1. SDK integration

To integrate your application with devtodev system please perform the following actions:
- add the application to the Space using the wizard for adding application
- download the latest version of devtodev SDK
- integrate SDK into your application. The integration may be whether partial or
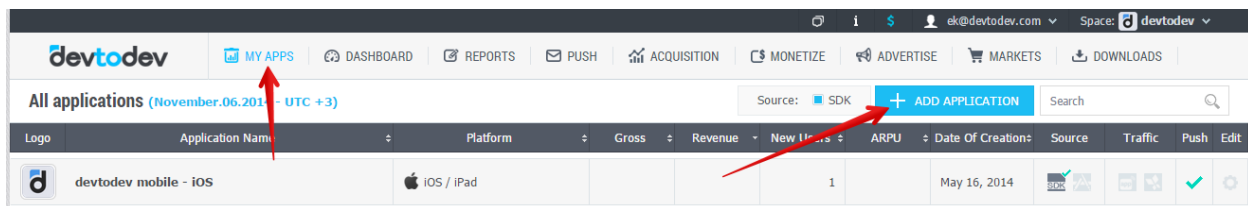including all the possibilities.

1.1. Adding application to the Space

If you haven't add any apps yet, follow this step-by-step guide to add a new one:

1.1.1 Adding new app

You must have "Admin" role in the Space to be able to add new apps
If you have aforesaid rights than "ADD APPLICATION" button will be displayed in "MY APPS"
tab.

After clicking it you'll get into the wizard for adding application.

1.1.2 Choosing platform

Depending on the chosen platform and the app store, you'll be suggested to choose one of
available connection types (integration with developer account available only for Apple App
Store and Google Play)
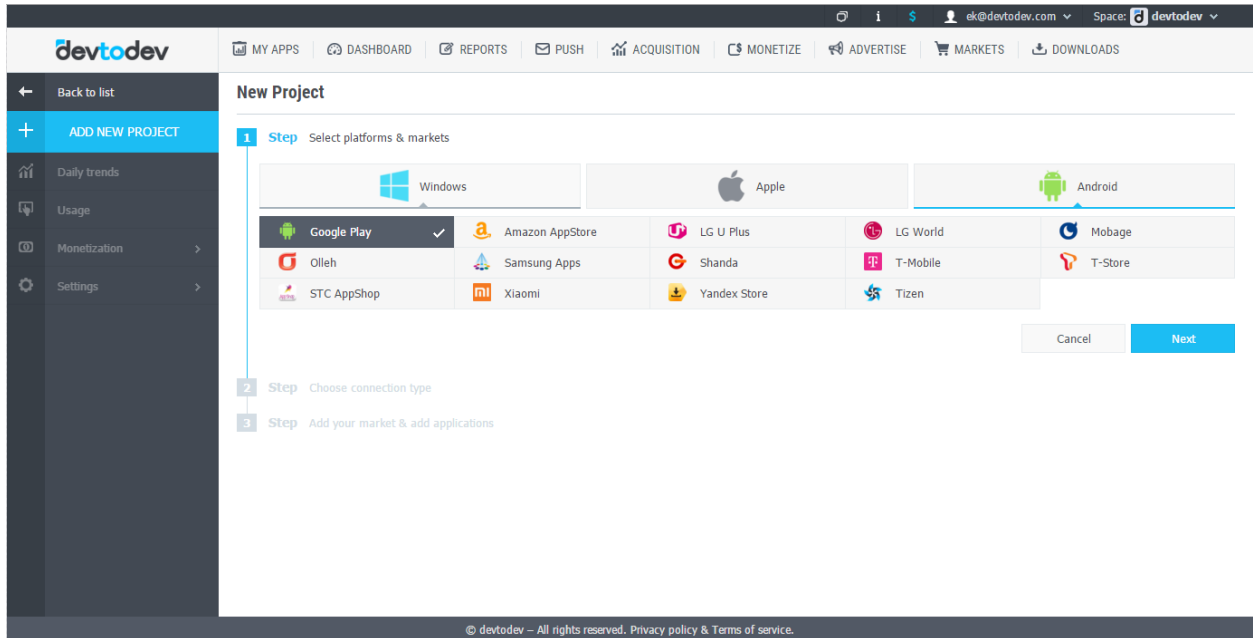
1.1.3 Choosing connection type

