Spis treści

[[**DOKUMENTACJA**](#_Toc93416694) **APLIKACJI QRSCANNER 1**](#_Toc93416695)

[1. Wprowadzenie 1](#_Toc93416696)

[2. Funkcje aplikacji 1](#_Toc93416697)

[3. Instrukcja użytkownika 2](#_Toc93416697)

[4. Architektura aplikacji 3](#_Toc93416697)

[5. Diagramy UML 4](#_Toc93416697)

[6. Opisanie kodu aplikacji 6](#_Toc93416697)

[7. Instalacja i konfiguracja 20](#_Toc93416697)

**Dokumentacja aplikacji QRScanner**

**Wprowadzenie** Moim projektem jest stworzona przeze mnie aplikacja do skanowania kodów QR i zarządzania stanem magazynowym w sklepie.

QRScanner to aplikacja umożliwiająca skanowanie kodów QR za pomocą aparatu w urządzeniu mobilnym. Aplikacja QRScanner została napisana w języku Java z użyciem platformy Android i jest przeznaczona do uruchamiania na urządzeniach mobilnych z systemem Android.  
  
**Funkcje Aplikacji**

1. **Skanowanie kodów QR** - Aplikacja umożliwia skanowanie kodów QR za pomocą aparatu w urządzeniu mobilnym. Po zeskanowaniu kodu QR, aplikacja odczytuje jego zawartość i wyświetla użytkownikowi odpowiednie informacje lub podejmuje odpowiednie działania, na podstawie typu zawartości kodu QR.
2. **Zarządzanie kodami QR**- Aplikacja umożliwia użytkownikowi przypisanie konkretnej nazwy produktu dla konkretnego kodu QR, edytowania tej nazwa oraz usuwania. Użytkownik może przeglądać listę swoich kodów QR.
3. **Zarządzanie magazynem** – Użytkownik po zeskanowaniu kodu QR dodanego do swojej bazy, może wybrać dla partii towaru datę ważności i umieścić ją w magazynie. Użytkownik może przeglądać swój magazyn, wyszukiwać produkty po nazwie, dacie oraz zakresie dat. Użytkownik może kasować towary z magazynu.

**Instrukcja użytkownika:**

1. Użytkownik klika przycisk skanuj aby przejść do ekranu skanowania.
2. Użytkownik skanuje kod kreskowy. Można włączyć aparat przyciskiem ściszania telefonu dla lepszej widoczności.
3. Użytkownik może również wpisać kod kreskowy ręcznie w pole tekstowe, które automatycznie reaguje na zawarte w nim dane.
4. Gdy użytkownik zeskanuje kod/wpisze go ręcznie aplikacja zwróci użytkownikowi informacje o kodzie kreskowym. Jeśli jest on już dodany do bazy, wtedy użytkownik będzie mógł dodać go do magazynu, wprowadzając przy tym datę ważności. Jeśli kodu kreskowego nie ma w bazie, użytkownik zamiast tego będzie miał teraz możliwość dodania kodu do swojej bazy przypisując mu odpowiednia nazwę.
5. Gdy użytkownik podaje datę dla towaru, chcąc dodać ja do magazynu, podaje datę formie „dd-mm-rr”. Z roku cyfra „20” jest pomijana, aplikacja dodaje ją później automatycznie. Data musi być pełna i zawierać obie cyfry dla miesiąca, dnia oraz roku, nawet gdy jest to data jednocyfrowa. Dla przykładu data 4.13.2024 zapisana musi zostać jako 04.13.24.
6. Użytkownik może kliknąć przycisk edytuj kod kreskowy w celu edytowania istniejącego kodu kreskowego.
7. Użytkownik przyciskiem Kody QR może przeglądać dodane do bazy kody kreskowe oraz je kasować. Ma on możliwość wyszukania kodu po nazwie lub ręcznie szukając na liście.
8. Użytkownik przyciskiem Magazyn może zajrzeć do magazynu. W nim może wyszukać produkt po nazwie lub kliknąć 3 kropki w prawym górnym rogu w celu przełączenia na wyszukiwanie po dacie. Wtedy użytkownik ujrzy przycisk wyszukiwania po dacie. Po jego naciśnięciu pojawi się komunikat z możliwymi 2 datami do wyboru. Wpisanie tylko jednej wyszuka dla konkretnej daty. Wpisanie 2 dat wyszuka zakres pomiędzy datą z pierwszego pola tekstowego i drugiego. Mniejsza data musi znajdować się w pierwszym polu tekstowym. Użytkownik może również kasować produkty z magazynu.

**Architektura aplikacji**

Aplikacja QRScanner składa się z kilku klas Java, które zostały zorganizowane w strukturze pakietów. Oto lista niektórych z głównych klas użytych w aplikacji:

**QRCodesActivity.java**: Klasa reprezentująca ekran główny aplikacji, gdzie użytkownik może skanować kody QR oraz przeglądać historię skanowanych kodów QR.

**MainActivity.java**: Klasa reprezentująca ekran startowy aplikacji, który jest wyświetlany po uruchomieniu aplikacji.

**MagazineActivity.java**: Klasa reprezentująca ekran wyświetlający zawartość zeskanowanego kodu QR typu "Magazine".

**CaptureAct.java**: Klasa obsługująca skanowanie kodów QR za pomocą aparatu w urządzeniu mobilnym.

**PrefConfig.java**: Klasa obsługująca zapisywanie i odczytywanie preferencji użytkownika, takich jak rodzaj wyszukiwania w magazynie.

