



Politechnika Wrocławskie

Wydział Informatyki i Zarządzania

kierunek studiów: informatyka

specjalność: brak

Praca dyplomowa – inżynierska

Aplikacja wspierająca inwestora w budowie domu jednorodzinnego

Jakub Jaszczuk

słowa kluczowe:

Aplikacja

Budowa

Android

Celem pracy jest przygotowanie aplikacji wspierającej decyzje podejmowane przez inwestorów na poszczególnych etapach budowy domu jednorodzinnego. Praca obejmuje analizę konkurencyjnych oraz zbliżonych rozwiązań, projekt i implementację aplikacji na urządzenia z systemem operacyjnym Android, przedstawienie problemu jakim jest finansowanie budowy domu jednorodzinnego.

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

pieczętka wydziałowa

Wrocław
2020

Streszczenie

Współcześnie ludzie coraz częściej budują własne domy. Jednocześnie koszty takiej inwestycji sukcesywnie rosły na przestrzeni ostatnich kilku lat, co powoduje że budowa własnego domu jest coraz bardziej problematyczna. W celu odpowiedzi na te wyzwania zaproponowano rozwiążanie programowe, które ma na celu wspomóc inwestora w budowie domu jednorodzinnego. Niniejsza praca przedstawia istotne statystyki związane z budową własnego domu, przegląd istniejących rozwiązań, propozycję nowej aplikacji oraz podsumowanie. W ramach pracy stworzono szczegółowy projekt aplikacji i bazy danych. Przedstawiono i zaimplementowano interfejsy wraz z realizowanymi przez nie przypadkami użycia. W rozwoju aplikacji istotny udział miały też jej testy. Przygotowana aplikacja może być wykorzystana przy budowie domu oraz w innych zbliżonych zastosowaniach.

Abstract

Nowadays, people more and more often build their own homes. At the same time, the costs of such an investment have been steadily increasing over the last few years, which means that building your own home is becoming more and more problematic. In order to respond to these challenges, a software solution has been proposed, which aims to support the investor in the construction of a single-family house. This paper presents important statistics related to the construction of one's own home, an overview of existing solutions, a proposal for a new application and a summary. As part of the paper, a detailed project of the application and database was created. Interfaces have been presented and implemented along with their use cases. Tests also played an important role in the development of the application. The prepared application can be used in the building of the house and in other similar applications.

Spis treści

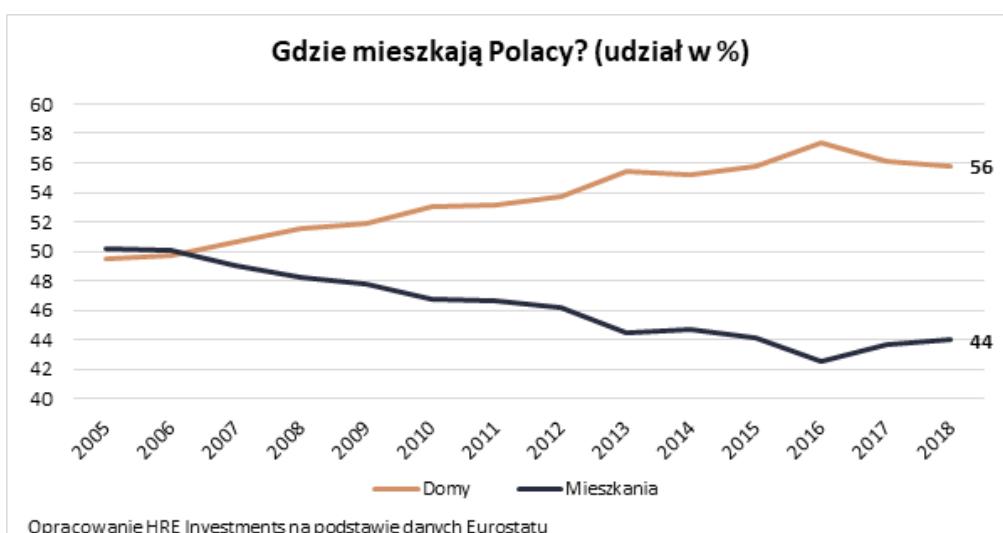
1. Wprowadzenie.....	2
1.1. Wstęp.....	2
1.2. Cel pracy.....	2
1.3. Przegląd rozdziałów.....	3
1.4. Przegląd istniejących rozwiązań.....	3
2. Projekt aplikacji.....	4
2.1. Analiza dziedziny problemowej.....	4
2.2. Specyfikacja wymagań.....	6
2.3. Model przypadków użycia.....	9
2.4. Specyfikacja przypadków użycia.....	10
2.5. Architektura logiczna i fizyczna.....	14
2.6. Model informacyjny.....	15
2.7. Interfejs użytkownika.....	19
2.8. Projekt bazy danych.....	24
2.9. Ideowe przedstawienie interakcji.....	25
3. Implementacja aplikacji.....	29
3.1. Wykorzystana technologia.....	29
3.1.1. Android Studio.....	29
3.1.2. Kotlin.....	31
3.1.3. SQLite.....	31
3.2. Implementacja dostępu do danych.....	31
3.3. Interfejs.....	33
3.4. Inne szczegóły implementacji.....	43
3.5. Testy.....	45
4. Zakończenie.....	48
4.1. Perspektywy rozwojowe aplikacji.....	48
4.2. Podsumowanie.....	48

1. Wprowadzenie

Niniejszy rozdział stanowi krótkie przedstawienie treści pracy, tematyki problemowej oraz omówienie proponowanego rozwiązania programowego.

1.1. Wstęp

Tematem niniejszej pracy inżynierskiej jest projekt i implementacja aplikacji wspomagającej inwestora w budowie domu jednorodzinnego. W dzisiejszych czasach może się okazać to szczególnie przydatnym oprogramowaniem. Według danych GUNB¹ w ostatnich latach mamy do czynienia ze wzrostem liczb wydanych pozwoleń na budowę budynków jednorodzinnych. W 2018 r. wydano ich 93714, czyli więcej o 3% względem roku 2017, w którym wydano ich 90968, i o 13,1% więcej niż w roku 2016, w którym wydano 82852 pozwoleń[1].



Rys. 1: Wykres przedstawiający rodzaj lokalu mieszkalnego w Polsce. Źródło: [2]

Ciekawym faktem jest to, że obecnie ponad połowa Polaków mieszka w zabudowie jednorodzinnej[2], co obrazuje wykres widoczny na Rys. 1. W przeciągu ostatnich 10 lat nastąpił znaczny wzrost liczby Polaków mieszkających w domach, podczas gdy w Europie widzi się trend odwrotny. Kolejnym faktem obrazującym przydatność proponowanej aplikacji jest ciągły znaczny wzrost kosztów budowy domu widoczny na przestrzeni ostatnich kilku lat. Od 2016 r. do 2019 r. ceny wzrosły o około 20%, co przy tak dużej i kosztownej inwestycji przekłada się na znaczne sumy pieniędzy[3]. Powyższe dane przedstawiają, że aplikacja pomagająca w zarządzaniu kosztami budowy domu może okazać się przydatna.

1.2. Cel pracy

Celem pracy jest przygotowanie aplikacji wspierającej decyzje podejmowane przez inwestorów na poszczególnych etapach budowy domu jednorodzinnego. Praca obejmuje

1 GUNB – Główny Urząd Nadzoru Budowlanego

analizę konkurencyjnych oraz zbliżonych rozwiązań, projekt i implementację aplikacji na urządzenie z systemem operacyjnym Android, przedstawienie problemu jakim jest finansowanie budowy domu jednorodzinnego.

1.3. Przegląd rozdziałów

Praca składa się z czterech rozdziałów, odpowiednio: wprowadzenie, projekt, implementacja, zakończenie. Pierwszy rozdział ma na celu wprowadzenie czytelnika w tematykę problemu jakim jest budowa domu jednorodzinnego. Ponadto krótko przedstawia ogólną strukturę pracy oraz konkurencyjne rozwiązania dostępne na rynku. Rozdział drugi przedstawia projekt proponowanej aplikacji. Kolejne jego podrozdziały skupiają się na poszczególnych etapach projektowania oprogramowania. Rozdział trzeci jest poświęcony omówieniu sposobu implementacji proponowanego rozwiązania programowego. Znajduje się w nim przedstawienie wykorzystanych narzędzi oraz zademonstrowanie działania aplikacji. Ostatni rozdział stanowi podsumowanie całości wykonanej pracy i pokazanie możliwych dalszych kierunków rozwoju aplikacji.

1.4. Przegląd istniejących rozwiązań

Obecnie na rynku jest dostępnych kilka rozwiązań przydatnych w zarządzaniu kosztami budowy domu. Jednym z nich jest kalkulator ze strony kb.pl[4]. Pozwala on na przeprowadzenie obliczeń kosztu domu. Użytkownik wybiera jeden z przygotowanych rodzajów domu, a następnie dodaje potrzebne pozycje z listy możliwych wydatków podzielonych na kategorie. Na pochwałę zasługuje również to, że są podane orientacyjne ceny, dzięki czemu nie trzeba wszystkiego samemu wprowadzać. Niestety aplikacja nie jest elastyczna i nie pozwala na wprowadzanie dowolnych danych. Ponadto wersja mobilna jest pełna reklam, a często nawet pełnoekranowych, co skutecznie odstrasza i zniechęca do korzystania z kalkulatora. Innym, problemem jaki może pojawić się na urządzeniu mobilnym jest niewłaściwe wyświetlanie treści strony, nachodzenie na siebie elementów. Kolejnym możliwym rozwiązaniem jest aplikacja mobiDOM[5], która jest dostępna na urządzenia mobilne z systemem Android lub iOS. Aplikacja zawiera bazę przydatnych informacji, przewodnik budowlany, opcję dziennika budowy, możliwość dodawania kontaktów. Istotnym elementem jest również grywalizacja, ponieważ za używanie aplikacji dostaje się punkty. Niestety mobiDOM nie posiada kalkulatora kosztów budowy, aplikacja nie jest zbyt popularna oraz od dawna nie została wydana żadna aktualizacja. Kolejnym rozwiązaniem może okazać się najprostszy arkusz kalkulacyjny. Zapewnia on bardzo dużą elastyczność w dodawaniu nowych pozycji, a ponadto wykonywanie obliczeń za jego pomocą jest bardzo proste. Jednak takie rozwiązanie ma też swoje wady. Przede wszystkim nie jest to rozwiązanie dedykowane do rozważanego problemu, przez co użytkownik może mieć problem z takim wykorzystaniem arkusza. Ponadto użytkownik może chcieć aplikację prostszą w użyciu i robiącą wszystko, co potrzeba, przy jak najmniejszym wkładzie własnym.

Aplikacja realizowana w ramach tej pracy ma odpowiadać na powyższe wady i łączyć najlepsze cechy wymienionych rozwiązań. Musi posiadać czytelną listę kosztów, możliwość dodawania dowolnych wydatków, które aplikacja będzie automatycznie podliczać, oraz zawierać porady dotyczące budowy domu.

2. Projekt aplikacji

Niniejszy rozdział przedstawia projekt proponowanego rozwiązania programowego z uwzględnieniem sztuki projektowania oprogramowania. Podrozdziały zostały ułożone w kolejności, która powinna pozwolić na łatwe zrozumienie projektu aplikacji.

2.1. Analiza dziedziny problemowej

Proces projektowania oprogramowania należy zacząć od analizy dziedziny problemowej oraz krótkiego przedstawienia co chcielibyśmy osiągnąć. Produktem, który ma powstać w wyniku procesu projektowania i implementacji jest aplikacja na urządzenia mobilne z systemem Android, ze szczególnym uwzględnieniem telefonów komórkowych, ale powinna działać bez problemów na m.in. tabletach. W treści tej pracy dziedziną problemową jest budowa domu jednorodzinnego oraz wsparcie inwestora w realizacji takiej inwestycji. W związku z tak przedstawioną dziedziną istotnymi bytami biznesowymi będą dom oraz wydatek, który musi ponieść inwestor czyli użytkownik aplikacji. Na podstawie dalszej analizy postanowiono wyróżnić następujące byty:

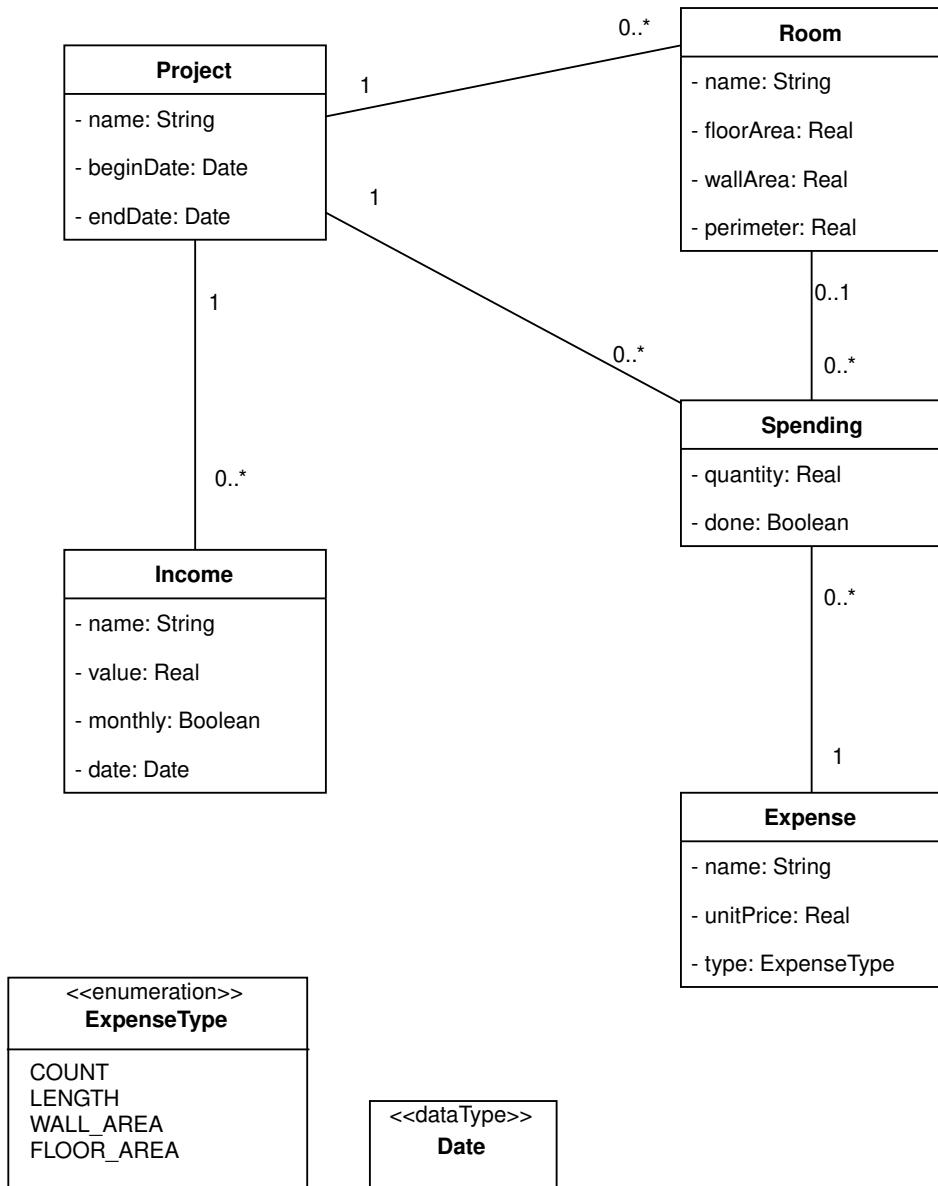
- **Projekt** – agregat zbierający wszystkie dane związane z domem, jego właściwościami, wydatkami. Użytkownik powinien mieć możliwość utworzenia wielu projektów. Taka właściwość może być przydatna w celu przedstawienia kilku różnych możliwych realizacji tego samego domu lub kiedy użytkownik buduje kilka domów jednocześnie.
- **Pomieszczenie** – aplikacja powinna wspierać inwestora nie tylko na etapie budowy bryły budynku ale też w pracach wykończeniowych i w umeblowaniu domu. Dlatego podział domu na pomieszczenia jest sensownym wyborem.
- **Przychód** – użytkownik powinien mieć możliwość podania spodziewanych przychodów w trakcie trwania realizacji inwestycji oraz ilości posiadanych środków finansowych. Dzięki tym informacjom aplikacja może przykładowo obliczyć spodziewany termin zakończenia prac albo zaalarmować kiedy środki nie będą wystarczające.
- **Wydatek** – jedną z podstawowych funkcjonalności aplikacji jest kontrola wydatków, dlatego musi mieć ona możliwość zbierania informacji o kosztach realizacji inwestycji od użytkownika. Jako że powinna być łatwa możliwość wielokrotnego dodania tego samego kosztu postanowiono wydzielić kolejny byt. Przykładem przydatności wyodrębnienia może być możliwość dodania tych samych paneli podłogowych do kilku pomieszczeń.
- **Wydatek globalny** – byt ten przedstawia pojedynczą rzecz albo usługę jaką można uznać za wydatek.

Po zdefiniowaniu bytów i krótkim określeniu do czego służą i w jakim celu zostały wyodrębnione należy je powiązać z regułami biznesowymi wynikającymi z badanej dziedziny oraz specyfiki aplikacji:

1. Każdy byt musi być łatwo identyfikowalny przez użytkownika aplikacji, dlatego musi posiadać nazwę.
2. Inwestycja jest realizowana w pewnych ustalonych przez użytkownika ramach czasowych.
3. Data rozpoczęcia nie może być późniejsza niż data zakończenia.

4. Każde pomieszczenie posiada ustalone wymiary.
5. Wymiary pomieszczenia nie mogą być ujemne.
6. Każdy przychód posiada nieujemną wartość pieniężną.
7. Każdy przychód może być cykliczny lub jednorazowy z określonym terminem zasilenia budżetu przeznaczonego na realizację inwestycji.
8. Pomieszczenie musi być częścią domu.
9. Wydatek może być przypisany do pomieszczenia lub do całości domu.
10. Przychód musi być powiązany z projektem.
11. Wydatek może być w stanie zrealizowanym lub nie.
12. Jeżeli ilość wydatku nie może zostać obliczona na podstawie wymiarów pomieszczenia to należy ją podać.
13. Ilość nie może być ujemna.
14. Wydatek musi posiadać odpowiadający mu wydatek globalny.
15. Wydatek globalny musi mieć podaną cenę jednostkową i swój rodzaj.

