Politechnika Poznańska
Wydział Informatyki i Telekomunikacji
Instytut Informatyki

Bank Scrapper - dokumentacja techniczna

Marcin Jaskulski, 136560
Paweł Pytel, 136786
Szymon Michalak, 136769

Pod opieką:
prof. dr hab. inż. Robert Wrembel
Mariusz Sienkiewicz - PKO BP

Poznań, 2021

Opis problemu

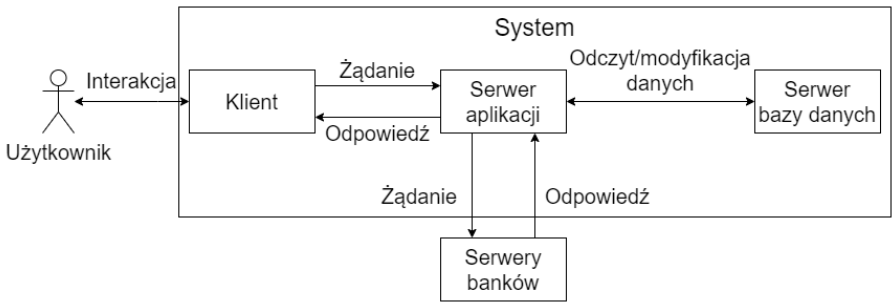
Placówki banków obsługują stałych klientów, ale są też ważnym elementem pozyskiwania nowych. Okazuje się, że wpływ na skuteczność tych działań mają godziny otwarcia tych placówek. Przykładowo potencjalny klient może iść ulicą wcześnie rano i wybrać ten bank, którego placówka jest otwarta. Jak zatem wybierać godziny otwarcia danej placówki? Zdaje się, że należałoby zwrócić uwagę na godziny otwarcia konkurencji. Pracownicy zajmujący się wyznaczaniem godzin dla danej placówki, aby zdobyć informacje o konkurencji, musieliby wejść na stronę każdego banku z osobna i w wyszukiwarce placówek wpisać odpowiedni adres, a następnie przepisać wyszukane dane z poszczególnych stron w jedno miejsce, co zajmowałoby zdecydowanie zbyt wiele czasu. Aby rozwiązać ten problem, powstał system, który pobiera informacje o godzinach otwarcia placówek różnych banków, a następnie dla danej placówki wyświetla godziny otwarcia konkurencji w zadanym sąsiedztwie. Dzięki temu pracownik w kilka sekund jest w stanie uzyskać interesujące go informacje i na ich podstawie wyznaczyć godziny otwarcia placówki.

Powstały system został zrealizowany na potrzeby banku PKO BP w ramach przedmiotu Hurtownie Danych i Przetwarzanie Analityczne. Przedmiot ten jest częścią kształcenia na specjalności Technologie Przetwarzania Danych na Politechnice Poznańskiej.

Architektura

Architektura systemu

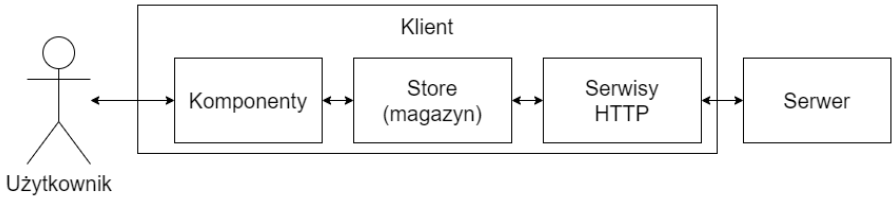
System został zaimplementowany w architekturze trójwarstwowej. Schemat architektury przedstawiony jest na rysunku nr 1. Klient wchodzi w bezpośrednią interakcję z użytkownikiem i na podstawie akcji wykonywanych przez użytkownika wysyła do serwera aplikacji odpowiednie żądania. Następnie serwer aplikacji komunikuje się z serwerem bazy danych, aby pobrać informacje potrzebne do zbudowania odpowiedzi i wysyła odpowiedź do klienta, a zadaniem klienta jest przekazanie odpowiedzi użytkownikowi w sposób czytelny i zrozumiały. Dodatkowo serwer aplikacji komunikuje się z serwerami wyszukiwarek poszczególnych banków, aby zdobyć informacje o placówkach. Dane zebrane podczas scrappingu przechowywane są na serwerze bazy danych i to na ich podstawie tworzone są odpowiedzi wysyłane do klienta.



