Politechnika Wrocławska

Wydział Informatyki i Zarządzania
kierunek studiów: informatyka

Praca dyplomowa - inżynierska

APLIKACJA WSPOMAGAJĄCA ROZLICZANIE DŁUGÓW

Tomasz Czubocha

słowa kluczowe:
React Native
Firebase
serverless

krótkie streszczenie:
Praca ma na celu rozwiązanie problemu zapamiętywania długów pieniężnych wobec innych osób i niezwróconych należności dla użytkownika przez zaprojektowanie i implementację wieloplatformowej aplikacji mobilnej działającej w architekturze serverless oraz wykorzystującej powiadomienia push. Omówiono m.in. istniejące rozwiązania, wymagania i założenia projektowe, a także problemy implementacyjne.

opiekun pracy dyplomowej	.....	.....	.....
	Tytuł/stopień naukowy/imię i nazwisko	*ocena*	*podpis*

Ostateczna ocena za pracę dyplomową			
Przewodniczący Komisji egzaminu dyplomowego	.....	.....	.....
	Tytuł/stopień naukowy/imię i nazwisko	*ocena*	*podpis*

*Do celów archiwalnych pracę dyplomową zakwalifikowano do:**

a) kategorii A (akta wieczyste)

b) kategorii BE 50 (po 50 latach podlegające ekspertyzie)

* *niepotrzebne skreślić*

pieczątka wydziałowa

Wrocław 2017

Rodzicom - za to, że praca ta mogła powstać.

Streszczenie

Praca ma na celu rozwiązanie problemu zapamiętywania długów pieniężnych wobec innych osób i niezwróconych należności dla użytkownika przez zaprojektowanie i implementację wieloplatformowej aplikacji mobilnej działającej w architekturze serverless oraz wykorzystującej powiadomienia push.

Aplikacja kierowana jest dla osób rozliczających się w grupie ludzi, w której zakupy nie są robione przez jedną osobę. Główną funkcją aplikacji jest dzielenie kwoty zakupów na wiele osób z grona tzw. znajomych i dodawanie długu dla każdej z nich z osobna, by stworzyć salda między użytkownikiem i jego dłużnikami. Celem pobocznym pracy jest stworzenie rozwiązania jak najmniejszym kosztem tj. przez unikanie pisania tzw. kodu boilerplate - skupieniu się na samej logice i wyglądzie aplikacji oraz minimalizację zarządzania powstałym produktem. Do wytworzenia oprogramowania wykorzystano platformę React Native pozwalającą na użycie jednego kodu uruchamianego na wielu systemach operacyjnych i Firebase udostępniającą usługi Back-end as a Service.

We wstępie wprowadzono czytelnika w problematykę, szczegółowo opisano rozwiązywany problem i zarysowano zakres pracy. Następnie sformułowano wymagania projektowe i omówiono istniejące już rozwiązania. Dalej określono założenia projektowe - przedmiot prac, wymagania funkcjonalne i niefunkcjonalne oraz architekturę systemu ze sposobem realizacji. Kolejno stworzono projekt aplikacji z przypadkami użycia, konceptem interfejsu i strukturą bazy danych oraz aplikacji. Ostatecznie opisano fazę implementacyjną łącznie z zaistniałymi problemami, by w ostatnim rozdziale podsumować całą pracę.

Abstract

Thesis aims to solve the problem of memorizing money debts to other people and charges not returned to the user by designing and implementing a cross platform mobile application running on a serverless architecture and using push notifications.

The application is addressed to people paying in a group of people in which purchases are not made by one person. The main function of the application is to divide the amount of purchases into many people from the group of friends and add the debt for each of them separately to create balances between the user and his debtors. The side aim is to create the solution at the least cost by avoiding writing the boilerplate code - focusing on the logic and look of the application and minimizing the management of the resulting product. The software was developed using the React Native platform, allowing one code to run on multiple operating systems and Firebase providing Back-end as a Service.

In the introduction, the reader is introduced to the problem, described in detail and outlined the scope of work. Next, the design requirements have been formulated and the existing solutions are discussed. Then, design assumptions are defined - subject of work, functional and non-functional requirements of the system, architecture and the way of implementation. Subsequently, the application is designed with use cases, interface concepts, database and application structure. Finally, the implementation phase is described together with the problems to in the last chapter, summarize all the work.

Spis treści

Spis rysunków

Rozdział 1

Wstęp

1.1 Wprowadzenie do problematyki

Niniejsza praca opisuje projekt oraz implementację aplikacji mobilnej wspomagającej rozliczanie długów. Celem aplikacji jest rozwiązanie problemu częstego oddawania niewielkich kwot pieniędzy oraz pamiętania o długach do spłacenia pojawiającego się w grupie ludzi, którzy często wymieniają między sobą dobra i kupującym nie jest zawsze jedna osoba. Jej główną funkcją jest dodawanie swoich wydatków na rzecz innych osób - znajomych użytkownika oraz dzielenie tych wydatków na więcej niż jedną osobę, np. w przypadku płacenia za lunch w restauracji. Oprócz rozwiązania wyżej wspomnianego problemu, celem pobocznym jest wytworzenie aplikacji w taki sposób, by była ona dostępna dla jak największej liczby osób i zapewniała jak najwięcej funkcji (także tych, wydawałoby się, bardziej zaawansowanych i trudnych do zaimplementowania) przy jak możliwie najmniejszym koszcie rozumianym przez ilość własnego kodu wyprodukowanego przez programistę. By go zrealizować wykorzystano najnowsze technologie, architekturę oraz usługi, które to umożliwiają. Autor pracy wychodzi z założenia, że aplikacja spełni swoją rolę i potrzeby potencjalnego użytkownika tylko wtedy, gdy będzie dostępna dla jej posiadacza i jego znajomych. Przyjmując, że tenże użytkownik i jego dłużnicy są posiadaczami smartfona dowolnej marki, aplikacja musiała zostać napisana tak, by można było ją uruchomić na dwóch najbardziej popularnych mobilnych systemach operacyjnych - Android oraz iOS - jednak pamiętając o nadmienionym celu dotyczącym minimalizacji kosztu wytworzenia. Złym rozwiązaniem byłoby zatem napisanie tej samej, natywnej (standardowej dla danej platformy) aplikacji w wersji dla jednego i drugiego systemu ze względu na podwojony nakład pracy. Technologią idealnie wpasowującą się w te wymagania jest platforma programistyczna (framework), której przyświeca idea „learn once, write anywhere”, czyli React Native. Korzystając z niej napisano aplikację samą w sobie, czyli klienta, którego wykorzystuje użytkownik do komunikacji z bazą danych. Dzięki React

Native można napisać jeden kod w języku programowania JavaScript, który wywołuje interfejs programistyczny Androida i iOS tworząc całkowicie natywne aplikacje, zachowujące standardy designu poszczególnych systemów. Do strony back-end'owej (typowo serwera przyjmującego zapytania klienta, przetwarzającego dane, zapewniającego logikę aplikacji oraz komunikację z bazą danych) wykorzystano środowisko do budowy aplikacji Firebase. Jest to platforma zapewniająca usługi w modelu BaaS, czyli Backend as a Service. Przez jej wykorzystanie całe przedstawiane rozwiązanie uzyskuje architekturę serverless, czyli w znaczącej części oparte jest na zewnętrznych rozwiązaniach chmurowych. Logika po stronie serwera pisana jest przez programistę, jednak w odróżnieniu do tradycyjnych architektur, uruchamiana jest w bezstanowych kontenerach i wywoływana przez określone zdarzenia.

1.2 Opis problemu

W życiu nie da obejść się bez pieniędzy. Codziennie mamy z nimi styczność, płacimy. Po prostu są one niezbędne. Warto więc zadbać o to, by kontakt z nimi był jak najbardziej przyjazny i wygodny. Często oznacza to jak najmniejsze użycie ich fizycznej formy - banknotów i monet. Odejście od takiego podejścia widać najlepiej przez rosnącą popularność „wirtualnych" pieniędzy - kont w bankach, płatności kartami debetowymi oraz kredytowymi. Rzadko który sklep nie obsługuje już płatności kartą. Każdy posiada swoje zasoby pieniężne, jednak życie pokazuje, że często korzystamy z czyichś lub sami tych zasobów użyczamy, po prostu pożyczając lub kupując coś drugiej osobie. Niestety równie częstą praktyką jak pożyczanie jest również nieoddawanie pieniędzy. Pamięć ludzka ma to do siebie, że nie przetrzymuje dobrze informacji o długach wobec innych, a z kolei psychika niechęć do przypominania innym o zobowiązaniach. Także z pozoru małe kwoty, potrafią urosnąć do liczących się sum. Sytuacje takie często pojawiają się wśród ludzi żyjących ze sobą na co dzień w jednym środowisku takim jak miejsce pracy lub jedno mieszkanie. Rozwiązaniem wszystkich problemów wyżej wymienionych problemów staje się aplikacja mobilna wspomagająca rozliczanie długów. Zapisując wszystkie należności za jej pomocą w prosty sposób zyskuje się nie tylko możliwość ich kontrolowania, ale także całkowite wykluczenie oddawania małych kwot pieniędzy. Jest to możliwe dzięki temu, że rzadko kiedy zakupy robi tylko jedna osoba, więc koszty rozkładają się na wszystkich w danym środowisku. Mając wszystkie dane o należnościach zapisane w jednym miejscu, współdzielone między obiema stronami „transakcji" zyskuje się pewność takich samych informacji, ich trwałość, zwolnienie z ich zapamiętywania oraz z problematycznego przypominania drugiej osobie o zaległych długach.

1.3 Cel pracy

Rozwiązanie problemu zapamiętywania długów pieniężnych wobec innych osób oraz niezwróconych należności dla użytkownika przez zaprojektowanie i implementację aplikacji mobilnej pracującej na systemie operacyjnym Android.

1.4 Zakres pracy

Zakres niniejszej pracy obejmuje określenie wymagań, przegląd istniejących rozwiązań, projekt oraz implementację aplikacji mobilnej wspomagającej rozliczanie długów.

Rozdział 1 to wstęp wprowadzający w problematykę pracy (podrozdział 1.1), opis problemu (1.2), cel (1.3) oraz zakres (1.4). W rozdziale 2 omówiono wymagania projektowe. Rozdział 3 to przegląd stanu wiedzy i techniki w zakresie tematyki pracy. Po wprowadzeniu (3.1) następuje analiza z podziałem na istniejące rozwiązania (3.2) i przydatne technologie (3.3) po czym w podrozdziale 3.4 wyciągane są wnioski. W rozdziale 4 wskazano założenia projektowe, czyli najpierw określono przedmiot prac (4.1), sformułowano wymagania funkcjonalne (4.2), niefunkcjonalne (4.3), opisano podstawową architekturę systemu (4.4), a także sposób realizacji (4.5). Wykorzystując powstałe założenia, w następnym rozdziale stworzono projekt aplikacji. Po wprowadzeniu (5.1), spisano przypadki użycia (5.2), przedstawiono projekty interfejsu (5.3), opisano bazę danych (5.4) oraz architekturę komponentów (5.5), by podsumować całość w podrozdziale 5.6. Następnie omawiając fazę implementacji wskazano wykorzystane środowiska i narzędzia programistyczne (6.2), pokazano zaprogramowany interfejs użytkownika (6.3), przeanalizowano komunikację z bazą danych (6.4), przedyskutowano powstałe problemy (6.5) i wyjaśniono proces instalacji i uruchomienia aplikacji (6.6). W ostatnim rozdziale 7 podsumowano wykonane prace (7.1), realizację celu pracy (7.2), wyciągnięto wnioski (7.3) oraz zaproponowano sugestie dalszego rozwoju aplikacji (7.4).

1.5 Podziękowania

W tym miejscu chciałbym podziękować mojemu koledze Wiktorowi i współlokatorom: Maćkowi, obu Michałom oraz Marcinowi za sam pomysł stworzenia aplikacji, który wykorzystałem w niniejszej pracy. Dzięki nim nigdy nie zabrakło mi motywacji, pisząc ją, czułem, że jest ona komuś naprawdę potrzebna, będzie wykorzystywana i rozwiąże realny problem moich znajomych i nie tylko. Jestem wdzięczny również za stworzenie warunków do pracy, wparcie mentalne i ich ocenę jako potencjalnych użytkowników.

Rozdział 2

Wymagania projektowe

Wytworzona aplikacja ma na celu rozwiązanie problemu nieoddawania pożyczonych pieniędzy między ludźmi żyjącymi w jednym środowisku takim jak mieszkanie lub miejsce pracy. Pożyczenie pieniędzy odbywa się nie wprost - nie przez przekazanie gotówki lub przelew środków, lecz przez kupno dóbr na rzecz grupy ludzi. Głównymi funkcjami aplikacji są:

- dzielenie określonej kwoty pieniędzy - wartości zakupów na wiele osób,
- zapisywanie historii długów
- obliczanie aktualnego salda między każdą parą osób

Aplikacja wykorzystywana przez współpracowników lub współlokatorów może być używana do rozliczeń za rzeczy codziennego użytku takie jak płyn do mycia naczyń, olej do smażenia, proszek do prania, papier toaletowy itp. Nie ma limitu co do ilości użytkowników aplikacji. Musi być ona powszechnie dostępna dla każdego posiadacza smartfona. Możliwościami rejestracji są:

- rejestracja przy pomocy konta w portalu Facebook
- rejestracja przy pomocy adresu e mail i hasła

Wymaga się, by instalacja oraz korzystanie z aplikacji było intuicyjne, bez konieczności czytania poradnika czy instrukcji przed użytkowaniem. Każdy użytkownik posiada swoje grono osób, wobec których długi chce zapisywać. Ma możliwość powiadomienia osoby o chęci wyrównania rozliczenia, a także o każdym nowym dodanym długu do jego rachunku. Innym sposobem poinformowania dłużnika jest skopiowanie wygenerowanego tekstu przy pomocy gotowego szablonu i wysłanie go inaczej np. przez sms lub email. Istnieje widok ostatnich wydarzeń na koncie takich jak dodanie do „znajomych" lub naliczenie długu. Dodatkową funkcją jest dodawanie kategorii do każdego zapisywanego długu i przeglądanie statystyk na ich podstawie np. na co pożyczano najwięcej pieniędzy albo komu było się dłużnym najwięcej. Użytkownik może dodać dług znajomemu bezpośrednio, bez dzielenia łącznej kwoty zakupów. W przypadku omyłkowego wpisania niepoprawnych danych posiadacz aplikacji może edytować wpis lub go usunąć. Może to zrobić także druga strona, czyli dłużnik. Dane takie jak nazwa użytkownika i hasło, także są edytowalne. Użytkownik ma także możliwość ustawienia kwoty limitu i dni, po których dłużnicy zostaną powiadomieni o przekroczonym limicie i potrzebie rozliczenia.

