**Politechnika Krakowska  
im. Tadeusza Kościuszki**

Wydział Fizyki, Matematyki i Informatyki

**Kraków 2018**

**Kamil Wanat**

Numer albumu: 114021

**Small business management – mobile application**

**Praca inżynierska   
 na kierunku Informatyka**

Praca wykonana pod kierunkiem:  
**dr hab. Zbisława Tabora**

**Uzgodniona ocena:**......................................

...........................................................................podpisy promotora i recenzenta





**Aplikacja mobilna wspomagająca zarządzanie małym przedsiębiorstwem**

**Spis treści**

[**1.** **Wstęp** 2](#_Toc503173982)

[**1.1** **Temat i cel pracy** 2](#_Toc503173983)

[**1.2** **Zakres pracy** 2](#_Toc503173984)

[**2.** **Część teoretyczna** 3](#_Toc503173985)

[**2.1** **Przegląd podobnych rozwiązań** 3](#_Toc503173986)

[**2.2** **Opis użytych technologii** 6](#_Toc503173987)

[**2.3** **Opis użytych wzorców projektowych** 9](#_Toc503173988)

[**2.4** **Dostęp do bazy danych** 12](#_Toc503173989)

[**2.5** **Mapowanie obiektowo-relacyjne** 12](#_Toc503173990)

[**3.** **Projekt aplikacji** 13](#_Toc503173991)

[**3.1** **Model opisowy** 13](#_Toc503173992)

[**3.2** **Przypadki użycia** 14](#_Toc503173993)

[**3.3** **Wymagania funkcjonalne i niefunkcjonalne** 17](#_Toc503173994)

[**3.4** **Architektura systemu** 18](#_Toc503173995)

[**4.** **Moduły aplikacji** 22](#_Toc503173996)

[**4.1** **Moduł użytkownika** 23](#_Toc503173997)

[**4.2** **Moduł grafików** 24](#_Toc503173998)

[**4.3** **Moduł urlopów** 25](#_Toc503173999)

[**4.4** **Moduł wiadomości** 27](#_Toc503174000)

[**4.5** **Moduł magazynowy** 28](#_Toc503174001)

[**5.** **Część implementacyjna** 29](#_Toc503174002)

[**5.1** **Widok paska nawigacji** 30](#_Toc503174003)

[**5.2** **Zdarzenia bazodanowe** 30](#_Toc503174004)

[**5.3** **Obiekty tabel w aplikacji** 32](#_Toc503174005)

[**5.4** **Sposób dostępu do bazy danych** 33](#_Toc503174006)

[**6.** **Podsumowanie** 34](#_Toc503174007)

[**6.1** **Wnioski** 34](#_Toc503174008)

[**6.2** **Propozycje rozwoju projektu** 35](#_Toc503174009)

[**7.** **Spis rysunków** 37](#_Toc503174010)

[**8.** **Bibliografia** 38](#_Toc503174011)

1. **Wstęp**
   1. **Temat i cel pracy**

Tematem mojej pracy inżynierskiej jest aplikacja mobilna wspomagająca zarządzanie małym przedsiębiorstwem. Program nazwałem MMA – Mobile Management Assistant. Do podjęcia tego tematu skłoniła mnie praca w jednym z Krakowskich przedsiębiorstw. Liczba pracowników zatrudnionych w tej działalności oscyluje w okolicach 20 osób. Jest więc to małe przedsiębiorstwo zarządzane bezpośrednio przez właściciela. Podczas pracy w tej firmie zauważyłem pewne trudności jakie napotykali zarówno pracownicy, jak i szefostwo. Każdy z pracowników musi przesyłać do pracodawcy swoją „dostępność” na kolejny miesiąc. Rozpisywane w niej są wszystkie dni miesiąca, wraz z godzinami w których pracownik może się pojawić. Następnie właściciel ustala grafiki na kolejne tygodnie, w których zawiera pracowników oraz ich godziny pracy. Cały proces jednak był chaotyczny. „Dostępności” napływały poprzez różne media, od wiadomości sms, poprzez komunikatory internetowe, aż po wiadomości e-mail oraz wydrukowane tabelki. Postanowiłem usystematyzować proces tworzenia grafików, tak aby mogły być one przechowywane w jednolitej postaci, oraz mogły być dostępne w jednym miejscu.

Dodatkowo chciałem usprawnić proces zarządzania magazynem prowadzonym

w przedsiębiorstwie. Nader często występowała sytuacja, w której brakowało pewnych artykułów niezbędnych do działania firmy. Dzięki aplikacji możliwe będzie kontrolowanie stanu magazynu.

Aplikacja posiada również inne, mniej istotne moduły takie jak wnioski urlopowe, bądź sekcję news, które zostaną omówione w dalszej części pracy.

* 1. **Zakres pracy**

Zakres niniejszej pracy obejmuje między innymi skontaktowanie się z właścicielem

wybranego przedsiębiorstwa w celu zebrania informacji dotyczących funkcjonowania firmy oraz potrzeb danego pracodawcy. Kolejnym krokiem było zaprojektowanie aplikacji, wybór odpowiednich technologii, oraz dalsze konsultacje z właścicielem przedsiębiorstwa. Dzięki współpracy z potencjalnym klientem aplikacji, mogła ona zostać dopasowana do potrzeb małych przedsiębiorstw. Następnie aplikacja była implementowana oraz testowana.

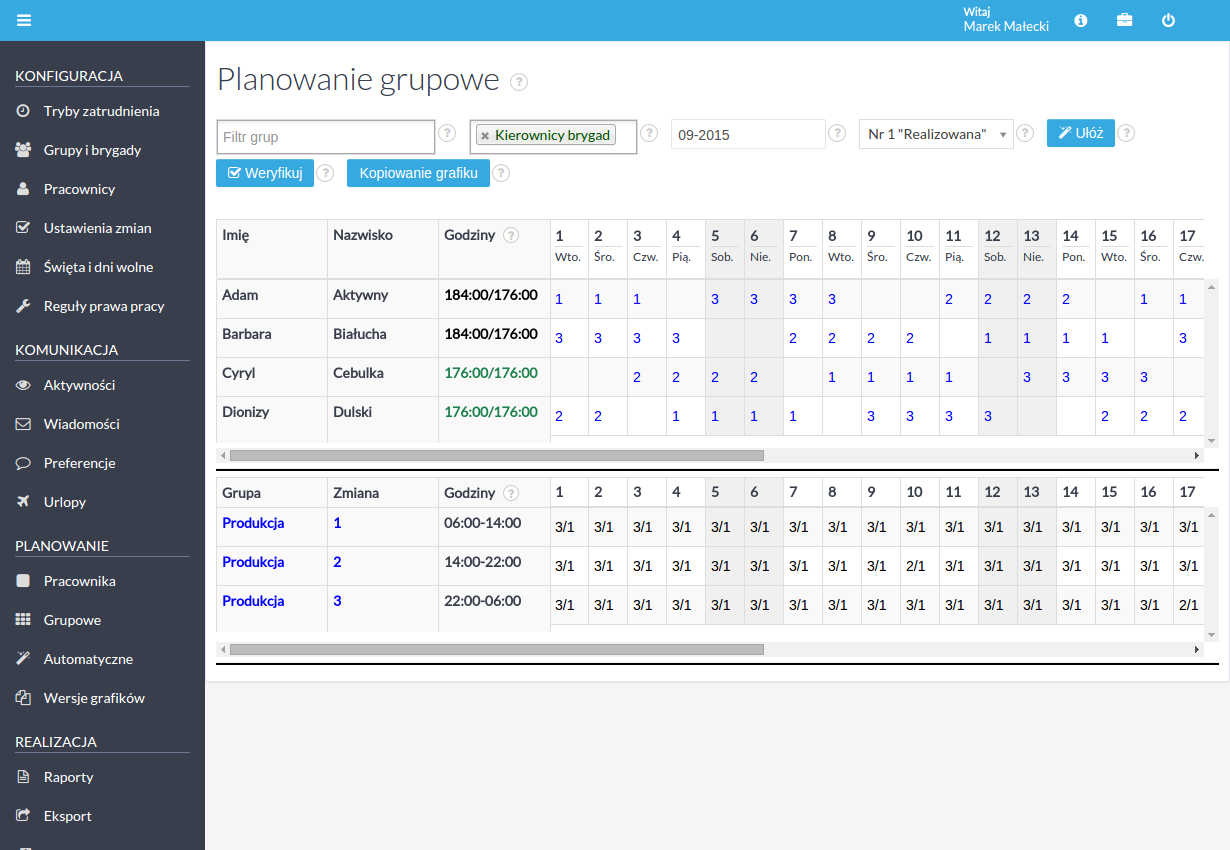
1. **Część teoretyczna**
   1. **Przegląd podobnych rozwiązań**

Na rynku istnieje obecnie niezliczona ilość aplikacji wspomagających zarządzanie małymi przedsiębiorstwami. Z łatwością znaleźć można także programy pozwalające na administrowanie przedsiębiorstwami średnimi, oraz systemy które ułatwiają prowadzenie dużych korporacji. Skupiając się jedynie na obszarze, w którym znajduję się moja praca inżynierska, użyć możemy kilkunastu programów pozwalających na efektywną administrację działalnością gospodarczą. Są to zarówno programy desktopowe, aplikacje mobilne dla różnych systemów mobilnych, a także serwisy internetowe.

Najpopularniejszym obecnie systemem do tworzenia grafików jest serwis sling.is. Pozwala on na łatwe przesyłanie dostępności oraz tworzenie grafików. Jego niewątpliwą zaletą jest wersja mobilna dzięki której system sling możemy mieć zawsze przy sobie, zainstalowany na swoim telefonie.

Kolejnych narzędziem umożliwiającym tworzenie grafików oraz wniosków urlopowych jest program „Grafik Urlopowy”. Za jego pomocą pracodawca może utworzyć plan pracy, oraz urlopów na cały rok. Program pozwala na wygenerowanie list pracowniczych, oraz ich rozesłanie do pracowników w celu ich sprawdzenia, jednak aby istniała możliwość jednoczesnej pracy kilku użytkowników, jedna z maszyn musi działać w trybie serwera. Pozwala to na bezpieczne połączenie się z bazą danych programu.

Ostatnim omawianym przeze mnie systemem wspomagającym tworzenie grafików jest serwis „ePlanist”, pozwalający na tworzenie grafików w oparciu o preferencje urlopowe pracowników, reguły prawa pracy oraz własne, zdefiniowane przez właściciela reguły. Program pozwala na zautomatyzowanie procesu tworzenia grafików poprzez wbudowane w aplikację algorytmy. Po skończonym procesie pracodawca ma możliwość edycji grafiku.



Rysunek : Przykładowy zrzut ekranu programu ePlanist

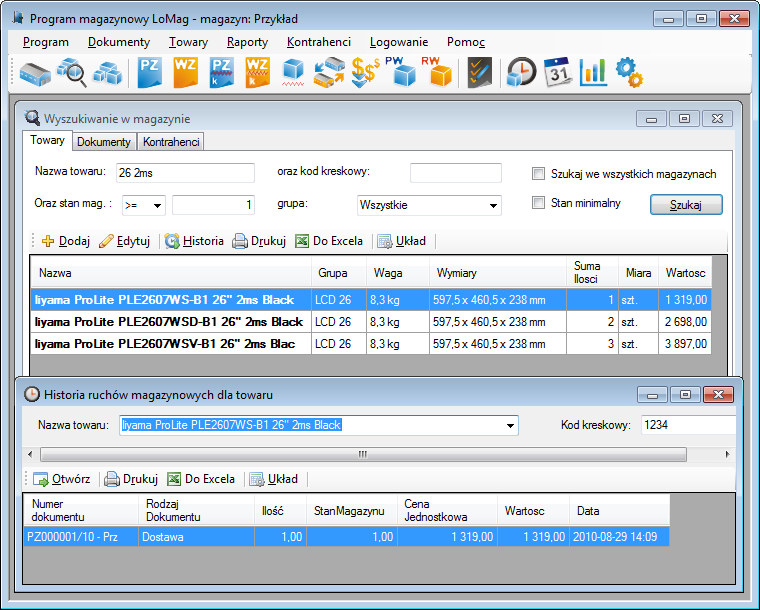
Kolejnymi omawianymi przeze mnie programami są aplikacje pozwalające na prostą obsługę magazynu. Dzięki takim rozwiązaniom przedsiębiorcy mogą w prosty sposób zarządzać swoimi magazynami oraz kontrolować ich stan. Podobnie jak w przypadku systemów ułatwiających tworzenie i przechowywanie grafików dla pracowników, tak i w przypadku programów zarządzających magazynem, wybór rozwiązań jest bardzo duży. Na rynku znaleźć można proste darmowe programy, pozwalające na zarządzanie małymi magazynami, a także duże komercyjne systemy umożliwiające zarządzanie dużymi magazynami. Omówię teraz kilka rozwiązań.

