 **Streszczenie**

Celem pracy jest projekt oraz implementacja aplikacji mobilnej do zarządzania zadaniami. Projekt oparto o przeanalizowanie popularności omawianego typu aplikacji, audyt istniejących rozwiązań oraz zestawienie niezbędnych narzędzi umożliwiających realizację pracy. Użyto także prototypowania interfejsu, w celu zaprezentowania środowiska dla użytkownika oraz designu. Część implementacyjną oparto o szczegółowy opis wdrożonej relacyjnej bazy danych, dokładny opis przypadków użycia oraz zestawienie problemów implementacyjnych, które autor napotkał w trakcie pracy. Założenia projektowe i implementacyjne zostały zrealizowane. Efektem pracy jest działająca aplikacja, służąca użytkownikowi do realizowania systemu zarządzania zadaniami oraz monitorowania ich terminowości.

**Abstract**

The purpose of this thesis is to create a project and implement an application for task management. Project is based on rigid analysis of task management popularity, inspection of existing solutions and collation of necessary development tools. Author therefore use a prototype of interface in order to show user’s visual environment and application design. Implementation is based on detailed description of created relational database, user cases and encountered implementation difficulties. Every kind of assumptions are realized. Application is working and providing advanced experience to user by allowing to manage user’s goals and task with monitoring them.

**Spis treści**

[Streszczenie 3](#_Toc436568156)

[**Rozdział 1 Wstęp** 5](#_Toc437187023)

[1.1 Wprowadzenie do problematyki oraz geneza pracy. 5](#_Toc437187024)

[1.2 Cel pracy 5](#_Toc437187025)

[**Rozdział 2 Stan wiedzy i techniki w zakresie tematyki pracy** 6](#_Toc437187026)

[2.1 Wprowadzenie 6](#_Toc437187027)

[2.2 Przegląd istniejących rozwiązań 6](#_Toc437187028)

[2.3 Przegląd przydatnych technik i technologii 8](#_Toc437187029)

[2.4 Podsumowanie 10](#_Toc437187030)

[**Rozdział 3 Wymagania projektowe** 11](#_Toc437187031)

[3.1 Użytkownicy 11](#_Toc437187032)

[3.2 Funkcjonalności aplikacji 11](#_Toc437187033)

[3.3 Środowisko i dostępność aplikacji 11](#_Toc437187034)

[**Rozdział 4 Założenia projektowe** 13](#_Toc437187035)

[4.1 Przedmiot pracy 13](#_Toc437187036)

[4.2 Wymagania funkcjonalne 13](#_Toc437187037)

[4.3 Wymagania niefunkcjonalne 13](#_Toc437187038)

[4.4 Opis podstawowej architektury systemu 14](#_Toc437187039)

[4.5 Sposób realizacji 14](#_Toc437187040)

[**Rozdział 5 Projekt** 14](#_Toc437187041)

[5.1 Wstęp 14](#_Toc437187042)

[5.2 Przypadki użycia 15](#_Toc437187043)

[5.3 Interfejs 21](#_Toc437187044)

[5.4 Wzorce architektoniczne. 27](#_Toc437187045)

[5.5 Baza danych 28](#_Toc437187046)

[5.6 Podsumowanie 29](#_Toc437187047)

[**Rozdział 6 Implementacja** 30](#_Toc437187048)

[6.1 Wprowadzenie 30](#_Toc437187049)

[6.2 Wykorzystane środowiska oraz narzędzia programistyczne 30](#_Toc437187050)

[6.3 Implementacja bazy danych 30](#_Toc437187051)

[6.4 Interfejs użytkownika 31](#_Toc437187052)

[6.5 Problemy implementacyjne 37](#_Toc437187053)

[6.6 Dalsze plany rozwoju aplikacji 43](#_Toc437187054)

[**Rozdział 7 Podsumowanie** 44](#_Toc437187055)

[**Bibliografia** 45](#_Toc437187056)

[**Spis rysunków** 46](#_Toc437187057)

[**Spis tabel** 47](#_Toc437187058)

# **Rozdział 1 Wstęp**

## 1.1 Wprowadzenie do problematyki oraz geneza pracy.

Rozpoczęcie projektu zaczyna się od planowania. Realizując zamierzone cele każdego dnia podchodzimy do sprawy w taki sposób, aby uzyskać maksymalną optymalizację podejmowanych działań i uzyskać jak najmniejszy koszt czasu. Zależy nam także na uzyskaniu minimalizacji błędów w podejmowanych decyzjach oraz posiadania czynnika zabezpieczającego nas przed porzuceniem istotnych zadań lub zwyczajnym zapomnieniem. Rzeczy tak błahe jak lista zakupów czy bardziej złożone jak opracowanie interfejsu dla aplikacji wymuszają na nas stworzenie chronologicznego ciągu zdarzeń, połączonych pewną logiką. Realizacja każdego kroku daje nam pewność, że idziemy we właściwym kierunku oraz możemy oszacować, czy nasze wysiłki są w stanie zmieścić nas w wyznaczonych przedziale czasowym. Aby osiągnąć tego typu założenia, autor zdecydował się na stworzenie aplikacji na systemy mobilne Android. Sposób jej realizacji jest domyślnie dostosowany dla każdego zainteresowanego uporządkowaniem codziennej pracy.

Geneza pracy opiera się na osobistych doświadczeniach autora z udziału w projekcie dla wrocławskiej firmy IT, realizowanego wraz z zespołem programistów. Zauważono, że im większa świadomość podejmowanych kroków i decyzji implementacyjnych, tym szybciej można zauważyć błędy w logice oraz poprawić podstawowe założenia dla danego problemu na samym początku pracy. Wykorzystanie tego sposobu planowania do codziennych obowiązków znacznie ułatwiła kontrolę pracy i naukę na uczelni. Okazało się, że wiele osób podchodzi do planowania w podobny sposób i aplikacja mobilna z pewnością będzie cieszyć się szerokim zainteresowaniem.

## 1.2 Cel pracy

Cel pracy obejmuje zaprojektowanie oraz zaimplementowanie prototypu aplikacji służącej do zarządzania zadaniami. Implementacja została oparta o język programowania Java oraz o wykorzystanie plików xml w środowisku Android. Rozwiązanie to ma dostarczyć pakiet funkcjonalności dla użytkownika, służących do zapisywania wybranych celów i zadań oraz monitorowania ich terminowości. Zadanie jest rozumiane jako element na liście, posiadający informacje o nazwie oraz i priorytecie wykonania. Dodatkowo każde zadanie może zostać rozbudowane o podzadania, służące za kolejne punkty odniesienia i informacje w toku realizacji głównego punktu. Dodatkowo została uwzględniona możliwość tworzenia i wybierania odpowiedniej kategorii dla realizowanego zadania. Tym sposobem użytkownik aplikacji ma do dyspozycji wyczerpującą formę zarządzania zadaniami.

# **Rozdział 2** **Stan wiedzy i techniki w zakresie tematyki pracy**

## 2.1 Wprowadzenie

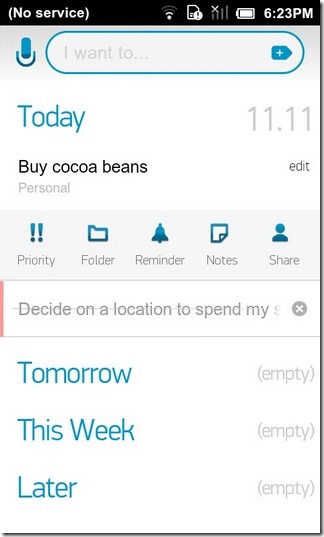
Aplikacje służące do zarządzania zadaniami określa się jako „Todo App”. Obecnie można znaleźć dość szeroki wybór programów zapewniających tego typu jakość usług. Jednakże temat zarządzania zadaniami i czasem jest mocno zindywidualizowany i nie istnieje jeden uniwersalny i optymalny sposób przekazania do rąk użytkownika aplikacji spełniającej wszelkie oczekiwania.

## 2.2 Przegląd istniejących rozwiązań

Analizując gotowe rozwiązania zwróciłem uwagę na dość popularny obecnie Todoist, następnie Any.do, Wunderlist. W pierwszej kolejności skupiłem się na charakterystyce Todoist (Rys.2.1, Rys.2.2). Mimo wielu przydatnych funkcji(drukowanie listy zadań) oraz kompatybilności z Windows Phone, aplikacja ta posiada szereg wad i niedogodności. Todoist wymaga od nas zakładania konta w celu każdorazowego logowania się przy korzystaniu z aplikacji. Aplikacja autora w stanie zapewnić łatwiejszy i szybszy dostęp do listy zadań dla użytkownika, w szczególności biorąc pod uwagę lokalny jej charakter. Dodatkowo system sortowania w Todoist nie sprawdza się tak, jak oczekuje się tego od podręcznej aplikacji. Warto jednak zaznaczyć, że bardzo dobrze zrealizowany jest Material Design, szczególnie widoczny w rozwijanym menu.

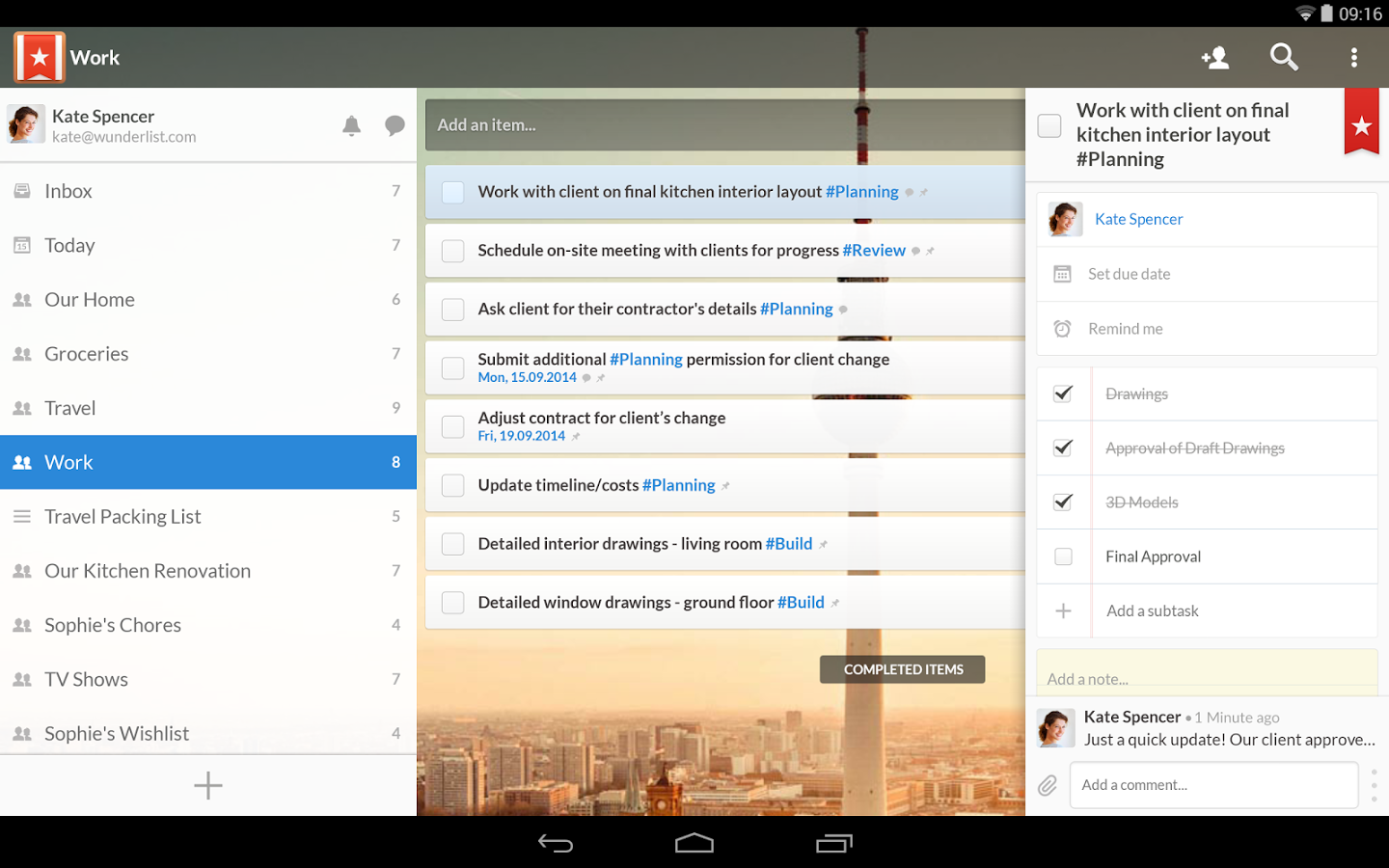
|  |  |
| --- | --- |
| C:\Users\Tric\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Screenshot_2015-11-22-14-25-14.png  Rys.2.1 Widok podsatwowy aplikacji Todoist | C:\Users\Tric\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Screenshot_2015-11-22-14-25-21.png  Rys.2.2 Widok menu aplikacji Todoist |

Kolejna aplikacja o której wspomniano wcześniej to Any.do (Rys.3.1). Cechuje się ona przejrzystym interfejsem oraz przyjazną usługa synchronizacji danych ze standardowej aplikacji kalendarza. Największą zaletą jest łatwy dostęp do elementów interfejsu.



Rys.2.3 Widok podsatwowy aplikacji Any.do

Ostatnią omawianą aplikacją będzie Wunderlist. Podobnie to poprzedników, aplikacja ta posiada prosty i łatwy w obsłudze interfejs. W przypadku programów tego typu, jest to najważniejsza cecha. Dodatkowo Wunderlist posiada kompatybilność z różnymi systemami mobilnymi oraz umożliwia tworzenie grup pracujących nad danymi zadaniami. Grupy te mogą być tworzone poprzez zaproszenia przez Facebook lub zwyczajnie przez emaila. Do dużch minusów zalicza się brak zoptymalizowanej współpracy z klawiaturą oraz ograniczona możliwość dostosowywania aplikacji dla potrzeb użytkowników. Dodatkowo Wunderlist zarzuca się brak przejrzystości w przypadku przeglądania zadań oraz podzadań. Jest to wyjątek dotyczący ogólnych zalet jego interfejsu.



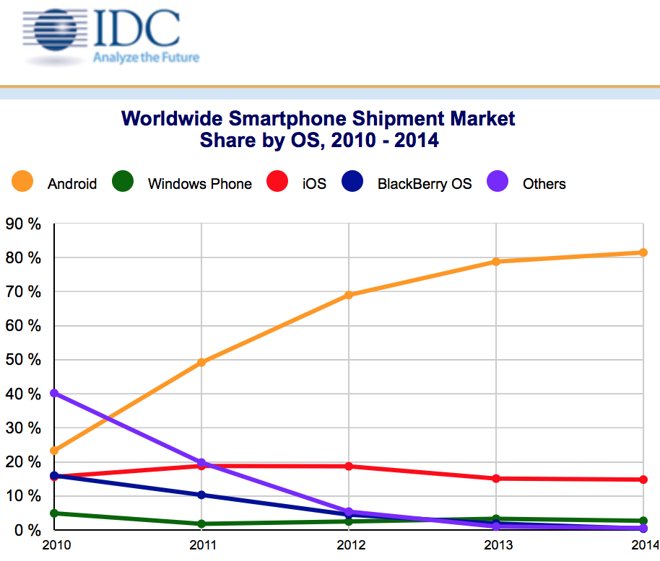
Rys.2.4: Przykład widoku z Wunderlist.

