Enterprise Frameworks Project - maps-ent

An enterprise level application to manage the content for a map-based RIA and provide a Web API

About

Enterprise Frameworks Project, MSc in Web Technologies (NCI)

Project Title	**maps-ent**
Team Members	Andrew Burnett -13115448
	Adam Harrington -13113305
	Fiona McAndrew -13121693
Tutor	Vikas Sahni
Technologies	.NET MVC5 EF
	C#
	Azure
Source	https://github.com/andburn/maps-ent
URL	http://mapsago.azurewebsites.net/
Licence	n/a

Contents

1. Background research and investigations

1.1. Purpose

The purpose of this document is to set out the requirements for the development of an enterprise grade application to support a public map-based educational rich internet application.

The intended customers are defined as follows:

User A: *RIA*

User A is a rich internet application (separate application) designed to display data it requests from the application

User B: *Data Curator*

User B is concerned with sourcing, managing and approval of data from third party sources for use in the application.

User C: *Owner*

User C is the overarching owner of the application and the related RIA. This user is concerned with how the system works together and with managing Data Curators (User B)

1.2. Project Scope

The scope of the project is to develop an application to facilitate the RIA through provision of structured data for user presentation. The application will also be concerned with the sourcing and management of the data.

The system shall also allow for abstracted data manipulation by the administrative users; the data curators. This will allow for the creation, reading, updating and deletion (CRUD) of data.

Data shall be sourced through programmable commands accessing third-party web content. Data persistence will be managed throughout the application.

1.2.1 - Constraints

- The client has stipulated use of the ASP.NET and C# technology stack.
- Date for completion has been tabled for the 29th April 2014.

1.3 - Definitions, Acronyms, and Abbreviations

Application	Used to refer to the application being developed as opposed to the RIA or the client application
RIA	Rich Internet Application, AKA the client Application
Client Application	Used to refer to the rich internet application which will be a primary user of the applications
Management Interface	Used to refer to the interface which will be operated by the Data Curator
Data Curator	Used to refer to the personnel in charge of the Management Interface.
Owner	Used to refer to the overarching owner of the application and the related RIA.
GUI	Graphic User Interface.
UR	Used to refer to the User Requirement
SR	Used to refer to the System Requirement
IR	Used to refer to the Interface Requirement
FR	Used to refer to the Functional Requirement
DR	Used to refer to the Documentation Requirement
DTR	Used to refer to the Data Requirement

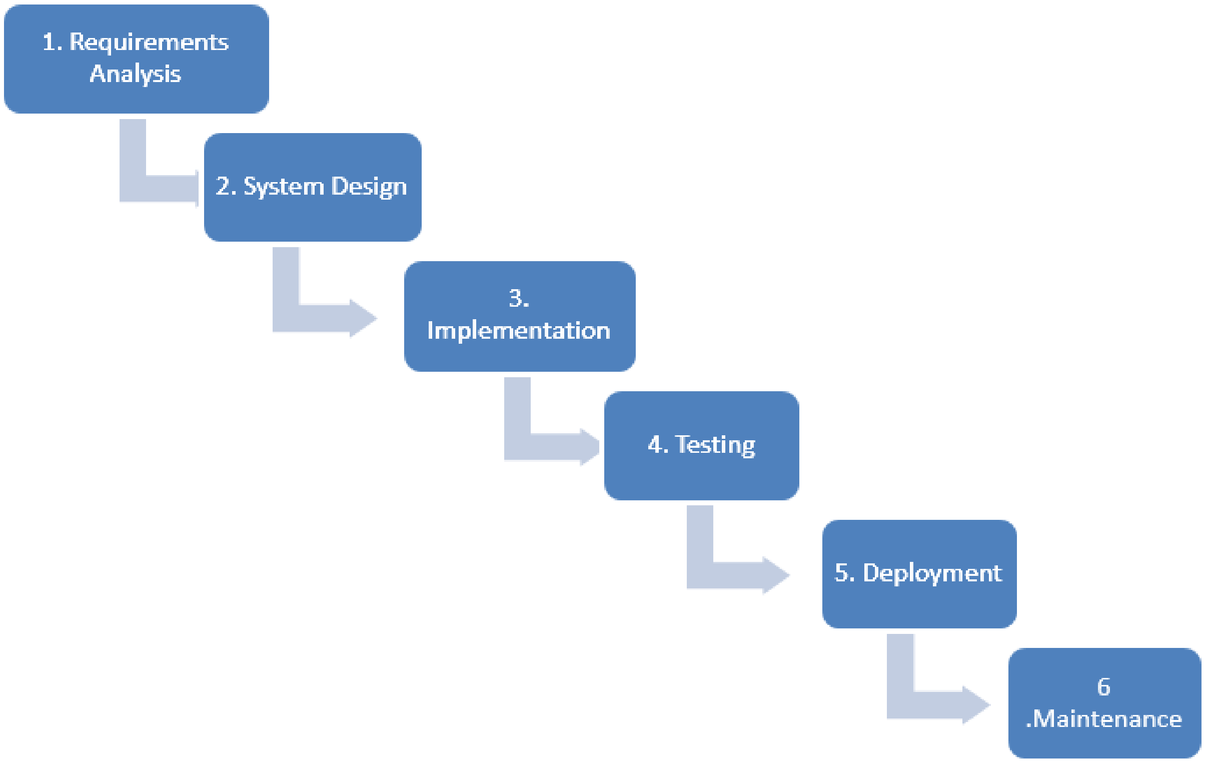
2. Project Plan

Project plan see **appendix I**. This includes a description of all tasks identified and explained. It also shows resource allocation and timelines.

3. Software development methodology employed

3.1. Development Methodology

Waterfall SDLC



I. **Requirement Gathering and analysis**:
   - Project Scope is defined,
   - Requirements specification:
     - Functional,
     - Interface requirements,
     - Documentation Requirement.

II. **System Design**:
    The requirement specifications from first phase are studied in this phase and system design is prepared. The System Architecture is defined, which will then inform the specification of the system's Data and Non-Functional Requirements.

III. **Implementation:** Writing code

IV. **Integration and Testing** :Functional and non-functional testing;
   - Unit testing,
   - integration testing,
   - User acceptance testing
   - And system testing.

V. **Deployment of system:** Once the functional and non-functional testing is done, the product is deployed.

VI. **Maintenance** - Out of scope for this project.

Waterfall Model Pros & Cons

Pros	Cons
Simple and easy to understand phases, therefore good for novice programmers	High amounts of risk and uncertainty.
Easy to manage due to the rigidity of the model, each phase has specific deliverables and a review process.	High amounts of risk and uncertainty.
Phases are processed and completed one at a time.	Cannot accommodate changing requirements.
Works well for smaller projects where requirements are very well understood.	Adjusting scope during the life cycle can end a project.
Well understood milestones.	Integration is traditionally done as at the very end, which doesn't allow identifying any technological or business bottleneck or challenges early.
Easy to arrange tasks.	No working software is produced until late during the life cycle.

Note: Test deployment was undertaken during the development phase, as the systems needed to be tested.

3.2. Technology

The project was developed using the following technologies.

- Framework: .NET MVC5 EF
- Language: C#
- Deployment: Azure

4. Requirements Analysis

User requirements

4.1. User A - (Client API - Machine)

This user is indifferent to data in the database, their concern lies in the delivery of the queried data between the interfaces.

User Requirement		UR	FR	SR	IR	DR	DTR
UR1	The User must to able to send queries to the dataset, based on a structured format	-	1		1		
UR2	The User must receive data back from a request in a structured and expected format	-	2		1		

4.2. User B - (Data Curator - Human):

This user is concerned with data in the database, it is their job to populate and curate it.

User Requirement		UR	FR	SR	IR	DR	DTR
UR3	The user must be able to source new content	-	3				
UR4	The User must be able to access and edit existing records within the application	-	4				
UR5	The User must be able to access the application, through most modern browsers	-	8				
UR6	The User must be able to log in to the admin only interface, by using secure and private authorisation	-	6				
UR7	The User should be able to view changes / conflicts to the database	-	4				
UR8	The User should be able to preview changes made by themselves and other members of the curation team	-	3,4				
UR9	The User should not need comprehensive technical knowledge to use the application	-	6				
UR10	The User should be able to use all the system functions after a total of two hours training	-				2	
UR12	The User should receive user friendly error messages to notify users of errors in the application	-	7				

4.3. User C – Owner (Admin)

This user is concerned with how the system works together and with managing Data Curators (User B).

User Requirement		UR	FR	SR	IR	DR	DTR
UR13	The User must be able to assign and manage roles	-					
UR14	The User can delete events	-					

Functional requirements definition

FR1	**PROVIDE A DOCUMENTED RESTFUL API TO ACCESS STORED DATASETS**				
DESCRIPTION	The system must provide a documented API in order for external systems to retrieve the datasets they require. The API should be designed using best practices to cater for future changes to the system.				
TRIGGER	The user (User A, client machine) sends requests using the API to acquire datasets.				
TECHNICAL ISSUES	The API needs to be designed in a robust and evolvable manner.				
RISKS	Invalid API requests need to be handled gracefully.				
PRIORITY	Level 1, essential to system operation.				
DEPENDENCIES	None				
RELATIONS	**UR**	**FR**	**SR**	**IR**	**DR**
	1	**-**	**-**	**-**	**-**

FR2	**API REQUESTS RETURN STRUCTURED DATA IN JSON FORMAT**				
DESCRIPTION	Client requests to the system, using the public API, will return the requested data in JSON format. The format of the return JSON data should be fully documented along with the API specification				
TRIGGER	The user (User A, client machine) gets a response in JSON format after a successful request using the API				
TECHNICAL ISSUES	Need to ensure all responses are valid JSON				
RISKS	Any errors in API requests must return valid JSON data in response. The returned JSON should indicate an error has occurred, so the client can handle it correctly				
PRIORITY	Level 1, essential to system operation				
DEPENDENCIES	FR1				
RELATIONS	**UR**	**FR**	**SR**	**IR**	**DR**
	2	**-**	**-**	**-**	**-**

