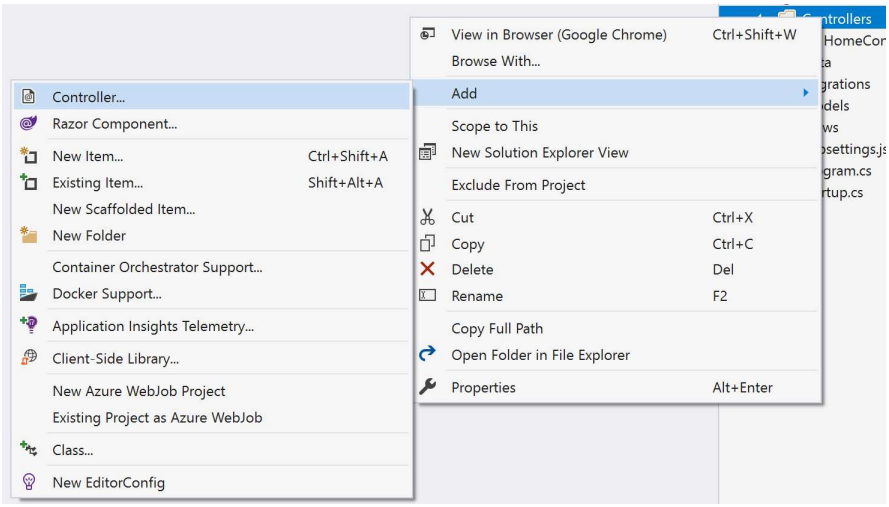
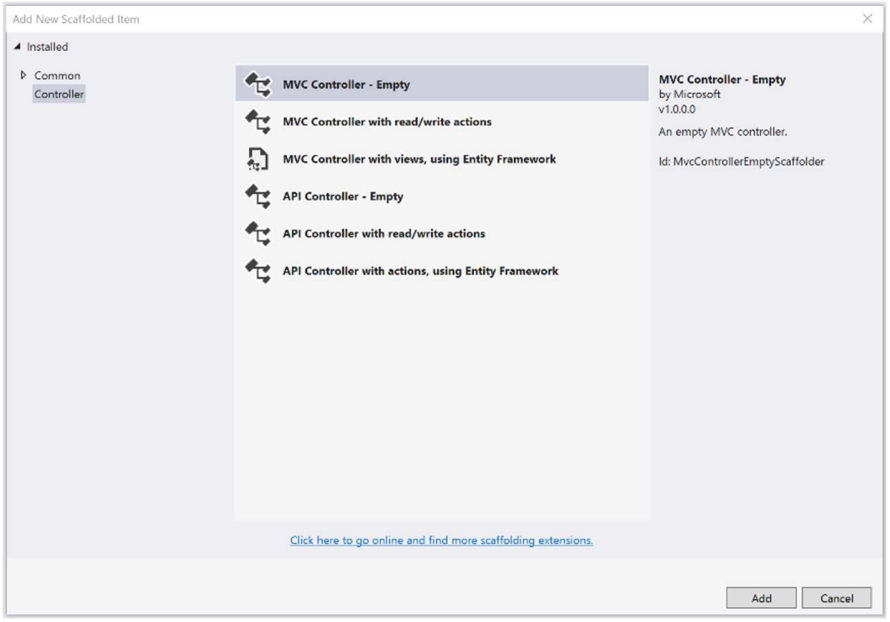
Creating a Controller using Visual Studio

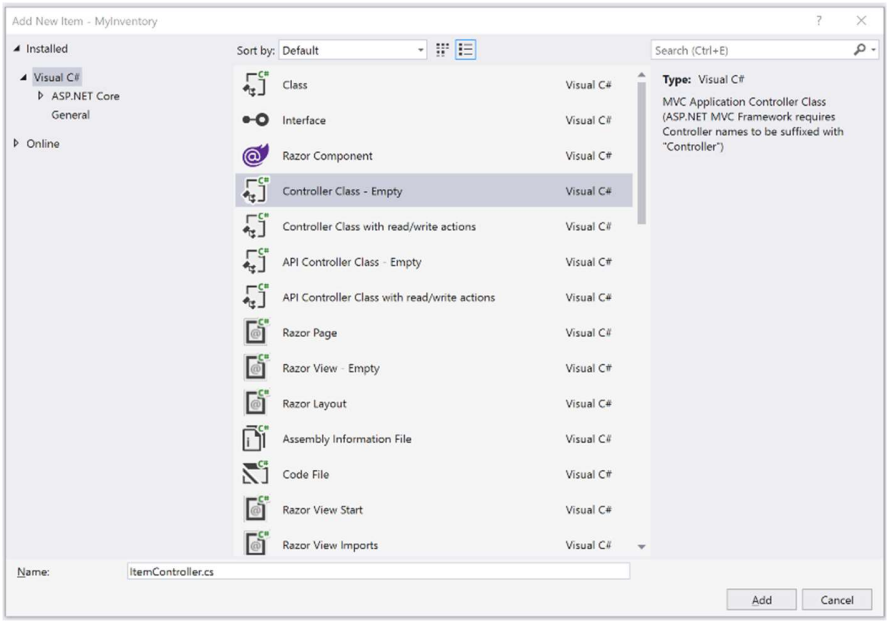
1. From the existing solution, right-click **Controllers > Add > Controller**.



