[SDPB] Wyznaczanie trasy

Dokumentacja końcowa projektu

Michał Aniserowicz, Jakub Turek

Temat projektu

Napisać aplikację wyznaczającą i porównującą trasę przejazdu na terenie Warszawy środkami komunikacji miejskiej i samochodem z opcją ustawienia godziny wyjścia z domu żeby zdążyć na czas. Aplikacja może być aplikacją mobilną lub przeznaczoną na komputer klasy PC.

Założenia

Temat został uszczegółowiony poniższymi założeniami:

- Aplikacja zaprojektowana w architekturze klient-serwer.
- Serwer posiada dwie odpowiedzialności:
  - Przechowuje dane.
  - Dostarcza logikę związaną z przeliczaniem tras.
- Klient jest aplikacją dostępową umożliwiającą wprowadzenie następujących danych:
  - Lokalizacja początkowa i docelowa (punkty na mapie).
  - Docelowy czas przyjazdu.
  - Przejazd komunikacją miejską lub samochodem.
- Klient na wyjściu prezentuje najszybszą trasę dla podanych danych wejściowych oraz godzinę wyjścia/wyjazdu, która pozwala zdążyć na czas.
- Dane nie są dostarczane z systemów zewnętrznych (np. *Google Maps*, *Jak dojadę*), ale składowane w bazie danych opracowanej w ramach projektu.
- Uproszczenie algorytmu wyznaczania trasy:
  - Założenie, że czas przejazdu tego samego odcinka drogi komunikacją miejską i samochodem jest identyczny.
  - Brak uwzględnienia informacji o godzinach przyjazdu środków komunikacji miejskiej na przystanki. Zakłada się, że czas wymagany na przesiadkę jest stały i jest parametrem algorytmu.

– Wyznaczana jest zawsze pojedyncza najszybsza trasa dla wybranego środka komunikacji.
– Jako punkt początkowy i końcowy wybierane są te lokalizacje spośród danych znajdujących się w bazie, które są najbliższe lokalizacjom wskazanym przez użytkownika.
– W przypadku wskazania komunikacji miejskiej jako środka transportu punktem początkowym i końcowym podróży są zawsze przystanki najbliższe wskazanym lokalizacjom. Nie jest uwzględniana możliwość dojścia do przystanku.

Model danych

Tradycyjny, oparty na tabelach model danych nie jest wydajny dla opisu struktury miasta. Struktura ta może być rozpatrywana jako zbiór punktów różnego typu (budynki, skrzyżowania, przystanki autobusowe). Punkty te mogą być ze sobą dowolnie połączone. Połączenie reprezentuje drogę, która umożliwia bezpośrednie przemieszczenie się pomiędzy punktami. Miasto można więc uprościć do przestrzennej struktury grafu planarnego.

Relacyjny model danych jest problematyczny w przypadku modelowania relacji wiele-do-wielu. Można wykorzystać dwie strategie:

- Tabela pośrednicząca, która przechowuje mapowanie kluczy obcych w relacji. Sprawdzanie relacji wymaga wykonania wielu złączeń.
- Denormalizacja. Na przykład dla struktury grafowej informację o wierzchołkach przechowuje się w tabeli z krawędziami. W tym przypadku narzut obliczeniowy na złączenia jest zastępowany narzutem pamięciowym, gdyż każdy wierzchołek jest opisany w bazie danych n razy, gdzie n to stopień tego wierzchołka.