Weaver WMS, to darmowe rozwiązanie przeznaczone dla przedsiębiorców, którzy prowadzą średniej wielkości magazyn. Program pozwala na łatwe zarządzanie stanem produktów, wystawianie faktur, tworzenie zamówień od dostawców, a także przygotowywanie list zakupowych dla klientów. System posiada też funkcję planowania magazynu, dzięki czemu użytkownik może w prosty sposób dowiedzieć się w którym miejscu magazynu należy szukać danego produktu. Ponadto, oprogramowanie automatycznie tworzy raporty podsumowujące, które towary cieszą się największym zainteresowaniem, a które z nich rotują najsłabiej.

Darmowe rozwiązanie ifirma.pl dostarczane jest przez spółkę IFIRMA SA, która

tworzy pakiety oprogramowania przeznaczone do zarządzania firmą. Serwis ifirma.pl pozwala na prowadzenie księgowości internetowej, biura rachunkowego, magazynu, a także wystawianie faktur. Część z funkcji jest dodatkowo płatna, jednak wykorzystanie samego programu magazynowego jest darmowe. Serwis internetowy do zarządzania magazynem pozwala między innymi na import magazynu z pliku CSV, łatwe zarządzanie stanem magazynu. W ramach jednego konta w systemie ifirma możliwe jest utworzenie kilku magazynów. Niewątpliwą zaletą jest możliwość zarządzania magazynem z każdego miejsca, ze względu na to, iż jest to serwis internetowy. Jednakże w sytuacji w której brakuje dostępu do sieci Internet, dostęp do zasobów jest niemożliwy.

Ostatnim omówionym tutaj rozwiązaniem jest komercyjny system LoMag dostarczany przez firmę LONGINT. Program został utworzony z wykorzystaniem bibliotek .NET Framework. Baza danych magazynu oparta jest o technologię Microsoft SQL Server Expres. LoMag to program pozwalający na zarządzie magazynem w małym lub średnim przedsiębiorstwie. Dzięki komercyjnemu charakterowi programu, udostępnia on użytkownikowi wiele przydatnych funkcji, ułatwiających kontrolowanie zasobów przedsiębiorstwa. Użytkownik systemu może za jego pomocą tworzyć najważniejsze dokumenty magazynowe takie jak: przyjęcie magazynowe, wydanie z magazynu, dokumenty rozchodowe i przychodowe, podsumowania, raporty, a także dokumenty remanentowe. Aplikacja pozwala na wielostanowiskowy dostęp do magazynu oraz na tworzenie i zarządzanie wieloma magazynami w ramach jednej aplikacji. Niezawodnym atutem aplikacji jest możliwość tworzenia i drukowania raportów dobowych oraz innych dokumentów bezpośrednio z programu LoMag.



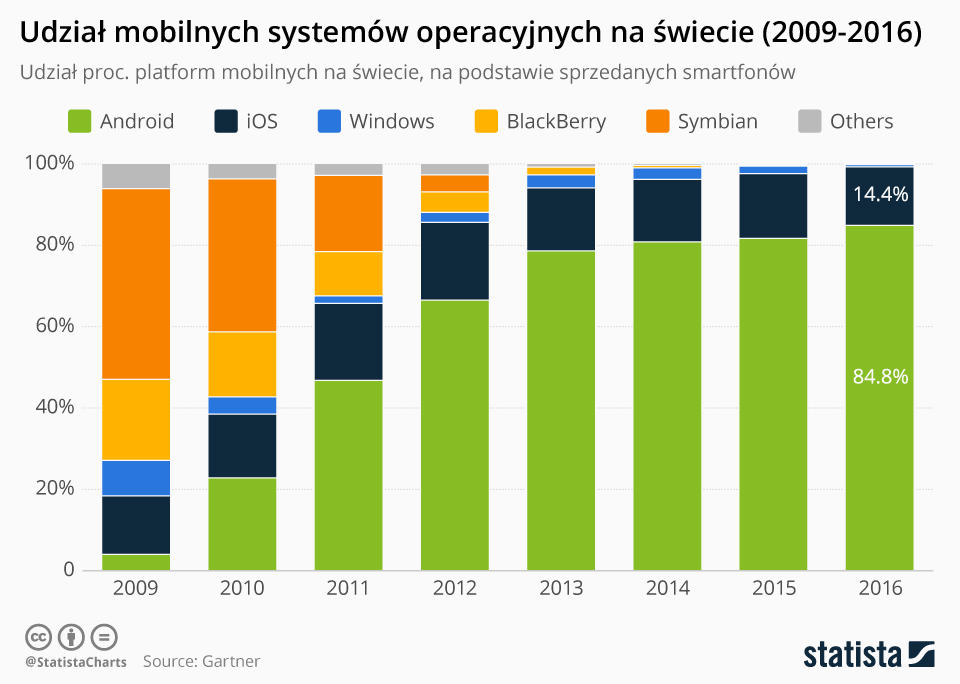
Rysunek : Przykładowy zrzut ekranu systemu LoMag.

* 1. **Opis użytych technologii**

W swojej pracy inżynierskiej wykorzystałem kilka technologii które pozwoliły mi, na efektywne zaprojektowanie oraz zaimplementowanie aplikacji. Dzięki wybranym technologiom projekt został poprawnie zaimplementowany, a podczas jego tworzenia możliwy był nadzór nad kolejnymi jego wersjami, dzięki zastosowaniu systemu kontroli wersji.

Android to system operacyjny przeznaczony dla urządzeń mobilnych tj. telefonów

komórkowych, tabletów, smartfonów, czy nowoczesnych telewizorów.[3] Android oparty jest o jądro systemu Linux. Początkowo rozwijany przez firmę Android Inc, obecnie utrzymywany przez Open Handset Alliance. Android od roku 2013 jest najpopularniejszym systemem operacyjnym na urządzenia mobilne, deklasując przy tym system iOS, czy Windows Mobile. Prostota tworzenia aplikacji współpracujących z systemem Android sprawiła, że obecnie zrzesza on wokół siebie bardzo dużą społeczność deweloperów. Dzięki temu Android może zostać rozszerzony przez miliony aplikacji przygotowanych dla użytkowników zarówno przez firmy deweloperskie, jak i samodzielnych programistów. Użytkownicy chętnie sięgają po urządzenia z systemem Android, ze względu na intuicyjny interfejs użytkownika, stabilne działanie oraz wspomnianą wcześniej liczbę aplikacji rozszerzających działanie systemu. W niniejszej pracy zdecydowałem się na wykorzystanie systemu Android ze względu na jego wielką popularność. Obecnie większość z nas ma dostęp do co najmniej jednego urządzenia z systemem Android. Dzięki temu każdy z pracowników będzie mógł w łatwy sposób korzystać z aplikacji przeznaczonej dla firmy.



Rysunek : Udział mobilnych systemów operacyjnych na świecie.

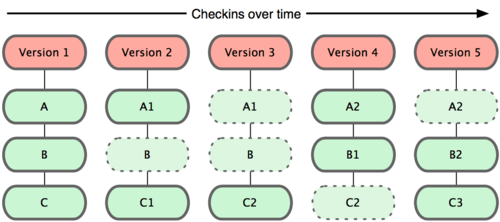
Java to nowoczesny, obiektowy język programowania utworzony przez firmę

Sun Microsystems. Java została zaprojektowana w latach 90 pod kierunkiem Jamesa Goslinga pod roboczą nazwą OAK. Składnia języka została zaczerpnięta z języków programowania takich jak: C, C++. W roku 1995 została wydana pierwsza publiczna implementacja języka, pod znaną nam dziś nazwą Java, wraz z obietnicą, że raz napisany program będzie mógł być uruchomiony na wielu platformach. Możliwe to było dzięki zastosowaniu kompilacji do kodu bajtowego a następnie jego uruchomieniu przez maszynę wirtualną Javy.[1] W 2007 roku została opublikowana całość kodu maszyny wirtualnej Javy na licencji GNU General Public License. Trzy lata później firma Oracle Corporation ogłosiła dalszy rozwój Javy. W pracy inżynierskiej zdecydowałem się na wykorzystanie Javy, ze względu na jej dużą popularność wśród developerów. Posiada ogromną społeczność programistów wspierającą rozwój języka. Dzięki temu wciąż poszerza się zbiór dostępnych bibliotek rozszerzających funkcjonalność Javy. Jest ona także obecnie najpopularniejszym językiem programowania wykorzystywanym w implementacji programów na systemy Android, choć nie jest to jedyny dostępny dla programistów język wykorzystywany do programowania aplikacji mobilnych przeznaczonych na wyżej wymienioną platformę.[4]

GIT to rozproszony system kontroli wersji. Utworzony został w 2005 roku przez

Linusa Torvaldsa. GIT początkowo został wykorzystany przy rozwoju jądra systemu Linux. GIT pozwala każdemu użytkownikowi na utrzymywanie lokalnej, pełnej kopii repozytorium, oraz pracę na niej. Dzięki temu każdy z użytkowników może pracować z projektem, bez wymaganego dostępu do sieci Internet. Po zakończeniu pracy zmiany dokonane na lokalnym repozytorium mogą zostać wysłane na repozytorium zdalne. Dodatkowo, GIT pozwala na tworzenie gałęzi, dzięki którym możliwe jest testowanie pewnych funkcjonalności poza główną gałęzią. Mechanizmy spójności danych, w jakie wyposażony jest system, pozwalają na bezpieczne jego wykorzystywanie bez ryzyka utraty danych, czy ich zmiany bez reakcji ze strony GIT. Do wyznaczania sum kontrolnych system kontroli wersji wykorzystuje funkcję skrótu SHA-1. System traktuje dane jak zestaw migawek. Po każdym zatwierdzeniu zmian tworzony jest obraz pokazujący jak wyglądają wszystkie pliki a następnie zapisuje referencję do tych obiektów. W celu optymalizacji wydajności, GIT nie zapisuje ponownie pliku które nie został zmieniony, a jedynie referencję do jego wersji poprzedniej.[5] W pracy inżynierskiej został wykorzystany system kontroli wersji GIT, ze względu na jego bezpieczeństwo. W każdej chwili możliwy jest powrót do poprzednich wersji.

Dzięki prostej i intuicyjnej obsłudze, system jest obecnie jednym z najpopularniejszych systemów kontroli wersji. Niewątpliwą jego zaletą jest także możliwość pracy na lokalnej kopii projektu, a następnie jej wysłanie na repozytorium zdalne.



Rysunek : Schemat działania systemu Git.

MySQL to system służący do zarządzania relacyjnymi bazami danych. Model

relacyjnej bazy danych powstał w roku 1970, a jego twórcą był Frank Edgar Codd. Model relacyjnych baz danych oparty został o tabele – podstawowe struktury przechowujące dane. Pomiędzy tabelami mogą zachodzić pewne powiązania, dzięki którym cała baza danych jest spójna. Wyróżniane są trzy fundamentalne związki pomiędzy tabelami: jeden do jeden, jeden do wielu i wiele do wielu. System MySQL dostępny jest obecnie dla większości systemów i różnych architektur sprzętowych.[6] W swojej pracy wykorzystałem bazę danych MySQL ze względu na jej dużą popularność oraz relacyjny charakter. Taka baza danych dobrze współpracuje z programami napisanymi w języku Java, a szybkość jej działania pozwala na lepszy odbiór aplikacji przez jej użytkowników.

