prework.md Raw

iOS Pre-work: Tip Calculator App

This pre-work incorporates the following steps:

1. Setup Xcode
2. Familiarize yourself with Swift
3. Build the initial Tip Calculator
4. Add a Settings Screen to the Tip Calculator
5. Submit your app for review via GitHub
6. Extend your app, improve UI, add features

1. Setup Xcode

Download Xcode from the Mac App Store.

Note: Membership in the iOS Developer Program is **not** required. This program costs $99 a year and is only required if you want to distribute your apps in the App Store and have access to beta iOS releases.

2. Familiarize yourself with Swift

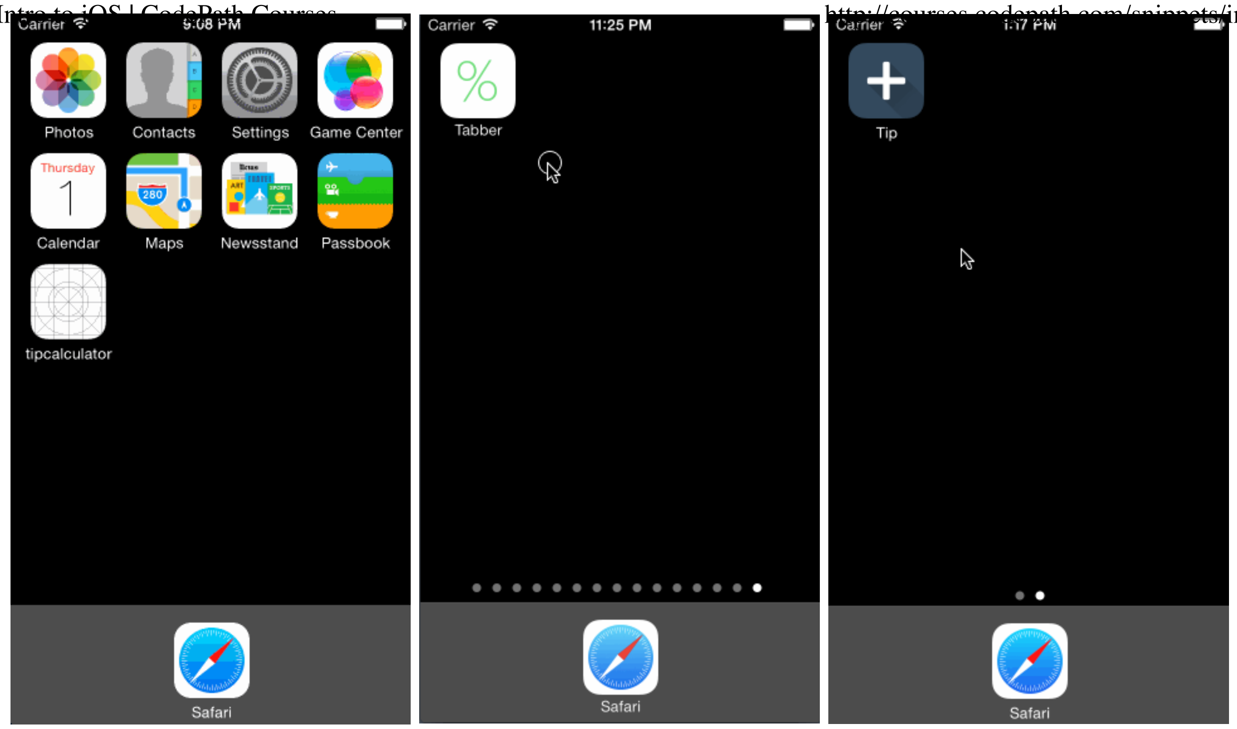
There are a bunch of resources out there to learn Swift. We recommend taking a look at the following options:

- Apple has the an official Swift guide.
- You can also find more resources here.

3. Build the initial Tip Calculator

Follow this video walkthrough.

