**Opis projektu**

Niniejszy projekt stanowi aplikację do zarządzania zadaniami, która wykorzystuje szereg technologii i wzorców projektowych w celu stworzenia skalowalnego, czytelnego oraz łatwego w rozbudowie rozwiązania. Kluczowymi elementami są tutaj:

• **FastAPI** do obsługi logiki webowej i REST API,

• **DynamoDB** (w projekcie uruchamiane lokalnie w kontenerze Docker) jako główna baza danych NoSQL,

• **Celery** do obsługi zadań asynchronicznych, które są kolejkowane i uruchamiane w tle,

• **RabbitMQ** jako broker komunikatów dla Celery,

• **Redis** do przechowywania i zarządzania wynikami zadań (tzw. *backend*) lub dodatkowych danych,

• **AuthX** do autoryzacji użytkowników w oparciu o JWT,

• **Docker Compose** do orkiestracji wielu kontenerów (m.in. wspomniane RabbitMQ, Redis, DynamoDB oraz aplikacja FastAPI).

Dzięki takiemu podejściu deweloperzy otrzymują spójne i powtarzalne środowisko uruchomieniowe, w którym wszystkie usługi działają w izolowanych kontenerach, co ułatwia wdrożenie i utrzymanie aplikacji.

**Wzorce projektowe**

Aplikacja wykorzystuje kilka wzorców projektowych, aby zachować czytelność i modułowość:

1. **Wzorzec Repozytorium (Repository Pattern)**

Zastosowany w TaskRepository i UserRepository, które odpowiadają za interakcję z bazą danych DynamoDB. Umożliwia to czytelne oddzielenie warstwy dostępu do danych od logiki aplikacji.

1. **Wzorzec Obserwator (Observer Pattern)**

W module observers mamy klasy, takie jak SlackNotifier, OverdueNotifier, PriorityEscalationNotifier oraz ChangeHistoryObserver. Dziedziczą one po abstrakcyjnej klasie TaskObserver.

Dzięki temu, gdy wykonywana jest operacja na zadaniu (utworzenie, aktualizacja, oznaczenie jako ukończone), wywoływana jest metoda update(...) wszystkich obserwatorów. Pozwala to np. wysyłać powiadomienia Slackowe lub e-mail w odpowiednim momencie.

1. **Wzorzec Strategii (Strategy Pattern)**

Klasy takie jak StatusFilterStrategy, DueDateFilterStrategy, SortByCreatedAtStrategy i SortByPriorityStrategy odpowiadają za różne sposoby filtrowania i sortowania zadań.

Możliwe jest łączenie wielu strategii w ramach np. CompositeFilterStrategy, co daje dużą elastyczność w dopasowywaniu wyświetlanych danych do wymagań użytkownika.

1. **Wzorzec Decorator (w formie dekoratora w Pythonie)**

W app/clients/dynamo\_client.py znajduje się dekorator dynamo\_error\_handler.

Z punktu widzenia wzorców GoF, to nieco inna implementacja niż klasyczny Decorator (tam w wersji obiektowej „opakowujemy” obiekt w inny obiekt), ale w Pythonie dekoratory funkcyjne często pełnią zbliżoną rolę.

Dekorator ten dodaje dodatkową funkcjonalność (obsługę wyjątków) do wybranych metod w klasie DynamoDBClient, bez ingerencji w ich oryginalny kod.

W pewnym sensie przypomina to wzorzec Dekorator, bo metoda jest „opakowywana” w dodatkową logikę – tutaj w postaci funkcji wewnętrznej wrapper().

1. **Dependency Injection**

Choć Dependency Injection (DI) nie pochodzi z klasycznych wzorców GoF, jest to powszechnie stosowany wzorzec/inżynierska praktyka, która pomaga oddzielać zależności i ułatwia testowanie.

W FastAPI mechanizm Depends() występuje we wszystkich routerach (np. app/routers/task.py, app/routers/users.py) i pozwala wstrzyknąć gotowy serwis (TaskService, UserService) lub obiekt repozytorium czy bieżącego użytkownika do funkcji obsługującej żądanie HTTP.

Dzięki temu unikamy ręcznego tworzenia obiektów w każdym endpointzie i możemy konfigurować zależności w jednym miejscu (np. w app/services/utils.py).

