Лабораторная работа №3

Частичные представления и компоненты представления. Передача данных представлению

1. Цель работы.

Изучение возможностей получения частичной разметки страницы. Знакомство с механизмом передачи данных представлению.

2. Общие сведения.

Для передачи данных между контроллером и представлением используются объекты ViewData и ViewBag. Они представляют собой словари, формируемые динамически, и доступные как свойства в контроллере и в представлении.

Пример использования ViewData:

```
ViewData["Text"] = "Лабораторная работа 2";
```

или

```
[ViewData]
public string Text { get; set; }
```

ViewData также предоставляет свойство **Model**, которое может представлять собой объект класса. В представлениях Razor свойство Model доступно динамически. Это значит, что можно получить доступ к свойствам модели, не зная типа класса модели. Однако, для использования механизма IntelliSense желательно указать, к какому классу относится модель. Для этого используется ключевое слово **@model.** Такое представление называется строго типизированным. Модель передается в представление как параметр.

Частичное представление – это разметка, которая помещается внутри другой разметки.

Для вызова частичного представления можно использовать вспомогательный метод **@Html.Partial()** или **@Html.RenderPartial()**. Однако, предпочтительнее использовать тэг <partial>, например:

```
<partial name="_UserPartial" />
```

Компоненты представления (ViewComponent), как и частичные представления, предназначены для создания части разметки, которая затем помещается на страницу. Однако, компонент позволяет реализовать сложную бизнес-логику. В этом работа компонента похожа на работу контроллера, но в отличие от контроллера класс компонента содержит один публичный метод

```
public IViewComponentResult Invoke()
```

Вызов компонента на странице осуществляется с помощью команды:

```
@await Component.InvokeAsync("Имя компонента")
```

Компоненты принято размещать в папке Components (необязательно).

Представления размещаются по пути:

*/Views/**контроллер**/Components/**ИмяКомпонента**/Default.cshtml*

или

*/Views/Shared/Components/**ИмяКомпонента**/Default.cshtml*

3. Выполнение работы

Используйте проект из лабораторной работы №2.

3.1. Задание №1

Требуется оформить меню сайта в виде компонента «Menu».

Для активного пункта меню предусмотреть свой CSS-стиль.

3.1.1. Рекомендации к заданию №1

В папке Models создайте класс MenuItem, описывающий параметры элементов меню сайта:

- является ли вызываемая ссылка страницей (Razor Page) или методом контроллера;
- текст надписи;
- имя контроллера;
- имя метода;
- имя страницы;

- имя области (Area);

- имя класса CSS для текущего (активного) пункта меню.

В классе компонента меню создайте коллекцию исходных данных главного меню: `List<MenuItem> items = new List<MenuItem> { . . .}`

Полученную коллекцию передавать представлению в качестве модели. В представлении компонента эта коллекция обходится в цикле «foreach» для создания элементов навигационной панели.

Для определения текущего пункта меню (для назначения соответствующего стиля кнопке меню) можно использовать свойство ViewContext:

```
ViewContext.RouteData.Values["controller"];
ViewContext.RouteData.Values["page"];
ViewContext.RouteData.Values["area"];
```

3.2. Задание №2

Информацию пользователя оформить в виде частичного представления _UserPartial.cshtml.

Информацию о корзине заказа оформить в виде компонента «Cart»

4. Контрольные вопросы

Как передать в представление одновременно несколько объектов разных классов?

Что такое строго типизированное представление?

5. Пример выполнения работы

5.1. Подготовка класса описания элемента меню

В папку Models добавьте класс MenuItem.cs:

```
public class MenuItem
{
    // является ли страницей или методом контроллера
    public bool IsPage { get; set; } = false;
    // имя области
    public string Area { get; set; } = "";
    // имя действия контроллера
```

```
        public string Action { get; set; } = "";
        // имя контроллера
        public string Controller { get; set; } = "";
        // имя страницы
        public string Page { get; set; } = "";
        // класс CSS для текущего пункта меню
        public string Active { get; set; } = "";
        // текст надписи
        public string Text { get; set; } = "";
}
```

5.2. Создание компонента

Добавьте в проект папку Components

В папку Components добавьте класс MenuViewComponent. Класс должен наследоваться от класса ViewComponent. Создайте в классе компонента коллекцию элементов списка меню:

```
// Инициализация списка элементов меню
private List<MenuItem> _menuItems = new List<MenuItem>
{
        new MenuItem{ Controller="Home", Action="Index", Text="Lab 2"},
        new MenuItem{ Controller="Product", Action="Index",
        Text="Каталог"},
        new MenuItem{ IsPage=true, Area="Admin", Page="/Index",
        Text="Администрирование"}
};
```