Na podstawie powyższych analiz dziedziny można pokazać następujący diagram domenowy.



Rys. 2: Model domenowy powstały w wyniku analizy wycinka rzeczywistości jakim jest budowa domu.

Na Rys. 2 przedstawiono w formie diagramu klas wymienione wcześniej byty, zależności pomiędzy nimi oraz dane jakie zawierają. Wyliczenie *ExpenseType* określa rodzaj wydatku globalnego. Ze względów programistycznych stosowane są nazwy angielskie, gdzie *Spending* to wydatek, a *Expense* reprezentuje wydatek globalny. Reszta nazw jest raczej zrozumiała.

2.2. Specyfikacja wymagań

Kolejnym etapem przygotowania projektu aplikacji jest analiza wymagań. Należy poznać wymagania oraz oczekiwania potencjalnego użytkownika finalnego produktu. Wymagania funkcjonalne związane z tworzoną aplikacją przedstawiają się następująco:

1. Użytkownik może dodać nowy projekt.

2. Użytkownik może edytować istniejący projekt.
3. Użytkownik może usunąć istniejący projekt.
4. Użytkownik może przeglądać listę istniejących projektów.
5. Użytkownik może przeglądać szczegółowe dane istniejących projektów.
6. Użytkownik może dodać nowe pomieszczenie do projektu.
7. Użytkownik może edytować istniejące pomieszczenie.
8. Użytkownik może usunąć istniejące pomieszczenie.
9. Użytkownik może przeglądać listę pomieszczeń dla wybranego projektu.
10. Użytkownik może dodać nowy przychód do projektu.
11. Użytkownik może usunąć istniejący przychód.
12. Użytkownik może edytować istniejący przychód.
13. Użytkownik może przeglądać listę przychodów dla wybranego projektu.
14. Użytkownik może dodać nowy wydatek do projektu.
15. Użytkownik może dodać nowy wydatek do wybranego pomieszczenia.
16. Użytkownik może edytować istniejący wydatek.
17. Użytkownik może usunąć istniejący wydatek.
18. Użytkownik może zmienić status istniejącego wydatku.
19. Użytkownik może przeglądać listę wydatków dla wybranego projektu.
20. Użytkownik może dodać nowy wydatek globalny.
21. Użytkownik może usunąć istniejący wydatek globalny.
22. Użytkownik może edytować istniejący wydatek globalny.
23. Użytkownik może przeglądać listę wydatków globalnych.

Powyższa lista wymagań składa się głównie z powtarzania dodaj, usuń, edytuj i przeglądaj. Są to bardzo ważne operacje z punktu widzenia aplikacji, ponieważ użytkownik musi mieć możliwość dowolnego manipulowania danymi wprowadzanymi przez siebie. Dostęp do tych wszystkich operacji powinien być łatwy i intuicyjny dla użytkownika. Lista wymagań funkcjonalnych stanowi trzon funkcjonalności aplikacji powstającej w ramach niniejszej pracy. Z kolei wymagania niefunkcjonalne są następujące:

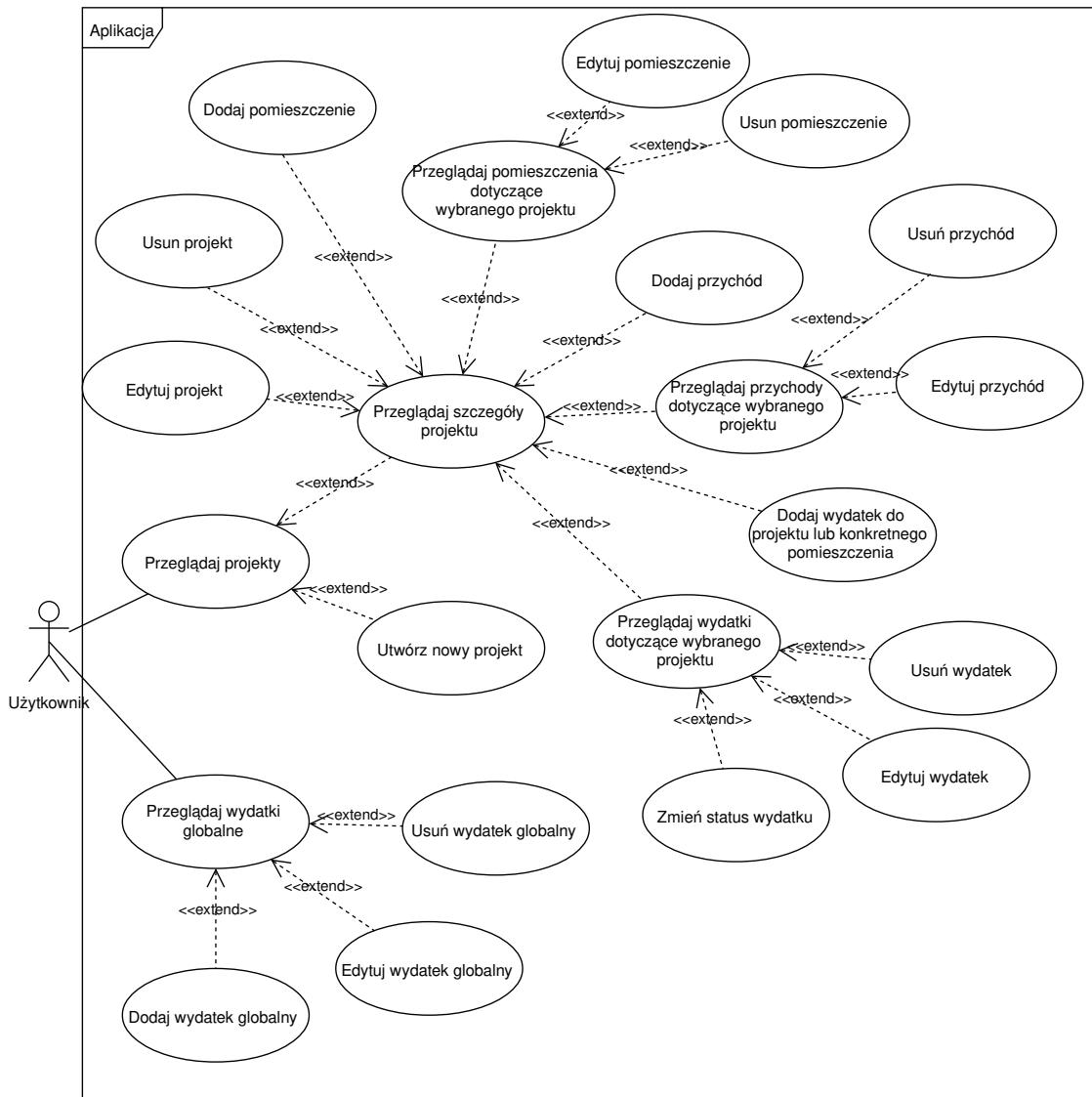
1. Aplikacja działa responsywnie.
2. Interfejs aplikacji jest dostosowany do wymiarów typowych urządzeń przenośnych.
3. Aplikacja jest dostępna w języku polskim.
4. Czas potrzebny na uruchomienie aplikacji jest mały.
5. Aplikacja posiada przejrzysty interfejs.
6. Okna aplikacji są proste w nawigacji.
7. Interakcje z elementami interfejsu są jasno określone.
8. Aplikacja powinna być przyjazna osobom niepełnosprawnym lub z innymi trudnościami.
9. Aplikacja działa na urządzeniach z systemem operacyjnym Android 4.0 lub nowszym.

Urządzenia mobilne mają swoją charakterystykę, na którą trzeba zwrócić uwagę tworząc na nie aplikacje[6]. Do głównych cech należą wielkość ekranu oraz mniejsza dostępność zasobów niż w wypadku komputerów stacjonarnych czy serwerów. Dlatego jednym z wymagań niefunkcjonalnych jest konieczność prawidłowego wyświetlania interfejsu na urządzeniach z różnymi rozmiarami, rozdzielczościami i proporcjami ekranów. Aplikacja musi również działać wydajnie, ale jednocześnie nie może zużywać za dużo energii. Należy również zwrócić uwagę na typowe sposoby nawigacji pomiędzy oknami na

urządzeniach przenośnych, dzięki czemu we własnej aplikacji można wykorzystać intuicję użytkownika. Jednym ze sposobów na dostosowanie aplikacji do potrzeb osób niepełnosprawnych jest dołączenie tekstowych objaśnień obrazków. Aplikacja musi działać na możliwie jak największej liczbie urządzeń, dlatego powinna wspierać starsze wersje systemu Android[7].

2.3. Model przypadków użycia

W celu lepszego zobrazowania zależności pomiędzy wymaganiami oraz wskazania podstawowego modelu interakcji z aplikacją przygotowano diagram przypadków użycia.



Rys. 3: Diagram przypadków użycia dla przedstawionych wymagań.

Widoczny na Rys. 3 diagram przypadek użycia odpowiada na wymagania przedstawione w poprzednim podrozdziale, a ponadto nakreśla podstawy zmian widoków w wyniku interakcji z użytkownikiem. Dwoma centralnymi punktami są widok szczegółów wybranego projektu oraz widok wydatków globalnych. Wydatki globalne nie są powiązane z żadnym konkretnym projektem, dzięki czemu można je wykorzystywać w wielu różnych projektach. Widok szczegółowy projektu pozwala na dostęp do większości operacji przedstawionych jako wymagania funkcjonalne. Jest to spowodowane koniecznością powiązania poszczególnych bytów z konkretnym projektem, czyli domem. Większość akcji możliwych do wykonania przez użytkownika to dodanie, usunięcie, zaktualizowanie,

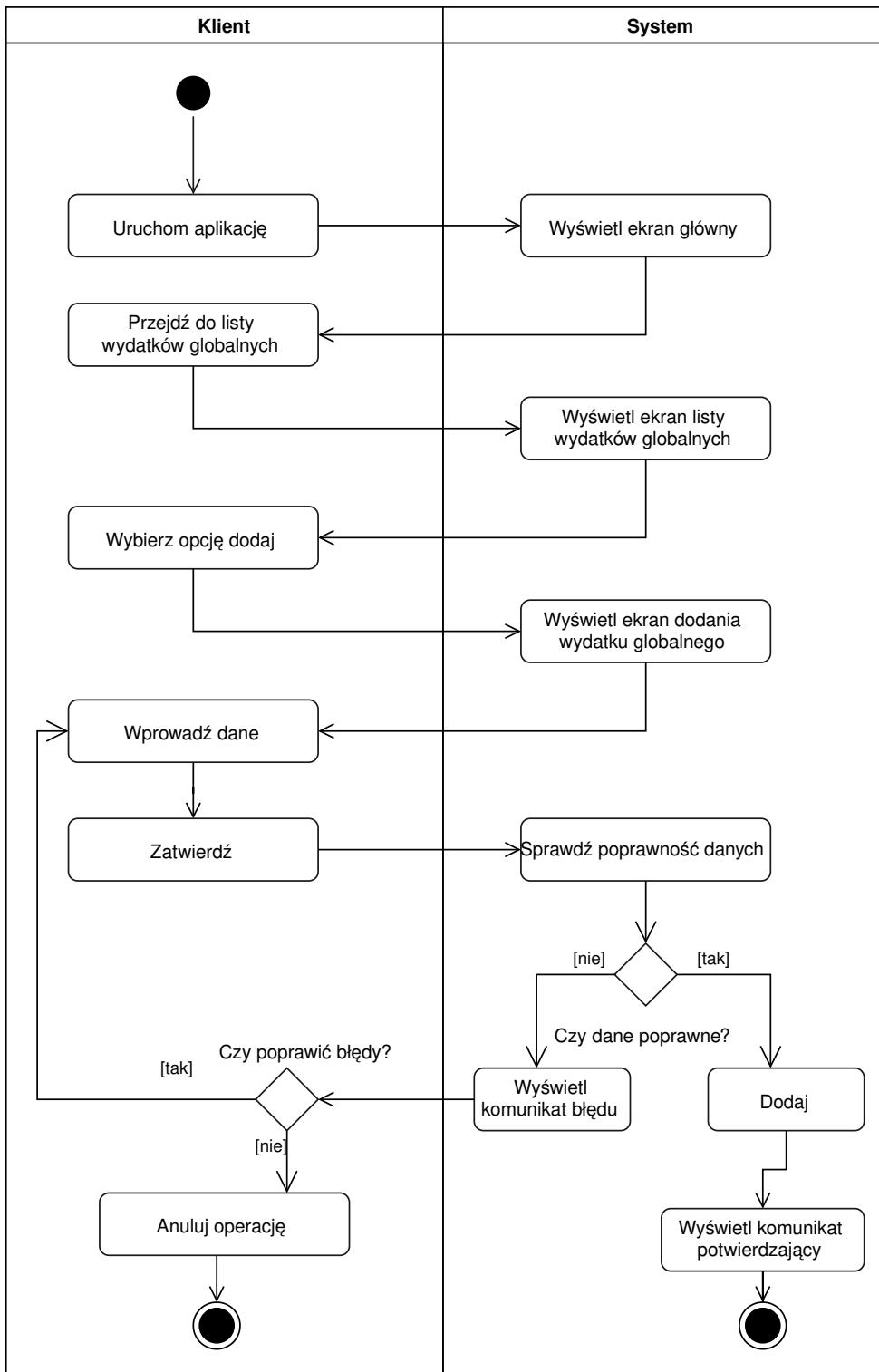
przeglądanie poszczególnych bytów przedstawionych w wcześniejszych podrozdziałach. Taki zbiór możliwych akcji odpowiada operacją CRUD².

2.4. Specyfikacja przypadków użycia

Przypadki użycia przedstawione na diagramie w rozdziale 2.3 należy doprecyzować. Jednym ze sposobów uszczegółowienia jest pokazanie ich realizacji na diagramie aktywności. Ze względu na powtarzalność możliwych interakcji użytkownika pokazano tylko po jednym typie PU³. Pominięto wyświetlanie danych, ponieważ jest to konieczne też w przypadku innych akcji.

2 CRUD – Create, Read, Update, Delete; Utwórz, Odczytaj, Aktualizuj, Usuń

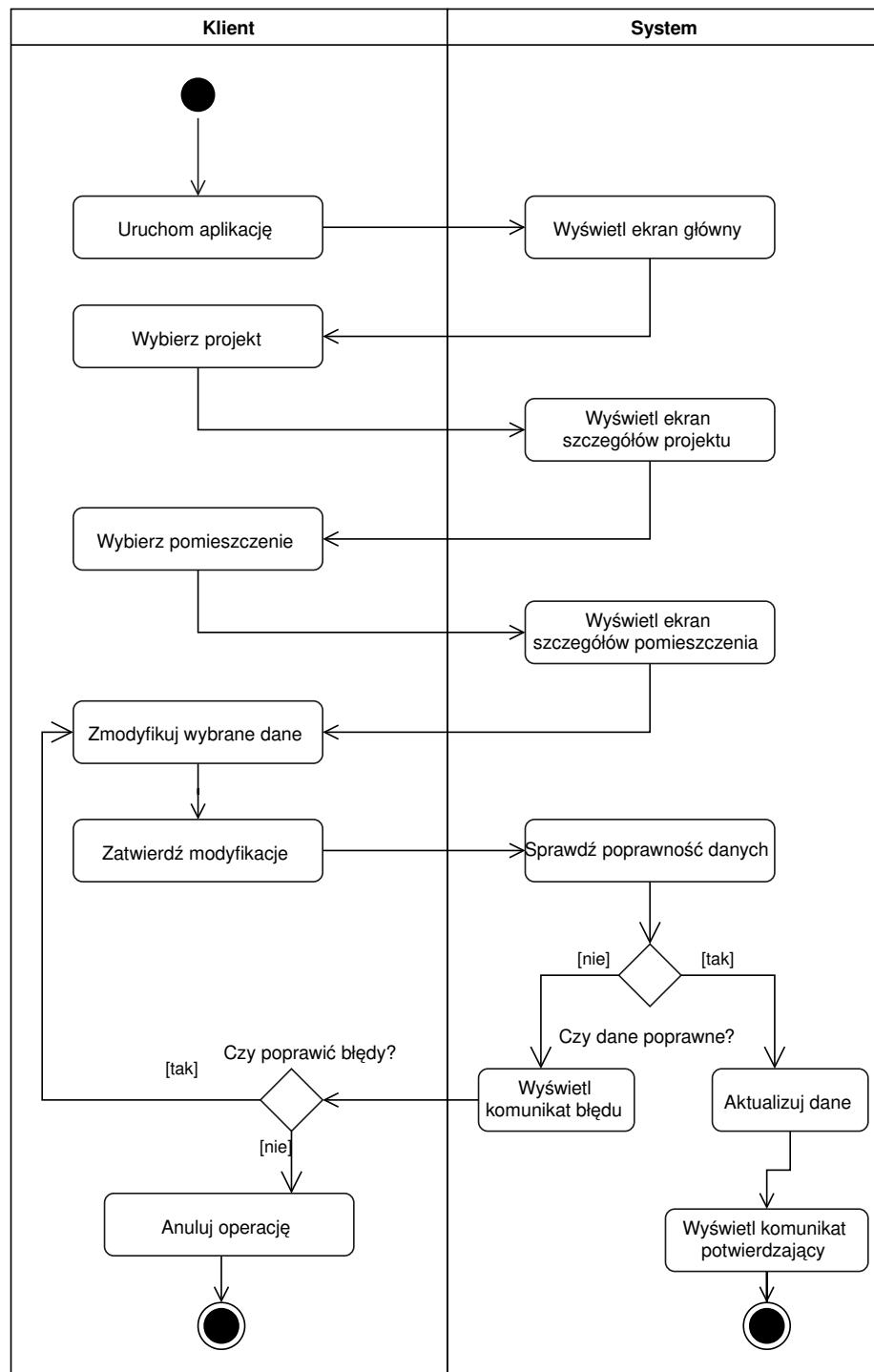
3 PU – przypadek użycia



Rys. 4: Diagram aktywności dla przypadku użycia „Dodaj wydatek globalny”.

