

# Click Tracking & S2S Postback System - Complete Documentation

## Table of Contents

- 1. [Overview](#)
- 2. [System Flow](#)
- 3. [API Endpoints](#)
- 4. [How It Works](#)
- 5. [Flutter Integration Guide](#)
- 6. [Example Flutter Code](#)
- 7. [Testing](#)

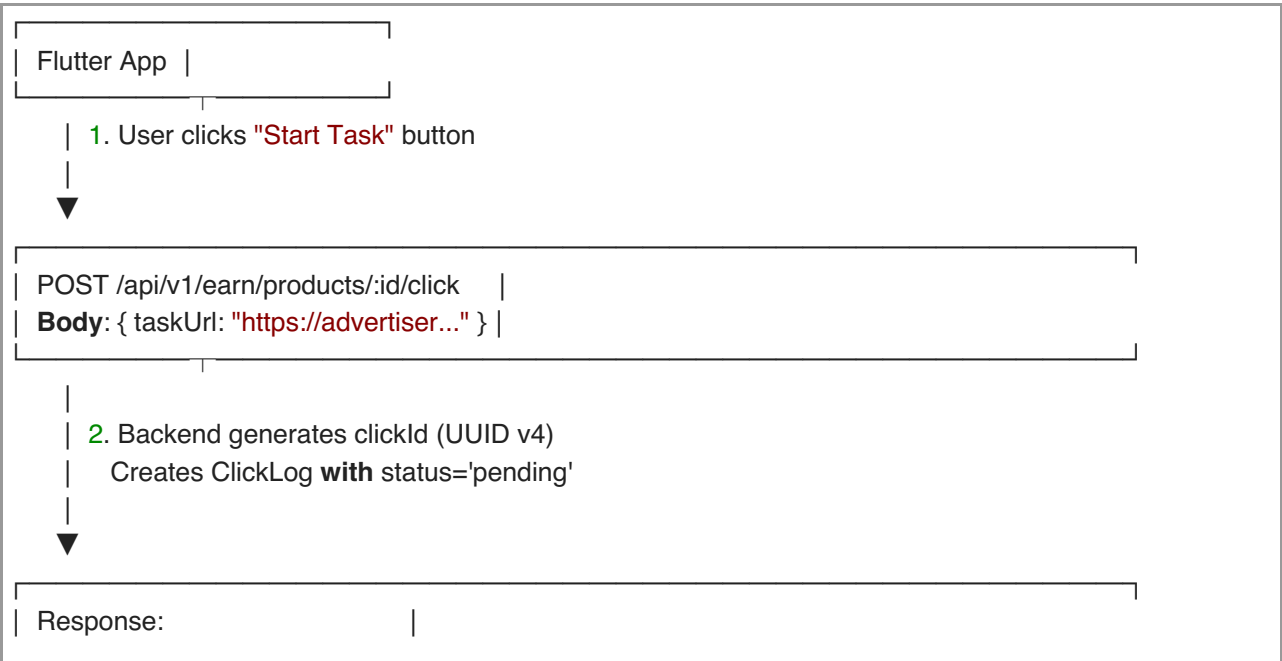
## Overview

This system tracks user clicks on offers/products and receives server-to-server (S2S) postbacks from advertisers when users complete tasks. It automatically credits users' wallets when conversions are confirmed.

### Key Components:

- **Click ID:** Unique UUID v4 identifier for each click
- **Tracking Link:** URL that redirects user to advertiser with tracking parameters
- **Postback URL:** S2S endpoint that advertisers call when conversion happens
- **Click Log:** Database record tracking each click's status
- **Earning:** Credit record created when postback is received

## System Flow



```
{
  clickId: "abc-123-...",
  redirectUrl: "https://advertiser...
    ?click_id=abc-123...
    &postback_url=https://...",
  expiresAt: "2024-01-01T12:00:00Z"
}
```

3. Flutter opens redirectUrl in WebView

Advertiser's Website  
User completes **task** (signup, install, etc)

4. Advertiser detects conversion

```
POST https://yourapp.com/api/v1/earn/
  postback?click_id=abc-123...
Body: {
  amount: 50.00,
  status: "completed",
  transactionId: "txn_123",
  conversionId: "conv_456"
}
```

5. Backend processes postback:

- Finds ClickLog by clickId
- Creates/Updates Earning **record**
- Credits user's wallet
- Updates ClickLog status='converted'
- Processes referral commission

User's wallet **is** credited automatically  
Earning appears **in** earnings list

