Mobile Application Development (MAD)

Student Project 1

29/03/2024

Introduction

This assignment gives you the role of a waiter/waitress in a coffee shop. The shop has 99 tables and 11 waiters. **You are responsible for accepting orders and payments from only 9 of these 99 tables**. Your colleagues will cover the rest of the tables. The orders will be sent to the **barman/barwoman** and the payments will be sent to the **cashier**. The roles of both the barman and the cashier will be played by a remote Web application, called **WebBar**. All communications with WebBar **must** include a personal token that will be used to identify the waiter/waitress.

Your tokens are given in the `tokens.pdf` file that accompanies this assignment.

The application will have three Activities. The first one will display your tables as Buttons. The second one will be activated by tapping on a table (Button), in order to accept an order from that table. The third one will be activated by long pressing on a table (Button), in order to accept a payment from that table.

Note 1: A table may make multiple orders (e.g. when the company orders another round, or a new customer joins the company). WebBar is responsible for merging these multiple orders.

Note 2: A table may pay an order partially (e.g. when a member of the company leaves earlier). Again, WebBar is responsible for monitoring the remainder of a table.

Note 3: Deadline is set to June 15, 2024.

Detailed instructions

1. Activities

You will create three activities: `TableActivity`, `OrderActivity`, and `PaymentActivity`. You will set `TableActivity`, to be the launcher Activity.

2.1. `TableActivity`

This Activity will serve as the starting point for your application. It will display the 9 tables that you are responsible for in the form of 3 X 3 grid (i.e. 3 rows, 3 columns, see Fig. 1).

Layout, Events, and Logic: You will create 9 `Buttons` in a 3 X 3 grid (Figure 2). Each `Button` will be assigned two events listeners:

- The first one will be a standard `onClick()` event listener that will create `OrderActivity`. `OrderActivity` must know the table that created (**Hint:** Create it with a `Bundle`).
- The second one will be an `onLongClick()` listener that will create `PaymentActivity`. `PaymentActivity` must know the table that created (**Hint:** Create it with a `Bundle`).

The buttons will not have a caption. Their caption will be filled up with the table IDs that will be sent later by WebBAR. Moreover, their initial color will be the default (GRAY). The background color will be changed according to the status of the table (also sent by WebBAR). Each table has a status:

1. `table_status=0` when it is empty, or when its order has been fully paid. In other words, **we do not expect money from this table**. We call these tables **Free**. This is denoted by a green background color. These tables cannot accept payments. Therefore, the long click event will do nothing for these tables. However, it will be able to make new orders. Therefore, the `onClick()` listener will be active.
2. `table_status=1` when the order has not been fully paid. In other words, **we expect money from this table**. We call these tables **Occupied**. This is denoted by a red background color. These tables can accept both new orders and payments. Therefore, both events will be active.

The following Table summarizes the discussion above:

Table 1: Table statuses

Property	Free Table	Occupied Table
`table_status`	`0`	`1`
Background color	Green	Red
`onClick` listener	Fires	Fires
`onLongClick` listener	Does nothing	Fires
Accepts new orders	Yes	Yes
Accepts payments	No	Yes
When	No unpaid orders exist	There exist unpaid orders

By the time this activity is created, it will contact WebBAR to retrieve the products and the tables you are responsible for. The following actions must take place `onCreate()`:

1. Contact WebBAR at http://mad.mywork.gr/get_coffee_data.php?t=XYZ where XYZ is your user token. If the token is invalid, WebBAR will respond with a `0-FAIL` status. In the opposite case (valid token), WebBAR will respond by sending a `3-OK` status and the following data inside an `<msg>` block.

Table 2: WebBAR response for the get_coffee_data request

```
<?xml version="1.0" encoding="utf-8"?>
<response>
      <status>3-OK</status>
      <msg>
            <tables>
                  <table>
                        <id>501</id>
                        <status>1</status>
                  </table>
                  <table>
                        <id>502</id>
                        <status>0</status>
                  </table>
                  ……….
            </tables>
            <products>
                  <product>
                        <id>14</id>
                        <title>Capuccino</title>
                        <price>2.00</price>
                  </product>
                  <product>
                        <id>9</id>
                        <title>Chamomile</title>
                        <price>3.00</price>
                  </product>
                  ……….
            </products>
      </msg>
</response>
```

3. There are two main blocks inside the response: `<tables>` and `<products>`. Only the `<tables>` block will be handled here. Specifically, the data inside the `<id>` tag must be placed on the caption of the next Button. The value of the `<status>` tag will determine the background color of the Button, according to Table 1 (see Fig. 2).