Rozdział 3

Stan wiedzy i techniki w zakresie tematyki pracy

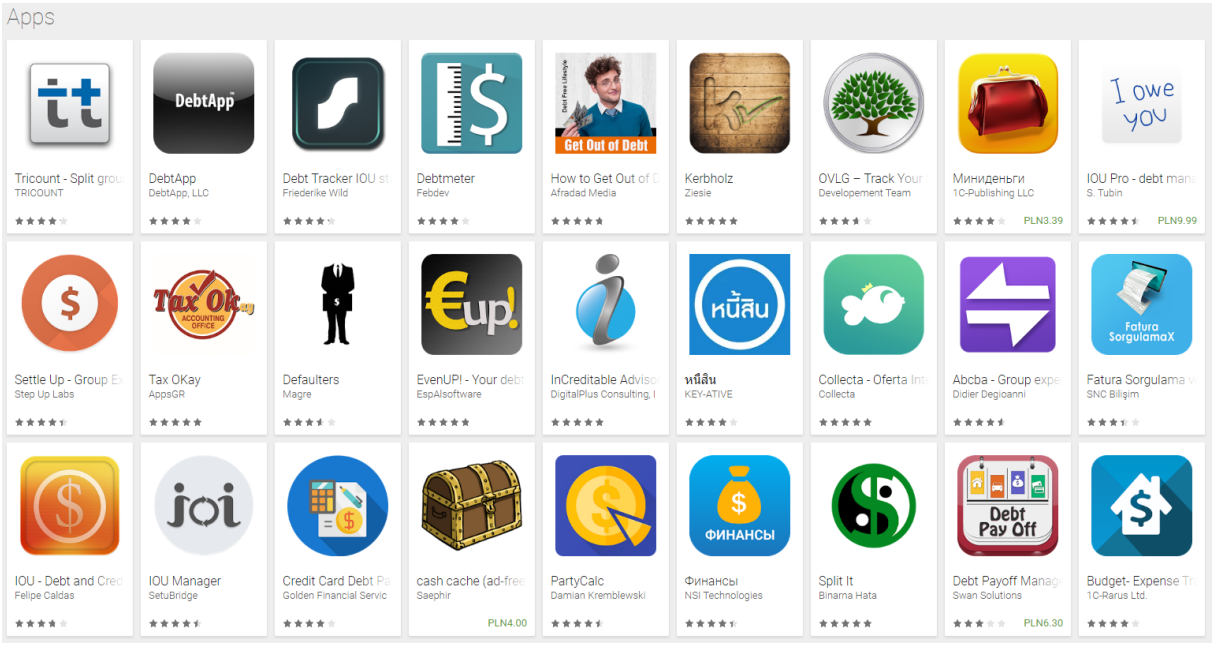
3.1 Wprowadzenie

W rozdziale tym omówiono istniejące, podobne rozwiązania problemu rozwiązywanego w niniejszej pracy. Porównywane zostały aplikacje z dwóch sklepów najpopularniejszych, mobilnych systemów operacyjnych - Google Play i App Store, jednak przedstawiono przykłady z pierwszego ze względu na analogiczną sytuację w drugim. Analizie zostaną poddane również techniki i technologie, które mogą być najbardziej przydatne do wytworzenia produktu. Omówiono rozwiązania dla strony klienta, czyli samej aplikacji oraz back-endu dla niej, czyli serwera i bazy danych.

3.2 Przegląd podobnych istniejących rozwiązań

3.2.1 Podział aplikacji o tematyce rozliczania długów

Wyszukując w sklepie aplikacji obecnym w systemie operacyjnym Android hasło „rozliczanie długów” znaleziono kilka pozycji porównywalnych do aplikacji omawianej w tej pracy, co widać na rysunku 3.1. Mimo użycia do wyszukiwania polskiego zwrotu nie znaleziono żadnych aplikacji z polską nazwą, a także pojawiły się pozycje służące do innych celów, więc w celu uzyskania bardziej trafnych wyników użyto angielskiego tłumaczenia

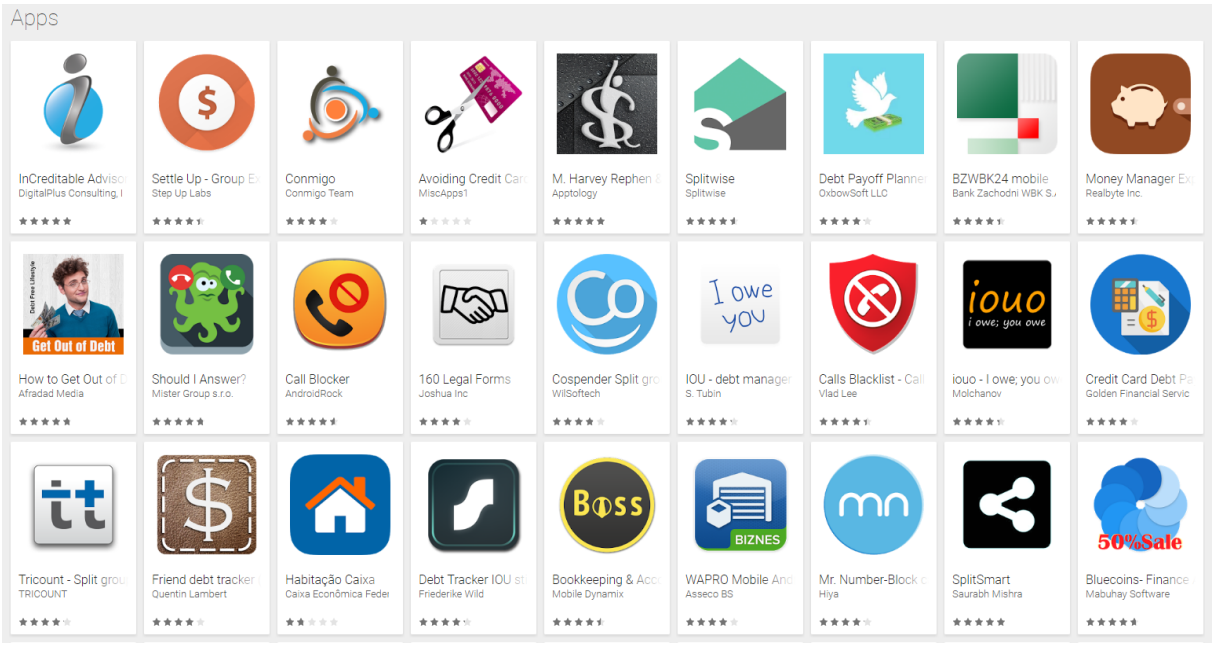


Rysunek 3.1: Wyszukanie aplikacji w sklepie Play z hasłem „rozliczanie długów”

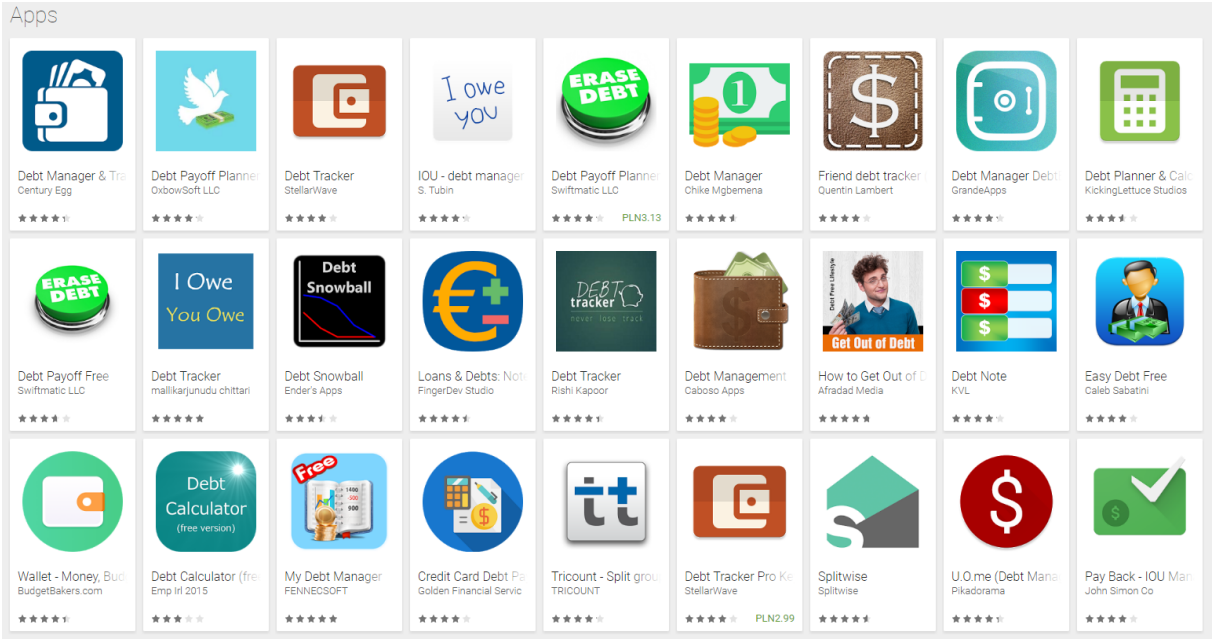
„debt settlement” (rys. 3.2). Co ciekawe wyniki tego wyszukania różnią się znacznie od poprzednich, ale również pojawiło się wiele aplikacji nieadekwatnych. Najwięcej aplikacji o ogólnej tematyce długów uzyskano przy haśle wyszukiwania „debt” co widać na rysunku 3.3. Po przeglądzie nawet samych tytułów mówiących o zastosowaniu aplikacji można zauważyć, że da się wydzielić dwie kategorie aplikacji o tematyce rozliczeń długów. Pierwsza z nich to tak zwane „trackery” lub „managery” - aplikacje offline (nie działające w sieci internet), które można porównać do zmodyfikowanych notatników, w których jest możliwość wpisania długu dla swoich kontaktów lub ręcznie wpisanej osoby. Rzadko się zdarza, by aplikacje jako pojedyncze pozycje występowały na dwóch platformach na raz, ze względu na brak takiej potrzeby - między użytkownikami nie występują interakcje, więc zapewnienie dostępności nie jest wymagane. Twórca może skupić się na jednym systemie i nie będzie to ujmą dla aplikacji. Druga kategoria natomiast to programy bardziej przypominające portal społecznościowy ze względu na możliwość zapraszania znajomych i wykonywania w stosunku do nich jakichś działań. W tym przypadku ważne jest, by była ona dostarczona do dwóch sklepów jednocześnie, gdyż niemożliwość zainstalowania jej przez choćby jedną osobę w grupie sprawia, że rozwiązanie problemu długów za jej pomocą staje się nierealne.

3.2.2 Aplikacje typu „tracker”

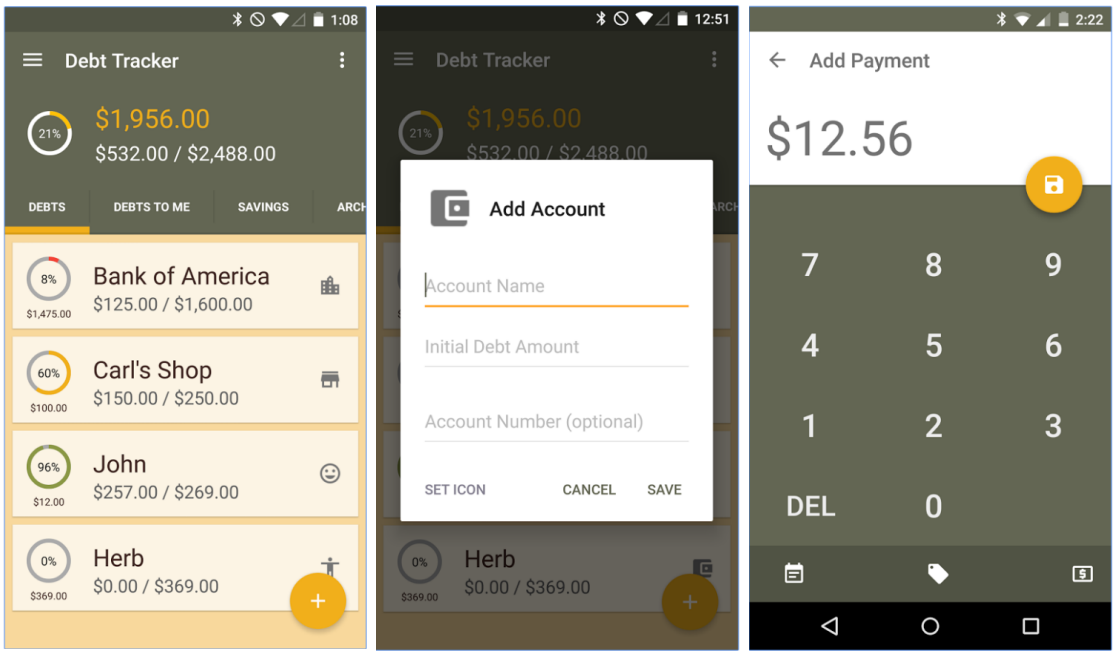
Przykładem aplikacji należącej do pierwszej kategorii jest np. Debt Tracker. Poniżej zaprezentowano zrzuty ekranu. Tak jak wspomniano, Debt Tracker można porównać do notatnika, ponieważ służy on jedynie do zapisywania informacji o kwocie należności względem ludzi lub firm. Informacje te znajdują się tylko na urządzeniu użytkownika, dlatego nie można stosować tego typu aplikacji do automatycznego wyrównywania długów. Nie są przechowywane informacje kontaktowe dłużników, ani nie mają oni także konta, więc nie można wysłać im automatycznego powiadomienia w postaci e mail’a, SMS’a czy notyfikacji push.



Rysunek 3.2: Wyszukanie aplikacji w sklepie Play z hasłem „debt settlement”



Rysunek 3.3: Wyszukanie aplikacji w sklepie Play z hasłem „debt”



Rysunek 3.4: Debt Tracker

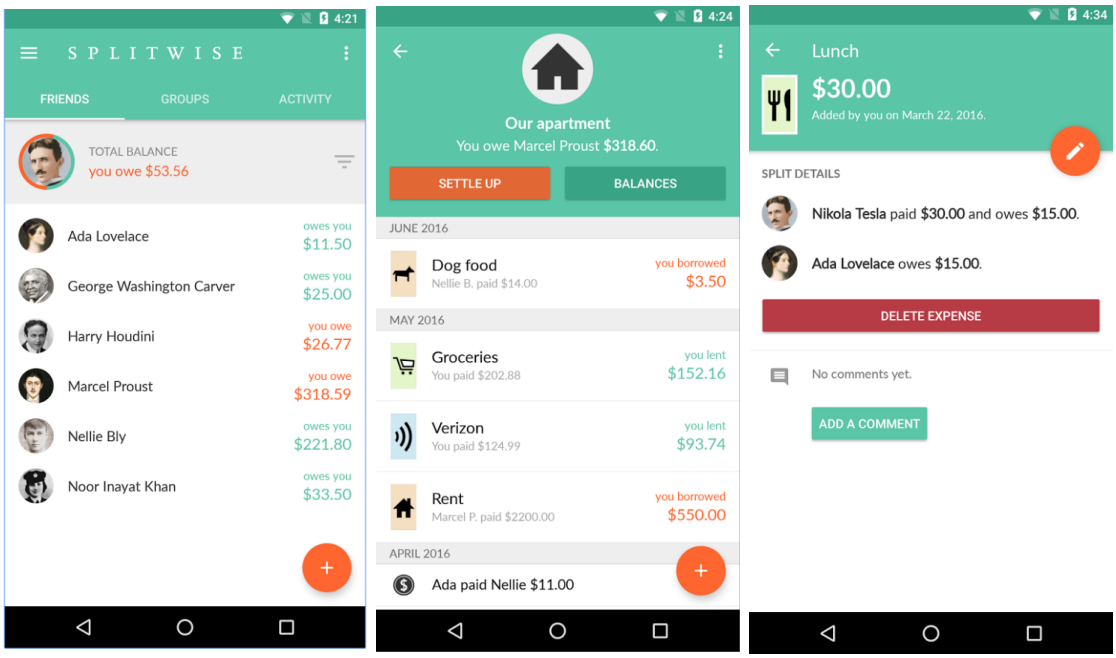
3.2.3 Aplikacje typu „portal społecznościowy”

Aplikacji należących do drugiej kategorii jest zdecydowanie mniej. Jest to spowodowane większym stopniem trudności wytworzenia ze względu na potrzebę zdalnego przechowywania danych, obsługę rejestracji, logowania użytkownika, a także potrzebą dostarczenia zarówno na system Android jak i iOS. Znalezienie takiej aplikacji nie jest proste co nie oznacza, że takowe nie istnieją. Za przykład może posłużyć Splitwise. Jest to aplikacja online, a zatem by z niej korzystać należy założyć konto. Następnie można dołączyć do grupy, w której chce się rozliczać z innymi lub dodać komuś dług bezpośrednio, nie w grupie. Istnieje także opcja „inteligentnego rozliczania”. Po jej włączeniu nie zawsze oddaje się pieniądze osobie, której jesteśmy coś dłużni, lecz zdarza się, że oddajemy je osobie, która jest pożyczkodawcą naszego pożyczkodawcy. Podejście takie wprowadza dużo zamieszania w rozliczenia i trzeba zaufać algorytmom stojącym za nimi. Aplikacja oferuje również przypomnienie o długu przez wysłanie e maila i wysyła powiadomienie push przy każdym dodaniu długu do wszystkich biorących udział w zakupach. Aplikacja występuje zarówno w sklepie Google Play jak i App Store.

