Step Mobile

Coding Exercise

Task

Congratulations! You've been tasked with building a prototype of the new (fictional) cash withdrawal feature for the Step app! This feature is a simple approximation of how cash deposits work in the real world, but in reverse.

The idea is simple: there are several buttons, which allow the user to withdraw different amounts. Each time the user hits a button, we will simulate contacting a server, and getting back a response with two pieces of information: a 4-digit code, and a dollar balance (or an error string). In real life, the user would then provide this code to a cashier at a convenience store, but we'll ignore that part for this exercise.

You will need to implement both the UI, and a mock service to simulate the server portion. For this exercise, the mock service will start the user off with $500. Two buttons will provide the ability to either withdraw $10 or $100 at a time.

You can implement this app using either React Native, Xcode (Swift or Objective-C) or Android Studio (Kotlin or Java). Other IDE's are welcome too, as long as the final code can be compiled by these tools.

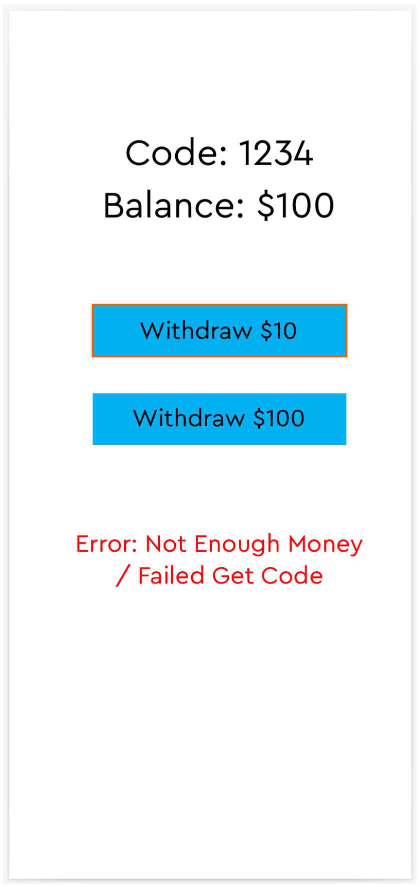
Service API

To simulate the service API, you will want to create a class that mocks out the network behavior. Here is pseudo-code:

```
class ServiceAPI {
  /**
   * Returns a 4-digit code, updated balance, or an error string if a
   * code could not be generated (for example, insufficient balance).
   *
   * Note: The two sleep’s are important, to simulate network delays.
   * Don’t forget to add them!
   *
   * @param {int} amount - The amount (in dollars) to withdraw.
   */
  getCode(int amount) => Response {
    // sleep(1 second)
    // update and check balance, compute new code or error response
    // sleep(1 second)
    // return the response
  }
}

class Response {
  /**
   * New generated code (can be random or sequential, but should change
   * for each new response).
   */
  int code

  /**
   * New updated balance after subtracting request amount. Should always
   * be >= 0.
   */
  int balance

  /**
   * Non-empty if something went wrong. The values of code and balance do
   * not matter if an error occurred.
   */
  string error
}
```

User Interface

Example screenshot is below. On startup, there will be no code, balance or error. Once the user hits a button, and we get back the response, the screen should update automatically.



Bonus Tasks

- Try tapping one of the buttons in rapid succession. What happens? Does the app stay responsive, or can the balance go negative? If either of these are true, how can we fix it?
- What happens if the server fails randomly? Let’s simulate it, by making the server call fail 20% of the time. (Create a new error string, such as “Failed Get Code” to distinguish from “Not Enough Money”.)

- Let’s suppose that we want to throttle generation of the codes. Add a 10 second timer to each code, so that user cannot generate a new code until the previous one has expired. Add a countdown timer to the user interface.