Proof of Concept

Aplikacja webowa napisana w Golang z grafową bazą danych Neo4J oraz prostą aplikacją użytkownika realizującą operacje CRUD napisaną w technologiach JQuery/AJAX.

Autor:
Dawid Chara

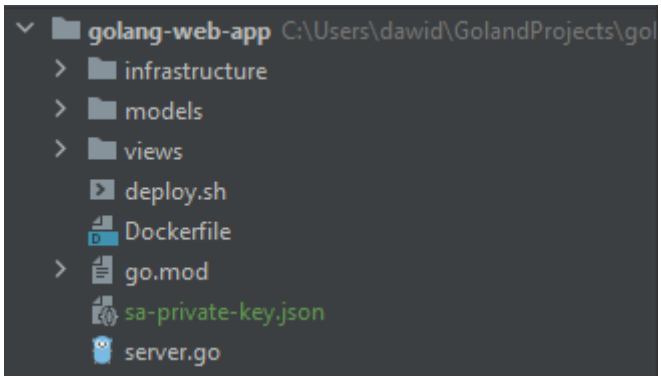
Repozytorium z kodem źródłowym:
https://github.com/dannyxn/golang-web-app

Link do serwisu udostępnionego w ramach usługi Cloud Run w Google Cloud Platform:
https://golag-neo4j-webapp-kxmtmt4m3q-lm.a.run.app

Spis treści

1. Infrastruktura aplikacji

Aplikacja składa się z kilku komponentów, które całościowo tworzą kompletne rozwiązanie, poniżej zdjęcie wycinka ekranu prezentujące katalog główny projektu.



Rysunek 1 Struktura projektu

W katalogu **infrastructure** znajdują się pliki konfiguracyjne narzędzie terraform opisujące infrastrukturę chmurową potrzebną do realizacji aplikacji. Katalog **models** służy do przechowywania plików źródłowych modeli – czyli struktur służących do operacji na danych aplikacji. **views** przechowuje kod źródłowy widoków aplikacji oraz folder **template**, który przechowuje szablon HTML z aplikacją kliencką, służącą do wykonywania zapytań do interfejsu eksponowanego przez aplikację. W katalogu głównym znajduję się skrypt **deploy.sh**, służący do realizowania operacji CI/CD stworzonej aplikacji, **Dockerfile** plik opisujący sposób tworzenia obrazu aplikacji, **go.mod** – plik aplikacji Golang zawierający dependencje używane przez moduł, **sa-private-key.json** (plik nie udostępniony w kodzie źródłowym

przesyłanym jako rozwiązanie) – klucz prywatny konta usług używanego do operacji na zasobach chmurowych, dzięki temu rozwiązaniu możliwa jest sprawna autoryzacja narzędzi wdrożeniowych, **server.go** zawiera rdzeń aplikacji, połączenie widoków z adresami URL, połączenie do bazy danych, pobieranie potrzebnych wartości tajnych z Google Secret Manager , oraz samo uruchamianie serwera http. W kolejnych punktach, powyższe komponenty zostaną opisane bardziej szczegółowo.

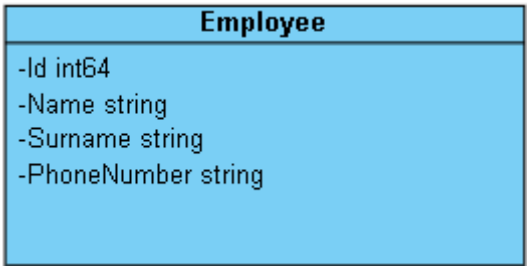
1) Kod serwera http

W tej sekcji zostanie opisany schemat uruchamiania serwera http aplikacji.

