DDD with ASP.NET Core – Labs

Steve Smith (@ardalis) **January 2017**

Lab 0: Set up

Goal

Have a working development environment and starter application.

Requirements

Visual Studio 2015 or Visual Studio Code plus CLI tools. A git client. **Note: Not VS2017 or .NET Core 1.1.**

It will also be helpful to install:

- Postman or Fiddler or another tool that allows you to GET/POST/PUT/DELETE to HTTP endpoints
- Smtp4Dev or Papercut or another local email testing tool

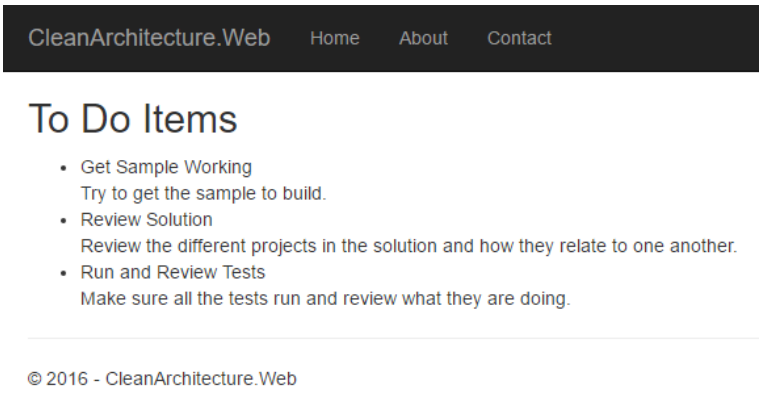
Instructions

We're going to be building a Guestbook application today using ASP.NET Core, Domain-Driven Design principles and patterns, and a domain-centric architecture. It's important that the solution be structured properly to support this approach. We will be starting from a Clean Architecture template which you'll find here:

https://github.com/ardalis/CleanArchitecture

Clone this sample to a working folder on your machine. If you download a ZIP of the repository, be sure to **Unblock** the zip file before you extract its contents (right click, Properties, Unblock).

You should be able to run the application and view the web app in your browser. You should be able to populate and view several ToDo items (see image below):



You should be able to run the tests in the tests folder, either from the command line or a test runner within your IDE. They should all pass.

For now the labs will only be using InMemory data, so no database configuration is required at this time. The application's data will reset each time it is started.

Rename the solution to **DDDGuestbook.sln**.

Lab 1: The Guest Book application

Goal

Create a domain model consisting of a Guestbook and GuestbookEntry Entities. Display (dummy) entries on the web application's home page.

Topics Used

Entities, ViewModels

Requirements

We are building an online guestbook. Anonymous users will be able to view entries on the guestbook, as well as (later) leave messages (GuestbookEntries) on the site. Each GuestbookEntry must include a value for the user's email address and a message, as well as a DateTimeCreated. For now these won't have much behavior; we'll add more later.

The home page should display the most recent ten (10) entries recorded on the guestbook, ordered by when the entry was recorded (most recent first). For now, have the home page just display several hard-coded entries.

Details

Entities

1. Guestbook and GuestbookEntry should inherit from BaseEntity, which defines an integer Id property.
2. Entities should be defined in the Core project, in the Entities folder.
3. Guestbook should include a string Name property.
4. Guestbook should include a List<GuestbookEntry> Entries property.
5. GuestbookEntry should include
    a. string EmailAddress
    b. string Message
    c. DateTimeOffset DateTimeCreated (defaults to Now)

Example:

```
0 references | 0 changes | 0 authors, 0 changes
public class Guestbook : BaseEntity
{
    0 references | 0 changes | 0 authors, 0 changes
    public string Name { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public List<GuestbookEntry> Entries { get; } = new List<GuestbookEntry>();
}

5 references | 0 changes | 0 authors, 0 changes
public class GuestbookEntry : BaseEntity
{
    0 references | 0 changes | 0 authors, 0 changes
    public string EmailAddress { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public string Message { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public DateTimeOffset DateTimeCreated { get; set; } = DateTime.UtcNow;
}
```

UI and ViewModel

6. The Home Page will display the guestbook name and up to 10 entries. Create a HomePageViewModel class in /ViewModels in the Web project. Include these properties:
    a. string GuestbookName
    b. List<GuestbookEntry> PreviousEntries
7. HomeController Index() should create a new Guestbook (Name="My Guestbook")
    a. Add several entries to its Entries collection (just hard-code them)
    b. Create a HomePageViewModel from a Guestbook and its entries
    c. Return a View, passing the viewmodel to the View
8. Modify the View/Index.cshtml view file in the Web project
    a. Specify @model CleanArchitecture.Web.ViewModels.HomePageViewModel at the top of the view file
    b. Display the Guestbook Name at the top of the page
    c. Loop through the Entries and display each entry, most recent first.

