Mobile Computing

Practical Assignment #1 / Design and Development

Order Android app for a cafeteria including loyalty campaigns

The Acme Café Order System

1. Scenario

A chain of cafeterias – the Acme Café – intends to implement a more efficient ordering and delivery system, supplying an Android app to its customers and incentivizing them to use it, through a loyalty campaign.

The idea is for the customers compose previously their orders in the app, choosing items from a menu and their 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 discount 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 cafeteria remote service, supplying some personal data (name, credit/debit bank card and NIF (fiscal identification number – 9 digits)).

The available items in the cafeterias (always the same) can be obtained from the remote service at any time, as well as available emitted vouchers to that customer. These loyalty vouchers are offered whenever the customer accumulates purchases of a certain item and/or a total value.

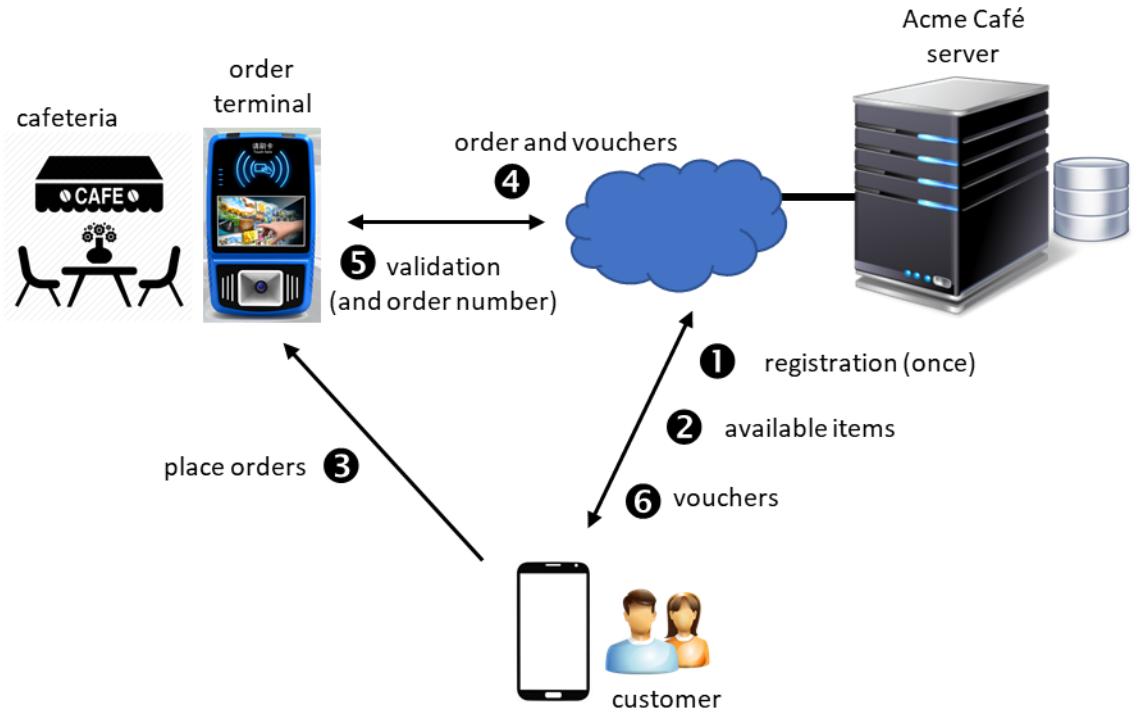
2. System applications

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: e.g., register customers, emit and validate vouchers, calculate an order value, …).
2. The Acme Café App, allowing him to register himself in the system, consult menus and prices, compose an order to the cafeteria eventually selecting and use the gift vouchers, and retrieving newly emitted and available vouchers.
3. For this experience, the cafeteria ordering terminal runs also an Android application, receiving from the customer app the ordered quantities of available products and selected vouchers that can be 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 total price payed. The customer then collects the products at the cafeteria counter, when the order number is called (in a visible display in the cafeteria).

