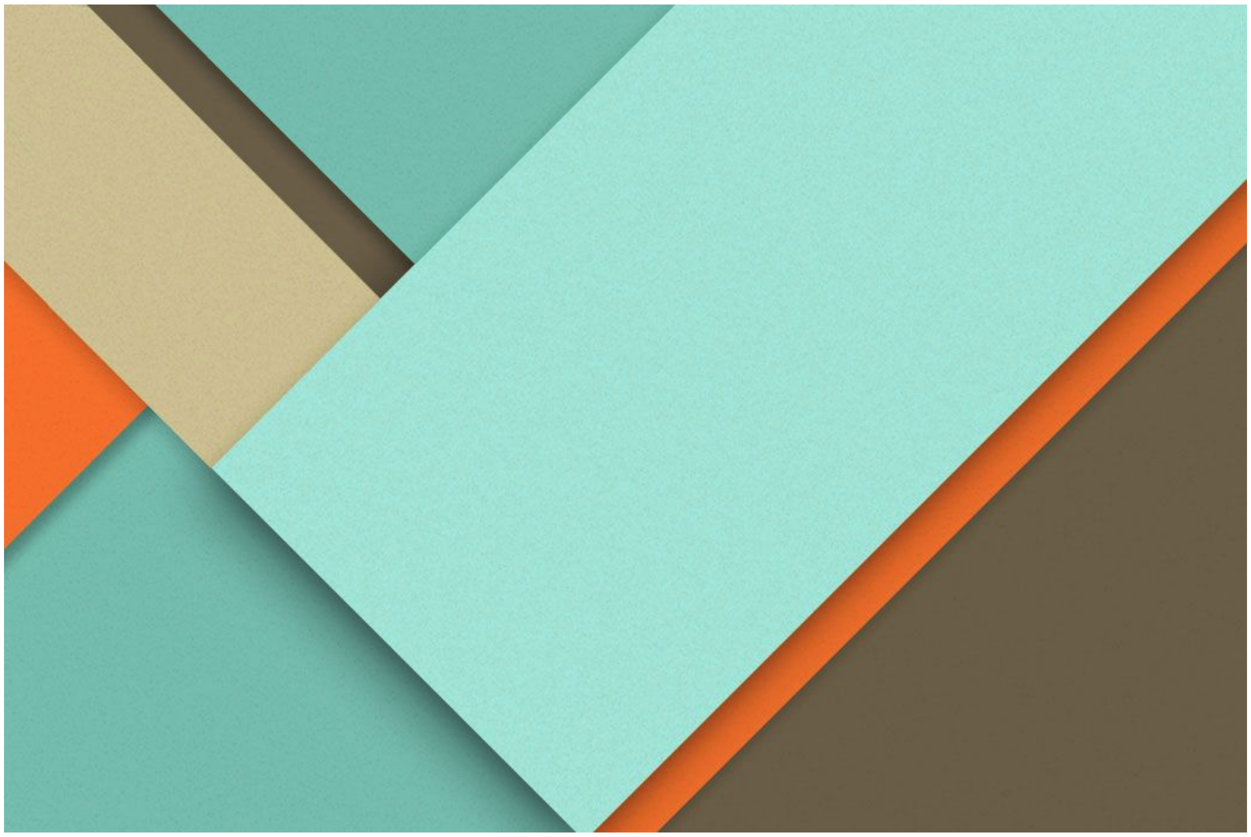

Freibier Android POS

TECHNICAL MANUAL

28.04.2017

Diego Ruiz
Bx Service GmbH

Table of Contents

1. Overview

Freibier Pos is an android app that allows to create orders in restaurants easily in iDempiere. It communicates with iDempiere using the web services included by default in iDempiere.

