✓	✓	✓
Owner Edit	✓	✓	✓
Owner Delete	✓	✓	✓
Moderator Read	✓	✓	✓
Moderator Create	✓	✓	✓
Moderator Edit	✓	✓	✓
Moderator Delete	✓	✓	✓
Comment Actions			
Anonymous Read	✓	✓	✓
Anonymous Create	✓	✓	X
Anonymous Delete	X	X	X
User Read	✓	✓	✓
User Create	✓	✓	✓
User Delete	X	X	X
Owner Read	✓	✓	✓
Owner Delete	✓	✓	✓
Moderator Read	✓	✓	✓
Moderator Create	✓	✓	✓
Moderator Delete	✓	✓	✓

Table 3.1: Requirements Specification

In addition to these roles there will be a third level which is “admin”. This is not an official role but will allow an admin user to bypass the privilege system.

3.4 Requirements Overview

To make sure this engine meets the requirements of being generic, the following decisions have been made. It should run on the widest possible number of platforms within reason, which is the reason for the decisions in section 3.1. The architecture of the system should be well structure to allow collaborative development and future expansion. It should also allow the post types to be created and included with no modification of the engine core and the permissions should be easily configurable through configuration files.

The end product should be tested compared to the functional requirements as specified in table 3.1. It must have the core functionality of a location-based social media including geospatial searching, content creation, edition and deletion as specified along with the ability to control the functionality depending on the location-based social media application which the engine is powering. The implementation and evaluation of an engine designed to meet this requirement is presented in chapters 4 and 5.

Chapter 4

Implementation

Figure 4.1 shows the overall design of the system. This chapter will start with an overview of an individual post type, in which the classes inherit from the post type abstracts in the library. This will be followed by a look at the configuration of the engine, then the location library for the geospatial representation and calculations the application core and configurations, and finally a look at the application core which ties all these together. This will not look at the Zend Framework directory as this is an unmodified version of Zend Framework.¹

In order to satisfy the core requirements identified in chapter 3, the design will follow the standard Zend Framework project structure: all core application code is inside the `applications/` directory, all files which should be exposed to the web are in the `public` directory and all library classes used are stored in the `library/`

¹Information on this version of Zend Framework can be found here: <http://framework.zend.com/manual/1.12/en/manual.html>

directory (labelled “Library” in figure 4.1). Inside the `application/` (labelled “Core” in figure 4.1) directory are a number of other directories which follow the MVC (Model-View-Controller) design pattern as discussed in section 2.5, however, on top of this there are the configuration files in `application/configs` and the modules directory in `application/modules` where modules can be dropped in and removed with ease. For this reason, the modules directory is going to be used for the post types, and any number of post types can be used at any one time.

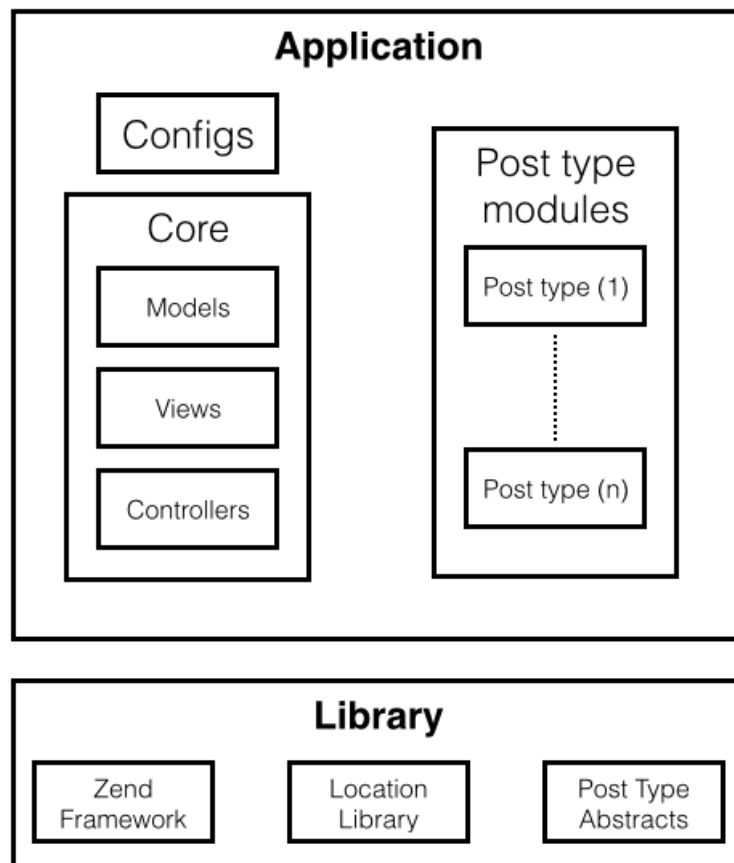


Figure 4.1: Overview of Application Structure

4.1 Modular Post Types

To allow the post types to have ultimate flexibility regarding post content, storage location and retrieval, each post type is developed as a separate Zend Framework module and is put in the `application/modules` directory. Each module will correspond to an individual post type and there can be an unlimited number of post types in use at any time. The only stipulation is that each post type needs a unique name. The file structure of an example post type is shown in figure 4.2.

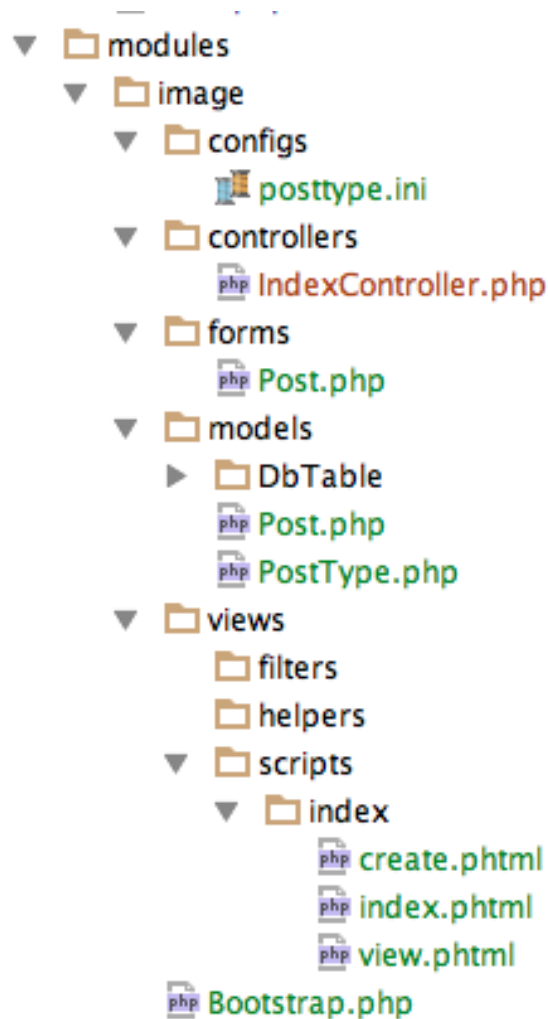


Figure 4.2: Post Type Module Structure

The structure follows a standard Zend Framework module MVC structure.

4.1.1 Required Classes

There are a number of classes which are required, and each of these classes extends abstract classes which are in the `library/My/PostModule` directory. These provide the common functionality required across all post types. In figure 4.3, the block highlighted in green is where all these abstract classes are located.

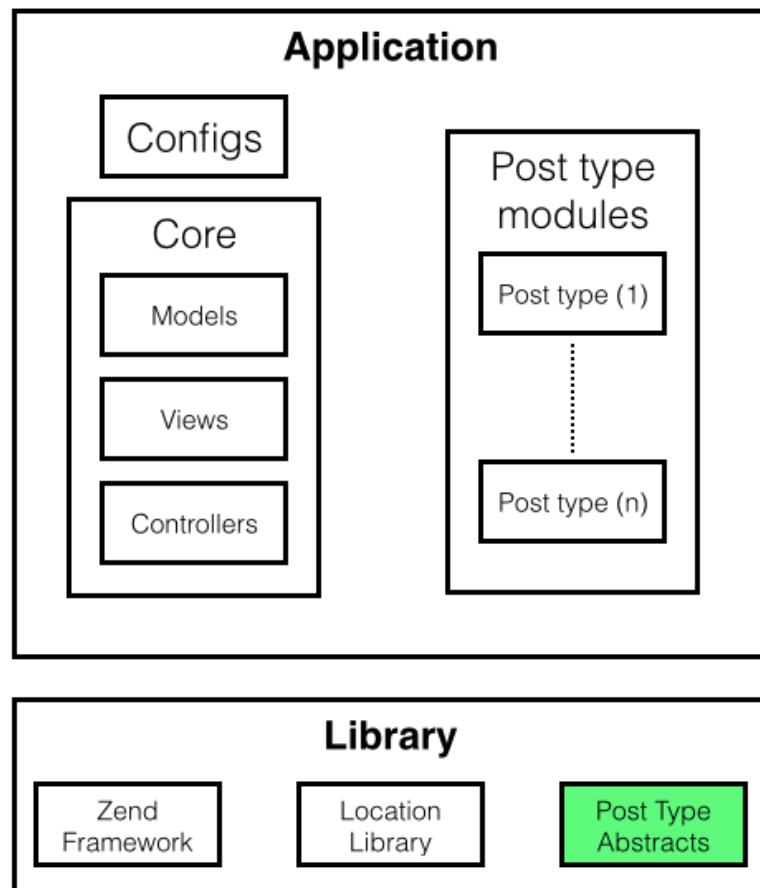


Figure 4.3: Overview of Application Structure (with the Post Abstracts location highlighted)

4.1.1.1 Post Class

The Post model in the individual post type extends the `My_PostModule_PostAbstract` class found in the post abstracts library. The only method that is required to be implemented is a `getTitle` method to retrieve the title of the post. This is because this is used when listing post search results, whereas the summary and the full post is rendered by the “PostType” class. The setting of the `_postType` property is also required which must be the name of the post type. The abstract class contains methods to handle the setting and retrieval of the individual post’s author and location (which is part of the application and therefore a developer should not need to write that functionality in themselves).

4.1.1.2 PostType Class

The PostType model needs to extend the `My_PostModule_PostTypeAbstract`. This is the class used to interact with the post module. It contains the properties for storing the post type name (a unique identifier for that post type), the module name (for to allow for the instantiation of classes from the module) and the title (which is the displayed name when creating a new post of that type). There are also two other properties which are used for controlling a post type’s abilities: “readonly” is used to allow a post type be defined as read only. This is particularly useful if the data is being retrieved from an external data source and therefore cannot be edited through this application. Another ability is “editable”, which is set to false if you only want a user to be able to create and delete and post type, but not have the editable functionality.

This class abstract contains two abstract methods which need to be implemented for each post type. The `getPost()` method expects a single post ID as a parameter, and expects the return of either null or a Post object. This has to be implemented by the developer of the post type, as the way the post data is retrieved and the object is constructed will differ between each implementation. This makes the storage and retrieval of the post as flexible as possible. The other abstract method is `renderPost` which takes a Post object as its parameter. This should return a string (probably containing HTML) of the full post rendered for display to the end user. Again, this is because how a post is rendered will be very variable from one implementation to another.

The `getPosts` method (as shown in figure 4.4) is the method that is called to retrieve posts from the location. This is passed the `Zend_Db_Table_Rowset` retrieved from the database query, the list of locations found from the application core and the bounding box. The latter parameter is not used by the default method, but could be useful if a developer created a post type which had an overriding method; particularly if the data source is external.

```

public function getPosts(Zend_Db_Table_Rowset $rowset,
    Array $locations, Location_Mbr $mbr = null)
{
    $postArray = array();
    foreach ($rowset as $row) {
        if ($row->posttype == $this->_name) {
            $post = $this->getPost($row['posts_id']);
            if ($post instanceof My_PostModule_PostAbstract) {
                $post->setLocation($locations[$row['locations_id']]);
                $postArray[] = $post;
            }
        }
    }

    return $postArray;
}

```

Figure 4.4: My_PostModule_PostTypeAbstract::getPosts method

The rowset contains related location IDs, post IDs and post types. The post type name is checked, and if it is the name of the current post type it calls the `getPost` method which is passed the post ID. If a post is returned, it then sets the location of the post using the `setLocation` method (which is defined in the post Abstract class) and appends it to an array of other posts. This array is then returned.

The `register` method in the `postTypeAbstract` is called by the bootstrap. This method uses a registry pattern to store all the post types which are in the application in a single place together. To achieve this `Zend_Registry` is used and an array of post types is stored in the registry key “posttypes”.

Figure 4.5 illustrates this registration process and also shows why the post type’s “name” must be unique, as it acts as the key of an associative array. An exception is thrown if a post type of the same name already exists in the registry.

```
public function register()
{
    $posttypes = Zend_Registry::get('posttypes');
    if (!isset($posttypes[$this->_name])) {
        $posttypes[$this->_name] = $this;
        Zend_Registry::set('posttypes', $posttypes);
    } else {
        throw new Exception('post type already exists');
    }
    return $this;
}
```

Figure 4.5: My_PostModule_PostTypeAbstract::register method

Finally there is a `parseConfig` method which is used for parsing a standard Ini configuration file. This allows the main information about the post type to be stored in an external, easily editable file rather than being hard coded into the PostType class. This is also invoked by the bootstrap.

