USN | | | | | | | | | | |

13MCA552

Fifth Semester MCA Degree Examination, Dec.2016/Jan.2017

Web 2.0 and Rich Internet Applications

Time: 3 hrs. Max. Marks:100

Note: ***Answer any FIVE full questions.***

1 a. List the limitations of classic web model. **(04 Marks)**
b. With a neat diagram, discuss the working of AJAX model compare with classic web model. **(06 Marks)**
c. How do you create and configure XML HTTP request object? Demonstrate the working of XML HTTP request object with an example program. **(10 Marks)**

2 a. What is a frame? How do you make a frame hidden? **(02 Marks)**
b. Explain the various steps of hidden frame communication with help of a diagram. **(10 Marks)**
c. Write a program to download HTTP headers using XML HTTP request object. **(08 Marks)**

3 a. When do you face browser cache problem? How to overcome this problem? **(04 Marks)**
b. Write a program to display number of children of a document element in an XML file. **(08 Marks)**
c. Briefly discuss the periodic refresh AJAX pattern. Give the real world applications of this problem. **(08Marks)**

4 a. List out any four server variables along with their purpose. **(04 Marks)**
b. Demonstrate a HTML program to send form data through an array to the server and PHP script to handle it in the server. **(10 Marks)**
c. Explain the text and integer input validation using PHP script. **(06 Marks)**

5 a. Differentiate traditional applications with flex applications. **(04 Marks)**
b. Discuss the working procedure of a flex application. **(06 Marks)**
c. Explain the two basic categories of components in a flex MXML application. Give example for each. **(10 Marks)**

6 a. How are synchronous and asynchronous errors handled in action script? Give an example. **(06 Marks)**
b. Describe the various types of value selector UI. Give an example for each. **(08 Marks)**
c. What is a collection object? Create a XML list collection object and bind it to a combo box. **(06 Marks)**

7 a. What are states and transitions? Explain with an example code. **(08 Marks)**
b. Discuss data binding in flex. **(06 Marks)**
c. How do you customize the validator messages? Demonstrate user name validation in a flex application. **(06 Marks)**

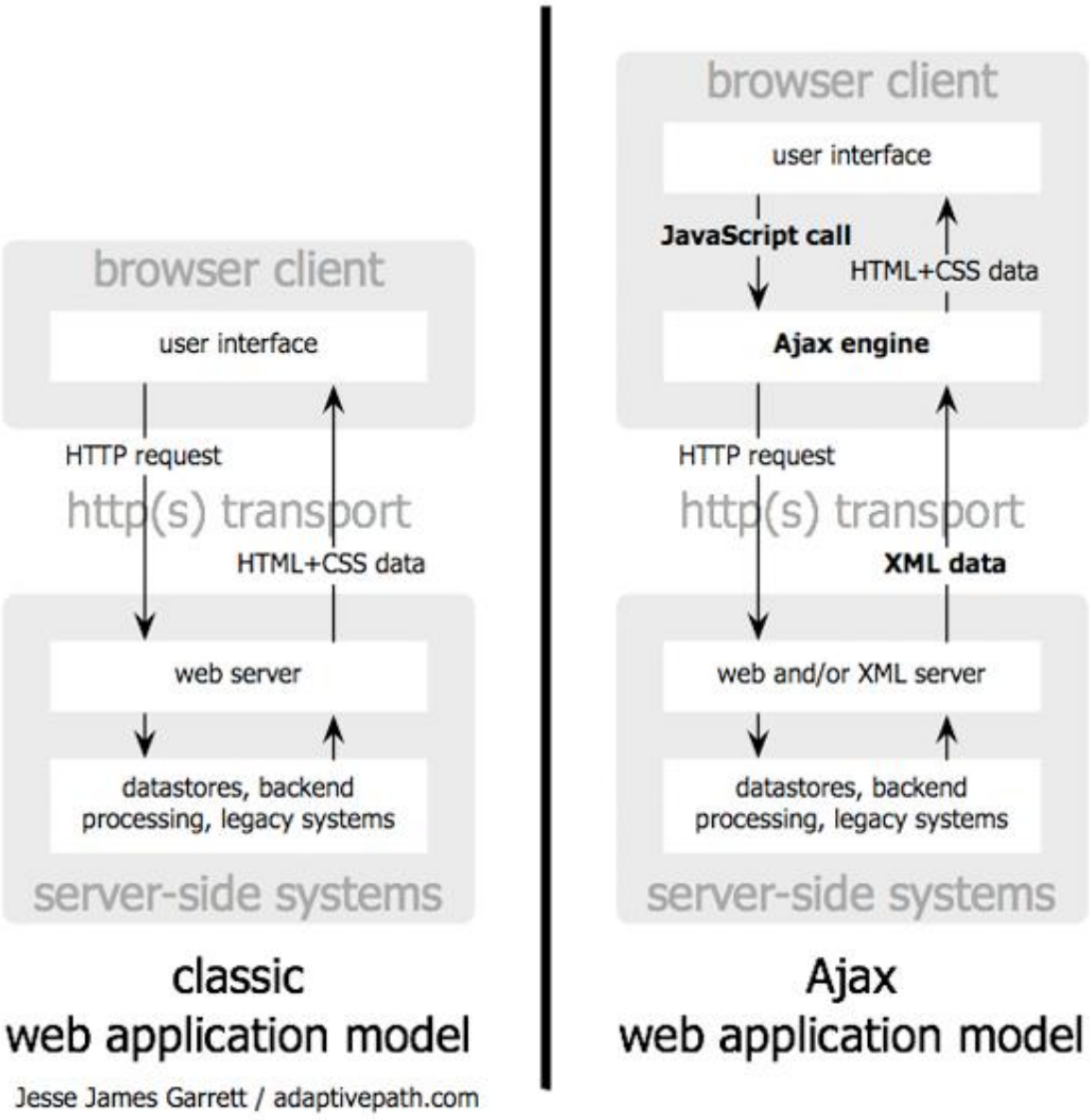
8 Write short notes on:
a. Business model for internet and web
b. Semantic web
c. Data models in flex
d. Replacing elements using DOM. **(20 Marks)**

