BeaconSmartStore Android library

Библиотека **BeaconSmartStore** 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"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

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

Для запуска библиотеки необходимо выполнить команду (*Есть варианты с различным набором параметров)

```
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)
```

Для использовать пререлизного API необходимо запустить библиотеку с помощью метода `SmartStoreHelper.initService(Context context, String apiKey, String objectKey, Class baseActivity, @DrawableRes int notificationIcon, int updateInterval, int pushTimeout, boolean usePrereleaseAPI)`

Установив `usePrereleaseAPI` в true.

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,
               true);
   }
}
```

При клике пользователя на нотификации, формируется следующий интент:

```
intent.setAction(SmartStoreHelper.getBroadcastAction());
intent.putExtra(INTENT_FIELD_ID, campaign._id);
intent.putExtra(INTENT_FIELD_TITLE, campaign.action.title);
intent.putExtra(INTENT_FIELD_SHORT_DESCRIPTION, campaign.action.shortDescription);
intent.putExtra(INTENT_FIELD_FULL_DESCRIPTION, campaign.action.fullDescription);
intent.putExtra(INTENT_FIELD_LINK, campaign.action.link);
intent.putExtra(INTENT_FIELD_DATA, campaign.action.params);
intent.putExtra(INTENT_FIELD_DELAY, campaign.action.delay);
if (campaign.action.customAction != null) {
   intent.putExtra(INTENT_FIELD_SCREEN, campaign.action.customAction.screen);
}
```

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_SHORT_DESCRIPTION),
             getIntent().getStringExtra(SmartStoreService.INTENT_FIELD_LINK),
             getIntent().getStringExtra(SmartStoreService.INTENT_FIELD_FULL_DESCRIPTION),
             getIntent().getStringExtra(SmartStoreService.INTENT_FIELD_ID),
             getIntent().getLongExtra(SmartStoreService.INTENT_FIELD_DELAY, 0));
        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) {
        Log.d("ActionReceiver", "received beacon " +
intent.getStringExtra(SmartStoreService.INTENT_FIELD_TITLE));
        String id = intent.getStringExtra(SmartStoreService.INTENT_FIELD_ID);
        String title = intent.getStringExtra(SmartStoreService.INTENT_FIELD_TITLE);
        String shortDescription =
intent.getStringExtra(SmartStoreService.INTENT_FIELD_SHORT_DESCRIPTION);
        String fullDescription =
intent.getStringExtra(SmartStoreService.INTENT_FIELD_FULL_DESCRIPTION);
        String link = intent.getStringExtra(SmartStoreService.INTENT_FIELD_LINK);
        long delay = intent.getLongExtra(SmartStoreService.INTENT_FIELD_DELAY, 0);
        String screen = intent.getStringExtra(SmartStoreService.INTENT_FIELD_SCREEN);
        Log.d("BSS broadcast", "screen: " + screen);
        if (title != null && shortDescription != null) {
            showPopup(title, shortDescription, link, fullDescription, id, delay);
        } 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)
```

Для того чтобы логировать время активности приложения, нужно добавить метод

```
SmartStoreHelper.setApplicationState(Context context, boolean isActive)
```

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);
    SmartStoreHelper.setApplicationState(this, true)
}

@Override
public void onStop(){
    super.onStop();
    // Enable notifications again
```