4.1.1.3 Bootstrap

A developer does not need to write any code for the Bootstrap.php file, they just need to extend the `My_PostType_Bootstrap` class. This class has a single method `_initRegister`. Methods in the bootstrap which start with “_init” are automatically called by Zend Framework, and this is where the PostType class is

initialised and added to the registry.

Figure 4.6 shows this method which uses reflection to retrieve the location for the configuration file, parses it and passes it to the PostType's `parseConfig` method followed by calling its `register` method from figure 4.5.

```
protected function _initRegister()
{
    $namespace = $this->getAppNamespace();
    $className = $namespace . '_Model_PostType';
    $ref = new Zend_Reflection_Class($this);
    $iniFile = dirname($ref->getFileName()) . '/configs/posttype.ini';
    $postType = new $className();
    if (file_exists($iniFile)) {
        $config = new Zend_Config_Ini($iniFile, APPLICATION_ENV);
        $postType->parseConfig($config);
    }
    $postType->register();
}
```

Figure 4.6: My_PostModule_Bootstrap::_initRegister method

4.1.2 Configuration File

The properties of the post type can be configured in a configuration file. This makes it easier to create and alter post types as you can set up the post type without the need to edit any PHP code. Figure 4.7 is an example post type configuration from the `config/posttype.ini` file in the figure 4.2.

```

[production]
posttype.name = "image"
posttype.module = "image"
posttype.title = "Image"
posttype.readonly = false
posttype.editable = true

[development : production]

```

Figure 4.7: An example Post Type configuration file

These configuration parameters correspond directly to the properties of the Post-Type class.

4.1.3 Post Type Controller

As well as specific models required, each post type requires an “IndexController” for which the abstract is called “My_PostControllerAbstract”. This is the default controller of the post type, and implements the ability to create, edit and delete posts of the post type. This abstract requires the implementation of three methods: “create”, “edit” and “delete”.

The abstract’s “createAction” (figure 4.8) method handles the location and authorisation (whether a user is allowed to create a post), and if they are, calls the custom “create” method. This method should handle everything that is required to create a post type (such as rendering of forms, handling of data submitted and insertion into database) and should either return “null” or an instance of the

PostAbstract class. If a post is returned, it is assumed that this is a brand new post has been created and it assigns that post to the location. Finally, it then redirects to the newly created post.

```

public function createAction()
{
    $post = null;
    $this->_isAuthorisedTo('create');

    $postId = $this->create();
    if ($postId != null) {

        $posttype = $this->_getPostType();
        $post = $posttype->getPost($postId);
        if ($post instanceof My_PostModule_PostAbstract) {
            $locationType = $this->getRequest()
                ->getParam('locationType', null);
            $locationVal = $this->getRequest()
                ->getParam('locationValue');
            if($locationType == 'locationId') {
                $location =
                    Application_Model_Location::fromId($locationVal);
            }
            else {
                $location =
                    Application_Model_Location::fromString($locationVal);
            }
            if($location instanceof Application_Model_Location) {
                $location->addPost($post);
                $this->_helper->redirector->gotoRoute(array(
                    'controller' => 'index',
                    'action' => 'post',
                    'postType' => $posttype->getName(),
                    'postId' => $post->_postId,
                    'module' => 'default'
                ),
                    'post',
                    true
                );
            }
        }
    }
}

```

Figure 4.8: The createAction method

The “editAction” does a very similar thing but for editing a post. This is simpler as a post is already allocated to a location, but it still handles the authorisation and redirect once edited. This calls the “edit” method to actually perform the editing of the post. Finally the “deleteAction” also calls the “delete” method after checking the authorisation, however, it does not redirect to the post as it is assumed the post no longer exists once called. The “delete” method should handle the actual deleting of the post.

4.2 Geospatial Library

To increase the flexibility of this engine, it should not rely upon third party services (such as Google Maps) for handling geospatial data and calculations. In order to do these, a geospatial library was written in PHP which was loosely based upon the geospatial data types from the Simple Features specification as discussed in 2.4.2. This library was written to follow Zend Framework naming conventions with each class being prefixed by `Location_`, and could be easily extracted from the location-based social media engine to be used completely standalone in a separate project. Figure 4.9 highlights the location of this library in the project.

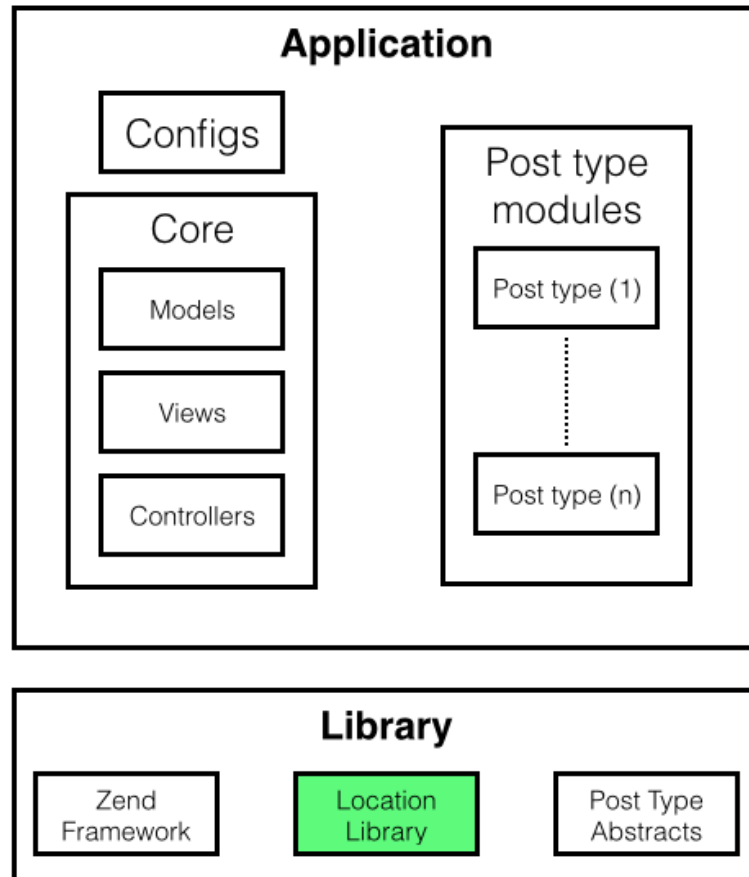


Figure 4.9: Overview of Application Structure (with highlighted geospatial library)

Figure 4.10 illustrates the location library file structure. This library comprises of a collection of classes. The main classes are based upon the Simple Features classes. These are: Point (relating to the Point class), Line and Multipoint Line (relating to the LineString class), and Polygon and Mbr (which are based upon the Polygon class). These classes all implement a `toSQL()` method which will convert the object to a WKT geospatial representation in a string format. This allows for objects to be easily inputted into an SQL database. In addition to these there are some other classes: Earth (for data specific to Earth) and Distance (used

for distance calculations and unit conversion).

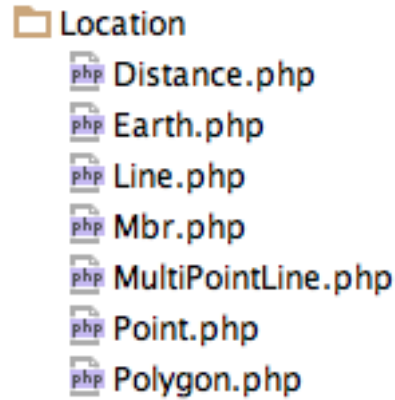


Figure 4.10: Location Library File Structure

4.2.1 Location_Earth Class

From a design perspective, a decision was made to include information specific to Earth in a single class to allow it to be used throughout the library. This class has a single static method `radius`, which takes one optional parameter which is the unit the in which the radius is required (this defaults to “km” if no parameter is passed) and returns the average radius of Earth in the provided unit.

This was developed as a method rather than as static constants as, due to Earth not being entirely spherical, the radius changes depending on the latitude. This means that it could be developed in the future to take a latitude as a parameter, and allow it to return the radius for that location. This would allow for more accurate distance calculations.

4.2.2 Location_Point Class

An instance of the point class is used to represent a single, dimensionless location on the surface of the planet. The constructor takes in the latitude and longitude in decimal format. This class has a number of methods: `thelineTo()` and `bearingTo()` methods take another `Location_Point` object as a parameter and relate to the `Location_Line` class, offering alternative interfaces to creating a new `Location_Line` and `Location_Line::getBearing()` respectively (see subsection 4.2.4 on page 60). The `longitudeToRad()` and `latitudeToRad()` methods convert the values of the latitude and longitude to radians and are generally used by other methods when performing calculations. The `distanceTo()` method also takes another `Location_Point` object and returns an instance of the `Location_Distance` class.

The `getRelativePoint` method takes a numerical distance, initial bearing (in degrees) and unit of the distance (defaults to “km”). To work out the latitude of the second point equation 4.1 is used:

$$\varphi_2 = \arcsin(\sin(\varphi_1) \cdot \cos\left(\frac{d}{R}\right) + \cos(\varphi_1) \cdot \sin\left(\frac{d}{R}\right) \cdot \cos(\theta)) \quad (4.1)$$

The longitude calculation is a little more complicated as the lines of longitude vary in distance depending on the latitude. To determine this, equation 4.2 is used:

$$\lambda_2 = \lambda_1 + \arctan 2(\sin(\theta) \cdot \sin\left(\frac{d}{R}\right) \cdot \cos(\varphi_1), \cos\left(\frac{d}{R}\right) - \sin(\varphi_1) \cdot \sin(\varphi_2)) \quad (4.2)$$

The method implements these two equations and returns a new `Location_Point` based on the results as demonstrated in figure 4.11.

```
$rad = Location_Earth::radius($unit);
$lat1 = $this->latitudeToRad();
$lon1 = $this->longitudeToRad();
$bearing = deg2rad($bearing);

$lat2 = sin($lat1) * cos($distance / $rad) +
        cos($lat1) * sin($distance / $rad) * cos($bearing);
$lat2 = asin($lat2);

$lon2y = sin($bearing) * sin($distance / $rad) * cos($lat1);
$lon2x = cos($distance / $rad) - sin($lat1) * sin($lat2);
$lon2 = $lon1 + atan2($lon2y, $lon2x);
return new Location_Point(rad2deg($lat2), rad2deg($lon2));
```

Figure 4.11: `Location_Point::getRelativePoint` Implementation

4.2.3 Location_Distance Class

The distance calculations were abstracted to their own class to allow ease of unit conversion. Although this class currently can only convert “miles” and “kilometres”, it can be easily expanded to include other units (such as “nautical miles” or “metres”). It is highly unlikely that the user will instantiate this class (although there is nothing stopping them), but it is more likely to be returned from the

`Location_Point::distanceTo()` and `Location_Line::getLength()` methods or used in the `Location_Line::getBearing()` method. The constructor takes two `Location_Point` objects and on construction uses the Haversine formula (equation 4.3) to determine the shortest distance between the two `Location_Point` objects.

$$a = \sin^2 \left(\frac{\Delta\varphi}{2} \right) + \cos(\varphi_1) \cdot \cos(\varphi_2) \cdot \sin^2 \left(\frac{\Delta\lambda}{2} \right)$$

$$\frac{d}{R} = 2 \cdot \arctan 2 \left(\sqrt{a}, \sqrt{1-a} \right) \quad (4.3)$$

This formula was written in JavaScript on moveable-type.co.uk, which was converted into PHP for this project as illustrated in figure 4.12.

```
$distance = sin($this->_distanceLat/2) * sin($this->_distanceLat/2) +
            cos($this->_firstLocation->latitudeToRad()) *
            cos($this->_secondLocation->latitudeToRad()) *
            sin($this->_distanceLong/2) * sin($this->_distanceLong/2);
$distance = 2 * atan2(sqrt($distance), sqrt(1 - $distance));
```

Figure 4.12: Distance calculation implementation

When converting this distance into miles and kilometres it retrieves the radius of Earth from the `Location_Earth` class. In order to add more units of measurement one would simply need to add another unit to the `Location_Earth` and pass the unit as a parameter into the `Location_Distance::to()` method.

Additionally, this class contains a `Location_Distance::getBearing()` method which returns the initial bearing of the line between the two points. This was delegated to this class as it uses some of the same as used in the distance calculation.

4.2.4 Location_Line

The `Location_Line` is a class that represents a one-dimensional, great-circle line between two `Location_Point` objects. The `getLength()` method creates a new instance of `Location_Distance` and passes in the start and end `Location_Point` objects.