Добавьте в класс компонента метод

```
        public IViewComponentResult Invoke()
```

Внутри метода нужно обойти коллекцию _menuItems и, если имя контроллера или имя области совпадает с текущим, то в данном пункте меню свойству Active присвоить значение «active». Полученную коллекцию нужно передать представлению в качестве модели:

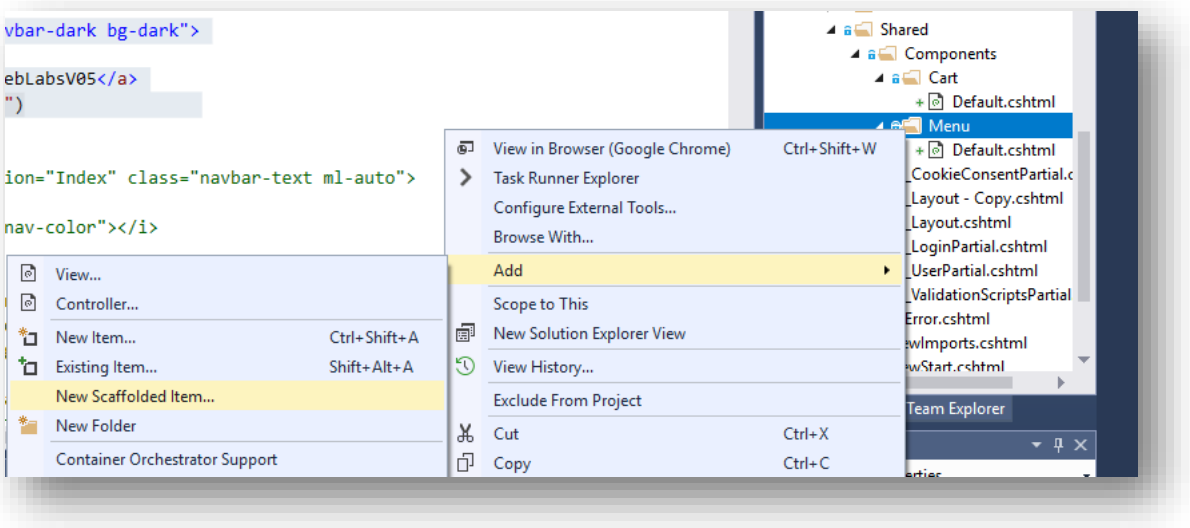
```
public IViewComponentResult Invoke()
{
        //Получение значений сегментов маршрута
        var controller = ViewContext.RouteData.Values["controller"];
        var page = ViewContext.RouteData.Values["page"];
        var area = ViewContext.RouteData.Values["area"];

        foreach(var item in _menuItems)
        {
                // Название контроллера совпадает?
                var _matchController = controller?.Equals(item.Controller)
                ?? false;
```

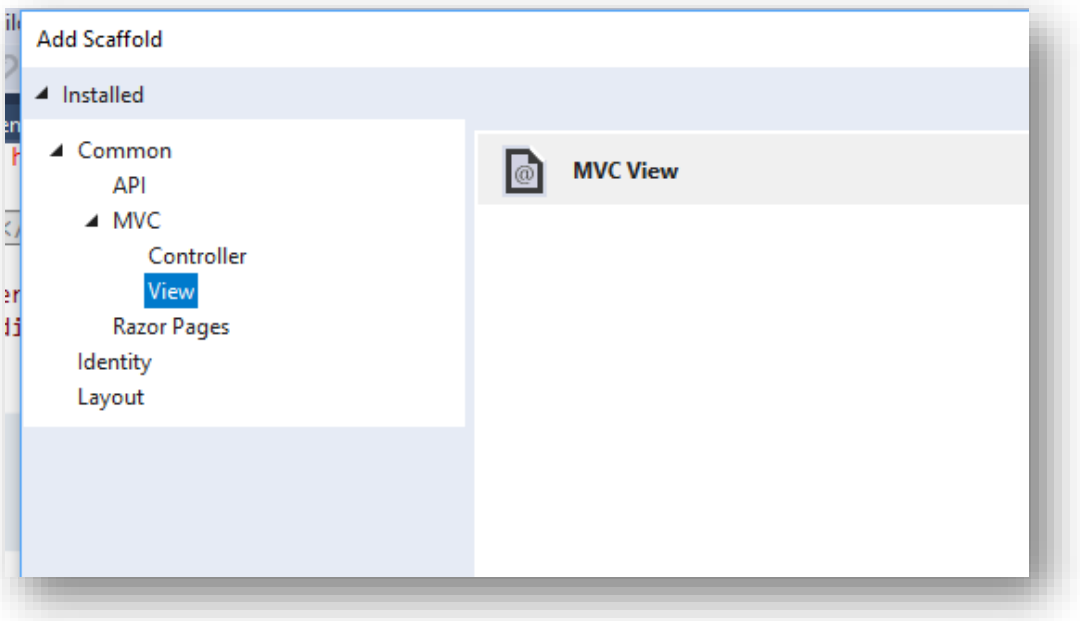
```
            // Название зоны совпадает?
            var _matchArea = area?.Equals(item.Area) ?? false;
            // Если есть совпадение, то сделать элемент меню активным
            // (применить соответствующий класс CSS)
            if(_matchController || _matchArea)
            {
                  item.Active = "active";
            }
      }
      return View(_menuItems);
}
```

