# CHAPTER TWO LITERATURE REVIEW

In the process of completing this project, the developers have taken upon to properly research the literature that already exists for the Web Based Application, Single-Page Application, Ajax technology, Payroll System, and Model-View-Controller (MVC) design pattern. The purpose is to get a better understanding on the subject before going further into the research and design phase of the development.

As such this paper will detail the sources in which the developer refer to and the knowledge gathered by the process. The main point of this research consists of example, and type of preexisting system that currently in used for the purpose of student information.

## 2.1: Web based application

A web based application is an application that is accessed by users over a network such as the Internet or an intranet. The term may also mean a computer software application that is coded in a browser-supported programming language (such as JavaScript, combined with a browser-rendered markup language like HTML) and reliant on a common web browser to render the application executable.

Web based applications are popular due to the ubiquity of web browsers, and the convenience of using a web browser as a client, sometimes called a thin client. The ability to update and maintain web based applications without distributing and installing software on potentially thousands of client computers is a key reason for their popularity, as is the inherent support for cross-platform compatibility. [1]

### 2.1.1: History

In earlier computing models, e.g. in client-server, the load for the application was shared between code on the server and code installed on each client locally. In other words, an application had its own client program which served as its user interface and had to be separately installed on each user's personal computer. An upgrade to the server-side code of the application would typically also require an upgrade to the client-side code installed on each user workstation, adding to the support cost and decreasing productivity.

In contrast, web applications use web documents written in a standard format such as HTML and JavaScript, which are supported by a variety of web browsers. Web applications can be considered as a specific variant of client-server software where the client software is downloaded to the client machine when visiting the relevant web page, using standard procedures such as HTTP. Client web software update may happen each time the web page is visited. During the session, the web browser interprets and displays the pages, and acts as the universal client for any web application.

In the early days of the Web each individual web page was delivered to the client as a static document, but the sequence of pages could provide an interactive experience, as user input is returned through web form elements embedded in the page markup.

In 1995 Netscape introduced a client-side scripting language called JavaScript allowing programmers to add some dynamic elements to the user interface that ran on the client side. So instead of sending data to the server in order to generate an entire web page, the embedded scripts of the downloaded page can perform various tasks such as input validation or showing/hiding parts of the page.

In 1996, Macromedia introduced Flash, a vector animation player that could be added to browsers as a plug-in to embed animations on the web pages. It allowed the use of a scripting language to program interactions on the client side with no need to communicate with the server.

In 1999, the "web application" concept was introduced in the Java language in the Servlet Specification version 2.2. At that time both JavaScript and XML had already been developed, but Ajax had still not yet been coined and the XMLHttpRequest object had only been recently introduced on Internet Explorer 5 as an ActiveX object.

In 2005, the term Ajax was coined, and applications like Gmail started to make their client sides more and more interactive. A web page script is able to contact the server for storing/retrieving data without downloading an entire web page.

In 2011, HTML5 was finalized, which provides graphic and multimedia capabilities without the need of client side plugins. HTML5 also enriched the semantic content of documents. The APIs and document object model (DOM) are no longer afterthoughts, but are fundamental parts of the HTML5 specification. WebGL API paved the way for advanced 3D graphics based on HTML5 canvas and JavaScript language. These have significant importance in creating truly platform and browser independent rich web applications. [1]

### 2.1.2: Interface

Web developers often use client-side scripting to add functionality, especially to create an interactive experience that does not require page reloading. Recently, technologies have been developed to coordinate client-side scripting with server-side technologies such as PHP. Ajax, a web development technique using a combination of various technologies, is an example of technology which creates a more interactive experience. [1]

### 2.1.3: Structure

Applications are usually broken into logical chunks called "tiers", where every tier is assigned a role. Traditional applications consist only of 1 tier, which resides on the client machine, but web applications lend themselves to a n-tiered approach by nature. Though many variations are possible, the most common structure is the three-tiered application. In its most common form, the three tiers are called presentation, application and storage, in this order. A web browser is the first tier (presentation), an engine using some dynamic Web content technology (such as ASP, ASP.NET, CGI, ColdFusion, JSP/Java, PHP, Perl, Python, Ruby on Rails or Struts2) is the middle tier (application logic), and a database is the third tier (storage). The web browser sends requests to the middle tier, which services them by making queries and updates against the database and generates a user interface. [1]

### 2.1.4: Business use

An emerging strategy for application software companies is to provide web access to software previously distributed as local applications. Depending on the type of application, it may require the development of an entirely different browser-based interface, or merely adapting an existing application to use different presentation technology. These programs allow the user to pay a monthly or yearly fee for use of a software application without having to install it on a local hard drive. A company which follows this strategy is known as an application service provider (ASP), and ASPs are currently receiving much attention in the software industry.

