MONASH University

FIT5032 - Internet Applications Development

Week 05C - Using Calendars

Last updated: 5th August 2018
Author: Jian Liew

Introduction

There is no need to complete this tutorial. It functions as a supplementary material to showcase how to use a simple JavaScript API (Full Calendar) to show events. Full Calendar can be found https://fullcalendar.io/

One of the **most complex use cases** has to do with the dealing of time. So, booking and scheduling uses cases are normally very challenging. This tutorial aims to show how to use Full Calendar (A JavaScript Library) to create simple events. It will not teach you how to handle the complexities of event scheduling and proper handling of time. Most developers tend to **misunderstand and underestimate** the complexities whenever time comes into the picture.

This tutorial will be done slightly different, instead of going Step by Step, it will give an explanation of what is done to achieve it. So, you will be given the final completed version of it here.

Objectives

Estimated Time To Complete - 30 mins

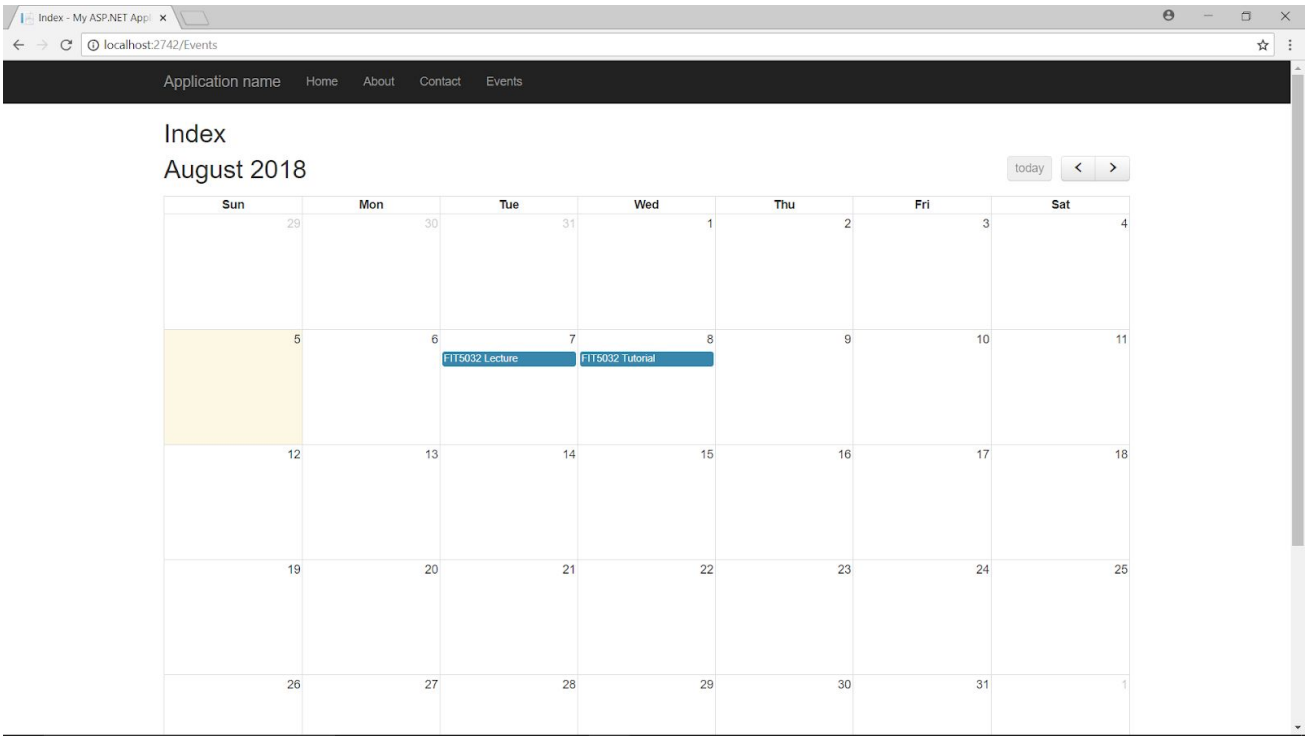
Upon the completion of this tutorial, you will gain a basic understanding of

