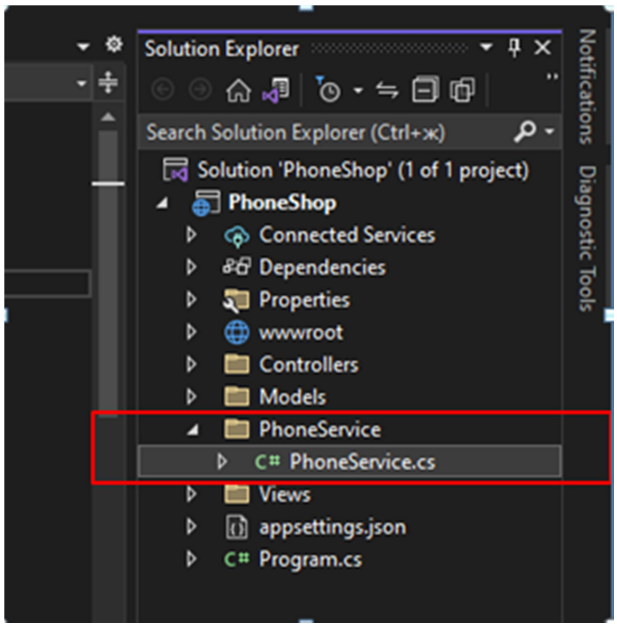
**Что значит папка PhoneService и внутри нее класс PhoneService?**




В папке Models должны храниться наши модели (модель может быть и классом, как это показано в коде). Это значит, если вы продаете технику, там будет храниться класс phone, класс computer, класс headphones, и тд. Создав класс (модель) под каждый продукт, мы хотели чтобы дальше эти модели были защищены от неявных\случайных изменений, которые могут произойти в процессе написания или изменения кода. Так вот. Папка **PhoneService и внутри нее класс PhoneService** это бизнес-логика, то есть там лежат все операции, которые мы будем производить с нашими моделями (классы phone, computer, headphones) – мы будем их показывать клиенту, удалять из базы данных при их продаже, добавлять как только на складе появляются новые.

**Почему не сделать все сразу в одном классе моделей, почему бы не написать логику добавления нового телефона в классе phone?** Представим, что все именно так и есть. Значит один и тот же код (добавление в базу данных, удаление из нее и выведение товаров на экран клиента) будет дублироваться в каждом классе, в нашем случае минимум трижды (классы phone, computer, headphones). Это не соответствует правилам code convention и в частности, SOLID. Но мы закроем на это глаза и продолжим. Всегда есть риск нечаянно изменить класс, например не заметить и стереть свойство Id.. можно накасячить если классы наследуются друг от друга или еще легче пропустить если класс наследуется от абстрактного класса (где в последнем пишутся поля, а в классах-наследниках вы просто должны догадаться, что там они уже есть). Даже если вам удастся преодолеть и этот порог, помните, что ваш клиент может в последний момент просто передумать, взять и попросить вас вернуть все, как было. А теперь представим, что вышеописанное было сделано не вами, а предыдущим сотрудником, а вам предстоит с этим хорошенько разобраться.

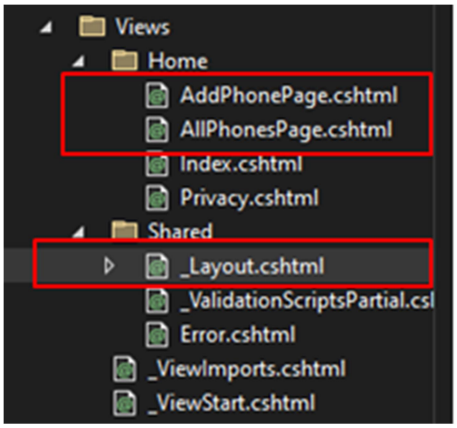
Бизнес-логика (будет вечно меняться) и asp.net должны знать ничего  друг о друге. Так мы храним яйца в разных корзинках: модели, на которых базируется бизнес (пусть это будут не телефоны, а, например врачи, и у вас больница) и бизнес-логику (врачам надо выплатить зарплаты, отпустить их в отпуск, наладить доступ в лаборатории, выдать униформу, посчитать их рабочие часы, умножить на то, сколько они потратили и сколько заработали и тп). Единственный, кто знает и о том и о другом – это контроллер, он тот самое соединительное звено. Чтобы контроллер знал о нашей бизнес-логике, надо  чтобы папка и класс **PhoneService были прописаны в конструкторе контроллера HomeController(подробнее см котроллер):**

```
public HomeController(ILogger<HomeController> logger, PhoneService service)
        {
            _logger = logger;
            _service= service;
        }
```