This class also has the `getBearing()` which retrieves the initial bearing of the line using the formula described in equation 4.4

$$b = \arctan 2(\sin(\Delta\lambda) \cdot \cos(\varphi_2), \cos(\varphi_1) \cdot \sin(\varphi_2) - \sin(\varphi_1) \cdot \cos(\varphi_2) \cdot \cos(\Delta\lambda)) \quad (4.4)$$

This is implemented in PHP as illustrated in figure 4.13.

```

$y = sin($this->_lonDiff()) * cos($this->_end->latitudeToRad());
$x = cos($this->_start->latitudeToRad()) * sin($this->_end->latitudeToRad())
    - sin($this->_start->latitudeToRad())
    * cos($this->_end->latitudeToRad()) * cos($this->_lonDiff());
$result = atan2($y, $x);

```

Figure 4.13: Get Bearing Implementation

4.2.5 Location_Mbr

MBR stands for Minimum Bound Rectangle. This class is used to generate the bounding box by being given a location (`Location_Point` object) and a radius when performing a search for posts. This will calculate a maximum and minimum latitude and longitude required for performing the search. This MBR can be converted to a `Location_Polygon` object by calling the `toPolygon` method. This is what is used by the SQL query. Matuschek (2013) published the equation for determining the bounding box on a spherical plane, this was implemented in the class method `_setLimits` (figure 4.14).

```

protected function _setLimits()
{
    $north = $this->_point->getRelativePoint($this->_radius,
        '0', $this->_unit);
    $south = $this->_point->getRelativePoint($this->_radius,
        '180', $this->_unit);

    $this->_limits['n'] = $north->lat;
    $this->_limits['s'] = $south->lat;

    $radDist = $this->_radius / Location_Earth::radius($this->_unit);
    $minLat = deg2rad($this->_limits['s']);
    $maxLat = deg2rad($this->_limits['n']);
    $radLon = $this->_point->longitudeToRad();
    if ($minLat > deg2rad(-90) && $maxLat < deg2rad(90)) {
        $deltaLon = asin(sin($radDist) /
            cos($this->_point->latitudeToRad()));
        $minLon = $radLon - $deltaLon;
        if ($minLon < deg2rad(-180)) {
            $minLon += 2 * pi();
        }
        $maxLon = $radLon + $deltaLon;
        if ($maxLon > deg2rad(180)) {
            $maxLon -= 2 * pi();
        }
    }

    $this->_limits['w'] = rad2deg($minLon);
    $this->_limits['e'] = rad2deg($maxLon);
}

```

Figure 4.14: MBR Calculation Implementation

4.3 Application Wide Permissions and Configuration

The configuration directory contains three “ini” files (the location of these files is highlighted in figure 4.15); “application.ini”, “lbsm.ini” and “permissions.ini”. The “application.ini” file contains the necessary configurations for the application such as the classes which are required for the autoloader, namespaces and routing information. This file should not be edited by a developer implementing the system.

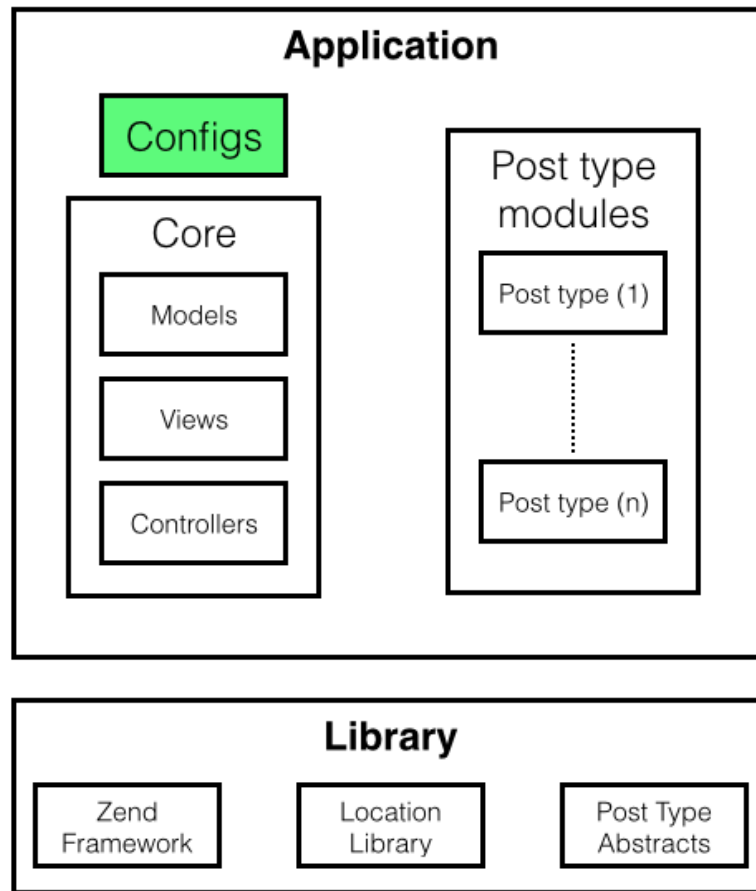


Figure 4.15: Overview of Application Structure with the configuration directory highlighted

The “lbsm.ini” (figure 4.16) file contains the implementation specific configuration. This is a file which a developer implementing the system would need to edit. This is a small file with the following configuration requirements:

```

[production]
; Set the database configuration
resources.db.adapter = "PDO_Mysql"
resources.db.params.username = "username"
resources.db.params.password = "password"
resources.db.params.dbname = "dbname"

; Set the site information
resources.frontController.baseUrl = "http://url.to.site/"
lbsm.title = "LBSM"
lbsm.default.role = member
lbsm.default.unit = "miles"
lbsm.default.radius = 5
lbsm.default.range.min = 0
lbsm.default.range.max = 100
lbsm.default.range.step = 0.1

[development:production]

```

Figure 4.16: lbsm.ini

The first group of configuration is standard database connection parameters. The second is involved with this application. the `resources.frontController.baseUrl` should contain the full URL to the root of the application. This allows the application not to be at the web root, but can be a subdirectory of a domain. The `lbsm.title` sets the title used throughout the application. It is displayed at the top-left of each page. The other `lbsm.default` configurations are all for the default values for the application. The default role is the role which is given to a newly registered user. The unit tells the application to query and display results in either miles or kilometres (“km”). The default radius value determines the radius of the bounding box of the initial query. Finally the range values determine the minimum and maximum radius which can be searched, along with the incrementation which the range slider can select.

The final configuration file is “permissions.ini”. This contains the user levels (known as roles) and the permissions. This uses a resource and privilege system to determine who (role) can perform an action (privilege) on what (resource). Roles can inherit from each other, for example if a standard “member” has a privilege on a resource, “moderator” role can inherit from that role which will also automatically allow these privileges for the moderator user as well.

The top section of figure 4.17 allocates the roles and inheritance. These roles are not fixed and there can be any number of roles. The only roles which are hard-written into this system are the role of “guest” which is the default role for a non-authenticated user and “admin” which will automatically be allowed to do everything. This example defines the role “guest” and does not inherit from any role, the next role is “member”, which inherits all privileges from “guest” and so on.

The next section is specifying the resources and privileges which roles have. In this case the resource is “post” and there are a number of privileges. The top line of this part states that a user of a “member” role is allowed to create a “post”. The third section is the same configuration as the previous one, but for the “comment” resource. Finally, there is a line which will prevent registration and can be used to prevent new users from registering. This was added as an additional feature as it is not required for the requirements specification.

```
[production]
acl.roles.guest = null
acl.roles.member = guest
acl.roles.moderator = member
acl.roles.admin = moderator

acl.resources.post.create.allow = member
acl.resources.post.delete.allow = member
acl.resources.post.edit.allow = moderator
acl.resources.post.read.allow = guest

acl.resources.comment.create.allow = guest
acl.resources.comment.delete.allow = moderator
acl.resources.comment.edit.allow = moderator
acl.resources.comment.read.allow = guest

acl.resources.user.register.allow = guest

[development : production]
```

Figure 4.17: An example “permissions.ini” configuration

4.4 Engine Design

There are multiple aspects to the design of the engine core, and the development of this is done in the main `application` directory (highlighted in figure 4.18).

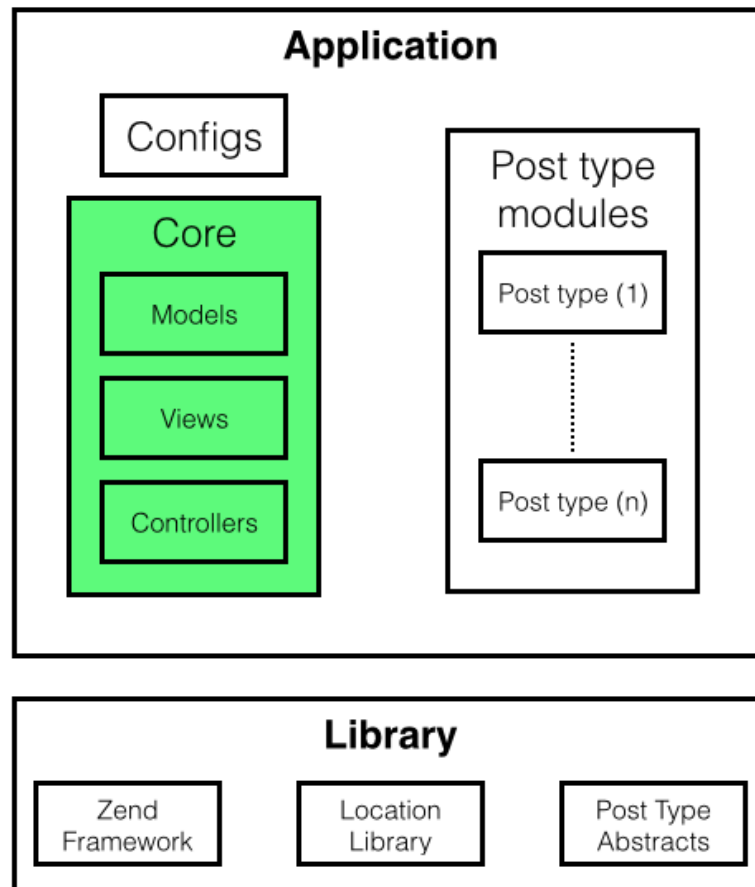


Figure 4.18: Overview of Application Structure with the core engine directory highlighted

4.4.1 User Authentication

This project uses the built in Zend Framework component `Zend_Auth` to perform the user authentication. This is all contained within the `AuthController` which handles the display of forms and comparison with the database. When a user enters their details into a registration form, the form is validated against the following criteria. The username and email must not already exist in the “users”

table, the password and password confirmation must be identical and the email address entered must be a valid email address. If it passes this validation a new row is entered into the table with all the details and the password hashed and salted. In addition, the user's role is entered into the database which is the default role as specified in the configuration.

Once registered a user can authenticate by filling out the login form. The password is hashed with the salt and compared to the hashed password stored in the database row with username.

4.4.2 Geospatial Searching

A key part of this engine's function is the ability to perform searches from location data.

4.4.2.1 Location Retrieval

The location retrieval is achieved by one of the small amounts of JavaScript in application. This uses the HTML 5 geolocation API which attempts to retrieve the web browser's location. As this is an HTML 5 standard, it works the same on any browser with geolocation capability (including mobile browsers).

On the index page (/), there is a large link with the text "What's around me?".

This is the HTML element which the JavaScript manipulates by replacing the “href” attribute’s value with a generated URL based on a pattern which contains placeholders for the latitude and longitude values. The original URL and the pattern URL are generated using the Zend Framework URL generator (figure 4.19(a)), which means if the URL does change, the JavaScript should continue working.

(a)

```
<a id="locate-me" class="btn btn-primary btn-large" href="<?php
    echo $this->url(array(
        'action' => 'index',
        'controller'=>'locate'
    ));
?>" data-geourl="<?php
    echo $this->url(array(
        'action'=>'geo',
        'controller'=>'locate',
        'lat' => ':lat:',
        'lon' => ':lon:'
    ), 'geolocate');
?>">What's around me?</a>
```

(b)

```
<a id="locate-me" class="btn btn-primary btn-large"
href="http://lbsm-wiki.local/locate"
data-geourl="http://lbsm-wiki.local/geo/%3Alat%3A/%3Alon%3A">
What's around me?</a>
```

Figure 4.19: The PHP template followed by the HTML which is generated from the template

The JavaScript (figure 4.20) uses the template from the `data-geourl` data attribute (figure 4.19(b)), which when HTML entities have been parsed has the path

/geo/:lat/:lon:, replaces the :lat: and :lon: placeholders with the actual latitude and longitude respectively. This is done by passing the HTML element (el) URL string (geoString) to a function called `generateUrl`.

```
var generateURL = function(el, geoString) {
    if(navigator.geolocation) {
        var watch = navigator.geolocation.watchPosition(
            function(position) {
                geoString = geoString.replace('%3Alat%3A',
                    position.coords.latitude);
                geoString = geoString.replace('%3Alon%3A',
                    position.coords.longitude);
                el.attr('href', geoString);
            }, null, {
                enableHighAccuracy: true
            });
    }
}
```

Figure 4.20: Generating the geographic URL

Geolocating a web browser is not instant as it has to be permitted by the user, it will then attempt to obtain as accurate a location as possible, but this in itself takes time. This script continues to run and regenerate the URL every time a new or more accurate location is retrieved. If no location is retrieved by the time the link is clicked however, the original URL in the `href` attribute is followed which takes the user to a form. The contents of this form attempts to be auto-populated with the geolocation, however, if the user has not permitted this, they can type their geographic coordinates in manually as a last resort.

