Developing the Views in MVC

Introduction
This tutorial will build on the practices and the concepts you were introduced to in the lecture.

Required Software
For this tutorial, you will need access to the following software applications:
MS Explorer
Visual Studio
SQL SERVER

Reference
Walther et al, 'ASP.NET MVC Framework Unleashed', SAMS

Breakdown of Tutorial
This tutorial consists of the following tasks:

- To improve Forest by adding a Controller that other controllers inherit from.
- To develop ViewHelpers.
- To add validation to input fileds in an entry form.

You will be given a series of practical exercises, along with selected questions to consolidate your learning.

Tutorial objectives
In this tutorial, we will carry on working on *Forest*. First we will extend forest to take advantage of a controller that the rest of the controllers will inherit from. In a previous tutorial, we built *Forest* to include functionality for adding a *Music_Recording*. In this exercise we will extend *Forest* to include new features.

Review questions

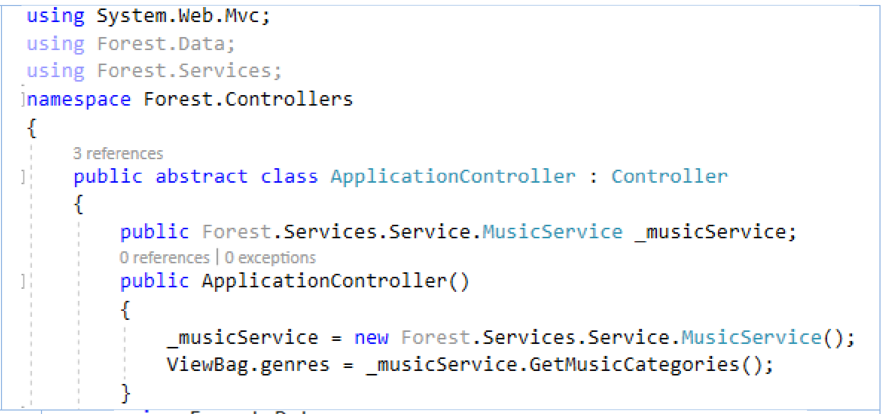
	What is 'DRY' principle? What is the application of principle in this context?
	In building *Forest* we made use of Interfaces. What do you understand an Interface to be? Under what circumstances do we use an Interface?
	In extending *Forest* here we will make use of Abstract class. What do you understand an Abstract class to be? Under what circumstances do we use an Abstract class?
	Elaborate on how you think how *Abstract* class and *Interface* compare and contrast.

Improving the Design of *Forest* – Creating the *ApplicationController*
Consider *Forest*. At the moment the application opens, showing the links *Music* and *Video*. You access *Music_Category* facilitated by *MusicController*. Data is passed to the *MusicController* using *viewBag* property that both the controller and its associated views share.

You may want to display the list of M*usic_Category* on your Master Page. You can make use of *viewBag* to pass data to your master page. What you do not want to do is to add the data to each of the controllers that will make use of that data. Instead you should create a base class for all controllers that all controllers would then implement. *ViewBag* is then modified in the base controller.

