Check Digit Calculator / Barcode Validator

Andrew Doepke

10/17/2021

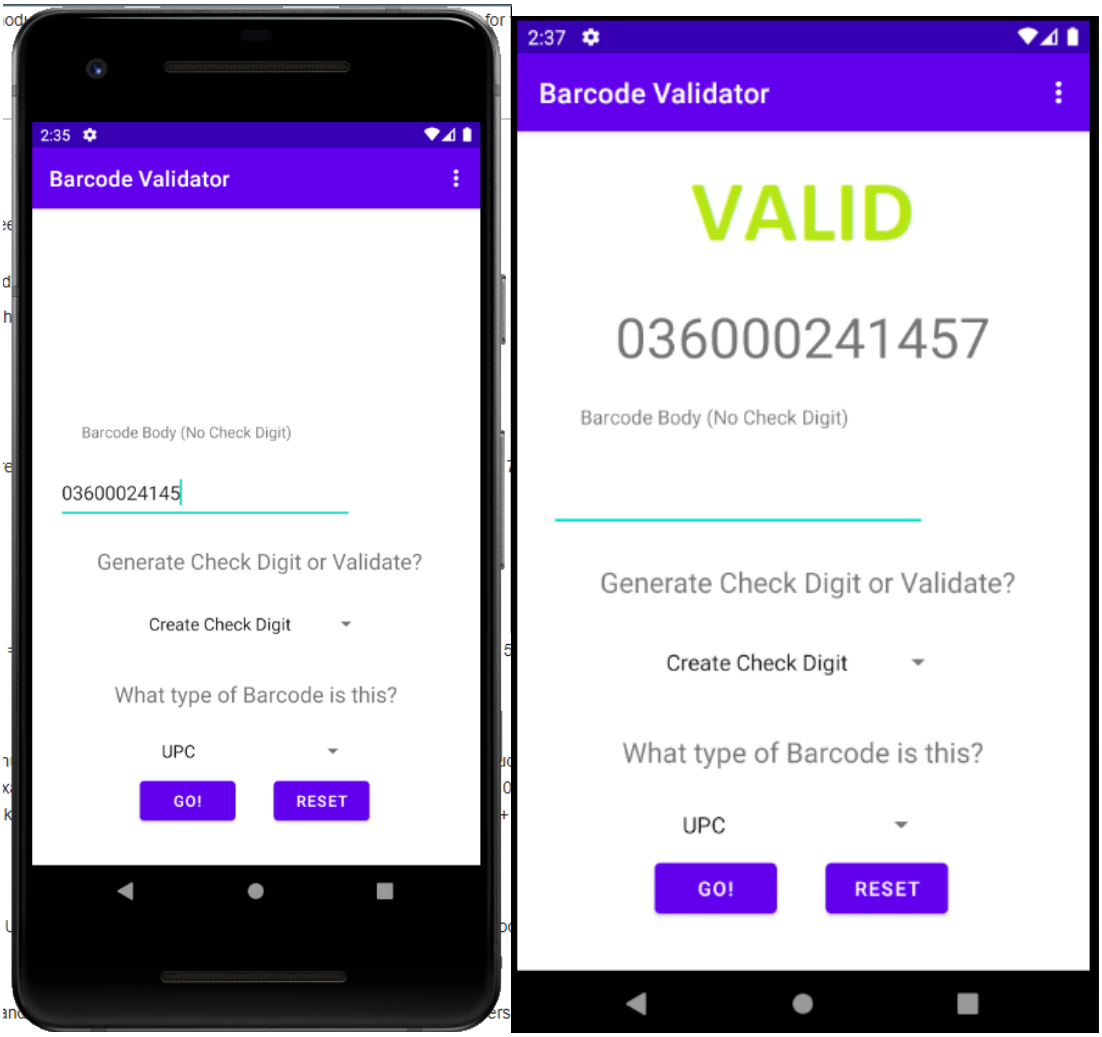
GitHub Repo: https://github.com/andrewdoepke/Barcode

I was assigned to create a specialized calculator application using Android Studio and the Kotlin language. It took me a while to finalize my ideas, but what I decided to go with was an app that allows one to either generate a check digit for a barcode body, or validate a barcode using the same algorithm. This application can currently generate and validate barcodes using the UPC algorithm and the ISBN 13 algorithm. I found information on how these work online, and implemented them into Kotlin.