Security breaches on these kinds of applications are a major concern because it can involve both enterprise information and private customer data. Protecting these assets is an important part of any web application and there are some key operational areas that must be included in the development process. This includes processes for authentication, authorization, asset handling, input, and logging and auditing. Building security into the applications from the beginning can be more effective and less disruptive in the long run.

In cloud computing model web applications are software as a service (SaaS). There are business applications provided as SaaS for enterprises for fixed or usage dependent fee. Other web applications are offered free of charge, often generating income from advertisements shown in web application interface. [1]

### 2.1.5: Web application development

Writing of web applications is often simplified by open source software such as Django, Ruby on Rails or Symfony called web application frameworks. These frameworks facilitate rapid application development by allowing a development team to focus on the parts of their application which are unique to their goals without having to resolve common development issues such as user management. While many of these frameworks are open source, this is by no means a requirement.

The use of web application frameworks can often reduce the number of errors in a program, both by making the code simpler, and by allowing one team to concentrate on the framework while another focuses on a specified use case. In applications which are exposed to constant hacking attempts on the Internet, security-related problems can be caused by errors in the program. Frameworks can also promote the use of best practices such as GET after POST.

In addition, there is potential for the development of applications on Internet operating systems, although currently there are not many viable platforms that fit this model. [1]

### 2.1.6: Benefits

* Web applications do not require any complex "roll out" procedure to deploy in large organizations. A compatible web browser is all that is needed.
* Browser applications typically require little or no disk space on the client.
* They require no upgrade procedure since all new features are implemented on the server and automatically delivered to the users.
* Web applications integrate easily into other server-side web procedures, such as email and searching.
* They also provide cross-platform compatibility in most cases (i.e., Windows, Mac, Linux, etc.) because they operate within a web browser window.
* With the advent of HTML5, programmers can create richly interactive environments natively within browsers. Included in the list of new features are native audio, video and animations, as well as improved error handling. [1]

### 2.1.7: Drawbacks

* In practice, web interfaces, compared to thick clients, typically force significant sacrifice to user experience and basic usability.
* Web applications absolutely require compatible web browsers. If a browser vendor decides not to implement a certain feature, or abandons a particular platform or operating system version, this may affect a huge number of users.
* Standards compliance is an issue with any non-typical office document creator, which causes problems when file sharing and collaboration becomes critical.
* Browser applications rely on application files accessed on remote servers through the Internet. Therefore, when connection is interrupted, the application is no longer usable. However, if it uses HTML5 API's such as Offline Web application caching,[10] it can be downloaded and installed locally, for offline use. Google Gears, although no longer in active development, is a good example of a third party plugin for web browsers that provides additional functionality for creating web applications.
* Since many web applications are not open source, there is also a loss of flexibility, making users dependent on third-party servers, not allowing customizations on the software and preventing users from running applications offline (in most cases). However, if licensed, proprietary software can be customized and run on the preferred server of the rights owner.
* They depend entirely on the availability of the server delivering the application. If a company goes bankrupt and the server is shut down, the users have little recourse. Traditional installed software keeps functioning even after the demise of the company that produced it (though there will be no updates or customer service).
* Likewise, the company has much greater control over the software and functionality. They can roll out new features whenever they wish, even if the users would like to wait until the bugs have been worked out before upgrading. The option of simply skipping a weak software version is often not available. The company can foist unwanted features on the users or cut costs by reducing bandwidth. Of course, companies will try to keep the good will of their customers, but the users of web applications have fewer options in such cases unless a competitor steps in and offer a better product and easy migration.
* The company can theoretically track anything the users do. This can cause privacy problems. [1]

## 2.2: Single-page application

A single-page application (SPA), also known as single-page interface (SPI), is a web application or web site that fits on a single web page with the goal of providing a more fluid user experience akin to a desktop application.