This step is only available for iOs apps and Android in Google Play
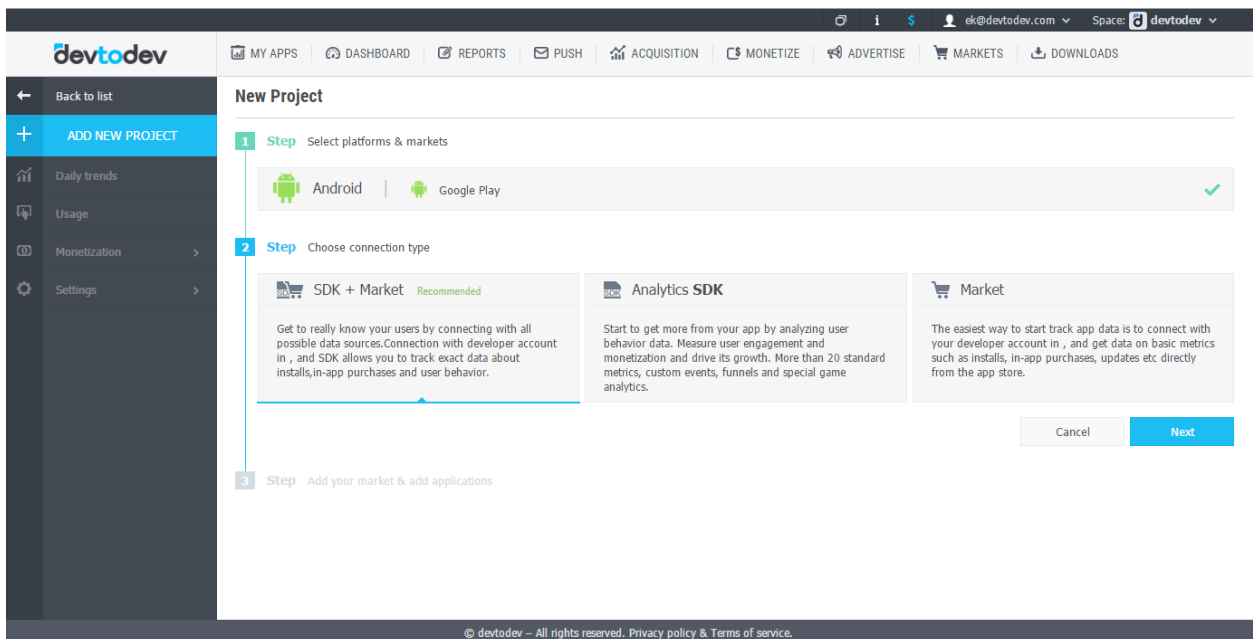
There are three types of connection available:

1. Connection with SDK and developer account in the app store

The optimal solution. This connection type allows to track both downloads and in-app purchases from markets and user-behavior data from SDK

2. SDK integration

Integration with analytic SDK allows to track more than 20 build-in engagement and monetization metrics, special game metrics and custom events.

3. Connection with developer account in app markets

Connection with developer account in iTunes Connect or Google Play – is the fastest way to start with devtodev. It allows to track downloads and in-app purchases directly from app market.

1.1.4 Filling in data for the account in the app store

If you have chosen "SDK+Market" or "Market" connection you'll be suggested to fill in your developer account details (login and password).

Notice: We only use Google Play financial subaccounts with turned off recovery options to collect data.

You also can limit devtodev access to iTunes Connect by setting up a separate account with Sales role only.

You can find detail instructions how to create subaccounts on iTunes - https://www.devtodev.com/forum/index.php?topic=22.0

After all necessary fields are filled in, the system automatically imports all available apps.

1.1.5a Choosing apps to import

Tick all apps you want to add to devtodev and click "Next"

Notice: all available for the current account apps will be ticked by default

1.1.5b Filling in app details

This step will be available if you have chosen SDK integration (without app market connection)

Fill in app name and upload its icon(optional).

1.1.6 Unique keys for SDK integration

As soon as all necessary fields are filled in, new app will be added to your space.

At the last step you get App ID and the Secret key. This is necessary for integrating analytic SDK into your application. Also you can download SDK files and instructions here.

Later, unique keys can be found in the application settings (My apps →App Name → Settings → Integration), all SDK files and instructions are stored in DOWNLOADS

1.2 Adding SDK into the application

1. Download the latest version of devtodev SDK from the "Download" section
2. Add devtodev.jar library to your application
3. Add the following permissions to your AndroidManifest.xml
   Necessary:

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

   (Required for sending analytics data to our server)
   Additionally:

   ```
   <uses-permission
   android:name="android.permission.ACCESS_WIFI_STATE" />
   ```

   (Required for devices' MAC addresses collection)

   ```
   <uses-permission
   android:name="android.permission.READ_PHONE_STATE" />
   ```

   (Required for cellular operator data collection)
4. Add init method into onCreate() method of your Activity

   ```
   DevToDev.init(Context context, String appKey, String
   secretKey);
   ```

5. Add startSession and stopSession methods into onStart and onStop methods of your Activity

   ```
   DevToDev.startSession(StartSessionEvent startSessionEvent);
   DevToDev.endSession(EndSessionEvent endSessionEvent);
   ```

2. Analytics integration

2.1. The list of basic methods

2.1.1. Passing the tutorial

The event allows to filter users in the course of passing the tutorial.The event is sent upon passing every step.

