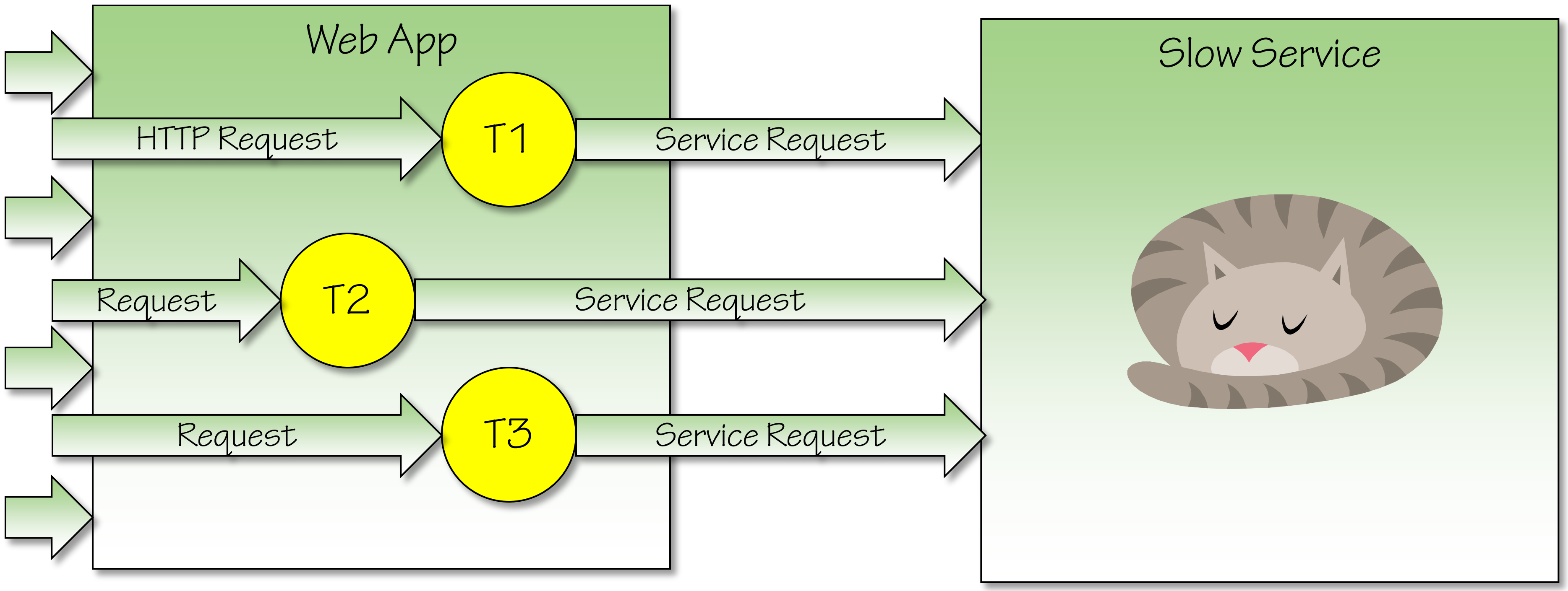
# MVC 4 – Async Actions

Asyncing and Awaiting

Scott Allen

pluralsight
hardcore developer training

# Scalability

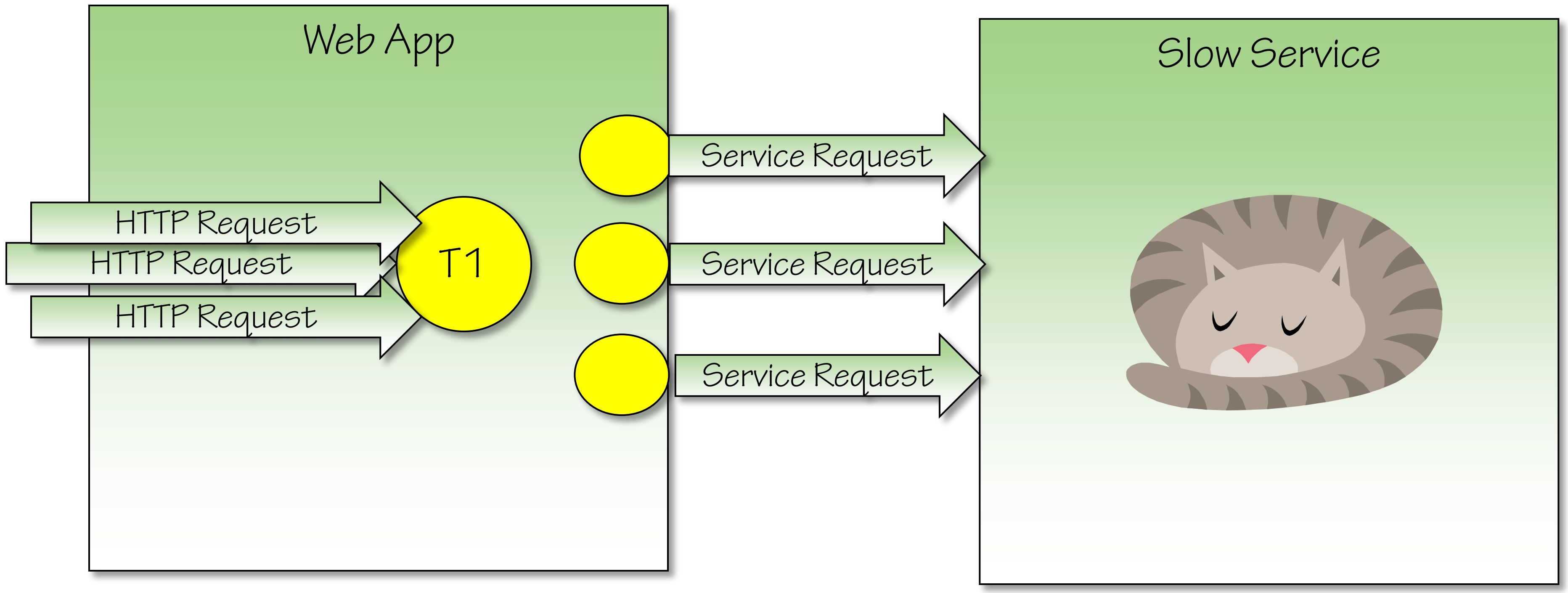


# AsyncController in MVC 3

```
public void IndexAsync()
{
    var model = new HomePageViewModel();
    var newsClient = new NewsServiceClient();
    var weatherClient = new WeatherServiceClient();

    AsyncManager.Parameters["model"] = model;

    AsyncManager.OutstandingOperations.Increment();
    newsClient.BeginGetHeadline(ar =>
    {
        model.Headline = newsClient.EndGetHeadline(ar);
        AsyncManager.OutstandingOperations.Decrement();