<table>
<tr><th>FR3</th><th colspan="5">PROVIDE ABILITY TO CREATE NEW DATA SETS</th></tr>
<tr><td><b>DESCRIPTION</b></td><td colspan="5">It must be possible to create new data sets easily from predefined sources</td></tr>
<tr><td><b>TRIGGER</b></td><td colspan="5">The user (User B, Data Curator) creates new data sets by interacting with a simple search like interface</td></tr>
<tr><td><b>TECHNICAL ISSUES</b></td><td colspan="5">It should be possible for a developer to add new sources to the system when required without altering the existing functionality</td></tr>
<tr><td><b>RISKS</b></td><td colspan="5">Any errors that may occur during the process of adding new data must be clearly communicated to the user. Incomplete additions to database must be rolled back</td></tr>
<tr><td><b>PRIORITY</b></td><td colspan="5">Level 1, essential to system operation</td></tr>
<tr><td><b>DEPENDENCIES</b></td><td colspan="5">None</td></tr>
<tr><td><b>RELATIONS</b></td><td><b>UR</b></td><td><b>FR</b></td><td><b>SR</b></td><td><b>IR</b></td><td><b>DR</b></td></tr>
<tr><td></td><td><b>3,8,9</b></td><td><b>-</b></td><td><b>-</b></td><td><b>-</b></td><td><b>-</b></td></tr>
</table>

FR4	**PROVIDE ABILITY TO VIEW AND EDIT EXISTING DATA SETS**				
DESCRIPTION	An interface should be provided to allow manipulation of existing data stored in the database. Different data sets could be displayed in a table and a simplified map to visualize the geospatial data				
TRIGGER	The user (User B, Data Curator) can view and edit existing data sets by using a simple browser based interface				
TECHNICAL ISSUES	Legacy browsers will not be supported				
RISKS	Errors must be handled and database integrity must be assured by rolling back any unsafe changes				
PRIORITY					
DEPENDENCIES	None				
RELATIONS	**UR**	**FR**	**SR**	**IR**	**DR**
	5,7,8	**-**	**-**	**-**	**-**

FR5	**NEED A SECURE AUTHENTICATION SYSTEM TO CONTROL ACCESS**				
DESCRIPTION	Only approved users should have access to the management interface. This should be achieved by requiring users to authenticate themselves with the system by using a secure login interface				
TRIGGER	The user logs in to the system using secure login credentials, such as username and password				
TECHNICAL ISSUES	Users need to be approved for access. An administrator user account is needed with privileges to add and remove users				
RISKS	Need to be able to recover or reset lost or forgotten credentials				
PRIORITY	Level 1, essential to system operation				
DEPENDENCIES	None				
RELATIONS	**UR**	**FR**	**SR**	**IR**	**DR**
	6	**-**	**-**	**-**	**-**

FR6	**DISPLAY AND LOG INFORMATIVE MESSAGES WHEN EXCEPTIONS OCCUR**
DESCRIPTION	All errors and exceptions that occur in the system should be handled properly. If the exception can be handled by the system, it should be logged with a clear description of what happened. If user interactions are affected a message should be displayed to the user indicating what happened and the steps to take, if any, to correct the situation
TRIGGER	User A should be able to determine an error has occurred based on the response received from an API request. Other users should be informed of relevant errors through the interface

TECHNICAL ISSUES	As far as possible the system should deal with all errors in a graceful manner without the need for user interaction				
RISKS	Errors may occur in the exception handling system itself leading to illegal operation being performed. These modules must be tested thoroughly				
PRIORITY	Level 2, not essential to system operation				
DEPENDENCIES	None				
RELATIONS	**UR**	**FR**	**SR**	**IR**	**DR**
	-	**-**	**-**	**-**	**-**

FR7 DELIVER USER CONTENT THROUGH A WEB BROWSER

DESCRIPTION	The management interface should be a web application accessible from all modern web browsers. There should be no reliance on a particular physical location or operating system. Support for legacy browser will not be supported				
TRIGGER	User's B and C will interact with the management interface from a web browser				
TECHNICAL ISSUES	Use cross-platform technologies to operate on all modern browsers				
RISKS	Possible difference in user experience on different platforms				
PRIORITY	Level 1, essential to system operation				
DEPENDENCIES	None				
RELATIONS	**UR**	**FR**	**SR**	**IR**	**DR**
	4	**-**	**-**	**-**	**-**

Interface requirements

This section describes how the software interfaces with other software systems and users; input and output. The diagram below illustrates the high lever goals of the system interfaces.

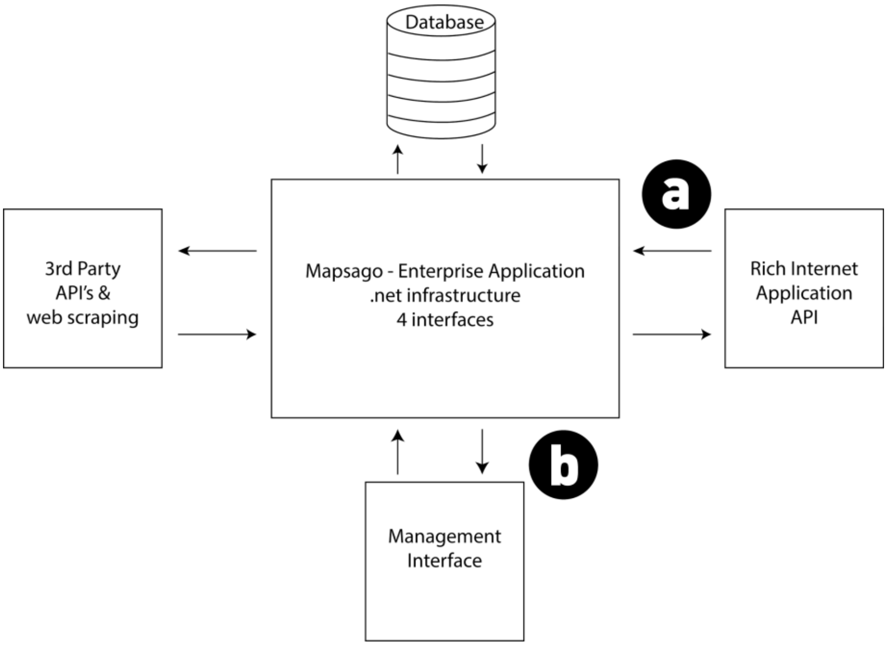

Database
a
3rd Party API's & web scraping
Mapsago - Enterprise Application .net infrastructure 4 interfaces
Rich Internet Application API
b
Management Interface

Figure 1 - Diagram of Interfaces

IR1	**PROVIDE DATA TO RIA**				
DESCRIPTION	Respond to structured requests with appropriate data output				
TRIGGER	User A will send a structured request to a pre-defined URI				
TECHNICAL ISSUES	The structure of the requests and responses must be defined and implemented in a meaningful way. Potentially using REST and JSON				
RISKS	Uncoordinated downtime of system could lead to poor service to clients				
PRIORITY	Level 1, essential to system operation				
DEPENDENCIES	FR1 & FR2				
RELATIONS	**UR**	**FR**	**SR**	**IR**	**DR**
	1,2	**-**	**-**	**-**	**-**

IR2	**PROVIDE A USER INTUITIVE GUI**				
DESCRIPTION	The system interface should conform to usability heuristics, particularly providing for use of knowledge in the world over knowledge in the head and providing meaningful feedback				
TRIGGER	Users B & C access the GUI management interface to perform tasks				
TECHNICAL ISSUES	Feedback should be provided without delay which in a browser may require asynchronous communication				
RISKS	Usability testing may not be feasible so heuristic evaluation may need to be completed by the developers which can lead to bias, assumptions and oversight				
PRIORITY	Level 2, not essential to system operation				
DEPENDENCIES	FR8				
RELATIONS	**UR**	**FR**	**SR**	**IR**	**DR**
	6-13	**-**	**-**	**-**	**-**

Documentation Requirements

There are three forms of documentation involved in the system:

- ★ Help documentation for the Management GUI interface
- ★ Reference documentation for the API provided for client applications
- ★ System documentation for later extensions or modifications to the system

DR1	**INTEGRATE HELP DOCUMENTATION FOR USER B INTO THE INTERFACE**				
DESCRIPTION	Help for User B will be integrated into the management interface using some of the following: informative error messages				
TRIGGER	-				
TECHNICAL ISSUES	-				
RISKS	-				
PRIORITY	Level 2, not essential to system operation				
DEPENDENCIES	None				
RELATIONS	**UR**	**FR**	**SR**	**IR**	**DR**
	-	**-**	**-**	**-**	**-**

DR2	**DOCUMENT STRUCTURE AND OPERATION OF API FOR USER A**				
DESCRIPTION	As the API is defined the documentation must be maintained with priority placed on accuracy, relevance and completeness				
TRIGGER	-				
TECHNICAL ISSUES	-				
RISKS	-				
PRIORITY	Level 1, essential to system operation				
DEPENDENCIES	None				
RELATIONS	**UR**	**FR**	**SR**	**IR**	**DR**
	11	**6**	**-**	**-**	**-**

Data requirements

The data requirements for the application weighed up the costs and benefits associated with local storage of data compared with those of processing frequent requests to external sites, making modifications to retrieved data and serving a response.

<table>
<tr><th>DTR1</th><th colspan="5">Data Acquired From External Sources Will Be Stored In The System Database</th></tr>
<tr><td><b>DESCRIPTION</b></td><td colspan="5">Data sets retrieved from external sources will be processed and shaped into an appropriate format for use within the system. This will allow more efficient transfer of data between the server and User A, by allowing a single request to contain data from multiple sources</td></tr>
<tr><td><b>TRIGGER</b></td><td colspan="5">-</td></tr>
<tr><td><b>TECHNICAL ISSUES</b></td><td colspan="5">-</td></tr>
<tr><td><b>RISKS</b></td><td colspan="5">-</td></tr>
<tr><td><b>PRIORITY</b></td><td colspan="5">Level 1, essential to system operation</td></tr>
<tr><td><b>DEPENDENCIES</b></td><td colspan="5">None</td></tr>
<tr><td><b>RELATIONS</b></td><td><b>UR</b></td><td><b>FR</b></td><td><b>SR</b></td><td><b>IR</b></td><td><b>DR</b></td></tr>
<tr><td></td><td><b>-</b></td><td><b>-</b></td><td><b>-</b></td><td><b>-</b></td><td><b>-</b></td></tr>
</table>

<table>
<tr><th>DTR2</th><th colspan="5">Data must be manually approved for addition to the database</th></tr>
<tr><td><b>DESCRIPTION</b></td><td colspan="5">Automatic addition of data into the system is fraught with danger. Incorrect or incomplete information will reduce the effectiveness of the system. User B must manually approve the data they are allowing into the system</td></tr>
<tr><td><b>TRIGGER</b></td><td colspan="5">-</td></tr>
<tr><td><b>TECHNICAL ISSUES</b></td><td colspan="5">-</td></tr>
<tr><td><b>RISKS</b></td><td colspan="5">-</td></tr>
<tr><td><b>PRIORITY</b></td><td colspan="5">Level 1, essential to system operation</td></tr>
<tr><td><b>DEPENDENCIES</b></td><td colspan="5">None</td></tr>
<tr><td><b>RELATIONS</b></td><td><b>UR</b></td><td><b>FR</b></td><td><b>SR</b></td><td><b>IR</b></td><td><b>DR</b></td></tr>
<tr><td></td><td><b>-</b></td><td><b>-</b></td><td><b>-</b></td><td><b>-</b></td><td><b>-</b></td></tr>
</table>

Non-Functional Requirements

Availability requirement

The data available to User A via an API should be available unless downtime is coordinated with the RIA.