**Product.java**: Klasa reprezentująca obiekt produktu, który może być odczytany z zeskanowanego kodu QR.

**Magazine.java**: Klasa reprezentująca obiekt magazynu, który może być odczytany z zeskanowanego kodu QR typu "Magazine".

**CardCaptionedImageMagazine.java**: Klasa reprezentująca widok karty z obrazkiem i opisem magazynu, który może być wyświetlany na ekranie wraz z informacjami o zeskanowanym kodzie QR typu "Magazine".

**CardCaptionedImageAdapterQR.java**: Klasa adaptera, która jest używana do wyświetlania listy kart z obrazkami i opisami zeskanowanych kodów QR na ekranie historii skanowanych kodów QR.

Aplikacja QRScanner jest napisana w języku Java i korzysta z frameworka Android do tworzenia aplikacji mobilnej. Architektura aplikacji opiera się na wzorcu architektonicznym Model-View-Presenter (MVP), gdzie QRCodesActivity pełni rolę widoku (View), CaptureAct jest modelem (Model), a MainActivity jest prezentatorem (Presenter). Pozostałe klasy, takie jak Product, Magazine, CardCaptionedImageMagazine, CardCaptionedImageAdapterQR i PrefConfig, są używane do przechowywania i zarządzania danymi w aplikacji.

**Diagramy UML**

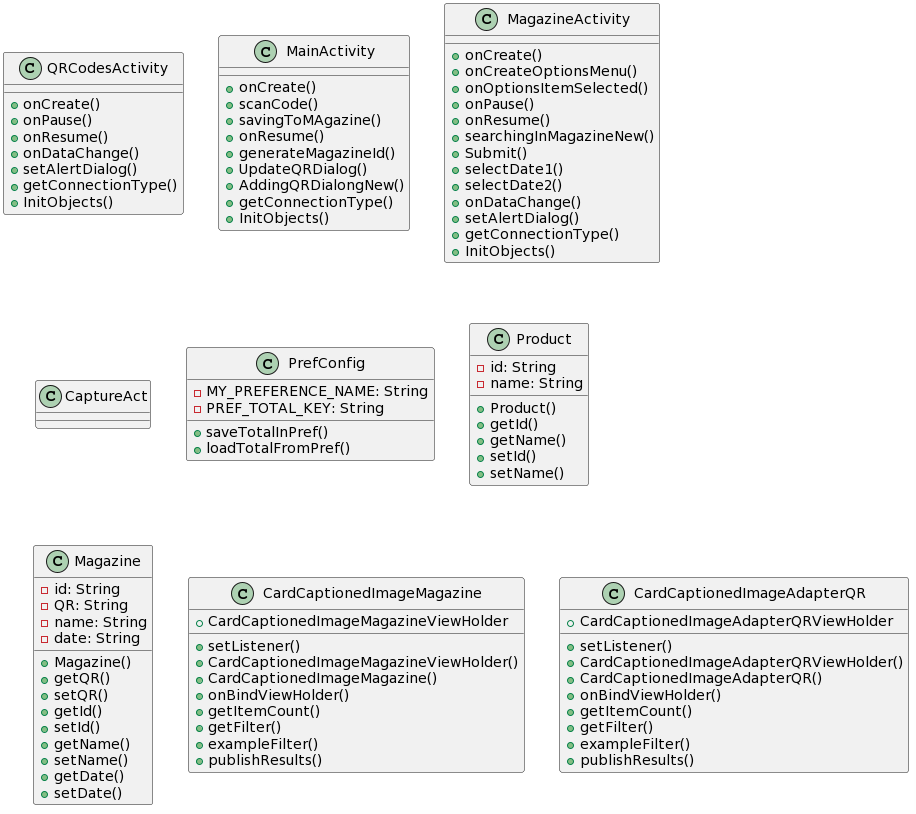
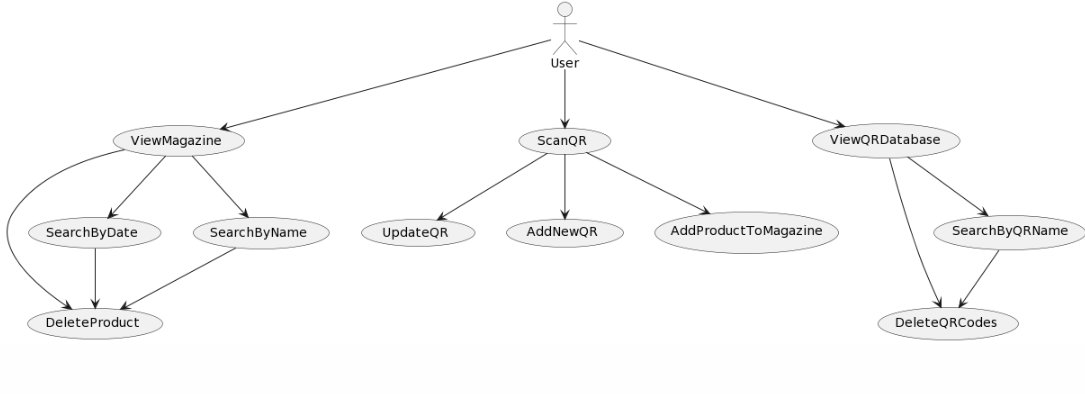
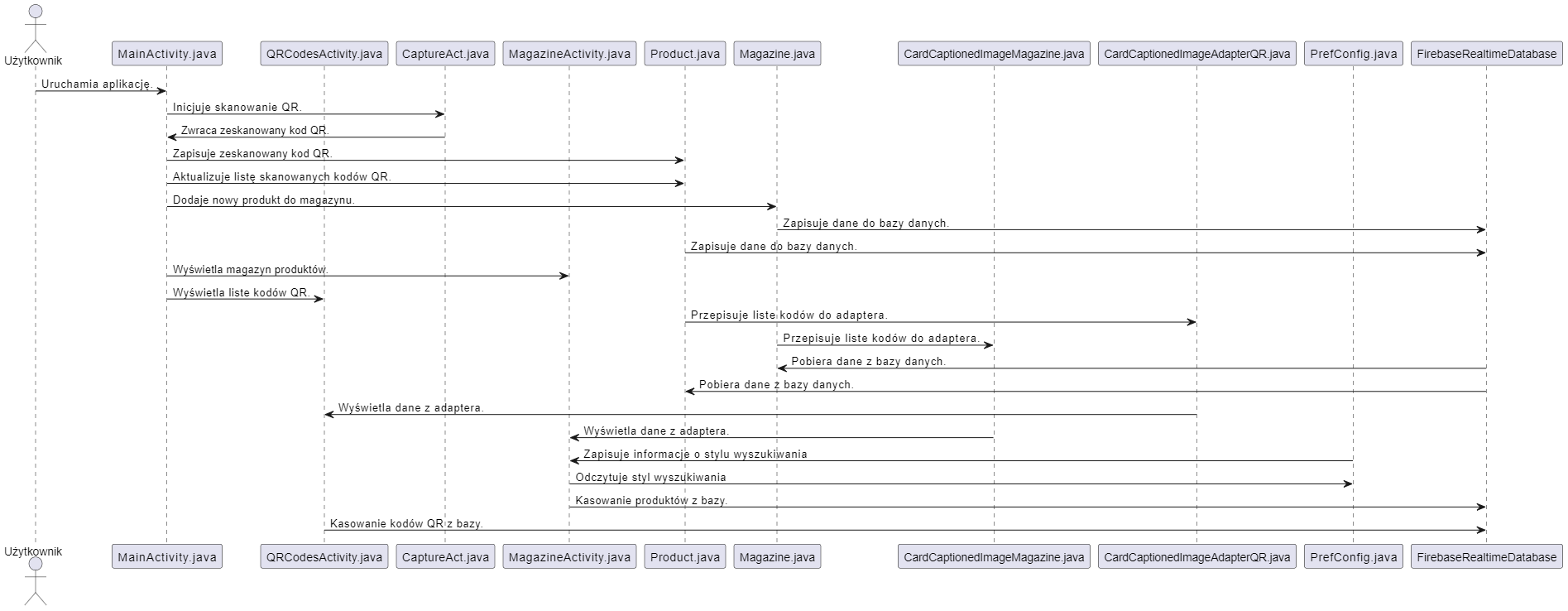
Diagram klas:

