ANTHONY DEBRUYN

Year Project

Wysiwyd

Wine cellar management,
almost without management

Gaëtan Podevijn PROJ-H-402

Contents

1 Introduction

Managing something composed of a large number of element can be tedious and complicated, event when you love doing it. This applies to oenologists as well when it comes to organising their wine cellar. This is where Wysiwyd enters in the game. The goal of Wysiwyd is to simplify the management of a wine cellar, to allow the oenologists and others to get the most of it.

Wysiwyd takes advantages of the new technologies available on the market, such as QR Code scanning and generating, NFC tags, to sort bottles. Photos of the bottles can be captured, so the user can see them right in the app as well. A search can be made on a lot of bottle properties to retrieve a desired bottle. The target OS is Android, leader on the smartphone market. The combination of the discovery of those technologies, the world of Android, the experience gained in mobile programming and design, is what makes this project fascinating.

2 The App

"What you scan is what you drink."
"Wysiwyd"

2.1 Functionalities

2.1.1 App Tour

Once the app is loaded, the user has to choose between 2 actions. To *scan a tag or a visual code*, or to *list directly all the bottles*. Those 2 actions are symbolised by 2 bottles on the screen (see fig **??**).

Figure 1: The Main Screen.

If the user touches the first bottle, a new choice pops up: *visual code*, or *NFC tag* scan. This time, it is

represented by glasses. The first one opens another app[1] to scan a bottle bar code, or QR code[2]. If the user does not have this separate application, he can download it on the playstore for free. This app is not mandatory for this one to work. However, it is highly recommended, as it leads to easier management of the cellar.

Behind the glasses, and if the app has at least one image of a bottle, is a blurred picture of a bottle. This picture is one that the user took to illustrate his mobile database.

Figure 2: Visual Code, or NFC Tag ?

By choosing the second glass, the user arrives in a third screen, only if NFC is available on the device (see fig. **??**). The NFC is in foreground mode as long as the user stays in this view. This means that no other app can intercept the NFC tag scan. Of course, if the user is not viewing this screen, the system still can intercept the tag if he places a bottle NFC tag near the NFC ship of the device. Then the system launches the app and directly lists the bottles with the corresponding code.

[1]Barcode Scanner, by ZXing Team
[2]This visual code is the traduction of the number just below the bar code on wine bottles.

Figure 3: NFC Tag Scan.

When the code has been captured, one way or another, the app shows a list of the corresponding bottles (fig. **??**). If the user decides to add a bottle, the code will be copied in the corresponding text field automatically. If on figure **??** the user selected the second bottle, the screen on figure **??** opens directly, displaying all the bottles in the order they were added to the database.