Recover requirement

Logs of changes must be kept by the system to help trace and/or undo unwanted changes to the system. Versioned backups of the database are also required to allow fast recovery of the system.

Security requirement

Authentication and authorisation must be employed for the management interface (user B and user C) to protect the system from unauthorised tampering.

Maintainability requirement

The code documentation should specify clearly the structure and organisation of the system to allow for speedy resolution to problems. The system should be modular and loosely coupled to ensure changes do not affect unrelated parts of the system.

Portability requirement

The API should be designed to return consistent results regardless of the origin of the caller. The management interface should also work consistently across all modern browsers.

Extendibility requirement

Loose coupling should ensure that adaptations or extensions can be made with confidence of not causing unexpected consequences within the system.

Reusability requirement

The application should be built with reusability in mind, except where efforts to generalise the code base conflicts with other user requirements.

Resource utilization requirement

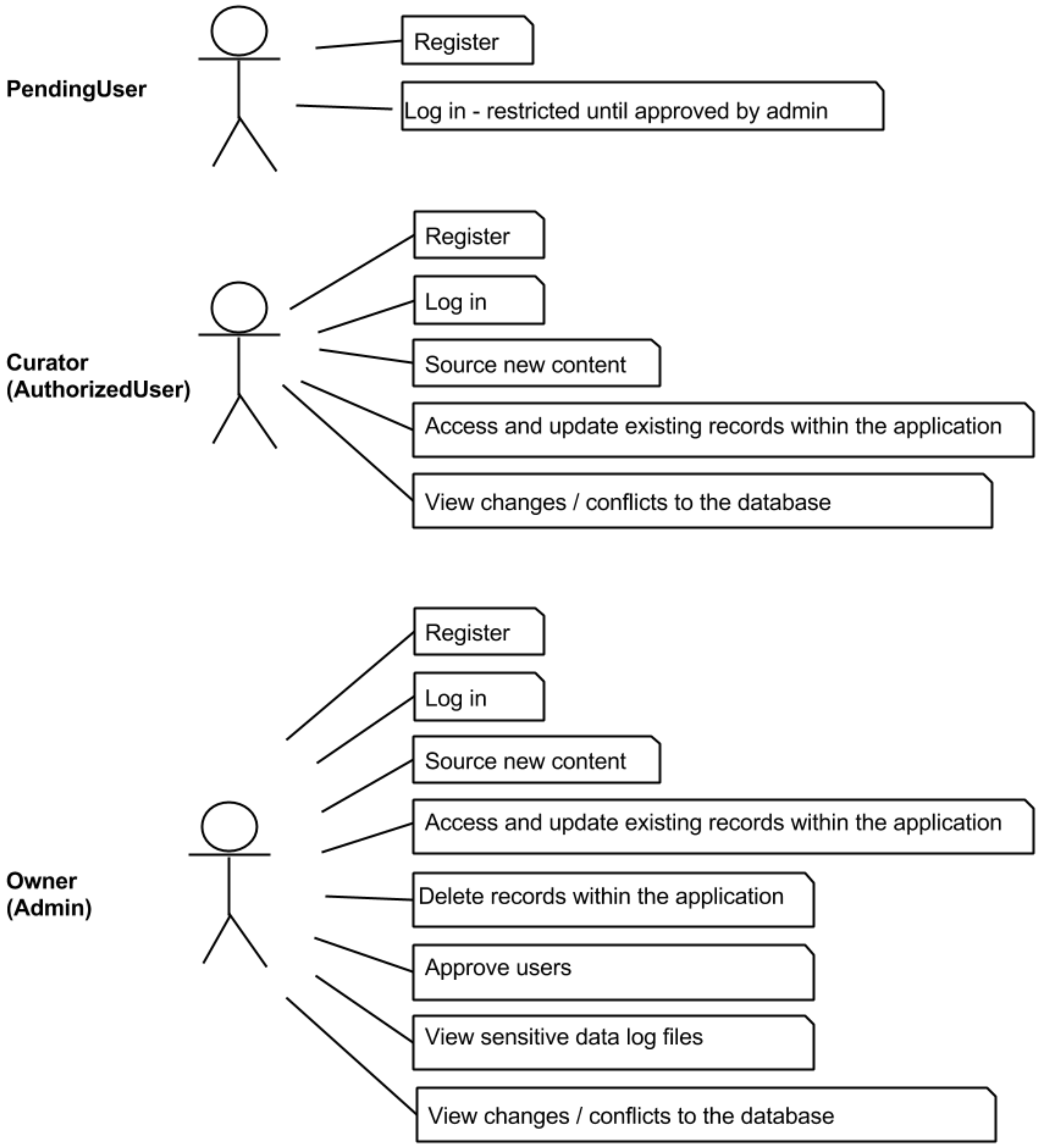
Storage of data sets is restricted to only the information required to feed the API, reducing data storage in the database.

Robustness requirement

The system must be scalable to tolerate a large number of request to the API. The management interface must also be able to accommodate multiple users accessing it at the same time.

5. Use cases

Actors:
Default User (Pending User)
Curator (Authorized)
Owner (Admin)


PendingUser
Register
Log in - restricted until approved by admin
Curator
(AuthorizedUser)
Register
Log in
Source new content
Access and update existing records within the application
View changes / conflicts to the database
Owner
(Admin)
Register
Log in
Source new content
Access and update existing records within the application
Delete records within the application
Approve users
View sensitive data log files
View changes / conflicts to the database

See user actions in appendix II.

6. Architecture/Design approach

The application is designed to employ a number of architectures including MVC and SOA. By providing well defined interfaces and loosely coupled layers the application will leverage performance of various platforms regardless of implementation, paving the way for scalability and adaptability.

This approach has major benefits for rich internet applications plugging into the API interface as all processing can be achieved in the business layer while a simplistic abstraction is exposed.

> *"Design to take advantage of client processing power. RIAs run on the client computer and can take advantage of all the processing power available there. Consider moving as much functionality as possible onto the client to improve user experience. Sensitive business rules should still be executed on the server, because the client logic can be circumvented". (Microsoft, 2013)*

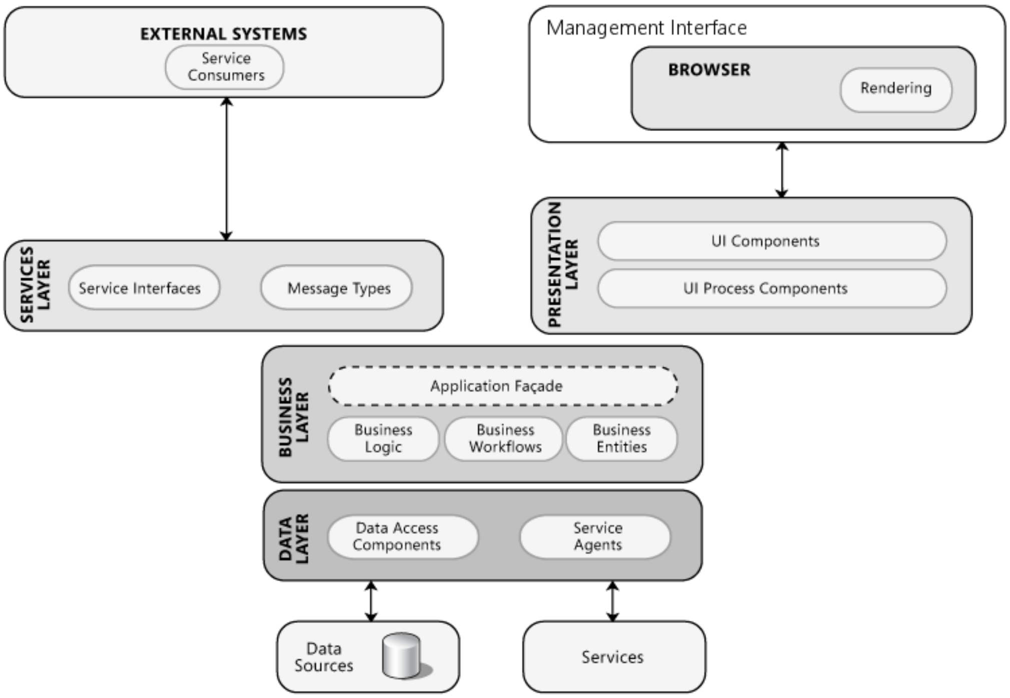


Figure 2 - Adapted from Microsoft's rich internet application and service archetypes

Aside from RIA interfaces, there is also the management interface will utilize multi-layered, or specifically an MVC architecture to provide a richer user experience (Microsoft, 2013).

Because much of the application involves sending messages without a need for a UI, the architecture will incorporate many of the characteristics of a service archetype (Microsoft, 2013). The service oriented elements include:

- Loosely coupling
- Complex
- Distributable
- Message oriented
- Technology agnostic

7. Models

We developed the models using the code-first approach to ASP.NET MVC. The models were created as POCO's (Plain Old CLR Objects). Developing in the fashion allows development to stay in the business domain as long as possible. There is also the added advantage of not having to deal with the direct creation of the database either using a visual designer or standard SQL commands. The .NET Entity framework takes of all that for the code-first developer.

The structure of the database tables and their relationships can be seen in the following diagram. In addition to our own models, the Identity framework that comes as part of MVC 5 provides a suite of functionality for user management and authentication. The models access this functionality using the code-first approach of navigation properties and foreign keys. This adds another set of models and associated database tables project.

Custom-shaped ViewModel classes

The MVC pattern makes it possible to efficiently manage common functionality centered around creating, deleting, updating and deleting content within a structured application; perfect for a management system. .Net's MVC framework implements this pattern with direct entity to mapping between models and display.

However, direct mapping of real database entities is not always desirable. Using class mappings directly exposes a model object and has security implications; how much end users know about how data is stored in the database leaves systems open to SQL injection attacks.

Supporting direct model mapping from properties is also a convenient approach and works well in scenarios where the rendering desired for views and UIs corresponds to data objects themselves; but this also is not always the case.