Rysunek 1: Architektura systemu

Architektura klienta

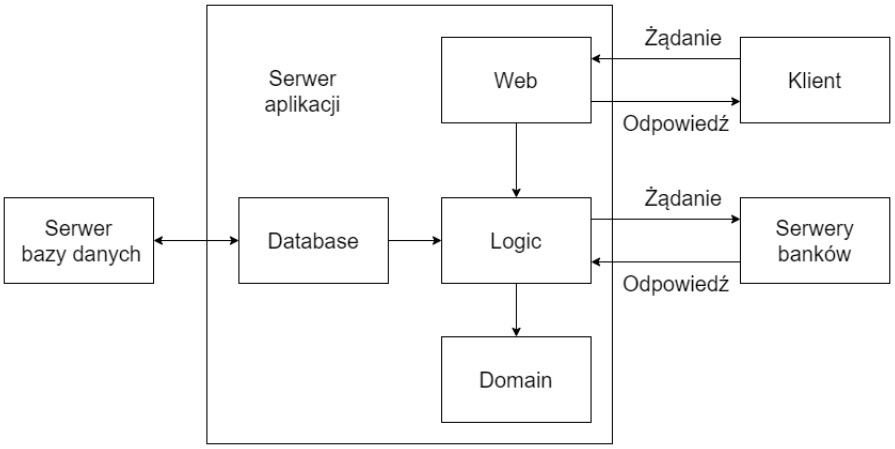
Architektura aplikacji klienckiej przedstawiona jest na rysunku nr 2. Część frontend-owa systemu składa się z trzech głównych elementów. Komponenty odpowiadają za wyświetlanie danych na ekranie, Służą do interakcji między komputerem, a użytkownikiem. Dzięki nim jest możliwość korzystania z aplikacji. Komponenty komunikują się ze Store. Jest to magazyn, który przechowuje aktualny stan aplikacji. Dane są tam przechowywane i jeśli nadejdzie żądanie z komponentu są przesyłane i wyświetlane na ekranie. Kolejnym elementem jest serwis HTTP. Służy on do połączenia aplikacji klienckiej i serwera z danymi. To tutaj tworzone są żądania HTTP, a odpowiedz z serwisu zapisywana jest w magazynie.



Rysunek 2: Architektura klienta

Architektura serwera aplikacji

Rozwiązanie serwerowe jest podzielone na kilka części opakowanych w osobne projekty. Jest to widoczne na rysunku nr 3. Projekt Domain opisuje fragment rzeczywistości istotnej dla systemu w postaci modeli. Projekt Logic odpowiedzialny jest za logikę biznesową: zarówno za przygotowywanie danych dla projektu Web, jak i za web scrapping. Projekt Database obsługuje komunikację z serwerem bazodanowym, a projekt Web wystawia dla klienta REST API.



Rysunek 3: Architektura serwera

Wykorzystywane technologie

ASP.NET Core 5.0 i Entity Framework 5.0

Do stworzenia API został wykorzystany wysokopoziomowy język C# i powiązana z nim platforma ASP.NET 5.0. Było to podyktowane znajomością technologi przez zespół programistyczny. Takie rozwiązanie pozwala na hostowanie Systemu zarówno na serwerach z zainstalowanym systemem operacyjnym Windows, jak i Linux.

Komunikacja z bazą danych została zrealizowana przy pomocy Entity Framework 5.0 (EF). Zapewnia on wsparcie dla podejścia Code First przy tworzeniu bazy danych. Na szczególną uwagę zasługuje wbudowany w framework mechanizm migracji pozwalający na modyfikację istniejącej bazy przy zachowaniu spójności dla całego zespołu deweloperskiego. Entity Framework daje również bezpośrednią możliwość tworzenia zapytań na obiektach. Takie zapytania są później bezpośrednio optymalizowane i transferowane na zapytania wykonywane bezpośrednio na bazie.