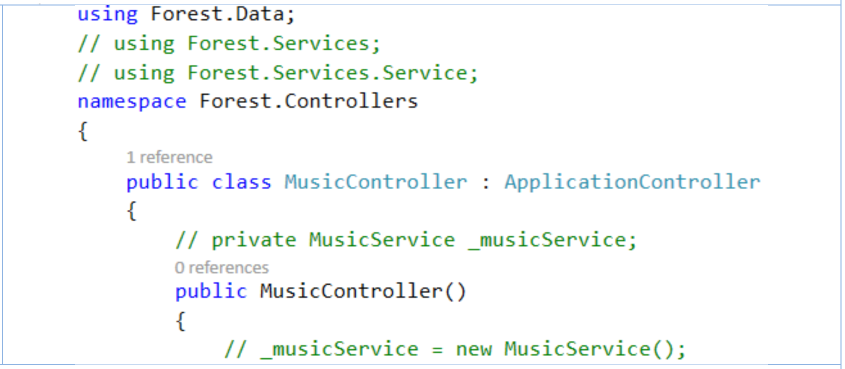
Working on *Forest;*

_ Right click on *controllers* folder and add a new *MVC Controller - Empty*
_ Make the *ApplicationController* class an *abstract* class
_ Ensure that you place the imports for the required *namespaces*
_ Ensure that class inheits *Controller* class
_ Create the *MusicService* object in the constructor
_ Place an *IList<Music_Category>* in the ViewBag and name it *cats*



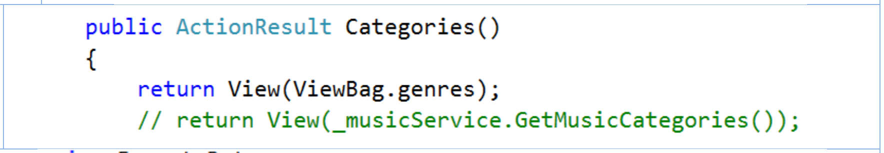
```
using System.Web.Mvc;
using Forest.Data;
using Forest.Services;
namespace Forest.Controllers
{
    3 references
    public abstract class ApplicationController : Controller
    {
        public Forest.Services.Service.MusicService _musicService;
        0 references | 0 exceptions
        public ApplicationController()
        {
            _musicService = new Forest.Services.Service.MusicService();
            ViewBag.genres = _musicService.GetMusicCategories();
        }
```

Working on the *MusicController*, this controller can now inherit from the '*ApplicationController*' as opposed to the '*Controller*' class.
_ Ensure that *MusicController* inherits *ApplicationController*
_ Comment out the lines that would create your *MusicService* object



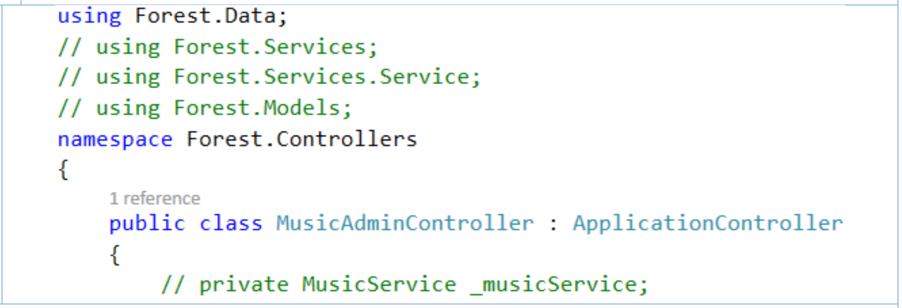
```
using Forest.Data;
// using Forest.Services;
// using Forest.Services.Service;
namespace Forest.Controllers
{
    1 reference
    public class MusicController : ApplicationController
    {
        // private MusicService _musicService;
        0 references
        public MusicController()
        {
            // _musicService = new MusicService();
```

Working on the *MusicController*, we can also amend the code in the *Categories* action method to take advantage of what we have placed in *ViewBag*;



```
public ActionResult Categories()
{
    return View(ViewBag.genres);
    // return View(_musicService.GetMusicCategories());
```

Working on the *MusicAdminController*, we can amend to take advantage of *ApplicationController*;



```
using Forest.Data;
// using Forest.Services;
// using Forest.Services.Service;
// using Forest.Models;
namespace Forest.Controllers
{
    1 reference
    public class MusicAdminController : ApplicationController
    {
        // private MusicService _musicService;
```

Test your application

The constructor in the *ApplicationController* adds to the Data Dictionary using the *ViewData* property of the *ApplicationController*. Since *MusicController* inherits from the *ApplicationController*, it will inherit the data dictionary and consequently the *IList<Music_category>*. Anchor to the list is *cats*.
Thi sis also the case for the *MusicService* object. This object si also inherited.