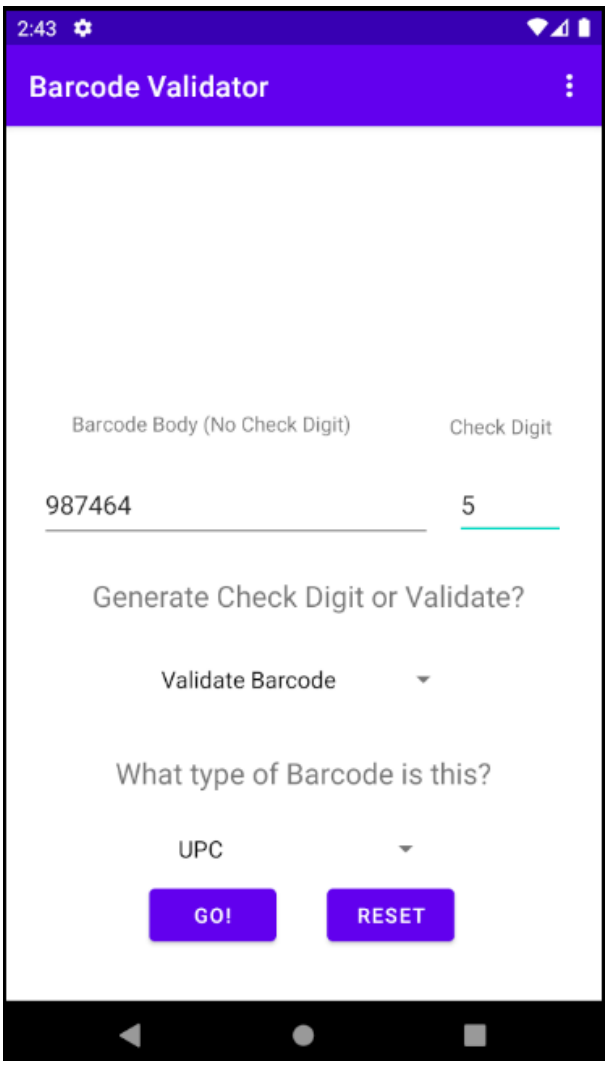
UPC ISBN 13

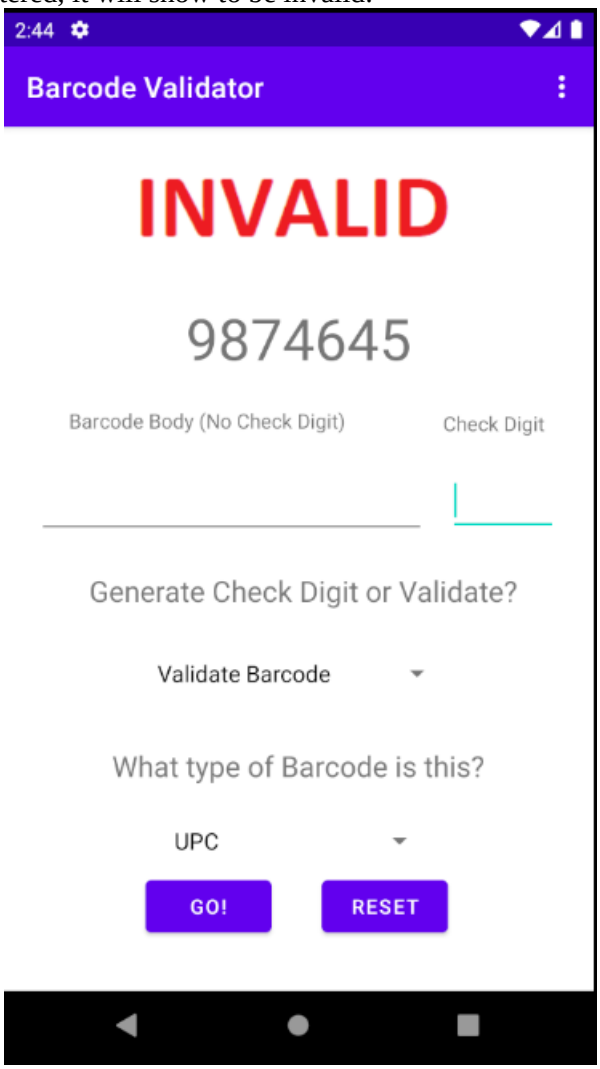
The basic flow of this application is to first open it, choose whether you're going to validate a barcode or calculate a check digit for a new barcode, and choose the type of barcode you are working with. Then, one would type in the barcode body in cases of both validation and generation, and if validating, a text box appears where the check digit for that code is typed. After all is entered, the "GO!" button will generate a Barcode object, clear the input fields, display the code, and display whether or not the code is valid. The "RESET" button can also be pressed at any moment, which will clear all data.