One common approach to represent data or accept data input in different forms is to implement a "custom-shaped ViewModel class" which acts as an interface between the real data, at a system level, and the data abstractions that are more effective for user interfaces.

This way the view can be completely independent of the database while still being strictly typed and benefiting from .Net's useful validation and security features.

Additional benefits, and the main ones desired for this system that this pattern facilitated, were:

- Request or present data belonging to multiple entities in a user-meaningful context.
- Process data from users in a context that makes sense to them, not how the system understands it.
- Produce a consistent and structured format for output for to machine users (GeoJson API)

Custom-shaped ViewModel classes are a less discussed implementation pattern in .Net MVC but they add significant advantages as described above.

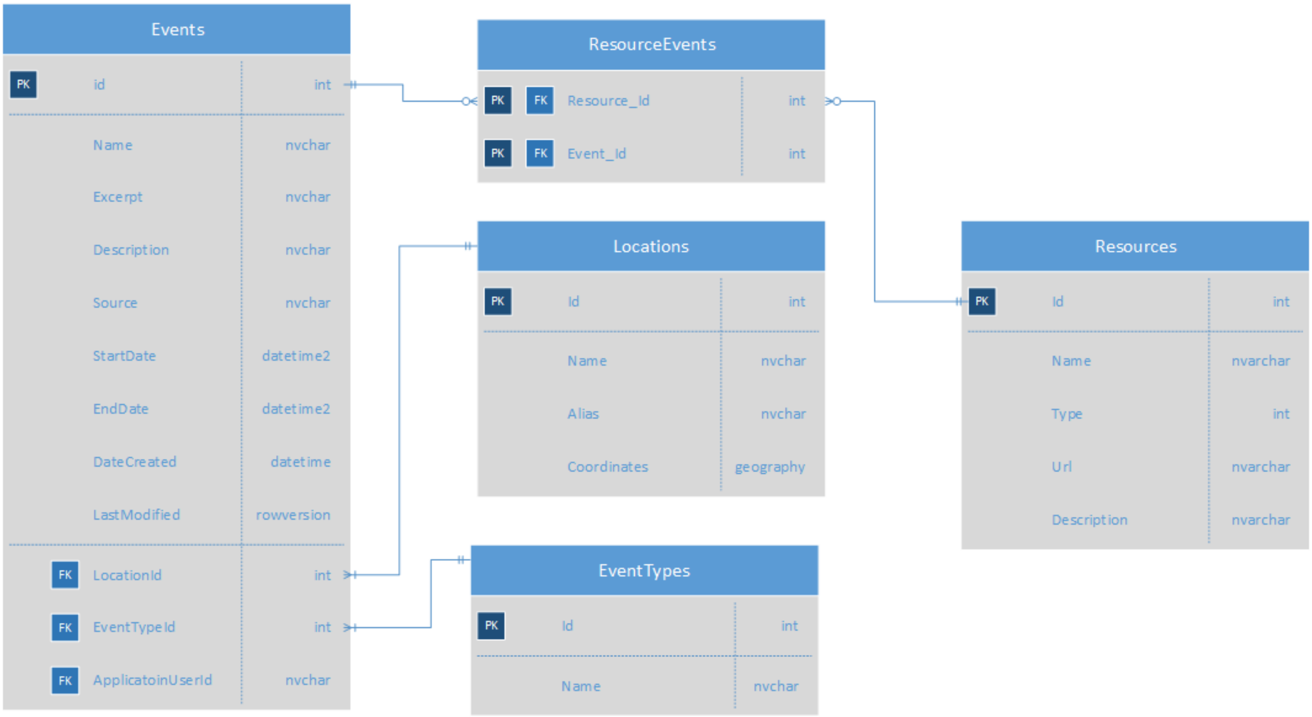


Figure 3 - Entity Diagram

8. Design patterns and architectural patterns implemented in the application

Principals

The SOLID design principles help developers adhere to good object oriented design methodologies. The acronym can be broken down into the following sections.

Single Responsibility Principle

The MVC pattern helps enforce this principle by separating out the different concerns of web application. We fully utilized this approach by sticking to the Model, View, Controller pattern.

Open-closed

The models and classes used in the application use encapsulation to hide the parts of the code that need to be kept private, while allowing extension through inheritance or composition.

Liskov substitution

The ASP.NET framework uses the Liskov substitution principle in many places allowing polymorphic replacement of object types.

Dependency inversion

Dependencies on concrete types should be avoided and programming to interfaces or abstract types is to be encouraged. Dependency injection is just one way to create concrete dependencies among objects at runtime, creating greater flexibility. The InfoGatherer class would benefit from dependency injection techniques as it structured in a way to allow to it.

Design Patterns

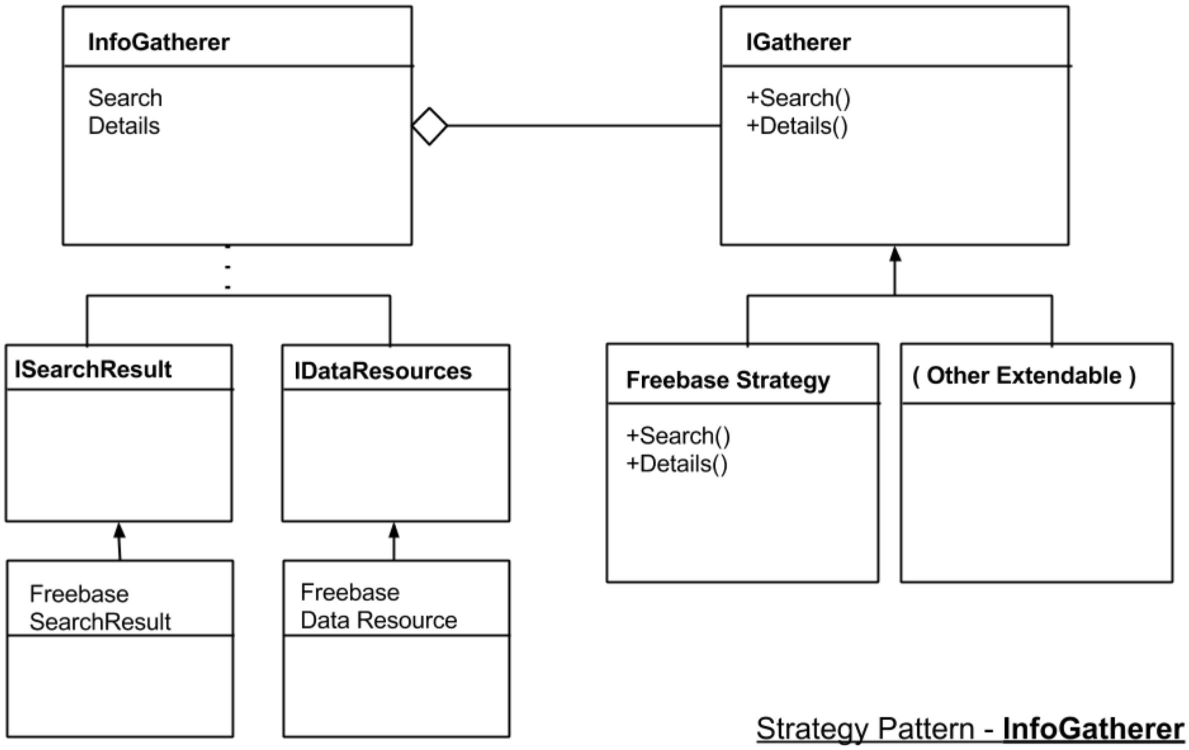
Design patterns are a repeatable solution for common problems. Due to the highly bespoke nature of this application, implementing design patterns proved difficult.

The patterns which were deployed were:

- MVC
- The Strategy pattern.

The strategy pattern

The strategy pattern is the process of disassociating an algorithm from its host and enabling the ability to swap algorithms dynamically in time. The strategy encapsulates algorithms as objects.



Strategy Pattern - **InfoGatherer**

The InfoGatherer and associated classes and interfaces implement the strategy pattern in order to allow behaviours to composed into the class and keeping it lightly coupled. The class defines a number of methods to query third party services. Using the strategy pattern allows different services to be queried using the same programming interface, while implementing them differently.

The repository pattern

The repository pattern is a commonly used pattern in the ASP.NET MVC community. It adds further abstraction to the data layer by creating a buffer between the ORM technology and the application models. Entity framework version six uses the unit of work pattern and now allows the DbSet to be modified to allow for the creation test doubles, one of the motivating factors for the use of the Repository pattern. Based on this it was decided not to incorporate the extra complexity the Repository pattern would involve into the application.

The InfoGatherer and associated classes and interfaces implement the strategy pattern in order to allow behaviours to composed into the class and keeping it lightly coupled. The class defines a number of methods to query third party services. Using the strategy pattern allows different services to be queried using the same programming interface, while implementing them differently.

9. ORM Tool Usage

The Entity framework Object Relationship Mapper is loaded by default in new ASP.NET MVC 5 projects. It creates an abstraction layer between the business layer and the database, allowing the database technology to be changed without effecting the business logic. It allows for different ways of creating the application models, by creating the database first or using a code first approach.

10. Dependency Injection

The use of dependency injection or inversion of control containers stops objects relying on concrete objects and becoming tightly coupled to them. The InfoGatherer class uses the strategy pattern and relies on object composition through interfaces, while dependency injection was not implemented at this point in the future this is an ideal area to introduce an inversion of control container like Ninject to manage dependencies.

11. Cross-Cutting Concerns

Cross cutting concerns are program functionalities that are available through all layers of the application. The ASP.NET MVC framework handles most cross-cutting concerns for the developer. The new Identity API introduced in MVC 5 provides user authentication and management and is used throughout the application.

12. Security of the application

View level security

For form authentication: there is a two part cross site request forgery (CSRF) prevention methods employed within the system.

- In View, add an antiforgery tokens to the forms

```
@html.AntiForgeryToken()
```

- In Controllers, validate the actions

```
[ValidateAntiForgery]
```

Public API security

Furthermore, as described further in section 13.1, the API was designed with security in mind to only accept GET request from cross-origin requests.

Data abstraction

Section 7 documents steps taken to abstract system entities from exposed input providing layers of abstractions between the user interface and the system model.

13. Configuration of the application

Authentication

3 Different types for ASP.NET MVC 4

1. Forms Authentication
2. OpenID / OAuth
3. Windows Authentication

Forms Authentication is suitable for Internet apps, is highly customisable; the developer of the site can control how the form looks and the password strength.

