

UNIWERSYTET IM. ADAMA MICKIEWICZA

W POZNANIU

Wydział Matematyki i Informatyki



**FindMyTutor - aplikacja wspomagająca komunikację między
studentami i nauczycielami akademickimi**

FindMyTutor - application enhancing communication between students and
academic teachers

Adam Domagalski 411201

Specjalność: Modelowanie i optymalizacja systemów

Marcin Jedyński 416084

Specjalność: Ogólna

Maciej Wanat 416196

Specjalność: Technologie informacyjne i internetowe

Mieszko Wrzeszczyński 416204

Specjalność: Ogólna

PRACA INŻYNIERSKA

w wykonana pod kierunkiem

dr Patryk Żywica

Poznań 2018

Poznań, dnia

OŚWIADCZENIE

Ja, niżej podpisany/a student/ka Wydziału Uniwersytetu im. Adama Mickiewicza w Poznaniu oświadczam, że przedkładaną pracę dyplomową pt:

..... napisałem/napisałam samodzielnie.
Oznacza to, że przy pisaniu pracy, poza niezbędnymi konsultacjami, nie korzystałem/am z pomocy innych osób, a w szczególności nie zlecałem/am opracowania rozprawy lub jej części innym osobom, ani nie odpisywałem/am tej rozprawy lub jej części od innych osób.

Oświadczam również, że egzemplarz pracy dyplomowej w wersji drukowanej jest całkowicie zgodny z egzemplarzem pracy dyplomowej w wersji elektronicznej.

Jednocześnie przyjmuję do wiadomości, że przypisanie sobie, w pracy dyplomowej, autorstwa istotnego fragmentu lub innych elementów cudzego utworu lub ustalenia naukowego stanowi podstawę stwierdzenia nieważności postępowania w sprawie nadania tytułu zawodowego.

[]* - wyrażam zgodę na udostępnianie mojej pracy w czytelni Archiwum UAM

[]* - wyrażam zgodę na udostępnianie mojej pracy w zakresie koniecznym do ochrony mojego prawa do autorstwa lub praw osób trzecich

*Należy wpisać TAK w przypadku wyrażenia zgody na udostępnianie pracy w czytelni Archiwum UAM, NIE w przypadku braku zgody. Niewypełnienie pola oznacza brak zgody na udostępnianie pracy.

.....

(czytelny podpis studenta)

Poznań, dnia

OŚWIADCZENIE

Ja, niżej podpisany/a student/ka Wydziału Uniwersytetu im. Adama Mickiewicza w Poznaniu oświadczam, że przedkładaną pracę dyplomową pt:

..... napisałem/napisałam samodzielnie. Oznacza to, że przy pisaniu pracy, poza niezbędnymi konsultacjami, nie korzystałem/am z pomocy innych osób, a w szczególności nie zlecałem/am opracowania rozprawy lub jej części innym osobom, ani nie odpisywałem/am tej rozprawy lub jej części od innych osób.

Oświadczam również, że egzemplarz pracy dyplomowej w wersji drukowanej jest całkowicie zgodny z egzemplarzem pracy dyplomowej w wersji elektronicznej.

Jednocześnie przyjmuję do wiadomości, że przypisanie sobie, w pracy dyplomowej, autorstwa istotnego fragmentu lub innych elementów cudzego utworu lub ustalenia naukowego stanowi podstawę stwierdzenia nieważności postępowania w sprawie nadania tytułu zawodowego.

[]* - wyrażam zgodę na udostępnianie mojej pracy w czytelni Archiwum UAM

[]* - wyrażam zgodę na udostępnianie mojej pracy w zakresie koniecznym do ochrony mojego prawa do autorstwa lub praw osób trzecich

*Należy wpisać TAK w przypadku wyrażenia zgody na udostępnianie pracy w czytelni Archiwum UAM, NIE w przypadku braku zgody. Niewypełnienie pola oznacza brak zgody na udostępnianie pracy.

.....
(czytelny podpis studenta)

Poznań, dnia

OŚWIADCZENIE

Ja, niżej podpisany/a student/ka Wydziału Uniwersytetu im. Adama Mickiewicza w Poznaniu oświadczam, że przedkładaną pracę dyplomową pt:

..... napisałem/napisałam samodzielnie. Oznacza to, że przy pisaniu pracy, poza niezbędnymi konsultacjami, nie korzystałem/am z pomocy innych osób, a w szczególności nie zlecałem/am opracowania rozprawy lub jej części innym osobom, ani nie odpisywałem/am tej rozprawy lub jej części od innych osób.

Oświadczam również, że egzemplarz pracy dyplomowej w wersji drukowanej jest całkowicie zgodny z egzemplarzem pracy dyplomowej w wersji elektronicznej.

Jednocześnie przyjmuję do wiadomości, że przypisanie sobie, w pracy dyplomowej, autorstwa istotnego fragmentu lub innych elementów cudzego utworu lub ustalenia naukowego stanowi podstawę stwierdzenia nieważności postępowania w sprawie nadania tytułu zawodowego.

[]* - wyrażam zgodę na udostępnianie mojej pracy w czytelni Archiwum UAM

[]* - wyrażam zgodę na udostępnianie mojej pracy w zakresie koniecznym do ochrony mojego prawa do autorstwa lub praw osób trzecich

*Należy wpisać TAK w przypadku wyrażenia zgody na udostępnianie pracy w czytelni Archiwum UAM, NIE w przypadku braku zgody. Niewypełnienie pola oznacza brak zgody na udostępnianie pracy.

.....
(czytelny podpis studenta)

Poznań, dnia

OŚWIADCZENIE

Ja, niżej podpisany/a student/ka Wydziału Uniwersytetu im. Adama Mickiewicza w Poznaniu oświadczam, że przedkładaną pracę dyplomową pt:

..... napisałem/napisałam samodzielnie. Oznacza to, że przy pisaniu pracy, poza niezbędnymi konsultacjami, nie korzystałem/am z pomocy innych osób, a w szczególności nie zlecałem/am opracowania rozprawy lub jej części innym osobom, ani nie odpisywałem/am tej rozprawy lub jej części od innych osób.

Oświadczam również, że egzemplarz pracy dyplomowej w wersji drukowanej jest całkowicie zgodny z egzemplarzem pracy dyplomowej w wersji elektronicznej.

Jednocześnie przyjmuję do wiadomości, że przypisanie sobie, w pracy dyplomowej, autorstwa istotnego fragmentu lub innych elementów cudzego utworu lub ustalenia naukowego stanowi podstawę stwierdzenia nieważności postępowania w sprawie nadania tytułu zawodowego.

[]* - wyrażam zgodę na udostępnianie mojej pracy w czytelni Archiwum UAM

[]* - wyrażam zgodę na udostępnianie mojej pracy w zakresie koniecznym do ochrony mojego prawa do autorstwa lub praw osób trzecich

*Należy wpisać TAK w przypadku wyrażenia zgody na udostępnianie pracy w czytelni Archiwum UAM, NIE w przypadku braku zgody. Niewypełnienie pola oznacza brak zgody na udostępnianie pracy.

.....
(czytelny podpis studenta)

Streszczenie

W erze postępującej cyfryzacji coraz więcej aspektów naszego życia jest informatyzowanych, co pozwala wprowadzać innowacyjne rozwiązania niedostępne dla tradycyjnych mediów. Rozwój technologii mobilnych dostarcza dużo potencjału w komputeryzacji konwencjonalnych problemów. Dzięki temu rośnie liczba użytkowników rezygnujących z klasycznych aplikacji na rzecz możliwości, które oferują dzisiejsze telefony.

Niniejsza praca opisuje projekt, który we współczesny sposób rozwiązuje klasyczny problem - komunikację studentów z profesorami na publicznej uczelni wyższej. Wykorzystanie wbudowanych w telefony modułów GPS umożliwiło stworzenie aplikacji, która pozwala kadrze dydaktycznej udostępniać informacje o swojej dostępności na dogodnym dla siebie poziomie. Studentom zaś dostarcza narzędzi, które pozwalają w łatwy sposób znaleźć danego profesora. Dzięki opartemu o architekturę REST wielodostępowemu API stworzony został interfejs zarówno mobilny jak i webowy, które razem w pełni wykorzystują cyfrowy potencjał do rozwiązania problemu.

Słowa kluczowe: *REST API, MVC, aplikacja mobilna, aplikacja webowa, aplikacje wieloplatformowe, ASP.net Core, Android, rxJava, MSSQL, nginx, docker, MVP, NativeScript, ReactNative, Xamarin, Ionic, SOLID, programowanie reaktywne, PWA, ciągłe dostarczanie, ciągła integracja*

Summary

In the age of fast-growing digitalization, more and more aspects of people's lives are being computerized, which facilitates an implementation of innovative solutions unavailable for the traditional media. The development in the field of mobile technologies provides a lot of potential in digitizing conventional problems. Thanks to that, one may observe an increase in the number of mobile users taking advantage of possibilities provided by modern mobile phones.

This paper describes a project which takes a modern approach in solving a classic problem – communication between students and professors at the public university. Having used built-in phone GPS modules, it was possible to create an application, which lets the teachers share information about their availability on the chosen level. Students on the other hand, can easily find any professor. REST based architecture made it possible to create both mobile and web interface, which together use the full digital potential to solve the problem.

Keywords: *REST API, MVC, mobil application, web application, cross-platform applications, ASP.net Core, Android, rxJava, MSSQL, nginx, docker, MVP, NativeScript, ReactNative, Xamarin, Ionic, SOLID, reactive programming, PWA, continuous delivery, continuous integration*

Spis treści

1.	Tworzenie współczesnych aplikacji serwerowych w technologii ASP.NET Core	13
1.1.	Wstęp	13
1.2.	Wprowadzenie.....	13
1.2.1.	Historia standardu .NET	13
1.2.2.	Porównanie frameworków .NET oraz .NET Core	15
1.3.	Wzorce projektowe i praktyki programistyczne	18
1.3.1.	MVC	18
1.3.2.	REST	20
1.3.3.	S.O.L.I.D.....	22
1.4.	Wybrane zagadnienia z ASP.NET Core	28
1.4.1.	Dokumentacja automatyczna oraz Swagger	28
1.4.2.	Zadania działające w tle.....	32
1.5.	Podsumowanie i wnioski	36
2.	Aplikacje wieloplatformowe jako alternatywa dla aplikacji natywnych.....	37
2.1.	Wstęp	37
2.2.	Analiza Rynku systemów operacyjnych	38
2.2.1.	Podsumowanie analizy	40
2.3.	Platformy Mobilne	40
2.3.1.	Android	40
2.3.2.	iOS.....	42
2.3.3.	Kod natywny	43
2.4.	Wieloplatformowość: hybrid vs cross-platform apps.....	44
2.4.1.	Mobilne aplikacje hybrydowe.....	44
2.4.2.	Mobilne aplikacje cross-platformowe	44
2.4.3.	Rozwiązania wieloplatformowe	50
2.5.	Problematyka wieloplatformowości	56
2.5.1.	Wydajność.....	56
2.5.2.	Bezpieczeństwo.....	56
2.5.3.	Zadania w tle	59
2.5.4.	Kompatybilność wstecz	62

2.6.	Profit z rozwiązania wieloplatformowego.....	63
2.6.1.	Single code base - jeden kod kilka aplikacji	63
2.6.2.	Szybkie dostarczenie MVP - niższe koszty	64
3.	Programowanie reaktywne na przykładzie platformy Android z użyciem biblioteki RxJava	65
3.1.	Wstęp	65
3.2.	Programowanie reaktywne	66
3.2.1.	Wzorzec projektowy obserwator	66
3.2.2.	Wzorzec projektowy iterator	68
3.2.3.	Biblioteka ReactiveX	71
3.2.4.	Elementy paradymatu funkcyjnego w programowaniu reaktywnym	74
3.2.5.	The Reactive manifesto	77
3.3.	Interfejs użytkownika w platformie Android.....	78
3.3.1.	Cykl życia aktywności.....	79
3.3.2.	Layout jak podstawowa jednostka widoku	82
3.3.3.	Format pliku szablonu	83
3.3.4.	Biblioteka ButterKnife	86
3.3.5.	Cykl życia widoku a biblioteka RxJava	87
3.4.	Przykłady zastosowanie biblioteki RxJava w projekcie FindMyTutor	88
3.4.1.	Obsługa wielowątkowości	88
3.4.2.	Serwisy HTTP	89
3.4.3.	Wyszukiwarka użytkowników	93
3.4.4.	Walidacja formularza logowania.....	94
3.5.	Podsumowanie	96
4.	Wdrożenia aplikacji serwerowych opartych o kontenery w metodyce ci/cd.....	97
4.1.	Wstęp – potrzeby zespołu deweloperskiego	97
4.2.	Biznesowa analiza konteneryzacji.....	98
4.2.1.	Udogodnienie dostarczania aplikacji dla klientów	98
4.2.2.	Udogodnienie procesu tworzenia oprogramowania	99
4.2.3.	Optymalizacja kosztów	100
4.3.	Kontenery	101
4.3.1.	Historia konteneryzacji	101

4.3.2.	Analiza wzrostu popularności kontenerów	104
4.4.	Kontenery Docker - wdrażanie aplikacji serwerowych	105
4.4.1.	Zasada działania	105
4.4.2.	Budowanie obrazów	106
4.4.3.	Uruchamianie i zarządzanie	108
4.5.	Ciągła integracja oraz ciągłe dostarczanie.....	110
4.5.1.	Definicja	110
4.5.2.	Narzędzie wspierające CI / CD – Buildbot	110
4.6.	Podsumowanie i wnioski	113
5.	Zakończenie	114
6.	Spis rysunków	115
7.	Bibliografia.....	118
8.	Dokumentacja.....	128
8.1.	Architektura systemu	128
8.2.	Dokumentacja REST API	129
8.3.	Instrukcja obsługi	131
	Ekran logowania.....	131
	Główny ekran aplikacji	132
	Ustawienia udostępniania.....	133
	Lista użytkowników.....	134

Wstęp

Projekt został zrealizowany dla Wydziału Informatyki i Matematyki Uniwersytetu im. Adama Mickiewicza w Poznaniu i został zakończony wdrożeniem, po którym został udostępniony kadrze dydaktycznej. Dzięki integracji z uczelnianym systemem LDAP każdy z użytkowników może zostać zautoryzowany bez potrzeby rejestracji czy też przechowywania haseł po stronie aplikacji. Aplikacja wie też, czy logowany jest student, czy profesor i na tej podstawie kieruje do odpowiedniej wersji. System automatycznie odnawia te informacje co semestr.

Strona internetowa WMI stanowi źródło informacji w oparciu, o które zapełniana jest baza danych aplikacji o profesorach. Dzięki temu możemy wyświetlać dane takie jak pokój danego profesora, adres email, czy też informacje o godzinach i dniach odbywających się dyżurów.

Aplikacji została stworzona przy pomocy usługi MapBox. Pozwoliła ona na stworzenie spersonalizowanej mapy wydziału, na której wyświetlane są markery symbolizujące profesorów. Udostępniany interfejs programistyczny pozwolił na zintegrowanie założonych funkcjonalności przy wykorzystaniu wszystkich funkcji usługi zarówno w wersji mobilnej jak i webowej.

Podział prac

Kolejne rozdziały pracy dyplomowej opisują zagadania, z którymi zespół inżynierski zapoznał się podczas tworzenia projektu FindMyTutor.

Rozdział 1, autor Maciej Wanat

Tworzenie współczesnych aplikacji serwerowych w technologii ASP.NET Core

Rozdział 2, autor Adam Domagalski

Aplikacje wieloplatformowe jako alternatywa dla aplikacji natywnych

Rozdział 3, autor Mieszko Wrzeszczyński

Programowanie reaktywne na przykładzie platformy Android z użyciem biblioteki RxJava

Rozdział 4, Marcin Jedyński

Wdrożenia aplikacji serwerowych opartych o kontenery w metodyce ci/cd

Rozdział 1

1. Tworzenie współczesnych aplikacji serwerowych w technologii ASP.NET Core

1.1. Wstęp

Pośród dominujących dziś technologii serwerowych jedną z zasługujących na szczególną uwagę jest ASP.NET Core. Rozwijany przez firmę Microsoft zestaw narzędzi pozwala na tworzenie kodu na wiele systemów operacyjnych (Linux, Windows, MacOS) w duchu idei open-source. Jego wydajność oraz szybkość działania sprawiają, iż jest to popularny wybór wśród programistów. Rozdział opisuje architekturę oraz proces tworzenia serwera opartego o model REST MVC w technologii .NET Core na przykładzie aplikacji FindMyTutor. Omawia również wiele wzorców projektowych oraz praktyk programistycznych charakterystycznych dla rozwiązań serwerowych w tej technologii.

W rozdziale rozważane są także różnice między starszą wersją framework'a a jego następcą. Na podstawie danych dotyczących obu rozwiązań autor ocenia która technologia jest korzystniejsza do stworzenia dziś nowego rozwiązania programistycznego.

1.2. Wprowadzenie

1.2.1. Historia standardu .NET

Prace nad .NET Framework rozpoczęły się w późnych latach 90', początkowo pod nazwą Generation Windows Services (NGWS). Rozwijany przez Microsoft produkt był częścią .NET Strategy - terminu marketingowego używanego przez firmę, pod którym tworzono szereg kompleksowych rozwiązań informatycznych. Były CEO Microsoft'u, Steve Ballmer, opisał .NET Strategy jako najambitniejsze przedsięwzięcie od czasu Internet Strategy Day w 1995 roku. Niemniej do 2003 roku marka upadła, porzucając związek z hasłem w produktach takich jak Windows Server czy też Visual Studio.

W czerwcu 2000 ogłoszono .NET Framework wraz z nowym językiem C#, a pod koniec tego samego roku została wydana pierwsza wersja beta .NET 1.0.

Celem technologii było zapewnienie szeregu rozwiązań, które razem pozwalały na łatwy rozwój różnorodnych aplikacji na platformę Windows. W dniu 24 kwietnia 2003 roku, Microsoft ogłosił globalną dostępność .NET Framework 1.1 oraz Visual Studio. Firma opisała swoje rozwiązanie słowami: „Together, Visual Studio .NET 2003, the Microsoft .NET Framework 1.1 and Windows Server 2003 provide developers with a robust, dependable application platform on which to build and deploy powerful, connected applications.” [1].

W grudniu 2007 Microsoft ogłosił, iż kod .NET Framework (nadchodzący był wtedy .Net Framework 3.5) będzie dostępny na licencji Microsoft Reference Source License (Ms-RSL). Mimo iż jest to najbardziej restryktywny z typów licencji Microsoft Shared Source, pozwolił on programistom przeglądać kod klas podczas debugowania. Od 16 stycznia 2008 .NET Framework dostępny jest właśnie w ramach tej licencji.

Poszczególne wersje .NET Framework oraz szczegółowe daty ich publikacji zaprezentowane są w poniższej tabeli:

Przegląd wersji .NET Framework			
Numer wersji	Data premiery	Zakończenie wsparcia	Narzędzie developerskie
1.0	2002-02-13	2009-07-14	Visual Studio .NET
1.1	2003-04-24	2015-06-14	Visual Studio .NET 2003
2.0	2005-11-07	2011-07-12	Visual Studio 2005
3.0	2006-11-06	2011-07-12	Expression Blend
3.5	2007-11-19	2028-10-10	Visual Studio 2008
4.0	2010-04-12	2016-01-12	Visual Studio 2010
4.5	2012-08-15	2016-01-12	Visual Studio 2012
4.5.1	2013-10-17	2016-01-12	Visual Studio 2013
4.5.2	2014-05-05	N/A	N/A
4.6	2015-07-20	N/A	Visual Studio 2015
4.6.1	2015-11-30	N/A	Visual Studio 2015 Update 1
4.6.2	2016-08-02	N/A	
4.7	2017-04-05	N/A	Visual Studio 2017
4.7.1	2017-10-17	N/A	Visual Studio 2017
4.7.2	2018-04-30	N/A	Visual Studio 2017
4.8	w rozwoju	N/A	Visual Studio 2019 (planowane)

Rysunek 1.1 Przegląd wersji .NET Framework [2]

W dniu 12 listopada 2014 Microsoft ogłosił .NET Core. Jego głównym celem było umożliwienie wytwarzanie wieloplatformowych aplikacji w technologii .NET. Miguel De Icaza opisał .NET Core słowami „redesigned version of .NET that is based on the simplified version of the class libraries” [3], a sam Microsoft określił go jako fundament przyszłości całej platformy. Technologia ta jest wytwarzana w modelu open-source, pozwalając społeczności programistów brać czynny udział w rozwoju platformy. Finalnie technologia została opublikowana 27 czerwca 2016.

Poszczególne wersje .NET Core oraz szczegółowe daty ich publikacji zaprezentowane są w poniższej tabeli.

Przegląd wersji .NET Core			
Numer wersji	Data premiery	Zakończenie wsparcia	Narzędzie developerskie
1.0	2016-06-27	2019-06-27	Visual Studio 2015, 2017
1.1	2016-11-18	2019-06-27	Visual Studio 2015, 2017
2.0	2017-08-14	2018-10-01	Visual Studio 2017
2.1	2018-05-30	2021-08-21	Visual Studio 2017
2.2	2018-12-04		Visual Studio 2017 15.9 i 2019 16.0 preview 1
3.0	w rozwoju		Visual Studio 2017 i 2019

Rysunek 1.2 Przegląd wersji .NET Core [4]

1.2.2. Porównanie frameworków .NET oraz .NET Core

Programista chcąc napisać dziś aplikację w technologii .NET staje przed niełatwym wyborem. Zarówno starsza jak i nowsza technologia zapewnia zestaw narzędzi niezbędny w procesie rozwoju aplikacji, a różnice mogą wydawać się nieoczywiste dla osób niezaznajomionych z frameworkami Microsoftu.

Pierwszą rzeczą, którą należy rozważyć przed decyzją na temat technologii, jest typ aplikacji, którą chcemy napisać. Obecnie .NET Core nie wspiera WPF (Windows Presentation Foundation) oraz WinForms, co może być przeszkodą, jeśli chcemy stworzyć program desktopowy na platformę Windows. Obie z wymienionych technologii są rozwijanymi przez Microsoft produktami pozwalającymi na proste tworzenie aplikacji klienckich działających bezpośrednio w systemie operacyjnym. .NET Core ma wspierać te typy technologii dopiero w wersji 3.0.

Kolejnym ograniczeniem narzuconym przez .NET Core jest brak wsparcia dla WebForms. Jest to technologia używana do tworzenia aplikacji webowych bazująca na ideach podobnych do WinForms. Przede wszystkim obie z tych technologii wykorzystują zdarzenia

(ang. events) jako podstawę swojego działania. Niemniej analogia do WinForms nie do końca sprawdziła się w kontekście aplikacji webowych, przede wszystkim uniemożliwiając łatwą separację części serwerowej i klienckiej. Z tego względu technologia ta jest stopniowo porzucana na rzecz MVC, które to spełnia współczesne standardy tworzenia aplikacji.

Jeśli zdecydujemy się na stworzenie aplikacji w technologii wspieranej przez oba frameworki, możemy skupić się na różnicach. Kluczową kwestią jest jeden z powodów, dla których powstał .NET Core – wieloplatformowe wsparcie. Jeśli chcemy, aby nasza aplikacja działała w środowiskach innych niż Windows, powinniśmy wybrać tę technologię.

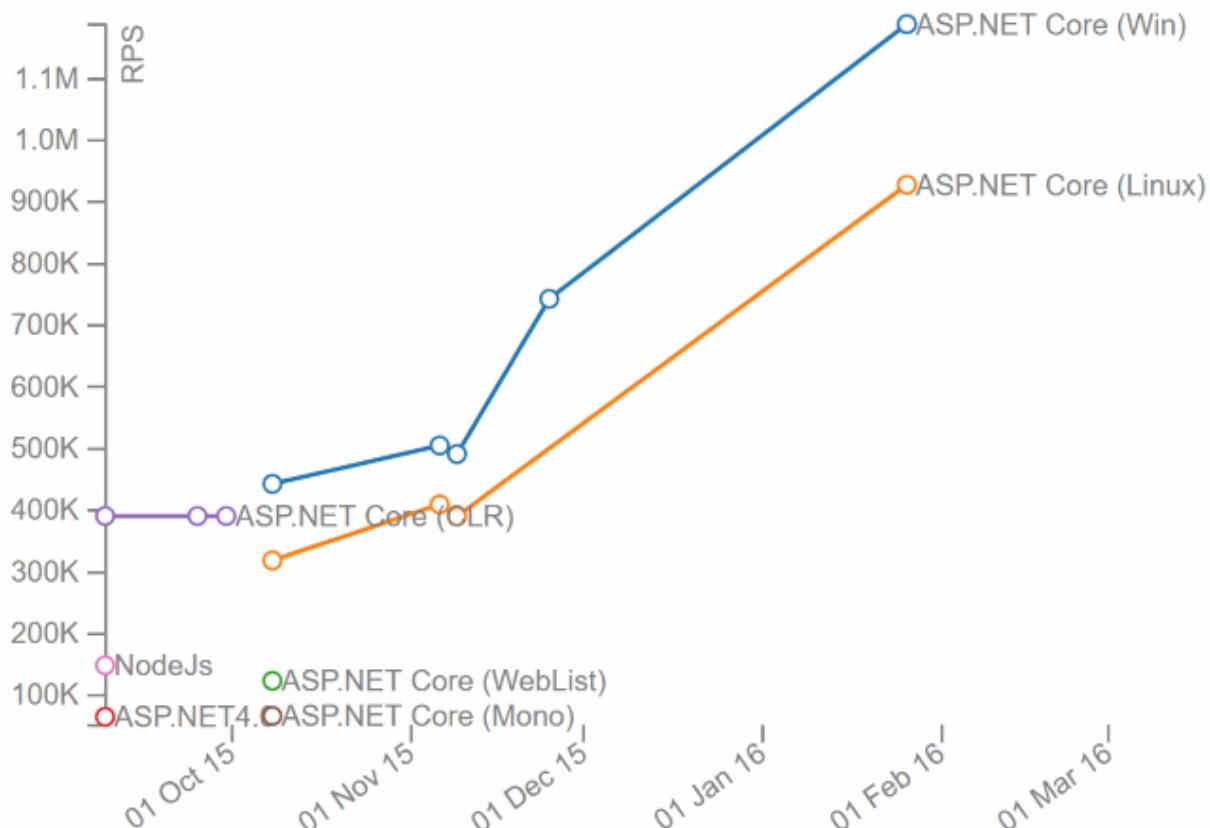
Dla programistów chcących pracować na systemie operacyjnym innym niż Windows istotny może okazać się fakt, iż jedynie .NET Core wspiera rozwój na takich systemach. Nie bez znaczenia pozostaje również, iż w tym framework'u zostało wprowadzone CLI (Command Line Interface). Umożliwia ono pracę bez związku z programem Visual Studio, a do całego procesu nie potrzebujemy już IDE (Integrated Development Environment).

W .NET Core nie musimy również korzystać z serwera http rozwijanego przez Microsoft (IIS), co pozwala nam na użycie technologii, która najbardziej nam odpowiada. Na potrzeby projektu FindMyTutor użyto serwera nginx hostowanego w systemie operacyjnym Linux.

Swoboda rozwijania aplikacji jaką oferuje .NET Core świetnie obrazuje ducha open-source, w którym rozwijany jest ten framework. Kod .NET Core jest publicznie dostępny jako repozytorium Git na serwerach serwisu GitHub, oraz jest rozwijany poprzez model publicznych pull requestów. Dzięki takiemu podejściu społeczność może mieć swój wkład w tę technologię, a cały kod jest darmowo ogólnodostępny dla dowolnego audytu.

Kolejną kwestią, która może być decydującą w wyborze technologii, jest wydajność. Na tym polu zdecydowanie wygrywa nowszy framework. Przykładowo dla prostej operacji jaką jest naprzemienne dodawanie i usuwanie elementów z kolejki (100 000 000 elementów), .NET Core poradzi sobie z zadaniem dwukrotnie szybciej. Niemniej na tym nie kończą się nowe możliwości. Test na strukturze SortedSet<int>, polegający na stworzeniu dużego zbioru tego typu (ponad ~400 000 elementów) pokazał, iż .NET Core potrafi czas ~7.7 sekund obniżyć do ~0.013 sekund, co daje około sześćsetkrotny wzrost wydajności. Wewnętrzne operacje na obiektach są więc dużo szybsze, zależnie od badanej struktury i rodzaju obliczeń, lecz zawsze zapewniają znaczny wzrost wydajności. Opisane testy oraz wyniki opisują porównanie .NET 4.7 oraz .NET Core 2.0.

Przekłada się to na testy wyższego poziomu. W lutym 2016 roku opublikowano wyniki testu, w którym osiągnął on wynik 2300% lepszy od poprzednika [5]. Mierzono wtedy zapytania na sekundę (ang. requests per second), a wynik osiągnięto dzięki zastosowaniu programowania asynchronicznego. Testowano proste zapytania niewymagające połączenia z bazą danych.



Rysunek 1.3 Tabela zapytań na sekundę
źródło: Ageofascent

W scenariuszach produkcyjnych różnice w wydajności nie są tak wysokie, lecz wciąż widoczne. Powyższy test ujawnia potencjał nowej technologii.

Niemniej nie bez powodu przy tworzeniu dużych, komercyjnych projektów nadal programiści nierzadko decydują się na .NET Framework. Pośród wielu dostępnych w galerii NuGet pakietów (menadżer pakietów .NET) wiele z nich wciąż dostępnych jest jedynie na starszą platformę i w tym aspekcie technologia ta może pochwalić się większą wszechstronnością. .NET Framework często traktowany jest jako technologia dojrzała i lepiej zbadana, niż stosunkowo młody .NET Core.

Ważąc wszystkie za i przeciw, do stworzenia serwera projektu FindMyTutor użyto nowszego framework'a. REST MVC idealnie nadaje się do stworzenia wymaganego rozwiązania, a .NET Core stanowi dojrzałą i współczesną opcję rozwoju aplikacji opartej o tę architekturę.

1.3. Wzorce projektowe i praktyki programistyczne

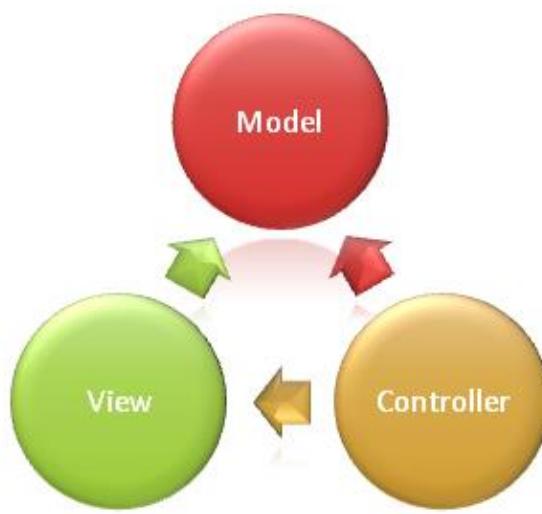
1.3.1. MVC

Wzorzec architektoniczny MVC stanowi dziś podstawę wytwarzania aplikacji serwerowych w technologii .NET Core. Idealnie nadaje się do potrzeb i wymagań współczesnych rozwiązań webowych.

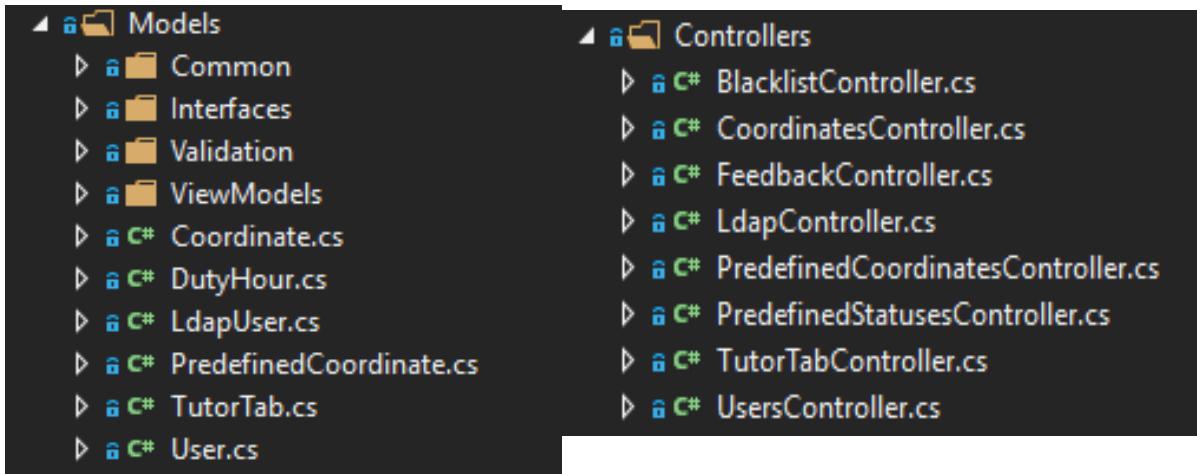
MVC stanowi akronim słów Model-View-Controller o które to opiera swoje działanie. Warto zaznaczyć, iż ten modny dziś standard powstał już w roku 1979, stworzony przez Norwega Trygve Reenskauga [6]. W aplikacjach tego typu zapytania użytkownika są przekierowane (ang. routed) do Kontrolera, który pracując na Modelach wykonuje żądane przez użytkownika operacje, a następnie wyświetla je w Widoku. Kontroler reprezentuje więc logikę sterującą aplikacją, model odzwierciedla dane a widok służy za interfejs.

Całość zorganizowana jest w myśl podziału odpowiedzialności (ang. separation of concerns). Jest to zasada powiązana ściśle z inną znaną każdemu programiście praktyką – zasadą jednej odpowiedzialności (ang. single responsibility principle) i oznacza jej wysokopoziomową implementację. Zasada ta została opisana w późniejszym podrozdziale (S.O.L.I.D.).

Zaletą separacji jest możliwość budowania poszczególnych komponentów w oparciu o różne technologie. Jako iż część serwerowa aplikacji FindMyTutor stanowi API, nie występują w niej widoki, których role pełnią osobne aplikacje. Dzięki temu program może wykorzystać jeden serwer na wielu platformach (Android, PC).



Rysunek 1.4 Model MVC [4]



Rysunek 1.5 Modele i kontrolery w aplikacji FindMyTutor

Ważne jest, aby metody kontrolera zawierały logikę sterującą aplikacją, lecz aby nie zawierały logiki biznesowej, która zazwyczaj znajduje się w serwisach. Innymi słowy, kontroler deleguje zadania operując na modelach, lecz nie zajmuje się ich wykonaniem.

```
// GET: api/userLogin/s519230
[HttpGet("userLogin/{ldapLogin}")]
[ProducesResponseType(400)]
[ProducesResponseType(typeof(string), 200)]
public async Task<IActionResult> GetUserLogin([FromRoute] string ldapLogin)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }

    var user = await _usersService.GetUserByLdap(ldapLogin);

    if (user == null)
    {
        return NotFound();
    }

    return Ok(user.Id);
}
```

Rysunek 1.6 Przykładowa metoda kontrolera w aplikacji FindMyTutor

MVC to współczesny wzorzec wytwarzania aplikacji wspierany przez Microsoft, dlatego stanowi dobry wybór do tworzenia aplikacji w technologii .NET Core. Możemy za jej pomocą tworzyć nie tylko API, lecz również komplementarne rozwiązanie, które do widoków wykorzystuje autorską technologię firmy Microsoft, RazorSharp. Jest to język stanowiący połączenie HTML oraz C#, pozwalający wprowadzać do HTML elementy logiki oraz programowania.

Niemniej, równie popularnym rozwiązaniem jest wykorzystanie .NET Core do stworzenia webowego API opartego o MVC. Właśnie w ten sposób stworzona część serwerowa aplikacji FindMyTutor.

1.3.2. REST

Jednym z szeroko wykorzystywanych komercyjnie standardów wytwarzania aplikacji serwerowych w technologii .NET Core jest REST.

REST (Representational State Transfer) został zdefiniowany w roku 2000 przez Roy'a Fielding'a w pracy pod tytułem "Architectural Styles and the Design of Network-based Software Architectures" która była również jego rozprawą doktorską [7]. Swoją popularność zawdzięcza on prostocie oraz kompatybilności z wdrożonymi już rozwiązaniami sieciowymi. Jest to styl architektoniczny, definiujący zestaw reguł w tworzeniu sieciowych serwisów. Opiera się on o 6 zasad, wokół których jest zbudowany:

1. Architektura klient – serwer

W zasadzie tej chodzi głównie o separację odpowiedzialności – w REST interfejs jest oddzielony od źródła danych. Pozwala na łatwe wdrożenie aplikacji na wiele platform oraz wspomaga skalowanie.