## API Endpoints

### 1. Generate Click (User-Facing)

**Endpoint:** POST /api/v1/earn/products/:productId/click

**Authentication:** Required (Bearer token)

**Request Body:**

```
{
  "taskUrl": "https://advertiser.com/offer",
  "offerId": "optional_offer_id"
}
```

**Response:**

```
{
  "success": true,
  "message": "Click generated successfully",
  "data": {
    "clickId": "550e8400-e29b-41d4-a716-446655440000",
    "redirectUrl": "https://advertiser.com/offer?
click_id=550e8400...&postback_url=https://yourapp.com/api/v1/earn/postback?click_id=550e8400...",
    "expiresAt": "2024-01-02T12:00:00.000Z",
    "trackingUrl": "https://yourapp.com/api/v1/earn/postback?click_id=550e8400..."
  }
}
```

**2. Track Click (Optional Analytics)**

**Endpoint:** GET /api/v1/earn/track/:clickId

**Authentication:** Not required

**Use Case:** Optional endpoint to track when user actually visits the redirect URL (for analytics)

**Response:**

```
{
  "success": true,
  "message": "Click tracked",
  "data": {
    "clickId": "550e8400-e29b-41d4-a716-446655440000",
    "status": "pending"
  }
}
```

**3. Postback (S2S - Called by Advertiser)**

**Endpoint:** POST /api/v1/earn/postback?click\_id={clickId}

**Authentication:** Not required (public endpoint for advertisers)

**Query Parameters:**

- click\_id (required): The click ID from the redirect URL

**Request Body:**

```
{
  "amount": 50.00,
  "status": "completed",
  "transactionId": "txn_123456",
  "conversionId": "conv_789012",
  "offerId": "optional_offer_id"
}
```

#### Response:

```
{
  "success": true,
  "message": "Postback received and processed",
  "data": {
    "earningId": "earning_123",
    "clickId": "550e8400-e29b-41d4-a716-446655440000",
    "conversionId": "conv_789012",
    "status": "completed",
    "amount": 50.00
  }
}
```

## How It Works

### Step 1: Click Generation

When a user wants to start a task:

1. **Flutter app calls** POST `/api/v1/earn/products/:productId/click` with the `taskUrl` (the advertiser's offer URL)
2. **Backend generates:**
  - Unique `clickId` using UUID v4
  - `postbackUrl`: `https://yourapp.com/api/v1/earn/postback?click_id={clickId}`
  - `redirectUrl`: The `taskUrl` with two query parameters:
    - `click_id`: The generated click ID
    - `postback_url`: The postback URL
3. **Backend creates a ClickLog record** with:
  - `clickId`: Unique identifier
  - `userId`: User who clicked
  - `productId`: Product/offer
  - `taskUrl`: Original advertiser URL
  - `redirectUrl`: URL with tracking parameters
  - `status`: 'pending'
  - `expiresAt`: 24 hours from now
4. **Response is sent** to Flutter app with `clickId`, `redirectUrl`, and expiration info

### Step 2: User Completes Task

1. **Flutter app opens** redirectUrl in a WebView or browser
2. **User completes the task** on advertiser's website (signup, install app, make purchase, etc.)
3. **Advertiser tracks the conversion** using the click\_id parameter

### Step 3: Postback Received

When advertiser confirms the conversion:

1. **Advertiser sends POST request** to the postback\_url with:
    - click\_id in query parameter
    - Conversion data in body (amount, status, transactionId, etc.)
  2. **Backend processes postback:**
    - Finds ClickLog by clickId
    - Validates click hasn't expired
    - Checks click hasn't already been converted
    - Creates/Updates Earning record
    - Credits user's wallet balance
    - Updates ClickLog status to 'converted'
    - Processes referral commission if applicable
  3. **Returns success response** to advertiser
- 

## Flutter Integration Guide

### Setup

1. **Add HTTP package** to pubspec.yaml:

```
dependencies:  
  http: ^1.1.0  
  url_launcher: ^6.2.1  
  webview_flutter: ^4.4.0 # For in-app WebView
```

2. **Create API service** for earnings endpoints
3. **Create WebView screen** to open tracking URLs