It relies on cookies by default. The cookie is stored in the clients browser so that they do not need to sign in again during a session.

SSL is required to make form authentication secure. Unless the site is hosted on a https address, instead of a regular http address, the username and password will be sent in plain text, making it vulnerable.

OpenID / OAuth is also suitable for Internet applications. Third Party Identity Providers authenticates a user and then redirects the user back to the ASP.NET MVC 4 site with a message verifying that they are who they say they are.

Windows Authentication is best suited to intranet application within an organisation. Users of the will avail of the "IntegratedAuth" provided by Mircosoft and will take advantage of a Single sign on.

Authentication	Internet apps	Features	Suitable for Application
Forms Authentication	yes	SSL required, supplies cookies	yes
OpenID / OAuth	yes	Third Party Indentity Providers	yes
Windows Authentication	no	"IntegratedAuth" provided by Mircosoft	no

Forms Authentication was chosen as the most appropriate Authentication implementation type for this application.

Roles

This web application requires users to register and be approved by an admin before the users are granted access to site.

There are 3 types of roles in this application.

Role	Description
PendingUser	**PendingUser** are users awaiting approval by the **administrator**.
AuthorizedUser	**AuthorizedUser** are Data Curators who are concerned with sourcing, managing and approval of data from third party sources for use in the application.
Admin	The **Administrator** is the overarching owner of the application. This user is concerned with how the system works together and with managing and approving Data Curators (ApprovedUser).

Setting up roles w/ AuthorizeAttribute Class

Launch application, if database is not present setup the following users:

```
user: JoeSoap password: testing //role assigned PendingUser user: admin password: testing // role assigned
admin user: RegisteredJoe password: testing // role assigned AuthorizedUser
```

Step 0:
In the table **AspNetRoles** assign the following:

id	Name
1	PendingUser
2	Admin
3	AuthorizedUser

In the table **AspNetUserRoles** assign the following:

Userid	Name
<\id of an Admin user>	2
<\id of an AuthorizedUser user>	3

(note a PendingUser does not need to be assigned a role, as everyone who registers is assigned the PendingUser status by default (id 1))

Step 1:
Add a filter to the `FilterConfig` file in the `App_Start folder`.
This filter will require a user to login to use the functionality of the web application.
Add: `filters.Add(new AuthorizeAttribute());`
Step 2:
Add `[Authorize]` to HomeController

```
//only authorized roles such as PendingUser, Admin and AuthorizedUser. [Authorize] public class HomeController : Controller { //code ommitted }
```

Step 3:
Add `[Authorize(Roles = "Admin , AuthorizedUser")]` to EventController

```
//only the Admin and AuthorizedUser users can view the Events page [Authorize(Roles = "Admin , AuthorizedUser")] public ActionResult Index() { //code ommitted }
```

Step 4:
Update **_Layout.cshtml** to only show application links if the site user `isAuthenticated`

```
@using Microsoft.AspNet.Identity @if (Request.IsAuthenticated) { <li>@Html.ActionLink("Home", "Index", "Home")</li> <li>@Html.ActionLink("About", "About", "Home")</li> <li>@Html.ActionLink("Contact", "Contact", "Home")</li> <li>@Html.ActionLink("Events", "Index", "Event")</li> } else { //no link options are dislayed }
```

14. Scalability of the application

Azure Deployment

Azure was chosen as a deployment platform, given the relatively abstracted integration with Visual Studio and the Asp.Net MVC framework.

Information regarding azure "virtual machines" vs. "websites" can be found on the platform website; a high level overview of the difference would be to liken Azure websites to the PaaS, or platform as a service, model. Virtual machines are more akin to Amazons IaaS offerings through AWS EC2 and have to be configured manually.

Websites still offer the full .Net potential with lower configuration overheads. A sample website was deployed to http://nci-testapi.azurewebsites.net/ to explore the platform potential prior to development. This sample application acted as proof of concept.

15. Testing Approach

This project uses the waterfall development model testing approach.

Testing Levels

Unit Testing

Unit testing are vital in the creation of a enterprise level web application, as the tests varify that the code that is built works as expected and operates correctly. The former holds short term advantages for the applications development.

There is also long time benifits of testing as they insure that changes to the code don't indvertently affect other parts of the system.

> "In an ASP.NET MVC application, controllers will interact with services or repositories to handle each request. As each piece is built, these interactions can be tested using unit tests to instill confidence that the system continues to work as new features are added or new versions of dependent libraries are supplied." (http://msdn.microsoft.com/en-us/library/hh404088.aspx#sec8)

Setting up Unit Tests

All testing work is carried out in the `mapsago.Tests` section.
Visual studio provides a testing framework called

All unit tests in this project a structured using the **"AAA"** testing pattern.

1. Arrange: Setup the unit being tested
2. Act: Exercise the unit under test and capture results
3. Assert: Verify the behavior

Add **comments** and meaningful **descriptions**.

Add a comments to the class that appropriately describe the business scenario leading to the tests, this will help focus the test as well as providing documentation for the application to be revisited at a later date.

The following is a unit test, which tests the InfoGatherer class.

It draws on the

```
using System; using Microsoft.VisualStudio.TestTools.UnitTesting; using MapsAgo.Domain; using
System.Collections.Generic; // This test the info gather namespace MapsAgo.Tests.Domain { [TestClass] public
class TestInfoGatherer { private IGatheringStrategy Strategy; private InfoGatherer Gatherer; [TestInitialize] public
void Initialize() { Strategy = new MockStrategy(); Gatherer = new InfoGatherer(Strategy); } [TestMethod] public
void Test_Search_Results() { IList<ISearchResult> results = Gatherer.Search("blah", "blah");
Assert.AreEqual("/m/0c55s", results[0].Id); Assert.AreEqual("Battle of Little Bighorn", results[1].Name); }
[TestMethod] public void Test_Details_Results() { IDataResource resource = Gatherer.Details("foo");
Assert.AreEqual(0, resource.EventEndYear); } } }
```

Component interface testing

Component interface testing does log in, button,etc work.

System testing

Scenerio based testing, based on user cases was carried out. All requirements were met according to the MoSoCoW requirement priority breakdown. Notable features include:

Must Have:

- Users can add and edit and otherwise manage the database of historical data.
- Admin User can manage users
- Provide machine access to data (API)

Won’t have:

Features due for implementation on future releases.

- Search third-party data sources within the system.
- Performance statistics systems administrator interfaces.

Acceptance testing

Client, or end-user acceptance testing is subject to approval of Vikas Sahini, NCI.

16. Other relevant features of the application

16.1. API - Development

Defining the interface between the two applications (Service provider, service user) required negotiation and consideration of the differing needs of each.

An API was designed to accept cross-origin GET requests in a number of formats which were defined in consultation with the primary service client. The needs of the client application at present can be broken into to categories, query-less and query-full requests, the later having many permutations of actual usage.

Request string format

The request sting must always be directed to HOST/api to be accepted for cross origin requests. The two route formats defined for use in this project follow a common structure. One designed to receive a general request with no query parameters, and one which accepts multiple variables used for querying entities and geo-data:

```
http://HOST/api/events

http://HOST/api/{altitude-zoom}/{latitude}/{longitude}?categories=1|2
```

By structuring the data in this way we present a stable, reliable interface to machine users accepting the data necessary to make informed decisions and return the desired responses.

Accepted query parameters are:

Categories

```
categories=battles|cheese|etc
```

A greedy search, results in any or all of the provided categories if provided

Keywords

```
keywords=hello\|world
```

A greedy search, results in any or all of the provided keywords in event descriptions if provided

startdate

```
startdate=01-01-2001
```

The end date for a query based on time range

Enddate

    enddate=01-08-2011
The end date for a query based on time range

A full request might look like this:

    http://HOST/api/9/-6/55?categories=battles|cheese|etc&keywords=hello\|world&startdate=01-01-2001&enddate=01-08-2011

Response format and structure

The response format needed to be interpreted by a JavaScript application, as such the returning of data structured as a JSON object of results suited the client needs over xml which would require additional parsing.

Creating a JSON response, in a predefined format in the .Net MVC framework requires careful planning and data entity abstraction, because true entity names or database structure should not be exposed to the client. To this end an API ViewModel was created to act as the "view" or response abstraction

The format below shows an example of this geo JSON.

~~~
```JavaScript
{ "type": "FeatureCollection",
    "features": [
      { "type": "Feature",
        "geometry": {"type": "Point", "coordinates": [102.0, 0.5]},
        "properties": {"prop0": "value0"}
        },
      { "type": "Feature",
        "geometry": {
          "type": "LineString",
          "coordinates": [
            [102.0, 0.0], [103.0, 1.0], [104.0, 0.0], [105.0, 1.0]
            ]
          },
        "properties": {
          "prop0": "value0",
          "prop1": 0.0
          }
        },
      { "type": "Feature",
         "geometry": {
           "type": "Polygon",
           "coordinates": [
             [ [100.0, 0.0], [101.0, 0.0], [101.0, 1.0],
               [100.0, 1.0], [100.0, 0.0] ]
~~~