* * * * *

WEB 2.0 AND RICH INTERNET APPLICATIONS

FIFTH SEMESTER MCA UNIVERSITY EXAM DEC 2016/JAN 2017

1 a	List the limitations of classic web model 1. Rich Assets This is the fundamental difference. A traditional website is mainly made up of Text (plus a few pictures). No matter what kind of site it is (blog, forum, or e-commerce 2. Rich Experience The richness of RIA means rich user experience. It is kind of paradigm shift from what we can provide more to how we can help users get more. 3. Rich Functionality Powered by the latest web technologies such as Adobe Flex/Flash, AJAX, or Microsoft SilverLight, A RIA can provide functions way beyond traditional web pages. 4. Rich Client A traditional web application often puts heavy workload on the server side. Regardless of its development platform (Java, .Net, or PHP), the server has to maintain the user session, process the request, and render the results. The client side is merely a browser to display the final page. 5. Rich Communication Whenever users ask for something, the browser has to send a HTTP request to the server, wait for the response back from the server, and refresh the page. This synchronous communication method inevitably creates negative impact on usability. 6. Poor Security While we enjoy the richness of RIA, it brings more security concerns than a traditional web application. The main reason is RIA is running on the client side. It tends to have more security flaws.
1 b	With a neat diagram discuss the working of AJAX model compare with classic web model



1 c

How do you create and configure xmlhttprequest object? Demonstrate the working of xmlhttprequest object with an appropriate program

A simple AJAX application

```
<!DOCTYPE html>
<html>
<head>
<script>
function loadXMLDoc()
{
var xmlhttp;
if (window.XMLHttpRequest)
  {// code for IE7+, Firefox, Chrome, Opera, Safari
        xmlhttp=new XMLHttpRequest();
  }
else
```

```
 {// code for IE6, IE5
            xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
 }
 xmlhttp.onreadystatechange=function()
 {
 if (xmlhttp.readyState==4 && xmlhttp.status==200)
   {
          document.getElementById("myDiv").innerHTML=xmlhttp.responseText;
   }
 }
xmlhttp.open("GET","ajax_info.txt",true);
xmlhttp.send();
}
</script>
</head>

<body>
<div id="myDiv"><h2>Let AJAX change this text</h2></div>
<button type="button" onclick="loadXMLDoc()">Change Content</button>
</body>
</html>
```

The XMLHttpRequest Object

The keystone of AJAX is the XMLHttpRequest object. The mechanism for sending data to and retrieving data from the server with Ajax is theXMLHttpRequest object. All modern browsers support the XMLHttpRequest object (IE5 and IE6 use an ActiveXObject). The XMLHttpRequest object is used to exchange data with a server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.

How to Create an XMLHttpRequest Object

All modern browsers (IE7+, Firefox, Chrome, Safari, and Opera) have a built-in XMLHttpRequest object.

Syntax to create an object:

```
variable=new XMLHttpRequest();
```

Old versions of Internet Explorer (IE5 and IE6) uses an ActiveX Object:

```
variable=new ActiveXObject("Microsoft.XMLHTTP");
```

To handle all modern browsers, including IE5 and IE6, check if the browser supports the XMLHttpRequest object. If it does, create an XMLHttpRequest object, if not, create an ActiveXObject:

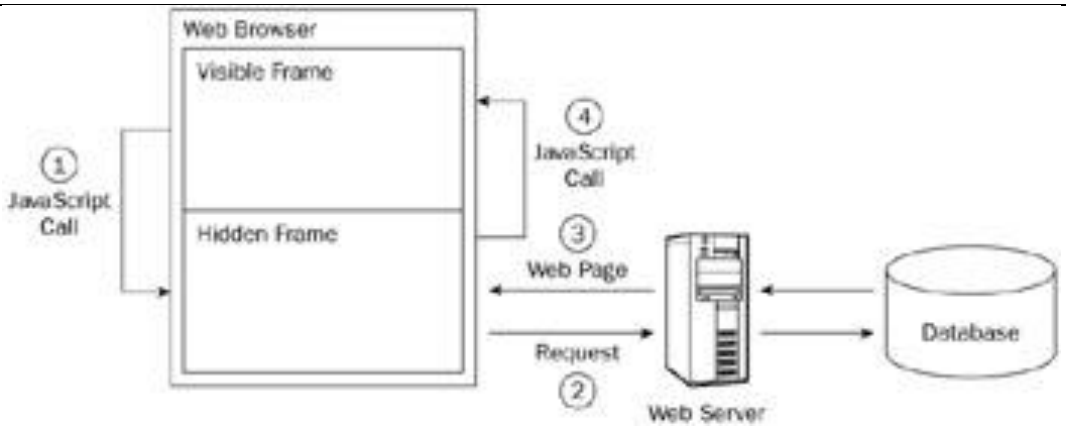
The XMLHttpRequest object is used to exchange data with a server.

How to Send a Request To a Server

To send a request to a server, we use the open() and send() methods of the XMLHttpRequest object:

```
xmlhttp.open("GET","ajax_info.txt",true);
xmlhttp.send();
```

	<table><tr><th>Method</th><th>Description</th></tr><tr><td>open(*method,url,async*)</td><td>Specifies the type of request, the URL, and if the request should be handled asynchronously or not. *method*: the type of request: GET or POST *url*: the location of the file on the server *async*: true (asynchronous) or false (synchronous)</td></tr><tr><td>send(*string*)</td><td>Sends the request off to the server. *string*: Only used for POST requests</td></tr></table> GET is simpler and faster than POST, and can be used in most cases. However, always use POST requests when: • A cached file is not an option (update a file or database on the server) • Sending a large amount of data to the server (POST has no size limitations) • Sending user input (which can contain unknown characters), POST is more robust and secure than GET
2 a	What is a frame? How do you make a frame hidden With the introduction of HTML frames, the hidden frame technique was born. The basic idea behind this technique is to create a frameset that has a hidden frame that is used for client-server communication. You can hide a frame by setting its width or height to 0 pixels, effectively removing it from the display. Although some early browsers (such as Netscape 4) couldn't fully hide frames, often leaving thick borders, this technique still gained popularity among developers. The Pattern The hidden frame technique follows a very specific, four-step pattern (see Figure 2-1). The first step always begins with the visible frame, where the user is interacting with a web page. Naturally, the user is unaware that there is a hidden frame (in modern browsers, it is not rendered) and goes about interacting with the page as one typically would. At some point, the user performs an action that requires additional data from the server. When this happens, the first step in the process occurs: a JavaScript function call is made to the hidden frame. This call can be as simple as redirecting the hidden frame to another page or as complicated as posting form data. Regardless of the intricacy of the function, the result is the second step in the process: a request made to the server.



