# C# Basic Web: Phonebook

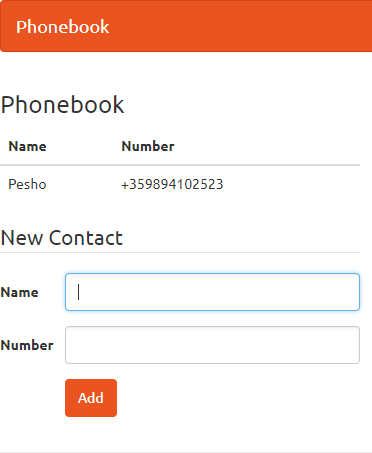
Problems for exercise for the ["Technology Fundamentals with C#" course @ SoftUni](https://softuni.bg/modules/57/tech-module-4-0)

## Problem

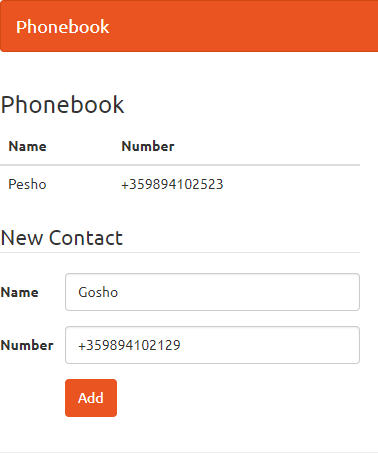
You have been tasked to create a simple **Phonebook** application. The application should hold **contacts** inside a **memory storage**.

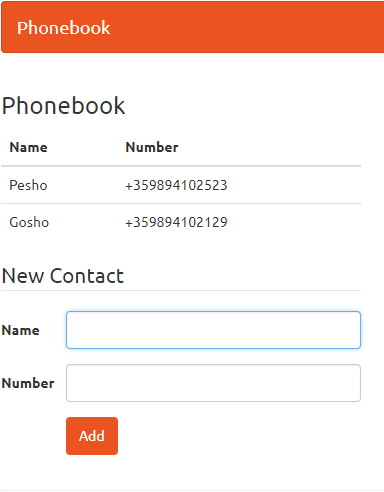
The functionality of the application should support:

* **Listing contacts**



* **Add Contact**





## Overview

### Data Model

The Contact entity holds **2 properties**:

* name – non-empty text
* number – non-empty text

### Project Skeletons

You will be given the applications' skeletons, which hold about **90%** of the logic. You’ll also be given **files**. The files have **partially implemented logic** and you need to write some code for the application to **function properly**.

The application’s views will be given to you fully implemented. You only need to include them in your business logic.

Everything that has been given to you inside the skeleton is **correctly implemented** and if you write your code **correctly**, the application should work just fine. You are free to change anything in the Skeleton on your account.

## Preparation

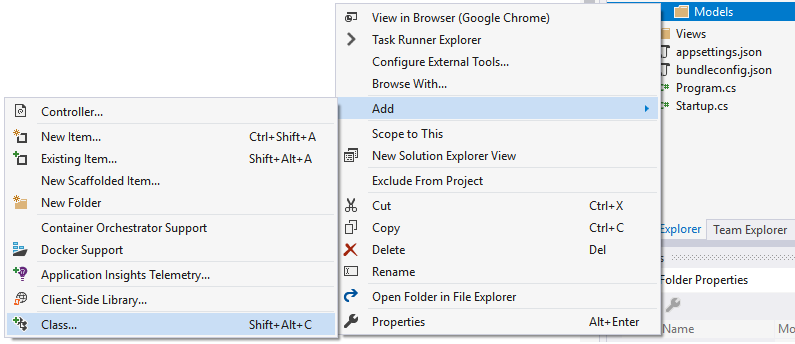
Extract the skeleton and open the **Phonebook.sln** file.