Microsoft SQL Server

Do przechowywania danych wykorzystano system zarządzania bazą danych Microsoft SQL Server. Znajomość technologii nie jest wymagana przy wykonywaniu prac nad Systemem, ponieważ wszystkie operacje są realizowane przy pomocy wcześniej wspomnianego EF.

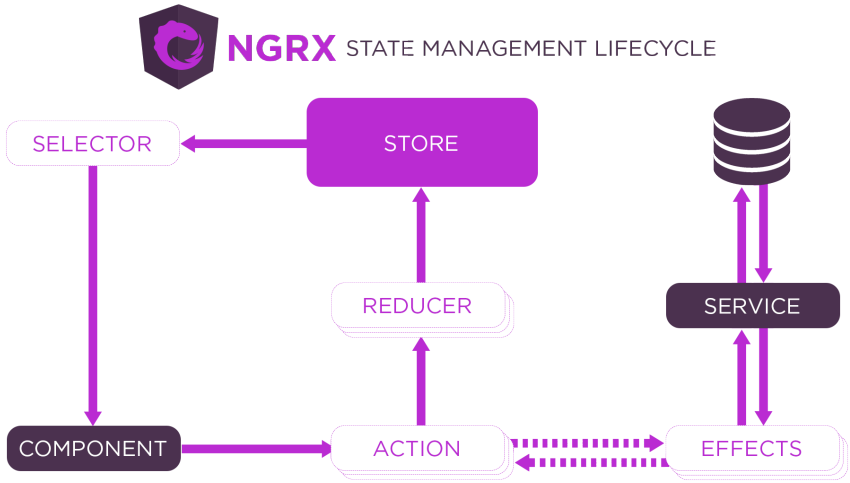
Angular 10

Frontend został wykonany przy pomocy frameworku Angular w wersji 10. Pozwala on na tworzenie SPA (Single Page Aplication). Cały widoczny interfejs składa się z komponentów, które są renderowane w odpowiednim miejscu po zadanej akcji.

Pojedynczy element składa się z trzech części:

- Plik HTML - szablon komponentu wykorzystujący hipertekstowy język znaczników
- Plik CSS - kaskadowy arkusz stylu pozwalający na wizualne stylizowanie
- Plik TS - pozwala dokonywać operacji na danych, które następnie są dynamicznie aktualizowane i wyświetlane

Dodatkowo frontend Systemu wyposażona jest w managera stanu, opartego o Redux Pattern. Jego obsługa jest realizowany przy pomocy biblioteki NgRx. Schemat jej działania został przedstawiony na Rysunku 4



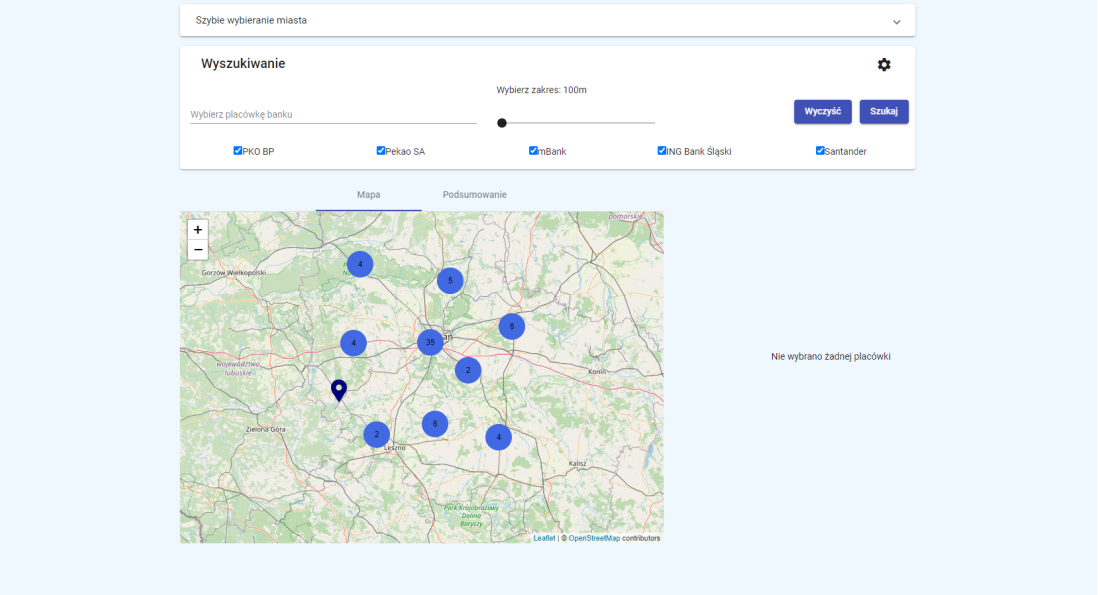
Rysunek 4: Zarządzanie stanem przy pomocy biblioteki NgRx

