Domain service injecting into domain entities

Asked 7 years, 3 months ago Active 7 years, 3 months ago Viewed 2k times

3

2

I'm learning about Domain-Driven-Design and I'm a little confused about entities and injecting domain services into them. I have found this blog and conclusion is that injecting services into entities is a bad idea. I partially agree with that, but what to do in this case: I have User entity which is an aggregate root, which has Password value object in it. It look like this:

Password value object:

```
public class Password
{
    public string Hash { get; private set; }

    public string Salt { get; private set; }

    private readonly IHashGeneratorService _hashGeneratorService;

    public Password(IHashGeneratorService hashGeneratorService)
    {
        _hashGeneratorService = hashGeneratorService;
    }

    public void GenerateHash(string inputString)
    {
        //Some logic

        Salt = hashGeneratorService.GenerateSalt();
        Hash = hashGeneratorService.GenerateHash(inputString);
    }
}
```

User entity:

```
            this.Password = new Password(hashGeneratorService);
        }
    }
```

In this case if I create User entity via factory I need to provide IHashGeneratorService implementation to factory constructor or `Create()` method. After that if my factory is used by, for ex. SomeUserService I have to provide implementation (ex. via ctor injection) to it. And so on... Honestly it smells form me, as lot of my classess are dependent on hash generator service implementation but only Password class uses it. And it also breaks SRP principle for Password class as I assume.

I've found out some solutions:

1. Use service locator. But it also smells as it's an anti-pattern and it is hard to test and manage entities if we use it.
2. Implement hashing alghoritm directly inside Password methods.
3. Stay with that what I have :) Cons are mentioned above, pros are that my classess are easier to test as I can provide mocked service instead of full implementation.

Personally I tend to refoactor my code to second solution as it does not break SRP (or it does? :)), classess are not dependent of hashing service implementation. Something more? Or do you have another solutions?

c# dependency-injection domain-driven-design

Share Improve this question Follow

edited Jan 27 '14 at 15:07

asked Jan 27 '14 at 10:47

Mad Max
157 10

+1 for infrastructure layer and use of appliaction layer and also for Value Object (that was my mistake and i corrected it in my post :)) But I'm not sure if appliaction layer should be responsible for creating User entity. I think it's better to leave it in factory, as factory creates User entity with initialized all needed fields including Password value object. – Mad Max Jan 27 '14 at 15:09

I posted an answer afterall =P Well, in case instanciating a `User` object is quite complex, it could be done in a factory (implemented as a domain

@Alexander Langer - I thought that value objects does not have their identity and cannot exist by itself, but their state can be changed – Mad Max Jan 27 '14 at 22:22

@MadMax Value objects should be immutable. If the value changes, create a new value object with the changed values and replace the original with the new one. Check martinfowler.com/bliki/ValueObject.html for reference. – Tuukka Haapaniemi May 6 '16 at 7:43

2 Answers

Active | Oldest | Votes

7

I am quite new to DDD, however I belive that hashing passwords is not a concern of the domain, but a technical concern, just like persistence. The hash service should have it's interface defined in the **domain**, but it's implementation in the **infrastructure** layer. The **application service** would then use the **hash service** to hash the password and create a `Password` instance (which should be a **value object**) before passing it to the `User` aggregate root.

There might be cases where an aggregate has to use a service like when the dependency resolutions are very complex and domain-specific. In this case, the application service could pass a domain service into the aggregate method. The aggregate would then double-dispatch to the service to resolve references.

For more information you can read the Implementing Domain-Driven Design book written by Vaughn Vernon. He speaks about this around page 362 (Model Navigation), but also at a few other places in the book.

Share Improve this answer Follow

answered Jan 27 '14 at 15:14

plalx
39.2k 5 63 83

0

I don't know reasons, why do you consider injection of constructor parameters only. AFAIK, it's a common feature for DI-containers to inject properties or fields. E.g., using MEF you could write something like this:

```
class SomeUserService : ISomeUserService
{
```

and inject a dependency only in those types, where you really need it.

Share Improve this answer Follow

answered Jan 27 '14 at 11:06

Dennis
34.9k 9 72 134

1 That's right, but I wonder if it is good idea to inject (not how to do it) service into entities. Moreover if I use property injection, it gets even harder to test then with ctor injection. – Mad Max Jan 27 '14 at 11:14

@MaciekKorzeniewski: personally, I don't like this idea. I prefer approach, where entities are just data containers, and logic is implemented outside in service layer. Th reason is that I don't want to test my domain entities at all. – Dennis Jan 27 '14 at 11:18

1 Well, but isn't your solution an anemic domain model? – Mad Max Jan 27 '14 at 11:20

And like I wrote above, my hashing service is used only inside Password entity, so I need to inject it only to Password class and property injection eases it as other classess are not dependent on hasing service implementation, but it does not get me less confused about injection services into entities. – Mad Max Jan 27 '14 at 11:23

Using property injection makes the class tightly coupled to a specific DI container implementation and makes your API lies about it's contract (it hides from it's user there is an additional dependency that needs to be provided). I consider this flavor of DI as the last resort where the application framework requires use of parameterless ctor. Like in MVC filters. – Bartłomiej Szypelow Jan 28 '14 at 10:53