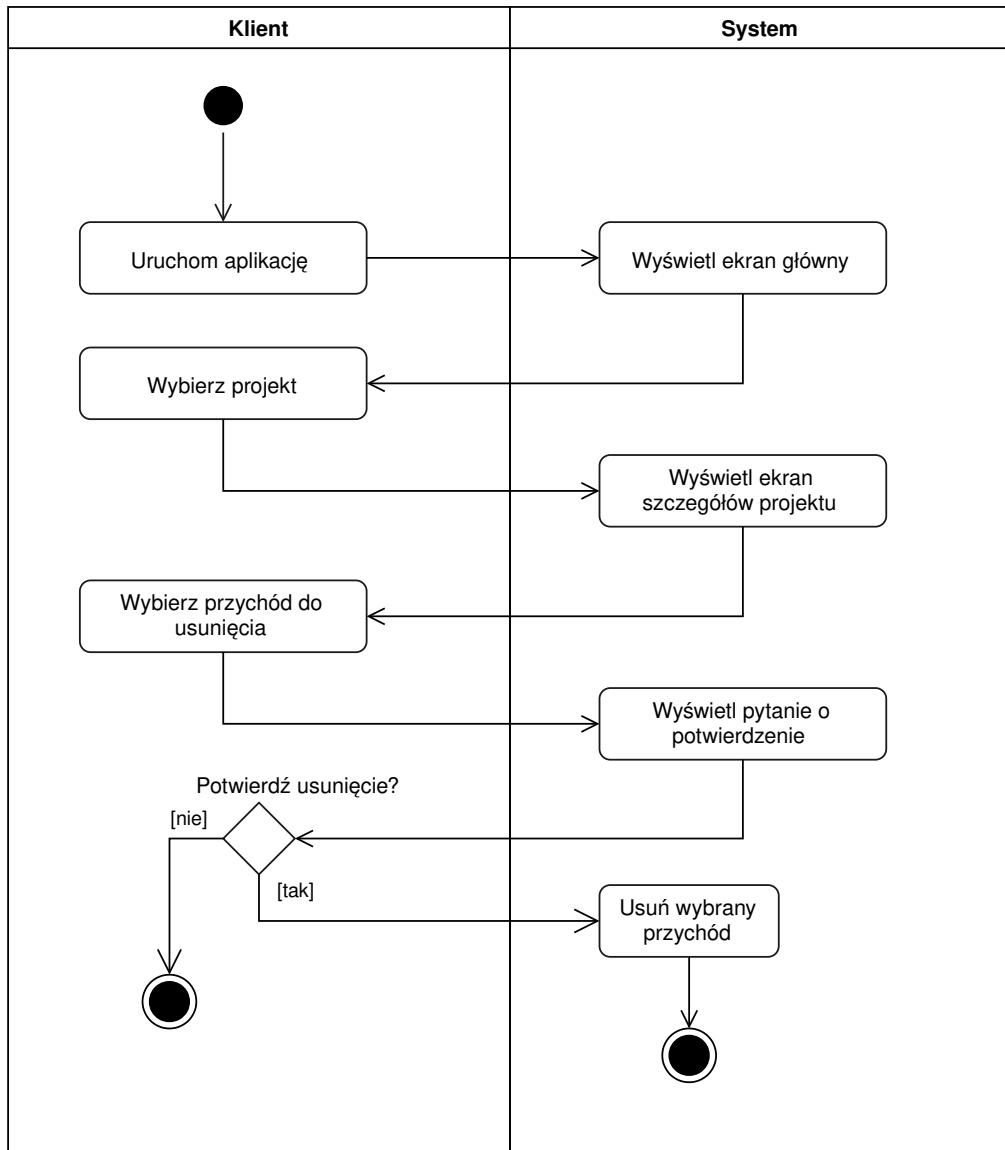
Na Rys. 4 przedstawiono diagram aktywności dla dodania wydatku globalnego, ale w przypadku dodawania innych rzeczy sekwencja operacji będzie bardzo zbliżona do pokazanej na diagramie. Po uruchomieniu aplikacji użytkownik nawiguje do ekranu dodania nowego wydatku globalnego, wprowadza dane i akceptuje dodanie. Aplikacja sprawdza

poprawność wprowadzonych danych zgodnie z regułami biznesowymi. Jeżeli dane są poprawne następuje dodanie ich do bazy danych aplikacji oraz wyświetlenie komunikatu powodzenia operacji. Alternatywnie dodanie nowego wydatku globalnego może zostać odrzucone ze względu na niepowodzenie walidacji, a użytkownik zostanie o tym poinformowany. W takiej sytuacji można poprawić dane zgodnie z otrzymanym komunikatem, albo zarzucić wykonanie całej operacji.



Rys. 5: Diagram aktywności dla przypadku użycia „Edytuj pomieszczenie”.

Rys. 5 przedstawia bardzo podobny przebieg akcji jak diagram z Rys. 4. Jednak w tym przypadku użytkownik wybiera konkretne pomieszczenie do zmodyfikowania, a przy wyświetlanie ekranu szczegółów pomieszczenia aplikacja musi załadować dane i wypełnić nimi interfejs.

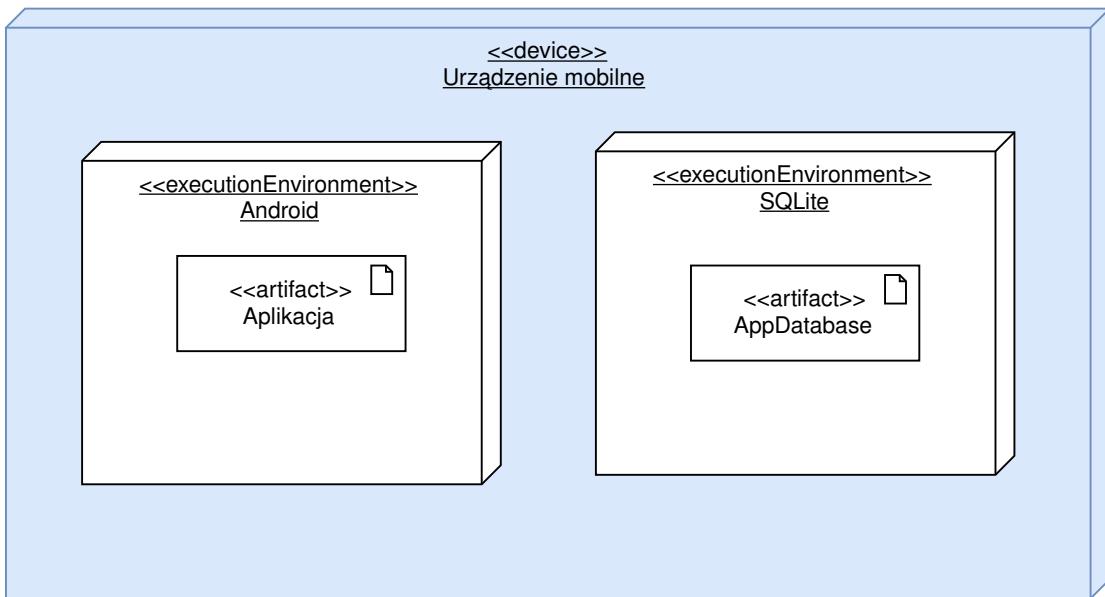


Rys. 6: Diagram aktywności dla przypadku użycia „Usuń przychód”.

Rys. 6 przedstawia diagram aktywności dla usunięcia przychodu z wybranego projektu. Użytkownik uruchamia aplikację, wybiera projekt, nawiguje do listy przychodów, a następnie usuwa wybraną pozycję. Aplikacja odpowiada zapytaniem czy na pewno użytkownik chce to zrobić. W wypadku odpowiedzi twierdzącej aplikacja usuwa wybrany przychód i wyświetla komunikat potwierdzający wykonanie tej operacji. Alternatywnie użytkownik może zmienić zdanie i żadna akcja nie zostanie wykonana przez aplikację.

2.5. Architektura logiczna i fizyczna

Istotnym elementem projektu aplikacji jest zaprojektowanie jej architektury logicznej i fizycznej. Aplikacja ma działać w całości na urządzeniu użytkownika i nie wymagać żadnych zewnętrznych serwisów czy baz danych. Pomimo tego architektura logiczna powinna pozwalać na łatwy i wymagający jak najmniej zmian w kodzie aplikacji sposób dostosowania programu do korzystania z komunikacji sieciowej z serwerem. Architektura fizyczna aplikacji została przedstawia na Rys. 7.



Rys. 7: Diagram architektury fizycznej aplikacji.

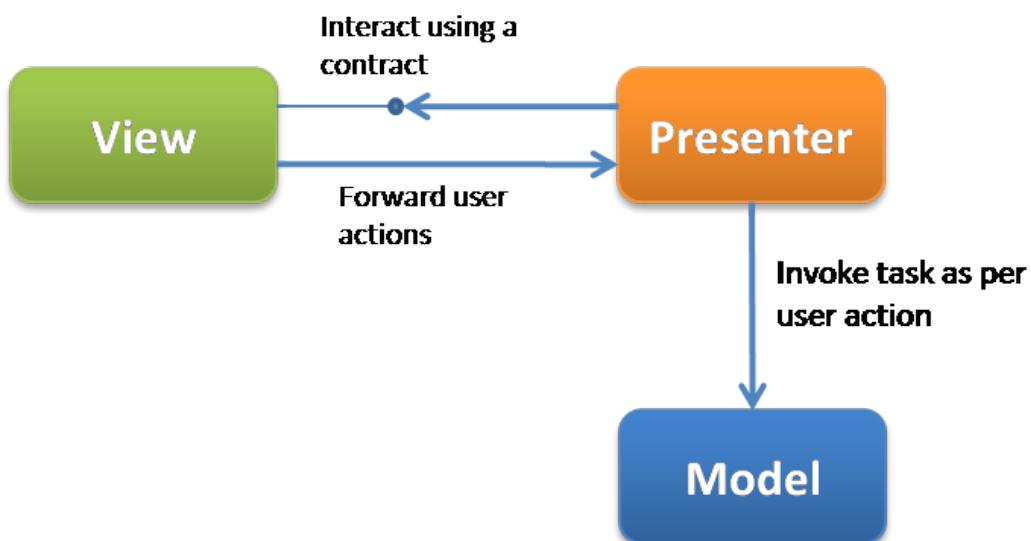
Widoczna na Rys. 7 architektura fizyczna przedstawia dwa artefakty: aplikacja, baza danych. Aplikacja działa pod kontrolą systemu operacyjnego Android, a baza danych wykorzystuje SZBD⁴ SQLite. Baza SQLite jest dostępna w systemie Android bez konieczności jej pobierania czy instalacji, co zmniejsza liczbę zależności aplikacji[8]. Oba artefakty znajdują się fizycznie na jednym urządzeniu.

Architekturę logiczną aplikacji należy dobrze zaprojektować. W tym celu może być przydatne wykorzystanie uznanych wzorców architektonicznych[9]:

- **MVC** (Model-View-Controller) – model reprezentuje dane i zawiera logikę biznesową, widok(view) na podstawie modelu prezentuje dane użytkownikowi, kontroler(controller) odpowiada na akcje użytkownika aktualizując model. Pojedynczy kontroler może obsługiwać wiele widoków.
- **MVP** (Model-View-Presenter) – w tej architekturze Presenter, zbliżony funkcjonalnie do kontrolera, odpowiada za całą komunikację pomiędzy widokiem, a modelem. Widok i prezenter są ze sobą ścisłe powiązane, ponieważ prezenter obsługuje tylko jeden widok.
- **MVVM** (Model-View-ViewModel) – wzorzec zbliżony do MVP, jednak powiązanie ViewModel z widokiem jest inne niż w przypadku MVP, odbywa się za pomocą mechanizmu wiązania danych. ViewModel może obsługiwać wiele widoków.

4 SZBD – System Zarządzania Bazą Danych

MVP - Passive View



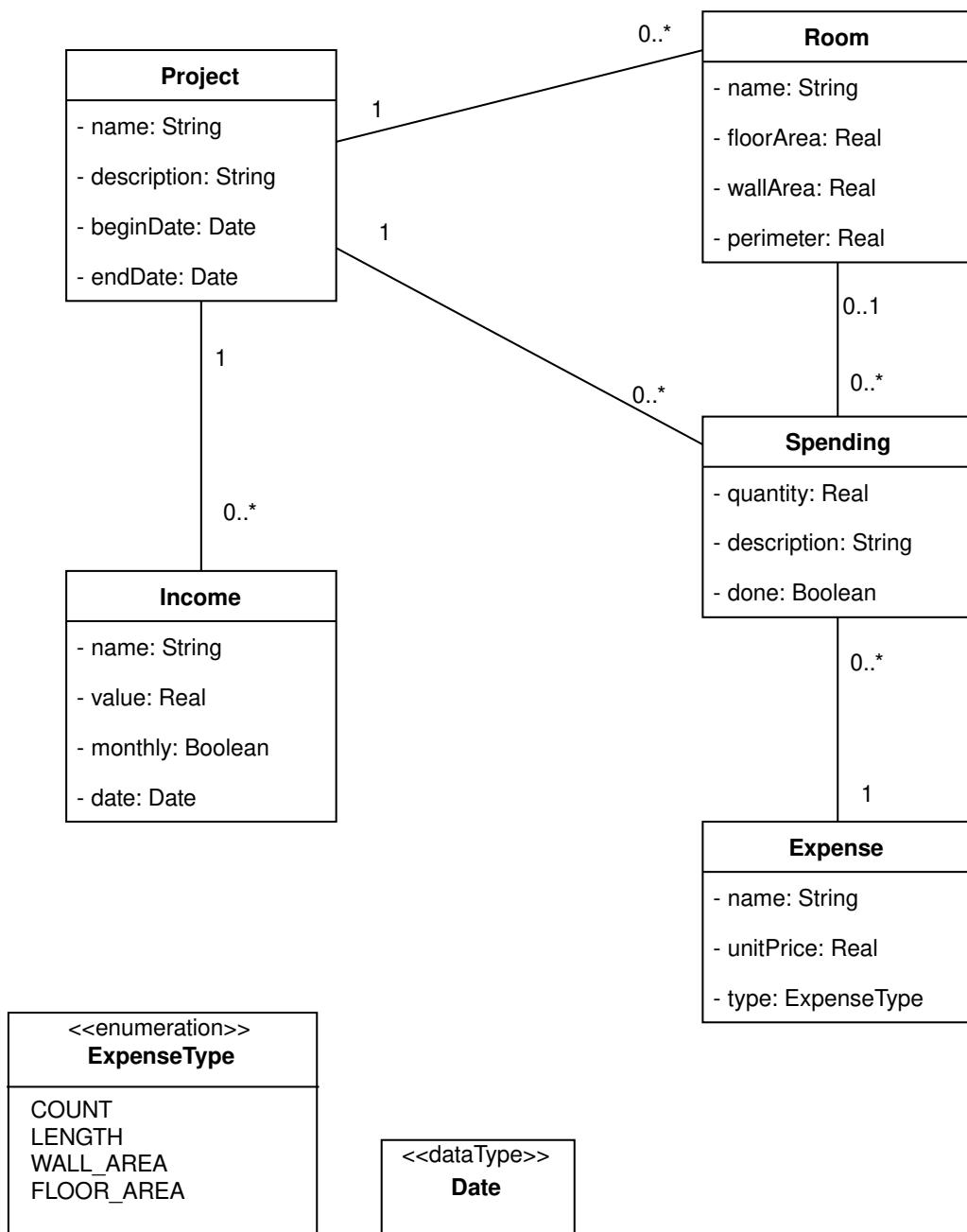
Rys. 8: Diagram przedstawiający relacje pomiędzy klasami w wzorcu MVP. Źródło: [9]

W wyniku analizy projektu aplikacji oraz zapoznania się z dodatkowymi materiałami postanowiono wykorzystać wzorzec MVP[10]. W tym wzorcu architektonicznym zastosowanym dla aplikacji rozwijanej dla systemu Android klasy typu *Activity* należą do widoku, a klasy prezenterów oraz modeli nie powinny zależeć od elementów API Android. Takie oddzielenie zapewnia lepszą testowalność aplikacji oraz łatwiejsze przenoszenie na inne rozwiązania. Prezenter oraz widok powinny się komunikować według ustalonego protokołu. Każdy widok i prezenter implementują klasy interfejsowe opisujące odpowiednie protokoły[11]. Dzięki wykorzystaniu interfejsów zamiast konkretnych klas łatwo można zmienić implementację, zwiększa to również jakość podziału obowiązków pomiędzy klasami. Takie podejście wykorzystania interfejsów zamiast konkretnych klas, pomimo że widok i prezenter są ze sobą w relacji 1 do 1, jest realizacją zasady DIP⁵. Wspomniana zasada mówi w skrócie: „Polegaj na abstrakcjach. Nie polegaj na konkretach”[12].

2.6. Model informacyjny

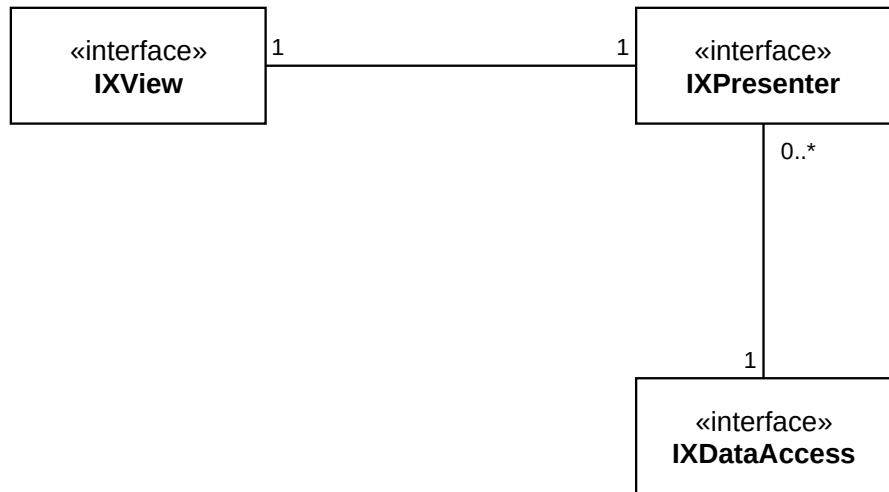
Celem tego rozdziału jest przedstawienie pełnego diagramu klas z uwzględnieniem bytów biznesowych, możliwej realizacji architektury aplikacji oraz innych istotnych klas.

