Mobile Computing

Practical Assignment #1 / Design and Development

Order and pay Android app for a cafeteria franchising

The Acme Café Order and Payment System

1. Scenario

A big cafeteria franchising – the Acme Café – have several cafeterias in several cities and intends to implement a more efficient ordering and payment system supplying an Android app to its customers.

The idea is to the customers compose previously their orders in the app, choosing items from the menu and quantities, and transmit them, together with possible vouchers and identification data, to a terminal inside the house. After that, the customers only need to collect the ordered items at the counter when they are ready.

Ordering and payment terminal

Cafeteria counter

Using the app the customers should first make a registration (only once when they use the app for the first time) in the franchising remote service, supplying some personal data and credit card data for payments.

The available items in the cafeterias can be obtained from the remote service at any time, as well as past transactions information and available emitted vouchers. This loyalty vouchers are offered

whenever a single order surpasses a certain amount or the customer accumulated orders value surpasses a multiple of another value.

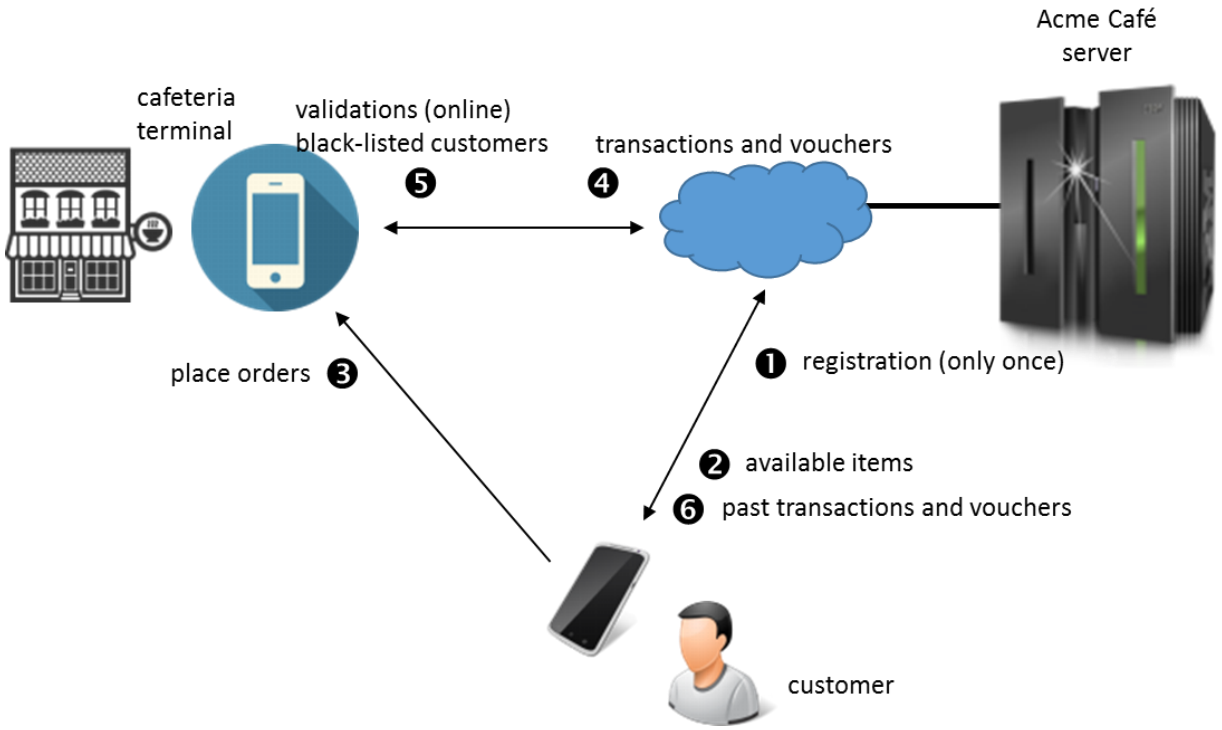
2. System applications

The ordering and payment system is composed of three different applications, namely:

1. The remote service (a REST service) located on the company server (it can be divided into several groups of operations: register and validate customers, emit and validate vouchers and perform and consult transactions).
2. The Customer App allowing him to register himself in the system, consult menus and prices, compose an order to the cafeteria eventually selecting and use gift vouchers, and also consult past transactions and available vouchers.
3. For this experience the cafeteria order terminal runs also an Android application receiving from the customer app the ordered quantities of available products and at most **three** vouchers used in the price calculation. After validation of the vouchers the terminal should show an order number (in big characters), products ordered, vouchers accepted and final price paid. The customer then collects the products at the cafeteria counter stating the order number.

3. Operations and interactions

The minimum set of operations and interactions between these software applications are depicted in the following diagram:



They should perform, at least, the following:

1. Registration - the first operation the customer app should do is to register the customer in the franchising service. The customer should supply his name, a user name, a password and credit card information, at least. If the operation succeeds a **unique** 'user id' (a generated **uuid** value – 16 bytes) and a PIN (4 digits) should be generated and returned to the app. The 'user id' should

be stored locally by the app (the value is not shown to the user) and the PIN should be shown (allowing the user remember it), only this time, and also stored. The server takes note of all this information in its database.

