Inversion of Control Proof of Concept (IocPoc)

The IocPoc solution attempts to demonstrate the advantages of a modular, loosely coupled software architecture based on abstractions, inversion of control (IoC), and dependency injection (DI). For more information, research SOLID software design and read the book “Dependency Injection in .NET” by Mark Seemann.

The IocPoc solution contains 3 Web API projects, a Domain project, and 2 Payment Gateways. The domain defines a payment gateway interface that is implemented by the PayPal and AuthorizeDotNet gateways.

1. The Web APIs
   1. IocPoc.NoDI
      1. References PayPal and AuthorizeDotNet projects (DLL’s).
         1. AuthorizeDotNet not used
      2. The controller creates an instance of PayPal and returns a message from the gateway
   2. IocPoc.DiOne
      1. References IocPoc.Domain
      2. Uses dependency injection (DI) to inject the PayPal gateway via the controller constructor
      3. PayPal DLL copied to bin folder, but no reference to it is made in the project
         1. UI finds the DLL and injects it
   3. IocPoc.DiTwo
      1. Controller method signature exactly the same as IocPoc.DiOne above
      2. web.config Unity mapping of AuthorizeDotNet to the IPaymentGateway interface

ASSUMPTIONS:

All the code is compiled and deployed.

THE STAKEHOLDER REQUEST:

IocPoc.NoDI and IocPoc.DiOne are to use AuthorizeDotNet instead of PayPal. Modifications and redeployment of the API DLLs are not allowed for various business reasons.

THE SOLUTION:

1. IocPoc.NoDI
   1. Because of a tightly coupled architecture, **the request cannot be met**. In addition, if the code were to be modified, the AuthorizeDotNet gateway class is marked internal and inaccessible to instantiation
2. IocPoc.DiOne
   1. **The request can be met** due to loose coupling and interchangeable modules
   2. Copy AuthorizeDotNet DLL to the bin folder
   3. Modify the Unity config section in web.config to map the AuthorizeDotNet DLL to the interface