Android Studio to środowisko programistyczne pozwalające na tworzenie aplikacji

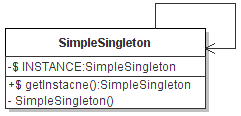
w języku Java dedykowanych dla urządzeń mobilnych z systemem Android. Pierwsza wersja środowiska została opublikowana w 2013 roku na konferencji Google. Android Studio oparte zostało na systemie IntelliJ. Środowisko do sprawnego funkcjonowania wymaga jedynie bibliotek JDK. Zaimplementowany został tutaj także menager, dzięki któremu użytkownicy mogą zarządzać zainstalowanymi składnikami systemu. W niniejszej pracy zdecydowałem się na wykorzystanie środowiska Android Studio, ze względu na jego stabilną pracę, oraz niezwykle bogatą funkcjonalność. Program udostępnia deweloperom wielu przydatnych narzędzi, dzięki którym implementacja aplikacji na system Android staje się o wiele prostsza. Narzędzia te ułatwiają tworzenie kolejnych elementów aplikacji, a graficzne środowisko projektowania, pozwala na intuicyjne tworzenie layoutów, co nierzadko jest dużym ułatwieniem dla programistów.[7]

* 1. **Opis użytych wzorców projektowych**

Wzorce projektowe, to opisy pewnych problemów często pojawiających się podczas

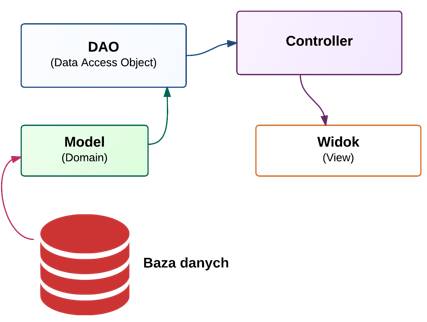
projektowania oraz implementacji oprogramowania, a także sposoby ich rozwiązania. Za pomocą wzorców projektowych, programista może w prosty sposób zaimplementować pewne rozwiązanie, które będzie przejrzyste i czytelne dla niego samego, oraz późniejszych developerów rozwijających dane oprogramowanie. Przy pomocy wzorców projektowych nazywamy standardowe struktury w projekcie, oraz tworzymy ich abstrakcje. Staramy się, aby dana struktura była przydatna w tworzeniu projektów i nadawała się do wielokrotnego użycia. We wzorcu projektowym określone są klasy, oraz ich role i współdziałanie z innymi klasami, a także podział zadań.

Wzorzec projektowy Singleton, to jeden z najprostszych wzorców projektowych. Gwarantuje on, że klasa będzie posiadać tylko jeden egzemplarz, i zapewnia globalny dostęp do niego. Singleton pozwala na zapewnienie kontroli dostępu do swojego jedynego egzemplarza, poprzez kapsułkowanie instancji swojej własnej klasy, a także pozwala na zmniejszenie zaśmiecenia przestrzeni nazw przez zmienne globalne, przechowujące swoje jedyne instancje. Dodatkowym atutem wzorca jest fakt, iż instancja klasy jest tworzona dopiero przy pierwszym pobraniu instancji Singletona. Dzięki temu instancja tworzona jest dopiero w momencie w którym jest potrzebna. Jeśli nie zajdzie taka potrzeba, instancja klasy Singleton nigdy nie zostanie utworzona.



Rysunek : Struktura wzorca Singleton

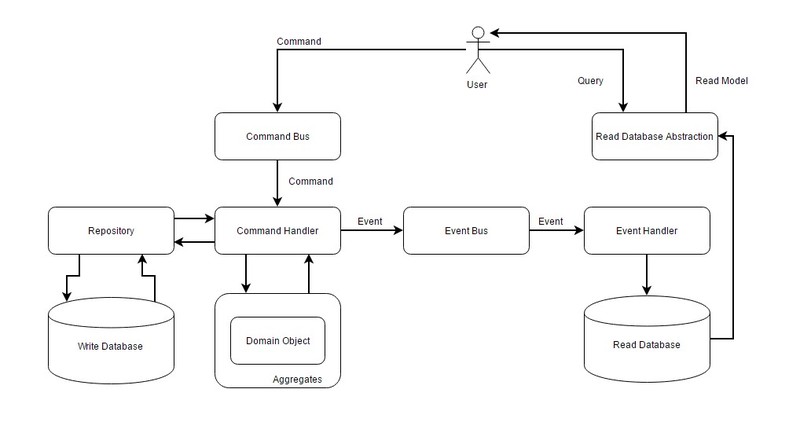
Wzorzec projektowy DAO (Data Access Object) pozwala programiście, na zapewnienie jednolitego dostępu do różnych źródeł danych. Dzięki DAO możliwe jest uniezależnienie warstwy logiki biznesowej od warstwy dostępu do danych. Wzorzec ten bardzo często wykorzystywany jest przy tworzeniu aplikacji, które zapewniają użytkownikowi dostęp do bazy danych. Dzięki DAO aplikacja nie musi znać miejsca składowania swoich danych, a jego zmiana nie pociąga za sobą zmiany kodu źródłowego logiki biznesowej. Wykorzystanie wzorca prowadzi do utworzenia nowej warstwy w architekturze systemu, jednak w aplikacjach, w których wysoka wydajności nie jest krytycznym czynnikiem, DAO może zostać z powodzeniem zaimplementowane, co znacznie uprości pracę przy implementacji oraz późniejszej rozbudowie systemu.



Rysunek : Sposób wykorzystania DAO w aplikacji.

Kolejnym wzorcem wykorzystanym przeze mnie do zaimplementowania tej aplikacji,

był CQRS (Command Query Responsibility Separation). Zanim jednak opiszę ten wzorzec, chciałbym wspomnieć kilka słów o koncepcie, z jakiego się on wywodzi. W roku 1986 Bertrand Mayer przedstawił koncept CQS (Command Query Separation). Zakładał on, że każda metoda w programie musi być zakwalifikowana do jednej z dwóch grup: Command (metody zmieniające coś w systemie, ale nie zwracające nic), oraz Query (metody nie zmieniające systemu, ale zwracające pewne dane). Blisko 20 lat później przedstawiony został następca CQS, czyli CQRS. Wzorzec ten niejako zmusza programistę do rozdzielenia komend i zapytań nie tylko na różne metody, ale także na różne obiekty. Dzięki temu, w projekcie powstają obiekty służące do zapisu pewnych informacji, oraz obiekty dzięki którym informacje te są odczytywane. Oczywiście przy takim podejściu, ilość kodu jest zwiększona, jednak jego zastosowanie pozwala nam na widoczne rozdzielenie operacji zapisu od odczytu.



Rysunek : Schemat wykorzystania wzorca CQRS.

* 1. **Dostęp do bazy danych**

W aplikacji MMA-Mobile Management Assistant dostęp do bazy danych

zrealizowany został przy wykorzystaniu wbudowanego w język Java interfejsu programowania JDBC (Java DataBase Connectivity). Interfejs umożliwia aplikacjom napisanym w języku Java (niezależnym od platformy) na porozumiewanie się z lokalnymi, a także zdalnymi bazami danych, z wykorzystaniem języka SQL. Jest on odpowiednikiem standardu ODBC (Open DataBase Connectivity), niezależnego od języka interfejsu, stworzonego przez SQL Access Group. Aby wykorzystać interfejs JDBC w projekcie tworzonym na system Android, konieczne jest pobranie biblioteki JDBC, oraz dołączenie jej do projektu.

* 1. **Mapowanie obiektowo-relacyjne**

Mapowanie obiektowo-relacyjne, to nowoczesne podejście do zagadnienia współpracy

aplikacji z bazą danych. Problemem jest tutaj sposób komunikacji oprogramowania, które bazuje na obiektach, z relacyjną bazą danych, która nadal jest najczęściej wykorzystywaną bazą danych, pomimo ciągle pojawiających się nowych rozwiązań. Do tej pory nie powstała akceptowana przez wszystkich koncepcja obiektowej bazy danych, choć oczywiście pojawiają różne jej wersje, takie jak repozytoria XML, czy próba rozszerzania języków programowania, w celu przechowywania zserializowanych obiektów. Jednak wciąż najczęściej, oraz najchętniej wykorzystywane są relacyjne bazy danych, ze względu na ich szybkość, stabilność oraz zapewniony poziom bezpieczeństwa. Przy wykorzystaniu relacyjnych baz danych pojawia się jednak pewien problem, związany z przeniesieniem logicznej struktury obiektowej, na strukturę relacyjną bazy danych. Sytuacja odwrotna (wykorzystanie tabelarycznej struktury bazy danych w programie opartym o logiczną strukturę obiektów) także nie jest możliwa. Aby zatem możliwa była współpraca aplikacji z bazą danych, wykorzystywane jest mapowanie obiektowo relacyjne. Dzięki temu programista przy zapisie do bazy danych może działać na obiektach, które są mapowane na relacje. Odczytane relacje z bazy danych, także są mapowane na obiekty, co pozwala na ich przypisanie do odpowiedniego obiektu.

W mojej pracy wykorzystałem framework OrmLite. Jest to pakiet przeznaczony dla języka Java, pozwalający na mapowanie obiektowo relacyjne. Jest to bardzo wygodne narzędzie, dzięki któremu programista może pominąć cały proces tworzenia własnego systemu mapowania obiektowo-relacyjnego. OrmLite pozwala na tworzenie obiektów tabel poprzez annotacje języka Java, co znacznie ułatwia proces ich tworzenia. Zapewnia także podstawową funkcjonalność DAO (Data Access Object), która następnie może być rozszerzona przez użytkownika. Operacje języka SQL takie jak: SELECT, INSERT, UPDATE, czy DELETE są mapowane na konkretne funkcje, dzięki czemu całą kwerendę wybierającą SELECT możemy zapisać, w postaci jednej linijki kodu języka Java. Zdecydowałem się na wybranie frameworku OrmLite ze względu na jego małą złożoność, a także unikanie nadmiarowości pakietów ORM.



Rysunek : Schemat działania Framework OrmLite.

1. **Projekt aplikacji**
   1. **Model opisowy**

Aplikacja MMA-Mobile Management Assistant pozwala na usprawnienie procesów

biznesowych zachodzących w małym przedsiębiorstwie. Aplikacja powstała z myślą o pracodawcach, oraz ich pracownikach, którzy na co dzień zmagają się z dokumentami składowanymi w różnych miejscach, oraz w różnej formie. Aplikacja mobilna ma za zadanie wspierać użytkowników w takich czynnościach, jak: składanie dostępności pracowniczej, oraz wyświetlanie grafików pracy, składanie wniosków urlopowych i ich akceptacja przez kierowników lub właścicieli. System zapewnia podział ról pomiędzy pracownikami. Wyróżniono w nim trzy poziomy użytkowników: właściciel, kierownik, oraz pracownik. Każdy z użytkowników ma w systemie dostęp do określonych funkcjonalności, w zależności od roli, jaką pełni w przedsiębiorstwie.

Dodatkowo aplikacja dostarcza podstawowej funkcjonalności magazynu, dzięki któremu stan wszystkich produktów wykorzystywanych w przedsiębiorstwie, jest dostępny w jednym miejscu. Dzięki temu możliwe jest kontrolowanie stanu magazynu w prosty sposób. Dodatkowo, funkcja wyświetlania produktów o niskim stanie magazynowym pozwala na łatwe utworzenie zamówienia na produkty, których zaczyna brakować w danym przedsiębiorstwie.

Oczywiście system zapewnia także funkcjonalności dotyczące użytkowników. Za pomocą systemu możliwe jest dodawanie nowych pracowników przez właściciela, nadawanie im uprawnień, zmiana ich danych, a także, jeśli zajdzie taka potrzeba, usunięcia pracowników z systemu. Każdy z pracowników dodatkowo może zmieniać swoje własne dane osobowe takie jak imię, nazwisko, pesel, adres oraz numer telefonu. Pracowników mogą także zmieniać swoje hasło logowania.

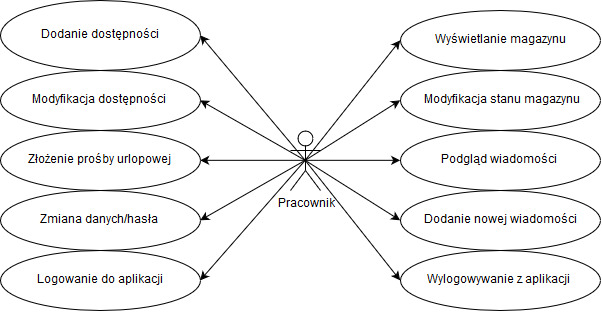
System zapewnia automatyczne zatwierdzanie grafików, na 7 dni przed rozpoczęciem danego dnia roboczego, oraz pozwala na automatyczne usuwanie z bazy danych wiadomości starszych niż 30 dni, czy przedawnionych wniosków urlopowych.