3.3 Przegląd przydatnych technik i technologii

3.3.1 Aplikacja

By zachować sens działania aplikacji w środowisku grupy ludzi należy dostarczyć ją do możliwie największej ilości potencjalnych odbiorów. Aby to zrobić wymagane jest napisanie jej na dwa najpopularniejsze mobilne systemy operacyjne - Android oraz iOS. Można by zrobić to pisząc osobno dwie natywne wersje aplikacji, jednak korzystając z obecnie najnowszych technologii nie jest to konieczne, szczególnie jeżeli mamy na myśli niezaawansowaną aplikację niewykorzystującą grafiki 3D lub innych bardziej skomplikowanych, specyficznych zagadnień. Do tego celu warto wykorzystać frameworki, które



Rysunek 3.5: Splitwise

zapewniają napisanie jednego kodu, który można uruchomić na wielu platformach. Przykładami takich rozwiązań są:

- Ionic
- Cordova
- PhoneGap
- Xamarin
- React Native

Ionic, Cordova, PhoneGap Aplikacje napisane za pomocą pierwszych 3 frameworków są aplikacjami hybrydowymi. Oznacza to, że uruchamiają one kod JavaScript'owy w kontenerze WebView. Są to tak naprawdę strony internetowe wbudowane w aplikację przez komponent WebView i udające natywne kontrolki aplikacji. By rozszerzyć funkcjonalność wykorzystują natywne rozszerzenia. Aplikacje hybrydowe mają największy problem z wydajnością. WebView nie zostało stworzone do wykonywania dużych aplikacji webowych, lecz do wyświetlania stron internetowych. O ile telefony z wyżej półki potrafią sobie poradzić z takim rozwiązaniem, na sprzęcie ze słabszym hardware'em (procesorem, pamięcią ram) aplikacje takie mogą się zacinać, a animacje nie będą tak płynne jak w aplikacjach natywnych. Innym kłopotem jest też kompatybilność. Różne systemy, wersje systemów mogą mieć różne błędy w wyświetlaniu stron internetowych. Przekłada się to na właśnie rozwiązania hybrydowe gdzie nie mamy pewności, że aplikacja będzie wyglądać prawidłowo na każdym urządzeniu.

Xamarin i React Native Xamarin oraz React Native w odróżnieniu od Ionic, Cordovy i PhoneGap nie tworzą aplikacji w kontenerze webowym [6]. W przypadku React

Native kod aplikacji również jest pisany w JavaScript'cie, jednak sama aplikacja jest renderowana używając natywnych komponentów, ponieważ kod aplikacji wywołuje natywne API platformy, na której aplikacja jest uruchamiana. Xamarin korzysta z języka C# i przeportowanych bibliotek Androida i iOS. W procesie kompilacji wywołania tych bibliotek w C# są zamieniane na natywne wywołania natywnego API platformy. Dzięki temu osiąga się nie tylko zgodność ze standardami wyglądu danego systemu operacyjnego, lecz wydajność zbliżoną do aplikacji napisanej w Objective-C w przypadku iOS lub Javie w przypadku Androida.

3.3.2 Back-end

Każda aplikacja mobilna będąca aplikacją online musi posiadać back-end, czyli minimum bazę danych odpowiedzialną za przechowywanie informacji użytkowników. W zależności od wybranej architektury aplikacja może być:

- cienkim klientem
- grubym klientem

Cienki klient jest niezależny od aplikacji serwerowej, gdyż porozumiewa się z back-endem za pomocą ustalonego protokołu, niezależnego od serwera. Plusem takiego rozwiązania są niewielkie wymagania co do aplikacji klienckiej, jednak minusem jest funkcjonalność ograniczona protokołem komunikacyjnym oraz opóźnienie wynikające z opóźnień sieci. Gruby klient natomiast przetwarza dane po swojej stronie i serwer wykorzystuje jedynie do ich przechowywania. Wadą takiego zastosowania jest większe obciążenie klienta.

Cienki klient w architekturze klient-serwer Decydując się na pierwsze rozwiązanie, czyli cienkiego klienta, zadaniem twórcy aplikacji jest napisanie aplikacji działającej na serwerze, komunikującej się z bazą danych oraz utrzymanie samego serwera. Rozwiązanie takie oferuje na przykład framework Spring, za pomocą którego można utworzyć endpointy (interfejs dzięki któremu aplikacja może komunikować się z serwerem). Wymaga to jednak wiedzy programisty na temat pisania aplikacji serwerowych, której często nie posiada specjalizując się w technologiach mobilnych. Drugim aspektem jest też ilość tzw. kodu boilerplate, czyli takiego, który napisać trzeba, a który nie zmienia się pisząc różne aplikacje. Ważny jest fakt, że trzeba mieć na względzie skalowalność aplikacji. Przy zwiększającym się ruchu pojedynczy serwer może nie oferować odpowiedniej wydajności co może odbić się na szybkości aplikacji. Za skalowalność, load balancing odpowiada właściciel aplikacji.

Gruby klient w architekturze serverless Wszystkich tych wad można pozbyć się stosując nową, dostępną od niedawna architekturę serverless [10, 11, 12, 13]. Wybierając ją aplikacja mobilna staje się grubym klientem, ponieważ jest zależna od wybranego dostawcy usług back-end'owych. Usługi te to niezależne produkty chmurowe, które programista może dowolnie wybierać i z nich korzystać. Do najbardziej podstawowych usług wykorzystywanych przy samym pisaniu aplikacji należy baza danych, uwierzytelnianie, miejsce na dysku czy tzw. cloud functions, czyli pojedyncze funkcje wywoływane przez określone zdarzenia. Zdarzenia te to przykładowo zalogowanie użytkownika, zapisanie dokumentu w określonej lokalizacji w bazie danych lub na dysku twardym. Jeśli chodzi

o skalowalność takiego rozwiązania to jest ona zapewniana przez dostawcę, więc zwalnia twórcę aplikacji z jej zapewnienia. Przykładami chmur, które oferują usługi, które można wykorzystać stosując architekturę serverless są Amazon Web Services oraz Google Cloud Platform, a dokładnie platforma Firebase [15, 3]. Wybierając takie rozwiązanie programista wykorzystuje bezpośrednio w aplikacji API (interfejs programistyczny) chmury wykorzystując je do komunikacji z np. bazą danych. W razie potrzeby można wykorzystać Cloud Functions jako endpointy tworzone na serwerze w tradycyjnej architekturze.

3.4 Podsumowanie i wnioski

Ograniczając wybór technologii do wieloplatformowych na pierwszy rzut oka opcji jest dużo, jednak po głębszej analizie tych rozwiązań okazuje się, że nowsze rozwiązania takie jak Xamarin i React Native oferują więcej benefitów od innych. Napisanie jednego kodu, który stworzy natywną aplikację na dwa systemy operacyjne jest ogromną zaletą zwłaszcza w przypadku, gdy duża dostępność aplikacji jest krytyczna dla jej sukcesu w rozwiązaniu problemu. Takie podejście idealnie wpisuje się również w cel poboczny traktujący o jak najmniejszym koszcie wytworzenia aplikacji. W niniejszej pracy wykorzystano framework React Native ze względu na jego wysokie podobieństwo do webowego odpowiednika - React'a. React Native jest oparty właśnie na nim co sprawia, że pisanie aplikacji mobilnych dla programistów webowych jest niezwykle proste. Wybór back-end'u aplikacji również został pokierowany celem wykorzystania jak największej ilości gotowych usług i kodu. Zdecydowano się na architekturę serverless z BaaS, czyli Back-end as a Service (back-end jako usługa). Jako dostawca usług przy darmowym planie dobrym wyborem jest Firebase, ponieważ oferuje wystarczającą ilość zasobów do wykorzystania oraz ma niski próg wejścia, który wraz z dobrze przygotowaną dokumentacją sprawia, że korzystanie z usług jest naprawdę proste i można skupić się na samej aplikacji.

Rozdział 4

Założenia projektowe

4.1 Przedmiot prac

Przedmiotem projektu jest wykonanie aplikacji wspomagającej rozliczanie długów. Aplikacja ma rozwiązywać problem zapamiętywania długów pieniężnych wobec innych osób oraz niezwróconych należności dla użytkownika. Główną funkcją jest gromadzenie informacji o należnościach między użytkownikami i obliczaniu salda między nimi. Chcąc korzystać z aplikacji należy założyć konto rejestrując się. Może to zrobić na dwa sposoby: korzystając z konta na portalu Facebook lub tradycyjnie - podając adres e-mail oraz hasło. Drugim krokiem rejestracji jest potwierdzenie swojej tożsamości przez kliknięcie linku w wiadomości poczty elektronicznej wysyłanej podczas pierwszego etapu rejestracji. Zalogowany posiadacz konta może dodawać inne osoby do grupy tzw. znajomych. Gdy obaj użytkownicy mają się w „znajomych" mogą dodać dług znajomemu bezpośrednio, wpisując kwotę, którą jest mu winien lub w sposób pośredni - rozdzielając łączną kwotę zakupu na wiele osób z możliwością uwzględnienia w rozliczeniu samego siebie. W aplikacji dostępny jest widok listy wszystkich znajomych z bieżącym stanem rozliczeń - saldem, które wskazuje która strona i jak dużo jest winna po uwzględnieniu wszystkich dotychczasowych należności. Użytkownik ma do dyspozycji również szczegółowy, chronologiczny wykaz wszystkich długów między nim i danym znajomym. Każdy dług ma kwotę, opis oraz kategorię służącą celom statystycznym. W momencie dodania, edycji lub usunięcia zobowiązania, a także dodania osoby do znajomych przez jedną stronę, druga dostaje powiadomienie w formie notyfikacji push. Ostatnie 3 takie zdarzenia są wyświetlane w formie „ostatnich wydarzeń". Istnieje również edytowalna kwota limitu długu i liczba dni, po których dłużnicy przekraczający limit zostaną powiadomieni o potrzebie rozliczenia. Aplikacja wytworzona zostanie na system mobilny Android i iOS za pomocą frameworka React Native. Back-end oparty będzie w całości na platformie Firebase, a dokładnie na następujących modułach: Firestore, Authentication oraz Cloud Functions.

4.2 Wymagania funkcjonalne

Jestem osobą, która kupuje pewne dobra dla grupy ludzi i się z nią rozliczam. Nie muszę otrzymywać w zamian gotówki, pozostałe osoby mogą kupić coś dla mnie przy następnych zakupach. Aby zapamiętać informację o ich długach muszę zapisywać je w miejscu, gdzie zarówno ja i oni będą mieli do niej dostęp. Nie chcę by była ona dostępna publicznie. Większość osób z tej grupy posiada konta na portalu Facebook, jednak nie wszyscy. Zdarza się, że kupuję coś tylko jednej osobie i nie chcę dzielić tego kosztu na kilka części. Często trzeba

ustalić kto teraz powinien płacić, więc potrzebuję listy moich znajomych z saldem pomiędzy poszczególnym znajomym i mną. Mam dużo znajomych, więc muszę szukać na liście, któremu znajomemu mam dopisać dług. Gdy to zrobię muszę powiedzieć mojemu dłużnikowi o dopisanym długu. Czasem, gdy mamy jakiekolwiek wątpliwości co do rozliczeń, spoglądamy w historię długów, by je rozwiać. Gdy źle policzymy całkowitą kwotę zakupów trzeba zmieniać kwotę długu lub całkowicie usunąć dług. Gdy byłem długo nieobecny to mam duży problem w dojściu co było kupowane w okresie mojej absencji. Sprawdzam szczegóły rozliczeń z każdym czy się nie pomylono na moją niekorzyść. Prowadzę również statystyki na temat kategorii rzeczy, które kupuję lub ktoś mi kupuje. Niekiedy w rachunku z jednym z moich znajomych nierozliczona kwota urośnie zbyt duża i w tym przypadku mówię mu o potrzebie rozliczenia.

Każde wymaganie ma ustalony priorytet: H - high (wysoki), M - medium (średni), L - low (niski).

1. Założenie konta za pomocą email'a i hasła. (H)
2. Założenie konta za pomocą konta w portalu Facebook. (M)
3. Logowanie za pomocą email'a i hasła. (H)
4. Logowanie za pomocą konta w portalu Facebook. (M)
5. Wylogowanie. (H)
6. Edycja nazwy użytkownika. (L)
7. Edycja hasła. (L)
8. Dodanie znajomego. (H)
9. Przeglądanie listy znajomych użytkownika z saldami. (H)
10. Wyszukiwanie na liście znajomych. (L)
11. Przypisanie długu do kategorii. (L)
12. Dodanie długu kilku znajomym przez rozłożenie kwoty zakupów. (H)
13. Dodanie długu znajomemu. (M)
14. Przeglądanie historii długów między użytkownikiem i jego znajomym. (H)
15. Edycja danego długu. (M)
16. Usunięcie danego długu. (M)
17. Przeglądanie ostatnich zdarzeń względem użytkownika. (L)
18. Przejście do historii długów między użytkownikiem i znajomym ze zdarzenia utworzonego przez znajomego. (L)
19. Przeglądanie statystyk dotyczących całej historii długów dotyczących użytkownika. (L)

4.3 Wymagania niefunkcjonalne

1. Brak potrzeby szkolenia użytkownika przed skorzystaniem z aplikacji.
2. Działanie pod systemem Android w wersji 4.4 lub wyższej.
3. Działanie pod systemem iOS w wersji 9 lub wyższej.
4. 1 GB miejsca w bazie danych.
5. Obsługa 20 000 zapisów do bazy danych dziennie.
6. Obsługa 50 000 odczytań z bazy danych dziennie.
7. Obsługa 20 000 usunięć z bazy danych dziennie.
8. Obsługa 10 GB transferu danych miesięcznie.

