# **CMPT 498: Historical Cemeteries Project - Web Application Development: Frameworks**

by Michael Hnatiw

With the need for our web application to provide an interfaces for different types of end-users to access the available central cemetery repository in an intuitive manner and to provide information visualization features to give end-users a more in-depth look at the information they want to view, it is imperative that the design and structure for the web application be thoroughly thought out to provide portability, expandability, and usability for the present and future. With this in mind, one of the biggest decisions that will have to be made will be the actual use of frameworks to simplify design for the developers, carry out the efficient processing of tasks, and give support modules with which to implement the specified features. While frameworks are used everywhere today, from providing front end aesthetics and easier user interactivity with back end components like a database (ie Twitter Bootstrap), to back end structured code modules and special syntax to help organize and reduce the complexity of processes to be carried out (ie Kohana), there are both pros and cons to using a framework.

General Advantages of Frameworks:

- Provide abstraction from underlying computation and code in some cases, making web development more available to the masses.
- Professional code modules that can be used by newer developers to learn from and develop complex implementations very quickly (would usually take extensive development processes from the ground up otherwise).
- Considerations such as security, user authentication, compatibility with protocols, and even content integration are taken care of by some code modules (less extent for developers to concern themselves with these issues)
- Extensive documentation, resources, and support from a community of developers for different frameworks
- Standardized structure (many websites built upon these frameworks, thus can implement a lot of the same features from existing websites for your own).
- Improvement of code (ie more readable, organized and structured, optimized, etc..)

General Cons:

- Can be viewed as "crutches" due to each framework having their own special syntax, implemented code modules, and rigorous structure which make web developers heavily rely on them in some cases (without learning anything else).

- Can be more time consuming and difficult  to learn specific syntax, semantics, idiosyncrasies and in some cases special languages (ie LESS in Twitter Bootstrap) in order to even uses basic features of a framework) than to start from scratch.
- Framework can limit web development:
  - Customizability of code modules often impossible in some frameworks   (non-portable code that cannot be changed to extend its functions for specific applications)
  - framework may not be good for some tasks (ie querying databases)
  - structure code modules a specific way (could limit implementations of features)
- Production of unnecessary semantics and "spaghetti code" (ie code modules that may not be used or unnecessary syntax and semantics used by the framework to structure code, creating code that could be nearly unreadable and understandable).
- Framework's are everywhere (with everyone using a framework, uniqueness of a websites layout and visuals lessens and everything becomes standardized).

The decision to use a framework comes down to the developers skills, time constraints, and the specific requirements and uses of the web application to be produced.

## <u>Some frameworks to consider:</u>

**<u>Drupal:</u>** is a backend, PHP based framework used for content management that allow s to use embedded modules to carry out features for their website.  Code modules are combinable to extend the functions of a feature.  Multiple layers (ie Data, Modules, Blocks/Menus, User Permissions, Templates) treat different content objects (ie blog post) as "nodes" that are all processed in a standard way to create their view on the webpage.  That means the developer has more flexibility and has a lot of the complexity removed by treating all objects the same. However, in order to do this, the information in nodes needs to be specified  and structures used to process the nodes (ie menus) need to be configured in order to handle and display them on the webpage.  Different templates and themes are integrated that allow users to quickly build and test features added a generic layout for a websites.
https://drupal.org/documentation/concepts

**Advantages:**

**- easily configurable to interact with other websites**
**- data collection and form creation from users are built in**
**- can define own content types (ie specify what makes up a record)**
**- diverse code modules used to implement features, and are combinable**
**- ability to display lists of organized information**

**Disadvantages:**

**- rigorous syntax and semantics that must be learned in order to use even the simplest modules.**
**- need extensive knowledge of the Drupal API in order to code your own custom modules from the ground up.**
**- unnecessary or spaghetti code can be produced due to the structured syntax of the framework.**
**- Some features may not be possible (ie data visualization) due to the limitations of the framework.**

**<u>Kohana</u>:** is a lightweight object oriented HMVC framework used to build components for web applications and structure/separate the user interface, the controls for it, and the core models for it to reduce coding complexity and easily allow users to provide content on their website. Kohana provides structured code plugin's that can be used for a variety of functions (ie user authentication and database access). Not as structured as most frameworks (not specific syntax or semantics other than the groupings of views, controllers, and models). Easily extendable with the freedom to write your own code and or modules (not forced to use existing modules). File system is "cascading", giving a hierarchical organization to how views and files can be presented on the webpage, and provides a more understandable look at how components work together. Also, the Kohana structure can be integrated with other more structured frameworks like Drupal or Zend (since kohana mainly provides a way for the file system to separate code modules).
http://kohanaframework.org/3.0/guide/kohana

**Advantages:**

**- separation of model, view, controller both conceptually and within the file system allows for an organize code base and better understanding of how each module works.**
**- No rigorously structured syntax, free to use existing code modules or to create your own.**
**- easy configuration**
**- OO (everything treated as an object and can be manipulated in ways specified by the developer)**
**- can be integrated with other frameworks (cause at lowest level, Kohana is a way to organize your websites file system)**

