Lab 4 Testing

Goal

Demonstrate the testability of the solution we've built, and how to test full stack ASP.NET Core MVC apps.

Topics Used

Testing, xUnit, Filters

Requirements

The Guestbook needs to support mobile and/or rich client apps, and thus requires an API. The API needs to support two methods initially:

- ListEntries: Should list the same entries as the current home page
- AddEntry: Should add a new entry (just like the form on the current home page)

Detailed Steps

- Add a new API Controller for `GuestbookController` in Web/Api/GuestbookController.cs
- Add a method to get a Guestbook
    - Return a 404 Not Found if the `Guestbook` doesn't exist

Example

```c# // GET: api/Guestbook/1 [HttpGet("{id:int}")] public IActionResult GetById(int id) { var guestbook = _guestbookRepository.GetById(id); if (guestbook == null) { return NotFound(id); } return Ok(guestbook); }

```
- Add a new integration test class for the `GetById` method (in
Tests/Integration/Web)
    - Use `ApiToDoItemsControllerListShould` as a reference, if necessary
    - Add test data to Web/Startup.cs `PopulateTestData` method
        - Use ``Entries.Add`` instead of ``AddEntry`` when populating test data
(this avoids throwing events)
        - Add one `Guestbook` with one test `GuestbookEntry`
        - Use a disposable TestServerFixture (sample at end of lab)
    - Confirm the 404 behavior
    - Confirm entries are returned correctly (both the Guestbook and the
GuestbookEntry)

Example

```c#
    public class ApiGuestbookControllerListShould :
IClassFixture<TestServerFixture>
```

```
    {
        private readonly TestServerFixture _fixture;

        public ApiGuestbookControllerListShould(TestServerFixture fixture)
        {
            _fixture = fixture;
        }

        [Fact]
        public void ReturnGuestbookWithOneItem()
        {
            var response = _fixture.Client.GetAsync("/api/guestbook/1").Result;
            response.EnsureSuccessStatusCode();
            var stringResponse = response.Content.ReadAsStringAsync().Result;
            var result = JsonConvert.DeserializeObject<Guestbook>(stringResponse);

            Assert.Equal(1, result.Id);
            Assert.Equal(1, result.Entries.Count());
        }

        [Fact]
        public void Return404GivenInvalidId()
        {
            string invalidId = "100";
            var response =
_fixture.Client.GetAsync($"/api/guestbook/{invalidId}").Result;

            Assert.Equal(HttpStatusCode.NotFound, response.StatusCode);
            var stringResponse = response.Content.ReadAsStringAsync().Result;

            Assert.Equal(invalidId.ToString(), stringResponse);
        }
    }
```

In Startup.cs

```c# private void PopulateTestData(IApplicationBuilder app) { var dbContext = app.ApplicationServices.GetService();

```

    // reset the database
    dbContext.Database.EnsureDeleted();

    dbContext.ToDoItems.Add(new ToDoItem()
    {
        Title = "Test Item 1",
        Description = "Test Description One"
    });
    dbContext.ToDoItems.Add(new ToDoItem()
    {
        Title = "Test Item 2",
        Description = "Test Description Two"
    });
    dbContext.SaveChanges();

    // add Guestbook test data; specify Guestbook ID for use in tests
```
```

```
    var guestbook = new Guestbook() { Name = "Test Guestbook", Id=1 };
    dbContext.Guestbooks.Add(guestbook);
    guestbook.Entries.Add(new GuestbookEntry()
    {
        EmailAddress = "test@test.com",
        Message = "Test message"
    });
    dbContext.SaveChanges();
}
```

~~~
- Add an API method to record an entry to a Guestbook
    - Accept a Guestbook ID and a GuestbookEntry
    - Return a 404 Not Found if no Guestbook exists for the ID
    - Return the updated Guestbook if successful
- Add a new integration test class for the AddEntry method
    - Confirm the 404 behavior
    - Confirm the entry is created and sent to the repository successfully

### Example

**In Api/GuestbookController.cs**

```c#
    // POST: api/Guestbook/NewEntry
    [HttpPost("{id:int}/NewEntry")]
    public async Task<IActionResult> NewEntry(int id, [FromBody] GuestbookEntry
entry)
    {
        var guestbook = _guestbookRepository.GetById(id);
        guestbook.AddEntry(entry);
        _guestbookRepository.Update(guestbook);

        return Ok(guestbook);
    }
~~~

ApiGuestbookControllerNewEntryShould.cs

```c# public class ApiGuestbookControllerNewEntryShould : IClassFixture { private readonly TestServerFixture _fixture; //private readonly HttpClient _client; public ApiGuestbookControllerNewEntryShould(TestServerFixture fixture) { _fixture = fixture; }

```
    [Fact]
    public void Return404GivenInvalidId()
    {
        string invalidId = "100";
        var entryToPost = new { EmailAddress = "test@test.com", Message = "test" };
        var jsonContent = new
StringContent(JsonConvert.SerializeObject(entryToPost), Encoding.UTF8,
            "application/json");
        var response =
_fixture.Client.PostAsync($"/api/guestbook/{invalidId}/NewEntry",
jsonContent).Result;
```
```