2. Get menu of items and prices - this interaction should be transparent to the user. Whenever the user wants to see the menu of available items and their prices or compose a new order, the app should confirm with the server if it already has the last menu. If not, the items are updated (and stored locally) before showing them to the user. For this test implementation consider at least half a dozen items, including **coffee** and **popcorn**.

3. Make and place an order – Whenever the customer wants to take something from a cafeteria, first he should compose an order and select appropriate vouchers if he possesses them, and transmit the order to the cafeteria terminal. The order can contain one or more of the available menu items and their quantities and up to **three** vouchers appropriate to the order. The vouchers can be an offer of a certain item or a discount applied to the total. Only **one** of this last type of voucher can be included. When the order is successfully transmitted, the included vouchers are **deleted** from the customer app storage. Also the transmitted information includes the '**user id**' stored in the customer app. To transmit an order, the app asks the user for the PIN and checks it locally.

   At the cafeteria terminal side, the validity of the vouchers is checked (usually online with the server) and the applicable ones are taken into account for calculating the final total to pay. Non applicable vouchers are ignored. Also the 'user id' is validated with the server (if online). If all is ok an order number is calculated and displayed together with the indication of used vouchers and prices.
   If the company server cannot be reached (which should be an infrequent situation) the vouchers are validated locally and the 'user id' is checked against a locally maintained black list of not accepted users. The transaction details are kept locally until they can be transmitted to the server later.

   The server should refuse customers and black list them if they have expired or invalid credit cards, or present false vouchers not emitted to them. The black list of clients is transmitted to the cafeteria terminals whenever there are changes.

   Voucher format and validation schema is explained later.

4. Check an order - The cafeteria terminals, by norm, check the orders with the company server, verifying the 'user id' and the validity of the appropriate vouchers. If the credit card info is correct (not expired) we can assume that the server performs the payment using the user credit card information with a third part banking service. If the credit card is invalid or a 'false' voucher is used the customer is black-listed and should not be further accepted. The result of the check and possible black-list updates are transmitted back to the terminal.
   If the server is not online at the time, the verifications described in 3. are performed locally. The transaction data is also kept locally and transmitted later to the server (and can result then is a black-list update).
   If the order is ok the price, taking into account the appropriate vouchers, is calculated and an order number is generated. All this information is shown at the terminal screen.

5. Validation result and black-list updates – If the previous checking is done online the server replies with the result, price charged, and possible black-list updates (from all transactions already transmitted to the server).

6. At any time, the customer can ask the company server for past transaction data (at least a certain number of those transactions should be transmitted back) using the user name and password (just one time per session). At this time also, the server transmits back all the vouchers that are unused and belonging to the user, that should replace the local list. If the user has used non appropriate vouchers in an order (therefore not effectively consumed) they will be recuperated at this point. Also new vouchers offered are transmitted to the user app.

4. Vouchers

Vouchers are used as a loyalty strategy. They are offered to the customer whenever he makes an order surpassing € 20.00 or whenever the accumulated value of all orders from the customer surpasses a new multiple of € 100.00. In the first case the voucher offers randomly a **coffee** or a **popcorn** pack. In the last case it offers a 5% discount in the total of a new order. All the vouchers can only be used after the order where they were originated, and subject to the maximums already stated.
A voucher is a sequence of bytes containing a unique serial number (can be an integer – 4 bytes), a type (1 byte) and a cryptographic signature calculated at the server using the company private key (46 bytes). Serial numbers should be different but not predictable. Consider, for instance, choosing the first one in the range 1 to 1000 chosen randomly, and then adding an increment from 100 to 500 also chosen randomly for each new emitted voucher.
To verify the signature, the server (or the terminal if the server is offline) should use the company public key which must transmitted and stored by each terminal. The key pair is generated by the server (once) and the private key is kept securely stored there. For the signature keys use the "RSA" algorithm and a length of 368 bits, and for the signature algorithm use "SHA1WithRSA". This produces a signature of 46 bytes which is the shortest with standard algorithms.

5. Communications

All the communications in all the operations, except operation 3., are done using the internet and the http protocol (in a REST service), over Wi-Fi or over the phone operator network. The communication between the customer app and terminal should use NFC or a QR code and camera.
If you have two physical Android phones supporting NFC use it. If not, use the QR code technique.
The QR code technique can also be used between a physical phone and an emulator. QR codes can only represent a small number of bytes, so the information coded should be the minimum possible.
If you don't have any Android physical phone available, you can use a TCP/IP connection between two different emulators.

Note: if you are using only the Google emulators the channel between them has to be TCP. See https://developer.android.com/studio/run/emulator-commandline.html#emulatornetworking and the section "Interconnecting emulator instances" for instructions. With one real phone and an emulator, the emulator should be the user phone presenting the QR code and the real phone can capture it with the camera.

6. Design and development

You should design and implement the set of applications capable of complying with the described functions and demonstrate its use. The applications should have a comfortable and easy to use interface. You can add any functionalities considered convenient and fill any gaps not detailed.

You should also write a report, describing the architecture, data schema, included features and performed tests. The applications way of use should also be included in the report, presenting a screen capture sequence.