Projektowanie aplikacji mobilnych

"AirQuality - Odczyt jakości powietrza ze stacji pomiarowych"

Adam Talarczyk

Wydział Nauk Ścisłych i Technicznych
Uniwersytet Śląski
Semestr letni 2020/2020

Spis treści

1 Wstęp **2**
1.1 Założenia projektowe 2
1.2 Wymagania funkcjonalne 2
1.3 Wymagania niefunkcjonalne 2

2 Specyfikacja zewnętrzna **3**
2.1 Instrukcja obsługi 3

3 Specyfikacja wewnętrzna **4**
3.1 Struktura aplikacji 4
3.2 Wzorce projektowe 5
3.3 Biblioteki i API 5
3.3.1 Interfejs API portalu "Jakość Powietrza" GIOŚ 5
3.3.2 Framework MapKit 6
3.4 Kod źródłowy 6
3.5 Uruchomienie aplikacji w środowisku deweloperskim 6

4 Podsumowanie **7**
4.1 Trudności 7
4.2 Wnioski 7

Spis rysunków **8**

Literatura **9**

1 Wstęp

1.1 Założenia projektowe

Głównym założeniem aplikacji jest pobieranie danych ze stacji pomiarowych oraz wyświetlanie ich w czytelny dla użytkownika sposób. Aplikacja przewiduje wyszukiwanie stacji na mapie, dodawanie lub usuwanie ich z zakładek, oraz prezentacja wybranych stacji na stronie głównej. Dane dostarczane przez aplikację pochodzą z publicznego interfejsu programistycznego aplikacji (https://powietrze.gios.gov.pl/php/content/api), który udostępnia wyniki automatycznych pomiarów dwutlenku siarki (SO2), dwutlenku azotu (NO2), pyłu PM10, pyłu PM2,5, tlenku węgla (CO), benzenu (C6H6), ozonu (O3). Dodatkowo, gdy poziom zanieczyszczenia powietrza dla najbliższej stacji pomiarowej przekroczy normę, aplikacja poinformuje o tym użytkownika przy pomocy powiadomienia.

Aplikacja przeznaczona jest wyłącznie na smartfony firmy Apple. Napisana została w języku Swift przy użyciu narzędzia Xcode.

1.2 Wymagania funkcjonalne

- Wyświetlanie na mapie wszystkich stacji pomiarowych na terytorium polski
- Dodanie stacji pomiarowej do zakładek
- Usuniecie stacji pomiarowej z zakładek
- Wyświetlenie szczegółowych pomiarów wybranej stacji
- Wysyłanie powiadomień użytkownikowi
- Wyświetlanie najbliższej stacji pomiarowej w lokalizacji użytkownika
- Obsługa trybu nocnego

1.3 Wymagania niefunkcjonalne

- Aplikacja automatycznie odświeża dane o pełnej godzinie (wynika to ze sposobu działania API)
- Działanie aplikacji oraz czas pobierania danych ograniczone są do przepustowości interfejsu programistycznego
- Komunikacja z serwerem API odbywa się poprzez protokół HTTP
- Interfejs użytkownika zaprojektowany jest zgodnie z wytycznymi Apple

2 Specyfikacja zewnętrzna

2.1 Instrukcja obsługi

Użytkownik po uruchomieniu aplikacji i zaakceptowaniu komunikatu o usłuchach lokalizacji znajduje się na stronie głównej, na której na samym początku widzi tylko najbliższą w okolicy stację pomiarową. Na dolnym pasku nawigacji ma on do dyspozycji:

- Stronę główną (na której się znajduje)
- Mapę
- Ustawienia

Po kliknięciu w zakładkę mapy zostanie przeniesiony do widoku, gdzie na mapie zaznaczone punktami są wszystkie dostępne stacje pomiarowe. Użytkownik może oddalać, przybliżać oraz przesuwać się po mapie. Jego lokalizacja również jest uwzględniona (z dokładnością do 100 metrów). Po kliknięciu w punkt ze stacją otwarte zostają szczegóły stacji pomiarowej, gdzie znajdują się wszystkie informacje o sensorach oraz pomiarach. Użytkownik może dodać stację pomiarową do zakładek, wówczas na stronie głównej zaraz pod najbliższą w okolicy stacją zostają wyświetlone wszystkie stacje z zakładek.

Użytkownik może również wyszukać stację po jej nazwie (która zazwyczaj zawiera nazwę miasta). W tym celu trzeba wejść na stronę główną i w prawym górnym rogu kliknąć przycisk lupy. Otworzy się wówczas widok z listą wszystkich stacji pomiarowych (około 190). By zainicjować wyszukiwanie trzeba przejechać palcem od góry do dołu, wtedy wyświetli się pasek wyszukiwania, w którym wpisać można odpowiednią frazę. Wybór stacji, tak jak w przypadku zakładki mapy otworzy nam widok ze szczegółami.