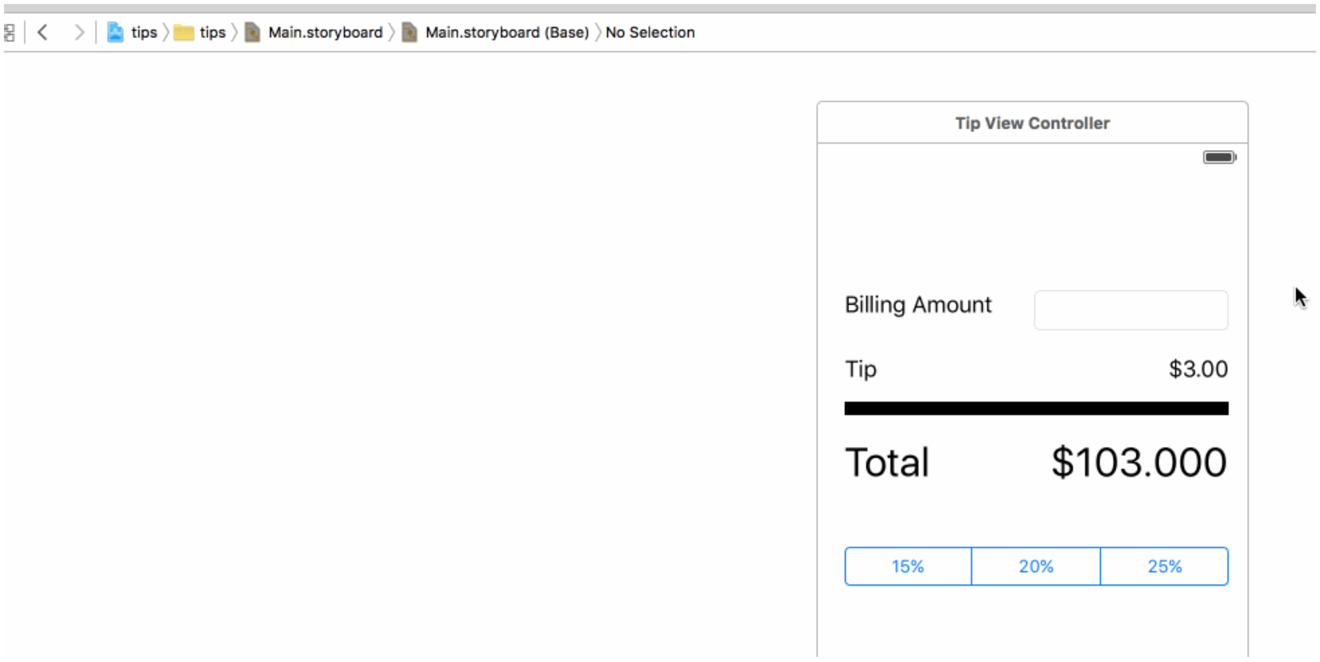
Here are some examples of excellent past submissions. It only takes a couple hours to implement the basic version, but you should spend more time exploring the iOS framework.



4. Add a Settings Screen to the Tip Calculator

Add a Navigation Bar to the main view

On the Interface Builder, select `TipViewController` , then embed it in a navigation controller (Editor -> Embed In -> Navigation Controller)