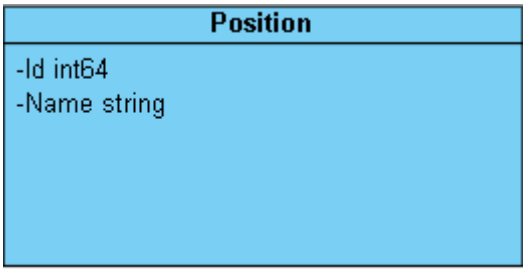
Najważniejszym elementem serwera jest funkcja **main**, to ona jest wykonywana jako pierwsza w całej aplikacji, na samym początku tej funkcji wywoływana jest metoda **initializeBackend** w której aplikacja za pomocą wcześniej wspomnianego pliku z kluczem prywatnym konta usług Google Cloud **sa-private-key.json** autoryzuje swoją obecność na platformie GCP, następnie pobiera najnowszą wartość sekretu **neo4j-golang-web-project-password**, czyli hasło do bazy danych stworzonych na potrzeby aplikacji. Po otrzymaniu hasła, aplikacja przystępuje do stworzenia nowego sterownika **neo4j.Driver** dzięki któremu będzie można tworzyć nowe sesje transakcyjne (lub nie) do bazy. Po pomyślnym stworzeniu sesji, aplikacja sprawdza czy można wykonać zapytanie powitalne i w zależności od tego loguje odpowiednią informację, pokazując administratorowi czy połączenie się powiodło. Jeżeli powiodła się inicjalizacja połączenia do bazy to następnie wskaźnik do sterownika połączenia jest przekazywana do widoków, zaś w samym kodzie serwera następuje powiązanie interfejsu adresów url ze stworzonymi widokami. Na samym końcu aplikacja uruchamia serwer nasłuchujący na porcie 8081 (nie widzimy tego ponieważ DNS platformy GCP steruje zapytaniami do adresu podanego na pierwszej stronie aplikacji).

2) Modele

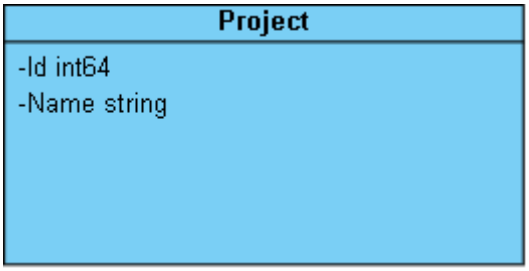
W aplikacji znajdziemy sześć modeli w tym trzy odpowiadające za operowanie na rodzajach węzłów, dwa odpowiadające za operacje na relacjach i jeden model pomocniczy który zostanie pominięty ze względu na wartość stricte implementacyjną. Modele zostaną przedstawione jako bloki diagramu klas.



