JÖNKÖPING UNIVERSITY

Lab 0

This exercise is briefly outlined below, we will go through it together step by step.

Part 1 - Creating a MVC application

Create an MVC application using Visual Studio – Name it "ProjectManager"

Add a controller ("HomeController") with an action-method "Index", the action method should come up default.

- To add a controller right-click the "Controllers" map and click "Add" then "Controller". In the window that comes up select "MVC 5 Controller – Empty". In the next window change the name to "HomeController".
- Change the returnvalue of the action-method from "ActionResult" to "ViewResult".

Add a view ("Index")

- To add a view you may right-click the action method it should be associated to, in this case "Index", and select "Add view"
- The view should have the name "Index", which should come up default. Do not use any layout page or model.

Populate the view with

- Some greeting information
- Use a ViewBag to send information about the date and time and display this information in the view

Test the application
The browser should show the greeting information. Note the URL the page uses, it should be Localhost:<port number>/Home/Index

Part 2 – Add an ActionLink

Add an "actionlink" which redirects to an action for listing departments. See to that the action link redirects to an action called "ListDepartments" as the second parameter.

Add a new action-method to the HomeController and name the action "ListDepartments"

- You can add a new action-method by placing the cursor in an empty space in the controller, write "mvc" and press tab-button twice.

Test the application
The browser should show the greeting information and a clickable link.
However, clicking the link will give us an error. Why?
The error gives us a list of searches the view engine tries out, to find the correct view.

Part 3 – Add an additional view

Add a view named “ListDepartments” and give it a heading “Departments”, just to show something. (We will later add some data dynamically).

If you test the application by “Start debugging” (F5) it will not start from the index page but start from the view you were editing. If you want to prevent this behaviour, of VS, you may set a specific start page.

- Under Project -menu in VS start “ProjectManager Properties”-window. Select the “Web”-section and check the “Specific page” option in the “Start action” category.

If you rename the view to something else, e.g. “TheListOfDepartments” the application will not find the view (of course). However you can force the action-method to find it by explicitly direct the action method to the view.

- To rename the view right click the view and select rename
- To direct the action method to the renamed view pass the new view-name, as a parameter, to the call of the View-method.

Part 4 – Add an additional controller

Add a second controller class called “UserLists”

- Right click the Controllers map and select Add -> Controller
- Select “MVC 5 Controller – Empty”
- Change default name to “UserListsController”

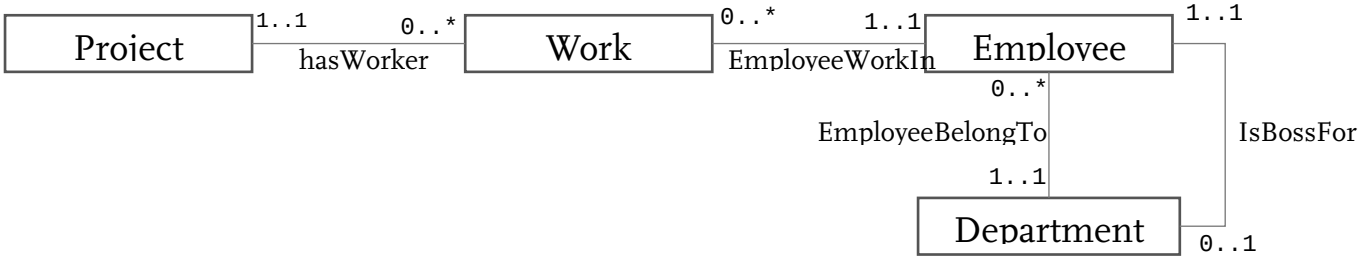
From the HomeController class cut the action-method “ListDepartments” and paste it into the UserListsController. When creating a controller, it will automatically create an action-method “Index”, however in this case you can simply remove it.
It should automatically create a folder “UserLists” under the Views-folder. However, if it does not show up you can easily create it by adding it manually.
Move the view “TheListOfDepartments” to the UserLists-folder.
For the Index – view in the Home – view folder to find the action-method ListDepartments, which now has been moved to the UserListsController class, it is necessary to change the ActionLink:

- Add “UserLists” as a third parameter to the ActionLink call.