5 DIP – Dependency Inversion Principle



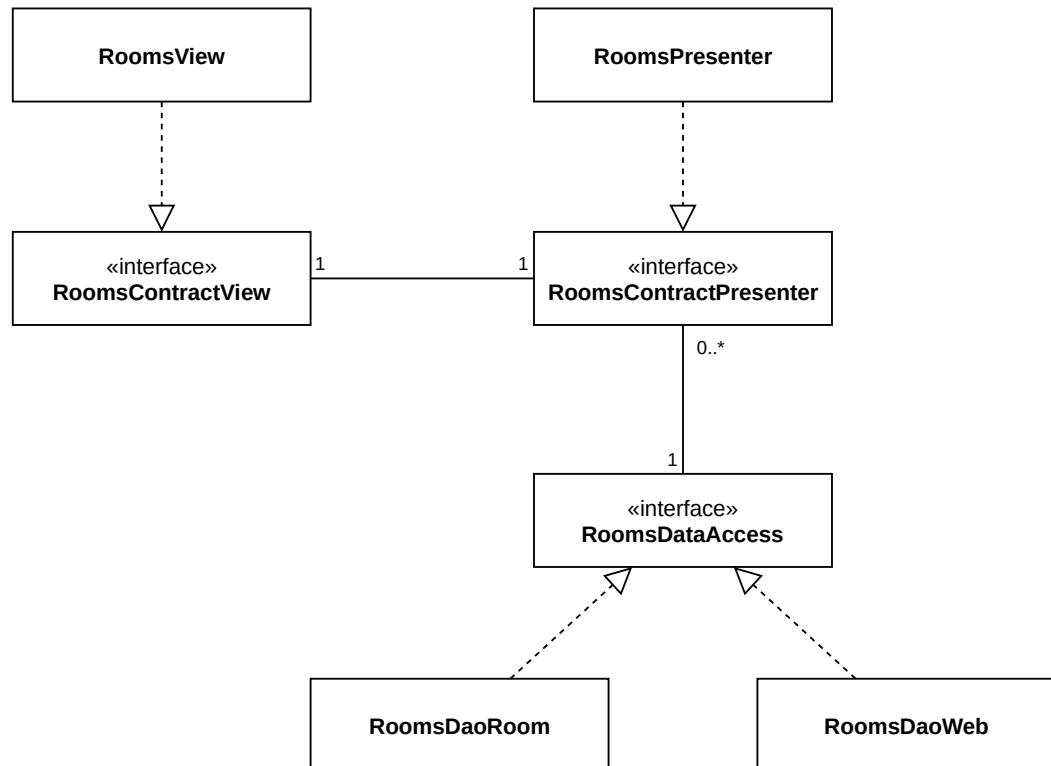
Rys. 9: Diagram klas dla bytów biznesowych.

Diagram widoczny na Rys. 9 prezentuje finalny diagram klas dla bytów biznesowych omówionych w rozdziale 2.1. W klasach *Project* i *Spending* doszły atrybuty reprezentujące dodatkowy opis, który można wykorzystać do przechowywania komentarza lub innych istotnych, z punktu widzenia użytkownika, informacji. Pozostałe klasy nie uległy zmianie.



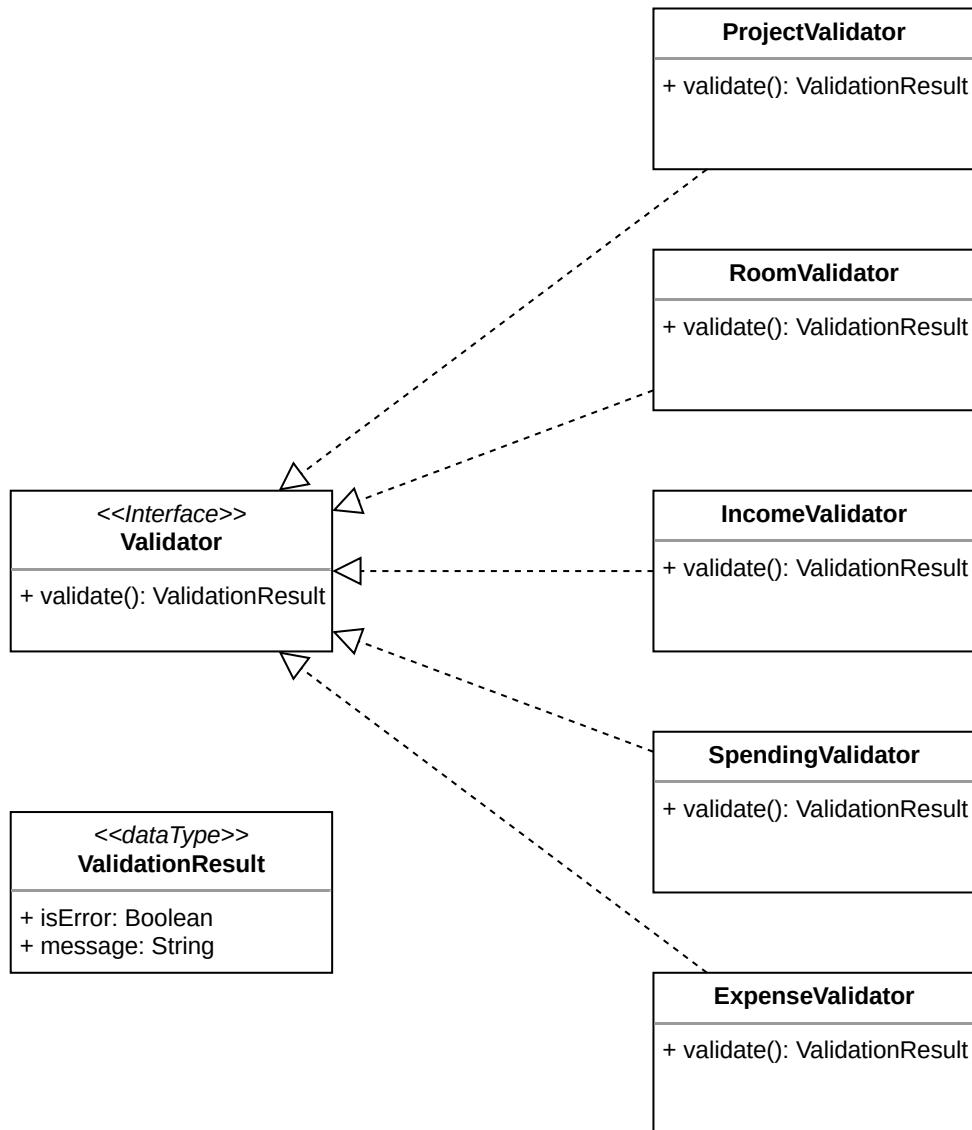
Rys. 10: Uogólniony diagram klas dla architektury MVP.

Diagram klas widoczny na Rys. 10 przedstawia ogólny schemat klas realizujących wzorzec architektoniczny MVP. Ze względu na realizację reguły DIP zamiast konkretnych klas zostały pokazane abstrakcyjne interfejsy, które stanowią protokoły komunikacji pomiędzy klasami realizującymi wybrany wzorzec. Ze względu na zmniejszenie rozmiaru diagramu zastosowano uogólnione nazwy zamiast konkretnych interfejsów. Przykład dokładnego przedstawienia pokazano na Rys. 11.



Rys. 11: Szczegółowy diagram klas dla widoku listy pomieszczeń.

Widoczne na Rys. 11 klasy interfejsowe stanowią protokół, zgodnie z którym komunikują się klasy konkretne. Pomiędzy konkretnymi klasami nie występują asocjacje ponieważ są one dziedziczne z interfejsów[13]. *RoomsView* odpowiada za interfejs użytkownika, *RoomsPresenter* komunikuje interfejs z danymi, a *RoomsDataAccess* odpowiada za pozyskiwanie danych. Ze względu na poleganie na abstrakcjach łatwo można zmienić implementację dostępu do danych i zamiast korzystać z lokalnej bazy danych SQLite można komunikować się z zewnętrznym serwerem poprzez sieć. Z tego względu diagram pokazuje dwie klasy implementujące interfejs *RoomsDataAccess*.



Rys. 12: Diagram klas walidatorów sprawdzających poprawność danych wprowadzanych przez użytkownika.

Diagram z Rys. 12 przedstawia walidatory sprawdzające poprawność danych wprowadzanych przez użytkownika. Walidatory zapewniają również zgodność danych z regułami biznesowymi. Dzięki ich zastosowaniu do bazy danych trafią tylko poprawne rekordy. W wyniku walidacji nie jest zwracana prosta informacja mówiąca czy dane są poprawne, czy nie, a struktura zawierająca wyjaśnienie problemu. Wykorzystanie takiej

struktury zamiast wartości binarnej jest łatwo rozszerzalne oraz pozwala na przekazanie jasnego komunikatu błędu.

2.7. Interfejs użytkownika

W celu zamodelowania interakcji użytkownika z aplikacją należy przygotować projekt UI⁶. Projektując interfejs aplikacji na urządzenia przenośne należy zwrócić uwagę na charakterystykę ekranów tych urządzeń. Pomimo niewielkich rozmiarów obecnie potrafią one mieć rozdzielcość większą niż monitory komputerowe, a ponadto nie zawsze są prostokątne (np.: wcięcie na aparat), co może spowodować problemy z prawidłowym wyświetlaniem treści. Interfejs aplikacji ma za zadanie realizować przypadki użycia wymienione w rozdziale 2.3. W trakcie prac implementacyjnych należy zwrócić również uwagę na to, aby wątek aplikacji obsługujący interfejs nie zajmował się innymi rzeczami, a w szczególności zadaniami wymagającymi dużych nakładów obliczeniowych.

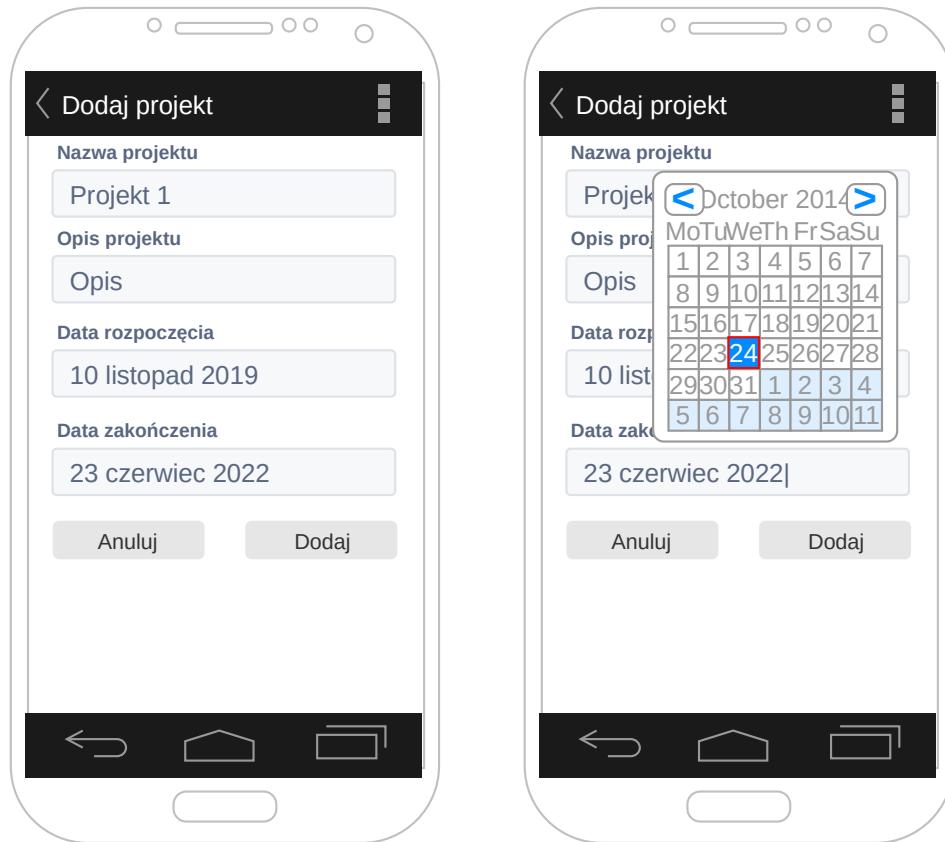


Rys. 13: Projekt ekranu listy projektów.

Na Rys. 13 pokazano ekran listy wszystkich projektów. Pojedyncza pozycja listy składa się z nazwy projektu oraz opcjonalnego opisu. Wybranie projektu przekierowuje do widoku

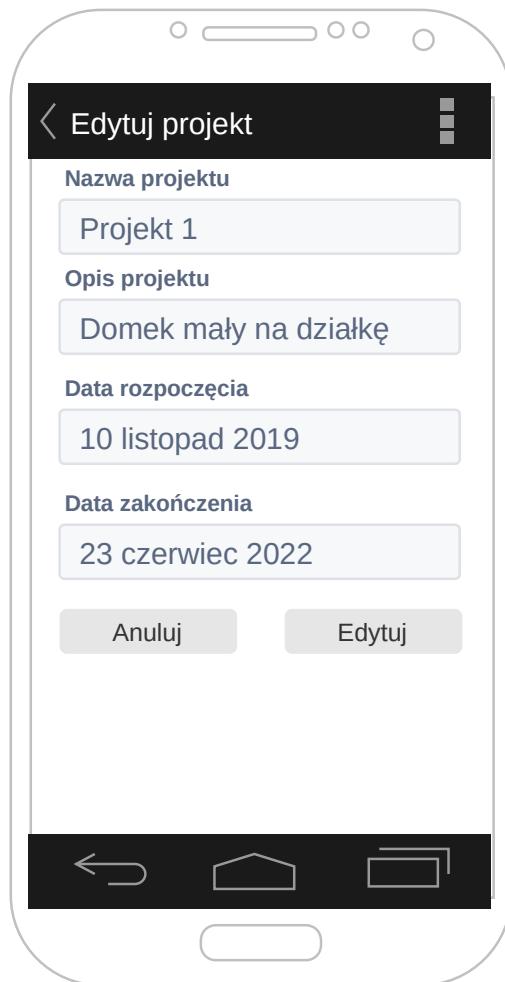
6 UI – User Interface, Interfejs użytkownika

szczegółów. Rozwijane menu oznaczone trzema kropkami w prawym górnym rogu pozwala dodać nowy projekt albo przejść do listy wydatków globalnych.



Rys. 14: Prototyp ekranu dodania nowego projektu.

Pokazany na Rys. 14 ekran pozwala na dodanie nowego projektu. Ten interfejs pozwala na podanie w polach tekstowych nazwy, opisu oraz dat. W przypadku naciśnięcia pola daty pojawia się widżet kalendarza pozwalający na wybranie daty w ten sposób. Użytkownik w każdej chwili może wrócić do poprzedniego ekranu wybierając jedną z trzech opcji: wstecz, do góry, przycisk „anuluj”. Po wybraniu „dodaj” aplikacja sprawdza poprawność danych i wyświetla odpowiedni komunikat informujący o stanie powodzenia operacji oraz ewentualnych błędach.



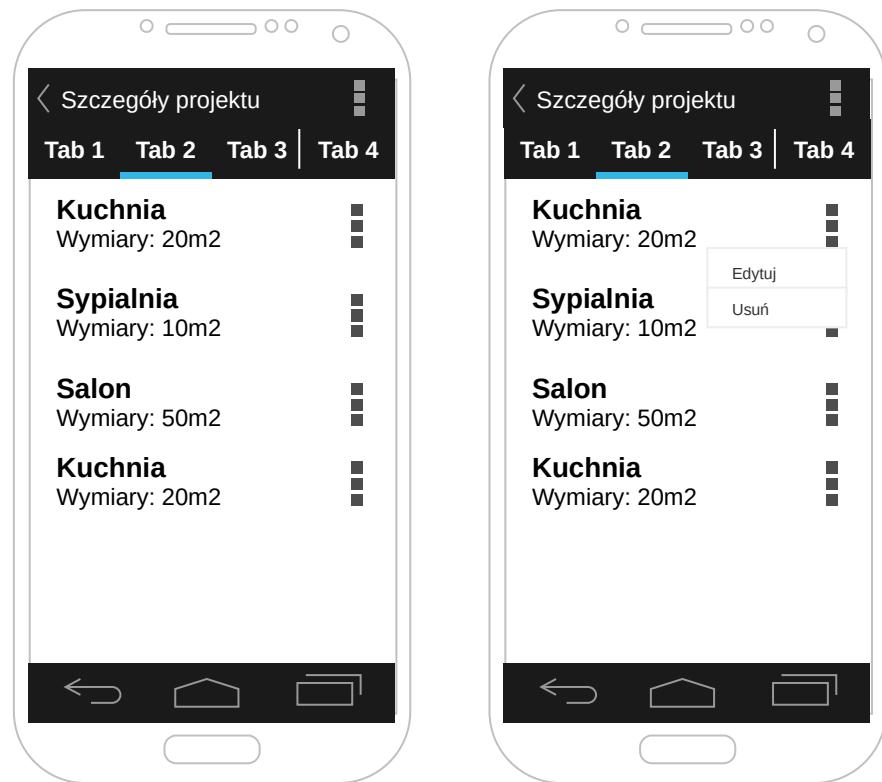
Rys. 15: Prototyp ekranu modyfikacji istniejącego projektu.

Widoczny na Rys. 15 prototyp interfejsu przedstawia okno modyfikacji informacji dotyczących wybranego projektu. Jest ono niemal identyczne z oknem dodania nowego projektu, jednak w tym przypadku zamiast przycisku „dodaj” jest przycisk „edytuj”. Kolejną różnicą jest to, że pola są już wypełnione danymi projektu.