2. Bezstanowość

Każde zapytanie musi zawierać wszystkie dane potrzebne do wykonania operacji – jest ono bezstanowe i nie może wykorzystywać żadnego kontekstu po stronie serwera.

3. Lokalne zasoby (ang. cacheability)

Architektura powinna mieć możliwość przechowywania części zasobów, zamiast każdorazowego o nieodpytywania (ang. caching). Dlatego każda z udostępnianych danych powinna definiować, czy jest przeznaczona do lokalnego przetrzymywania w pamięci.

4. System warstwowy

Żaden komponent aplikacji nie powinien widzieć niczego poza swoją „warstwą” – nie powinien wiedzieć, czy stanowi pośrednictwo pomiędzy serwerem końcowym, czy jest do niego bezpośrednio podłączony.

5. Jednolity interfejs

Zasada ta zawiera zestaw reguł, według których w architekturze REST definiujemy interfejsy. Są to:

- **Identyfikacja zasobów**

Każdy zasób jest rozpoznawany przez specyficzne zapytanie – na przykład zgodne ze standardem URI.

- **Manipulacja zasobami przez ich reprezentacje**

Poprzez standard http możemy dokładnie opisać komunikację. Na przykład GET oznacza, iż chcemy pobrać dane o zasobie zdefiniowanym przez URI. Kiedy klient posiadający reprezentację zasobu, powinien mieć dość informacji, aby go usunąć lub zmodyfikować.

- **Samoopisowe wiadomości**

Każda wiadomość powinna zawierać dość informacji, aby opisać w jaki sposób przetworzyć wiadomość. Na przykład – który z parserów użyć dla określonego typu zawartości (ang. media type).

- **Hipermateria jako silnik stanu aplikacji (HATEOAS)**

Aplikacja powinna pozwalać na sterowanie poprzez dodatkowe informacje pozwalające na poruszanie się po API – linki do zasobów. Oznacza to, iż moglibyśmy teoretycznie zbudować system, który zawiera jeden endpoint do wysłania zapytania, które następnie kieruje w głąb aplikacji do kolejnych zasobów. Przykładowa odpowiedź z aplikacji implementującej HATEOAS mogłaby wyglądać następująco:

```
{  
    "resourceId": "efe56a80-f5c3-46a7-ad87-1536130be6fa" ,  
    "resourceName": "example",  
    "_links": [  
        { "rel": "anotherResource", "href": "/app/anotherResource", "method": "GET" },  
        { "rel": "anotherResource2", "href": "/app/anotherResource2", "method": "POST" }  
    ]  
}
```

Rysunek 1.7 Przykładowa odpowiedź z aplikacji implementującej HATEOAS

Kod na żądanie (opcjonalnie)

Zasada ta oznacza, iż serwer może rozszerzać funkcjonalność klienta poprzez umożliwienie ściągania i egzekucji kodu w formie skryptów.

W oparciu o powyższy zestaw reguł tworzone są systemy webowe oferujące separację odpowiedzialności, skalowalność oraz implementację współczesnych reguł architektonicznych i standardów wytwarzania kodu. Ze względu na korzyści, które zapewnia, aplikacja FindMyTutor została stworzona w oparciu o ten właśnie standard.

1.3.3. S.O.L.I.D.

SOLID stanowi mnemonik stworzony przez Roberta C.Martina, który opisuje pięć zasad którymi powinien się kierować programista, programując obiekty. Dotyczy on nie tylko programowania w technologii .NET Core, niemniej stanowi ona nieodzowny wzorzec pisania aplikacji o nią opartej.

Litera S odnosi się do zasady pojedynczej odpowiedzialności (ang. single responsibility principle). Zasada pojedynczej odpowiedzialności mówi, iż klasa lub moduł powinna być odpowiedzialna za jedynie jedną funkcjonalność i w całości zawierać jej implementację. Jest to zasada, której autorstwo przypisuje się Robertowi C. Martinowi, który opisał ją słowami: „A class should have only one reason to change” [8]. Oznacza to, iż jeśli klasę można zmodyfikować więcej niż z jednego powodu, łamiemy zasadę pojedynczej odpowiedzialności.

```
public async Task<TutorTab> GetTutorTabForUser(string userId)
{
    using (var scope = _scopeFactory.CreateScope())
    {
        TutorDbContext context = scope.ServiceProvider.GetRequiredService<TutorDbContext>();
        return await context.TutorTabs
            .Where(e => e.UserId == userId)
            .Include(e => e.DutyHours)
            .SingleOrDefaultAsync();
    }
}
```

Rysunek 1.8 Przykład metody spełniającej zasadę pojedynczej odpowiedzialności – pobranie zakładki profesora dla danego użytkownika

Litera O oznacza zasadę otwarte/zamknięte (ang. open/closed principle). Mówiąc o niej, klasy powinny być otwarte na rozszerzenia, lecz zamknięte na modyfikacje. Powinno być możliwe rozszerzanie encji bez ingerencji w jej istniejący kod. Dodawanie kolejnych funkcji klasy nie powinno wpływać na już istniejące implementacje.

```
public class FeedbackService
{
    private readonly object _lock = new object();
    private readonly object _autoLock = new object();

    public string GetFeedback()...
    public void AppendFeedback(Feedback feedback)...
    public void AppendAutoFeedback(string feedback, User user)...
}
```

Rysunek 1.9 Przykład klasy spełniającej zasadę otwarte/zamknięte – klasa FeedbackService

Na przykładzie powyższej klasy, możemy zobaczyć, że dodawanie kolejnych metod nie wpłynie na istniejący już kod. Chcąc rozszerzyć klasę feedback o np. pobieranie feedbacku dodanego jedynie przez profesorów, można to w łatwy sposób zrealizować, implementując metodę GetTutorsFeedback(). Widać jak zasada otwarte/zamknięte łączy się z regułą pojedynczej odpowiedzialności – często realizowanie pojedynczej funkcjonalności jest warunkiem niezbędnym, aby klasy nie trzeba modyfikować w celu jej rozszerzenia.

Litera L odnosi się do zasady podstawienia Liskov (ang. Liskov substitution principle). Nazwa pochodzi od nazwiska Barbary Liskov, która sformułała ją po raz pierwszy w książce Data Abstraction and Hierarchy. Mówi ona, iż funkcje, które używają wskaźników lub referencji do klas bazowych, muszą być w stanie używać również obiektów klas dziedziczących po klasach bazowych, bez dokładnej znajomości tych obiektów.

Oznacza ona, iż klasa dziedzicząca powinna implementować wszystkie metody klasy bazowej i zapewniać wszystkie funkcjonalności, które zapewnia klasa bazowa. Jego łamanie oznacza zazwyczaj źle zaprojektowaną hierarchię dziedziczenia. Klasycznym problemem obrazującym zasadę podstawienia Liskov jest problem koła i elipsy (ang. circle-ellipse problem). Jego treść wygląda następująco.

Rozważmy dwie klasy:

```
public class Ellipse
{
    private int X { get; set; }
    private int Y { get; set; }

    public void StretchX(int x)
    {
        this.X = x;
    }

    public void StretchY(int y)
    {
        this.Y = y;
    }
}

public class Circle : Ellipse
{}
```

Rysunek 1.10 Przykład klas łamiących zasadę podstawienia Liskov w .NET Core

Pozornie sensownym wydaje się dziedziczenie koła po elipsie, jako iż koło stanowi specjalny przypadek elipsy (o równych współrzędnych X i Y, gdzie wartości te stanowią średnicę w pionie i w poziomie). Bardzo łatwo zauważyc, iż nie mamy możliwości implementacji metod StretchX oraz StretchY w klasie Circle, ponieważ modyfikacja którejś z tych wartości spowodowałaby, iż koło przestałoby być kołem. Nie możemy zaimplementować metod klas bazowych odwzorowując ich działanie, tak więc łamiemy zasadę podstawienia Liskov. W związku z tym, kod wykorzystujący metody klasy bazowej, nie będzie mógł być wykonany. Zostanie zwrócony błąd lub efekt będzie niezgody z zamierzeniem – często nieoczekiwany dla programisty.

```
public class RunCode
{
    public void Main()
    {
        var circle = new Circle();

        ManipulateObject(circle);
    }

    public void ManipulateObject(Ellipse ellipse)
    {
        ellipse.StretchX(12);
        ellipse.StretchY(13);
    }
}
```

Rysunek 1.11 Kod, który powinien zostać poprawnie wykonany dla proponowanej w fig.x struktury dziedziczenia

Litera I oznacza zasadę segregacji interfejsów (ang. interface segregation principle). Mówi ona, iż wiele dedykowanych interfejsów jest lepsze niż jeden ogólny. Żadna encja nie powinna być zmuszona do korzystania z metod, których nie używa.

Często określenie dobrej abstrakcji obiektu jest nieoczywiste. Niemniej zła abstrakcja przynosi mniej korzyści niż jej absolutny brak. Rozważmy przykład interfejsu obiektu działającego na plikach.

```
interface IFileWorker
{
    void Read();
    void Write();
    void Trim();
}
```

Rysunek 1.12 Przykład interfejsu działającego na plikach w .NET Core

Taki interfejs pozornie jest poprawny, lecz łatwo zauważyc, iż jest bardzo ogólny. Pisząc klasę, która ma za zadanie jedynie odczytywać z pliku, przy implementacji interfejsu IFileWorker musielibyśmy implementować dodatkowe, niepotrzebne metody. Dlatego zgodnie z zasadą segregacji interfejsów, lepszym rozwiązaniem jest utworzenie trzech interfejsów.

```
interface ITrimmer interface IWriter   interface IReader
{
    void Trim();           {
                           void Write();    {
                           }                   void Read();
}
```

Rysunek 1.13 Przykład interfejsów zgodnych z zasadą segregacji interfejsów w .NET Core

Przy tej implementacji możemy stworzyć obiekt realizujący bardziej atomowe funkcjonalności, jak i za pomocą dziedziczenia wielu interfejsów możemy stworzyć model realizujący je wszystkie.

Litera D oznacza regułę odwrócenia zależności (ang. dependency inversion principle). Reguła ta jest oparta o dwie poprzednie – zasadę otwarte/zamknięte oraz zasadę podstawienia Liskov. Mówi ona, iż moduły wysokiego poziomu (zapewniające złożoną logikę) powinny być wielokrotnego użytku (ang. reusable), a zmiany w modułach niskiego poziomu nie powinny wpływać na funkcjonalności, które one zapewniają. Innymi słowy – moduły wysokiego poziomu nie powinny zależeć od modułów niskiego poziomu. Drugą zasadą kryjącą się za odwróceniem zależności jest reguła, iż abstrakcje nie powinny zależeć od szczegółów, lecz odwrotnie, implementacja powinna zależeć od abstrakcji.

Przykładem realizacji tej zasady w aplikacji FindMyTutor są niestandardowe (ang. custom) atrybuty walidacji modelu.

The screenshot shows a GitHub code diff for a file named 'LongitudeRangeAttribute.cs'. The commit message is '4 references | Maciej Wanat, 48 days ago | 1 author, 1 change'. The code implements a validation attribute for longitude ranges:

```
public class LongitudeRangeAttribute : ValidationAttribute
{
    protected override ValidationResult IsValid(object value, ValidationContext validationContext)
    {
        if (AppConfiguration.Current.Security.SecurityOverride)
        {
            return ValidationResult.Success;
        }

        var minimalLongitude = AppConfiguration.Current.Security.AllowedCoordinatesRange.Longitude.First();
        var maximalLongitude = AppConfiguration.Current.Security.AllowedCoordinatesRange.Longitude.Last();

        ICoordinate coordinate = (ICoordinate)validationContext.ObjectInstance;

        if (coordinate.Longitude >= minimalLongitude && coordinate.Longitude <= maximalLongitude)
        {
            return ValidationResult.Success;
        }

        return new ValidationResult("Longitude must be between " + minimalLongitude + " and " + maximalLongitude);
    }
}
```

Rysunek 1.14 Atrybut walidacji LongitudeRangeAttribute w aplikacji FindMyTutor

Atrybut ten odpowiada za sprawdzenie, czy podane wartości koordynatów mieszczą się w zdefiniowanym w ustawieniach aplikacji zakresie. Jego działanie opiera się o interfejs ICoordinate. Jako moduł wysokiego poziomu, nie zależy on od szczegółów implementacyjnych klasy (w tym przypadku modelu), lecz jedynie od ich abstrakcji. Dzięki temu możliwe jest wielokrotne użycie atrybutu w wielu modelach, implementujących wymieniony interfejs. W przypadku FindMyTutor jest to na przykład koordynat oraz koordynat predefiniowany.

```
26 references | Maciej Wanat, 48 days ago | 3 authors, 14 changes
public class Coordinate : ICoordinate
{
    14 references | Maciej Wanat, 48 days ago | 2 authors, 2 changes
    public Guid CoordinateId { get; set; }
    [Required]
    [LatitudeRange]
    16 references | Maciej Wanat, 48 days ago | 3 authors, 5 changes
    public float Latitude { get; set; }
    [Required]
    [LongitudeRange]
    18 references | Maciej Wanat, 48 days ago | 3 authors, 4 changes
    public float Longitude { get; set; }

5 references | Maciej Wanat, 24 days ago | 3 authors, 4 changes
public class PredefinedCoordinate : ICoordinate
{
    1 reference | Maciej Wanat, 48 days ago | 2 authors, 2 changes
    public Guid PredefinedCoordinateId { get; set; }
    [Required]
    [LatitudeRange]
    16 references | Maciej Wanat, 48 days ago | 2 authors, 2 changes
    public float Latitude { get; set; }
    [Required]
    [LongitudeRange]
    18 references | Maciej Wanat, 48 days ago | 2 authors, 2 changes
    public float Longitude { get; set; }
```

Rysunek 1.15 Modele Coordinate oraz PredefinedCoordinate, wykorzystujące te same atrybuty walidacji

Stosowanie się do reguł opisanych przez Robert'a C. Martin'a jest kluczowe w pisaniu łatwego w utrzymaniu, zrozumiałego i wydajnie działającego kodu. Jest czynnikiem niezbędnym w stosowaniu przy tworzeniu architektury serwerowej opartej o .NET Core.

1.4. Wybrane zagadnienia z ASP.NET Core

1.4.1. Dokumentacja automatyczna oraz Swagger

Swagger stanowi niezwykle popularny framework wykorzystywany przy tworzeniu webowego API. Jest to rozwiązanie otwarte (ang. open-source), zawierające zestaw narzędzi do budowania, dokumentacji oraz wywoływania sieciowych serwisów REST. Dzięki niemu możemy bardzo niskim narzutem prac uzyskać automatycznie generowany zestaw dokumentów i funkcji, które usprawniają rozwój projektu oraz ułatwiają jego utrzymanie.

Dodanie Swagger'a do projektu w ASP.NET Core 2.1 jest bardzo proste. Najpierw musimy usługę skonfigurować w metodzie `ConfigureServices()` klasy `Startup.cs`. W podstawowej formie, wystarczy do tego poniższy kod:

```
services.AddSwaggerGen(c =>
{
    c.SwaggerDoc("v1", new Info
    {
        Version = "v1",
        Title = "My API"
    });
});
```

Rysunek 1.16 Fragment kodu metody `ConfigureServices()`, odpowiedzialny za konfigurację Swagger'a

W projekcie FindMyTutor użyto kilku niestandardowych opcji. Dodatkową funkcją, która została wykorzystane jest funkcja autoryzacji – w UI możemy dodać token aby „zalogować się” jako dany użytkownik. Dodana została również obsługa komentarzy dodawanych w kodzie w języku XML, które są potem widoczne jako opisy konkretnych endpointów i wzbogacają dokumentację. Finalnie konfiguracja Swaggera w projekcie FindMyTutor wygląda więc następująco:

```

services.AddSwaggerGen(c =>
{
    c.SwaggerDoc("v1", new Info { Version = "v1", Title = "FindMyTutor", });

    // Swagger 2.+ support
    var security = new Dictionary<string, IEnumerable<string>>
    {
        {"Bearer", new string[] { }},
    };

    c.AddSecurityDefinition("Bearer", new ApiKeyScheme
    {
        Description = "JWT Authorization header using the Bearer scheme. Example: \"Bearer {token}\\"",
        Name = "Authorization",
        In = "header",
        Type = "apiKey"
    });
    c.AddSecurityRequirement(security);

    // Set the comments path for the Swagger JSON and UI.
    var xmlFile = $"{Assembly.GetExecutingAssembly().GetName().Name}.xml";
    var xmlPath = Path.Combine(ApplicationContext.BaseDirectory, xmlFile);
    c.IncludeXmlComments(xmlPath);
});

```

Rysunek 1.17 Konfiguracja frameworku Swagger w projekcie FindMyTutor

Kolejnym krokiem jest dodanie odpowiedniego oprogramowania pośredniczącego (middleware). W ASP.NET Core dodajemy je za pomocą następującego kodu w metodzie **Configure()** klasy **Startup.cs**:

```

app.UseSwagger();
app.UseSwaggerUI(c =>
{
    c.SwaggerEndpoint("v1/swagger.json", "FindMyTutor V1");
});

```

Rysunek 1.18 Fragment kodu dodający oprogramowanie pośredniczące Swagger'a w projekcie FindMyTutor

Swagger na podstawie kodu projektu generuje plik json, zawierający opis całego API. Bazując na tymże pliku Swagger buduje interfejs UI oraz dokumentację. Jest on domyślnie dostępny pod adresem: {adres_serwera}/swagger.

```

[ ... ]  

  • swagger: "2.0",  

  • info:  

    -  

    { ...  

      o version: "v1",  

      o title: "FindMyTutor"  

    },  

  • paths:  

    -  

    { ...  

      o /api/users/blacklist/{tutorId}:  

        -  

        { ...  

          • get:  

            -  

            { ...  

              • tags:  

                -  

                [ ...  

                  • "Blacklist"  

                ],  

              • operationId: "ApiUsersBlacklistByTutorIdGet",  

              • consumes: [ ],  

              • produces:  

                -  

                [ ...  

                  • "application/json"  

                ],  


```

Fragment pliku json generowanego przez swaggera

Rysunek 1.19 Fragment dokumentacji automatycznej wygenerowanej przez framework Swagger w projekcie FindMyTutor

Dokumentacja zawiera spis wszystkich endpointów dostępnych w aplikacji, z podziałem na kontrolery. Na samym dole strony obecne są wszystkie wykorzystywane modele oraz ich struktura. Po kliknięciu w zapytanie wyświetcone zostaną wszystkie o nim informacje – jaki model przyjmuje (z przykładowymi wartościami), jakie odpowiedzi może zwrócić serwer, razem z modelami, a także opis, jeśli został dołączony w kodzie. Ważną funkcjonalnością zapewnianą przez framework jest możliwość testowania. Po wciśnięciu przycisku „Try it out” mamy możliwość wpisania własnych danych oraz wysłania zapytania. Odpowiedź serwera pojawia się poniżej, w osobnym oknie.

POST /api/ldap/validate Login / register user with this endpoint

It should be used like plain logging - user should not know he is registering at all.
"Registration" is in fact just copying user into our database from LDAP.

Method work scheme:

1. Check if user has isLocal flag set to true, if yes attempt to log him in locally (with local db password).
2. Check password/login for correctness in LDAP
3. If user is validated correctly, check if it exists in our DB.
4. If user is in our DB, log him in (return token).
5. If not, scrap data about him from LDAP, register him (add to our DB), and log in (return token).

Parameters

Name Description

IdapLoginModel (body)

Example Value | Model

```
{
  "login": "string",
  "password": "string"
}
```

Parameter content type application/json-patch+json

Responses

Code Description Response content type

200 Success application/json

Example Value | Model

```
{
  "token": "string"
}
```

400 Bad Request

Rysunek 1.20 Przykład metody Validate() kontrolera LdapController wygenerowany przez framework Swagger

```

// POST: ldap/validate
/// <summary>
/// Login / register user with this endpoint
/// </summary>
/// <remarks>
/// It should be used like plain logging - user should not know he is registering at all.
/// "Registration" is in fact just copying user into our database from LDAP.
/// Method work scheme:
/// 0. Check if user has isLocal flag set to true, if yes attempt to log him in locally (with local db password).
/// 1. Check password/login for correctness in LDAP
/// 2. If user is validated correctly, check if it exists in our DB.
/// 3. If user is in our DB, log him in (return token).
/// 4. If not, scrap data about him from LDAP, register him (add to our DB), and log in (return token).
/// </remarks>
/// <returns>Token for the authorized user</returns>
[AllowAnonymous]
[HttpPost("validate")]
[ProducesResponseType(400)]
[ProducesResponseType(typeof(TokenModel), 200)]
0 references | Maciej Wanat, 22 days ago | 2 authors, 14 changes
public async Task<IActionResult> PostValidate([FromBody] LdapLoginModel ldapLoginModel)...

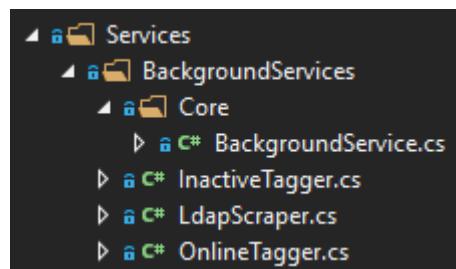
```

Rysunek 1.21 Dokumentacja metody Validate() w kodzie FindMyTutor

Swagger pozwala w prosty sposób stworzyć bogatą, generowaną automatycznie dokumentację. Jest również sposobem na łatwe testowanie stworzonych wcześniej zapytań. Jego interfejs pozwala programistom front-end'owym na łatwy dostęp do modeli oraz ścieżek wymaganych metod. Stanowczo ułatwia to pracę nad projektem oraz umożliwia jego utrzymanie. Integracja framework'a z kodem w ASP.NET Core jest prosta i szybka, dlatego też narzędzie to zostało wykorzystane w projekcie FindMyTutor.

1.4.2. Zadania działające w tle

Waczną częścią kodu, o którego działanie opiera się aplikacja FindMyTutor są zadania działające w tle (ang. background tasks). Za ich pomocą możliwe było zrealizowanie oznaczanie profesorów jako dostępnych, niedostępnych, aktywnych, nieaktywnych, czy też automatyczne ściąganie listy profesorów z systemu LDAP, jak i ściąganie danych na temat dyżurów ze strony WMI. Wiele projektów charakteryzuje się problemami specyficznymi dla tego zadań działających w tle, a .NET Core 2.1 stanowi framework oferujący łatwą oraz wydajną implementację ich rozwiązań.



Rysunek 1.22 Zadania działające w tle w aplikacji FindMyTutor

Działanie zadań w tle oparte jest o klasę **BackgroundService()** – każdy z serwisów zadań w tle po niej dziedziczy. Klasa ta definiuje przede wszystkim metody **ExecuteAsync()**, **StartAsync()**, oraz **StopAsync()**. Metody te stanowią interfejs pozwalający na rozpoczęcie lub zakończenie zadania, oraz jak i na przetworzenie jego części logicznej.

```
public abstract class BackgroundService : IHostedService, IDisposable
{
    protected readonly IServiceScopeFactory _scopeFactory;
    private Task _executingTask;
    private readonly CancellationTokenSource _stoppingCts =
        new CancellationTokenSource();

    3 references | Maciej Wanat, 53 days ago | 1 author, 1 change | 0 exceptions
    public BackgroundService(IServiceScopeFactory scopeFactory)...

    4 references | Maciej Wanat, 53 days ago | 1 author, 1 change | 0 exceptions
    protected abstract Task ExecuteAsync(CancellationToken stoppingToken);

    0 references | Maciej Wanat, 53 days ago | 1 author, 1 change | 0 exceptions
    public virtual Task StartAsync(CancellationToken cancellationToken)...

    0 references | Maciej Wanat, 53 days ago | 1 author, 1 change | 0 exceptions
    public virtual async Task StopAsync(CancellationToken cancellationToken)...

    0 references | Maciej Wanat, 53 days ago | 1 author, 1 change | 0 exceptions
    public virtual void Dispose()...
}
```

Rysunek 1.23 *BackgroundService* – klasa bazowa dla wszystkich zadań działających w tle

Metoda **ExecuteAsync()** odpowiada za główną część logiki zawartej w klasie. Przyjmuje ona obiekt klasy **CancellationToken**, który pozwala na anulowanie wątku z klasy, która ówcześnie go stworzyła. Pozwala to w razie potrzeby na zatrzymanie działania metody „z zewnątrz”.

Działanie metody opiera się o periodyczne wykonywanie zadanej logiki. W przypadku **OnlineTagger** – klasy odpowiadającej za oznaczanie użytkowników jako aktywnych, jeśli w ciągu ostatnich dwóch minut udostępnili oni swoją lokalizację, czas wykonania wynosi 2 minuty. Co taki odstęp czasu wykonywana jest metoda **ProcessCoords**, która odpowiada za opisaną wcześniej logikę. Za zachowanie interwału odpowiada metoda statycznej klasy **Task** – **Delay()**. Pozwala ona na zatrzymanie pracy metody poprzez stworzenie wątku działającego przez zadany przez na czas. Dodatkowo działanie klasy jest zawarte w bloku **try / catch**, dzięki czemu błąd w wykonaniu metody nie powoduje zakończenia działania obiektu. W przypadku pojawięcia się błędu, informacja o nim jest zapisywana do logów aplikacji.

```

protected override async Task ExecuteAsync(CancellationToken stoppingToken)
{
    while (!stoppingToken.IsCancellationRequested)
    {
        try
        {
            using (var scope = _scopeFactory.CreateScope())
            {
                var dbContext = scope.ServiceProvider.GetRequiredService<TutorDbContext>();

                _logger.LogInformation("Online tagger Service is Running");

                await ProcessCoords(dbContext);
            }
        }
        catch (Exception e)
        {
            _logger.LogError("Error in online tagger: " + e.Message + " || CancellationRequested: " + stoppingToken.IsCancellationRequested)
        }

        await Task.Delay(TimeSpan.FromSeconds(_runIntervalSeconds), stoppingToken);
    }
}

```

Rysunek 1.24 Nadpisana metoda ExecuteAsync() w klasie OnlineTagger.cs

OnlineTagger jest przykładem bardzo prostej implementacji zadania działającego w tle, lecz zaprezentowany wzorzec pozwala również tworzyć bardziej złożone rozwiązania. Przykładem takiej klasy może być LdapScraper, odpowiedzialny za pobieranie profesorów z systemu LDAP, w celu posiadania aktualnego katalogu.

Logikę tej klasy musimy wykonywać w znacznie większych odstępach czasu – jako iż katalog LDAP jest uaktualniany semestralnie, rozsądny interwał stanowi pół roku. Skierowanie tak długiego opóźnienia do metody Task.Delay() byłoby niezgodne z przeznaczeniem metody, jak iż nie jest ona dostosowana do działania na krótszych odstępach czasu. Lepszym wzorcem służącym do wykonywania zadań w tak długich odstępach jest standard Cron.

Cron jest wywodzącym się z systemów Unixowych standardem opisu interwałów czasu. Pozwala on na łatwe definiowanie okresowości. Składa się z pięciu wyrazów, które odpowiadają odpowiednio za: minutę, godzinę, dzień, miesiąc oraz dzień tygodnia wykonania zadania. Istnieje również sześciocyfrowy standard, który pozwala również na zdefiniowanie roku.

Przykładowym wyrażeniem użytym do definicji interwału w klasie LdapScraper jest „0 0 1 4,11 *”. Oznacza ono wykonanie o północy, pierwszego dnia kwietnia oraz listopada, dowolnego dnia tygodnia.

Do implementacji tego standardu w .NET Core 2.0 możemy użyć pakietu NuGet NCrontab. Pozwala on na stworzenie obiektu klasy CrontabSchedule. Do jego konstruktora podajemy wyrażenie Cron, a następnie za pomocą metody GetNextOccurrence możemy w łatwy sposób uzyskać datę następnego wystąpienia.

Dodatkowo, klasa implementuje retryCounter. Jest to obiekt, który odpowiada za zliczanie prób wywołania metody. Jeśli podczas zadanego wykonania metody wystąpi błąd, po określonym interwale czasu metoda spróbuje wykonać się raz jeszcze, zliczając kolejną próbę. Po określonej ilości prób program zakończy działanie, pobierając datę następnego wykonania z wyrażenia Cron. Dzięki temu chwilowe problemy nie sprawią, iż zadanie nie zostanie

wykonane (np. przejściowy brak połączenia z bazą danych), lecz jednocześnie nie sprawi, iż serwer będzie podejmował próby wykonania zadania w nieskończoność, obciążając się obliczeniowo - w przypadku, jeśli problem wymaga ingerencji programisty. Informacje o podejmowanych próbach i błędach są zapisywane do logów serwera.

Wszystkie parametry wykonania klasy są zawarte w pliku appsettings.json, dzięki czemu dostęp do nich jest prosty i łatwo konfigurowalny, bez potrzeby dokładnej znajomości kodu.

```
"LdapScraper": {  
    "Retries": 3,  
    "Cron": "0 0 1 4,11 *",  
    "RunIntervalHours": 3  
},
```

Rysunek 1.25 Ustawienia dla metody LdapScraper w appsettings.json

```
protected override async Task ExecuteAsync(CancellationToken stoppingToken)  
{  
    while (!stoppingToken.IsCancellationRequested)  
    {  
        try  
        {  
            var now = DateTime.Now;  
  
            if (now > _nextRun)  
            {  
                _logger.LogInformation("LdapScraper Service is Running");  
  
                await _ldapService.DeleteUsersOutOfLdap();  
                await _ldapService.ScrapAllFaculty();  
                await _tutorTabService.ScrapAllTutorTabs();  
  
                _retryCounter = 0;  
                _nextRun = _schedule.GetNextOccurrence(DateTime.Now);  
            }  
        }  
        catch (Exception e)  
        {  
            _retryCounter++;  
            _logger.LogError("LdapScraper error: " + e.Message + " || CancellationRequested: " +  
                + stoppingToken.IsCancellationRequested + " || Attempt " + _retryCounter);  
  
            if (_retryCounter >= _appConfiguration.BackgroundTasks.LdapScraper.Retries)  
            {  
                _logger.LogError("LdapScraper - all retries failed. Aborting and rescheduling. " +  
                    + _retryCounter + " attempts made");  
                _retryCounter = 0;  
                _nextRun = _schedule.GetNextOccurrence(DateTime.Now);  
            }  
        }  
  
        await Task.Delay(TimeSpan.FromHours(_runIntervalHours), stoppingToken);  
    }  
}
```

Rysunek 1.26 Metoda ExecuteAsync klasy LdapScraper

Ostatnim krokiem w implementacji rozwiązania jest jego rejestracja w klasie Startup.cs. Klasy stanowiące zadania działające w tle dodajemy jako HostedService – stanowiący cukier syntaktyczny dla typu Transient. Jest to jeden z cyklów życia usług, dostępnym w ASP.NET Core 2.1. Wszystkimi dostępnymi typami są:

- **Transient** – nowa instancja jest tworzona za każdym razem, kiedy występuje żądanie. Jest więc ona różna w obrębie jednego zapytania.
- **Scoped** – nowa instancja jest tworzona raz na każde zapytanie. Jest więc ona wspólna w obrębie zapytania, lecz różna w obrębie dwóch różnych zapytań.
- **Singleton** – nowa instancja jest tworzona raz na całą aplikację. Jest więc ona wspólna dla wszystkich zapytań.

```
services.AddHostedService<OnlineTagger>();
services.AddHostedService<InactiveTagger>();
services.AddHostedService<LdapScraper>();
```

Rysunek 1.27 Rejestracja zadań działających w tle w aplikacji FindMyTutor

Zadania działające w tle nie powinny być zależne od zapytania, jednocześnie zapewniając nową instancję przy każdym żądaniu – dlatego odpowiednim cyklem życia takiego obiektu będzie Transient.

Wykorzystując opisane wyżej metody możemy zaimplementować w .NET Core 2.1 zadania w tle, które wykonają się w zadanym przez nas w formacie Cron interwale czasu. Łatwość ich konfiguracji zapewnia proste utrzymanie, a asynchroniczne wywołanie wydajne działanie. Nie jest to jedyny sposób na tworzenie zadań działających w tle – bardzo popularne rozwiązanie stanowi na przykład framework Hangfire. .NET Core 2.1 umożliwia tworzenie wydajnych i prostych rozwiązań problemów charakterystycznych dla zadań działających w tle, co sprawia, iż stanowi on świetny framework do tworzenia rozwiązań o takich wymaganiach.

1.5. Podsumowanie i wnioski

.NET Core stanowi dziś dojrzałą i rozbudowaną technologię do budowania aplikacji serwerowych. Pozwala na wykorzystanie najnowszych praktyk programistycznych i wzorców projektowych, w oparciu, o które stworzymy szybkie i multiplatformowe rozwiązania. Tworzenie framework'a w modelu open-source pozwala na rozwój przy udziale całej społeczności, a narzędzia konsolowe zapewniają wygodną drogę dla wielu programistów preferujących ten styl pisania aplikacji, w oparciu o IDE własnego wyboru. Z tego względu serwer aplikacji FindMyTutor został zbudowany w oparciu o tę właśnie technologię, która pozwoliła stworzyć wydajne rozwiązanie dostosowane do potrzeb użytkowników aplikacji.

Rozdział 2

2. Aplikacje wieloplatformowe jako alternatywa dla aplikacji natywnych

2.1. Wstęp

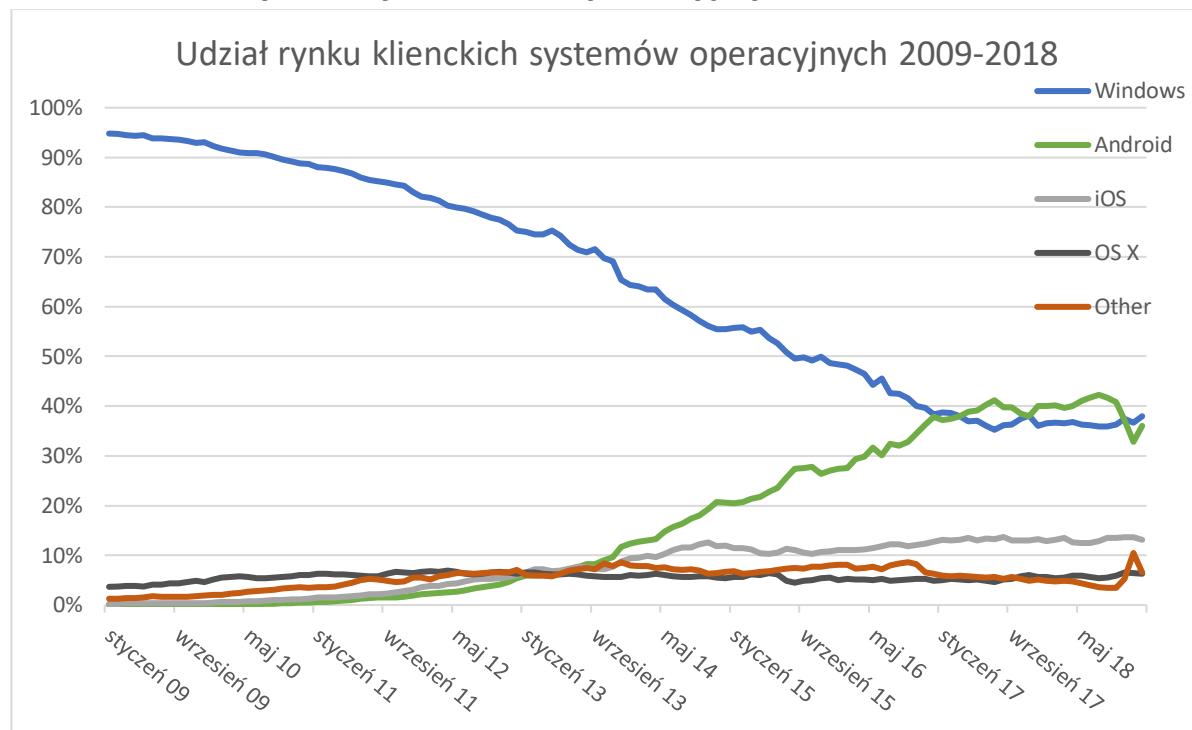
Urządzenia mobilne stały się nieodłącznym elementem naszego życia prywatnego oraz zawodowego. Dzięki smartfonom czy tabletom, jesteśmy w stanie w znaczny sposób zwiększyć swoją produktywność zawodową, uzyskać dostęp do dowolnych usług w celu zaspokojenia swoich potrzeb konsumenckich. Aplikacje systemów mobilnych stały się ogólnodostępnym narzędziem ułatwiającym dostęp do ofert niemalże każdego usługodawcy. Rynek aplikacji mobilnych obejmuje niemalże połowę rynku aplikacji konsumenckich i wciąż się rozwija. Tworząc aplikację na urządzenia mobilne, problematyczny jest wybór sposobu implementacji produktu między aplikacją natywną lub wieloplatformową.

