Jeffrey Palermo 8:14 am on July 30, 2008

Tags: onion architecture (4)

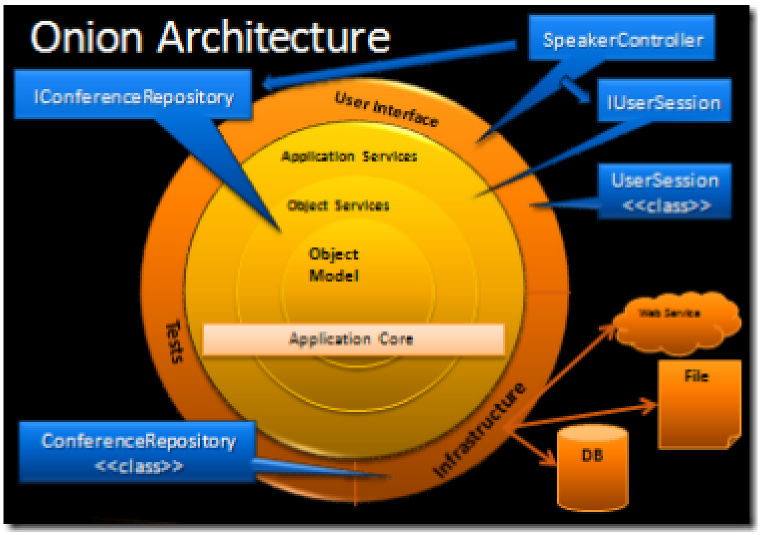
The Onion Architecture : part 2

part 1. This is part 2. part 3. part 4. My feed (rss).

In part 1, I introduced an architectural pattern that I have named "Onion Architecture". The object-oriented design concepts are not new, but I'm pulling together a lot of techniques and conventions into a single pattern and giving it a name. My hope is that the industry can use this name to communicate the architectural approach where appropriate.

Part 2: Practical example:

CodeCampServer uses the Onion Architecture. If you are looking for a full, working application as an example, please have a look. The practical example I put before you is taken directly from CodeCampServer. It is a narrow, vertical slice of an example. I'm keeping the scope small as to be digestible. I'll start with a diagram so you can understand where all the code resides within the layers of the onion.



CodeCampServer uses the ASP.NET MVC Framework, so SpeakerController is part of the user interface. This controller is coupled to the ASP.NET MVC Framework, and there is no getting around that. SpeakerController depends on IConferenceRepository and IUserSession (and IClock, but we'll omit that). The controller only depends on interfaces, which are defined in the application core. Remember that **all dependencies are toward the center**.

Turn your attention to the ConferenceRepository and UserSession classes. Notice that they are in layers outside of the application core, and they depend on the interfaces as well, so that they can implement them. These two classes each implement an interface closer to the center than itself. At runtime, our Inversion of Control container will look at its registry and construct the proper classes to satisfy the constructor dependencies of SpeakerController, which is the following:

```
public SpeakerController(IConferenceRepository conferenceRepository,
                         IUserSession userSession, IClock clock)
    : base(userSession)
{
    _conferenceRepository = conferenceRepository;
    _clock = clock;
    _userSession = userSession;
}
```

At runtime, the IoC container will resolve the classes that implement interfaces and pass them into the SpeakerController constructor. At this point in time, the SpeakerController can do its job.

Based on the rules of the Onion Architecture, the SpeakerController *could* use UserSession directly since it's in the same layer, but it cannot use ConferenceRepository directly. It must rely on something external passing in an instance of IConferenceRepository. This pattern is used throughout, and the IoC container makes this process seamless.

At the end of this series, I plan on publishing a full working system that adheres to the Onion Architecture pattern. The systems we build for clients use this approach, but I'm not at liberty to discuss that code, so I will craft a reference application for those of you who prefer a concrete Visual Studio solution to digest.