Grandstream Networks, Inc.

WP856 NetSuite ERP Demo Source-code Guide

Overview

This guide provides a structured breakdown of the NetSuite ERP integration demo application. The application demonstrates authentication via OAuth 2.0 and supports limited inventory transfer functionalities. Developers can modify and expand upon this base to fit their organization's needs.

Prerequisites

Before setting up the NetSuite ERP application, ensure you have the following:

- Netsuite ERP demo source code package. (Click on this link for download)
- A NetSuite account with admin access.
- API access enabled in your NetSuite account.
- Basic knowledge of Android development and Kotlin.
- Android Studio installed on your development machine.
- Retrofit and OkHttp libraries included in your project dependencies.

Configuration & Setup

NetSuite Platform Configuration

Enable Required Features

1. Navigate to **NetSuite** → **Setup** → **Company** → **Enable Features**.
2. Under the **SuiteCloud** tab, enable **OAuth 2.0** authentication.
3. Refer to the official documentation for more details.

Create an Integration for the App

1. Go to **Setup** → **Integration** → **Manage Integrations**.
2. Create a new integration and configure it for **OAuth 2.0**.
3. Check official documentation for guidance.

Android Project Configuration

Android Studio will be used extensively in this section for editing project files, implementing the authentication logic, and debugging the application.

Update build.gradle

Modify the *build.gradle* file to include **NetSuite API** authentication details:

```
android {
    android.buildFeatures.buildConfig = true

    buildTypes {
        // The CONSUMER_KEY and CONSUMER_SECRET of the integration (They only appear when you create the integration)
        val NETSUITE_CONSUMER_KEY = "XXXXXX" // Replace with your consumer key
        val NETSUITE_CONSUMER_SECRET = "XXXXXX" // Replace with your consumer secret
        // The company's Account ID: Such as "11111_DEMO1"
        val NETSUITE_ACCOUNT_ID = "XXXXXX" // Replace with your account ID
        // The company's Account ID in the URL of NetSuite API: Such as "11111-demo1", "https://11111-demo1.suitetalk.api.netsuite.com
        val NETSUITE_ACCOUNT_ID_IN_URL = "XXXXXX" // Replace with your account ID
        // The callback URI for the login authorization with OAuth 2.0:
        // 1. The authorization sign-in result is returned to the app via this URI.
        // 2. The app needs to add the corresponding intent-filter in the AndroidManifest.xml for the Activity that receives the login authorization result:
        //    Such as, "xxx://xxx/xxx" (grandstream://com.exaple/login/callback)
        val NETSUITE_REDIRECT_URI = "XXXXXX" // Replace with your redirect url

        debug {
            buildConfigField("String", "NETSUITE_CONSUMER_KEY", "\"$NETSUITE_CONSUMER_KEY\"")
            buildConfigField("String", "NETSUITE_CONSUMER_SECRET", "\"$NETSUITE_CONSUMER_SECRET\"")
            buildConfigField("String", "NETSUITE_ACCOUNT_ID", "\"$NETSUITE_ACCOUNT_ID\"")
            buildConfigField("String", "NETSUITE_ACCOUNT_ID_IN_URL", "\"$NETSUITE_ACCOUNT_ID_IN_URL\"")
            buildConfigField("String", "NETSUITE_REDIRECT_URI", "\"$NETSUITE_REDIRECT_URI\"")
        }
        release {
            isMinifyEnabled = false
            proguardFiles(
                    getDefaultProguardFile("proguard-android-optimize.txt"),
                    "proguard-rules.pro"
            )
            signingConfig = signingConfigs.getByName("release")

            buildConfigField("String", "NETSUITE_CONSUMER_KEY", "\"$NETSUITE_CONSUMER_KEY\"")
            buildConfigField("String", "NETSUITE_CONSUMER_SECRET", "\"$NETSUITE_CONSUMER_SECRET\"")
            buildConfigField("String", "NETSUITE_ACCOUNT_ID", "\"$NETSUITE_ACCOUNT_ID\"")
            buildConfigField("String", "NETSUITE_ACCOUNT_ID_IN_URL", "\"$NETSUITE_ACCOUNT_ID_IN_URL\"")
            buildConfigField("String", "NETSUITE_REDIRECT_URI", "\"$NETSUITE_REDIRECT_URI\"")
        }
    }
}
```