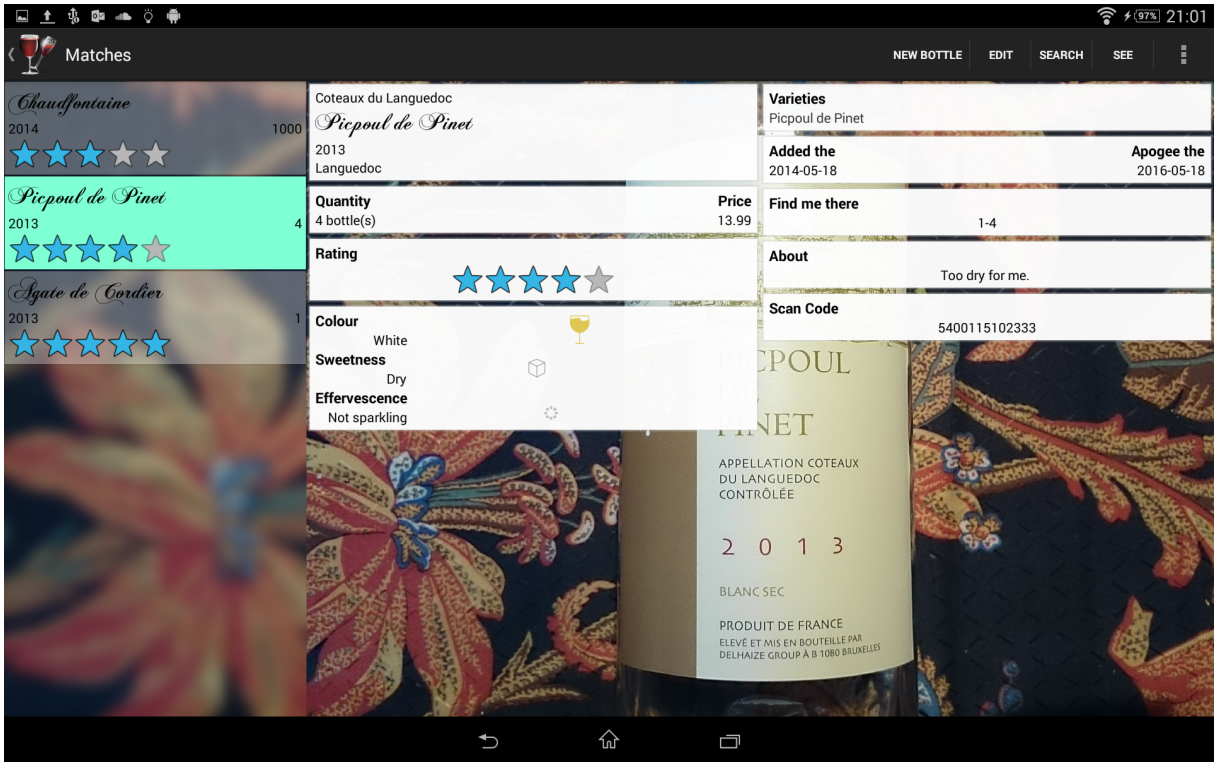


Figure 4: List of results.

On this figure, one can see all the details about a bottle. This is the view shown on a large screen device. On smaller devices, only one view is displayed at a time, either the details or the list.

Several buttons are aligned at the top right of the screen:

New Bottle: To create a new bottle. Fill all the necessary text fields and other information, and click on *Done*.

Edit: To edit an existing bottle. Click on *Done* when finished.

Search: To search for bottles. Write down the desired properties, and click on *Ok*. The list will refresh to display only the matching bottles.

See: To see the captured image of the bottle lying behind the view. This hides the details to show the background.

A *long touch on the code section* in the details will open the code exportation functionality. It will allow you to export the code to a QR code or/and to NFC tag. The layout of the corresponding view is similar to the figure **??**. To generate a QR code and send it, the user is supposed to click on the "QR glass", while to write a NFC tag, the user has to push the "NFC glass". After generating the QR code, one can send it to a web service like Drive from Google, or email it.

2.1.2 Adding a Bottle

There are several ways to add a bottle in the database:

1. The user *can touch the "search bottle"* on the main screen (fig. **??**), then *touch New Bottle*. This is the preferred method if no visual code is on the bottle, like a bar code or a QR code. If the user wants to add a bottle that has a lot in common with another saved bottle in the database, he can just *keep his finger on that bottle* for a while. After a short time, the new bottle view will appear with all the fields already completed with the information from the previous bottle.

2. If the bottle has a bar/QR code, then the user is advised to go with the scan solution. After the "scan bottle" has been touched, he then chooses the QR/bar code scan. The scan app will open, or a pop up will offer to download it on the Playstore if it is not on the phone. He can now *aim at the printed code* on the bottle with the camera of the device, and wait for the code to be recognised. If the device has no camera, this feature will not be accessible.

2.1.3 Finding a bottle

To retrieve a bottle in the database, one can choose between different methods:

1. If the code of the bottle has been correctly entered in the database, and a NFC tag and/or a bar/QR code is on the bottle, the best option is to *choose the first action* on figure **??**. Then another choice is available to the user to choose between the 2 technologies offered with the app.

2. If no code/tag is on the bottle, or if the user wants to retrieve a bigger amount of bottles matching certain properties, the second action of figure **??** is the best. In the list screen, one can open the search view by *touching Search*.