Example:

```
4 references | 0 changes | 0 authors, 0 changes
public class HomePageViewModel
{
    2 references | 0 changes | 0 authors, 0 changes
    public string GuestbookName { get; set; }
    3 references | 0 changes | 0 authors, 0 changes
    public List<GuestbookEntry> PreviousEntries { get; } = new List<GuestbookEntry>();
}
```

```
0 references | Steve Smith, 3 hours ago | 1 author, 1 change
public class HomeController : Controller
{
    0 references | Steve Smith, 3 hours ago | 1 author, 1 change
    public IActionResult Index()
    {
        var guestbook = new Guestbook() {Name = "My Guestbook"};
        guestbook.Entries.Add(new GuestbookEntry() { EmailAddress = "steve@devig.com", Message = "Hi!", DateTimeCreated = DateTime.UtcNow.AddHours(-2) });
        guestbook.Entries.Add(new GuestbookEntry() { EmailAddress = "mark@devig.com", Message = "Hi again!", DateTimeCreated = DateTime.UtcNow.AddHours(-1)});
        guestbook.Entries.Add(new GuestbookEntry() { EmailAddress = "michelle@devig.com", Message = "Hello!" });

        var viewModel = new HomePageViewModel();
        viewModel.GuestbookName = guestbook.Name;
        viewModel.PreviousEntries.AddRange(guestbook.Entries);

        return View(viewModel);
    }
```

cs ToDoItem.cs HomePageViewModel.cs Index.cshtml

```
@{
    ViewData["Title"] = "DDD Guestbook";
}
@model CleanArchitecture.Web.ViewModels.HomePageViewModel

<h2>Guestbook: @Model.GuestbookName</h2>
<h3>Messages:</h3>
<ol class="round">
    @if (!Model.PreviousEntries.Any())
    {
        <li class="zero">
            <h5>No Messages</h5>
            Nobody has left any messages.  How sad. :(
        </li>
    }
    @foreach (var entry in Model.PreviousEntries.OrderByDescending(p => p.DateTimeCreated))
    {
        <li class="arrow">
            <h5>@entry.EmailAddress - @entry.DateTimeCreated</h5>
            @entry.Message
        </li>
    }
</ol>
```

Lab 2: Adding Entries to the Guestbook

Goal

Add a form to the page that allows users to add new entries to the Guestbook.

Topics Used

Repository

Requirements

Display up to 10 of the most recent entries *from persistence* on the home page. For now, there is only ever one instance of a Guestbook. If none exists in persistence, create one when the home page loads.

Add a form that accepts an **EmailAddress** and a **Message** and POSTs to the IndexController. Persist new entries to the Guestbook.

Details

The CleanArchitecture template includes an IRepository<T> and an EFRepository<T> implementation already. You can see an example of it in action in the Web/Controllers/TodoController.cs class. Follow this example to pass an IRepository<Guestbook> into HomeController's constructor and assign it to a private local field.

Determine if there is already a Guestbook in persistence. If not, create and save one in the HomeController's Index() method, using the repository (you can use the code you have from lab 1 that creates a guestbook with several entries). In any case, fetch the first Guestbook from persistence and use it to populate the HomePageViewModel.