The third step in the pattern is a response received from the server. Because you are dealing with frames, this response must be another web page. This web page must contain the data requested from the server as well as some JavaScript to transfer that data to the visible frame. Typically, this is done by assigning an onload event handler in the returned web page that calls a function in the visible frame after it has been fully loaded (this is the fourth step). With the data now in the visible frame, it is up to that frame to decide what to do with the data.

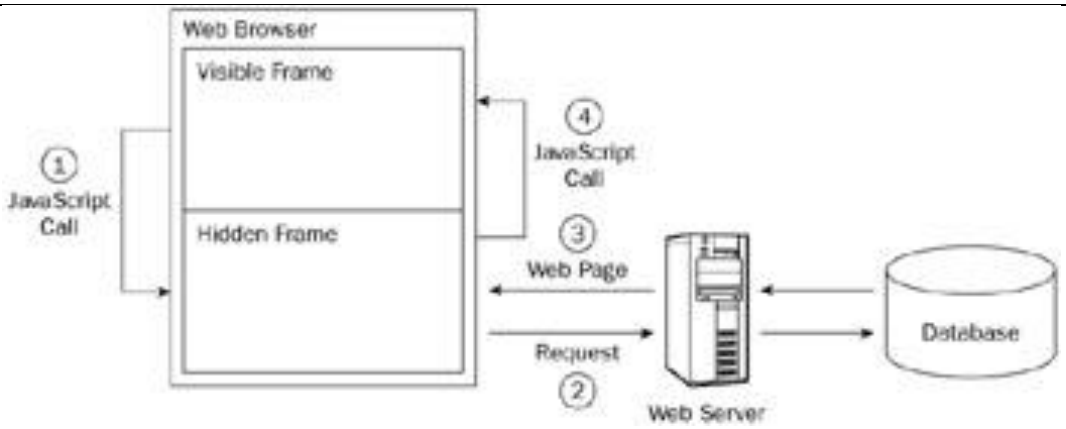
2b Explain the various stages of hidden frame technique with help of a diagram

The Hidden Frame Technique

With the introduction of HTML frames, the hidden frame technique was born. The basic idea behind this technique is to create a frameset that has a hidden frame that is used for client-server communication. You can hide a frame by setting its width or height to 0 pixels, effectively removing it from the display. Although some early browsers (such as Netscape 4) couldn't fully hide frames, often leaving thick borders, this technique still gained popularity among developers.

The Pattern

The hidden frame technique follows a very specific, four-step pattern (see Figure 2-1). The first step always begins with the visible frame, where the user is interacting with a web page. Naturally, the user is unaware that there is a hidden frame (in modern browsers, it is not rendered) and goes about interacting with the page as one typically would. At some point, the user performs an action that requires additional data from the server. When this happens, the first step in the process occurs: a JavaScript function call is made to the hidden frame. This call can be as simple as redirecting the hidden frame to another page or as complicated as posting form data. Regardless of the intricacy of the function, the result is the second step in the process: a request made to the server.



The third step in the pattern is a response received from the server. Because you are dealing with frames, this response must be another web page. This web page must contain the data requested from the server as well as some JavaScript to transfer that data to the visible frame. Typically, this is done by assigning an onload event handler in the returned web page that calls a function in the visible frame after it has been fully loaded (this is the fourth step). With the data now in the visible frame, it is up to that frame to decide what to do with the data.

2c Write a program to download HTTP headers using xmlhttprequest object

```
<!DOCTYPE html>
<html>
<head>
<script>
function loadXMLDoc()
{
var xmlhttp;
xmlhttp=new XMLHttpRequest();
xmlhttp.onreadystatechange=function()
 {
 if (xmlhttp.readyState==4 && xmlhttp.status==200)
   {
   document.getElementById("myDiv").innerHTML=xmlhttp.responseText;
   }
 }
xmlhttp.open("GET","ajax_info.txt",true);
xmlhttp.send();
}
function loadDoc()
{
var xmlhttp;
xmlhttp = new XMLHttpRequest();
xmlhttp.onreadystatechange=function()
        {
                if(xmlhttp.readyState == 4 && xmlhttp.status == 200)
                        {

        document.getElementById("div1").innerHTML=xmlhttp.getAllResponseHeaders();
```

	}
	}
	xmlhttp.open("GET","ajax_info2.txt",true);
	xmlhttp.send();
	}
	</script>
	</head>
	
	<body>
	
	<button type="button" onclick="loadXMLDoc()" ondblclick="loadDoc()">Change Content</button>
	  Let AJAX change this text
	  Content changed again
	</body>
	</html>

3a	When do you face browser cache problem . How can you overcome this problem?
	When it comes to whitespace, Firefox by default acts differently than Internet Explorer. In Firefox, whitespace that you use to indent the elements in your XML counts as text nodes. So when navigating, we have to take all the whitespace nodes into account in Firefox, by default.To strip out indentation whitespace before Firefox gets its hands on it. To do that, we might write a JavaScript function named removeWhitespace, which is passed a JavaScript XML document object If the current node is an element node, which we can check by seeing if its nodeType property equals 1 (see Table 6-1), it might have child nodes—and we've got to remove the whitespace from those child nodes as well, so we pass the current node to the removeWhitespace function again:
	function removeWhitespace(xml)
	{
	var loopIndex;
	for (loopIndex = 0; loopIndex < xml.childNodes.length;
	loopIndex++) {
	var currentNode = xml.childNodes[loopIndex];
	if (currentNode.nodeType == 1) {
	removeWhitespace(currentNode);
	}
	.
	.
	.
	}
	}
	
	function removeWhitespace(xml)
	{
	var loopIndex;
	for (loopIndex = 0; loopIndex < xml.childNodes.length;
	loopIndex++) {
	var currentNode = xml.childNodes[loopIndex];
	if (currentNode.nodeType == 1) {
	removeWhitespace(currentNode);
	}
	if (((/^\s+$/.test(currentNode.nodeValue))) &&

	(currentNode.nodeType == 3)) {
	.
	.
	.
	}
	}
	}
	If the if statement's condition is satisfied, you know you have a text node that's pure whitespace, and should be removed.