Źródło: http://www.softwarecrew.com/wp-content/uploads/2014/08/wunderlist3.png

Podsumowując, wymienione wyżej aplikacje posiadają szereg cech, które powinny sugerować kompletną i właściwie zaimplementowaną aplikację do zarządzania zadaniami. Jednakże biorąc pod uwagę docelowy profil odbiorcy, autor chciałby dostosować swoje rozwiązanie w bardziej przystępny i łatwiejszy do obsługi sposób. Skupiono się na dostosowaniu listy zadań do formy łatwej w aktualizacji i manipulacji danymi oraz ograniczono ilość notyfikacji, które niejednokrotnie mogą zirytować użytkownika (co jest widoczne np. w Todoist).

## 2.3 Przegląd przydatnych technik i technologii

Aktualnie Android jest najpopularniejszym systemem operacyjnym dla urządzeń mobilnych. Jego udziały na tym rynku wciąż rosną, więc można prognozować, że trend popularności w najbliższym czasie nie ulegnie pogorszeniu. [Jillian D'Onfro](http://www.businessinsider.com/author/jillian-donfro) [12] posłużyła się porównaniem dwóch wiodących marek na rynku mobilnym. Porównanie Androida oraz iOS prowadzi do interesujących wniosków. Wciąż iOS nie jest w stanie zagwarantować map jakościowo zbliżonych do tych, udostępnionych przez Google. Biorąc pod uwagę istotność nawigacji dzisiaj, jest to wielki minus. Dodatkowo użytkownicy Androida posiadają uniwersalne rozwiązanie co do ładowania ich urządzeń. Rodzaj kabla do ładowarki jest kompatybilny praktycznie z każdym urządzeniem opartym o system Android. Konkurent nie posiada takiego udogodnienia. Oczywistym czynnikiem przemawiającym za projektowaniem aplikacji z myślą o Android, jest cena. Urządzenia mobilne z systemem iOS są znacznie droższe niż ich odpowiedniki z systemem opracowanym przez Google.



Rys.2.5: Udział mobilnych systemów operacyjnych na rynku[13].

Autor artykułu [13], zamieszczając wykres udziału systemów mobilnych na rynku, przedstawił obowiązujący trend, dotyczący przewagi Androida nad innymi systemami. Analizując powyższe dane dla aplikacji wybrano system opracowany przez Google. Rozwiązanie to, ma na celu zagwarantowanie większego zainteresowania aplikacją w przyszłości. Biorąc także pod uwagę liczne grono użytkowników systemu Android istnieje większe prawdopodobieństwo uzyskania odpowiednich narzędzi i pomocy służących do zrealizowania celu pracy.

Spośród dostępnych narzędzi wybrane zostało środowisko developerskie Android Studio. Jest ono darmowe oraz zapewnia niezbędny pakiet narzędzi umożliwiających wygodne pisanie aplikacji. Ponadto autor posiada wiedzę zdobytą w trakcie realizowania toku nauczania na wydziale Informatyki i Zarządzania, dotyczącą pracy z systemami mobilnymi. Zajęcia laboratoryjne wykorzystywały funkcjonalność środowiska Android Studio. Na drodze implementacyjnej najczęściej wykorzystywaną funkcją była funkcja debugowania. Android Studio udostępnia to narzędzie w sposób bardzo łatwy i logiczny. Istnieje możliwość ewaluowania wyrażeń kodu w momencie zatrzymania na ustawionym wcześniej punkcie zatrzymania. Warto też zaznaczyć, że Android Studio pozwala użytkownikowi na dostęp do większości plików i katalogów znajdujących się na urządzeniu. Tym sposobem programista jest w stanie pobrać np. stworzoną bazę danych aplikacji i przetestować ją w innym środowisku programistycznym. Operacja ta nie jest domyślnie możliwa z poziomu powłoki konsolowej w Android Studio oraz z poziomu przeglądania plików na urządzeniu.

Baza danych została stworzona w oparciu o SQLite. Samo w sobie środowisko Android Studio nie pozwala na podgląd bazy, więc wykorzystałem nakładkę na Firefox w postaci SQLite Manager w celu inspekcji stworzonej bazy danych i analizy wprowadzanych rekordów. Zastosowanie i przykładową implementację omówiono w książce [3] w rozdziale 9. Autor książki opisał trzy istotne zalety zastosowanie SQLite.

Pierwsza cecha dotyczy jego dostępności. Jest on w pełni darmowy, stąd nie istnieje jakikolwiek problem z prawami autorskimi. Silnik SQLite jest też stosunkowo niewielki, zajmuje 150 KB. Ostatnia, a zarazem najciekawsza cecha, to brak wymagania konfiguracji oraz administrowania silnikiem.

Styl graficzny aplikacji oparto o motyw Material Design. Jest to styl opracowany i zaprezentowany przez Google w 2014 roku. Opiera się on o wiele czynników związanych z interfejsem np. wzbogacenie palety kolorów, cienie. Material Design wyznacza tak naprawdę nową drogę do tworzenia interfejsów i wyglądów dla aplikacji. Wspomina o tym autor artykułu [11], dodając także, że w przyszłości wszelkie aplikacje i programy tworzone przez Google będą oparte o ten styl graficzny. Pierwszym rzucającym się w oczy elementem, jest forma wykorzystania przycisku w kształcie plusa. Ma on za zadania informować od razu użytkownika tym, że w tym miejscu będzie w stanie stworzyć lub dodać coś nowego. Warto posłużyć się także artykułem [10], w którym to autor podkreśla zastosowanie Material Design jako wzorca dla przyszłych aplikacji. Zmiana ta będzie widoczna w urządzeniach takich jak telewizor, komputer, tablet oraz na wszelkich platformach korzystających z Androida. Autor artykułu mówi także, o rosnącym trendzie wśród programistów, dotyczącym zastosowania Material Design w implementowanych aplikacjach. Sposób wykorzystania tej formy designu można łatwo zauważyć na urządzeniach mobilnych korzystających z Androida w wersji 5.0 i większej.

Na potrzeby implementacji posłużono się darmowymi ikonami dostępnymi na stronie internetowej <http://findicons.com/>.

## 2.4 Podsumowanie

Wybierając zaprezentowaną technologię autor stworzył warunki konieczne do sprawnego zaimplementowania aplikacji jak i całej logiki towarzyszącej realizacji zadań. W momencie szukania właściwych technologii należy zwracać szczególną uwagę na założenia projektowe oraz trzeba uważnie śledzić aktualne zachowania wśród docelowych użytkowników. Jeżeli istotnym czynnikiem tworzonej aplikacji jest jej sukces i nastawienie na dotarcie do jak najszerszego grona, należy dokładnie rozważyć sposoby dotarcia to osób mogących zainteresować się programem. To założenie jest spełnione poprzez wykorzystanie najpopularniejszego systemu operacyjnego dla urządzeń mobilnych. Android nie tylko jest najpowszechniejszy, ale okazuje się, że urządzenia oparte o ten system są w zasięgu finansowym większości osób.

# **Rozdział 3 Wymagania projektowe**

## 3.1 Użytkownicy

Bazowo aplikacja jest przeznaczona dla indywidualnego użytkownika, który posiada dostęp do programu jedynie na urządzeniu mobilnym z systemem Android. Aplikacja jest zbudowana lokalnie na urządzeniu użytkownika, stąd istnieje ograniczenie w przenoszeniu danych między urządzeniami tego samego typu.

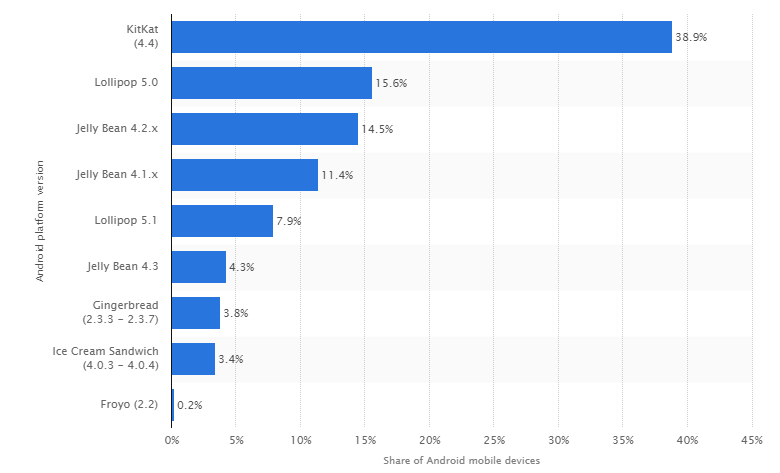
Podsumowując, użytkownikiem jest każda osoba, która posiada aplikację na swoim urządzeniu mobilnym.

## 3.2 Funkcjonalności aplikacji

1. Najważniejszą funkcjonalnością aplikacji jest dodawanie zadań stworzonych przez użytkownika do listy zadań posortowanej według priorytetu zadania. Dodawanie jest zrealizowane w formie wypełniania formularza informującego o poszczególnych składowych danych zadania.
2. Dodawanie kategorii zadań oraz ich możliwe edytowanie.
3. Dodawanie podzadań wybranego zadania w celu uszczegółowienia.
4. Przedstawienie zadań i podzadań w formie rozszerzalnej listy z wyróżnionymi polami do finalizowania podzadań oraz ikonami sugerującymi stan realizacji zadania.
5. Sortowanie listy według wybranej kategorii, co pozwala na łatwiejszy dostęp do interesujących nas informacji.
6. Dynamiczne zmienianie statusu zadania na podstawie zrealizowanych podzadań.
7. System notyfikacji informujących o pozostałym czasie na wykonanie zadania oraz wskazujących na zadania o przekroczonym czasie wykonania.

## 3.3 Środowisko i dostępność aplikacji

Aplikacja została zaprojektowana z myślą o użytkownikach urządzeń mobilnych z systemem operacyjnym Android. Kluczowym elementem okazał się dobór odpowiedniego poziomu API dla mojej aplikacji.



Rys.3.3 Udział procentowy wersji systemów Android

Źródło: <http://www.statista.com/statistics/271774/share-of-android-platforms-on-mobile-devices-with-android-os/>

Wykres pochodzi ze strony statista.com i jest datowany na 5 października 2015. Na podstawie informacji z diagramu postanowiłem zbudować swoją aplikację na poziomie API już od wersji 15(jest to Ice Cream Sandwich).

Docelowy poziom API to 22, czyli urządzenia już z systemem Lolipop 5.1 będą współpracować z aplikacją. Dość łatwo zauważyć, że powyższy zabieg pozwoli na szerszy dostęp do użytkowników, co może skutkować zwiększoną popularnością w finalnej wersji udostępnionej na Google Play.

Aplikacja będzie dostępna na platformie Google Play, jako bezpłatna.

# **Rozdział 4 Założenia projektowe**

## 4.1 Przedmiot pracy

Przedmiotem pracy jest wykonanie projektu oraz implementacja aplikacji służącej użytkownikowi jak menadżer jego prywatnych celów i zadań. Rozwiązanie to ma dostarczyć łatwy w obsłudze pakiet operacji, służący do zapisywania oraz prezentowania zadań wraz z informacjami tyczący się sposobu ich realizacji. Poprzez sposób realizacji autor rozumie stosowanie systemu notyfikacji monitorujących terminowość użytkownika oraz pozwalających w szybki sposób odnaleźć najistotniejsze zadania. Samo zadanie przyjmuje formę elementu wyświetlonego na liście zadań, posortowanej według priorytetu danego zadania.

## 4.2 Wymagania funkcjonalne

1. Wprowadzenie kategorii do systemu.

2. Edycja kategorii oraz możliwość ich usuwania.

3. Walidacja usuwania kategorii. Usuwanie możliwe jedynie wtedy, kiedy kategoria nie posiada żadnych zadań.

4. Wprowadzenie zadania do systemu.

5. Edycja oraz usuwanie zadania z systemu. Walidacja na pola obligatoryjne przy zapisywaniu zadania oraz edytowaniu.

6. Wprowadzenie podzadań do stworzonego zadania.

7. Edycja oraz usuwanie podzadań.

8. System zmienia status zadania na ukończone, jeżeli wszystkie podzadania są zrealizowane.

9. System stosuje ikony przy zadaniach do informowania o statusie.

10. System oblicza czas do realizacji zadania oraz podaje informacje o pozostałym czasie.

11. System informuje o zadaniach, których czas realizacji przekroczył wyznaczony termin.

## 4.3 Wymagania niefunkcjonalne

1. Aplikacja jest przewidziana dla pojedynczego, indywidualnego użytkownika.

2. Baza danych jest zbudowana lokalnie na urządzeniu.

3. Kompatybilność aplikacji dotyczy urządzeń od wersji Androida na poziome API 15(Ice Cream Sandwich).

## 4.4 Opis podstawowej architektury systemu

Aplikacja zostanie zbudowana z myślą o indywidualnym użytkowniku. Nie wymaga ona przechowywania osobistych informacji o posiadaczu urządzenia mobilnego, stąd nie istnieje też konieczność logowania oraz rejestrowania użytkownika. W dużym stopniu ogranicza to zużycie pamięci urządzenia, co czyni moją aplikację atrakcyjniejszą dla urządzeń o słabszych parametrach. Baza danych jest zbudowana lokalnie, stąd dostęp do danych jest możliwy jedynie z docelowego urządzenia mobilnego.

## 4.5 Sposób realizacji

Aplikacja od początku będzie tworzona w środowisku developerskim Android Studio. Do inspekcji oraz testowania bazy danych wykorzystam nakładkę na przeglądarkę Firefox w postaci SQLite Manager. Udostępnia ona wygodny podgląd wprowadzonych danych oraz możliwość edytowania rekordów, w celu weryfikacji typów danych. Architekturę systemu oparłem o wzorzec MVC. Tym sposobem autor stworzył implementację zachowującą standard DAO. Każdy model posiada odpowiedni wzorzec DAO, który ułatwia i optymalizuje pracę oraz operacje na bazie danych w dalszym etapie logiki aplikacji. Ten standard posiada też istotnie cechy, które pozwalają na ograniczenie ilości kodu oraz zachowanie transakcyjnego sposobu realizowania operacji. Jest to bardzo popularna praktyka, widoczna także w sposobach implementacji strony internetowej. Rozdzielanie logiki w mniejszych aplikacjach nie jest tak istotne jak w przypadku dużych projektów, jednakże zachowanie czytelności kodu oraz przejrzystości umożliwi w przyszłości łatwy powrót do opracowywanego projektu w postaci wdrażania kolejnych funkcjonalności.