Aplikacje natywne są tworzone pod konkretny system operacyjny, który musi spełniać pewne założenia środowiska uruchomieniowego (ang. runtime) z którym się komunikuje, implementacja może się różnić również językiem programowania, a co za tym idzie – deweloper jest zmuszony do tworzenia tej samej aplikacji wielokrotnie, z pomocą odpowiednich narzędzi deweloperskich i języków programowania

Aplikacje wieloplatformowe biblioteki narzucają wyższy poziom abstrakcji oprogramowania, dzięki czemu programista pisze jeden kod źródłowy na wiele systemów operacyjnych – następnie ta biblioteka jest odpowiedzialna za sprowadzenie kodu do postaci natywnej danego systemu operacyjnego. Technologie wieloplatformowe umożliwiają pokrycie dużej części rynku. W czasach, kiedy dostarczenie rozwiązania w jak najkrótszym czasie oraz na jak najwięcej urządzeń ma kluczowe znaczenie z perspektywy biznesu. Firmy nieustannie poszukują rozwiązań pozwalających na rozwijanie produktu z użyciem jak najmniejszej ilości technologii w celu optymalizacji kosztów rozwoju oraz utrzymania. Aplikacje wieloplatformowe wydają się idealnym rozwiązaniem do projektów o małej złożoności, jeden kod źródłowy - wiele platform. Wymagają jednak większego nakładu pracy we wczesnych etapach tworzenia produktu pod względem architektury, znaczące jest również doświadczenie zespołu deweloperskiego.

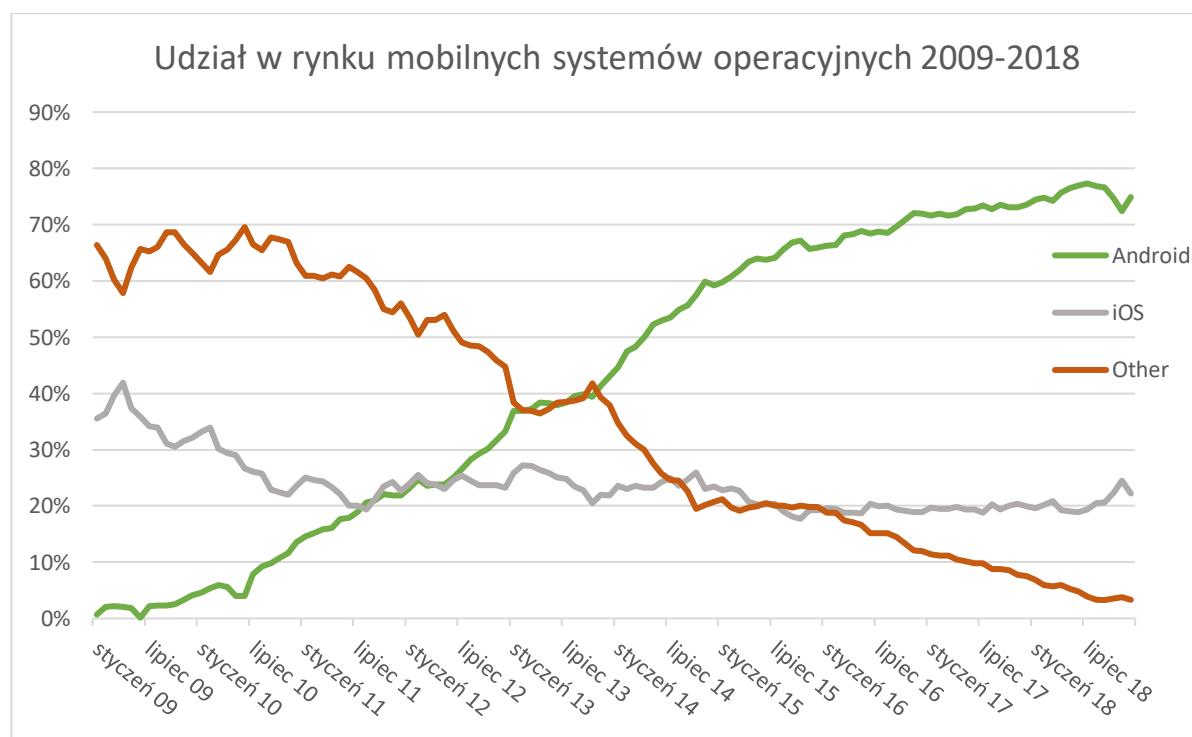
Rozdział ma na celu przedstawianie wieloplatformowych w kontekście mobilnych systemów operacyjnych, zaprezentowanie dostępnych bibliotek wieloplatformowych oraz ich wad oraz zalet.

2.2. Analiza Rynku systemów operacyjnych



Rysunek 2.1 Udział rynku klienckich systemów operacyjnych

Źródło Statcounter



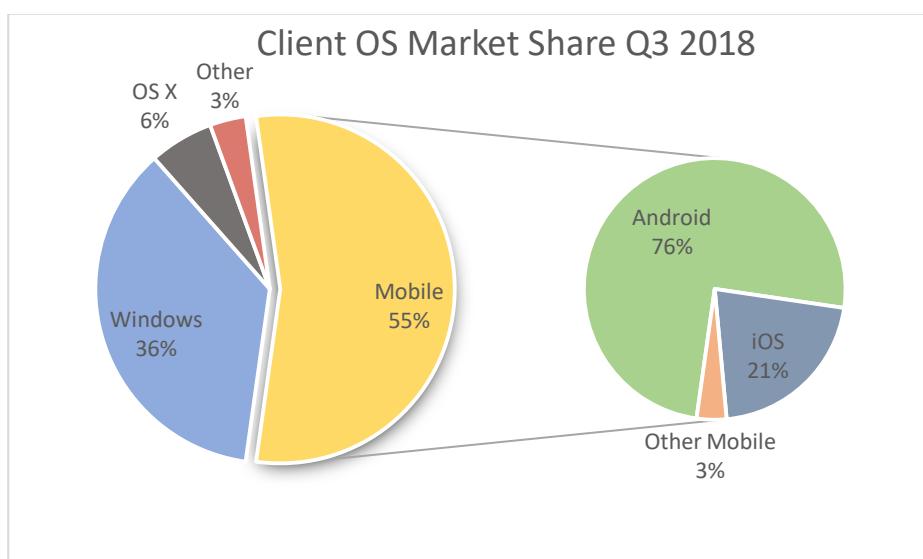
Rysunek 2.2 Udział w rynku mobilnych systemów operacyjnych

Źródło: Statcounter

W 2007 roku „krajobraz” rynku systemów mobilnych całkowicie się zmienił wraz z pierwszym iPhone'em. Urządzenie Apple było innowacyjne na wiele sposobów. Na początek było to urządzenie skierowane do zwykłego użytkownika. Jego powierzchnia ekranu dotykowego może być obsługiwana palcem, a nie rysikiem, jak każdy inny smartfon, który pojawił się wcześniej. Pomyślowość marketingowa firmy wywołała rewolucję w smartfonie i była przełomowa dla rynku. Informacja prasowa, która mówi, że Apple "wymyśliła telefon na nowo", okazała się trafna [9].

Z drugiej strony, Android firmy Google miał znacznie wolniejsze początki. Przez pierwsze kilka lat były one wyboiste, ale byli tacy, którzy wierzyli, że otwarta natura systemu operacyjnego będzie korzystna w dłuższej perspektywie. W 2010 r. idea otwartości systemu Androida została potwierdzona, gdy popularność Androida gwałtownie wzrosła. W ostatnim kwartale 2010 roku Symbian został zdeterminowany z wiodącej pozycji w rynku mobilnym [10].

Na początku 2012 roku Android został oficjalnym przodownikiem na światowym rynku smartfonów. Pomimo wielu prób dołączenia do rynku mobilnych systemów operacyjnych podjętych przez inne firmy, takich jak Nokia i Microsoft z systemem Windows Phone, to iOS i Android pozostały na stanowisku liderów aż do dziś z 97% udziałem rynkowym.



Rysunek 2.3 Kwartal Q3 2018
Źródło statcounter

Dominację Androida w rynku systemów mobilnych potwierdza również wykres kwartalny z września 2018.

Z analizy zaprezentowanych danych statystycznych wynika, że Android jest najbardziej rozpowszechnionym mobilnym systemem operacyjnym. Sprzedaż smartfonów z Androide rośnie z roku na rok.

Dane statystyczne wskazują także na wyraźny trend spadkowy w kwestii sprzedaży smartfonów z innymi niż wiodące systemami mobilnymi, takich jak Windows Phone.

2.2.1. Podsumowanie analizy

W 2015 roku niezależne przedsiębiorstwo analityczno-badawcze Gartner – specjalizujące się w zagadnieniach strategicznego zarządzania i wykorzystywania technologii, opublikowała badania, prognozujące, że do końca 2017 roku popyt rynkowy na usługi rozwoju aplikacji mobilnych wzrośnie co najmniej pięć razy szybciej niż zdolność wewnętrznych organizacji IT do ich dostarczenia.

Z badań tej samej firmy z czerwca 2017, zapotrzebowanie na aplikacje mobilne wzrasta wykładniczo, do 2022 roku, 70% wszystkich interakcji aplikacji typu enterprise, będzie zachodzić na urządzeniach mobilnych [11].

Na rynku systemów mobilnych przoduje dwóch liderów – Google (Android) oraz Apple (iOS), żadna inna firma nie stanowi dla nich konkurencji.

2.3. Platformy Mobilne

2.3.1. Android

Android to system operacyjny oparty na jądrze systemu Linux z architekturą procesorów ARM, x86, x86-64 lub podobne, który jest obecnie własnością Google. W 2005 roku Google kupił wyrafinowany system operacyjny od Android Inc., którego właścicielem był Andy Rubin. Przejęcie Androida było pierwszym krokiem gigantycznej firmy Google, która wkroczyła w świat telefonów komórkowych. 22 października 2008 r. Pierwszy telefon komórkowy z systemem Android został wprowadzony na rynek jako HTC Dream [12]. Po pierwszej wersji w 2008 roku wprowadzono wiele aktualizacji dodawanych po kolej, dodając nowe funkcje i funkcje. Android wydaje główne wersje w kolejności alfabetycznej (nazwy słodkości), obecnie najnowszym systemem wersji 9 Android jest Oreo.

Nazwa kodowa	Numery wersji	Wersja jądra Linux	Data pierwszego wydania	Poziomy API
Brak nazwy	1.0	?	23 września 2008	1
Petit Four	1.1	2.6	9 lutego 2009	2
Cupcake	1.5	2.6.27	27 kwietnia 2009	3
Donut	1.6	2.6.29	15 września 2009	4
Eclair	2.0-2.1	2.6.29	26 października 2009	5-7
Froyo	2.2-2.2.3	2.6.32	20 maja 2010	8
Gingerbread	2.3-2.3.7	2.6.35	6 grudnia 2010	9-10
Honeycomb	3.0-3.2.6	2.6.36	22 lutego 2011	11-13
Ice Cream Sandwich	4.0-4.0.4	3.0.1	18 października 2011	14-15
Jelly Bean	4.1-4.3.1	3.0.31 - 3.4.39	9 lutego 2012	16-18
KitKat	4.4-4.4.4	3.10	31 października 2013	19-20
Lollipop	5.0-5.1.1	3.16	12 listopada 2014	21-22
Marshmallow	6.0-6.0.1	3.18	5 października 2015	23
Nougat	7.0-7.1.2	4.4	22 sierpnia 2016	24-25
Oreo	8.0-8.1	4.10	21 sierpnia 2017	26-27
Pie	9.0	4.4.107, 4.9.84, 4.14.42	06 sierpnia 2018	28

Legenda: Stara wersja – brak wsparcia, Stara wersja - wsparcie, Najnowsza wersja

Tabela 2.1 Historia wersji systemu Android [13]

Google rozpoczęło współpracę z takimi firmami jak LG i HTC, aby rozszerzyć rynek Androida, po siedmiu latach rozwoju systemu, odnotowano ponad 1200 firm rozwijających urządzenia Android [14]. Obecnie system operacyjny nie jest używany tylko w telefonach komórkowych, ale także w urządzeniach takich jak tablety, telewizory, lodówki i inne. W początkowej fazie rozwoju Nokia była głównym rywalem, ponieważ telefony z systemem Symbian były najczęściej używane przez ludzi.

Instalacja aplikacji na urządzeniach z systemem Android jest możliwa poprzez sklepy z aplikacjami lub z nieznanego źródła tzw. sideloading.

Sideloading polega na umieszczeniu pliku instalacyjnego na urządzeniu oraz instalacji po ówczesnym nadaniu odpowiednich uprawnień w ustawieniach systemu.

Najpopularniejszym sklepem z aplikacjami dla systemu Android jest Google Play (wcześniej nazwa - Android Market). Oprócz możliwości pobierania aplikacji darmowych i płatnych, oferuje również sprzedaż e-booków, muzyki oraz filmów. Dostępnych jest ponad 2.7 miliona aplikacji [15]. Każda aplikacja jest sprawdzana pod kątem złośliwego oprogramowania. Alternatywami Google Play są sklepy takie jak Galaxy Apps (Samsung), Amazon Appstore czy F-droid.

2.3.1.1. *Android Studio*

Darmowe, zintegrowane środowisko programistyczne (IDE) oparte na IntelliJ IDEA od firmy JetBrains na systemy Linux, Windows oraz MacOS. Celem Android Studio jest pomoc programistom w zwiększeniu produktywności i pomóc w procesie rozwoju. Android Studio ma wbudowaną obsługę usług Google, takich jak aparat aplikacji Google i usługa notyfikacji Firebase Cloud Messaging. Android Studio zapewnia także rozszerzone wsparcie dla różnych urządzeń, takich jak smartfony, urządzenia przenośne i Android TV. Przesyłanie aplikacji na Androida do Google Play wymaga tylko kilku kroków: Android Studio daje możliwość pobrania pliku .apk. Użytkownik musi następnie przesłać plik na osobiste konto sklepu Google Play. Android Studio jest rekomendowanym narzędziem deweloperskim, lecz nie jedynym.

2.3.2. *iOS*

iOS opracowany przez Apple Inc. to system operacyjny dla smartfonów i innych urządzeń, takich jak iPad (tablet) i iPod (przenośny odtwarzacz muzyki). System operacyjny był wcześniej znany jako iPhone OS, bazujący na jądrze XNU systemu Darwin oraz usług zaczerpniętych z systemu FreeBSD. System operacyjny po raz pierwszy ujrzał światło dzienne w roku 2007, wprowadzając nową erę smartfonów. Nie mając silnego konkurenta na rynku, liczba sprzedanych iPhone'ów wyniosła 70 milionów do roku 2010 [3.] Począwszy od 2007 roku, istnieje wiele wersji systemu operacyjnego uruchamianych w systemie iOSX (X to numer wersji). Najnowsza wersja systemu iOS jest 12. iOS to jeden z przodujących systemów mobilnych na świecie i drugi popularny po systemie Android. Pod względem ilości tabletów był liderem na rynku do roku 2013, gdy tablety z Androidem wyprzedziły jego sprzedaż.

Instalacja aplikacji na urządzenia iPhone jest w pełni zcentralizowana – nie ma możliwości instalacji aplikacji z innego źródła. Sklep z aplikacjami „App Store”, dostępnego jest ponad 2.2 miliona aplikacji [16] i odpowiada za podpisywanie aplikacji, sprawdzanie czy nie jest szkodliwa, wydanie oraz aktualizacje.

2.3.2.1. *Xcode*

Oficjalne zintegrowane środowisko programistyczne Apple (IDE) dla programistów komputerów Mac i iOS. Jest to jedynie oprogramowanie, które umożliwia programistom pisanie kodu działającego na produktach Apple. Podobnie jak w Android Studio, Xcode oferuje wsparcie dla różnych urządzeń Apple, chociaż Xcode działa tylko na systemach operacyjnych MacOS, nie może działać na żadnym innym systemie operacyjnym. Ponadto Xcode odgrywa istotną rolę w publikacji aplikacji. Apple ma skrupulatny sposób sprawdzania kodu programistów, z różnymi krokami, które rozpoczynają się bezpośrednio na Xcode przed faktycznym przesłaniem aplikacji na serwery Apple.

2.3.3. Kod natywny

Rozwój aplikacji mobilnych możliwy jest za pomocą dostarczonych przez producenta narzędzi deweloperskich i wymaga znajomości natywnych języków programowania danej platformy w celu osiągnięcia wydajnej aplikacji

Czym dokładnie jest kod natywny?

Zwykle jest skompilowany, co czyni go szybszym niż języki interpretowane, takie jak JavaScript. Z natywnym kodem, odwołujemy się bezpośrednio do natywnego API platformy, udostępnionego przez twórcę.

2.3.3.1. *Android*

W nomenklaturze środowiska Android – kodem natywnym nazywamy skompilowany kodem Java (Kotlin) do kodu bajtowego(maszynowego), który jest wykonywany przez wirtualną maszynę Javy (ART. lub wcześniej Dalvik) [17].

2.3.3.2. *iOS*

Kod natywny to skompilowany kod Objective-C(Swift) i jest wykonywany bezpośrednio przez środowisko wykonawcze Objective-C [18].

2.4. Wieloplatformowość: hybrid vs cross-platform apps

Technologie wieloplatformowe można podzielić na dwa rodzaje pod względem sposobu realizacji aplikacji. Są to środowiska hybrydowe oparte na technologiach webowych oraz środowiska cross-platformowe natywne.

2.4.1. Mobilne aplikacje hybrydowe

Aplikacje hybrydowe są podobne do aplikacji natywnych, pobierane są z tego samego źródła, mają dostęp do tych samych natywnych funkcji oraz sprzętowej akceleracji wydajnościowej (ang. hardware-based performance acceleration), tak jak aplikacja zbudowana przy pomocy natywnego SDK.

Kluczową różnicą jest fakt, że aplikacje hybrydowe bazują na technologiach webowych, takich jak JavaScript, HTML oraz CSS zamiast specjalistycznych języków używanych przez iOS (Obj-C, Swift) czy Android (Java, Kotlin). Oznacza to, że każdy, kto ma zestaw umiejętności programistów aplikacji internetowych, może zbudować aplikację przy użyciu podejścia hybrydowego.

Aplikacje hybrydowe działają w trybie pełnoekranowym przeglądarki internetowej (ang. WebView), która jest niewidoczna dla użytkownika – naszą aplikacją jest de facto aplikacja internetowa, sprawiająca wrażenie aplikacji mobilnej. Dzięki dostosowanym natywnym wtyczkom mają one dostęp do rodzimych funkcjonalności określonych urządzeń mobilnych (takich jak kamera lub TouchID), bez kluczowego kodu związanego z tym urządzeniem [19].

2.4.2. Mobilne aplikacje cross-platformowe

Aplikacje cross-platformowe, mimo stosowanych na ogół tych samych technologii, co w przypadku aplikacji hybrydowych (tj. JavaScript, HTML i CSS), w odróżnieniu od aplikacji hybrydowych, nie bazują na przeglądarce internetowej. Nowoczesne biblioteki cross-platformowe, podczas budowania aplikacji na wybrany mobilny system operacyjny (iOS, Android), interpretują kod źródłowy napisany w JavaScript, na kod konkretnej platformy – Swift dla iOS/Swift bądź w przypadku Androida – Java/Kotlin. Dzieje się to za pomocą tzw. API (Application Programming Interface) danej biblioteki, gdzie są jasno zdefiniowane powiązania dla natywnego iOS API oraz Android API. Dla przykładu, gdy stosujemy przycisk, którego nazwą klasy jest `<Button>` (w API biblioteki), biblioteka dostarczy odpowiednią klasę interfejsu użytkownika podczas budowania aplikacji dla iOS – klasa `UIButton` oraz dla Android – klasa `ANDROID.WIDGET.BUTTON`.

Według przeprowadzonych badań w roku 2017 przez największy serwis społecznościowy dla programistów StackOverflow, zaledwie 6.5% developerów miało styczność z Swift i Objective C, dla nadania kontrastu, aż 62.5% społeczności korzystało z języka JavaScript (który pojawił się jako najczęściej używany język programowania w badaniu) [20].

Cecha	Aplikacje Natywne	Aplikacje wieloplatformowe
Wymagany zestaw umiejętności programistów	Obj-C, Swift z iOS SDK Java, Kotlin z Android SDK	HTML, CSS, JavaScript
Metoda dystrybucji	Sklepy z aplikacjami	Sklepy z aplikacjami,
Szybkość rozwoju aplikacji	Wolny	Szybki
Koszt tworzenia	Wysoki	Niski
Utrzymanie Aplikacji	Wysoki	Średni/Niski
Wydajność Graficzna	Wysoka	Średnia
Wydajność Aplikacji	Wysoka	Zależna od przypadku użycia
Dostęp do natywnych funkcjonalności	Pełny	Zależny od dodatkowych bibliotek
Spójność UX	Osobne Aplikacje	Tak

Rysunek 2.4 Zestawienie różnic między mobilnymi aplikacjami natywnymi a wieloplatformowymi

		Xamarin	React Native	Native Script	Ionic (Cordova)
Kod Aplikacji		C# ze środowiskiem .net	JavaScript + Swift, Objective-C lub Java, Kotlin	JavaScript + TypeScript	HTML, CSS JavaScript + TypeScript
Kompilacja	iOS	AOT	Interpreter	Interpreter	JIT + WKWebView
	Android	JIT/AOT	JIT	JIT	JIT
Przenoszalność		iOS Android Mac OS, Windows	iOS Android	iOS Android	iOS Android
Współdzielenie kodu		Do 96%	Do 70%	Do 90%	Do 100%
UI		Natywne kontrolery UI	Natywne kontrolery UI	Natywne kontrolery UI	HTML, CSS
Gwiazdki GitHub		10k	72k	16k	36k
Społeczność		Duża	Bardzo Duża	Mała	Duża
Cennik		Open-Source (MIT)/ Visual Studio licencja komercyjna \$539-2999	Open-Source (MIT)	Open-Source (Apache 2.0)/ Sidekick cloud service \$19-249	Open-Source (MIT)
Przypadek użycia aplikacji		Proste	Wszystkie	Wszystkie	Proste
Wydajność		Zbliżona do natywnej	Zbliżona do natywnej	Zbliżona do natywnej	Słaba Umiarkowana
Popularne aplikacje		Skulls of the Shogun, Alaska Airlines, Storyo, Olo, The World Bank	Facebook Ads Manager, Instagram, Pinterest, Bloomberg, UberEATS, Discord	MyPuma, California Court Access App	JustWatch, Pacifica, Nationwide

Rysunek 2.5 Porównanie najpopularniejszych technologii wieloplatformowych na urządzenia mobile

2.4.2.1. Runtime Environment - Środowisko Uruchomieniowe

Każda z poszczególnych platform mobilnych posiada swój własny ekosystem oraz oferuje zupełnie inne narzędzia developerskie oraz języki programowania – Java (Kotlin) dla Android i Objective-C(Swift) dla iOS. Aby przetłumaczyć kod JavaScript na odpowiednie natywne API, potrzebne jest dostarczenie pewnego rodzaju mechanizmu proxy(mostu). To właśnie za to odpowiedzialne są części „Runtime” we bibliotekach cross-platformowych. Środowisko wykonawcze iOS (Android) można uważać za „most” między światem JavaScript a światem iOS (Android). Aplikacja cross-platformowa dla iOS (Android) jest standardowym pakietem natywnym .IPA (.APK), w którym oprócz plików JavaScript osadza także środowisko wykonawcze.

2.4.2.2. Interpreter a Kompilator

Interpreter jest programem komputerowym, który bezpośrednio wykonuje instrukcje napisane w języku programowania lub skryptowym, bez konieczności uprzedniego ich skompilowania do programu języka maszynowego. Interpreter zazwyczaj używa jednej z następujących strategii do realizacji programu:

1. analizuje kod źródłowy (ang. parse) i wykonywa go bezpośrednio
2. tłumaczy kod źródłowy na wydajną reprezentację pośrednią i natychmiast go wykonuje
3. jawnie wykonuje zapisany, skompilowany kod utworzony przez kompilator, który jest częścią systemu interpretera

Kompilator to program, który tłumaczy kod źródłowy z języka programowania wysokiego poziomu na język niższego poziomu (np. Język asembler, kod obiektowy lub kod maszynowy) w celu utworzenia wykonywalnego programu. W środowisku Android kod kompilowany jest do kodu byteowego, który wykonywany jest przez wirtualną maszynę Javy (JVM) [21].

Podstawową różnicą jest to, że kompilator, generuje samodzielny program kodu maszynowego, podczas gdy interpreter wykonuje działania opisane przez program wysokiego poziomu.

Kod źródłowy programu jest kompilowany z wyprzedzeniem (AOT - Ahead-of-time compilation) i przechowywany jako niezależny od systemu operacyjnego kod, który jest następnie łączony w czasie wykonywania aplikacji i wykonywany przez interpreter (iOS) [22] lub kompilator - dla systemów z kompilacją typu Just-in-time (Android) [23].

2.4.2.3. Silnik JavaScript

JavaScript Engine, zwany również JavaScript Virtual Machine - program lub interpreter wykonujący kod JavaScript z poziomu natywnej aplikacji mobilnej.

Podstawowym zadaniem silnika JavaScript, jest wczytanie i komplikacja kodu JavaScript, który może być interpretowany lub osadzony w aplikacji. Dodatkowo silnik JS analizuje, optymalizuje, interpretuje kod JS, zawiera również garbage collector.

Dokładniej, każdy silnik JavaScript implementuje wersję ECMAScript, której JavaScript jest dialektem. W miarę ewolucji ECMAScript, ewoluują również silniki JavaScript. W obydwu systemach mobilnych (iOS, Android) mamy dostarczony silnik JS. Framework odpowiada za obsługę operacji na obiekcie JS oraz dynamiczne wywołuje interfejsy API iOS (Android) w razie potrzeby.

2.4.2.3.1. Android – Chrome V8

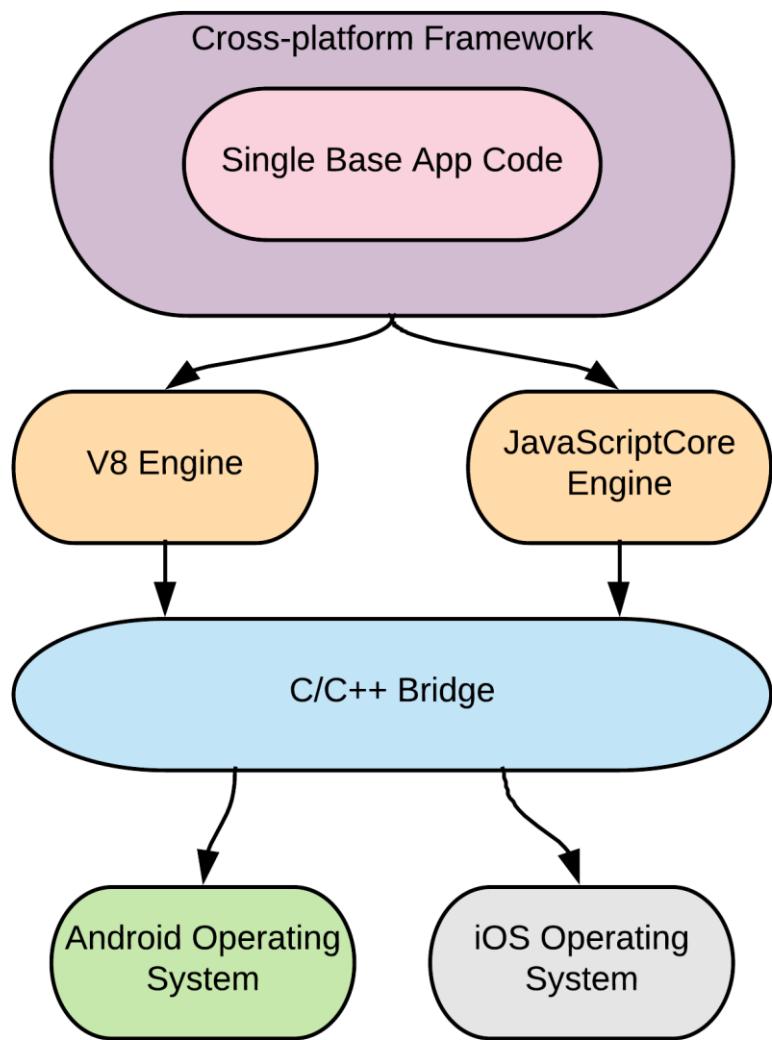
Jest odpowiedzialny za przetwarzanie kodu JavaScript. Dobrą analogią jest Node.js - przetwarza JavaScript i tłumaczy go na bazowe systemowe API. Maszyna wirtualna jest ładowana w procesie aplikacji i działa w głównym wątku UI. V8 używa techniki kompilacji JIT (Just-In-Time) do przetwarzania kodu JavaScript [24].

2.4.2.3.2. iOS – JavaScriptCore

W odróżnieniu od Chrome V8, JavaScriptCore zapewnia możliwość ewaluacji(interpretacji) kodu JavaScript w aplikacjach Swift, Objective-C i C. Ze względów bezpieczeństwa, Apple narzuciło ograniczenie kompilacji metodą JIT, dzięki któremu wykonywanie dynamicznie generowanego kodu nie jest możliwe. JavaScriptCore stosuje metodę kompilacji AOT ze względu na narzucone restrykcje [25].

2.4.2.4. Bridge

Bridge zapewnia komunikację między silnikami JavaScript i Systemami Operacyjnymi. W przypadku Androida, komunikacje między kodem C++ a Java API zapewnia Java Native Interface (JNI), natomiast iOS - kod C++ może bezpośrednio wywoływać interfejsy API Objective-C.



Rysunek 2.6 Uogólniona budowa mostu rozwiązań wieloplatformowych

Biblioteki cross platformowe polegają w dużej części na zaimplementowaniu takiego mostu – tak zwany middleware, który jest asynchronicznym kodem zapewniającym komunikacje między API [26].

2.4.3. Rozwiązania wieloplatformowe

2.4.3.1. *Progressive Web Apps (PWA)*

Progressive web apps - jest to zbiór zasad dla aplikacji internetowych, dzięki której aplikacja sprawia wrażenie natywnej. Zbiór zawiera wytyczne odnośnie UX, architektury, jak i również zbioru technologii [27].

Fundament/Koncept PWA obejmuje:

- ***Niezawodność (Reliable)***

Po uruchomieniu aplikacji z ekranu głównego użytkownika, service worker umożliwiają uruchomienie aplikacji webowej, niezależnie od stanu sieci, a nawet bez dostępu do sieci – w trybie offline.

Service Worker - napisany w języku JavaScript, uruchamiający się cyklicznie, działa jak serwer proxy po stronie klienta i zapewnia kontrolę nad pamięcią podręczną oraz sposób reagowania na żądania zasobów – synchronizuje dane w tle oraz sprawdza powiadomienia PUSH. Poprzez wstępne buforowanie kluczowych zasobów (pre-caching) można wyeliminować zależność od sieci, zapewniając natychmiastową i niezawodną obsługę dla użytkowników.

- ***Szybkość (Fast)***

Po załadowaniu, użytkownicy oczekują, że interfejs będzie działać szybko. 53% użytkowników opuści witrynę, jeśli załadowanie strony zajmie więcej niż 3 sekundy.

- ***Zintegrowany (Integrated)***

Aplikacja internetowa może zostać zainstalowana na ekranie głównym urządzenia, bez konieczności pobierania jej ze sklepu aplikacji (App Store lub Google Play).

Uproszczenie płatności dzięki API Payment Request [28], bez potrzeby uruchamiania stron trzecich.

- ***Ujmujący i Angażujący (Engaging and Retain)***

UX aplikacji jest starannie przemyślany, prosty i intuicyjny dla użytkownika. Odpowiednie wykorzystanie zarówno mechanizmu push z serwera, jak i notyfikacji, w celu wyświetlenia w określony sposób i w dogodnym momencie wartościowych informacji dla użytkownika jest kluczowym aspektem. Aplikacja powinna przyciągać użytkownika.

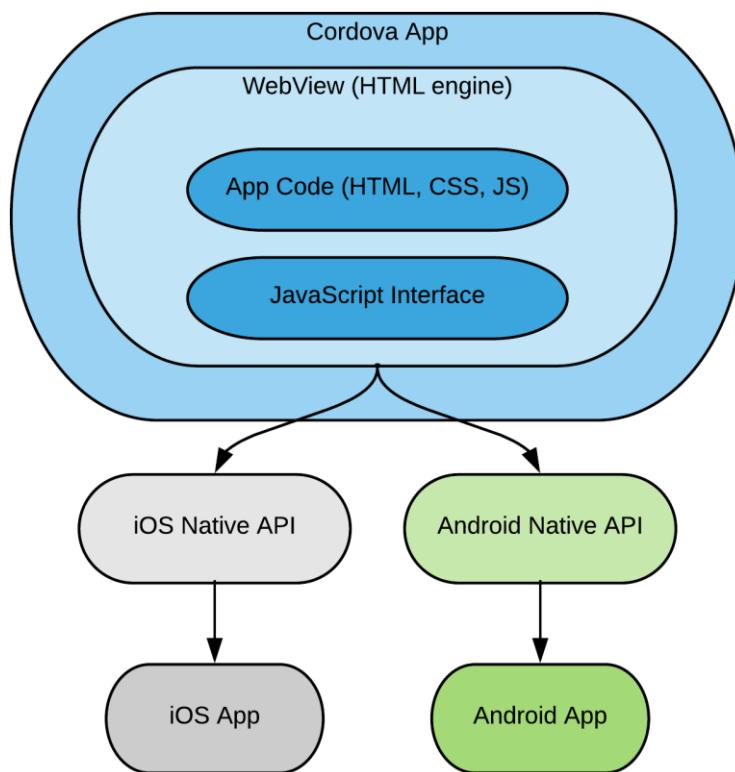
2.4.3.2. Ionic (Cordova)

Apache Cordova to biblioteka open-source do tworzenia hybrydowych aplikacji mobilnych, umożliwia korzystanie ze standardowych technologii webowych - HTML5, CSS3 i JavaScript.

Apache Cordova jest odpowiedzialny za budowanie paczki z aplikacją HTML5 w aplikacje natywne, która może działać na systemach Android, iOS i innych platformach za pomocą osadzania aplikacji w natywny widok takie jak WebView(Android) lub UIWebView(iOS).

Aplikacje są wykonywane w paczkach przeznaczonych dla każdej z platform i polegają na zgodnych z normami powiązaniach API w celu uzyskania dostępu do natywnych funkcji każdego urządzenia, takich jak kamera, GPS, sieć czy dostępu do plików.

Ionic to frontendowa, zoptymalizowana pod kątem urządzeń przenośnych biblioteka, dzięki której aplikacja może wyglądać jak natywna. Ionic umożliwia wykorzystanie nowoczesnych bibliotek *Single Page Application* (SPA) – dostępna jest oficjalna integracja z AngularJS oraz Angular. Wsparcie dla bibliotek Vue.js i React są w fazie rozwoju [29].



Rysunek 2.7 Architektura biblioteki Ionic Cordova

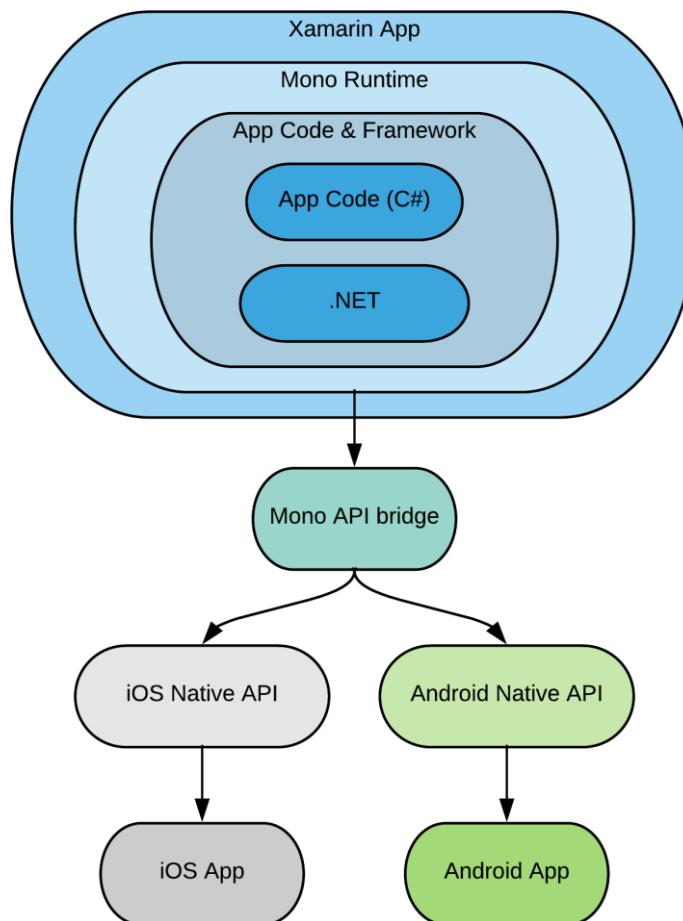
Ionic 2.0 umożliwia tworzenie aplikacji PWA, dzięki czemu deweloperzy aplikacji są w stanie dostarczyć produkt, który w znacznym stopniu wpływa pozytywnie na User Experience [30].