Example:

```
using System;
using System.Linq;
using CleanArchitecture.Core.Entities;
using CleanArchitecture.Core.Interfaces;
using CleanArchitecture.Web.ViewModels;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore.Migrations;

namespace CleanArchitecture.Web.Controllers
{
    1 reference | Steve Smith, 30 minutes ago | 1 author, 2 changes
    public class HomeController : Controller
    {
        private readonly IRepository<Guestbook> _guestbookRepository;

        0 references | 0 changes | 0 authors, 0 changes
        public HomeController(IRepository<Guestbook> guestbookRepository)
        {
            _guestbookRepository = guestbookRepository;
        }

        0 references | Steve Smith, 30 minutes ago | 1 author, 2 changes
        public IActionResult Index()
        {
            if (!_guestbookRepository.List().Any())
            {
                var newGuestbook = new Guestbook() {Name = "My Guestbook"};
                newGuestbook.Entries.Add(new GuestbookEntry()
                {
                    EmailAddress = "steve@deviq.com",
                    Message = "Hi!" });
                _guestbookRepository.Add(newGuestbook);
            }

            //var guestbook = _guestbookRepository.List().FirstOrDefault();
            var guestbook = _guestbookRepository.GetById(1);
            var viewModel = new HomePageViewModel();
            viewModel.GuestbookName = guestbook.Name;
            viewModel.PreviousEntries.AddRange(guestbook.Entries);

            return View(viewModel);
        }
```

Implement a Repository for Guestbook

The CleanArchitecture template comes with a generic EF repository, but it requires some updates to support Guestbook. Specifically, the dbcontext needs to have a DbSet of Guestbooks and when a Guestbook is fetched, it needs to include the associated Entries collection with it.

1. Add a DbSet<Guestbook> property 'Guestbooks' to the Infrastructure/Data/AppDbContext.cs class.
2. Modify the EfRepository to make its _dbContext protected and its GetById method virtual.
3. Create a new GuestbookRepository class that inherits from EfRepository<Guestbook>
    a. Provide a constructor that passes a dbContext to the base constructor
    b. Override GetById as shown:
    c. Include `using Microsoft.EntityFrameworkCore;`
4. In Startup, configure IRepository<Guestbook> to use GuestbookRepository.

```
public class GuestbookRepository : EfRepository<Guestbook>
{
    public GuestbookRepository(AppDbContext dbContext) : base(dbContext)
    {
    }

    // Since Guestbook is an Aggregate Root we need it to include its children
    public override Guestbook GetById(int id)
    {
        return _dbContext.Guestbooks
            .Include(g => g.Entries)
            .FirstOrDefault(g => g.Id == id);
    }

}
```

```
For<IRepository<Guestbook>>().Use<GuestbookRepository>();
```

At this point you should be able to view the home page and see your test data displayed, which is being stored in persistence (on the first request) and then fetched from persistence (on each request).

Implement Adding Entries

To support adding new entries to the guestbook, you need an HTML form in the view and a new controller action method to handle POSTs.

1. Update /Views/Home/Index.cshtml to include a form above the message list. ASP.NET Core tag helpers provide a replacement for HTML Helpers used in previous versions of ASP.NET MVC.

Example:

```
<form asp-controller="Home" asp-action="Index" method="post" class="form-horizontal">
    <h4>Sign the Guestbook</h4>
    <hr />
    <div asp-validation-summary="All" class="text-danger"></div>
    <div class="form-group">
        <label asp-for="NewEntry.EmailAddress" class="col-md-2 control-label"></label>
        <div class="col-md-10">
            <input asp-for="NewEntry.EmailAddress" class="form-control" />
            <span asp-validation-for="NewEntry.EmailAddress" class="text-danger"></span>
        </div>
    </div>
    <div class="form-group">
        <label asp-for="NewEntry.Message" class="col-md-2 control-label"></label>
        <div class="col-md-10">
            <input asp-for="NewEntry.Message" class="form-control" />
            <span asp-validation-for="NewEntry.Message" class="text-danger"></span>
        </div>
    </div>
    <div class="form-group">
        <div class="col-md-offset-2 col-md-10">
            <button type="submit" class="btn btn-default">Save</button>
        </div>
    </div>
</form>
```

Note that the form **posts** to the **Home** controller's **Index** method, and the two values are EmailAddress and Message from a NewEntry property on the viewmodel.