2.2. OrderActivity

This Activity must provide an interface for the waiter to create new orders and send them to the barman (WebBAR). As stated earlier, a table may make multiple orders. However, there will be no distinction between the first and the subsequent orders. All the products in this Activity will be displayed with an initial quantity value equal to 0.

Layout, Events & Logic: The layout is shown in Fig. 4 and consists of:

- 1 `TextView` for displaying the table ID (top left).
- 1 `TextView` for displaying the value of the current order (top center).
- 1 `ImageButton` for sending the order to the barman (top right).
- 1 `RecyclerView` for displaying the products and changing their quantity. The layout of the RecyclerView is exactly the same as that of MAD-14. You may copy the appropriate

files from the MADCourse2024 project that resides in the eLearning platform. (**Hint:** You will need one layout file, `product_layout.xml`. You will also need two classes: `Product.class`, and `ProductListAdapter.class`). Still, some slight changes may be required.

When the `OrderActivity` gets created, another request to http://mad.mywork.gr/get_coffee_data.php?t=XYZ is required (XYZ is your user token). You will now process the `<products>` block in the XML response. You must retrieve all the products from that block and insert them to the `RecyclerView` in exactly the same way as it was shown in MAD-14.

The plus and the minus `Buttons` of the `RecyclerView` will change the quantity of the product. They will also immediately update the value of the order `onClick()`.

In addition, an `onClick()` event listener must be attached to the top right `ImageButton`. This listener shall:

1. Check whether all quantities are zero (i.e. no product has been selected).
2. If all quantities are zero, display a `Toast` object with an error message.
3. If there is at least one product with non zero quantity, then the `Button` must submit the selections of the waiter to the Barman. You will contact WebBAR for that. The URL that will be called must be of the form:

   `http://mad.mywork.gr/send_order.php?t=[XYZ]&tid=[TABLE_ID]&oc=[CONTENTS]`

   where:

   - `XYZ`: is your token ID,
   - `TABLE_ID`: is the ID of the table which made the order (**Hint:** get it from the top-left TextView), and
   - `CONTENTS`: are the contents of the order formatted as `{PRODUCT_ID, QUANTITY}` pairs delimited by a semicolon. For example, `oc=1,3;8,1;10,2` means that the order contains: i) 3 items for product ID=1 (Freddo Espresso), ii) 1 item for product ID=8 (Green Tea), and iii) 2 items for product ID=10 (Black Tea).

   Full example:

   `http://mad.mywork.gr/send_order.php?t=1546&tid=501&oc=1,3;8,1;10,2`

4. If WebBAR responds with a `4-FAIL` status, then employ a `Toast` object to display what it is sent within the `<msg>` block (there could be multiple reasons why WebBARs responds like this - it must protect itself from bad calls including malicious ones).
5. If WebBAR responds with a `4-OK`, then:
   a. The status of the table will be set equal to 1.
   b. The background color of the table will become red.
   c. `OrderActivity` must be destroyed returning the user to `TableActivity`.

Hints on how to get the order contents: Getting the order contents may be tricky. One obvious way is to iterate through the elements of the `RecyclerView` when the 'send order' Button is pressed. During the iteration, you will get the quantity for each product from the corresponding `TextView`. If this quantity is non-zero, then you will get the product ID from the left-most `TextView` of the element's layout and you will create a substring of the form `product_id, quantity;` be careful to use the correct symbols to delimit the data. Commas are used to separate a product ID from its respective quantity. Semicolons are used to delimit different `product_id, quantity` pairs.

Iterating through the elements of a `RecyclerView` is expensive especially when the `RecyclerView` accommodates numerous products. Another more efficient way is to get the desired data at the `onClick()` listeners of the plus and minus buttons which control the quantities of the products. You can employ a hidden `TextView` element where you will store the CONTENTS string and update it when the user clicks on a plus or minus button. Alternatively, you can expand our `Product` Java class with a quantity variable. That is the variable that must be updated during these events. Then, when the 'send order' Button is pressed, you will iterate through the dataset and check which products have a non-zero quantity.

2.3. PaymentActivity

This Activity must provide an interface for the waiter to receive payments from the customers and send them to the cashier (WebBAR). There could be multiple payment phases for an order until its balance gets equal to zero. Although in real-world applications negative balances are perfectly valid, we will not allow such cases here.

Layout, Events & Logic: The layout is shown in Fig. 5 and consists of:

- 1 `TextView` for displaying the table ID (top left).
- 3 `TextViews` for displaying the cost (`tv_cost`), the paid amount (`tv_paid`), and the balance (`tv_balance`) of the order, respectively.
- 1 `EditText` for typing the collected amount (`et_am`). It must allow only decimal numbers which cannot exceed the current balance of the order (shown in the third `TextView`).
- 1 `ImageButton` for sending the payment to the cashier (top right).
- 1 `RecyclerView` for displaying the products of the order. Here, the quantity controls (i.e. the 2 Buttons) must be hidden. The Adapter must be extended for this; this is also exactly identical to what was presented in Lecture 10, where the Adapter was modifying itself according to the calling activity.