- **Store** - magazyn przechowujący aktualny stan Systemu, czyli wszystkie informacje, które zostały pobrane z bazy danych lub wygenerowane przez użytkownika. Są one gotowe do wyświetlenia, gdy nadejdzie taka potrzeba,
- **Selector** - element pobierający dane z konkretnego pola magazynu. Przekazuje dane do komponentu,
- **Component** - komponent zgłasza żądanie o dane i pobiera je. Następnie odpowiednio formatuje lub edytuje i wyświetla użytkownikowi,
- **Action** - akcje mają dwa główne zastosowania: przekazują dane z komponentu do magazynu (aktualizacja stanu) lub komunikują się z bazą danych (pośrednio),
- **Reducer** - dane które dostanie od akcji są przypisywane do magazynu,
- **Effect** - jest pewnego rodzaju rozgałęzieniem i sterownikiem. Decyduje którędy mają poruszać się dane pomiędzy frontend-em, a backend-em. Z jednej strony wchodzi akcja z parametrami do zapytania HTTP, a następnie gdy informacje zostaną pobrane decyduje, w jakie pole magazynu należy je zapisać,
- **Service** - serwis służy do wykonywania podstawowych operacji HTTP, takich jak GET, POST czy DELETE.

Do tworzenia widoków został również wykorzystany framework Bootstrap oraz Angular Material. Są to biblioteka zawierająca gotowe komponenty wizualne np. wyskakujące okienka, zakładki, rozszerzane panele czy gotowe stylizacje elementów. Widok mapy został osiągnięty dzięki bibliotece Leaflet.js. Jest to gotowy komponent, który posiada operacje takie jak dodawanie markerów, klastrowanie ich czy dodawanie okręgu do zaznaczania obszaru.

Interfejs

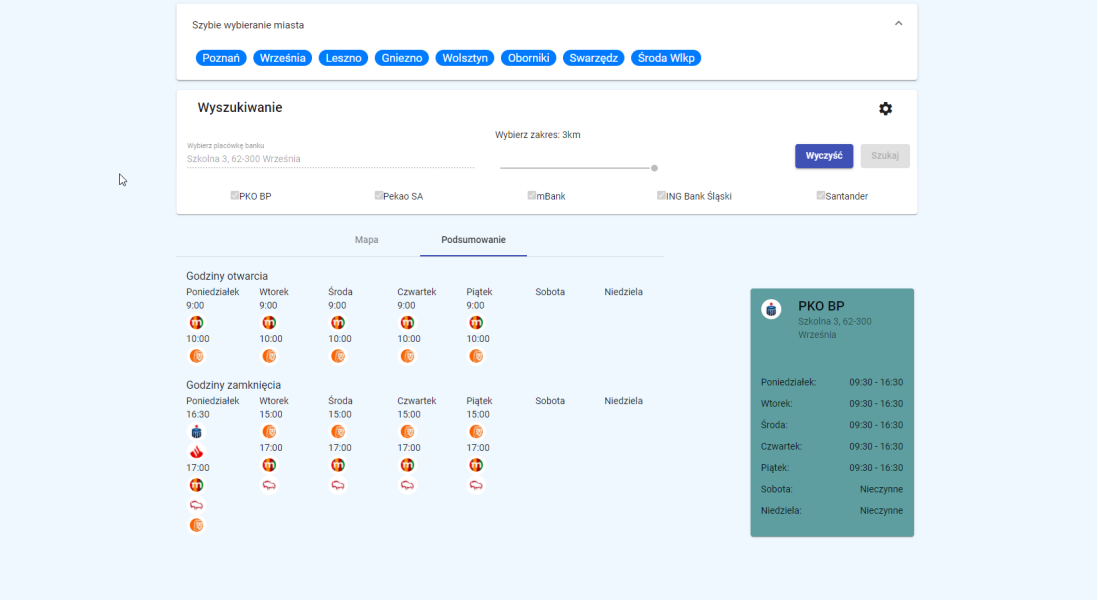
Zgodnie z wymaganiami postawionymi przez przedstawiciela banku PKO BP interfejs użytkownika został wyposażone w mapę. Podczas uruchomienia strony na tę mapę nanoszą się markery odpowiadające lokalizacjom placówek banku, dla którego chcemy zrobić wyszukanie. Placówkę można wybrać na dwa sposoby: albo klikając marker na mapie, albo używając pola tekstowego, gdzie należy wpisać adres - ulica, kod pocztowy czy miasto. Po wybraniu placówki po prawej stronie pojawiają się informacje odnośnie jej godzin otwarcia. Następnie należy wybrać zakres, dla którego chcemy zrobić wyszukanie (używając suwaka) oraz używając checkbox-ów wybrać banki konkurencji, które chcemy sprawdzić.