5.3. Подготовка представления

В папку ~/Views/Shared добавьте папку Conponents. В полученную папку добавьте папку Menu.

В папку Menu добавьте файл представления Default.cshtml.





В качестве модели укажите `IEnumerable<MenuItem>`

Обойдите модель в цикле foreach и выполните разметку для каждого пункта меню:

```
@model IEnumerable<MenuItem>

<div class="navbar-nav">
    @foreach (var item in Model)
    {
        @if (item.IsPage)
        {
            <a class="nav-item nav-link @item.Active"
               asp-area="@item.Area"
               asp-page="@item.Page">
                @item.Text
            </a>
        }
        else
        {
            <a class="nav-item nav-link @item.Active"
               asp-controller="@item.Controller"
               asp-action="@item.Action">
                @item.Text
            </a>
        }
    }
</div>
```

5.4. Подготовка шаблона

Сделайте резервную копию файла _Layout.cshtml.

В файле _Layout.cshtml:

Замените разметку главного меню на:

```
<!-- меню сайта -->
<a class="navbar-brand" asp-action="Index" asp-
controller="Home">WebLabsV06</a>
<div class="navbar-nav">
      @await Component.InvokeAsync("Menu")
</div>
<!-- меню сайта - конец -->
```

Проверьте результат

5.5. Информация пользователя

В папку ~/Views/Shared добавьте частичное представление _UserPartial.cshtml

Поместите в него разметку (информацию пользователя) из страницы макета.

На странице макета уберите разметку информации пользователя и замените ее на:

```
<!-- Информация пользователя -->
<partial name="_UserPartial" />
<!-- Информация пользователя - конец -->
```

Проверьте результат.

5.6. Компонент корзины заказа

В папку Components проекта добавьте класс CartViewComponent.

Добавьте в класс метод Invoke.

На данном этапе компонент просто возвращает представление без предварительной обработки данных:

```
public class CartViewComponent:ViewComponent
{
      public IViewComponentResult Invoke()
      {
            return View();
      }
}
```

В папку ~/Views/Shared/Components добавьте папку Cart. В полученную папку добавьте частичное представление Default.cshtml.

В полученное представление скопируйте разметку корзины из представления _UserPartial.cshtml.

```
<a asp-controller="Cart" asp-action="Index"
           class="navbar-text ml-auto">
    00,0 руб.(0)
    <i class="fa fa-shopping-cart nav-color"></i>
</a>
```

В представлении _UserPartial вместо разметки корзины укажите:

```
@await Component.InvokeAsync("Cart")
```

Проверьте результат.

Пример окончательной разметки заголовка страницы:

```
<header>
        <div class="container">
            <!-- Панель навигации -->
            <nav class="navbar navbar-expand-md navbar-dark bg-dark">
                <!-- меню сайта -->
                <a class="navbar-brand" asp-action="Index" asp-
controller="Home">WebLabsV06</a>
                <div class="navbar-nav">
                    @await Component.InvokeAsync("Menu")
                </div>
                <!-- меню сайта - конец -->
                <!-- Информация пользователя -->
                <partial name="_UserPartial"/>
                <!-- Информация пользователя - конец -->
            </nav> <!-- Панель навигации - конец -->
        </div><!-- container - конец -->
</header><!-- header - конец -->
```