**Теперь подробнее о самом коде в классе PhoneService что там и почему оно именно там:**

```
espace PhoneShop.Service

3 references
public class PhoneService // здесь лежит вся бизнес логика, модели в папке models должны оставаться нетронутыми,
                          // а здесь мы пишем все что с моделями делать хотим
{
    private readonly string pathToFile = @".\wwwroot\db\phones.json"; // адрес где лежит наш файл в формате json, в будущем это база данных
    List<Phone> phones= new List<Phone>(); // лист куда мы будем записывать, удалять или показывать наши телефоны

    1 reference
    public void AddPhoneToList(Phone phone)
    {
        DeserializeList();// выгрузим из базы данных свеженький лист со всеми телефонами
        phones.Add(phone); //добавим в этот свеженький лист новый телефон
        SerializeList(); // загрузим обновленный лист обратно в базу данных
    }

    1 reference
    public List<Phone> ShowPhones()
    {
        DeserializeList(); // выгрузим из базы данных свеженький лист со всеми телефонами
        return phones; // вернем этот лист тому методу, кто его попросит дальше
    }

    1 reference
    private void SerializeList() // json язык сервера, этот метод позволяет загрузить лист(базу данных) с телефонами с сервера
    {
        string json = JsonConvert.SerializeObject(phones);
        File.WriteAllText(pathToFile, json);
    }

    2 references
    private void DeserializeList() // а этот метод позволяет загрузить лист(базу данных) с телефонами обратно на сервер
    {
        string json = ReadFileToString();
        phones = JsonConvert.DeserializeObject<List<Phone>>(json) ?? new List<Phone>();
    }

    1 reference
    private string ReadFileToString() // этот метод превращает данные с сервера из формата json (которые c# не может считать корретно)
                                      // в формат строки (а вот это c# может)
    {
        if (!File.Exists(pathToFile)) // если такой файл НЕ существует (например, путь "../document/repos/database.json"), погугли File.Exists
        {
            File.WriteAllText(pathToFile, ""); //создание нового файла, запись указанной строки в файл,
                                               //а затем закрытие файла(путь pathToFile, содержимое "") гугли  File.WriteAllText
        }

        return File.ReadAllText(pathToFile); // если он существует - прочти весь текст в искомом файле и верни его в строковом формате
                                             // тому методу, кто его попросит дальше
    }
```

**Папка Views что там происходит?**




CSHTML файл _Layout - хранит код странички, это буквально скелет главной странички нашего вэб-приложения. Формат CSHTML читается как CS = c-sharp и HTML, то есть внутри блоков html мы можем вписать объекты нашего c-sharp кода, например классы, листы или даже совершать операции например c = a++ и пр.

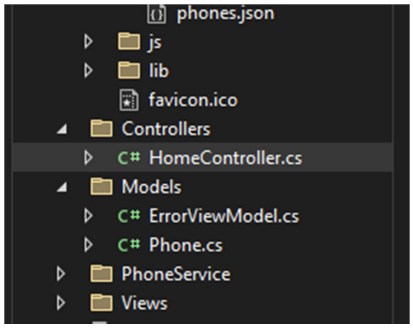
Папка Home обозначает нашу главную и пока единственную страничку приложения. Внутри нее есть две формы – AddPhonesPage и AllPhonesPage (приставка Page чтобы не путать эти названия с названиями методов, например вспомним что у нас есть метод AddPhones). Первая (AddPhonesPage) – это форма для добавления телефона в базу данных, а вторая для отображения добавленных телефонов в базе данных. Мы в главном лейауте прописываем что на вкладке AddPhones будет отображена форма, которая находится в файле AddPhonesPage и аналогично с вкладкой AllPhones и формой AllPhonesPage. Вот где именно прописаны эти формы в главном лейауте(файл _Layout в папке Shared ):

```
<div class="navbar-collapse collapse d-sm-inline-flex justify-content-between">
    <ul class="navbar-nav flex-grow-1">
        <li class="nav-item">
            <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Index">Home</a>
        </li>
        <li class="nav-item">
            <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="AddPhonePage">Add Phone</a>
        </li>
        <li class="nav-item">
            <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="AllPhonesPage">All Phones</a>
        </li>
    </ul>
</div>
```

Название контроллера – Home(полное название HomeController), дальше экшен – название формы для добавления телефона AddPhonesPage и название самой вкладки обозначаем на свой вкус, у нас это просто AddPhone. Вторая строка аналогично: название контроллера, название формы html, и как будет называться эта вкладка. Для того, чтобы стилизовать формы, например красивыми кнопками или ровными таблицами – используем по умолчанию подключенную библиотеку https://getbootstrap.com/

Например здесь все по кнопкам https://getbootstrap.com/docs/5.3/components/buttons/#examples

Поисковик на сайте библиотеки удобно использовать, чтобы искать быстрее. В ютубе полно видео, рассказывающих про эту библиотеку. Освежить знания о стилях css и, скелетах html и библиотеках можно, например, вот тут https://www.youtube.com/watch?v=W4MIiV4nZDY&t=9307s




**Что происходит в контроллере и как мы можем его изменить в лучшую сторону?**

Первые две строчки это подключение логгирования, а вторая строчка это подключение бизнес-логики **PhoneService** (о ней подробно выше).

Так же ниже конструктор контроллера, который подключает логгер и

```
private readonly ILogger<HomeController> _logger;
private PhoneService _service; // в папке PhoneService лежит класс отражающий наш

0 references
public HomeController(ILogger<HomeController> logger, PhoneService service) // co
{
    _logger = logger;
    _service= service;
}
```

бизнес-логику к котроллеру (смотри рисунок слева). Вспомним, что бизнес логика и ваш asp.net не должны знать друг о друге, связующим звеном между ними является контроллер, который знает об обоих. Дальше комментарии по каждому блоку кода контроллера смотри в коде самого контроллера.  Что можно улучшить? Если наш магазин начнет продавать не только телефоны, но и телевизоры, игровые приставки, наушники и компьютеры, и все методы для каждого из этих классов/моделей мы будем писать в HomeController, то его длина и низкая удобочитаемость  значительно усложнит работу по развитию проекта. Чтобы магазин мог расти и масштабироваться, под каждую модель, думаю можно запилить отдельный контроллер (тогда не забудем еще в html layout поменять название контроллера). Так, вместо одного контроллера с 25-ю методами внутри у нас появится 5 контроллеров под каждую модель внутри которых будет всего по 5 методов. Работать с такой архитектурой станет значительно легче.