Rysunek 5: Główny widok aplikacji klienckiej

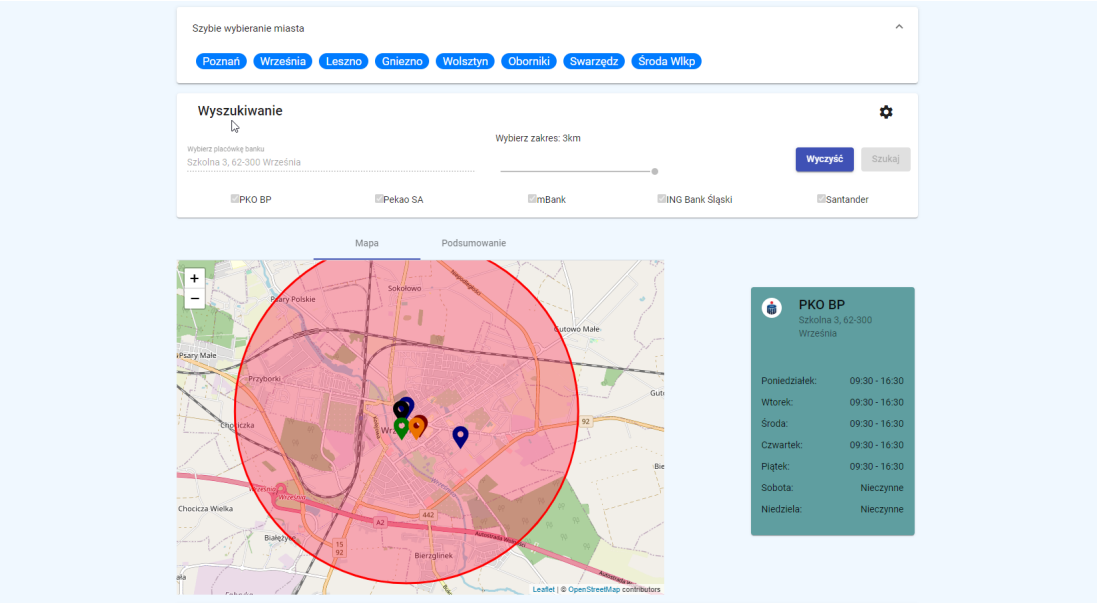
Po wybraniu placówki, zakresu i banków konkurencji można przejść do szukania informacji. W tym celu należy użyć przycisku "Szukaj". Gdyby brakowało któregoś z parametrów użytkownik zostanie poinformowany o tym przy pomocy powiadomienia u dołu ekranu.

Po wyszukaniu informacji widok mapy zostaje przełączony na podsumowanie.