2. Add a **NewEntry** property of type GuestbookEntry to the HomePageViewModel.
3. Add a new action method to /Controllers/HomeController.cs, Index(HomePageViewModel model)
    a. Add the **[HttpPost]** attribute to the method
    b. Check if ModelState.IsValid. If it is:
        i. Fetch the appropriate guestbook (in this case the first one, or ID 1)
        ii. Add the NewEntry from the model to the guestbook's Entries collection.
        iii. Update the guestbook using the _guestbookRepository.
        iv. Update the model's PreviousEntries collection to match guestbook.Entries
    c. Return View(model)

Example:

```
[HttpPost]
0 references | 0 changes | 0 authors, 0 changes
public IActionResult Index(HomePageViewModel model)
{
    if (ModelState.IsValid)
    {
        var guestbook = _guestbookRepository.GetById(1);
        guestbook.Entries.Add(model.NewEntry);
        _guestbookRepository.Update(guestbook);

        model.PreviousEntries.Clear();
        model.PreviousEntries.AddRange(guestbook.Entries);
    }
    return View(model);

}
```

Discussion/Review Questions:

1. What do you think about this design so far?
2. Is it following separation of concerns?
3. Is it testable?
4. Is there anything you would refactor to improve it?
5. How is model validation working at this point?
6. Why do you think we have a Guestbook in the model, rather than just Entries?

Lab 3: Notifying Users of New Entries

Goal

When a new entry is added to the guestbook, notify anyone who has previously signed the guestbook, but only within the last day.

Topics Used

Domain Events, Services

Requirements

Implement logic in the controller's POST action to loop through previous entries and send an email to any that have been made within the last day. You can use the "Mailkit" nuget package to send emails; add it to the Web **project.json**.

See: http://stackoverflow.com/questions/33496290/how-to-send-email-by-using-mailkit

```
    "CleanArchitecture.Core": "1.0.0-*",
    "CleanArchitecture.Infrastructure": "1.0.0-*",
    "MailKit": "1.8.1"
  },
```

Manually test that your solution works, using Smtp4Dev/Papercut or similar. If you get an error with your mail server, see: http://stackoverflow.com/a/40140468 (make sure STARTTLS is unchecked)

- What impact does this approach have on testability of the controller?
- Where does the business logic for this requirement live? How valuable is the current model in representing this logic?

Refactor to use a domain service:

Step 1: Encapsulate Email Sending in an Infrastructure service.

- Create an interface to represent sending email (e.g. IMessageSender)
  - Place in Core/Interfaces
  - Keep as simple as possible(e.g. `void SendGuestbookNotificationEmail(string toAddress, string messageBody);`)
- Implement the interface in Infrastructure (e.g. EmailMessageSenderService : IMessageSender)
  - Don't forget to add Mailkit to project.json in Infrastructure

```
using CleanArchitecture.Core.Interfaces;
using MailKit.Net.Smtp;
using MimeKit;

namespace CleanArchitecture.Infrastructure
{
    1 reference | Steve Smith, 37 minutes ago | 1 author, 1 change
    public class EmailMessageSenderService : IMessageSender
    {
        2 references | Steve Smith, 37 minutes ago | 1 author, 1 change
        public void SendGuestbookNotificationEmail(string toAddress, string messageBody)
        {
            string fromAddress = "donotreply@guestbook.com";
            var message = new MimeMessage();
            message.From.Add(new MailboxAddress("Guestbook", fromAddress));
            message.To.Add(new MailboxAddress(toAddress, toAddress));
            message.Subject = "New Message on GuestBook";
            message.Body = new TextPart("plain") { Text = messageBody };
            using (var client = new SmtpClient())
            {
                client.Connect("localhost", 25);
                client.Send(message);
                client.Disconnect(true);
            }
        }
    }
}
```

- In Startup.cs, configure IMessageSender to use EmailMessageSender.
- Inject IMessageSender into HomeController and re-test your email sending logic

Step 2: Encapsulate Sending Notifications
We want to pull the logic of deciding who should get a notification out of the POST action method and into a service in Core. **Note**, this approach is better than leaving the logic in the UI project, but not ideal because it splits business functionality related to Guestbook into a service, tending to make Guestbook *anemic*.

- Create an interface IGuestbookService
  - Place in Core/Interfaces
  - void RecordEntry(Guestbook guestbook, GuestbookEntry entry)
  - (later you could add additional methods, if desired)
- Create a new class, GuestbookService, in Core/Services
  - Implement IGuestbookService
  - Inject a repository to access guestbook (IRepository<Guestbook>)
  - Inject an IMessageSender to send emails
  - Implement RecordEntry, moving the logic for saving the entry and looping through previous entries from HomeController to this method.
  - Note: If you named your interface and implementation correctly, StructureMap will map them to one another automatically by convention.