For example, if I want to generate a check digit for the UPC barcode "03600024145", I can enter the values as such. This code will evaluate to a check digit of 7. After hitting the Go button, a complete and Valid Barcode will be generated.

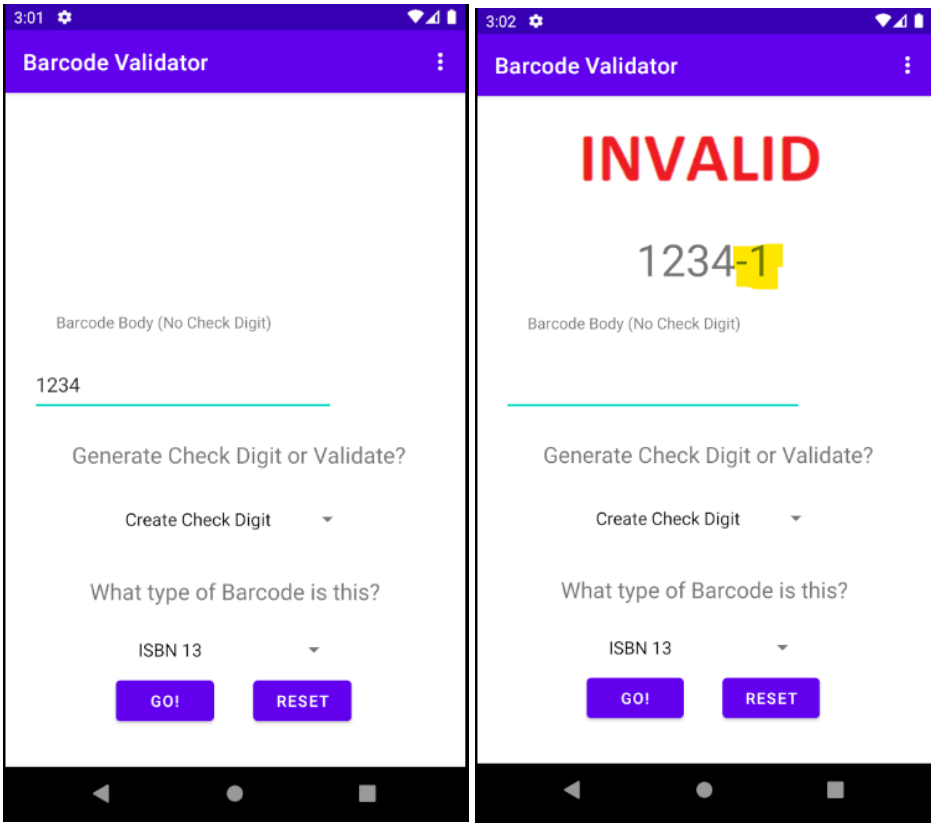


If I wanted to validate this (or any other code), I can enter it in with the check digit this time and it will show to be valid. If a full and Invalid Barcode is entered, it will show to be invalid:



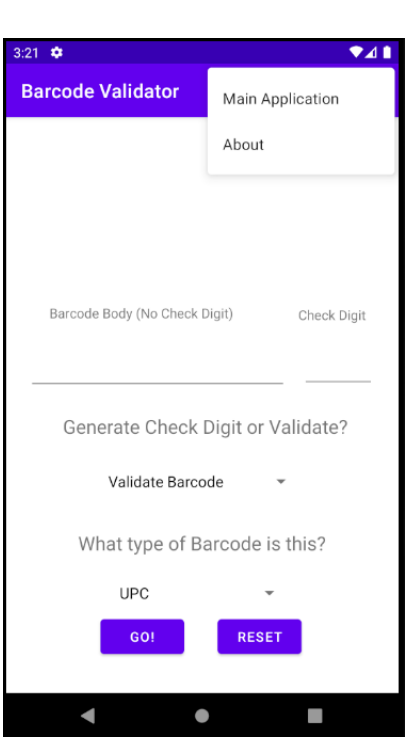


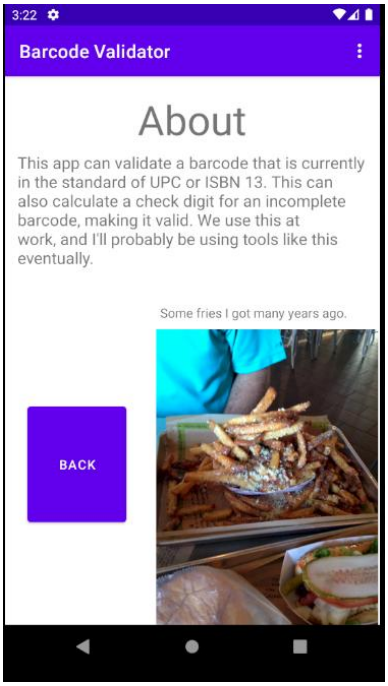
All of this also functions with ISBN 13 as well. One quirk with this is that the barcodes have to be 13 digits long, so it will show an invalid “-1” code when the body isn’t 12 digits on validation, and will show the code as only -1 when creating a check digit or if a check digit is not present during validation. I chose to default the check digit to “-1” to catch errors within a barcode, to invalidate a barcode that encountered an error. This applies to every barcode as it is built into the Barcode functionality.

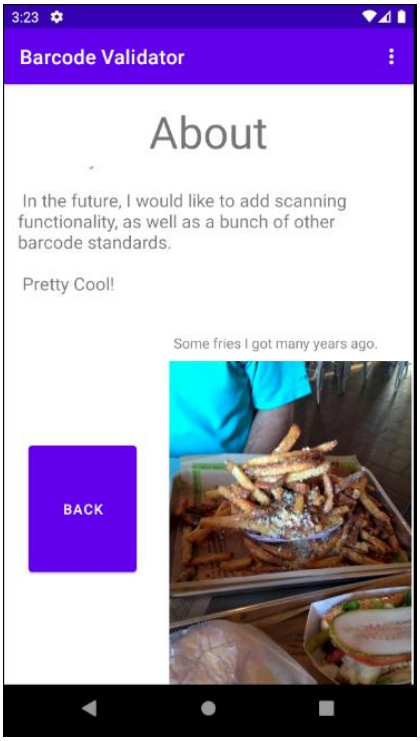


I decided to model a Barcode as a Kotlin Class that handles functionality in its initialization, so the application can interface with it easily, without the hassle of writing logic in the Activity code. The application just creates the Barcodes based on input, and is able to refrence the fields within that Barcode to draw onto the screen. This is handled with minimal code in the Go button’s listener function. The Barcode class itself takes the Barcode body, a check digit, and the algorithm type as parameters, and on initialization, will generate a check digit if one isn’t passed, and will then append the passed or generated check digit and validate. When a check digit is generated, a function is called to switch on the Type parameter, and the validation function will generate a new check digit and test against the digit passed to it.

This application also has a menu bar and a separate page with some information, called the About page. This page has a header, a text box with a paragraph that one can scroll through and read in full. It also contains a back button and a photo I felt I would share with the user. One may navigate to this page by pressing the menu (burger) icon on the upper right-hand side of the screen, and clicking 'About'. To return from this page, one may navigate back to the menu and click 'Main Application', or press the 'Back' button on the screen. Navigation between these pages doesn't save any data, although this is something I may implement in the future.

