Add Dashboard

Contents

[1. Introduction 2](#_Toc14183748)

[2. Create a DashBoard Class 3](#_Toc14183749)

[3. Implementations for Classes 6](#_Toc14183750)

[4. Create View for Dashboard 8](#_Toc14183751)

[5. AggregateBase 10](#_Toc14183752)

[6. Attachments 14](#_Toc14183753)

[Class TestDashBoard 14](#_Toc14183754)

[Class TestClass with Dashboard Methods and using IAggregate Interface 15](#_Toc14183755)

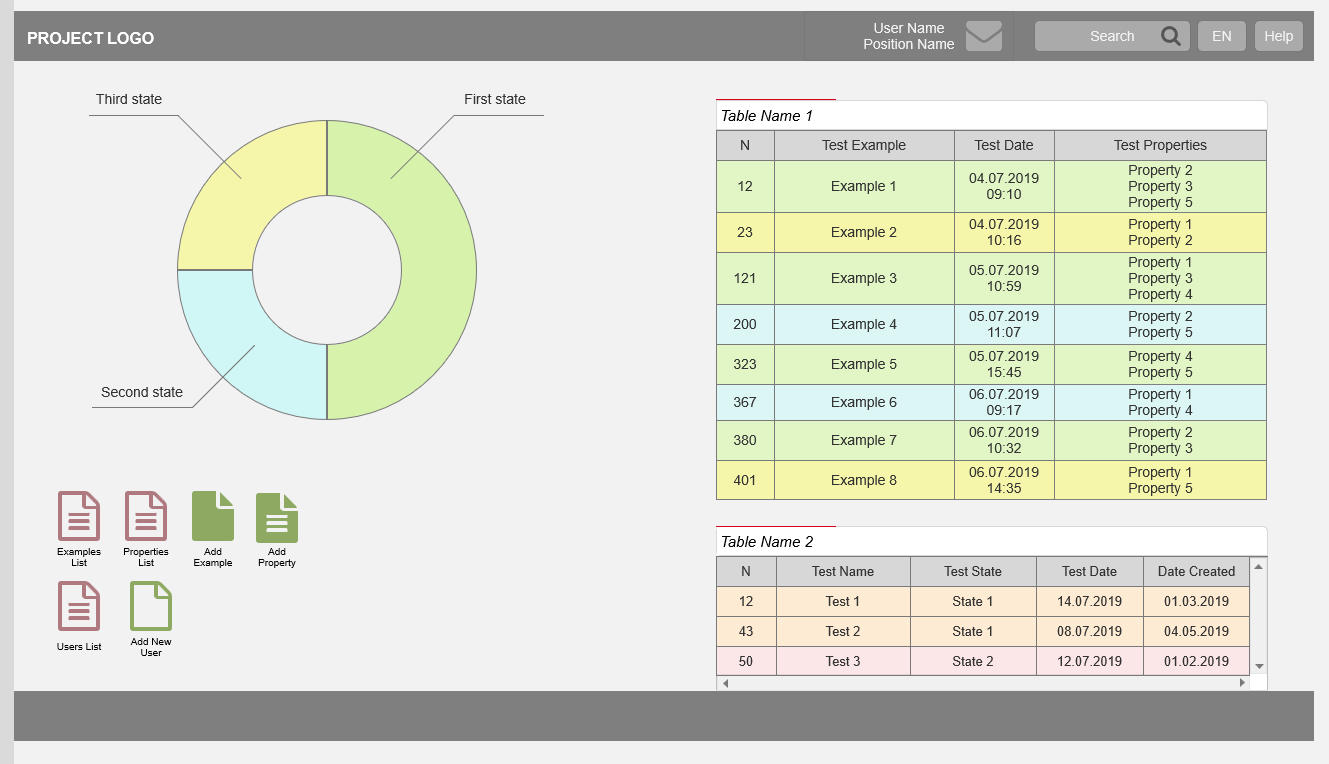
# 1. Introduction

Dashboard is the main page of the web site, which is unique for each user. Here will be located the following content:

* Pie Chart (a pie diagram, that will display the definite objects and those statistic)
* Icons (links to various pages)
* List (list of data, displayed in a table)

For realization of these components, it is necessary to create a class of the dashboard, to implement the methods, which help to make the list and pie chart, create the db procedures for them, create the view for dashboard and add the required icons.