Test that the application works as before.

Part 5 – Add a model

For our application to use some data we will add some classes in the Models folder. These classes will be used to fake some data which, at this point, will be hardcoded, instead of read from a database. (Later in the course we will add the database support).
The following ER-model serves as guide for the context of the data:



Add a class for Department in the Models folder:

```
public class Department
    {
        public int depid { get; set; }
        public string name { get; set; }
        public int bossid { get; set; }
```

- Right click Models folder and do Add -> New item
- In C# - code select Class
- Name the class “Department.cs”

Build the solution. (Otherwise the class will not be recognised)

Part 6 – Create a strongly typed view which add a department

We will create a strongly typed view for adding a new department. To do that we need an action-method for adding the department. Do the following:

- Add the action-method “AddDepartment” to the UserListsController class
- Add an ActionLink, with the text “Add department”, for calling the action-method from the Index view
- Add a view to be started by the action-method addDepartment (a view with the same name). In the Add View window set the Template to “Empty” and set Model class to “Employee (ProjectManager Models)”.

The first row in the created view should now have the following line:

```
@model ProjectManager.Models.Department
```

For the controller to have access to the model classes add, in the UserListsController, the following using-statement:

```
using ProjectManager.Models;
```

In the action-method AddDepartment create a department object:

- Add the following code

  ```
  Department depObj = new Department();
  ```

  Which creates an empty object.

Pass the Department object to the view by adding it to the parameterlist when the view is called

```
return View(depObj);
```

The Department object will now be the Model object in the view.

In the AddDepartment view add a form using the helper method Html.BeginForm() by adding the following code:

```
<h1> Add a department</h1>
@using(Html.BeginForm())
{
    @:Department id: @Html.TextBox("depid", Model.depid)
    @:Name: @Html.TextBox("name", Model.name)
    @:Boss id: @Html.TextBox("bossid",Model.bossid)
    <input type="submit" value="Add"/>
}
```

The form will automatically use a POST for sending the data to the AddDepartment action-method. We therefore need to create a second method with the same name, but preceded with the attribute

[HttpPost].

```
        [HttpPost]
        public ViewResult AddDepartment(Department depObj)
        {
            return View("ShowDepartment",depObj);
        }
```

The metoden should in the real case add the new data to the database and perhaps call the view TheListOfDepartments to update the view's list.

For the time being we might add a temporary strongly typed view (`"ShowDepartment"`) to display what we just entered.

Part 7 – Creating mockup data

In the class UserListsController add the mockup code creating a list of departments:

```
static private List<Department> departmentList = new List<Department>
{
   new Department {depid = 1, name = "Administration", bossid = 106},
   new Department {depid = 2, name = "Manufactoring", bossid = 104},
   new Department {depid = 3, name = "Advertisement", bossid = 107}
};
```

(The property depid is unique id for the department and bossid is a reference to empid in Employee. For now we will just treat the values entered just as values without any references).

By adding it in the class outside the action-methods the departmenList is an object available in all methods.

Part 8– Call the view to list all departments

The view TheListOfDepartments was not created as a strongly typed view. We can manually add the row

```
@model List<ProjectManager.Models.Department>
```

in the top of the view so that the view becomes strongly typed, and will as Model handle an object containing a list of Departments.

Change the return statement in the method ListDepartments to send the list-object:

```
return View("TheListOfDepartments",departmentList);
```

In the view TheListOfDepartments add the following code:

```
 <h1>Departments</h1>
 <ul>
      @foreach (var department in Model)
      {
         <li>
             Department id: @department.depid, Department name: @department.name
         </li>
      }
 </ul>
```

Test the application