    }, null);

    AsyncManager.OutstandingOperations.Increment();
    weatherClient.BeginGetCurrentTemperature(ar =>
    {
        model.Temperature = weatherClient.EndGetCurrentTemperature(ar);
        AsyncManager.OutstandingOperations.Decrement();
    }, null);

}
```

```
public ViewResult IndexCompleted(HomePageViewModel model)
{
    return View(model);
}
```

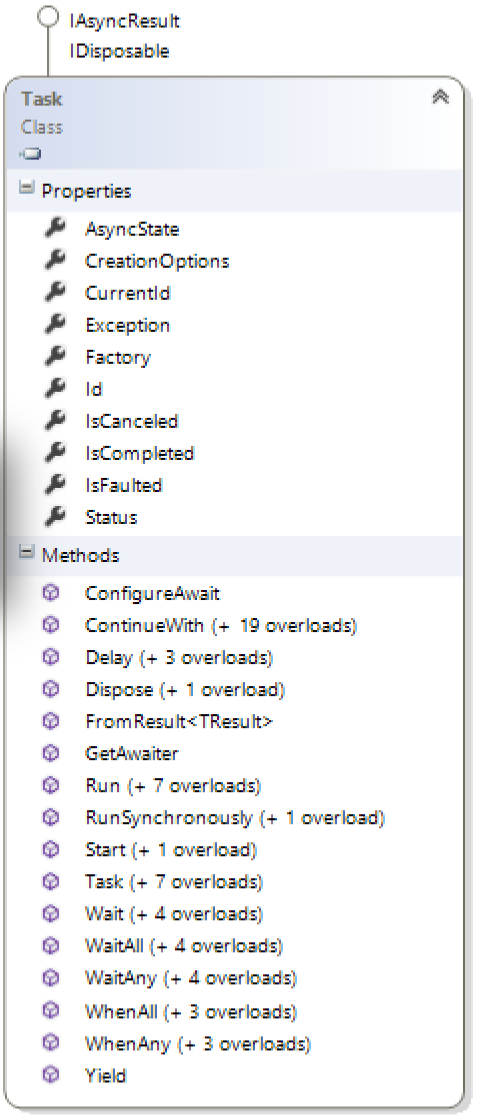
# Scalability



# The Task Parallel Library

- Parallel processing
- Asynchronous processing
- A better abstraction for threads

```
var task = Task.Factory.StartNew<int>(SlowOperation);
// ...
task.Wait();
```