\* Replace Test by your class

**

# 2. Create a DashBoard Class

1.1. Add to \Enums\DashboardEnum.cs:

Test = 4

Where:

* Test – the name of dashboard
* Number – the serial number of the dashboard (number, integer)

1.2. Add Dashboard class to \mvc\Dashboards:

namespace MedCore.Dashboards

{

public class TestDashBoard : BaseDashboard

{

public DashboardEnum DashboardType = DashboardEnum.UserAdmin;

}

}

This class represents the dashboard. The DashboardType represents which user owns the dashboard

1.3. Implement in Dashboard class the function Load – function that is responsible for initial load:

public override void Load(ViewDataDictionary ViewData, User currentUser)

{

var context = new HttpContextWrapper(HttpContext.Current);

var currentPerson = LIB.Tools.Security.Authentication.GetCurrentUser().Person;

ViewData["List"] = Test.LoadLatest(currentPerson);

ViewData["GroupedList"] = TestState.LoadLatests(currentPerson.Id);

ViewData["Dashboard"] = DashboardType;

ViewData["DashboardControl"] = "\_TestList";

}

**ViewData["List"]** – represents the list on dashboard, to which is assigned the method “Load Latests()” from the required class.

**ViewData["GroupedList"]** –represents the Pie Chart on dashboard, to which is assigned the method “Load Latests()” from the required class.

**ViewData["Dashboard"]** –is written the type of dashboard (which user owns the dashboard)

**ViewData["DashboardControl"]** – represents the part of view that will display the list

1.4. (Optional) Implement in Dashboard class **RefreshWidget** – function that is responsible for ajax refresh:

public override Dictionary<string, object> RefreshWidget(ViewDataDictionary ViewData, ControllerContext ControllerContext, TempDataDictionary TempData, User currentUser, long lastId, int count, string widgetitems)

1.5 (Optional) To create the possibility to access the dashboard by another user, it is necessary to add this part of code into the **DashBoardController.cs** in getter of **Dashboard**:

else if (currentUser.HasAtLeastOnePermission((long)BasePermissionenum.TestAccess))

{

return new TestDashBoard();

}

# 3. Implementations for Classes

The following tool is necessary for displaying the List and the Pie Chart on the dashboard. For each element is necessary to implement the method LoadLatest (one for List and another for Pie Chart), as well and the procedures for them.

2.1 Implement in the class the function **LoadLatest** by this exemple:

public static List<Test> LoadLatest(Person pOwner, long lastestId = 0)

{

var conn = DataBase.ConnectionFromContext();

var tests = new List<Test>();

var cmd = new SqlCommand("Test\_Populate\_Latests", conn) { CommandType = CommandType.StoredProcedure };

cmd.Parameters.Add(new SqlParameter("@OwnerId", SqlDbType.BigInt) { Value = pOwner.Id});

using (var rdr = cmd.ExecuteReader(CommandBehavior.SingleResult))

{

while (rdr.Read())

{

var od = (Test)(new Test()).FromDataRow(rdr);

tests.Add(od);

}

rdr.Close();

}

return tests;

}

*Description:*

* The row :

var cmd = new SqlCommand("Test\_Populate\_Latests", conn) { CommandType = CommandType.StoredProcedure };

Is responsible for connecting a specific procedure that will display data from the database in the list.

* The row :

cmd.Parameters.Add(new SqlParameter("@OwnerId", SqlDbType.BigInt) { Value = pOwner.Id});

Is responsible for sending the necessary parameters to the procedure.

2.2 To display data in the list for the previous method, you need to create the procedure **Test\_Populate\_Latests**:

CREATE PROCEDURE [dbo].[Test\_Populate\_Latests]

@OwnerId bigint

AS

BEGIN

SELECT v.TestId,v.Name,v.DateStart

,v.TestStateId,vs.Name as TestStateName

,pc.PredefinedColorId, pc.Code as PredefinedColorCode

FROM Test v

INNER JOIN TestState vs ON vs.TestStateId=v.TestStateId

INNER JOIN PredefinedColor pc ON pc.PredefinedColorId=vs.PredefinedColorId

WHERE v.OwnerId = @OwnerId AND v.DeletedBy IS NULL

END

# 4. Create View for Dashboard

1. Add the view for the list on Dashboard **/Views/DashBoard/\_TestList.cshtml:**

In this view should be written the name of columns of the table.

@using LIB.BusinessObjects

@using MedCore.Models.Objects

@if (((List<Test>)ViewData["List"]).Count > 0)

{

<script type="text/javascript">

var gLatestDashboardId = @(((List<Test>)ViewData["List"]).First(i => i.Id>0).Id);

var gDashboardItemsCount = @(((List<Test>)ViewData["List"]).Count);

</script>

<form class="table-widget">

<div class="dashboard-ico">

<div class="outer-ico">

<div class="widget-caption">

Tests

</div>

<div class="data-grid-container">

<div class="data-grid">

<div class="data-grid-title-row">

<div class="data-grid-title">N</div>

<div class="data-grid-title">Name</div>

<div class="data-grid-title">Date Start</div>

<div class="data-grid-title">Curent State</div>

</div>

@Html.Partial("Controls/\_TestsRows")

</div>

</div>

</div>

</div>

<div class="clear" style="height:1px"></div>

</form>

}

2. Add Rows for created view **Views/DashBoard/Controls/(\_TestsRows.cshtml):**

In this view should be written the values for each column of the table.