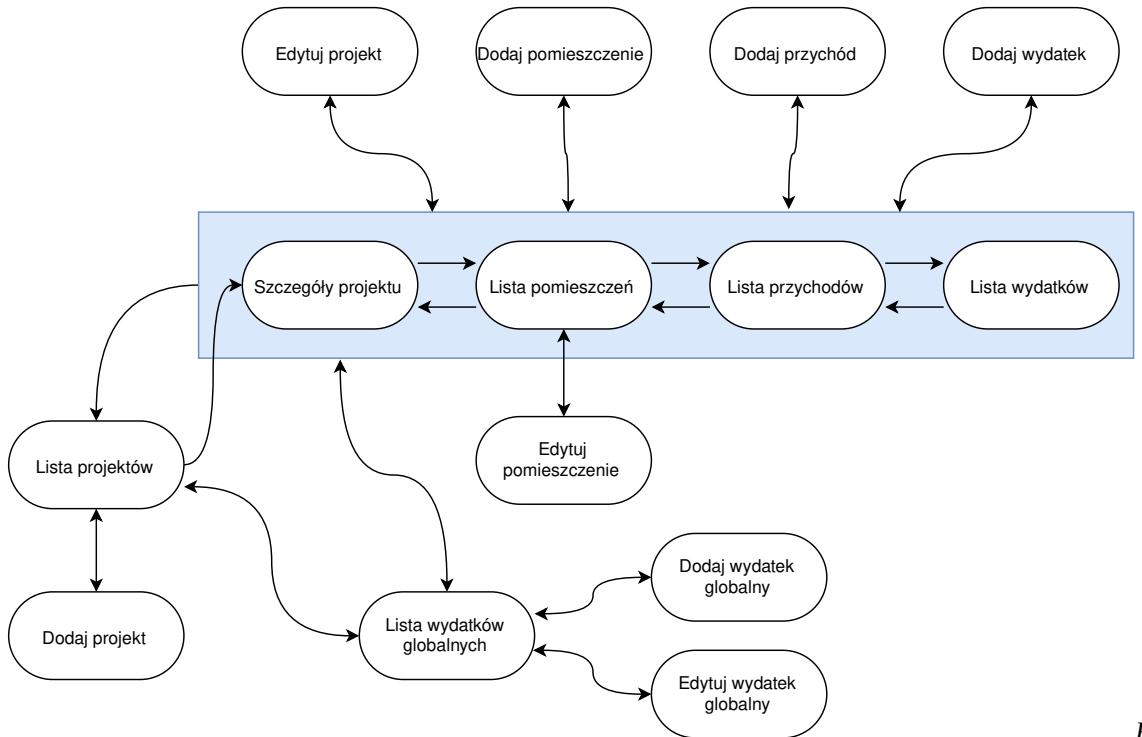
Rys. 16: Prototyp ekranu szczegółów projektu.

Znajdujący się na Rys. 16 prototyp interfejsu przedstawia jedną zakładkę ekranu szczegółów projektu. Aplikacja na podstawie danych związanych z projektem może obliczyć dodatkowe informacje i przedstawić je użytkownikowi. W prawym górnym rogu znajduje się też rozwijane menu zawierające dodatkowe opcje.



Rys. 17: Prototyp ekranu listy pomieszczeń.

Przedstawiony na Rys. 17 projekt ekranu wyświetla listę pomieszczeń powiązanych z wybranym projektem. Pojedyncza pozycja składa się z nazwy oraz informacji o wymiarach pomieszczenia. Obok każdej pozycji listy znajduje się też przycisk pozwalający wyświetlić menu zawierające dodatkowe akcje. Dzięki temu można przejść do ekranu modyfikacji wybranego pomieszczenia albo je usunąć.



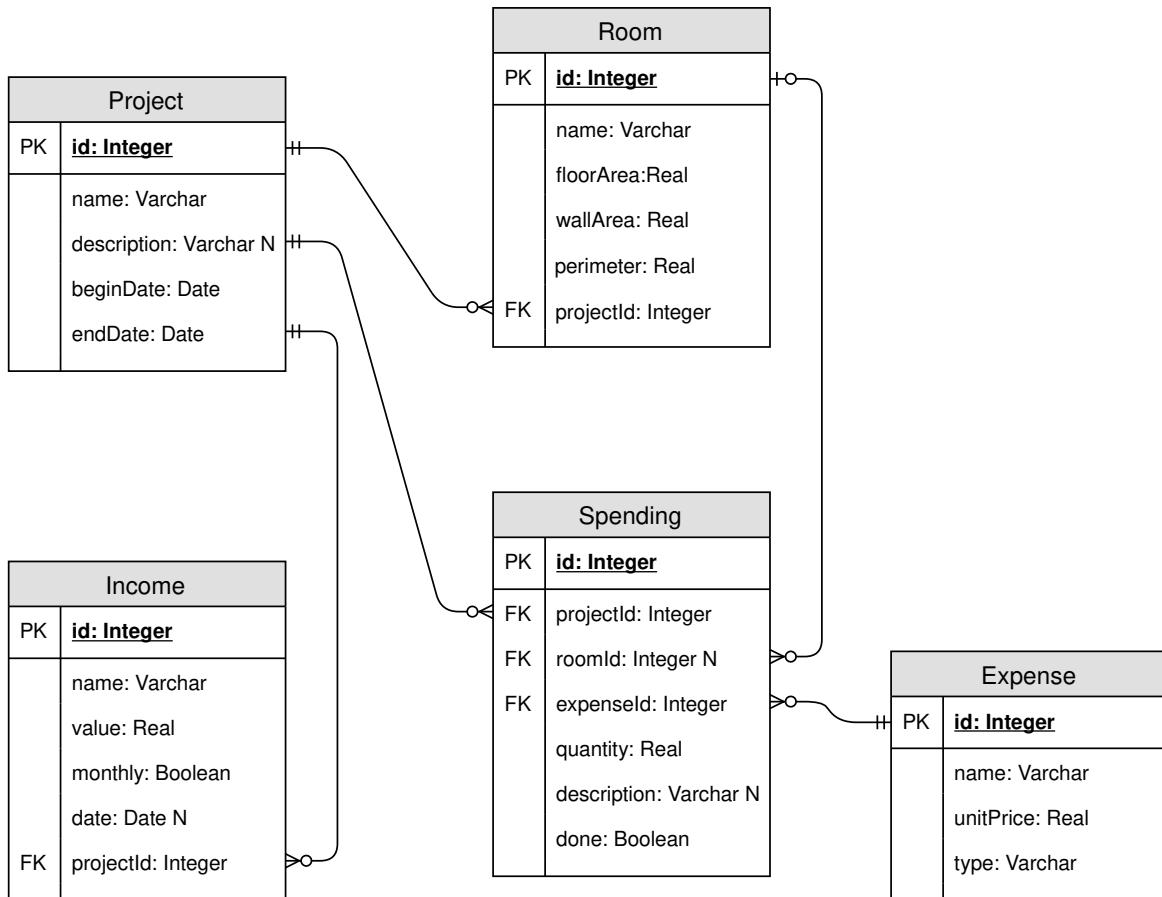
Rys.

18: Diagram przepływu nawigacji.

Rys. 18 ma za zadanie zademonstrować przepływ nawigacji po interfejsie. Okno szczegółów projektu zostało pokazane jako niebieski obszar, który składa się z wielu zakładek. Można się pomiędzy nimi przemieszczać, a każda z nich zawiera inną informację.

2.8. Projekt bazy danych

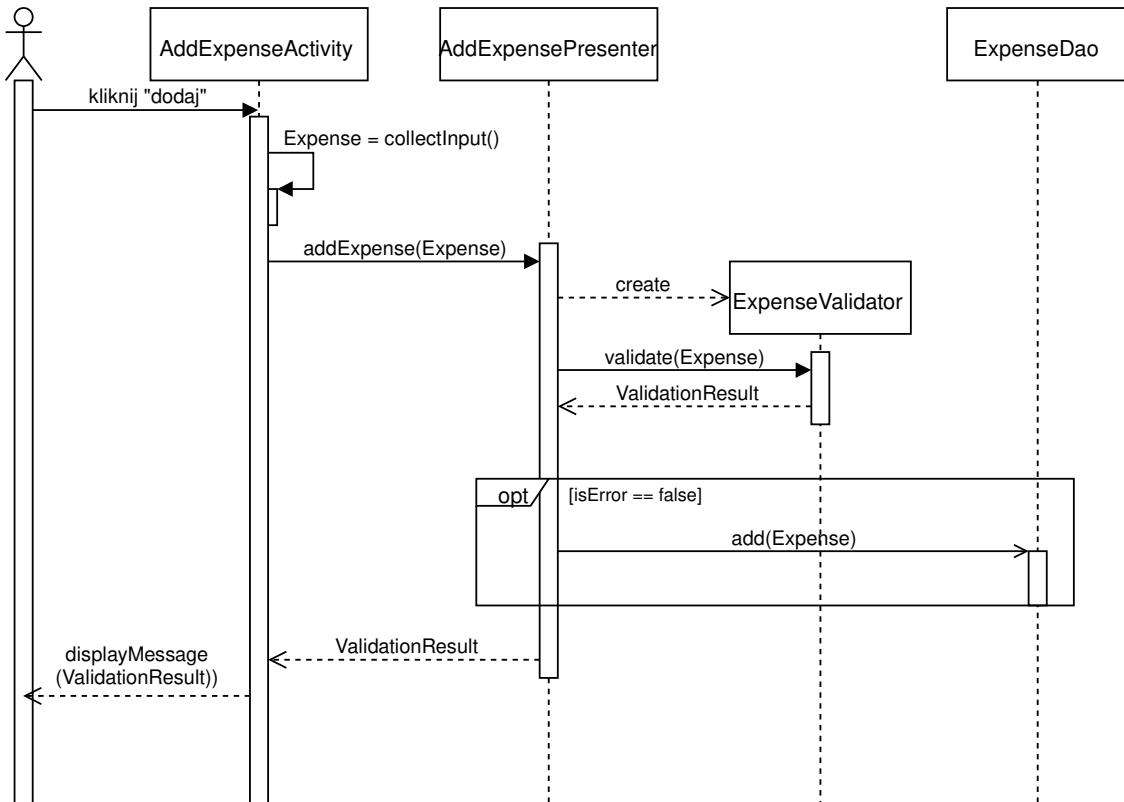
Baza danych została zrealizowana na podstawie modelu domenowego i przedstawiona na diagramie Rys. 19. Encje są ze sobą powiązane w opisany we wcześniejszych rozdziałach sposób. Typy danych wymienione przy atrybutach są rodzaju ogólnego, nie mają powiązania z żadnym konkretnym SZBD. Dzięki temu projekt można łatwo przenieść na różne implementacje bazy danych. Każda encja jest jednoznacznie identyfikowana przez klucz sztuczny będący liczbą całkowitą. Wykorzystanie liczb całkowitych pozwala na szybsze, niż w wypadku użycia ciągów znakowych (np. nazwa), łączenie tabel oraz wyszukiwanie, zajmuje mniej miejsca w pamięci, a ponadto pozwala na bezproblemową zmianę reprezentacji danych istotnych dla obiektu biznesowego[14]. Wartość opisującą rodzaj wydatku globalnego jest reprezentowana przez identyfikujący ją jednoznacznie ciąg znaków. Innym podejściem byłoby utworzenie dodatkowej tabeli przechowującej rodzaje wydatków globalnych.



Rys. 19: Diagram ERD dla wykonanej bazy danych.

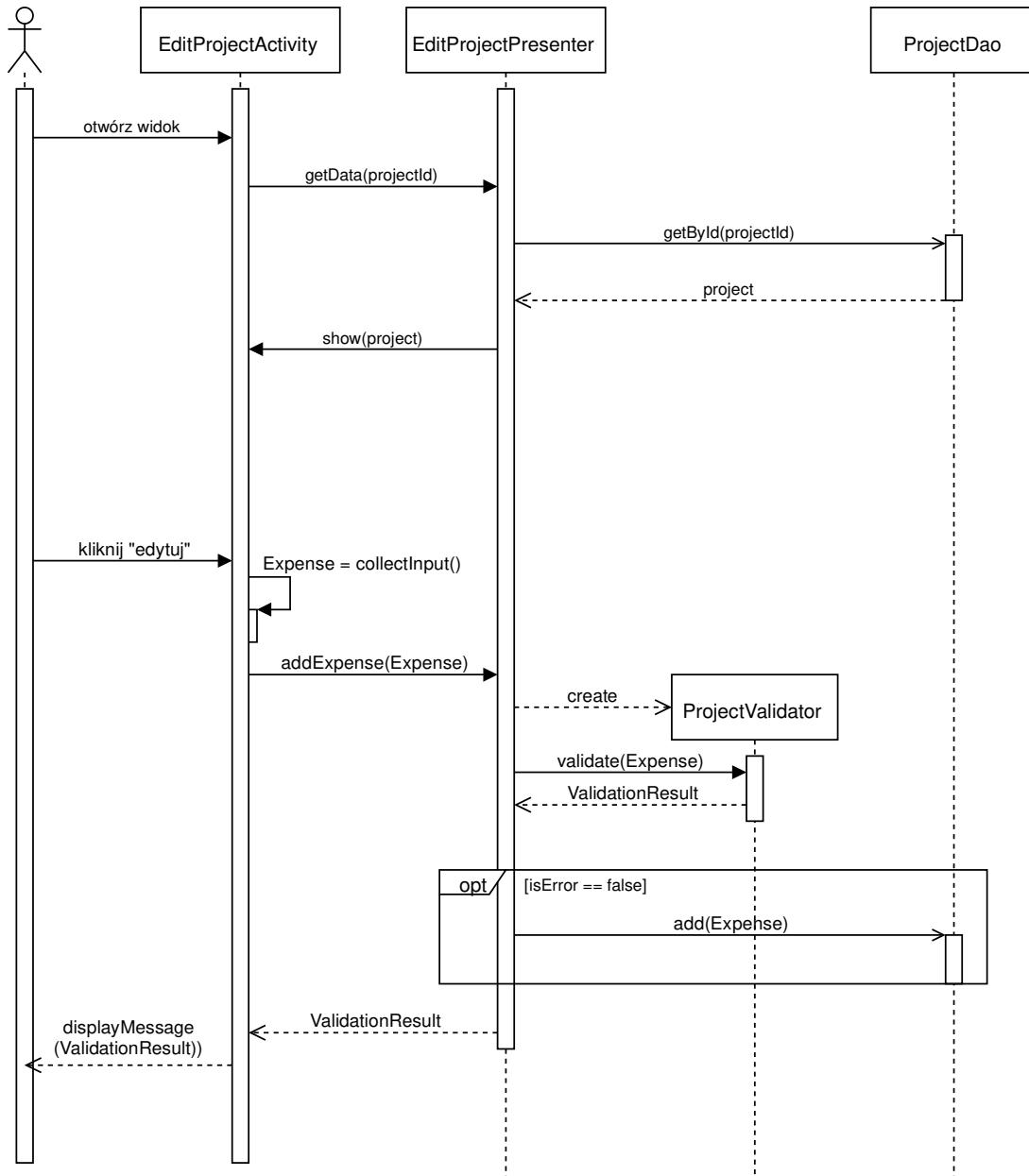
2.9. Ideowe przedstawienie interakcji

Na tak zaawansowanym etapie prac projektowych można sporządzić bardziej szczegółowy opis interakcji pomiędzy poszczególnymi klasami i w jaki sposób współpracują by osiągnąć zamierzony cel. Jednym ze sposobów jest przedstawienie tego przy pomocy diagramów sekwencji. Głównymi akcjami dokonywanymi na obiektach biznesowych są: utwórz, przeglądaj, modyfikuj, usuń. Z tego powodu postanowiono pokazać właśnie te akcje na diagramach. Przeglądanie zostało pominięte ponieważ jest też częścią pozostałych operacji. Dzięki zastosowanej architekturze zestaw potrzebnych klas dla różnych widoków jest praktycznie identyczny. Miejscami, dla jasności diagramów, pominięto przepływy alternatywne. Ograniczają się one do zamknięcia widoku, wybrania opcji wstecz lub anulowania, co powoduje zaniechanie całej operacji.



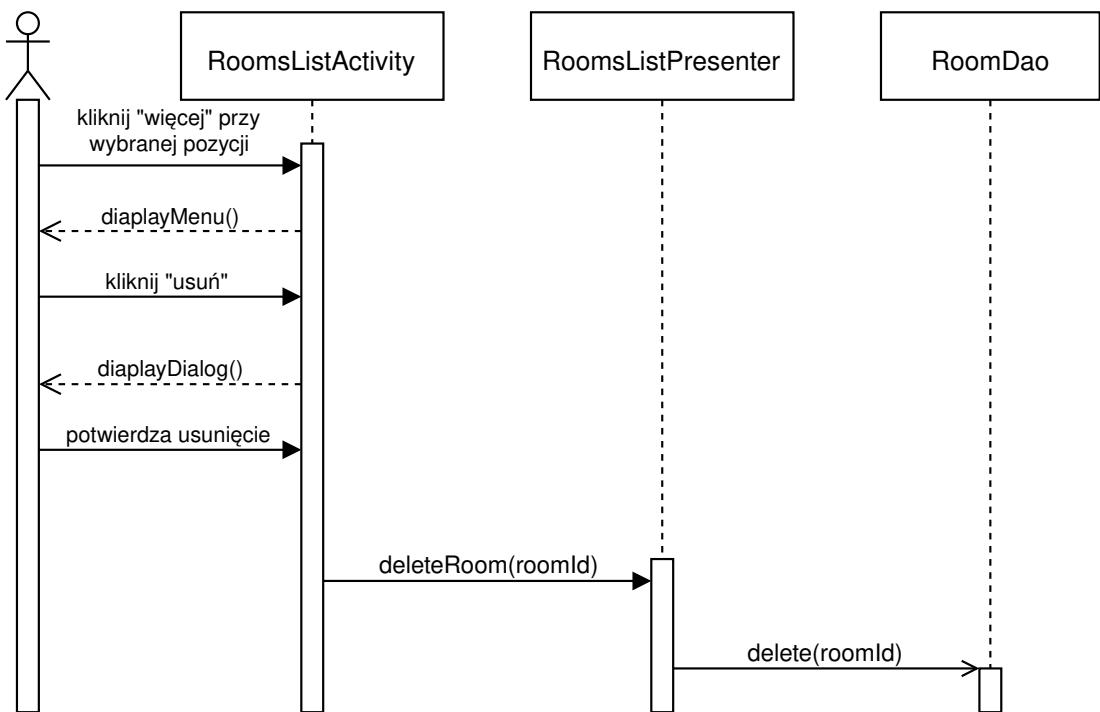
Rys. 20: Diagram sekwencji dla dodania wydatku globalnego.

Widoczny na Rys. 20 diagram przedstawia przebieg komunikacji poszczególnych klas i w jaki sposób współpracują w celu dodania nowego wydatku globalnego. Widok zbiera dane wprowadzone przez użytkownika i przekazuje je prezenterowi w celu dodania nowego wydatku globalnego do bazy danych aplikacji. Prezenter najpierw sprawdza poprawność danych przy pomocy obiektu klasy *ExpenseValidator*. Jeżeli dane nie zawierają błędów zostają przekazane do obiektu *ExpenseDao*, który ma za zadanie połączyć się z bazą danych i dodać do niej nowe rekordy. Użytkownik otrzymuje komunikat informujący o powodzeniu operacji lub wskazujący czego dotyczył błąd walidacji.



Rys. 21: Diagram sekwencji dla edycji projektu.