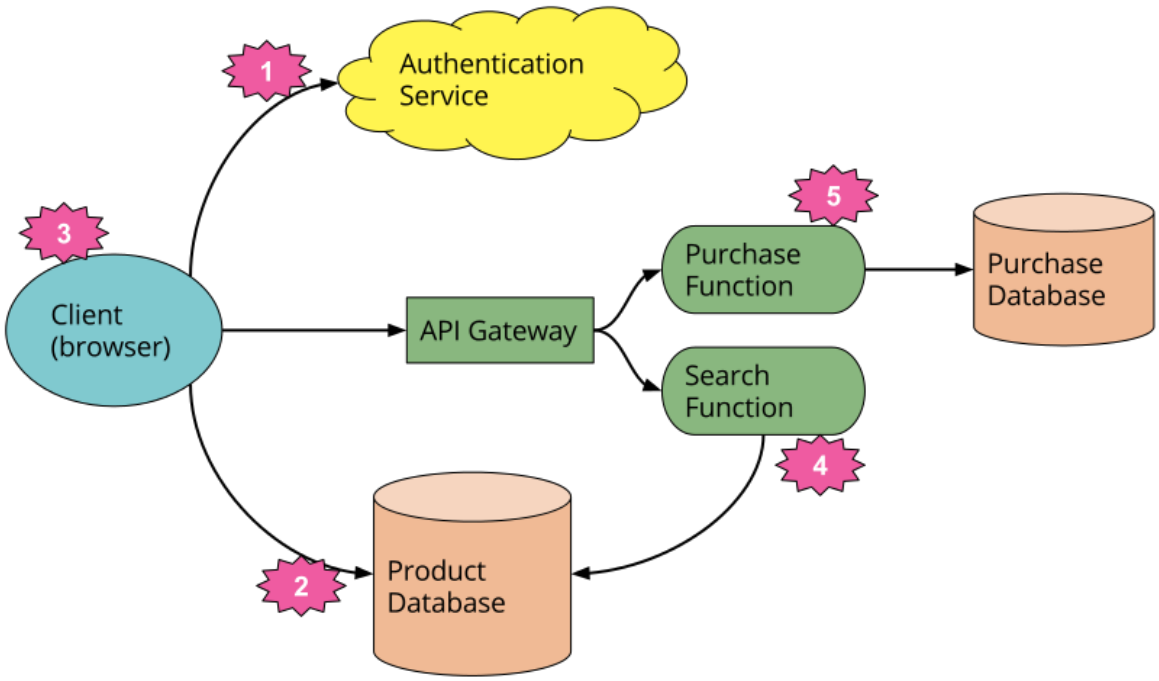
4.4 Opis podstawowej architektury systemu

Rozwiązanie oparte jest na architekturze serverless. Aplikacja kliencka (nr 3 na rysunku 4.1) na telefonie użytkownika komunikuje się z usługami Back-end as a Service przez JavaScript SDK (Software Development Kit). Aplikacja pracuje jako gruby klient. Użytkownik korzystając z interfejsu graficznego wywołuje API usług chmurowych - wymienia informacje bezpośrednio z bazą danych (2) oraz usługą uwierzytelniającą (1). Baza danych jest skalowalną bazą NoSQL przechowującą dane w dokumentach, które zawierają pola i ich wartości w formacie JSON. Usługa autentykacyjna zawiera z kolei informacje niezbędne do uwierzytelniania użytkowników. Część przetwarzania danych wykonuje się w aplikacji, a część w Cloud Functions (pojedynczych funkcji JavaScript'owych, nr 4 i 5 na rysunku 4.1) wdrożonych w chmurze. Schemat architektury przedstawiono na rysunku 4.1. Do wysyłania powiadomień push aplikacja wykorzystuje zapytania HTTP do serwerów, które w zależności od systemu operacyjnego urządzenia, do którego kierowane jest powiadomienie, wysyła je korzystając z usługi Google Cloud Messaging lub Apple Push Notification (rysunek 4.2).

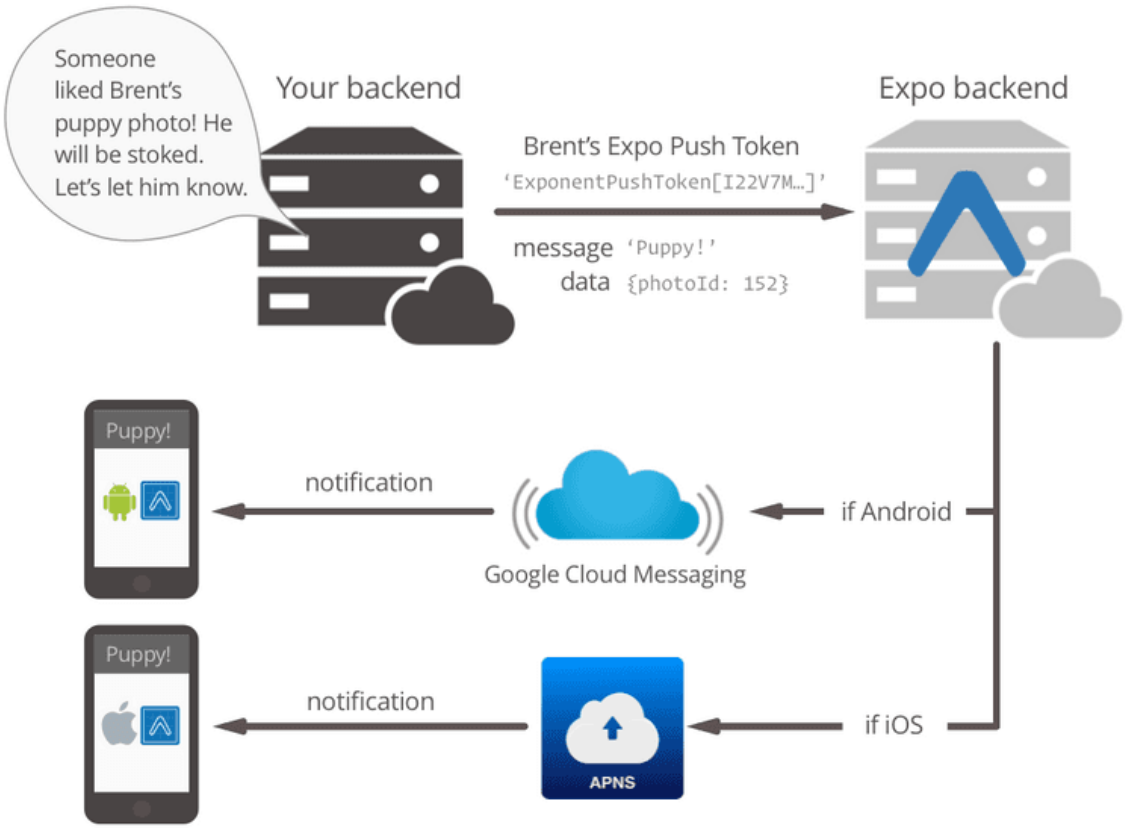
4.5 Sposób realizacji

4.5.1 Główne narzędzia i technologie

Aplikacja zostanie wykonana za pomocą frameworku React Native [1, 16] wraz z narzędziami Expo [14], które usprawnią proces jej wytwarzania, zapewnią wysyłanie powiadomień oraz umożliwią wdrożenie. Kod aplikacji zostanie napisany w języku JSX (rozszerzenie JavaScript'u [2] umożliwiające definiowanie w jednym pliku rozmieszczenia, wyglądu i zachowania komponentów React'owych), natomiast kod funkcji Cloud Functions w czystym JavaScript'cie. By umożliwić użytkownikom rejestrację i logowanie z kontem Facebook zostanie wykorzystane także API dostarczone przez ten portal. Jako dostawcę usług chmurowych niezbędnego do zastosowania architektury serverless wybrano platformę Firebase [3, 15] istniejącą na infrastrukturze Google Cloud. Wykorzystane zostaną komponenty Authentication, Cloud Firestore oraz Cloud Functions.



Rysunek 4.1: Przykład architektury serverless [10]



Rysunek 4.2: Schemat działania wysyłania powiadomień push [14]

4.5.2 Narzędzia wspierające

Aby zagwarantować czytelność, jednolitość oraz zgodność kodu z dobrymi praktykami programistycznymi zastosowane zostanie narzędzie ESLint [23], czyli tzw. linter. Możliwość wykorzystania usprawnień JavaScript'u [2] wprowadzonych w standardzie ECMAScript 2015 i późniejszych, czyli głównie funkcji asynchronicznych przy użyciu składni async/await zapewni narzędzie Babel. Przekompiluje ono, także JSX do czystego JavaScript'u. By wyeliminować całą kategorię błędów w czasie działania programu związanych z typami, a wynikających z tego, że JavaScript jest językiem o typowaniu słabym, wykorzystane zostanie narzędzie Flow [24], które po dodaniu typów do zmiennych, statycznie je sprawdza i informuje o błędach już w czasie pisania kodu.

4.5.3 Środowisko programistyczne i repozytorium

Jako zintegrowane środowisko programistyczne wybrano Webstorm 2017.3 firmy JetBrains. Wspiera ono zarówno JavaScript jak i JSX. Git będzie wykorzystany jako system kontroli wersji, natomiast jako repozytorium zdalne GitHub. Do zarządzania zależnościami oraz do instalacji np. narzędzi wymienionych wcześniej typu ESLint lub programów konsolowych do zarządzania usługami chmurowymi i wdrażania aplikacji użyty zostanie menedżer pakietów npm [21].

4.5.4 Narzędzia do projektowania

Do stworzenia diagramów UML [4] i nie tylko, niezbędnych do stworzenia projektu aplikacji posłuży program Visual Paradigm 14.1 oraz narzędzie online Creately. Do wykonania projektu interfejsu zostanie użyty Balsamiq Mockups 3.5.15.

Rozdział 5

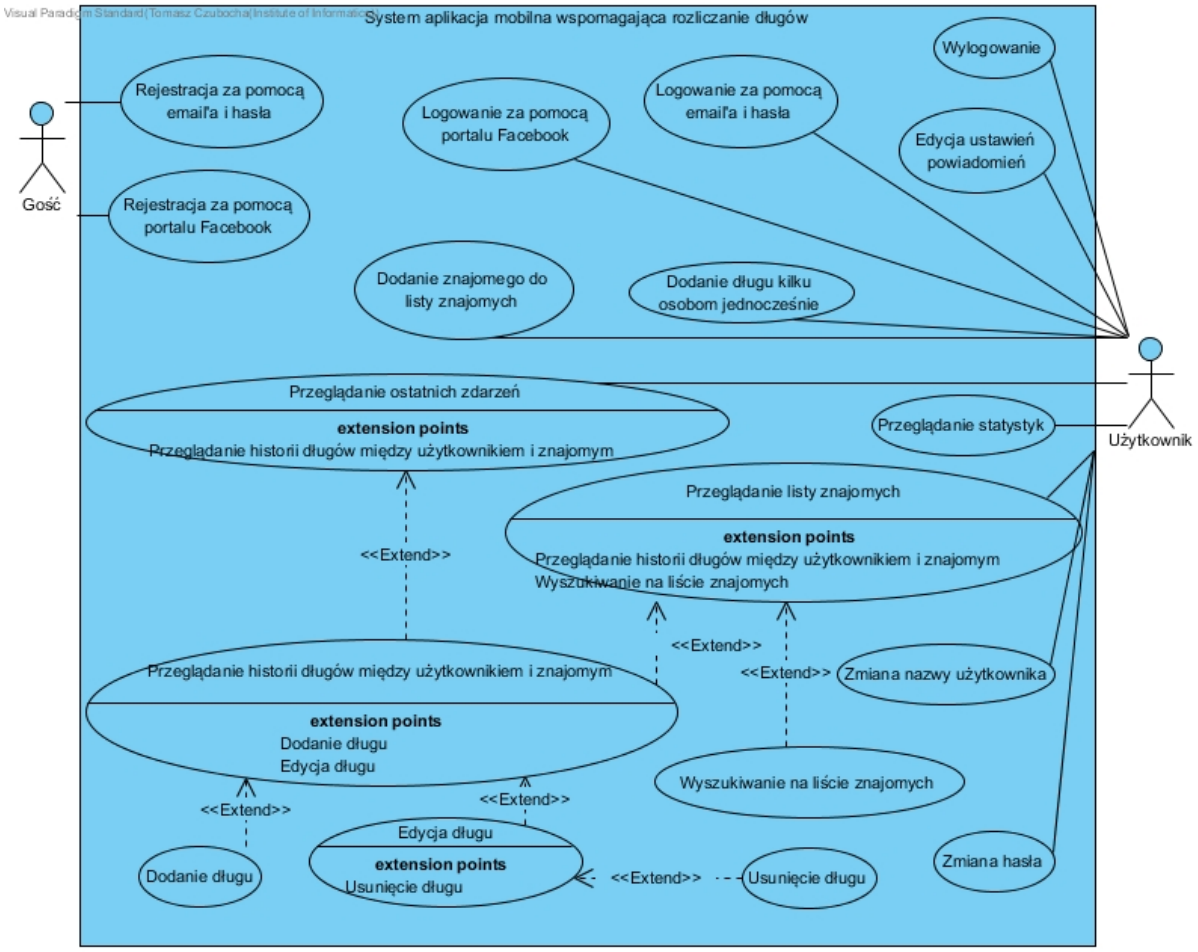
Projekt aplikacji

5.1 Wprowadzenie

W rozdziale poprzednim przyjęto założenia co do wymagań funkcjonalnych, niefunkcjonalnych, architektury sytemu oraz sposobu realizacji oprogramowania realizowanego w ramach niniejszej pracy. Pozwala to rozpocząć prace projektowe. Projekt obejmuje przypadki użycia, koncepty interfejsu użytkownika, krótkie omówienie bazy danych i jej struktury, a także architekturę komponentów framework'u React Native. Ukończony projekt umożliwi rozpoczęcie prac implementacyjnych.

5.2 Przypadki użycia

UC1: Rejestracja za pomocą email'a i hasła
Główny aktor: Gość
Priorytet: Wysoki
Warunki wstępne: System wyświetla ekran logowania
Główny scenariusz:
1. Gość wybiera opcję rejestracji
2. System wyświetla formularz
3. Gość podaje żądane dane
4. Gość potwierdza rejestrację
5. System wyświetla komunikat o potwierdzeniu rejestracji przez link w email'u
6. System powraca do ekranu głównego



Rysunek 5.1: Diagram przypadków użycia

Rozszerzenia:
3.A. Gość nie podaje adresu email lub podaje niepoprawny adres email
3.A.1. System wyświetla wiadomość o błędnym adresie email
3.B. Gość podał adres email, ale nie podał hasła lub hasło było za krótkie
3.B.1. System wyświetla wiadomość o zbyt krótkim haśle
3.C. Gość podaje email, za pomocą którego już ktoś się rejestrował
3.C.1. System wyświetla wiadomość, że istnieje już konto z takim adresem email

UC2: Rejestracja za pomocą portalu Facebook
Główny aktor: Gość
Priorytet: Średni
Warunki wstępne: System wyświetla ekran logowania
Główny scenariusz:
1. Gość wybiera opcję rejestracji za pomocą portalu Facebook
2. System wyświetla okno do uwierzytelniania z Facebook'iem
3. Gość uwierzytelnia się
4. System wyświetla komunikat o potwierdzeniu rejestracji przez link w email'u
5. System powraca do ekranu logowania

UC3: Logowanie za pomocą email'a i hasła
Główny aktor: Użytkownik
Priorytet: Wysoki
Warunki wstępne: System wyświetla ekran logowania
Główny scenariusz:
1. Użytkownik podaje email i hasło
2. Użytkownik wybiera opcję logowania
3. System wyświetla ekran główny
Rozszerzenia:
2.A. Użytkownik nie podał adresu email lub podał źle sformatowany adres email
2.A.1. System wyświetla wiadomość, że użytkownik podał źle sformatowany adres email
2.B. Użytkownik podał poprawnie sformatowany adres email, ale nie podał lub podał złe hasło
2.B.1. System wyświetla wiadomość, że hasło jest niepoprawne.