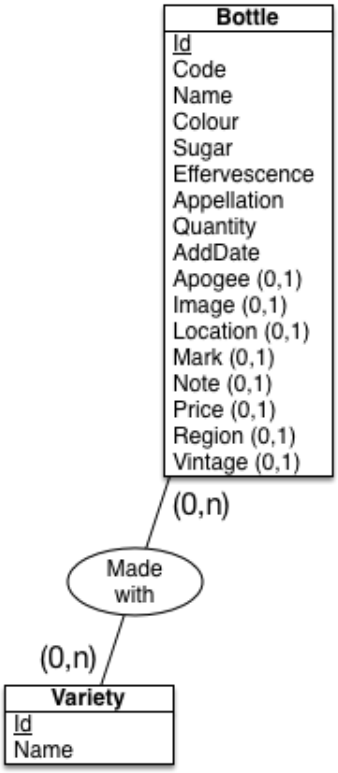
3 The Architecture

3.1 The Database

The database used in this project is made with SQLite. The global architecture of the database is simple, as we can see on the entity relationship diagram.

3.1.1 The Entity Relationship Diagram

Here is the ERD. The diagram contains 2 entities, linked with a relation. All attributes characterising one bottle are in the Bottle entity, they are not represented by separate entities. This would result in more tables in the database, and more resource consuming queries. Having more tables for these attributes would have been useful if these attributes were changed a lot by the user and if a lot of bottles shared same values for them. Here we supposed that the user will not change these values a lot after entering the bottle information, as the information associated to a bottle of wine rarely change in the market.



Relationship DB diagram.png

Figure 5: The ERD. Key is underlined, and optional attributes are together with a (0,1).

Constraints

- If the effervescence is 0, then the vintage attribute is mandatory.
- Id, Code, Quantity, Vintage are a natural numbers.
- Price is a positive real number.
- Colour is a natural number in $\{0, 1, 2\}$.
- Sugar is a natural number in $\{0, 1, 2, 3\}$.
- Effervescence is a natural number in $\{0, 1, 2, 3\}$.
- Mark is a natural number in $\{1, 2, 3, 4, 5\}$.

3.1.2 The Relational Model

The translation of the ENR into the relational model:

Bottle (Id, Code, Name, Colour, Sugar, Effervescence, Appellation, Quantity, AddDate, *Apogee*, *Image*, *Location*, *Mark*, *Note*, *Price*, *Region*, *Vintage*)

Variety (Id, Name)

BottleVarieties (Bottle Id, Variety Id)

BottleVarieties.Bottle Id foreign key to *Bottle.Id*
BottleVarieties.IdVariety foreign key to *Variety.Id*

Constraints

- If the type is "Champagne", then the Year attribute is not mandatory.
- The AddDate must be $\leq$ the current date.
- Id, Code, Quantity, Vintage are a natural numbers.
- Price is a positive real number.
- Colour is a natural number in $\{0, 1, 2\}$.
- Sugar is a natural number in $\{0, 1, 2, 3\}$.
- Effervescence is a natural number in $\{0, 1, 2, 3\}$.
- Mark is a natural number in $\{1, 2, 3, 4, 5\}$.
- For each Bottle Id in BottleVariety, there exists an Id in Bottle.
- For each Variety Id in BottleVariety, there exists an Id in Variety.

Remark Numbers are given to colours, varieties, etc. to allow the use of string resources. This is recommended by the Android best practices, and allows to create multiple versions of the string resources for multiple languages.

3.1.3 Normalisation

No normalisation was necessary.

3.2 The Code

The implementation of the app was made by trying to respect the best practices in Android application development. All information concerning the UI was stored in xml files (layout files). The hole architecture was thought in an optimisation and object oriented perspective, by taking advantages of the given Android callbacks and life cycle methods.

The code is split in several packages. Each package providing a functionality. These packages are described after. Here is the class diagram of the code:

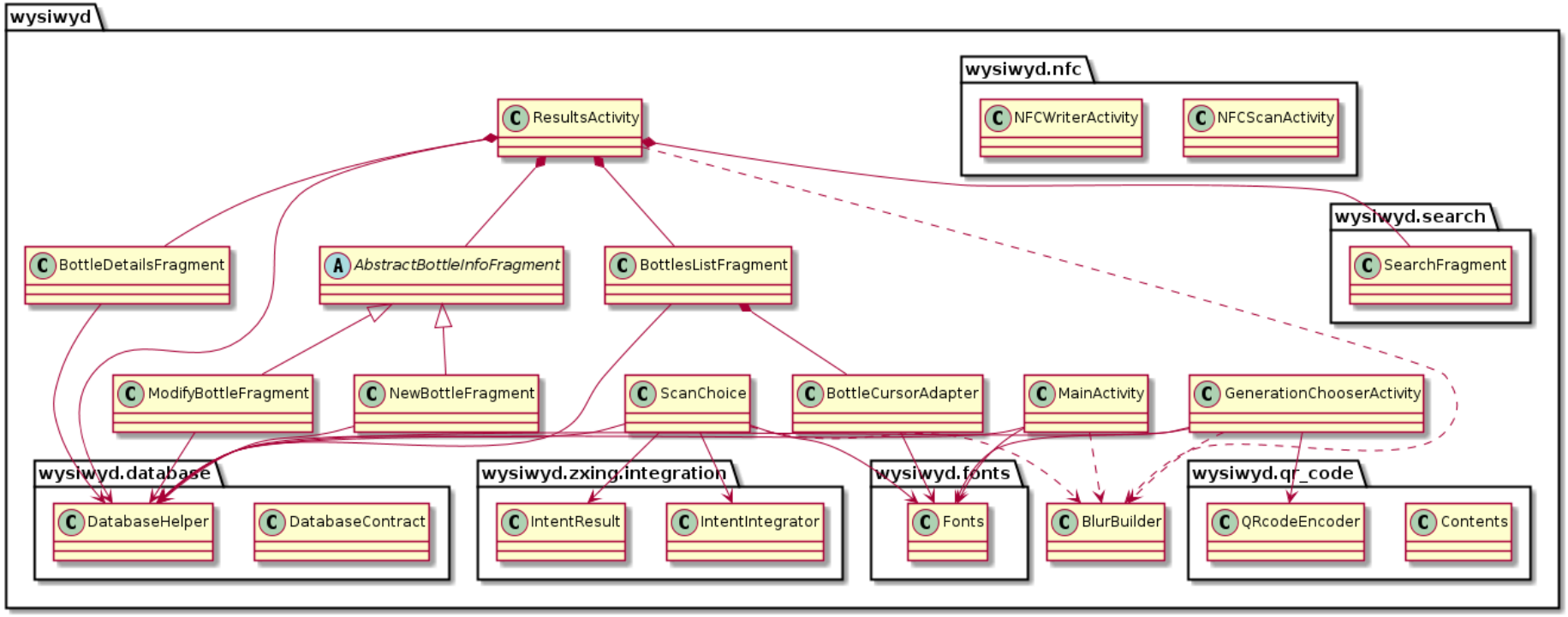


Figure 6: The Class Diagram of the code architecture.

3.2.1 Root package (wysiwyd)

The root package (root means the highest level of the src packages hierarchy) contains the main navigation Android activities, from the welcome screen activity to the export chooser for the code exportation. Each situation in the app is associated with another activity in the code.

The results activity, the one that shows the list of bottles is special in the sense that it uses fragments. This allows a better modularisation of the code, and easier management of the view. Indeed, 2 modes are currently available in the app, depending on what type of device the app is running on. If the app is run on a large screen device, then the app will use a different layout to take advantage of the screen. 2 fragments will be displayed at the same time: the list and the details/modification/new bottle/search fragment.

To be able to communicate with their parent activities, each fragment defines an interface. This interface must be implement by the interface, and this is checked by the fragment each time the fragment is attached to the activity, in the life cycle callback *onAttach*. This interface contains the callbacks methods that are needed by the fragment when it needs to send a message to the activity. As an example, the search fragment has a callback method to give the activity a Bundle object containing all the information needed to build the search query. That way the activity can contact the list fragment, so this latter can build and execute the query.

All the information displaying fragment, like the modifying bottle fragment, use callbacks method to save their current state when it is destroyed (not explicitly by the user). This helps reduce resources consumption. It is useful in cases like screen rotation.

The use of an abstract class, AbstractBottleInfoFragment, was decided to factorise the code, and take advantage of the polymorphism. This class is inherited by NewBottleFragment and ModifyBottleFragment. These 2 classes have a lot in common, since they both offer tools to specify information on a bottle.

3.2.2 Database Package (wysiwyd.database)

The database package contains 2 classes:

The Contract Class This class contains the structure of the database, the constants used in other places in the project to build queries, and the strings for the scripts to create the tables.

The Database Helper Class This class inherits from SQLiteOpenHelper from the Android framework, and is used to get access to the database. A singleton pattern is used in the class to only have one instance of this class created the hole time. This guaranties that no leaks will occur if the access is asked several times. This class also deals with the creation of the database and its upgrade.