Diagram przypadków użycia



Diagram sekwencyjny

Diagramy zaprogramowano w kodzie plantuml.

**Opisanie kodu**   
**CardCaptionedImageMagazine.java** - jest adapterem dla RecyclerView w aplikacji Android. Jest używana do wyświetlania listy elementów typu Magazine w formie kart z obrazkami i podpisami. Klasa implementuje interfejs Filterable, co umożliwia filtrowanie listy elementów na podstawie określonych kryteriów.

Pola klasy:

exampleList: Lista elementów typu Magazine, które będą wyświetlane w RecyclerView.

exampleListFull: Kopia oryginalnej listy exampleList, używana do przechowywania pełnej listy przed zastosowaniem filtrów.

listener: Obiekt implementujący interfejs Listener, który jest wywoływany po kliknięciu na element listy.

scaleUp, scaleDown: Obiekty animacji do skalowania elementów.

clickAlert: Obiekt MediaPlayer, używany do odtwarzania dźwięku po kliknięciu na element.

filterInt: Zmienna przechowująca informację o wybranym rodzaju filtru.

context: Kontekst aplikacji.

date1, date2, date3: Obiekty Date, używane do przechowywania dat w procesie filtrowania.

cal1, cal2, cal3: Obiekty Calendar, używane do porównywania dat w procesie filtrowania.

dateMin, dateMax: Stringi przechowujące daty w formacie "yyyy-MM-dd" na podstawie wartości constraint w procesie filtrowania.

Metody klasy:

CardCaptionedImageMagazineViewHolder: Wewnętrzna klasa ViewHolder, używana do przechowywania widoków elementów listy.

CardCaptionedImageMagazine: Konstruktor klasy, przyjmuje listę elementów typu Magazine i inicjalizuje pola exampleList i exampleListFull.

onCreateViewHolder: Metoda dziedziczona z RecyclerView.Adapter, odpowiedzialna za tworzenie i zwracanie nowego obiektu CardCaptionedImageMagazineViewHolder.

onBindViewHolder: Metoda dziedziczona z RecyclerView.Adapter, odpowiedzialna za przypisanie danych z obiektu Magazine do widoku ViewHoldera oraz obsługę kliknięcia na element.

getItemCount: Metoda dziedziczona z RecyclerView.Adapter, zwraca liczbę elementów w liście exampleList.

getFilter: Metoda dziedziczona z Filterable, zwraca obiekt filtru, który jest używany do filtrowania listy elementów.

exampleFilter: Obiekt klasy Filter, implementujący filtrowanie na podstawie wybranego rodzaju filtru. Filtruje listę elementów na podstawie nazwy lub daty elementów typu Magazine.

performFiltering: Metoda dziedziczona z Filter, implementująca logikę filtrowania. Tworzy nową listę zawierającą tylko elementy spełniające wybrane kryteria filtru.

publishResults: Metoda dziedziczona z Filter, aktualizuje listę exampleList na podstawie wyników filtrowania i powiadamia adapter o zmianach w danych.  
  