**Disadvantages:**

**- Imposes MVC on your code base (must adhere to these concepts in order to make things work).**
**- If modules are not used, they unnecessarily complex your file system and take up space.**

**- need to use specific syntax for existing code modules**
**- No real support for Jquery and JS plugins (although it plugin capability, apparently is finicky)**
**- Documentation and resources (ie tutorials) are not as well defined and documented as with other frameworks.**

**Twitter Bootstrap:** is a collection of built-in components (ie grid layouts) for easily building the frontend of web applications. Contains HTML and CSS structures for aesthetics, and JavaScript extensions for backend interactivity. Uses the LESS stylesheet language that allows for variables, nesting of structures, and functional operations (ie addition) as sort of a meta language to make the building of the websites CSS easier. Is easy to use and only requires minor configuration to integrate onto website HTML pages (ie include scripts).
http://getbootstrap.com/2.3.2/

Advantages:
- easy to use built in components within ones HTML and CSS
- allows for quick prototyping and tests of the front end of the website
- minor configuration to existing code to import and use

Disadvantages:
- Heavy use of LESS stylesheet language (difficult to breakdown existing modules to learn from and use bits and pieces from).
- components are generic and are used everywhere (takes away creativity and uniqueness of webpage design)

**Razor JS (Razor Engine):** is a loosely based framework to embed Razor markup syntax (based on ASP.NET) that lets you insert server-based code (ie C#) into WebPages within JavaScript files. What Razor does is create dynamic content on the fly as the server executes the server-based code before the page is loaded to the browser. This means tasks such as accessing a database can be done dynamically before the page is loaded. What Razor JS allows is the use of this syntax within client-side JavaScript files carry out various tasks for clients (ie Alter results from Google Map search).
http://john.katsiotis.com/blog/razorjs---write-razor-inside-your-javascript-files
http://haacked.com/archive/2011/01/06/razor-syntax-quick-reference.aspx

Advantages:
- Gives more flexibility for tasks to be carried out within the JavaScript code
- Allows tasks to be carried out from the client-side over the server-side
- Can dynamically carry out server-side tasks as the webpage is interacted with.

Disadvantages:
-Overhead of coordinated computation from client-to-server side back and forth.

-Use of potential confusing syntax throughout JavaScript code.

**Node JS:** is a platform built upon Google Chrome's V8 JavaScript Engine to utilize the JavaScript language for easily building scalable network applications. This is possible due to its event-driven, non-blocking I/O API. The platform contains a built in server library capable of running a server without external software like Apache. This platform's libraries also contain built in JavaScript models that allow for easier network programming compared to other languages.
http://nodejs.org/

Advantages:

- Due to its extensive library, daunting networking tasks that could be carried out in other languages are simplified with less complexity and code.
- Event-driven I/O model allows for efficient data processing among distributed systems (allows other processes to run on top of I/O).
- Extensive documentation, tutorials, and examples provided on the Web

Disadvantages:

- High learning curve for inexperienced network programmers to produce any substantial application (platform syntax and semantics are extensive)
- Heavily structured semantics, syntax, and code modules that may limit custom functionality of applications

**ASP.NET with MVC4 Entities:** is an updated framework that utilizes the MVC principles with the established server-side framework ASP.NET to build dynamic and scalable web pages and applications. Using the power of the design pattern of Web-forms provided by ASP.NET, a structured MVC architecture is implemented to provide a better separation among application tasks, allow for extendible modules, lessens the complexity of development, and for the integration of existing ASP.NET files.
http://www.asp.net/mvc/mvc4
http://en.wikipedia.org/wiki/ASP.NET

Advantages:

- easier to manage complexity of application with the separation among controllers, views, and models.
- Supports multiple views for each web page
- Web-form model allows information to persist state over HTTP(S) connections (ie good for application that store sensitive information in form format inputted by end-users)
- Front Controller Patterns allow for information to be processed through a single controller (ie robust routing capabilities due and infrastructure)
- Page Controller pattern can add different functionality to different pages

- Extensive documentation and large community support

Disadvantages:

- Does not save the view state of the current page (need to learn and use the web-form model for page information to persist over a connection)
-imposed MVC framework that must be complied with in order for implementations to work, lose support for integrating framework with existing applications as well.
- If syntactical languages are used for functionality (ie Razor), framework does not support this and could break existing code.
-Added complexity of MVC framework integrated with ASP.NET can making learning the syntax and semantics difficult and time consuming.

## Discussion:

Given the discussed pros and cons of each framework, the functionality and attributes of each, and the relevance of each framework to this current project; the decision to use which (or any at all) is still a complex one. The current project requires and entails:

- **End-users interacting with the site to query cemetery records and information from the database, as well as the insertion and integration of data provided by the end-users into the central database**
    1. **Searches**: using typical search boxes, radio buttons, dropdown menus, etc.. for the user to choose and pick **what to search and how to filter the search**.
       - 2 levels of searching through records that query different information from the database:
         -**Cemetery**: search for specific cemetery info
         - **Individual**: search for a specified individual gravestone within a cemetery, across cemeteries, bring up related individuals, etc..
       - **Filters** encompass many different functionalities (from ranges of values, to reducing results returned, to specifying certain area to search in, etc..)
    2. **Submitting**: using some sort of form API to prompt cemetery record information from users and store it in the central database.
       - use of **required fields** (minimum information o submit)
       - requirement for **persistent information to remain on the webpage** (if user doesn't meet required fields at first, goes back a page, etc..)