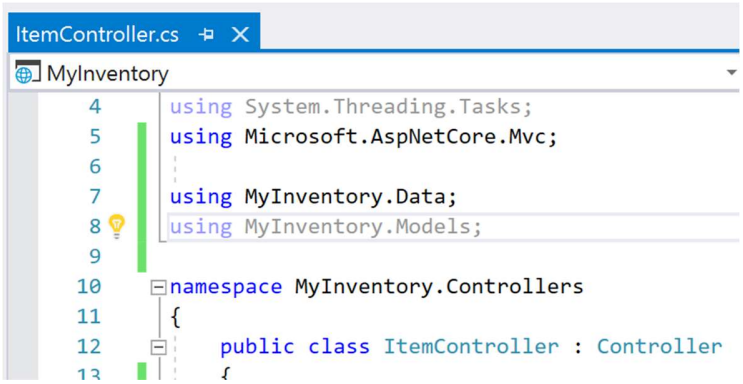
2. From the **Add New Scaffold Item** window, select **MVC Controller – Empty**, then click the **Add** button.



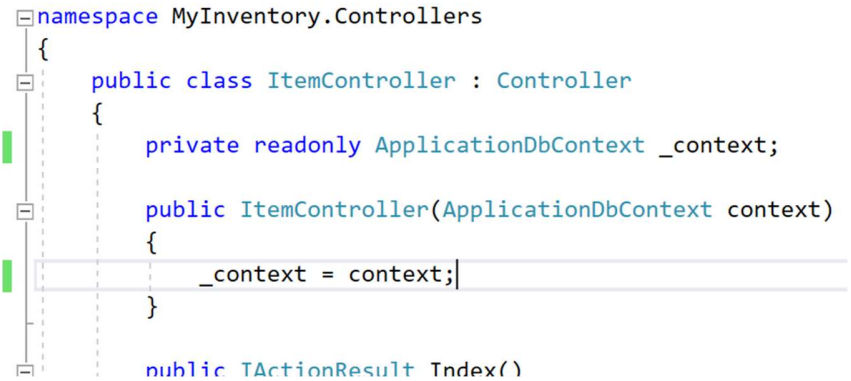
3. Name the controller file as **ItemController.cs.**



4. Inside **ItemController.cs**, add the namespace **MyInventory.Data** and **MyInventory.Models**.



5. Inside the constructor, include the **DbContext** class object using **Dependency Injection**. Add the **ApplicationDbContext** class object and set a public property value to it.



6. Inside the **Index** method, declare a variable **list** that displays records from the **Items** table. The model will be included to be rendered by the **View** method.

```
public IActionResult Index()
{
    var list = _context.Items.ToList();
    return View(list);
}
```

7. Create a **Create** method returning to view.

```
public IActionResult Create()
{
    return View();
}
```

8. Override the existing **Create** method by replicating another **Create** method using the **Item** object as the parameter.

```
public IActionResult Create()
{
    return View();
}

public IActionResult Create(Item record)
{

}
```

9. Include an action verb HttpPost on top of the second **Create** method.

```
[HttpPost]
public IActionResult Create(Item record)
{

}
```

10. Inside the **Create** method, initialize a variable **item** from the **Item** class.

```
[HttpPost]
public IActionResult Create(Item record)
{
    var item = new Item();
}
```

11. Identify the list of properties and set their value from the **record** parameter.

```
[HttpPost]
public IActionResult Create(Item record)
{
    var item = new Item();
    item.Name = record.Name;
    item.Code = record.Code;
    item.Description = record.Description;
    item.Price = record.Price;
    item.DateAdded = DateTime.Now;
    item.Type = record.Type;
}
```

Or

```
[HttpPost]
public IActionResult Create(Item record)
{
    var item = new Item()
    {
        Name = record.Name,
        Code = record.Code,
        Description = record.Description,
        Price = record.Price,
        DateAdded = DateTime.Now,
        Type = record.Type
    };
}
```

12. Add the existing values, then save the record from the database table.

```
[HttpPost]
public IActionResult Create(Item record)
{
    var item = new Item()
    {
        Name = record.Name,
        Code = record.Code,
        Description = record.Description,
        Price = record.Price,
        DateAdded = DateTime.Now,
        Type = record.Type
    };

    _context.Items.Add(item);
    _context.SaveChanges();
}
```

13. Return the view redirecting back to the **Index** action.

```
[HttpPost]
public IActionResult Create(Item record)
{
    var item = new Item()
    {
        Name = record.Name,
        Code = record.Code,
        Description = record.Description,
        Price = record.Price,
        DateAdded = DateTime.Now,
        Type = record.Type
    };

    _context.Items.Add(item);
    _context.SaveChanges();

    return RedirectToAction("Index");
}
```

14. Create an **Edit** method using a nullable integer **id** as the parameter.

```
public IActionResult Edit(int? id)
{

}
```