# **Rozdział 5 Projekt**

## 5.1 Wstęp

W rozdziale poświęconym projektowi autor przedstawił i opisał przypadki użycia realizowane w ramach aplikacji. Zostały one zaprojektowane w oparciu o rozdział 7 książki [5]. Zaprezentowano także prototyp interfejsu oraz opisano zaimplementowaną bazę danych wraz z opisem użytego wzorca architektonicznego. Opisano także, dlaczego sposób zbudowania kluczowych tabeli został zrealizowany w dany sposób.

# 

## 5.2 Przypadki użycia

Rys.4.1: Diagram przypadków użycia

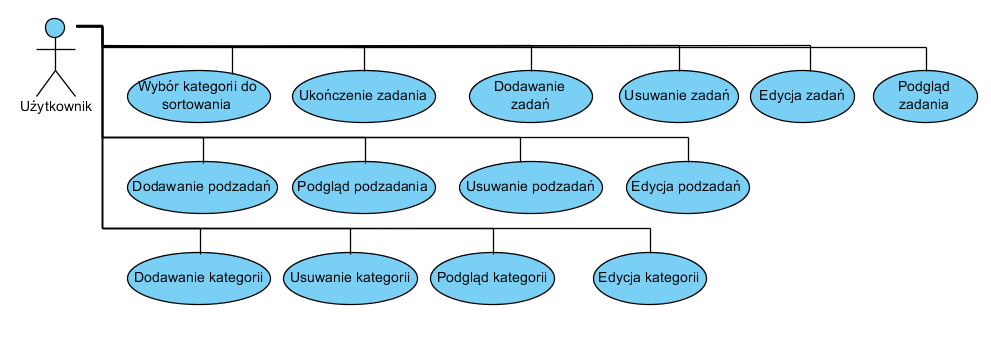


Tabela 5.1: PU-1 - Dodawanie kategorii.

|  |  |
| --- | --- |
| Lp. | PU-1 |
| Nazwa: | Dodawanie kategorii. |
| Warunki początkowe: | Nie może już istnieć kategoria w bazie o wprowadzanej nazwie. |
| Opis: | Użytkownik chce dodać kategorię do listy kategorii. |
| Ścieżka główna: | 1. Użytkownik wypełnia pola dla kategorii.  2. Użytkownik kończy dodawanie przyciskiem „Save”.  3. Aplikacja tworzy kategorię o nazwie podanej przez użytkownika. |
| Ścieżka alternatywna: | Użytkownik otrzymuje komunikat o błędzie spowodowanym błędnie wypełnionym formularzem kreatora lub dotyczącym próby dodania kategorii o istniejącej już nazwie. |
| Warunki końcowe: | Nowa kategoria została dodana. |

Tabela 5.2: PU-2 - Edycja kategorii.

|  |  |
| --- | --- |
| Lp. | PU-2 |
| Nazwa: | Edycja kategorii. |
| Warunki początkowe: | Użytkownik znajduje się w oknie podglądu kategorii oraz wybrał opcję edycji. |
| Opis: | Użytkownik chce edytować wybraną kategorię. |
| Ścieżka główna: | 1. Użytkownik zmienia wybrane wartości dla pól edytora kategorii.  2. Użytkownik kończy edytowanie przyciskiem „Save”.  3. Aplikacja nadpisuje dane kategorii. |
| Ścieżka alternatywna: | Użytkownik otrzymuje komunikat o błędzie spowodowanym błędnym wypełnieniem pól kategorii. |
| Warunki końcowe: | Kategoria zostaje edytowana. |

Tabela 5.3: PU-3 - Usuwanie kategorii.

|  |  |
| --- | --- |
| Lp. | PU-3 |
| Nazwa: | Usuwanie kategorii. |
| Warunki początkowe: | Użytkownik znajduje się w oknie podglądu kategorii. |
| Opis: | Użytkownik chce usunąć wybraną kategorię z bazy danych. |
| Ścieżka główna: | 1. Użytkownik usuwa kategorię poprzez przycisk „Delete”. |
| Ścieżka alternatywna: | Użytkownik otrzymuje komunikat o błędzie spowodowanym próbą usunięcia kategorii posiadającej istniejące zadania. |
| Warunki końcowe: | Kategoria zostaje usunięta z bazy danych. |

Tabela 5.4: PU-4 - Wybór kategorii do sortowania.

|  |  |
| --- | --- |
| Lp. | PU-4 |
| Nazwa: | Wybór kategorii do sortowania. |
| Warunki początkowe: | Użytkownik znajduje się w widoku listy zadań. |
| Opis: | Użytkownik chce posortować listę zadań według kategorii. |
| Ścieżka główna: | 1. Użytkownik sortuje listę zadań poprzez wybranie odpowiedniej kategorii ze spinnera. |
| Ścieżka alternatywna: |  |
| Warunki końcowe: | Lista zadań zostanie posortowana. |

Tabela 5.5: PU-5 - Podgląd kategorii.

|  |  |
| --- | --- |
| Lp. | PU-5 |
| Nazwa: | Podgląd kategorii. |
| Warunki początkowe: | Użytkownik znajduje się w widoku listy kategorii. |
| Opis: | Użytkownik chce podejrzeć wybraną kategorię i zobaczyć jej szczegóły. |
| Ścieżka główna: | 1. Użytkownik przechodzi do podglądu kategorii poprzez kliknięcie wybranej kategorii na liście. |
| Ścieżka alternatywna: |  |
| Warunki końcowe: |  |

Tabela 5.6: PU-6 - Podgląd podzadania.

|  |  |
| --- | --- |
| Lp. | PU-6 |
| Nazwa: | Podgląd podzadania. |
| Warunki początkowe: | Użytkownik znajduje się w widoku listy zadań oraz rozwinął szczegółowy widok wybranego zadania. |
| Opis: | Użytkownik chce podejrzeć szczegóły wybranego podzadania |
| Ścieżka główna: | 1. Użytkownik poprzez longClick na wybranym podzadaniu otwiera kontekstowe menu.  2. Użytkownik wybiera odpowiednią pozycję z kontekstowego menu.  3. Użytkownik przechodzi do widoku wybranego podzadania. |
| Ścieżka alternatywna: |  |
| Warunki końcowe: |  |

Tabela 5.7: PU-7 - Podgląd zadania.

|  |  |
| --- | --- |
| Lp. | PU-7 |
| Nazwa: | Podgląd zadania. |
| Warunki początkowe: | Użytkownik znajduje się w widoku listy zadań. |
| Opis: | Użytkownik chce podejrzeć szczegóły wybranego zadania |
| Ścieżka główna: | 1. Użytkownik poprzez longClick na wybranym zadaniu otwiera kontekstowe menu.  2. Użytkownik wybiera odpowiednią pozycję z kontekstowego menu.  3. Użytkownik przechodzi do widoku wybranego podzadania. |
| Ścieżka alternatywna: |  |
| Warunki końcowe: |  |

Tabela 5.8: PU-8 - Ukończenie podzadania.

|  |  |
| --- | --- |
| Lp. | PU-8 |
| Nazwa: | Ukończenie podzadania. |
| Warunki początkowe: | Użytkownik znajduje się w widoku listy zadań oraz rozwinął szczegółowy widok wybranego zadania. |
| Opis: | Użytkownik chce zakończyć wybrane podzadanie. |
| Ścieżka główna: | 1.Użytkownik kończy podzadanie poprzez zaznaczenie checkboxa.  2. Aplikacja uaktualnia bazę danych. |
| Ścieżka alternatywna: |  |
| Warunki końcowe: |  |

Tabela 5.9: PU-9 - Ukończenie zadania.

|  |  |
| --- | --- |
| Lp. | PU-9 |
| Nazwa: | Ukończenie zadania. |
| Warunki początkowe: | Użytkownik znajduje się w widoku listy zadań. |
| Opis: | Użytkownik chce zakończyć zadanie niezależnie od tego czy posiada podzadania czy nie. |
| Ścieżka główna: | 1. Użytkownik poprzez longClick otwiera kontekstowe menu na wybranym zadaniu.  2. Użytkownik wybiera odpowiednią pozycję z menu kontekstowego i kończy zadanie.  3. Aplikacja uaktualnia bazę danych. |
| Ścieżka alternatywna: |  |
| Warunki końcowe: | Zadanie w bazie danych otrzymuje status ukończonego a ewentualne podzadania zostają ukończone. |

Tabela 5 10: PU-10 - Usunięcie zadania.

|  |  |
| --- | --- |
| Lp. | PU-10 |
| Nazwa: | Usunięcie zadania. |
| Warunki początkowe: | Użytkownik znajduje się w widoku podglądu zadania. |
| Opis: | Użytkownik chce usunąć wybrane zadanie z listy zadań. |
| Ścieżka główna: | 1. Użytkownik poprzez przycisk „Delete” usuwa zadanie z listy zadań.  2. Aplikacja uaktualnia bazę danych. |
| Ścieżka alternatywna: |  |
| Warunki końcowe: | Zadanie wraz z ewentualnymi podzadaniami zostaje usunięte z bazy danych. |

Tabela 5.11: PU-11 - Edycja zadania.

|  |  |
| --- | --- |
| Lp. | PU-11 |
| Nazwa: | Edycja zadania. |
| Warunki początkowe: | Użytkownik znajduje się w widoku listy zadań. |
| Opis: | Użytkownik chce edytować dane o zadaniu. |
| Ścieżka główna: | 1. Użytkownik poprzez longClick na wybranym zadaniu otwiera menu kontekstowe.  2. Użytkownik wybiera odpowiednią pozycję z menu kontekstowego i przechodzi do podglądu zadania.  3. Użytkownik zmienia wybrane pola oraz zapisuje edytowane zadanie poprzez przycisk „Save”.  4. Aplikacja uaktualnia bazę danych. |
| Ścieżka alternatywna: | Użytkownik otrzymuje komunikat o błędzie spowodowany niepoprawnym wypełnieniem lub modyfikacją pól dla zadania. |
| Warunki końcowe: | Zadanie zostaje zaktualizowane o wprowadzone nowe dane. |

Tabela 5.12: PU-12 - Dodawanie zadania.

|  |  |
| --- | --- |
| Lp. | PU-12 |
| Nazwa: | Dodawanie zadania. |
| Warunki początkowe: | Użytkownik znajduje się w widoku listy zadań. |
| Opis: | Użytkownik chce dodać zadanie do listy zadań. |
| Ścieżka główna: | 1. Użytkownik poprzez ikonę dodawania przechodzi do widoku dodawania zadania.  2. Użytkownik wypełnia pola dla zadania oraz zapisuje je poprzez przycisk „Save”.  3. Aplikacja uaktualnia bazę danych. |
| Ścieżka alternatywna: | Użytkownik otrzymuje komunikat o błędzie spowodowany niepoprawnym wypełnieniem pól dla zadania. |
| Warunki końcowe: | Zadanie zostaje zapisane w bazie danych. |

Tabela 5.13: PU-13 - Dodawanie podzadania.

|  |  |
| --- | --- |
| Lp. | PU-13 |
| Nazwa: | Dodawanie podzadania. |
| Warunki początkowe: | Użytkownik znajduje się w widoku listy zadań. |
| Opis: | Użytkownik chce dodać podzadanie do wybranego zadania. |
| Ścieżka główna: | 1. Użytkownik poprzez longClick na wybranym zadaniu otwiera kontekstowe menu.  2. Poprzez wybranie odpowiedniej pozycji z menu kontekstowego użytkownik przechodzi do widoku dodawania podzadania.  3. Użytkownik wypełnia pola formularza dodawania podzadania oraz zapisuje je poprzez przycisk „Save”.  4. Aplikacja zapisuje wprowadzone podzadanie. |
| Ścieżka alternatywna: | Użytkownik otrzymuje komunikat o błędzie spowodowany błędnym wprowadzeniem danych do formularza dodawania podzadania. |
| Warunki końcowe: | Podzadanie zostaje zapisane w bazie danych. |

Tabela 5.14: PU-14 - Edycja podzadania.

|  |  |
| --- | --- |
| Lp. | PU-14 |
| Nazwa: | Edycja podzadania. |
| Warunki początkowe: | Użytkownik znajduje się w widoku listy zadań. |
| Opis: | Użytkownik chce edytować podzadanie dla wybranego zadania. |
| Ścieżka główna: | 1. Użytkownik poprzez longClick na wybranym podzadaniu otwiera menu kontekstowe.  2. Poprzez wybranie odpowiedniej pozycji z menu kontekstowego użytkownik przechodzi do widoku podglądu podzadania.  3. Po wyborze opcji edytowania użytkownik zmienia wartości w polach podzadania oraz zapisuje zmiany poprzez przycisk „Save”.  4. Aplikacja dodaje podzadanie do bazy danych. |
| Ścieżka alternatywna: | Użytkownik otrzymuje komunikat o błędzie spowodowany błędnym wprowadzeniem wartości do pól w oknie edytowania podzadania. |
| Warunki końcowe: | Wybrane podzadanie zostaje zmodyfikowane i zaktualizowane w bazie danych. |

Tabela 5.15: PU-15 - Usuwanie podzadania.

|  |  |
| --- | --- |
| Lp. | PU-15 |
| Nazwa: | Usuwanie podzadania. |
| Warunki początkowe: | Użytkownik znajduje się w widoku listy zadań. |
| Opis: | Użytkownik chce usunąć podzadanie z wybranego zadania. |
| Ścieżka główna: | 1. Użytkownik poprzez longClick na wybranym podzadaniu otwiera menu kontekstowe.  2. Poprzez wybranie odpowiedniej pozycji z menu kontekstowego użytkownik przechodzi do widoku podglądu podzadania.  3. Poprzez przycisk „Delete” użytkownik usuwa podzadanie z bazy danych.  4. Aplikacja usuwa podzadanie. |
| Ścieżka alternatywna: |  |
| Warunki końcowe: | Wybrane podzadanie zostaje usunięte z bazy danych. |

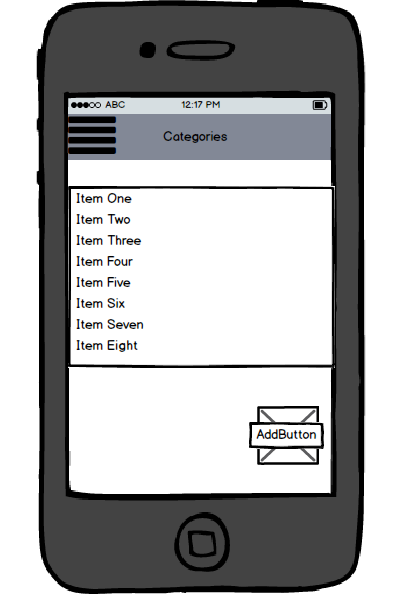
## 5.3 Interfejs

Budując interfejs dla aplikacji mobilnej opartej na systemie Android należy zwrócić uwagę na zastosowanie podstawowych elementów widoków tj. Aktywności oraz Fragmentów.

