Assignment 2

Music Renting Application

Github repository link: https://github.com/SoftDevMobDev-2024-Classrooms/assignment02-TranVuDuc04-

Introduction

This is a simple app called Music Renting Application; it stores information of music instruments, gives users further options to choose for renting and finalises the last receipt.

Development plan

1. Requirements Analysis:

How to use

This app provides an experience for user who wants to rent an instrument. The user can navigate through different instruments using a navigation bar, Each instrument is displayed with its name, color options, quantity, rating, and price per month.

Users can adjust the quantity and color of the instrument, and then confirm their booking by pressing the "Borrow" button. This action takes them to a Booking tab, displaying the selected instrument's details, where they can finalize the rental with “Save” button.

After saving the order, user can see their order receipt in a invoice form.

The app manages data in memory and passes it between activities using Parcelable objects.

User stories and sketches

User Story 1: Liam wants to browse through available musical instruments one at a time, seeing key details: name, color, rating, and price, so that he can make an informed decision on which instrument to rent.

Use Case:

User launches the app and starts browsing through the instruments.

The user is able to see the first instrument's details (name, color options, rating, and price).

Hence, the user can adjust the color and quantity.

The process continues until the user finds an instrument he wants to rent.

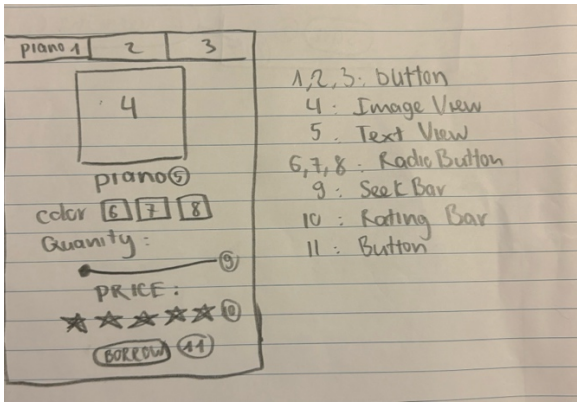


Figure 1. Sketch 1

User Story 2: Duc wants to confirm the rental of an instrument and be sure that the total price does not exceed my available credit, so that he can make a responsible financial decision.

Use Case: The user has selected an instrument and clicks "Borrow" to proceed.

The user confirms their selection by reviewing the instrument's details (name, color, quantity, and total price). Then the application direct to a new tab for user to confirm further detailed information.

After all, the app checks if the total price exceeds the user's available credit.

If valid, the user can click "Save" to finalize the rental. Otherwise, an error message is shown, and they are prompted to adjust their choices.

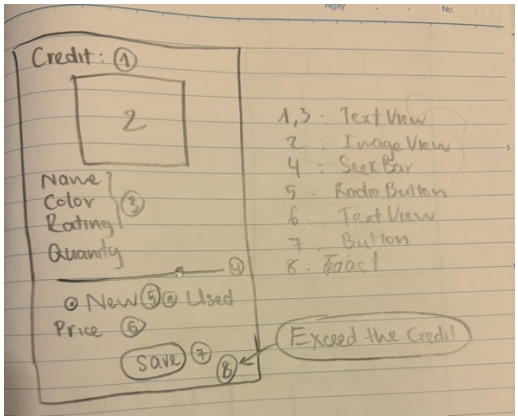


Figure 2. Sketch 2

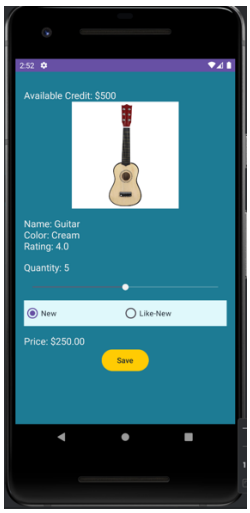
2. Design

UI Layout:

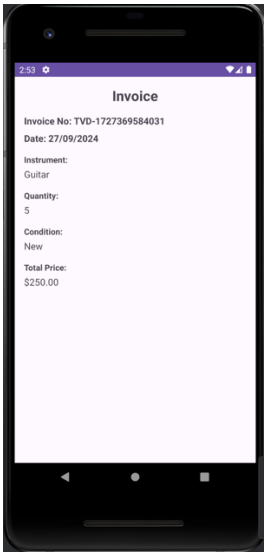
MainActivity: Displays one instrument at a time. The screen includes an ImageView to display the instrument, a RadioGroup to allow users to select a color, a SeekBar to adjust the

quantity, and a RatingBar to display the instrument's rating. There are also buttons to navigate to the next instrument and to proceed with the booking ("Borrow" button).

BookingActivity: Displays the details of the selected instrument, such as the name, color, condition, quantity, and total price. Users can confirm or cancel their booking on this screen.