- How to use FullCalendar at a basic level
- How to use query strings to process GET request

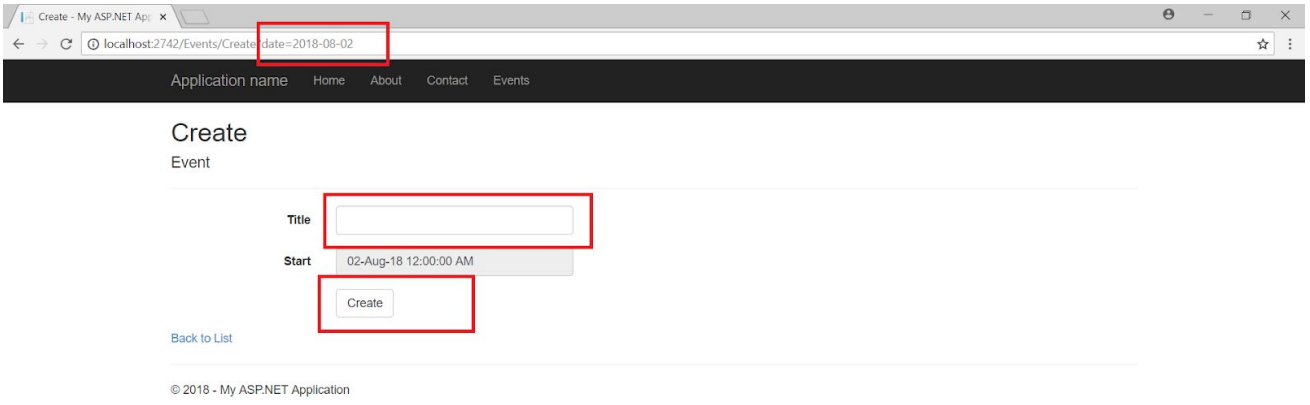
DoubtFire Submission

- None

Upon the completion of this tutorial. The end result of your web application would look like the following.



When a day on the Calendar is clicked, it will create a GET request with a Query String of the Day.



Step 1

This tutorial will be done slightly different, instead of going Step by Step, it will give an explanation of what is done to achieve it. So, you will be given the final completed version on Moodle. In a way, there is no need to follow these instructions unless you are interested how it was done.

The following, is the method and explanation of the various design decisions I have made to complete it.

I will create my Models first based on a Database First Approach. The provided SQL is as follows.

```
CREATE TABLE [dbo].[Events] (
    [Id]    INT           IDENTITY (1, 1) NOT NULL,
    [Title] VARCHAR (MAX) NOT NULL,
    [Start] DATETIME      NOT NULL,
    PRIMARY KEY CLUSTERED ([Id] ASC)
);
```

I have a selected a very simple database for this demonstration as it has only a Start time. You can further expand on this to include an End time as well. Of course these should have a constraint on them if you plan to do so. If you plan to include an End Time, how do you think can write the SQL constraint?

You will also realise that I have used the **DateTime** data type for this. In MS SQL, there are other data types that handles Date as well. For example what is the difference between DateTime and Date?

Normally when dealing with Dates, people often approach it from different ways. One of the more common approaches that I have seen is to use a "Data Warehouse" like methodology where the everything about the date is stored including the Unix timestamps. So, there are multiple fields in the operational database. (So, a single date is broken down into day, month, year, unix time stamp, and etc)

In fact, there are also times when Noda time is used. Link here. In JavaScript there is moment.js. In Java, there is JODA time.

Step 2

After that, I generated the Models based on that. Remember that there always needs to be a Model.

Step 3

I will then use the scaffolding feature to generate both the Controllers and Views. Remember that the purpose of this feature is to aid you. It does not mean that everything is 100% working out of the box. You should test if everything is working as intended.