2.4.3.3. Xamarin

Xamarin jest platformą oferującą rozwiązania cross-platformowe umożliwiającą tworzenie oprogramowania przeznaczone dla różnych platform w języku C# za pomocą bibliotek .NET oraz zintegrowanego środowiska deweloperskiego (IDE) Visual Studio lub Xmarain Studio.

W lutym 2013 został wydany Xamarin 2.0 a wraz z nim Xamarin.Android i Xamarin.iOS, umożliwiający pisanie natywnych aplikacji na systemy Android oraz iOS w języku C#. Programiści mają możliwość ponownego wykorzystania istniejącego kodu C# oraz współdzielić go między różne platformy urządzeń.

Xamarin wydał także sklep z komponentami do integracji systemów backendowych, bibliotek zewnętrznych, usług w chmurze i elementów UI bezpośrednio dla aplikacji mobilnych.

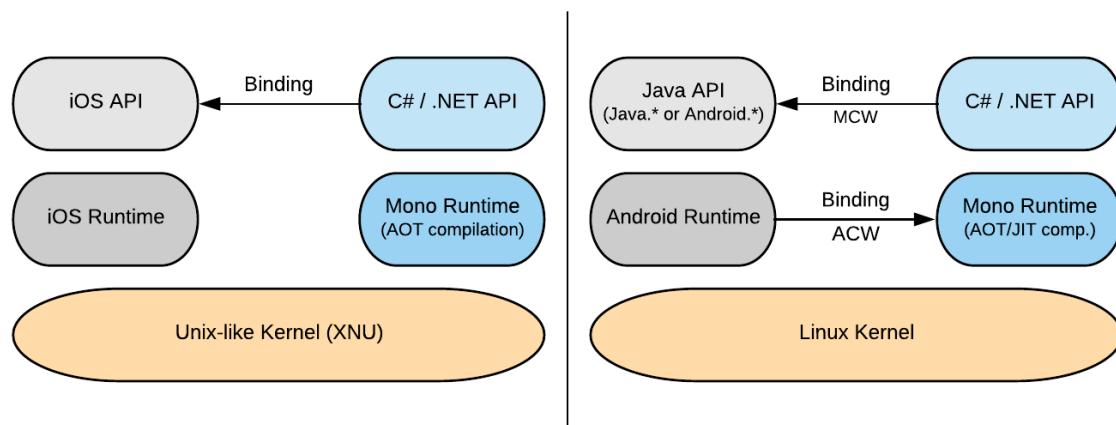


Rysunek 2.8 Architektura biblioteki Xamarin iOS i Android

2.4.3.3.1. Mono Runtime

Środowisko wykonawcze Mono implementuje wspólną infrastrukturę językową ECMA (CLI) dla języka C# i bibliotek .NET [31].

Mono stanowi środowisko wykonawcze aplikacji Xamarin.iOS i Xamarin.Android. Środowisko to, działa równolegle z Android Runtime (ART) w systemie Android [32] lub w przypadku iOS – równolegle ze środowiskiem wykonawczym iOS Runtime (ARM) [33].

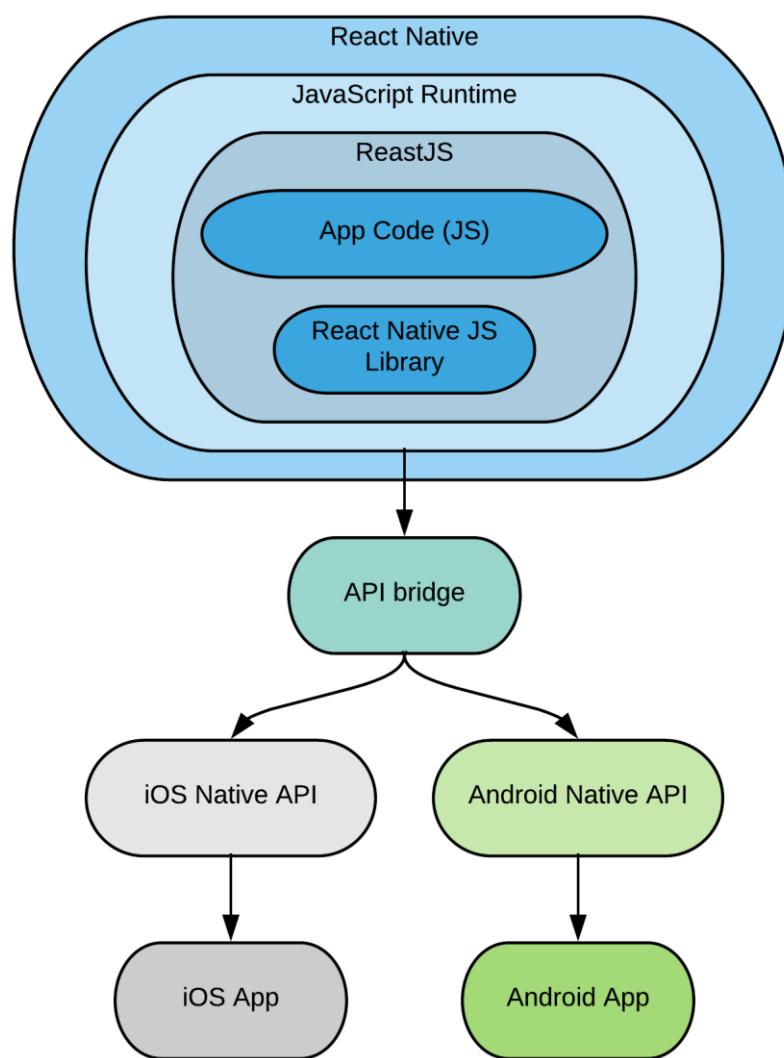


Rysunek 2.9 Mono Runtime [22]

2.4.3.4. *React Native*

React Native został wydany przez Facebooka w 2015 r., który zastosował architekturę biblioteki React do natywnych aplikacji Android, iOS i UWP.

Zasady działania React Native są w zasadzie takie same jak w React, z tym wyjątkiem, że nie manipuluje DOMem za pośrednictwem VirtualDOM, lecz na specjalnie przygotowanych, natywnych widokach. Proces drugoplanowy z ReactNative (który interpretuje kod JavaScript napisany przez deweloperów) bezpośrednio na urządzeniu i komunikuje się z natywną platformą za pośrednictwem sterylizowanego oraz asynchronicznego Mostu (ang. Bridge). Technologia ReactNative nie używa HTML, stosuje tylko JavaScript oraz natywne SDK.



Rysunek 2.10 Architektura ReactNative

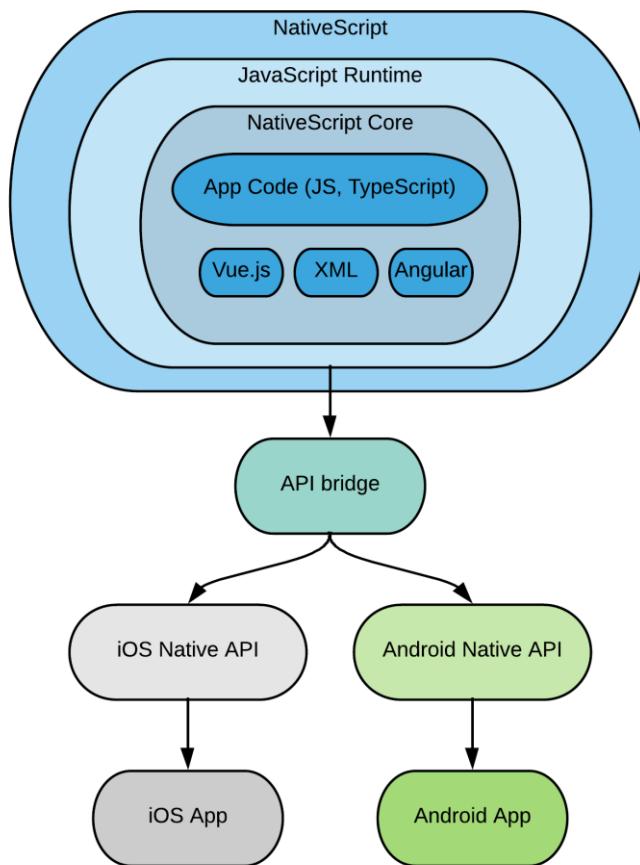
2.4.3.5. NativeScript

NativeScript to biblioteka open-source do tworzenia aplikacji na platformy Apple iOS i Android. Stworzony w roku 2015 i wspierany przez firmę Progress, specjalizującą się opracowaniach i wdrożeniach aplikacji biznesowych.

Budowanie interfejsu użytkowania może być budowanie za pomocą języka znaczników XML lub z pomocą nowoczesnych bibliotek SPA – Angular i Vue.js.

Schemat działania NativeScript jest zbliżony do ReactNative. Bazuje on na natywnych SDK platform, silniku JavaScript i asynchronicznym moście zapewniającym komunikacje z natywnym API.

W odróżnieniu od ReactNative, dostarcza również serwis SideKick umożliwiający testowanie, budowanie i wdrażanie aplikacji iOS w chmurze, bez potrzeby posiadania urządzenia Apple Mac.



Rysunek 2.11 Architektura NativeScript

2.5. Problematyka wieloplatformowości

2.5.1. Wydajność

Problem wydajnościowy aplikacji cross-platformowych, a w szczególności hybrydowych jest niewątpliwie jednym z najbardziej istotnych. Zwiększenie poziomu abstrakcji, jak i dodanie kolejnych zależności technologicznych, w znaczny sposób upraszcza rozwój aplikacji, lecz nigdy nie zwiększy wydajności rozwiązań natywnych.

Ciężkie jest również miarodajne porównanie bibliotek hybrydowych i cross-platformowych z natywnymi. Proste porównywanie wydajności poprzez zastosowanie dla przykładu funkcji haszującej SHA1 dla 60000 iteracji, funkcji sprawdzającej liczby pierwsze dla zakresu od 2 do 100000 czy wielu innych funkcji matematycznych, które skupiają się na wykonywaniu wielu operacji matematycznych niewątpliwie wykazałyby przewagę rozwiązań natywnych.

W realnych aplikacjach konsumentckich, które zawierają znacznie większą ilość kodu, środowisko uruchomieniowe musi radzić sobie z operacjami alokacji/zwalniania pamięci, przetwarzania łańcuchów znaków, tablic i wielu innych struktur danych, które nie znajdują to odzwierciedlenia w surowych wynikach matematycznych.

Kompilacja AOT, redukuje wiele wąskich gardeł (ang. bottlenecks) – redukcja RTTI (runtime type information), funkcje inline (wstawianie funkcji pod adres) czy doprecyzowanie kodu generycznego – przez co kod cross-platformowy całkiem dobrze pasuje do urządzenia.

Większość bibliotek cross-platformowych umożliwia dodawanie natywnych modułów danej platformy, jak i również pisanie kodu w natywnym języku danej platformy – Objective-C/Swift dla iOS oraz w przykładzie Android, Java/Kotlin (SDK) a nawet C/C++(NDK [34]).

2.5.2. Bezpieczeństwo

Aplikacje hybrydowe i cross-platformowe wywołują natywne API platformy. Bezpieczeństwo aplikacji zależy w dużej mierze zależy od twórców aplikacji, większość problemów związanych z bezpieczeństwem jest spowodowana przez samych programistów, którzy nie podejmują żadnych środków bezpieczeństwa. Technologie webowe i bramki płatnicze działają dobrze bez względu na to, że kod jest całkowicie otwarty i ktokolwiek jest go w stanie odczytać.

2.5.2.1. *Współczesna Kryptografia*

Nowoczesne biblioteki wieloplatformowe dostarczają biblioteki zapewniające dostęp do natywnego API, zbioru kryptograficznego, implementującego powszechnie stosowane standardy, takie jak Advanced Encryption Standard (AES), Rivest-Shamir-Adleman (RSA), Digital Signature Algorithm (DSA), and Secure Hash Algorithm (SHA) [35].

2.5.2.2. *Uwierzytelnianie i autoryzacja*

Dostępne interfejsy natywne API dla protokołów wyższego poziomu, takich jak Secure Socket Layer (SSL) i HTTPS, które wraz z protokołami do autoryzacji i uwierzytelniania SAML, OAuth2 wraz z OpenID Connect czy rozwiązania typu enterprise Azure Active Directory, zapewniają bezpieczną komunikację klient-serwer [36].

2.5.2.3. *Ochrona kodu*

Łatwo pobrać opublikowany pakiet aplikacji z publicznego sklepu iOS i Android. W takim przypadku każdy może otworzyć paczkę aplikacji i sprawdzić wszystkie zasoby – kod źródłowy, obrazy, wszystko, co jest częścią paczki.

Problem dotyczy również natywnych aplikacji na systemy iOS i Android – w Internecie dostępnych jest dużo darmowego oprogramowania do inżynierii wstępnej. Proces ten różni się w przypadku iOS i Androida, ze względu na różnicę w sposobie komplikacji kodu Java i Objective-C.

2.5.2.3.1. *Zaciemnianie kodu*

W systemie Android dodatkową warstwę zabezpieczeń zapewnia narzędzie o nazwie ProGuard. ProGuard jest częścią Android Studio i może zaciemnić kodu źródłowego Java, aby bardziej było go odtworzyć, ale znowu nie jest niemożliwe. Podobne rozwiązania dostępne są dla technologii .net, iOS i web:

Nazwa	Technologia	Język
Dotfuscator [37]	Xamarin .net	C#
Jscrambler [38]	Web, hybrid, cross-platform	JavaScript
UglifyJS [39]	Web, hybrid, cross-platform	JavaScript
Swiftshield [40]	iOS	Objective-C, Swift
iXGuard [41]	iOS	Objective-C, Swift

Rysunek 2.12 Zestawienie bibliotek do zaciemniania kodu

W rozwiązaniach wieloplatformowych, wykorzystujących język JavaScript jako języka aplikacji, który jest językiem interpretowanym, nie kompiluje się podczas procesu kompilacji, a pliki są dostępne jako część pakietu aplikacji - zaciemnianie kodu [42] ma również znaczny wpływ na ochronę własności intelektualnych oraz redukuje rozmiar aplikacji.

2.5.2.4. Bezpieczne przechowywanie danych - Secure Storage

Nowoczesne biblioteki mobile dostarczają również mechanizmy wiążące (ang. binding) do natywnych mechanizmów bezpiecznego przechowywania danych (ang. Secure Storage) – KeyStore (Android) [43], KeyChain(iOS) [44]. Klucze kryptograficzne przechowywane są w kontenerze zabezpieczonym sprzętowo, aby utrudnić wyodrębnianie z urządzenia. Gdy klucze znajdują się w magazynie kluczy, można je wykorzystać do operacji kryptograficznych, przy czym kluczowy materiał nie może zostać wyeksportowany. Ponadto oferuje możliwość ograniczenia czasu i sposobu użycia kluczy, np. Wymaganie uwierzytelnienia użytkownika w celu użycia klucza lub ograniczenie użycia kluczy tylko w niektórych trybach kryptograficznych.

2.5.2.5. Podsumowanie bezpieczeństwa

Aplikacje wieloplatformowe są równie bezpieczne co natywne, jeżeli deweloper o to zadba.

Każda metoda bezpieczeństwa jest do złamania. Zabezpieczanie kodu wykonującego operacje na wrażliwych danych po stronie serwera oraz przechowywanie możliwe jak najmniej istotnych informacji po stronie klienta – dotyczy się to każdego systemu klienckiego. Zaleca się stosowanie wytycznych polecanych przez konkretne platformy, czy ogólnie wykorzystywanie dobrych praktyk rekomendowanych przez organizacje OWASP dotyczących bezpieczeństwa aplikacji [45].

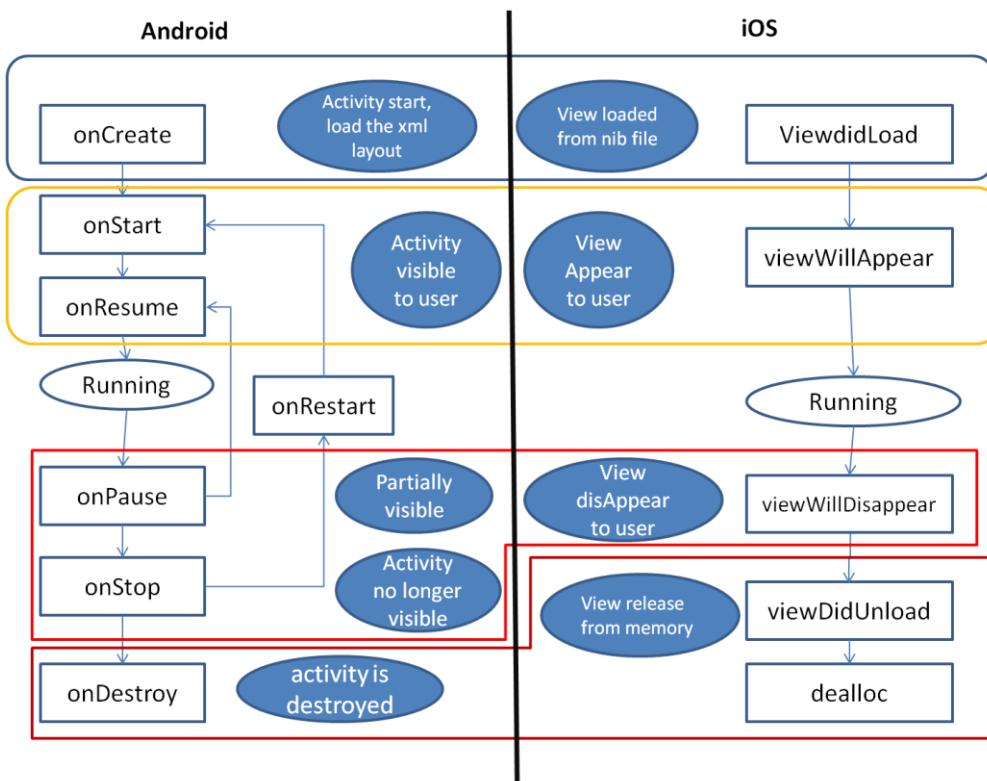
2.5.3. Zadania w tle

Zarówno cykl życia aplikacji jak i limity narzucone przez systemem mobilny ograniczają dostęp do zasobów i mają na celu zwiększenie bezpieczeństwa i wydajności urządzenia, wymuszając tym samym większy nakład pracy.

2.5.3.1. Cykl życia

Uruchomienie aplikacji w systemie mobilnym pozwala jej na pracę w różnych trybach, takich jak: aktywna, nieaktywna, wstrzymana czy działająca w tle. Aplikacja widoczna na ekranie, która działa na pierwszym planie nosi status aktywnej, natomiast pozostałe mają inne statusy, co oznacza, że tylko jedna aplikacja w danym czasie może posiadać status aktywnej.

Aplikacja uruchomiona na systemie mobilnym, przechodzi w różne stany, takie jak aktywna, nieaktywna, wstrzymana czy działająca w tle. Aplikacja widoczna na ekranie, która działa w pierwszym planie, nosi status aktywnej, natomiast pozostałe mają inne statusy, co oznacza, że tylko jedna aplikacja w danym czasie posiada status aktywnej.



Rysunek 2.13 Zestawienie cyklu życia aplikacji Android oraz iOS [46]

Prawidłowa implementacja cyklu życia może pomóc w uniknięciu aplikacji [47]:

- zawieszenie, jeśli użytkownik odbierze połączenie telefoniczne lub przełączy się na inną aplikację.
- zużywania cennych zasobów systemowych (pamięć operacyjna, bateria)
- utratę postępu użytkownika, jeśli opuści aplikację i powróci do niej w późniejszym czasie
- zawieszenie lub utratę postępu użytkownika po obróceniu ekranu między orientacją poziomą a pionową

Aplikacje wieloplatformowe znacznie gorzej wspierają procesy wykonujące się w tle. Dodatkowe biblioteki pomocnicze (ang. plugins) rozwijane przez społeczność deweloperów, zazwyczaj jako open-source, działają bezproblemowo w przypadku podstawowych funkcjonalności, na przykład pobieranie obrazu, system push, notyfikacje czy pre-caching danych.

2.5.3.2. *Procesy cykliczne*

Problem zachodzi w przypadku cyklicznie wykonywanych procesów w tle, takich jak udostępnianie lokalizacji urządzenia w czasie rzeczywistym. Ze względu na narzucone ograniczenia systemu mobilnego, deweloper zmuszony jest do osobnej implementacji mechanizmu dla poszczególnych platform (iOS/Android [48]), przy użyciu natywnych języków programowania i natywnych modułów platformy.

2.5.3.3. *GPS w tle i restrykcje platform*

Podczas budowania prototypu aplikacji FindMyTutor została wybrana biblioteka NativeScript jako technologia cross-platformowa. Projekt FindMyTutor zetknął się z problemem udostępniania cyklicznego dokładnej lokalizacji w tle.

Obydwie platformy uniemożliwiają wykonywanie kodu JavaScript (JS Runtime) i C# (Mono Runtime) gdy aplikacja nie jest w stanie aktywnym. Co za tym idzie, funkcjonalność ta musi zostać napisana w natywnym języku platformy.

2.5.3.3.1. iOS

Standard location service - konfigurowalne, uniwersalne rozwiązanie umożliwiające uzyskanie lokalizacji użytkownika w czasie rzeczywistym [49]. Ta usługa zużywa znacznie więcej energii niż inne usługi lokalizacyjne, zapewnia najdokładniejsze i natychmiastowe informacje o lokalizacji. System iOS uniemożliwia zastosowanie najdokładniejszego serwisu Standard location w tle aplikacji [50].

Zastosowanie serwisu *Significant-Change Location* - oferuje energooszczędną alternatywę dla aplikacji, które wymagają danych o lokalizacji, ale nie wymagają częstych aktualizacji lub precyzyji GPS. Usługa polega na alternatywach o niższej mocy (takich jak Wi-Fi i sieć komórkowa) w celu ustalenia lokalizacji użytkownika. Następnie dostarcza aktualizacje lokalizacji do aplikacji tylko wtedy, gdy pozycja użytkownika zmienia się o znaczącą wartość, na przykład 500 metrów lub więcej. Usługa *Significant-Change Location* dostępna jest w procesach wykonywanych w tle [51].

2.5.3.3.2. Android

Klasa *IntentService* dostarcza prostą strukturę do wykonywania operacji jednowątkowych w tle. Pozwala to na obsługę długotrwałych operacji bez wpływu na szybkość reakcji interfejsu użytkownika. Android po nadaniu odpowiednich uprawnień aplikacji, zezwala stworzenie procesu w tle z udostępnianiem lokalizacji, jednakże proces ten ma znaczny wpływ na konsumpcję baterii i nie zaleca się stosowania takiej metody.

Dodatkowym problemem są wersje Android, które wymagają modyfikacji stosowanych metod API w takim procesie, deweloper jest zmuszony do sprawdzenia różnych wersji Androida oraz poprawienie błędów związanych z tymi wersjami.

W celu zmniejszenia zużycia energii system Android 8.0 (poziom interfejsu API 26) ogranicza częstotliwość pobierania bieżącej lokalizacji urządzenia w tle. Aplikacje mogą otrzymywać aktualizacje lokalizacji tylko kilka razy na godzinę.

Standaryzacja Androida zmierza w tym samym kierunku, co standardy bezpieczeństwa iOS. Odpowiednikiem usługi iOS *Significant-Change Location* jest API *FusedLocationProviderClient* dostarczona w usłudze Google Play 11.6.0 lub wyższej. [52]

2.5.4. Kompatybilność wstecz

Rozwiązania wieloplatformowe pokrywają w znacznej ilości starsze wersje systemów mobilnych.

Aplikacja jest w stanie działać na smartfonach sprzed 7 lat - Apple iPhone 4S (iOS 8), czy Android Google Nexus 4 (Kitkat 4.4).

Biblioteka	Android		iOS	
	minimalne wersje	Udział na rynku	minimalne wersje	Udział na rynku
Xamarin	4.4 Kitkat (API 19)	94.7 %	8	98.1 %
NativeScript	5.1 Lollipop (API 22)	77.9 %	9	95.75 %
ReactNative	5.0 Lollipop (API 21)	82.9 %	9	95.75 %
Ionic	4.4 Kitkat (API 19)	94.7 %	8	98.1 %

Rysunek 2.14 Kompatybilność wsteczna bibliotek wieloplatformowych
 Źródło: Apteligent

Niemniej jednak nowe funkcje wprowadzane wraz z aktualizacją systemu operacyjnego mogą być wprowadzone ze znacznym opóźnieniem oraz z ograniczoną funkcjonalnością.

2.6. Profit z rozwiązania wieloplatformowego

2.6.1. Single code base - jeden kod kilka aplikacji

Istotną częścią rozwoju aplikacji jest zarządzanie repozytorium kodu źródłowego z pomocą systemów kontroli wersji (ang. Version Control System) takich jak git czy Subversion. Dzięki przemyślanym wyborem odpowiednich technologii wraz z rozwiązaniem wieloplatformowym, możliwe jest zredukowanie ilości repozytoriów kodu nawet do jednego [53].

Stosowanie jednego repozytorium kodu źródłowego do zarządzania aplikacjom serwerową (ang. backend) oraz aplikacjami klienckimi (ang. frontend), w które wlicza się aplikacje internetowa oraz aplikacje mobilne (iOS oraz Android), ma następujące zalety:

- umożliwia zachować spójność oraz ułatwia utrzymanie projektu
- szybsze wykrycie oraz wprowadzanie poprawek – w przypadku aplikacji wieloplatformowych naprawa usterki w jednym miejscu, skutkuje rozwiązaniem problemu na wielu platformach
- współdzielenie kodu źródłowego – dobrze przemyślana struktura projektu umożliwia wyodrębnienie komponentów, które możemy zastosować jednocześnie w aplikacji internetowej i aplikacji wieloplatformowej (mobilnej iOS i Android)
- znaczaco upraszcza wdrożenie (ang. deploy) aplikacji na innych środowiskach uruchomieniowych
- ułatwia przygotowanie oraz uruchomienie środowiska deweloperskiego na dowolnym urządzeniu spełniające wszystkie warunki wstępne (ang. prerequisite)

2.6.2. Szybkie dostarczenie MVP - niższe koszty

Przedsiębiorstwa noszące miano firm startupowych, które poprzez ograniczoną ilość zasobów ludzkich oraz środków finansowych, gdzie ryzyko inwestycyjne jest znacznie wyższe, w porównaniu z "tradycyjnymi" firmami, dążą do jak najszybszego wydania na rynek produktu, który jest najprostszą działającą wersją produktu.

Podejście MVP – ang. Minimum Viable Product – produkt o minimalnej funkcjonalności, dzięki któremu jesteśmy w stanie zaprezentować produkt potencjalnym klientom bądź użytkownikom w celu:

- sprawdzenia hipotezy czy produkt ma potencjał biznesowy
- uzyskania realnych danych od interesantów produktu
- pozyskania funduszy na dalszy rozwój produktu

Metodyka MVP umożliwia w krótki czasie z minimalnym nakładem pracy uzyskać informacje zwrotną o naszym produkcie i pozwala obrać lub zmienić ścieżkę rozwoju produktu we wczesnych stadiach projektowych. Kluczowym założeniem modelu MVP jest ciągła interakcja z klientem, iteracyjne rozwijanie produktu oraz zrozumienie potrzeb klienta [54].

Benefitem stosowania rozwiązań wieloplatformowych w kontekście MVP jest możliwość stworzenia przez mały zespół deweloperski, poruszający się w tym samym obrębie stosu technologicznego (ang. technology stack) do dotarcia do większej puli użytkowników w stosunkowo krótkim czasie oraz przy pomniejszonych kosztach dostarczenia produktu MVP.

Rozdział 3

3. Programowanie reaktywne na przykładzie platformy Android z użyciem biblioteki RxJava

3.1. Wstęp

Nie sposób zaprzeczyć, iż jednym z najszybciej rozwijających się obszarów w świecie IT jest obszar budowania interfejsów aplikacji. Podejście programistów zarówno do interfejsów webowych, jak i mobilnych cały czas ewoluje, a coraz to nowsze biblioteki i frameworki wymuszają ciągłą potrzebę poznawania nowych rozwiązań. Jednym z kluczowych aspektów budowania interfejsu użytkownika jest możliwość zapanowania nad zdarzeniami, które wywołuje użytkownik aplikacji. Z pomocą przychodzi podejście reaktywne, które łączy w sobie paradygmat programowania funkcyjnego i takie wzorce projektowe jak obserwator i iterator. Następujący rozdział opisuje elementy, z których składa się programowanie reaktywne jak i jego zastosowanie. W dalszej części przedstawia także proces budowania interfejsu użytkownika w platformie Android z użyciem podejścia reaktywnego.

3.2. Programowanie reaktywne

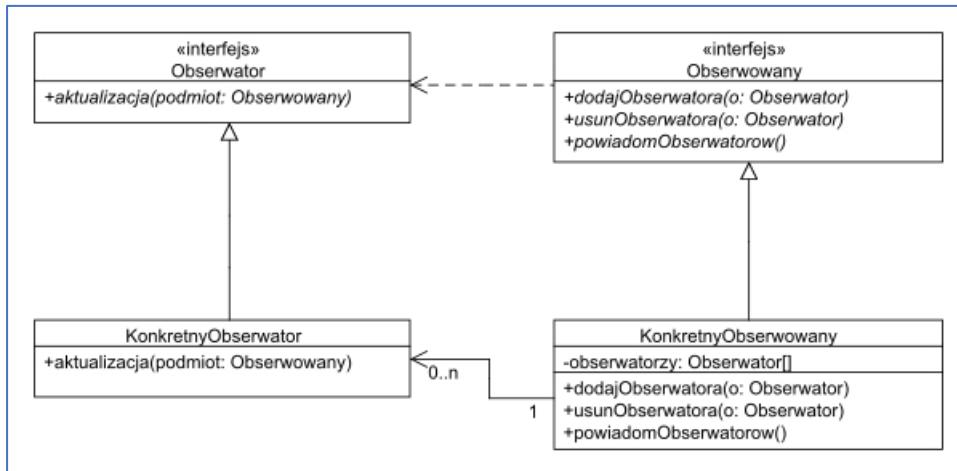
W tej części rozdziału przedstawione zostaną teoretyczne pojęcia, które kryją się pod pojęciem programowania reaktywnego. Raczej każdy programista rozpoczyna swoją przygodę z programowaniem od paradygmatu imperatywnego, w którym kroki w programie są realizowane w sposób sekwencyjny. Taki sposób programowania mocno odbiega od podejścia reaktywnego, które na pierwszy rzut oka może wydawać się złożone. Samo pojęcie nie jest jednak niczym nowym i istnieje już od pewnego czasu w świecie IT. Istota programowania reaktywnego to próba odzwierciedlenia otaczającego nas świata, który zbudowany jest z asynchronicznych zdarzeń. Takie same asynchroniczne zdarzenia zachodzą także w interfejsie użytkownika jak i w całej aplikacji. Takimi zdarzeniami w aplikacji mogą być: pobranie danych z pliku/serwera, wypełnienie pola tekstowego, naciśnięciu przycisku. Podejście reaktywne pozwala nam zapanować nad asynchronicznymi zdarzeniami za pomocą paradygmatu funkcyjnego i kilku wzorców projektowych. Funkcyjne operacje na danych zapewniają nam zwięłość w sposobie manipulacji nimi a wzorzec obserwator zapewnia wygodny sposób reagowania na zdarzenia występujące w aplikacji.

3.2.1. Wzorzec projektowy obserwator

Jednym z najważniejszych elementów programowania reaktywnego jest wzorzec obserwator (ang. Observer Pattern). Jest to wzorzec typu behavioralnego, który odpowiada za nasłuchiwanie czy też obserwację jakiegoś zdarzenia. Głównymi podmiotami tego wzorca jest obiekt obserwowany oraz obiekty go obserwujące. Pomiędzy podmiotami zachodzi, więc relacja jeden-do-wielu. Jeżeli obiekt, który jest obserwowany zmieni swój stan to wszystkie obiekty, który zdecydowały się na obserwowanie go zostaną powiadomione o tym fakcie.

Jednym najprostszych przykładów może być stacja pogodowa, która po otrzymaniu nowych danych, może automatycznie powiadomić o aktualizacji swojego stanu takich obserwatorów jak strona internetowa czy wyświetlacz. Podstawowym sposobem implementacji tego wzorca jest zaprojektowanie dwóch interfejsów, które odpowiedzialne są za deklaracje odpowiednich metod [55]. Aby jeszcze lepiej zrozumieć budowę tego wzorca poniżej zaprezentowany został diagram klas wzorca obserwator.

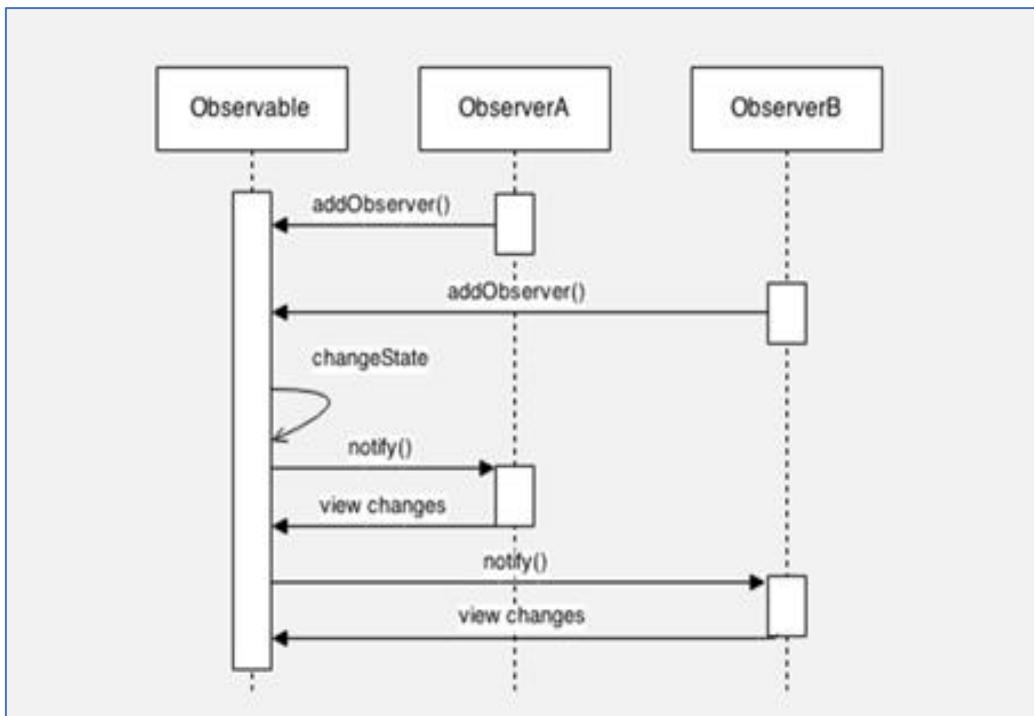
3.2.1.1. Schemat wzorca projektowego obserwator



Rysunek 3.1 Diagram klas wzorca obserwator [56]

Wyróżnić należy dwa podstawowe typy obiektów, które składają się na strukturę tego wzorca:

- Obserwowany (ang. Observable, Subject) – obiekt, który odpowiedzialny jest za publikowanie zmian do wszystkich swoich subskrybentów.
- Obserwator (ang. Observer, Listener) – obiekt, który oczekuje na powiadomienie opublikowane przez obiekt obserwowany.



Rysunek 3.2 Diagram sekwencji wzorca obserwator [57]

Nowe obiekty, które zdecydują się na obserwacje mogą zostać dodane w dowolnym momencie. Obserwujący obiekt może zaprzestać obserwowania, lecz może zostać także usunięty listy obserwatorów w obiekcie obserwowanym.

Język Java posiada własną implementację wzorca projektowego obserwator (`java.util.Observable`). Niestety w tym przypadku jest to klasa a nie interfejs, co znaczco ogranicza jego zastosowanie. Aby skorzystać z gotowej implementacji należy utworzyć klasy podrzędne, co skutkuje tym, że nie można dodać zachowania klasy `Observable` do innej już istniejącej [55].

3.2.1.2. *Wady i zalety*

Jedną z największych zalet tego wzorca projektowego jest osłabienie zależności pomiędzy obiektami, które obserwują obiekt obserwowany. Skutkuje to odpowiednią izolacją, ponieważ obserwowany i obserwator nie muszą mieć dużej wiedzy o sobie. Jedyną informacją, jaką posiada obserwowany podmiot o obiektach go obserwujących jest fakt, że implementują one odpowiedni interfejs [55]. Niestety nie można uzależnić aplikacji od określonej kolejności nadchodzących powiadomień. Powiadomienie może także dotrzeć do obserwatora zbyt późno.