Register OAuth Redirect URI in AndroidManifest.xml

Ensure the app properly handles OAuth callbacks by adding an intent-filter:

```
<intent-filter>
    <action android:name="android.intent.action.VIEW" />
    <category android:name="android.intent.category.DEFAULT" />
    <category android:name="android.intent.category.BROWSABLE" />
    <data
        android:host="com.exaple"
        android:path="/login/callback"
        android:scheme="grandstream" />
</intent-filter>
```

Network Communication

The application uses **RetrofitClient** for API communication.

Define NetSuite API Service

```
interface NetsuiteServiceApi {
    // Get the access token
    @FormUrlEncoded
    @POST(NetworkConfig.OAUTH2_STEP2_TOKEN_PATH)
    fun getToken(
        @Header(NetworkConfig.HEADER_AUTHORIZATION) authorization: String,
        @Field(NetworkConfig.OAUTH2_STEP2_CODE) code: String,
        @Field(NetworkConfig.OAUTH2_STEP2_REDIRECT_URI_KEY) redirectUri: String,
        @Field(NetworkConfig.OAUTH2_STEP2_GRANT_TYPE_KEY) grantType: String,
        @Field(NetworkConfig.OAUTH2_STEP2_CODE_VERIFIER_KEY) codeVerifier: String
    ): Call<OAuth2Token>
}
```

Implement Retrofit Client

```
object RetrofitClient {
    // network timeout
    private const val TIME_OUT = 20

    // gson
    private val gson = Gson().newBuilder().setLenient().serializeNulls().create()

    // network api: Setting base url by BuildConfig
    val api by lazy { getService(NetsuiteServiceApi::class.java, String.format(NetworkConfig.BASE_REST_URL,
BuildConfig.NETSUITE_ACCOUNT_ID_IN_URL)) }

    private val client: OkHttpClient
        get() {
            val builder = OkHttpClient.Builder()
            builder
                .sslSocketFactory(SSLSocketManager.sslSocketFactory, SSLSocketManager.x509TrustManager!!)
                .hostnameVerifier(SSLSocketManager.hostnameVerifier)
                .connectTimeout(TIME_OUT.toLong(), TimeUnit.SECONDS)
                .writeTimeout(TIME_OUT.toLong(), TimeUnit.SECONDS)
                .readTimeout(TIME_OUT.toLong(), TimeUnit.SECONDS)
            return builder.build()
        }

    // Create Retrofit service instance dynamically
    private fun <S> getService(serviceClass: Class<S>, baseUrl: String): S {
        return Retrofit.Builder()
            .callFactory(TrackCallFactory(client))
            .addConverterFactory(GsonConverterFactory.create(gson))
            .baseUrl(baseUrl)
            .build()
            .create(serviceClass)
    }
}
```

Making API Calls

```
fun getToken(authorize: OAuth2Authorize, codeVerify: String): Call<OAuth2Token> {
    return RetrofitClient.api.getToken(
        Credentials.basic(BuildConfig.NETSUITE_CONSUMER_KEY, BuildConfig.NETSUITE_CONSUMER_SECRET), // Basic
auth header
        authorize.code, // Authorization code received from login
        BuildConfig.NETSUITE_REDIRECT_URI, // Redirect URI for callback
        NetworkConfig.OAUTH2_STEP2_GRANT_TYPE_VALUE, // Grant type for OAuth2
        codeVerify // Code verifier for PKCE
    )
}
```