**CardCaptionedImageAdapterQR.java** - jest adapterem dla RecyclerView w aplikacji Android. Jest używana do wyświetlania listy elementów typu Product w formie kart z obrazkami i podpisami. Klasa implementuje interfejs Filterable, co umożliwia filtrowanie listy elementów na podstawie określonych kryteriów.

Pola klasy:

exampleList: Lista elementów typu Product, które będą wyświetlane w RecyclerView.

exampleListFull: Kopia oryginalnej listy exampleList, używana do przechowywania pełnej listy przed zastosowaniem filtrów.

listener: Obiekt implementujący interfejs Listener, który jest wywoływany po kliknięciu na element listy.

scaleUp, scaleDown: Obiekty animacji do skalowania elementów.

clickAlert: Obiekt MediaPlayer, używany do odtwarzania dźwięku po kliknięciu na element.

Metody klasy:

CardCaptionedImageAdapterQRViewHolder: Wewnętrzna klasa ViewHolder, używana do przechowywania widoków elementów listy.

CardCaptionedImageAdapterQR: Konstruktor klasy, przyjmuje listę elementów typu Product i inicjalizuje pola exampleList i exampleListFull.

onCreateViewHolder: Metoda dziedziczona z RecyclerView.Adapter, odpowiedzialna za tworzenie i zwracanie nowego obiektu CardCaptionedImageAdapterQRViewHolder.

onBindViewHolder: Metoda dziedziczona z RecyclerView.Adapter, odpowiedzialna za przypisanie danych z obiektu Product do widoku ViewHoldera oraz obsługę kliknięcia na element.

getItemCount: Metoda dziedziczona z RecyclerView.Adapter, zwraca liczbę elementów w liście exampleList.

getFilter: Metoda dziedziczona z Filterable, zwraca obiekt filtru, który jest używany do filtrowania listy elementów.

exampleFilter: Obiekt klasy Filter, implementujący filtrowanie na podstawie nazwy z obiektu Product.

performFiltering: Metoda dziedziczona z Filter, implementująca logikę filtrowania. Tworzy nową listę zawierającą tylko elementy spełniające kryteria filtru.

publishResults: Metoda dziedziczona z Filter, aktualizuje listę exampleList na podstawie wyników filtrowania i powiadamia adapter o zmianach w danych.

**Magazine.java** - Klasa Magazine jest klasą reprezentującą magazyn w systemie. Zawiera pola takie jak id, QR, name oraz date, które określają identyfikator magazynu, kod QR, nazwę oraz datę magazynu.

Konstruktory:

Magazine(String id, String QR, String name, String date): Konstruktor parametryczny, który tworzy obiekt klasy Magazine z przekazanymi wartościami id, QR, name oraz date.

Magazine(): Konstruktor domyślny, który tworzy pusty obiekt klasy Magazine.

Metody:

getQR(): Metoda zwracająca kod QR magazynu.

setQR(String QR): Metoda ustawiająca kod QR magazynu na podstawie przekazanego parametru QR.

getId(): Metoda zwracająca identyfikator magazynu.

setId(String id): Metoda ustawiająca identyfikator magazynu na podstawie przekazanego parametru id.

getName(): Metoda zwracająca nazwę magazynu.

setName(String name): Metoda ustawiająca nazwę magazynu na podstawie przekazanego parametru name.

getDate(): Metoda zwracająca datę magazynu.

setDate(String date): Metoda ustawiająca datę magazynu na podstawie przekazanego parametru date.

**Product.java** - Klasa Product jest klasą reprezentującą produkt w systemie. Zawiera pola takie jak id oraz name, które określają identyfikator produktu oraz jego nazwę.

Konstruktory:

Product(String id, String name): Konstruktor parametryczny, który tworzy obiekt klasy Product z przekazanymi wartościami id oraz name.

Product(): Konstruktor domyślny, który tworzy pusty obiekt klasy Product.

Metody:

getId(): Metoda zwracająca identyfikator produktu.

getName(): Metoda zwracająca nazwę produktu.

setId(String id): Metoda ustawiająca identyfikator produktu na podstawie przekazanego parametru id.

setName(String name): Metoda ustawiająca nazwę produktu na podstawie przekazanego parametru name.

**PrefConfig.java** - Klasa PrefConfig jest klasą pomocniczą do obsługi preferencji w systemie. Zawiera metody do zapisywania i odczytywania danych z preferencji.

Pola:

MY\_PREFERENCE\_NAME: Prywatne, statyczne pole typu String, które przechowuje nazwę preferencji.

PREF\_TOTAL\_KEY: Publiczne, statyczne pole typu String, które przechowuje klucz do zapisywania/odczytywania wartości z preferencji.