3b	Write a program to display number of children of a document element in n XML file
	
	<!DOCTYPE html>
	<html>
	<body>
	
	 <button onclick="loadXMLDoc()">Get CD info</button>
	
	<table id="demo" border="1">
	<tr><th>Artist</th><th>Title</th></tr>
	</table>
	
	<script>
	function loadXMLDoc() {
	var xmlhttp = new XMLHttpRequest();
	xmlhttp.onreadystatechange = function() {
	if (this.readyState == 4 && this.status == 200) {
	myFunction(this);
	}
	};
	xmlhttp.open("GET", "cd_catalog.xml" , true);
	xmlhttp.send();
	}
	
	function myFunction(xml) {
	var x, i, xmlDoc, table;
	xmlDoc = xml.responseXML;
	table = "<tr><th>Artist</th><th>Title</th></tr>";
	x = xmlDoc.getElementsByTagName("CD")
	for (i = 0; i < x.length; i++) {
	table += "<tr><td>" +
	x[i].getElementsByTagName("ARTIST")[0].childNodes[0].nodeValue +
	"</td><td>" +
	x[i].getElementsByTagName("TITLE")[0].childNodes[0].nodeValue +
	"</td></tr>";
	}
	document.getElementById("demo").innerHTML = table;
	}
	</script>
	
	</body>

	</html> Here x.length gives number of children in document root
3c	Briefly discuss the periodic refresh AJAX pattern. Give the real world application for this problem 1. Periodic Refresh ▪ It is a design pattern ▪ Describes the process of checking for new server information in specific intervals ▪ This approach is also called polling ▪ Used to increase user experience ▪ Notify users of updated information 2. Polling Mechanism Requires the browser to keep track of when another request to the server should take place 3. Real World Examples ▪ ESPN : Update online scoreboards automatically ▪ Gmail : Notify users of the new mail ▪ XHTML Live Chat : Implement a chat room ▪ Magnetic Ajax Demo : Re-creates online the experience of magnetic poetry ▪ White Chat & Lace Chat ▪ JotSpot LiveWiki ▪ Claude Hussnet's Portal 4. ESPN: Update Online Scoreboards Automatically ▪ Shows up-to-the-minute scores ▪ Drive charts for every game being played at that time ▪ Uses XHR objects and a little bit of Flash ▪ Page repeatedly updates itself with new information ▪ For example: the IPL Scoreboard ▪ http://www.espncricinfo.com/ci/engine/current/match/scores/live.ht ml 5. Gmail: Notify Users Of The New Mail ▪ Repeatedly checks the server to see if new mail has arrived ▪ This is done without notification unless there is new mail ▪ The number of new e-mails received is displayed in parentheses next to the Inbox menu item ▪ http://gmail.google.com
4a	List out four server variables along with their purpose AUTH_TYPE When running under the Apache web server and doing HTTP authentication, holds the authentication type (such as password authentication). DOCUMENT_ROOT Contains the document root directory where the script is. GATEWAY_INTERFACE Contains the version of the CGI (Common Gateway Interface—how servers communicate with browsers) specification the server is using. HTTP_ACCEPT Contains the text in the Accept: header from the current request. HTTP_ACCEPT_CHARSET Contains the text in the Accept-Charset: header from the current request.

4b	Demonstrate an HTML program to send form data through an array to the server and PHP script to handle it in the server
	
	<html>
	<head>
	<title>Sending Data to the Server With POST</title>
	<script language = "javascript">
	var XMLHttpRequestObject = false;
	if (window.XMLHttpRequest) {
	XMLHttpRequestObject = new XMLHttpRequest();
	} else if (window.ActiveXObject) {
	XMLHttpRequestObject = new ActiveXObject("Microsoft.XMLHTTP");
	}
	function getData(dataSource, divID, data)
	{
	if(XMLHttpRequestObject) {

```
var obj = document.getElementById(divID);
XMLHttpRequestObject.open("POST", dataSource);
XMLHttpRequestObject.setRequestHeader('Content-Type',
'application/x-www-form-urlencoded');
XMLHttpRequestObject.onreadystatechange = function()
{
if (XMLHttpRequestObject.readyState == 4 &&
XMLHttpRequestObject.status == 200) {
obj.innerHTML = XMLHttpRequestObject.responseText;
}
}
XMLHttpRequestObject.send("data=" + data);
}
}
</script>
</head>
<body>
<H1>Sending Data to the Server With POST</H1>
<form>
<input type = "button" value = "Fetch message 1"
onclick = "getData('dataresponderpost.php', 'targetDiv', 1)">
<input type = "button" value = "Fetch message 2"
onclick = "getData('dataresponderpost.php', 'targetDiv', 2)">
</form>
<div id="targetDiv">
<p>The fetched message will appear here.</p>
</div>
</body>
</html>
```