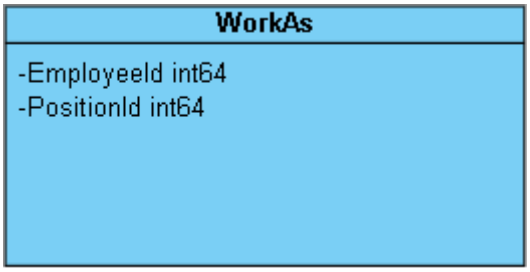
*Rysunek 2 Model **Employee***



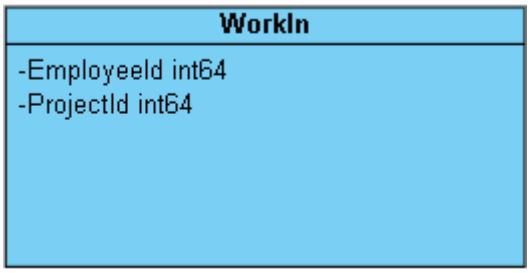
*Rysunek 3 Model **Position***



*Rysunek 4 Model **Project***



*Rysunek 5 Model **WorkAs***



*Rysunek 6 Model **WorkIn***

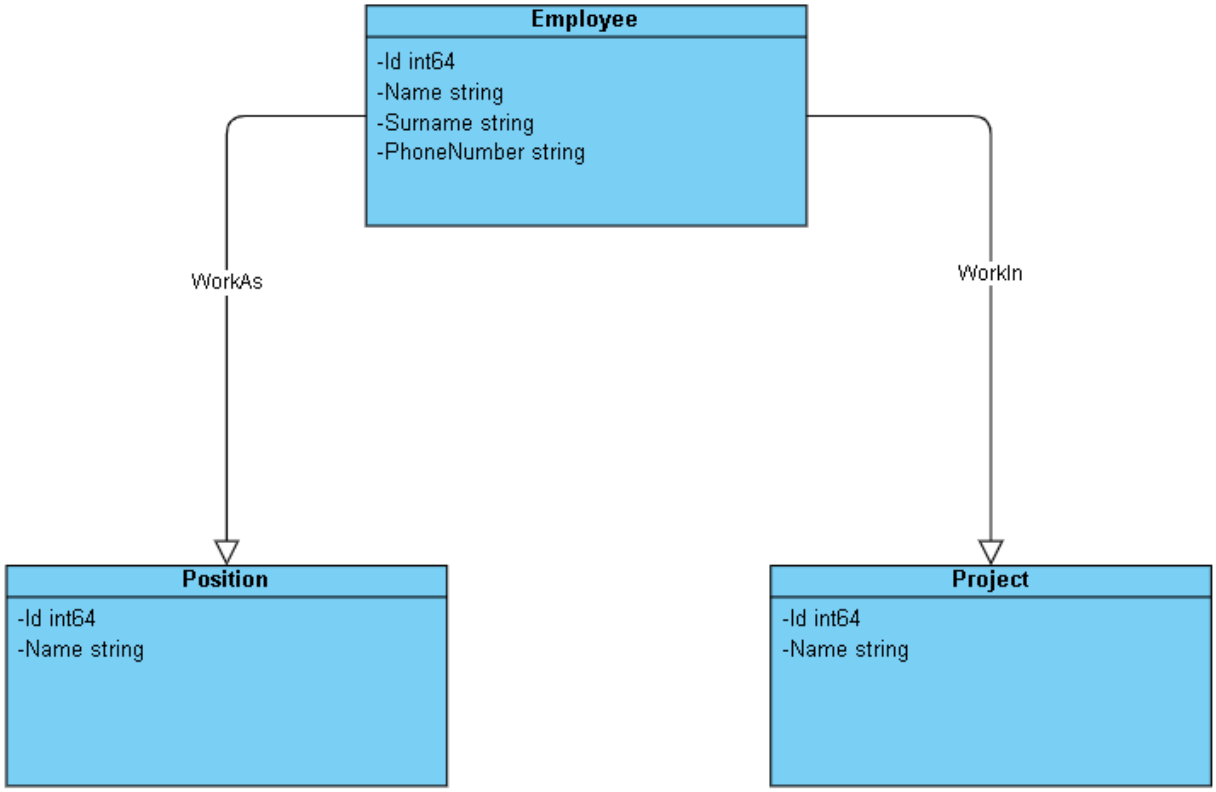
3) Widoki

Każdy model reprezentujący rodzaj węzła posiada zdefiniowane w widokach operacje CRUD oraz listowania, zaś każdy model reprezentujący relację posiada zdefiniowaną operację dodawania i listowania, pozostałe operacje na relacjach zostaną dodane w pełnej wersji (to tylko POC). Widoki oczekują otrzymania danych typu **json** dla operacji CREATE oraz PUT, dane muszą być zgodne z strukturą modeli, aczkolwiek zostało to zagwarantowane w aplikacji klienckiej, stąd użytkownik nie musi się zajmować np. dopisywaniem sztucznych indeksów do danych **json** zawierających dane nowego węzła do utworzenia.

4) Aplikacja kliencka

Aplikacja kliencka składa się pliku index.html ze zdefiniowanymi formularzami oraz odpowiednich przycisków gwarantujących wysłanie odpowiedniego zapytania do API udostępnionego przez serwer. Zapytania wysyłane są dzięki technologii JQuery/AJAX, funkcje obsługujące te zapytania zaimplementowane są w Javascript'cie zagnieżdżonym w pliku index.html.

2. Diagram bazy danych



Rysunek 7 Diagram bazy danych

3. Infrastruktura chmurowa

Przygotowaniem aplikacji do wdrożenia w chmurze zajmuję się skrypt deploy.sh, dzieli się on na dwie główne części **prepare_image** oraz **deploy**, owe funkcje są opisane w odpowiednio 1) i 2) punkcie.

1) Stworzenie obrazu kontenera i wdrożenie do Google Container Registry

Skrypt tworzy obraz kontenera na podstawie pliku **Dockerfile**, jednocześnie umieszczając w nim wszystkie potrzebne komponenty jak i kod źródłowy aplikacji, do uruchomienia jej w zamkniętym środowisku chmurowym. Po zbudowaniu obrazu jest on następnie wysłany do Google Container Registry, tak aby w drugim etapie wdrożenia, narzędzie terraform potrafiło go znaleźć i wykorzystać do uruchomienia konteneryzowanej aplikacji.

2) Uruchomienie aplikacji w usłudze Cloud Run

Drugą częścią wdrożenia aplikacji jest uruchomienie narzędzia terraform, tak aby zbudowało ono potrzebne zasoby chmurowe używając do tego obrazu stworzonego i wysłanego do Google

Container Registry w pierwszym etapie. Terraform tworzy różne zasoby potrzebne do realizacji wdrożenia, lecz najważniejszym z nich jest instancja zasobu Cloud Run, czyli usługi uruchamiającej konteneryzowane aplikacje. Cloud Run jest to tani i bardzo wygodny sposób na tworzenie tego typu aplikacji, koszty generowane są przez uruchomienia aplikacji (w tym przypadku wywołania zapytań do serwera uruchomionego przez aplikację), lecz w praktyce w tej skali projektu, koszty nie przekraczają darmowego limitu.