```
             ]
         },
         "properties": {
           "prop0": "value0",
           "prop1": {"this": "that"}
           }
         }
       ]
     }
```

GeoJson format uses Longitude/Latitude ordering rather than the Latitude/Longitude ordering ISO 6709 standard employed by most other systems including Google Maps. Despite this, it was decided to put systems implementation secondary to adherence to the GeoJson format in order to increase accessibility of the API for other systems.

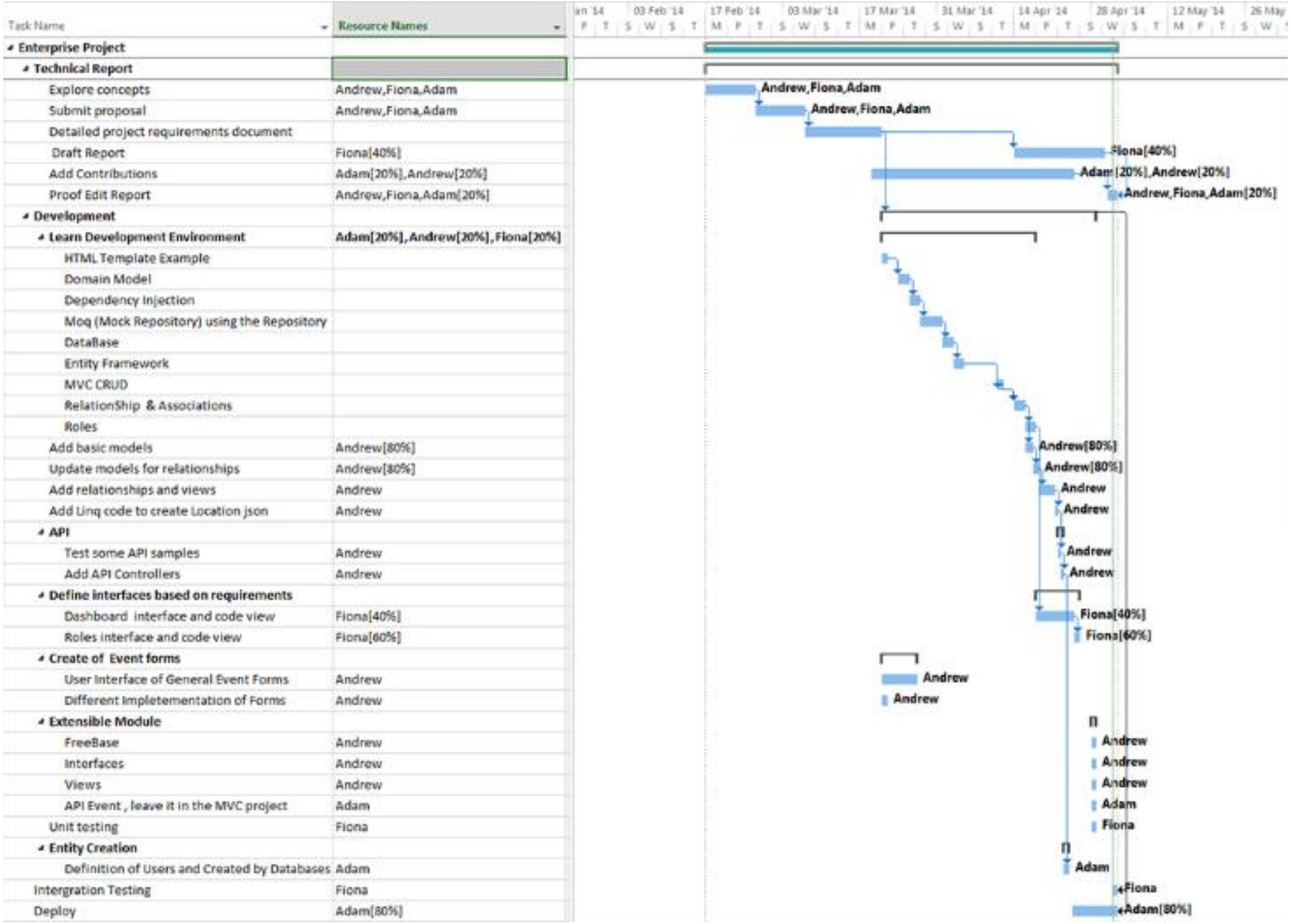
To compensate, the MapsAgo API includes a `location` property with `latitude` and `longitude` properties