Rysunek 6: Widok podsumowania

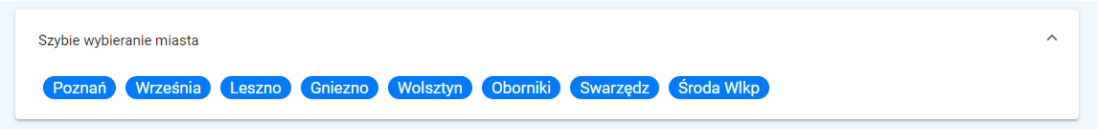
Można stąd odczytać w jakim przedziale czasowym otwierają i zamykają swoje oddziały banki w poszczególnych dniach. Na podstawie tego można przeanalizować, czy opłaca się wydłużyć godziny pracy czy też nie. Po przełączeniu z powrotem na widok mapy ukazują się kolorowe markery (jeden kolor odpowiada jednemu bankowi).



Rysunek 7: Widok mapy z wyszukaniem

Po najechaniu kursorem na marker pojawia się dodatkowe okienko z informacjami o godzinach otwarcia palcówki.

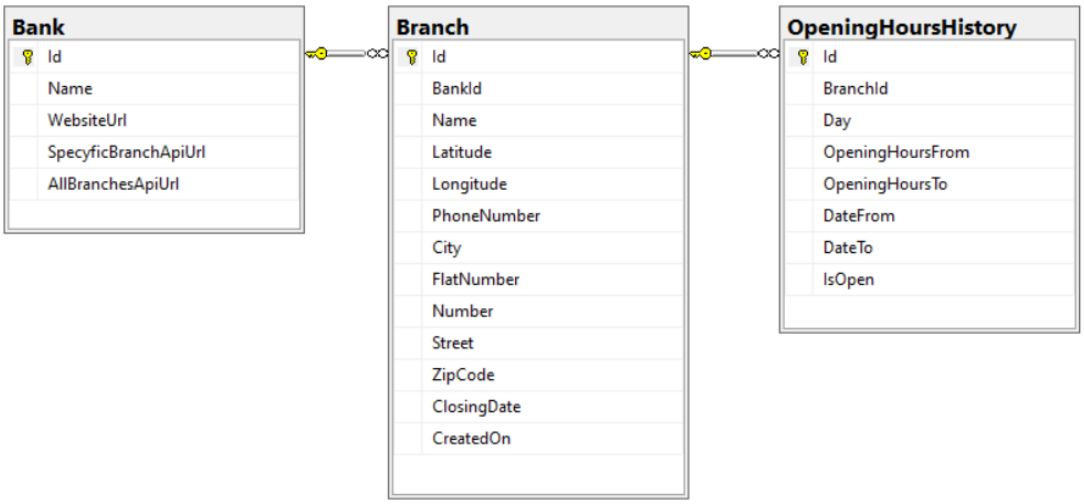
Aby przejść do następnego wyszukania należy kliknąć przycisk “Wyczyść”, aby zrestartować wszystkie dane do fazy początkowej.
Dodatkowo interfejs wyposażony jest w szybką zmianę lokalizacji mapy. Służy do tego rozszerzany panel “Szybkie wybieranie miasta”. Po otwarciu go, pojawiają się miasta do wyboru, gdzie kliknięcie powoduje szybkie przeniesienie mapy do danej lokalizacji.



Rysunek 8: Szybkie wybieranie miasta - lokalizacje

Ikonka ustawień pozwala na zmianę banku, który analizujemy. Dzięki temu, można analizować godziny otwarcia z kilku różnych punktów widzenia.

Schemat bazy danych



Rysunek 9: Schemat bazy danych

Metody pozyskiwania danych

Zdecydowana większość banków posiada wyszukiwarki placówek wraz z mapami. Tak, jak człowiek może z nich korzystać, tak samo może z nich korzystać program komputerowy, np. przy pomocy Web Scrapping-u.

Klasyczny Web Scrapping jest trudny do wykonania ze względu na to, że większość stron jest zaimplementowanych we frameworkach JavaScript i strony zwracają jedynie skompilowany JS, a nie statyczny HTML.