In an SPA, either all necessary code – HTML, JavaScript, and CSS – is retrieved with a single page load, or partial changes are performed loading new code on demand from the web server, usually driven by user actions. The page does not automatically reload during user interaction with the application, nor does control transfer to another page. Updates to the displayed page may or may not involve interaction with a server. [2]

### 2.2.1: Architectural characteristics

#### 2.2.1.1: The problem

The way traditional web applications work causes disruption in the user experience and workflow.

Traditional web applications work by reloading the entire web page. In order to advance through a workflow, the user interacts with page elements (such as hyperlinks and form submit buttons) that cause the browser to issue a request to the server for a completely new page.

**User experience**: Continual page redraws disrupt the user experience because the network latencies cannot be hidden from the user. There is typically a perceivable transitional jolt from one page to the next. The next page's data is retrieved from the server, the old page is unloaded, and the new page is rendered to screen.

Stable UI affordances, such as toolbars, navigation elements, database query results and so forth, continually disappear and reappear.

**Performance**: The complete page reload that occurs on each user interaction results in unnecessary re-transmission of data over the wire. This can make the overall performance of the web-site, when the entire session is taken into account, slower.

#### 2.2.1.2: How SPAs address the problem

SPAs address these issues by requiring no page reload by the browser during an application session. All user interaction and changes of the application state are handled in the context of a single Web document.

The user experience becomes more continuous and fluid, and network latencies can be hidden more easily. [2]

### 2.2.2: Technical approaches

There are various techniques available that enable the browser to retain a single page even when the application requires server communication.

**AJAX**

The most prominent technique currently being used is Ajax. Predominantly using the XMLHttpRequest object from JavaScript, other AJAX approaches include using IFRAME or script HTML elements. Popular libraries like jQuery, that normalize AJAX behavior across browsers from different manufacturers, have further popularized the AJAX technique.

**Node.js/SignalR**

Asynchronous calls to the server may also be achieved using Node.js or SignalR in conjunction with Socket.io.

**Browser plugins**

Asynchronous calls to the server may also be achieved using browser plug-in technologies such as Silverlight, Flash or Java applets.

**Data transport (XML, JSON and AJAX)**

Requests to the server typically result in either raw data (e.g. XML or JSON), or new HTML being returned. In the case where HTML is returned by the server, JavaScript on the client updates a partial area of the DOM (Document Object Model). When raw data is returned, oftentimes a client-side JavaScript XML / (XSL) process (and in case of JSON a template) is used to translate the raw data into HTML, which is then used to update a partial area of the DOM.

**Thin server architecture**

An SPA moves logic from the server to the client. This results in the role of the web server evolving into a pure data API or web service. This architectural shift has, in some circles, been coined "Thin Server Architecture" to highlight that complexity has been moved from the server to the client, with the argument that this ultimately reduces overall complexity of the system.

**Thick server architecture**

The server keeps the necessary state in memory of the client state of the page. In this way, when any request hits the server (usually user actions), the server sends the appropriate HTML and/or JavaScript with the concrete changes to bring the client to the new desired state (usually adding/deleting/updating a part of the client DOM). At the same time the state in server is updated. Most of the logic is executed in server and HTML is usually also rendered in server. In some ways the server simulates a web browser, receiving events and performing delta changes in server state which are automatically propagated to client. This approach needs more server memory and server processing, but the advantage is a simplified development model because a) the application is usually fully coded in server, b) data and UI state in server are shared in the same memory space with no need of custom client/server communication bridges. [2]

### 2.2.3: Running locally

Some SPAs may be executed from a local file using the file URI scheme. This gives users the ability to download the SPA from a server and run the file from a local storage device, without depending on server connectivity. If such an SPA wants to store and update data, it must be self-modifying. That is, the SPA must be capable of writing itself to disk, including a representation of the state that is to be persisted. These applications benefit from advances available with HTML5, particularly Web Storage. [2]

### 2.2.4: Challenges with the SPA model

Because the SPA is an evolution away from the stateless page-redraw model that browsers were originally designed for, some new challenges have emerged. Each of these problems has an effective solution with:

* Client-side JavaScript libraries addressing various issues.
* Server side web frameworks that specialize in the SPA model.
* The evolution of browsers and the HTML5 specification aimed at the SPA model.