Available languages:

- English
- Spanish
- German

Tested devices:

- Nexus 4
- Motorola Moto G (3rd Gen)
- Acer Iconia Tab10
- Printer Zebra MZ320

2. Prerequisites

1. Install AndroidStudio >=1.5.1
2. Download the source code from here
3. Having an android device with android API version 21 or above. (Lollipop / 5)
4. A running iDempiere server

3. How to get Freibier POS running

Download the source code from bitbucket and import it in Android Studio. Once the project had already been imported successfully to Android Studio, you can connect your device to the usb port and install the app in the device.

You already have the app in your device. However to be able to login and use the app properly, you must do the following:

Install the push notification plugin in iDempiere

Install the FCM-Server plugin in iDempiere, you can find it here. This plugin allows the user to do two things:

- Send an update request to the registered devices with the "Update Request Notification" process.
- Synchronize table status among multiple devices. Once a tables is occupied or busy, it replicates the status change to all the connected devices.

If you want to use the push notification functionalities, you must create a Sysconfig in iDempiere named **BXS_POS_APIKEY** with the api key of the app (you can get it following these steps).

The plugin also creates the necessary elements in iDempiere to be able to run the pos (tables, views, trees ...).

Import the necessary web service security records in iDempiere

Freibier POS communicates to iDempiere via web services, therefore, in the server you have to allow the web services to execute. Import the pack in, containing the web service security records:

- QueryProductCategory
- QueryProduct
- QueryTable
- QueryProductPrice
- TestLogin
- CompositeCreateOrder
- CreateSalesOrder
- CreateSalesOrderLine
- DocActionOrder
- QueryPOSData
- CreatePosPayment
- CreateDeviceToken
- UpdateBXSTable
- QueryTenderType

Import the new elements in iDempiere

Freibier POS uses new things to get the right data, please import the pack in that imports those elements into your iDempiere client, the pack in will create the following:

- Table Tree (Garden)
- TenderTypes (Garden)

All of this can be done by importing in Garden the freibierPosClientSetup.zip pack out.

Configure the data in iDempiere

The app reads all the data from iDempiere, however, it must be configured in a special way to show only the data related to the POS and avoid to show wrong data or unnecessary data. The way to create these data in iDempiere is explained in the user manual, please refer to it.

4. How to maintain the multi-language

The app will be shown in the phone's language (German, english, spanish). When you add a new string to the code, make sure that it exist in the three string.xml files.

5. Project structure

As every android app the project is divided in two main folders: java and res. The java folder containing all the classes used in the project, and the res folder containing all the resource files used in the app, such as: images, xml files.

5.1 Java structure

The java part of the project is divided in three main layers: logic, persistence and ui. Logic having the model layer, persistence the layer that communicates with the database, and ui is the view layer in charge of managing the user interface objects. If you want to add code to the project, please keep in mind this organization to keep a clean code and avoid cohesion. Every layer is subdivided in different packages as well.

6. Reports in the app

The app creates reports to show the user meaningful information. The app is designed to be able to evolve with the time, allowing the creation of further reports.

6.1 How to add a new report?

To add a new report there are some things you need to consider, these steps are as follows:

1. Create the string resource and string value: in the strings.xml file (including the multi languages), add the new report name to the string array ***report_types_titles*** and its respective value in ***report_types_values***.
2. Create the corresponding model class: create the class that will manage the report in the package de.bxservice.bxpos.logic.model.report, the new model class must extend the class Report. The new class will bring the desired data for the report, thus, you must create also the corresponding methods to read the data from the database
3. Set the html code that will be displayed in the report: the class ReportHtmlTemplate provides a standard way to create the html code, it provides methods to: create the title, create rows, create tables with n number of rows. The class creates generic code, you just need to replace the content tags to display the right data. The use of this class allows a generic report layout across the different reports. (see code example in the classes: SalesReport - VoidItemReport).
4. Subscribe the report in the ReportFactory class: in the class de.bxservice.bxpos.logic.model.report.ReportFactory, add the corresponding code that you added as a value, and add the creation of the Report class in the switch code segment.