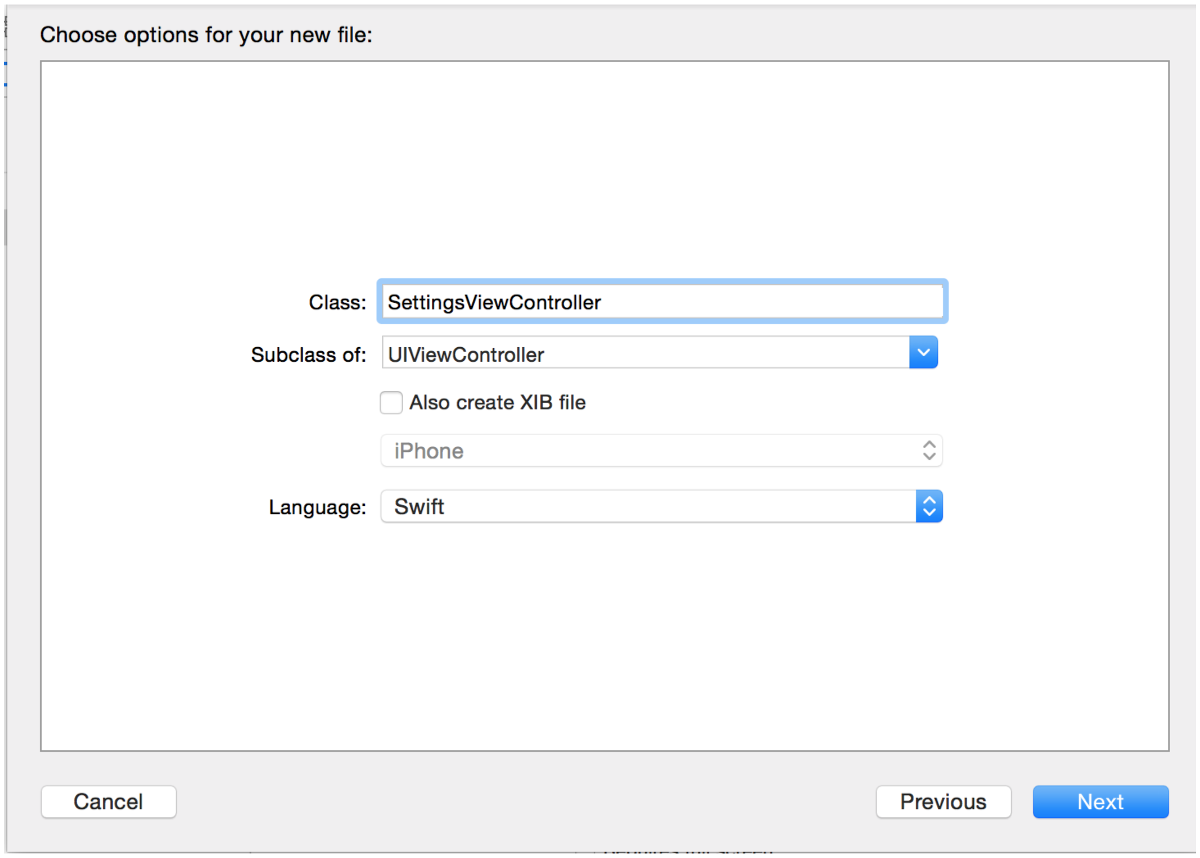


Create a Settings View Controller

Every screen in your application is implemented by a view controller. A view controller has two parts: the designed part that's in the Storyboard and an Swift class that contains the code for that screen.

If a screen in the application doesn't require any code, then it can use the default UIViewController. However, as soon as you need to handle events like a user tapping on the screen, then you need to create a custom view controller for that screen.