Whichever method is used, the end result is the user is redirected to a URL containing a set of geographic coordinates. This URL is for a page which uses

those coordinates to perform the search around that area.

4.4.2.2 Geolocation Storage

The storage of locations was designed using the concepts of a “many to many” relation, but not actually implemented as such. This is to keep the engine generic as it gives the ability for multiple posts to be at the same location, but those posts may be of any different “post type”. Furthermore, a location may not necessarily have a physical or static geographic location. Figure 4.21 shows the design of the two tables.

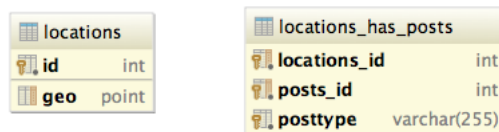


Figure 4.21: Locations and Locations to Posts Table

A row in the “locations” table must have a numeric ID and may have a geolocation which is stored as a Spatial Point data type. The “locations_has_posts” table contains a foreign key to the “locations” table called “locations_id”, and the other two columns denote the post type, and the post ID. These are both required as the posts are implemented by third parties, there is no guarantee that post IDs will be unique across post types.

4.4.2.3 Geospatial Query

The coordinates are retrieved from the URL for the geospatial query. The geospatial query is done through URL parameters rather than a JavaScript geolocation retrieval because this method enables a location to be bookmarked and also manually called (such as by a third party application), thus increasing the flexibility of how this engine can be used. These coordinates are then used to create a bounding box (or Minimum Bound Rectangle) based on either a preset radius or a custom radius also passed in the URL.

The controller instantiates a new instance of the `Application_Model_PostList` class which is then injected with a `Location_Mbr` object (see section 4.2.5 for more information). Once this has been set the method `getPostsByGeoLocation` is called which calls the query, and gets the post IDs from the database.

The PHP method which performs the query is in the `Application_Model_DbTable_Locations` class and is called `getLocationsFromMbr` as show in figure 4.22, which is passed the `Location_Mbr` object.

```

public function getLocationsFromMbr(Location_Mbr $mbr) {
    $polygon = $mbr->toPolygon()->toSql();
    $sql = $this->select()
        ->from($this, array('id',
            'g' => new Zend_Db_Expr('ASTEXT(geo)'))
        ->where('MBRContains(GeomFromText(?), geo) = 1', $polygon);
    $locations = array();

    foreach($this->fetchAll($sql) as $row) {
        $loc = new Application_Model_Location();

        $locations[$row['id']] = $loc->setDatabaseLocation(
            $row['id'], $row['g']
        );
    }

    return $locations;
}

```

Figure 4.22: The getLocationsFromMbr method

This method converts the MBR (which contains the minimum and maximum longitudes and latitudes for a geographic location with a distance radius) to a `Location_Polygon` object which is then converted to a WKT (Well-Known Text) representation. A `Zend_Db_Select` object is used to construct the SQL select statement using the `MBRContains` MySQL function to return all the rows where the “point” is contained within the polygon. The SQL which is generated will be similar to figure 4.23.

```

SELECT id, ATEXT(geo) FROM `locations` WHERE
MBRContains(GeomFromText('POLYGON((
53.499879484892 -2.2972739174845,
53.499879484892 -2.2504200825155,
53.472000515108 -2.2504200825155,
53.472000515108 -2.2972739174845,
53.499879484892 -2.2972739174845))'), geo) = 1

```

Figure 4.23: Example of generated SQL

This query converts each row to `Application_Model_Location` object. This object is used to convert handle change of location formats throughout the application, and contains the geographic location and the location ID (if it has one). This row is then returned and then the related rows in the “locations_has_posts” are retrieved. The application now has a list of locations and posts in those locations.

The post types are retrieved from the registry, and the `getPosts` methods are called from each, with the rowset, the locations array and MBR object being passed. Each of post types return an array of any number of Post objects. These arrays are then merged and stored in the post list object.

4.4.2.4 Sorting

Due to the retrieval being a bounding box query, the order in which they are returned is not the order they should be in (which is distance). The sorting could be achieved in the database query, but this is an incredibly inefficient process as

the database is not optimised to perform complex calculations.

This is all performed in the `Application_Model_PostList::sortByDistance` method (figure 4.24). It uses the PHP `usort` function which implements a merge sort algorithm in which an anonymous function is passed as an argument to provide the search criteria. The criteria provided compares the distances of the posts from the centre of the bounding box. There will be occasions when multiple posts may be identical distances away, particularly as you can add multiple posts to the same location. If this is the case, then it is sorted by alphabetical order from the post's title.

```

public function sortByDistance()
{
    $pl = $this->_mbr->getLocation();
    usort(
        $this->_postArray,
        function ($a, $b) use ($pl) {
            $locationa = $a->getLocation()->getGeo();
            $locationb = $b->getLocation()->getGeo();
            if ($locationa == null) {
                $distancea = 0;
            } else {
                $distancea = $locationa->distanceTo($pl)->toKm();
            }

            if ($locationb == null) {
                $distanceb = 0;
            } else {
                $distanceb = $locationb->distanceTo($pl)->toKm();
            }
            if ($distancea == $distanceb) {
                return strcmp($a->getTitle(), $b->getTitle());
            }
            return ($distancea < $distanceb) ? -1 : 1;
        }
    );

    return $this;
}

```

Figure 4.24: Sort the list of posts by distance method

4.4.3 User Commenting

The action where the post is rendered also contains the functionality to comment on the post. Since how the post is rendered could vary dramatically depending on the post type, the actual rendering of the post is left entirely to the post type

itself. This prints the return string from the “render” method of the appropriate post type.

Below the rendered post type is the list of comments (if there are any and if the user has the right permissions to read the comments) and a form for a user to create a comment (again, if the user has permissions to create a comment). There are one of two forms displayed. If the user has not been authenticated, then a form to type a name, email address and comment is display. If the user is authenticated, however, then it just displays a comment box as the username is used for the name. These comments will act as the discussion functionality for a Wiki and the commenting functionality for a blog or content sharing site as discussed in section 2.3.

The testing and evaluation of the engine as described in this chapter is presented in chapter 5.

Chapter 5

Testing and Evaluation

As this is a “generic engine” which can power a range of location-based social media applications, the testing needs to focus on assessing the functionality of the engine that has been produced. The accuracy of the location library needs to be determined, the ability for the engine to handle multiple post types and have different configurations for different implementations need to be covered.

5.1 Methods of Testing

As this engine has been built by the person who is testing it, the tests need to be automated and repeatable based on the specification and with the author’s

influence on the testing outcome removed. These specifications are the geospatial abilities and the configurability of the engine to be implemented in different applications.

5.1.1 Functional Acceptance Testing

To test the engine itself, it needs to be implemented in applications for the three different social media types identified as: a blog, a wiki and a content sharing site. This then needs to be tested for functionality (using functional testing techniques).

In software development it is common practice to have in-house software testing before it is delivered to the client. As this is a web application, Selenium IDE will be used which is an automated test-suite built into the Firefox web browser. It is used to automate tasks and make assertions based on the tasks performed as a proof of functionality.

Function	Blog	Wiki	Content-Sharing
User Management			
Registration	✓	✓	✓
Authentication	✓	✓	✓
Content Searching			
Search around Location	✓	✓	✓
Post Actions			
Anonymous Read	✓	✓	✓
Anonymous Create	X	✓	X
Anonymous Edit	X	✓	X
Anonymous Delete	X	✓	X
User Read	✓	✓	✓
User Create	✓	✓	✓
User Edit	X	✓	X
User Delete	X	✓	X
Owner Read	✓	✓	✓
Owner Edit	✓	✓	✓
Owner Delete	✓	✓	✓
Moderator Read	✓	✓	✓
Moderator Create	✓	✓	✓
Moderator Edit	✓	✓	✓
Moderator Delete	✓	✓	✓
Comment Actions			
Anonymous Read	✓	✓	✓
Anonymous Create	✓	✓	X
Anonymous Delete	X	X	X
User Read	✓	✓	✓
User Create	✓	✓	✓
User Delete	X	X	X
Owner Read	✓	✓	✓
Owner Delete	✓	✓	✓
Moderator Read	✓	✓	✓
Moderator Create	✓	✓	✓
Moderator Delete	✓	✓	✓

Table 5.1: Requirements Specification

Table 5.1 is a copy of the functional requirements that were originally presented in chapter 3 as table 3.1. The plan is to run a set of identical to table tests which correspond to each row in this table, and then run on each implementation which represents the columns in said table. The result should reflect the ability to perform that task as that user level in each test. Normally, a test suite should pass in its entirety, but in this case the plan is to run the same test suite across all applications and the tests should pass or fail depending on whether that ability is permitted in that application. Another way would be to write individual test suites per implementation, but this would remove the guarantee that the same functionality is tested across them all and hence, not help prove the engine's generality.

5.1.2 Geospatial Library Testing

The geospatial calculations need to be checked for accuracy, so unit tests are most appropriate for this. The expected results will be based upon calculations done prior using an application which is known to be correct. These will then be compared to the results which the geospatial library returns and the test will pass or fail depending on this accuracy.

The unit test coverage will not need to be 100%, but it should at least cover any geospatial equation which has been implemented. Additionally, rounding errors may occur, but it should be accurate to a reasonable resolution, in this case deemed to be three decimal places as this will provide resolution to one metre.

5.2 Testing Environment

Although multiple applications need to be implemented to test this engine functionally, there still needs to be proof that this is the same engine powering all three. Therefore a structure was devised whereby the web base directory would be duplicated three times, as would the configuration directory (one for each application). This would allow different domains to be created for each and allow the inclusion of separate configuration files. The application core, however, would be the same files across all three implementations.

5.2.1 Separate Implementations

The web-base directory is the directory which is exposed by the web server, and in this case is called “public”. This directory will be duplicated and named as “public-blog”, “public-wiki” and “public-share” to represent the blog, wiki and content sharing site respectively. These directories will be the base directory of multiple Apache Web Server vhosts. The domains used for testing will be “lbsm-blog.local”, “lbsm-wiki.local” and “lbsm-share.local” and the Apache configuration is as described in figure 5.1.


```

<VirtualHost *:80>
    ServerAdmin webmaster@dummy-host2.example.com
    DocumentRoot "/Users/rick/Dev/lbsm/public-blog"
    ServerName lbsm-blog.local
    ErrorLog "/private/var/log/apache2/lbsm_error_log"
    CustomLog "/private/var/log/apache2/lbsm_access_log" common
    <Directory "/Users/rick/Dev/lbsm/public-blog">
        AllowOverride All
        Order allow,deny
        Allow from all
    </Directory>
</VirtualHost>
<VirtualHost *:80>
    ServerAdmin webmaster@dummy-host2.example.com
    DocumentRoot "/Users/rick/Dev/lbsm/public-wiki"
    ServerName lbsm-wiki.local
    ErrorLog "/private/var/log/apache2/lbsm_error_log"
    CustomLog "/private/var/log/apache2/lbsm_access_log" common
    <Directory "/Users/rick/Dev/lbsm/public-wiki">
        AllowOverride All
        Order allow,deny
        Allow from all
    </Directory>
</VirtualHost>
<VirtualHost *:80>
    ServerAdmin webmaster@dummy-host2.example.com
    DocumentRoot "/Users/rick/Dev/lbsm/public-share"
    ServerName lbsm-share.local
    ErrorLog "/private/var/log/apache2/lbsm_error_log"
    CustomLog "/private/var/log/apache2/lbsm_access_log" common
    <Directory "/Users/rick/Dev/lbsm/public-share">
        AllowOverride All
        Order allow,deny
        Allow from all
    </Directory>
</VirtualHost>

```

Figure 5.1: Apache Configuration for the three test environments

Apart from a small alteration of the “index.php” file which contains the includes,

these three directories are identical. This alteration allows the inclusion of different configuration files for the different implementations.

The configuration directory inside the application directory is also duplicated, and named “configs-blog”, “configs-wiki” and “configs-share”. The configuration files inside are designed to be altered by the developer who is implementing the engine for their own application. The “application.ini” file is the same across the three directories, the “lasm.ini” file has had a single entry altered which is the `lasm.title` entry. This changes the application title to distinguish the different implementations for the testing.

The main alteration between the configurations is in the “permissions.ini” file which contains the access controls. Each of these have been altered to meet the general configurations of each of these social media types as stated in table 3.1 and 5.1.