Use constants to record base evetns of the tutorial:

- **DevToDevTutorialState.Start** - at the beginning, before the first step is completed;
- **DevToDevTutorialState.Finish** - instead of the last step number;
- **DevToDevTutorialState.Skipped** - if user skipped the tutorial.

In other cases use step numbers. Make sure you use numbers above 0 numerating the steps.

```
/**
 * The event allowing to track the stage of tutorial a player
is on.
 * @param int tutorialStep - the latest successfully completed
tutorial step.
 */

DevToDev.tutorialCompleted(int tutorialStep);
```

2.1.2. Levelling up

You can analyse the distribution of the players over the levels. The event is sent over reaching a new level by a player.

```
/* *
 * Player has reached a new level
 * @param int level - level reached by the player.
 * /
DevToDev.levelUp(int level);
```

To track the average account balance of in-game currency by the end of each level, please also provide the list of currency names and their amount.

```
/**
* @param int level - уровень
* @param HashMap<String, Integer> resources - hashmap with the
currency names and amounts */
HashMap<String, Integer> resources = new HashMap<String,
Integer>();
resources.put("Currency name 1", 1000);
resources.put("Currency name 2", 10);
DevToDev.levelUp(level, resources);
```

To track the average amount of in-game currency earned or purchased during a level, it is necessary to send a special event after each time an in-game account is replenished.

```
/**
* @param String currencyName - currency name
* @param float currencyAmount - the amount an account has been
credited with
* @param AccrualType accrualType - the way the currency was
obtained: earned or purchased
*/
DevToDev.currencyAccrual(currencyName, currencyAmount,
accrualType);
```

AccrualType can take one of following values:

```
public enum AccrualType {
    Earned,
    Purchased
}
```

2.1.3. Payment

To track the payments, add this event right after the platform confirms that the payment went through.

```
/**
* Register transactions made through the platform's payment
system.
*
* @param String paymentId - transaction ID
* @param float inAppPrice - product price (in user's currency)
* @param String inAppName - product name
* @param String inAppCurrencyISOCode - transaction currency (in
ISO 4217 format)
* /

DevToDev.realPayment(String paymentId, float inAppPrice, String
inAppName, String inAppCurrencyISOCode);
```

2.1.4. Connecting to social network

```
/**
* Tracks the existence of a connection with a social network.
* Use pre-defined or custom values as an identifier.
* @param SocialNetwork socialNetwork - social network id
**/
DevToDev.socialNetworkConnect(SocialNetwork socialNetwork);
```

Use the current constants to specify social network:

```
SocialNetwork.Facebook
SocialNetwork.Twitter
SocialNetwork.GooglePlus
SocialNetwork.Vk
and so on...
```

Otherwise, create social network the object of your own.

```
SocialNetwork socialNetwork = SocialNetwork.Custom(String
networkName);
```

2.1.5. Posting to social network

Track publications to social networks from the app to improve viral marketing efficiency.The event is sent after the social network confirms the publication.

```
/**
* Tracks the existence of posts to a social network.
* @param socialNetwork - social network Id
* @param reason - the reason of posting
* /
DevToDev.socialNetworkPost(SocialNetwork socialNetwork, String
reason);
```

As a «reason» parameter we recommend that you indicate actions which encourage users to make publication.

 For example:
- Start playing
- New level reached
- New building
- New ability
- Quest completed
- New item
- Collection completed
- Invitation

- Asking for help
- New Record
- Acheivement
- URL sharing
- Recommendation
- Review

and so on...

Use the current constants to specify social network:

```
SocialNetwork.Facebook
SocialNetwork.Twitter
SocialNetwork.GooglePlus
SocialNetwork.Vk
and so on...
```

Otherwise, create social network the object of your own.

```
SocialNetwork socialNetwork = SocialNetwork.Custom(String
networkName);
```

2.1.6. In-app purchases

To track expenditures of in-game currency and the popularity of the products, add this event right after a purchase is made.

```
/**
 * In-app purchase with a definite article ID.
 *
 * @param purchaseId - unique purchase Id
 * @param purchaseType - purchase type or group
 * @param purchaseAmount - count of purchased goods
 * @param purchasePrice - cost of purchased goods (total cost -
 if several goods were purchased)
 * @param purchaseCurrency - currency name
 */

DevToDev.inAppPurchase(String purchaseId, String purchaseType,
int purchaseAmount, int purchasePrice, String
purchaseCurrency);
```