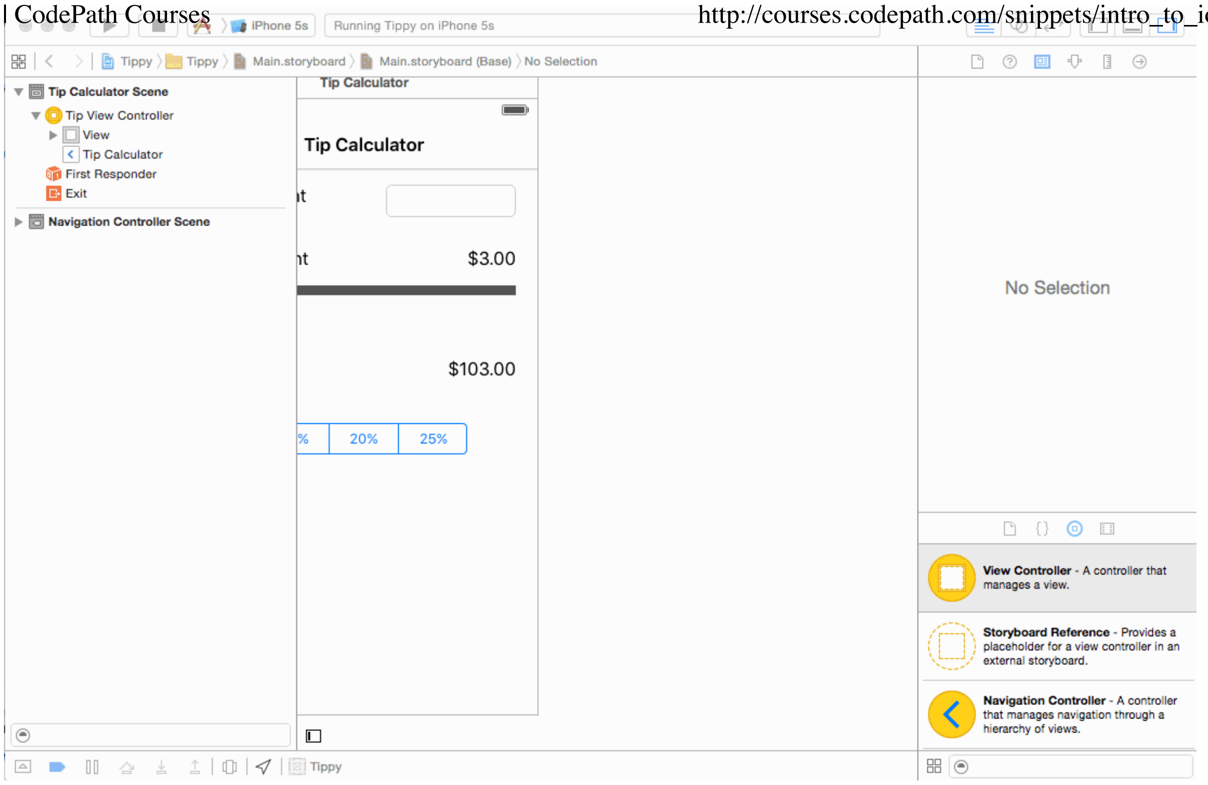
1. For our settings screen, we need a custom view controller (`SettingsViewController`):
   - Create the class (File -> New -> File -> Cocoa Touch Class) with the following options:



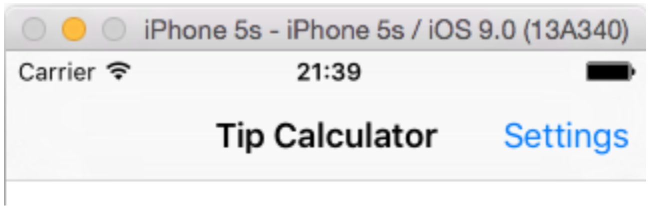
Choose options for your new file:

Class: SettingsViewController
Subclass of: UIViewController
[ ] Also create XIB file
iPhone
Language: Swift