* 1. **Przypadki użycia**

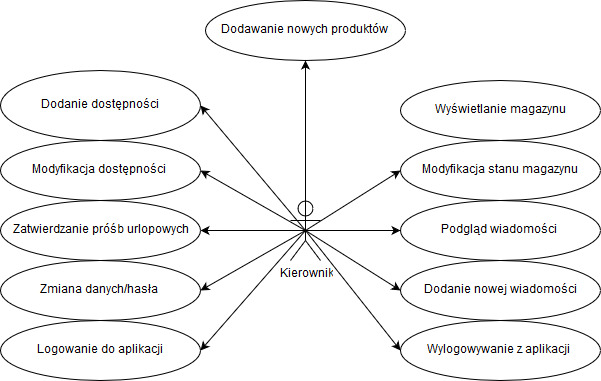
Diagramy przypadków użycia pozwalają na graficzne przedstawienie funkcjonalności

systemu, w sposób w jaki są one widziane od strony użytkownika aplikacji. Diagramy składają się z aktorów, oraz przypadków użycia połączonych ze sobą, za pomocą asocjacji skierowanych. Diagramy przypadków użycia pomimo swojej prostoty są jednym z ważniejszych narzędzi wykorzystywanych podczas projektowania aplikacji.

W niniejszej aplikacji można wyróżnić trzech podstawowych aktorów, którymi są: właściciel, kierownik, oraz pracownik. Diagramy przypadków użycia dla tych aktorów, zostały przedstawione na poniższych schematach. Część funkcjonalności jest wspólna dla wszystkich użytkowników aplikacji, jednak dla zachowania przejrzystości zostały one rozdzielone dla poszczególnych aktorów.



Rysunek : Diagram przypadków użycia dla pracownika.



Rysunek : Diagram przypadków użycia dla kierownika.



Rysunek : Diagram przypadków użytkownika dla właściciela.

Na powyższych diagramach łatwo można zauważyć fakt, iż większość funkcji jest taka sama zarówno dla pracownika, kierownika jak i właściciela. Kierownik jest rozszerzeniem pracownika i posiada dodatkową funkcję, pozwalającą na zatwierdzanie urlopów innych pracowników. Sam jednak nie musi składać wniosków o urlopy poprzez aplikację, ponieważ w małym przedsiębiorstwie prośby takie kierowane są zazwyczaj bezpośrednio do właściciela firmy. Dodatkowo, Kierownik ma możliwość dodawania nowych produktów do magazynu oraz wyświetlania produktów, których ilość przekroczyła dopuszczalny stan krytyczny. Największe uprawnienia w aplikacji, a co za tym idzie dostęp do największej liczby funkcji posiada właściciel. Konto właściciela jest swoistym rozszerzeniem konta Kierownika. Właściciel odpowiedzialny jest za zatwierdzenie, oraz poprawę grafików. Dodatkowo właściciel jest zwolniony z odpowiedzialności dodawania grafiku dla siebie. Użytkownik z takimi uprawnieniami ma możliwość tworzenia kont dla pozostałych pracowników, oraz nadawania im uprawnień. Jako jedyny również może edytować dane każdego z pracowników a także usuwać użytkowników z systemu MMA.

* 1. **Wymagania funkcjonalne i niefunkcjonalne**

Analiza wymagań funkcjonalnych oraz niefunkcjonalnych jest istotnym punktem

w tworzeniu oprogramowania. Pozwala ona na określenie wszelkich funkcjonalności systemu, oraz jego standardów.

Wymagania funkcjonalne to konkretne możliwości jakie zapewnia system. Są to

wszystkie funkcjonalności tworzonego oprogramowania. Wymagania funkcjonalne nie dotyczą jedynie użytkowników, korzystających z danej aplikacji, ale także całego systemu, ze względu na fakt, iż tworzony system sam w sobie może zapewniać pewne funkcjonalności, które są niedostępne dla użytkownika. Poniżej przedstawiona została lista wymagań funkcjonalnych systemu.

* Logowanie użytkownika do systemu
* Odczytywanie wiadomości
* Dodawanie nowych wiadomości
* Możliwość zmiany danych użytkownika
* Możliwość zmiany hasła użytkownika
* Składanie wniosków urlopowych przez pracowników
* Zatwierdzanie/odrzucanie wniosków urlopowych przez kierowników i właściciela
* Dodawanie grafików przez pracowników oraz kierowników
* Zatwierdzanie i zmiana grafików przez właściciela
* Wyświetlanie produktów z magazynu
* Zmiana ilości dostępnych produktów przez każdego pracownika
* Dodawanie nowych produktów przez kierowników i właściciela
* Wyświetlanie produktów o niskim stanie magazynowym przez kierownictwo i właściciela
* Dodawanie nowych pracowników przez właściciela
* Usuwanie pracowników z bazy danych przez właściciela
* Nadawanie i zmiana uprawnień pracownikom przez właściciela
* Automatyczne usuwanie z bazy danych wiadomości starszych niż 30 dni
* Automatyczne usuwanie wniosków urlopowych które uległy przedawnieniu
* Automatyczne zatwierdzanie grafików, na 7 dni przed rozpoczęciem pracy
* Utrzymywanie sesji użytkownika przez 14 dni.

Analiza wymagań niefunkcjonalnych pozwala projektantowi aplikacji, na określenie

tego, jak system powinien się zachowywać. W odróżnieniu od wymagań funkcjonalnych, nie definiujemy tutaj funkcjonalności udostępnianej przez aplikację, a jedynie charakterystykę działania danego oprogramowania. Lista wymagań niefunkcjonalnych została przedstawiona poniżej.

* Łatwa rozbudowa aplikacji
* Bezpieczeństwo danych
* Odporność aplikacji na sytuacje awaryjne
* Wykorzystanie systemu kontroli wersji w celu przechowywania kolejnych wersji aplikacji
* Przyjazny i intuicyjny interfejs graficzny użytkownika
  1. **Architektura systemu**

Aplikacja MMA-Mobile Management Assistant została zaprojektowana

z wykorzystaniem warstwowej architektury systemu. Pozwala to, na rozdzielenie poszczególnych części systemu, na pojedyncze odpowiedzialności. Każda warstwa systemu odpowiedzialna jest za wykonywanie w nim pewnych zadań, a wyniki swojej pracy przekazuje do kolejnych warstw systemu. Dzięki zastosowaniu takiego podejścia, każda z warstw może być osobno implementowana, a w późniejszym czasie także zarządzana i rozbudowywana o kolejne moduły, bez ingerencji w pozostałe warstwy systemu. Aplikacja składa się z następujących warstw:

* Baza danych
* Warstwa modelu danych
* Warstwa dostępu do danych
* Warstwa logiki biznesowej
* Warstwa graficznego interfejsu użytkownika

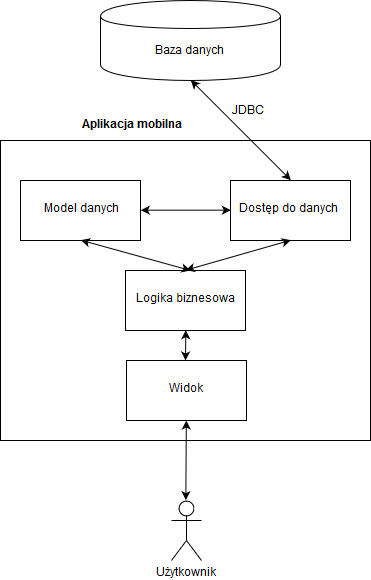
Pierwsza a zarazem najniższa warstwa to baza danych. Służy ona do przechowywania wszelkich informacji wykorzystywanych przez aplikację w swoim działaniu. Baza danych zapewnia bezpieczne i spójne przechowywanie rekordów o użytkownikach, grafikach oraz urlopach. Dodatkowo w bazie danych przechowywane są wszelkie produkty składowane w magazynie przedsiębiorstwa.

Kolejną warstwą architektury systemu jest model danych. Przedstawia on zmapowane do postaci obiektowej relacyjne rekordy pobierane z bazy danych. Dzięki zastosowaniu warstwy modelu danych, wszelkie informacje potrzebne do sprawnego działania aplikacji, mogą być użyte przez wyższe warstwy systemu w prosty sposób, poprzez pobranie obiektowego modelu danych. Pozwala to na łatwe zarządzanie danymi zarówno w przypadku ich pobierania z bazy danych, jak i ich wykorzystywania w trakcie działania systemu.

Warstwa dostępu do danych zapewnia aplikacji dostęp do informacji zapisanych w bazie danych. Moduł ten w dużej części połączony jest z frameworkiem OrmLite, który pozwala na mapowanie obiektowo-relacyjne, oraz zdalny dostęp do bazy danych. Warstwa odpowiedzialna jest za komunikację pomiędzy dwoma warstwami niższymi: modelu danych i bazy danych. Odpowiada za utworzenie połączenia z bazą danych, pobranie informacji wymaganych przez warstwy wyższe, w postaci relacyjnej, ich zmapowanie do postaci obiektowej, oraz przekazanie do warstwy logiki biznesowej. W przypadku zapisu informacji do bazy danych proces przebiega analogicznie, z ta różnicą jednak że dinformacje mapowane są z postaci obiektowej do postaci relacyjnej, a następnie zapisywane do bazy danych.

Warstwa logiki biznesowej odpowiedzialna jest w systemie, za przetwarzanie wszelkich żądań użytkownika, komunikację z warstwą dostępu dodanych a także jeśli zajdzie taka potrzeba, z warstwą modelu danych. Logika biznesowa zapewnia pobieranie danych od użytkownika, ich przetwarzanie, a następnie zwrot do widoku, bądź zapis do bazy, przy wykorzystaniu warstwy dostępu do danych. To właśnie w tym miejscu inicjowane jest żądanie zapisu do bazy, lub odebrania od niej pewnych informacji. Moduł może także komunikować się z warstwą modelu danych, w celu przygotowania nowych obiektów do zapisu w zdalnej bazie, w postaci relacyjnej. Jest on także odpowiedzialny, za zarządzanie widokami, co oznacza, ze decyduje on jakie dane należy wyświetlić, oraz jak w danym momencie powinien wyglądać interfejs graficzny naszej aplikacji.

Najwyższą warstwą aplikacji, a jednocześnie jedyną warstwą z którą użytkownik prowadzi interakcję jest warstwa widoku. To dzięki tej warstwie użytkownik może komunikować się z całością systemu. Wprowadzone przez użytkownika informacje, są następnie przekazywane do warstwy logiki biznesowej, w celu ich przetworzenia i wykonania odpowiedniego żądania, zleconego przez użytkownika. Widok odbiera także polecenia od modułu niższego, dzięki czemu dane mogą być zwrócone do użytkownika. Widok jest statyczny i nie decyduje o tym jakie akcje mają być wykonywane w dalszej części pracy programu.



Rysunek : Schemat architektury systemu.