4c Explain the text and integer input validation using PHP script

```
<body>

<?php
$temp = "";
$msg = "";
$rst = "";

if (isset($_POST["submit"])) {
$number = $_POST["custid"];

if(empty($number)) {
$msg = '<span class="error"> Please enter a value</span>';
} else if(!is_numeric($number)) {
$msg = '<span class="error"> Data entered was not numeric</span>';
} else if(strlen($number) != 6) {
$msg = '<span class="error"> The number entered was not 6 digits long</span>';
} else {
echo "valid";
```

```
}

}

?>

<h1>Customer Information Collection <br /></h1>

<form method="POST" action="<?php echo $_SERVER["PHP_SELF"];?>" id="custinfo" >
<table>
<tr>
   <td><label for="custid">Customer ID (integer value): </label></td>
   <td><input type="text" id="custid" name="custid" value="<?php echo   $temp ?>" size=11 /><?php echo
$msg; ?></td>
</tr>

<tr>
   <td><label for="customerfname">Customer Frist Name: </label></td>
   <td><input type="text" id="customerfname" name="fname" size=50/></td>
</tr>
<tr>
   <td><label for="customerlname">Customer Last Name: </label></td>
   <td><input type="text" id="customerlname" name="lname" size=50/></td>
</tr>
<tr>
   <td><label for="customeraddress">Customer Address: </label></td>
   <td><input type="text" id="customeraddress" name="custaddress" size=65/></td>

   <td><label for="suburb"> Suburb: </label></td>
<td><input type="text" id="suburb" name="suburb"/></td>
</tr>
<tr>
<td>
State:<select name="state" id="state">
   <option value="select">--</option>
   <option value="ACT">ACT</option>
   <option value="NSW">NSW</option>
   <option value="NT">NT</option>
   <option value="QLD">QLD</option>
   <option value="SA">SA</option>
   <option value="TAS">TAS</option>
   <option value="VIC">VIC</option>
    <option value="WA">WA</option>
 </select>
</td>
<td><label for="postcode"> Post Code: </label><input type="text" id="postcode" name="postcode"
size=4/></td>
</tr>
</table>
<p><input type="submit" name="submit" value="Save Data"/> <input type="reset" value="Clear
```

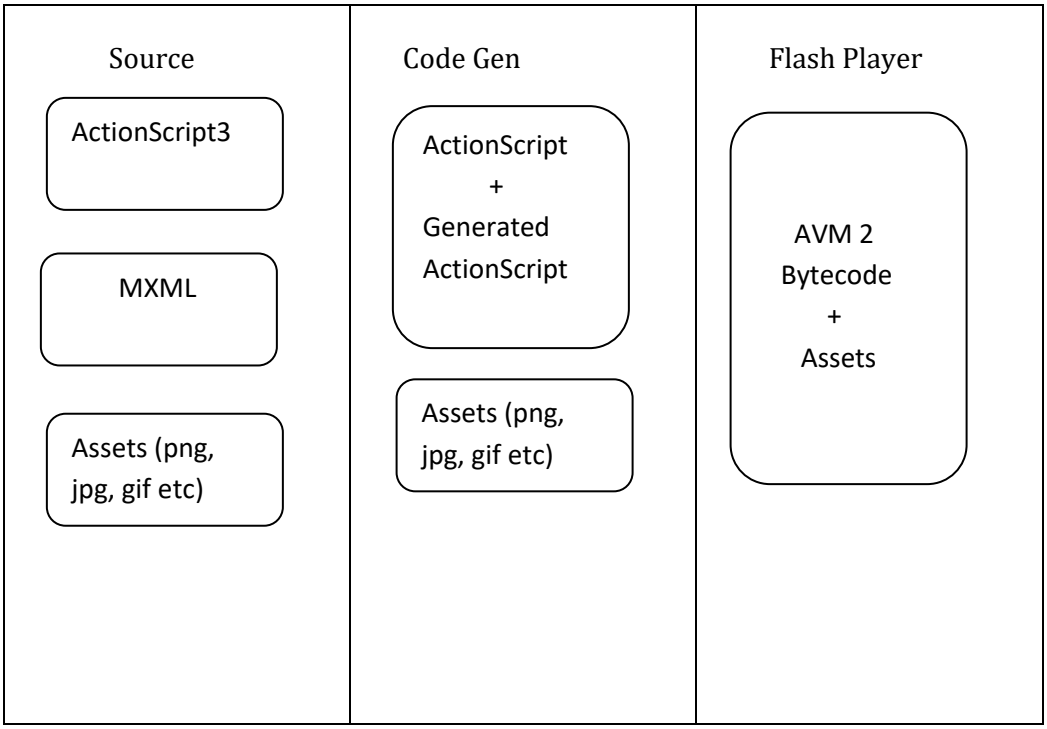
<table>
<tr><td></td><td>
Form" />
</tr>

</form>

</body>
</td></tr>
<tr><td>5a</td><td>
Differentiate traditional applications with flex web applications
Differences between Traditional and Flex Web Applications
<table>
<tr><th>Traditional Web Application</th><th>Flex Web Application</th></tr>
<tr><td>1. Traditional web applications have data tier, business tier, and presentation tier.
The presentation tier consists of HTML, CSS, JavaScript, JSP, ASP, PHP, or similar documents.
A request is made from the user's web browser for a specific presentation tier resource, and the web server runs any necessary interpreters to convert the resource to HTML and JavaScript, which is then returned to the web browser running on the client computer. Technically the HTML rendered in the browser is a client tier in a traditional web application.</td><td>Flex applications have data tier, business tier and client tier. The client tier of flex application enables enables clients to offload computation from the server, freeing up network latency and making for responsive and highly interactive user interfaces.

Data tiers generally consist of databases or similar resources. Business tiers consist of the core application business logic. As an example, a business tier may accept requests from a client or presentation tier, query the data tier, and return the
requested data.

Flex applications generally reside embedded within the presentation tier. In addition, Flex applications can integrate with the presentation tier to create tightly coupled client-side systems. Flex applications use Flash Player to run sophisticated
client-tier portions of the application.</td></tr>
</table>
</td></tr>
</table>

		2. The client tier of a traditional web application is stateless and fairly nonresponsive, it is generally not considered a full-fledged tier.	2. The Flex application client is *stateful*, which means that it can make changes to the view without having to make a request to the server. the Flex application client is responsive. For example, Flash Player can respond to user interaction such as mouse movement, mouse clicks, and keyboard presses, and it can respond to events such as notifications from the business tier when data is returned or pushed to the client. Flash Player also can respond to timer events. Flex applications can walk the user through a step-based or wizard-like interface, collect and validate data, and allow the user to update and edit previous steps, all without having to make requests to the business tier until the user wants to submit the data. All this makes Flex clients potentially far more compelling, responsive, and engaging than traditional web applications.
5b	Discuss the working procedure of flex application 		

