2 Pisanie ściśle powiązanego kodu

W rozdziale:

- Pisanie ściśle powiązanej aplikacji
- Ocena strukturalności aplikacji
- Analiza braku strukturalności w aplikacji

Tak, jak wspomnieliśmy o tym w rozdziale 1, *sos berneński* to zemulgowany sos na bazie żółtek i masła, ale ta wiedza nie powoduje, że w magiczny sposób ktoś potrafi przygotować taki dodatek do dań. Najlepszym sposobem, by się tego nauczyć, jest próbowanie, ale często jeden przykład może przybliżyć pustą przestrzeń między teorią a praktyką oraz pozbyć się jej. Oglądanie profesjonalnego kucharza przygotowującego sos berneński jest pomocne, zanim samemu wypróbuje się przygotowanie własnej wersji.

Kiedy wprowadziliśmy Wstrzykiwanie zależności we wcześniejszym rozdziale, zabraliśmy czytelników na przejażdżkę na wysokim poziomie, żeby każdy zrozumiał cel i ogólne zasady DI. Ale to proste wytłumaczenie nie oddaje sprawiedliwości Wstrzykiwaniu zależności. DI to sposób na włączenie luźnego wiązania, a luźne wiązanie to przede wszystkim skuteczny sposób na poradzenie sobie ze złożonością.

Większość oprogramowania jest tak złożona, że musi adresować wiele kwestii jednocześnie. Oprócz zagadnień biznesowych, które mogą być kompleksowe same w sobie, oprogramowanie musi również rozwiązywać kwestie związane z bezpieczeństwem, diagnostyką, operacjami, wydajnością i rozszerzalnością. Zamiast odnosić się do wszystkich tych zagadnień w jednej wielkiej kuli błota (*a big ball of mud*), luźne wiązanie zachęca do odnoszenia się do każdego zagadnienia osobno. Łatwiej robić to w odizolowaniu, ale ostatecznie i tak aplikacja musi zostać zbudowana ze złożonego zbioru tych zagadnień.

W tym rozdziale przyjrzymy się bardziej złożonemu przykładowi. Pokażemy, jak łatwo napisać ściśle powiązany kod. Wraz z nami będzie można przeanalizować, dlaczego taki ściśle powiązany kod jest problematyczny z perspektywy utrzymywalności. W rozdziale 3 użyjemy DI do całkowitego przepisania ściśle powiązanego kodu na taki, który jest luźno powiązany. Jeśli ktoś chce przyjrzeć się temu drugiemu kodowi od razu, może pominąć rozdział 3. A jeśli nie, kiedy skończy czytać o tym zagadnieniu, powinien zacząć rozumieć, co sprawia, że ściśle powiązany kod jest tak problematyczny.

2.1. Budowanie ściśle powiązanej aplikacji

Pomysł budowania luźno powiązanego kodu nie jest szczególnie kontrowersyjnym pomysłem, ale istnieje duża dysproporcja między teorią a praktyką. Zanim pokażemy w następnym rozdziale, jak użyć DI do zbudowania luźno powiązanej aplikacji, chcemy wskazać, jak łatwo może to pójść nie po czyjejś myśli. Częstym podejściem do luźno powiązanego kodu jest budowanie aplikacji złożonej z warstw. Każdy może narysować diagram trójwarstwowej aplikacji, a rys. 2.1 dowodzi, że i my to potrafimy.

Rysunek 2.1. Standardowa architektura trójwarstwowej aplikacji. Jest to najprostszy i najczęściej spotykany wariant architektury n-warstwowej aplikacji, gdzie aplikacja jest stworzona z *n* różnych warstw

Stworzenie takiego trójwarstwowego diagramu jest pozornie proste, ale samo rysowanie jakiegokolwiek diagramu jest podobne do zażyczenia sobie sosu berneńskiego do steku: jest to deklaracja chęci, która nie niesie za sobą żadnej gwarancji odnośnie do finalnego produktu. Można uzyskać coś innego. Będzie można to zobaczyć już za chwilę.

Istnieje więcej niż jeden sposób patrzenia i tworzenia elastycznej i utrzymywalnej złożonej aplikacji, ale *n*-warstwowa architektura aplikacji to podejście „wypróbowane i sprawdzone". Wyzwaniem jest zaimplementowanie jej poprawnie. Będąc uzbrojonym w trójwarstwowy diagram, taki jak ten na rys. 2.1, można zacząć budowanie aplikacji.