#### 2.2.4.1: Browser history

With an SPA being, by definition, "a single page", the model breaks the browser's design for page history navigation using the Forward/Back buttons. This presents a usability impediment when a user presses the back button, expecting the previous screen state within the SPA, but instead the application's single page unloads and the previous page in the browser's history is presented.

The traditional solution for SPA's has been to change the browser URL's hash fragment identifier in accord with the current screen state. This can be achieved with JavaScript, and causes URL history events to be built up within the browser. As long as the SPA is capable of resurrecting the same screen state from information contained within the URL hash, the expected back button behavior is retained.

To further address this issue, the HTML5 specification has introduced pushState and replaceState providing programmatic access to the actual URL and browser history. [2]

### 2.2.5: Page lifecycle

An SPA is fully loaded in the initial page load and then page regions are replaced or updated with new page fragments loaded from the server on demand. To avoid excessive downloading of unused features, an SPA will often progressively download more features as they become required, either small fragments of the page, or complete screen modules.

In this way an analogy exists between "states" in an SPA and "pages" in a traditional web site. Because "state navigation" in the same page is analogous to page navigation, in theory any page based web site could be converted to single-page replacing in the same page only the changed parts result of comparing consecutive pages in a non-SPA.

The SPA approach on the web is similar to the Single Document Interface (SDI) presentation technique popular in native desktop applications. [2]

## 2.3: Ajax

Ajax (also AJAX; an acronym for Asynchronous JavaScript and XML) is a group of interrelated web development techniques used on the client-side to create asynchronous web applications. With Ajax, web applications can send data to, and retrieve data from, a server asynchronously (in the background) without interfering with the display and behavior of the existing page. Data can be retrieved using the XMLHttpRequest object. Despite the name, the use of XML is not required (JSON is often used instead), and the requests do not need to be asynchronous.

Ajax is not a single technology, but a group of technologies. HTML and CSS can be used in combination to mark up and style information. The DOM is accessed with JavaScript to dynamically display, and to allow the user to interact with the information presented. JavaScript and the XMLHttpRequest object provide a method for exchanging data asynchronously between browser and server to avoid full page reloads. [3]

[](http://www.adaptivepath.com/uploads/archive/images/publications/essays/ajax-fig1.png)

Figure 1: The traditional model for web applications (left) compared to the Ajax model (right).

*http://www.adaptivepath.com/uploads/archive/images/publications/essays/ajax-fig1\_small.png*

### 2.3.1: History

In the 1990s, most web sites were based on complete HTML pages. Each user action required that the page be reloaded from the server (or a new page loaded). This process was inefficient, as reflected by the user experience: all page content disappeared then reappeared. Each time a page was reloaded due to a partial change, all of the content had to be re-sent, even though only some of the information had changed. This placed additional load on the server and used excessive bandwidth.

In 1996, the iframe tag was introduced by Internet Explorer to load content asynchronously.

In 1998, Microsoft Outlook Web Access team implemented the first component XMLHTTP by client script.

In 1999, Microsoft utilized its iframe technology to dynamically update the news stories and stock quotes on the default page for Internet Explorer, and created the XMLHTTP ActiveX control in Internet Explorer 5, which was later adopted by Mozilla, Safari, Opera and other browsers as the XMLHttpRequest JavaScript object. Microsoft has adopted the native XMLHttpRequest model as of Internet Explorer 7, though the ActiveX version is still supported. The utility of background HTTP requests to the server and asynchronous web technologies remained fairly obscure until it started appearing in full scale online applications such as Outlook Web Access (2000) and Oddpost (2002).

Google made a wide deployment of standards-compliant, cross browser Ajax with Gmail (2004) and Google Maps (2005).

The term Ajax was coined on 18 February 2005 by Jesse James Garrett in an article entitled "Ajax: A New Approach to Web Applications", based on techniques used on Google pages.