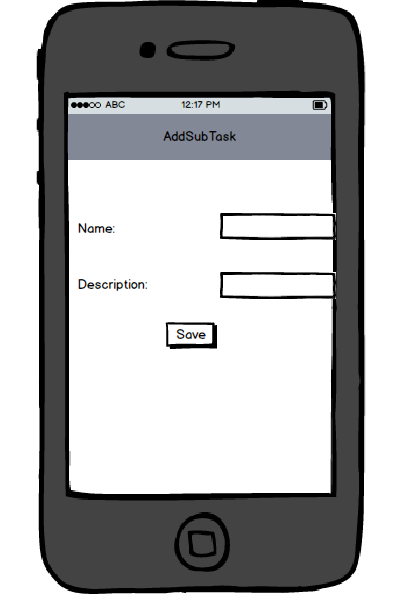
Omawiając Aktywności chciałbym posłużyć się informacjami zawartymi w książce *Android w praktyce*[1], dotyczącymi zastosowania Aktywności. Autorzy książki w podrozdziale 2.6 opisują Aktywność jako definicja widocznego ekranu oraz wszelkich komponentów znajdujących się na nim. Innymi słowy, tworzy ona interfejs umożliwiający kontakt pomiędzy użytkownikiem a widocznymi zasobami na ekranie. Aktywność może dotyczyć wielu widoków i często jest indywidualna dla każdego ekranu. Zarządza ona także cyklem życia, obsługuje zdarzenia na interfejsie oraz odpowiada za kontrolę menu. Na stronie internetowej[8] Cooper Bruce opisuje intencję jako swoistą wiadomość wysyłaną przez system operacyjny Android do właściwych elementów interfejsu telefonu. Zabieg ten umożliwia komunikację różnych aplikacji. Najprostszy przykład użycia intencji to start aplikacji oraz wywołanie innej aktywności w ramach aplikacji poprzez przesyłanie intencji.

Aktywność poza reagowaniem na zdarzenia interfejsu musi jednocześnie odpowiednio kreować na ekranie zaimplementowane rozwiązanie. Jest to dość mocno obciążający proces dla telefonu, dlatego stworzono możliwość oddzielenia części logicznej od wizualnej interfejsu. Tym kluczowym czynnikiem okazują się Fragmenty, odpowiadające za odciążenie aktywności oraz za odpowiednie renderowanie widoku dla użytkownika.

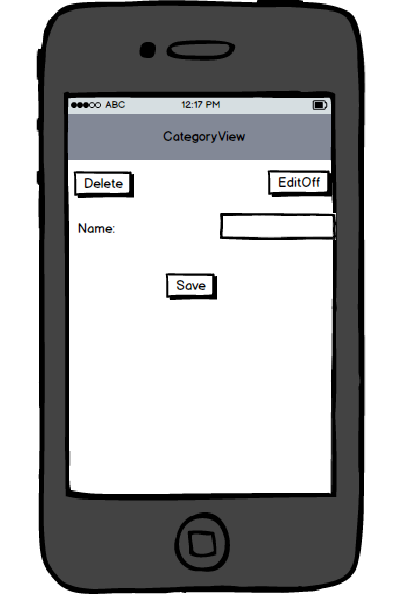
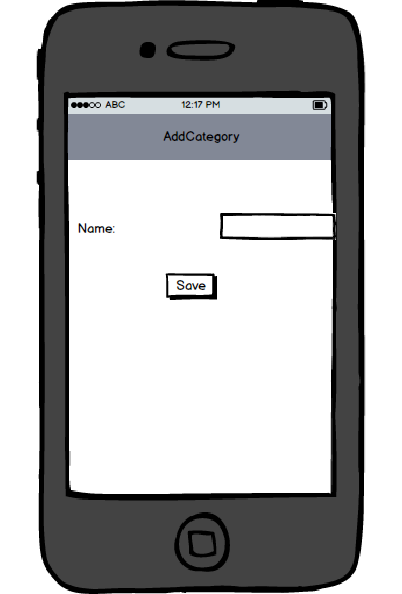
Poniżej autor zamieścił prototypy interfejsu dla aplikacji.



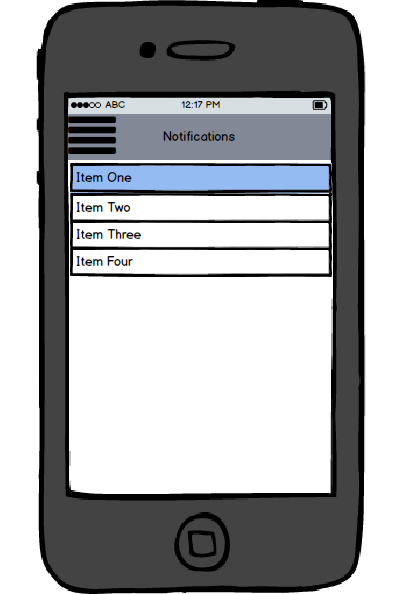
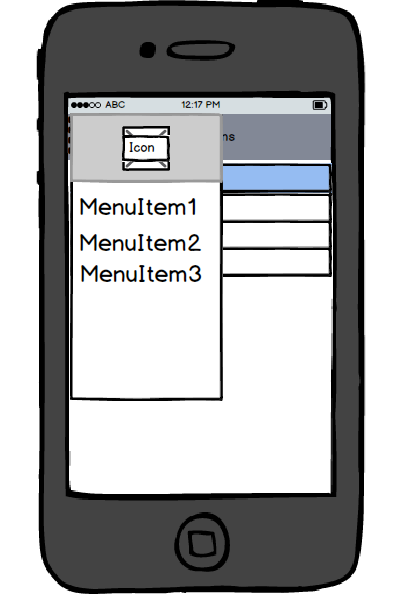
Rys.5.1: Widok listy kategorii. Rys.5.2: Widok dodawania zadania.



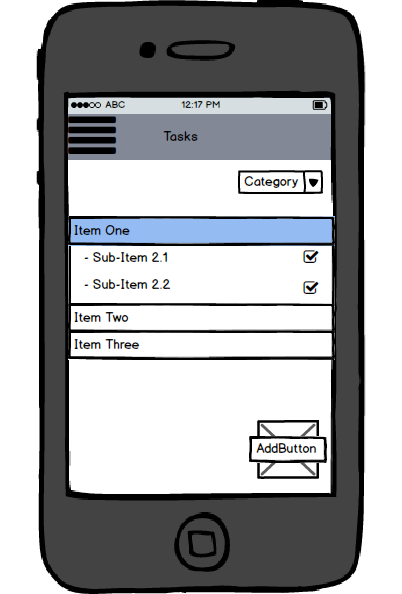
Rys.5.3: Widok dodawania podzadania. Rys.5.4: Widok dodawania podzadania.



Rys.5.5: Widok dodawania kategorii. Rys.5.6: Podgląd kategorii.



Rys.5.7: Widok menu aplikacji. Rys.5.8: Widok listy notyfikacji.



Rys.5.9: Widok listy zadań oraz podzadań.

## 5.4 Wzorce architektoniczne.

Autor w swojej aplikacji zastosował wzorzec MVC. Odpowiada on kolejno za zarządzanie modelem, widokiem oraz kontrolerem. Zabieg ten ma na celu oddzielenie logiki zarządzania bazą danych, reagowania na zachowania użytkownika w aplikacji oraz prezentowania widoku. Android oparty jest o język JAVA natomiast w przypadku implementacji widoku stosuje się pliki xml.

W modelu zawarłem całą logikę związaną z lokalną bazą danych. Posiada on klasę wykorzystującą biblioteki SQLite oraz interfejs DAO(data access object) oraz jego implementację. Zarządza on obiektami stworzonymi w ramach modelu oraz udostępnia metody pozwalająca na podstawowe działania w obrębie bazy danych. Ponadto tworzone obiekty są oparte o wygodne rozwiązanie getterów i seterów, co znacznie upraszcza w późniejszym etapie implementacji stosowanie logiki dodawania czy edytowania rekordów w bazie.

Widok jest ściśle związany z plikami xml oraz klasami JAVA, służącymi do komunikacji pomiędzy aplikacją a xml. Jest to powszechne rozwiązanie w programowaniu aplikacji.

W kontrolerze zawarto całą logikę dotyczącą reagowania na zachowania użytkownika oraz zaimplementowano sposób prezentowania menu w Material Design. Biorąc pod uwagę, że większość operacji odbywa się na listach zadań oraz kategorii, wykorzystano klasy służące za adapter dla wdrożonych list. Adapter ma za zadanie reagować na zmiany w obrębie swoich kompetencji oraz dynamicznie zmieniać swój wygląd lub zawartość. Odpowiada on także za sposób wyświetlenia danych.

Sam model MVC posiada wiele zalet. Najistotniejszą jest oddzielenie logiki biznesowej od warstwy prezentacji oraz od warstwy służącej za reagowanie na zachowanie użytkownika. Ułatwia to znacznie zachowanie logiki implementacji oraz pomaga przy tworzeniu testów dla aplikacji. W takim wypadku kontynuacja pracy w przyszłości może odbyć się w szybszym tempie bez konieczności ponownego wdrożenia w problematykę rozwiązań. Za wadę można uznać czas poświęcony na operacje realizowane w tle aplikacji. Złożoność implementacji wzorca MVC jest najlepiej dopasowany do większych aplikacji. W przypadku mniejszych programów zastosowanie tego model wymusza tworzenie wielu osobnych klas w obrębie projektu w celu zachowania założeń architektury.

## 5.5 Baza danych

**Model logiczny bazy danych SQLite:**

Przykład: **Encja**(kluczGłówny, atrybut, *kluczObcy).*

1. **Task** (TASK\_ID, TASK\_NAME, TASK\_DESCRIPTION, TASK\_SCHEDULED\_DATE, TASK\_EXECUTION\_DATE, TASK\_CREATION\_DATE, TASK\_MODIFICATION\_DATE. TASK\_PRIORITY, *TASK\_CATEGORY\_ID*).

Zawiera dane o wprowadzonych zadaniach.

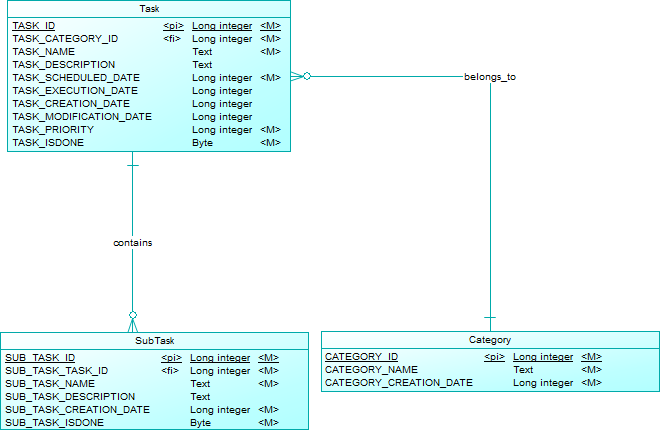
2. **SubTask** (SUB\_TASK\_ID, SUB\_TASK\_NAME, SUB\_TASK\_DESCRIPTION, SUB\_TASK\_CREATION\_DATE, SUB\_TASK\_ISDONE, *SUB\_TASK\_TASK\_ID*).

Zawiera dane o podzadaniach dla zadania.

3. **Category** (CATEGORY\_ID, CATEGORY\_NAME, CATEGORY\_CREATION\_DATE).

Zawiera dane o kategoriach w aplikacji.

**Model fizyczny bazy danych SQLite:**



Rys.5.11: Model bazy danych.

## 5.6 Podsumowanie

Autor skupił się na dokładniejszym przedstawieniu zadania wprowadzanego przez użytkownika, w porównaniu do zaimplementowanego modelu podzadania. Zadanie samo w sobie posiada najważniejszą grupę szczegółów i informacji, stąd rola podzadania jest ograniczona jedynie do formy informowania o ewentualnych krokach na drodze do realizacji celu. Za przykład można przyjąć listę zakupów, która będzie figurować jako zadanie „Zakupy”. Następnie możemy do tego zadania dopisać podzadania, które będą reprezentować produkty, w które chcemy się zaopatrzyć.

Autor zrealizował strukturę zadania na podstawie doświadczeń z obecnymi aplikacjami na rynku oraz na podstawie założeń aplikacji, mających na celu stworzenie potencjalnie wygodnego środowiska dla odbiorcy. Budowa encji Task**,** zaprezentowanej na Rys.5.11, umożliwia stworzenie responsywnej logiki tworzenia, ewaluowania oraz kończenia zadania. Istotną rolę odgrywa flaga ISDONE, która informuje aplikację o tym, czy dane zadanie uznano za zakończone, czy jest dalej w trakcie realizowania. Istotnym elementem są także pola dat, które znajdują zastosowanie do obliczania czasu realizacji oraz na ich podstawie budowane są notyfikacje

Podzadanie nie posiada możliwości zaplanowania w czasie i sprawia to, że użytkownik zwraca większą uwagę na główne pojęcie zadania. Zabieg ten pozwala także ułatwić dostęp do żądanych informacji, gdyż drugorzędność podzadań ukierunkowuje użytkownika na analizę danych zadania. Także autor zadeklarował formę podzadań bez stosowania sortowania oraz priorytetów. Jednakże na drodze rozwoju aplikacji może okazać się to konieczne, gdyż tak naprawdę potencjalni użytkownicy aplikacji determinują kolejne zmiany i ulepszenia. Na ten moment, biorąc też pod uwagę kompaktowość aplikacji i poręczność, struktura modeli zadań oraz podzadań jest zaprojektowana z myślą o spełnieniu najważniejszych wymagań dla odbiorcy.

# **Rozdział 6 Implementacja**

## 6.1 Wprowadzenie

Autor w tym rozdziale omówi szczegóły implementacyjne. W pierwszej kolejności zostanie przedstawione środowisko oraz narzędzia programistyczne. Kolejne etapy omówienia implementacji będą dotyczyć formy wdrożenia bazy danych, opisu wzorca architektonicznego, zaprezentowania interfejsu wraz z przedstawieniem problemów implementacyjnych oraz finalnie podsumowanie rozdziału w formie omówienia możliwych dróg rozwoju aplikacji.

## 6.2 Wykorzystane środowiska oraz narzędzia programistyczne

Do realizacji projektu wykorzystano środowisko Android Studio zapewniające właściwą jakość pracy w systemie Android. Do implementacji bazy danych zastosowany SQLite wraz z pomocniczą nakładką do przeglądarki internetowej Firefox – SQLite Manager. Służył on do podglądu bazy danych na etapie jej tworzenia. Prototypowanie interfejsu oparłem o wykorzystanie frameworku środowiska Android Studio służącego do zarządzania widokami oraz częścią graficzną programu.