Test your application again. It should still work as it did before. The controller method should be much smaller now than it was when it was responsible for identifying recipients, building messages, and sending emails.

```
[HttpPost]
0 references | Steve Smith, 46 minutes ago | 1 author, 3 changes
public IActionResult Index(HomePageViewModel model)
{
    if (ModelState.IsValid)
    {
        var guestbook = _guestbookRepository.GetById(1);
        _guestbookService.RecordEntry(guestbook, model.NewEntry);

        model.PreviousEntries.Clear();
        model.PreviousEntries.AddRange(guestbook.Entries);
    }
    return View(model);
}
```

Step 3: Use a Domain Event

Eliminate the need for a service that only works with one Entity by putting the logic into the Entity itself and raising a *domain event* to trigger additional behavior.

- Create a new domain event in Core/Events called EntryAddedEvent.
- Inherit from BaseDomainEvent.
- Include the entry and the guestbook ID in its constructor; expose these as properties.
- Modify Core/Entities/Guestbook.cs:
  - Add a new public void AddEntry(GuestbookEntry entry) method that adds the entry to Entries.
  - Create a new EntryAddedEvent and added it to the Events collection.
  - *(later) Change Entries to make it harder for clients to manipulate it directly – we only want new entries to be added using the AddEntry method.*
- Create a GuestbookNotificationHandler in Core/Handlers
  - Implement IHandle<EntryAddedEvent>
  - Inject IRepository<Guestbook> and IMessageSender
  - Copy notification logic from GuestbookService to Handle method
  - Update HomeController POST method to call Guestbook.AddEntry()

Test the application again – its behavior should remain unchanged.

Samples

```
namespace CleanArchitecture.Core.Events
{
    4 references | 0 changes | 0 authors, 0 changes
    public class EntryAddedEvent : BaseDomainEvent
    {
        2 references | 0 changes | 0 authors, 0 changes
        public int GuestbookId { get; set; }
        1 reference | 0 changes | 0 authors, 0 changes
        public GuestbookEntry Entry { get; set; }

        1 reference | 0 changes | 0 authors, 0 changes
        public EntryAddedEvent(int guestbookId, GuestbookEntry entry)
        {
            GuestbookId = guestbookId;
            Entry = entry;
        }
    }
}
```

In Guestbook:

```
public void AddEntry(GuestbookEntry entry)
{
    Entries.Add(entry);
    Events.Add(new EntryAddedEvent(this.Id, entry));
}
```

```
using System;
using System.Linq;
using CleanArchitecture.Core.Entities;
using CleanArchitecture.Core.Events;
using CleanArchitecture.Core.Interfaces;

namespace CleanArchitecture.Core.Handlers
{
    1 reference | 0 changes | 0 authors, 0 changes
    public class GuestbookNotificationHandler : IHandle<EntryAddedEvent>
    {
        private readonly IRepository<Guestbook> _guestbookRepository;
        private readonly IMessageSender _messageSender;

        0 references | 0 changes | 0 authors, 0 changes
        public GuestbookNotificationHandler(IRepository<Guestbook> guestbookRepository,
            IMessageSender messageSender)
        {
            _guestbookRepository = guestbookRepository;
            _messageSender = messageSender;
        }

        3 references | 0 changes | 0 authors, 0 changes
        public void Handle(EntryAddedEvent entryAddedEvent)
        {
            var guestbook = _guestbookRepository.GetById(entryAddedEvent.GuestbookId);

            // send updates to previous entries made within last day
            var emailsToNotify = guestbook.Entries
                .Where(e => e.DateTimeCreated > DateTimeOffset.UtcNow.AddDays(-1))
                .Select(e => e.EmailAddress);
            foreach (var emailAddress in emailsToNotify)
            {
                string messageBody = "{entry.EmailAddress} left new message {entry.Message}";
                _messageSender.SendGuestbookNotificationEmail(emailAddress, messageBody);
            }
        }
    }
}
```

Reference:

https://github.com/ardalis/ddd-guestbook *see Tags for different Lab starting points*