**Technologie i uzasadnienie wyboru**

* **FastAPI**: Umożliwia szybkie budowanie usług webowych z wykorzystaniem Pythona i daje czytelny system obsługi endpointów oraz wbudowaną walidację Pydantic.
* **DynamoDB (lokalne uruchomienie)**: Baza NoSQL, która świetnie skaluje się i pozwala na utrzymywanie dużych wolumenów danych przy zachowaniu wysokiej przepustowości. W środowisku deweloperskim korzystamy z kontenera, aby unikać kosztów i skomplikowanych konfiguracji.
* **RabbitMQ i Redis** w połączeniu z **Celery**: Realizują mechanizmy kolejkowania zadań i ich asynchroniczną realizację (np. wysyłanie maili, notyfikacje Slack). Dzięki temu zadania długotrwałe nie blokują głównej obsługi aplikacji.
* **AuthX**: Dostarcza gotowe rozwiązania oparte na JWT (JSON Web Tokens), co usprawnia uwierzytelnianie i autoryzację użytkowników.
* **Docker & Docker Compose**: Pozwalają na spójne zarządzanie infrastrukturą aplikacji – nie trzeba instalować każdej usługi lokalnie. Mamy pewność, że środowisko produkcyjne i deweloperskie są do siebie maksymalnie zbliżone.
* **DevContainers (VS Code Remote – Containers)**: Za pomocą konfiguracji zawartej w folderze .devcontainer/ można błyskawicznie uruchomić cały stos technologiczny (łącznie z bazą, brokerem wiadomości i usługami asynchronicznymi) w zdefiniowanym kontenerze. Dzięki temu deweloper nie musi ręcznie konfigurować środowiska, co znacząco zwiększa spójność i skraca czas potrzebny na rozpoczęcie pracy z projektem.

**Podsumowanie funkcjonalności:**

Aplikacja oferuje szeroki zakres możliwości w zakresie zarządzania zadaniami i obsługi kont użytkowników. Przede wszystkim:

1. **Rejestracja i logowanie**

Użytkownicy mogą założyć konto, podając niezbędne dane (m.in. e-mail, hasło) i następnie zalogować się, by korzystać z pozostałych funkcjonalności aplikacji.

1. **Tworzenie nowych zadań**

Każdy zalogowany użytkownik może dodać zadanie do systemu, określając tytuł, opis, priorytet (1–5) oraz termin ukończenia. Tak stworzone zadanie powiązane jest z kontem użytkownika i widoczne tylko dla niego.

1. **Podgląd i aktualizacja zadań**
   1. Lista zadań: Użytkownik może wyświetlić wszystkie swoje zadania, a także skorzystać z mechanizmów filtrowania i sortowania (np. po priorytecie lub dacie utworzenia).
   2. Edycja: Można zmieniać najważniejsze parametry zadania, takie jak tytuł, opis, status czy priorytet.
2. **Oznaczanie zadania jako ukończone**
   1. Gdy użytkownik uzna, że zadanie jest zrealizowane, może jednym kliknięciem (lub wywołaniem właściwego endpointu) oznaczyć je jako „completed”.
3. **Usuwanie zadań**
   1. W dowolnym momencie można trwale usunąć zadanie, jeśli nie jest już potrzebne.
4. **Powiadomienia**
   1. Powiadomienie o nadchodzącym terminie: Jeżeli data realizacji zadania jest blisko, aplikacja może automatycznie (w tle) wygenerować powiadomienie, np. e-mail z przypomnieniem.
   2. Powiadomienia Slack: W zależności od konfiguracji (np. integracji z webhookiem), status zadania może zostać przesłany do kanału Slackowego (np. ukończenie ważnego zadania).
5. **Zarządzanie kontem**
   1. Edycja profilu: Można zmieniać takie dane jak nazwa użytkownika, imię i nazwisko, a w razie potrzeby hasło.
   2. Role i uprawnienia (dla administratorów): Aplikacja umożliwia nadawanie i odbieranie ról, dzięki czemu niektóre akcje (np. usuwanie użytkowników) są dostępne tylko dla osób z odpowiednim poziomem uprawnień.