## 6.3 Implementacja bazy danych

W implementacji bazy danych wykorzystano pomocniczą klasę SQLiteOpenHelper. Oprócz zarządzania bazą danych odpowiada także za wersjonowanie bazy. Innymi słowy, inkrementacja numeru bazy umożliwia zachowanie poprawnego działania bez konieczności usuwania i kreowania bazy danych.

Do stworzenia tabel posłużyłem się poniższym kodem:

private static final String CREATE\_TABLE\_TASK = "CREATE TABLE " + TABLE\_TASK + "("

+ TASK\_ID + " INTEGER PRIMARY KEY AUTOINCREMENT,"

+ TASK\_NAME + " TEXT NOT NULL, "

+ TASK\_DESCRIPTION + " TEXT,"

+ TASK\_SCHEDULED\_DATE + " INTEGER,"

+ TASK\_EXECUTION\_DATE + " INTEGER,"

+ TASK\_CREATION\_DATE + " INTEGER,"

+ TASK\_MODIFICATION\_DATE + " INTEGER,"

+ TASK\_PRIORITY + " INTEGER NOT NULL, "

+ TASK\_ISDONE + " INTEGER NOT NULL, "

+ TASK\_CATEGORY\_FOREIGN\_KEY\_ID + " INTEGER, "

+ "FOREIGN KEY (" + TASK\_CATEGORY\_FOREIGN\_KEY\_ID + ") REFERENCES "

+ TABLE\_CATEGORY + " (" + CATEGORY\_ID + ")"

+ ")";

private static final String CREATE\_TABLE\_CATEGORY = "CREATE TABLE " + TABLE\_CATEGORY + "("

+ CATEGORY\_ID + " INTEGER PRIMARY KEY AUTOINCREMENT,"

+ CATEGORY\_NAME + " TEXT NOT NULL UNIQUE,"

+ CATEGORY\_CREATION\_DATE + " INTEGER"

+ ")";

private static final String CREATE\_TABLE\_SUB\_TASK = "CREATE TABLE " + TABLE\_SUB\_TASK + "("

+ SUB\_TASK\_ID + " INTEGER PRIMARY KEY AUTOINCREMENT,"

+ SUB\_TASK\_NAME + " TEXT NOT NULL,"

+ SUB\_TASK\_DESCRIPTION + " TEXT,"

+ SUB\_TASK\_CREATION\_DATE + " INTEGER,"

+ SUB\_TASK\_TASK\_FOREIGN\_KEY + " INTEGER, "

+ SUB\_TASK\_ISDONE + " INTEGER NOT NULL, "

+ "FOREIGN KEY (" + SUB\_TASK\_TASK\_FOREIGN\_KEY + ") REFERENCES "

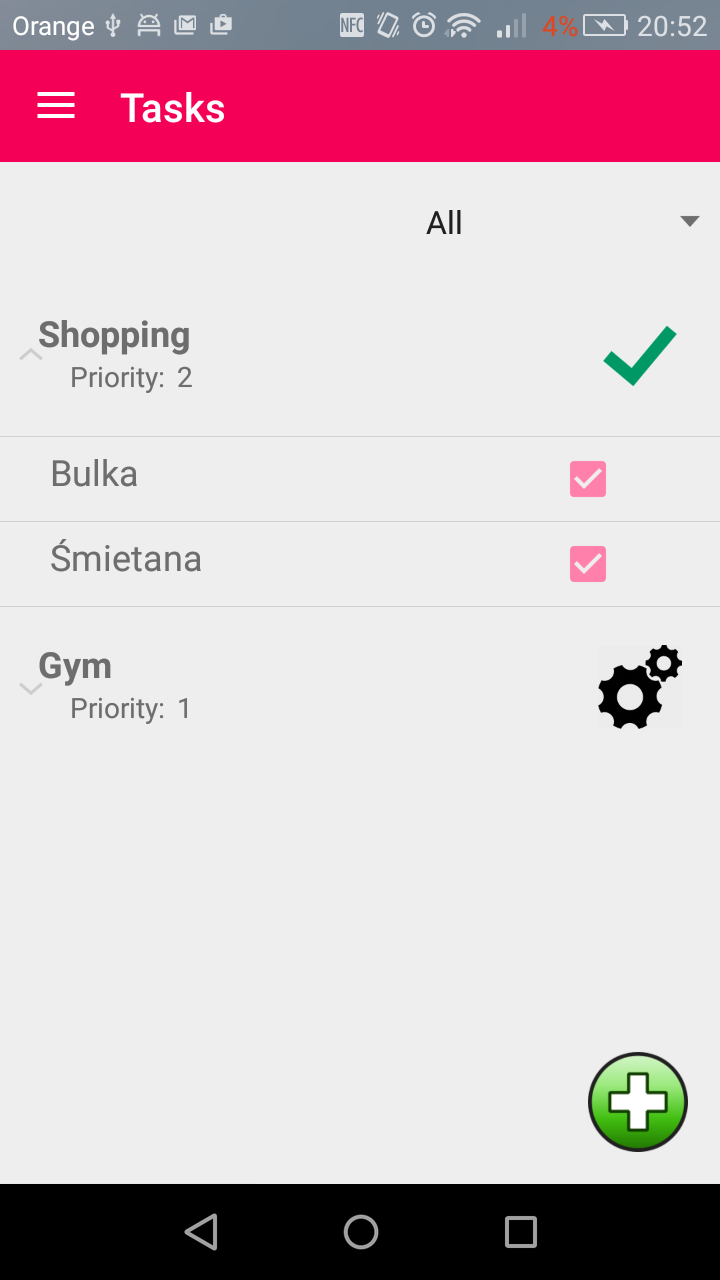
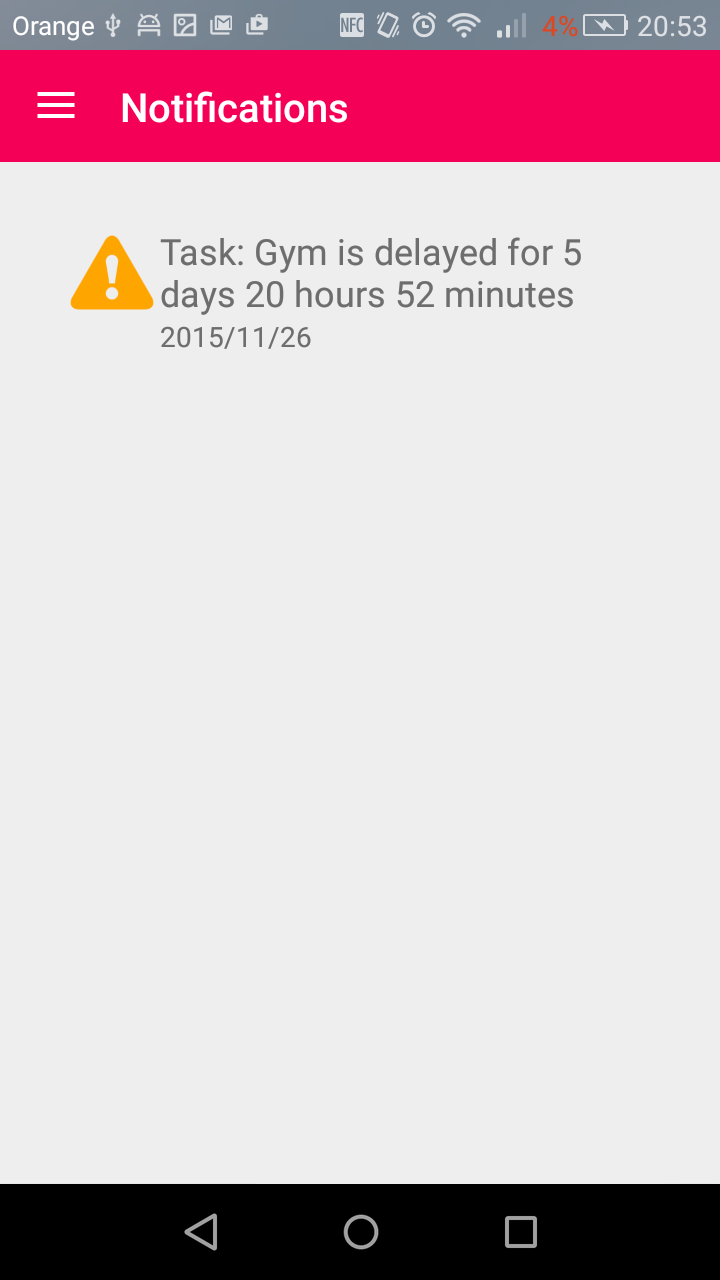
+ TABLE\_TASK + " (" + TASK\_ID + ")"

+ ")";

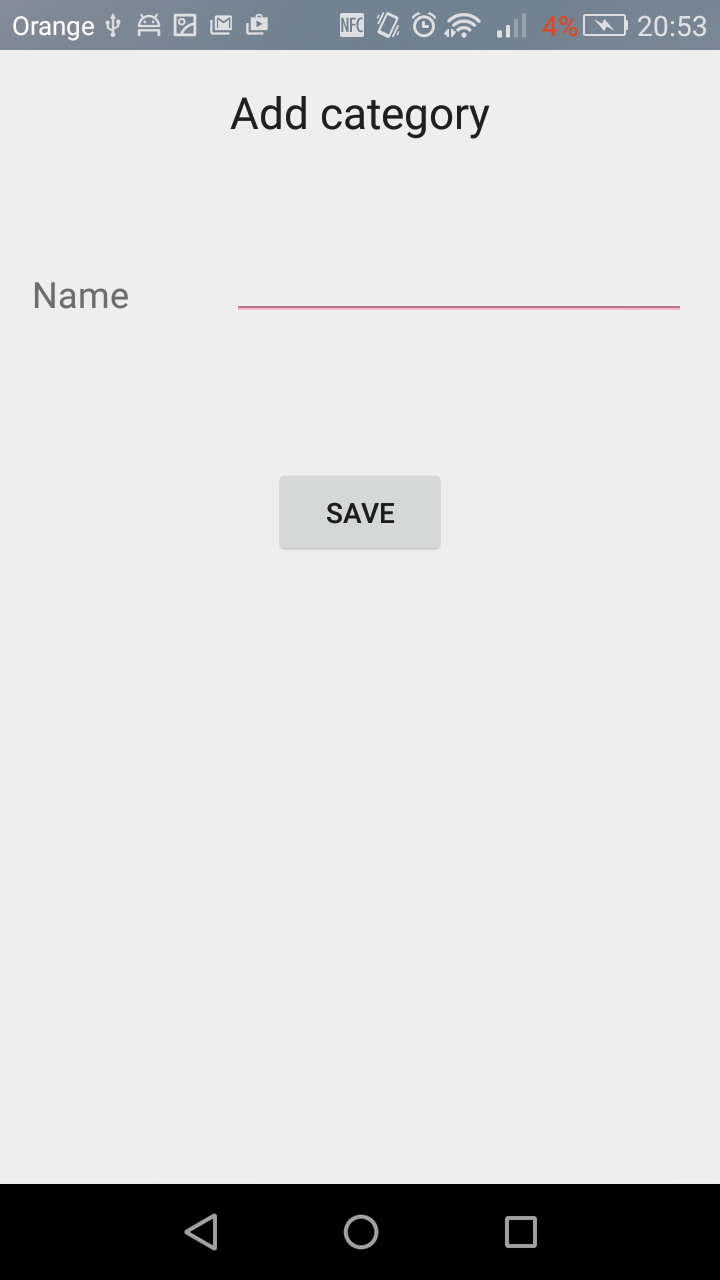
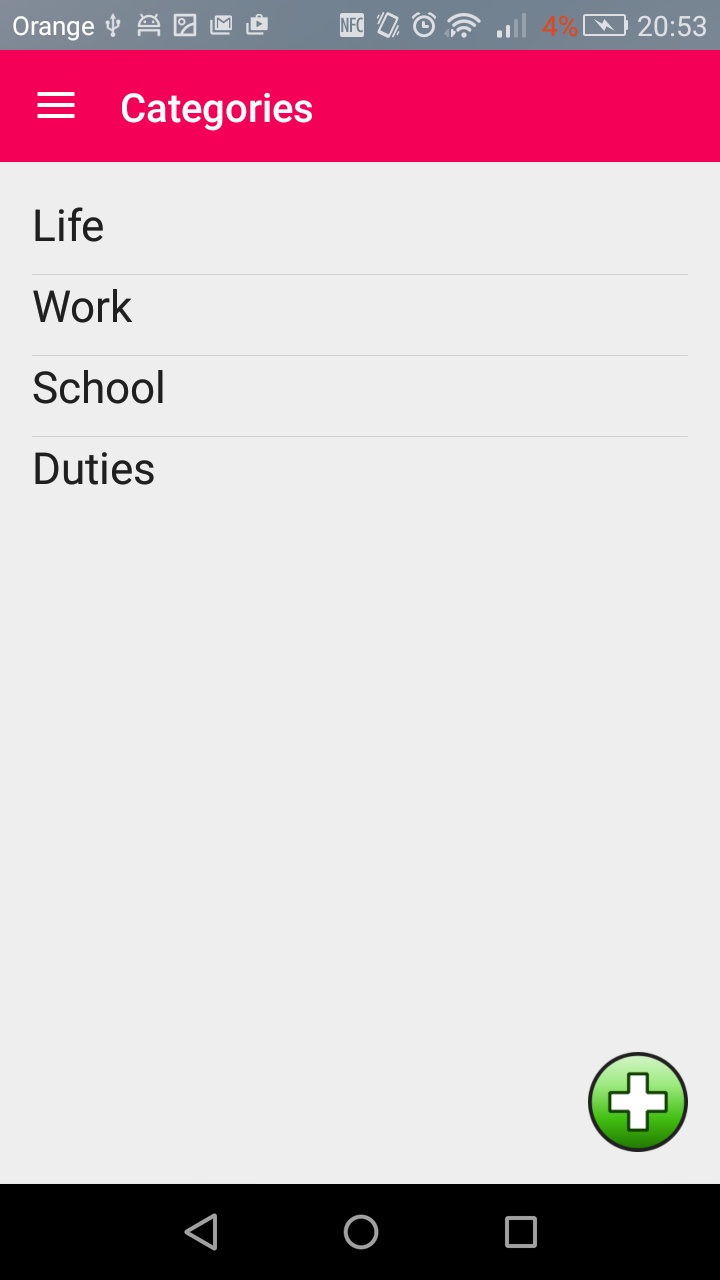
W trakcie implementacji tego rozwiązania autor zauważył, że w przypadku tworzenia połączeń między tabelami i dodawania klucza obcego konieczne jest umieszczenie ów kluczy na samym końcu tworzonej tabeli. Inne działania uniemożliwiają poprawne skompilowanie kodu, a co za tym idzie, baza danych nie zostanie stworzona.