Na szczęście takie dane również są możliwe do wyciągnięcia. Opcją jest wykorzystanie biblioteki Selenium, która pozwala na wykonywanie JavaScript i wyrenderowanie pożądanego HTML-a.

Alternatywą jest korzystanie z endpointów serwera strony internetowej banku i wyciągnięcie z nich potrzebnych informacji. Takie rozwiązanie przyjął zespół.

Algorytm wyznaczania okręgu od placówki

Część kliencka systemu wysyła do backend-u informacje o lokalizacji placówki i odległości w kilometrach. Niestety Entity Framework nie obsługuje skomplikowanych zapytań do bazy danych. Pobieramy dane z kwadratu, którego środki krawędzi wyznacza odległość od punktu centralnego. Pomiędzy każdą z placówek, a punktem centralnym jest wyliczana odległość przy pomocy biblioteki GeoCoordinate.NetCore. Jeśli dystans jest większy niż nas interesuje placówka zostaje odrzucona z listy wynikowej.

Instrukcja instalacji

Frontend

Aplikacja kliencka została upubliczniona przy pomocy systemu chmurowego Azure. Zalecane jest, aby do publikowania zmian i uruchamiania aplikacji użyć CI/CD. Pipeline, który zarządza tworzeniem aplikacji składa się z czterech elementów: instalacja środowiska (npm install), budowanie (ng build), kompresowanie zbudowanej wersji (archive files) oraz upublicznienie aktualnej wersji (Publish artifacts). System operacyjny, który zostanie użyty do budowania nie ma znaczenia, może to być Linux jak i Windows.

Program można również uruchomić lokalnie. Potrzebne jest do tego zainstalowanie framework-u Angular oraz managera pakietów npm. W folderze z projektem potrzebne jest użycie dwóch komend: npm install oraz ng serve. W ten sposób przechodząc w przeglądarce pod adres http://localhost:4200 ukaże się lokalnie zbudowana wersja.

Backend

Instalacja została wykonana na Azure, z użyciem systemu operacyjnego Windows. Aby aplikacja działała należy podmienić adres bazy danych, zainstalować wymagane biblioteki (Nugety), oraz wykonać komendę update-database pozwalającą zaktualizować bazę danych jeśli zaszły zmiany w jej strukturze. Obie te czynności są zautomatyzowane przy zaproponowanych przez zespół rozwiązaniach.

Instalacji można dokonać na dwa sposoby. Pierwszym sposobem (zalecanym) jest wykorzystanie skonfigurowanego CI/CD. Każdy commit do gałęzi mastera powoduje automatyczne wgranie aplikacji na serwer. Drugim sposobem jest pobranie ze środowiska Azure profilu publikacji, który po instalacji w Visual Studio automatycznie konfiguruje wszystkie ustawienia. Następnie należy kliknąć przycisk Publish. Spowoduje to rozpoczęcie procesu publikacji. Po przesłaniu plików aplikacja uruchomi się automatycznie, co kończy cały proces.

Podsumowanie

Realizacja projektu przebiegła bardzo pomyślnie. Cel został osiągnięty. Działanie aplikacji klienckiej okazało się bardzo intuicyjne oraz pod względem stylistycznym prezentuje się ona bardzo dobrze. Dzięki czemu pracownicy banku prawie od zaraz będą mogli z niego korzystać, jeśli nadejdzie taka potrzeba.

Pobieranie danych ze stron banków tylko w jednym przypadku okazało się uciążliwe. Dane w mBanku prawdopodobnie są uzupełniane przez pracowników ręcznie, nie przy pomocy formularzy i dane nie zawsze mają taką samą strukturę.

Było to nasze pierwsze doświadczenie z usługą Azure, dzięki któremu stworzony system jest dostępny dla wszystkich, a nie tylko lokalnie. Pierwsze wrażenia oceniamy bardzo pozytywnie. Jest wiele opcji do wykorzystania, wiele poradników jak osiągnąć wskazany cel, a sam interfejs usługi jest bardzo intuicyjny.

Dodatkowo każdy z nas podszkolił się i zyskał kolejne doświadczenie w technologiach, które już znał.