PrintingReceiptActivity: Shows a final receipt after booking confirmation, displaying instrument name, condition, quantity, total price, unique invoice number, and date.



Widgets:

ImageView: Displays the image of the selected instrument based on color choice.

RadioGroup: Allows users to choose the color and condition of the instrument.

SeekBar: Used for selecting the quantity of the instrument to rent.

RatingBar: Displays the instrument's rating.

TextView: Used to show instrument name, price, quantity, and total cost.

Button: Allow users to navigate through invoice and main page.

App Logic:

Data Storage: The data for each instrument (name, price, color, rating, quantity) is maintained in memory and passed between activities using Parcelable.

Navigation: The navigation provides user the screen with the chosen instrument's details. The "Borrow" button navigates to a screen where users can confirm or cancel the rental. The "Booking" shows the receipt of the last order.

Error Validation: The app validates that the user has selected a valid quantity (greater than zero) and that the total price does not exceed the available credit before confirming the rental.

3. Implementation

UI Layout (XML):

- MainActivity XML: the layout for the instrument details. It includes 3 fragments with a navigation bar, ImageView, TextView for displaying name, RadioGroup for color selection, SeekBar for adjusting quantity, RatingBar for rating, and buttons to navigate to further details.

- BookingActivity XML: Displays the selected instrument's details and allows users to choose more condition, confirm or cancel the booking. Includes TextViews for showing instrument details, SeekBar for quantity selection, and buttons for confirming or canceling the booking.
- PrintingReceiptActivity XML: Presents the final receipt after the rental is confirmed. It displays the instrument name, quantity, total price, invoice number, and date.

Data Passing with Parcelable:

The Instrument class implements the Parcelable interface, ensuring completelt passing the instrument details like name, quantity, condition, and total price from main activity to others.

Advantages:

1. Optimized for Android: Parcelable is designed specifically for Android, making it faster and more memory-efficient.
2. In-Memory Data Transfer: The data is passed without needing to store it persistently in a database or file. This ensures that interactions like booking and confirming an order happen quickly and without delays.

Error Handling:

In BookingActivity, the app ensures that the total price does not exceed the available credit. If any required fields are missing, the user cannot save the rental.

4: Testing and Debugging

- Text and Button Testing: Using Espresso, tests verify that the "Borrow" buttons function correctly.

```
@RunWith(AndroidJUnit4::class)  VuDucTran04 *
class MainActivityTest {
    @Test  VuDucTran04 *
    fun testBorrowButtonFunctionality() {
        //launch MainActivity
        ActivityScenario.launch(MainActivity::class.java)
        //click the Borrow button
        onView(withId(R.id.borrowButton)).perform(click())
        //check that the creditTextView is displayed in BookingActivity
        //if the creditTextView is displayed, that means borrow button navigates to BookingActivity
        onView(withId(R.id.creditTextView)).check(matches(isDisplayed()))
    }
}
```

- Parcelable Testing: Tests confirm that the data passed between MainActivity and BookingActivity via Parcelable objects is correctly transferred and displayed.

```
class ParcelableTest {
    @Test  new *
    fun testParcelableDataTransfer() {
        ActivityScenario.launch(MainActivity::class.java)
        //select instrument type
        onView(withId(R.id.buttonFragment1)).perform(click())
        //set quantity to 5
        onView(withId(R.id.quantitySeekBar)).perform(setProgress(5))
        //click borrow button
        onView(withId(R.id.borrowButton)).perform(click())
        //check that the quantity passed to BookingActivity is displayed correctly
        onView(withId(R.id.quantityLabel))
            .check(matches(withText( text: "Quantity: 5")))
    }
```

- Browsing between Instrument tab Testing
- Validation Testing

5. Finalization

- User Testing: Get feedback from peers or users.
- Submission: Document a report.

Tools and Resources

- StackOverflow
- Android Studio
- ImageColorPicker
- Android Developer
- Chatgpt

Key design

State Management:

- Decision: Use onSaveInstanceState and onCreate to ensure that crucial variables are restored after configuration changes, orientation changes.
  ⇨ maintain consistent user experience

Toast and AlertDialog:

- Toasts are brief, non-interactive messages and automatically disappear after a short time, it doesn't cause any trouble for user to interact with the application. It is just

simply a quick notification. The dialog makes sure the user is fully aware of their decision.

- With the combination, I think it would be appropriate since toasts are displayed only after the user has confirmed his order using alertdialog

Reflection

- Clear and consistent use of Kotlin-specific features
- Well-documented with comments that clarify crucial functions, making the code easier to understand and maintain.
- The use of onSaveInstanceState was effective in maintaining app state during configuration changes, ensuring a smooth user experience.
- Updating UI: based on the current instruments and price, providing required feedback to users.
- Testing: crucial in spotting and tackling problem