In case a product was bought for several game currencies at once, it is necessary to make a hashmap including the names and amounts of the paid currencies.

```
HashMap<String, Integer> resources = new HashMap<String,
Integer>();
resources.put("currency_1", 120);
resources.put("currency_2", 29);
…and so on…

DevToDev.inAppPurchase(String purchaseId, String purchaseType,
int purchaseAmount, HashMap<String, Integer> resources);
```

2.1.7. Custom Events

If you want to count the events you need but they are not among the basic ones, use custom events. An event should have a unique name and can include up to 10 parameters.
You can use up to 300 unique events' names. Every event can have up to 100 records (it means that an event can run up to 100 times at the same time on the single device).

The maximum length of an event name is 72 symbols. The maximum length of a parameter name is 32 symbols and length of parameter value - 255 symbols.

```
/**
* @param String eventName - event name
**/
DevToDev.customEvent(eventName);
```

Also 10 parameter names may be associated with any event:

```
CustomEventParams params = new CustomEventParams();
params.putDate("date", new Date());
params.putDouble("double", 1.12);
params.putFloat("float", 9.99f);
params.putInteger("int", 145);
params.putLong("long", 123L);
params.putString("string","start");
```

Then use method:

```
/**
* @param String eventName - event name
* @param CustomEventParams params - event parameters
**/
DevToDev.customEvent(eventName, params);
```

2.2. The list of secondary methods

2.2.1. Age (secondary method).

```
/**
* Track user's age
* @param int age - user's age
* /
DevToDev.age(int age);
```

2.2.2. Gender (secondary method).

```
/**
* Track's user's gender
* @param Gender gender - predefined value.
* /
DevToDev.gender(Gender gender);
```

2.2.3. Cheater

In case you have your own methods of determining cheaters in the application, you can have such users marked. Payments made by them will not be taken into account in the statistics.

```
/**
* Mark user if it's cheater.
* @param boolean isCheater
* /
DevToDev.cheater(boolean isCheater);
```

2.2.4. User identifier

To track the users by your own ID use this method:

```
/**
* @param String userUdid
* /
DevToDev.setUserUdid(String userUdid);
```

To see what identifier is used at the moment:

```
/**
* @return Custom User Identifier
* /
DevToDev.getUserUdid();
```

2.2.5. OpenUdid

```
/**
 * @return Open Udid
 * /
DevToDev.getOpenUdid();
```

2.2.6. ODIN1

```
/**
 * @return ODIN1
 * /
DevToDev.getOdin1();
```

2.2.7. UUID

```
/**
 * @return UUID
 * /
DevToDev.getUUID();
```

2.2.8. Debug mode

To enable the debug mode and make SDK notifications displayed in the console use this method:

```
/**
 * @param logLevel
 */
DevToDev.setLogLevel(LogLevel logLevel);
```

2.2.9. Forced sending

To send events pack before it is filled or before its formation period you can use immediate dispatch:

```
DevToDev.sendBufferedEvents();
```

2.2.10. Current SDK version

To get the version of integrated SDK use this method:

```
/**
 * @return SDKVersion
 * /
DevToDev.getSdkVersion();
```

2.3. Validation of payments and time adjustments

2.3.1. Payments validation

To be protected from fraudulent transactions, we recommend you to use devtodev Anticheat service
Use this method, and devtodev will check the transaction validity with the payment platform, and the response will be returned to the application.
Call following method when GooglePlay returns transaction to your onActivityResult:

```
DevToDevCheat.verifyPayment(String receipt, String signature,
String publicKey, OnVerifyListener onVerifyListener);
```

You can get sharedSecret key here:
1. Go to the Google Play Developer Console and sign in. Make sure that you sign in to the account from which the application you are licensing is published (or will be published).
2. In the application details page, locate the Services & APIs link and click it.
3. In the Services & APIs page, locate the Licensing & In-App Billing section.

Your public key for licensing is given in the Your License Key For This Application field.
Result can take one of following values:

```
public enum VerifyStatus {
Valid,
Invalid,
InternalError,
ServerError}
```

In case of successful check call following main SDK method:

```
DevToDev.realPayment(String pPaymentId, float pInAppPrice,
String pInAppName, String pInAppCurrencyISOCode);
```

2.3.2. Time cheats check

To check for time cheats call checkTime method every time when app is being launched

```
DevToDevCheat.verifyTime(OnTimeVerifyListener
onTimeVerifyListener);
```

Result can take one of following values:

```
public enum TimeStatus {
Valid,
Forward,
Rewind
}
```

2.4. Checking the events sent

You can control the events sent from the application in "SDK logs" section. To do so, go to the application settings ("My Apps" → App Name → "Settings" → "SDK logs").
Attention! By default the application settings section is available only to the owner of the space and to the users in "Administrator" role.
Choose date and time, type of the event. If necessary, specify the device ID. Up to 500 events will be displayed.