Metody:

saveTotalInPref(Context context, int total): Publiczna, statyczna metoda do zapisywania wartości total do preferencji. Przyjmuje jako parametry kontekst Context oraz wartość całkowitą total, które zostają zapisane w preferencji.

loadTotalFromPref(Context context): Publiczna, statyczna metoda do odczytywania wartości total z preferencji. Przyjmuje jako parametr kontekst Context i zwraca odczytaną wartość całkowitą.

**Capture.java** - Klasa CaptureAct jest klasą dziedziczącą po klasie CaptureActivity i reprezentuje aktywność do przechwytywania obrazów za pomocą kamery w celu odczytania kodów QR lub kodów kreskowych.

Dziedziczenie:

CaptureAct dziedziczy po klasie CaptureActivity.

Metody:

Klasa CaptureAct dziedziczy metody z klasy CaptureActivity. Nie zawiera dodatkowych metod.

**QRCodesActivity.java** - jest klasą w języku Java, która rozszerza klasę AppCompatActivity i implementuje interfejs ValueEventListener. Jest to aktywność w aplikacji, która obsługuje wyświetlanie dodanych kodów QR dla produktów.

Dziedziczenie:

AppCompatActivity: Jest to klasa bazowa dostarczana przez bibliotekę Android Support, która dostarcza funkcje związane z obsługą aktywności w systemie Android.

ValueEventListener: Jest to interfejs dostarczany przez bibliotekę Firebase, który pozwala na nasłuchiwanie na zdarzenia zmiany wartości w bazie danych Firebase.

Pola:

searchView: Pole typu SearchView, które reprezentuje widok wyszukiwania w aktywności.

progress: Pole typu ProgressBar, które reprezentuje pasek postępu w aktywności.

id: Pole typu String, które przechowuje aktualnie wybrany identyfikator produktu.

name: Pole typu String, które przechowuje aktualnie wybraną nazwę produktu.

productList: Pole typu List<Product>, które przechowuje listę produktów pobranych z bazy danych.

recyclerView: Pole typu RecyclerView, które reprezentuje widok listy produktów w aktywności.

firebaseDatabase: Pole typu FirebaseDatabase, które reprezentuje instancję bazy danych Firebase.

loadingText: Pole typu TextView, które reprezentuje tekst informujący o ładowaniu danych w aktywności.

emptyList: Pole typu TextView, które reprezentuje tekst informujący o pustej liście produktów w aktywności.

toolbar: Pole typu Toolbar, które reprezentuje pasek narzędziowy w aktywności.

loadingDialog: Pole typu AlertDialog, które reprezentuje okno dialogowe wyświetlane podczas ładowania danych w aktywności.

clickAlert: Pole typu MediaPlayer, które reprezentuje dźwięk odtwarzany po kliknięciu w aktywności.  
  
Metody:

onCreate(Bundle savedInstanceState): Metoda ta jest wywoływana podczas tworzenia aktywności. Inicjalizuje interfejs użytkownika, takie jak ProgressBar, RecyclerView, SearchView, i Toolbar. Ustawia także nasłuchiwanie na bazie danych Firebase.

onPause(): Metoda ta jest wywoływana, kiedy aktywność jest pauzowana. Usuwa nasłuchiwanie z bazy danych Firebase.

onResume(): Metoda ta jest wywoływana, kiedy aktywność jest wznowiona po pauzie. Dodaje nasłuchiwanie do bazy danych Firebase.

onDataChange(DataSnapshot snapshot): Metoda ta jest wywoływana, kiedy dane w bazie danych Firebase ulegają zmianie. Aktualizuje listę produktów na podstawie otrzymanych danych i aktualizuje interfejs użytkownika.

onCancelled(DatabaseError error): Metoda ta jest wywoływana, kiedy odczyt danych z bazy danych Firebase zostaje przerwany z powodu błędu.

setAlertDialog(String IdText, String NameText): Metoda ta tworzy i wyświetla dialog z pytaniem o potwierdzenie usunięcia produktu z bazy danych Firebase.

getConnectionType(Context context): Metoda ta zwraca typ połączenia sieciowego na podstawie aktualnego połączenia urządzenia.

**MagazineActivity.java** – jest klasą Java, która dziedziczy po klasie AppCompatActivity i implementuje interfejs ValueEventListener. Jest to aktywność w aplikacji mobilnej, która jest odpowiedzialna za wyświetlanie listy produktów w magazynie oraz obsługę wyszukiwania magazynów na podstawie nazwy lub daty.

Dziedziczenie

AppCompatActivity: Jest to klasa bazowa dostarczana przez bibliotekę Android Support, która dostarcza funkcje związane z obsługą aktywności w systemie Android.

ValueEventListener: Jest to interfejs dostarczany przez bibliotekę Firebase, który pozwala na nasłuchiwanie na zdarzenia zmiany wartości w bazie danych Firebase.