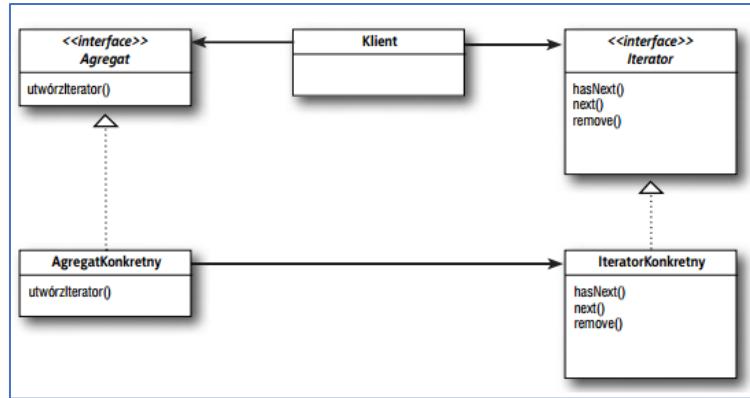
3.2.2. *Wzorzec projektowy iterator*

Jego celem stworzenie jest jednolitego interfejsu, który zapewnia sekwencyjny dostęp do obiektów znajdujących się w danej kolekcji bez ujawniania ich wewnętrznej reprezentacji [55].

3.2.2.1. *Schemat wzorca projektowego iterator*

Wzorzec ten składa się z:

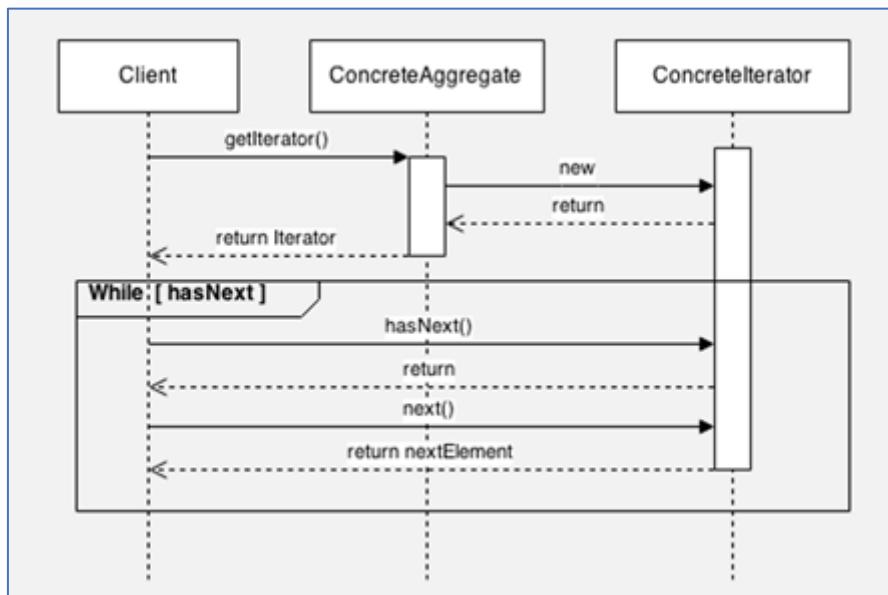
- Dwóch klas abstrakcyjnych: Agregat i Iterator
- Dwóch klas konkretnych: AgregatKonkretny i IteratorKonkretny



Rysunek 3.3 Diagram klas wzorca iterator

Źródło: Wzorce projektowe rusz głową, Freeman, E. F. K. S. B. B. Elisabeth 2017

Wszystkie kolekcje stworzone za pomocą tego wzorca implementują interfejs Aggregate. Klient, który odwoła się do metody CreateIterator(), otrzymuje klasę, która implementuje interfejs Iterator. Skutkuje to tym, że klient zna tylko interfejs, do, którego musi się odwołać. Dokładnie taki scenariusz działania zachodzi w obrębie biblioteki Collections w języku Java. Każda kolekcja posiada swój własny iterator, lecz dostęp do niego realizowany jest za pomocą wspólnego interfejsu Iterator [58]. Iteratory posiadają metody, które pozwalają na sekwencyjny dostęp do wszystkich elementów kolekcji. W niektórych implementacjach pozwala on także na modyfikację kolekcji.



Rysunek 3.4 Diagram sekwencji wzorca iterator [59]

Język Java posiada domyślną implementację wzorca projektowego Iterator z metodami: hasNext, next, remove i forEachRemaining. Poniższy przykład ilustruje użycie wbudowanego iteratora celu wyświetlenia elementów przechodzonej kolekcji na standardowe wyjście programu [60].

```
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;

public class ExternalIteratorDemo
{
    public static void main(String[] args)
    {
        List<String> names = new LinkedList<>();
        names.add("Rams");
        names.add("Posa");
        names.add("Chinni");

        // Getting Iterator
        Iterator<String> namesIterator = names.iterator();

        // Traversing elements
        while(namesIterator.hasNext()){
            System.out.println(namesIterator.next());
        }
    }
}
```

Rysunek 3.5 Przykład użycia wbudowanego iteratora w języku Java [61]

3.2.2.2. *Po słowie*

W wielu językach programowania powszechna stała się konstrukcja for-each, która tak naprawdę w niejawny sposób wywołuje Iterator. Dowodzi to, że wzorzec Iterator jest już niejako standardem zaimplementowanym praktycznie w każdym języku programowania [62].

3.2.3. Biblioteka ReactiveX

Jest to biblioteka, która łączy w sobie zastosowanie zarówno wzorców obserwator i iterator, a ponadto zawiera też implementacje elementów paradymatu funkcyjnego. Połącznie tych trzech głównych składowych pozwala na manipulacje danymi w bardziej deklaratywny sposób, co skutkuje wysokim poziomem abstrakcji w zakresie wielowątkowości, synchronizacji danych jak i operacjami, które mogą blokować działania aplikacji. Biblioteka ta została zaimplementowana w wielu językach programowania takich jak: .Net, Java, JavaScript, Scala czy Swift [63]. W przypadku projektu FindMyTutor została wykorzystana implementacja w języku Java nosząca nazwę RxJava.

3.2.3.1. Zastosowania wzorca obserwator w bibliotece RxJava

W bibliotece RxJava wyróżnić możemy następujące typy obiektów obserwowanych:

Typ obserwowanego obiektu (ang. Observable)	Opis
Flowable<T>	Emituje on od 0 do N elementów. Może zakończyć swoje działanie w dwojakim sposobie, zwracając sukces lub błąd. Wspiera sprzężenie danych (ang. Backpressure), co skutkuje możliwością kontrolowania jak szybko dane zostaną opublikowane.
Observable<T>	Bardzo podobny do Flowable, lecz nie wspiera sprzężenia danych.
Single<T>	Emituje pojedynczą wartość lub błąd. Przydatny podczas tworzenia serwisów HTTP. Przykładowo zapytanie GET może zwrócić wynik zapytania lub błąd.
Maybe<T>	Kończy swoje działanie emitując jeden ze stanów: pojedynczy element, brak elementu lub błąd.
Completable<T>	Kończy swoje działanie zwracając sukces lub błąd. Nigdy nie emittuje elementów. Przykładem zastosowania może być obsługa zapytania POST w serwisie HTTP.

Rysunek 3.6 Opis poszczególnych typów obserwatorów w bibliotece RxJava [64]

Sama biblioteka dostarcza też wiele metod za pomocą, których można utworzyć różne rodzaje obserwatorów. Warto zaznaczyć jednak, że nie wszystkie podane niżej metody są dostępne dla typów wyszczególnionych w powyższej tabeli. Przykładowo metoda fromArray dostępna jest jedynie dla typów: Flowable i Observable [65].

Metoda	Opis
Observable.just()	Pozwala na utworzenie obiektu obserwowanego po przez opakowanie innego typu danych.
Observable.fromIterable()	Jako argument oczekuje typu java.lang.Iterable<T> i emituje elementy tego typu z zachowaniem kolejności elementów.
Observable.fromArray()	Jako argument przyjmuje tablicę i emituje jej zawartość z zachowaniem kolejności elementów.
Observable.fromCallable()	Pozwala na utworzenie obiektu obserwowanego z typu java.util.concurrent.Callable<V>.
Observable.fromFuture()	Pozwala na utworzenie obiektu obserwowanego z typu java.util.concurrent.Future.
Observable.interval()	Tworzy obiekt obserwowany, który emituje nieskończoną sekwencję obiektów typu Long w podanym odstępie czasowym.

Rysunek 3.7 Opis poszczególnych metod umożliwiających tworzenie obserwatorów w bibliotece RxJava [64]

W celu utworzenia obserwatora należy posłużyć się operatorem Subscribe, który odpowiedzialny jest za połączenie ze sobą obserwatora i obiektu obserwowanego. Wyróżnia się cztery podstawowe typy obserwatorów: Observer, SingleObservable, MaybeObservable i CompletableObserver [66]. Ich powiązanie z obiektem obserwowanym zostało zaprezentowane w poniższej tabeli.

Typ obiektu obserwowanego	Typ obserwatora
Observable	Observer
Single	SingleObserver
Maybe	MaybeObserver
Flowable	Observer
Completable	CompletableObserver

Rysunek 3.8 Typy obiektów obserwowanych wraz z typami obserwatorów [67]

Prawie każdy typ obiektu obserwującego musie zawierać implementacje metod wyszczególnionych w poniższej tabeli.

Metoda obserwatora (ang. Subscriber)	Opis
onNext()	Metoda ta zostaje wywołana, gdy obserwator wyemituje element.
onError()	Obserwator wywoła tę metodę, gdy nastąpi błąd podczas emitowania danych. Wywołanie tej metody zatrzymuje dalsze przetwarzanie kolejnych elementów i skutkuje brakiem wywołań pozostałych metod: onNext i onComplete.
onComplete()	Metoda ta zostanie wywołana, gdy po raz ostatni zostanie wywołania metoda onNext i niewywołana została metoda onError w trakcie emitowania danych.

Rysunek 3.9 Podstawowe metody obserwatorów dostępne w bibliotece RxJava [67]

3.2.4. Elementy paradygmatu funkcyjnego w programowaniu reaktywnym

Najważniejszym komponentem, z którego zbudowany jest paradygmat funkcyjny są funkcje. W podejściu tym nie występują pętle ani zmienne. Zamiast tego posługiwać musimy się zdefiniowanymi stałymi i rekurencją. Jednymi z największych zalet tego paradygmatu są funkcje wyższego rzędu, strumienie oraz funktry. W rezultacie otrzymany kod zawiera mniej efektów ubocznych i jest bardziej przewidywalny w działaniu. To z tej dziedziny pochodzą takie pojęcia jak czysta funkcja (ang. Pure Function) lub niezmienne dane (ang. Immutable Data). Programując funkcyjnie odpowiadamy raczej na pytanie: „Co chcemy osiągnąć?” (Jest to paradygmat deklaratywny), a nie „W jaki sposób chcemy to osiągnąć?” (To pytanie zadajemy sobie programując w paradygmacie imperatywnym). Poznając programowanie reaktywne bardzo często spotykamy się z takimi funkcjami wyższego rzędu jak: map, filter czy fold, które znajdują swoje odzwierciedlenie w programowaniu reaktywnym [68].

3.2.4.1. Czyste funkcje

Jako czystą funkcję należy rozumieć funkcję, która nie wywołuje efektów ubocznych tj. może wpływać na otoczenie tylko poprzez swój wynik a wynik ten jest tylko i wyłącznie zależny od danych wejściowych (ang. Referential Transparency). Jako efekty uboczne można rozumieć m.in. wyświetlenie tekstu na ekran, wysłanie danych do serwera etc. Efekty uboczne, które mogą wystąpić podczas użycia funkcji, która nie jest czysta mogą powodować sprzężenia i zależności czasowe, co skutkuje niepoprawnym działaniem aplikacji.

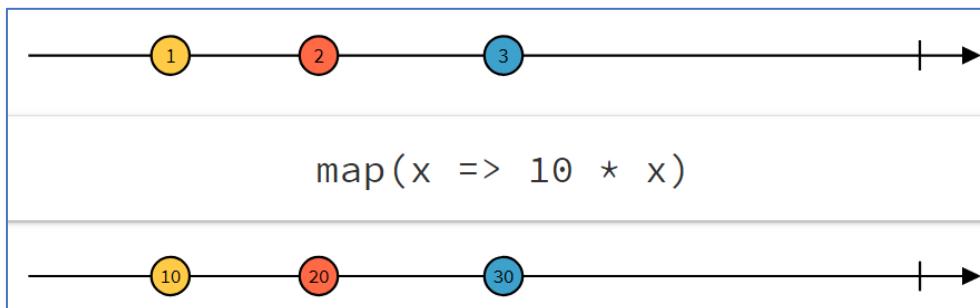
Czyste funkcje są łatwiejsze w testowaniu, ponieważ ich wyjście zależy tylko od danych wejściowych. Ponadto może je składać i są łatwiejsze do zrównoleglania [69]. Oczywiście niemożliwym jest operowanie tylko czystymi funkcjami, ponieważ prawie każdy program wypisuje dane na ekran, co samym w sobie jest efektem ubocznym. Należy jednak dążyć do jak największej liczby takich funkcji w programie. Efekty uboczne mogą zostać zniwelowane za pomocą innych narzędzi i definicji matematycznych takich jak na przykład Monady [70].

3.2.4.2. Funkcje wyższego rzędu

Innym podstawowym mechanizmem używanym w programowaniu funkcyjnym jest mechanizm funkcji wyższego rzędu (ang. Higher order function). Funkcja wyższego rzędu to funkcja, która przyjmuje jako argument inną funkcję lub zwraca funkcję. Można wyróżnić kilka rodzajów takich funkcji:

- Map

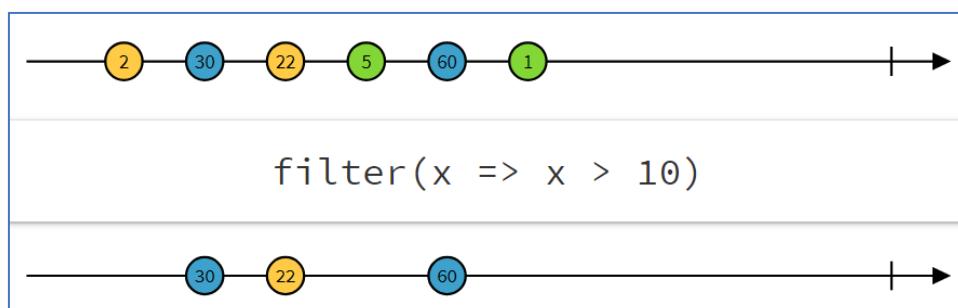
Ten typ funkcji wyższego rzędu przyjmuje jako argument funkcję i listę. Podczas przechodzenia po liście na każdym jej elemencie aplikowana jest funkcja podana w argumencie. W rezultacie wynikiem funkcji map jest nowa lista, która składa się z elementów zwróconych przez funkcje, która została zaaplikowana [68].



Rysunek 3.10 Obrazowe przedstawienie działania funkcji map [71]

- Filter

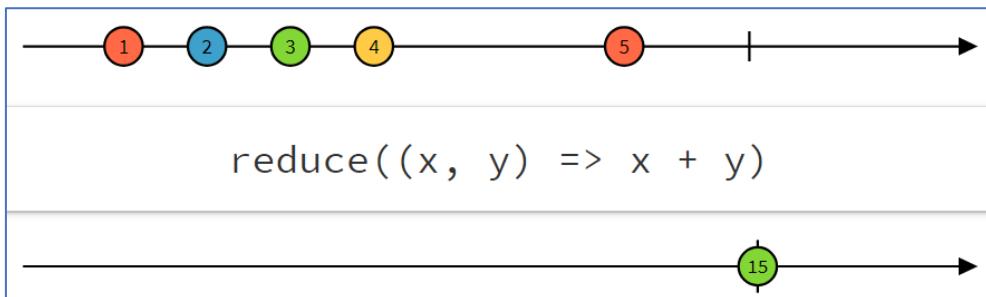
Podobnie jak w funkcji map argumentami są funkcja i lista. Jednak w tym przypadku jako argument musi zostać podana funkcja boolowska (funkcja, która zwraca tylko wartość true lub false). Funkcja zostaje zaaplikowana do każdego elementu z listy, aby następnie zwrócić elementy, dla których została zwrócona wartość true [68].



Rysunek 3.11 Obrazowe przedstawienie działania funkcji filter [71]

- Reduce

Funkcja reduce przyjmuje jako argumenty: funkcję, wartość początkową i listę. Jej zadaniem jest obliczenie pewnej wartości na podstawie wszystkich elementów listy. Funkcja podana jako argument operuje na wartości obliczonej do n-tego elementu listy i jej następnym elemencie. Przykładowym zastosowaniem funkcji reduce może być obliczenie sumy wszystkich elementów listy [68].



Rysunek 3.12 Obrazowe przedstawienie działania funkcji reduce [71]

3.2.4.3. Po słowie

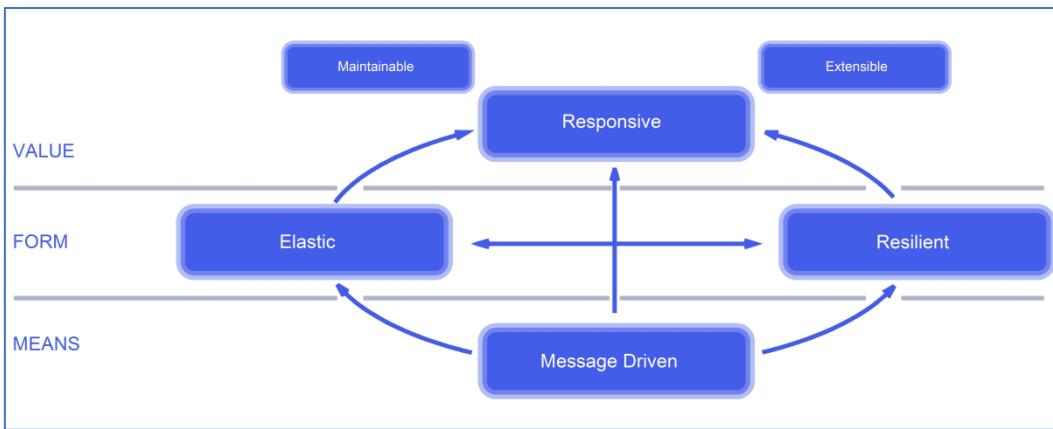
Implementacja funkcji wyższego rzędu może zależeć od danego języka programowania. Przykładowo w języku JavaScript funkcje te są zdefiniowane w prototypie danego obiektu, co skutkuje tym, że zbędny jest argument z listą, po której operujemy. Z racji tego, że funkcje wyższego rzędu zwracają nową listę możliwe jest ich składania, przez co otrzymany kod jest jeszcze bardziej przejrzysty. Kod napisany z użyciem paradygmatu funkcyjnego w większości przypadków jest bardziej zwięzły od kodu napisanego w paradygmacie imperatywnym, co skutkuje mniejszą złożonością i ułatwia jego zrozumienie przez programistę [72].

3.2.5. The Reactive manifesto

Nie sposób zaprzeczyć, iż wymagania, które twórcy oprogramowania stawali przed sobą jeszcze kilka lat temu uległy drastycznej zmianie. Nie tak dawno temu istniały aplikacje, w których czas odpowiedzi serwera był liczony w sekundach a liczba godzin niedostępności aplikacji była wyrażana w godzinach. W dzisiejszym świecie, w którym aplikację uruchamiane są na niezliczonej ilości różnych urządzeń a użytkownik oczekuje czasu odpowiedzi serwera w milisekundach i dostępności serwisu 24 godziny na dobę, wymagania stawiane przed twórcami oprogramowania uległy znaczącej zmianie. The reactive manifesto to próba odpowiedzenia na problemy, z którymi musi zmierzyć się system informatyczny w dzisiejszych czasach.

System reaktywny to system, który jest:

- Responsywny – system reaguje tak szybko jak to tylko możliwe. Responsywność jest podstawą użyteczności. Ponadto pozwala ona na szybkie wykrycie potencjalnych błędów. Systemy reaktywne zapewniają szybki i niezmienny czas reakcji, który buduje zaufanie użytkownika systemu [73].
- Odporny – system pozostaje responsywny nawet w przypadku wystąpienia błędów. Odporność uzyskiwana jest za pomocą takich pojęć jak: replikacja, izolowanie czy delegowanie. Wystąpienie błędu w jednym z komponentów systemu nie może skutkować pojawiением się błędu w innych obszarach [63].
- Elastyczny – system jest w stanie dostosować swoje działanie do obciążenia. Jest on w stanie zarządzać swoimi zasobami w celu odpowiedzenia na napotkane zapotrzebowanie [63].
- Sterowany wiadomościami (ang. Message driven) – systemy reaktywne opierają się na asynchronicznym przekazywaniu danych w celu ustalenia granic pomiędzy danymi komponentami. Skutkuje to izolacją i przejrzystością ich lokalizacji. Ustalenie granic pomiędzy komponentami systemu umożliwia przekazywanie błędów jako wiadomości. Skutkuje to możliwością zarządzania obciążeniem systemu i lepszą obsługę awarii poszczególnych komponentów [63].



Rysunek 3.13 Zależności pomiędzy komponentami w systemie reaktywnym [74]

3.3. Interfejs użytkownika w platformie Android

Jednym z podstawowych komponentów, który używany jest w procesie budowania aplikacji jest aktywność (ang. Activity). Jest to klasa, której zadaniem jest umożliwienie interakcji użytkownika z aplikacją. Za jej pomocą można utworzyć okno, w którym umieszony zostanie widok interfejsu użytkownika. Posługując się przykładem aplikacji, która pełni rolę klienta poczty elektronicznej, zakłada się, że lista z wiadomościami będzie prezentowana za pomocą jednej aktywności, a inna aktywność posłuży do utworzenia wiadomości. Istnieje możliwość grupowania kilku aktywności w grupy, ale zazwyczaj należy utożsamiać widoczne okno aplikacji z jedną instancją pod klasy Activity.

Definiując własną aktywność należy stworzyć podklasę klasy Activity dziedzicząc w ten sposób niezbędne metody, za pomocą których programista może wpływać na interfejs użytkownika [75]. Jednym z kolejnych podstawowych elementów UI w systemie Android jest klasa Fragment. Można używać jej w wieloraki sposób, lecz najczęściej jest ona ściśle połączona z Activity. W odróżnieniu od klasy Activity klasa Fragment ma raczej za zadanie odpowiadać za pewną mniejszą część interfejsu użytkownika. Warto zaznaczyć, iż zachowanie instancji Fragmentu jest ściśle powiązane z zachowaniem jej rodzica, czyli Activity. W przypadku zakończenia działania danej aktywności, wszystkie instancje klasy Fragment, które utworzone zostały w danym Activity również zakończą swoje działanie [76]. Aby lepiej zrozumieć zachowanie się interfejsu użytkownika w systemie Android należy zapoznać się z cyklem życia aktywności.

3.3.1. Cykl życia aktywności

Podczas używania aplikacji użytkownik może wprowadzać aktywność w różnego rodzaju stany. Sam system operacyjny także może oddziaływać na stan, w jakim znajduje się aplikacja. Taką sytuację może być przykładowo wyświetlenie natywnego ekranu z przychodzącem połączeniem telefonicznym. W takiej sytuacji programista musi posiadać narzędzia, które umożliwią mu reagowanie na zmianę stanu interfejsu użytkownika. Za pomocą odpowiednich metod zdefiniowanych w klasie Activity (Często nazywanych również z ang. callbackami) programista jest w stanie określić, jak zachowa się aplikacja podczas przechodzenia w nowy stan [77].

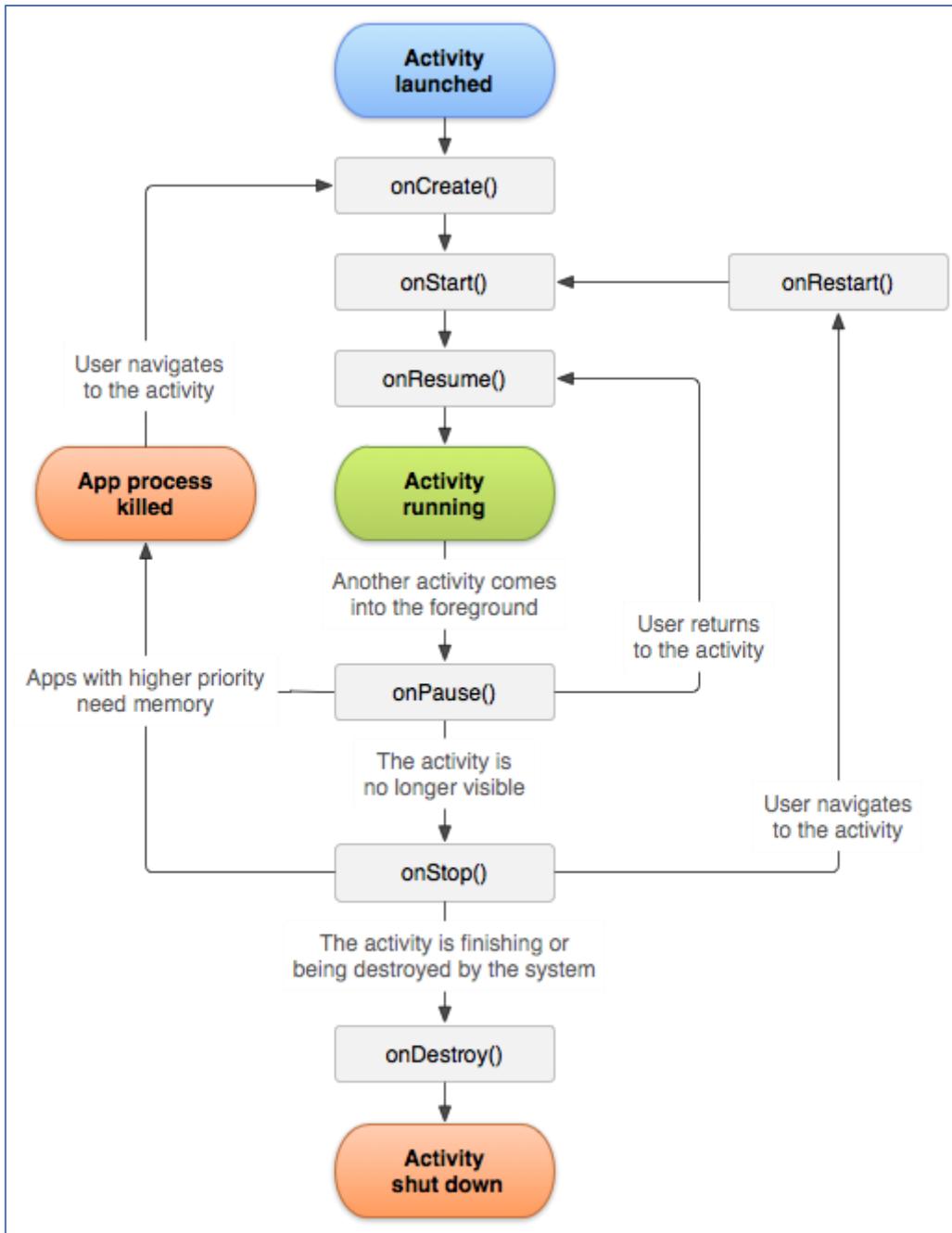
Aby zapanować nad zmianami, które zachodzą w obrębie aplikacji, zdefiniowanych zostało siedem zapytań zwrotnych (ang. Callbacks) za pomocą, których możemy reagować na zachodzące zmiany. Cykl życia instancji klasy Fragment nie różni się znacząco od cyklu życia aktywności, dlatego omówiony zostanie tylko cykl życia dotyczący aktywności. Poszczególne metody zostały wyszczególnione i opisane szerzej poniżej.

- *OnCreate*

Jest to obligatoryjne wywołanie zwrotne, który zostaje wywołane podczas utworzenia danego Activity. W tej metodzie powinien znajdować się taki kod programu, który zostanie wywołany tylko raz w ramach danego cyklu życia Activity. To w obrębie tej metody należy zdefiniować referencję do wszelkiego rodzaju przycisków, pól tekstowych i innych elementów interfejsu użytkownika. Po zakończeniu wykonywania tej metody system wykonuje następny callback *OnStart* [78].

- *OnRestart*

Metoda ta wywoływana jest po zatrzymaniu aktywności w celu przygotowania jej do restartu. Zawsze następuje przed metodą *onStart* [78].



Rysunek 3.14 Cykl życia aktywności [79]

- *OnStart*

Ta metoda skutkuje pojawiением się interfejsu aplikacji na ekranie urządzenia. Jest ona zawsze wywoływana po metodach *OnCreate* lub *OnRestart*. W przeciwieństwie do metody *OnCreate* może ona zostać wywołana wiele razy w trakcie życia aplikacji. W tym miejscu należy zainicjalizować takie byty jak *BrodcastReceiver*, który pozwala na odbieranie powiadomień z całego systemu, innej aplikacji czy też procesu [78].

- *OnResume*

Metoda ta zostaje wywołana, gdy użytkownik powraca do aplikacji na przykład po odebraniu połączenia lub po ponownym włączeniu ekranu telefonu. W tym miejscu należy umieść kod odpowiedzialny za odświeżenie widoku lub pobranie nowych danych. Po zakończeniu działania tej metody następuje aktywny czas życia aplikacji (ang. The Active Lifetime) [80].

- *OnStop*

Metoda ta zostaje wywołana, kiedy aplikacja nie jest już widoczna dla użytkownika. W tym miejscu umieszczony może zostać kod programu odpowiedzialny za zatrzymanie wszelkiego rodzaju animacji, serwisów i innych bytów, które związane są z interfejsem użytkownika [80].

- *OnPause*

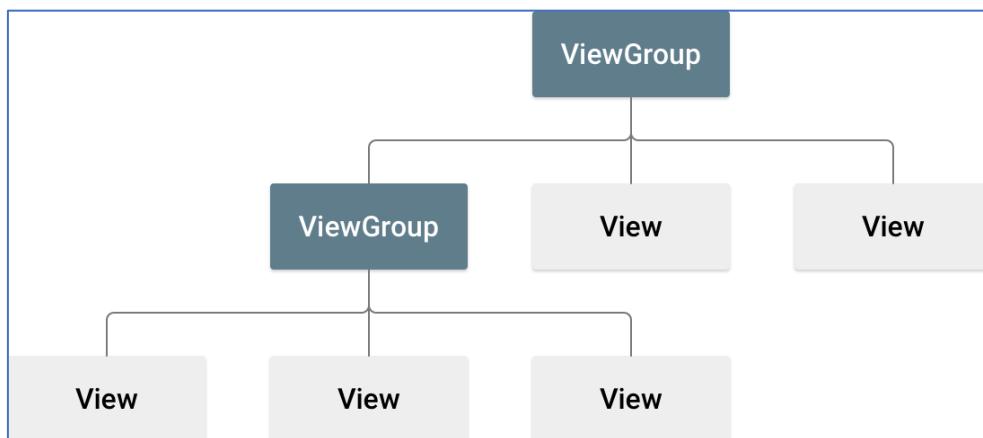
Wywołanie tej metody może nastąpić, kiedy użytkownik zacznie opuszczać aktywność lub nastąpi przysłonięcie jej przez inną aktywność. Aplikacja nadal może być widoczna na ekranie, dlatego nie zaleca się spowalnia wywoływanego OnPause różnego rodzaju operacjami takimi jak zapis danych. Po tej metodzie mogą zostać wywołane metody: *OnResume* lub *OnStop* [80].

- *OnDestroy*

Metoda ta zostaje wywołana przed zakończeniem działania aktywności. W tym miejscu aplikacji powinna zadbać o zwolnienie zasobów, z których korzysta. Może to być na przykład zamknięcie połączenia z bazą danych.

3.3.2. Layout jak podstawowa jednostka widoku

Podstawowym narzędziem, które umożliwia zdefiniowane interfejsu użytkownika jest szablon (ang. Layout). Wszystkie elementy używane podczas tworzenia UI budowane są za pomocą hierarchii klas View i ViewGroup. Klasa View odpowiedzialna jest za renderowanie bytów, z którymi użytkownik może wchodzić w interakcję. Natomiast klasa ViewGroup to rodzaj niewidzialnego kontenera, który odpowiedzialny jest za definicje struktury interfejsu użytkownika. Poniżej przedstawiono prosty schemat budowy szablonu w systemie Android.



Rysunek 3.15 Przykładowy schemat layoutu [79]

Deklarowanie UI może zostać zrealizowane na dwa sposoby:

- Poprzez zdefiniowanie elementów UI bezpośrednio w pliku XML.
- Poprzez stworzenie poszczególnych elementów UI podczas działania programu.

Zazwyczaj stosuje się pierwsze podejście w celu odseparowania widoku od logiki aplikacji. W ten sposób łatwiejsze staje się też, zdefiniowanie różnego rodzaju szablonów w zależności od szerokości ekranu lub jego orientacji.

3.3.3. Format pliku szablonu

Każdy plik zawierający definicje szablonu UI musi być zdefiniowany w formacie XML. Ponadto musi zawierać dokładnie jeden element nadzędny (rodzica), który jest typu View lub ViewGroup. Poniżej zaprezentowany został szablon odpowiedzialny za wygląd ekranu logowania w projekcie FindMyTutor. Szablon składa się z rodzica, którym jest LinearLayout i takich elementów potomnych jak: ImageView, ProgressBar czy Button.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:fontFamily="@font/lato_regular"
    android:gravity="center_horizontal"
    android:orientation="vertical"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    tools:context=".activity.LoginActivity">
    <ImageView
        android:id="@+id/imageView"
        android:layout_width="match_parent"
        android:layout_height="120dp"
        android:contentDescription="@string/logo_find_my_tutor"
        app:srcCompat="@drawable/logo_design_black2" />
    <ProgressBar
        android:id="@+id/login_progress"
        style="?android:attr/progressBarStyleLarge"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginBottom="8dp"
```

```
    android:visibility="gone" />

<ScrollView
    android:id="@+id/login_form"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <LinearLayout
        android:id="@+id/email_login_form"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical">
        <android.support.design.widget.TextInputLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content">
            <android.support.design.widget.TextInputEditText
                android:id="@+id/email"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:hint="@string/prompt_login"
                android:inputType="textEmailAddress"
                android:maxLines="1"
                android:singleLine="true" />
        </android.support.design.widget.TextInputLayout>
        <android.support.design.widget.TextInputLayout
            android:layout_width="match_parent"
            app:passwordToggleEnabled="true"
            android:layout_height="wrap_content">
            <android.support.design.widget.TextInputEditText
                android:id="@+id/password"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:hint="@string/prompt_password"
                android:imeActionId="6"
                android:imeActionLabel="@string/action_sign_in_short"
                android:imeOptions="actionUnspecified"
```

```
        android:inputType="textPassword"
        android:maxLines="1"
        android:singleLine="true" />
    </android.support.design.widget.TextInputLayout>
    <Button
        android:id="@+id/email_sign_in_button"
        style="?android:textAppearanceSmall"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="16dp"
        android:text="@string/action_log_in"
        android:textStyle="bold" />
</LinearLayout>
</ScrollView>
</LinearLayout>
```

Rysunek 3.16 Schemat widoku ekranu logowania w aplikacji FindMyTutor

Podczas komilacji programu plik z szablonem jest komplikowany do zasób typu View. Następnie w metodzie *OnCreate* możemy załadować dany szablon za pomocą metody *setContentView()*, która przyjmuje nazwę pliku zawierającego szablon interfejsu użytkownika. W celu umożliwienia stworzenia referencji do danego elementu zdefiniowanego w szablonie należy w kodzie programu przypisać do każdego elementu unikalny identyfikator. Następnie, aby stworzyć instancję danego elementu należy wyszukać go za pomocą zdefiniowanego wcześniej identyfikatora (Powinno odbywać się to w metodzie *OnCreate()*). Poniżej zaprezentowany został przykładowy kod w języku Java, który odpowiada za utworzenie instancji przycisku zdefiniowanego w szablonie.