* 1. **Baza danych**

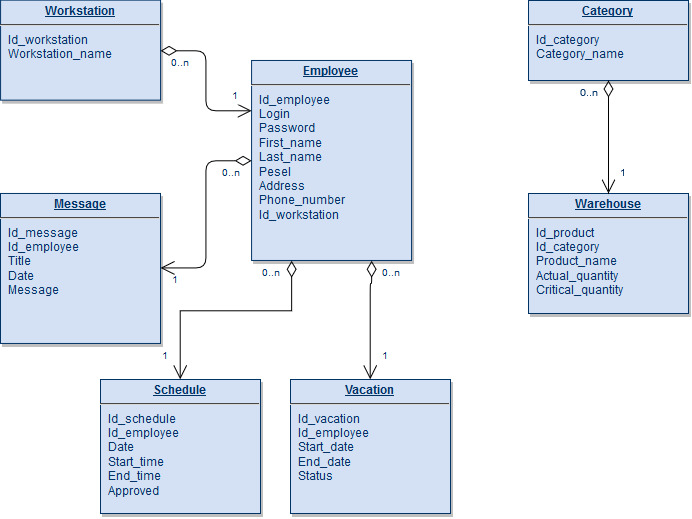
Aplikacja w swoim działaniu wykorzystuje relacyjną bazę danych napisaną

w języku SQL. Jest to relacyjna baza danych zarządzana przez system MySQL. Została ona utworzona na zdalnym serwerze, z wykorzystaniem systemu operacyjnego Linux Debian. Taka konfiguracja zapewnia stabilną pracę bazy danych, oraz zwiększa jej bezpieczeństwo. Jednym z wymagań było umieszczenie jej, na zewnętrznym serwerze. Konieczność taka dyktowana była faktem, iż MMA-Mobile Management Assistant jest aplikacją wieloużytkownikową. Każdy z użytkowników może zatem w tym samym czasie wykonywać pewne operacje na bazie danych. Nie jest możliwe wykorzystanie także lokalnej bazy danych, a następnie jej wysyłanie na serwer zdalny, ze względu na mogące się pojawiać konflikty w przesyłanych informacjach. Struktura bazy danych składa się z dwóch zasadniczych modułów: użytkownika oraz magazynu.

Moduł użytkownika odpowiedzialny jest za przechowywanie danych, związanych z działalnością użytkownika w systemie. Składa się ze zbioru 5 tabel, powiązanych ze sobą relacjami jeden do wielu. W skład modułu użytkownika wchodzą tabele przechowujące informacje o użytkowniku, jego stanowisku, a także modułach aplikacji takich jak grafiki, wiadomości czy żądania urlopowe. Związane jest to z faktem, iż w każdym z wymienionych modułów wymagane jest przechowywanie wiadomości o użytkowniku, który dokonał wpisu informacji do danej tabeli

Drugi z modułów został nazwany „magazynowym”, ze względu na charakter informacji jakie przechowuje. Składa się z kolekcji jedynie dwóch tabel, co w zupełności pozwala na efektywne zarządzanie magazynem małego przedsiębiorstwa. W skład magazynu wchodzi tabela „Category”, pozwalająca na przechowywanie w spójny sposób kategorii produktów, oraz tabela „Warehouse” której zadaniem jest przechowywanie informacji, o przedmiotach przechowywanych w magazynie: ich nazw, ilości, oraz informacji o tym, jaka jest ilość krytyczna dla danego produktu.

Baza danych zapewnia także automatyczne usuwanie przestarzałych wiadomości, oraz wniosków urlopowych. Dodatkową funkcjonalnością zaimplementowaną w systemie MySQL, jest skrypt pozwalający na automatyczne zatwierdzanie wstawionego grafiku, na 7 dni przed nadejściem danego dnia roboczego. Funkcjonalności takie zostały utworzone z wykorzystaniem zdarzeń które mogą zostać zaprogramowane w taki sposób, aby automatycznie wykonywały pewne akcje w określonych momentach czasu.



Rysunek : Diagram ERD bazy danych

1. **Moduły aplikacji**

Aplikacja mobilna została zaimplementowana tak, aby możliwe było jej łatwe

rozbudowywanie, a także zmiana jej funkcjonalności, bez potrzeby zmiany całego systemu. Z tego względu zdecydowałem się na utworzenie swego rodzaju modułów, które są w całości niezależne od siebie. Pozwala to na rozwój jednego z modułów bez ingerencji w pozostałe. Możliwe też jest odłączenie pewnych modułów od aplikacji, lub dodawanie kolejnych. W każdym z tych przypadków, aplikacja nie musi być zmieniana w całości, i może nadal spełniać swoje funkcje. Takie podejście pozwala na zapewnienie ciągłego działania aplikacji. W systemie wydzielonych zostało 5 podstawowych modułów, które postaram się teraz opisać.

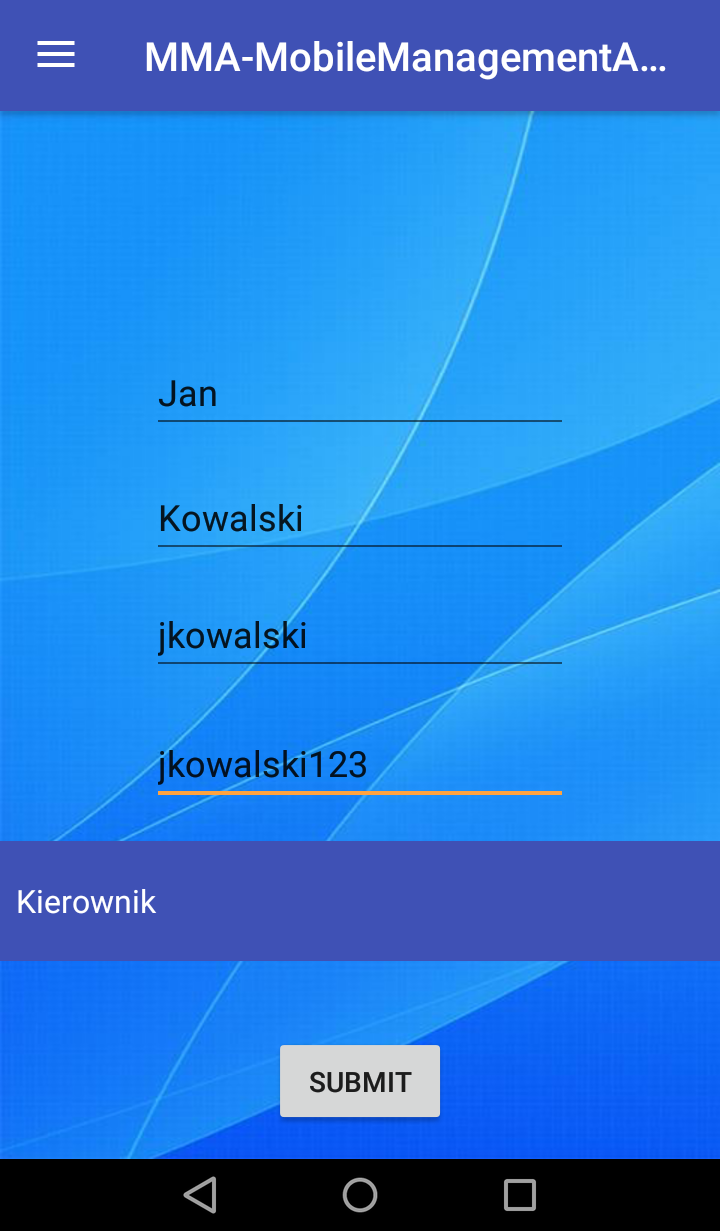
* 1. **Moduł użytkownika**

Aplikacja MMA-Mobile Management Assisstant zaopatrzona została w moduł

Użytkownika, który pozwala na zarządzanie użytkownikami obecnymi w systemie. Pracownik może oczywiście zalogować się do aplikacji za pomocą loginu i hasła, które jest przechowywane w bazie danych w postaci zaszyfrowanej. Sesja zalogowanego użytkownika utrzymywana jest przez 14 dni od momentu ostatniego otwarcia aplikacji, lub też do momentu wylogowania użytkownika z systemu. Użytkownik systemu może dowolnie zmieniać swoje dane, takie jak: imię, nazwisko, adres, pesel oraz numer telefonu. System posiada także funkcję zmiany hasła, dzięki czemu użytkownik może zwiększać bezpieczeństwo swojego konta, przy pomocy częstej zmiany haseł.

Większa funkcjonalność w module użytkownika udostępniona została kontu właściciela, który może dodawać nowych użytkowników do istniejącego już systemu, lub też ich z niego usuwać, w przypadku wygaśnięcia umowy o pracę pomiędzy pracodawcą, a pracownikiem. Właściciel może także zmieniać uprawnienia pozostałych użytkowników. Dzięki temu użytkownicy w systemie mogą spełniać różne role, oraz mieć dostęp do różnych zasobów. Użytkownik z najwyższymi prawami dostępu, ma dostęp do listy wszystkich użytkowników systemu, gdzie może przeprowadzać takie operacje jak: usunięcie użytkownika z systemu, czy zmiana jego danych.

W module użytkownika konta kierowników zostały zrównane w prawach z kontami pracowników. Wynika to z faktu, iż w większości małych przedsiębiorstw, to właśnie właściciele nadają stanowiska pracownikom, a także osobiście zajmują się zatrudnianiem pracowników, czy też przykrym obowiązkiem ich zwalniania.



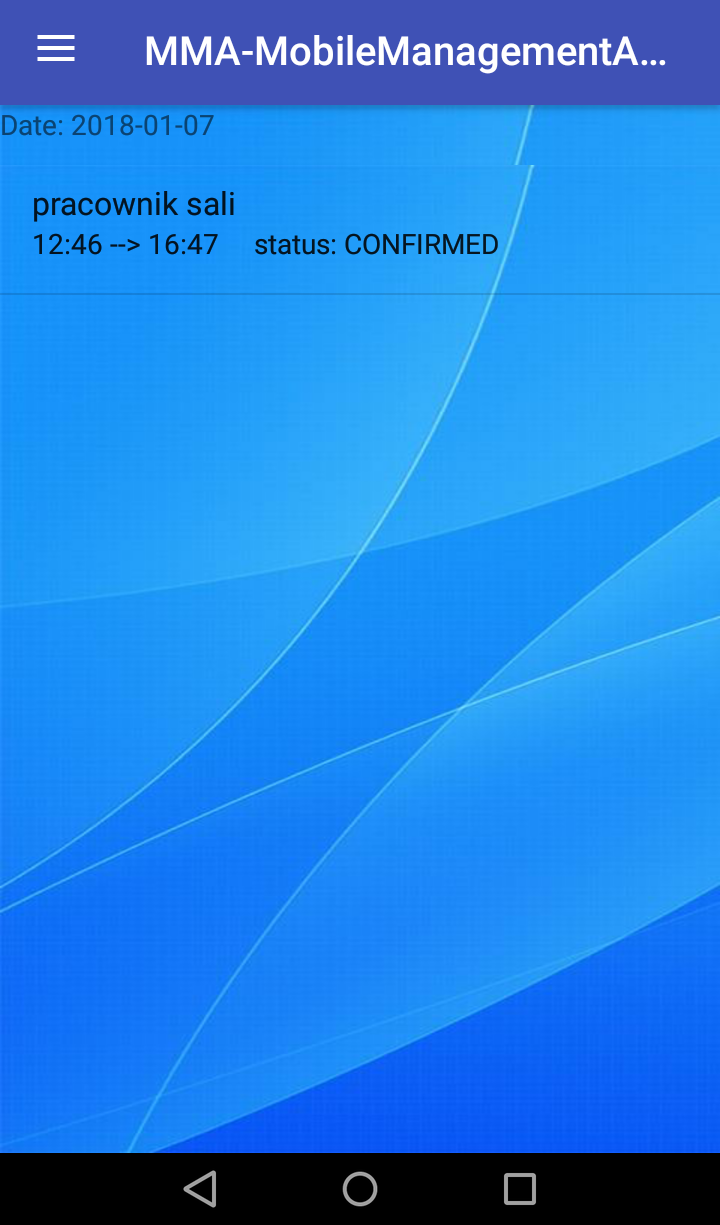
Rysunek : Przykładowa funkcjonalność modułu użytkownika (dodawanie nowych pracowników)

* 1. **Moduł grafików**

Moduł grafikowy jest głównym elementem aplikacji mobilnej

MMA-Mobile Management Assistant. Pozwala on na dodawanie przez pracowników dostępności na kolejne dni pracy, w prosty i przyjemny sposób. Pracownik po wyborze w kalendarzu interesującego go dnia, proszony jest o wybranie godziny rozpoczęcia pracy, oraz jej zakończenia. Godziny te mogą być zmieniane przez pracownika aż do momentu ich zatwierdzenia przez pracodawcę, lub automatycznego potwierdzenia przez system na 7 dni przed rozpoczęciem danego dnia roboczego. Po zatwierdzeniu danych, mogą być one przez pracownika jedynie przeglądane. Konto kierownika nie odróżnia się także w tym module, od konta zwykłego pracownika, ponieważ to właściciele zwykle ustalają godziny pracy w małych przedsiębiorstwach, natomiast kierownictwo sprawuje nadzór nad pracownikami, w czasie ich pracy.