5.2.2 Commonality

All three implementations will be using the same application core, database and post type. The reason for using the same application core is to prove that the engine is generic, and it is exactly the same code powering all three implementations. The reason for the same database and post type is to provide consistency for the testing environment. This allows the same tests to be run across all implementations (albeit with different results), but without having to worry about the databases not being in sync.

5.3 Functionality Tests

The following subsections will describe four of the twenty nine test cases written (figure 5.2) in Selenium IDE, corresponding to the twenty nine functional requirements listed in table 5.1, to give an overview of how the tests are run. The test case instructions are a mixture of actions (which selenium must perform such as “type” or “click” a link) and assertions (assert the existence of an HTML element or assert that the text of an element is a specific value). Each instruction is rendered as a row with three columns; the first column is the command (e.g. “clickAndWait”, “type” or “assertElementPresent”), the second column is the selector (this can use element IDs, CSS selector or an Xpath query) and the final column is a an optional column which is only used for certain commands (such as the text to type in or the value of the text of an element). If this final column contains data, but the command does not require it, the text is ignored.

Test Suite
Register
Authentication
Find Posts
Read A Text Post
Create A Text Post
Edit A Text Post
Delete A Text Post
User Read A Text Post
User Create A Text Post
User Edit A Text Post
User Delete A Text Post
Owner Read A Text Post
Owner Edit A Text Post
Owner Delete A Text Post
Moderator Read A Text Post
Moderator Create A Text Post
Moderator Edit A Text Post
Moderator Delete A Text Post
Read Comment
Create Comment
Delete Comment
User Read Comment
User Create Comment
User Delete Comment
Owner Read Comment
Owner Delete Comment
Moderator Read Comment
Moderator Create Comment
Moderator Delete Comment

Figure 5.2: List of Selenium test cases

For the tests to work, there needs to be an initial state for the applications to be in, which need to be reset each time the test suite is run. This contains some initial users and specific roles: “user1” and “user2” are pre-existing users which have the roles of ordinary application “members”, “moderator” is a user with the role of “moderator” for testing, and finally the user “rick” is set as “admin” (which is guaranteed to have permissions for everything).

There is also an initial “Text Post” post type with a known ID (which is used for direct access) and geographic coordinates. This post also has a single comment attached to it to start with. This is used for the search tests, the read tests and for all the comment tests.

5.3.1 Test User Registration

To test the registration system, a test case was written in Selenium IDE as shown in figure 5.3. The two lines (first open and ClickAndWait commands) are used to reset the login. This precedes most test cases as it makes sure that browser is not logged in. After this the site root is opened and the “Register/Login” button is clicked. The script then types a new username, password, password confirmation and email in the registration form then clicks submit (ClickAndWait means the script will wait until the HTTP response has fully loaded).

To check if the registration has been achieved, the same process is then repeated, except this time the test is expected to get an error stating that both the username and email address has already been registered. If this error appears, the test is passed.

Register		
open	/	
clickAndWait	id=login-button	
open	/	
clickAndWait	id=login-button	
type	xpath=("//input[@id='username'])[2]	testuser2
type	xpath=("//input[@id='password'])[2]	testpass24
type	id=confirmPassword	testpass24
type	id=email	test2@ricklab.net
clickAndWait	id=register	
open	/	
clickAndWait	id=login-button	
assertElementNotPresent	//span[contains(text()),"A record matching 'test2@ricklab.net' was found"]	
assertElementNotPresent	//span[contains(text()),"A record matching 'testuser2' was found"]	
type	xpath=("//input[@id='username'])[2]	testuser2
type	xpath=("//input[@id='password'])[2]	testpass24
type	id=confirmPassword	testpass24
type	id=email	test2@ricklab.net
clickAndWait	id=register	
assertElementPresent	//span[contains(text()),"A record matching 'test2@ricklab.net' was found"]	
assertElementPresent	//span[contains(text()),"A record matching 'testuser2' was found"]	

Figure 5.3: The Register Test Case

5.3.2 Testing Geospatial Searching

Figure 5.4 is a simple test case which performs a search of a specific set of geographical coordinates. This set of coordinates is chosen because the place-holder post will be within the search radius (which, as it is not specified, is the default search radius of 5 miles). It asserts that this post is found.

The search radius is then narrowed to a radius known to not contain the place holder post. Another check is then performed to assert that that post does not come up in the results. This is only a crude test as the geospatial calculations are properly tested in the unit tests (as described in section 5.4.1).

Find Posts		
open	/	
clickAndWait	id=login-button	
open	/geo/53.4197467/-2.1864255	
selectWindow	null	
assertElementPresent	link=Comment Test Post	
type	id=radius	3.2
clickAndWait	css=input[type="submit"]	
assertElementNotPresent	link=Comment Test Post	

Figure 5.4: Find Post from Geographic Coordinates

5.3.3 Testing If A User can Edit another User’s Post

Figure 5.5 shows the instruction set for testing if a standard user (with the role “member”) is allowed to edit another user’s post. After the initial authentication reset as described in section 5.3.1, the test case logs in as standard user “user1” and create’s a new “Text Post” post type at a specific location with the title “Member Edit A Text Post”. This is verified as being created and then the script logs out.

The script then logs back in, but this time as “user2”; another standard user which does not have any moderator privileges. The test case then finds the post by the title, and attempts to edit it. An assertion is made that the content has been edited and if it has the test case is passed.

Member Edit A Text Post		
store	Member Edit A Text Post	title
open	/	
clickAndWait	id=login-button	
open	/	
clickAndWait	id=login-button	
type	id=username	user1
type	id=password	user1
clickAndWait	id=login	
assertText	id=login-button	Logout
open	/geo/53.472780699999994/-2.2974819	
assertElementPresent	link=Create here:	
open	/create/location/53.4197467%2C-2.1864255/postText	
type	id=title	\${title}
type	id=content	Testing create text post.
clickAndWait	id=submit	
assertText	css=div.container > h1	\${title}
assertText	css=div.main-content	Testing create text post.
open	/geo/53.4197467/-2.1864255	
assertElementPresent	link=\${title}	
open	/	
assertText	id=login-button	Logout
clickAndWait	id=login-button	
assertText	id=login-button	Register/Login
clickAndWait	id=login-button	
type	id=username	user2
type	id=password	user2
clickAndWait	id=login	
assertText	id=login-button	Logout
open	/geo/53.472780699999994/-2.2974819	
selectWindow	null	
clickAndWait	link=\${title}	
clickAndWait	link=Edit	
type	id=content	Testing create text post. Now edited.
clickAndWait	id=submit	
assertText	css=div.main-content	Testing create text post. Now edited.

Figure 5.5: A standard user editing another user’s text post.

5.3.4 Moderator Delete Comments

The test case in figure 5.6 logs in as a normal user (“user1”) and creates a comment against the place-holder post. Then it is asserted that an element exists on the web page containing the text which was written as the comment. The test case then logs out and logs in as moderator. The same post is visited, and attempts to click the “delete” link which is next to the piece of text written as the comment. Finally it is asserted that the element which contained the text now does not

exist.

Moderator Delete Comment		
clickAndWait	id=login-button	
open	/auth	
type	id=username	user1
type	id=password	user1
clickAndWait	id=login	
open	/display/postText/78	
type	id=comment	Moderator delete comment.
clickAndWait	id=addcomment	
assertElementPresent	//p[contains(text(),"Moderator delete comment.")]	
clickAndWait	id=login-button	
open	/auth	
type	id=username	moderator
type	id=password	moderator
clickAndWait	id=login	
open	/display/postText/78	
clickAndWait	//p[contains(text(),"Moderator delete comment.")]/../p/a[contains(text(),"delete")]	
assertElementNotPresent	//p[contains(text(),"Moderator delete comment.")]	
clickAndWait	id=login-button	

Figure 5.6: A moderator has the ability to delete a comment.

5.3.5 Test Results

Figure 5.7 shows the results from the test suite run on the blog, wiki and content sharing website implementations. The green background indicates that particular test case passed, and red background indicates that it failed. These results need to be compared with table 5.1. The expectations are that where there is a tick (✓) it is expected that the test will pass, but where there is a cross (X) the test is expected to fail. Table 5.2 is a side-by-side comparison of each functionality with the expected and actual results (actual results in green).

Test Suite Results
Register
Authentication
Find Posts
Read A Text Post
Create A Text Post
Edit A Text Post
Delete A Text Post
User Read A Text Post
User Create A Text Post
User Edit A Text Post
User Delete A Text Post
Owner Read A Text Post
Owner Edit A Text Post
Owner Delete A Text Post
Moderator Read A Text Post
Moderator Create A Text Post
Moderator Edit A Text Post
Moderator Delete A Text Post
Read Comment
Create Comment
Delete Comment
User Read Comment
User Create Comment
User Delete Comment
Owner Read Comment
Owner Delete Comment
Moderator Read Comment
Moderator Create Comment
Moderator Delete Comment

Test Suite Results
Register
Authentication
Find Posts
Read A Text Post
Create A Text Post
Edit A Text Post
Delete A Text Post
User Read A Text Post
User Create A Text Post
User Edit A Text Post
User Delete A Text Post
Owner Read A Text Post
Owner Edit A Text Post
Owner Delete A Text Post
Moderator Read A Text Post
Moderator Create A Text Post
Moderator Edit A Text Post
Moderator Delete A Text Post
Read Comment
Create Comment
Delete Comment
User Read Comment
User Create Comment
User Delete Comment
Owner Read Comment
Owner Delete Comment
Moderator Read Comment
Moderator Create Comment
Moderator Delete Comment

Test Suite Results
Register
Authentication
Find Posts
Read A Text Post
Create A Text Post
Edit A Text Post
Delete A Text Post
User Read A Text Post
User Create A Text Post
User Edit A Text Post
User Delete A Text Post
Owner Read A Text Post
Owner Edit A Text Post
Owner Delete A Text Post
Moderator Read A Text Post
Moderator Create A Text Post
Moderator Edit A Text Post
Moderator Delete A Text Post
Read Comment
Create Comment
Delete Comment
User Read Comment
User Create Comment
User Delete Comment
Owner Read Comment
Owner Delete Comment
Moderator Read Comment
Moderator Create Comment
Moderator Delete Comment

(a) Blog Results (b) Wiki Results (c) Content Sharing Results

Figure 5.7: Results of the functionality testing.

Function	Blog		Wiki		Content-Sharing	
User Management						
Registration	✓	✓	✓	✓	✓	✓
Authentication	✓	✓	✓	✓	✓	✓
Content Searching						
Search around Location	✓	✓	✓	✓	✓	✓
Post Actions						
Anonymous Read	✓	✓	✓	✓	✓	✓
Anonymous Create	X	X	✓	✓	X	X
Anonymous Edit	X	X	✓	✓	X	X
Anonymous Delete	X	X	✓	✓	X	X
User Read	✓	✓	✓	✓	✓	✓
User Create	✓	✓	✓	✓	✓	✓
User Edit	X	X	✓	✓	X	X
User Delete	X	X	✓	✓	X	X
Owner Read	✓	✓	✓	✓	✓	✓
Owner Edit	✓	✓	✓	✓	✓	✓
Owner Delete	✓	✓	✓	✓	✓	✓
Moderator Read	✓	✓	✓	✓	✓	✓
Moderator Create	✓	✓	✓	✓	✓	✓
Moderator Edit	✓	✓	✓	✓	✓	✓
Moderator Delete	✓	✓	✓	✓	✓	✓
Comment Actions						
Anonymous Read	✓	✓	✓	✓	✓	✓
Anonymous Create	✓	✓	✓	✓	X	X
Anonymous Delete	X	X	X	X	X	X
User Read	✓	✓	✓	✓	✓	✓
User Create	✓	✓	✓	✓	✓	✓
User Delete	X	X	X	X	X	X
Owner Read	✓	✓	✓	✓	✓	✓
Owner Delete	✓	✓	✓	✓	✓	✓
Moderator Read	✓	✓	✓	✓	✓	✓
Moderator Create	✓	✓	✓	✓	✓	✓
Moderator Delete	✓	✓	✓	✓	✓	✓

Table 5.2: Requirements Specification compared to results

The results as illustrated in table 5.2 show that the functional requirements pass across all three social media types as the test results match the original functional requirements. Since the engine itself has not been altered for each implementation, this demonstrates that the engine is generic enough to power the three social media types.

5.4 Spatial Tests

Another factor of the engine which needs to be tested is the ability to calculate distances correctly and retrieve all results within a bounding box.