```
Button mEmailSignInButton = findViewById(R.id.email_sign_in_button);
```

Rysunek 3.17 Utworzenie referencji do przycisku logowania w aplikacji FindMyTutor

Powyższy przykład tworzenia referencji do obiektów zdefiniowanych plikach XML wydaje się proste jednak posiada jedną zasadniczą wadę, utworzenie referencji musi odbywać się w metodzie *OnCreate*. Pozornie nie wydaje się to problemem jednak warto pamiętać, że w takim przypadku wszelkiego rodzaju obsługa zdarzeń takich jak naciśnięcie przycisku lub wpisanie wartości do pola tekstowego także, musi zostać zdefiniowana w tej metodzie. Aby uniezależnić tworzenie referencji do obiektów wyszczególnionych w szablonie XML w projekcie FindMyTutor została użyta biblioteka *ButterKnife*.

3.3.4. Biblioteka ButterKnife

Celem zastosowania tej biblioteki jest zmniejszenie ilości kodu, który musi zostać wygenerowany podczas tworzenia referencji i obsługi zdarzeń elementów budujących widok aplikacji. Za jej pomocą możliwe staje się nie tylko wyekstrahowanie kodu odpowiedzialnego za obsługę interfejsu użytkownika z metody OnCreate, ale także grupowanie poszczególnych obiektów w listy, co skutkuje większą zwięzłością.

```
public class WhiteList extends AppCompatActivity {

    @BindView(R.id.recycler_view_whitelist)
    RecyclerView recyclerView;

    @BindView(R.id.white_list_empty_text_view)
    TextView noNotesView;

    @BindView(R.id.switch_whitelist_toggle)
    Switch aSwitch;

    @BindView(R.id.add_to_white_list_fab)
    FloatingActionButton addToWhiteListFab;

    @BindView(R.id.loader)
    SpinKitView loader;

    ...
}
```

Rysunek 3.18 Przykład użycia biblioteki ButterKnife w projekcie FindMyTutor

Jak widać w zaprezentowanym powyżej kawałku kodu, logika odpowiedzialna za zdefiniowanie odwołań do elementów interfejsu użytkownika nie została zdefiniowana w metodzie OnCreate. Ponadto możliwe stało się także, zadeklarowanie w tym samym miejscu logiki odpowiedzialnej za obsługę zdarzeń.

3.3.5. Cykl życia widoku a biblioteka RxJava

Decydując się na utworzenie obiektu obserwowanego i jego obserwatorów w aktywności lub innym komponencie odpowiedzialnym za interfejs użytkownika należy zapewnić odpowiednie powiązanie pomiędzy nimi a cyklem życia widoku aplikacji. Obserwatorzy nie powinni emitować żadnych danych, gdy widok przestaje być już dostępny dla użytkownika. Aby uzależnić stan obserwatorów od stanu, w którym znajduje się aktualnie widok należy posłużyć się obiektem Disposable z biblioteki RxJava.

```
private void loginProcess(String email, String password) {  
    ValidateUser user = new ValidateUser(email, password);  
  
    disposable.add(ldapService.validate(user)  
        .subscribeOn(Schedulers.io())  
        .observeOn(AndroidSchedulers.mainThread())  
        .subscribe(this::handleResponse, this::handleError));  
}
```

Rysunek 3.19 Przykład dodania obserwatora do obiektu typu Disposable w aplikacji FindMyTutor

Następnie, aby obserwatorzy zaprzestali obserwowania należy w odpowiedniej metodzie cyklu życia aplikacji anulować wszystkie aktualne subskrypcje. Zabezpiecza to przed potencjalnymi problemami na przykład z wyciekami pamięci.

```
@Override  
public void onDestroy() {  
    super.onDestroy();  
    disposable.dispose();  
}
```

Rysunek 3.20 Przykład anulowania wszystkich subskrypcji danego obserwatora w metodzie onDestroy

3.4. Przykłady zastosowanie biblioteki RxJava w projekcie FindMyTutor

Jednym z głównych zastosowań biblioteki RxJava było utworzenie serwisów HTTP komunikujących się z warstwą backendową aplikacji. Podejście reaktywne zostało także zastosowane w niektórych elementach aplikacji odpowiedzialnych za interfejs użytkownika takich jak: wyszukiwarka użytkowników obsługa przycisków i pól tekstowych. Aby zamieszczone dalej przykłady zostały w pełni zrozumiałe należy omówić kilka dodatkowych operatów, które są niezbędne podczas budowania wydajnych rozwiązań za pomocą biblioteki RxJava.

3.4.1. Obsługa wielowątkowości

W celu obsługi wielowątkowości w bibliotece RxJava należy posłużyć się odpowiednim dyspozycorem (ang. Scheduler). Przekazując wszelkiego rodzaju kosztowne obliczenia, które mogą zachodzić na danych emitowanych przez obiekt obserwowany, do odpowiedniego rodzaju despozytora możemy zapewnić, że główny wątek aplikacji nigdy nie zostanie zablokowany. Idea wydaje się prosta, zamiast wykonywać przetwarzanie danych np. w tym samym wątku, który odpowiedzialny jest za obsługę interfejsu użytkownika, należy kosztowne operacje oddelegować do innego wątku. Aby wybrać, w jakim rodzaju dyspozytora dane zostaną przetworzone należy posłużyć się metodą SubscribeOn. W bibliotece RxJava jednymi z najważniejszych dyspozytorów są:

- `Schedulers.computation()` – służy do obsługi kosztownych manipulacji na danych. Domyślne liczba wątków w tym dyspozytorze jest taka sama jak liczba wątków procesora.
- `Schedulers.io()` – przeznaczony jest do obsługi operacji wejścia/wyjścia takich jak asynchroniczny zapis czy odczyt danych.

Kolejnym bardzo przydatnym operatorem jest operator `ObserveOn`. Jest on odpowiedzialny za określenie dyspozytora, w którym obserwator będzie mógł obserwować dane wyemitowane przez obiekt obserwowany.

3.4.2. Serwisy HTTP

W ramach obsługi serwisów HTTP w projekcie FindMyTutor dane zwracane z serwera aplikacji najpierw były kierowane do dyspozytora IO, a później obserwowane były w głównym wątku systemu Android `AndroidSchedulers.mainThread()`, (aby użyć wspomnianego dyspozytora niezbędne jest użycie biblioteki RxAndroid). Poniżej zaprezentowany został kod aplikacji odpowiedzialny za zdefiniowanie serwisu HTTP, który służy do obsługi modelu WhiteList. Model ten pozwala na dodawanie osób w aplikacji, dla który użytkownik chce być widoczny. Do podstawowej implementacji serwisu wykorzystana została biblioteka Retrofit w wersji drugiej.

```
package com.uam.wmi.findmytutor.service;

import com.uam.wmi.findmytutor.model.UsingListBool;
import com.uam.wmi.findmytutor.model.StudentIdModel;
import com.uam.wmi.findmytutor.model.User;

import java.util.List;

import io.reactivex.Completable;
import io.reactivex.Observable;
import io.reactivex.Single;
import retrofit2.http.Body;
import retrofit2.http.DELETE;
import retrofit2.http.GET;
import retrofit2.http.POST;
import retrofit2.http.PUT;
import retrofit2.http.Path;

public interface WhiteListService {

    @GET("api/users/{id}")
    Single<User> getUserById(@Path("id") String userID);

    @GET("api/users/whitelist/{tutorId}")
    Single<List<String>> getTutorWhitelist(@Path("tutorId") String tutorID);
}
```

```

    @PUT("api/users/whitelist/{tutorId}")
    Completable setTutorWhitelist(@Path("tutorId") String tutorID, @Body
    IsUsingListBool isUsing);

    @POST("api/users/whitelist/{tutorId}")
    Observable<User> addStudentToWhitelist(@Path("tutorId") String tutorID,
    @Body StudentIdModel student);

    @DELETE("api/users/whitelist/{tutorId}")
    Completable removeStudentFromWhitelist(@Path("tutorId") String tutorID,
    @Body StudentIdModel student);
}

```

Rysunek 3.21 Przykład definicji serwisu HTTP w aplikacji FindMyTutor

W powyższym przykładzie serwisu należy zwrócić uwagę na typy obserwatorów użytych dla poszczególnych metod. Przykładowo metoda getTutorWhitelist zwraca obserwator typu Single, ponieważ wykonywanie jest zapytanie GET. Wszystkie użyte typy obserwatorów i ich zastosowania zostały szerzej umówione w rozdziale 5.

Następny krokiem w projekcie było użycie wyżej wyszczególnionego serwisu w aplikacji. Aby wyświetlić użytkowników, którzy zostali dodani do białej listy należało pobrać listę, która zawiera ich identyfikatory i następnie dla każdego z nich pobrać ich szczegółowe dane. Pobranie listy użytkowników zostało zrealizowane w następujący sposób:

```

private Observable<List<String>> getListOfWhitelistedUsers(String userId) {
    return whiteListService.getTutorWhitelist(userId)
        .toObservable()
        .subscribeOn(Schedulers.io())
        .observeOn(AndroidSchedulers.mainThread());
}

```

Rysunek 3.22 Metoda zwracająca listę identyfikatorów osób, które znajdują się w białej liście

W zdefiniowanej metodzie warto zwrócić uwagę na zwracany typ. W definicji serwisu HTTP, metoda getTutorWhiteList zwraca typ Single jednak w wyżej wymienionym przykładzie użycie metody getListOfWhiteListedUsers skutkuje zwróceniem typu Observable. Odpowiedzialny jest za to operator toObservable, który pozwala na utworzenie obserwatora, który emitować będzie elementy listy o typie String (w tym przypadku są to identyfikatory użytkowników). Utworzenie obserwatora jest w tym przypadku niezbędne, ponieważ w dalszej części aplikacji dla każdego emitowanego elementu należy na jego podstawie wykonać kolejne zapytanie GET. W identyczny sposób zdefiniowała została metoda odpowiedzialna za pobranie szczegółowych informacji o danym użytkowniku.

```
private Observable<User> getUserObservable(String userId) {  
    return whiteListService  
        .getUserById(userId)  
        .toObservable()  
        .subscribeOn(Schedulers.io())  
        .observeOn(AndroidSchedulers.mainThread());  
}
```

Rysunek 3.23 Metoda zwracająca szczegółowe dane na temat danej użytkownika

Aby ostatecznie otrzymać listę użytkowników należy dla każdego wyemitowanego elementu o typie String (identyfikator użytkownika) z metody getListOfWhitelistedUsers wywołać metodę getUserObservable, która pozwali na otrzymanie listy z obiektami o typie User. Poniżej zaprezentowano implementację tego rozwiązania.

```
private void fetchWhiteListedUsers() {  
    disposable.add(getListOfWhitelistedUsers(tutorId)  
        .doOnSubscribe(this::handleDoOnSubscribe)  
        .subscribeOn(Schedulers.io())  
        .observeOn(AndroidSchedulers.mainThread())  
        .flatMap(Observable::fromIterable)  
        .flatMap(this::getUserObservable)  
        .subscribe(user -> whitelistedUsers.add(user),  
            this::handleError,  
            this::handleComplete)  
    );  
}
```

Rysunek 3.24 Metoda odpowiedzialna za pobranie listy osób dodanych do białej listy

Oto kroki, które zostają wywołane w metodzie `fetchWhiteListedUsers`:

1. Pobranie listy elementów o typie `String`, która zawiera identyfikatory użytkowników, którzy zostali dodani do bieżącej listy. W tym miejscu zostaje zwrócony obserwator.
2. Skierowanie danych do odpowiedniego dyspozytora i obserwacja ich w głównym wątku aplikacji.
3. Podczas rozpoczęcia obserwowania wykonanie metody `handleDoOnSubscribe`. Jest to metoda odpowiedzialna m.in. za wyświetlenie informacji o rozpoczęciu pobierania danych w interfejsie użytkownika.
4. Używając operatora `flatMap` przekształcenie otrzymanych elementów z obserwatora zwróconego przez metodę `getListOfWhitelistedUsers` w jedno-wymiarową listę, której elementami będą nowo utworzone obiekty typu obserwator. W tym przypadku, aby zmapować listę składającą się z elementów o typie `String` użyta została metoda `FromIterable`, która skutkuje utworzeniem obserwatora, który emitować będzie kolejne obserwatory o typie `User`. Jest to niezbędny zabieg, który pozwala na równoległe odpytywanie warstwy serwerowej aplikacji.
5. Powtórne użycie operatora `flatMap` dla każdego obserwatora wraz z metodą `getUserObservable` skutkuje otrzymaniem obserwatora emitującego już elementy o typie `User`.
6. Używając operatora `subscribe` stworzenie obserwatora, który otrzymane elementy o typie `User` zapisuje do listy połączonej z interfejsem użytkownika.
7. Postługując się metodą `onComplete` w obserwatorze odebrana zostaje informacja o zakończeniu przetwarzania danych, co skutkuje wywołaniem metody `handleComplete`, która odpowiedzialna jest za aktualizację interfejsu użytkownika.

```
private void handleComplete() {  
    Collections.sort(whitelistedUsers, this::sortByUserName);  
    didFetched = true;  
    refreshUI();  
}
```

Rysunek 3.25 Przykład metody wywołanej po zakończeniu przetwarzania danych

3.4.3. Wyszukiwarka użytkowników

W celu stworzenia wyszukiwarki użytkowników utworzona została specjalna klasa RxSearchObservable, w której zaimplementowano logikę odpowiedzialną za obsługę interfejsu użytkownika. Działanie tej klasy opiera się o obiekt typu Subject, który pochodzi z biblioteki RxJava. Subject jest rodzajem połączenia ze sobą bytu obserwowanego i obserwatora. Ponadto jako byt obserwowany emituje on swoje dane dla obserwatorów tylko od momentu rozpoczęcia obserwowania przez danego obserwatora. Jest to, więc obiekt typu „hot” [81]. W ten sposób dany obserwator nie jest w stanie otrzymać danych, które zostały wyemitowane przed jego subskrypcją.

```
package com.uam.wmi.findmytutor.utils;

import android.support.v7.widget.SearchView;

import io.reactivex.Observable;
import io.reactivex.subjects.PublishSubject;
public class RxSearchObservable {

    public static Observable<String> fromView(SearchView searchView) {
        final PublishSubject<String> subject = PublishSubject.create();
        searchView.setOnQueryTextListener(new SearchView.OnQueryTextListener() {
            @Override
            public boolean onQueryTextSubmit(String s) {
                subject.onNext(s);
                searchView.clearFocus();
                return false;
            }
            @Override
            public boolean onQueryTextChange(String text) {
                subject.onNext(text);
                return false;
            }
        });
        return subject;
    }
}
```

Rysunek 3.26 Klasa odpowiedzialna za implementację obserwatora emitującego wpisane wartości w pole wyszukiwarki

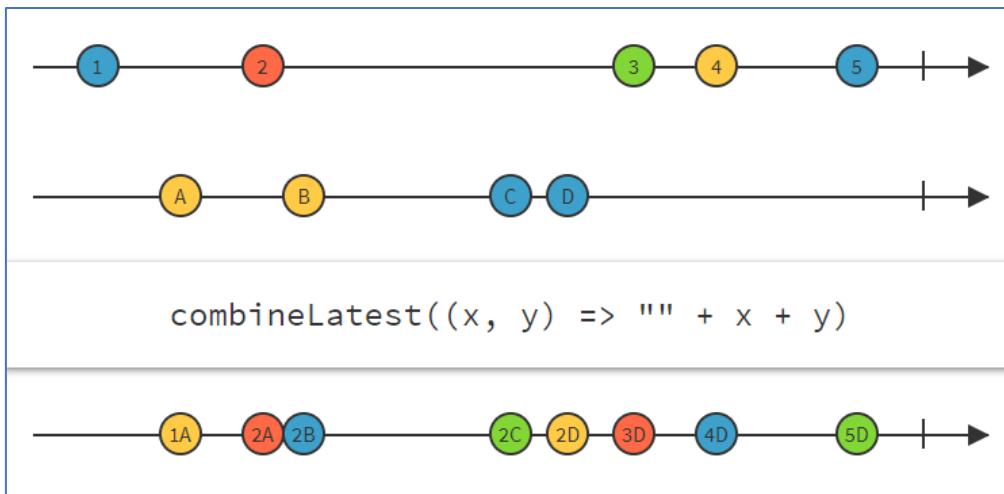
Aby przekształcić dane pochodzące z pola odpowiedzialnego za wyszukiwanie użytkowników posłużono się kilkoma operatorami, które odpowiedzialne były między innymi za konwersję wpisanych danych tekstowych do mały liter i odfiltrowanie pustych wartości. Jednym z najważniejszych operatorów, który został zastosowany podczas implementacji wyszukiwania użytkowników był operator debounce. Sprawia on, iż obserwator wyemitemuje dane dopiero po upływie danego interwału czasowego, co skutkuje tym, że zapytanie o wyszukanie nie nastąpi po wpisaniu każdego kolejnego znaku, lecz zostanie wywołane tylko po zdefiniowanej przerwie od wprowadzenia ostatniej wartości w polu wyszukiwarki.

```
RxSearchObservable.fromView(searchView)
    .skip(0)
    .map(String::toLowerCase)
    .filter(t -> !t.isEmpty())
    .debounce(250, TimeUnit.MILLISECONDS)
    .subscribeOn(Schedulers.io())
    .observeOn(AndroidSchedulers.mainThread())
    .subscribe(this::executeSearch);
```

Rysunek 3.27 Przykład kodu, który obrazuje przetwarzanie danych emitowanych przez obserwatora wyszukiwania

3.4.4. Weryfikacja formularza logowania

Celem walidacji formularza służącego do logowania użytkowników było zablokowanie lub odblokowanie przycisku, który umożliwia próbę zalogowania się w aplikacji. Aby ułatwić implementację obserwatora emitującego dane z pól tekstowych w projekcie FindMyTutor użyta została biblioteka RxBinding, która odpowiedzialna jest za tworzenie obserwatorów na podstawie danego elementu interfejsu użytkownika [82]. W aplikacji stworzone zostały dwa obiekty, które emitują zmianę stanu pola tekstowego. Jeden z nich odpowiedzialny jest za obsługę pola login a drugi za pole password. Obsługa przycisku umożliwiającego zalogowanie została zaimplementowana za pomocą operatora CombineLatest. Łączy on ze sobą ostatnio wyemitowane dane dwóch obserwatorów za pomocą podanej funkcji, która zwraca nowe dane [83].



Rysunek 3.28 Obrazowe przedstawienie działania funkcji `combineLatest` [84]

W przypadku projektu FindMyTutor przycisk logowania jest dostępny tylko, gdy obydwa pola formularza zostaną pomyślnie zwalidowane.

```

private void validateForm() {
    sign_in_button.setEnabled(false);
    mLoginNameView.setError(null);
    mPasswordView.setError(null);
    rx.Observable<TextViewTextChangeEvent> emailChangeObservable =
        RxTextView.textChangeEvents(email);
    rx.Observable<TextViewTextChangeEvent> passwordChangeObservable =
        RxTextView.textChangeEvents(password);
    rx.Observable.combineLatest(emailChangeObservable,
        passwordChangeObservable,
        (emailObservable, passwordObservable) -> {
            boolean emailCheck = isValidEmail(emailObservable.text());
            boolean passwordCheck = passwordObservable.text().length() > 0;
            if (!emailCheck) {
                mLoginNameView.setError(getString(R.string.error_invalid_login_name));
            }
            return emailCheck && passwordCheck;
        }).subscribe(aBoolean -> sign_in_button.setEnabled(aBoolean));
}
  
```

Rysunek 3.29 Metoda odpowiedzialna za walidację formularza w ekranie logowania aplikacji FindMyTutor

3.5. Podsumowanie

Użycie programowania reaktywnego wraz z biblioteką RxJava w projekcie FindMyTutor ukazuje szeroki wachlarz możliwych zastosowań paradygmatu reaktywnego. Omawiane podejście wraz z takimi bibliotekami jak RxAndroid i RxBinding pozwoliło na prostą integrację omawianego paradygmatu ze środowiskiem Android. Biblioteka RxJava okazała się być stabilnym i dojrzałym narzędziem, którego użycie poskutkowało mniejszą złożonością aplikacji. Warto zaznaczyć, że tego typu podejście do budowania aplikacji nie musi kojarzyć się tylko z budowaniem interfejsu użytkownika. Wieloplatformowe implementacje biblioteki ReactiveX ukazują wysokie zainteresowanie i zapotrzebowanie na tego typu narzędzia przez programistów.

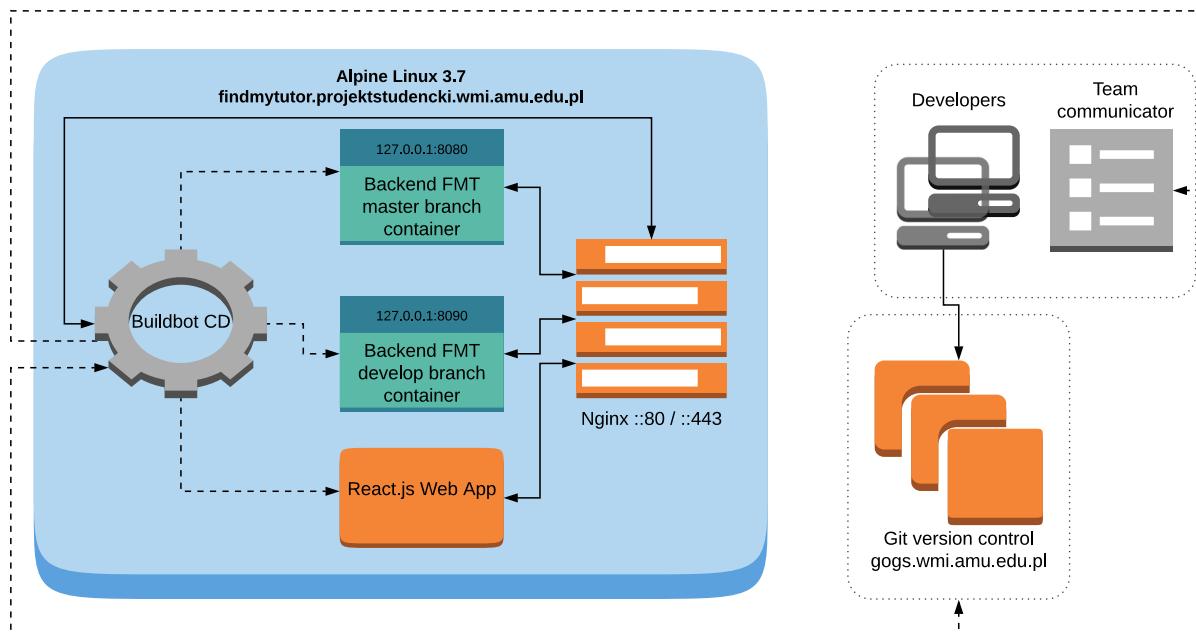
Rozdział 4

4. Wdrożenia aplikacji serwerowych opartych o kontenery w metodyce ci/cd

4.1. Wstęp – potrzeby zespołu deweloperskiego

W trakcie prac nad projektem FindMyTutor zespół napotkał problemy które przyczyniły się do wdrożenia systemu CD¹opartego o kontenery Docker. Głównym celem przedsięwzięcia było uniezależnienie procesu dostarczania kolejnych wersji aplikacji serwerowej od jednej osoby w taki sposób by zużyć jak najmniej zasobów otrzymanej od uczelni maszyny wirtualnej.

Takie ograniczenie wynikało z niedużych zasobów sprzętowych przydzielonej maszyny, która oprócz obsługi ciągłego dostarczania miała również obsługiwać serwer www, kontenery API FindMyTutor oraz wersję webową interfejsu do aplikacji. Zamieszczony poniżej diagram wizualizuje ogólną strukturę środowiska deweloperskiego.



Rysunek 4.1 Diagram środowiska deweloperskiego w projekcie FindMyTutor

¹ CD - ciągłego dostarczania

Ze względu na asynchronousność pracy poszczególnych członków zespołu szybko stało się jasne, że jeśli wdrażanie zmian po stronie serwerowej będzie odbywało się ręcznie przez jedną osobę to zmniejszy to znaczco produktywność całego zespołu. W początkowych etapach projektu niejednokrotnie zachodziły sytuację w których zmiany wypchnięte do repozytorium części serwerowej czekały na osobę odpowiedzialną za wdrożenie podczas gdy programiści zajmujący się widokiem aplikacji byli gotowi do przetestowania nowych zmian na serwerze.

Ponieważ tempo rozwoju projektu stale narastało postanowiono wykorzystać dostępną technologię konteneryzacji razem z serwisem ciągłego dostarczania. Automatyzacja procesu pozwoliła zaoszczędzić czas deweloperów jednocześnie zwiększając okno czasowe na testowanie nowych funkcjonalności oprogramowania.

W następnych podrozdziałach przedstawię obecny krajobraz technologii konteneryzacji aplikacji pod systemem Linux oraz procesów wdrażania korzystających z ich zalet.

4.2. Biznesowa analiza konteneryzacji

Ponieważ pojęcie kontenerów jest często wiązane z konkretną implementacją technologiczną, warto zastanowić się czym są kontenery w szerszym bardziej abstrakcyjnym pojęciu. Pomimo różnic jakie zachodzą pomiędzy rozwiązaniami obecnymi na rynku możemy wyodrębnić zestaw funkcjonalnych cech, które są wspólne dla każdej technologii konteneryzacji.

Te podobieństwa wynikają przede wszystkim z narastających potrzeb branży technologicznej na coraz szybsze i wydajniejsze finansowo dostarczanie usług i wprowadzanie nowych funkcjonalności do swoich projektów. Dlatego pomimo różnic w implementacji technicznej poszczególnych systemów konteneryzacji funkcjonalnie oraz biznesowo większość cechuje podobna charakterystyka. Aby objaśnić jak dokładnie działają musiałbym się wdać w techniczną analizę, jednakże na razie skupię się na tym by opisać je szerzej od strony korzyści jakie niosą dla branży technologicznej. Zdaje się to być kluczowym elementem potrzebnym do zrozumienia jak ważnym elementem są kontenery we współczesnym procesie tworzenia oprogramowania przez firmy.

Kontenery w swoim założeniu od strony biznesowej są odpowiedzią na trzy problemy:

4.2.1. Udogodnienie dostarczania aplikacji dla klientów

Aby osiągnąć te cele technologie konteneryzacji skupiają się na opakowaniu środowiska, w którym uruchamiane jest oprogramowanie dostarczone przez deweloperów w zamknięty

układ funkcjonujący w podobny sposób jak osobny system operacyjny. Różnica polega na tym, że spakowane w ten sposób środowiska można uruchomić na jednym systemie służącym jako gospodarz (ang. *host*) tak jak standardowe procesy systemu operacyjnego.

Pojawia się zatem pytanie po co rozpowszechniać oprogramowanie w postaci kontenerów, skoro w efekcie końcowym uzyskujemy rezultat zbliżony do standardowej dystrybucji oprogramowania, gdzie po prostu publikujemy kod źródłowy bądź pliki binarne wraz z opisem zależności i instrukcją uruchomienia.

Jeśli na etapie planowania projektu firma deweloperska zdecyduje się na wykorzystanie technologii kontenerów w architekturze swojego rozwiązania to minimalizuje w ten sposób ryzyko potencjalnych problemów wdrożeniowych w końcowej fazie projektu.

Kontenery ze względu na swoją naturę enkapsulacji wyprodukowanego oprogramowania wraz z odpowiednimi narzędziami do zarządzania tworzą warstwę abstrakcji nad systemem operacyjnym na którym działają. Zależności oraz ich odpowiednie wersje potrzebne do uruchomienia oprogramowania znajdują się w odizolowanym systemu plików. Dzięki temu aplikacja rozwijana i testowana na innym systemie operacyjnym może być bezproblemowo przeniesiona i wdrożona w infrastrukturze klienta bez większych wymagań co do systemów operacyjnych z jakich korzysta.

Dzięki temu po przekazaniu projektu dział utrzymania może zarządzać infrastrukturą w bardziej swobodny sposób systemy, na których działają skonteneryzowane aplikacje można aktualizować nie martwiąc się tym, że aktualizacja łamie wstępную kompatybilność i któryś z elementów wdrożonego projektu przestanie działać.

4.2.2. Udogodnienie procesu tworzenia oprogramowania

Te same cechy, które ułatwiają dostarczanie aplikacji opakowanych w kontenery zwiększają również produktywność zespołu odpowiedzialnego za tworzenie oprogramowania. Narzędzia używane przez programistów zazwyczaj mają swoje zależności niezbędne do korzystania z oprogramowania. Problem staje się jeszcze bardziej widoczny, jeśli zespół korzysta z niszowej technologii, która posiada szczątkową dokumentację i wymaga wielu etapów komplikacji do działania. Opakowując takie narzędzie w kontener możemy pomóc sobie oraz przyszłym członkom zespołu którzy dołączając do projektu będą mogli poświęcić mniej swojego czasu na konfigurację narzędzi potrzebnych do pracy. Przykładem takiej sytuacji może być firma Lyft która postawiła sobie za cel ułatwienie procesu wdrożenia nowych członków do tego stopnia, aby już pierwszego dnia pracy nowy pracownik mógł wypchnąć do repozytorium kodu swoje pierwsze zmiany [85].

4.2.3. Optymalizacja kosztów

Dobrą praktyką wdrażania systemów informatycznych jest izolacja ich poszczególnych komponentów od siebie. Powodem takiego podejścia są względy bezpieczeństwa, w razie nieautoryzowanego dostępu lub losowej usterki tylko część systemu staje się obarczona konsekwencjami podczas gdy reszta pozostaje funkcjonalna.

Wdrażanie systemów informatycznych przed popularyzacją kontenerów opierano zazwyczaj na maszynach wirtualnych które często wiążą się z alokowaniem zasobów do obsługi systemu ponad miarę co generuje niepotrzebne koszty. Mając na uwadze wspomniane wcześniej kryterium izolacji przy wdrażaniu projektów za pomocą maszyn wirtualnych mamy zazwyczaj do czynienia, że środowiskami które składają się z kilku niezależnych maszyn, każda zawierająca pełnoprawny system operacyjny wymagający do działania własnego procesora, pamięci oraz przestrzeni dyskowej. Są to zasoby w części konsumowane przez systemy operacyjne w wyniku wymagań jakie stawiają maszyny wirtualne bez żadnego biznesowego zysku.

Opierając wdrożenie na kontenerach cały system można szybko wdrożyć korzystając tylko z jednego systemu operacyjnego który będzie środowiskiem uruchomieniowym dla opakowanych komponentów naszego systemu informatycznego oszczędzając przy tym ilość czasu potrzebną na stworzenie licznych maszyn wirtualnych oraz koszty ich utrzymania.

Serwisy które mają do czynienia z dużym ruchem dodatkowo korzystają z maksymalnej utylizacji przydzielonych zasobów do obsługi kontenerów oraz przy odpowiedniej architekturze serwisu skalowalności takiego systemu. Ze względu na krótki czas potrzebny na stworzenie nowego kontenera istnieje wiele rozwiązań umożliwiających badanie natężenia ruchu z jakim zmaga się serwis i tworzenie lub niszczenie nowych instancji. Takie podejście umożliwia firmom zapewnienie usług, które będą w stanie obsłużyć nieprzewidywalny wzrost zainteresowania tym samym maksymalizując zyski a jednocześnie otrzymując możliwość redukcji swojej infrastruktury podczas spadku liczby konsumentów minimalizując swoje niepotrzebne koszty.

Kwestie finansowe zdają się być wystarczającym kryterium sprawiającym, że coraz więcej firm decyduje się na migrację swoich systemów na rozwiązania kontenerowe, przykładem takich firm może być MapBox [86] albo Financial Times [87], które po transformacji upubliczniły wartość redukcji kosztów na poziomie 50-80% w porównaniu do infrastruktury opartej na maszynach wirtualnych.

Ze względu na wymienione powyżej cechy w ostatnich latach duże firmy technologiczne oraz inwestorzy przeznaczają spore środki finansowe na rozwój kontenerów lub zakup firm posiadających gotowe rozwiązania konteneryzacji.

Przykładem takich zjawisk może być wzrost wartości obecnego lidera pod względem udziału rynkowego [88] firmy Docker Inc. w którą na przestrzeni lat 2010-2017 zainwestowano łącznie ponad 270 milionów USD [89]. Docker Inc. nie jest jedyną tak wysoko cenioną firmą w dziedzinie konteneryzacji, w styczniu 2018 roku Red Hat Inc. nabył firmę CoreOS Inc.,

producenta alternatywnego rozwiązania technologii kontenerów w transakcji o wartości 250 milionów USD [90].

4.3. Kontenery

Aby omówić to jak obecnie działają mechanizmy odpowiedzialne za tworzenie i uruchamianie skonteneryzowanych aplikacji warto spojrzeć na historię rozwiązań technicznych które zbliżyły informatykę do konteneryzacji jaka jest nam obecnie znana. Jednocześnie spróbuje przedstawić czemu omawiana technologia stała się tak powszechnie używana we współczesnych projektach.

4.3.1. Historia konteneryzacji

Za pierwszy krok w stronę rozwiązań pozwalających na izolację i enkapsulację uznaje się wprowadzenie polecenia *chroot* w systemie Unix V7 w roku 1979. Jest to komenda, która weszła na stałe do kanonu poleceń współczesnych systemów opartych o jądro Linux. Dzięki mechanizmowi zaimplementowanemu w poleceniu umożliwiono nadanie uruchamianemu procesowi innego katalogu głównego tym samym dając mu dostęp tylko do danego poddrzewa całego systemu plików. Jest to krok symboliczny w stronę separacji procesów współdzielących wspólne jądro systemu Linux.

Ponieważ współczesna konteneryzacja jest zestawem różnych technologii powiązanych ze sobą razem chciałbym przedstawić chronologicznie rozwiązania, które przybliżały nas do kontenerów jakie znamy wraz z rokiem ich debiutu [91].

2000 – FreeBSD Jails

Komenda systemu FreeBSD bazująca na *chroot* rozszerzając bezpieczeństwo i uniemożliwiająca ucieczkę z poddrzewa katalogu. Dodatkowo wprowadziła możliwość przypisania danej jednostce izolacji własnego adresu IP oraz wskazanie pliku wykonywalnego który powinien zostać uruchomiony przy włączeniu jednostki [92].

2002 – Linux kernel namespaces

Moduł jądra system Linux odpowiedzialny za rozgraniczanie przestrzeni nazw identyfikatorów obecnych w systemie operacyjnym. Umożliwia uruchomienie procesu z zestawem zdefiniowanych przestrzeni nazw. Obecnie wspierane są następujące przestrzenie nazw [93]:

- *User* - rozgranicza jakich użytkowników proces widzi
- *Network* – narzuca ograniczenie na widoczność dostępnych skonfigurowanych sieci
- *Mount* – odpowiada za widoczność dostępnych zamontowanych systemów plików
- *Cgroups* – rozgranicza jakie grupy kontrolne dany proces widzi
- *Pid* – odpowiada za ograniczenie widoczności innych procesów uruchomionych w systemie, dzięki temu nie dochodzi do kolizji procesów oznakowanych takim samym numerycznym identyfikatorem
- *Ipc* – rozróżnia widoczność systemowych kolejek przerwań i komunikacji pomiędzy procesami
- *Uts* – umożliwia rozgraniczenie nazwy hosta widocznej dla procesu

2004 – Oracle Solaris Containers (Zones)

Technologia konteneryzacji wprowadzona w płatnym systemie Solaris 10 od firmy Oracle. Rozwiązanie umożliwia podział systemu na tzw. zony które są od siebie niezależne i nie wiedzą nawzajem o swoim istnieniu. Dodatkowo korzystając z wprowadzonego systemu plików ZFS dodano możliwość tworzenia obrazów zon i tworzenia nowych instancji na ich podstawie [94].

2006 – Process Containers (cgroups)

Projekt stworzony przez Google mający na celu dodanie możliwości podziału uruchomionych w systemie procesów na grupy, z których każda grupa może posiadać inne prawa dostępu do sprzętowych zasobów systemowych. Jest to kluczowy dla konteneryzacji podsystem jądra Linux. Rok później zmiany zaproponowane przez Google zostały zaakceptowane i wcielone do jądra w wersji 2.6.24 [95].

2008 – LXC

LXC (akronim od LinuX Containers) to rozwiązanie bazujące na opisanych *namespaces* oraz *cgroups* realizujące tworzenie wirtualnych środowisk systemów bazujących na jednym wspólnym jądrze systemu Linux, dzięki niemu jesteśmy w stanie uruchomić na dystrybucji Ubuntu środowisko, które będzie zawierało system CentOS zachowując przy tym wydajność stworzonego środowiska bliską natywnej instalacji. Ze względu na ten sposób funkcjonowania możemy potocznie nazwać jednostki LXC jako „kontenery pełno systemowe” [96].

2013 – Docker

Najpopularniejsza w tej chwili technologia dostarczająca zestaw narzędzi do budowania, uruchamiania oraz zarządzania kontenerami. Z początku rozwiązanie bazowało na podstawie LXC które zostało porzucone przez Docker Inc. w 2014 roku na rzecz własnego środowiska uruchomieniowego o nazwie *runC*. Pomimo nowej architektury, środowisko wciąż bazuje w dużej mierze na możliwościach jakie zapewniają *cgroups* oraz *namespaces*. W odróżnieniu od LXC kontenery Docker skupiają się na jak najlżejszych wyizolowanych środowiskach zalecając podejście „jeden kontener = jeden uruchomiony w środku serwis” [97].

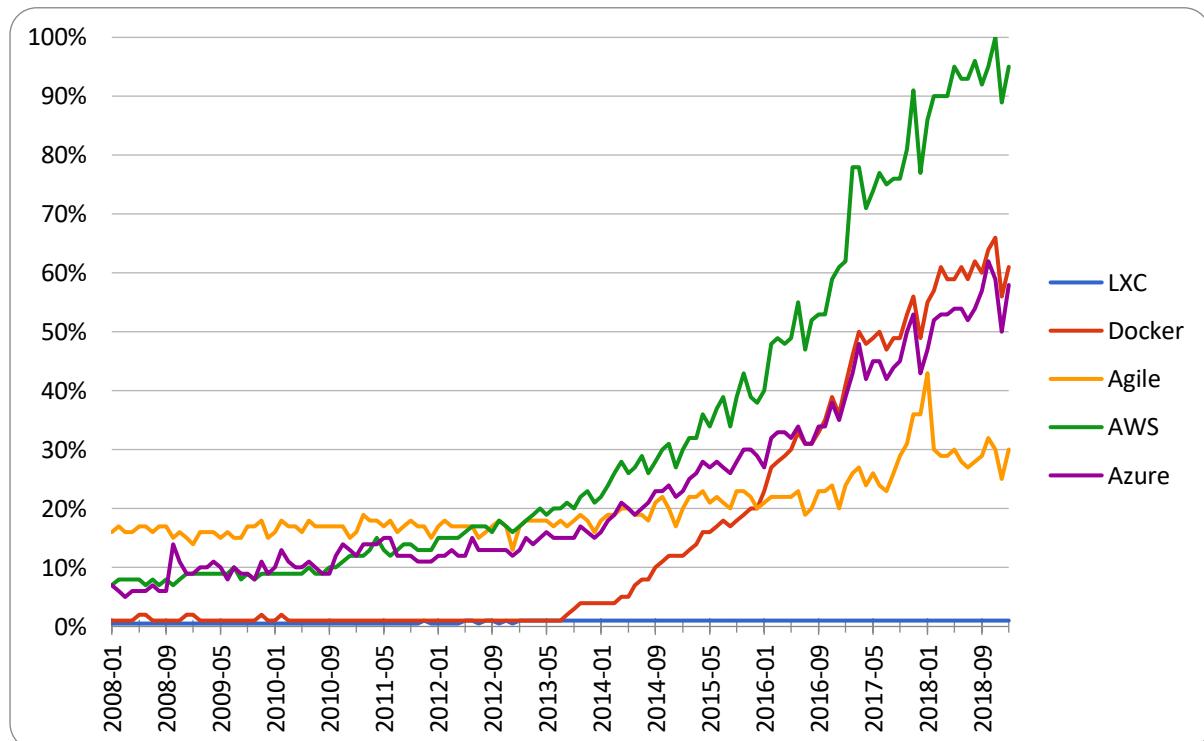
2014 – rkt

Rozwiązanie zainspirowane wzrostem popularności kontenerów Docker zostało według twórców zaprojektowane w taki sposób, aby zaadresować problemy architektury Docker'a. Główną różnicą był brak centralnego demona zarządzającego kontenerami [98] oraz wbudowana integracja z systemami rozruchowymi np. *systemd* umożliwiająca zarządzanie kontenerami jak zwykłymi procesami systemu operacyjnego [99]. Obecnie Docker również przeszedł na model bez centralnego demona zarządzającego. Podobnie jak wspomniany wyżej Docker, rkt również w dużej mierze utylizuje mechanizmy zaimplementowane w *cgroups* oraz *namespaces*.

4.3.2. Analiza wzrostu popularności kontenerów

Warto *pokrótko* zastanowić się nad tym, dlaczego kontenery stały się tak popularne na przestrzeni kilku ostatnich lat pomimo tego, że mechanizmy z których korzystają są w większości znane od dawna. Aby zwizualizować trendy w technologii konteneryzacji poniżej możemy zobaczyć wykres popularności wyszukiwań w wyszukiwarce Google.

Na wykresie zostały załączone wyniki względnego rozkładu ilości zapytań.



Rysunek 4.2 Wykres względnej ilości zapytań w serwisie google.com

Źródło: Google Trends

Analizując powyższy wykres można zauważać, że początek popularyzacji kontenerów Docker będący ich premierą zbiega się w czasie z rozkwitem metodyk zwinnych oraz środowisk chmurowych. Wiąże się to z coraz szybszym tempem pisania oprogramowania oraz wdrażaniem gotowych projektów na zdalnej infrastrukturze. Docker Inc. dostarczając przystępny użytkownikowi interfejs oraz sporo dokumentacji zaadresował oba te problemy. Deweloperzy zamiast czekać na wewnętrzny dział IT aż stworzy im maszynę wirtualną by sprawdzić coś nowego mogli uruchomić interesujące ich oprogramowanie w kilka sekund lokalnie z natywną wydajnością. Co więcej korzystając z platformy internetowej do współdzielenia obrazów dostarczonej przez Docker Inc. o nazwie Docker Hub programiści z różnych stron świata mogli korzystać z gotowych obrazów oraz publikować swoje własne kreacje.

Jednocześnie możemy zwrócić uwagę na dość stałe w czasie minimalne zainteresowanie technologią LinuX Containers która prawdopodobnie zadebiutowała wyprzedzając nieco swoje czasy.

Ze względu na omówioną popularyzację, w 2015 roku z inicjatywy Docker Inc, CoreOS Inc. oraz 21 innych firm pod opieką Linux Foundation została zawiązana organizacja Open Container Initiative skupiająca się na standaryzacji środowisk uruchomieniowych kontenerów oraz ich formatu [100].

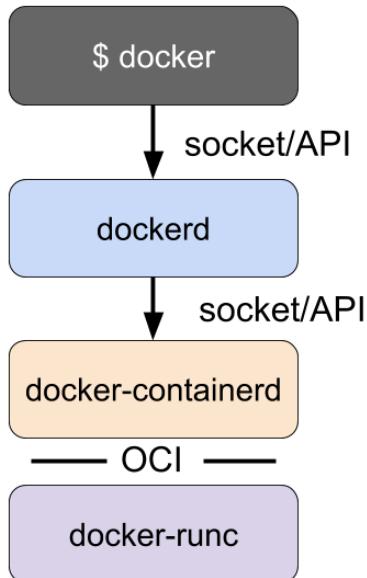
4.4. Kontenery Docker - wdrażanie aplikacji serwerowych

Ze względu na opisaną wcześniej popularność narzędzi oferowanych przez Docker Inc. oraz fakt wykorzystania tej technologii w projekcie inżynierskim chciałbym przybliżyć procesy zachodzące podczas budowania oraz uruchamiania procesów w kontenerach Docker pod systemem Linux.

4.4.1. Zasada działania

Zaczynając od poziomu użytkownika interakcja z kontenerami Docker zachodzi głównie przez wywołania zestawu poleceń składających się na *docker-cli* wchodzących w interakcję z Docker Engine który odpowiada za tłumaczenie poleceń na standard zrozumiały dla demona *containerd* – projektu rozwijanego przez wspomnianą wcześniej Open Container Initiative. To właśnie *containerd* jest finalnie odpowiedzialne za pobieranie obrazów, start kontenerów i zarządzanie ich cyklem życia [101].

Jest to narzędzie terminalowe umożliwiające budowanie i uruchamianie kontenerów które inicjuje łańcuch bardziej niskopoziomowych wywołań.



Rysunek 4.3 łańcuch wywołań inicjowany przez narzędzia *docker-cli* [102]

4.4.2. Budowanie obrazów

Przenośność oraz łatwość w dostarczaniu skonteneryzowanego oprogramowania wynika z formatu w jakim są eksportowane. Obraz kontenera jest jego spakowaną jednostką uruchomieniową zawierającą system plików, zmienne środowiskowe oraz metadane określające właściwości zdefiniowane podczas budowania.

Aby ułatwić ten proces można skorzystać z narzędzi *docker build* oraz pliku konfiguracyjnego *Dockerfile* korzystającego z dedykowanego języka określającego poszczególne etapy budowy. Po wykonaniu komendy plik jest interpretowany przez *dockerd* odpowiadającego za wygenerowanie obrazu zgodnego z standardem OCI.