```javascript
{
    "type": "FeatureCollection",
    "features": [
      {
          "type": "Feature",
          "geometry": {
              "type": "Point",
              "coordinates": [53.365, -6.21]
          },
          "properties": {
              "name": "Battle of Clontarf",
              "excerpt": "The Battle of Clontarf took place on 23 April 1014 between the forces of Brian Boru and the...",
              "startDate": "1014-04-23",
              "endDate": "1014-04-23",
              "category": "Battle",
              "resources": [
                {
                    "type": "image",
                    "name": "Www",
                    "url": "http://upload.wikimedia.org/wikipedia/commons/e/e9/Www.wesleyjohnston.com-users-ireland-maps-historical-map1014.gif"
                }, {
                    "type": "link",
                    "name": "Wikipedia Link",
                    "url": "http://en.wikipedia.org/wiki/index.html?curid=155550"
                }],
              "location": {
                  "alias": null,
                  "name": "Clontarf, Dublin",
                  "latitude": 53.365,
                  "longitude": -6.21
              },
              "description": "The Battle of Clontarf took place on 23 April 1014 between the forces of Brian Boru and the forces led by the King of Leinster, Máel Mórda mac Murchada: composed mainly of his own men, Viking mercenaries from Dublin and the Orkney Islands led by his cousin Sigtrygg."
          }
      },
    {
        "type": "Feature"
    }]
}
```

In this way each response has two properties, `type` with value `FeaturesCollection` and `features` as an array of points.

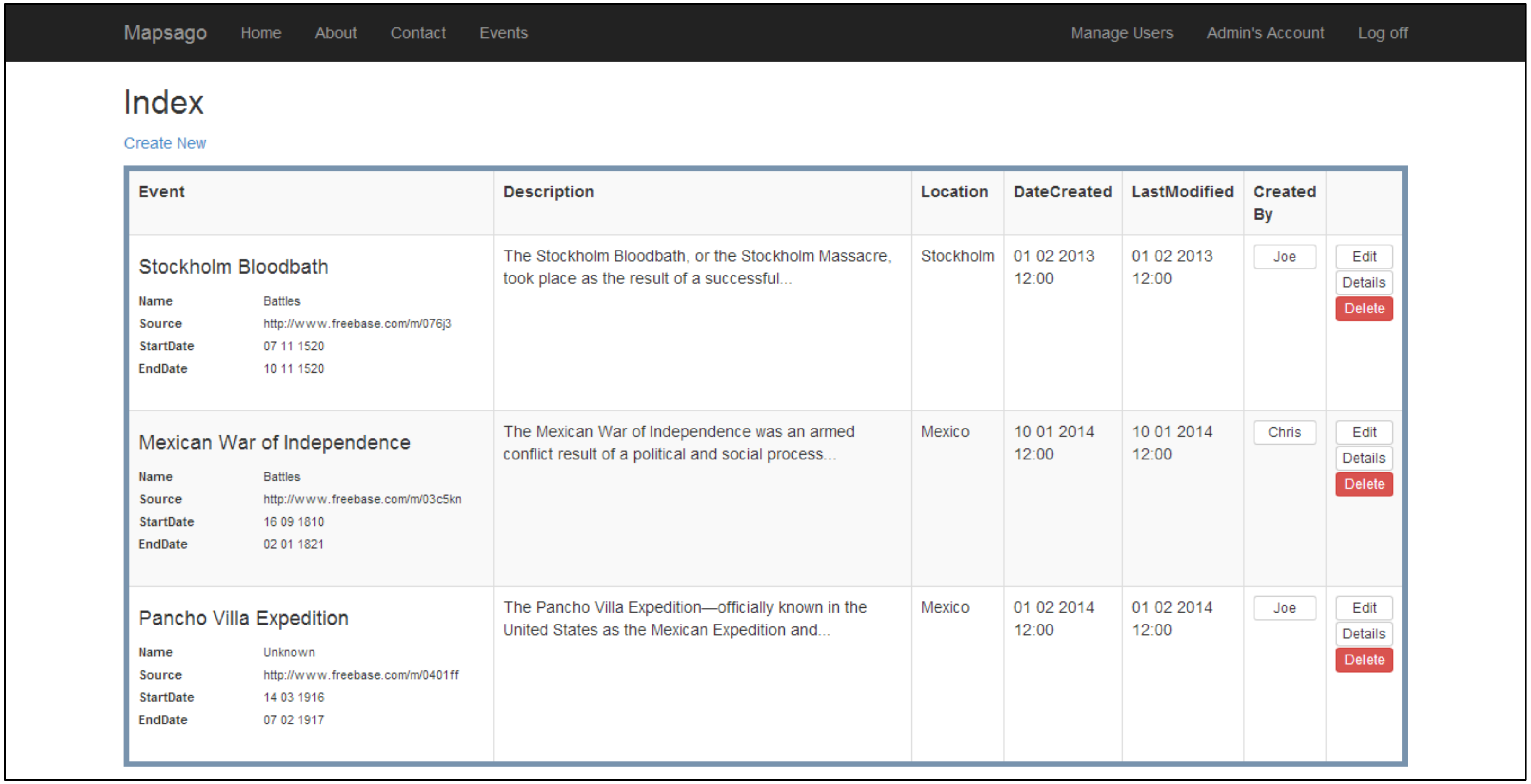
Each Point has three properties: "type", "geometry" (which stores coordinates) and "properties" (which contains all other metadata. Although this format is more complex than is strictly necessary to communicate effectively, it is hoped that the conformity to standards will increase the value and usefulness of the service to third-parties.

Appendix I



Task Name	Resource Names
◢ Enterprise Project	
◢ Technical Report	
Explore concepts	Andrew,Fiona,Adam
Submit proposal	Andrew,Fiona,Adam
Detailed project requirements document	
Draft Report	Fiona[40%]
Add Contributions	Adam[20%],Andrew[20%]
Proof Edit Report	Andrew,Fiona,Adam[20%]
◢ Development	
◢ Learn Development Environment	Adam[20%],Andrew[20%],Fiona[20%]
HTML Template Example	
Domain Model	
Dependency Injection	
Moq (Mock Repository) using the Repository	
DataBase	
Entity Framework	
MVC CRUD	
RelationShip & Associations	
Roles	
Add basic models	Andrew[80%]
Update models for relationships	Andrew[80%]
Add relationships and views	Andrew
Add Linq code to create Location json	Andrew
◢ API	
Test some API samples	Andrew
Add API Controllers	Andrew
◢ Define interfaces based on requirements	
Dashboard interface and code view	Fiona[40%]
Roles interface and code view	Fiona[60%]
◢ Create of Event forms	
User Interface of General Event Forms	Andrew