IAsyncResult
IDisposable

Task
Class

Properties

- AsyncState
- CreationOptions
- CurrentId
- Exception
- Factory
- Id
- IsCanceled
- IsCompleted
- IsFaulted
- Status

Methods

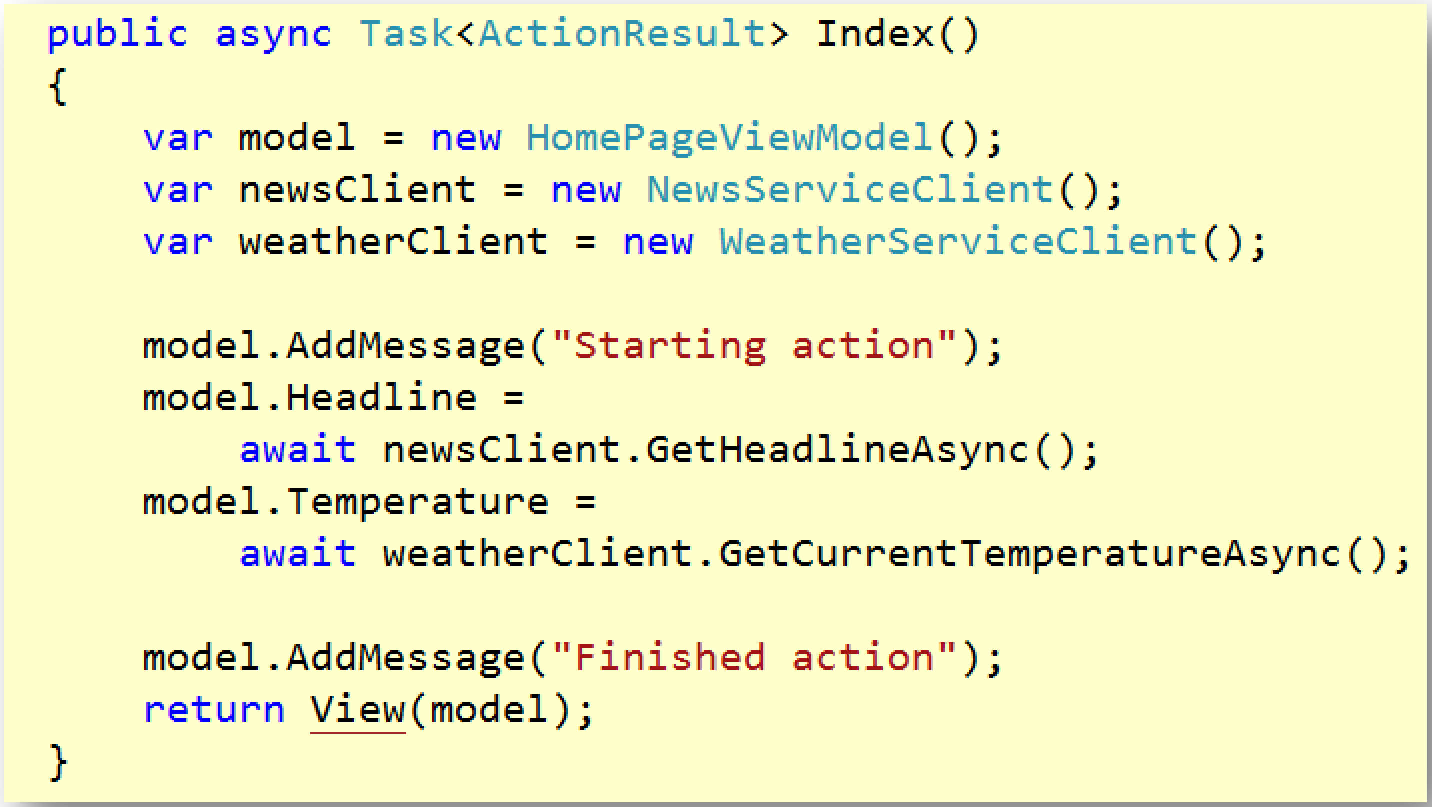
- ConfigureAwait
- ContinueWith (+ 19 overloads)
- Delay (+ 3 overloads)
- Dispose (+ 1 overload)
- FromResult<TResult>
- GetAwaiter
- Run (+ 7 overloads)
- RunSynchronously (+ 1 overload)
- Start (+ 1 overload)
- Task (+ 7 overloads)
- Wait (+ 4 overloads)
- WaitAll (+ 4 overloads)
- WaitAny (+ 4 overloads)
- WhenAll (+ 3 overloads)
- WhenAny (+ 3 overloads)
- Yield