Zakładka ustawienia na chwile obecną przechowuje informacje o autorze, w przyszłości jednak planowane jest zaimplementowanie ustawień lokalizacji i powiadomień.

3 Specyfikacja wewnętrzna

3.1 Struktura aplikacji

Szkielet aplikacji opiera się o kilka rodzajów klas, które spełniają różne zadania:

- Kontrolery widoków - odpowiedzialne za wyświetlanie danych zgodnie z ich przeznaczeniem, włączają się do tego klasy wyświetlające mapę, widok strony głównej oraz widok wyszukiwania stacji pomiarowych: `MapViewController.swift`, `HomeViewController.swift`,

  `SearchTableViewController.swift`

- Kontrolery logiki aplikacji - klasy, które realizują niezbędną logikę biznesową, czyli pobranie danych z odpowiedniego węzła końcowego API, przypisanie danych do odpowiednich struktur. W skład tej kategorii wchodzą klasy realizujące pobieranie danych:

  `StationsController.swift`

- Modele danych - odpowiednio przygotowane struktury danych, na których potem wykonywane są operacje. Modele zostały zaprojektowane na podstawie danych, jakie dostarcza API "Jakość Powietrza" GIOŚ. Zastosowane przeze mnie struktury danych: `StationModel.swift`,

  `CityModel.swift`, `CommuneModel.swift`, `SensorModel.swift`,

  `IndexModel.swift`, `ParamModel.swift` oraz `ValueModel.swift`.

Po uruchomieniu odpowiedniego widoku (np. kliknięcie w zakładkę mapy) kontroler widoku inicjalizuje kontroler logiki, który zwraca odpowiednie dane "zapakowane" w odpowiednią strukturę danych. Ponieważ dane z API zwracane są w formacie JSON, kontroler ma za zadanie zdekodować je oraz ustawić ich odpowiedni format. Posiadając komplet danych, kontroler widoku jest w stanie wyświetlić na ekranie smartfona gotowy widok. Dodatkowo, aplikacja posiada odpowiedni folder na zasoby (grafiki, ikony, itp.) o nazwie `Assets.xcassets`, plik konfiguracyjny `info.plist` (property list - który swoją składnią przypomina pliki XML) oraz specjalną tablicę do projektowania widoków oraz interakcji między nimi `Main.storyboard`.

Z powyżej listy interesującą pozycją może wydawać się plik konfiguracyjny, ponieważ do poprawnego działania aplikacji wymagane było dodanie następujących kluczy:

- "Privacy - Location When In Use" - komunikat podczas żądania lokalizacji od użytkownik

- "App Tranport Security Settings", "Exception Domains", "NSIncludesSubdomains", "NSExceptionAllowInsecureHTTPLoads" - biała lista adresów, z którymi aplikacja może się komunikować - bez tego klucza nie istaniała by możliwość by pobrać dane z publicznego API.

3.2 Wzorce projektowe

W programowaniu aplikacji mobilnych w języku Swift najbardziej popularnym wzorcem projektowym są delegacje ("Delegate pattern"). Delegacja jest wzorcem z programowania obiektowego, w którym obiekt zamiast sam wykonać pewną operację deleguje ją (zleca do wykonania) innemu obiektowi pomocniczemu (delegatowi). Następuje tu więc odwrócenie odpowiedzialności. Delegat jest odpowiedzialny za wykonanie zadania obiektu delegującego. Delegacje realizują podobne założenia do dziedziczenia jednak stosuje się je w miejscach, gdzie dziedziczenie może być niewygodne w implementacji.

3.3 Biblioteki i API

3.3.1 Interfejs API portalu "Jakość Powietrza" GIOŚ

Interfejs API portalu "Jakość Powietrza" GIOŚ umożliwia dostęp do danych dotyczących jakości powietrza w Polsce, wytwarzanych w ramach Państwowego Monitoringu Środowiska i gromadzonych w bazie JPOAT2,0

Dostęp do danych odbywa się poprzez zapytanie HTTP GET na podany w dokumentacji adres. Udostępniane dane zwracane są w formiacie JSON. Interfejs udostępnia dane o stacjach pomiarowych, stanowiskach pomiarowych (lista dostępnych stanowisk pomiarowych na wybranej stacji pomiarowej), dane pomiarowe oraz indeks jakości powietrza.