@using MedCore.Models.Objects

@using MedCoreLib.BusinessObjects

@{

Layout = "~/Views/Master/\_Ajax.cshtml";

}

@{ var index = ((List<Test>)ViewData["List"]).Count + (int)ViewData["DashboardListCount"];}

@foreach (var test in ((List<Test>)ViewData["List"]))

{

<a href="@LIB.Tools.Utils.URLHelper.GetUrl("DocControl/Test/" + test.Id )" target="\_blank" class="data-grid-data-row data-grid-data-row-@(test.Id) data-grid-data-row-type-@(test.TestState.PredefinedColor.Code)">

<div class="data-grid-data"><b>@(test.Id)</b></div>

<div class="data-grid-data">@(test.Name)</div>

<div class="data-grid-data">@(test.DateStart.ToString("dd.MM.yyyy HH:mm"))</div>

<div class="data-grid-data">@(test.TestState.Name)</div>

<input type="hidden" name="widgetitems" value="@(test.Id):@(test.TestState.Id)" />

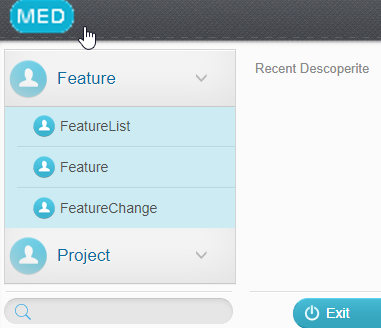
</a>

}

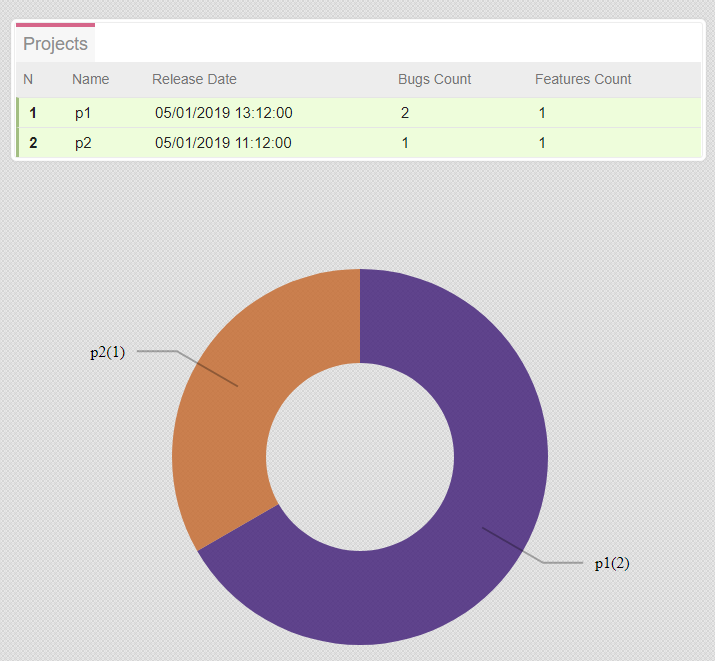
\* For adding the icons source the Document “AddIcon”

# 5. AggregateBase

This utility is made to avoid the problem of not displaying certain pages in the quick access menu.



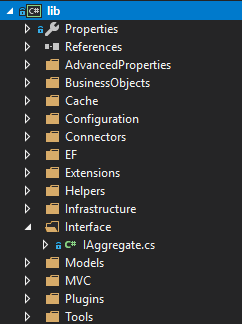
This problem occurs when it is necessary to implement certain methods for creating a pie chart and lists on a dashboard.



Inherited classes conflict, thus it was decided to create an interface that will be inherited by your class without going into conflict with the implemented class ItemBase.

1. Create new Interface

Create new folder in the directory \Lib\**Interfaces**.