Pola:

searchView: Obiekt typu SearchView, który reprezentuje pole do wprowadzania zapytania filtrowania w widoku listy magazynów.

progress: Obiekt typu ProgressBar, który reprezentuje wskaźnik postępu podczas ładowania danych z bazy danych.

id: Zmienna typu String, przechowująca identyfikator zalogowanego użytkownika.

name: Zmienna typu String, przechowująca nazwę zalogowanego użytkownika.

magazineId: Zmienna typu String, przechowująca identyfikator wybranego magazynu.

date: Zmienna typu String, przechowująca datę wybranego magazynu.

magazineList: Lista obiektów typu Magazine, przechowująca listę magazynów pobranych z bazy danych Firebase.

recyclerView: Obiekt typu RecyclerView, który reprezentuje widok listy magazynów.

firebaseDatabase: Obiekt typu FirebaseDatabase, który jest używany do komunikacji z bazą danych Firebase.

loadingText: Obiekt typu TextView, który reprezentuje tekst wyświetlany podczas ładowania danych z bazy danych.

emptyList: Obiekt typu TextView, który reprezentuje tekst wyświetlany w przypadku pustej listy magazynów.

toolbar: Obiekt typu Toolbar, który reprezentuje pasek narzędziowy aktywności.

searchingType: Zmienna typu int, przechowująca typ wyszukiwania, czyli datę początkową, końcową lub brak wyszukiwania.

searchingButton: Obiekt typu Button, który reprezentuje przycisk do aktywowania wyszukiwania na podstawie daty.

s: Obiekt typu AnimationSet, który reprezentuje zestaw animacji używanych podczas ładowania danych.

scaleUp: Obiekt typu Animation, który reprezentuje animację powiększania elementu.

scaleDown: Obiekt typu Animation, który reprezentuje animację pomniejszania elementu.

calendarView: Obiekt typu CalendarView, który reprezentuje widok kalendarza używany do wyboru daty początkowej i końcowej w dialogu wyszukiwania.

adapter: Obiekt typu CardCaptionedImageMagazine, który jest adapterem listy magazynów.

searchingDate: Zmienna typu String, przechowująca zapytanie filtrowania na podstawie wybranych dat w dialogu wyszukiwania.

buttonDate1: Obiekt typu Button, który reprezentuje przycisk wyboru daty początkowej w dialogu wyszukiwania.

buttonDate2: Obiekt typu Button, który reprezentuje przycisk wybór daty końcowej w dialogu wyszukiwania.

dialog: Obiekt typu AlertDialog, który reprezentuje dialog wyszukiwania z datami początkową i końcową.

loadingDialog: Obiekt typu AlertDialog, który reprezentuje dialog ładowania danych z bazy danych.

clickAlert: Obiekt typu MediaPlayer, który jest używany do odtwarzania dźwięku kliknięcia przy akcji związanej z dialogiem.

Metody:

onCreate(Bundle savedInstanceState): Jest to metoda cyklu życia Activity, która jest wywoływana przy tworzeniu (inicjalizacji) Activity. W tej metodzie inicjalizowane są różne elementy interfejsu użytkownika, takie jak widoki, animacje, pasek narzędzi, obiekt MediaPlayer, oraz ustawienia początkowe dla niektórych zmiennych. Wykorzystywane są również metody z klasy PrefConfig do odczytywania i zapisywania ustawień preferencji. Metoda ta ma jedną opcjonalną argument savedInstanceState, który przechowuje stan Activity w przypadku jego odtworzenia po ewentualnym zniszczeniu, np. w wyniku zmiany orientacji ekranu.

onCreateOptionsMenu(Menu menu): Jest to metoda, która jest wywoływana podczas tworzenia menu akcji w pasku narzędzi Activity. Metoda ta jest odpowiedzialna za nadmuchanie menu z zasobów XML do obiektu typu Menu, a następnie zwraca wartość true w celu pokazania menu na ekranie.

onOptionsItemSelected(MenuItem item): Jest to metoda, która jest wywoływana w odpowiedzi na wybranie jednego z elementów menu akcji w pasku narzędzi Activity. Metoda ta jest odpowiedzialna za obsługę kliknięć na poszczególne elementy menu. W zaimplementowanym kodzie obsługiwane są dwa elementy menu o identyfikatorach R.id.m\_lo i R.id.m\_assets, które reprezentują opcje wyboru rodzaju wyszukiwania (po nazwie magazynu lub po dacie). W zależności od aktualnie wybranego rodzaju wyszukiwania i stanu progress bara, wykonywane są odpowiednie akcje, takie jak zmiana ustawień preferencji, zmiana widoczności niektórych widoków, odtworzenie dźwięku kliknięcia, oraz filtrowanie listy magazynów.

onPause(): Jest to metoda cyklu życia Activity, która jest wywoływana przy pauzowaniu Activity (np. w przypadku przejścia do innego Activity lub przy zamknięciu aplikacji). W tej metodzie jest usuwane nasłuchiwanie zmian w bazie danych Firebase na gałęzi "Magazine" z wykorzystaniem metody removeEventListener(this), gdzie this odnosi się do obiektu, który implementuje interfejs ValueEventListener (czyli klasy MagazineActivity). Dzięki temu, nie będziemy otrzymywać powiadomień o zmianach w bazie danych, gdy Activity jest wstrzymane.

onResume(): Ta metoda jest wywoływana, gdy aktywność ponownie staje się widoczna dla użytkownika po zakończeniu innej aktywności. W tej metodzie ponownie dodajemy nasłuchiwacz (EventListener) do bazy danych Firebase Realtime Database za pomocą metody addValueEventListener(this). Dzięki temu, jeśli wystąpią zmiany w danych w bazie danych, zostaną one odzwierciedlone na ekranie w czasie rzeczywistym.

getConnectionType(Context context): Metoda ta zwraca typ połączenia sieciowego na podstawie aktualnego połączenia urządzenia.  
InitObjects(): Ta metoda tworzy i wyświetla okno dialogowe z paskiem postępu, informujące użytkownika o trwającym procesie łączenia z bazą danych. Okno dialogowe jest niemożliwe do zamknięcia przez użytkownika i zawiera przycisk "Odśwież aplikacje", który po kliknięciu odświeża aplikację.  
searchingInMagazineNew(): Jest to metoda, która tworzy i wyświetla okno dialogowe z funkcją wyszukiwania w magazynie. Okno dialogowe jest tworzone za pomocą klasy AlertDialog.Builder i zawiera niestandardowy widok załadowany z pliku XML alert\_magazine. Okno dialogowe zawiera przyciski do wyboru daty, przyciski "Zastosuj" i "Resetuj", oraz pole do wprowadzania tekstu do wyszukiwania.