```
        Assert.Equal(HttpStatusCode.NotFound, response.StatusCode);
        var stringResponse = response.Content.ReadAsStringAsync().Result;

        Assert.Equal(invalidId, stringResponse);
    }

    [Fact]
    public void ReturnGuestbookWithOneItem()
    {
        int validId = 1;
        string message = Guid.NewGuid().ToString();
        var entryToPost = new { EmailAddress = "test@test.com", Message = message };
        var jsonContent = new
StringContent(JsonConvert.SerializeObject(entryToPost), Encoding.UTF8,
            "application/json");
        var response =
_fixture.Client.PostAsync($"/api/guestbook/{validId}/NewEntry", jsonContent).Result;
        response.EnsureSuccessStatusCode();
        var stringResponse = response.Content.ReadAsStringAsync().Result;
        var result = JsonConvert.DeserializeObject<Guestbook>(stringResponse);

        Assert.Equal(validId, result.Id);
        Assert.True(result.Entries.Any(e => e.Message == message));
    }
}
```

~~~
**Note:** If you get test failures due to no mail server being found, make sure you
have smtp4dev / postman running. Otherwise, you may wish to configure your
TestServer to use a different/mock implementation of `IMessageSender`.

- Extract the 404 behavior into a new `VerifyGuestbookExistsAttribute`

**Add a filter to handle 404 policy**
- Create a new Web/Filters/VerifyGuestbookExistsAttribute.cs file
- See https://github.com/ardalis/GettingStartedWithFilters for reference
- Inherit from `TypeFilterAttribute`
- Create a constructor that chains to base() passing in the
`typeof(VerifyGuestbookExistsFilter)`
- Create a private class `VerifyGuestbookExistsFilter`
- Inherit from `IAsyncActionFilter`
- Create a constructor that takes an `IGuestbookRepository`
- Implement `OnActionExecutionAsync`:

```c#
    public async Task OnActionExecutionAsync(ActionExecutingContext context,
                                                ActionExecutionDelegate next)
    {
        if (context.ActionArguments.ContainsKey("id"))
        {
            var id = context.ActionArguments["id"] as int?;
            if (id.HasValue)
            {
                if ((await _guestbookRepository.GetById(id.Value)) == null)
                {
                    context.Result = new NotFoundObjectResult(id.Value);
                    return;
~~~

```
                }
            }
        }
        await next();
    }
```

- Add the attribute to the API action methods that should return 404 when no guestbook is found
  - `[VerifyGuestbookExists]`
  - Can be applied to a Controller to apply it to every Action of the Controller
- Remove the logic from the API methods to do guestbook existence checks and 404 responses
- Confirm that the behavior remains the same (re-run integration tests)

Example:

TestServerFixture

```` ```c# using System; using System.IO; using System.Net.Http; using System.Net.Http.Headers; using CleanArchitecture.Web; using CleanArchitecture.Infrastructure.Data; using Microsoft.AspNetCore.Hosting; using Microsoft.AspNetCore.TestHost; using Microsoft.EntityFrameworkCore; using Microsoft.Extensions.DependencyInjection; using Microsoft.Extensions.Logging; ````

```
namespace CleanArchitecture.Tests.Integration.Web
{
    //
http://www.stefanhendriks.com/2016/04/29/integration-testing-your-dot-net-core-app-
with-an-in-memory-database/
    public class TestServerFixture : IDisposable
    {
        public TestServer Server { get; }
        public HttpClient Client { get; }

        public TestServerFixture()
        {
            var builder = new WebHostBuilder()
                .UseContentRoot(Directory.GetCurrentDirectory())
                .ConfigureServices(services =>
                {
                    services.AddDbContext<AppDbContext>(options =>
                        options.UseInMemoryDatabase(Guid.NewGuid().ToString()));
                })
                .ConfigureLogging(lf =>
                {
                    lf.AddConsole(LogLevel.Warning);
                })
                .UseStartup<Startup>()
                .UseEnvironment("Testing"); // ensure ConfigureTesting is called in
Startup

            Server = new TestServer(builder);
            Client = Server.CreateClient();
```
````

```
            // client always expects json results
            Client.DefaultRequestHeaders.Clear();
            Client.DefaultRequestHeaders.Accept.Add(
                new MediaTypeWithQualityHeaderValue("application/json"));
        }

        public void Dispose()
        {
            Server.Dispose();
            Client.Dispose();
        }
    }
}
```

\`\`\`