Aplikacja składa się z trzech warstw:

* warstwy aplikacji klienckiej napisanej w języku TypeScript z wykorzystaniem środowiska Angular
* warstwy serwerowej zbudowanej w oparciu o architekturę mikrousług, składającą się z 3 niezależnych mikroserwisów
* Warstwy danych. W tym przypadku wykorzystano bazę danych MongoDB dostępną w modelu SaaS.

Aplikacja kliencka komunikuje się z warstwą serwerową za pomocą interfejsu REST API, natomiast komunikacja pomiędzy poszczególnymi mikrousługami może być zrealizowana na dwa sposoby: synchronicznie – z wykorzystaniem interfejsu REST API oraz asynchronicznie z wykorzystaniem brokera wiadomości RABBIT MQ.

1. Na kolejnym slajdzie zaprezentowano wykorzystane w aplikacji technologie.

Technologie podzielono na 3 grupy, tj. technologie wykorzystane w implementacji mikroserwisów: m.in. język Java, biblioteki Spring Boot oraz bibliotek reaktywne Reactor.

Technologie wykorzystane do stworzenia aplikacji klienckiej, m.in. TypeScript, Angular oraz biblioteki reaktywne RxJS.

Oraz technologie wykorzystane do wdrożenia aplikacji czyli oprogramowanie do konteneryzacji Docker, oraz platforma Okteto umożliwiającą uruchamianie aplikacji z wykorzystaniem systemu do zarządzania kontenerami jakim jest Kubernetes.

--------

Poszczególne mikroserwisy zostały zbudowane z wykorzystaniem architektury warstwowej. Najwyższą warstwę stanowi tutaj warstwa API, na którą składają się punkty końcowe REST API oraz klasy odpowiedzialne za odbieranie i wysyłanie wiadomości asynchronicznych.

Środkową warstwę stanowi warstwa serwisów zawierających logikę biznesową.

Najniższą warstwę stanowią klasy odpowiedzialne za obsługę dostęp do bazy danych (repozytoria danych).

1. W celu umożliwienia logowania się użytkowników oraz zapewnienia bezpiecznego sposobu uwierzytelniania i autoryzacji wykorzystano zewnętrzną aplikację Keycloak która pełni funkcję dostarczyciela tożsamości użytkowników. Uwierzytelnianie i autoryzacja możliwa jest dzięki wykorzystaniu protokołu OAuth 2.0 oraz Open ID Connect.

Na slajdzie pokazano diagram sekwencji ilustrujący proces logowania użytkownika na który składa się m.in. uzyskanie tokenu JWT oraz wysłanie żądania zawierającego token w nagłówku.

1. Na kolejnym slajdzie przedstawiono diagram sekwencji dla jednego z wybranych przypadków użycia, którym jest konkretnie przypadek „Dodaj przychód/wydatek”.

W celu obsłużenia tego przypadku użycia jak widać na diagramie wykorzystane zostaną dwa mikroserwisy: Transaction management oraz Planning, a także aplikacja kliencka oraz broker wiadomości.

PREZENTACJA APLIKACJI

1. Na następnym slajdzie z kolei przedstawiono diagram przypadku użycia „Dodaj Depozyt”. W tym przypadku komunikacja pomiędzy mikroserwisami odbywa się nie tylko asynchronicznie, ale również synchronicznie. Jak widać na diagramie mikroserwis Asset management, wykonuje serie synchronicznych żądań do serwisu Planning w celu pobrania aktualnego stanu środków oraz sprawdzenia czy kategoria Asset jest utworzona (w przypadku jej braku wysłane zostanie żądanie utworzenia, co zostało pominięte na schemacie).

Następnie wykonywane jest synchroniczne żądanie do serwisu Transaction management w celu utworzenia transakcji. Serwis transaction Management z kolei w trakcie przetwarzania żądania wysyła asynchroniczną wiadomość do serwisu Planning, z informacją o szczegółach utworzonej transakcji.

1. W ramach pracy przygotowano również dokumentację punktów końcowych REST API. W tym celu wykorzystano bibliotekę Spring OpenAPI Webflux umożliwiającą automatyczne generowanie dokumentacji bezpośrednio z kodu punktów końcowych, z wykorzystaniem adnotacji które umożliwiają rozszerzanie dokumentacji czy tworzenie dodatkowych opisów.  
   Spring automatycznie tworzy dokumentację w formacie JSON a także udostępnia je online w formie graficznej.
2. W celu sprawdzenia poprawności działania poszczególnych funkcji zaimplementowano testy jednostkowe. W tym celu wykorzystano środowisko testowe Spock oraz język Groovy.

Przykładowy prosty test jednostkowy służący do testowania statycznej metody pokazany jest na slajdzie. Wykorzystano tutaj metodę DDA, pozwalającą na sprawdzenie wielu przypadków w jednej metodzie testowej.

Testy jednostkowe wykorzystano także do testowania bardziej skomplikowanych funkcji systemu wymagających zbudowania kontekstu aplikacji oraz stworzenia protez (mocków) m.in. dla metod które zwracają dane z zewnętrznych systemów.  
  
W aplikacji zastosowano także testy integracyjne których celem jest sprawdzenie prawidłowości współpracy aplikacji z zewnętrznymi systemami takimi jak bazy danych czy też zewnętrzne interfejsy API.

Aby przetestować łączność z zewnętrznymi systemami w tym przypadku z bazą danych MongoDB oraz brokerem wiadomości RabbitMQ skorzystano z bibliotek umożliwiających uruchamianie tych systemów w tymczasowych kontenerach testowych z wykorzystaniem Dockera.

1. Ostatnim etapem pracy było wdrożenie aplikacji. W tym celu skorzystano z platformy Okteto, która oferuje możliwość uruchamiania aplikacji z wykorzystaniem systemu Kubernetes. Platforma Okteto wspiera przy tym możliwość uruchamiania/wdrażania aplikacji z wykorzystaniem plików Docker Compose.   
     
   Na platformie Okteto uruchamiane są wszystkie aplikacje w tym aplikacja kliencka oraz 3 mikroserwisy. Oprócz tego na platformie Okteto uruchamiana jest aplikacja RabbitMQ.  
     
   Jak pokazuje diagram wdrożenia, baza danych znajduje się na zewnętrznym systemie. W tym przypadku skorzystano z bazy danych MongoDB w modelu Software as a Service.  
     
   Dodatkowo na serwerze zewnętrznym również z wykorzystaniem Docker Compose uruchamiana jest aplikacja Keycloak pełniąca funkcję dostarczyciela tożsamości.

Ostatecznym efektem pracy jest działający system o podstawowej funkcjonalności, który jednak dzięki modułowej budowie może być łatwo rozbudowany. Dzięki zastosowaniu architektury mikroserwisowej, programowanie reaktywnego, nierelacyjnej bazy danych oraz komunikacji asynchronicznej, system taki może być w łatwy sposób skalowalny, przez co nadaje się do obsługi ruchu o zróżnicowanym nasileniu.