5.4.1 Unit Tests on Calculations

Unit tests are an accepted way of testing software (Runeson, 2006). They run specific parts of the software (normally methods in classes), and compare the output of the software to known values (these are called assertions). A test case normally consists of one or more assertions. If all the assertions in a test case are true, then it is considered to have passed the test case. If any single assertion is false, even if all other assertions in the test case are true, it is still considered to have failed the test. In this case, there will be a number of unit tests which correspond to each of the developer-instantiated classes from the “Location” library and each unit test will consist of a test case for each geospatial calculation. The methods that contain the geospatial calculations, and therefore

the ones that need to be tested are as follows:

- `Location_Distance::__construct()`
- `Location_Distance::_trigCalc()`
- `Location_Point::getRelativePoint()`
- `Location_Line::getMidPoint()`
- `Location_Line::getBaring()`
- `Location_Mbr::_setLimits()`

PHPUnit is the software used to run these tests. There is a bootstrap file which sets up the environment for the unit tests which means that the unit tests can use the Zend Framework autoloader. PHPUnit is instructed to use this file by the `phpunit.xml` file, which also instructs PHPUnit to automatically generate an HTML report of code coverage of the unit tests.

The Unit test assertions have been retrieved from Veness (2012) as this is a widely cited source, and would have been corrected if there were errors. The tests themselves are very simple, and just match the actual output with an expected output. Here are two of the tests written.

5.4.1.1 Test for Location_Point::getRelativePoint

Before each unit test, a `setUp()` method instantiates a new `Location_Point` object and saves it to a property of the test class for use in test cases as shown in figure 5.8.

```
protected function setUp()
{
    parent::setUp();
    $this->_point = new Location_Point(53.485722, -2.273644);
}
```

Figure 5.8: Location_PointTestL::setUp()

This point is then used in the test case for getting a relative point.

```
public function testGetRelativePoint()
{
    $relative = $this->_point->getRelativePoint(1, 120, 'km');
    $this->assertInstanceOf('Location_Point', $relative);
    $this->assertEquals(53.481, round($relative->latitude, 3));
    $this->assertEquals(-2.261, round($relative->longitude, 3));
}
```

Figure 5.9: Location_PointTest::testGetRelativePoint

The test in figure 5.9 retrieves a point at a known distance and bearing, and makes three assertions: the first assertion is that the object returned from the method is an instance of the `Location_Point` class and the second and third are asserting the latitude and longitude respectively. This test passes if all three assertions are met.

5.4.1.2 Test for the Location_Distance Class

The `setUp()` method of this test creates a new `Location_Distance` object from two `Location_Point` objects. There are then two tests, one to test the distance returned is as expected in miles, and one to test that the distance returned is as expected in kilometres rounded to three decimal places. The test in figure 5.10 is testing the calculation in kilometres.

```
public function testDistanceInKm()  
{  
    $this->assertEquals(2.276, round($this->_distance->toKm(), 3));  
}
```

Figure 5.10: Testing the Distance calculation in kilometres

5.4.1.3 Unit Test Results

In total there are fifteen test cases with thirty five assertions across them all (due to some tests having multiple assertions as illustrated in figure 5.9), with figure 5.11 displaying the overall code coverage of the Location library. The coverage is far from 100%, however, when the results of the individual classes are analysed, the key methods with the calculations in have full coverage.



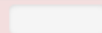



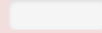



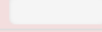



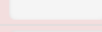



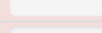

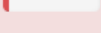
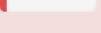
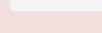

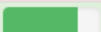

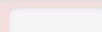



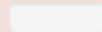
	Code Coverage								
	Lines			Functions and Methods			Classes and Traits		
Total		74.52%	117 / 157		47.73%	21 / 44		0.00%	0 / 7
 Distance.php		84.21%	16 / 19		66.67%	4 / 6		0.00%	0 / 1
 Earth.php		66.67%	2 / 3		0.00%	0 / 1		0.00%	0 / 1
 Line.php		95.00%	19 / 20		83.33%	5 / 6		0.00%	0 / 1
 Mbr.php		97.30%	36 / 37		80.00%	4 / 5		0.00%	0 / 1
 MultiPointLine.php		7.69%	2 / 26		9.09%	1 / 11		0.00%	0 / 1
 Point.php		77.14%	27 / 35		46.15%	6 / 13		0.00%	0 / 1
 Polygon.php		88.24%	15 / 17		50.00%	1 / 2		0.00%	0 / 1

Figure 5.11: Overall code coverage of the Location library

As can be seen in figure 5.12, the `getRelativePoint()` method is covered by the unit test which is the only method in this class which contains a calculation. Any other methods which return calculations call methods from the other classes (such as `bearingTo()`).

	Code Coverage									
	Classes and Traits			Functions and Methods				Lines		
Total	<div></div>	0.00%	0 / 1	<div></div>	46.15%	6 / 13	CRAP	<div></div>	77.14%	27 / 35
Location_Point	<div></div>	0.00%	0 / 1	<div></div>	46.15%	6 / 13	20.45	<div></div>	77.14%	27 / 35
__construct(\$lat, \$long)				<div></div>	0.00%	0 / 1	3.21	<div></div>	71.43%	5 / 7
latitudeToRad()				<div></div>	100.00%	1 / 1	1	<div></div>	100.00%	1 / 1
longitudeToRad()				<div></div>	100.00%	1 / 1	1	<div></div>	100.00%	1 / 1
__toString()				<div></div>	100.00%	1 / 1	1	<div></div>	100.00%	1 / 1
getLatitude()				<div></div>	0.00%	0 / 1	2	<div></div>	0.00%	0 / 1
getLongitude()				<div></div>	100.00%	1 / 1	1	<div></div>	100.00%	1 / 1
distanceTo(Location_Point \$point2)				<div></div>	100.00%	1 / 1	1	<div></div>	100.00%	2 / 2
__get(\$request)				<div></div>	0.00%	0 / 1	3.04	<div></div>	83.33%	5 / 6
getRelativePoint(\$distance, \$bearing, \$unit = 'km')				<div></div>	100.00%	1 / 1	1	<div></div>	100.00%	11 / 11
bearingTo(Location_Point \$point2)				<div></div>	0.00%	0 / 1	2	<div></div>	0.00%	0 / 1
lineTo(Location_Point \$point)				<div></div>	0.00%	0 / 1	2	<div></div>	0.00%	0 / 1
getMbr(\$distance, \$unit = 'km')				<div></div>	0.00%	0 / 1	2	<div></div>	0.00%	0 / 1
toSql()				<div></div>	0.00%	0 / 1	2	<div></div>	0.00%	0 / 1

Figure 5.12: Unit test coverage of the Location_Point class

Figure 5.13 shows the results from the unit tests for the Location_Line class. The two methods in this which contain calculations are the `getMidPoint` and `getBearing` methods. As this figure shows, both those methods have been 100% tested.



















	Code Coverage									
	Classes and Traits			Functions and Methods				Lines		
Total		0.00%	0 / 1		83.33%	5 / 6	CRAP		95.00%	19 / 20
Location_Line		0.00%	0 / 1		83.33%	5 / 6	6		95.00%	19 / 20
__construct(Location_Point \$start, Location_Point \$end)					100.00%	1 / 1	1		100.00%	3 / 3
getLength()					100.00%	1 / 1	1		100.00%	1 / 1
getMidPoint()					100.00%	1 / 1	1		100.00%	8 / 8
getBearing()					100.00%	1 / 1	1		100.00%	6 / 6
_latDiff()					0.00%	0 / 1	2		0.00%	0 / 1
_lonDiff()					100.00%	1 / 1	1		100.00%	1 / 1

Figure 5.13: Unit test coverage of the Location_Line class

The Location_Distance class contains two methods which need testing, the constructor `__construct` and the protected method `_trigCalc`. The `getBearing` method has not been tested, as it calls the `getBearing` from the Location_Line class, so the calculation has already been tested. As figure 5.14 shows, the two core methods have 100% coverage.



















	Code Coverage									
	Classes and Traits			Functions and Methods				Lines		
Total		0.00%	0 / 1		66.67%	4 / 6	CRAP		84.21%	16 / 19
Location_Distance		0.00%	0 / 1		66.67%	4 / 6	7.19		84.21%	16 / 19
__construct(Location_Point \$firstLocation, Location_Point \$secondLocation)					100.00%	1 / 1	1		100.00%	6 / 6
_trigCalc()					100.00%	1 / 1	1		100.00%	5 / 5
getBearing()					0.00%	0 / 1	2		0.00%	0 / 2
toMiles()					100.00%	1 / 1	1		100.00%	1 / 1
toKm()					100.00%	1 / 1	1		100.00%	1 / 1
to(\$unit)					0.00%	0 / 1	2.06		75.00%	3 / 4

Figure 5.14: Unit test coverage of the Location_Distance class

Finally, the testing of the `Location_Mbr` class, which requires the `_setLimits` method to be tested. Figure 5.15 illustrates that this has also 100% coverage of that method. However, this needs to be tested further to make sure that posts from within the bounding box are actually retrieved.

















	Code Coverage									
	Classes and Traits			Functions and Methods				Lines		
Total		0.00%	0 / 1		80.00%	4 / 5	CRAP		97.30%	36 / 37
Location_Mbr		0.00%	0 / 1		80.00%	4 / 5	10		97.30%	36 / 37
<code>__construct(Location_Point \$point, \$radius, \$unit = 'km')</code>					100.00%	1 / 1	1		100.00%	5 / 5
<code>_setLimits()</code>					100.00%	1 / 1	5		100.00%	22 / 22
<code>getLocation()</code>					0.00%	0 / 1	2		0.00%	0 / 1
<code>toPolygon()</code>					100.00%	1 / 1	2		100.00%	8 / 8
<code>__get(\$offset)</code>					100.00%	1 / 1	1		100.00%	1 / 1

Figure 5.15: Unit test coverage of the `Location_Mbr` class

5.4.2 Bounding Box Tests

In the unit tests, the calculation for the `Location_Mbr` class was compared with other MBR implementations, which matched. However, it was tested within the application as well to ensure that the results were correct and therefore would only return posts whose locations were inside the box. A test environment was set up with nine posts posted around various points of the University of Salford campuses, and the distance unit was set to kilometres to increase the granularity of the results.

Figure 5.16 shows the locations of the posts, and the green marker is also where

the geospatial search is being performed from (the centre of the bounding box). The furthest post away is the MediaCityUK post which is 2.11km away (bottom left of figure 5.16), and the second furthest is Castle Irwell (very top of figure 5.16).

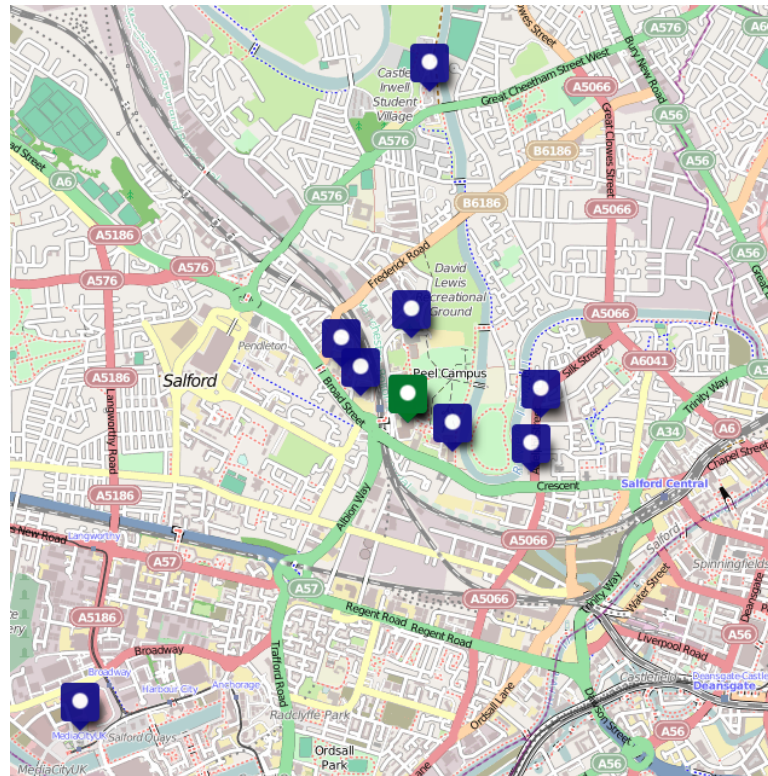


Figure 5.16: Post Locations

The first test performed was set at a 2.2km radius and should encompass all these posts, as shown in figure 5.17.