Find Devtodev UDID correspondent to your test device on the UDID event report page. Using the search by Devtodev UDID you will easily track any other events coming solely from your test device.

3. PUSH-notifications

3.1 General information

To enable PUSH-notifications you will have to perform the following actions:
- Add the application to your space in devtodev system
- Get API key from Google APIs Console. It is nessesary to activate Google Cloud Messaging for Android before key generation.
- Submit the data to the application settings in devtodev system
- Integrate devtodev SDK to the application (see the "SDK integration" division to learn more about integrating and initializing devtodev SDK)
- Add several lines of the code to switch in the push-notification to the SDK
- Create a campaign for sending push-notifications in "Push" section

3.2. API Key generation

- Make sure that Google Play Services are added into your app.
- Create Google Api Project.
- Open Google Developer Console, and if you haven't created an API project yet, click Create Project.
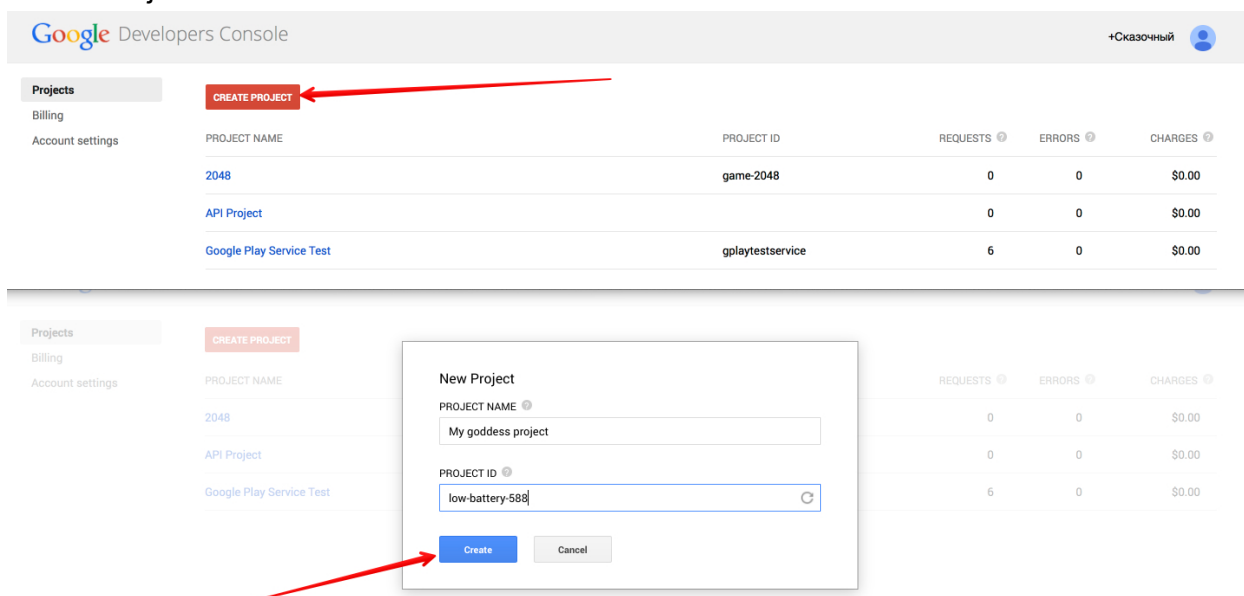
Supply a project name and click Create
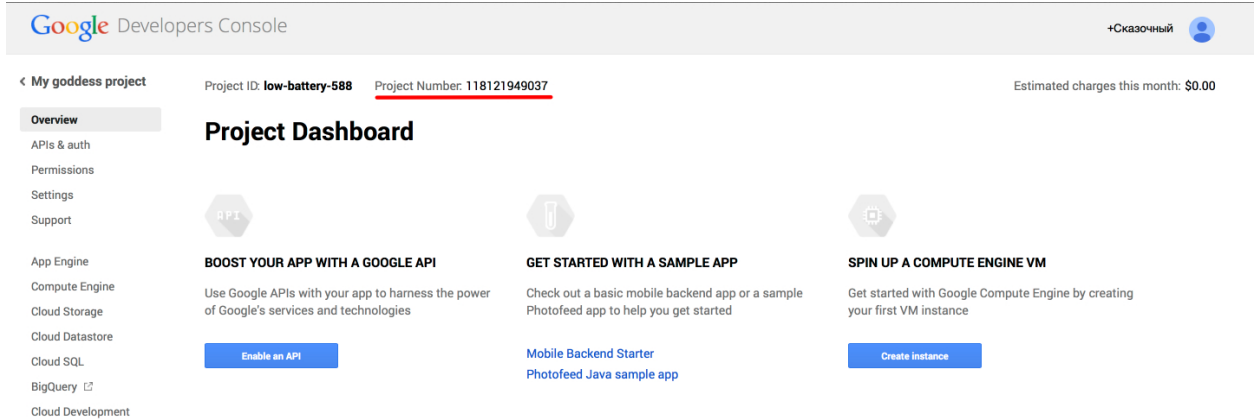
Once the project has been created, a page appears that displays your project ID and project number. For example, Project Number: 118121949037