3. Operations and interactions

The minimum set of operations and interactions between these software applications are depicted in the following diagram (you can include more):



They should perform, at least, the following:

1. Registration - the first operation the customer app should do is to register the customer in the cafeteria service. The customer should supply his name, NIF, and credit/debit card information, at least, and also a nick name / password, for local login. Also, the app should generate a cryptographic RSA key pair, and transmit the public key to the server (in the form of a certificate). If the operation succeeds a **unique** 'user id' (a **uuid** value – 16 bytes) should be generated in the server and returned to the app. The 'user id' and generated private key should be stored locally by the app (the values are never shown to the user, and the key should be generated and stored in the Android key store, for safety). The server takes note of the registration 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 only a few items (coffee and other drinks, and foods like pastry and sandwiches), including **coffee**. These items can be represented by an icon, a name, and the current price.

3. Make and place an order – Whenever the customer wants to take something from the 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 some vouchers appropriate to the order. The vouchers can be an offer of a coffee 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. All this info is signed with the user private key before sending, and signature appended.

   At the cafeteria terminal side, the order is sent to the server, where the user is identified, the signature verified, and the validity of the vouchers is checked. The applicable ones are considered for calculating the final total to pay which is done using the user pay information (consider that the server always succeeds in performing that payment). Non-applicable vouchers are ignored but can be retrieved again. Invalid vouchers are ignored and discarded. If all is ok, the terminal calculates and displays an order number (sequential) together with the indication of used vouchers and total price. Otherwise the order is rejected, and that is indicated in the terminal by a very visible, large red icon. The server also generates a receipt and keep it associated with the user.

   Voucher format and validation schema is explained later.

4. Check an order - The cafeteria terminals check the orders with the company server, verifying the 'user id', the validity and applicability of the appropriate vouchers and the user signature.

   If the order is ok, the price, considering the appropriate vouchers, is calculated and an order number is generated. All this information is shown at the terminal screen.

5. Validation result - The server replies with the validation result, accepted vouchers and total value payed.

6. At any time, the customer can ask the café server for newly emitted vouchers, transmitting a signed user-id for authentication. To avoid capture and replay attacks a timestamp should also

be included in the authentication. The server should define a small tolerance for verification (usually Android clocks are precise, being frequently synchronized with a time server). If the user is properly authenticated, 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.

The user can also ask for the receipts of previous purchases. If the server have them they are transmitted to the user (only once) and deleted from the server. The user app identifies the user the sane way as for asking the vouchers.

4. Vouchers

Vouchers are used as a loyalty strategy and as an incentive to use the app. One free coffee voucher is offered to the customer whenever he consumes 3 payed coffees (so the server should track these purchases). Also, a discount voucher is offered whenever the accumulated payed value of all orders from the customer surpasses a new multiple of €100.00 (the server should take note, for each customer, of this total). The discount voucher 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 at most one discount per new order.
A voucher contains a unique serial number (**uuid**), and a type (1 byte) (there are only two types). When they are generated the server associates them to the owner's 'user id'.

5. Signatures

To verify a signature, the server should use the customer public key which was transmitted at the registration phase. The key pair is generated by the app (once) and the private key is kept stored there in the Android Key Store. For the signature keys use the "RSA" algorithm with a length of 512 bits, and for the signature algorithm use "SHA256WithRSA". This produces a signature of 64 bytes which is the shortest, using standard recommended cryptography algorithms.

6. 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 and/or a QR code and camera.
If you have available two physical Android phones supporting NFC, use them. 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 (use binary values, not strings; in the Android or Java side you have the ByteBuffer class for easy manipulation of such values).
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 must be TCP. See https://developer.android.com/studio/run/emulator-networking.html 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.

7. 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 (for instance, emission of receipts electronically by the server, consultation of past transactions (in the server or locally), gains obtained so far, …), and fill any gaps not detailed.

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