Diagram widoczny na Rys. 21 został przygotowany dla operacji zmodyfikowania istniejącego projektu. Druga część diagramu składa się z zebrania wprowadzonych przez użytkownika informacji, sprawdzenia ich poprawności oraz dodania nowych rekordów do bazy danych. Jest to bardzo podobne do diagramu z Rys. 20. Z kolei pierwsza część to przygotowanie okna. Po przejściu przez użytkownika na ekran edycji projektu należy wypełnić go już istniejącymi informacjami. W tym celu prezenter pobiera dane dzięki pomocy DAO, a otrzymanymi wartościami wypełnia odpowiednie elementy interfejsu.



Rys. 22: Diagram sekwencji dla usunięcia pomieszczenia.

Przedstawiony na Rys. 22 diagram pokazuje przepływ komunikacji obiektów dla przypadku usunięcia wybranego pomieszczenia. Widok pyta użytkownika czy ten jest w pełni świadomy co chce zrobić. Jeżeli otrzyma odpowiedź twierdzącą, przekazuje zapytanie do prezentera, a ten wywołanie funkcji usuń do `RoomDao`. W wyniku tej operacji z bazy danych zostają usunięte odpowiednie rekordy.

3. Implementacja aplikacji

Niniejszy rozdział ma za zadanie przedstawić implementację rozwiązania na podstawie wykonanego projektu oraz, w razie konieczności, wskazać różnice. W treści rozdziału znajduje się również omówienie wykorzystanego środowiska programistycznego, technologii i języka programowania w jakim zaimplementowano aplikację.

3.1. Wykorzystana technologia

3.1.1. Android Studio

W trakcie prac nad aplikacją wykorzystano IDE⁷ Android Studio. Środowisko to zostało specjalnie stworzone w celu wspomagania rozwoju aplikacji dla urządzeń z systemem operacyjnym Android. Android Studio jest tworzone we współpracy Google z firmą JetBrains i rozwijane na podstawie jej środowiska IntelliJ IDEA[15]. W tracie prac implementacyjnych zauważone m.in. następujące cechy i funkcje wykorzystanego środowiska[16]:

- Zaawansowane opcje nawigacji po kodzie.
- Bardzo dobrze działające autouzupełnianie.
- Kontrola stylu pisania kodu.
- Refaktoryzacja zasobów i odwołań do nich w kodzie.
- Graficzny edytor interfejsów.
- Wspomaganie wykonywania tłumaczeń.
- Porady dotyczące indeksowania w sklepie Google Play oraz w wyszukiwarce Google.
- Porady dotyczące wsparcia różnych wersji systemu Android.
- Wsparcie dla języków Java oraz Kotlin.
- Konfiguracja projektu.
- Automatyczne pobieranie zależności projektu.
- Możliwość pobierania z poziomu środowiska dodatkowych zasobów, ikon.
- Zamienianie w kodzie nazw zasobów tekstowych na ich wartość tekstową.

⁷ IDE – Integrated Development Environment

```
package riper.housebuilder

import ...

interface EditProjectContract {
    interface Presenter {
        fun detach()
        fun updateProject(project: Project): ValidationResult
        fun getOldData(): Int
    }

    interface View {
        fun show(project: Project)
    }
}

class EditProjectPresenter(private val view: EditProjectContract.View, private val dao: ProjectDao) : CoroutineScope, EditProjectContract.Presenter {

    private val job = Job()
    override val coroutineContext: CoroutineContext = job + Dispatchers.IO

    override fun detach() {
        job.cancel()
        view = null
    }

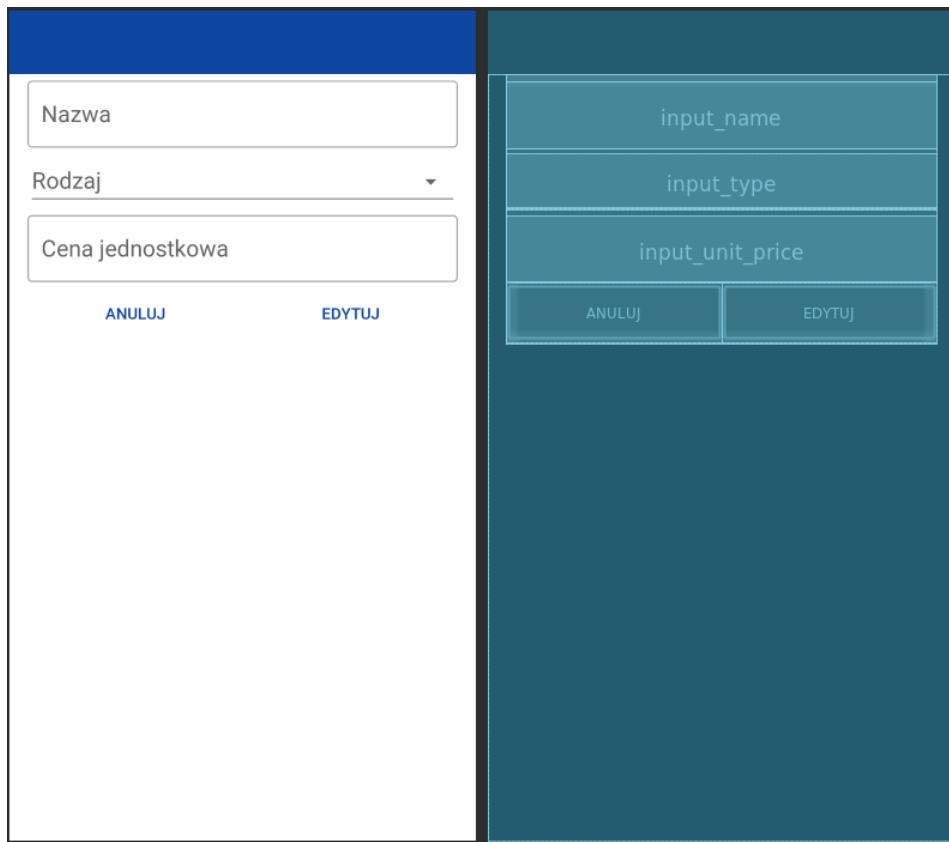
    override fun updateProject(project: Project): ValidationResult {
        val validator = ProjectValidator2(project)
        val validationResult = validator.validate()
        if (validationResult.isError) {
            launch(Dispatchers.IO) { (this.coroutineScope) }
            dao.update(project)
        }
        return validationResult
    }

    override fun getOldData(): Int {
        launch(Dispatchers.Main) { (this.coroutineScope)
            val data = dao.getOldData()
            if (data != null) {
                view?.show(data)
            }
        }
    }
}

class EditProjectActivity : AppCompatActivity(), EditProjectContract.View {
    private lateinit var currentProject: Project
    private lateinit var beginData: LocalDate
    private lateinit var endData: LocalDate

    private lateinit var editPresenter: EditProjectPresenter
    private lateinit var editData: EditData
}
```

Rys. 23: Przykładowe okno środowiska Android Studio.



Rys. 24: Przykład okna aplikacji w graficznym edytorze środowiska Android Studio.

3.1.2. Kotlin

Rozważanymi przy implementacji aplikacji językami programowania były Java oraz Kotlin. Na podstawie wcześniejszych doświadczeń z językiem Java oraz krótkiego przeglądu możliwości jakie daje Kotlin[17] postanowiono wykorzystać ten drugi. Kotlin jest statycznie typowanym językiem wspierającym wiele paradygmatów programowania. Język ten można wykorzystać do rozwijania aplikacji na platformę JVM⁸, system Android, przeglądarki internetowe wspierające JavaScript oraz do wytwarzania aplikacji natywnych[18]. Do głównych jego zalet należą, większa niż w porównaniu z Java, przejrzystość kodu oraz rozróżnienie typów zmiennych mogących przyjąć wartość null. Jest to bardzo dobrze zaimplementowana właściwość języka. Dzięki niej można uniknąć wielu błędów, bardzo prosto wywołać metodę tylko jeżeli obiekt nie ma wartości null, a ponadto w wyrażeniach warunkowych następuje automatyczna konwersja na typ niemogący przyjąć wartości null jeżeli można tego dokonać. Te cechy języka pozwalają uniknąć poważnych błędów programistycznych[19]. W 2017 roku Google ogłosiło oficjalne wsparcie dla Kotlina na platformę Android[20]. Język ten posiada również rozbudowaną dokumentację, przykłady użycia oraz materiały do jego nauki[21].

3.1.3. SQLite

SQLite jest wbudowaną w system Android plikową bazą danych. Ten SZBD wspiera tylko kilka typów danych ze standardu SQL. Dzięki wykorzystaniu dynamicznych rozmiarów danych nie jest wymagane doprecyzowania i określania zakresów wartości[22]. Na systemie Android aplikacja nie potrzebuje mieć przyznanych przez użytkownika uprawnień do korzystania ze swojej bazy SQLite.

3.2. Implementacja dostępu do danych

Do łączenia z bazą danych aplikacji wykorzystano wchodząca w skład zestawu rozszerzeń Android Jetpack nakładkę na SQLite o nazwie Room[23]. Room zapewnia warstwę abstrakcji nad bezpośrednim wykorzystaniem SQLite, dzięki czemu dostęp do bazy danych jest dużo łatwiejszy, a mapowanie obiektowo-relacyjne przebiega dużo sprawniej i wymaga pisania mniej kodu. Zastosowana technologia na podstawie umieszczonych w kodzie adnotacji generuje kod języka Java, który pozwala na dostęp do danych oraz tworzy automatycznie wymagane tabele. Wykorzystanie Room w projekcie jest łatwe i wymaga tylko utworzenia kilku klas lub interfejsów z odpowiednimi adnotacjami:

- Klasa abstrakcyjna reprezentująca bazę danych.
- Interfejs oznaczony adnotacją `@Dao` zawierający metody pozwalające na dostęp do bazy danych. Podstawowe zapytania mogą zostać automatycznie wygenerowane, a bardziej zaawansowane można samodzielnie napisać w języku SQL. W zapytaniach można wykorzystać argumenty metod. Dobrą praktyką jest utworzenie po jednym takim interfejsie na encję.
- Oznaczona adnotacją `@Entity` klasa reprezentuje pojedynczą encję. W parametrach adnotacji można wprowadzić m.in. nazwę tabeli, klucze obce, indeksy.

```

@Entity(tableName = "Rooms",
    foreignKeys = [ForeignKey(entity = Project,
        indices = [Index(...value: "project_id")])
)
data class Room(
    @PrimaryKey(autoGenerate = true)
    val id: Int,
    val name: String,
    val floorArea: Double,
    val wallArea: Double,
    val perimeter: Double,
    val project_id: Int
) {

```

Rys. 25: Przykładowa definicja encji z wykorzystaniem adnotacji Room.

```

@Dao
interface RoomDao {
    @Insert
    suspend fun add(room: Room): Long

    @Update
    suspend fun update(room: Room): Int

    @Delete
    suspend fun delete(room: Room): Int

    @Query(value: "SELECT * FROM Rooms;")
    suspend fun getAll(): Array<Room>

    @Query(value: "SELECT * FROM Rooms WHERE id = :id;")
    suspend fun getById(id: Int): Room?
}

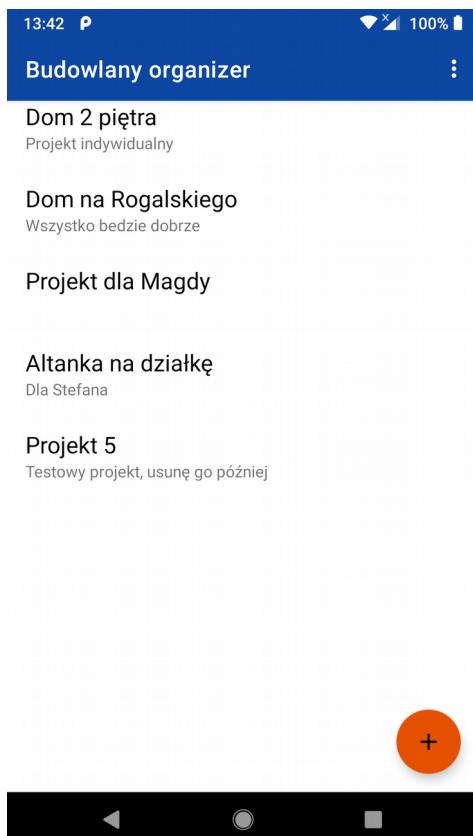
```

Rys. 26: Fragment definicji DAO dla encji Room. Widać automatycznie generowanie zapytania oraz samodzielnie napisane.

Rys. 25 przedstawia klasę Room, która reprezentuje pomieszczenie, przygotowaną do wykorzystania z Room. Jako parametry adnotacji `@Entity` można podać nazwę tabeli, zbiór kluczy obcych oraz zalecane przez środowisko indeksy. Ponadto można wskazać pole reprezentujące klucz główny i opcjonalnie określić czy ma być automatycznie generowany przez SZBD. Widoczny na Rys. 26 fragment programu przedstawia sposób opisywania obiektów DAO na potrzeby Room. Korzystając z języka Kotlin można dodatkowo opatrzyć metody zapytań słowem kluczowym `suspend`, dzięki czemu mogą być one wykonywane asynchronicznie za pomocą mechanizmu coroutine[24]. Taki sposób interakcji z bazą danych nie powoduje obciążania wątku renderującego interfejs użytkownika, a tym samym zapewnia responsywność aplikacji nawet w wypadku pobierania danych z serwera sieciowego. Dostęp do obiektu reprezentującego bazę danych jest dokonywany za pomocą wzorca singleton. Wzorzec ten został wykorzystany zgodnie z zaleceniami znajdującymi się w dokumentacji Androida, ponieważ utworzenie obiektu bazy danych jest kosztowne i zabiera dużo czasu[25][26].

3.3. Interfejs

W trakcie implementacji interfejsu użytkownika starano się kierować zasadami material design[27]. Android studio pozwala na bezpośrednie pobieranie dodatkowych ikon i grafik ze stron projektu. Ponadto na stronie [27] znajduje się duża biblioteka gotowych komponentów wykonanych w stylu material design. Komponenty te można bezproblemowo wykorzystać we własnej aplikacji, a środowisko programistyczne zajmie się pobraniem odpowiednich plików opisujących użyte elementy.



Rys. 27: Aktywność listy projektów.

Ekran przedstawiony na Rys. 27 przedstawia listę wszystkich projektów znajdujących się w bazie danych aplikacji. W prawym górnym rogu znajduje się rozwijane menu z możliwością dodania nowego projektu albo przejścia do listy wydatków globalnych. Pomarańczowy przycisk z symbolem plusa również pozwala na dodanie nowego projektu. Aktywność ta realizuje przypadek użycia „Przeglądaj projekty”.

13:43 100%
Nowy projekt

Nazwa projektu
projekt

Opis
testy

Data rozpoczęcia
16 października 2019

Data zakończenia
17 kwiecień 2021

ANULUJ DODAJ

13:44 100%
Edytuj projekt

Nazwa projektu
Dom 2 piętra

Opis
Projekt indywidualny

Data rozpoczęcia
22 sierpień 2019

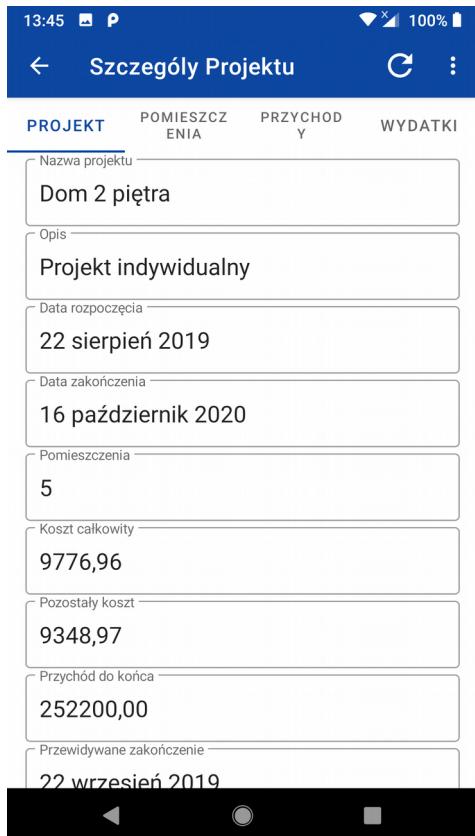
Data zakończenia
16 października 2020

ANULUJ EDYTUJ

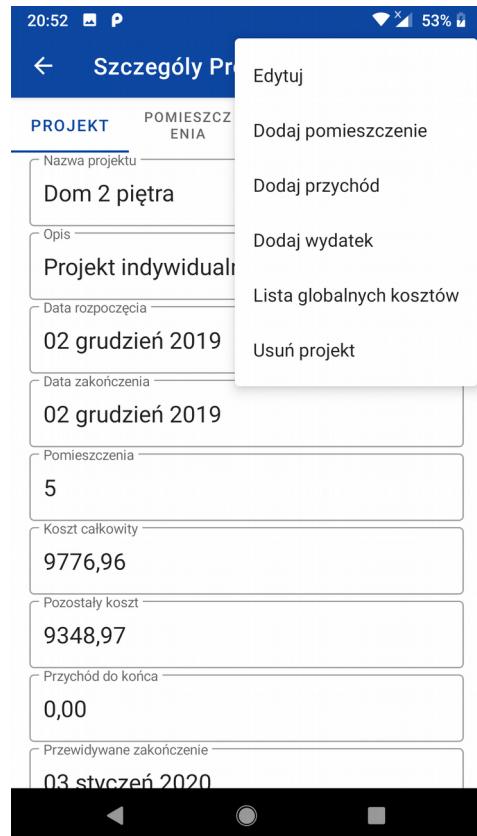
Rys. 28: Aktywność dodania nowego projektu.

Rys. 29: Aktywność edycji istniejącego projektu.

Ekran widoczny na Rys. 28 przedstawia interfejs dodania nowego projektu. Pola tekstowe pozwalają na wprowadzenie danych, a w wypadku dat pojawia się widżet kalendarza. Po przyciśnięciu przycisku „dodaj” aplikacja sprawdza poprawność danych i w razie konieczności informuje użytkownika o błędach. Aktywność ta realizuje przypadek użycia „Utwórz nowy projekt”. Rys. 29 przedstawia bardzo podobne wizualnie i funkcjonalnie okno do Rys. 28, jednak w tym przypadku służy ono do edycji istniejącego już projektu. Widok z Rys. 29 realizuje przypadek użycia „Edytuj projekt”.



Rys. 30: Aktywność widoku szczegółów projektu.



