Architectural Choices

Syed Waqas Mahmood

Vanhack Evaluation Forum App

24-Jan-2017

- Technologies
  - .NET Framework
  - Entity Framework
  - Ninject
  - Microsoft Identity
  - OWIN
  - MS Sql Server
  - nUnit
- Design Patterns
  - Domain Driven Design
  - Repository Pattern
  - CQRS - Command Query Responsibility Segregation
  - Dependency Injection

Technologies

The following are the technologies used in the backend API of the application and the rationale behind their usage:

.NET Framework

The main framework used. I have used the .NET framework because this is the technology in which I have the most experience in and this is the main technology that I will like to work upon in the future as well. The projects use .NET Framework version 4.6.1.

Entity Framework

The ORM. The reason for using it is because it integrates seamlessly with the .NET framework and also the ease of use it provides to get up and running instantly. I have used Code-First migrations so it is very easy to automatically setup the database using the domain model in the code.

Ninject

The dependency Injection Framework used. All the configuration happens inside the code without the need of XML configuration et al and the code configuration is as easy to write just a few lines and we are set.

Microsoft Identity

Provides the membership system for ASP.NET, that provides all the security features that are required in modern web apps, including built-in support for different aspects of authentication, authorization and redirection-based log-ins to authentication providers such as Facebook, Twitter etc.
One huge benefit is that it integrates with OWIN seamlessly.

OWIN

OWIN is a standard for an interface between .NET Web applications and Web servers. Different middleware components can be integrated into the pipeline regardless of other components, which ends up in a loosely-coupled yet neatly knit middleware pipeline.

MS Sql Server

For database, I have used MS Sql Server as it the standard database in the .NET stack.

nUnit

For writing Unit test, I have used nUnit because it is the dominant framework of choice in the .NET world.

Design Patterns

Domain Driven Design

The app revolves around the concept of Domain Driven Design. Thus employing the concept of 'Bounded Contexts' where I have introduced 2 Bounded Contexts (BCs); Identity and Forum.

Identity BC is handles all the Identity And Access related operations and the Forum BC covers the operations for Pots, Comments & Categories.

I have used the **Layered architecture**, with the **Domain.Model** residing at the core, while the **Persistence layer** handling database operations for the Domain object. The **Application layer** directs traffic while the **Ports layer** handles in the incoming REST calls.

Single Responsibility Principle is honored, starting from the entities inside the Domain model, where the aggregate root encapsulates all of its own operations as well as of entities that reside within its aggregate. This is why you will see private setter in the entities, so that only the aggregate root (Post in the case of Forum BC)can modify the attributes within it's aggregate.

Identity BC uses Microsoft Identity so it is not required to define its Domain.Model.

Repository Pattern

Even though Entity Framework use Repository pattern and unit of work patterns on its own, it is a good idea to use the repository pattern to have a neat layout and to introduce new persistence modules in the future without the need to refactor the existing ones. We have a base implementation which is a generic implementation for any type, resulting in code reuse.
As this is a RESTful application, I have committed the transaction in each write operation as soon as we add/update/delete any entity, as in RESTful calls we are performing one operation per request anyways, so

CQRS - Command Query Responsibility Segregation

We do not want to expose our Domain Model to the outside world, hence, we have created Commands that serve as the request objects and Representation, that act as the query objects. Although this project was simple enough, I still have implemented CQRS in order to demonstrate it application and proof-of-concept.

Dependency Injection

Dependency injection is an integral part of the application and the lifetime of objects in this RESTful application depends on its use, for example, some of the objects' lifetime is transient, while most of them arse created per request.