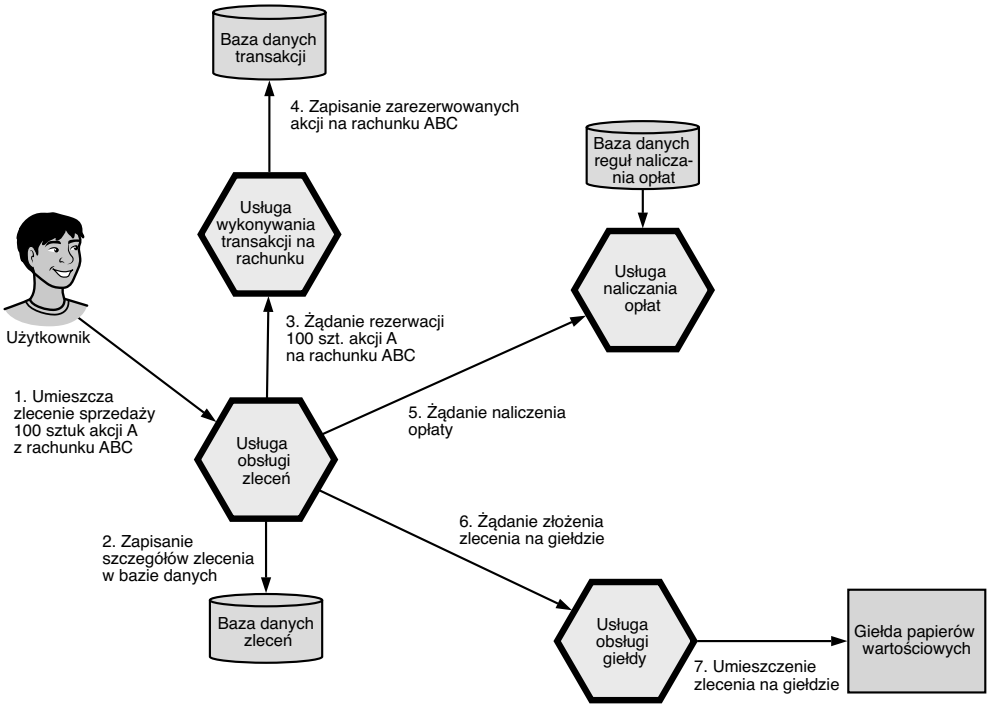
2.3.2. Współpraca usług

Zidentyfikowaliśmy kilku kandydatów na mikroserwisy. Te usługi muszą ze sobą współpracować, aby zrobić coś użytecznego dla klientów SimpleBanku.

Jak już wiemy, współpraca usług może być realizowana zarówno jako punkt–punkt, jak i sterowana zdarzeniami. Komunikacja punkt–punkt jest zazwyczaj synchroniczna, a komunikacja sterowana zdarzeniami jest asynchroniczna. Wiele aplikacji mikroserwisowych zaczyna od użycia komunikacji synchronicznej. Motywacje do tego są dwojakie:

- połączenia synchroniczne są zazwyczaj prostsze i bardziej zrozumiałe niż interakcja asynchroniczna; nie myślmy, że mają one te same cechy, co lokalne wywołania funkcji wykonywane w trakcie procesu – żądania w sieci są znacznie wolniejsze i bardziej nieprzewidywalne;
- większość, jeśli nie wszystkie, ekosystemy programistyczne obsługują już prosty, językowo agnostyczny mechanizm transportowy z szerokim spektrum programistycznym – HTTP, który jest wykorzystywany głównie do synchronicznych wywołań, ale można go również użyć asynchronicznie.

Rozważmy proces składania zleceń w SimpleBanku. Usługa obsługi zleceń jest odpowiedzialna za rejestrację i złożenie zlecenia na giełdę. Aby to zrobić, musi wejść w interakcje z usługami dotyczącymi giełdy, opłat i transakcji na rachunku. Ta współpraca została przedstawiona na rysunku 2.6.



Rysunek 2.6. Usługa obsługi zleceń kieruje zachowaniem kilku innych usług w celu złożenia zlecenia na giełdę

Wcześniej wskazaliśmy, że mikroserwisy powinny być autonomiczne, a żeby to osiągnąć, usługi powinny być luźno powiązane. Osiągniemy to częściowo przez zaprojektowanie naszych usług, gromadzenie razem rzeczy, które zmieniają się z tych samych powodów w celu zminimalizowania szansy, że zmiany w jednej usłudze wymagają zmian u jej współpracowników. Musimy również wziąć pod uwagę *kontrakty usług* oraz *odpowiedzialność usług*.

KONTRAKTY USŁUG

Komunikaty akceptowane przez każdą usługę i zwracane przez nią odpowiedzi tworzą kontrakt między tą usługą a usługami, które na niej polegają, a które możemy nazwać *współpracownikami wyższego poziomu* (*nadrzędnymi*). Kontrakty umożliwiają traktowanie każdej usługi jako czarnej skrzynki przez jej współpracowników – wysyłamy żądanie i otrzymujemy coś z powrotem. Jeśli dzieje się to bezbłędnie, usługa robi to, co ma robić.

Chociaż wdrożenie usługi może się z czasem zmieniać, utrzymanie zgodności na poziomie kontraktu zapewniają dwie rzeczy:

- zmiany są tak nieznaczne, że nie powodują braku zrozumienia wśród konsumentów;
- zależności między usługami można jednoznacznie określić i nimi zarządzać.

Z naszego doświadczenia wynika, że kontrakty często są zawierane w naiwnych lub wczesnych wdrożeniach mikroserwisów; są sugerowane przez dokumentację i praktykę, a nie wyraźnie skodyfikowane. Wraz ze wzrostem liczby usług można osiągnąć znaczną korzyść dzięki standaryzacji interfejsów między nimi w formacie zrozumiałym dla maszyn. Na przykład REST API może korzystać ze Swagger/OpenAPI. Poza dodaniem testów zgodności poszczególnych usług, publikowanie standardowych kontraktów pomaga zrozumieć inżynierom w firmie, jak korzystać z dostępnych usług.

ODPOWIEDZIALNOŚĆ USŁUG

Na rysunku 2.6 widać, że usługa zleceń ma dużą odpowiedzialność. Bezpośrednio zarządza działaniami każdej innej usługi związanej z procesem składania zlecenia. Jest to koncepcyjnie proste, ale ma wady. W najgorszym przypadku nasze pozostałe usługi staną się anemiczne, a wiele z nich będzie kontrolowanych przez niewielką liczbę inteligentnych usług, przez co te inteligentne usługi będą stawać się coraz większe.

Takie podejście może prowadzić do ścisłego powiązania. Jeśli będziemy chcieli wprowadzić nowy etap tego procesu – powiedzmy, że będziemy chcieli powiadomić menedżera rachunku klienta o złożeniu dużego zlecenia – będziemy musieli wdrożyć nowe zmiany w usłudze zleceń. Zwiększy to koszt zmiany. Teoretycznie, jeśli usługa obsługi zleceń nie będzie musiała synchronicznie potwierdzać wyniku działania – a tylko potwierdzić otrzymanie żądania – nie powinna potrzebować żadnej wiedzy o tych dalszych krokach.

2.3.3. Choreografia usługi

W aplikacji mikroserwisowej usługi będą oczywiście miały różne poziomy odpowiedzialności. Ale musimy zrównoważyć orkiestrację z *choreografią*. W systemie choreograficznym usługa nie musi bezpośrednio sterować i uruchamiać działań w innych serwisach.