## Contact Model

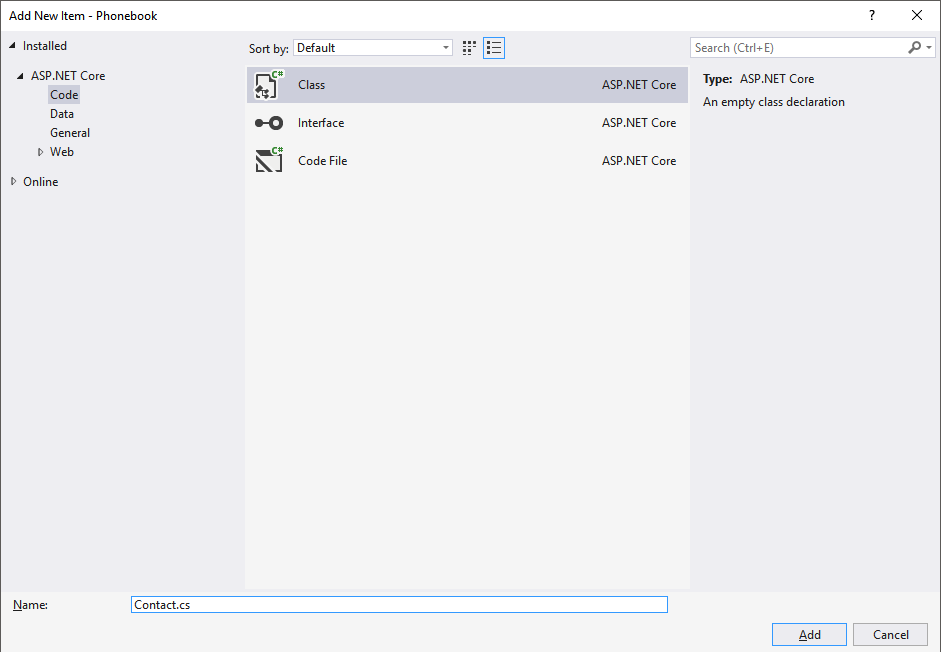
Now it’s time to create our **contact entity class**. Our **contact** will be simple. It will have **2 properties**:

* **Name** – the **name** of the contact, stored as a string.
* **Number** – the **number** of the contact, stored as a string.

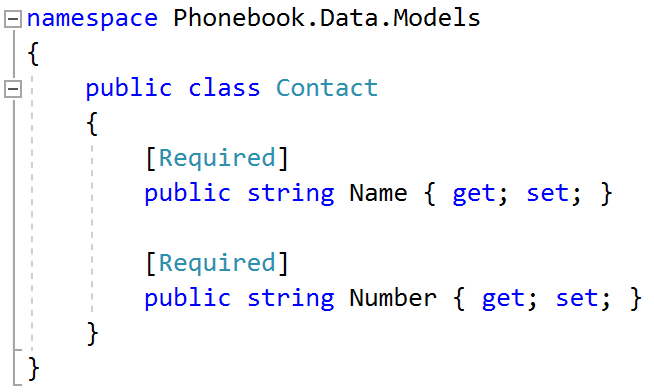
Let’s go in our **Data/**Modelsfolder and **add a Contact class**:



In the menu, which popped up, select Classand name it Contact.cs:



All that’s left is to **add the properties** into our new file:

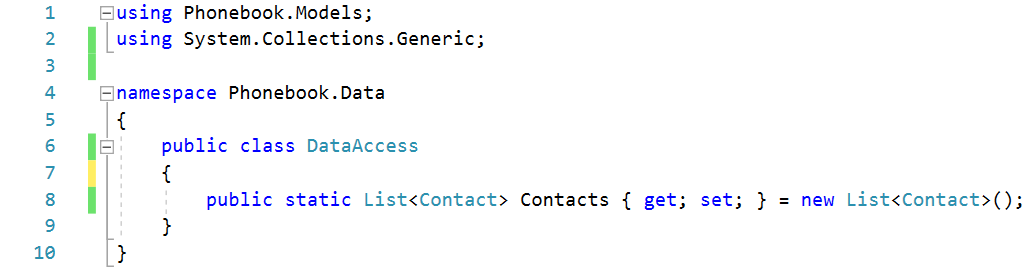


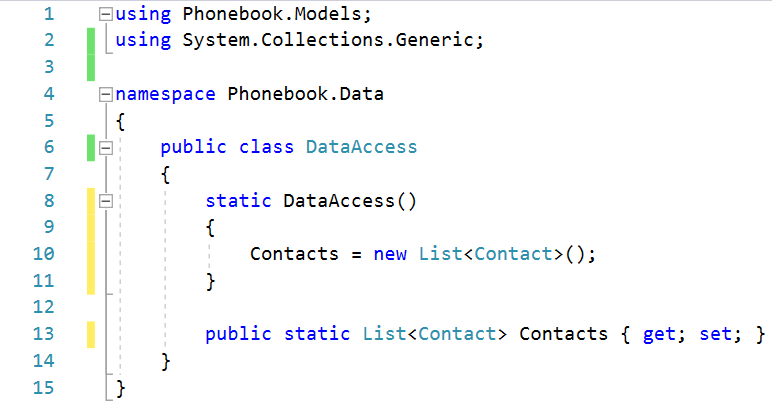
We’re using the [Required] **attribute** on our **Name and Number properties**, because we don’t want to have **contacts without a name or a number**.