Submit(): Jest to metoda, która jest wywoływana po kliknięciu przycisku "Zastosuj" w oknie dialogowym. Sprawdza, czy wybrane daty są poprawne, a następnie wywołuje metodę getFilter().filter() na adapterze, aby przefiltrować dane w zależności od wybranych dat.

selectDate1() i selectDate2(): Są to metody, które są wywoływane po kliknięciu przycisków wyboru daty w oknie dialogowym. Wyświetlają okno dialogowe do wyboru daty za pomocą klasy DatePickerDialog i aktualizują wybrane daty na przyciskach.

onDataChange(): Jest to metoda obsługi zmiany danych, która jest wywoływana po zmianie danych w źródle danych. Aktualizuje interfejs użytkownika na podstawie nowych danych, takich jak ukrywanie/zmiana widoczności elementów interfejsu, ustawienie nowych danych w adapterze, ustawienie słuchacza na polu wyszukiwania itp.  
setAlertDialog() - Tworzy i wyświetla okno dialogowe z pytaniem o potwierdzenie usunięcia pozycji z magazynu. Okno dialogowe zawiera informacje o nazwie, kodzie i dacie pozycji, którą użytkownik chce usunąć. Metoda posiada dwa przyciski - "Usuń" i "Anuluj". Po kliknięciu przycisku "Usuń", metoda inicjalizuje obiekty niezbędne do wyświetlenia okna dialogowego z paskiem postępu (loadingDialog) i usuwa wybraną pozycję z bazy danych Firebase. Jeśli nie ma połączenia z internetem, zostaje wyświetlony komunikat o braku łączności i metoda ponownie wywołuje się, oczekując na akcję użytkownika. Po kliknięciu przycisku "Anuluj", okno dialogowe zostaje zamknięte.  
 **MainActivity.java** - jest klasą Java, która dziedziczy po klasie AppCompatActivity. Jest to aktywność w aplikacji mobilnej, przez którą użytkownik skanuje dodaje kody kreskowy, edytuje, dodaje nowe towary do magazynu oraz uruchamia inne aktywności jak magazyn produktów i magazyn kodów QR.

Pola:

Button btn\_scan: Przycisk do skanowania kodów QR.

Button btn\_add\_QR: Przycisk do dodawania kodów QR.

Button btn\_QR\_codes: Przycisk do wyświetlania listy kodów QR.

Button btn\_Magazine: Przycisk do wyświetlania listy pozycji w magazynie.

Button btn\_addNew: Przycisk do dodawania nowych pozycji do magazynu.

Button btn\_reset: Przycisk do resetowania skanu.

EditText editTextDate: Pole tekstowe do wprowadzania daty.

String scanName: Zmienna przechowująca nazwę zeskanowanej pozycji.

String productId: Zmienna przechowująca identyfikator produktu (kod QR).

String magazineId: Zmienna przechowująca identyfikator magazynu.

String date: Zmienna przechowująca datę.

int scanOperation: Zmienna określająca rodzaj operacji skanowania.

int inDatabaseScan: Zmienna przechowująca informację o tym, czy zeskanowana pozycja jest już w bazie danych.

Toolbar toolbar: Pasek narzędzi.

AlertDialog loadingDialog: Okno dialogowe z paskiem postępu.

Animation scaleUp, scaleDown: Animacje do powiększenia i zmniejszenia elementów interfejsu użytkownika.

MediaPlayer clickAlert: Dźwięk kliknięcia.

AnimationSet s: Zestaw animacji.

AlertDialog dialog: Okno dialogowe.

FirebaseDatabase firebaseDatabase: Obiekt klasy FirebaseDatabase do obsługi bazy danych Firebase.

EditText editTextAdd: Pole tekstowe do dodawania nowych pozycji.

TextView textView\_result: Pole tekstowe do wyświetlania wyników.

ProgressBar progressBar\_auto: Pasek postępu do automatycznego ładowania.

Metody:

onCreate() - jest metodą cyklu życia, która jest wywoływana, gdy aktywność jest tworzona. Wewnątrz tej metody inicjalizowane są różne elementy interfejsu użytkownika za pomocą odpowiednich metod findViewById. Ustawiany jest również nasłuchiwacz zmian tekstu (text watcher) na elemencie editTextAdd, który nasłuchuje zmian w wprowadzonym tekście. Jeśli wprowadzony tekst istnieje, zapytuje Firebase Realtime Database, aby sprawdzić, czy wprowadzone ID produktu istnieje w bazie danych. Jeśli ID produktu istnieje, pobiera szczegóły produktu i wyświetla je w elemencie textView\_result. Następnie wywoływana jest metoda savingToMAgazine. Jeśli ID produktu nie istnieje, wyświetlany jest komunikat informujący, że produkt nie został znaleziony w bazie danych.

scanCode() – uruchamia skaner kodów oraz inicjuję AcitivityResultLauncher<>.