On 5 April 2006 the World Wide Web Consortium (W3C) released the first draft specification for the XMLHttpRequest object in an attempt to create an official web standard. [3]

### 2.3.2: Technologies

The term Ajax has come to represent a broad group of web technologies that can be used to implement a web application that communicates with a server in the background, without interfering with the current state of the page. The following technologies are incorporated in Ajax:

* HTML (or XHTML) and CSS for presentation
* The Document Object Model (DOM) for dynamic display of and interaction with data
* XML for the interchange of data, and XSLT for its manipulation
* The XMLHttpRequest object for asynchronous communication
* JavaScript to bring these technologies together

There have been a number of developments in the technologies used in an Ajax application, and the definition of the term Ajax. XML is not required for data interchange and therefore XSLT is not required for the manipulation of data. JavaScript Object Notation (JSON) is often used as an alternative format for data interchange, although other formats such as preformatted HTML or plain text can also be used. [3]

### 2.3.3: Drawbacks

* In pre-HTML5 browsers, pages dynamically created using successive Ajax requests did not automatically register themselves with the browser's history engine, so clicking the browser's "back" button may not have returned the browser to an earlier state of the Ajax-enabled page, but may have instead returned to the last full page visited before it. Such behavior — navigating between pages instead of navigating between page states — may be desirable, but if fine-grained tracking of page state is required then a pre-Ajax workaround was to use invisible iframes to trigger changes in the browser's history. A workaround implemented by Ajax techniques is to change the URL fragment identifier (the part of a URL after the '#') when an Ajax-enabled page is accessed and monitor it for changes. HTML5 provides an extensive API standard for working with the browser's history engine.
* Dynamic web page updates also make it difficult to bookmark and return to a particular state of the application. Solutions to this problem exist, many of which again use the URL fragment identifier. The solution provided by HTML5 for the above problem also applies for this.
* Depending on the nature of the Ajax application, dynamic page updates may interfere disruptively with user interactions, especially if working on an unstable Internet connection. For instance, editing a search field may trigger a query to the server for search completions, but the user may not know that a search completion popup is forthcoming, and if the internet connection is slow, the popup list may show up at an inconvenient time, when the user has already proceeded to do something else.
* Because most web crawlers do not execute JavaScript code, publicly indexable web applications should provide an alternative means of accessing the content that would normally be retrieved with Ajax, thereby allowing search engines to index it.
* Any user whose browser does not support JavaScript or XMLHttpRequest, or simply has this functionality disabled, will not be able to properly use pages which depend on Ajax. Devices such as smartphones and PDAs may not have support for the required technologies, though this is becoming less of an issue. The only way to let the user carry out functionality is to fall back to non-JavaScript methods. This can be achieved by making sure links and forms can be resolved properly and not relying solely on Ajax.
* Similarly, some web applications which use Ajax are built in a way that cannot be read by screen-reading technologies, such as JAWS. The WAI-ARIA standards provide a way to provide hints in such a case.
* Screen readers that are able to use Ajax may still not be able to properly read the dynamically generated content.
* The same origin policy prevents some Ajax techniques from being used across domains, although the W3C has a draft of the XMLHttpRequest object that would enable this functionality. Methods exist to sidestep this security feature by using a special Cross Domain Communications channel embedded as an iframe within a page, or by the use of JSONP.
* The asynchronous callback-style of programming required can lead to complex code that is hard to maintain, to debug and to test. [3]

## 2.4: Payroll System

A payroll system involves everything that has to do with the payment of employees and the filing of employment taxes. This includes keeping track of hours, calculating wages, withholding taxes and other deductions, printing and delivering checks and paying employment taxes to the government.

Withholding and paying taxes is one of the most important responsibilities of the payroll system. In the United States, the following are the major withholdings required by the government:

* Federal income tax
* State and local income taxes (where applicable)
* Social Security tax
* Medicare tax

When an employer withholds taxes from a paycheck, he acts as the trustee for those funds until they are paid to the IRS, the Social Security Administration (SSA) or other government agency. To avoid confusing this money with profits or other business income, all withheld taxes must be held in a separate bank account or trust fund.

In the case of Social Security and Medicare withholdings, when it's time to hand that money over to the government, the employer is required to match the employee's contributions. For example, if an employee is paying 6.2 percent of every check for Social Security, then the employer has to pay an equal 6.2 percent.

In addition to matching Social Security and Medicare contributions, the employer has to pay federal and state unemployment taxes (FUTA and SUTA) for each employee. The employer pays these taxes himself, meaning nothing is withheld from the paycheck.