## Create Data Context

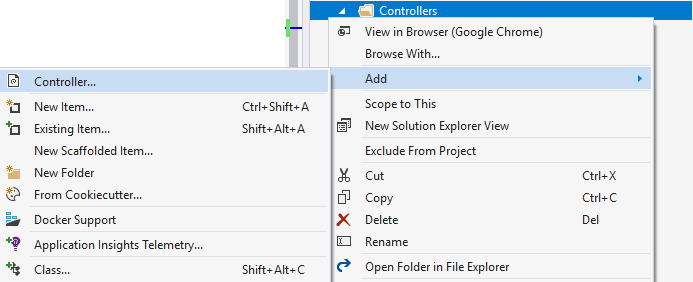
Now it’s time to create the **data context**, which **keeps all the data** about the contacts. Right click on the Data folder and create a new **Class.** Name it **DataAccess.**

Now, we have to write the list, which will keep all the data about all the contacts. This can be done in **one of two ways**:

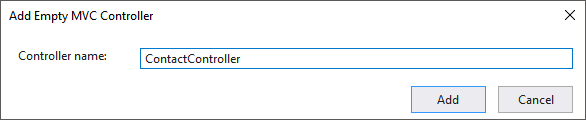




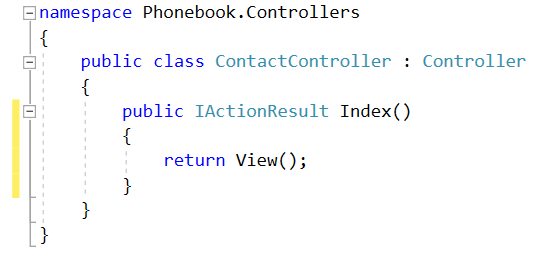
## Create Contact Controller

Now it’s time to create the controller, which **adds** **contacts**. Right-click the Controllers folder and click on Add 🡺 Controller:  


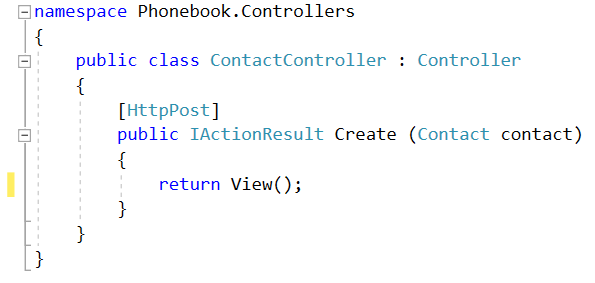
In the popup, select “MVC Controller – Empty”, then name it ContactController:



If we look at our newly-created controller, it looks like this:



We don’t need the Index() action, so just **remove it**.

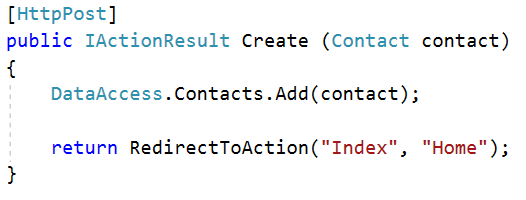
Now, it’s time to write the logic for both of the actions. 

### Write Logic for Adding Contacts

Let’s make the action for **creating** contacts. This action will have a Contactas a parameter, letting ASP.NET automatically fill in the properties of the contact, before inserting it into the memory storage:

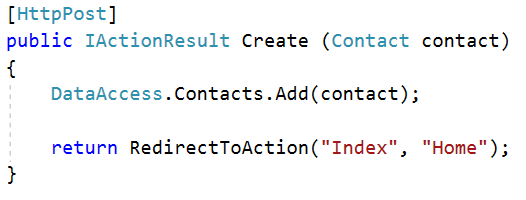
We’re using the [HttpPost] attribute, because we’re **sending** data to the server, not retrieving it.

Let’s **add the task to the memory storage**:



What we are doing here is calling the **DataAccess** class and adding the new **contact** to the **Contacts** list. This way every time someone inserts **contact information** it will go in the list.

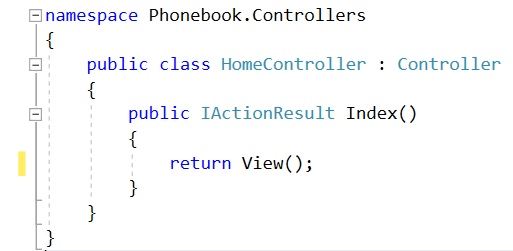
At last, we need to **redirect the user** to the Index() action in the HomeController:



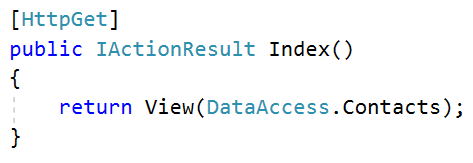
Now if our user wants to **add a contact**, all they have to do is send a **POST request** to "/Create" with their **contact name and number**. Alternatively, they could just use the **HTML form** we’ll create in a few minutes.

### Write Logic for Listing Contacts

Let’s go into the Controllers/HomeController.cs file:



Not much going on here… Let’s retrieve all the **contacts** and give them to the **index** view:



We retrieve all the contacts, which are in the **RAM** and we can render them in the view.

With that we finished our **C# Phonebook**. Feel free to **build on your project even further**. ☺

For example, you can add **validation** to the contact name and number using **Regular Expressions** (which you have learned about in the previous lecture) so that you know that the information given is always valid.