## 6.4 Interfejs użytkownika

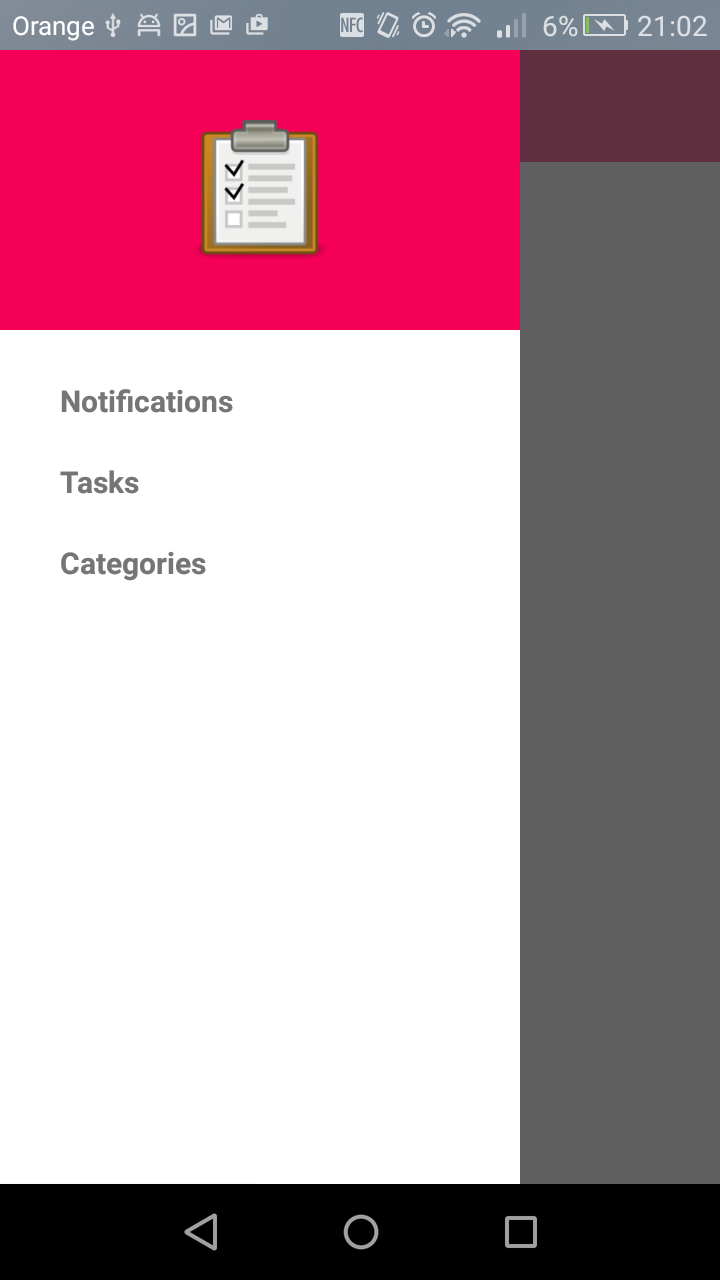
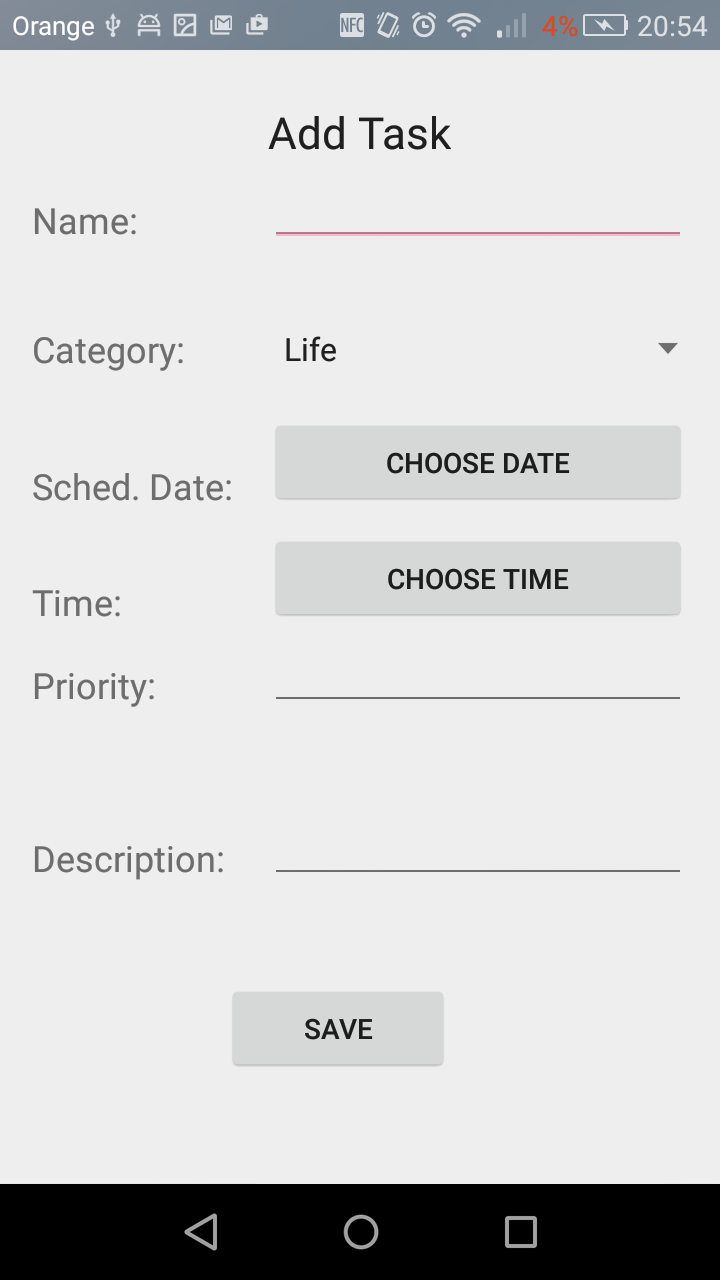
W tym podrozdziale przedstawiono obecny wygląd interfejsu aplikacji. Jest on spójny z założeniami projektowymi oraz jego forma sprzyja dalszemu rozwojowi aplikacji. Autor zakłada rozszerzenie funkcjonalności i logiki realizacji zadań oraz poszerzenie menu o elementy opisane w końcowych sugestiach odnośnie pracy. W przypadku realizacji interfejsu dla aplikacji, wykorzystano informacje zawarte w książce[2] *Android. Podstawy tworzenia aplikacji.* W rozdziale 4 autor książki opisuje zachowanie odpowiednich komponentów dla widoku. Autor skupił się na informacjach dotyczących ScrollView, opisujących jak dokładnie powinien zachować się przesuwalny ekran oraz użyteczna okazała się także przykładowa implementacja takiego rozwiązania. W przypadku tworzenia aplikacji, te informacje są w stanie zaoszczędzić dużo czasu i podstawy zawarte w tym rozdziale ułatwiają dalszą pracę z kodem. Poniżej autor przedstawił wybrane elementy interfejsu aplikacji:

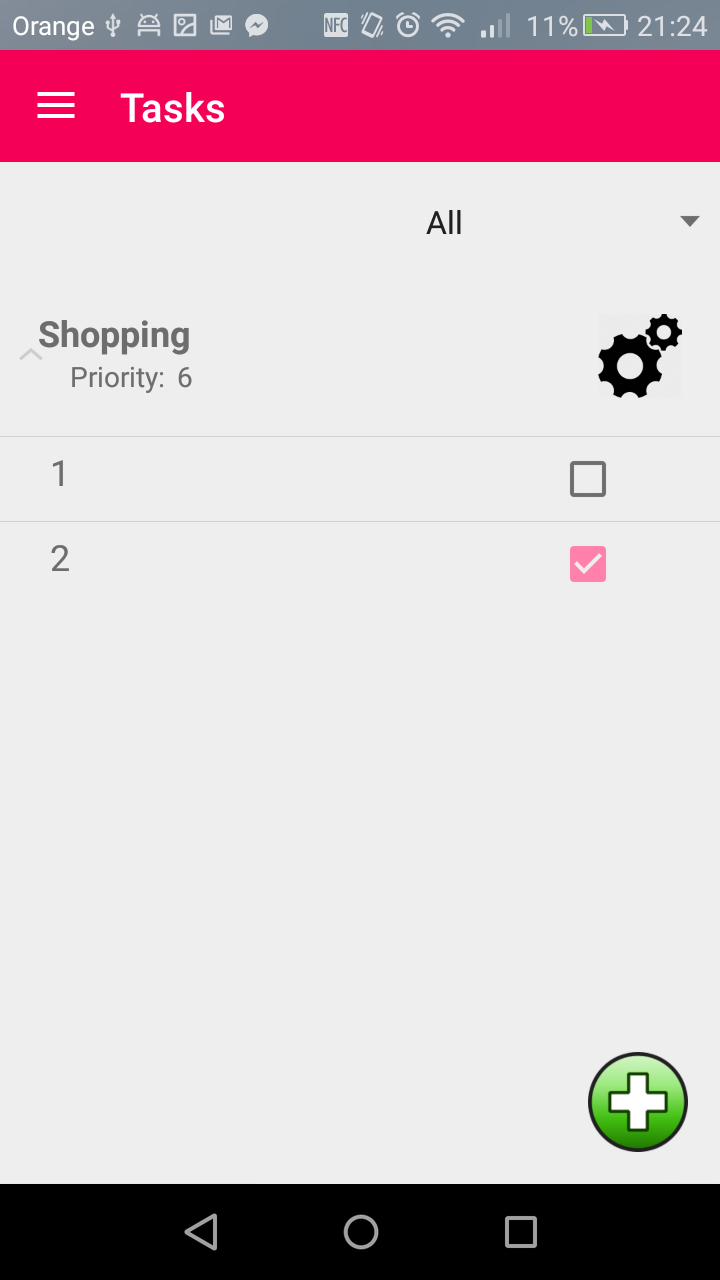
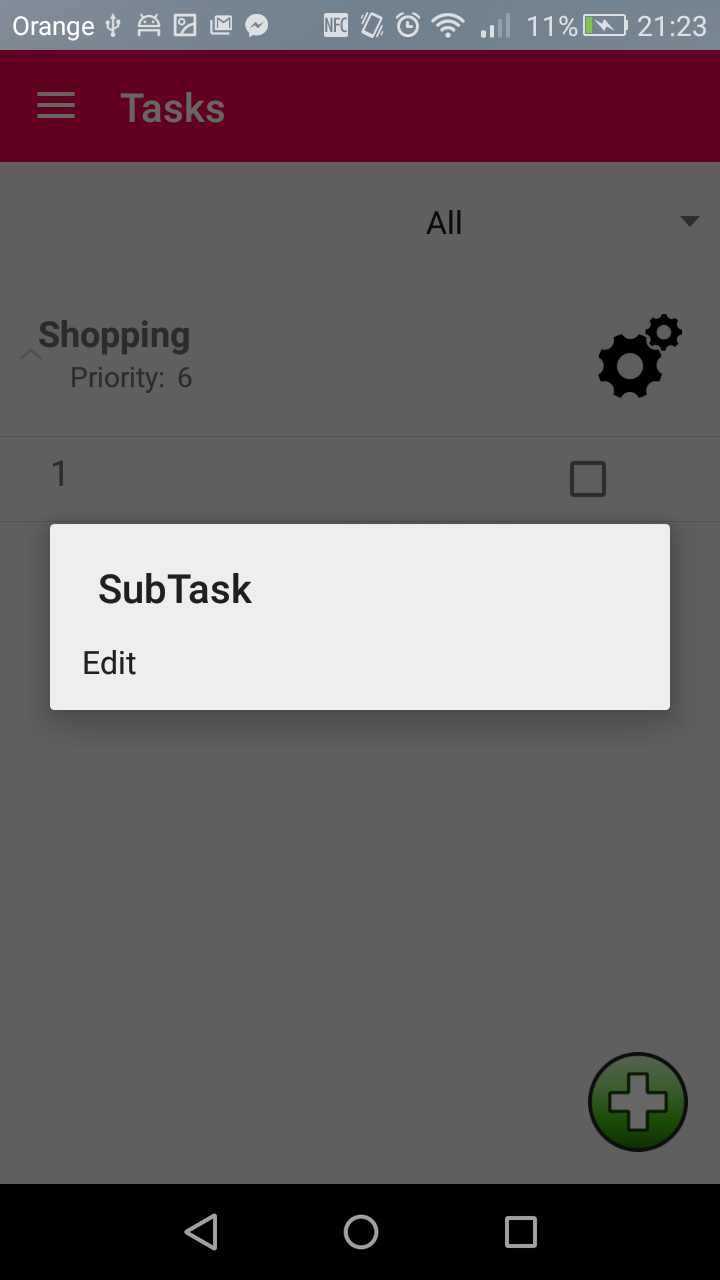
Rys. 6.1: Widok listy zadań. Rys. 6.2: Widok notyfikacji.



Rys. 6.3: Widok kategorii. Rys. 6.4: Widok dodawania kategorii.