When the Activity gets created, it must contact WebBAR to get the details of the pending order for a table. The URL to be called is http://mad.mywork.gr/get_order.php?t=XYZ&tid=TAB_ID where XYZ is your token and TAB_ID is the ID of the table which makes the request. The server will respond by either sending `0-FAIL` (rejected token), or `5-FAIL` (for a number of reasons - e.g. bad table ID, or no pending order for this table ID), or `5-OK`. If the server returns a FAILED response, then a `Toast` object will display what the server sends inside the `<msg>` block and the Activity must be immediately destroyed. In the case of `5-OK`, the response will have the form of Table 3.

Table 3: WebBAR response for the get_order request

```
<?xml version="1.0" encoding="utf-8"?>
<response>
        <status>5-OK</status>
        <msg>
                <order>
                        <id>3</id>
                        <table_id>507</table_id>
```

```
                    <cost>29.50</cost>
                    <payment>5.00</payment>
                    <balance>24.50</balance>
                    <products>
                          <product>
                                <id>1</id>
                                <title>Freddo Espresso</title>
                                <price>2.50</price>
                                <quantity>5</quantity>
                          </product>
                          <product>
                                <id>2</id>
                                <title>Freddo Capuccino</title>
                                <price>3.00</price>
                                <quantity>2</quantity>
                          </product>
                    </products>
              </order>
        </msg>
</response>
```

The response fields include:

- `<id>`: this is the order ID in WebBAR's database. You will not need it.
- `<table_id>`: self-explanatory
- `<cost>`: The cost of the products of the order. This is the amount that must be collected. It will be placed in the `tv_cost TextView`.
- `<payment>`: The amount collected so far. It will be placed in `tv_paid TextView`.
- `<balance>`: The current balance of the order (= cost - payment). It will be placed in `tv_balance` and `et_am`. You cannot collect an amount greater than this.. The user is free to replace the content of `et_am` with a value that must always be smaller than the current balance. In this way our app will support partial payments (e.g. when a customer leaves earlier and desires to pay for his/her coffee). If the user enters a greater value and presses the top right `ImageButton`, a `Toast` object will report the error.
- `<products>`: This block contains the products of the order and their quantity. The items of the block will be inserted into the `RecyclerView` of this activity. The quantity controls (i.e. the plus and minus Buttons) must be hidden.

The waiter can immediately press the top right ImageButton to submit the payment. Also, he/she can change the recommended value of `et_am` before pressing the Button. Now when that Button is pressed, the following actions must take place.

1. check whether the amount entered in `et_am` is greater than zero. If it is not, create a `Toast` object and display the error.
2. check whether the amount entered in `et_am` is lower than, or equal to the current balance of the order. If it is not, create a `Toast` object and display the error.
3. If the amount in `et_am` complies with the previous restrictions, then submit the payment to the cashier. More specifically, contact WebBAR by following the URL http://mad.mywork.gr/send_payment.php?t=XYZ&tid=TAB_ID&a=AMOUNT, where:
   - XYZ: is your token ID,
   - `TABLE_ID`: is the ID of the table which made the order (**Hint:** get it from the top-left TextView), and
   - AMOUNT is the amount of payment, that is, the value in `et_am`.

4. If WebBAR responds with a `6-FAIL` status, then employ a `Toast` object to display what it is sent within the `<msg>` block (there could be multiple reasons why WebBARs responds like this - it must protect itself from bad calls including the malicious ones).
5. If WebBAR responds with a `6-OK`, then it will also send the new balance for the order in the fashion illustrated in Table 4.

Table 4: WebBAR response for the send_payment request

```
<?xml version="1.0" encoding="utf-8"?>
<response>
      <status>6-OK</status>
      <msg>
            <table_id>505</table_id>
            <new_balance>10.50</new_balance>
      </msg>
</response>
```

6. If the value enclosed by the `<new_balance>` tags is greater than zero, then just destroy `PaymentActivity` returning the user to `TableActivity`.
7. If the value enclosed by the `<new_balance>` tags is equal to zero, then
   a. Update the status of the table in the database by setting it equal to zero.
   b. The background color of the table will become green.
   c. `PaymentActivity` will be destroyed returning the user to `TableActivity.`

