NekloSmartStore Android library

Библиотека **NekloSmartStore** работает автономно при условии, что будут обеспечены ниже описание шаги и обеспечивает функционирование с удаленно базой данных, обнаружение маяков, сбора статистики, вывод нотификаций.

Для интеграции библиотеки, нужно в Android Studio выполнить import AAR, далее в build.gradle вашего проекта, прописать импортированную библиотеку и дополнительно ее зависимости:

```
compile project(':smartstore')
compile 'com.google.code.gson:gson:2.3'
compile 'com.loopj.android:android-async-http:1.4.9'
compile 'org.altbeacon:android-beacon-library:2.6.1'
```

Библиотека запрашивает следующие права:

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.BLUETOOTH"/>
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>
```

Так же сканирование биконов будет работать только на Android 14+ на устройствах с поддержкой Bluetooth LE

Для запуска библиотеки необходимо выполнить команду

```
SmartStoreHelper.initService(Context context, String apiKey, String objectKey, Class baseActivity, @DrawableRes int notificationIcon)
```

apiKey - будет вам выдан и является уникальным ключом для запросов данного приложения.

objectKey - будет вам выдан и является уникальным ключом для взаимодействия с вашей панелью управления.

У библиотеки имеются следующие параметры для настройки:
baseActivity - класс активити, которая будет открываться, при клике на нотификацию
notificationIcon - идентификатор ресурса иконки, которая будет отображаться в нотификации

Есть расширенная версия метода, если надо укачать свои интервалы обновления данных с сервера и таймаут обнаружения биконов в миллисекундах

```
SmartStoreHelper.initService(Context context, String apiKey, String objectKey, Class baseActivity, @DrawableRes int notificationIcon, int updateInterval, int pushTimeout)
```

Данный метод можно вызвать в классе Application вашего приложения

```
public class DemoApp extends Application {

   private final static int UPDATE_INTERVAL_DEFAULT = 6 * 60 * 60 * 1000; // 6 hours
   private final static int PUSH_TIMEOUT_DEFAULT = 30 * 60 * 1000; // 30 mins

   @Override
   public void onCreate() {
       super.onCreate();
       // init service
       SmartStoreHelper.initService(this,
               getString(R.string.api_key),
               getString(R.string.object_key),
               MainActivity.class,
               R.mipmap.ic_launcher,
               UPDATE_INTERVAL_DEFAULT,
               PUSH_TIMEOUT_DEFAULT);
   }
}
```

При клике пользователя на нотификации, формируется следующий интент:

```
Intent intent = new Intent(this, baseActivity);
intent.setAction(campaign._id); // need use unique action to prevent intent caching
intent.putExtra(INTENT_FIELD_TITLE, campaign.action.title);
intent.putExtra(INTENT_FIELD_TEXT, campaign.action.text);
intent.putExtra(INTENT_FIELD_LINK, campaign.action.link);
```

Эти данные можно использовать в открывающейся активити:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_main);

   checkMessage();
}

@Override
protected void onNewIntent(Intent intent) {
   super.onNewIntent(intent);
   Log.i(TAG, "onNewIntent");
   setIntent(intent);
   // Check incoming message
```

```
    checkMessage();
}

/**
* Check intent on message
*/
private boolean checkMessage() {
    if (getIntent().hasExtra(SmartStoreService.INTENT_FIELD_TITLE)){
        showPopup(
                getIntent().getStringExtra(SmartStoreService.INTENT_FIELD_TITLE),
                getIntent().getStringExtra(SmartStoreService.INTENT_FIELD_TEXT),
                getIntent().getStringExtra(SmartStoreService.INTENT_FIELD_LINK));
        return true;
    }
    return false;
}
```

Если вы хотите подписаться на уведомления от Action биконов, то необходимо зарегистрировать броадкаст ресирвер с фильтром SmartStoreHelper.*getBroadcastAction*()

```
// Register receiver for action beacons
IntentFilter filter = new IntentFilter(SmartStoreHelper.getBroadcastAction());
registerReceiver(actionReceiver, filter);
```

В самом ресирвере можно получить данные о пришедшей нотификации:

```
/**
* Receiver for action beacon
*/
BroadcastReceiver actionReceiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        LogHelper.d("ActionReceiver", "received beacon " +
intent.getStringExtra(SmartStoreService.INTENT_FIELD_TITLE));
        String title = intent.getStringExtra(SmartStoreService.INTENT_FIELD_TITLE);
        String text = intent.getStringExtra(SmartStoreService.INTENT_FIELD_TEXT);
        String link = intent.getStringExtra(SmartStoreService.INTENT_FIELD_LINK);
        if (title != null && text != null){
            showPopup(title, text, link);
        }else{
            String params = intent.getStringExtra(SmartStoreService.INTENT_FIELD_DATA);
            if (params != null){
                // TODO actions
            }
        }
    }
};
```

Для того чтобы не дублировать нотификации от сервиса и сообщения внутри приложения, можно отключить нотификации сервисе через метод
`SmartStoreHelper.setNotificationPosting(Context context, boolean needPosting)`

Рекомендуется это делать в следующих методах активити:

```
@Override
public void onStart(){
   super.onStart();
   // Disable notifications because we start activity and will handle it self
   SmartStoreHelper.setNotificationPosting(this, false);
}

@Override
public void onStop(){
   super.onStop();
   // Enable notifications again
   SmartStoreHelper.setNotificationPosting(this, true);
}
```

Для того чтобы совсем отключить обработку сообщений, используется метод
`SmartStoreHelper.setNotificationEnable(Context context, boolean enable)`
С помощью его можно оргранизовать опцию отключения нотификаций пользователем

Остановить работу сервиса можно с помощью метода
`SmartStoreHelper.stopService(Context context)`