```
1  #Build stage
2
3  FROM microsoft/dotnet:2.1-sdk-alpine AS build-env
4  WORKDIR /app
5  COPY findMyTutorBackend.csproj .
6  COPY nuget.config .
7  RUN ["dotnet", "restore","--verbosity","diag","--configfile","nuget.config"]
8  COPY ..
9  RUN ["dotnet", "publish","-c","Release","-o","/publish"]
10
11 #Runtime Stage
12
13 FROM microsoft/dotnet:2.1.3-aspnetcore-runtime-alpine3.7
14 RUN apk add --no-cache icu-libs
15 ENV DOTNET_SYSTEM_GLOBALIZATION_INARIANT=false
16 WORKDIR /publish
17 COPY ca_labs.wmi.amu.edu.pl.pem /usr/share/ca-certificates/ca_labs.wmi.amu.edu.pl.crt
18 RUN echo ca_labs.wmi.amu.edu.pl.crt >> /etc/ca-certificates.conf
19 RUN ["update-ca-certificates"]
20 COPY --from=build-env /publish .
21 EXPOSE 80/tcp
22 ENV ASPNETCORE_ENVIRONMENT=Production
23 ENTRYPOINT [ "dotnet","findMyTutorBackend.dll" ]
```

Rysunek 4.4 Dockerfile odpowiedzialnego za budowę aplikacji serwerowej FindMyTutor.

Polecenia wykorzystane w powyższym pliku tłumaczą się na następujące akcje [103]:

- **FROM** – instrukcja wskazująca pierwszą warstwę naszego kontenera, zawiera rozwiązywanie nazw obrazów w formacie <autor>/<obraz>:<wersja> przeszukując skonfigurowane rejestyry obrazów. Jeśli zostanie znalezione dopasowanie to obraz podczas budowy zostanie pobrany ze zdalnego rejestru do lokalnego indeksu.
- **WORKDIR** – ustawia ścieżkę do katalogu roboczego który obowiązuje wszystkie komendy występujące po tej instrukcji
- **COPY** – kopiuje wskazane pliki z obecnej warstwy pośredniej tworząc z nich następną warstwę pośrednią
- **RUN** – wykonuje podane komendy
- **ENV** – ustawia zmienne środowiskowe udostępniając je następnym instrukcjom oraz zawierając je w finalnym obrazie
- **EXPOSE** – ustawia metadane budowanego obrazu informujące na jakich portach może nastąpić spakowana w środku aplikacja
- **ENTRYPOINT** – ustawia ścieżkę do pliku binarnego bądź polecenia, które zostanie wykonane podczas startu kontenera

Podczas procesu składania obrazu kontenera stosowana jest strategia rozbijania etapów budowy na przyrostowe różnice budowanego systemu plików i tworzenie warstw z zarejestrowanych różnic. Tym sposobem uzyskana warstwowa konstrukcja jest składowana w lokalnym indeksie obrazów wraz z obliczoną funkcją skrótu sha256 która jest jej identyfikatorem.

Składnia *Dockerfile* została zaprojektowana w taki sposób, aby jedna linia pliku konfiguracyjnego odpowiadała jednej zmianie w budowanym systemie plików tym sposobem każdy etap jest odkładany podczas budowy jako warstwa nazywana obrazem pośrednim. Dzięki temu, jeśli nastąpi przebudowanie obrazu i nie zostaną wykryte żadne zmiany w danym kroku to mechanizm budujący skorzysta z odłożonej warstwy pośredniej tym samym znacznie skracając czas potrzebny na zbudowanie pełnego obrazu kontenera.

```
1
2 Step 1/18 : FROM microsoft/dotnet:2.1-sdk-alpine AS build-env
3    |--> c751b3a7f4de
4 Step 2/18 : WORKDIR /app
5    |--> 9d04c8562bdb
6 Step 3/18 : COPY findMyTutorBackend.csproj .
7    |--> b22097260de6
8 Step 4/18 : COPY nuget.config .
9    |--> cb49ac2c8a11
10 Step 5/18 : RUN ["dotnet", "restore","--verbosity","diag","--configfile","nuget.config"]
11    |--> Running in 0c17a016d266
12
13
14
15 Step 18/18 : ENTRYPOINT [ "dotnet","findMyTutorBackend.dll" ]
16    |--> Running in d8dacd65ce02
17 Removing intermediate container d8dacd65ce02
18    |--> b970ac0d7528
19 Successfully built b970ac0d7528
20 Successfully tagged mjedynski/fmt-backend-develop:latest
21 Successfully tagged mjedynski/fmt-backend-develop:2019-01-17-234c3a0
22 program finished with exit code 0
23 elapsedTime=164.216391
```

Rysunek 4.5 Proces budowy z pustym indeksem obrazów pośrednich.

Powyżej został umieszczony wycinek procesów tworzenia obrazu kontenera aplikacji FindMyTutor po wcześniejszym wyczyszczeniu lokalnego indeksu z obrazów pośrednich. W logach produkowanych przez polecenie *docker build* jest widoczne, że plik *Dockerfile* jest interpretowany linia po linii i wykonaniu każdego kroku towarzyszy wygenerowany ze zmian obraz pośredni oznaczony wartością funkcji skrótu.

Wykonując jeszcze raz to samo polecenie w wygenerowanych logach zachodzą zmiany w porównaniu do pierwszego przebiegu.

```
1
2 Step 1/18 : FROM microsoft/dotnet:2.1-sdk-alpine AS build-env
3 |--> c751b3a7f4de
4 Step 2/18 : WORKDIR /app
5 |--> Using cache
6 |--> 9d04c8562bdb
7 Step 3/18 : COPY findMyTutorBackend.csproj .
8 |--> Using cache
9 |--> b22097260de6
10 Step 4/18 : COPY nuget.config .
11 |--> Using cache
12 |--> cb49ac2c8a11
13 Step 5/18 : RUN ["dotnet", "restore","--verbosity","diag","--configfile","nuget.config"]
14 |--> Using cache
15
16 .....
17
18 Step 18/18 : ENTRYPOINT [ "dotnet","findMyTutorBackend.dll" ]
19 |--> Running in 23ead50f06c
20 Removing intermediate container 23ead50f06c
21 |--> cbb7c28c697f
22 Successfully built cbb7c28c697f
23 Successfully tagged mjedynski/fmt-backend-develop:latest
24 Successfully tagged mjedynski/fmt-backend-develop:2019-01-17-234c3a0
25 program finished with exit code 0
26 elapsedTime=30.807558
```

Rysunek 4.6 Proces budowy z wykorzystaniem obrazów pośrednich.

Można zauważyć ponad pięciokrotne przyśpieszenie procesu, jeśli Docker Engine znajdzie potrzebne obrazy w swoim lokalnym indeksie.

4.4.3. Uruchamianie i zarządzanie

Aby wykorzystać stworzony obraz pakiet *docker-cli* dostarcza polecenie *doker run* przekazujące parametry startowe do Docker Engine który z kolei odpowiada za przekazanie polecenia do *containerd* który sprawdza lokalne i zdalne rejestracje obrazów i jeśli znajdzie dopasowanie uruchamia niskopoziomowe środowisko uruchomieniowe *runC*.

Po uruchomieniu kontenera podobnie jak w procesie jego budowy dodawana jest jeszcze jedna warstwa systemu plików która służy za nietrwały obszar do odczytu i zapisu przetwarzanych danych wewnętrz kontenera. Takie podejście wynika z tego, że wszystkie warstwy składające się na obraz kontenera są traktowane jako warstwy dostępne w trybie tylko do odczytu. Dzięki temu jeden obraz może być bazą uruchomieniową dla wielu kontenerów, z których każdy posiada swoją własną warstwę do zapisu i nie ingeruje w system plików innych kontenerów współdzielących ten sam obraz [104].

Po usunięciu kontenera dane przechowywane w tej warstwie zostaną również utracone. Dlatego do przechowywania istotnych danych została zaimplementowana funkcjonalność wolumenów danych które podczas uruchamiania mogą zostać podane jako parametr do polecenia *docker run*. W ten sposób wolumen zostaje podmontowany wewnątrz kontenera jako osobny system plików i dane modyfikowane na nim będą zachowane nawet po zakończeniu cyklu życia danego kontenera. Aby przybliżyć format składnię polecenia *docker run* omówię ją na przykładzie użycia w projekcie FindMyTutor.

```
1
2 docker run -itd -p 127.0.0.1:8090:80 \
3   -v fmtBackendDevelopLogs:/publish/Logs \
4   -v fmtBackendDevelopFeedback:/publish/Feedback \
5   --name=fmt-backend-develop \
6   --restart always \
7   mjedynski/fmt-backend-develop:2019-01-23-234c3a0
```

Rysunek 4.7 Polecenie uruchamiające kontener aplikacji FindMyTutor zbudowany z gałęzi develop.

Flagi zastosowane w powyższym poleceniu są odpowiedzialne za następujące parametry uruchomieniowe [105]:

- **-i** utrzymuje otwarty strumień standardowego wejścia.
- **-t** alokuje i przypisuje do kontenera pseudo terminal systemu Linux.
- **-d** tryb *detached* wymusza zakończenie procesu kontenera, jeśli proces, który się w nim wykonywał zostanie zakończony.
- **-p** opcja umożliwia mapowanie portów systemu hostującego Docker Engine na porty wewnątrz kontenera. Przyjmowane argumenty mają format:
`<adres_hosta>:<port_hosta>:<port_kontenera>`
Jeśli chcemy ustawić mapowanie kilku portów flaga **-p** musi zostać powtórzona w poleceniu *docker run* dla każdego mapowania osobno.
- **-v** flaga odpowiadająca za podmontowanie wewnątrz kontenera wolumenu Docker'a. Przyjmowane argumenty mają format:
`<nazwa_wolumenu>:<ścieżka_podmontowania_wew_kontenera>`
- **--name** przypisuje podaną nazwę do kontenera.
- **--restart** określa politykę zarządzania cyklem życia kontenera w wypadku jego zakończenia, dopuszcza się: *no*, *always*, *on-failure* oraz *unless-stopped*.

Ostatni argument polecenia wskazuje nazwę obrazu, który ma zostać wykorzystany.

Szybkość tworzenia i uruchamiania zbudowanych obrazów sprawiły, że kontenery są idealnym elementem rozwiązań wspierających ciągłe dostarczanie. W następnym rozdziale przybliżę jakimi wartościami określa się ta metodyka pracy oraz jakie narzędzia zostały skonfigurowane, aby zaistniała jako stały element procesów budowania aplikacji FindMyTutor.

4.5. Ciągła integracja oraz ciągłe dostarczanie

4.5.1. Definicja

Ciągła integracja (ang. *continuous Integration*) to metodyka pracy nad projektem stawiająca na celu zminimalizowanie kosztów oraz trudów jakie są ponoszone w trakcie dołączania nowych zmian do projektu. Dodatkowo wymaga od osób prowadzących projekt takiej organizacji pracy i wewnętrznych procesów deweloperskich, aby w każdym momencie pracy nad produktem zespół mógł dostarczyć jego wersję zdatną do opublikowania [106].

Ciągłe dostarczanie (ang. *continuous deployment*) jest procesem niejako wspomagającym wspomnianą wyżej ciągłą integrację [107]. U jego podstaw leży ukierunkowanie na możliwe jak największe skrócenie czasu od dodania zmian przez dewelopera do momentu, w którym użytkownicy systemu mogą skorzystać z nowej funkcjonalności. Postawione założenia są zazwyczaj realizowane za pomocą zautomatyzowanej infrastruktury skonfigurowanej w taki sposób by wykryła nowe zmiany w projekcie i przebudowała aplikację oraz wdrożyła nową wersję jednocześnie wycofując starą.

4.5.2. Narzędzie wspierające CI / CD – Buildbot

Aby wprowadzić do projektu inżynierskiego metodologię ciągłego dostarczania postanowiłem poszukać gotowego narzędzia spełniającego nasze wymagania. Po dłuższej analizie dostępnych rozwiązań oraz zarezerwowanego czasu przeznaczonego na wdrożenie CD wybór padł na narzędzie Buildbot. Jest to otwartoźródłowy serwer automatyzacji o architekturze rozproszonej napisany w języku Python. Dodatkowo istnieje możliwość zainstalowania dodatkowych modułów które pozwalają na zarządzanie procesami z poziomu przeglądarki internetowej. Jeśli zachodzi potrzeba dodania systemu powiadomień informującego o statusie wykonywanych automatyzacji to zapewniają to obiekty typu *notifiers* obsługujące różne kanały informacji na przykład takie jak http POST, Email, IRC albo status push dla repozytoriów kodu.

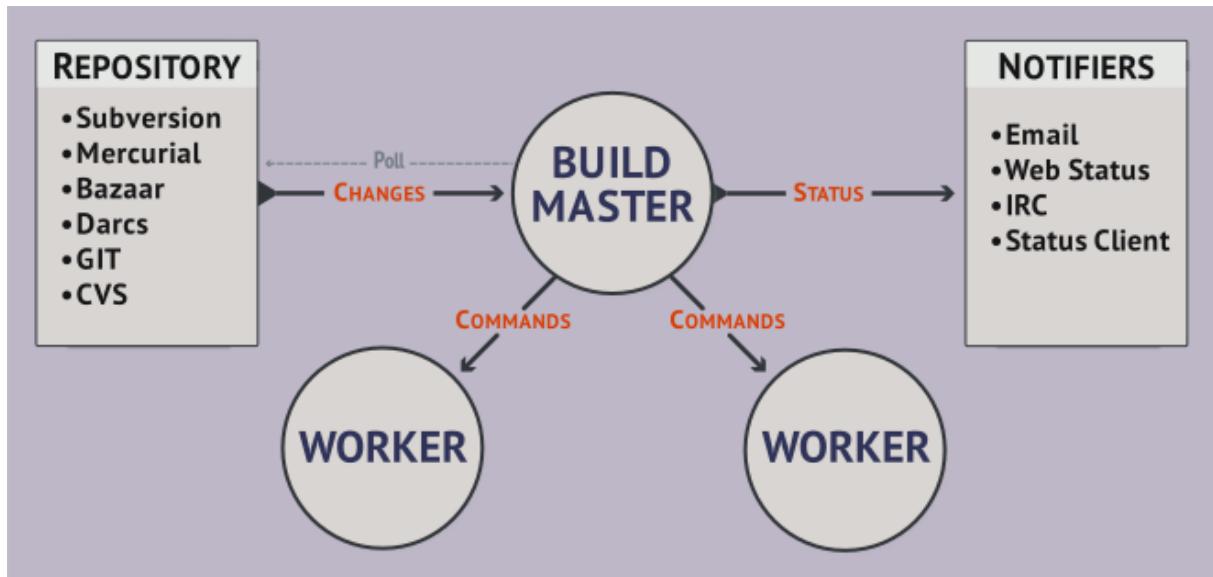
The screenshot shows the 'Find My Tutor CI' web interface. On the left is a dark sidebar with navigation links: Profiler, Home, Console View, Builds (selected), About, and Settings. The main area displays a 'Welcome to buildbot' message, a summary of '0 builds running currently', and '20 recent builds'. Two build status cards are shown: 'deployBackendDevelop' (with two failed builds) and 'deployWebMaster' (with three successful and one failed build). A 'ForceBuild' button is located in the top right corner.

Rysunek 4.8 interfejs webowy systemu Buildbot.

Niewątpliwą zaletą tego rozwiązania jest szybkość instalacji, odbywająca się za pomocą menadżera pakietów języka Python *pip*. Co więcej konfiguracja po pobraniu i zainstalowaniu odbywa się poprzez modyfikację pliku, który korzysta ze składni struktur danych i obiektów języka Python które w momencie uruchamiania są wczytywane do pamięci programu [108].

Do podstawowej funkcjonalności wymagana jest konfiguracja węzła *master* oraz co najmniej jednego węzła *worker*, które pełnią odpowiednio role dyspozytora zadań oraz wykonawcy. Dodatkowo będzie potrzebne skonfigurowanie źródła zmian tzw. repozytorium.

Poniżej znajduje się na opisywaną architektura w wariancie z dwoma węzłami *worker*.



Rysunek 4.9 Ogólny zarys architektury systemu Buildbot
Źródło: <https://buildbot.net/>

Żeby zautomatyzować dany proces budowania kontenerów aplikacji trzeba skonfigurować trzy łączące się ze sobą elementy które opiszę korzystając z przykładów.

Change source utrzymuje dostęp do repozytorium kodu projektu, nasłuchiwa zmian i w razie wykrycia różnic generuje obiekt z potrzebnymi informacjami które przekazuje do *Scheduler'a*.

```
34 c['change_source'].append(changes.GitPoller(  
35     repourl='git@git.wmi.amu.edu.pl:s416084/find-my-tutor-backend.git',  
36     project='fmt-backend',  
37     workdir='fmt-backend',  
38     pollAtLaunch=False,  
39     branches=['develop', 'master'],  
40     pollInterval=250))
```

Rysunek 4.10 Przykład konfiguracji *change source* w programie Buildbot.

W powyższym przykładzie widać interwał odpytywania repozytorium kodu aplikacji serwerowej ustawiony na 250 sekund oraz wyszczególnione gałęzie, które nas interesują. Taki mechanizm aktywnego odpytywania nie jest najlepszą praktyką, bardziej optymalnym rozwiązaniem byłaby reakcja *Change source* na powiadomienia http POST wysyłane przez repozytorium przy zdarzeniu wystąpienia nowych zmian. Niestety jednak ze względu na trudność konfiguracji takiej metody wdrożony został prostszy mechanizm.

Scheduler to obiekt odpowiedzialny za podejmowanie decyzji na podstawie zmian dostarczonych przez *Change source*. Akcje są podejmowane, jeśli zdefiniowany filtr znajdzie dopasowanie na zasadzie wyrażenia regularnego do któregoś z określonych atrybutów obiektu. Tym sposobem istniejące obiekty typu *Scheduler* nasłuchują zmian rozgłoszanych przez skonfigurowane *Change source* i reagują tylko te których filtr znalazł dopasowanie.

```
75 c['chedulers'].append(schedulers.SingleBranchScheduler(  
76     name="BackendDevelopSched",  
77     change_filter=filter_backend_develop,  
78     treeStableTimer=None,  
79     builderNames=["deployBackendDevelop"]))  
80
```

Rysunek 4.11 Przykład konfiguracji *scheduler* w programie Buildbot.

Dodatkowo możemy zauważać pole *builderNames* dostarczające listę zawierającą nazwy fabryk których procesy automatyzacji zostaną aktywowane w sytuacji dopasowania filtru do otrzymanych zmian.

Konfiguracja kroków która mają się wykonać w danym przypadku jest realizowana za pomocą funkcji *addStep()* wywoływanej na instancji obiektu *BuidFactory*. Powtarzając to wywołanie kilka razy zostaje utworzona uporządkowana sekwencja akcji składająca się na proces automatyzacji.

Najprostszym, ale dającym również największe możliwości jest krok typu *ShellCommand* który pozwala na wykonywanie dowolnych poleceń powłoki systemowej oraz wcześniej przygotowanych skryptów.

Aby podsumować omówioną wiedzę z zakresu kontenerów oraz automatyzacji w tym rozdziale poniżej możemy zobaczyć fragment kodu odpowiedzialny za budowanie nowego obrazu, wygaszanie starej wersji oraz uruchomienie nowej.

