**Using AJAX In ASP.NET MVC**

**What is Ajax?**

As we all know, AJAX means Asynchronous JavaScript and XML. It is a client-side script that communicates to and from a server/database without the need for a postback or a complete page refresh. The Ajax speeds up response time.

In other words, Ajax is the method of exchanging data with a server, and updating parts of a web page, without reloading the entire page.

**Benefits of Ajax**

* Callbacks
* Making Asynchronous Calls
* User-Friendly
* Improve the speed, performance and usability of a web application

**Implementation of Ajax can be done in two way in ASP.Net Application**

* using Update Panel and,
* using jQuery

**What Advances have Been Made to Ajax?**

JavaScript is the client-side scripting language and XML is a mark-up language to define data. And we have, JSON(JavaScript Object Notation) as another mark-up language to define data as well. JSON is much easier to use with JavaScript than XML. After the combination of JavaScript and Ajax, the XML Web Services are being replaced by JSON Web Services.

Another major advance to JavaScript and Ajax is the JavaScript object library called jQuery, which is the free, open-source software. It is a wrapper for JavaScript. jQuery is used to write the JavaScript to navigate and manipulate a page and make asynchronous Ajax callbacks.

Hence, Ajax callbacks have become standard programming practices by using jQuery and JSON Web Services for designing and developing web applications.

**Demonstration: Implementation of Ajax using jQuery**

**Step 1**

Create a new Project and choose ASP.NET MVC web application.

**Step 2**

Just Ignore the built-in Models and Controllers and make your own model.

Here I am creating a model called "Students" having properties studentID, studentName and studentAddress as shown below,

public class Student

{

[Key]

public int studentID { get; set; }

[Required]

public string studentName { get; set; }

[Required]

public string studentAddress { get; set; }

}

C#

Add "using System.ComponentModel.DataAnnotations;" Attributes to Validate Model Data and then build the project once.

**Step 3**

Let's create another model by inheriting the DbContext. It is a class that manages all the database operations, like database connection, and manages various entities of the Entity Model.

We can also say DbContext is a wrapper of ObjectContext. So, DbContext is a lightweight version of the ObjectContext and is exposes only the common features that are really required in programming.

Here I am creating a model called "StudentContext" as shown below,

public class StudentContext : DbContext

{

public DbSet<Student> Students { get; set; }

}

C#

Add "using System.Data.Entity;" that provides access to the core functionality of the entity framework.

**Step 4**

Now create a controller to written the code for inserting data into database, displaying data into view.

Here I am creating "Student" controller. Inside the Controller, I am creating an object of StudentContext for inserting and retrieving data from database. Also add the necessary namespace.

StudentContext context = new StudentContext();

C#

**Step 5**

Now I'm creating the action methods for Inserting and retrieving the data to/from the database.

Here I am creating an [HttpPost] action method "createStudent" for inserting the JSON-Formatted data to database. I am Using [HttpPost] attribute to Save/Post the data as below:

[HttpPost]

public ActionResult createStudent(Student std)

{

context.Students.Add(std);

context.SaveChanges();

string message = "SUCCESS";

return Json(new { Message = message, JsonRequestBehavior.AllowGet });

}

C#

Similarly, I am also creating JSON method "getStudent" to retrieve data from database and returning raw JSON data to be consumed by JavaScript in view as shown below,

public JsonResult getStudent(string id)

{

List<Student>students = new List<Student>();

students = context.Students.ToList();

return Json(students, JsonRequestBehavior.AllowGet);

}

C#

**Step 6**

Now add a view to display the data and data inserting field. Here, I am adding a view named "Index.cshtml". Then, write the HTML codes for making the data input field, submit button and also for displaying the data in same page. You can use bootstrap classes for designing. Here I am using bootstrap. My designing code is as below:

<div class="col-md-12">

<div class="panel panel-primary">

<div class="panel-heading">

<h3 class="panel-title">Please enter the details below.</h3>

</div>

<div class="panel-body">

<div class="form-group col-md-5">

<label>Student Name</label>

<input type="text" name="StudentName" id="StudentName" class="form-control" placeholder="Enter Student Name" required="" />