- Copy down your project number. You will use it later on as the PUSH_SENDER_ID.
- Generate Server Key:
  - Choose APIs & auth in [Google APIs Console](#) main page and turn on Google Cloud Messaging for Android.
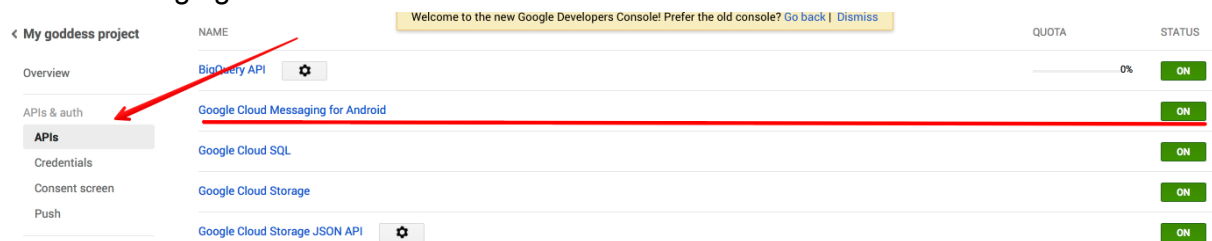
  - Create a server key: open Credentials tab, click "Create new key" and choose Server key button.
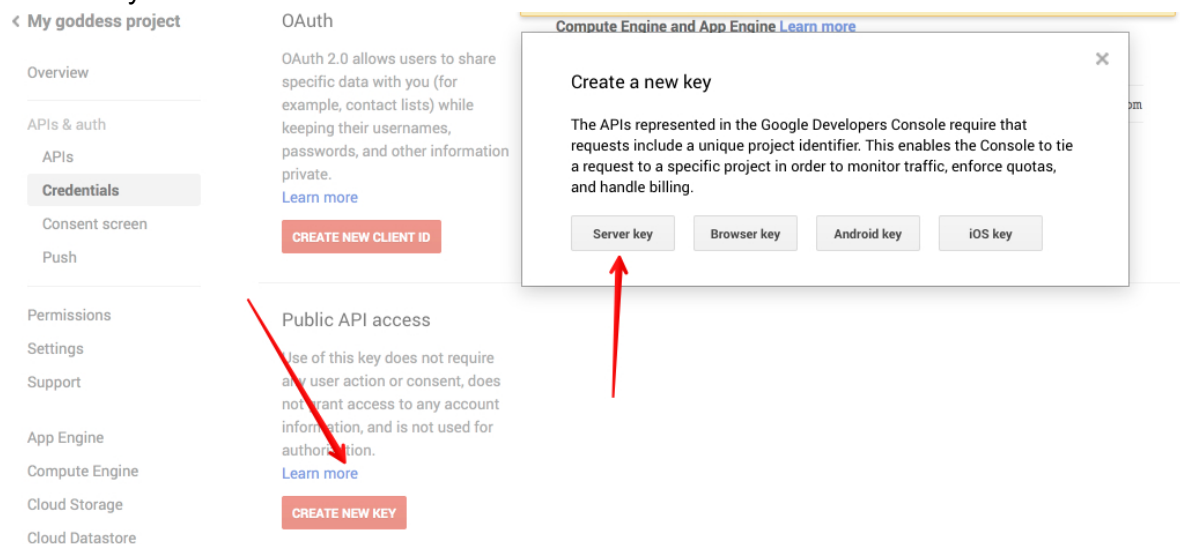
○ Click "Create"
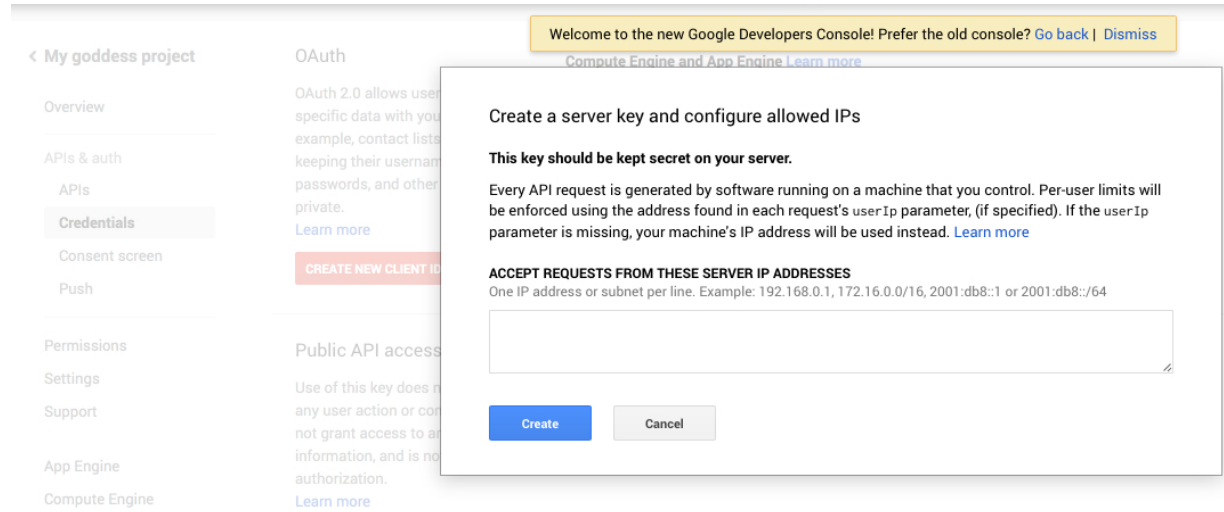
○ Your key will be in API key field.