### Implementation Steps

#### Step 1: Create API Service

```
// lib/services/earnings_api.dart  
import 'package:http/http.dart' as http;  
import 'dart:convert';  
  
class EarningsApi {  
  final String baseUrl = 'https://yourapp.com/api/v1';  
  final String? token; // Your auth token  
  
  EarningsApi(this.token);
```

```

// Generate click and get tracking URL
Future<ClickResponse> generateClick({
  required String productId,
  required String taskUrl,
  String? offerId,
}) async {
  final url = Uri.parse('$baseUrl/earn/products/$productId/click');

  final response = await http.post(
    url,
    headers: {
      'Content-Type': 'application/json',
      'Authorization': 'Bearer $token',
    },
    body: jsonEncode({
      'taskUrl': taskUrl,
      if (offerId != null) 'offerId': offerId,
    }),
  );

  if (response.statusCode == 201) {
    final data = jsonDecode(response.body);
    return ClickResponse.fromJson(data['data']);
  } else {
    throw Exception('Failed to generate click: ${response.body}');
  }
}

// Track click (optional analytics)
Future<void> trackClick(String clickId) async {
  final url = Uri.parse('$baseUrl/earn/track/$clickId');

  await http.get(url);
  // Ignore response - this is just for analytics
}

// Response model
class ClickResponse {
  final String clickId;
  final String redirectUrl;
  final DateTime expiresAt;
  final String trackingUrl;

  ClickResponse({
    required this.clickId,
    required this.redirectUrl,
    required this.expiresAt,
    required this.trackingUrl,
  });
}

```

```

factory ClickResponse.fromJson(Map<String, dynamic> json) {
  return ClickResponse(
    clickId: json['clickId'],
    redirectUrl: json['redirectUrl'],
    expiresAt: DateTime.parse(json['expiresAt']),
    trackingUrl: json['trackingUrl'],
  );
}

```

## Step 2: Create WebView Screen

```

// lib/screens/task_webview_screen.dart
import 'package:flutter/material.dart';
import 'package:webview_flutter/webview_flutter.dart';

class TaskWebViewScreen extends StatefulWidget {
  final String redirectUrl;
  final String clickId;

  const TaskWebViewScreen({
    Key? key,
    required this.redirectUrl,
    required this.clickId,
  }) : super(key: key);

  @override
  State<TaskWebViewScreen> createState() => _TaskWebViewScreenState();
}

class _TaskWebViewScreenState extends State<TaskWebViewScreen> {
  late WebViewController controller;
  bool isLoading = true;

  @override
  void initState() {
    super.initState();

    controller = WebViewController()
      ..setJavaScriptMode(JavascriptMode.unrestricted)
      ..setNavigationDelegate(
        NavigationDelegate(
          onPageStarted: (String url) {
            setState(() => isLoading = true);
          },
          onPageFinished: (String url) {
            setState(() => isLoading = false);
          },
          onWebResourceError: (WebResourceError error) {
            // Handle error
            print('WebView error: ${error.description}');
          },
        ),
      );
  }
}

```

```

    },
  ),
)
..loadRequest(Uri.parse(widget.redirectUrl));
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text('Complete Task'),
      actions: [
        IconButton(
          icon: Icon(Icons.refresh),
          onPressed: () => controller.reload(),
        ),
      ],
    ),
    body: Stack(
      children: [
        WebViewWidget(controller: controller),
        if (isLoading)
          Center(
            child: CircularProgressIndicator(),
          ),
      ],
    ),
  );
}
}

```

### Step 3: Create Task Button/Widget

```

// lib/widgets/task_button.dart
import 'package:flutter/material.dart';
import '../services/earnings_api.dart';
import '../screens/task_webview_screen.dart';

class TaskButton extends StatefulWidget {
  final String productId;
  final String taskUrl;
  final String? offerId;
  final String? token;

  const TaskButton({
    Key? key,
    required this.productId,
    required this.taskUrl,
    this.offerId,
    this.token,
  }) : super(key: key);
}

```

```
@override
State<TaskButton> createState() => _TaskButtonState();
}
```

```
class _TaskButtonState extends State<TaskButton> {
  bool isLoading = false;

  Future<void> _startTask() async {
    if (widget.token == null) {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text('Please login to continue')),
      );
      return;
    }

    setState(() => isLoading = true);

    try {
      final api = EarningsApi(widget.token);

      // Generate click and get tracking URL
      final clickResponse = await api.generateClick(
        productId: widget.productId,
        taskUrl: widget.taskUrl,
        offerId: widget.offerId,
      );

      // Optional: Track click for analytics
      await api.trackClick(clickResponse.clickId);

      // Navigate to WebView with tracking URL
      if (mounted) {
        Navigator.push(
          context,
          MaterialPageRoute(
            builder: (context) => TaskWebViewScreen(
              redirectUrl: clickResponse.redirectUrl,
              clickId: clickResponse.clickId,
            ),
          ),
        );
      }
    } catch (e) {
      if (mounted) {
        ScaffoldMessenger.of(context).showSnackBar(
          SnackBar(content: Text('Error: $e')),
        );
      }
    } finally {
      if (mounted) {
        setState(() => isLoading = false);
      }
    }
  }
}
```

```

    }
  }
}

@override
Widget build(BuildContext context) {
  return ElevatedButton(
    onPressed: isLoading ? null : _startTask,
    child: isLoading
      ? SizedBox(
        width: 20,
        height: 20,
        child: CircularProgressIndicator(strokeWidth: 2),
      )
      : Text('Start Task'),
  );
}
}

