Если мы создаем компонент с помощью наследования от класса ViewComponent, то нам становится доступным контекст компонента через ряд свойств:

* **HttpContext**: представляет контекст, который описывает полученный запрос, а также отправляемый ответ
* **ModelState**: представляет состояние модели в виде объекта ModelStateDictionary
* **Request**: возвращает контекст запроса в виде объекта HttpRequest
* **RouteData**: возвращает данные маршрута
* **Url**: представляет объект IUrlHelper, который используется для генерации адресов URL
* **User**: представляет текущего пользователя в виде объкта IPrincipal
* **ViewBag**: представляет динамический объект, который может использоваться для передачи данных в представление
* **ViewContext**: описывает контекст главного представления, в котором вызывается компонент
* **ViewComponentContext**: представляет объект ViewComponentContext, который инкапсулирует контекст компонента
* **ViewData**: возвращает объект ViewDataDictionary, который применяется для передачи данных в представление

Используем некоторые и этих свойств. Пусть код компонента будет выглядеть следующим образом:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39 | using Microsoft.AspNetCore.Mvc;  using System;  using System.Collections.Generic;  using System.Linq;    namespace ViewComponentsApp.Components  {      public class PhonesList : ViewComponent      {          Dictionary<string, int> phones;          public PhonesList()          {              phones = new Dictionary<string, int>              {                  {"iPhone 7", 56000 },                  {"Alcatel Idol S4", 26000 },                  {"Samsung Galaxy S7", 50000 },                  {"HP Elite x3", 56000 },                  {"Xiaomi Mi5S", 22000 },                  {"Meizu Pro 6", 22000 },                  {"Huawei Honor 8", 23000 },                  {"Google Pixel", 30000 }              };          }          public IViewComponentResult Invoke()          {              int maxPrice = phones.Max(x => x.Value);                // если передан параметр id              if(RouteData.Values.ContainsKey("id"))                  Int32.TryParse(RouteData.Values["id"]?.ToString(), out maxPrice);                ViewBag.Phones = phones.Where(p => p.Value <= maxPrice).ToList();              ViewData["Header"] = $"Список смартфонов с ценой от {maxPrice.ToString("c")} и меньше";                return View();          }      }  } |

В методе Invoke фильтруем значения в словаре phones по цене. При этом цену получаем из параметров маршрута. Для этого берем параметр "id", который определен в маршруте по умолчанию. А с помощью объектов ViewBag и ViewData передаем данные в представление.

В представлении компонента получим все переданные данные:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | <h3>@ViewData["Header"]</h3>  <ul>      @foreach (var phone in ViewBag.Phones)      {          <li>Модель: <b>@phone.Key</b>  Цена:@phone.Value</li>      }  </ul> |

Вызовем компонент в главном представлении:

|  |  |
| --- | --- |
| 1 | @await Component.InvokeAsync("PhonesList") |

И после этого через параметр id мы сможем передавать цену для фильтрации: