CSCI 321/521 Assignment 2

There seems to be an unwritten rule that every mobile device course has a tip calculator as an early assignment, and this course will be no exception!

Create a new iOS App in Xcode using the Storyboard / UIKit App Delegate interface and life cycle.

Model

The Model for this app will be quite simple, consisting of the following three properties, which may be stored in the view controller:

- A bill amount (floating point number, initial value 0.00)
- A tip percentage (integer, initial value 20)
- A party size (integer, initial value 1)

View

This app will have only a single view / Storyboard scene. You will need the following user interface elements at minimum:

Inputs

- The bill amount (a positive floating-point value)
- The desired tip percentage (an integer value between 0 and 40, in increments of 5)
- The party size (an integer value between 1 and 10)

Use a **Text Field** (an object of the `UITextField` class) to allow the user to enter the bill amount. Use **Steppers** (objects of the `UIStepper` class) or **Sliders** (objects of the `UISlider` class) or to allow the user to enter the tip percentage and party size.

(Or use one of each, it's up to you.)

Outputs

- The tip percentage selected
- The party size selected
- The total bill amount (including tip)
- The share of the bill each person in the party needs to pay

Outputs can be displayed using **Labels** (objects of the `UILabel` class). Dollar amounts should be formatted so that they are displayed as currency (dollars and cents). See the documentation on the `NumberFormatter` class and the `NumberFormatter.Style` enum for details on doing this.

Other

- Use other **Labels** as needed to make the purpose of your UI elements clear to the user.
- Set up constraints to ensure that your UI elements will be properly positioned in all both portrait and landscape orientations and across all of the different iPhone platforms.

Controller

The single View Controller class can potentially contain all of the code that you write for this assignment. The controller essentially has two jobs in this app:

1. Handle events generated by the user interacting with the input UI elements of the view. Generally, the controller's response to those events should be to update the appropriate model property. For example, when the user moves the Slider that selects the party size, the party size property should be updated with the new value of the Slider.
2. Update the output UI elements of the view in response to changes in the model. For example, when the party size property changes, the Label that displays the share of the bill each member of the party needs to pay should be updated.

Messages from the View to the Controller

Much like Buttons, Sliders and Steppers in iOS are Views that use the "Target-Action" pattern to send messages to the Controller. Moving a Slider results in a "Value Changed" event. If you connect that Sent Event for the Slider or Stepper to an Action method, you can then access the new value of the Slider or Stepper through its `value` property inside that method. The value can then be assigned to the corresponding model property in the View Controller or model class (some type conversion may be necessary).

For the Text Field, you will need to decide on a way for the user to signal that they are done entering input. There are several possibilities:

1. Allow the user to press a Button next to the Text Field to signal that they are done entering input.
2. If the keyboard for the Text Field has a Return key, pressing that key can signal that the user is done entering input. Make the View Controller a `UITextViewDelegate` and connect the Text Field's `delegate` outlet to the View Controller. The delegate protocol method `textFieldShouldReturn(_:)` will be called when the user presses the Return key.
3. If the keyboard lacks a Return key and you don't want to use a Button, you can allow the user to signal that they are done entering input by tapping anywhere outside the Text Field, as discussed in class.

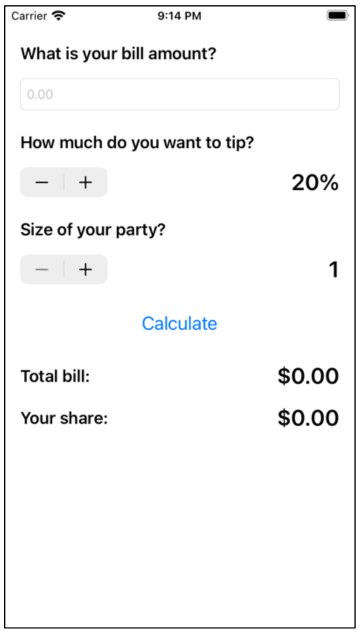
When the user is done entering input, you will need to dismiss the keyboard, retrieve the text in the Text Field, and convert the `String` to a `Double` so that it may be assigned to the bill amount property.

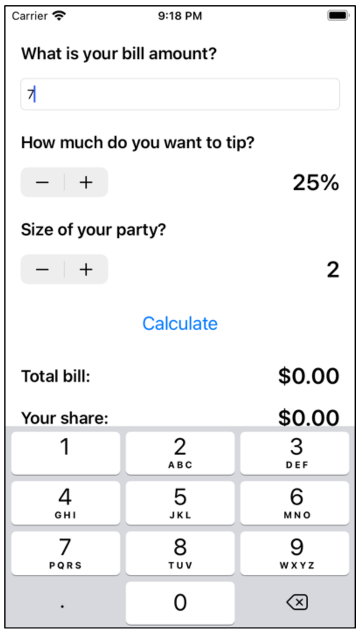
Programming Notes

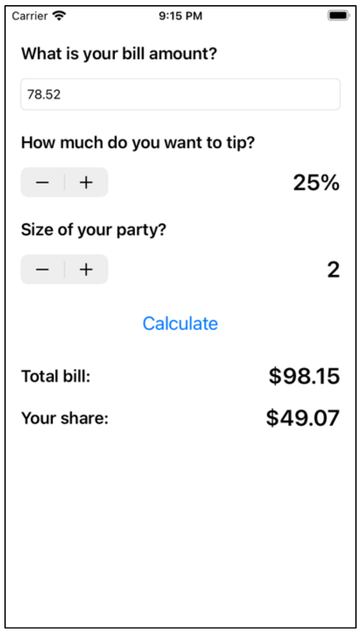
- Design for the lowest common denominator. You should support the iPhone SE 2nd Generation and up. Of those devices, the iPhone SE 2nd Generation has the least amount of available screen real estate, so if your UI works on that device, it should be fine on the larger members of the iPhone family as well.
- Ideally, the **Text Field** should prevent the user from entering any invalid characters. If you use a keyboard that allows the user to input an invalid value and they do so, use an Alert Controller to display an error message and do not accept the invalid value.

Sample UI Screen Shots

Here's an example of what the user interface for this app might look like on the iPhone SE 2nd Generation. You do not need to match the style shown here as long as all of the required elements are in place and your app is functional.







Extra Credit

- Create your own set of app icons (you are welcome to use art found on the Internet). (5 points)
- Create a launch screen. (5 points)