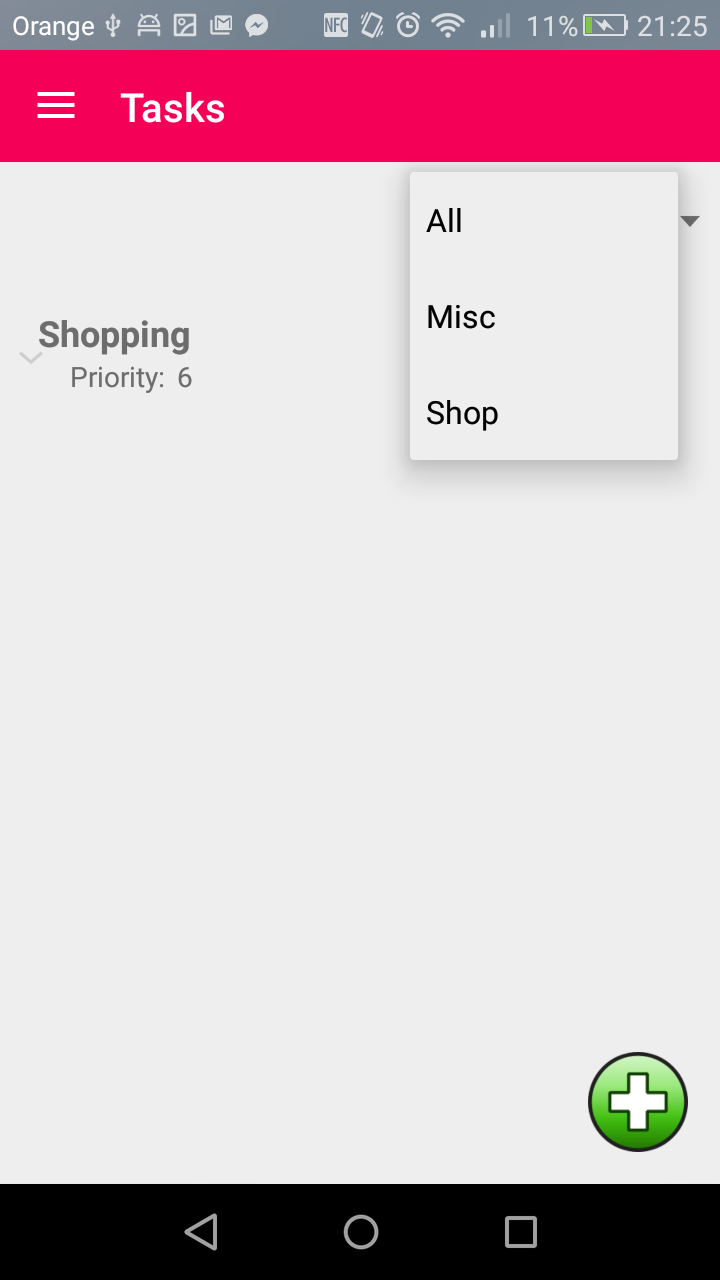
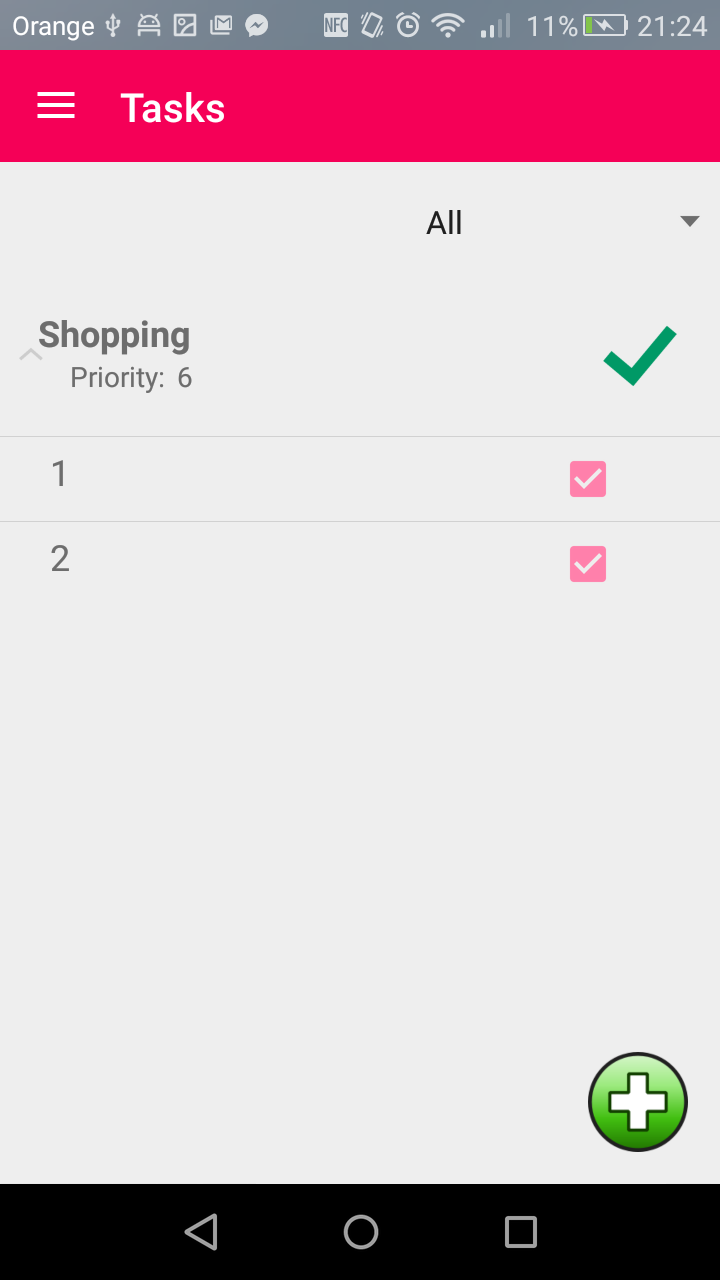


Rys. 6.5: Widok dodawania zadań. Rys. 6.6: Widok menu aplikacji.



Rys. 6.7: Widok menu kontekstowego dla podzadań. Rys. 6.8: Widok listy zadań z jednym

wykonanym podzadaniem.



Rys. 6.9: Widok ze zrealizowanym zadaniem. Rys. 6.10: Widok filtra kategorii.

Zastosowanie Material Design można zaobserwować w postaci prostej kompozycji elementów oraz ograniczonej ilości kolorów. Na podstawie Rys.6.8 oraz Rys.6.9 można zauważyć, jak przebiega logika zmiany charakteru zadania. W przypadku niezakończonego podzadania, zadanie wciąż pozostaje niezakończone. W momencie sfinalizowania wszystkich podzadań, automatycznie zadanie otrzymuje status zakończonego.

Autor zakłada zmianę formy przycisku dodawania np. kategorii oraz zaplanowano zmianę koloru ikon, w zależności od kolejnych statusów zadania. Wiąże się to także z rozszerzeniem bazy danych i samego modelu zadania w aplikacji. Dodatkowo dalszy etap rozwoju interfejsu będzie dotyczył także rozszerzenia opcji sortowania. Jednakże pierwszą najistotniejszą zmianą okaże się dodanie interfejsu wyboru dnia i realizowania w kolejnych widokach logiki dodawania i ewaluowania zadań.

## 6.5 Problemy implementacyjne

Autor w tym podrozdziale omawia wybrane napotkane problemy w implementacji aplikacji oraz szczegółowo omówi sposób ich rozwiązania.

Tabela 6.1: PI-01 - Fragmenty oraz komunikacja między nimi.

|  |  |
| --- | --- |
| Lp. | PI-01 |
| Nazwa | Fragmenty oraz komunikacja między nimi. |
| Opis problemu | Problem dotyczy przechodzenia pomiędzy widokami zbudowanymi na fragmentach a kolejną aktywnością. W momencie powrotu z innej aktywności do poprzedniego fragmentu zaobserwowano powtarzanie ostatnio zapisanego schematu listy elementów bez uwzględnienia jej zmiany, która nastąpiła np. w momencie dodania zadania w aktywności, z której wracamy.  Fakt ten uniemożliwia sprawne posługiwanie się przyciskiem powrotu. |
| Rozwiązanie | Autor zastosował poniższy kod w momencie przechodzenia z fragmentu do aktywności dodawania zadania:  FragmentManager fm = getActivity().getSupportFragmentManager();  for(int i = 0; i < fm.getBackStackEntryCount(); ++i)  {  fm.popBackStack();  }  getActivity().finish();  startActivity(intent);  Zastosowanie tego rozwiązania w pierwszej kolejności czyści, ze stosu przechowywanych stanów fragmentu, wszystkie elementy oraz kończy życie obecnej aktywności. Finalnie następuje przejście do następnej aktywności. W przypadku powrotu aktywność pierwotna zostaje uruchomiona. |

Tabela 6.2: PI-02 - Typy danych w SQLite oraz przykład boolean.

|  |  |
| --- | --- |
| Lp. | PI-02 |
| Nazwa | Typy danych w SQLite oraz przykład boolean. |
| Opis problemu | Zastosowanie SQLite wymaga odpowiedniego zaplanowania typów danych dla tworzonej bazy. W przeciwieństwie do klasycznego SQL, posiadamy jedynie 5 klas typów danych. Implementacja skryptów do tworzenia bazy danych i tabel nie mogła zawierać typów danych takich jak boolean. |
| Rozwiązanie | W przypadku typu boolean autor zastosował typ Integer dla tworzonej tabeli. W dalszym procesie implementowania logiki został on wykorzystany do przechowywania wartości 0 i 1, odpowiadających kolejno za wartości false oraz true. Dodatkowe wskazówki można znaleźć w dokumentacji technicznej SQLite[9]. Ponadto dość obszerny materiał, mówiący o zastosowaniu i implementacji bazy danych w SQLite można znaleźć w Dodatku E w książce [4]. Autorzy udostępnili obszerne wyjaśnieine zastosowanie powłoki ADB do przeglądania katalogów aplikacji. Tym sposobem można także dostać się do bazy danych zapisanej na telefonie dla aplikacji. Autorzy także szczegółowo omawiają cechy stosowania SQLite, między innymi informacje o braku możliwości zagnieżdżania transakcji, braku możliwości złączeń typu RIGHT OUTER JOIN oraz informacja o ograniczonej wbudowanej funkcjonalności. Dodatek E zawiera też kluczową informację o sposobie dostępu do bazy danych. W związku z tym, że baza jest przechowywana jako plik, dostęp do danych należy realizować sekwencyjnie, a nie współbieżnie. |

Tabela 6.3: PI-03 - Typ danych dla dat.

|  |  |
| --- | --- |
| Lp. | PI-03 |
| Nazwa | Typ danych dla dat. |
| Opis problemu | Kolejny problem dotyczący przechowywania dat dotyczy typu danych dla dat. Początkowo autor zakładał typ danych jako DATETIME. W praktyce okazało się, że sposób przechowywania daty w bazie danych wymuszał stosowanie osobnych funkcji pozwalających na konwertowanie typu DateTime do SimpleDateFormat. Powoduje to dalsze komplikacje implementacyjne, w szczególności przy odczytywaniu danych z tabeli przy zastosowaniu interfejsu Cursor. |
| Rozwiązanie | Najlepszym rozwiązaniem okazało się przechowywanie daty jako typu liczbowego. W momencie zapisu do bazy, data jest konwertowana do postaci milisekund, co jest znacznie łatwiejszym sposobem niż stosowanie SimpleDateFormat. Tak spreparowana data nie jest też problemem w przypadku odczytu danych z tabeli.  Przykład zastosowania interfejsu Cursor do odczytywania danych z tabeli:  private Category cursorToItem(Cursor cursor)  {  Category category = new Category();  category.setId(cursor.getLong(0));  category.setName(cursor.getString(1));  category.setCreationDate(cursor.getLong(2));  return category;  }  Analizując Cursor można zauważyć, że nie posiada on metody pozwalającej bezpośrednio. Ponadto zastosowanie typu liczbowego daje możliwość łatwego sortowania rekordów według daty. |

Tabela 6.4: PI-04 - Zastosowanie klasy abstrakcyjnej Calendar.

|  |  |
| --- | --- |
| Lp. | PI-04 |
| Nazwa | Zastosowanie klasy abstrakcyjnej Calendar. |
| Opis problemu | Autor aplikacji napotkał problem implementacyjny dotyczący sprawnego zarządzania logiką aktualizowania dat oraz zapisywania ich w momencie określonego zachowania użytkownika aplikacji. Wymagało to zastosowania rozwiązania implementacyjnego nie obciążającego obliczeniowo. |
| Rozwiązanie | W przypadku przechowywania dat jako typ liczbowy najlepszym rozwiązaniem okazała się klasa Calendar. Posiada ona metodę getInstance, zwracającą aktualną datę. Narzędzie to jest prostym sposobem do zapisywania rekordów z koniecznością zachowania kompatybilności z wprowadzonymi ustawieniami godziny i daty na telefonie. Ta funkcjonalność w przyszłości zapewni szersze perspektywy rozwoju aplikacji. Ponadto ta klasa posiada także metodę zwracającą czas w milisekundach.  Przykład zastosowania klasy abstrakcyjnej Calendar:  final Calendar cal = Calendar.getInstance();  year\_x = cal.get(Calendar.YEAR);  month\_x = cal.get(Calendar.MONTH);  day\_x = cal.get(Calendar.DAY\_OF\_MONTH);  W tym przypadku autor korzysta z elementu kodu odpowiedzialnego za wyświetlanie daty w widoku dodawania lub edytowania zadania. Zmienne odpowiedzialne za rok, miesiąc oraz dzień są reprezentowane jako wartości liczbowe. |

Tabela 6.5: PI-05 - Reprezentacja daty i godziny przy pomocy DatePicker oraz TimePicker.

|  |  |
| --- | --- |
| Lp. | PI-05 |
| Nazwa | Reprezentacja daty i godziny przy pomocy DatePicker oraz TimePicker. |
| Opis problemu | Aplikacja w przypadku dodawania oraz edytowania zadania musi zapewnić łatwy w obsłudze interfejs wybierania daty oraz godziny. Wpisywanie ręczne daty do stworzonego szablonu nie jest dobrym rozwiązaniem. |
| Rozwiązanie | Autor zastosował rozwiązanie w formie widgetów DatePicker oraz TimePicker. Odpowiadają one za wyświetlanie okna służącego do wybrania kolejno daty oraz godziny. Poniżej został zaprezentowany efekt implementacji tych widgetów.    Rys.6.1. Widok widgetów.  Dużą pomocą implementacyjną okazuje się artykuł [7]. Autor artykułu opisuje szczegółowo ścieżki implementacji tego rozwiązania oraz wyczerpująco objaśnia sposób działania widgetów. |

Tabela 6.6: PI-06 - Dobór formatu godziny.

|  |  |
| --- | --- |
| Lp. | PI-06 |
| Nazwa | Dobór formatu godziny. |
| Opis problemu | W przypadku zastosowania widgetów, udostępniających interfejs wyboru daty, pojawia się problem związany z właściwym prezentowaniem wybranej godziny. Konieczne jest zastosowanie osobnych metod modyfikujących wybraną godzinę do odpowiedniego formatu. |
| Rozwiązanie | Klasa abstrakcyjna Calendar umożliwia zweryfikowanie aktualnej daty i godziny. Dzięki temu łatwo jest określić czy dana godzina powinna pojawić się z informacją „am” czy „pm”. Poniższy kod przedstawia sposób poprawnej realizacji zapisu godziny w słuchaczu dla TimePicker:  Calendar cal = Calendar.getInstance();  cal.set(year\_x, month\_x , day\_x , hour , minute);  if(cal.get(Calendar.AM\_PM) == 0)  {  am\_pm = "AM";  }  else  {  am\_pm = "PM";  }  String curTime = String.format("%02d:%02d", cal.get(Calendar.HOUR), cal.get(Calendar.MINUTE)) + " " + am\_pm;  Warunek sprawdzający obiekt “cal” umożliwia zweryfikowanie, który ze skrótów godzinowych powinien zostać użyty. Następnie odpowiednia wartość zostaje przypisana do wcześniej zadeklarowanego String am\_pm. Ostatniecznie, poprzez zastosowanie formatowania Stringa, kreowana jest zmienna curTime. Reprezentuje ona godzinę wraz ze skrótem „am” lub „pm”. |

Tabela 6.7: PI-07 - Wykorzystanie ExpandableListView.