UC4: Logowanie za pomocą portalu Facebook
Główny aktor: Użytkownik
Priorytet: Średni
Warunki wstępne: System wyświetla ekran logowania
Główny scenariusz:
1. Użytkownik wybiera opcję logowania za pomocą portalu Facebook
2. System wyświetla okno uwierzytelniania Facebook'a
3. Użytkownik uwierzytelnia się
4. System wyświetla ekran główny

UC5: Wylogowanie
Główny aktor: Użytkownik
Priorytet: Wysoki
Warunki wstępne: System wyświetla ekran ustawień

Główny scenariusz:
1. Użytkownik wybiera opcję wylogowania
2. System wyświetla ekran logowania

UC6: Edycja nazwy użytkownika
Główny aktor: Użytkownik
Priorytet: Niski
Warunki wstępne: System wyświetla ekran ustawień
Główny scenariusz:
1. Użytkownik wybiera opcję zmiany nazwy użytkownika
2. System wyświetla modal edycji nazwy użytkownika
3. Użytkownik podaje żądane dane
4. System powraca do ekranu ustawień

UC7: Edycja hasła użytkownika
Główny aktor: Użytkownik
Priorytet: Niski
Warunki wstępne: System wyświetla ekran ustawień
Główny scenariusz:
1. Użytkownik wybiera opcję zmiany hasła użytkownika
2. System wyświetla modal edycji hasła użytkownika
3. Użytkownik podaje żądane dane
4. System powraca do ekranu ustawień

UC8: Edycja ustawień powiadomień
Główny aktor: Użytkownik
Priorytet: Niski
Warunki wstępne: System wyświetla ekran ustawień
Główny scenariusz:
1. Użytkownik wybiera opcję zmiany ustawień powiadomień
2. System wyświetla modal ustawień powiadomień
3. Użytkownik podaje żądane dane
4. System powraca do ekranu ustawień

UC9: Dodanie znajomego do listy znajomych
Główny aktor: Użytkownik
Priorytet: Wysoki
Warunki wstępne: System wyświetla ekran znajomych
Główny scenariusz:
1. Użytkownik wybiera opcję dodania znajomego do listy znajomych
2. System wyświetla modal dodawania do znajomych
3. Użytkownik podaje wymagane dane
4. System wyświetla komunikat o dodaniu znajomego
5. System powraca do ekranu znajomych
Rozszerzenia:
3.A. Użytkownik podaje dane o nieistniejącym znajomym
3.A.1. System wyświetla komunikat, że taki znajomy nie istnieje
3.A.2. System powraca do ekranu znajomych

UC10: Wyszukiwanie w liście znajomych
Główny aktor: Użytkownik
Priorytet: Niski
Warunki wstępne: System wyświetla ekran znajomych, lista znajomych nie jest pusta
Główny scenariusz:
1. Użytkownik podaje dane w polu wyszukiwania
2. System wyświetla odfiltrowywaną listę znajomych

UC11: Przeglądanie historii długów między użytkownikiem i znajomym
Główny aktor: Użytkownik
Priorytet: Wysoki
Warunki wstępne: System wyświetla ekran znajomych, lista znajomych nie jest pusta
Główny scenariusz:
1. Użytkownik wybiera nazwę znajomego z listy
2. System wyświetla ekran znajomego z danymi o znajomym i listą długów

UC12: Dodanie długu znajomemu
Główny aktor: Użytkownik
Priorytet: Średni
Warunki wstępne: System wyświetla ekran znajomych, lista znajomych nie jest pusta
Główny scenariusz:
1. Użytkownik wybiera nazwę znajomego z listy
2. System wyświetla ekran znajomego z danymi o znajomym i listą długów
3. Użytkownik wybiera opcję dodania długu znajomemu
4. Użytkownik podaje żądane dane
5. Użytkownik potwierdza dodanie długu
6. System powraca do ekranu znajomego
Rozszerzenia:
5.A. Użytkownik chce dodać dług znajomemu, który nie dodał go do znajomych
5.A.1. System wyświetla komunikat, że znajomy nie dodał go do znajomych

UC13: Edycja długu
Główny aktor: Użytkownik
Priorytet: Średni
Warunki wstępne: System wyświetla ekran znajomych, lista znajomych nie jest pusta, historia długów nie jest pusta
Główny scenariusz:
1. Użytkownik wybiera nazwę znajomego z listy
2. System wyświetla ekran znajomego z danymi o znajomym i listą długów
3. Użytkownik wybiera dług z listy
4. Użytkownik zmienia wybrane dane
5. Użytkownik potwierdza zapisanie długu
6. System powraca do ekranu znajomego

UC14: Usunięcie długu
Główny aktor: Użytkownik
Priorytet: Średni

Warunki wstępne: System wyświetla ekran znajomych, lista znajomych nie jest pusta, historia długów nie jest pusta
Główny scenariusz:
1. Użytkownik wybiera nazwę znajomego z listy
2. System wyświetla ekran znajomego z danymi o znajomym i listą długów
3. Użytkownik wybiera dług z listy
4. Użytkownik wybiera opcję usunięcia długu
5. System powraca do ekranu znajomego

UC15: Wygenerowanie tekstu powiadomienia o chęci rozliczenia
Główny aktor: Użytkownik
Priorytet: Średni
Warunki wstępne: System wyświetla ekran znajomych, lista znajomych nie jest pusta, historia długów nie jest pusta
Główny scenariusz:
1. Użytkownik wybiera nazwę znajomego z listy
2. System wyświetla ekran znajomego z danymi o znajomym i listą długów
3. Użytkownik wybiera opcję wygenerowania szablonu SMS
4. System kopiuje wygenerowany tekst do schowka

UC16: Przejście do ekranu znajomego z listy ostatnich zdarzeń
Główny aktor: Użytkownik
Priorytet: Niski
Warunki wstępne: System wyświetla ekran główny, lista ostatnich zdarzeń nie jest pusta
Główny scenariusz:
1. Użytkownik wybiera zdarzenie z listy
2. System wyświetla ekran znajomego z danymi o znajomym i listą długów

UC17: Przeglądanie statystyk
Główny aktor: Użytkownik
Priorytet: Niski
Warunki wstępne: System wyświetla ekran główny
Główny scenariusz:
1. Użytkownik wybiera opcję przeglądania statystyk
2. System wyświetla ekran statystyk

UC18: Dodanie długu wielu znajomym
Główny aktor: Użytkownik
Priorytet: Wysoki
Warunki wstępne: System wyświetla ekran główny, lista znajomych nie jest pusta
Główny scenariusz:
1. Użytkownik wybiera opcję dodania długu
2. System wyświetla listę znajomych z polami wyboru
3. Użytkownik wybiera, na których znajomych chce rozłożyć kwotę oraz uzupełnia pola dotyczące szczegółów długu
4. Użytkownik zaznacza pole wyboru decydujące czy ma on również zostać uwzględniony w rozliczeniu i zatwierdza dodanie długu
5. System powraca do ekranu głównego

Rozszerzenia:
4.A. Użytkownik wybrał minimum jednego znajomego, który nie dodał go do znajomych
4.A.1. System wyświetla komunikat, że znajomy nie dodał go do znajomych

5.3 Interfejs

W aplikacji można wyróżnić 8 różnych ekranów. W tym rozdziale przedstawiono działanie każdego z nich wraz z zaprojektowanym interfejsem. W procesie tym postawiono na prostotę, minimalizm i intuicyjność. Ze względu na wieloplatformowość aplikacji przedstawione prototypy interfejsu mogą się różnić w rzeczywistych aplikacjach ze względu na inne standardy wyglądu na różnych systemach operacyjnych.

Ekran logowania i rejestracji (rysunek 5.2) Do ekranów tych ma dostęp Użytkownik jak i Gość, który jest niezalogowanym użytkownikiem aplikacji. By się zarejestrować z email'em i hasłem użytkownik wybiera opcję rejestracji w dolnej części ekranu logowania, natomiast gdy chce się zarejestrować za pomocą konta w portalu Facebook wybiera opcję logowania z Facebook'iem, nawet jeśli nie ma jeszcze konta w aplikacji - ma on podwójne działanie, zarówno rejestracji jak i logowania. Jeżeli użytkownik chce się zalogować za pomocą email'a i hasła wpisuje dane w formularz i wybiera opcję logowania.

Menu główne (rysunek 5.3) Po zalogowaniu system wyświetla menu główne, które składa się z 3 ekranów: główny, znajomych i ustawień. By przejść do kolejnego ekranu z menu należy wybrać opcję na pasku menu lub wykonać gest z jednej strony ekranu do drugiej.

Ekran główny (rysunek 5.3a) Na tym ekranie użytkownik widzi ostatnie zdarzenia, opcję przejścia do statystyk oraz dodania nowego długu wielu znajomym jednocześnie. Po kliknięciu na dane zdarzenie system wyświetla ekran znajomego powiązanego z wydarzeniem. Po wybraniu opcji pokazania statystyk pokazuje się ekran statystyk, natomiast po wybraniu opcji dodania długu - ekran dodawania długu wielu znajomym.

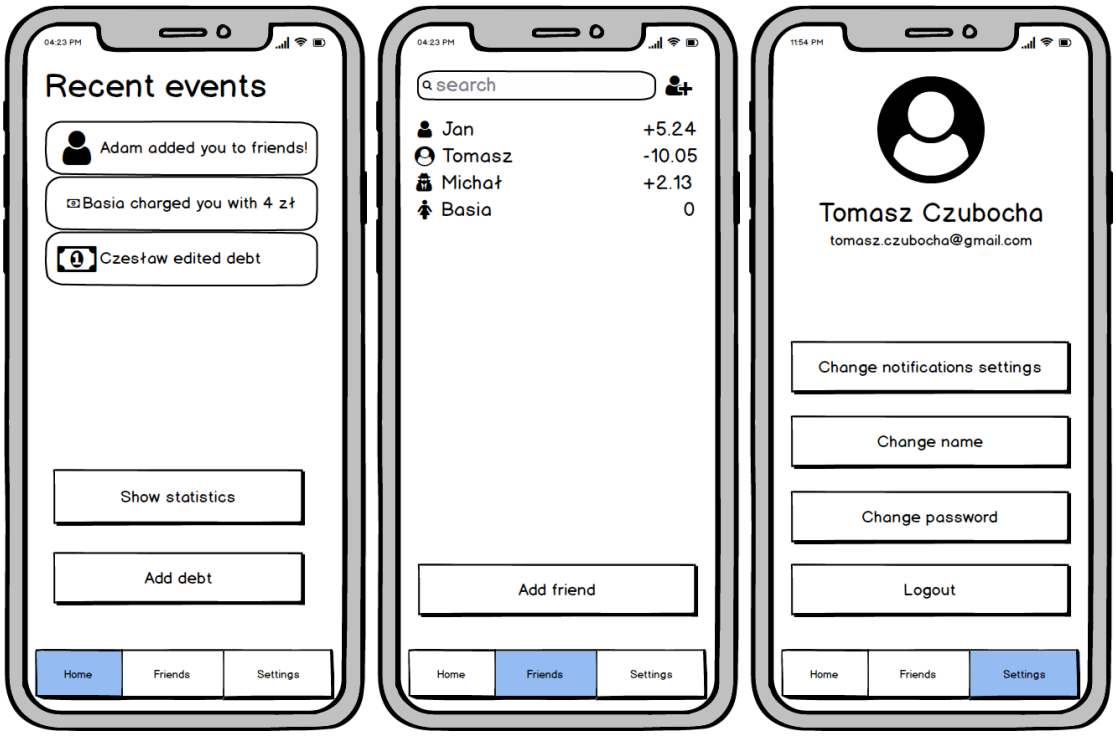
Ekran statystyk (rysunek 5.4a) Na ekranie statystyk nie ma żadnych kontrolek poza tekstem ze statystykami. Użytkownik może powrócić do ekranu głównego gestem lub strzałką w lewym, górnym rogu.

Ekran dodawania długu wielu znajomym (rysunek 5.4b) Ekran ten umożliwia rozdzielenie kwoty zakupów na wielu znajomych. W tym celu należy zaznaczyć na liście znajomych komu dodać dług zaznaczając pole wyboru znajdujące się przy nazwie znajomego. Następnie trzeba wpisać opis, kwotę oraz wybrać kategorię spośród możliwych do wyboru z rozwijanego menu. Ostatnim krokiem przed wybraniem opcji dodania jest zaznaczenie lub też nie pola wyboru mówiącego o tym czy użytkownik ma być uwzględniony przy podziale kwoty zakupów.

Ekran znajomych (rysunek 5.3b) Ekran znajomych przedstawia listę znajomych wraz z saldami. Oprócz niej widoczny jest pasek wyszukiwania na samej górze, pod paskiem powiadomień. Przy wpisywaniu w nim wartości lista znajomych zostaje przefiltrowana (rysunek 5.5b). Niżej, pod listą znajomych znajduje się przycisk dodania nowego

(a) Ekran logowania (b) Ekran rejestracji

Rysunek 5.2: Ekran logowania i rejestracji



(a) Ekran główny (b) Ekran znajomych (c) Ekran ustawień

Rysunek 5.3: Menu główne

znajomego. Po jego naciśnięciu system wyświetla modal (rysunek 5.5a) z polem do wpisania adresu email i przycisk potwierdzający dodanie. Po wybraniu znajomego z listy system wyświetla ekran znajomego.

Ekran znajomego (rysunek 5.6a) Ekran znajomego wyświetla jego dane - awatar, nazwę oraz email. Użytkownik ma też opcję dodania długu znajomemu bezpośrednio - poprzez przycisk z górnej części ekranu. Po jego naciśnięciu pojawia się modal z formularzem (rysunek 5.6b), w którym należy podać opis, kwotę oraz kategorię długu. Po wybraniu opcji dodania dług staje się widoczny w historii długów na ekranie znajomego. Obok przycisku dodania długu widoczny jest również przycisk odpowiedzialny za wygenerowanie tekstu powiadomienia znajomego o chęci rozliczenia.

Ekran ustawień (rysunek 5.3c) Na ekranie ustawień widoczne są informacje o aktualnie zalogowanym użytkowniku, czyli awatar, nazwa i email oraz przyciski umożliwiające edycję ustawień powiadomień, nazwy użytkownika, hasła, a także wylogowanie.

Wysuwanie wirtualnej klawiatury (rysunek 5.7) W niektórych przypadkach wysuwająca się klawiatura zasłania istotną część interfejsu - na przykład formularz, który jest właśnie uzupełniany. W takim przypadku, aby zapewnić poprawne działanie należy przesunąć elementy interfejsu. W przypadku ekranu rejestracji (rys. 5.7a) nie ma takiej potrzeby, ponieważ kontrolki znajdują się w górnej części ekranu, lecz ekran logowania jest zapełniony w całości. By rozwiązać problem wszystkie kontrolki przesuwają się wyżej robiąc miejsce klawiaturze, a niemieszczące się już na ekranie logo staje się niewidoczne (rys. 5.7b).