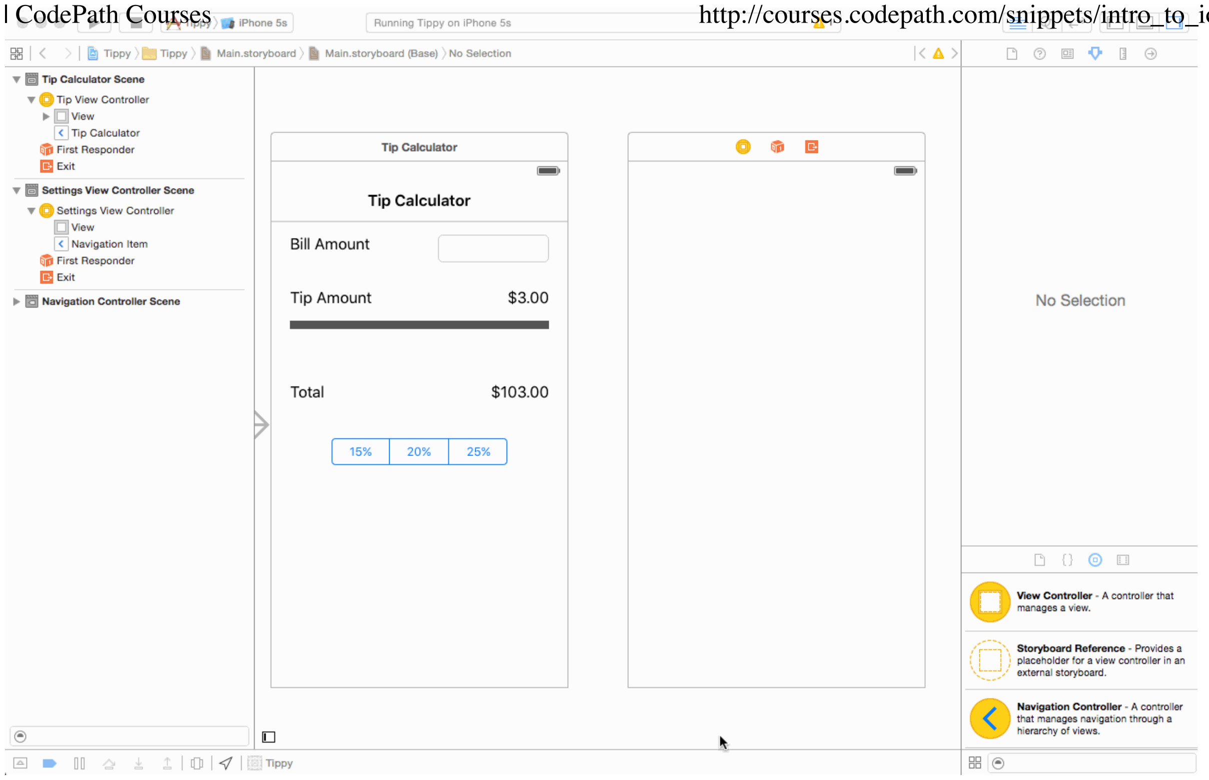
   - Then, use the "Object Library" to add a View Controller to the `Main.storyboard` and associate it with the `SettingsViewController` class:


iPhone 5s
Running Tippy on iPhone 5s
Tippy 〉 Tippy 〉 Main.storyboard 〉 Main.storyboard (Base) 〉 No Selection
Tip Calculator Scene
Tip View Controller
View
Tip Calculator
First Responder
Exit
Navigation Controller Scene
Tip Calculator
Tip Calculator
$3.00
$103.00
20%
25%
No Selection
View Controller - A controller that manages a view.
Storyboard Reference - Provides a placeholder for a view controller in an external storyboard.
Navigation Controller - A controller that manages navigation through a hierarchy of views.
Tippy

2. Now we need a way to get from our main screen to the settings screen:
   - We'll do this by adding a button to the navigation bar:



   - Let's go back to the `Main.storyboard` and use the "Object Library" to add a `BarButtonItem` to the navigation bar. Tapping on this button should "segue" us to the `SettingsViewController` :


Tippy
Main.storyboard
Main.storyboard (Base)
No Selection
Tip Calculator Scene
Tip View Controller
View
Tip Calculator
First Responder
Exit
Settings View Controller Scene
Settings View Controller
View
Navigation Item
First Responder
Exit
Navigation Controller Scene
Tip Calculator
Bill Amount
Tip Amount
$3.00
Total
$103.00
15%
20%
25%
No Selection
View Controller - A controller that manages a view.
Storyboard Reference - Provides a placeholder for a view controller in an external storyboard.
Navigation Controller - A controller that manages navigation through a hierarchy of views.

- **Note:** You'll need to ctrl-drag from the `BarButtonItem` to the `SettingsViewController` to create the push segue.

Additional info: In iOS, navigation stacks are one of the most common navigation mechanisms. You've seen it in many apps when you tap a button a new view slides in from the right. When you tap the back button, the previous view slides in from the left. A UINavigationController maintains the stack of view controllers and you can push new view controllers onto the stack. The back button is automatically added and tapping on the back button will pop the most recent view controller.

Designing the Settings Screen

The purpose of the settings screen is to configure the default tip percentage. In the `SettingsViewController` we added to the Main Storyboard, design what you think the SettingsViewController should look like.

Loading and Saving

There are several different persistence mechanisms in iOS. The simplest to use is a persistent key-value store called NSUserDefaults. You might use NSUserDefaults for similar purposes as cookies in web development. They can store things like application settings, the current user, or a flag for whether a user has already seen a helpful hints popover.