3.3. Insert API Key to the site

Insert API Key into Integration section of application settings panel ("My apps" →App Name → "Settings" → "Integration")
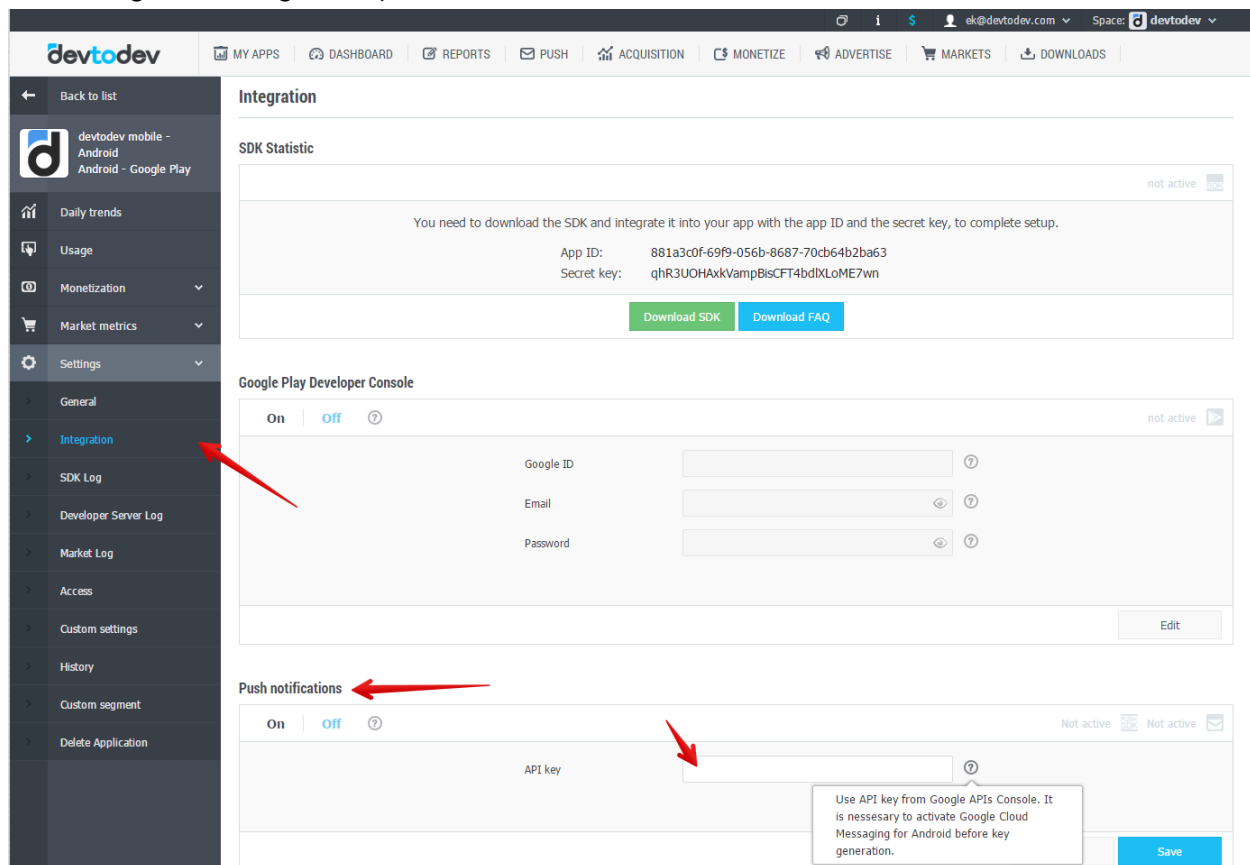
3.4. SDK Integration