	Flex applications deployed on the Web work differently than HTML-based applications. Every Flex application deployed on the Web utilizes Flash Player as the deployment platform. all Flex applications use the Flex framework at a minimum to compile the application. All Flex applications require at least one MXML file or ActionScript class file, and most Flex applications utilize both MXML and ActionScript files. The MXML and ActionScript class files comprise the source code files for the application. Flash Player does not know how to interpret MXML or uncompiled ActionScript class files. Instead, it is necessary to compile the source code files to the *.swf* format, which Flash Player can interpret. A typical Flex application compiles to just one *.swf* file. You then deploy that one *.swf* to the server, and when requested, it plays back in Flash Player. That means that unlike HTML-based applications, the source code files remain on the development machine, and you do not deploy them to the production server. Asset files such as MP3s, CSS documents, and PNGs can be embedded within a *.swf*, or they can be loaded at runtime. Data services are requested at runtime. That means that the services must be available at a valid URL when requested at runtime. For example, if a Flex application utilizes a web service, that web service must be accessible from the client when requested.
5c	Explain the two categories of components in a flex application. Give example for each Components A component is an ActionScript class or an MXML component document that has been mapped to an identifier via a manifest file so that it can be instantiated via MXML. There are many different types of components, but in terms of the Flex framework components, there are two basic categories: visual and nonvisual. The visual components consist of the following: • Containers • User interface controls The nonvisual components consist of the following: • Data components • Utility components Visual Components Containers Containers are types of components that can contain other components. Every application must use containers. The Application element itself is a container because you can place other components within it. Containers are used for layout. There are containers for vertical layout, horizontal layout, grids, tiles, and all sorts of layout configurations. When you use layout containers, you place other components within them using nested tags. There are containers for vertical layout, horizontal layout, grids, tiles, and all sorts of layout configurations. When you use layout containers, you place other components within them using nested tags.

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute">
<mx:VBox>
        <mx:Button label="Example Button 1" />
        <mx:Button label="Example Button 2" />
</mx:VBox>
</mx:Application>
```

UI controls

User interface controls are visible interface elements such as buttons, text inputs, lists, and data grids. For example, a button component allows you to apply a label by setting a property. Every component type has its own unique set of properties that you can set. For example, a button and a VBox clearly have different properties because they do different things. The simplest and most common way to set properties for a component is to use the tag attributes. For instance, the Application tag allows you to set a layout property using a tag attribute as in the following example:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute">
</mx:Application>
```

A tag can have many attributes, each separated by spaces. Almost all components (all visible components) have an id property. The id property for a component should consist only of letters, numbers, and underscores, and it should
start with either an underscore or a letter, but not a number. The following assigns an id value to a button:

```
<mx:Button id="exampleButton" label="Example Button" />
```

The dataProvider property of a combo box must be some sort of collection of values. The following example creates a combo box and uses a nested dataProvider tag to populate it with values:

```
<mx:ComboBox id="exampleComboBox">
<mx:dataProvider>
        <mx:ArrayCollection>
                <mx:String>A</mx:String>
                <mx:String>B</mx:String>
                <mx:String>C</mx:String>
                <mx:String>D</mx:String>
        </mx:ArrayCollection>
</mx:dataProvider>
</mx:ComboBox>
```

Nonvisual components

There are two types of nonvisual components: data components and utility components.
Data components are used to create data structures, such as arrays and collections, and for making remote procedure calls with protocols such as SOAP for web services or AMF for Flash Remoting

Utility components are components used to achieve functionality. Examples of utility components are those used for creating repeating components and for creating data binding between components.

6a	How are synchronous and asynchronous errors handled in actionscript.
	
	Handling Synchronous Errors
	Synchronous errors occur immediately when trying to execute a statement. You can use try/catch/finally to handle synchronous errors.
	When you have some code that may throw runtime errors, surround it with a try statement:
	try {
	// Code that might throw errors
	}
	You must then include one or more catch blocks following a try. If the code in the try block throws an error, the application attempts to match the error to the catch blocks in the order in which they appear. Every catch block must specify the specific type of error that it handles. The application runs the first catch block that it encounters to see if it matches the type of error thrown. All error types are either flash.errors.Error types or subclasses of Error. Therefore, you should try to catch more specific error types first, and more generic types (e.g., Error) later; for example:
	try {
	// Code that might throw errors
	}
	catch (error:IOError) {
	// Code in case the specific error occurs
	}
	catch (error:Error) {
	// Code in case a non-specific error occurs
	}
	In addition, you can add a finally clause that runs regardless of whether the try statement is successful:
	try {
	// Code that might throw errors
	}
	catch (error:IOError) {
	// Code in case the specific error occurs
	}
	catch (error:Error) {
	// Code in case a non-specific error occurs
	}
	finally {
	// Code to run in any case
	}
	Handling Asynchronous Errors
	Many objects in ActionScript can potentially throw *asynchronous errors*. Asynchronous errors are those that occur in response to network operations. For example, if a requested file is not found, the network operation fails asynchronously, and an asynchronous error is thrown. All asynchronous errors are in the form of events, and they use the same event model as standard events. For example, if a URLLoader object attempts to load data outside the Flash Player security sandbox, it dispatches a securityError event. The following example illustrates how to handle error events:

	<?xml version="1.0" encoding="utf-8"?>
	<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute"
	initialize="initializeHandler(event)">
	<mx:Script>
	<![CDATA[
	private function initializeHandler(event:Event):void {
	var loader:URLLoader = new URLLoader( );
	// In order to test this you'll need to specify a URL of a file that
	// exists outside of the security sandbox.
	loader.load(new URLRequest("data.xml"));
	loader.addEventListener(SecurityErrorEvent.SECURITY_ERROR,
	securityErrorHandler);
	}
	
	private function securityErrorHandler(event:SecurityErrorEvent):void
	{
	errors.text += event + "\n";
	}
	]]>
	</mx:Script>
	<mx:TextArea id="errors" />
	</mx:Application>