3.2.3 Fonts Package (wysiwyd.fonts

Only one class is in this package, and it is used to get access to fonts that are not in the Android framework. A singleton pattern is used for the class, to guarantee that only a single instance will be made.

3.2.4 NFC Package (wysiwyd.nfc)

It contains the classes needed to communicate with NFC tags. One class to read and one to write, 2 Android activities. They inherit from classes in an external library (NDEF Tools for Android, see **??**).

3.2.5 QR Code Package (wysiwyd.qr_code)

Two classes from an external library populate this package (ZXing, see **??**). These classes are used to communicate with the ZXing library to generate QR codes.

3.2.6 Search Package (wysiwyd.search)

Only one class: the search fragment, displayed when the user wants to find a bottle with some properties.

3.2.7 ZXing Integrator Package (zxing.integration)

Rather than importing all the code needed to scan visual codes, like QR/bar codes, these 2 classes look for an external application, made by the ZXing developers. This external application is launched each time a scan is needed, and return the result to the calling activity. This way of implementing the code was chosen over the hole importation, since it is easier, and guarantees the last version of the scanning functionality code at all times.

3.3 Used Libraries

CWAC LoaderEx (commonsguy) Loaders are a way to load information in a separate thread, and not in the UI thread to avoid lag in the interface. Since the use of loaders was encouraged by the Android best practices, searches were made in that direction. Apparently, the use of loaders with SQLite queries was depreciated, and only their use with content providers was possible. Content providers are useful when it comes to data interaction between applications, and seemed a bit overkill in this app. So an external library was used to compensate this: CWAC LoaderEx from commonsguy. See `https://github.com/commonsguy/cwac-loaderex`.

NDEF Tools for Android This library offers high level programming for NFC communications, as the Android framework does not provide it. This made the implementation of the NFC communication easier, and faster. See `https://code.google.com/p/ndef-tools-for-android/`.

ZXing This is an open-source, multi-format barcode image processing library. Barcodes can also be generated. It was used for the bar code scanning and generating functionalities. See `https://github.com/zxing/zxing/`.

4 Difficulties

Here are explained the main difficulties met during the implementation of the app.

Layout Creation One of the biggest difficulties encountered was to find the good adjustments for the layout parameters. These parameters, like the weight of a view, the scaling type for an image, the size of views, have to be correctly defined to obtain the desired layout. In xml layout files, like the one for the scan choice activity, the bottle details fragment, this definition of values was made through a process of trial and errors, trying values that theoretically would lead to success. In these files, the nesting of layout made the job more complicated.

Use of 2 Different Screen Sizes The use of 2 different screen sizes led to addition of code in the results activity. Tests to know in what size it was were added to behave differently. Indeed, different layout files have to be loaded and, in large screen size, some buttons have to be hidden manually. In single pane mode (small screens), these buttons are hidden because the fragment associated to them is hidden. This dual mode also caused problems with the rotation of the screen, since an activity is re-created each time the device configuration changes. This was discovered after some research on the problem, and solutions like using the state saving callback method were exploited.

Finding Help A lot of search was made to find solutions to the problems found while coding. Some of them did not take long, but others did. Sometimes some solutions had to be mixed to create an adapted solution. See section **??**.

Android The constant evolution of the mobile OS framework causes some implementations to become depreciated. Alternatives had to be found. External libraries were used, like CWAC LoaderEx, to compensate.

5 Some Ideas for the Future

Cloud Database The current database is local, on the device itself. Putting the database in the cloud would allow the user to use different devices to manage his wine cellar, these devices being synchronised. The user could also share its cellar content with other people. This could be used by shops to offer another view on their stock.

Notification System A notification system would alert the user that some bottles are going to enter apogee. Another filter could also be based on the apogee criterion in the sorting preference for the results list.

6 References

Here are some links that helped implementing the app by offering solutions:

6.1 General

- `http://developer.android.com/training/index.html`
  The developer training.
- `http://stackoverflow.com/questions/3769762/android-color-xml-resource-file`
  Contains a lot of predined colors.
- `http://developer.android.com/guide/index.html`
  Android guide.
- `http://stackoverflow.com/questions/2025282/difference-between-px-dp-dip-and-sp-in-android`
  To get the different measure units.