Follow these steps to integrate Push-notifications into your app using devtodev SDK:

- Integrate devtodev SDK to the application (see the "SDK integration" division to learn more about integrating and initializing devtodev SDK)
- Add following strings to AndroidManifest.xml:

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission
android:name="com.google.android.c2dm.permission.RECEIVE" />
<permission
android:name="com.example.gcm.permission.C2D_MESSAGE"
android:protectionLevel="signature" />
<uses-permission
android:name="com.devtodev.gcm.permission.C2D_MESSAGE" />
```

to <application>…</application>

```
<receiver
android:name="com.devtodev.push.logic.GcmBroadcastReceiver"
android:permission="com.google.android.c2dm.permission.SEND" >
<intent-filter>
<action android:name="com.google.android.c2dm.intent.RECEIVE"
/>
<category android:name="com.devtodev.gcm" />
</intent-filter>
</receiver>
<service
android:name="com.devtodev.push.logic.GcmIntentService" />
<meta-data android:name="PUSH_SENDER_ID" android:value="\
XXXXXXXXXX" />
```

Replace XXXXXXX with your key received from google developer console.
If you need vibro signal when push is delivered

```
<uses-permission android:name="android.permission.VIBRATE" />
```

If you need a device to awake from notification

```
<uses-permission android:name="android.permission.WAKE_LOCK" />
```

- Initialize DevToDevPushManager call in main activity
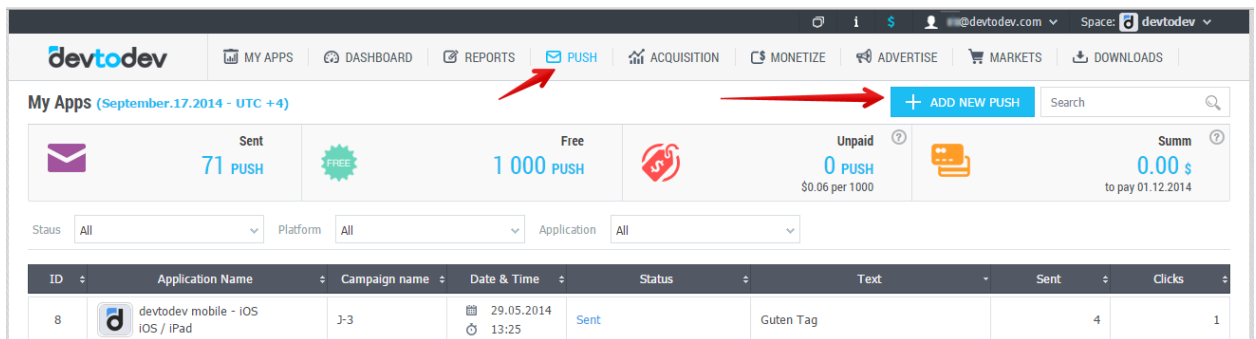
```
private PushListener listener = new PushListener() {
@Override
    public void onRegisteredForPushNotifications(String
deviceId) {
        Log.d("PushListener", "onRegisteredForPushNotifications:
" + deviceId);
    }
    @Override
    public void onFailedToRegisteredForPushNotifications(String
error) {
        Log.d("PushListener",
"onFailedToRegisteredForPushNotifications: " + error);
    }
    @Override
    public void onPushNotificationsReceived(HashMap<String,
Object> params) {
        Log.d("PushListener", "onPushNotificationsReceived: " +
params);
    }
};


@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.main);
DevToDev.init(getBaseContext(), ApplicationKey,
ApplicationSecret);
DevToDevPushManager.init(getIntent());
DevToDevPushManager.setPushListener(listener);
}
```

3.5. Creating a new push-notification in devtodev interface

1. Open PUSH tag and click on ADD NEW PUSH button.

2. Fill in campaign name, select an app for delivery.
   Attention! You can create a campaign only after at least one push token comes from

devtodev SDK integrated to your application. Otherwise the app will not be displayed in the list.

3. Choose user group to send a message. You can choose existing segment or create a new one.

4. Enter notification details

5. Schedule the delivery

6. Thats it!