(a) Ekran statystyk

(b) Ekran dodawania długu wielu znajomym

Rysunek 5.4: Ekrany dostępne z ekranu głównego

(a) Ekran znajomych - dodanie znajomego (b) Ekran znajomych - wyszukiwanie

Rysunek 5.5: Ekran znajomych - opcje

(a) Ekran znajomego (b) Ekran znajomego - dodanie długu

Rysunek 5.6: Ekran znajomego

(a) Ekran rejestracji z wysuniętą klawiaturą

(b) Ekran logowania z wysuniętą klawiaturą

Rysunek 5.7: Ekrany z wysuniętą klawiaturą

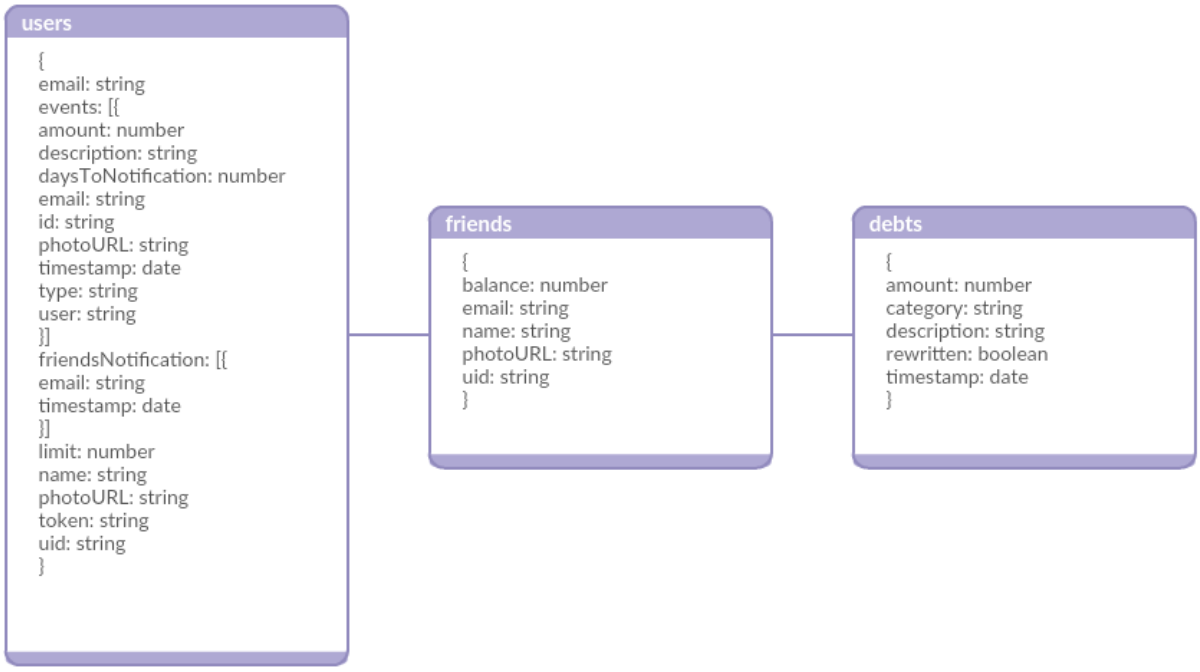
5.4 Baza danych

5.4.1 Wprowadzenie

Do tworzonego oprogramowania wybrano Cloud Firebase - bazę danych będącą częścią usług BaaS należących do Firebase. Jest to „document-oriented database", a więc baza danych, która przechowuje dane w dokumentach. W przypadku Firebase są one zapisywane w formacie JSON. Usługa ta oferuje zapytania na podstawie wartości pól dokumentu, synchronizację danych w czasie rzeczywistym, a także automatyczną replikację danych na wiele regionów. Dzięki listener'om, czyli słuchaczom, można zapewnić, aby informacje w aplikacji były aktualne dzięki powiadomieniom o zmianach w nasłuchiwanych danych i zamiast wszystkich danych, otrzymywać same nowe zmiany. Baza ta zapewnia również integrację z pozostałymi produktami wchodzącymi w skład platformy Firebase. Firestore został zaprojektowany tak, by łatwo się skalował - korzystając z wydajnej infrastruktury Google Cloud Platform może obsłużyć ruch nawet globalnych aplikacji, więc nie trzeba przejmować się tym problemem. Do komunikacji z Firestore wykorzystany zostanie JavaScript SDK - zestaw narzędzi programistycznych, dzięki którym jest możliwe wywoływanie funkcji odpowiedzialnych za wstawianie i pobieranie danych z bazy z kodu aplikacji.

5.4.2 Struktura

Jak wspomniano wcześniej Cloud Firestore przechowuje dane w dokumentach zawierających pola z wartościami. Dokumenty mogą przechowywać różne typy danych - od prostych ciągów znaków po skomplikowane, zagnieżdżone obiekty. Każdy dokument posiada swoje unikalne ID, dzięki któremu jest rozróżniany w bazie danych. ID może być nadane automatycznie przez bazę lub programista może ustawić je samemu przy tworzeniu dokumentu. Dokumenty muszą być przechowywane w kolekcjach, które są kontenerami na nie i których można używać do organizacji danych i tworzenia zapytań. Kolekcje są tworzone niejawnie - wystarczy zapisać dokument w nieistniejącej kolekcji, a zostanie ona automatycznie stworzona. Można także tworzyć podkolekcje w dokumentach i budować hierarchiczne struktury danych, które skalują się wraz ze wzrostem bazy danych. Cloud Firestore jest pozbawiony schematów, projektant ma całkowitą dowolność w decydowaniu jakie pola wstawić do każdego dokumentu i jakiego typu dane będą przechowywane w tych polach. Dokumenty w tej samej kolekcji mogą posiadać różną strukturę, jednakże dobrym pomysłem jest używanie tych samych pól i typów danych w tej samej kolekcji, by w łatwy sposób pobierać z nich dane. Firestore jest zoptymalizowany do przechowywania dużych kolekcji małych dokumentów. Pamiętając o tym oraz w celu łatwego dostępu do danych struktura projektowanej aplikacji będzie hierarchiczna. Kolekcją na najwyższym poziomie hierarchii będzie kolekcja „users". Będzie ona zawierać informacje o zarejestrowanych użytkownikach aplikacji. Każdy dokument użytkownika będzie posiadać kolekcję „friends" zawierającą dokumenty znajomych. Te z kolei będą zawierać kolekcję „debts", która będzie przechowywać długi między użytkownikiem, a danym znajomym. Diagram struktury bazy danych przedstawiono na rysunku 5.8.



Rysunek 5.8: Diagram struktury bazy danych

5.5 Architektura komponentów React Native

Wprowadzanie Cała aplikacja stworzona w React Native opiera się na komponentach pisanych w języku JSX. Komponent zawiera w sobie zarówno strukturę innych komponentów, którą definiuje się za pomocą znaczników podobnie do języka HTML, wygląd tych komponentów, który określa się za pomocą stylów podobnych do CSS oraz samo zachowanie komponentu kontrolowane przez funkcje JavaScript'owe. Komponent może, lecz nie musi posiadać pewien stan, który jest w nim przechowywany. W stanie przechowywane są informacje takie jak dane wprowadzone przez użytkownika do formularza, informacja o zaznaczonym lub nie polu wyboru albo lista znajomych pobrana z bazy danych. Komponenty przechowujące stan to komponenty stateful, natomiast te które go nie przechowują to komponenty stateless [9]. Do przekazywania danych między komponentami wykorzystuje się tzw. props. Są to dane, które można przekazać do komponentu dziecka, czyli tego, będącego częścią innego komponentu. Komponenty można tworzyć wieloma metodami. Najpopularniejszy z nich to tworzenie klasy JavaScript'owej (które pojawiły się w standardzie ECMAScript 6) rozszerzającej klasę Component. Dzięki wykorzystaniu JavaScript'owych arrow functions można tworzyć tzw. functional components, które nie są całą klasą, a jedynie funkcją. Z tego powodu mają one ograniczone możliwości - nie mogą przechowywać stanu, a także nie można korzystać z ich lifecycle hooks, ale za to nie pisze się kodu boilerplate, więc będą one stosowane tam gdzie będzie to możliwe. Komponenty można podzielić na różne sposoby, w niniejszej pracy przyjęto podział [8] na:

- komponenty prezentacyjne
- komponenty kontenerowe

Komponenty prezentacyjne Ten rodzaj komponentów skoncentrowany jest na tym jak wyglądają rzeczy w aplikacji. Najczęściej są to komponenty stateless, ponieważ nie

przechowują żadnych danych tylko wyświetlają te, które otrzymują od komponentu rodzica przez props. Nie określają one jak dane są ładowane lub przetwarzane.

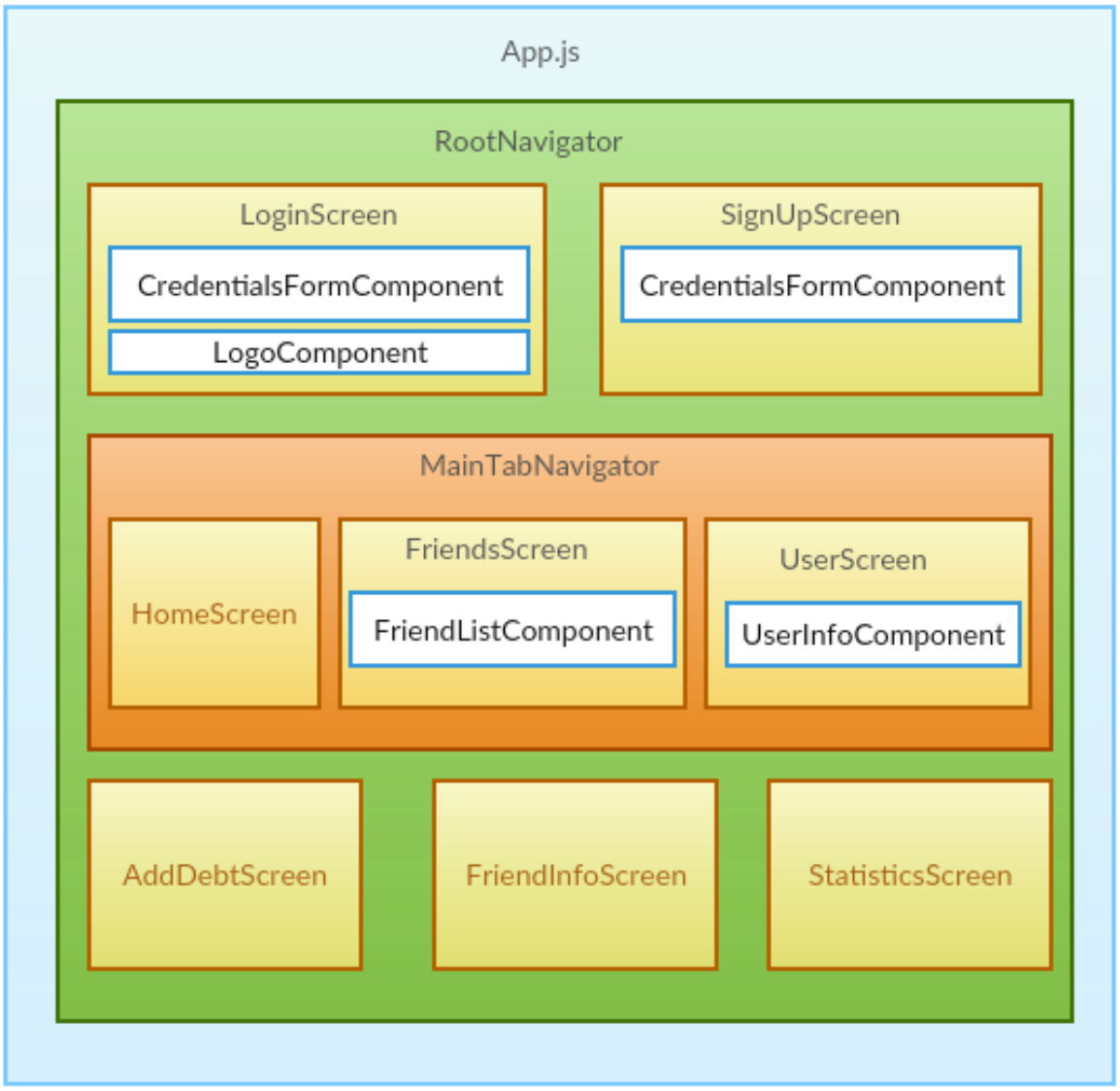
Komponenty kontenerowe Komponenty kontenerowe skoncentrowane są na tym jak pracują rzeczy w aplikacji. Odpowiedzialne są za logikę aplikacji, pobieranie danych z bazy oraz obsługę nawigacji po aplikacji. Tak jak podpowiada nazwa, stanowią one kontenery na komponenty prezentacyjne, którym przekazują dane do wyświetlenia przez props.

Nawigacja Nawigacja jest ważnym zagadnieniem, gdyż to dzięki niej użytkownik może poruszać się po aplikacji. Aktualnie istnieje wiele bibliotek do nawigacji w React Native, jednak jeszcze do niedawna nie było żadnej polecanej przez twórców frameworku. W prezentowanym rozwiązaniu będzie wykorzystana biblioteka zalecana w dokumentacji - React Navigation [17]. Opiera się ona na tzw. nawigatorach. Mogą być one różnego typu. Każdy nawigator może nawigować do zawartych w nim ekranów, które są komponentami.

Architektura komponentów aplikacji Na rysunku 5.9 przedstawiono diagram komponentów, na którym widać wszystkie komponenty aplikacji razem z nawigatorami. Po załadowaniu aplikacji widoczny zostaje ekran logowania z nawigatora root.

5.6 Podsumowanie

Na podstawie przypadków użycia widać jakie funkcje aplikacji muszą zostać zaimplementowane. Na ich podstawie wykonano projekt interfejsu użytkownika, dzięki któremu wiadomo jak rozmieścić ekrany i kontrolki, którymi będzie posługiwać się posiadacz aplikacji. Przy budowie interfejsu największą wagę przykładano do intuicyjności i minimalizmu. Baza danych NoSQL nie musi mieć zdefiniowanej struktury, jednak dla łatwego dostępu do danych zaproponowano ją i przedstawiono na diagramie. Zaproponowano również architekturę komponentów, która została zaprezentowana na diagramie. Na podstawie projektu dokonanego w niniejszym rozdziale można podjąć prace implementacyjne.



Rysunek 5.9: Diagram komponentów

Rozdział 6

Implementacja

6.1 Wprowadzenie

Aplikacja została zaimplementowana zgodnie z projektem z poprzedniego rozdziału oraz z dobrymi praktykami programowania [5]. Ze względu na konieczność zmian narzędzi programistycznych, które zostały zaplanowane w projekcie w następnym podrozdziale (6.2) opisano wykorzystane środowiska i narzędzia. Przedstawiono komunikację z bazą danych (6.4), a powstały interfejs aplikacji zaprezentowano w podrozdziale 6.3. Następnie opisano problemy (6.5), które pojawiły się w fazie implementacyjnej. Ponieważ wytworzonej aplikacji nie uruchamia się w sposób standardowy w podrozdziale 6.6 wyjaśniono sposób jej instalacji i uruchamiania.

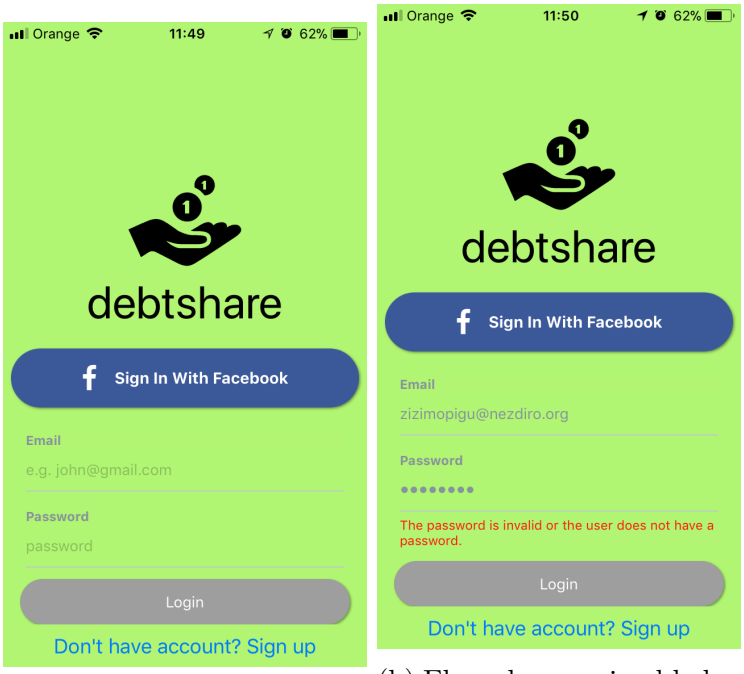
6.2 Wykorzystane środowiska i narzędzia programistyczne

Do zaimplementowania aplikacji wykorzystano zintegrowane środowisko programistyczne dla JavaScript'u Webstorm. Integruje się ono dobrze z narzędziem Flow do statycznej kontroli typów i podjęto próbę zastosowania typowania w kodzie aplikacji. Niestety w związku ze słabym wsparciem Flow do zewnętrznych zależności zrezygnowano z tego rozwiązania (miało to również swoje plusy [7]), co opisano w podsekcji 6.5.1. Zmiana nastąpiła również w wyborze menadżera pakietów (6.5.2) - zastosowano yarn [22] zamiast npm ze względu na niewłaściwą pracę tego drugiego pod systemem Windows.