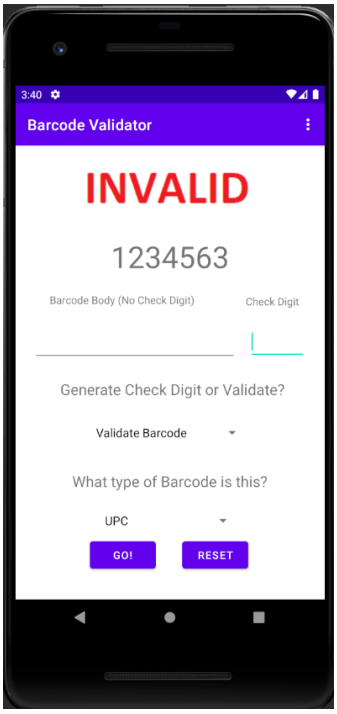




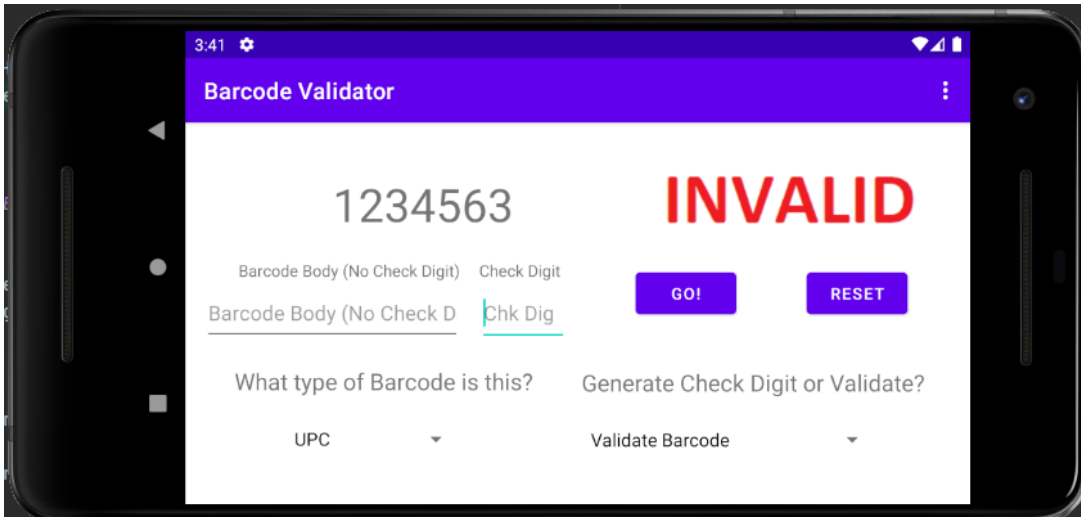


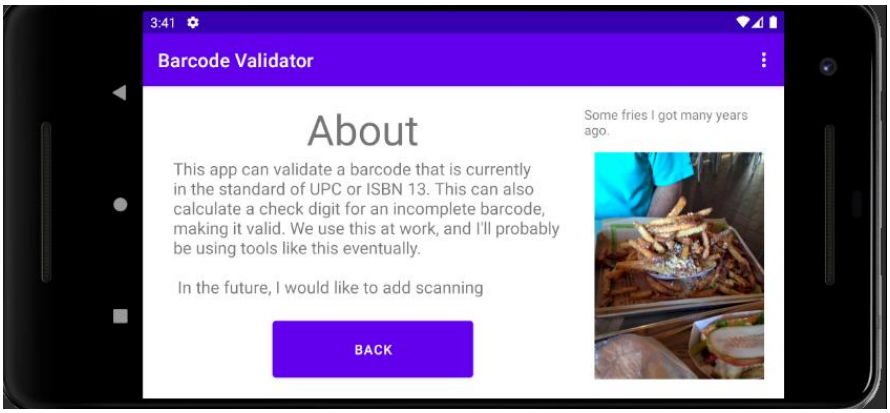
This is scrollable, so as much information as needed can be written into here.

All of the activities in this application support a landscape layout as well. These layouts are defined in separate XML files in the land layout folder. When switching between orientations, any values input or changed will stay between the switch. I did this by overriding the onSaveInstanceState function and passing a few variables into the registry. I was limited by the functionality of ImageView, where I had to create integer values corresponding to each image ID and call this back to figure out which image will display after the change.