Using View Helpers

In MVC Web Applications we do not have the ever useful ASP.Net Web Controls. Instead we have *Html Helpers*. Essentially *Html Helpers* render Html tags for us in exactly the same way that an ASP.Net TextBox WebControl renders a TextBox. There are standard helpers that we are provided with but the list is finite. We often need to build our own custom Html Helpers to render the required tags.

Review Questions

Open the *Forest* project. Under *../Views/MusicAdmin/* locate *EditMusicRecording.cshtml*. Inspect the source code and answer the following questions. You can get help at http://msdn.microsoft.com/en-us/library/system.web.mvc.htmlhelper.aspx.

What HTML Helper methods do you see in the code? State and describe the functionality of as many as you can.

We have used a *Using* with *Html.BeginForm()*. What is the use of *Using*? Can we do away with it?

Adding the 'Terms and Conditions' Download

Review Questions

How would you create a download in a conventional non-MVC Web page?

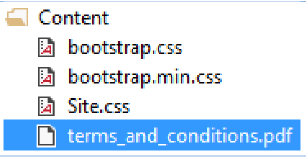
This method will not work for MVC applications. Why is that?

One type of *ActionResult* is *FileResult*. *FileResult* inherits *ActionResult*. Helper method for *FileResult* is *File*. *File* helper method of the controller returns a *FileResult*.

To add the file download we have to do the following:

- Create the '*Terms and Conditions*' document
- Create an action method in a controller for the download
- Create a link to the action method

_ Create the *Terms and Conditions* document and place it in the *Content* folder of your MVC project
Note: For this you need to decide on a *MIME* type for the file. Examples of MIME types maybe .doc and .pdf.



```
Content
    bootstrap.css
    bootstrap.min.css
    Site.css
    terms_and_conditions.pdf
```

_ Create an action method in a controller.
Note: In this case I have decided to use the *HomeController* and a *MIME* type of .pdf. Note that you need to pass three parameters to the *File* helper method; Location of the file, *MIME* type of the file, name of the file. For a comprehensive list of *MIME* types, see appendix below.

```
public ActionResult DownloadTerms()
{
    return File("~/Content/terms_and_conditions.pdf",
        "application/pdf", "terms_and_conditions.pdf");
}
```

_ Add a link to the action method
Note: I have decided to add this link to the *site master*. The site master in *Forest* project is *_Layout.cshtml*. You will find in *Forest/Views/Shared/* folder.

```
<li>@Html.ActionLink("Terms & Conditions",
                "DownloadTerms", "Home")</li>
```

_ Test your application

Rendering an Image Link ViewHelper

The object of this exercise is to work on *Recordings* view and to display images in palce of hyperlinks. To render an image in place of text as a hyperlink, we need to;

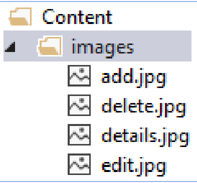
- Create a *<a>* tag. The standard form of this tag is `<a href="url">Link text</a>`.
- The *href* of the tag must now point to a *ControllerAction* and the *Link* text of the tag must point to an image, i.e. an *<img>* tag.

We cannot use the *Html.ActionLink()* helper to render an image link. This is because this helper HTML encodes its link text automatically and therefore you are not able to pass an *<img>* tag to this method and expect it to render as an image. Instead we need to use the *Url.Action()* helper to generate the link.

'*Url.Action()*' helper supports a set of parameters similar to those of '*Html.ActionLink()*' helper.
In this case the helper is invoking the '*Delete*' action and its passing '*id*' of the current item to the action.