6.3 Interfejs

Do wykonania interfejsu użytkownika wykorzystano biblioteki NativeBase [18] oraz React Native Elements [19]. Umożliwiają one wykorzystanie gotowych elementów takich jak przyciski, listy czy formularze. Do łatwego zaimplementowania modala (wysuwający się ekran będący nad innym) użyto zaś biblioteki react-native-modal [20].

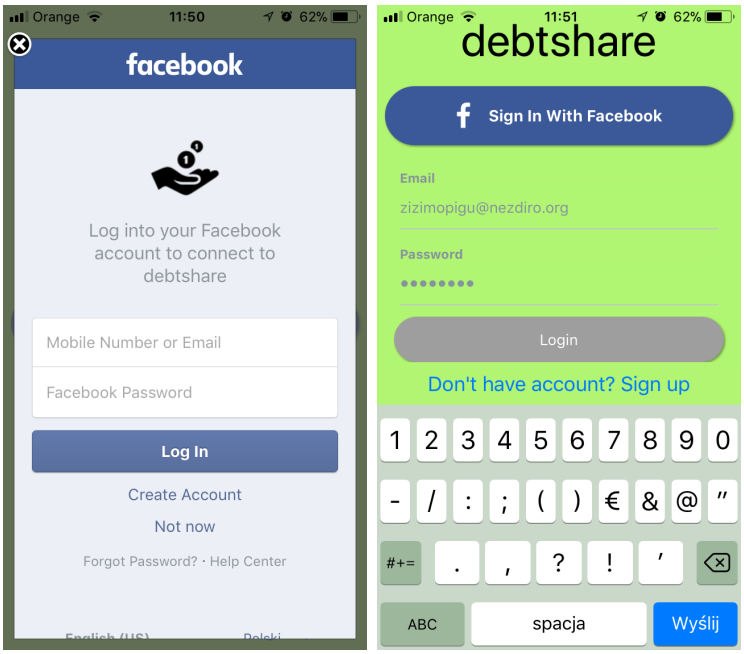
6.3.1 Ekran logowania



(a) Ekran logowania

(b) Ekran logowania - błędne hasło

Rysunek 6.1: Ekran logowania - obsługa błędnego hasła

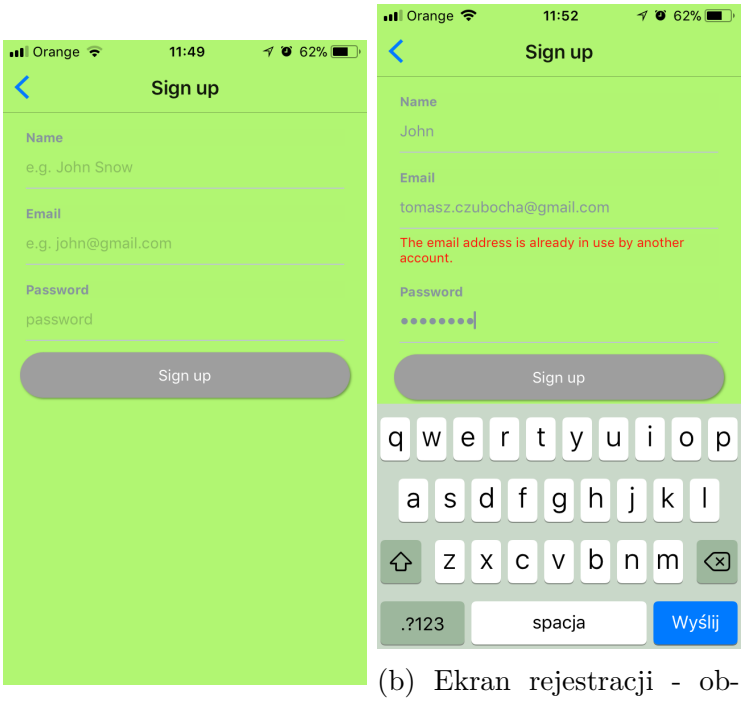


(a) Ekran logowania przez portal Facebook

(b) Ekran logowania - obsługa wysuniętej klawiatury

Rysunek 6.2: Ekran logowania - logowanie przez portal Facebook i obsługa wysuniętej klawiatury

6.3.2 Ekran rejestracji

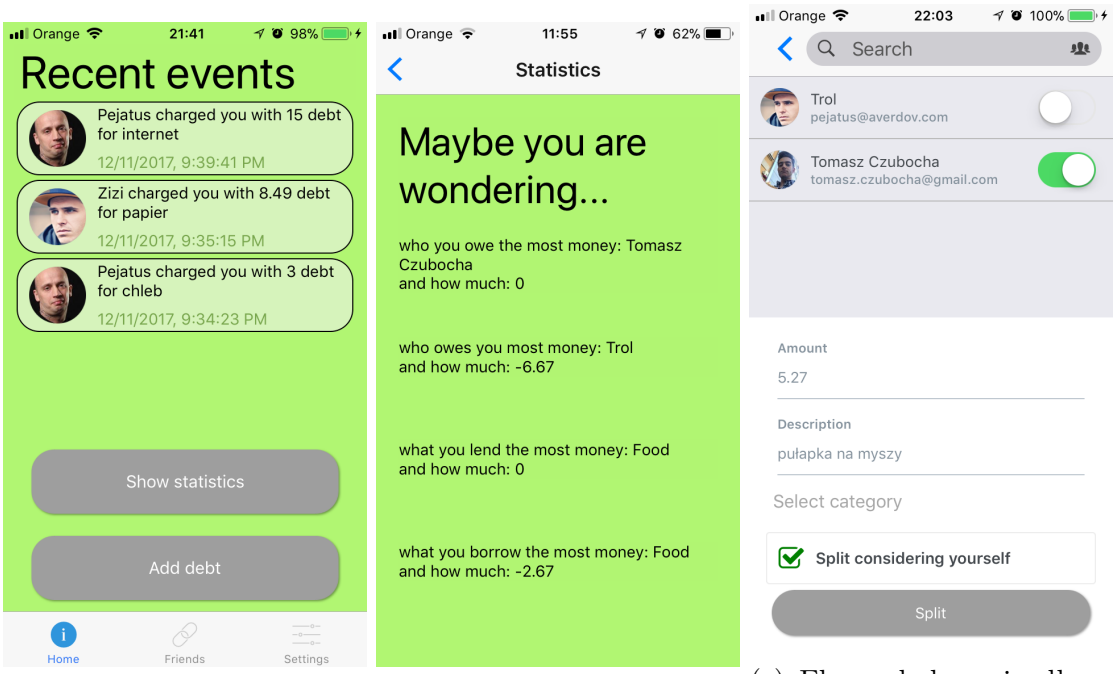


(a) Ekran rejestracji

(b) Ekran rejestracji - obsługa rejestracji z zarejestrowanym już email'em

Rysunek 6.3: Ekran rejestracji - obsługa błędu istniejącego konta

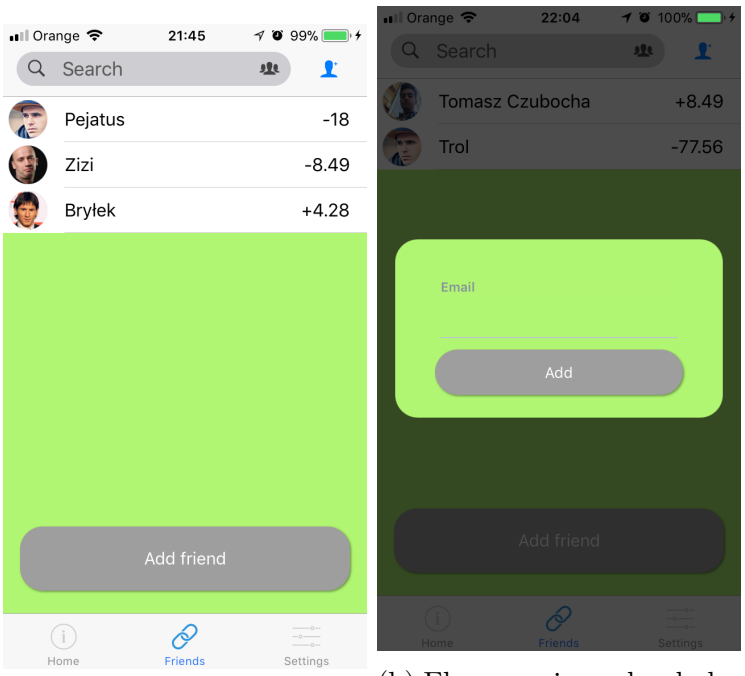
6.3.3 Ekran główny



(a) Ekran główny (b) Ekran statystyk (c) Ekran dodawania długu wielu osobom

Rysunek 6.4: Ekran główny, statystyk, dodawania długu wielu osobom

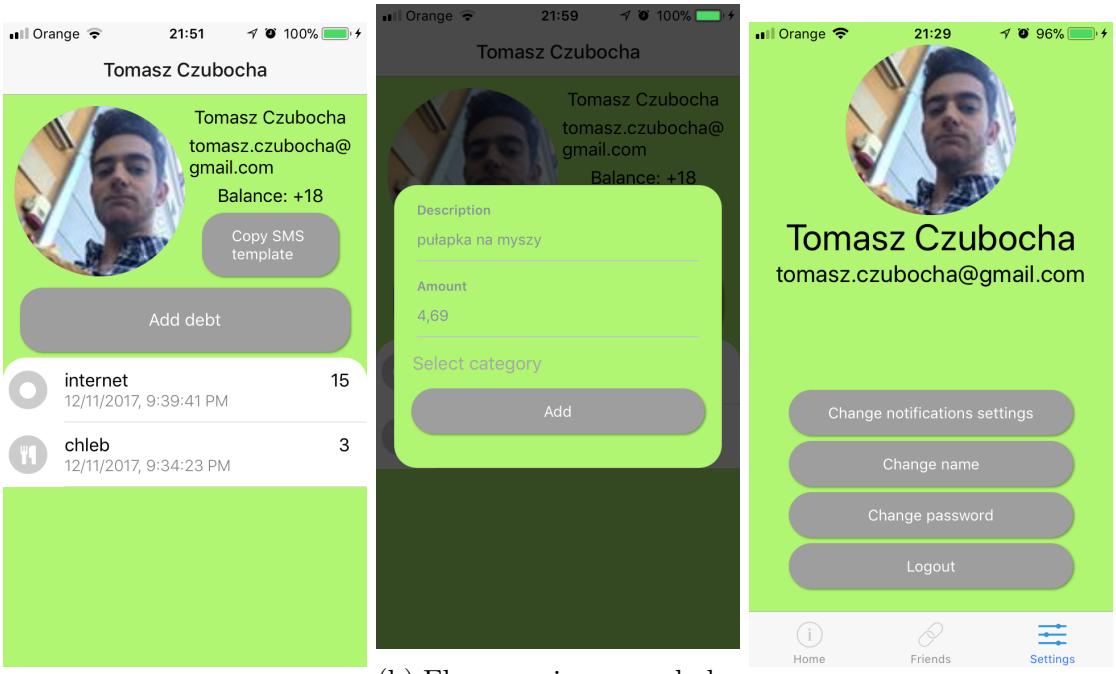
6.3.4 Ekran znajomych



(a) Ekran znajomych (b) Ekran znajomych - dodawanie znajomego

Rysunek 6.5: Ekran znajomych - lista i dodawanie znajomego

6.3.5 Ekran znajomego i ustawień



(a) Ekran znajomego

(b) Ekran znajomego - dodawanie długu

(c) Ekran ustawień

Rysunek 6.6: Ekran znajomych - ekran znajomego i ustawień

6.4 Komunikacja aplikacji z bazą danych

Baza danych Firestore oferuje dwa sposoby otrzymywania danych: wywołanie metody lub ustawienie słuchacza na zdarzenia zmiany danych. Podczas implementacji korzystano z drugiego rozwiązania, ponieważ oferuje ono szereg zalet. Przy jakichkolwiek zmianach w bazie danych, czyli np. przy operacjach dodawania, edycji czy usuwania długu po wykonanej czynności nie trzeba pamiętać o pobraniu na nowo danych z bazy. Ustawiając słuchacza backend powiadomi aplikację o zmianie w danych, które zostaną otrzymane automatycznie. Przy takim podejściu aplikacja aktualizuje się w czasie rzeczywistym dzięki czemu można pozbyć się wszelkich przycisków typu odśwież oraz widoczne informacje pojawiają się w momencie wykonanej czynności przez innego użytkownika np. gdy znajomy użytkownika doda dług, pojawia się on w czasie rzeczywistym na ekranie posiadacza aplikacji, bez jakiejkolwiek interakcji. Innym plusem jest możliwość pobierania tylko danych, które się zmieniły, a nie całego zapytania. Znakomicie ogranicza to transfer danych generowany przez program. Jedyne o czym należy pamiętać to wypisanie się z nasłuchiwania, ponieważ gdy się tego nie wykona dane nadal będą otrzymywane.

6.5 Problemy

6.5.1 Flow

Flow to narzędzie pozwalające na statyczną kontrolę typów w języku JavaScript. By skorzystać z korzyści oferujących to narzędzie należy oczywiście typować zmienne, które

definiuje się w kodzie. Problem pojawia się, gdy korzysta się z zewnętrznych bibliotek, ponieważ one nie są otypowane. Nic nie robiąc z taką sytuacją pojawia się ogrom błędów, które sprawiają, że rzeczywiste błędy wynikające z pomyłki programisty giną w gąszczu tych spowodowanych brakiem typów. Rozwiązania są dwa: albo samodzielne stworzenie interfejsu biblioteki z typami „any", albo pobranie otypowanego interfejsu zrobionego przez osobę trzecią. Pierwsze rozwiązanie jest proste, ale nie Flow w tym wypadku daje żadnych korzyści. Jeżeli chodzi natomiast o drugie rozwiązanie tu pojawia się sedno problemu, gdyż takich otypowanych interfejsów jest bardzo niewiele. To ten czynnik sprawia, że wielu programistów chcących pisać otypowany JavaScript wybiera TypeScript zamiast Flow, ponieważ TypeScript posiada bogate repozytorium definicji typów [25].

6.5.2 npm

Początkowo korzystając z npm jako menadżera pakietów nie było z nim problemu, jednak po pewnym czasie przy próbie instalacji jakiejkolwiek dependencji zaczął się zawieszać. Było to niezwykle uciążliwe. Ogólnie zachowanie npm można nazwać jako niestabilne. Po wyszukaniu problemu w sieci natrafiono na wątek na GitHub'ie z dyskusją o problemie [26]. Pokierowano się sugestią, by użyć innego menadżera o nazwie yarn. Po zapoznaniu się z porównaniami obu produktów, zaczęto korzystać z poleconego programu. Jego pliki konfiguracyjne są w pełni kompatybilne z npm, więc nie trzeba było zmieniać już utworzonych plików, wystarczyło zapoznanie się z innymi komendami konsolowymi.