```

#### Step 4: Use in Your App

```

// In your product/offer detail screen
TaskButton(
  productId: product.id,
  taskUrl: product.taskUrl, // URL from product/offer data
  offerId: offer?.id, // Optional
  token: userToken, // User's auth token
)

```

## Testing

#### Test Click Generation

```

curl -X POST http://localhost:3000/api/v1/earn/products/PRODUCT_ID/click \
-H "Authorization: Bearer YOUR_TOKEN" \
-H "Content-Type: application/json" \
-d '{
  "taskUrl": "https://example.com/offer"
}'

```

#### Test Postback (Simulate Advertiser)

```
# Get clickId from generateClick response
curl -X POST "http://localhost:3000/api/v1/earn/postback?click_id=YOUR_CLICK_ID" \
-H "Content-Type: application/json" \
-d '{
  "amount": 50.00,
  "status": "completed",
  "transactionId": "test_txn_123",
  "conversionId": "test_conv_456"
}'
```

## Database Schema

### ClickLog Collection

```
{
  clickId: string (UUID v4, unique),
  userId: ObjectId,
  productId: ObjectId,
  offerId?: ObjectId,
  taskUrl: string,
  redirectUrl: string,
  ipAddress: string,
  userAgent: string,
  referrer?: string,
  clickedAt: Date,
  expiresAt: Date,
  status: 'pending' | 'converted' | 'expired' | 'rejected',
  conversionId?: string,
  postbackReceived: boolean,
  postbackReceivedAt?: Date,
  createdAt: Date,
  updatedAt: Date
}
```

### Earning Collection

```
{
  userId: ObjectId,
  productId: ObjectId,
  offerId?: ObjectId,
  applicationId?: ObjectId,
  clickId: string, // Links to ClickLog
  conversionId: string, // From postback
  amount: number,
  status: 'pending' | 'completed' | 'cancelled',
  type: string,
  earnedAt: Date,
  creditedAt?: Date,
  approvalStatus: 'pending' | 'auto_approved' | 'manual_approved' | 'rejected',
  postbackReceived: boolean,
  postbackReceivedAt?: Date,
  postbackData?: object, // Raw postback data
  // ... other fields
}
```

---

## Security Considerations

1. **Postback Endpoint:** Public endpoint but validates clickId to prevent unauthorized credits
2. **Click Expiration:** Clicks expire after 24 hours to prevent old conversions
3. **Duplicate Prevention:** System checks if conversion already exists using clickId and conversionId
4. **Status Validation:** Prevents processing already converted/rejected clicks
5. **Amount Validation:** Uses postback amount or product's earnUpTo value

---

## Best Practices

1. **Always use redirectUrl:** Never send users directly to taskUrl - always use the redirectUrl from the API response
2. **Store clickId:** Save clickId locally if you need to check status later
3. **Handle expiration:** Inform users if a click has expired before they start
4. **Error handling:** Implement proper error handling for network issues
5. **Loading states:** Show loading indicators while generating click
6. **WebView settings:** Configure WebView properly (JavaScript enabled, cookies, etc.)

---

## Troubleshooting

### Click not generating

- Check authentication token is valid
- Verify productId exists
- Ensure taskUrl is a valid URL

### Postback not received

- Verify advertiser is calling the correct postback\_url

- Check click\_id parameter is included
- Verify click hasn't expired (24 hour limit)
- Check server logs for errors

### **User not credited**

- Check if postback was received (check postbackReceived in ClickLog)
  - Verify status in postback is 'completed' or 'approved'
  - Check if click was already converted (duplicate prevention)
- 

## **Support**

For issues or questions, check:

- Server logs for postback errors
- ClickLog collection for click status
- Earning collection for credit records