```
128 ##### BACKEND DEVELOP #####
129 factory_backend_develop = util.BuildFactory()
130 factory_backend_develop.addStep(steps.Git(
131     | repourl='git@git.wmi.amu.edu.pl:s416084/find-my-tutor-backend.git', name='Pull changes', mode='full', branch='develop'))
132 factory_backend_develop.addStep(steps SetPropertyFromCommand(
133     | name='Set timestamp', command='date -I', property='timestamp'
134 ))
135 factory_backend_develop.addStep(steps SetPropertyFromCommand(
136     | name='Set commit tag', command='git log -n 1 --pretty=format:%h', property='version'
137 ))
138 factory_backend_develop.addStep(steps SetPropertyFromCommand(
139     | name='Get commit message', command='git log -1 --pretty=%B', property='message'
140 ))
141 factory_backend_develop.addStep(
142     steps.ShellCommand(
143         name='Build new image',
144         command=['docker', 'build', 'findMyTutorBackend/','-t','mjedynski/fmt-backend-develop:latest',
145         '-t',util.Interpolate('mjedynski/fmt-backend-develop:{(prop:timestamp)s-(prop:version)s}')])
146 factory_backend_develop.addStep(
147     steps.ShellCommand(
148         name='Stop old container',
149         command=['docker', 'stop', 'fmt-backend-develop'])
150 )
151 factory_backend_develop.addStep(
152     steps.ShellCommand(
153         name='Remove old container',
154         command=['docker', 'rm', 'fmt-backend-develop'])
155 )
156 factory_backend_develop.addStep(
157     steps.ShellCommand(
158         name=util.interpolate('Run container {(prop:version)s}'),
159         command=['docker', 'run', '-i', '-p', '127.0.0.1:8090:80', '-v', 'fmtBackendDevelopLogs:/publish/Logs',
160         '-v', 'fmtBackendDevelopFeedback:/publish/Feedback', '--name=fmt-backend-develop', '--restart', 'always',
161         util.interpolate('mjedynski/fmt-backend-develop:{(prop:timestamp)s-(prop:version)s}')])
162 factory_backend_develop.addStep(
163     steps SetPropertyFromCommand(
164         name=util.interpolate('Set status property of {(prop:version)s}'),
165         property='status',
166         command="docker ps -l --format '{{.Status}} | awk '{print $1}'")
167 )
168 factory_backend_develop.addStep(
169     steps.ShellCommand(
170         name=util.interpolate('Status of {(prop:version)s} container is {(prop:status)s}'),
171         command=['curl', '-X', 'POST', '-H', 'Content-type: application/json', '--data',
172         util.interpolate('{ "text": "Backend develop container {(prop:version)s} is {(prop:status)s}!\nCommit message: {(prop:message)s}" }'),
173         'https://hooks.slack.com/services/T7X3LJZPB/BAK93RHA9/vsWzcDtceZARwCmQSDsGrVP'])
```

Rysunek 4.12 Przykład konfiguracji kroków budowy obrazu kontenera a następnie uruchomienia zbudowanej aplikacji.

4.6. Podsumowanie i wnioski

Obecne konteneryzacja zdają się idealnie wychodzić naprzeciw potrzebom deweloperów oraz biznesu. Dopóki branża wytwarzania oprogramowania będzie podążać za nurtem metodyk zwinnych oraz towarzyszących im filozofii ciągłej integracji i dostarczania można spodziewać się jeszcze większego wzrostu zainteresowania tą technologią.

W projekcie FindMyTutor rola jaka została odegrana przez kontenery oraz automatyzację dostarczania ma bardzo wymierny efekt. Czas opóźnienia pomiędzy wprowadzeniem zmian do repozytorium a upublicznieniem nowej wersji API został zredukowany z kilku godzin do niezawodnych kilku minut. Tym sposobem cały zespół zyskał na komforcie płynnej pracy a użytkownicy aplikacji mogli liczyć na szybkie aktualizacje zidentyfikowanych błędów. Niewątpliwie osiągnięcie takiego efektu bez przystępnej technologii kontenerów oraz gotowych narzędzi automatyzacji byłoby trudne.

5. Zakończenie

Celem niniejszej pracy było opisanie technologii oraz technik wykorzystanych przy tworzeniu aplikacji FindMyTutor. Projekt został wdrożony poprzez platformę Google Play (aplikacja mobilna), jak i na serwerach Wydziału Matematyki i Informatyki Uniwersytetu im. Adama Mickiewicza w Poznaniu (wersja webowa, serwer). Zaimplementowane zostały funkcjonalności, które były wcześniej konsultowane zarówno z profesorami jak i studentami – analiza objęła ponad 150 studentów oraz 30 profesorów. Dostosowanie produktu do ich wymagań było rzeczą kluczową w biznesowym rozwoju aplikacji. Dzięki oparciu aplikacji o architekturę REST możliwe było stworzenie interfejsu zarówno webowego, jak i mobilnego. Wdrożenie oparte zostało o kontenery Docker na systemie Linux.

Projekt ten może być z powodzeniem dalej utrzymywany oraz rozwijany przez Wydział Matematyki i Informatyki Uniwersytetu im. Adama Mickiewicza w Poznaniu. Jego wysoko zautomatyzowana architektura pozwala na proste wprowadzanie modyfikacji, bez potrzeby szczegółowej znajomości struktury projektu. Projekt stwarza także duże pole do dalszego rozwoju funkcjonalności, dlatego stanowi atrakcyjny produkt dla klienta, na rzecz którego został wykonany.

6. Spis rysunków

Rysunek 1.1 Przegląd wersji .NET Framework [2].....	14
Rysunek 1.2 Przegląd wersji .NET Core [4]	15
Rysunek 1.3 Tabela zapytań na sekundę	17
Rysunek 1.4 Model MVC [4]	18
Rysunek 1.5 Modele i kontrolery w aplikacji FindMyTutor	19
Rysunek 1.6 Przykładowa metoda kontrolera w aplikacji FindMyTutor	19
Rysunek 1.7 Przykładowa odpowiedź z aplikacji implementującej HATEOAS.....	21
Rysunek 1.8 Przykład metody spełniającej zasadę pojedynczej odpowiedzialności – pobranie zakładki profesora dla danego użytkownika.....	22
Rysunek 1.9 Przykład klasy spełniającej zasadę otwarte/zamknięte – klasa FeedbackService	23
Rysunek 1.10 Przykład klas łamiących zasadę podstawienia Liskov w .NET Core	24
Rysunek 1.11 Kod, który powinien zostać poprawnie wykonany dla proponowanej w fig.x struktury dziedziczenia	25
Rysunek 1.12 Przykład interfejsu działającego na plikach w .NET Core.....	25
Rysunek 1.13 Przykład interfejsów zgodnych z zasadą segregacji interfejsów w .NET Core ...	25
Rysunek 1.14 Atrybut walidacji LatitudeRangeAttribute w aplikacji FindMyTutor	26
Rysunek 1.15 Modele Coordinate oraz PredefinedCoordinate, wykorzystujące te same atrybuty walidacji	27
Rysunek 1.16 Fragment kodu metody ConfigureServices(), odpowiedzialny za konfigurację Swagger'a.....	28
Rysunek 1.17 Konfiguracja frameworku Swagger w projekcie FindMyTutor.....	29
Rysunek 1.18 Fragment kodu dodający oprogramowanie pośredniczące Swagger'a w projekcie FindMyTutor.....	29
Rysunek 1.19 Fragment dokumentacji automatycznej wygenerowanej przez framework Swagger w projekcie FindMyTutor.....	30
Rysunek 1.20 Przykład metody Validate() kontrolera LdapController wygenerowany przez framework Swagger.....	31
Rysunek 1.21 Dokumentacja metody Validate() w kodzie FindMyTutor	32
Rysunek 1.22 Zadania działające w tle w aplikacji FindMyTutor	32
Rysunek 1.23 BackgroundService – klasa bazowa dla wszystkich zadań działających w tle....	33
Rysunek 1.24 Nadpisana metoda ExecuteAsync() w klasie OnlineTagger.cs	34

Rysunek 1.25 Ustawienia dla metody LdapScraper w appsettings.json	35
Rysunek 1.26 Metoda ExecuteAsync klasy LdapScraper	35
Rysunek 1.27 Rejestracja zadań działających w tle w aplikacji FindMyTutor	36
Rysunek 2.1 Udział rynku klienckich systemów operacyjnych	38
Rysunek 2.2 Udział w rynku mobilnych systemów operacyjnych.....	38
Rysunek 2.3 Kwartał Q3 2018	39
Rysunek 2.4 Zestawienie różnic między mobilnymi aplikacjami natywnymi a wieloplatformowymi	45
Rysunek 2.5 Test.....	Błąd! Nie zdefiniowano zakładki.
Rysunek 2.6 Uogólniona budowa mostu rozwiązań wieloplatformowych	49
Rysunek 2.7 Architektura biblioteki Ionic Cordova	51
Rysunek 2.8 Architektura biblioteki Xamarin iOS i Android	52
Rysunek 2.9 Mono Runtime.....	53
Rysunek 2.10 Architektura ReactNative	54
Rysunek 2.11 Architektura NativeScript.....	55
Rysunek 2.12 Zestawienie bibliotek do zaciemniania kodu	58
Rysunek 2.13 Zestawienie cyklu życia aplikacji Android oraz iOS.....	59
Rysunek 2.14 Kompatybilność wsteczna bibliotek wieloplatformowych	62
Rysunek 3.1 Diagram klas wzorca obserwator	67
Rysunek 3.2 Diagram sekwencji wzorca obserwator	67
Rysunek 3.3 Diagram klas wzorca iterator	69
Rysunek 3.4 Diagram sekwencji wzorca iterator	69
Rysunek 3.5 Przykład użycia wbudowanego iteratora w języku Java	70
Rysunek 3.6 Opis poszczególnych typów obserwatorów w bibliotece RxJava	71
Rysunek 3.7 Opis poszczególnych metod umożliwiających tworzenie obserwatorów w bibliotece RxJava	72
Rysunek 3.8 Typy obiektów obserwowanych wraz z typami obserwatorów	73
Rysunek 3.9 Podstawowe metody obserwatorów dostępne w bibliotece RxJava.....	73
Rysunek 3.10 Obrazowe przedstawienie działania funkcji map	75
Rysunek 3.11 Obrazowe przedstawienie działania funkcji filter	75
Rysunek 3.12 Obrazowe przedstawienie działania funkcji reduce	76
Rysunek 3.13 Zależności pomiędzy komponentami w systemie reaktywnym.....	78

Rysunek 3.14 Cykl życia aktywności	80
Rysunek 3.15 Przykładowy schemat layoutu	82
Rysunek 3.16 Schemat widoku ekranu logowania w aplikacji FindMyTutor	85
Rysunek 3.17 Utworzenie referencji do przycisku logowania w aplikacji FindMyTutor	85
Rysunek 3.18 Przykład użycia biblioteki ButterKnife w projekcie FindMyTutor	86
Rysunek 3.19 Przykład dodania obserwatora do obiektu typu Disposable w aplikacji FindMyTutor.....	87
Rysunek 3.20 Przykład anulowania wszystkich subskrypcji danego obserwatora w metodzie onDestroy.....	87
Rysunek 3.21 Przykład definicji serwisu HTTP w aplikacji FindMyTutor	90
Rysunek 3.22 Metoda zwracająca listę identyfikatorów osób, które znajdują się w białej liście	90
Rysunek 3.23 Metoda zwracająca szczegółowe dane na temat danej użytkownika	91
Rysunek 3.24 Metoda odpowiedzialna za pobranie listy osób dodanych do białej listy	91
Rysunek 3.25 Przykład metody wywołanej po zakończeniu przetwarzania danych	92
Rysunek 3.26 Klasa odpowiedzialna za implementację obserwatora emitującego wpisane wartości w pole wyszukiwarki.....	93
Rysunek 3.27 Przykład kodu, który obrazuje przetwarzanie danychemitowanych przez obserwatora wyszukiwania.....	94
Rysunek 3.28 Obrazowe przedstawienie działania funkcji combineLatest.....	95
Rysunek 3.29 Metoda odpowiedzialna za walidację formularza w ekranie logowania aplikacji FindMyTutor.....	95
Rysunek 4.1 Diagram środowiska deweloperskiego w projekcie FindMyTutor	97
Rysunek 4.2 Wykres względnej ilości zapytań w serwisie google.com	104
Rysunek 4.3 Łańcuch wywołań inicjowany przez narzędzia docker-cli.....	105
Rysunek 4.4 Dockerfile odpowiedzialnego za budowę aplikacji serwerowej FindMyTutor. .	106
Rysunek 4.5 Proces budowy z pustym indeksem obrazów pośrednich.....	107
Rysunek 4.6 Proces budowy z wykorzystaniem obrazów pośrednich.	108
Rysunek 4.7 Polecenie uruchamiające kontener aplikacji FindMyTutor zbudowany z gałęzi develop.	109
Rysunek 4.8 interfejs webowy systemu Buildbot.	111
Rysunek 4.9 Ogólny zarys architektury systemu Buildbot	111
Rysunek 4.10 Przykład konfiguracji change source w programie Buildbot.....	112
Rysunek 4.11 Przykład konfiguracji scheduler w programie Buildbot.....	112

7. Bibliografia

- [1] T. Anderson, „The day Microsoft ‘embraced and extended’ Java,” 2007. [Online]. Available:
https://www.theregister.co.uk/Print/2007/12/07/microsoft_java_sun_infamy/.
- [2] Wikipedia, „.NET_Framework,” [Online]. Available:
https://en.wikipedia.org/wiki/.NET_Framework.
- [3] M. D. Icaza, „Microsoft Open Sources .NET and Mono,” 2014. [Online]. Available:
<https://tirania.org/blog/archive/2014/Nov-12.html>.
- [4] Wikipedia, „ASP.NET_Core,” [Online]. Available:
https://en.wikipedia.org/wiki/ASP.NET_Core.
- [5] B. Adams, „ASP.NET Core – 2300% More Requests Server Per Second,” 2016. [Online]. Available:
<https://www.ageofascent.com/2016/02/18/asp-net-core-exceeds-1-15-million-requests-12-6-gbps/>.
- [6] T. M. Reenskaug, „MVC: Model View Controller,” 1979. [Online]. Available:
<http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html>.
- [7] R. T. Fielding, „Representational State Transfer (REST),” 2000. [Online]. Available:
https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm.
- [8] J. M. R. A. P. C. B. N. Robert C. Martin, „Agile Software Development: Principles, Patterns, and Practices,” 2003. [Online].
- [9] Apple Inc., "Apple Reinvents the Phone with iPhone," 2007. [Online]. Available:
https://www.apple.com/uk/newsroom/2007/01/09Apple-Reinvents-the-Phone-with-iPhone/?utm_content=buffer9ce8e&utm_medium=social&utm_source=twitter.com&utm_campaign=buffer.

- [10] P. Adam, „The History Of The Smartphone,” 2016. [Online]. Available: <http://www.mobileindustryreview.com/2016/10/the-history-of-the-smartphone.html>.
- [11] Gartner, "Gartner Says Demand for Enterprise Mobile Apps Will Outstrip Available Development Capacity Five to One," 2015. [Online]. Available: <https://www.gartner.com/newsroom/id/3076817>.
- [12] W. Mark, „T-Mobile G1: Full Details of the HTC Dream Android Phone,” 2008. [Online]. Available: <https://gizmodo.com/t-mobile-g1-full-details-of-the-htc-dream-android-phone-5053264>.
- [13] Wikipedia, „Wersje systemu Android,” [Online]. Available: https://pl.wikipedia.org/wiki/Wersje_systemu_Android.
- [14] Opensignal, „Android Fragmentation,” 2015. [Online]. Available: <https://opensignal.com/reports/2015/08/android-fragmentation/>.
- [15] AppBrain, „Number of Android applications,” 2017. [Online]. Available: <https://web.archive.org/web/20170210051327/https://www.appbrain.com/stats/number-of-android-apps>.
- [16] L. Goode, „Apple's App Store just had the most successful month of sales ever,” 2017. [Online]. Available: <https://www.theverge.com/2017/1/5/14173328/apple-december-2016-app-store-record-phil-schiller>.
- [17] Android, „ART and Dalvik,” 2018. [Online]. Available: <https://source.android.com/devices/tech/dalvik>.
- [18] Apple Inc., „Objective-C Runtime,” 2018. [Online]. Available: https://developer.apple.com/documentation/objectivec/objective-c_runtime.
- [19] Ionic, „Hybrid vs. Native. An introduction to cross-platform hybrid development for architects and app development leaders,” Ionic eBook, 2018.
- [20] stackoverflow, „Developer Survey Results,” 2017.
- [21] V. Anudit, „iOS vs Android memory management,” 2018. [Online]. Available: <https://www.anudit.in/2018-01-13-android-vs-ios-memory-management/>.
- [22] Xamarin.iOS, „iOS Architecture,” 2018. [Online]. Available: <https://docs.microsoft.com/pl-pl/xamarin/ios/internals/architecture>.

- [23] Android, „Implementing ART Just-In-Time (JIT) Compiler,” 2018. [Online]. Available: <https://source.android.com/devices/tech/dalvik/jit-compiler>.
- [24] NativeScript, „JavaScript Android Runtime,” 2018. [Online]. Available: <https://docs.nativescript.org/core-concepts/android-runtime/overview>.
- [25] Apple Inc., „JavaScriptCore,” 2018. [Online]. Available: <https://developer.apple.com/documentation/javascriptcore>.
- [26] V. TJ, „How NativeScript Works,” 2015. [Online]. Available: <https://developer.telerik.com/featured/nativescript-works/>.
- [27] Google Developers, „Progressive Web Apps,” 2018. [Online]. Available: <https://developers.google.com/web/progressive-web-apps/>.
- [28] Google Developers, „Payments,” 2019. [Online]. Available: <https://developers.google.com/web/fundamentals/payments/>.
- [29] Ionic, „Dokumentacja Ionic,” 2018. [Online]. Available: <https://ionicframework.com/docs/>.
- [30] Ionic, „The Architect's Guide to PWA,” 2018. [Online]. Available: https://cdn2.hubspot.net/hubfs/3776657/PWA_WP_v6.pdf.
- [31] Mono Project, „The Mono Runtime,” 2018. [Online]. Available: <https://www.monoproject.com/docs/advanced/runtime/>.
- [32] Xamarin, „Android apps architecture,” 2018. [Online]. Available: <https://docs.microsoft.com/pl-pl/xamarin/android/internals/architecture>.
- [33] Xamarin, „iOS app architecture,” 2018. [Online]. Available: <https://docs.microsoft.com/pl-pl/xamarin/ios/internals/architecture>.
- [34] Android, „Android NDK Documentation,” 2018. [Online]. Available: <https://developer.android.com/ndk/>.
- [35] V. Stoychev, „How secure is NativeScript,” 2016. [Online]. Available: <https://www.nativescript.org/blog/how-secure-is-nativescript>.
- [36] NativeScript, „Configuring OpenID Connect Authentication,” 2018. [Online]. Available: <https://docs.nativescript.org/sidekick/user-guide/enterprise-auth/openid>.

- [37] J. Sewell, „Dotfuscator Community Edition (CE),” 2017. [Online]. Available: <https://docs.microsoft.com/en-gb/visualstudio/ide/dotfuscator/?view=vs-2017>.
- [38] Jscrambler, „Jscrambler ensures the integrity of your Web Application on the Client-Side,” [Online]. Available: <https://jscrambler.com/>.
- [39] M. Bazon, „UglifyJS – a JavaScript parser/compressor/beautifier,” 2014. [Online]. Available: <https://github.com/mishoo/UglifyJS>.
- [40] B. Rocha, „Swift Obfuscator that protects iOS apps against reverse engineering attacks.,” 2018. [Online]. Available: <https://github.com/rockbruno/swiftshield>.
- [41] Guardsquare, „Cutting-edge protection for your iOS apps and SDKs,” [Online]. Available: <https://www.guardsquare.com/en/products/ixguard>.
- [42] T. VanToll, „Protecting Your Source Code with Jscrambler,” 2018. [Online]. Available: <https://www.nativescript.org/blog/protecting-your-source-code-with-jscrambler>.
- [43] Android, „Android keystore system,” 2018. [Online]. Available: <https://developer.android.com/training/articles/keystore>.
- [44] Apple Inc., „Keychain Services,” [Online]. Available: https://developer.apple.com/documentation/security/keychain_services.
- [45] OWASP, „Projects/OWASP Mobile Security Project - Security Testing,” 2013. [Online]. Available: https://www.owasp.org/index.php/Projects/OWASP_Mobile_Security_Project_-_Security_Testing.
- [46] Abdellah, „What the equivalet of activity life cycle in ios,” 2015. [Online]. Available: <https://stackoverflow.com/questions/28969032/what-the-equivalent-of-activity-life-cycle-in-ios>.
- [47] Android, „Understand the Activity Lifecycle,” 2019. [Online]. Available: <https://developer.android.com/guide/components/activities/activity-lifecycle>.
- [48] Android, „Background Execution Limits,” 2018. [Online]. Available: <https://developer.android.com/about/versions/oreo/background>.
- [49] Apple Inc., „Using the Standard Location Service,” 2018. [Online]. Available: https://developer.apple.com/documentation/corelocation/getting_the_user_s_location/using_the_standard_location_service?language=objc.

- [50] Apple Inc., „Handling Location Events in the Background,” 2018. [Online]. Available: https://developer.apple.com/documentation/corelocation/getting_the_user_s_location/handling_location_events_in_the_background?language=objc.
- [51] Apple Inc., „Using the Significant-Change Location Service,” 2018. [Online]. Available: https://developer.apple.com/documentation/corelocation/getting_the_user_s_location/using_the_significant-change_location_service.
- [52] Android, „FusedLocationProviderClient,” 2018. [Online]. Available: <https://developers.google.com/android/reference/com/google/android/gms/location/FusedLocationProviderClient>.
- [53] H. Kevin, „Using a single codebase for your cloud-native app,” 2016. [Online]. Available: <https://www.oreilly.com/ideas/using-a-single-codebase-for-your-cloud-native-app>.
- [54] I. Goździeniak, „MVP (ang. Minimal Viable Product) to sposób działania,” 2017. [Online]. Available: <http://www.agile247.pl/mvp/>.
- [55] E. F. K. S. B. B. E. Freeman, „Wzorce projektowe rusz głową,” 2017. [Online].
- [56] T. Tomczykiewicz, „Wzorzec projektowy obserwator,” 2017. [Online]. Available: <https://tomasz-tomczykiewicz.blog/2017/03/28/wzorzec-projektowy-obserwator>.
- [57] O. Blancarte, „BEHAVIORAL PATTERN,” 2016. [Online]. Available: Źródło: <https://reactiveprogramming.io/books/design-patterns/en/catalog/observer> .
- [58] Oracle, „Interface Iterator documentation,” [Online]. Available: <https://docs.oracle.com/javase/8/docs/api/java/util/Iterator.html>.
- [59] reactiveprogramming.io, „Design Patterns,” [Online]. Available: <https://reactiveprogramming.io/books/design-patterns/en>.
- [60] O. Blancarte, „Iterator behavioral pattern,” 2019. [Online]. Available: <https://reactiveprogramming.io/books/design-patterns/en/catalog/iterator>.
- [61] R. POSA, „Java Iterator,” [Online]. Available: <https://www.journaldev.com/13460/java-iterator>.
- [62] M. Nowak, „Iterator - Wzorce projektowe,” [Online]. Available: <http://androidcode.pl/blog/wzorce/iterator/>.

- [63] R. Contributors., „ReactiveX Intro,” [Online]. Available: <http://reactivex.io/intro.html>.
- [64] Vogella, „Using RxJava 2 - Tutorial,” [Online]. Available: <http://www.vogella.com/tutorials/RxJava/article.html>.
- [65] R. Contributors, „Creating observables,” [Online]. Available: <https://github.com/ReactiveX/RxJava/blob/2.x/docs/Creating-Observables.md>.
- [66] R. Tamada, „RxJava Understanding Observables,” [Online]. Available: <https://www.androidhive.info/RxJava/rxjava-understanding-observables/>.
- [67] R. TAMADA, „RxJava Understanding Observables,” [Online]. Available: <https://www.androidhive.info/RxJava/rxjava-understanding-observables/>.
- [68] S. Thompson, „Haskell the craft of functional programming 3rd edition,” Pearson Education, 2011. [Online].
- [69] A. Alexande, „The Benefits of Pure Functions,” [Online]. Available: <https://alvinalexander.com/scala/fp-book/benefits-of-pure-functions>.
- [70] A. Zawłock, „Monady,” [Online]. Available: <https://www.mimuw.edu.pl/~zawlocki/haskell/Monady.html>.
- [71] A. Stoltz, „RxJS Marbles,” 2017. [Online]. Available: <https://rxmarbles.com/>.
- [72] M. Foundation, „JavaScript Documentation,” [Online]. Available: <https://developer.mozilla.org/pl/docs/Web/JavaScript/Referencje/Obiekty/Array/map>.
- [73] D. F. R. K. a. M. T. J. Bonér, „The Reactive Manifesto,” [Online]. Available: <https://www.reactivemanifesto.org/>.
- [74] reactivemanifesto, „The Reactive Manifesto,” 2014. [Online]. Available: <https://www.reactivemanifesto.org/>.
- [75] A. O. S. Project, „Application Fundamentals,” 2018. [Online]. Available: <https://developer.android.com/guide/components/fundamentals>.
- [76] A. O. S. Project, „Fragment documentation,” [Online]. Available: <https://developer.android.com/reference/android/app/Fragment>.

- [77] A. O. S. Project, „Activity documentation,” [Online]. Available: <https://developer.android.com/reference/android/app/Activity#activity-lifecycle>.
- [78] A. O. S. Project, „Understand the Activity Lifecycle,” 2018. [Online]. Available: <https://developer.android.com/guide/components/activities/activity-lifecycle>.
- [79] Android Inc., „Understand the Activity Lifecycle,” 2019. [Online]. Available: <https://developer.android.com/guide/components/activities/activity-lifecycle>.
- [80] M. Stanek, „Activity – podstawowe informacje, cykl życia,” [Online]. Available: <http://www.android4devs.pl/2011/07/activity-podstawowe-informacje-cykl-zycia/>.
- [81] R. Contributors, „Subject,” [Online]. Available: <http://reactivex.io/documentation/subject.html>.
- [82] J. Wharton., „RxBinding,” [Online]. Available: <https://github.com/JakeWharton/RxBinding>.
- [83] R. Contributors., „Combinelatest operators,” [Online]. Available: <http://reactivex.io/documentation/operators/combinelatest.html>.
- [84] ReactiveX, „CombineLatest,” [Online]. Available: <http://reactivex.io/documentation/operators/combinelatest.html>.
- [85] Codeship, „Speeding Up Development at Lyft,” 2018. [Online]. Available: <https://blog.codeship.com/dockercon-2015-speeding-up-development-at-lyft/>.
- [86] E. McAfee, „We switched to Amazon ECS and you won’t believe what happened next,” 2017. [Online]. Available: <https://blog.mapbox.com/we-switched-to-amazon-ecs-and-you-wont-believe-what-happened-next-6cadbbf7282c>.
- [87] C. Joe, „How containers cut server costs at the Financial Times by 80 percent,” [Online]. Available: <https://www.computerworlduk.com/infrastructure/how-containers-cut-server-costs-for-financial-times-by-80-percent-3676757/>.
- [88] E. Carter, „2018 Docker Usage Report,” 2018. [Online]. Available: <https://sysdig.com/blog/2018-docker-usage-report/>.
- [89] Crunchbase Inc., „Docker Overview,” 2019. [Online]. Available: <https://www.crunchbase.com/organization/docker>.

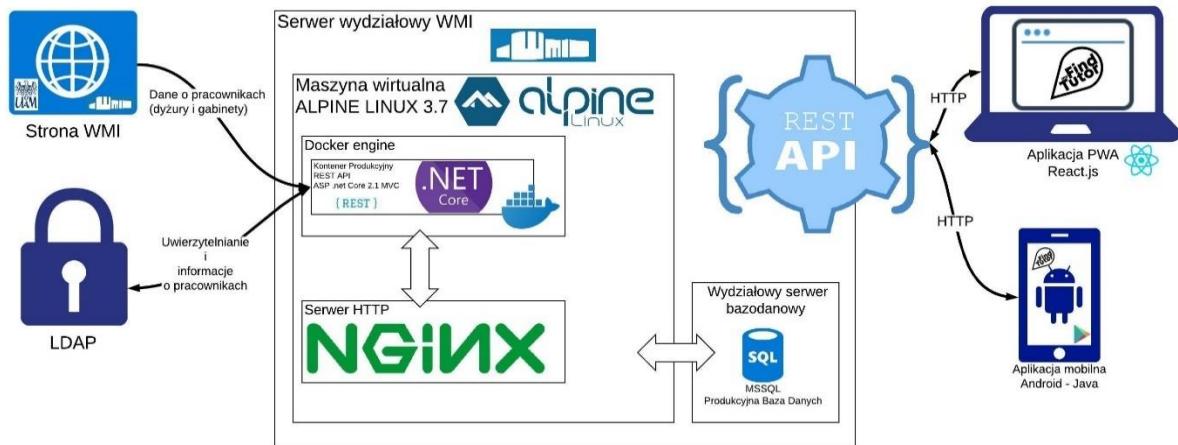
- [90] Red Hat Inc., „Red Hat to Acquire CoreOS, Expanding its Kubernetes and Containers Leadership,” 2018. [Online]. Available: <https://www.redhat.com/en/about/press-releases/red-hat-acquire-coreos-expanding-its-kubernetes-and-containers-leadership>.
- [91] Pivotal, „Moments in Container History,” [Online]. Available: <https://content.pivotal.io/infographics/moments-in-container-history>.
- [92] Wikipedia, „FreeBSD jail,” [Online]. Available: https://en.wikipedia.org/wiki/FreeBSD_jail.
- [93] M. Kerrisk, „Namespaces in operation, part 1: namespaces overview,” 2013. [Online]. Available: <https://lwn.net/Articles/531114/>.
- [94] J. Topjian, „Contain your enthusiasm – Part Two: Jails, Zones, OpenVZ, and LXC,” 2013. [Online]. Available: <https://www.cybera.ca/news-and-events/tech-radar/contain-your-enthusiasm-part-two-jails-zones-openvz-and-lxc/>.
- [95] Wikipedia, „cgroups,” [Online]. Available: <https://en.wikipedia.org/wiki/Cgroups>.
- [96] J. Lonan, „LXC: A Quick Overwiev,” 2016. [Online]. Available: <http://blog.hde.co.jp/entry/2016/05/23/120114>.
- [97] Docker Inc., „Best practices for writing Dockerfiles,” [Online]. Available: https://docs.docker.com/develop/develop-images/dockerfile_best-practices/.
- [98] CoreOS, „rkt architecture,” [Online]. Available: <https://coreos.com/rkt/docs/latest-devel/architecture.html>.
- [99] CoreOS, „Using rkt with systemd,” [Online]. Available: <https://coreos.com/rkt/docs/latest/using-rkt-with-systemd.html>.
- [100] The Linux Foundation, „About,” [Online]. Available: <https://www.opencontainers.org/about>.
- [101] M. Crosby, „WHAT IS CONTAINERD ?,” 2017. [Online]. Available: <https://blog.docker.com/2017/08/what-is-containedr-runtime/>.
- [102] I. Lewis, „Container Runtimes Part 3: High-Level Runtimes,” 2018. [Online]. Available: <https://www.ianlewis.org/en/container-runtimes-part-3-high-level-runtimes>.

- [103] Docker Inc., „Dockerfile reference,” [Online]. Available: <https://docs.docker.com/engine/reference/builder/>.
- [104] Docker Inc., „About storage drivers,” [Online]. Available: <https://docs.docker.com/storage/storagedriver/>.
- [105] Docker Inc., „Docker run reference,” [Online]. Available: <https://docs.docker.com/engine/reference/run/>.
- [106] Agile Alliance, „Continuous Integration,” [Online]. Available: <https://www.agilealliance.org/glossary/continuous-integration/>.
- [107] Agile Alliance, „Continuous Deployment,” [Online]. Available: <https://www.agilealliance.org/glossary/continuous-deployment/>.
- [108] Buildbot, „2.5.1. Configuring Buildbot,” [Online]. Available: <http://docs.buildbot.net/current/manual/configuration/intro.html>.
- [109] Microsoft, „Microsoft Announces Visual Studio .NET 2003 Worldwide Availability,” 2003. [Online]. Available: <https://news.microsoft.com/2003/04/23/microsoft-announces-visual-studio-net-2003-worldwide-availability/>.
- [110] S. Toub, „Performance Improvements in .NET Core,” 2017. [Online]. Available: <https://blogs.msdn.microsoft.com/dotnet/2017/06/07/performance-improvements-in-net-core/>.
- [111] Steve Smith, „Overview of ASP.NET Core MVC,” 2018. [Online]. Available: <https://docs.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-2.2>.
- [112] M. Stasch, „HATEOAS. Czy Twoje API może być lepsze?,” 2018. [Online]. Available: <https://geek.justjoin.it/hateoas-api-moze-byc-lepsze/>.
- [113] T. Dalling, „SOLID Class Design: The Liskov Substitution Principle,” 2009. [Online]. Available: <https://www.tomdalling.com/blog/software-design/solid-class-design-the-liskov-substitution-principle/>.
- [114] F. Despoudis, „Understanding SOLID Principles: Interface Segregation Principle,” 2017. [Online]. Available: <https://codeburst.io/understanding-solid-principles-interface-segregation-principle-b2d57026cf6c>.
- [115] „Swagger, dokumentacja,” 2018. [Online]. Available: <https://swagger.io/docs/specification/2-0/what-is-swagger/>.

- [116] L. L. Scott Addie, „Dependency injection in ASP.NET Core,” 2018. [Online]. Available: <https://docs.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection?view=aspnetcore-2.2>.
- [117] L. Latham, „Background tasks with hosted services in ASP.NET Core,” 2018. [Online]. Available: <https://docs.microsoft.com/en-us/aspnet/core/fundamentals/host/hosted-services?view=aspnetcore-2.2>.
- [118] Wikipedia, „ASP.NET_Core,” [Online]. Available: https://en.wikipedia.org/wiki/ASP.NET_Core.

8. Dokumentacja

8.1. Architektura systemu



Rysunek 85 Architektura systemu FMT

Na rysunku została przedstawiona architektura aplikacji FindMyTutor.

Warstwa kliencka została zrealizowana w dwóch technologiach:

- Natywna aplikacja mobilna dla systemu Android - Java.
- Aplikacja webowa - JavaScript, biblioteka React.js

Komunikacja aplikacji z warstwą serwerową REST API odbywa się za pomocą protokołu HTTP. Serwer został stworzony w technologii ASP.NET Core 2.1 MVC, z wykorzystaniem relacyjnej bazy danych MSSQL. Instancja REST API uruchamiana jest w wirtualnym kontenerze Docker. Uwierzytelnianie użytkownika odbywa się za pomocą wydziałowego serwera LDAP.

Cały projekt został wdrożony na wirtualnej maszynie Alpine Linux 3.7 na fizycznej maszynie na Wydziale Matematyki i Informatyki Uniwersytetu im. Adama Mickiewicza w Poznaniu, posiada system ciągłego dostarczania oraz system notyfikacji budowy kontenerów deweloperskich i produkcyjnych zintegrowany z komunikatorem Slack. Za kierowanie ruchem HTTP jest odpowiedzialny serwer nginx.

8.2. Dokumentacja REST API

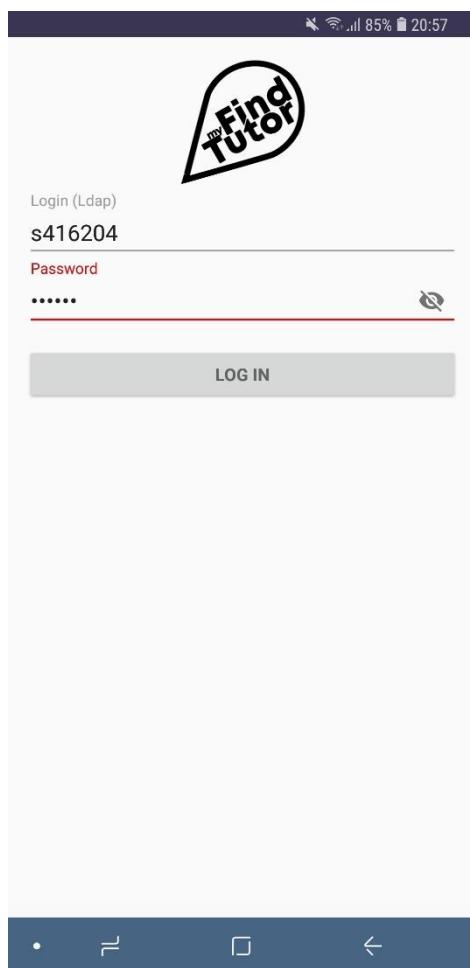
Blacklist		
GET	/api/users/blacklist/{tutorId}	🔒
PUT	/api/users/blacklist/{tutorId} Turn on/off usage of blacklist for tutor	🔒
POST	/api/users/blacklist/{tutorId} Add student to tutor's blacklist	🔒
DELETE	/api/users/blacklist/{tutorId}	🔒
GET	/api/users/whitelist/{tutorId}	🔒
PUT	/api/users/whitelist/{tutorId} Turn on/off usage of whitelist for tutor	🔒
POST	/api/users/whitelist/{tutorId} Add student to tutor's whitelist	🔒
DELETE	/api/users/whitelist/{tutorId}	🔒
GET	/api/users/blacklist/getBlacklisters/{studentId} Get a list of tutors who blacklist you	🔒
Coordinates		
GET	/api/coordinates/user/{userId}	🔒
GET	/api/coordinates/userTop/{userId}	🔒
GET	/api/coordinates/top Get all top coordinates for all users	🔒
GET	/api/coordinates/top/online Get all top coordinates for all online users	🔒
PUT	/api/coordinates/{userId} Edit specific coordinate	🔒
Feedback		
GET	/api/Feedback Get feedback	🔒
POST	/api/Feedback Post feedback	🔒
POST	/api/Feedback/autoFeedback Post auto feedback	🔒
Ldap		
POST	/api/ldap/fakeValidate Login / register user with this endpoint - for development	🔒
POST	/api/ldap/validate Login / register user with this endpoint	🔒
GET	/api/ldap/getUser/{login} Get user by login	🔒
GET	/api/ldap/getUser/email/{email} Get user by email	🔒
POST	/api/ldap/ldapScraper Initialize ldapScraper	🔒
POST	/api/ldap/logout Logout user with this endpoint	🔒
PredefinedCoordinates		
GET	/api/users/predefined/coordinate/{tutorId}	🔒
POST	/api/users/predefined/coordinate/{tutorId}	🔒
DELETE	/api/users/predefined/coordinate/{tutorId}	🔒

TutorTab		
GET	/api/users/tutorTab/{tutorId}	🔒
PUT	/api/users/tutorTab/{tutorId}	🔒
POST	/api/users/tutorTab/{tutorId} Add tutor tab	🔒
DELETE	/api/users/tutorTab/{tutorId}	🔒
POST	/api/users/scrapTutorTab Scrap all tutor tabs	🔒
POST	/api/users/scrapTutorTab/{tutorId} Scrap single tutor tab	🔒
Users		
GET	/api/users	🔒
POST	/api/users Add user to DB	🔒
GET	/api/users/tutors/search Search among tutors for given name and surname. Works basing on "title" field in user model (not actual roles), which should be exact as role, but may vary.	🔒
GET	/api/users/tutors/active Get active tutors. Works basing on "title" field in user model (not actual roles), which should be exact as role, but may vary.	🔒
GET	/api/users/tutors/online Get online tutors. Works basing on "title" field in user model (not actual roles), which should be exact as role, but may vary.	🔒
GET	/api/users/tutors/offline Get offline tutors. Currently returns offline + inactive tutors. Works basing on "title" field in user model (not actual roles), which should be exact as role, but may vary.	🔒
GET	/api/users/tutors	🔒
GET	/api/users/page/{pageNum} Get a page of 10 users - all users	🔒
GET	/api/users/tutors/page/{pageNum} Get a page of 10 users - tutors	🔒
GET	/api/users/students/page/{pageNum} Get a page of 10 users - students	🔒
GET	/api/users/{id}	🔒
PUT	/api/users/{id} Edit user	🔒
DELETE	/api/users/{id}	🔒
GET	/api/users/self/{id} Get all data about user	🔒
GET	/api/users/userLogin/{ldapLogin}	🔒
PUT	/api/users/role/makeTutor/{ldapLogin} Make user a tutor	🔒
PUT	/api/users/role/makeStudent/{ldapLogin} Make user a student	🔒

8.3. Instrukcja obsługi

Ekran logowania

Po uruchomieniu użytkownik zostanie poproszony o zalogowanie się do aplikacji za pomocą danych używanych w wydziałowym systemie LDAP. W zależności od jego roli w tym systemie zostaną udostępnione mu takie same uprawnienia w aplikacji FindMyTutor.



Dodatkowo po zalogowaniu się do aplikacji użytkownik zostanie poinformowany o potrzebie dodania aplikacji FindMyTutor do wyjątków zarządzania baterią.



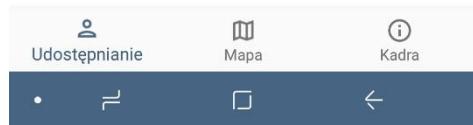
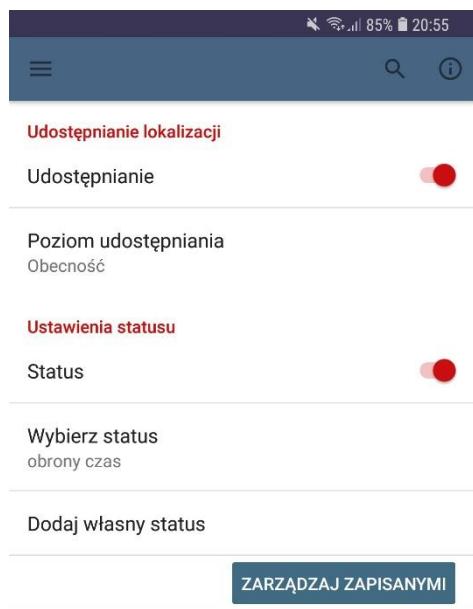
Główny ekran aplikacji

Po zalogowaniu się użytkownikowi ukaże się mapa budynku, na której znajdować się będą markery informujące o pozycji poszczególnego nauczyciela akademickiego w obrębie budynku.



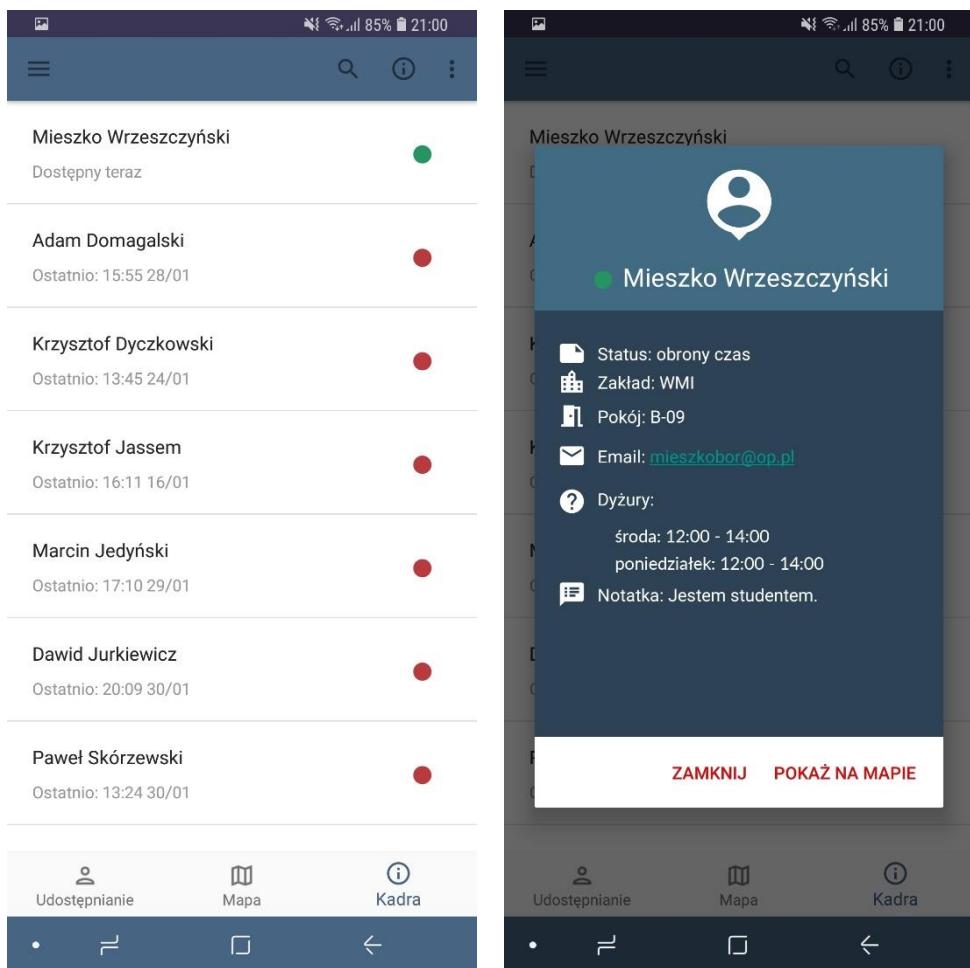
Ustawienia udostępniania

W tym ekranie użytkownik aplikacji może ustalić poziom udostępniania swojej lokalizacji. Dodatkowo możliwe jest także zdefiniowanie własnych predefiniowanych lokalizacji i statusu, który dostępny będzie dla innych użytkowników.



Listy użytkowników

Ekran ten odpowiedzialny jest za wyświetlanie użytkowników aplikacji. Oprócz samych danych o danym użytkowniku możliwe jest także ustalenie, kiedy ostatnio użytkownik był dostępny w aplikacji.



Po na naciśnięciu na danego użytkownika widoczny stanie się dodatkowy ekran, który zawiera bardziej szczegółowe dane takie jak: godziny dyżurów czy pokój.