Konto właściciela ma tutaj oczywiście wyższe uprawnienia i dostęp do innych funkcjonalności niż konta pracowników i kierowników. Właściciel przedsiębiorstwa nie ma dostępu do części w której pracownicy składają swoje dostępności. Funkcjonalność ta, zarezerwowana jest jedynie dla użytkowników, ponieważ właściciel przedsiębiorstwa nie musi składać dostępności sam dla siebie. Pozwala to na zachowanie logiki i spójności aplikacji, oraz odwzorowuje rzeczywiste procesy biznesowe zachodzące w firmach. Właściciel może jednak po wyborze danego dnia z kalendarza, wyświetlić listę przesłanych grafików przez swoich pracowników. Lista taka składa się z informacji o personaliach pracownika, oraz proponowanych przez niego godzinach pracy. Właściciel, po naciśnięciu interesującego go grafiku, może wybrać z menu opcję zatwierdzenia danego grafiku, lub jego edycji. Po zatwierdzeniu grafiku przez pracodawcę, możliwość jego edycji jest zablokowana dla pracowników; pracodawca jednak, może nadal edytować wybrane godziny pracy.



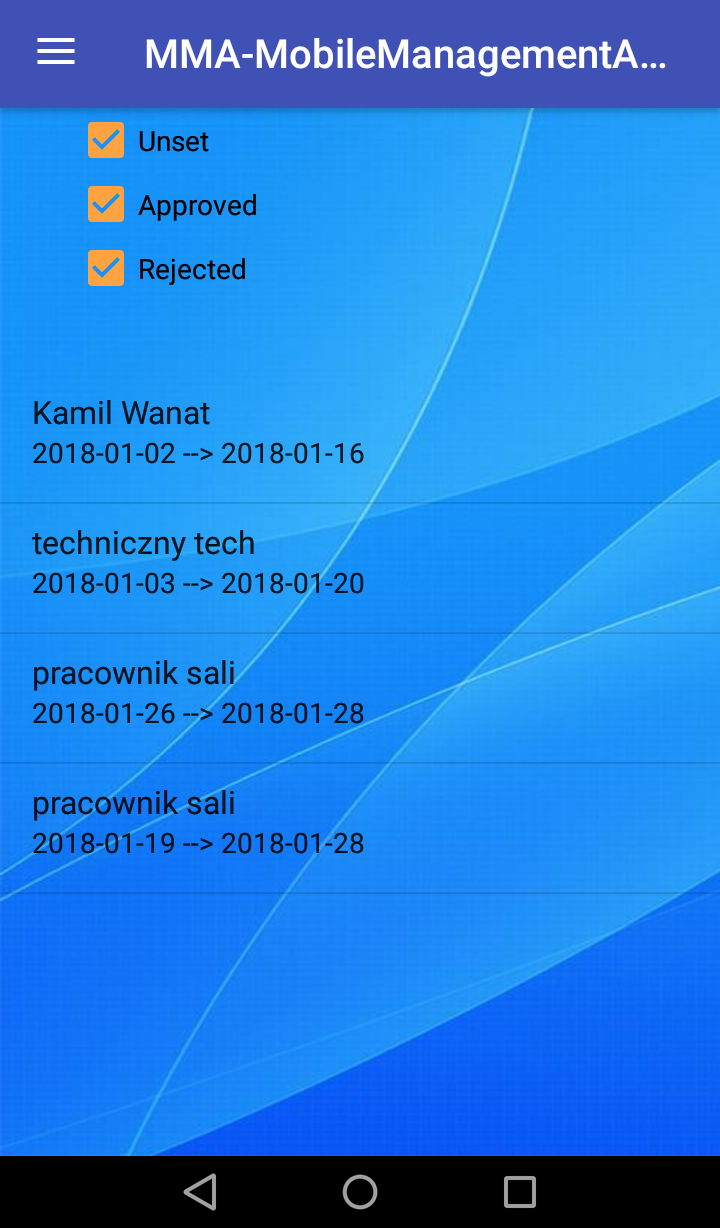
Rysunek : Lista złożonych dostępności przez pracowników na dany dzień.

* 1. **Moduł urlopów**

Aplikacja pozwala także na składanie wniosków urlopowych przez pracowników,

oraz ich zarządzanie przez użytkowników o podniesionych uprawnieniach. Cała funkcjonalność została zawarta w module urlopowym. Pracownicy mogą w prosty sposób złożyć wniosek urlopowy. Wystarczy tylko za pomocą widżetu kalendarza wybrać dzień rozpoczęcia urlopu, oraz dzień jego zakończenia. W aplikacji nie jest wymagane podawanie powodu urlopu. Użytkownik ma także możliwość przeglądania listy własnych wniosków urlopowych. Dane pobrane z bazy danych są wyświetlane w postaci listy, w kolejności od najstarszego wniosku, aż po najnowsze. Dodatkowo użytkownik może zobaczyć jaki status został przypisany do danego wniosku. Aplikacja mobilna w swoim działaniu wyróżnia trzy stany wniosku urlopowego: „nieustawiony”, „zaakceptowany” oraz „odrzucony”. Domyślnie, po złożeniu wniosku przyjmuje on stan „nieustawiony”.

Konta kierownika i właściciela w module urlopowym mają tożsame uprawnienia, i dostęp do tych samych funkcjonalności. Kierownicy nie muszą składać wniosków urlopowych. Zazwyczaj kwestie urlopów kierowniczych są rozstrzygane bezpośrednio u właściciela przedsiębiorstwa, ze względu na wymaganą obecność co najmniej jednego kierownika danego dnia. Użytkownicy o podniesionych uprawnieniach mają zatem dostęp, do pełnej listy wniosków urlopowych. Dane wyświetlane są tutaj w postaci listy, podobnej do listy wniosków urlopowych użytkownika. Możliwe jest tutaj sortowanie elementów listy, według statusu wniosku. Kierownictwo, poprzez zaznaczenie odpowiednich „checkboxów”, może zdecydować jakie typy wniosków mają być obecnie widoczne. Po zaznaczeniu danego wniosku, otwiera się menu dialogowe, za pomocą którego kierownik bądź właściciel, może nadać nowy stan (zaakceptowany/odrzucony) wybranemu wnioskowi. Stany te mogą być dowolnie zmieniane, jednak po pierwszej zmianie stanu, nie ma możliwości powrotu do pozycji: „nieustawiony”. Wnioski urlopowe są przechowywane w bazie danych, do momentu ich przedawnienia, tzn. do czasu, w którym przewidywana data zakończenia urlopu jest datą wcześniejszą niż dzisiejsza. Kasowaniem takich danych zajmuje się zaprojektowany w bazie danych „event” który uruchamiany jest każdego dnia.



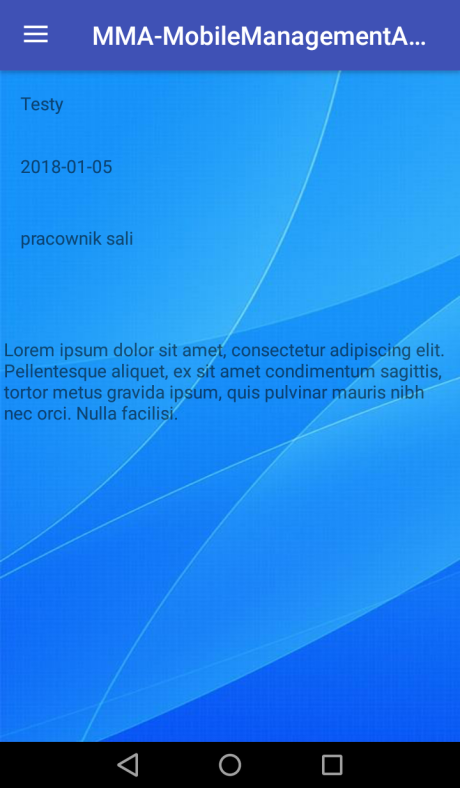
Rysunek : Lista wniosków urlopowych (widok z poziomu kierownika)

* 1. **Moduł wiadomości**

Aplikacja została przeze mnie wyposażona także w moduł wiadomości.

Jest to najmniejszy z obszarów aplikacji, pozwalający na dodawanie, oraz wyświetlanie wiadomości. Dzięki temu możliwe jest przesyłanie prostych komunikatów które dotyczą całego zespołu pracowników. Nie ma tutaj rozróżnienia pomiędzy różnymi typami użytkowników. Każdy ma takie same uprawnienia. Użytkownicy mogą w prostu sposób dodać wiadomość/komunikat, poprzez wpisanie jego tytułu, oraz treści. Po zatwierdzeniu, do wiadomości automatycznie dodawana jest data, oraz identyfikator użytkownika, który wiadomość utworzył. Komunikaty wyświetlane są w postaci listy w głównym menu aplikacji. Każdy z użytkowników może tą listę przeglądać. Zawarte są w niej jedynie podstawowe informacje, takie jak: tytuł wiadomości oraz data jej dodania. Wiadomości są posortowane w kolejności od najnowszej. Po naciśnięciu w element listy, następuje automatyczne przekierowanie użytkownika do szczegółów wiadomości, w których może zobaczyć wszelkie informacje związane z komunikatem, tj. autora, datę dodania, tytuł oraz zasadniczą treść wiadomości.

System ze swojej strony zapewnia automatyczne usuwanie wiadomości starszych niż 30 dni. Funkcjonalność taka została zaimplementowana z wykorzystaniem zdarzeń wbudowanych w system zarządzania bazą danych MySQL. Czas po którym usuwane są wiadomości został ustawiony na 30 dni, ze względu na częste wprowadzanie do systemu wiadomości, które dotyczą przyszłego zdarzenia.





Rysunek : Widok listy wiadomości ekranu startowego (po lewej), oraz szczegółów wiadomości (po prawej).

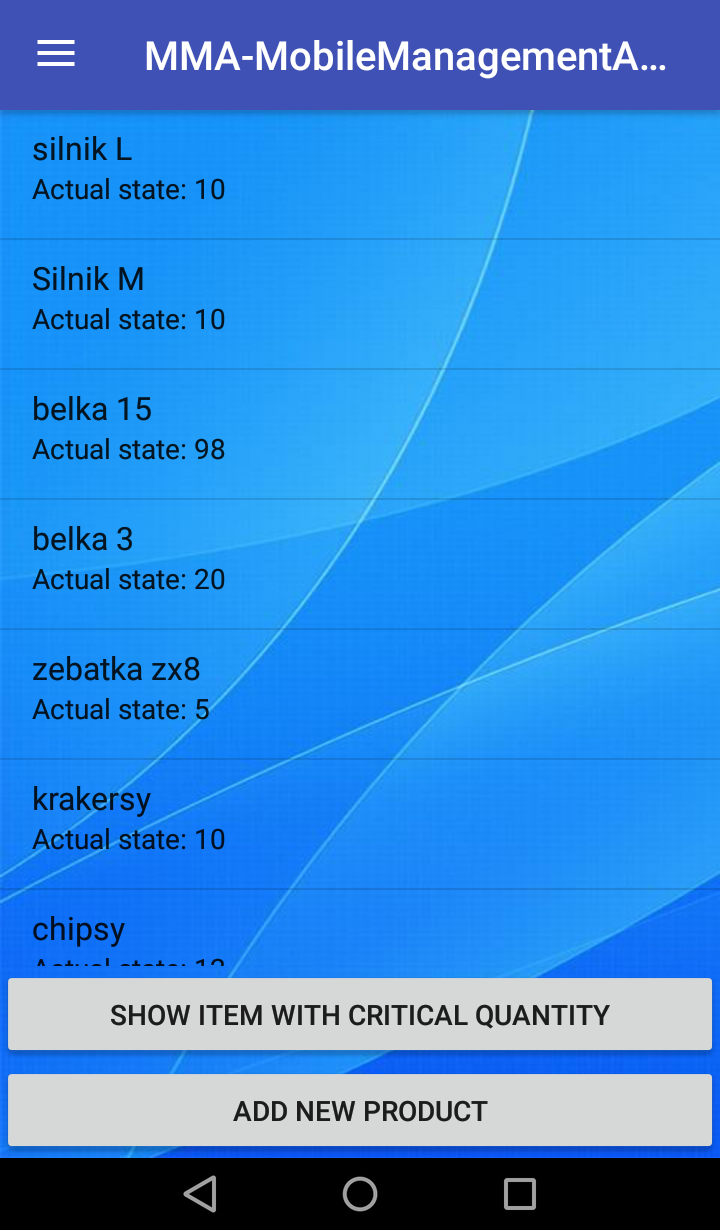
* 1. **Moduł magazynowy**

Ostatnim z modułów jakie obecnie zostały zaimplementowane w aplikacji

jest magazyn. Dostarcza on bardzo podstawowych funkcjonalności, związanych z przechowywaniem produktów w magazynie. Jest to jednak funkcjonalność wystarczająca, do znacznego usprawnienia zarządzania produktami składowanymi w przedsiębiorstwie. Tak jak w poprzednich modułach, tutaj również dokonano podziału funkcjonalności pomiędzy różnych użytkowników.

