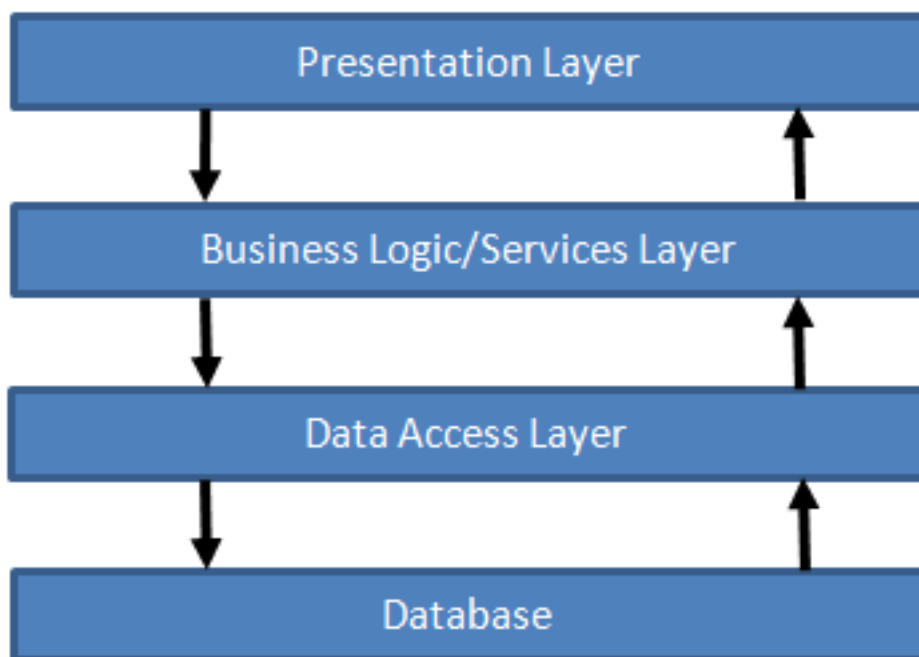


مقدمه

نوشتن تست برای کدها بسیار عالی است، در صورتیکه بدانید چگونه این کار را بدرستی انجام دهید. متأسفانه بسیاری از منابع آموزشی موجود، این مطلب که چگونه کد قابل تست بنویسیم را رها می‌کنند؛ بدلیل اینکه آنها مراقبت در بین لایه‌هایی که در کدهای واقعی وجود دارند گیر نکنند، جایی که شما لایه‌های خدمات (Service Layer)، لایه‌های داده، و غیره را دارید. به ضرورت، وقتی میخواهید کدی را تست کنید که این وابستگی‌ها را دارد، تست‌ها بسیار کند و برای نوشتن دشوار هستند و اغلب بدلیل وابستگی‌ها شکست می‌خورند و نتیجه غیر قابل انتظاری خواهند داشت.

پیش زمینه

کدی که به خوبی نوشته شده باشد از لایه‌های جداگانه‌ای تشکیل شده است که هر لایه مسئول یک قسمت متفاوت از وظایف برنامه خواهد بود. لایه‌های واقعی بر اساس نیاز و نظر توسعه دهندگان متفاوت است، ولی یک ساختار رایج به شکل زیر خواهد بود.



لایه نمایش / رابط کاربری : این قسمت کد منطق نمایش و تعامل رابط کاربری می‌باشد. **منطق تجاری / لایه خدمات :** این قسمت منطق تجاری کد شما می‌باشد. برای نمونه در کد مربوط به یک کارت خرید، این کارت خرید میداند چگونه جمع کارت را محاسبه نماید و یا چگونه اقلام موجود در سفارش را شمارش کند. **لایه دستیابی به داده / لایه ماندگاری :** این کد میداند چگونه به منبع داده متصل شود و یک کارت خرید را بازگرداند و یا چگونه یک کارت را در منبع داده ذخیره نماید. **منبع داده :** اینجا جایی است که محتویات کارت در آن ذخیره میشود. **مزیت مدیریت وابستگی‌ها**