## async & await

- Method with async keyword can use an await
- await can suspend an async method
- await can free the calling thread
- Execution can resume where it left off

```
static async Task<int> SomeWorkAsync()
{
    var result = await ServiceCallAsync();
    return result;
}
```

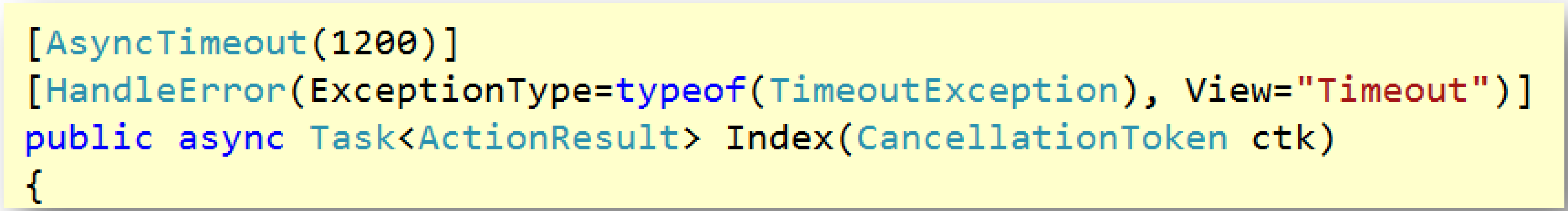
# async actions



```
public async Task<ActionResult> Index()
{
    var model = new HomePageViewModel();
    var newsClient = new NewsServiceClient();
    var weatherClient = new WeatherServiceClient();

    model.AddMessage("Starting action");
    model.Headline =
        await newsClient.GetHeadlineAsync();
    model.Temperature =
        await weatherClient.GetCurrentTemperatureAsync();

    model.AddMessage("Finished action");
    return View(model);
}
```

## Timeouts

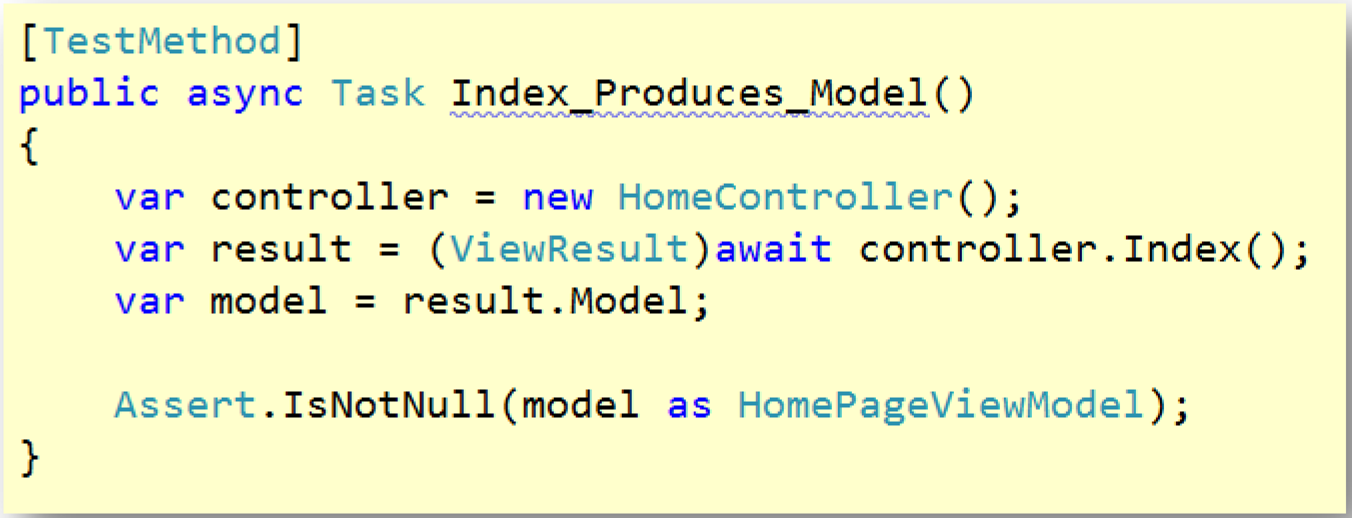
- **Use AsyncTimeout attribute**
- **Requires CancellationToken parameter**
    - Pass token to other async operations



```
[AsyncTimeout(1200)]
[HandleError(ExceptionType=typeof(TimeoutException), View="Timeout")]
public async Task<ActionResult> Index(CancellationToken ctk)
{
```