</div>

<div class="form-group col-md-5">

<label>Student Address</label>

<input type="text" name="StudentAddress" id="StudentAddress" class="form-control" placeholder="Enter Student Address" required="" />

</div>

<div class="form-group col-md-1">

<div style="float: right; display:inline-block;">

<input class="btn btn-primary" name="submitButton" id="btnSave" value="Save" type="button">

</div>

</div>

</div>

</div><hr />

<table id="tblStudent" class="table table-bordered table-striped table-responsive table-hover">

<thead>

<tr>

<th align="left" class="productth">ID</th>

<th align="left" class="productth">Student Name</th>

<th align="left" class="productth">Student Address</th>

</tr>

</thead>

<tbody></tbody>

</table>

</div>

Markup

We can simply use the bootstrap class and call jQuery functions in ASP.NET MVC because during the project creation it will by default added to project and also linked to the template.

**Step 7**

Now I will write the script for inserting the data as well as retrieving and displaying it to the view. Just after the HTML code finished add the script shown below,

@section Scripts

{

<script type="text/javascript">

$(function () {

LoadData();

$("#btnSave").click(function () {

//alert("");

var std = {};

std.studentName = $("#StudentName").val();

std.studentAddress = $("#StudentAddress").val();

$.ajax({

type: "POST",

url: '@Url.Action("createStudent")',

data: '{std: ' + JSON.stringify(std) + '}',

dataType: "json",

contentType: "application/json; charset=utf-8",

success: function () {

alert("Data has been added successfully.");

LoadData();

},

error: function () {

alert("Error while inserting data");

}

});

return false;

});

});

function LoadData() {

$("#tblStudent tbody tr").remove();

$.ajax({

type: 'POST',

url: '@Url.Action("getStudent")',

dataType: 'json',

data: { id: '' },

success: function (data) {

var items = '';

$.each(data, function (i, item) {

var rows = "<tr>"

+ "<td class='prtoducttd'>" + item.studentID + "</td>"

+ "<td class='prtoducttd'>" + item.studentName + "</td>"

+ "<td class='prtoducttd'>" + item.studentAddress + "</td>"

+ "</tr>";

$('#tblStudent tbody').append(rows);

});

},

error: function (ex) {

var r = jQuery.parseJSON(response.responseText);

alert("Message: " + r.Message);

alert("StackTrace: " + r.StackTrace);

alert("ExceptionType: " + r.ExceptionType);

}

});

return false;

}

</script>

}

JavaScript

We are using Ajax to refresh "tblStudent" so it refreshes only the particular table, rather than refreshing entire page. After clicking the "submitButton" the data from the input fields are taken in variable and redirected to action "createStudent" of "StudentController" to insert into the database. And, I have written a function "LoadData" to display the data from database to view continuously after inserting data. Inside "LoadData" function I am calling "getStudent" method which returns result in JSON and the JSON-Formatted data are presented in html by the underlying statements.

**Step 8**

Now, configure the Database connectionStrings in "web.config" file of the particular project as shown below,

<connectionStrings>

<add name="StudentContext" connectionString="Data Source=RAVI-KANDEL;Initial Catalog=Student;Integrated Security=True"

providerName="System.Data.SqlClient" />

</connectionStrings>

Markup

In the above connectingString "StudentContext" is the name of class inherited from abstract Class "DbContext" in "StudentContext.cs" file. "RAVI-KANDEL" is the name of the Database Server and "Student" is the name of the Database. Change "RAVI-KANDEL" with your Database Server name. Finally, run your application and navigate to Student controller. Also you can configure "RouteConfig.cs" and set the controller to "Student" for direct navigation to the Student controller when loading the application.

You can add the records and can see added data in the table after the form-control as shown below,

A screenshot of a computer

AI-generated content may be incorrect.

Also, you can open the database server and see the data stored in database over there.