Lista adresów potrzebna do pobrania danych:

- `http://api.gios.gov.pl/pjp-api/rest/station/findAll` - lista wszystkich stacji pomiarowych
- `http://api.gios.gov.pl/pjp-api/rest/station/sensors/{stationId}` - lista dostępnych sensorów dla stacji pomiarowej na podstawie identyfikatora stacji
- `http://api.gios.gov.pl/pjp-api/rest/data/getData/{sensorId}` - dane pomiarowe dla sensora o podanym w adresie id
- `http://api.gios.gov.pl/pjp-api/rest/aqindex/getIndex/{stationId}` - indeks jakości powietrza na podstawie identyfikatora stacji pomiarowej

3.3.2 Framework MapKit

Narzędzia pozwalające na wyświetlanie map wraz ze wszystkimi niezbędnymi elementami, takimi jak adnotacje, punkty, nakładki.

Implementacja wymaga sprawdzenia, czu urządzenie użytkownika posiada włączone usługi lokalizacyjne oraz czy użytkownik nadał uprawnienia do pobierania jego lokalizacji.

3.4 Kod źródłowy

Komentarze do kodu źródłowego (mogą być wygenerowane automatycznie)

3.5 Uruchomienie aplikacji w środowisku deweloperskim

Do uruchomienia aplikacji wymagany jest komputer firmy Apple z zainstalowaną aplikacją Xcode. Po ściągnięciu i otwarciu projektu wystarczy wybrać urządzenie z listy na górnym pasku a następnie wcisnąc przycisk odpowiedzialny za uruchomienie. Jeżeli wybrane urządzenie nie jest fizycznym smartfonem, uruchomiony zostanie emulator, który w ciągu kilku minut zainicjalizuje naszą aplikację. Warto zwrócić uwagę, że korzystając z emulatora można doświadczać problemów z poprawnym wyświetlaniem lokalizacji użytkownika, dlatego w celu przetestowania aplikacji sugerowane jest użycie fizycznego urządzenia.

4 Podsumowanie

4.1 Trudności

Podczas tworzenia aplikacji napotkałem wiele trudności i problemów, które wymagały zasięgnięcia porady w internecie. Niektóre z tych problemów wynikają ze specyfiki języka Swift (który bazuje na Objective-C) lub z nawyków, które nabyłem programując w innych technologiach.

Język Swift posiada bardzo specyficzną składnię czerpiącą wiele z nowoczesnych języków. Ciężko było przywyknąć mi do nietypowych deklaracji zmiennych, np. `let` oraz `var`, które w jęzuku JavaScript nie różnią się za bardzo od siebie - tutaj jednak, ta pierwsza definiuje nam stałą wartość, której nie można już później zmienić, druga definicja to deklaracja zwykłej zmiennej. Ciekawą rzeczą, którą poznałem jest również wyrażenie
`guard [wyrażenie] else { kod aplikacji }`. Jest to nic innego jak `if`, który wykona kod w klamrach tylko wtedy, gdy wartość wyrażenia przyjmie fałsz. Będąc przy temacie składni nie może zabraknąć informacji o tzw. "optional chaining", które w kwietniu 2020 dopiero zostało wprowadzone w języku JavaScript. Polega to na tym, że nie musimy każdorazowo sprawdzać instrukcją `if` czy nasza zmienna, do której chcemy się odnieść istnieje Wykorzystuje się do tego znak zapytania. Pozwala to zredukować ilość nadmiarowego kodu. Przykład:
Wyrażenie `let sensors = obj.station && obj.station.sensors`
Możemy zastąpić poprzez: `let sensors = obj.station?.sensors`
Jest to ułatwienie, które w ostatnim czasie jest implementowane również w innych językach.

Obsługa asynchronicznych funkcji też znacznie różni się od tych, do których przyzwyczaił mnie JavaScript. Pobierając dane z API oczekuje, że widok wygeneruje się dopiero po tym jak otrzymamy odpowiedź. We wspomnianym wcześniej JavaScripcie możliwe jest to za pomocą wyrażeń `async` oraz `await`. Tutaj niestety sytuacja jest dużo trudniejsza. W języku Swift wymagane jest zaimplementowanie funkcji `completion`. Oficjalna dokumentacja określa to jako "funkcje domknięcia (closures)" i przyrównuje je do bloków w Objective-C lub wyrażeń lambda w języku Java

Ostatnią trudnością w programowaniu aplikacji na iOS jest fakt, że można je tworzyć tylko na urządzeniach Apple. Według mnie praca na laptopie i małym ekranie jest dużo bardziej męcząca niż wykonywanie tego samego zadania na komputerze stacjonarnym.

4.2 Wnioski

1. API GIOŚ które nie jest rest 2. Łatwy w użyciu debugger, który nie wymaga konfiguracji 4. Swift skupia się bardzo na delegatach w sposób, w który nie był mi dotąd znany

Spis rysunków

Literatura

[1] https://developer.apple.com/documentation/mapkit/

[2] https://developer.apple.com/documentation/corelocation/cllocationmanager

[3] http://programowanie.siminskionline.pl/resource/wp/ood_strategy.pdf