There are numerous other possible deductions, withholdings and contributions that can be subtracted from an employee's gross wages and that need to be tracked by the payroll system:

* Health insurance or life insurance premiums
* 401(k) or other retirement fund contributions
* Workman's compensation
* Union dues
* Vacation days
* Sick days
* Employee loans
* Court-ordered wag­e garnishments (for outstanding debts)
* Child support payments

At the end of the year, an employer uses the payroll system to take all of the wage and withholding information from the previous year and summarize it on a W-2 form for full-time employees or a 1099 form for contract workers. Copies of that form must be sent to the employee, the IRS and the SSA. [4]

## 2.5: Model-view-controller (MVC) design pattern

Model–view–controller (MVC) is a software architecture pattern that separates the representation of information from the user's interaction with it. The model consists of application data, business rules, logic, and functions. A view can be any output representation of data, such as a chart or a diagram. Multiple views of the same data are possible, such as a bar chart for management and a tabular view for accountants. The controller mediates input, converting it to commands for the model or view. The central ideas behind MVC are code reusability and separation of concerns. [5]

### 2.5.1: History

The model-view-controller pattern was originally formulated in the late 1970s by Trygve Reenskaug at Xerox PARC, as part of the Smalltalk system. [5]

### 2.5.2: Component interactions

In addition to dividing the application into three kinds of components, the MVC design defines the interactions between them.

* A controller can send commands to its associated view to change the view's presentation of the model (e.g., by scrolling through a document). It can also send commands to the model to update the model's state (e.g., editing a document).
* A model notifies its associated views and controllers when there has been a change in its state. This notification allows the views to produce updated output, and the controllers to change the available set of commands. A passive implementation of MVC omits these notifications, because the application does not require them or the software platform does not support them.
* A view requests from the model the information that it needs to generate an output representation. [5]

[](http://en.wikipedia.org/wiki/File:MVC-Process.png)

Figure 2: A typical collaboration of the MVC components.

*http://upload.wikimedia.org/wikipedia/commons/thumb/f/fd/MVC-Process.png/200px-MVC-Process.png*

### 2.5.3: Dependency hierarchy

There is usually a kind of hierarchy in the MVC pattern. The Model knows only about itself. That is, the source code of the Model has no references to either the View or Controller.

The View however, knows about the Model. It will poll the Model about the state, to know what to draw. That way, the View can draw something that is based on what the Model has done. But the View knows nothing about the Controller.

The Controller knows about both the Model and the View. To take an example from a game: If you click on the "fire" button on the mouse, the Controller knows what fire function in the Model to call. If you press the button for switching between first and third person, the Controller knows what function in the View to call to request the display change.

The reason to keep it this way is to minimize dependencies. No matter how the View class is modified, the Model will still work. Even if the system is moved from Windows to an app in a smart phone, the Model can be moved with no changes. But the View probably needs to be updated, as will the Controller. [5]

### 2.5.4: Use in web applications

Although originally developed for personal computing, Model View Controller has been widely adapted as architecture for World Wide Web applications in all major programming languages. Several commercial and noncommercial application frameworks have been created that enforce the pattern. These frameworks vary in their interpretations, mainly in the way that the MVC responsibilities are divided between the client and server.

Early web MVC frameworks took a thin client approach that placed almost the entire model, view and controller logic on the server. In this approach, the client sends either hyperlink requests or form input to the controller and then receives a complete and updated web page (or other document) from the view; the model exists entirely on the server. As client technologies have matured, frameworks such as JavaScriptMVC and Backbone have been created that allow the MVC components to execute partly on the client. [5]

# References

|  |  |
| --- | --- |
| [1] | "Web application - Wikipedia, the free encyclopedia," [Online]. Available: http://en.wikipedia.org/wiki/Web\_application. [Accessed March 2013]. |
| [2] | "Single-page application - Wikipedia, the free encyclopedia," [Online]. Available: http://en.wikipedia.org/wiki/Single-page\_application. [Accessed March 2013]. |
| [3] | "Ajax (programming) - Wikipedia, the free encyclopedia," [Online]. Available: http://en.wikipedia.org/wiki/Ajax\_(programming). [Accessed March 2013]. |
| [4] | D. Roos, "HowStuffWorks "What is a Payroll System?"," [Online]. Available: http://money.howstuffworks.com/payroll-system1.htm. [Accessed March 2013]. |
| [5] | "Model–view–controller - Wikipedia, the free encyclopedia," [Online]. Available: http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller. [Accessed March 2013]. |