6b	Describe the various tyoes of value selectors UI. The Flex framework includes not only the base validator, Validator, but also many validators for common sorts of data formats. The framework ships with the following validators: • StringValidator • NumberValidator • DateValidator • EmailValidator All the validator types are subtypes of Validator, and they inherit all the same properties and methods StringValidator The StringValidator allows you to verify that a string value length is within a specific range. You can define minLength and maxLength property values for StringValidators, as in the following example: <mx:StringValidator source="{sourceObject}" property="sourceProperty" minLength="5" maxLength="10" /> You can also specify custom error messages, just in case the validation fails, using the tooShortError and tooLongError properties: <mx:StringValidator source="{sourceObject}" property="sourceProperty" minLength="5" maxLength="10" tooShortError="You gotta use a longer number, buddy" tooLongError="Whoa! Shorter numbers, please" /> NumberValidator The NumberValidator allows you to validate all sorts of number values. You can specify a range of allowable values using the minValue and maxValue properties:

	<mx:NumberValidator source="{sourceObject}" property="sourceProperty" minValue="-5" maxValue="5" /> The default value for minValue and maxValue is NaN (not a number), which means that no limit is placed on the range. If you set a value for either property and you later want to remove the limit, just assign a value of NaN to the validator, as shown here: numberValidator.minValue = NaN; If you want to allow or disallow negative numbers, you can use the allowNegative property. The default value is true, but setting it to false disallows negative numbers: <mx:NumberValidator source="{sourceObject}" property="sourceProperty" allowNegative="false" /> By default, a NumberValidator allows any number type, but you can explicitly specify whether you want to accept all number types (real) or just integers (int) using the domain property. The default value is real, but the following example allows only integers: <mx:NumberValidator source="{sourceObject}" property="sourceProperty" domain="int" /> DateValidator DateValidator allows you to validate values as dates. There are several basic properties you can configure to customize this type of validator, and there are advanced properties that allow you to use the validator with several inputs at once. The basic properties you can use with a DateValidator are allowedFormatChars and inputFormat. The allowedFormatChars property allows you to specify whichcharacters are allowable as delimiters between year, month, and date. The default value is /\-., which means any of the following is valid: The following allows only the asterisk character as a delimiter: <mx:DateValidator source="{sourceObject}" property="sourceProperty" allowedFormatChars="*" /> The inputFormat property determines the order of the year, month, and date parts. You can use the strings YYYY, MM, and DD to represent each of those parts. You can use any of the default delimiter characters as delimiters in the inputFormat string.The default value is MM/DD/YYYY. The following example requires that the date appear with the year followed by the month followed by the date: <mx:DateValidator source="{sourceObject}" property="sourceProperty" inputFormat="YYYY/MM/DD" /> EmailValidator The EmailValidator is easy to implement, as it doesn't require any additional properties aside from the standard source and property. It simply validates the source value as a valid email address: <mx:EmailValidator source="{sourceObject}" property="sourceProperty" />
6c	What is collection obect Create an xml list collection onject and bind it to a combo box *Collections* are objects that provide a uniform way to access and represent the data contained in a data source object, such as an Array or an XMLList object. Collections provide a level of abstraction between components and the data objects that you use to populate them. The standard collection types in the Flex framework, the ArrayCollection and XMLListCollection classes, extend the mx.collections.ListCollectionView class, which implements the mx.collections.ICollectionView and mx.collections.IList interfaces. These interfaces provide the underlying functionality for viewing and

	modifying data objects. An ArrayCollection object takes an Array as its source object. An XMLListCollection object take an XMLList object as its source object. Collections provide the following features: ▪ Ensure that a component is properly updated when the underlying data changes. Components are not updated when noncollection data objects change. (They *are* updated to reflect the new data the next time they are refreshed.) If the data provider is a collection, the components are updated immediately after the collection change occurs. ▪ Provide mechanisms for handling paged data from remote data sources that may not initially be available and may arrive over a period of time. ▪ Provide a consistent set of operations on the data, independent of those provided by the raw data source. For example, you can insert and delete objects by using an index into the collection, independently of whether the underlying data is, for example, in an Array or an Object. ▪ Provide a specific *view* of the data that can be in sorted order, or filtered by a developer-supplied method. This is only a view of the data; it does not change the data. ▪ Use a single collection to populate multiple components from the same data source. ▪ Use collections to switch data sources for a component at run time, and to modify the content of the data source so that changes are reflected by all components that use the data source. ▪ Use collection methods to access data in the underlying data source.

7a	What are states and transitions? Explain with example
	Flex applications always consist of one or more user interface and/or container components. At a minimum, a Flex application has an application container, but usually it has many additional components. Although the default behavior for components is fairly static, you can liven up an application with the use of effects. An *effect* is an action that takes place, such as moving, fading, or zooming into or out of a component. An effect can even be a nonvisual behavior, such as playing a sound. Using effects, you can create applications that are more visually (and audibly) interesting. Perhaps more importantly, you can use effects to direct focus and help users better understand how to use applications.
	
	States
	
	The Flex <mx:State> is a very good way to change the appearance of an itemRenderer. States are easy to use, and when combined with transitions, give the user feedback and pleasent experience.
	
	In this example, you'll create a new MXML itemRenderer (and remember, you can do this completely in ActionScript if you prefer) for the list. All this itemRenderer shows is the image, title, author, price, and a Button to purchase the book.
	