```
    SmartStoreHelper.setNotificationPosting(this, true);
    SmartStoreHelper.setApplicationState(this, false)
}
```

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

Для регистрации пользователя со сторонним id воспользуйтесь методом
`SmartStoreHelper.registerExternalUser(String userId, String name, String email, final ApiDataCallback<String> callback)`

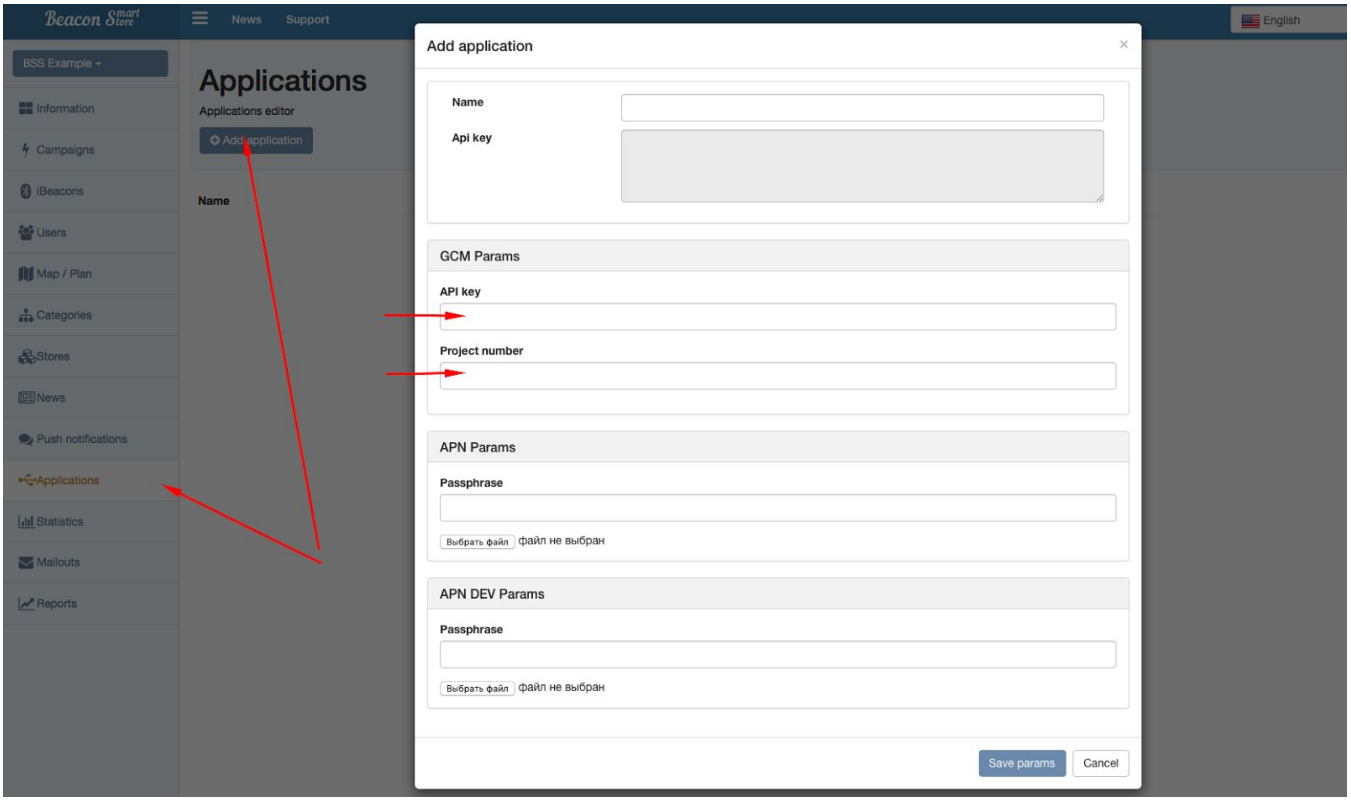
Для авторизации пользователя со сторонним id воспользуйтесь методом
`SmartStoreHelper.loginExternalUser(final Context context, String userId, final ApiDataCallback<LoginData> callback)`

Для выхода и сессии авотризованного пользователя со сторонним id воспользуйтесь методом
`SmartStoreHelper.logoutExternalUser(final Context context, final ApiDataCallback<String> callback)`

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

Библиотека поддерживает работу с push-нотификациями через GCM. Нотификации позволяют отображать информацию для пользователей, а также инициировать обновление данных в библиотеке.

Для настройки необходимо внести параметры в панель управления.



Для передачи device token в панель утправления необходимо вызвать метод

`StatsHelper.getInstance().postDeviceToken(String deviceToken):`

Для обработки внутри приложение в GcmListenerService необходимо выполнить проверку принятых данных. Данные могут быть 2-х видов:

- Уведомление на обновление данных (содержит в data **`content-available = 1 и collapse_key = bss`**)
- Уведомление для отображения пользователю (data с данными **`gcm.notification.`**)

```
/**
* Called when message is received.
*
* @param from SenderID of the sender.
* @param data Data bundle containing message data as key/value pairs.
*             For Set of keys use data.keySet().
*/
// [START receive_message]
@Override
public void onMessageReceived(String from, Bundle data) {
   boolean isForceUpdate = false;
   try {
```

```
            isForceUpdate = data.getString("content-available").equalsIgnoreCase("1")
                    && data.getString("collapse_key").equalsIgnoreCase("bss");
        } catch (Exception e) {
            e.printStackTrace();
        }
        if (isForceUpdate) {
            SmartStoreHelper.forceUpdateData(getApplicationContext());
        } else {
            String title = data.getString("gcm.notification.title");
            String message = data.getString("gcm.notification.body");
            Log.d(TAG, "From: " + from);
            Log.d(TAG, "Title: " + title);
            Log.d(TAG, "Message: " + message);

            if (from.startsWith("/topics/")) {
                // message received from some topic.
            } else {
                // normal downstream message.
            }

            // [START_EXCLUDE]
            /**
             * Production applications would usually process the message here.
             * Eg: - Syncing with server.
             *     - Store message in local database.
             *     - Update UI.
             */

            /**
             * In some cases it may be useful to show a notification indicating to the user
             * that a message was received.
             */
            sendNotification(title, message);
            // [END_EXCLUDE]
        }
    }
    // [END receive_message]

    /**
    * Create and show a simple notification containing the received GCM message.
    *
    * @param message GCM message received.
    */
    private void sendNotification(String title, String message) {
        Intent intent = new Intent(this, MainActivity.class);
        intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
```

```
    PendingIntent pendingIntent = PendingIntent.getActivity(this, 0 /* Request code */,
intent,
            PendingIntent.FLAG_ONE_SHOT);

    Uri defaultSoundUri = RingtoneManager.getDefaultUri(RingtoneManager.TYPE_NOTIFICATION);
    NotificationCompat.Builder notificationBuilder = new NotificationCompat.Builder(this)
            .setSmallIcon(getApplication().getApplicationInfo().icon)
            .setContentTitle(title)
            .setContentText(message)
            .setAutoCancel(true)
            .setSound(defaultSoundUri)
            .setContentIntent(pendingIntent);

    NotificationManager notificationManager =
            (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);

    notificationManager.notify(0 /* ID of notification */, notificationBuilder.build());
}
```