Rys. 31: Aktywność widoku szczegółów projektu z rozwiniętym menu.

Aktywność przedstawiona na Rys. 30 i Rys. 31 pokazuje interfejs widoku szczegółów projektu. Widoczne informacje pochodzą bezpośrednio z bazy danych albo są wyliczane na podstawie pozostałych rzeczy powiązanych z danym projektem. Widoczne na Rys. 31 menu zawiera opcje pozwalające na wykonanie wielu różnych akcji. Aktywność ta realizuje przypadek użycia „Przeglądaj szczegóły projektu”.

PROJEKT	POMIESZCZENIA	PRZYCHOD Y	WYDATKI
Kuchnia	14,000 m ²	30,000 m ²	10,000 m ²
	Wymiary: Koszt całkowity: 443,97		⋮
Salon	40,000 m ²	78,000 m ²	26,000 m ²
	Wymiary: Koszt całkowity: 575,00		⋮
Schowek	2,000 m ²	12,000 m ²	6,000 m ²
	Wymiary: Koszt całkowity: 0,00		⋮
Łazienka	8,000 m ²	12,000 m ²	6,000 m ²
	Wymiary: Koszt całkowity: 0,00		⋮
Sypialnia	10,500 m ²	12,000 m ²	8,400 m ²
	Wymiary: Koszt całkowity: 7,99		⋮

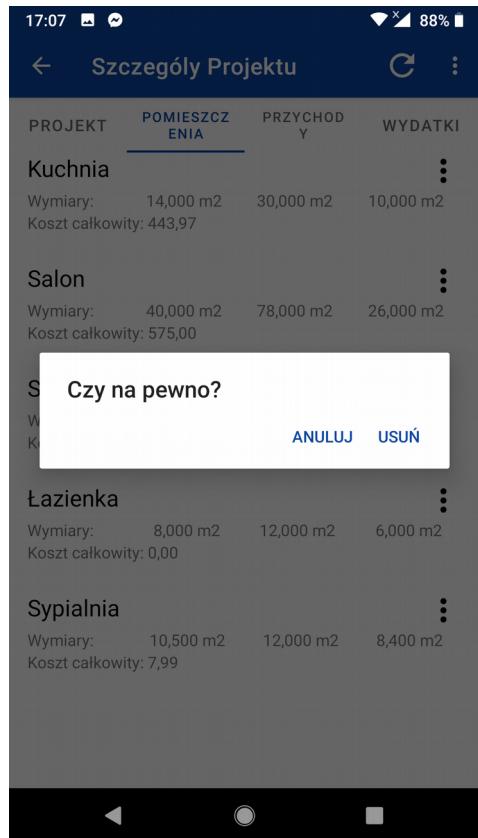


Rys. 32: Aktywność widoku listy pomieszczeń.

Ekran przedstawiony na Rys. 32 pokazuje listę pomieszczeń powiązanych z wybranym projektem. Każda pozycja listy składa się z nazwy pomieszczenia, jego wymiarów, kosztu obliczanego na podstawie powiązanych wydatków oraz menu rozwijanego. Pomiędzy oknami informacji o projekcie można się przemieszczać przyciskając zakładki lub przesuwając cały ekran palcem (swipe). Aktywność z Rys. 32 realizuje przypadek użycia „Przeglądaj pomieszczenia dotyczące wybranego projektu”.

PROJEKT	POMIESZCZENIA	PRZYCHOD Y	WYDATKI	
Kuchnia				⋮
Wymiary:	14,000 m2	30,000 m2	10,000 m2	
Koszt całkowity:	443,97			
Salon				⋮
Wymiary:	40,000 m2			
Koszt całkowity:	575,00			
Schowek				⋮
Wymiary:	2,000 m2	12,000 m2	6,000 m2	
Koszt całkowity:	0,00			
Łazienka				⋮
Wymiary:	8,000 m2	12,000 m2	6,000 m2	
Koszt całkowity:	0,00			
Sypialnia				⋮
Wymiary:	10,500 m2	12,000 m2	8,400 m2	
Koszt całkowity:	7,99			

Rys. 33: Aktywność widoku listy pomieszczeń z rozwiniętym menu.



Rys. 34: Aktywność widoku listy pomieszczeń z komunikatem usunięcia pomieszczenia.

Okno widoczne na Rys. 33 przedstawia listę pomieszczeń i co się stanie jak użytkownik rozwinie menu znajdujące się przy każdej pozycji. Wybranie opcji edytuj przeniesie do ekranu modyfikacji pomieszczenia, a opcja usuń wyświetli komunikat widoczny na Rys. 34. Jest to realizacja przypadku użycia „Usuń pomieszczenie”.

20:52 P 53% 53%

← Dodaj pomieszczenie

Nazwa pokoju

Powierzchnia

Powierzchnia ścian
|

Obwód

ANULUJ DODAJ

Rys. 35: Aktywność utworzenia nowego pomieszczenia.

20:52 P 53% 53%

← Edytuj pomieszczenie

Nazwa pokoju
Kuchnia

Powierzchnia
14,000

Powierzchnia ścian
30,000

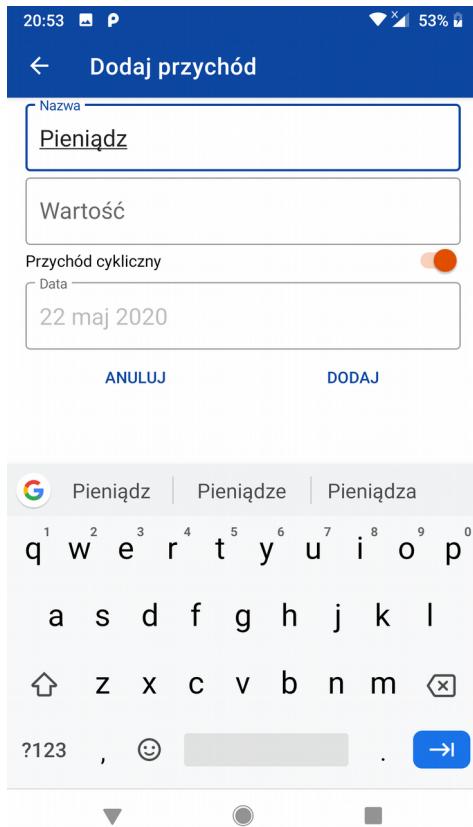
Obwód
10,000

ANULUJ EDYTUJ

Kuchnia | Kuchnią | Kuchniach
q w e r t y u i o p
a s d f g h j k l
z x c v b n m
?123 , . →

Rys. 36: Aktywność modyfikacji istniejącego pomieszczenia.

Okno widoczne na Rys. 35 pokazuje ekran dodania nowego pomieszczenia i realizuje przypadek użycia „Dodaj pomieszczenie”. Natomiast Rys. 36 pokazuje okno edycji pomieszczenia, które realizuje przypadek użycia „Edytuj pomieszczenie”. Oba widoki posiadają takie same pola służące do wpisywania wartości, jednak w przypadku edycji są one już wypełnione danymi pochodzącyimi z bazy danych.

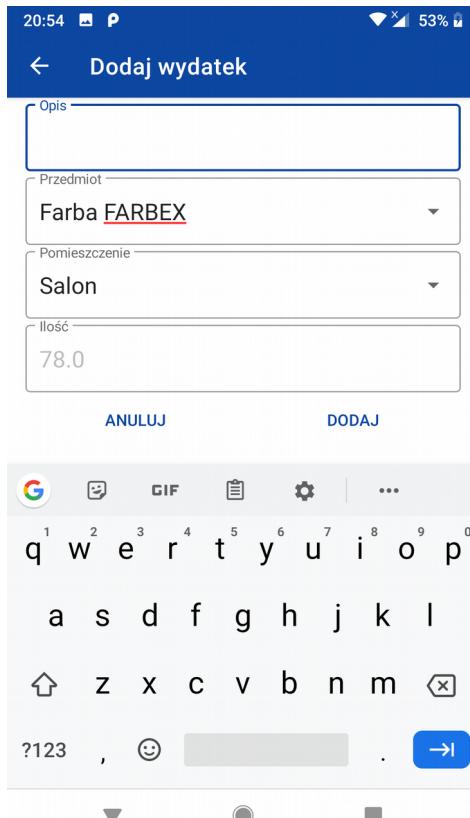


Rys. 37: Aktywność utworzenia nowego przychodu.

PROJEKT	POMIESZCZENIA	PRZYCHOD Y	WYDATKI
Pod choinkę	23000,00 zł	25 czerwiec 2020	⋮
Odsetki	200,00 zł	Co miesiąc	⋮
Moje	5200,00 zł	Co miesiąc	⋮
Karolina 15k	12000,00 zł	Co miesiąc	⋮
Nagroda za zlecenie	3000,00 zł	25 styczeń 2020	⋮

Rys. 38: Aktywność listy przychodów.

Ekran przedstawiony na Rys. 37 pokazuje interfejs dodania nowego przychodu do wybranego projektu. Użytkownik może podać nazwę, wartość pieniężną oraz datę pojawienia się tego przychodu. Ponadto jest możliwość wskazania, że środki będą wpływać cyklicznie jako pensja. Aktywność z Rys. 37 realizuje PU „Dodaj przychód”. Widoczny na Rys. 38 ekran pozwala na sprawdzenie listy przychodów powiązanych z danym projektem. Każda pozycja listy składa się z nazwy, wartości, określenia cykliczności lub jednorazowej daty wpłynięcia środków oraz rozwijanego menu. Menu pozwala na usunięcie wskazanej pozycji. Aktywność z Rys. 38 realizuje przypadki użycia „Przeglądaj przychody dotyczące wybranego projektu” oraz „Usuń przychód”.

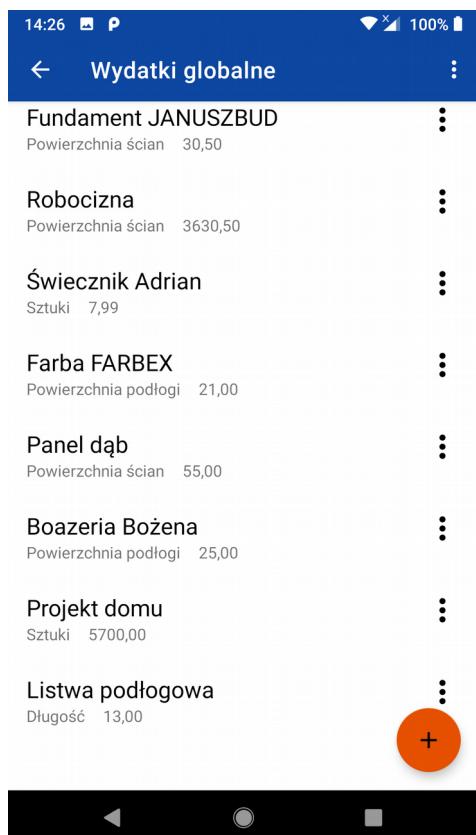


Rys. 39: Aktywność dodania nowego wydatku.

PROJEKT	POMIESZCZENIA	PRZYCHOD Y	WYDATKI
Fundament JANUSZBUD	DOM 100	3050,00zł	
	Fundamenty		
Świecznik Adrian	Kuchnia 3	23,97zł	
	Świeczniki dla Stacha		
Zrobione - Świecznik Adrian	Sypialnia 1	7,99zł	
	Świeczniki do sypialni		
Zrobione - Farba FARBEX	Kuchnia 20	420,00zł	
Panel dąb	Salon 10	550,00zł	
	Panel dąb fajny jest ale można bać panel ze sosny i		
	bedzie taniej :D		
Projekt domu	DOM 1	5700,00zł	
	Projekt od architekta		

Rys. 40: Aktywność listy wydatków.

Okno widoczne na Rys. 39 pokazuje interfejs dodania nowego wydatku do wybranego projektu. Użytkownik może podać opcjonalny opis, wybrać odpowiadający wydatek globalny, wskazać pomieszczenie lub przypisać wydatek do całego domu oraz podać ilość. Ponadto jeżeli aplikacja jest w stanie obliczyć potrzebną ilość to użytkownik nie musi jej podawać. Obliczenie opiera się na wymiarach wybranego pomieszczenia. Aktywność z Rys. 39 realizuje PU „Dodaj wydatek do projektu lub konkretnego pomieszczenia”. Widoczny na Rys. 40 ekran pozwala na sprawdzenie listy wydatków powiązanych z danym projektem. Każda pozycja listy składa się z nazwy powiązanego wydatku globalnego, konkretnego pomieszczenia lub wskazania na cały dom, kosztu całkowitego, opisu oraz rozwijanego menu. Menu pozwala na usunięcie wskazanej pozycji i wyświetla komunikat pytający czy użytkownik na pewno chce to zrobić. Aktywność z Rys. 40 realizuje przypadki użycia „Przeglądaj wydatki dotyczące wybranego projektu” oraz „Usuń wydatek”.



Rys. 41: Aktywność listy wydatków globalnych.

Widoczna na Rys. 41 aktywność wyświetla listę wydatków globalnych, a tym samym realizuje przypadek użycia „Przeglądaj wydatki globalne”. Menu w prawym górnym rogu oraz pomarańczowy przycisk pozwalają na dodanie nowego wydatku globalnego. Każda pozycja listy zawiera nazwę, cenę jednostkową oraz rodzaj jednostki, który określa sposób obliczania kosztu całkowitego. Menu znajdujące się po prawej każdej pozycji listy pozwala przejść do widoku edycji lub usunąć wybrany wydatek globalny.

The image consists of two side-by-side screenshots of a mobile application interface. Both screens have a blue header bar at the top with white text. The left screen's header reads "14:17" and "Dodaj wydatek globalny". The right screen's header reads "14:17" and "Edytuj wydatek globalny".

Left Screen (Rys. 42):

- Nazwa:** coś
- Rodzaj:** Sztuki
- Cena jednostkowa:** 235.36

Right Screen (Rys. 43):

- Nazwa:** Fundament JANUSZBUD
- Rodzaj:** Sztuki
- Cena jednostkowa:** 30,50

Both screens feature standard UI elements like "ANULUJ" (Cancel) and "DODAJ" (Add) buttons at the bottom left and right respectively, and a "EDYTUJ" (Edit) button at the bottom right.

Rys. 42: Aktywność dodania nowego wydatku globalnego.

Rys. 43: Aktywność edycji istniejącego wydatku globalnego.

Okno widoczne na Rys. 42 pokazuje ekran dodania nowego wydatku globalnego i realizuje przypadek użycia „Dodaj wydatek globalny”. Natomiast Rys. 43 pokazuje okno edycji wydatku globalnego, które realizuje przypadek użycia „Edytuj wydatek globalny”. Oba widoki posiadają takie same pola służące do wpisywania wartości, jednak w przypadku edycji są one już wypełnione danymi pochodzącyimi z bazy danych. Użytkownik może podać nazwę, cenę jednostkową oraz wybrać z listy rodzaj jednostki.

3.4. Inne szczegóły implementacji

Jedną z kluczowych rzeczy w trakcie implementacji aplikacji było trzymanie się zaproponowanej wcześniej architektury całości rozwiązania, która pozwala na łatwiejsze rozwijanie i utrzymywanie finalnego produktu. Każdy widok w aplikacji posiada odpowiadający mu prezenter, a komunikują się one przez interfejsy. Zastosowanie interfejsów pozwala na ewentualną bezproblemową wymianę komponentów.

```
interface EditProjectContract {  
    interface Presenter {  
        fun detach()  
        fun updateProject(project: Project): ValidationResult  
        fun getData(dataId: Int)  
    }  
  
    interface View {  
        fun show(project: Project)  
    }  
}
```

Rys. 44: Implementacja przykładowego protokołu dla prezentera i widoku.

Na Rys. 44 został przedstawiony interfejs jaki muszą spełniać prezenter i widok dla okna edycji projektu. W wypadku pozostałych par zaimplementowano podobne protokoły, jednak nie mają one wspólnego interfejsu bazowego, a przez to posiadają większą elastyczność.

```
class EditProjectPresenter(private var view: EditProjectContract.View?,
                           private val dao: ProjectDao)
    : CoroutineScope, EditProjectContract.Presenter
{
    private val job = Job()
    override val coroutineContext: CoroutineContext = job + Dispatchers.IO

    override fun detach() {
        job.cancel()
        view = null
    }

    override fun updateProject(project: Project): ValidationResult {
        val validator = ProjectValidator2(project)
        val validationResult = validator.validate()
        if(!validationResult.isError) {
            launch(Dispatchers.IO) { this: CoroutineScope
                dao.update(project)
            }
        }
        return validationResult
    }

    override fun getData(dataId: Int) {
        launch(Dispatchers.Main) { this: CoroutineScope
            val data = dao.getById(dataId)
            if(data != null) {
                view?.show(data)
            }
        }
    }
}
```

Rys. 45: Implementacja prezentera dla okna edycji projektu.

Widoczny na Rys. 45 kod źródłowy aplikacji przedstawia implementację prezentera dla okna edycji projektu. Metoda *getData* odpowiada za pobranie z bazy danych już istniejących danych i wypełnienie nimi interfejsu użytkownika. Wykonuje to przy pomocy przekazanego w konstruktorze obiektu DAO. Metoda *updateProject* na podstawie otrzymanych z widoku danych aktualizuje bazę danych pod warunkiem, że nowe dane przejdą proces sprawdzenia ich poprawności. Użytkownik w każdym wypadku dostaje informację zwrotną. Prezenter obsługuje również mechanizm coroutine, dlatego powinien posiadać możliwość przerwania operacji asynchronicznej jeżeli widok już nie potrzebuje danych, przykładowo na skutek zamknięcia okna. Taką rolę pełni metoda *detach*, która jest wywoływana w funkcji niszczącej widok.

```
class ProjectValidator(private val project: Project) {
    fun validate(): ValidationResult {
        return when {
            project.name.isEmpty() -> {
                ValidationResult(isError: true, R.string.validation_name_error)
            }
            project.endDate < project.beginDate -> {
                ValidationResult(isError: true, R.string.validation_date_error)
            }
            else -> ValidationResult(isError: false, R.string.edited)
        }
    }
}
```

Rys. 46: Przykładowa implementacja validatora.