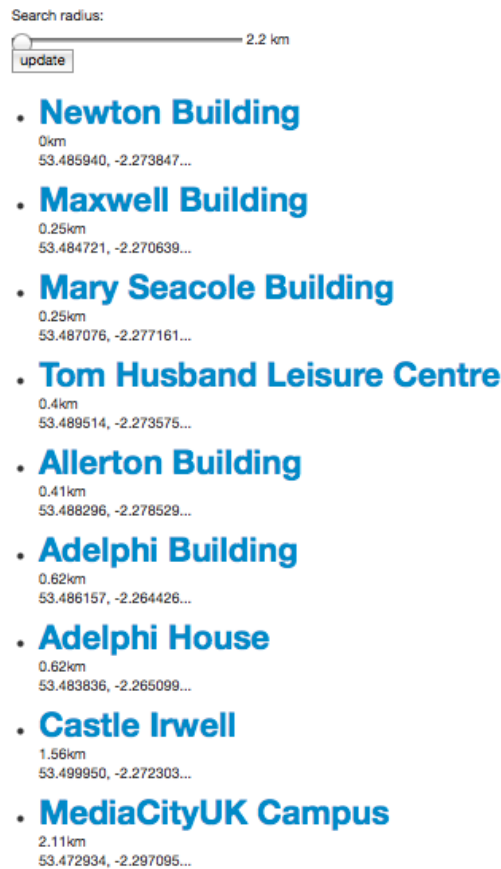


Figure 5.17: Results from 2.2km search

This is the correct result. If the radius is reduced to 2km, then the query is expected to return all posts except MediaCityUK. The actual result is shown in figure 5.18 which it can be seen that the results still include MediaCityUK.

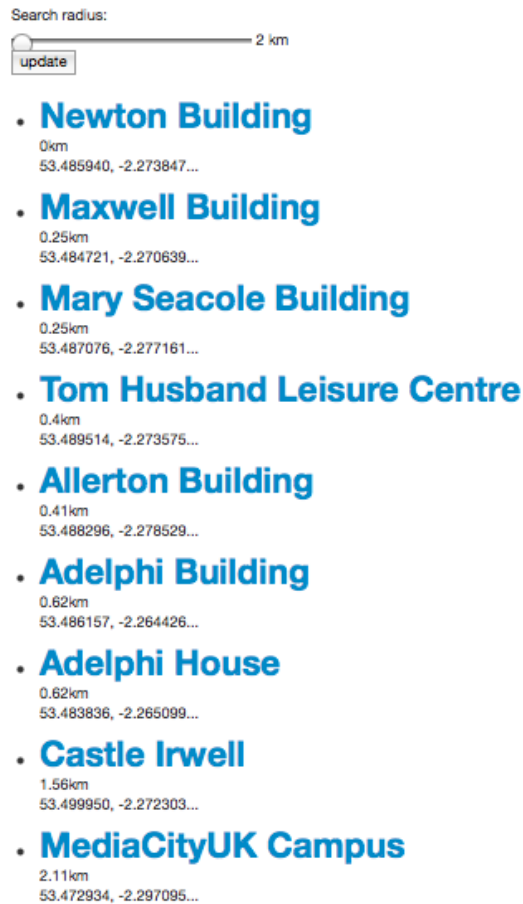


Figure 5.18: Results from 2km search

In fact the query can go as low as 1.55km and still include MediaCityUK as shown in figure 5.19. Interestingly though, the Castle Irwell post is now missing, as that is 1.56km away, which is the correct result.

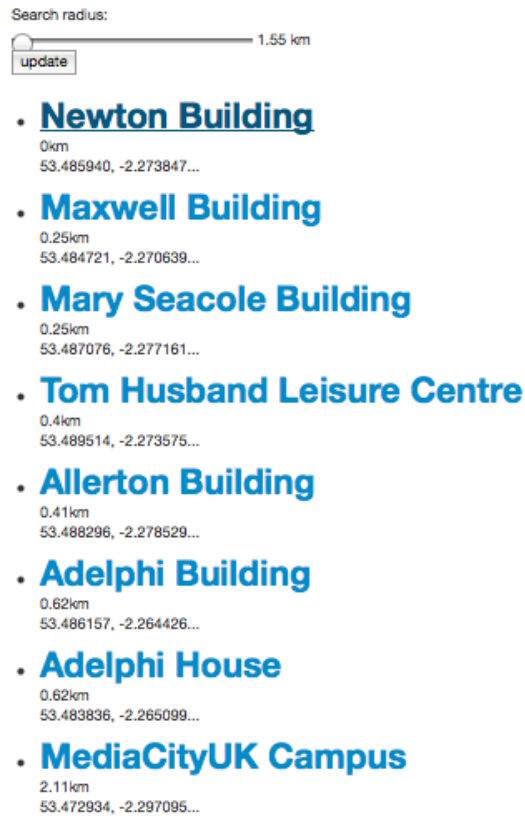


Figure 5.19: Results from 1.55km search

The reason for this is due to the query finding everything inside the bounding box, and although the sides of the box are 1.55km away from the centre, the corners of the box will return posts which are further away. This is shown in figure 5.20.

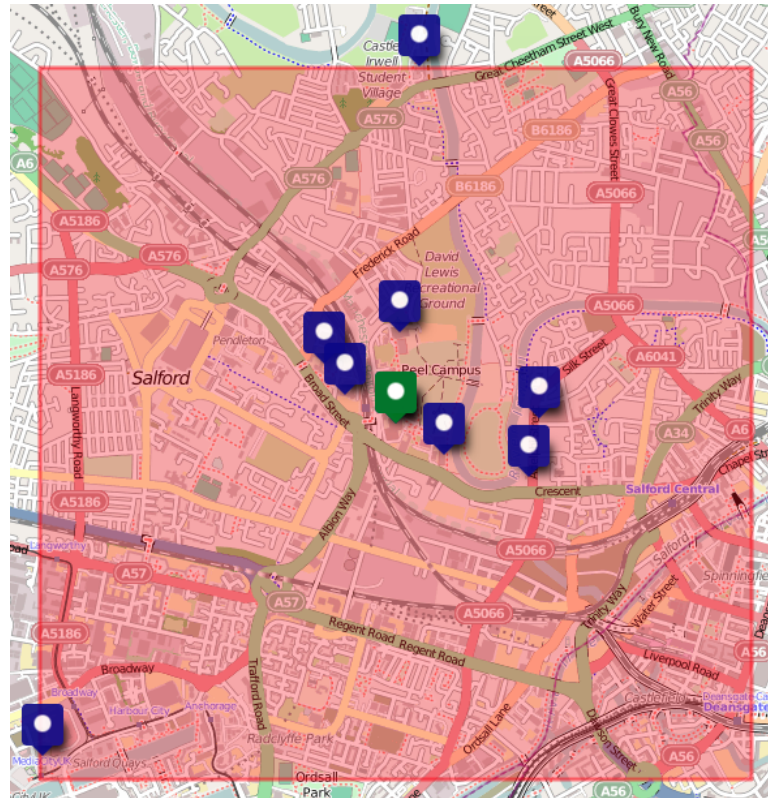


Figure 5.20: The 1.55km bounding box

Finally, the box is reduced to 0.5km, which should definitely exclude MediaCityUK and also exclude Adelphi House and Adelphi Campus as those are both 0.6km from the search location as shown in figure 5.21.

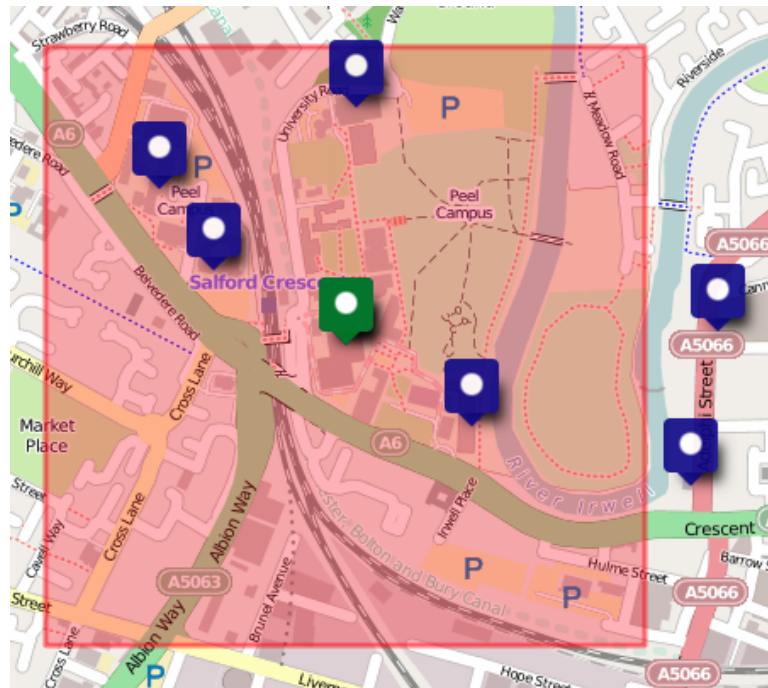


Figure 5.21: The 0.5km bounding box

This is confirmed by the results from the search query as is shown in figure 5.22.

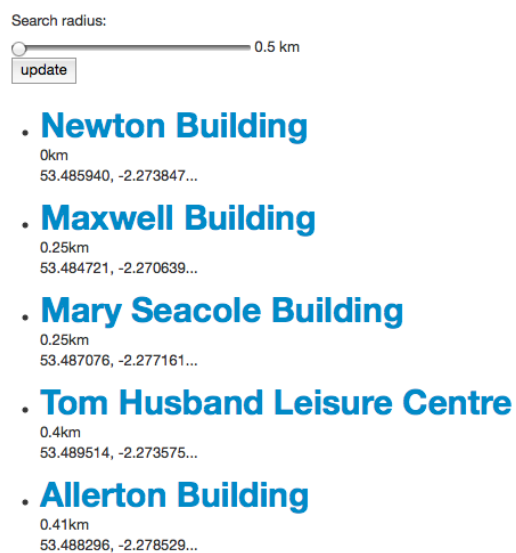


Figure 5.22: Results from 0.5km search

Overall the bounding box query is returning all the results within the box, and therefore the geospatial searching is working as expected.

5.5 Results Evaluation

With a created post type it was possible to create new posts in a specific location, edit and delete the posts along with the ability to add comments to the post. These posts are able to be queried by location and retrieved with the results being ordered by proximity to search location. The ability to control user levels and permissions makes this flexible enough to be implemented in a variety of different social media types.

Due to use of a bounding box SQL query, the geospatial searching prioritises returning all data from within an area over returning too much data (i.e. outliers). This could be solved easily in the application core by checking the distance before the render of the post item in the list, however, the data would still be retrieved from the database query so would not make the query any more efficient. This in itself is not a problem as it is better to return more posts than required than miss out posts which are within the radius. The other alternative is to pre-calculate the distances of all locations before retrieval, however this would be inefficient as this is not a functionality the database is designed to have.

5.5.1 Further Implementation

This engine was used to implement an application as part of the FIRM project, an EPSRC funded project involving the University of Salford, Lancaster University, Goldsmiths and the BBC. In this project there was a web-based video editing application called Storisphere where users could upload video clips and create stories by combining and editing these video clips. The editing is done through a collaborative editing workspace through the web and the videos can then be published for others users to view.

This engine was used as a method of giving these videos geographic locations and being able to search through the geographically tagged videos based on the location. To implement this a new post type was created (named Storisphere), which would interface with the Storisphere API and serve up videos which were not locally stored by the location-based social media application implementation.

The conclusions drawn from this evaluation of the engine, further work required and the future of location-based social media are presented in chapter 6.

Chapter 6

Conclusion & Further Work

The original aim of this project was to create a truly generic engine to power a location-based social media application. Although the engine is not perfect and could now be implemented more easily, it is possible to create multiple location-based social media applications with different abilities and content, without modifying the main code of the engine. This was evidenced by the testing of the three main social media types presented in section 5.3.

The first objective was to identify the characteristics which differentiate the social media types. The conclusion of this was primarily the differing of permissions between levels of users and guests. Another difference which was identified was the differing content which users could post up. This latter difference, however, was not limited to different social media types, but differed between implementations of the same social media type. These were summarised in tables 2.1 (page 32)

and 3.1 (page 40).

The second objective was to create a generic engine which would allow location-centric applications to be implemented of these social media types. This was implemented successfully in chapter 4 and tested, with the results of the tests being shown in 5.

6.1 Technology

Web and mobile technology is a fast moving field. With this project commencing in 2010 and only being worked on part time, this application was created using technology which was out of date by the time it was finished. This section will discuss key technology areas which may be used if the project commenced in 2014. Although the technology has changed, the key architectural concepts from the requirements analysis and the design approach adopted in the implementation are still valid and would still be used with these newer technologies.

6.1.1 Database Backend

MySQL is still the most common open-source platform around but databases such as MongoDB are becoming increasingly supported. MongoDB would be a much better database for geospatial searches as it would allow searching for the

closest as opposed to using a bounding box. This would allow pagination and remove the need for altering the bounding box for searching.

In MongoDB version 2.4 the “2dsphere” index was added (MongoDB Inc., 2014a) to allow distance calculations across a spherical plane (such as Earth). This means that the results could be ordered by distance from search location in the database query by using the `$near` proximity operator (MongoDB Inc., 2014b), rather than this having to be done within the application itself. Also, because MongoDB is schemaless (no tables are needed to be created), it would make the creation of new post types easier as the developer would not need to create new database tables and the queries across all post-types could be done in a single database call.

With cloud based “Database as a Service” hosting companies appearing over recent years (such as mongohq.com), it is possible to use MongoDB for an application even if MongoDB was not provided by the web host. Therefore MongoDB would be a much more sensible choice for the application back-end than MySQL if this application was written today.