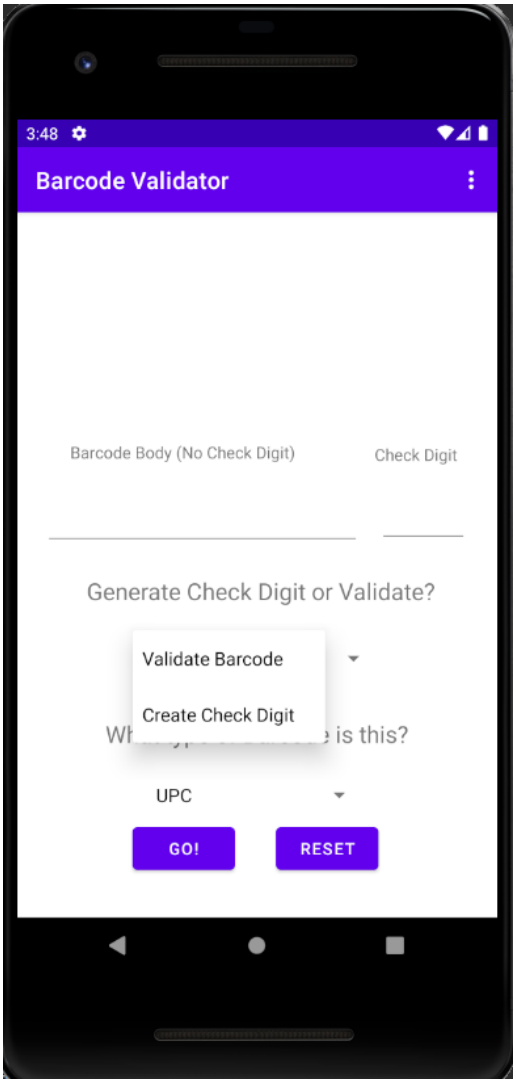
Values will stay between rotations, and any text that hasn't been submitted will also stay.