6.5.3 React Navigation

React Navigation [17] jest w momencie pisania pracy polecaną biblioteką do nawigacji w React Native. Jednak nie oznacza to, że jest to biblioteka najlepsza, a tym bardziej idealna. Przekonano się o tym, gdy podczas implementacji zaszła potrzeba cofnięcia się z głębszego poziomu nawigacji do pierwszego ekranu aplikacji (przycisk wyloguj). Okazuje się, że nawigując do nawigatora, który znajduje się w innym nawigatorze nie można wyjść z niego w inny sposób niż przez komendę back powracającą o jedną akcję nawigacji wstecz. Po bezpośredniej rozmowie z jednym twórców React Native wyszło na jaw, że społeczność React Native ma duży problem z bibliotekami nawigacyjnymi i nie ma w tym momencie złotego środka. Rekomendowana w rozmowie biblioteka wykorzystuje natywne elementy systemu bezpośrednio co uniemożliwia korzystanie z niej w połączeniu z Expo. Z tego powodu zastosowano obejście problemu polegające na zapisaniu referencji do nawigatora najwyższego w strukturze (z którego można przejść do ekranu logowania), a następnie zaimportowaniu jej w ekranie ustawień, gdzie jest potrzebna do reakcji na naciśnięcie przycisku wylogowywania.

6.5.4 Firestore na systemie Android

Najpoważniejszy problem, bo uniemożliwiający poprawne działanie aplikacji na systemie Android powiązany był z bazą danych Firestore. Podczas testów na systemie Android żadne dane nie były pobierane z bazy, w momencie gdy na iOS aplikacja działała poprawnie. W czasie pisania pracy produkt ten jest w fazie beta, więc wszelkie błędy są bardziej zrozumiałe niż w innych produktach oznaczonych jako stabilne. Autor pracy będąc jednym z pierwszych osób dotkniętych błędem (publiczne wydanie wersji beta - 3.10.2017r.) potwierdził go na forum Expo, a następnie w założonym wątku na GitHub'ie [27]. Problem

okazał się leżeć po stronie Google. W momencie pisania pracy zostało podane tymczasowe obejście, które zostało zastosowane w aplikacji, by docelowo wyeliminować je przez podwyższenie wersji Firestore SDK, w której ma być już zaimplementowane rozwiązanie błędu [28].

6.5.5 Zapętlanie Cloud Functions

Cloud Functions to pojedyncze funkcje wdrożone na serwerach Google Cloud Platform, które są uruchamiane w standardowym środowisku Node.js. Mogą być one wyzwalane na kilka sposobów: przy wydarzeniach zapisu, edycji, usuwania z bazy danych, przy logowaniu użytkownika, przez wysłanie zapytania do endpoint'u i jeszcze inaczej. W przypadku niniejszego rozwiązania korzysta się m.in. z wyzwalacza na podstawie zapisu do Firestore. Użytkownik dodając dług znajomemu zapisuje go w lokalizacji w bazie w swoim profilu z kwotą ze znakiem dodatnim. Aby drugi użytkownik zobaczył dodany mu dług Cloud Function wyzwala się przy zapisie do lokalizacji users/userId/friends/friendId/debts/* i zapisuje dług z odwrotnym znakiem do users/friendId/friends/userId/debts/*. Jednak to zapisanie z kolei znowu wywołuje funkcję. Aby pozbyć się nieskończonej pętli zastosowano pole „rewritten" w dokumencie długu, które wskazuje czy dług został już przepisany przez funkcję chmurową. Pierwszy zapis jest dokonywany z wartością tego pola false i na początku wykonania funkcji sprawdzany jest warunek czy dług został przepisany - jeśli tak - należy ją zakończyć - jeżeli nie - wykonać zapis.

6.5.6 Zewnętrzne zapytania z Cloud Functions

Problem pojawił się, także z Cloud Functions przy próbie wysyłania powiadomień. Jak można przeczytać w cenniku Firebase [30], przy darmowym planie niestety nie można wysyłać zapytań do serwerów innych niż Google. Nie można wykorzystać zatem funkcji w chmurze do notyfikacji push np. przy zapisie, edycji lub usunięciu długu z bazy danych, lecz trzeba wysyłać zapytanie do serwerów Expo bezpośrednio z aplikacji. Zaimplementowano takie rozwiązanie za pomocą Fetch API obecnym w React Native używając żądania POST z danymi powiadomienia.

6.6 Instalacja i uruchomienie aplikacji

By umożliwić wygodne wypróbowanie aplikacji wdrożono ją w trybie publicznym na serwery Expo. Wykorzystano do tego narzędzie o identycznej nazwie. Działanie jest następujące: przez sieć CDN (Content Delivery Network) uzyskuje się „wiązkę" z JavaScript'owym kodem aplikacji, za której pomocą w kliencie Expo jest renderowana aplikacja odpowiednio na system Android lub iOS. Klient ten jest oczywiście dostępny zarówno w Google Play jak i App Store skąd należy go pobrać i zainstalować. Następnie po kliknięciu w znak plusa w kliencie, oznaczający dodanie nowego projektu, trzeba wpisać jako URL `exp://exp.host/@czubocha/debtshare`. Opcjonalnie, mając dostęp do komputera lub innego urządzenia poza smartfonem, wybierając odpowiednią opcję w kliencie Expo, można zeskanować kod QR znajdujący się na stronie projektu [29]. Po otwarciu projektu lub bezpośrednio po zeskanowaniu QR kodu „wiązka" aplikacji zostanie pobrana i aplikacja zostanie uruchomiona.

6.7 Podsumowanie

Dla kogoś nowego zarówno w pracy z framework'iem React jak i React Native implementacja opisywanej w projekcie aplikacji nie jest prostym zadaniem. Oprócz zrozumienia działania platformy należy również dużo doszkolić się na temat przechowywania stanu komponentów, wymiany danych między nimi oraz o nawigacji między ekranami. Dobrym wyborem okazał się wybór architektury serverless, gdyż za pomocą platformy Firebase niewielkim kosztem wzbogacono aplikację o dużą wartość dodaną. Dużo czasu można z kolei stracić na próbach integracji z narzędziami mającymi za zadanie ulepszyć proces kodowania aplikacji, a nie zawsze wysiłek w to wkładany jest opłacalny, szczególnie przy niewielkich projektach. Podsumowując, posiadając nawet podstawowe umiejętności, w połączeniu z nieskomplikowaną platformą Firebase można stworzyć dobrze wyglądającą, wydajną aplikację o zaawansowanych funkcjach znanych z dużych serwisów.

Rozdział 7

Podsumowanie pracy

7.1 Wykonane prace

W ramach niniejszej pracy szczegółowo przeanalizowano oraz opisano problem zapamiętywania długów pieniężnych wobec innych osób oraz niezwróconych należności dla użytkownika. Zgłębiono się w istniejące produkty próbujące rozwiązać omawiane zagadnienie i skategoryzowano je. Dokonano przeglądu przydatnych technologii i technik zarówno dla aplikacji klienckiej jak i dla backend'u rozwiązania. W jego zakresie rozpatrzono dostępne framework'i pozwalające pisać aplikacje wieloplatformowe oraz zalety i wady różnych typów klientów i architektur systemów mobilnych. Ustalono założenia projektowe - określono przedmiot prac, sformułowano wymagania funkcjonalne i niefunkcjonalne, opisano podstawową architekturę systemu oraz wybrano sposób realizacji - technologie, narzędzia i środowisko programistyczne. Sporządzono projekt aplikacji zawierający przypadki użycia wraz z diagramem, koncepty interfejsu użytkownika, opis wybranej bazy danych i jej struktury, a także diagram architektury komponentów używanych do zaprogramowania aplikacji. W ramach implementacji wytworzono w pełni funkcjonujące rozwiązanie działające na systemie Android i iOS. Zaprezentowano interfejs zgodny z projektem, opisano sposób komunikacji z bazą danych, powstałe problemy i ich rozwiązania jak i również metodę instalacji i uruchomienia aplikacji.

7.2 Realizacja celu pracy

Cel pracy został zrealizowany. Oprócz aplikacji na system Android stworzono jej odpowiednik na system iOS. Użytkownik dzięki powstałemu produktowi zwolniony jest z zapamiętywania długów swoich oraz niezwróconych należności swoich dłużników. Może on również w wygodny sposób dzielić kwotę zakupów na wielu znajomych. Przy każdej akcji otrzymuje powiadomienie push. Udało wykonać się także cel poboczny pracy, czyli wykonanie rozwiązania problemu niskim kosztem programistycznym tj. ilością własnego

kodu przez użycie najnowszych technologii pozwalających na wielokrotne wykorzystanie jednego kodu i użycie gotowych usług zapewniających zaawansowane funkcje. Stało się to możliwe dzięki zastosowaniu platformy React Native tworzącej aplikacje na dwa systemy operacyjne z jednego kodu oraz platformy Firebase udostępniającej usługi Backend as a Service, które zapewniły uwierzytelnianie użytkowników, skalowalną, elastyczną bazę danych oraz Cloud Functions jako zastępstwo tradycyjnego serwera.

7.3 Wnioski

Użyty stos technologiczny pozwala na zbudowanie zaawansowanego produktu w postaci szeroko dostępnej aplikacji mobilnej wykorzystującej gotowe usługi wydajnie działające zarówno dla niewielkich projektów jak i tych bardzo dużych, dzięki całkowicie automatycznemu, niewymagającemu ingerencji programisty skalowaniu. React Native powstał na bazie jego webowego odpowiednika - React'a, więc szczególnie proste będzie wdrożenie się w takie rozwiązanie programistom webowym. Jednakże również osoby spoza tej dziedziny z powodzeniem mogą tworzyć produkty podobne do zaprezentowanego w niniejszej pracy - przykładem może być sam jej autor, który na co dzień nie zajmuje się ani front-end'em czy aplikacjami mobilnymi, a back-end'em w języku Java. Przedstawiony i rozwiązany w tymże tekście problem był tylko przykładowym - jak pokazano - te dwie technologie to doprawdy efektywne pod względem kosztu wytworzenia i efektowne zarazem połączenie, potrafiące stworzyć zaawansowany produkt borykający się z o wiele trudniejszymi zagadnieniami.

7.4 Sugestie odnośnie dalszych prac

By uczynić powstałe rozwiązanie lepszym można dodać szereg usprawnień czyniąc rozliczanie długów wśród grupy znajomych jeszcze łatwiejszym. Z pewnością byłby to moduł płatności, który umożliwiałby natychmiastowe wyrównanie długu, gdy stanie się zbyt duży lub gdy druga osoba o takowy poprosi. Innym pomysłem jest algorytm upraszczający rozliczanie na podstawie grafu utworzonego między utworzoną grupą wspólnych znajomych, który działałby na podstawie następującego przykładu: gdy Janek jest dłużny Agacie 10 zł, a Agata jest dłużna Adamowi 10 zł to z dwóch długów tworzy się jeden - Janek jest dłużny Adamowi 10 zł. Ciekawą funkcją aplikacji byłaby także możliwość udostępniania zdjęć paragonów w celu uwiarygodnienia listy i kwoty zakupów. Usługą Firebase, która idealnie nadawałaby się do jej zaimplementowania jest Cloud Storage, które jest miejscem dyskowym dostępnym w chmurze. Na chwilę obecną jednak nie jest możliwe wykorzystanie jej razem z React Native z powodu braku implementacji typu Blob (Binary Large OBject), ale zgodnie z zapowiedziami twórców framework'a ma się to zmienić w najbliższej przyszłości.

Bibliografia

[1] Masiello, E., Friedmann, J. (2017). *Mastering React Native*. Packt Publishing.

[2] Prusty, N. (2015). *Learning ECMAScript 6*. Packt Publishing.

[3] Moroney, L. (2017). *The Definitive Guide to Firebase: Build Android Apps on Google's Mobile Platform*. Apress.

[4] Pilone, D. (2003). *UML Pocket Reference*. O'Reilly.

[5] Martin, R. (2008). *Clean Code* (1st ed.). Prentice Hall.

[6] Cruxlab Inc. (2017). *Xamarin vs Ionic vs React Native: differences under the hood*. https://medium.com/swlh/xamarin-vs-ionic-vs-react-native-differences-under-the-hood-6b9cc3d2c826, dostęp 2.12.2017.

[7] Elliott, E. (2016). *You Might Not Need TypeScript (or Static Types)*. https://medium.com/javascript-scene/you-might-not-need-typescript-or-static-types-aa7cb670a77b, dostęp 2.12.2017.

[8] Abramov, D. (2015). *Presentational and Container Components*. https://medium.com/@dan_abramov/smart-and-dumb-components-7ca2f9a7c7d0, dostęp 2.12.2017.

[9] Jefferson, D. (2016). *Stateless components in React Native*. https://medium.com/front-end-hacking/stateless-components-in-react-native-e9034f2e3701, dostęp 2.12.2017.

[10] Roberts, M. (2016). *Serverless Architectures*. https://martinfowler.com/articles/serverless.html, dostęp 2.12.2017.

[11] Williams, M. (2017). *How serverless changes application development*. https://search.proquest.com/docview/1906407616, dostęp 2.12.2017.

[12] Maruti Techlabs. *Serverless architecture the future of business computing*. https://www.marutitech.com/serverless-architecture-business-computing, dostęp 2.12.2017.

[13] Avram, A. (2016). *FaaS, PaaS, and the Benefits of the Serverless Architecture*. https://www.infoq.com/news/2016/06/faas-serverless-architecture, dostęp 2.12.2017.

[14] *Dokumentacja Expo*. https://docs.expo.io, dostęp 2.12.2017.

[15] *Dokumentacja Firebase*. https://firebase.google.com/docs, dostęp 2.12.2017.

[16] *Dokumentacja React Native*. https://facebook.github.io/react-native/docs/getting-started, dostęp 2.12.2017.

[17] *Dokumentacja React Navigation*. https://reactnavigation.org/docs, dostęp 2.12.2017.

[18] *Dokumentacja NativeBase*. https://docs.nativebase.io, dostęp 2.12.2017.

[19] *Dokumentacja React Native Elements*. https://react-native-training.github.io/react-native-elements, dostęp 2.12.2017.

[20] *Dokumentacja react-native-modal*. https://github.com/react-native-community/react-native-modal, dostęp 2.12.2017.

[21] *Dokumentacja npm*. https://docs.npmjs.com, dostęp 2.12.2017.

[22] *Dokumentacja yarn*. https://yarnpkg.com/en/docs, dostęp 2.12.2017.

[23] *Dokumentacja ESLint*. https://eslint.org/docs/user-guide/getting-started, dostęp 2.12.2017.

[24] *Dokumentacja Flow*. https://flow.org/en/docs, dostęp 2.12.2017.

[25] *Repozytorium DefinitelyTyped*. http://definitelytyped.org, dostęp 2.12.2017.

[26] *Dyskusja nt. błędu npm*. https://github.com/npm/npm/issues/13729, dostęp 2.12.2017.

[27] *Dyskusja nt. działania Firestore na systemie Android*. https://github.com/firebase/firebase-js-sdk/issues/283, dostęp 2.12.2017.

[28] *Kod naprawiający działanie Firestore na systemie Android*. https://github.com/google/closure-library/commit/9d6c2de7549e0211a822cb94b8b0eb59007d7ec5, dostęp 2.12.2017.

[29] *Strona projektu opublikowanego w Expo*. https://expo.io/@czubocha/debtshare, dostęp 2.12.2017.

[30] *Cennik Firebase*. https://firebase.google.com/pricing, dostęp 2.12.2017.