6.1.2 Application Framework

When this project was started, Zend Framework was considered the most common and complete PHP framework around. Over recent years, however, a new set of “second generation” frameworks have been developed. These frameworks harness the power of PHP 5.3 or higher, as PHP 5.2 is now unsupported. Examples of

these frameworks include Zend Framework 2 and Symfony2.

These newer frameworks implement dependency injection with a service handler, meaning that the code is much more loosely coupled and any dependencies are automatically injected in from data in configuration files. If this location-based social media engine was implemented with this, it would probably allow much easier implementation of new post types, and potentially allow a completely new post type to be created simply from a configuration file. Additionally, these frameworks have Object Relational Mapping (ORM) for SQL databases or Object Document Mapping (ODM) for NoSQL databases such as MongoDB which would allow for an easier development of database independent code, as the current implementation can only work on MySQL.

6.2 Further Work

For this engine to be released there are a number of areas which should be improved, this section will outline the key areas that could be improved upon both for the location-based social media engine and for its components.

6.2.1 Missing Features

A useful feature to add, particularly for something which is location based, is adding a “nearby” feature to the post types, meaning that when displaying the

post type, it should have a list of other posts which may be of interest. This is very similar to the “see also” feature which some blogging platforms have. Additionally, being able to categorise posts would provide additional functionality and allow people to filter out posts belonging to categories which they were not interested in. These categories should be dynamic with the ability to create new categories on creation of a new post.

The ability to rate comments would also be useful to allow the highest rated comments to be displayed at the top (similar to YouTube). This would be less useful for posts as the search is based on geographic proximity and would not use the ratings to affect the results.

A full administrative dashboard would be useful to allow an admin to manage users, including (but not necessarily limited to): creating a new user, editing users’ details, altering the roles of users and deleting/suspending user accounts. A user password reset would also be useful and potentially other methods of authenticating such as OAuth with another social media platform.

Finally, it would be nice to have an API which would allow external applications to consume it. This would increase the flexibility of the application and allow it to power not just web applications but mobile applications as well as being embedded into other web sites. The API should use a modern and consistent architecture (such as RESTful) and should be easy to use with OAuth 2 for authentication of the client.

This could potentially be a huge piece of software, so the opportunities for expanding the features are endless.

6.2.2 Expansion of the Location Library

The location library was designed to be kept as a separate project to the rest of the location-based social media application. As this library was useful in another project, it has already been forked and uploaded onto GitHub. This version has been updated and requires PHP 5.3 as a minimum as it uses namespaces and conforms to the PSR-0 specifications. This version is also installable through Composer as it is published in the Packagist repository.¹ Additional features have also been added to the Location library, primarily the support of GeoJSON which is a standard data format for storing geospatial data. MongoDB is one of the spatial databases that uses this format. At time of writing this library has been installed thirteen times for use in other applications.

6.2.3 Front End Design

Although this front-end uses Twitter Bootstrap, which was used primarily to create a usable interface out of the box and easily customisable by a developer implementing the engine, the actual design is very minimal. Additionally, the only JavaScript really used is for the HTML 5 geospatial API. Many modern web

¹This is viewable at <http://packagist.org/packages/ricklab/location>

applications are using front-end MVC frameworks such as Backbone or AngularJS. These frameworks give an impression of an application on the web rather than a website, often only consisting of a single HTML page which is altered by the JavaScript.

For this engine, it would make sense to move to this model as it would aid its usability on a small screen such as a smartphone, which for a location-based social media application would make sense.

6.3 The Future of location-based social media

Location-based social media is not going away. As well as mobile devices, new wearable devices are starting to appear on the market. These devices (such as Google Glass and the Pebble smartwatch) are likely to increase the number of location aware applications and ease the interaction with the applications. This means there is likely to be a need for an engine like this, to allow developers to rapidly develop new location-based social media applications to be used with these devices. This engine can also be locked down with the permissions so it could power a non-social location-based application. This would increase variety of location-based applications which could be developed based on this engine and enable more of these types of services to be created.

Bibliography

- Agarwal, A. (2011). How to Find Videos on YouTube by Location. Retrieved October 3, 2013, from <http://www.labnol.org/internet/find-videos-by-location/19742/>
- Alexa. (2013). Wikipedia.org Site Info. Retrieved March 26, 2013, from <http://www.alexa.com/siteinfo/wikipedia.org>
- Burkard, R. K. (1984). THE WORLD GEODETIC SYSTEM. In *Geodesy for the layman* (4th ed., Chap. 8). Retrieved from http://www.ngs.noaa.gov/PUBS%5C_LIB/Geodesy4Layman/TR80003E.HTM%5C#ZZ11
- C2.com. (2013). Wiki History. Retrieved February 27, 2013, from <http://c2.com/cgi/wiki?WikiHistory>
- Charlesworth, A. (2009). The ascent of smartphone. *Engineering & technology*, 4(February 2009), 32–33. Retrieved from <http://digital-library.theiet.org/content/journals/10.1049/et.2009.0306>
- Dennis, M. A. (2013). wiki (Web site) – Encyclopedia Britannica. Retrieved May 17, 2013, from <http://www.britannica.com/EBchecked/topic/1192819/wiki>

- EBizMBA. (2013). Top 15 Most Popular Social Networking Sites. Retrieved October 3, 2013, from <http://www.ebizmba.com/articles/social-networking-websites>
- Google. (2006). Google To Acquire YouTube for \$1.65 Billion in Stock - News announcements - News from Google - Google. Retrieved April 22, 2014, from http://googlepress.blogspot.co.uk/2006/10/google-to-acquire-youtube-for-165%5C_09.html
- Google. (2008). New feature: Geotagging. Retrieved August 21, 2013, from <http://bloggerindraft.blogspot.co.uk/2008/12/new-feature-geotagging.html>
- Google. (2013). YouTube API v2.0 - API Query Parameters - YouTube - Google Developers. Retrieved June 26, 2013, from https://developers.google.com/youtube/2.0/developers%5C_guide%5C_protocol%5C_api%5C_query%5C_parameters
- Holmes, A. & Kellogg, M. (2006). Automating Functional Tests Using Selenium. In *Agile 2006 (agile'06)* (pp. 270–275). IEEE. doi:10.1109/AGILE.2006.19
- Kaasinen, E. (2003, May). User needs for location-aware mobile services. *Personal and Ubiquitous Computing*, 7(1), 70–79. doi:10.1007/s00779-002-0214-7
- Kaplan, A. M. & Haenlein, M. (2010, January). Users of the world, unite! The challenges and opportunities of Social Media. *Business Horizons*, 53(1), 59–68. Retrieved from <http://www.sciencedirect.com/science/article/B6W45-4XFF2S0-1/2/600db1bd6e0c9903c744aaf34b0b12e1>
- Kietzmann, J. H., Hermkens, K., McCarthy, I. P. & Silvestre, B. S. (2011, May). Social media? Get serious! Understanding the functional building blocks of social media. *Business Horizons*, 54(3), 241–251. doi:10.1016/j.bushor.2011.01.005
- Kim, J.-T., Lee, J.-H., Lee, H.-K. & Paik, E.-H. (2010). Design and Implementation of the Location-Based Personalized Social Media Service. In *Internet*

- and web applications and services (iciw), 2010 fifth international conference on* (pp. 116–121). doi:10.1109/ICIW.2010.25
- Leff, A. & Rayfield, J. (2001). Web-application development using the Model/View/Controller design pattern. In *Proceedings fifth ieee international enterprise distributed object computing conference* (pp. 118–127). IEEE Comput. Soc. doi:10.1109/EDOC.2001.950428
- Matuschek, J. P. (2013). Finding Points Within a Distance of a Latitude/Longitude Using Bounding Coordinates. Retrieved March 7, 2014, from <http://janmatuschek.de/LatitudeLongitudeBoundingCoordinates%5C#Longitude>
- Maximilien, E. & Williams, L. (2003). Assessing test-driven development at IBM. In *25th international conference on software engineering, 2003. proceedings.* (pp. 564–569). IEEE. doi:10.1109/ICSE.2003.1201238
- Mnkandla, E. (2009, September). About software engineering frameworks and methodologies. In *Africon 2009* (pp. 1–5). IEEE. doi:10.1109/AFRCON.2009.5308117
- MongoDB Inc. (2013). BSON - Binary JSON. Retrieved September 4, 2013, from <http://bsonspec.org/>
- MongoDB Inc. (2014a). 2dsphere Indexes - MongoDB Manual 2.4.9. Retrieved March 16, 2014, from <http://docs.mongodb.org/manual/core/2dsphere/>
- MongoDB Inc. (2014b). \$NEAR - MONGODB MANUAL 2.4.9. Retrieved March 16, 2014, from http://docs.mongodb.org/manual/reference/operator/query/near/%5C#op.%5C_S%5C_near
- Nardi, B. A., Schiano, D. J., Gumbrecht, M. & Swartz, L. (2004, December). Why we blog. *Communications of the ACM*, 47(12), 41. doi:10.1145/1035134.1035163
- Noble, J. (1998). Classifying relationships between object-oriented design patterns. In *Proceedings 1998 australian software engineering conference (cat.*

- no.98ex233*) (pp. 98–107). IEEE Comput. Soc. doi:10.1109/ASWEC.1998.730917
- Open Geospatial Consortium Inc. (2010). OpenGIS Implementation Standard for Geographic information - Simple feature access - Part 2 : SQL option.
- Oracle. (2008). MySQL :: Market Share. Retrieved September 2, 2013, from <http://www.mysql.com/why-mysql/marketshare/>
- Oracle Corporation. (2013). MySQL :: MySQL 5.5 Reference Manual :: 14.3 The InnoDB Storage Engine. Retrieved September 4, 2013, from <https://dev.mysql.com/doc/refman/5.5/en/innodb-storage-engine.html>
- Ordnance Survey. (2013). Surveying guidelines | Business and government | Ordnance Survey. Retrieved September 4, 2013, from <http://www.ordnancesurvey.co.uk/business-and-government/help-and-support/navigation-technology/os-net/surveying.html>
- O'Reilly, T. (2005). What Is Web 2.0 - O'Reilly Media. Retrieved March 26, 2013, from <http://oreilly.com/web2/archive/what-is-web-20.html>
- Perks, M. (2006, August). Best practices for software development projects. IBM. Retrieved from http://www.ibm.com/developerworks/websphere/library/techarticles/0306%5C_perks/perks2.html
- Rethans, D. (2014). Finding a Pub with MongoDB and OpenStreetMap. Retrieved April 11, 2014, from <http://derickrethans.nl/talks/mongo-osm-conf0014.pdf>
- Runeson, P. (2006, July). A survey of unit testing practices. *IEEE Software*, 23(4), 22–29. doi:10.1109/MS.2006.91
- Sagar, I. (2012). Before iPhone and Android Came Simon, the First Smartphone. Retrieved August 19, 2013, from <http://www.businessweek.com/articles/2012-06-29/before-iphone-and-android-came-simon-the-first-smartphone>

- Schonfeld, E. (2011). Gowalla Versus Foursquare: Why Pretty Doesn't Always Win | TechCrunch. Retrieved June 27, 2013, from <http://techcrunch.com/2011/12/05/gowalla-versus-foursquare/>
- Semenik, M. (2013). GeoData: a new age of geotagging on Wikipedia. Retrieved August 21, 2013, from <https://blog.wikimedia.org/2013/01/31/geodata-a-new-age-of-geotagging-on-wikipedia/>
- The Associated Press. (2013). Number of active users at Facebook over the years - Yahoo! News. Retrieved June 27, 2013, from <http://news.yahoo.com/number-active-users-facebook-over-230449748.html>
- Veness, C. (2012). Calculate distance and bearing between two Latitude/Longitude points using Haversine formula in JavaScript. Retrieved March 14, 2014, from <http://www.movable-type.co.uk/scripts/latlong.html>
- W3Techs. (2013). Usage Statistics and Market Share of Server-side Programming Languages for Websites, October 2013. Retrieved October 2, 2013, from http://w3techs.com/technologies/overview/programming%5C_language/all
- Wikipedia. (2013a). Wikipedia:Statistics - Wikipedia, the free encyclopedia. Retrieved March 26, 2013, from <http://en.wikipedia.org/wiki/Wikipedia:Statistics>
- Wikipedia. (2013b). Wikipedia:Welcome unregistered editing - Wikipedia, the free encyclopedia. Retrieved June 6, 2013, from http://en.wikipedia.org/wiki/Wikipedia:Welcome%5C_unregistered%5C_editing
- Wordpress. (2013a). GNU General Public License. Retrieved June 21, 2013, from <https://wordpress.org/about/gpl/>
- Wordpress. (2013b). Requirements. Retrieved June 21, 2013, from <http://wordpress.org/about/requirements/>

Wordpress. (2013c). Stats - WordPress.com. Retrieved May 16, 2013, from <http://en.wordpress.com/stats/>