Different Impletementation of Forms	Andrew
◢ Extensible Module	
FreeBase	Andrew
Interfaces	Andrew
Views	Andrew
API Event , leave it in the MVC project	Adam
Unit testing	Fiona
◢ Entity Creation	
Definition of Users and Created by Databases	Adam
Intergration Testing	Fiona
Deploy	Adam[80%]

Appendix II

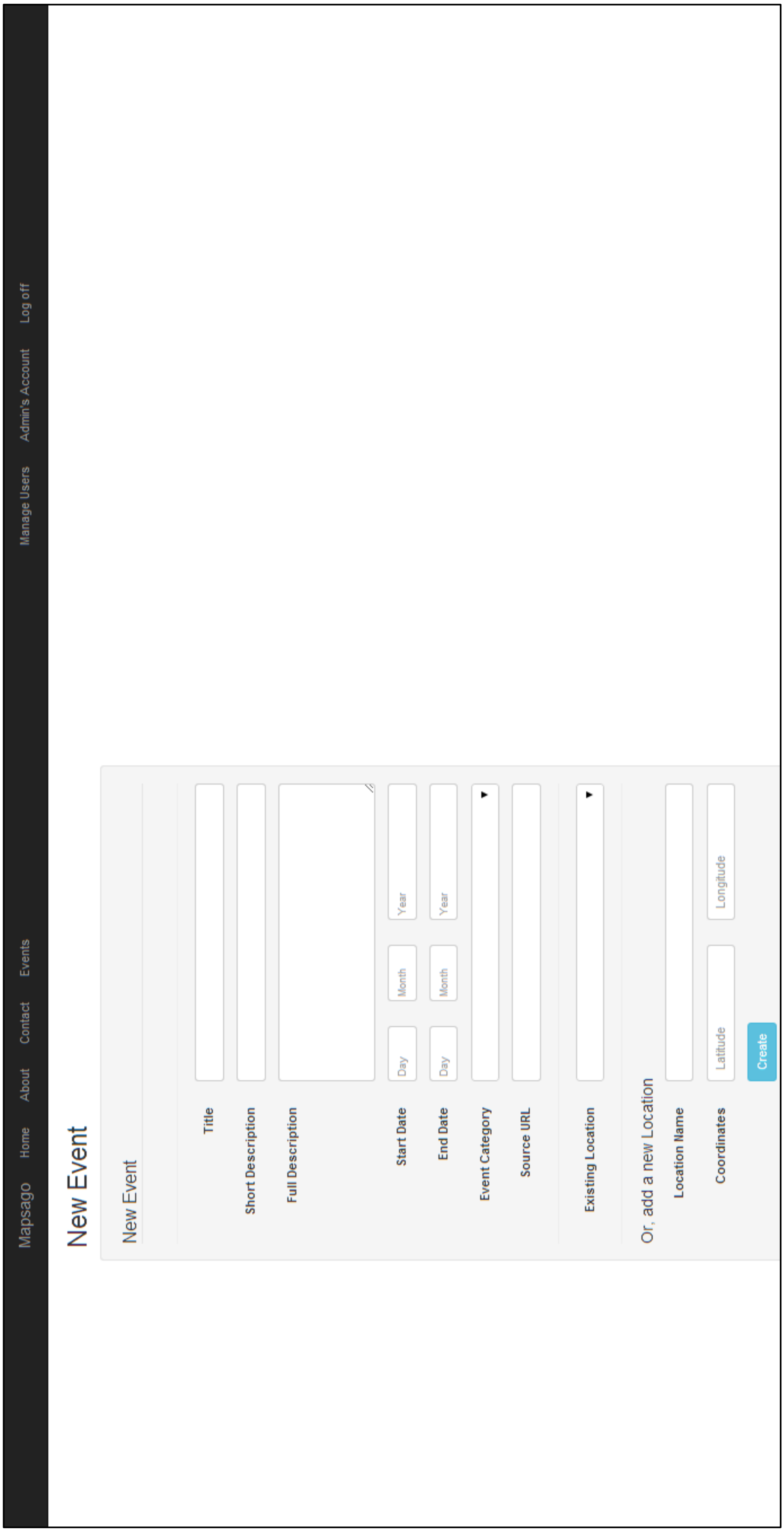


Index

Create New

Event	Description	Location	DateCreated	LastModified	Created By	
Stockholm Bloodbath Name Battles Source http://www.freebase.com/m/076j3 StartDate 07 11 1520 EndDate 10 11 1520	The Stockholm Bloodbath, or the Stockholm Massacre, took place as the result of a successful...	Stockholm	01 02 2013 12:00	01 02 2013 12:00	Joe	Edit Details Delete
Mexican War of Independence Name Battles Source http://www.freebase.com/m/03c5kn StartDate 16 09 1810 EndDate 02 01 1821	The Mexican War of Independence was an armed conflict result of a political and social process...	Mexico	10 01 2014 12:00	10 01 2014 12:00	Chris	Edit Details Delete
Pancho Villa Expedition Name Unknown Source http://www.freebase.com/m/0401ff StartDate 14 03 1916 EndDate 07 02 1917	The Pancho Villa Expedition—officially known in the United States as the Mexican Expedition and...	Mexico	01 02 2014 12:00	01 02 2014 12:00	Joe	Edit Details Delete

Figure 4 Events list view



New Event

New Event

Title

Short Description

Full Description

Start Date Day Month Year

End Date Day Month Year

Event Category

Source URL

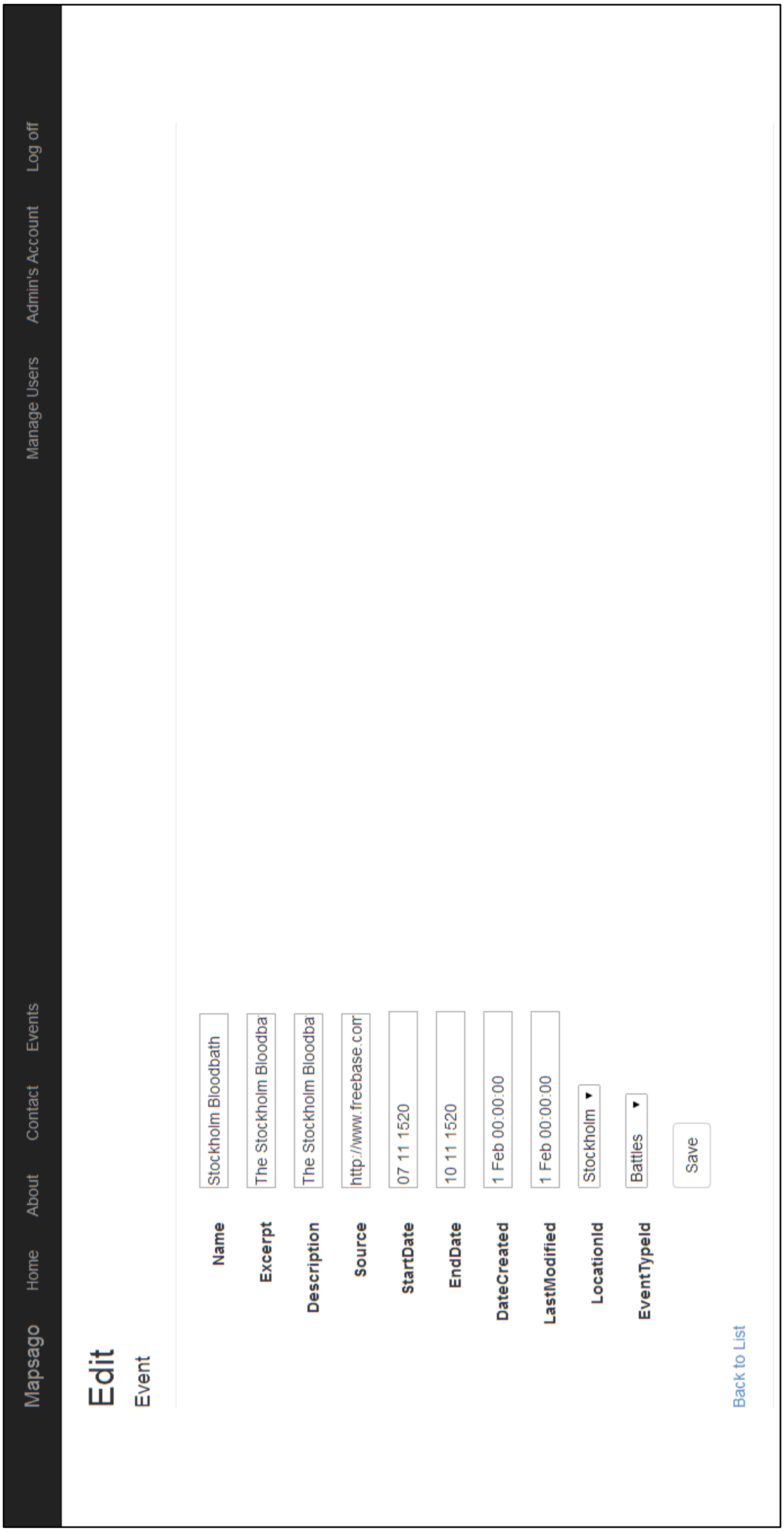
Existing Location

Or, add a new Location

Location Name

Coordinates Latitude Longitude

Figure 5 New event view

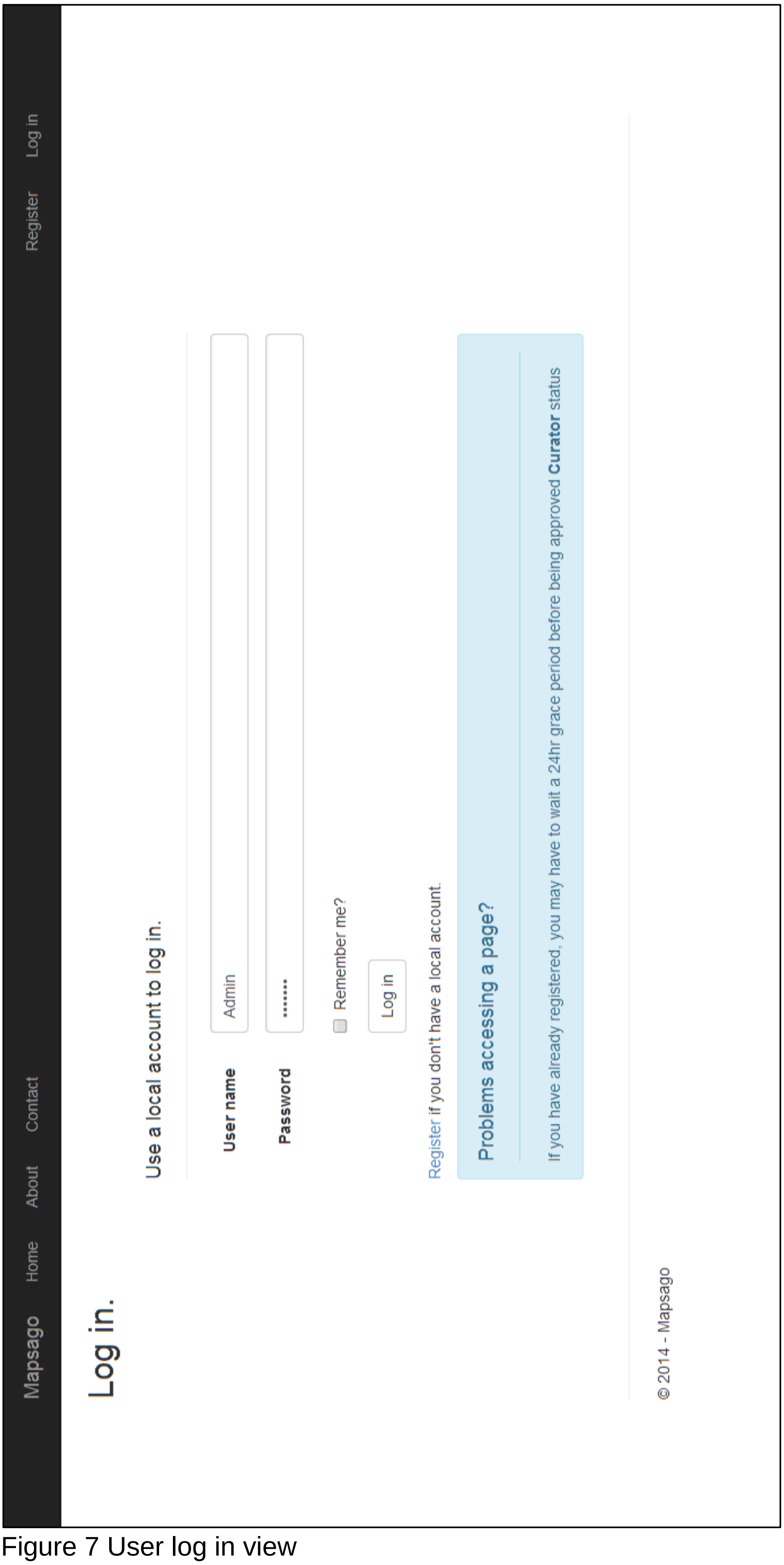


Edit

Event

Name	Stockholm Bloodbath
Excerpt	The Stockholm Bloodba
Description	The Stockholm Bloodba
Source	http://www.freebase.com
StartDate	07 11 1520
EndDate	10 11 1520
DateCreated	1 Feb 00:00:00
LastModified	1 Feb 00:00:00
LocationId	Stockholm ▾
EventTypeId	Battles ▾

Figure 6 Edit Event view



Log in.

Use a local account to log in.

User name Admin

Password •••••••

□ Remember me?

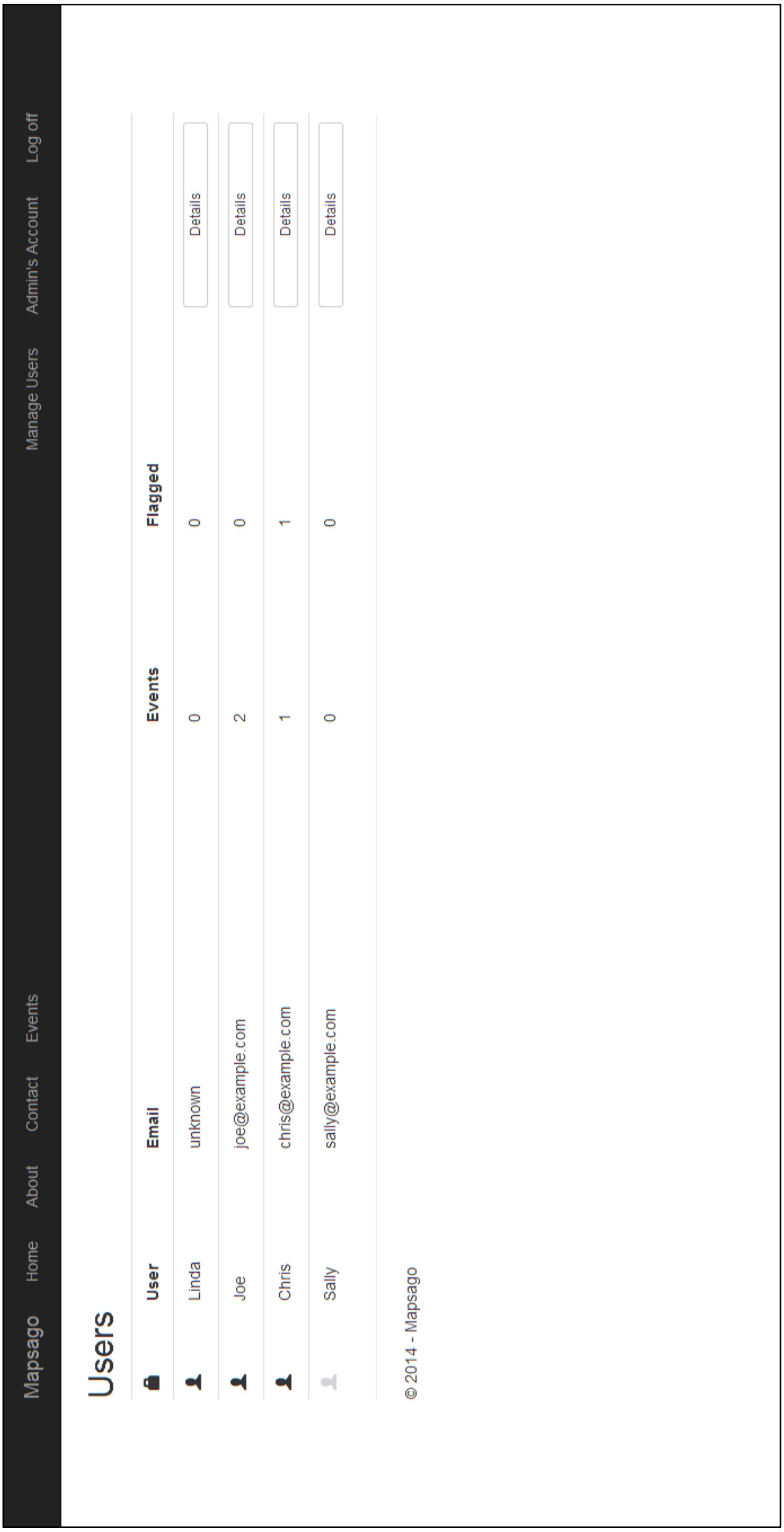
Register if you don't have a local account.

Problems accessing a page?

If you have already registered, you may have to wait a 24hr grace period before being approved **Curator** status

© 2014 - Mapsago

Figure 7 User log in view

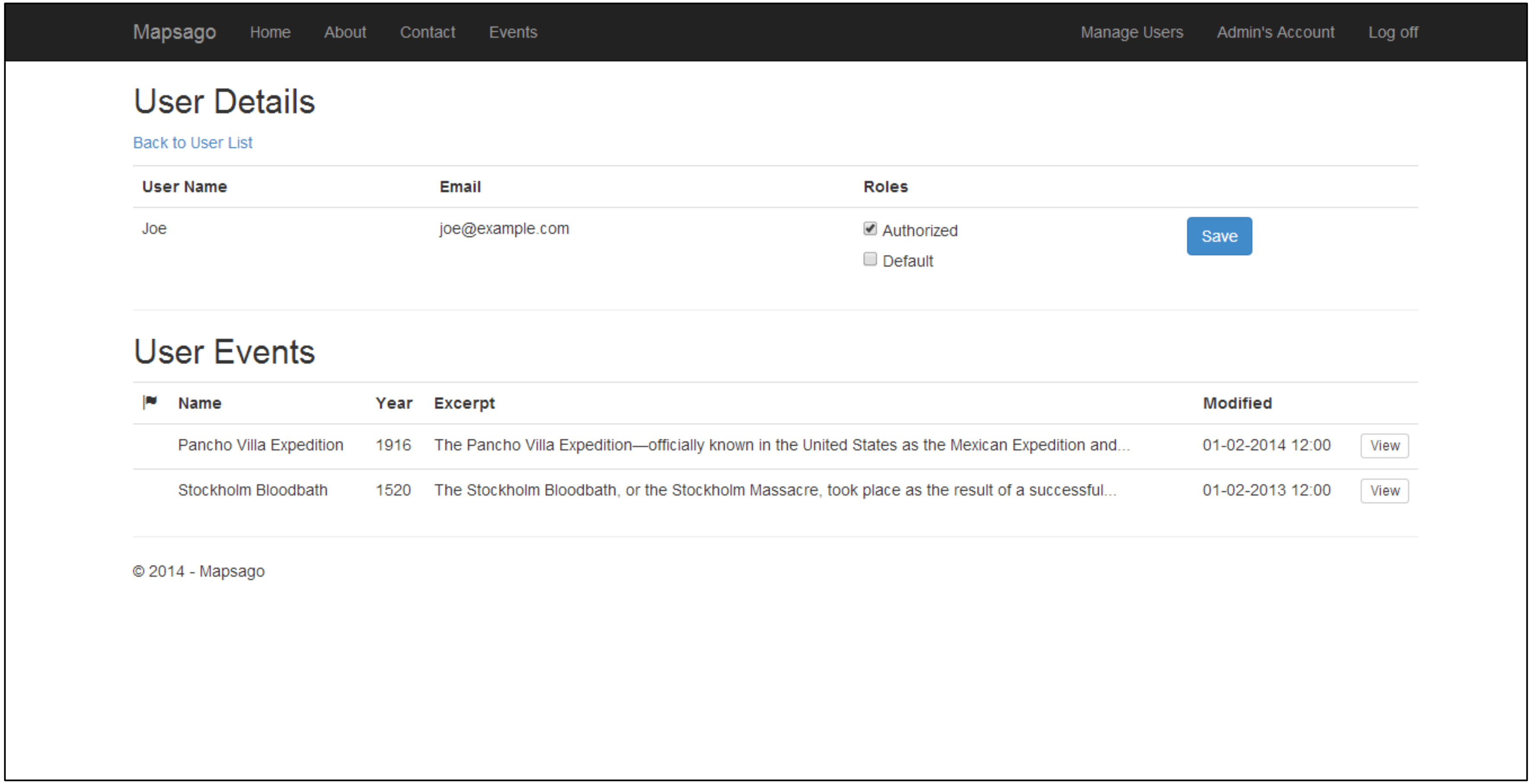


Users

🔒	User	Email	Events	Flagged	
👤	Linda	unknown	0	0	Details
👤	Joe	joe@example.com	2	0	Details
👤	Chris	chris@example.com	1	1	Details
👤	Sally	sally@example.com	0	0	Details

© 2014 - Mapsago

Figure 8 Users view



User Details

User Name	Email	Roles	
Joe	joe@example.com	☑ Authorized □ Default	Save

User Events

⚑	Name	Year	Excerpt	Modified	
	Pancho Villa Expedition	1916	The Pancho Villa Expedition—officially known in the United States as the Mexican Expedition and...	01-02-2014 12:00	View
	Stockholm Bloodbath	1520	The Stockholm Bloodbath, or the Stockholm Massacre, took place as the result of a successful...	01-02-2013 12:00	View

© 2014 - Mapsago

Figure 9 Admin user view