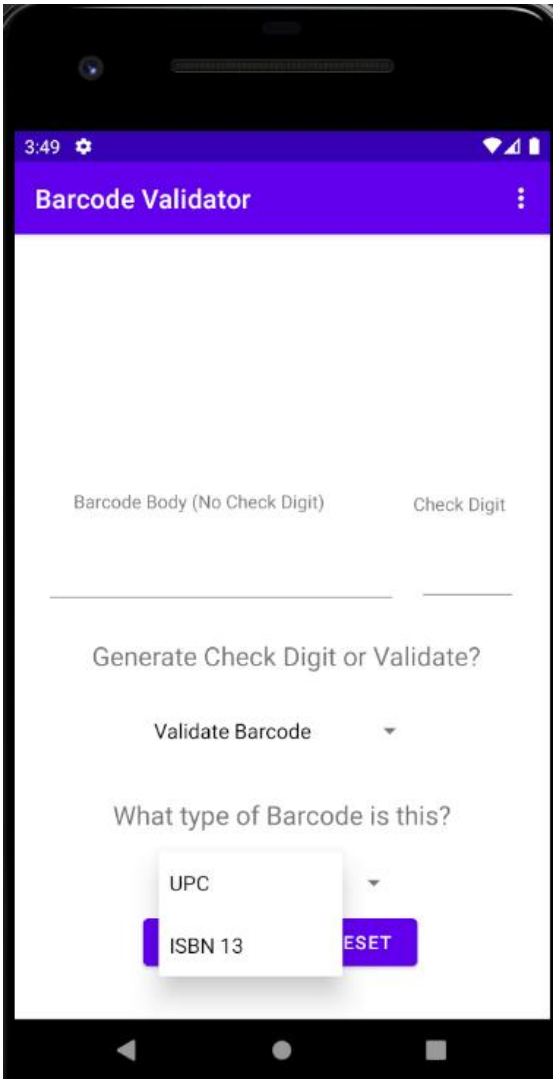




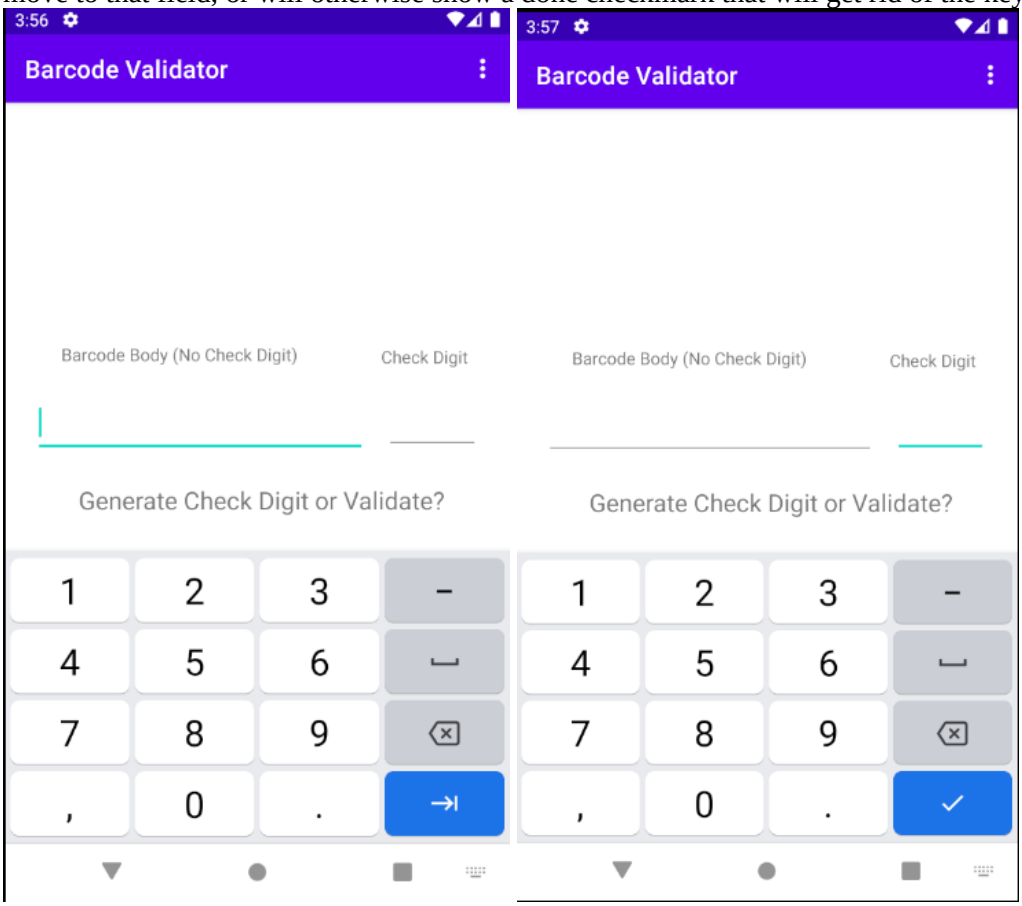
I handled the re-filling out of values in the onCreate function, although I am not sure if this was the best choice, but I was getting errors when overriding onRestoreInstanceState that may or may not have been related, but I got it working with onCreate which is fine for this application.

I implemented Android Spinners for the dropdown menus that allow selection of Barcode type and Method for the application to follow. Each of these dropdown items have different functionality and set different values to change to the desired state of the application. The state of these is also held through rotations.





The input is handled only with selection from these dropdowns and a number pad. If there is another field to be filled out (Check Digit after barcode body), the keypad will allow to click next and move to that field, or will otherwise show a done checkmark that will get rid of the keypad.



Currently, this application only supports two check digit algorithms. I would like to add more in the future as this would give this application more use, especially if someone was to use this on the job. This app also doesn't display the actual barcodes (only corresponding numbers) which works, but a future feature that would be useful is to actually draw the barcodes on the screen and give some export functionality (Google Drive, MS Sharepoint, etc.), and even a scanning functionality. This app was thought out, but not created entirely with the use cases of actual jobs in mind. This would be something to discuss with the target audience of the app, to further refine it to something that would be very useful on the job. Even so, using this app is far easier than doing the calculations by hand, or getting to a computer to find a website that does this. It is usable without internet, though, so on a tablet or other device that doesn't have fast internet (or at all), this could prove very useful.