	`<?xml version="1.0" encoding="utf-8"?>`
	`<mx:HBox xmlns:mx="/2006/mxml">`
	
	`<mx:Image id="bookImage" source="{data.image}" />`
	`<mx:VBox height="115" width="100%" verticalAlign="top" verticalGap="0" paddingRight="10">`
	`<mx:Text text="{data.title}" fontWeight="bold" width="100%"/>`
	`<mx:Label text="{data.author}" />`
	`<mx:HBox id="priceBox" width="100%">`

	<mx:Label text="{data.price}" width="100%"/> <mx:Button label="Buy" /> </mx:HBox> </mx:VBox> </mx:HBox>

7b	Discuss data binding
	What is Data Binding?
	Data Binding is a process in which data of one object is tied to another object. Data binding requires a source property, a destination property and a triggering event which indicates when to copy the data from source to destination.
	Flex provides three ways to do Data Binding
	Curly brace syntax in MXML Script
	<fx:binding> tag in MXML
	BindingUtils in ActionScript
	Data Binding - Using Curly Braces in MXML
	Following example demonstrates using curly braces to specify data binding of a source to destination.
	<s:TextInput />
	<s:TextInput />
	Data Binding - Using <fx:Binding> tag in MXML
	Following example demonstrates using <fx:Binding> tag to specify data binding of a source to destination.
	<fx:Binding source="txtInput1.text" destination="txtInput2.text" />
	<s:TextInput />
	<s:TextInput />
	Data Binding - Using BindingUtils in ActionScript
	Following example demonstrates using BindingUtils to specify data binding of a source to destination.
	<fx:Script>
	<![CDATA[
	im port m x.binding.utils.BindingUtils;
	im port m x.events.FlexEvent;
	protected function txtInput2_preinitializeHandler(event:FlexEvent):void
	{
	BindingUtils.bindProperty(txtInput2,"text",txtInput1, "text");
	}
	]]>
	</fx:Script>
	<s:TextInput />
	<s:TextInput
	preinitialize="txtInput2_preinitializeHandler(event)"/>

7c	How do you customize validator messages All Flex validators define default error messages. In most cases, you can override these messages with your own. The default error messages for all validators are defined by using resource bundles so that you can easily change them as part of localizing your application. You can override the default value of an error message for all validator objects created from a validator class by editing the resource bundles associated with that class.

	You edit the error message for a specific validator object by writing a String value to a property of the validator. For example, the PhoneNumberValidator defines a default error message to indicate that an input phone number has the wrong number of digits. You can override the default error message for a specific PhoneNumberValidator object by assigning a new message string to the wrongLengthError property, as the following example shows:
	
	`<?xml version="1.0"?>`
	`<!-- validators\PNValidatorErrMessage.mxml -->`
	`<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"`
	`xmlns:s="library://ns.adobe.com/flex/spark"`
	`xmlns:mx="library://ns.adobe.com/flex/mx">`
	`<s:layout>`
	`<s:VerticalLayout/>`
	`</s:layout>`
	
	`<fx:Declarations>`
	`<!-- Define the PhoneNumberValidator. -->`
	`<mx:PhoneNumberValidator id="pnV"`
	`source="{phoneInput}" property="text"`
	`wrongLengthError="Please enter a 10-digit number."/>`
	`</fx:Declarations>`
	
	`<!-- Define the TextInput control for entering the phone number. -->`
	`<s:TextInput id="phoneInput"/>`
	`<s:TextInput id="zipCodeInput"/>`
	`</s:Application>`

8a	Write short notes on Business model for internet an web a business model is the method of doing business by which a company can sustain itself -- that is, generate revenue. The business model spells-out how a company makes money by specifying where it is positioned in the value chain. Some models are quite simple. A company produces a good or service and sells it to customers. If all goes

well, the revenues from sales exceed the cost of operation and the company realizes a profit. Other models can be more intricately woven. Broadcasting is a good example. Radio and later television programming has been broadcasted over the airwaves free to anyone with a receiver for much of the past century. The broadcaster is part of a complex network of distributors, content creators, advertisers (and their agencies), and listeners or viewers. Who makes money and how much is not always clear at the outset. The bottom line depends on many competing factors.

Semantic web
The Semantic Web is an extension of the Web through standards by the World Wide Web Consortium (W3C). The standards promote common data formats and exchange protocols on the Web, most fundamentally the Resource Description Framework (RDF). The Semantic Web provides a common framework that allows data to be shared and reused across application, enterprise, and community boundaries. The Semantic Web is regarded as an integrator across different content, information applications and systems. It has applications in publishing, blogging, and many other areas. The **Semantic Web** is the extension of the World Wide Web that enables people to share content beyond the boundaries of applications and websites. It has been described in rather different waysas a web of data, or merely as a natural paradigm shift in our daily use of the Web. Most of all, the Semantic Web has inspired and engaged many people to create innovative semantic technologies and applications.
The Semantic Web is a vision: the idea of having data on the Web defi ned and linked in a way that it can be used by machines not just for display purposes, but for automation, integration, and reuse of data across various applications. The Semantic Web is an extension of the current Web in which information is given well-defi ned meaning, better
enabling computers and people to work in cooperation.

Data models in flex

You can define a data model in an MXML tag, an ActionScript function, or an ActionScript class. In general, you should use MXML-based models for simple data structures, and use ActionScript for more complex structures and client-side business logic. *The <fx:Model> and <fx:XML> tags are Flex compiler tags and do not correspond directly to ActionScript classes.* The most common type of MXML-based model is the <fx:Model> tag, which is compiled into an ActionScript object of type mx.utils.ObjectProxy, which contains a tree of objects when your data is in a hierarchy, with no type information. The leaves of the Object tree are scalar values. Because models that are defined in <fx:Model> tags contain no type information or business logic, you should use them only for the simplest cases. Define models in ActionScript classes when you need the typed properties or you want to add business logic.

The following example shows an employee model declared in an <fx:Model> tag:

```
<?xml version="1.0"?>
<!-- Models\ModelsModelTag.mxml -->
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
   xmlns:mx="library://ns.adobe.com/flex/mx"
   xmlns:s="library://ns.adobe.com/flex/spark">
```

```
    <fx:Declarations>
      <fx:Model id="employeemodel">
        <employee>
          <name>
            <first/>
            <last/>
          </name>
          <department/>
          <email/>
        </employee>
      </fx:Model>
    </fx:Declarations>
  </s:Application>
```

Replacing elements using mxml

The replaceChild() method replaces a child node with a new node.

The new node could be an existing node in the document, or you can create a new node.

```
// Create a new <li> element
var elmnt = document.createElement("li");

// Create a new text node called "Water"
var textnode = document.createTextNode("Water");

// Append the text node to <li>
elmnt.appendChild(textnode);

// Get the <ul> element with id="myList"
var item = document.getElementById("myList");

// Replace the first child node (<li> with index 0) in <ul> with the newly created <li> element
item.replaceChild(elmnt, item.childNodes[0]);
```