AcitivityResultLauncher<ScanOptions>barLaucher - Metoda ta jest obiektem klasy ActivityResultLauncher, który został zarejestrowany dla kontraktu ScanContract. Metoda ta jest odpowiedzialna za obsługę wyników skanowania kodów kreskowych za pomocą skanera kodów kreskowych w aplikacji. Po otrzymaniu wyniku, metoda sprawdza, czy otrzymany kod kreskowy istnieje w bazie danych Firebase. W zależności od wartości zmiennej scanOperation, podejmuje odpowiednie działania, takie jak wyświetlenie informacji o produkcie, zapisanie produktu w magazynie, aktualizację informacji o produkcie lub dodanie nowego produktu, w zależności od tego, czy produkt istnieje w bazie danych czy nie.

savingToMAgazine() – Metoda ta odpowiedzialna jest za wyświetlanie okna dialogowego z formularzem do dodawania przedmiotu do magazynu. Metoda ta tworzy AlertDialog, ustawia jego tytuł, styl, oraz widok za pomocą LayoutInflater. Następnie, dodaje nasłuchiwacze na przyciski "Dodaj" i "Anuluj", oraz na zmiany w polu tekstowym z datą. Po kliknięciu przycisku "Dodaj", sprawdza czy wprowadzona data jest w prawidłowym formacie, tworzy nowy obiekt Magazine z wprowadzonymi danymi i dodaje go do bazy danych Firebase. Po kliknięciu przycisku "Anuluj", zamyka okno dialogowe i czyści pola tekstowe.

generateMagazineId() – metoda ta generuje unikalny identyfikator dla nowego elementu magazynowego. Metoda ta używa aktualnej daty i czasu w formie liczby milisekund od epoki Unix (tzw. timestamp) jako podstawy do generowania identyfikatora. Następnie generuje trzy losowe dwucyfrowe liczby, dodaje je do identyfikatora, oraz łączy je ze sobą i z timestampem, tworząc finalny identyfikator w formacie "timestamp-liczba1-liczba2-liczba3". Identifikator ten jest przypisywany do zmiennej "magazineId", która jest używana później przy dodawaniu nowego elementu magazynowego do bazy danych.

UpdateQRDialog() – Metoda ta tworzy i wyświetla okno dialogowe (AlertDialog) w aktywności MainActivity w celu aktualizacji nazwy produktu na podstawie zeskanowanego kodu QR. Okno dialogowe zawiera tytuł "Wynik skanowania", treść z informacjami o nazwie produktu i kodzie produktu, oraz pole EditText do wprowadzenia nowej nazwy. Po kliknięciu przycisku "Nadpisz", metoda sprawdza, czy wprowadzona nazwa nie jest pusta, a następnie aktualizuje bazę danych Firebase i zamyka okno dialogowe. Jeśli nazwa jest pusta, metoda wywołuje się ponownie. Po kliknięciu przycisku "Anuluj", okno dialogowe jest zamykane.

AddingQRDialogNew() - Metoda ta tworzy i wyświetla okno dialogowe (AlertDialog) w aktywności MainActivity w celu dodania nowego produktu na podstawie zeskanowanego kodu QR. Okno dialogowe zawiera pole EditText do wprowadzenia nazwy produktu, hint z informacją "Wprowadź nazwę produktu", tytuł "Wynik skanowania" oraz treść z informacją o braku produktu w bazie danych i kodzie produktu. Po kliknięciu przycisku "Dodaj", metoda sprawdza, czy wprowadzona nazwa nie jest pusta, a następnie dodaje nowy produkt do bazy danych Firebase i zamyka okno dialogowe. Pole EditText zostaje wyczyszczone. Jeśli nazwa jest pusta, metoda wywołuje się ponownie. Po kliknięciu przycisku "Anuluj", okno dialogowe jest zamykane, a pole EditText zostaje wyczyszczone.

getConnectionType(Context context): Metoda ta zwraca typ połączenia sieciowego na podstawie aktualnego połączenia urządzenia.

InitObjects(): Ta metoda tworzy i wyświetla okno dialogowe z paskiem postępu, informujące użytkownika o trwającym procesie łączenia z bazą danych. Okno dialogowe jest niemożliwe do zamknięcia przez użytkownika i zawiera przycisk "Odśwież aplikacje", który po kliknięciu odświeża aplikację.

**Instalacja i konfiguracja**

Krok 1: Pobranie i instalacja aplikacji

1. Pobierz najnowszą wersję aplikacji QRScanner.
2. Zainstaluj aplikację na swoim urządzeniu, postępując zgodnie z instrukcjami na ekranie.

Krok 2: Uruchomienie aplikacji

1. Uruchom aplikację QRScanner, znajdź jej ikonę na ekranie swojego urządzenia i dotknij ją, aby ją otworzyć.
2. Po uruchomieniu aplikacji zostaniesz przekierowany do ekranu głównego, gdzie będziesz mógł rozpocząć konfigurację aplikacji.

Krok 3: Skonfiguruj preferencje aplikacji

1. W głównym ekranie aplikacji znajdź przycisk „Skanuj produkt”.
2. Naciśnij przycisk.
3. Potwierdź uprawnienia, aby umożliwić aplikacji korzystanie z aparatu i kamery.