Working on *Forest* project;

_ Downlod *JPG* images for the actions
_ Add a sub folder to Content folder and name it *images*
_ Place the downloaded images in the *images* folder



```
Content
  ◢ images
      add.jpg
      delete.jpg
      details.jpg
      edit.jpg
```

Working on *GetMusicRecordings* view and *Edit* hyperlink;

_ Create *Url.Action()* helper

```
@Url.Action("EditMusicRecording",
    new { id = item.Id, Controller = "MusicAdmin"})
```

_ Create the *<img>* tag.

```
<img src="~/Images/edit.jpg"
    alt="Edit" style="height:20px;width:20px" />
```

Putting it together;
_ Create a *<a>* tag

Note; *Url.Action* is passed as *href* of *<a>* tag. *<img>* tag becomes the value that is enclosed by the *<a>* tag.

```
<a href=@Url.Action("EditMusicRecording",
       new { id = item.Id, Controller = "MusicAdmin" })>
    <img src="~/Images/edit.jpg"
         alt="Edit" style="height:20px;width:20px" />
</a>
```

_ Repeat the exercise for the *Details* and the *Delete* actions

_ Test your application

What is the difference between *Html.ActionLink* and *Url.Action*?

Rendering a DropDownList ViewHelper

The object of this exercise is to use a *DropDownList ViewHelper* to display genres of music. This *ViewHelper* will be employed in the *AddMusicRecording* view to facilitate opting for a genre. Recall that we pass *genre* to this action. It would be good to initialise the *DropDownListFor* to the passed *genre*.

To render a *DropDownList* we have to do the following:
Work on the *AddMusicRecording* action in controller;

- Prepare a *List* object, Items of which are the genres and it is initialised to the passed genre.
- Place the *List* object inside the *ViewBag*.

Work on the *AddMusicRecording* view;

- Work on the *AddMusicRecording* view and for the *Genre* property to replace the *EditorFor* HTML-Helper for a *DropDownListFor* HTML-helper.
- Retrieve the *List* object from the *ViewBag* and pass it to the *SelectList* property of the *DropDownListFor* HTML-helper

Working on *MusicAdminController* and the *HTTPGet AddMusicRecording* action:

_ Prepare a *List* object, Items of which are the genres and it is initialised to the passed genre
Note; Note that we have renamed the parameter that is passed to the action to *selectedGenre*.

_ Place the *List* inside inside the *ViewBag*
_ Return a *View*

```
[HttpGet]
0 references
public ActionResult AddMusicRecording(string selectedGenre)
{
    List<SelectListItem> genreList = new List<SelectListItem>();
    foreach(var item in _musicService.GetMusicCategories())
    {   genreList.Add   (
            new SelectListItem()
            {   Text = item.Genre,
                Value = item.Id.ToString(),
                Selected = (item.Genre == (selectedGenre) ? true : false)
            }       );
    }
    ViewBag.genreList = genreList;
    return View();
```

Working on *HttpGet-AddMusicRecording* view:
_ Replace the *EditorFor* HTML-Helper for a *DropDownListFor* HTML-helper
_ Retrieve the *List* object from the *ViewBag* and pass it to the *SelectList* property of the *DropDownListFor* HTML-helper

```
@*@Html.EditorFor(model => model.Genre ,
     new { htmlAttributes = new { @class = "form-control" } })*@
@Html.DropDownListFor(model => model.Genre,
     (List<SelectListItem>)ViewBag.genreList)
```

_ Test your application

Appendix

Included here is a list of MIME types. This list Is not exhaustive.

.xlsx	application/vnd.openxmlformats-officedocument.spreadsheetml.sheet
.xltx	application/vnd.openxmlformats-officedocument.spreadsheetml.template
.potx	application/vnd.openxmlformats-officedocument.presentationml.template
.ppsx	application/vnd.openxmlformats-officedocument.presentationml.slideshow
.pptx	application/vnd.openxmlformats-officedocument.presentationml.presentation

.sldx	application/vnd.openxmlformats-officedocument.presentationml.slide
.docx	application/vnd.openxmlformats-officedocument.wordprocessingml.document
.dotx	application/vnd.openxmlformats-officedocument.wordprocessingml.template
.xlam	application/vnd.ms-excel.addin.macroEnabled.12
.xlsb	application/vnd.ms-excel.sheet.binary.macroEnabled.12
.pdf	application/pdf