Step 4

After that, I used the NuGet package manager and download the needed packages. Remember the downloading or updating the packages does not mean everything will be working out of the box.

Step 5

I then, created a bundle just for this purpose.

```
bundles.Add(new ScriptBundle("~/bundles/fullcalendar").Include(
            "~/Scripts/lib/jquery.min.js",
            "~/Scripts/lib/moment.min.js",
            "~/Scripts/fullcalendar.js",
            "~/Scripts/calendar.js"
            ));
```

Step 6

I will then write the calendar.js script needed to load the calendar and add extra classes to the view created. You do not normally need to include this in the bundle if you have different functions elsewhere.

```
var events = [];
$(".events").each(function () {
    var title = $(".title", this).text().trim();
    var start = $(".start", this).text().trim();
    var event = {
        "title": title,
        "start": start
    };
    events.push(event);
});

$("#calendar").fullCalendar({
    locale: 'au',
    events: events,

    dayClick: function (date, allDay, jsEvent, view) {
        var d = new Date(date);
```

```
            var m = moment(d).format("YYYY-MM-DD");
            m = encodeURIComponent(m);
            var uri = "/Events/Create?date=" + m;
            $(location).attr('href', uri);
        }
});
```

Index.cshtml of Events

```
@model IEnumerable<FIT5032_Week05D.Models.Event>
@{
    ViewBag.Title = "Index";
    Layout = "~/Views/Shared/_Layout.cshtml";
}
<h2>Index</h2>
<div id="calendar"></div>
<div style="display: none">
    <table class="table">
        <tr>
            <th>
                @Html.DisplayNameFor(model => model.Title)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Start)
            </th>
            <th></th>
        </tr>
        @foreach (var item in Model)
        {
            <tr class="events">
                <td class="title">
                    @Html.DisplayFor(modelItem => item.Title)
                </td>
                <td class="start">
                    @Html.DisplayFor(modelItem => item.Start)
                </td>
                <td>
                    @Html.ActionLink("Edit", "Edit", new { id = item.Id }) |
                    @Html.ActionLink("Details", "Details", new { id = item.Id }) |
                    @Html.ActionLink("Delete", "Delete", new { id = item.Id })
                </td>
            </tr>
        }
    </table>
</div>
@section Scripts {
    @Scripts.Render("~/bundles/fullcalendar")
}
```

Explanation

Using the same concept as done previously, I create an empty array to store the events in which a jQuery selector will be used to extract the values.

I also used an inline style to hide these values. Normally it is not such a good practice to use inline styles but it is acceptable in this scenario.

The array created earlier will be used to set the information in the calendar. You will need to refer to the FullCalendar documentation to see how to load the values correctly.

I also introduced a new function to in which whenever a day is clicked it will redirect the user to the "Create" page. Here, I passed in a **query string in which it is encoded.** Please refer to https://en.wikipedia.org/wiki/Query_string for what a Query String is. I also used moment js to format my date to a specific format. Then I will just JavaScript redirect with a relative URL to the Create page. I did not pass in the time in this scenario.

I will then, modifier my controller and change how it uses the GET request to accept a Query String.

```
// GET: Events/Create?date=YYYY-MM-DD
public ActionResult Create(String date)
{
    if (null == date)
        return RedirectToAction("Index");
    Event e = new Event();
    DateTime convertedDate = DateTime.Parse(date);
    e.Start = convertedDate;
    return View(e);
}
```

In this specific controller, I have changed to accept a query parameter date. If the value is null, it will redirect the user to the Index. If it not null, it will proceed and show the create View.

Conclusion

The objective of this supplementary material, is

- To showcase how to use FullCalendar.io at a basic level.
- How to Send Query String and use them in the Controller.
- To inform you that handling dates are normally very challenging if you are interested you can ask your tutor why this is so. (Or ask Google why)