After that create in that folder new Interface: **IAggregate.cs**

public interface IAggregate

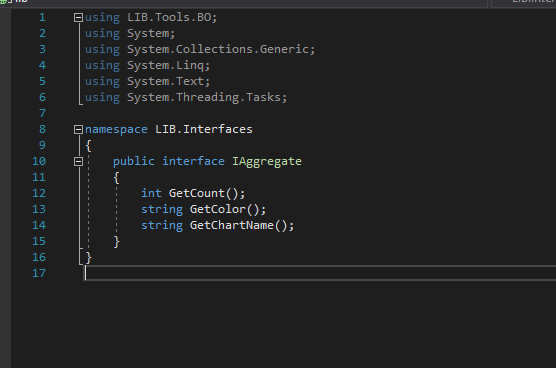
{

int GetCount();

string GetColor();

string GetChartName();

}

2. Copy the following code into this interface

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace LIB.Interface

{

public interface IAggregate

{

int GetCount();

string GetColor();

string GetChartName();

}

}

Methods explanation:

* GetCount() – the count number on PieChart
* GetColor() – the color that is generated on PieChart
* GetChartName() – the names of the PieChart sections

3. In the class, which will be used for PieChart, change the parent class, implementing the IAggregate Interface:



Your class will inherit the parent class PrintBase and the created interface IAggregate at the same time.

4. Implement these methods into the class:

public int GetCount()

{

return TestCount;

}

public string GetColor()

{

return ColorHelper.HexConverter(ColorHelper.generateRandomColor(new System.Drawing.Color(), Convert.ToInt32(Id)));

}

public string GetChartName()

{

return GetName();

}

public override string LoadReport(ControllerContext ControllerContext, ViewDataDictionary ViewData, TempDataDictionary TempData, ExportType type = ExportType.None)

{

throw new NotImplementedException();

}

And the propery:

[Common(EditTemplate = EditTemplates.Hidden), Db(\_Editable = false, \_Populate = false, \_Ignore = true)]

public string Color { get; set; }

5. In the view \DachBoard\Controls\\_Chart.cshtml change the code by the following example:

@using Weblib.Models

@using GofraVersionsweblib.Enums

@using GofraVersionsLib.BusinessObjects

@using System.Collections.Generic;

@using LIB.Interface;

@using LIB.Tools.BO;

@{

Layout = "~/Views/Master/\_Ajax.cshtml";

var Dashboard = ViewData["Dashboard"] is DashboardEnum ? (DashboardEnum)ViewData["Dashboard"] : DashboardEnum.None;

var GroupedList = ((Dictionary<long, IAggregate>)ViewData["GroupedList"]);

var index = 0;

}

gChartDataArray = d3.scale.ordinal()

.domain(

[

@if (GroupedList != null)

{

foreach (var item in GroupedList.Values)

{

@: @(index == 0 ? "" : ",")"@(item.GetChartName()):@(item.GetCount()):@(((ItemBase)item).Id)"

index++;

}

}

]

)

.range(

[

@if (GroupedList != null)

{

index = 0;

foreach (var item in GroupedList.Values)

{

@:@(index == 0 ? "" : ",")"@(item.GetColor())"

index++;

}

}

]);

# 6. Attachments

## Class TestDashBoard

namespace MedCore.Dashboards

{

public class TestDashBoard : BaseDashboard

{

public DashboardEnum DashboardType = DashboardEnum.UserAdmin;

public override void Load(ViewDataDictionary ViewData, User currentUser)

{

var context = new HttpContextWrapper(HttpContext.Current);

var currentPerson = LIB.Tools.Security.Authentication.GetCurrentUser().Person;

ViewData["List"] = Test.LoadLatest(currentPerson);

ViewData["GroupedList"] = TestState.LoadLatests(currentPerson.Id);

ViewData["Dashboard"] = DashboardType;

ViewData["DashboardControl"] = "\_TestList";

}

}

}

## Class TestClass with Dashboard Methods and using IAggregate Interface

public class TestState : AggregateBase

{

#region Constructors

public TestState()

: base(0)

{

}

public TestState(long id)

: base(id)

{

}

#endregion

public override int GetCount()

{

return TestCount;

}

public static Dictionary<long, AggregateBase> LoadLatests(long pOwner)

{

var conn = DataBase.ConnectionFromContext();

var testStates = new Dictionary<long, AggregateBase>();

var cmd = new SqlCommand("TestState\_Populate\_Latests", conn) { CommandType = CommandType.StoredProcedure };

if (pOwner > 0)

cmd.Parameters.Add(new SqlParameter("@OwnerId", SqlDbType.BigInt) { Value = pOwner });

using (var rdr = cmd.ExecuteReader(CommandBehavior.SingleResult))

{

while (rdr.Read())

{

var state = (TestState)(new TestState()).FromDataRow(rdr);

state.Color = rdr["Color"].ToString();

testStates.Add(state.Id, state);

}

rdr.Close();

}

return testStates;

}

#region Properties

[Template(Mode = Template.Name)]

public string Name { get; set; }

[Common(EditTemplate = EditTemplates.Hidden), Db(\_Editable = false, \_Populate = false)]

public int TestCount { get; set; }

[Template(Mode = Template.DropDown)]

public PredefinedColor PredefinedColor { get; set; }

#endregion }}