|  |  |
| --- | --- |
| Lp. | PI-07 |
| Nazwa | Wykorzystanie ExpandableListView. |
| Opis problemu | Autor aplikacji na etapie projektowania wybrał sposób prezentowania zadań w formie listy z łatwym dostępem do podzadań dla każdego zadania. Wybrane rozwiązanie musi posiadać cechy łatwego i szybkiego dostępu do informacji oraz przejrzystości interfejsu. |
| Rozwiązanie | W przypadku tworzenia dynamicznych list rozważano zastosowanie formy DragAndDrop. Polega ona na przesuwaniu widocznych elementów w ramach listy w dowolnej kolejności w celu zmiany ich porządku wyświetlania bądź pozycji. Rozwiązanie to ma wiele zalet, lecz nie jest w stanie zapewnić dynamicznego rozwijania listy w celu wyświetlania dodatkowych informacji. Taką cechą charakteryzuje się ExpandableListView. Autor zastosował to rozwiązanie z powodzeniem budując odpowiedni adapter dla tej listy w oparciu o informacje zawarte w dokumentacji technicznej [6]. Można znaleźć tam istotne informacje o sposobie budowania konstruktora dla adaptera oraz w jaki sposób działają jego metody. W podrozdziale 4.4 *Android w praktyce* [1] autorzy udostępniają istotne informacje dotyczące informowania ListView co do zmiany w strukturze wyświetlanych elementów. Jest to kluczowy element w przypadku implementacji dynamicznej listy na systemie operacyjnym Android. |

Tabela 6.8: PI-08 - Podgląd implementowanej bazy danych.

|  |  |
| --- | --- |
| Lp. | PI-08 |
| Nazwa | Podgląd implementowanej bazy danych. |
| Opis problemu | Android Studio dostarcza jedynie powłokę poleceń SQLite, natomiast nie istnieje rozwiązanie pozwalające na użycie graficznego interfejsu do zarządzania bazą danych. |
| Rozwiązanie | Autorzy książki *Android w praktyce* [1] w podrozdziale 7.4 udostępniają sposób graficznego zobrazowania bazy danych zaimplementowanej w SQLite. Polecają zastosowanie z narzędzia dedykowanego dla przeglądarki Firefox o nazwie SQLiteManager. Nie istnieje bezpośrednie połączenie pomiędzy programem a bazą danych aplikacji, stąd istnieje konieczność zapisania jej lokalnie na komputerze. Następnie zapisany plik można otworzyć z poziomu interfejsu SQLiteManager. Jest to według autora najwygodniejszy mechanizm prezentowania danych i samej bazy danych. |

Tabela 6.9: PI-09 - Kaskadowe usuwanie danych.

|  |  |
| --- | --- |
| Lp. | PI-09 |
| Nazwa | Kaskadowe usuwanie danych. |
| Opis problemu | Aplikacja przypisuje każde zadanie do wybranej kategorii, a także każde podzadanie ma przypisane dokładnie jedno zadanie. W przypadku konieczności usunięcia wybranej kategorii, aplikacja musi zareagować na wszystkie obiekty powiązane z daną kategorią. |
| Rozwiązanie | Biorąc pod uwagę relację tabeli i stworzony schemat, należy zadbać o konieczność reagowania na usunięcie kategorii w sposób kaskadowy. Oznacza to, że usunięcie kategorii spowoduje usunięcie zadań dotyczących tej kategorii oraz podzadań. Mechanizm ten jest konieczny, gdyż w przypadku niezastosowania tej metody, baza danych uniemożliwi wykonanie operacji, a tym samym aplikacja zostanie zamknięta z powodu krytycznego błędu. Autor poniżej przedstawił sposób implementacji usuwania kaskadowego.  TaskDaoImpl taskDao = new TaskDaoImpl(getApplicationContext());  taskDao.open();  String where = PersonalManagerDBHelper.TASK\_CATEGORY\_FOREIGN\_KEY\_ID + " = " + category.getId();  List<Task> tasks = taskDao.findTasks(where, null);  if(tasks.size() == 0)  {  return true;  }  else  {  return false;  }  Jest to implementacja funkcji boolowskiej korzystającej z zaimplementowanego przez autora wzorca DAO. Poprzez odpytanie bazy danych dla określonej kategorii, aplikacja jest w stanie sprawdzić czy dana kategoria posiada zadania. Warunek sprawdzający zwraca kolejno wartości True dla pustej kategorii oraz False dla kategorii posiadającej zadania. Następnie w słuchaczu dla przycisku „Delete” aplikacja weryfikuje otrzymane dane i prezentuje odpowiedni komunikat, w zależności od wdrożonej operacji. |

## 6.6 Dalsze plany rozwoju aplikacji

Autor zakłada rozwój aplikacji na poziomie zarówno Frontendu jak i Backendu. Innymi słowy, wszelkie kolejne projekty i plany rozszerzenia aplikacji będą dotyczyć zarówno części odpowiadającej za implementację, jak i za elementy interfejsu. Na podstawie dotychczasowych prac sformułowano następujące ścieżki rozwoju:

a) W przypadku możliwości wykorzystania drugiego urządzenia mobilnego pracującego na danych już wprowadzonych przez użytkownika do pierwotnego urządzenia konieczne jest zastosowanie zewnętrznego środowiska przechowującego stworzone dane. Do aktualnego zastosowania lokalnej bazy danych zostanie dodana możliwość eksportowania bazy na konto Dropbox. W ten sposób użytkownik będzie mógł zapisać dane aplikacji oraz w łatwy sposób importować je.

b) Jeżeli rozwiązanie z punktu **a** nie będzie spełniać wymagań i oczekiwań użytkowników, zostanie zrealizowana forma bazy danych przechowywanej na zewnętrznym serwerze, a nie na lokalnym, jak to dotychczas jest zrealizowane. W dalszym etapie zakłada się dodanie funkcjonalności edytowania i realizowania zadań wraz z osobami upoważnionymi do pobrania oraz edytowania danych aplikacji użytkownika. Ścieżka ta doprowadzi do stworzenia środowiska umożliwiającego współpracę dla zespołów np. projektowych.

c) Dodanie widoku kalendarzowego, umożliwiającego wybranie dnia oraz zaplanowanie zadań w nim.

d) System logowania i rejestracji w momencie uzyskania działającej bazy danych opartej o serwer.

e) Rozszerzenie systemu notyfikacji o automatyczne wysyłanie maili w formie przypomnienia lub ostrzeżenia w cyklu realizacji zadań.

f) Wdrożenie modułu umożliwiającego pracę w systemie SCRUM. Będzie to przydatne narzędzie dla zespołów programistów oraz menadżerów.

g) Przechowywanie zdjęć oraz zrzutów ekranów dotyczących danego zdania lub podzadania.

h) Dodawania informacji w formie notatek. Będzie można je wpisać samodzielnie, lub pobrać bezpośrednio z Internetu w formie linku lub tekstu.

i) System grywalizacji. Aplikacja będzie reagować na terminowość użytkownika i każda udana i zaplanowana działalność będzie skutkować otrzymaniem punktów. Suma punktów będzie możliwa do porównania z innymi użytkownikami w formie rankingu.

# **Rozdział 7 Podsumowanie**

Celem pracy inżynierskiej autora było zaprojektowanie oraz zaimplementowanie aplikacji służącej do zarządzania zadaniami. Realizacja pracy obejmowała część odpowiadającą za dobór i zastosowanie właściwego modelu graficznego z interfejsem oraz część odpowiadającą za wdrożenie logiki służącej do wdrożenia logiki, odpowiadającej za reagowanie na zachowanie użytkownika w obrębie aplikacji.

Autor zastosował się do założeń projektowych oraz wykorzystał wszystkie omawiane wcześniej elementy interfejsu. Zwrócono szczególną uwagę na zachowanie właściwej, dla Material Design, konwencji graficznej przy dodaniu autorskich ikon oraz formy ich reagowania na status zadania. Warto zaznaczyć, że odpowiedni dobór elementów interfejsu oraz właściwa ich kompozycja odpowiadają za zrealizowanie założeń co do prostego i przystępnego środowiska dla użytkownika aplikacji.

Założenia implementacyjne zostały spełnione oraz zrealizowane zgodnie z przedstawionymi przypadkami użycia. Każdy z nich posiada swoje działające odzwierciedlenie w aplikacji. Program został przetestowany przez autora oraz posiada także odpowiednie funkcje zabezpieczające przez niepożądanym działaniem użytkownika. Pracochłonne okazało się odnalezienie przyczyny występowania problemu z *Tabela 6.1.* Autor poświęcił dużo czasu na odszukanie rozwiązania poprzez użycie narzędzi debuggera w środowisku programistycznym. Tym sposobem udało się odnaleźć przyczynę oraz zastosować odpowiednią modyfikację dla aplikacji.

Geneza projektu inżynierskiego jest ściśle związana z co raz większą popularnością aplikacji i programów służących do uporządkowania codziennych celów i zadań. Ponadto rozwijający się trend tworzenia designu dla aplikacji jest interesującą drogą do zaprezentowania na przykładzie aplikacji autora. Uwzględniono również dalsze ścieżki rozwoju aplikacji w podrozdziale 6.6, które dokładnie definiują cel i dalszą możliwą drogą dla projektu inżynierskiego. Najistotniejszym krokiem w stronę rozwoju aplikacji jest wprowadzenie zewnętrznej bazy danych przechowywanej na serwerze.

# **Bibliografia**

[1] Charlie Collins, Michael Galpin, Matthias Kaeppler: *Android w praktyce*, HELION, Gliwice, 2012

[2] Andrzej Stasiewicz: *Android. Podstawy tworzenia aplikacji,* HELION, Gliwice, 2013

[3] Ed Burnette: *Hello Android*, HELION, Gliwice, 2011

[4] Shane Conder, Lauren Darcey: *Android programowanie aplikacji na urządzenia mobilne wydanie II*, HELION, Gliwice, 2011

[5] Joseph Schmuller, *UML dla każdego.* HELION, Gliwice 2003

[6] [Android](https://plus.google.com/111484694481085486134), *Dokumentacja techniczna – rozdział ExpandableListView.* [dostęp 5 listopada 2015]. Dostępny w Internecie: *http://developer.android.com/reference/android/widget/ExpandableListView.html*

[7] Mkyong, *Android date picker example.* [dostęp 5 listopada 2015]. Dostępny w Internecie: [*http://www.mkyong.com/android/android-date-picker-example/*](http://www.mkyong.com/android/android-date-picker-example/)

[8] Cooper, Bruce: [*Activities, Tasks and Intents, Oh My!*](http://www.sitepoint.com/activities-tasks-and-intents-oh-my/)*.* [dostęp 17 października 2015]. Dostępny w Internecie: *http://www.sitepoint.com/activities-tasks-and-intents-oh-my/*

[9] SQLite. *Dokumentacja techniczna SQLite*[dostęp 5 listopada 2015] : Dostępny w Internecie: *https://www.sqlite.org/datatype3.html*

[10] Nick Heath, *Material Design: 10 apps that show off Google's new UI language.* [dostęp 14 października 2015].   
Dostępny w Internecie*:* [*http://www.techrepublic.com/pictures/material-design-10-apps-that-show-off-googles-new-ui-language/*](http://www.techrepublic.com/pictures/material-design-10-apps-that-show-off-googles-new-ui-language/)

[11] Chris Smith, *Google’s Material Design is about to change the way we look at the worldwide web.* [dostęp 14 października 2015].  
Dostępny w Internecie: [*http://bgr.com/2014/07/30/google-drive-material-design-update/*](http://bgr.com/2014/07/30/google-drive-material-design-update/)

[12] [Jillian D'Onfro](http://www.businessinsider.com/author/jillian-donfro), *16 reasons Android phones are better than iPhones* [dostęp 18 listopada]. Dostępny w Internecie [*http://www.businessinsider.com/android-phones-vs-iphone-2015-9*](http://www.businessinsider.com/android-phones-vs-iphone-2015-9)

[13] [AppleInsider Staff](mailto:news@appleinsider.com), *Android dominate smartphone market with 96% combined stake*. [dostęp 18 listopada]. Dostępny w Internecie *http://appleinsider.com/articles/15/02/24/ios-android-dominate-smartphone-market-with-96-combined-stake*

# **Spis rysunków**

Rys.2.1 Widok podstawowy aplikacji Todoist 6

Rys.2.2 Widok menu aplikacji Todoist 6

Rys.2.3 Widok podstawowy aplikacji Any.do 7

Rys.2.4: Przykład widoku z Wunderlist. 8

Rys.2.5: Udział mobilnych systemów operacyjnych na rynku[13]. 9

Rys.3.3 Udział procentowy wersji systemów Android 12

Rys.4.1: Diagram przypadków użycia 15

Rys.5.1: Widok listy kategorii. 22

Rys.5.2: Widok dodawania zadania. 22

Rys.5.3: Widok dodawania podzadania. 23

Rys.5.4: Widok dodawania podzadania. 23

Rys.5.5: Widok dodawania kategorii. 24

Rys.5.6: Podgląd kategorii. 24

Rys.5.7: Widok menu aplikacji. 25

Rys.5.8: Widok listy notyfikacji. 25

Rys.5.9: Widok listy zadań oraz podzadań. 26

Rys. 6.3: Widok kategorii. 27

Rys. 6.4: Widok dodawania kategorii. 33

Rys. 6.5: Widok dodawania zadań. 34

Rys. 6.6: Widok menu aplikacji. 34

Rys. 6.7: Widok menu kontekstowego dla podzadań. 35

Rys. 6.8: Widok listy zadań z jednym wykonanym podzadaniem. 35

Rys. 6.9: Widok ze zrealizowanym zadaniem. 36

Rys. 6.10: Widok filtra kategorii. 36

Rys.6.11. Widok widgetów. 40

# **Spis tabel**

Tabela 5.1: PU-1 - Dodawanie kategorii. 16

Tabela 5.2: PU-2 - Edycja kategorii. 16

Tabela 5.3: PU-3 - Usuwanie kategorii. 16

Tabela 5.4: PU-4 - Wybór kategorii do sortowania. 17

Tabela 5.5: PU-5 - Podgląd kategorii. 17

Tabela 5.6: PU-6 - Podgląd podzadania. 17

Tabela 5.7: PU-7 - Podgląd zadania. 18

Tabela 5.8: PU-8 - Ukończenie podzadania. 18

Tabela 5.9: PU-9 - Ukończenie zadania. 18

Tabela 5 10: PU-10 - Usunięcie zadania. 19

Tabela 5.11: PU-11 - Edycja zadania. 19

Tabela 5.12: PU-12 - Dodawanie zadania. 19

Tabela 5.13: PU-13 - Dodawanie podzadania. 20

Tabela 5.14: PU-14 - Edycja podzadania. 20

Tabela 5.15: PU-15 - Usuwanie podzadania. 21

Tabela 6.1: PI-01 - Fragmenty oraz komunikacja między nimi. 37

Tabela 6.2: PI-02 - Typy danych w SQLite oraz przykład boolean. 38

Tabela 6.3: PI-03 - Typ danych dla dat. 38

Tabela 6.4: PI-04 - Zastosowanie klasy abstrakcyjnej Calendar. 39

Tabela 6.5: PI-05 - Reprezentacja daty i godziny   
 przy pomocy DatePicker oraz TimePicker. 39

Tabela 6.6: PI-06 - Dobór formatu godziny. 40

Tabela 6.7: PI-07 - Wykorzystanie ExpandableListView. 41

Tabela 6.8: PI-08 - Podgląd implementowanej bazy danych. 41

Tabela 6.9: PI-09 - Kaskadowe usuwanie danych. 42