Zwykli pracownicy mają dostęp jedynie do przeglądania magazynu, oraz zmiany ilości dostępnych produktów po zabraniu czegoś z magazynu, lub dostarczeniu nowych produktów. Po otworzeniu magazynu pracownicy zobaczą listę z produktami które są dla nich dostępne. Mogą oni przeglądać produkty, które odpowiadają ich stanowisku pracy. Każdy produkt opatrzony jest informacją o tym, jaka jest jego aktualna ilość. Pracownicy po naciśnięciu na dany element mogą wybrać czy chcą dodać coś do magazynu, czy coś z niego zabrać. Oczywiście w aplikacji zastosowano ograniczenia, które nie pozwalają użytkownikowi, na pobranie większej ilości danego produktu niż stan magazynowy. Takie działanie mogłoby prowadzić do poważnych błędów w aplikacji, jak również w działaniu przedsiębiorstwa.

Dostęp do większej liczby funkcjonalności został przyznany użytkownikowi kierownik oraz właściciel. Już przy przeglądaniu zawartości magazynu, mają dostęp do pełnej listy magazynowej z wszystkich kategorii. Dodatkowo mogą oni dodawać nowe produkty do magazynu, poprzez podanie nazwy, kategorii, oraz stanu początkowego i stanu krytycznego. Ustawienie takiego stanu pozwala na łatwe przefiltrowanie produktów w magazynie, które wkrótce się skończą. Daje to właścicielowi możliwość, na wcześniejsze zamówienie brakujących produktów, aby zapewnić nieprzerwane działanie przedsiębiorstwa. Użytkownicy o podniesionych uprawnieniach w stosunku do pracowników, mogą także filtrować listy magazynowe. Za pomocą jednego przycisku mogą przełączać widok z pełną lista magazynową, na listę produktów których stan magazynowy przekroczył dopuszczalną wartość krytyczną.



Rysunek : Widok listy magazynowej z poziomu kierownika/właściciela.

1. **Część implementacyjna**

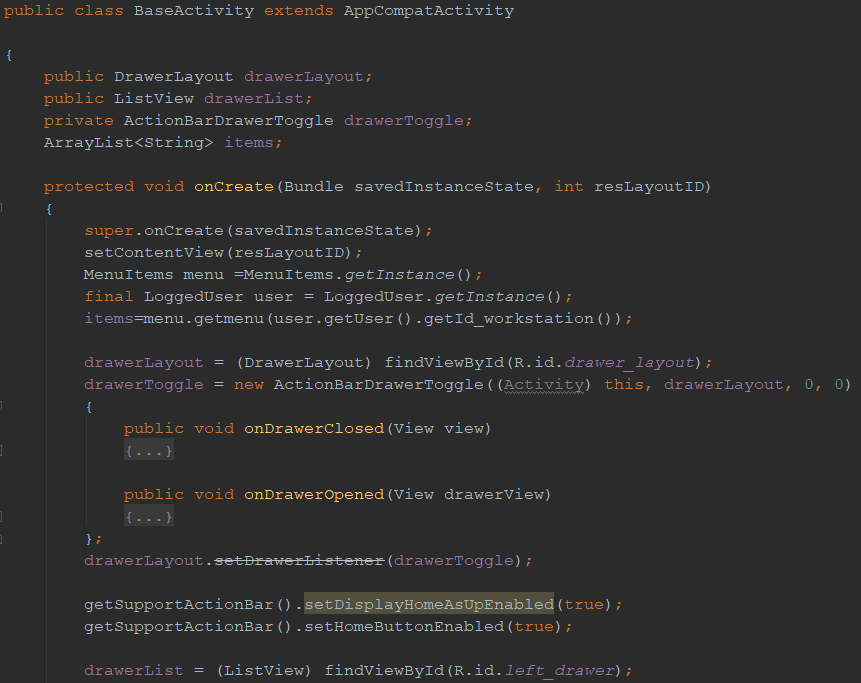
Aplikacja mobilna MMA-Mobile Management Assistant została zaimplementowana

zgodnie z przedstawionym wcześniej projektem aplikacji. Do zaimplementowania programu zostało użyte środowisko Android studio, oraz języki programowania Java, oraz SQL. Obiektowy język Java został wykorzystany do zaimplementowania aplikacji mobilnej na system Android. Za pomocą języka SQL została utworzona baza danych, pod kontrolą systemu MySQL, oraz wszelkie „eventy” wykorzystane w aplikacji. W tym rozdziale postaram się przedstawić najciekawsze aspekty implementacji systemu.

* 1. **Widok paska nawigacji**

Pasek menu w aplikacji został utworzony poprzez zastąpienie bazowej klasy

aktywności w systemie Android, własnoręcznie napisana klasą „BaseActivity”. Dzięki takiemu zabiegowi, każda kolejno utworzona aktywność mogła dziedziczyć po utworzonej wcześniej klasie „BaseActivity”. Skutkowało to domyślnym utworzeniem menu nawigacji dla każdego, nowoutworzonego widoku, co znacznie uprościło implementację kolejnych aktywności użytkownika. Dodatkowo, każda z utworzonych aktywności musiała posiadać wcześniej zdefiniowaną bazową strukturę widoku, co pozwalało na automatyczne wstawienie listy menu, w danym widoku. Na poniższych obrazach przedstawiono część implementacji klasy BaseActivity która zastępuje klasę bazową dla wszelkich aktywności, oraz implementuje menu boczne.



Rysunek : Implementacja klasy bazowej wraz z funkcjonalnością menu.

* 1. **Zdarzenia bazodanowe**

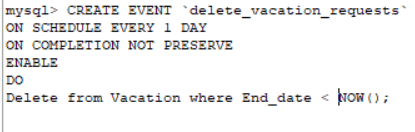
Przy pomocy zdarzeń bazodanowych, znanych szerzej jako „eventy”,

zaimplementowane zostały trzy funkcjonalności systemu. Eventy to funkcje tworzone przez programistę w języku SQL, które uruchamiane są automatycznie poprzez tzw. „event scheduler”, w określonym przez programiście czasie. Zdarzenie może być wykonywane jednokrotnie, bądź też według wskazanego harmonogramu np. każdego dnia.[8]

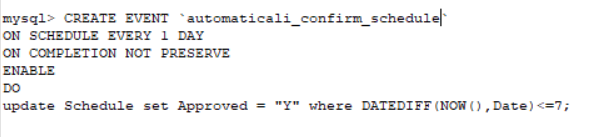
Pierwsza z funkcji odpowiada za automatyczne usuwanie wiadomości starszych, niż 30 dni. Dzięki temu unikamy przechowywania danych, które nie zostaną już wykorzystane w aplikacji. Dodatkowo zabezpieczamy aplikację przed niepotrzebnym pobieraniem do pamięci wiadomości które są już nieaktualne.

Kolejną funkcją która została wdrożona w postaci zdarzeń bazodanowych, jest automatyczne zatwierdzanie przesłanych grafików. Każdy grafik przesłany przez użytkownika może być automatycznie zatwierdzony przez bazę danych, na 7 dni przed rozpoczęciem danego dnia roboczego. Pozwala to na ostateczne ustalenie listy grafikowej przez pracodawcę i blokuje pracownikom możliwość jej edycji. Działanie takie mogłoby być niepożądane przez pracodawcę. Ostatnią funkcją zaimplementowaną po stronie bazy danych jest „event”, automatycznie usuwający przedawnione wnioski urlopowe. Moduł taki pozwala na zmniejszenie zaśmiecenia aplikacji, oraz bazy danych. Zbędne wnioski urlopowe są usuwane, dzięki czemu nie następuje ich ładowanie do aplikacji, oraz wyświetlanie w liście aktualnych wniosków urlopowych.

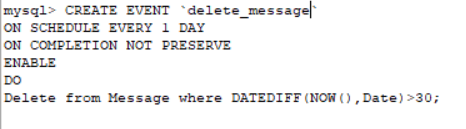
Wykorzystanie zdarzeń pozwoliło na łatwe zaimplementowanie wymaganych funkcji. Niestety zaimplementowanie takich funkcjonalności, z poziomu aplikacji było trudne, a także nieefektywne. Nie można było jasno określić które z kont miałoby takie operacje wywoływać, a także jak ustalić harmonogram aktualizacji bazy danych. Żadne z kont użytkowników nie dawało gwarancji codziennego logowania, a co za tym idzie, niemożliwe było codzienne uruchamianie określonych funkcjonalności. Wbudowany w system MySQL „event scheduler” doskonale nadawał się do tej roli, ze względu na swoją niezależność od aplikacji mobilnej, a także nieprzerwane działanie. Na poniższych diagramach przedstawione zostały skrypty pozwalające na utworzenie „eventów” obsługujących wymienione powyżej funkcjonalności.



Rysunek : Skrypt tworzący zdarzenie usuwania przedawnionych wniosków urlopowych



Rysunek : Skrypt tworzący zdarzenie automatycznego zatwierdzania grafików.



Rysunek : Skrypt tworzący zdarzenie usuwania wiadomości starszych niż 30 dni.

* 1. **Obiekty tabel w aplikacji**

Istotnym punktem w implementacji całego systemu było przygotowanie warstwy

modelu danych w taki sposób aby była ona przejrzysta i łatwa w zarządzaniu. Dzięki zastosowaniu frameworku OrmLite, warstwa modelu danych mogła zostać zaimplementowana w przejrzysty sposób, z wykorzystaniem jedynie annotacji języka Java, które odpowiednio wskazywały na nazwę tabeli oraz wszelkich kolumn zawartych w danej tabeli. Dzięki zastosowaniu prostych funkcji pobierających, oraz zapisujących informacje do atrybutów klasy (zwanych potocznie „getterami” oraz „setterami”) możliwe było łatwe zarządzanie danymi zawartymi wewnątrz instancji danej klasy. Na poniższym schemacie przedstawiono przykładową implementację takiej klasy.

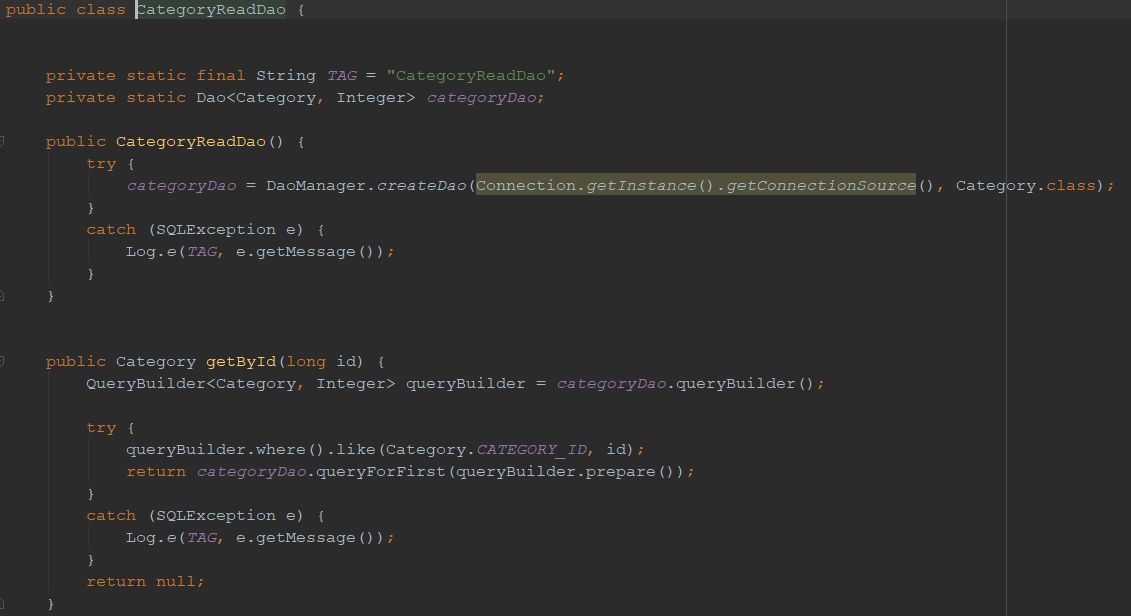


Rysunek : Przykładowa implementacja klasy tabel.