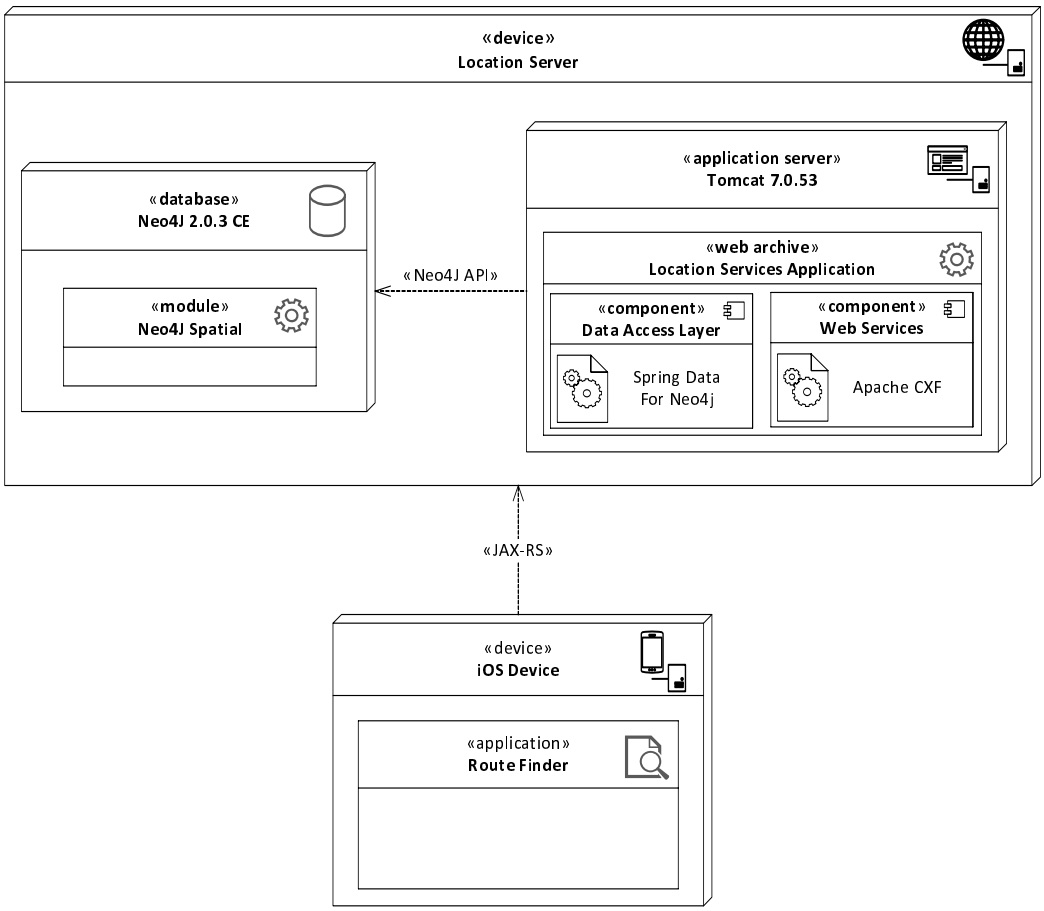
Interesującą alternatywę zapewnia ruch **Not Only SQL**, a konkretnie jego podzbiór - grafowe bazy danych. Ich wewnętrzny model danych jest spójny z przedstawioną w poprzednim akapicie koncepcją opisu struktury miasta. Jednym z najbardziej dojrzałych projektów grafowych baz danych jest **Neo4J** (`http://www.neo4j.org`). Zaletą tej bazy w kontekście projektu jest posiadanie warstwy abstrakcji do obliczeń przestrzennych - **Neo4J Spatial**. Ze względu na wymienione zalety i odpowiedniość struktury do dziedziny problemu, do implementacji projektu zostanie wykorzystana właśnie baza Neo4J.

Architektura aplikacji

Aplikacja została zaprojektowana w architekturze klient-serwer. Szczegółowy schemat architektury przedstawia rysunek 1.

Budowanie i uruchamianie aplikacji

Moduł serwerowy oraz kliencki aplikacji budowane są osobno. Część serwerowa jest budowana przy użyciu aplikacji Maven (`http://maven.apache.org`). Uruchomienie polecenia **mvn clean install** w ścieżce projektu spowoduje ściągnięcie wszystkich zależności,



Rysunek 1: Schemat architektury aplikacji.

uruchomienie testów jednostkowych oraz integracyjnych, a także wygenerowanie pliku web-archive (**.war**), który następnie trzeba *zdeployować* na serwer aplikacji Java.

Budowanie aplikacji klienckiej jest realizowane przy pomocy menadżera zależności CocoaPods (`http://cocoapods.org`). Uruchomienie polecenia **pod install** w podfolderze z klientem spowoduje ściągnięcie wszystkich zależności oraz wygenerowanie pliku projektu programu Xcode, za pomocą którego można skompilować i wyeksportować aplikację.