# async Testing

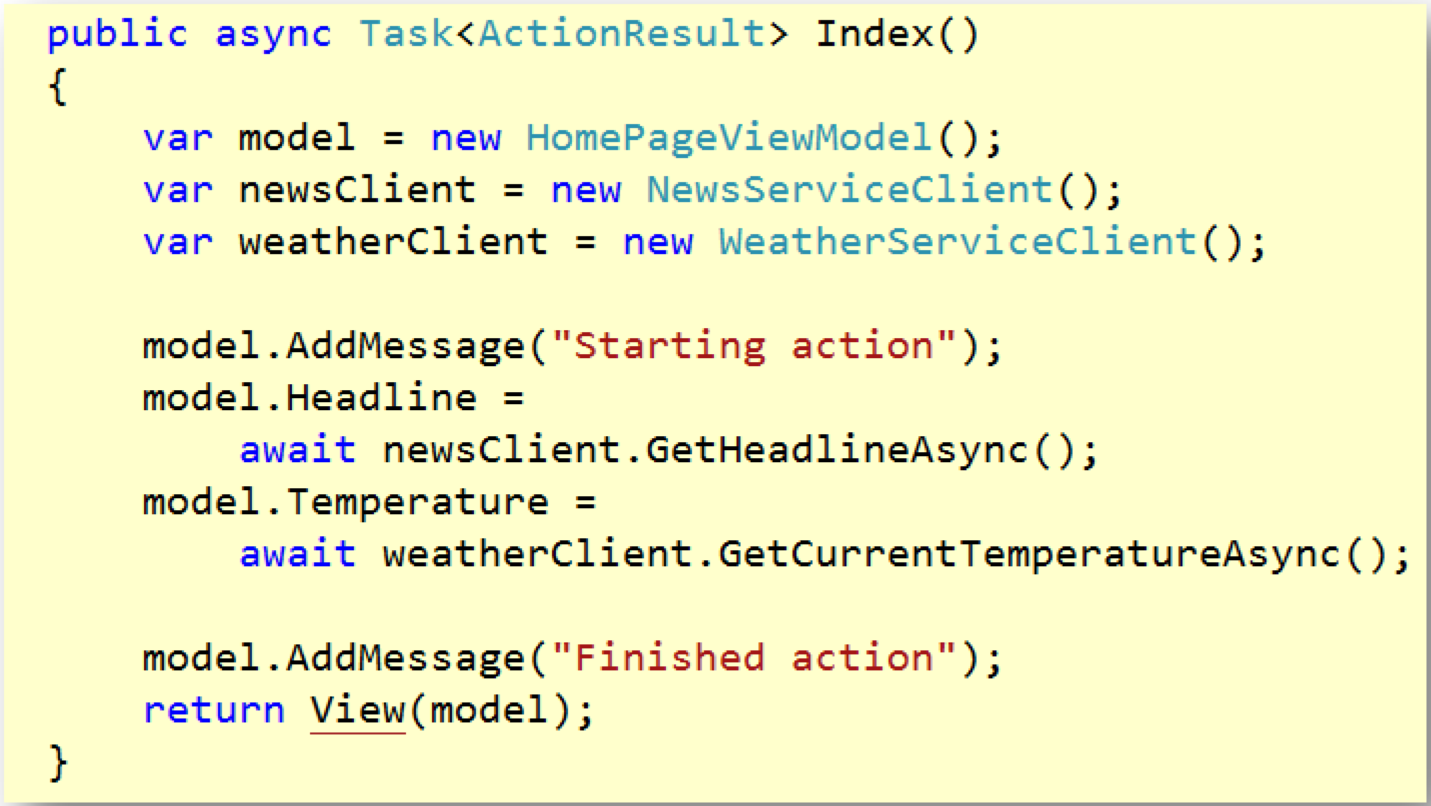
- **Use a test runner that supports async test methods**
  - MSTest
  - XUnit



```
[TestMethod]
public async Task Index_Produces_Model()
{
    var controller = new HomeController();
    var result = (ViewResult)await controller.Index();
    var model = result.Model;

    Assert.IsNotNull(model as HomePageViewModel);
}
```

# Summary

- async await make for easy asynchrony
- Structure awaits to suit the processing model
- Test async methods with a async capable test runner



```
public async Task<ActionResult> Index()
{
    var model = new HomePageViewModel();
    var newsClient = new NewsServiceClient();
    var weatherClient = new WeatherServiceClient();

    model.AddMessage("Starting action");
    model.Headline =
        await newsClient.GetHeadlineAsync();
    model.Temperature =
        await weatherClient.GetCurrentTemperatureAsync();

    model.AddMessage("Finished action");
    return View(model);
}
```