* 1. **Sposób dostępu do bazy danych**

Ostatnim przedstawionym elementem implementacji systemu jest sposób dostępu

do bazy danych. Wykorzystany został tutaj wzorzec CQRS, który zapewnia separację pomiędzy obiektami zajmującymi się zapisem do bazy danych, a obiektami których zadaniem jest odczyt. Dzięki takiej implementacji warstwy dostępu do danych, kolejne operacje mogły być wykonywane w prosty sposób. Unikamy tutaj sytuacji, w której metody przyjmują niedeterministyczne postaci. Nie można w takiej sytuacji jednoznacznie ustalić, czy funkcje zajmują się obecnie zapisem informacji podanych przez użytkownika, czy może właśnie pobierają informacje z bazy danych z zamiarem ich zwrócenia do aplikacji. Dodatkowe rozdzielenie metod zapisujących, od odczytujących na różne klasy, pozwala na uzyskanie większej przejrzystości, a w przypadku rozbudowy systemu, pozwoli programiście na szybkie zrozumienie napisanego wcześniej kodu. Na poniższym zrzucie ekranu pokazany został przykład klasy implementującej funkcje odczytywania danych z bazy danych. Na ekranie dobrze widoczny jest konstruktor klasy w którym tworzone jest połączenie ze zdalną bazą danych, oraz jedna z funkcji pobierająca dane bezpośrednio z tabeli. W pokazanym przykładzie jest to funkcja pobierająca dane z tabeli „Category”, która wyszukuje wiersz o identyfikatorze kategorii podanym przez użytkownika.



Rysunek : Klasa odpowiedzialna za pobieranie danych z tabeli "Category"

1. **Podsumowanie**
   1. **Wnioski**

Aplikacja MMA-Mobile Management Assistant została poprawnie zaprojektowana

oraz zaimplementowana. Tworzenie aplikacji mobilnej opartej na zewnętrznej bazie danych było dla mnie ciekawym doświadczeniem, oraz sporym wyzwaniem. Problemem było tutaj znalezienie takiego rozwiązania które pozwoli na najszybsze ustanowienie połączenia ze zdalną bazą danych, oraz zapewni mapowanie obiektowo relacyjne. Dodatkowo starałem się znaleźć takie rozwiązanie które będzie stabilne, oraz samo w sobie nie będzie większe niż całość mojej aplikacji. Takim rozwiązaniem okazał się Framework OrmLite, który zapewnia wszelkie z wymienionych przeze mnie funkcjonalności.

Bardzo ważnym elementem pracy było rzetelne zaprojektowanie aplikacji. Dzięki temu już od początku projektu wiedziałem jak aplikacja powinna wyglądać, jakie wymagania powinna spełniać oraz jakie ograniczenia należy na nią nałożyć. Dzięki dobrze zaprojektowanej aplikacji od strony teoretycznej, późniejsza implementacja aplikacji okazała się zadaniem o wiele prostszym. Pomogło to również w dobraniu odpowiednich technologii, dzięki czemu nie możliwa była sytuacja, w której w trakcie implementacji aplikacji okazuje się, że wykorzystanie danej technologii jest całkowicie bezsensowne, lub wręcz niemożliwe.

Oczywiście w trakcie projektowania systemu pojawiły się pewne problemy. Okazało się bowiem, że moje założenie o działaniu aplikacji bez dostępu do sieci Internet nie jest możliwe do zrealizowania. W początkowej fazie projektu założeniem było, aby aplikacja wymagała dostępu do Internetu jedynie podczas jej uruchomienia, w celu pobrania zdalnej bazy danych bezpośrednio do pamięci aplikacji, a następnie ich zapisu po skończonej pracy użytkownika z aplikacją. Nie było to jednak możliwe ze względu na wieloużytkownikowy charakter aplikacji. Takie podejście do połączenia z bazą danych zwiększyłoby znacznie ilość wykorzystywanej przez aplikację pamięci, oraz co ważniejsze generowałoby konflikty podczas pracy wielu użytkowników na raz. Przykładem może być tutaj moduł magazynowy. Załóżmy, że dwóch użytkowników loguje się do aplikacji w tym samym czasie. Na ich urządzenia zostaje pobrana lokalna kopia bazy danych, na której przeprowadzają operacje. Każdy z nich chce z magazynu pobrać pewną ilość produktu „A”. Po zatwierdzeniu zmian obaj zamykają aplikację, a na serwer wysyłane są zmiany przeprowadzone przez obu użytkowników. Takie działanie generowałoby błąd w bazie danych, ponieważ stan magazynowy produktu A byłby zaburzony.

Z tego względu zdecydowałem się, na bezpośrednie połączenie do zdalnej bazy danych z poziomu aplikacji mobilnej. Niestety niesie to za sobą konsekwencje, w postaci wymagania nieprzerwanego dostępu do sieci Internet, w trakcie działania programu. Jest to jednak o wiele lepsze rozwiązanie niż początkowy zamysł projektowy, ze względu na powszechny dostęp do sieci Internet.

* 1. **Propozycje rozwoju projektu**

Uważam, że zaimplementowana przeze mnie aplikacja ma bardzo duży potencjał,

który zamierzam wykorzystać. Aplikacja może zostać rozbudowana na kilka różnych sposobów. Postaram się pokrótce przedstawić kilka z nich.

Ze względu na modułową budowę aplikacji, mogą być do niej dołączane nowe moduły, w zależności od potrzeb pracodawcy/właściciela danego przedsiębiorstwa. Mógłby to być na przykład moduł pozwalający na automatyczne zapisywanie czasu pracy pracowników. Taka funkcjonalność jest możliwa przez zastosowanie kodów QR. Każdy z pracowników po przyjściu do pracy skanowałby swój indywidualny kod QR, a aplikacja dodawałaby do bazy danych informację o godzinie rozpoczęcia pracy. Po wyjściu z pracy ten sam kod służyłby do automatycznego zakończenia czasu pracy, oraz podliczenia godzin spędzonych w pracy. Ostatniego dnia miesiąca mógłby być generowany raport dla właściciela, z danymi przedstawiającymi sumaryczny czas pracy dla poszczególnych pracowników, w danym miesiącu.

Modułowa budowa aplikacji pozwala także na rozwój jednego z już dostępnych modułów, bez przerw w działaniu innych modułów. Przykładem może być tutaj rozwój modułu urlopowego. W przyszłości system taki mógłby automatycznie sprawdzać złożone już wnioski urlopowe oraz porównywać je z dostępnościami pracowników na dany okres. Dzięki temu możliwe byłoby uzyskanie od systemu podpowiedzi, czy taki wniosek może być zaakceptowany, a liczba pracowników w danym dniu będzie wystarczająca, czy też nie jest to możliwe, ze względu na zbyt małą obsadę kadry na dany dzień roboczy.

Oczywiście każdy z modułów może być dopasowany, oraz rozwijany bezpośrednio do potrzeb danego przedsiębiorstwa. Dzięki temu aplikacja będzie mogła być spersonalizowana, co mogłoby wiązać się z jej lepszym odbiorem na rynku aplikacji dla małych przedsiębiorstw.

Kolejną propozycją rozwoju projektu jest utworzenie oddzielnego oprogramowania, na systemy desktopowe, które jednak współpracowałoby z aplikacją MMA-Mobile Management Assistant. Rdzeniem spajającym obie aplikacje byłaby oczywiście zdalna baza danych. Propozycją takiego programu jest aplikacja magazynowa. Dzięki takiej aplikacji, możliwe byłoby powiększenie funkcjonalności związanych z magazynem. Taki system mógłby automatycznie wysyłać przypomnienia z produktami które się kończą, zapewniać automatyczne generowanie raportów, oraz pozwolić na łatwiejsze oraz wygodniejsze zarządzanie całym magazynem. Oczywiście lista funkcji jakie mógłby zapewniać taki system jest bardzo duża, i nie jest możliwe ich wymienienie bez wcześniejszego wstępnego projektu systemu.

1. **Spis rysunków**

[Rysunek 1: Przykładowy zrzut ekranu programu ePlanist 4](#_Toc503174012)

[Rysunek 2: Przykładowy zrzut ekranu systemu LoMag. 5](#_Toc503174013)

[Rysunek 3: Udział mobilnych systemów operacyjnych na świecie. 6](#_Toc503174014)

[Rysunek 4: Schemat działania systemu Git. 8](#_Toc503174015)

[Rysunek 5: Struktura wzorca Singleton 9](#_Toc503174016)

[Rysunek 6: Sposób wykorzystania DAO w aplikacji. 10](#_Toc503174017)

[Rysunek 7: Schemat wykorzystania wzorca CQRS. 11](#_Toc503174018)

[Rysunek 8: Schemat działania Framework OrmLite. 13](#_Toc503174019)

[Rysunek 9: Diagram przypadków użycia dla pracownika. 15](#_Toc503174020)

[Rysunek 10: Diagram przypadków użycia dla kierownika. 15](#_Toc503174021)

[Rysunek 11: Diagram przypadków użytkownika dla właściciela. 16](#_Toc503174022)

[Rysunek 12: Schemat architektury systemu. 20](#_Toc503174023)

[Rysunek 13: Diagram ERD bazy danych 22](#_Toc503174024)

[Rysunek 14: Przykładowa funkcjonalność modułu użytkownika (dodawanie nowych pracowników) 23](#_Toc503174025)

[Rysunek 15: Lista złożonych dostępności przez pracowników na dany dzień. 25](#_Toc503174026)

[Rysunek 16: Lista wniosków urlopowych (widok z poziomu kierownika) 26](#_Toc503174027)

[Rysunek 17: Widok listy wiadomości ekranu startowego (po lewej), oraz szczegółów wiadomości (po prawej). 27](#_Toc503174028)

[Rysunek 18: Widok listy magazynowej z poziomu kierownika/właściciela. 29](#_Toc503174029)

[Rysunek 19: Implementacja klasy bazowej wraz z funkcjonalnością menu. 30](#_Toc503174030)

[Rysunek 20: Skrypt tworzący zdarzenie usuwania przedawnionych wniosków urlopowych 31](#_Toc503174031)

[Rysunek 21: Skrypt tworzący zdarzenie automatycznego zatwierdzania grafików. 31](#_Toc503174032)

[Rysunek 22: Skrypt tworzący zdarzenie usuwania wiadomości starszych niż 30 dni. 32](#_Toc503174033)

[Rysunek 23: Przykładowa implementacja klasy tabel. 32](#_Toc503174034)

[Rysunek 24: Klasa odpowiedzialna za pobieranie danych z tabeli "Category" 33](#_Toc503174035)

1. **Bibliografia**

[1]Cay S. Horstmann, Gary Cornell „Java Podstawy wydanie IX”,

HELION, Gliwice 2013

[2]Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides „Wzorce projektowe.

Elementy oprogramowania obiektowego wielokrotnego użytku”,

HELION, Gliwice 2010

[3]Charlie Coliins, Michael Galpin, Matthias Kaeppler, „Android w praktyce,

HELION, Gliwice 2012

[4]Ian F. Darwin, Android. Receptury, Gliwice, HELION, 2013,

ISBN: 978-83-246-6269-2

[5]Scott Chacon, „Pro Git”, Apress, New York, 2009

[6]Sasha Pachew, „MySQL. Mechanizmy wewnętrzne bazy danych”,

HELION, Gliwice, 2007

[7]Zapata Cruz Belen, „Android Studio. Podręcznik użytkownika”,

HELION, Gliwice, 2015

[8]Paul DuBois, „MySQL. Vademecum profesjonalisty”, Gliwice, HELION, 2014