15. Inside the **Edit** method, include a selection statement that checks if a valid value is not present. If the condition is valid, the view will redirect to the **Index** action.

```
public IActionResult Edit(int? id)
{
    if (id == null)
    {
        return RedirectToAction("Index");
    }
}
```

16. Declare a variable **item** that gets the existing record from the **Items** table.

```
public IActionResult Edit(int? id)
{
    if (id == null)
    {
        return RedirectToAction("Index");
    }

    var item = _context.Items.Where(i => i.ItemId == id).SingleOrDefault();
}
```

17. Include a selection statement that checks if the **item** record is not present. If the condition is valid, the view will redirect to the **Index** action.

```
public IActionResult Edit(int? id)
{
    if (id == null)
    {
        return RedirectToAction("Index");
    }

    var item = _context.Items.Where(i => i.ItemId == id).SingleOrDefault();
    if (item == null)
    {
        return RedirectToAction("Index");
    }
}
```

18. The **item** model object will be included to be rendered by the **View** method.

```
public IActionResult Edit(int? id)
{
    if (id == null)
    {
        return RedirectToAction("Index");
    }

    var item = _context.Items.Where(i => i.ItemId == id).SingleOrDefault();
    if (item == null)
    {
        return RedirectToAction("Index");
    }

    return View(item);
}
```

19. Override the existing **Edit** method by replicating another **Edit** method using a nullable integer **id** and the **Item** object as the parameters.

```
public IActionResult Edit(int? id, Item record)
{

}
```

20. Include an action verb **HttpPost** on top of the second **Edit** method.

```
[HttpPost]
public IActionResult Edit(int? id, Item record)
{

}
```

21. Declare a variable item that gets the existing record from the Items table based on the parameter id value.

```
[HttpPost]
public IActionResult Edit(int? id, Item record)
{
    var item = _context.Items.Where(i => i.ItemId == id).SingleOrDefault();
}
```

22. Identify the list of properties and set their value from the **record** parameter.

```
[HttpPost]
public IActionResult Edit(int? id, Item record)
{
    var item = _context.Items.Where(i => i.ItemId == id).SingleOrDefault();
    item.Name = record.Name;
    item.Code = record.Code;
    item.Description = record.Description;
    item.Price = record.Price;
    item.DateModified = DateTime.Now;
    item.Type = record.Type;
}
```

23. Update the existing values, then save the record from the database table.

```
[HttpPost]
public IActionResult Edit(int? id, Item record)
{
    var item = _context.Items.Where(i => i.ItemId == id).SingleOrDefault();
    item.Name = record.Name;
    item.Code = record.Code;
    item.Description = record.Description;
    item.Price = record.Price;
    item.DateModified = DateTime.Now;
    item.Type = record.Type;

    _context.Items.Update(item);
    _context.SaveChanges();
}
```

24. Return the view redirecting back to the **Index** action.

```
[HttpPost]
public IActionResult Edit(int? id, Item record)
{
    var item = _context.Items.Where(i => i.ItemId == id).SingleOrDefault();
    item.Name = record.Name;
    item.Code = record.Code;
    item.Description = record.Description;
    item.Price = record.Price;
    item.DateModified = DateTime.Now;
    item.Type = record.Type;

    _context.Items.Update(item);
    _context.SaveChanges();

    return RedirectToAction("Index");
}
```

25. Create a **Delete** method using a nullable integer **id** as the parameter.

```
public IActionResult Delete(int? id)
{

}
```

26. Inside the **Delete** method, include a selection statement that checks if a valid value is not present. If the condition is valid, the view will redirect to the **Index** action.

```
public IActionResult Delete(int? id)
{
    if (id == null)
    {
        return RedirectToAction("Index");
    }
}
```

27. Declare a variable **item** that gets the existing record from the **Items** table.

```
public IActionResult Delete(int? id)
{
    if (id == null)
    {
        return RedirectToAction("Index");
    }

    var item = _context.Items.Where(i => i.ItemId == id).SingleOrDefault();
}
```

28. Include a selection statement that checks if the **item** record is not present. If the condition is valid, the view will redirect to the **Index** action.

```
public IActionResult Delete(int? id)
{
    if (id == null)
    {
        return RedirectToAction("Index");
    }

    var item = _context.Items.Where(i => i.ItemId == id).SingleOrDefault();
    if (item == null)
    {
        return RedirectToAction("Index");
    }
}
```

29. Remove the existing record from the database table.

```
public IActionResult Delete(int? id)
{
    if (id == null)
    {
        return RedirectToAction("Index");
    }

    var item = _context.Items.Where(i => i.ItemId == id).SingleOrDefault();
    if (item == null)
    {
        return RedirectToAction("Index");
    }

    _context.Items.Remove(item);
    _context.SaveChanges();
}
```

30. Return the view redirecting back to the **Index** action.

```
public IActionResult Delete(int? id)
{
    if (id == null)
    {
        return RedirectToAction("Index");
    }

    var item = _context.Items.Where(i => i.ItemId == id).SingleOrDefault();
    if (item == null)
    {
        return RedirectToAction("Index");
    }

    _context.Items.Remove(item);
    _context.SaveChanges();

    return RedirectToAction("Index");
}
```