بدون مدیریت وابستگی‌ها، وقتی شما برای لایه نمایش تست می‌نویسید، کد شما در خدمات واقعی که به کد واقعی دستیابی به منبع داده وابسته است، گرفتار می‌شود و سپس به منبع داده اصلی متصل می‌شود. در واقع، وقتی شما در حال تست نویسی برای

گزینه "اضافه به کارت" و یا "دریافت تعداد اقلام" هستید، می‌خواهید کد را به صورت مجزا تست کنید و قادر باشید نتایج قابل پیش بینی را تضمین نمایید. بدون مدیریت وابستگی‌ها، تست‌های نمایش شما برای گزینه "اضافه به کارت" کند هستند و وابستگی‌ها نتایج غیر قابل پیش بینی بازمیگردانند که میتوانند باعث پاس نشدن تست شما شوند.

راه حل تزریق وابستگی است

راه حل این مسأله تزریق وابستگی است. تزریق وابستگی برای کسانی که تا بحال از آن استفاده نکرده اند، اغلب گیج کننده و پیچیده به نظر میرسد، اما در واقع، مفهومی بسیار ساده و فرآیندی با چند اصل ساده است. آنچه می‌خواهیم انجام دهیم مرکزیت دادن به وابستگی هاست. در این مورد، استفاده از شیء کارت خرید است و سپس رابطه بین کدها را کم می‌کنیم تا جاییکه وقتی شما برنامه را اجرا می‌کنید، از خدمات و منابع واقعی استفاده کند و وقتی آنرا تست می‌کنید، می‌توانید از خدمات جعلی (mocking) استفاده نمایید که سریع و قابل پیش بینی هستند. توجه داشته باشید که رویکردهای متفاوت بسیاری وجود دارند که می‌توانید استفاده کنید، ولی من برای ساده نگهداشتن این مطلب، فقط رویکرد Constructor Injection را شرح می‌دهم.

گام 1 - وابستگی‌ها را شناسایی کنید

وابستگی‌ها وقتی اتفاق می‌افتند که کد شما از لایه‌های دیگر استفاده می‌نماید. برای نمونه، وقتی لایه نمایش از لایه خدمات استفاده می‌نماید. کد نمایش شما به لایه خدمات وابسته است، ولی ما می‌خواهیم کد لایه نمایش را به صورت مجزا تست کنیم.

```
public class ShoppingCartController : Controller
{
    public ActionResult GetCart()
    {
        //shopping cart service as a concrete dependency
        ShoppingCartService shoppingCartService = new ShoppingCartService();
        ShoppingCart cart = shoppingCartService.GetContents();
        return View("Cart", cart);
    }
    public ActionResult AddItemToCart(int itemId, int quantity)
    {
        //shopping cart service as a concrete dependency
        ShoppingCartService shoppingCartService = new ShoppingCartService();
        ShoppingCart cart = shoppingCartService.AddItemToCart(itemId, quantity);
        return View("Cart", cart);
    }
}
```

گام 2 - وابستگی‌ها را مرکزیت دهید

این کار با چندین روش قابل انجام است؛ در این مثال من می‌خواهم یک متغیر عضو از نوع ShoppingCartService ایجاد کنم و سپس آنرا به وهله ای که در Constructor ایجاد خواهم کرد، منتسب کنم. حال هر جا ShoppingCartService نیاز باشد بجای آنکه یک وهله جدید ایجاد کنم، از این وهله استفاده می‌نمایم.

```
public class ShoppingCartController : Controller
{
    private ShoppingCartService _shoppingCartService;
    public ShoppingCartController()
    {
        _shoppingCartService = new ShoppingCartService();
    }
    public ActionResult GetCart()
    {
        //now using the shared instance of the shoppingCartService dependency
        ShoppingCart cart = _shoppingCartService.GetContents();
        return View("Cart", cart);
    }
    public ActionResult AddItemToCart(int itemId, int quantity)
    {
        //now using the shared instance of the shoppingCartService dependency
        ShoppingCart cart = _shoppingCartService.AddItemToCart(itemId, quantity);
        return View("Cart", cart);
    }
}
```