- validation of information (ie that it conforms to the database structure) before insertion)

*** restricting only permitted users to be able to submit data.**

3. __**Visualizations**__: support for visualizations to be carried out from returned results of a search (ie display layout of graves when searching for specified cemetery).

    *** not sure what would be best to support this at this point.

- **The creation and management of different types of User accounts that have differing permission boundaries when accessing database information and using available features, and different account features.**
    - 3 different types of users **(Trusted Public, Researcher, Administrator)**
    - **Authentication process**
        - **Login and Account Creation** Pages (and subsequent prompting to login into an account to submit data)
        - **Checking user permissions to signal what information will be available to display**
    - **Account Interface/Dashboard**
        - **Tabbed menu** for features that users can access (ie view submitted data, export data, etc..)
            - Trusted Public: can view previously submitted data
            - Researchers: same as Trusted Public and can **export data**
            - Administrators: same as Researcher and can Manage User accounts (ie rewrite permission, modify/remove accounts)
        - **Personal Account management** (ie account settings, profile, etc..).

- **Ability to accommodate source repository changes, changes to the central repository, to import/export data to and from the central repository, and the ability to expand the type/kinds of data that can be elicited from Alberta's Historical Cemeteries**

    *** **Importer and Exporter Modules that will be built out of scripts do not apply to the web Interface** (at least not right now)

    - **Expandability**: The ability for the web application to handle new features if need be (ie support a forum) and for the functions of the existing features to be expanded (ie if new types of data found, integrate into visualizations and maps etc..)

- **Capacity for frameworks to either support or can integrate visualization modules need for the Map features (ie Overview, Layout)**.

Given the specified requirements and features to be implemented for this project, each framework can provide the following things:

- **Drupal**: While Drupal provides functionality for form-input and the ability to organize and display listed information easily, the fact that it has rigorous syntax, innate restrictions for features due to modules (ie for visualizations), and that developers will need a good knowledge of the API to extend the existing modules or create your own; creates alot of extra complexity that isn't needed in this project. In my opinion, **I don't think Drupal will be suitable for this project** (at least given the time constraints and the skills of its developers).
- **Kohana**: This framework provides a logically sound way to organize how will work (through the use of the HMVC method), is easily configurable and does not require rigorous syntax to accommodate its structure, is Object oriented, and has a file system layout that is well organized and simplifies things for developers. Given all these advantages, Kohana does not reliably support JS and Jquery plug-ins that will probably be essential for Data visualizations. As well, there may be other MVC frameworks with features more suitable to support our Database and user authentication needs**.** **Kohana is a plausibility to use** (at least some aspects).
- **Twitter Bootstrap**: provides developers with a quick and easy way to integrate already made UI components into their web application with support for both JS and Jquery. While this module may be good for prototyping web layouts for pages during development, the generic UI components take the originality away from the application. As well, the heavy reliance on the LESS stylesheet language may make it hard for new developers to decipher how a UI element works to either build upon it's functionality or deduce what might be wrong if a fault occurs while using it. **Depending on whether it is decided that this current project will need a clean generic aesthetic provided by bootstrap, then this framework will not be used.**
- *** RazorJS:** this markup syntax language that can be used within JS scripts allows for processes and tasks to be executed before the webpage is loaded, which can be ideal for database querying and loading visualizations for them. This also allows for the client-side to take care of some of the dynamic computation. While coordinating the code's syntax to work with the razor engine will take a little studying, **I think this syntax will give great support for database querying and data visualization for this project**.

- **NodeJS**: is a framework that centers its focus around providing easy modules for networking among applications. With its even-drive I/O framework, it may be useful if our application need to communicate with other websites, and not just the source repositories. Since the communication will be taken care of on the backend, it is not necessary for the web application. Thus, unless in the future the website is expanded and requires heavy networking with other applications, **I don't see the need for this highly structured framework to be used here**.
- *** ASP.NET MVC4**: is once again an MVC principled framework, but built on ASP.NET. The web-form and web-page interfaces allow for the acquisition of information from users and can add different functionalities for the same webpage. The problem though is that data does not persist among WebPages (meaning user inputted information will not persist if pages are changed). There might be a way around this fault, but more research is needed. As well, existing code that uses syntactical languages like Razor may have their functionality broken if this framework is integrated (not really a concern unless they want to expand the project in the future and try to integrate other code into it). Despite these faults, I think the usefulness of an MVC structured framework, that allows for good data processing from users and modules for almost anything can have its benefits for both parsing, querying, and displaying the information to users. **So the decision stands that this framework might be used**.

  * Most useful when coupled with maybe **RazorJS** from a ground up implementation.