3. Figures

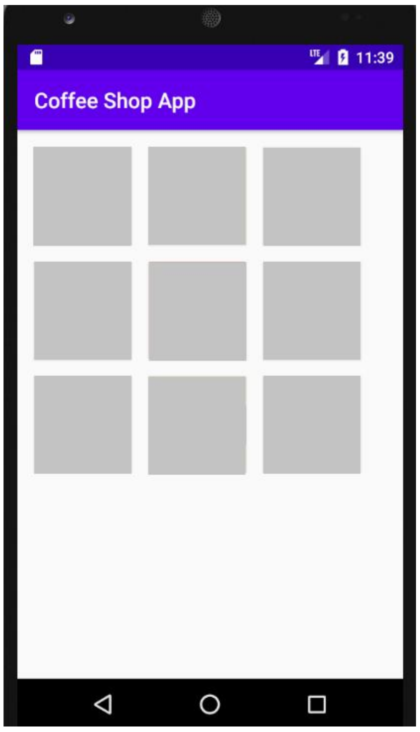


Figure 1: `TableActivity` (layout)

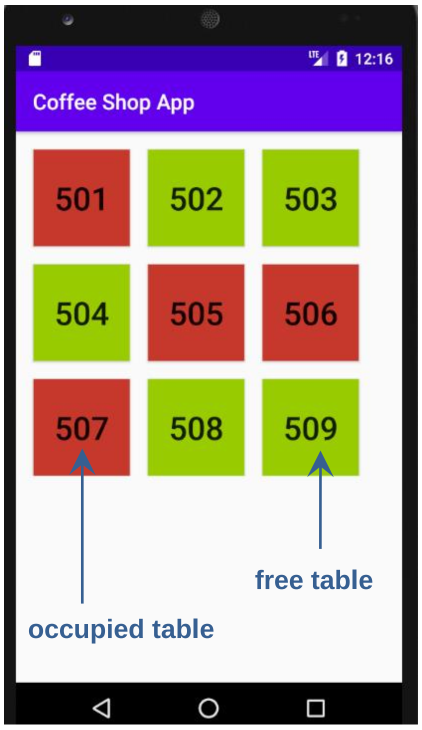
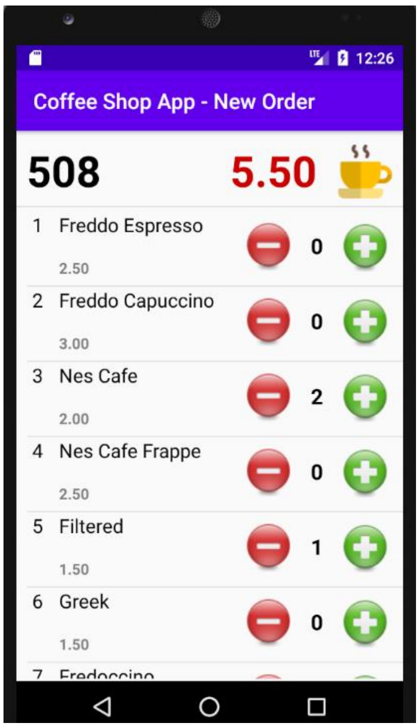


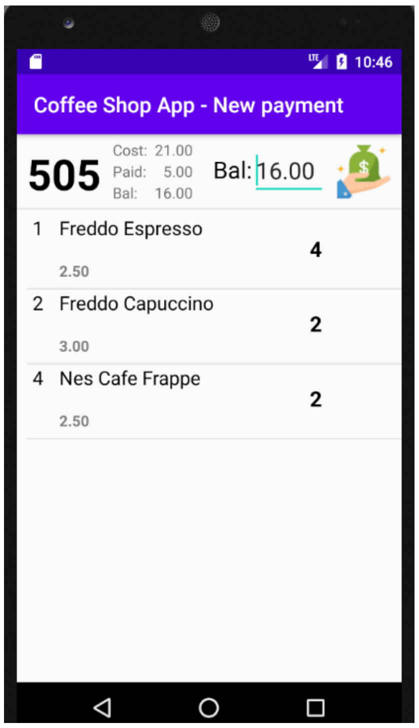
Figure 2: `TableActivity` (captions and colors)



508 5.50

1 Freddo Espresso 2.50 0
2 Freddo Capuccino 3.00 0
3 Nes Cafe 2.00 2
4 Nes Cafe Frappe 2.50 0
5 Filtered 1.50 1
6 Greek 1.50 0
7 Fredoccino

Figure 3: `OrderActivity` (layout)



505 Cost: 21.00 Paid: 5.00 Bal: 16.00 Bal: 16.00

1 Freddo Espresso 2.50 4
2 Freddo Capuccino 3.00 2
4 Nes Cafe Frappe 2.50 2

Figure 4: `PaymentActivity`