Widoczna na Rys. 46 klasa odpowiada z sprawdzenie poprawności danych wprowadzonych przez użytkownika aplikacji. Jedyna metoda sprawdza kolejne warunki błędu i jeżeli przekazany obiekt nie spełnia ich to zostaje uznany za poprawny. Obiekt będący wynikiem sprawdzenia poprawności zawiera informację czy wystąpił błąd oraz zasób tekstowy reprezentujący informację zwrotną. Nie wykorzystano łańcuchów znakowych ponieważ nie podlegają one tłumaczeniu w aplikacji, a dzięki użyciu numerów identyfikacyjnych zasobów możliwe jest wyświetlenie komunikatu w języku preferowanym przez użytkownika.

3.5. Testy

Istotnym elementem zapewniania wysokiej poprawności i utrzymywalności wytwarzanego produktu jest przygotowanie testów. W tym celu można wykonać testy jednostkowe, integracyjne, z użytkownikiem oraz wiele innych. W trakcie prac implementacyjnych przygotowano odpowiednie testy, które miały pilnować zgodności aplikacji ze specyfikacją i projektem. W celu wykonania testów jednostkowych wykorzystano bibliotekę JUnit[28] oraz Mockito[29], które odpowiada za tworzenie obiektów imitujących rzeczywistą implementację. Przygotowano również automatyczne testy interfejsu użytkownika przy pomocy biblioteki Espresso[30].

```

@Test
fun projectValidator_project_begin_greater_than_end() {
    val validator = ProjectValidator2(Project( id: 0, name: "Domek mój", description: "Domek mój"))
    val result = validator.validate()
    assertTrue(result.isError)
}

@Test
fun projectValidator_project_all_wrong() {
    val validator = ProjectValidator2(Project( id: 0, name: "", description: ""))
    val result = validator.validate()
    assertTrue(result.isError)
}

@Test
fun room_total_area_good() {
    assertEquals(Room( id: 1, name: "Kuchnia", floorArea: 20.0, wallArea: 54.0, doorArea: 0.0))
}

```

Rys. 47: Fragment kodu kilku testów.

Widoczny na Rys. 47 fragment kodu źródłowego aplikacji przedstawia kilka prostych testów jednostkowych. Każdy przypadek testowy jest opatrzony odpowiednią adnotacją, dzięki czemu zostaje on uruchomiony przez środowisko wykonawcze testów. W ogólnym przypadku każdy test składa się z testowanej jednostki oraz sprawdzenia czy rezultat wykonania jest identyczny z pewnym znanym wynikiem lub spełnia inne zależności. Istotnym zagadnieniem na jakie należy zwrócić uwagę jest odpowiednie pokrycie wielu kombinacji możliwych argumentów funkcji. Programista piszący testy jednostkowe musi wziąć pod uwagę wartości typowe, rzadkie, błędne oraz skrajne. Testy te są niezależne od platformy Android.

```

@Test
fun add_new_good() {
    onView(withId(R.id.input_name)).perform(typeText("Test dodania projektu"))
    onView(withId(R.id.input_description)).perform(typeText("Testy :D"))
    onView(withId(R.id.input_date_end)).perform(closeSoftKeyboard(), scrollTo(), click())
    onView(withClassName(Matchers.equalTo(DatePicker::class.java.name))).perform(PickerAction)
    onView(withText("OK")).perform(click())
    onView(withId(R.id.button_accept)).perform(click())
    assertTrue(activityRule.activity.isFinishing)
}

@Test
fun add_new_error() {
    onView(withId(R.id.input_description)).perform(typeText("Testy :D"))
    onView(withId(R.id.button_accept)).perform(click())
    onView(withText(R.string.add_project_fail)).inRoot(withDecorView(not(
        activityRule.activity.window.decorView))).check(matches(isDisplayed()))
}

```

Rys. 48: Fragment kodu testów interfejsu.

Przedstawiony na Rys. 48 fragment kodu pokazuje dwa testy automatyczne interfejsu użytkownika dla okna dodania nowego projektu. Wykonanie tych testów wymaga działania w środowisku Android, gdzie automat samodzielnie wykonuje podane akcje. Zadaniem każdego takiego testu jest zasymulowanie możliwych sekwencji akcji podejmowanych przez użytkownika aplikacji i sprawdzenie czy ich wynik jest zgodny z spodziewanym.

Espresso umożliwia przygotowanie testów w bardzo prosty sposób. Można wyszukiwać elementy interfejsu, wykonywać kilka podstawowych akcji oraz sprawdzać wartości atrybutów i porównać je ze spodziewanymi. Takie testy pozwalają na pokrycie prawie wszystkich możliwych interakcji i sprawdzenie czy zawsze działają tak samo. Zaleta testów automatycznych jest też odciążenie testera z pracy oraz ich powtarzalność. Niestety przygotowując takie testy jesteśmy ograniczeni do możliwości na jakie pozwala wykorzystana biblioteka.

Aplikacja w trakcie implementacji przeszła też przez wiele testów manualnych. Testy manualne polegają na sprawdzaniu działania interfejsu użytkownika samodzielnie przez człowieka. Dodanie każdej nowej funkcjonalności czy zmiana w interfejsie musiała zostać sprawdzona i przebadana pod kątem poprawności działania, a w wypadku elementów interfejsu użytkownika pod kątem wyglądu i dopasowania do innych kontrolek.

4. Zakończenie

4.1. Perspektywy rozwojowe aplikacji

Aplikacja zawiera wszystkie wymagane funkcjonalności, jednak w obecnej formie może nadal być rozwijana o nowe funkcje i poprawiana. Możliwe jest też polepszenie jakości użytkowania produktu. Poniższa lista zbiera kilka z nich:

- Wyszukiwanie – wszystkie listy w aplikacji powinny zawierać opcję filtrowania i wyszukiwania. Dzięki temu nawigacja została ułatwiona.
- Synchronizacja z serwerem chmurowym – możliwe jest wykorzystanie bazy danych, która nie znajduje się na urządzeniu użytkownika. Takie rozwiązanie zapewniłoby łatwą synchronizację pomiędzy wieloma urządzeniami oraz opcję wymieniania się projektami. Niestety takie podejście ma też wady. Wykorzystanie serwera własnego lub od zewnętrznego dostawcy wiąże się z poniesieniem kosztów ich działania. W celu opłacenia takiego rozwiązania w aplikacji mogłyby pojawić się reklamy, opłaty za korzystanie z aplikacji lub płatna wersja premium zawierająca możliwość trzymania danych na serwerach.
- Poradnik budowlany – zbiór porad, pułapek możliwych przy budowie, porady i informacje prawne. Obecnie jest to realizowane poprzez początkowe wypełnienie bazy danych najważniejszymi wydatkami globalnymi.
- Przydatne zasoby – linki do przydatnych stron i artykułów.
- Lepsza szata graficzna – aplikacja znacznie by skorzystała na lepszej i czytelniejszej oprawie wizualnej. Dodanie specjalnie przygotowanych ikon, grafik oraz logo zdecydowanie zwiększyłoby profesjonalizm wykonania produktu.

4.2. Podsumowanie

Aplikacja w obecnej formie spełnia prawie wszystkie wymagania funkcjonalne przedstawione w treści pracy. Nie dokonano implementacji modyfikacji przychodów i wydatków, ponieważ ich usunięcie i ponowne utworzenie jest proste i nie pociąga za sobą żadnych dalszych usunięć czy innych operacji. Kolejną rzeczą nad jaką należałoby popracować jest przyjemność z użytkowania aplikacji, czyli user experience. W tym względzie bardzo pomogłyby zaimplementowanie wyszukiwania i filtrowania widoków list. Dodatkowo przydałyby się grafiki i animacje, które poprawiłyby wygląd interfejsu użytkownika. Aplikacja jest zdatna do użytku i może okazać się przydatna przy budowie domu jednorodzinnego. Ponadto dzięki dużej elastyczności można ją wykorzystać w innych celach, jak na przykład remonty. W trakcie prac implementacyjnych znaczna część aplikacji została zrealizowana zgodnie z wcześniej przygotowanym projektem. W trakcie prac nad projektem i implementacją praktycznie wykorzystano zdobytą wcześniej wiedzę oraz nauczono się wielu nowych rzeczy przydatnych w dalszej pracy zawodowej. Tak powstała aplikacja możliwe konkurować z obecnie istniejącymi rozwiązaniami na rynku. Jedną z głównych zalet jest jej elastyczność i znajdowanie się zawsze pod ręką, dzięki byciu aplikacją mobilną.

Bibliografia

- [1] Główny Urząd Nadzoru Budowlanego, „Ruch budowlany w 2018 r.”. [Online]. Adres: <https://www.gunb.gov.pl/aktualnosc/ruch-budowlany-w-2018-r>. [Dostęp: 9.12.2019]
- [2] Infor Biznes Sp z o.o., „Europa wybiera bloki. Tymczasem większość Polaków mieszka w... domach jednorodzinnych”. [Online]. Adres: <https://nieruchomosci.dziennik.pl/news/artykuly/605121,polska-domy-bloki-nieruchomosci-mieszkanie-statystyki.html>. [Dostęp: 9.12.2019]
- [3] Kalkulatory Budowlane, „Koszty budowy domu w ciągu ostatnich 3 lat wzrosły prawie o 20%”. [Online]. Adres: <https://kb.pl/porady/koszty-budowy-domu-w-ciagu-ostatnich-3-lat-wzrosly-prawie-o-14/>. [Dostęp: 9.12.2019]
- [4] Kalkulatory Budowlane, „Koszt budowy domu - kalkulator”. [Online]. Adres: <https://kb.pl/budowa/>. [Dostęp: 9.12.2019]
- [5] Archipelag.pl, „mobiDOM”. [Online]. Adres: <http://mobi-dom.pl/>. [Dostęp: 9.12.2019]
- [6] V. Ng, „Non-Functional Requirement of the Mobile Development system.”. [Online]. Adres: <https://medium.com/@vishwasng/non-functional-requirement-of-the-mobile-development-system-e0ed98f2a872>. [Dostęp: 9.12.2019]
- [7] Google and Open Handset Alliance, „Distribution dashboard”. [Online]. Adres: <https://developer.android.com/about/dashboards/index.html>. [Dostęp: 9.12.2019]
- [8] A. Klusiewicz, „Baza SQLite w androidzie”. [Online]. Adres: <http://andrzejklusiewicz-android.blogspot.com/2014/02/baza-sqlite-w-androidzie.html>. [Dostęp: 9.12.2019]
- [9] M. Jaggavarapu, „Presentation Patterns : MVC, MVP, PM, MVVM”. [Online]. Adres: <https://manojjaggavarapu.wordpress.com/2012/05/02/presentation-patterns-mvc-mvp-pm-mvvm/>. [Dostęp: 9.12.2019]
- [10] E. Maxwell, „The MVC, MVP, and MVVM Smackdown”. [Online]. Adres: <https://academy.realm.io/posts/eric-maxwell-mvc-mvp-and-mvvm-on-android/>. [Dostęp: 9.12.2019]
- [11] F. Cervone, „Model-View-Presenter: Android guidelines”. [Online]. Adres: <https://medium.com/@cervonefrancesco/model-view-presenter-android-guidelines-94970b430ddf>. [Dostęp: 9.12.2019]
- [12] R. Martin, „Design Principles and Design Patterns”. [Online]. Adres: https://web.archive.org/web/20150906155800/http://www.objectmentor.com/resources/articles/Principles_and_Patterns.pdf. [Dostęp: 9.12.2019]
- [13] Object Management Group, „OMG Unified Modeling Language”. [Online]. Adres: <https://www.omg.org/spec/UML/2.2/Superstructure/PDF/>. [Dostęp: 9.12.2019]
- [14] G. Larsen, „SQL Server: Natural Key Verses Surrogate Key”. [Online]. Adres: <https://www.databasejournal.com/features/mssql/article.php/3922066/SQL-Server-Natural-Key-Verses-Surrogate-Key.htm>. [Dostęp: 9.12.2019]
- [15] X. Ducrohet, „Android Studio: An IDE built for Android ”. [Online]. Adres: <https://android-developers.googleblog.com/2013/05/android-studio-ide-built-for-android.html>. [Dostęp: 9.12.2019]
- [16] Google and Open Handset Alliance, „Android Studio”. [Online]. Adres: <https://developer.android.com/studio>. [Dostęp: 9.12.2019]

- [17] S. Webber, „Learn Kotlin in Y minutes”. [Online]. Adres: <https://learnxinyminutes.com/docs/kotlin/>. [Dostęp: 9.12.2019]
- [18] JetBrains, „Kotlin”. [Online]. Adres: <https://kotlinlang.org/>. [Dostęp: 9.12.2019]
- [19] T. Hoare, „Null References: The Billion Dollar Mistake”. [Online]. Adres: <https://www.infoq.com/presentations/Null-References-The-Billion-Dollar-Mistake-Tony-Hoare/>. [Dostęp: 9.12.2019]
- [20] M. Shafirov, „Kotlin on Android. Now official”. [Online]. Adres: <https://blog.jetbrains.com/kotlin/2017/05/kotlin-on-android-now-official/>. [Dostęp: 9.12.2019]
- [21] JetBrains, „Kotlin Reference”. [Online]. Adres: <https://kotlinlang.org/docs/reference/>. [Dostęp: 9.12.2019]
- [22] SQLite Consortium, „SQLite HomePage”. [Online]. Adres: <https://www.sqlite.org/index.html>. [Dostęp: 9.12.2019]
- [23] Google and Open Handset Alliance, „Room Persistence Library”. [Online]. Adres: <https://developer.android.com/topic/libraries/architecture/room>. [Dostęp: 9.12.2019]
- [24] Google and Open Handset Alliance, „Accessing data using Room DAOs”. [Online]. Adres: <https://developer.android.com/training/data-storage/room/accessing-data#kotlin-coroutines>. [Dostęp: 9.12.2019]
- [25] Google and Open Handset Alliance, „Save data in a local database using Room”. [Online]. Adres: <https://developer.android.com/training/data-storage/room>. [Dostęp: 9.12.2019]
- [26] Google and Open Handset Alliance, „Add a Room database”. [Online]. Adres: <https://codelabs.developers.google.com/codelabs/android-room-with-a-view-kotlin/index.html?index=..%2F..index#6>. [Dostęp: 9.12.2019]
- [27] Google, „Material Design”. [Online]. Adres: <https://material.io/>. [Dostęp: 9.12.2019]
- [28] The JUnit Team, „JUnit”. [Online]. Adres: <https://junit.org/junit5/>. [Dostęp: 9.12.2019]
- [29] S. Faber, „Mockito”. [Online]. Adres: <https://site.mockito.org/>. [Dostęp: 9.12.2019]
- [30] Google and Open Handset Alliance, „Espresso”. [Online]. Adres: <https://developer.android.com/training/testing/espresso>. [Dostęp: 9.12.2019]

Spis rysunków

Rys. 1: Wykres przedstawiający rodzaj lokalu mieszkalnego w Polsce. Źródło: [2].....	2
Rys. 2: Model domenowy powstały w wyniku analizy wycinka rzeczywistości jakim jest budowa domu.....	6
Rys. 3: Diagram przypadków użycia dla przedstawionych wymagań.....	9
Rys. 4: Diagram aktywności dla przypadku użycia „Dodaj wydatek globalny”.....	11
Rys. 5: Diagram aktywności dla przypadku użycia „Edytuj pomieszczenie”.....	12
Rys. 6: Diagram aktywności dla przypadku użycia „Usuń przychód”.....	13
Rys. 7: Diagram architektury fizycznej aplikacji.....	14
Rys. 8: Diagram przedstawiający relacje pomiędzy klasami w wzorcu MVP. Źródło: [9]..	15
Rys. 9: Diagram klas dla bytów biznesowych.....	16
Rys. 10: Uogólniony diagram klas dla architektury MVP.....	17
Rys. 11: Szczegółowy diagram klas dla widoku listy pomieszczeń.....	17

Rys. 12: Diagram klas validatorów sprawdzających poprawność danych wprowadzanych przez użytkownika.....	18
Rys. 13: Projekt ekranu listy projektów.....	19
Rys. 14: Prototyp ekranu dodania nowego projektu.....	20
Rys. 15: Prototyp ekranu modyfikacji istniejącego projektu.....	21
Rys. 16: Prototyp ekranu szczegółów projektu.....	22
Rys. 17: Prototyp ekranu listy pomieszczeń.....	23
Rys. 18: Diagram przepływu nawigacji.....	24
Rys. 19: Diagram ERD dla wykonanej bazy danych.....	25
Rys. 20: Diagram sekwencji dla dodania wydatku globalnego.....	26
Rys. 21: Diagram sekwencji dla edycji projektu.....	27
Rys. 22: Diagram sekwencji dla usunięcia pomieszczenia.....	28
Rys. 23: Przykładowe okno środowiska Android Studio.....	30
Rys. 24: Przykład okna aplikacji w graficznym edytorze środowiska Android Studio.....	30
Rys. 25: Przykładowa definicja encji z wykorzystaniem adnotacji Room.....	32
Rys. 26: Fragment definicji DAO dla encji Room. Widać automatycznie generowanie zapytania oraz samodzielnie napisane.....	32
Rys. 27: Aktywność listy projektów.....	33
Rys. 28: Aktywność dodania nowego projektu.....	34
Rys. 29: Aktywność edycji istniejącego projektu.....	34
Rys. 30: Aktywność widoku szczegółów projektu.....	35
Rys. 31: Aktywność widoku szczegółów projektu z rozwiniętym menu.....	35
Rys. 32: Aktywność widoku listy pomieszczeń.....	36
Rys. 33: Aktywność widoku listy pomieszczeń z rozwiniętym menu.....	37
Rys. 34: Aktywność widoku listy pomieszczeń z komunikatem usunięcia pomieszczenia.	37
Rys. 35: Aktywność utworzenia nowego pomieszczenia.....	38
Rys. 36: Aktywność modyfikacji istniejącego pomieszczenia.....	38
Rys. 37: Aktywność utworzenia nowego przychodu.....	39
Rys. 38: Aktywność listy przychodów.....	39
Rys. 39: Aktywność dodania nowego wydatku.....	40
Rys. 40: Aktywność listy wydatków.....	40
Rys. 41: Aktywność listy wydatków globalnych.....	41
Rys. 42: Aktywność dodania nowego wydatku globalnego.....	42
Rys. 43: Aktywność edycji istniejącego wydatku globalnego.....	42
Rys. 44: Implementacja przykładowego protokołu dla prezentera i widoku.....	43
Rys. 45: Implementacja prezentera dla okna edycji projektu.....	44
Rys. 46: Przykładowa implementacja walidatora.....	45
Rys. 47: Fragment kodu kilku testów.....	46
Rys. 48: Fragment kodu testów interfejsu.....	46