To save a key to NSUserDefaults, do something like this:

```
let defaults = NSUserDefaults.standardUserDefaults()
defaults.setObject("some_string_to_save", forKey: "some_key_that_you_choose")
defaults.setInteger(123, forKey: "another_key_that_you_choose")
defaults.synchronize()
```

To load a key from NSUserDefaults, do something like this:

```
let defaults = NSUserDefaults.standardUserDefaults()
let stringValue = defaults.objectForKey("some_key_that_you_choose") as! String
let intValue = defaults.integerForKey("another_key_that_you_choose")
```

Note the `synchronize` call. NSUserDefaults automatically and periodically synchronizes, but to manually flush the keys and values to disk, call synchronize to guarantee that your updates are saved.

View Controller Lifecycle

When returning to the main `TipViewController` from the settings screen, it would be good to have the tip percentage reflect the new default value. One way to do that is load the tip percentage from NSUserDefaults whenever the view appears.

A view controller has a series of "lifecycle" methods that are called at various stages. When the view controller is initially set up, `viewDidLoad` is called. As it is shown `viewWillAppear` and `viewDidAppear` are called. As it is hidden `viewWillDisappear` and `viewDidDisappear` are called. The lifecycle methods are split into two methods (i.e. `viewWillAppear` and `viewDidAppear`) because sometimes you want some behavior to happen before the transition animation starts or after the transition animation ends.

In `TipViewController.swift`, try adding the following methods and watch the console output as you navigate into the settings view and back.

```
override func viewWillAppear(animated: Bool) {
    super.viewWillAppear(animated)
    print("view will appear")
}

override func viewDidAppear(animated: Bool) {
    super.viewDidAppear(animated)
    print("view did appear")
}

override func viewWillDisappear(animated: Bool) {
    super.viewWillDisappear(animated)
    print("view will disappear")
}

override func viewDidDisappear(animated: Bool) {
    super.viewDidDisappear(animated)
    print("view did disappear")
}
```

Animating View Properties (Optional)

Many UIView properties can be animated, including frame, center, backgroundColor, alpha, transform, etc.

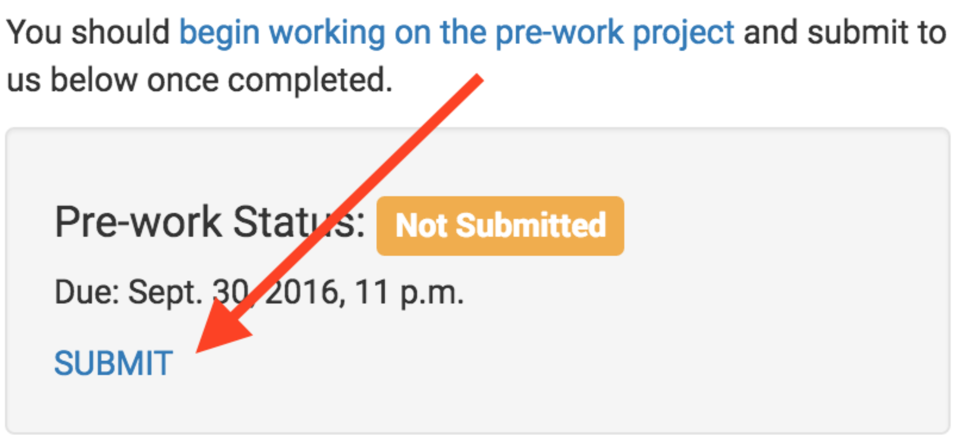
In order to animate one or more views, simply create an animation block. The animation block will animate from the current value to the value that's set within the block. You can manipulate multiple views within the block. For example, in the snippet below, firstView is fading in, while secondView is fading out. This is assuming that you've created outlets for firstView and secondView.

```
// Optionally initialize the property to a desired starting value
self.firstView.alpha = 0
self.secondView.alpha = 1
UIView.animateWithDuration(0.4, animations: {
    // This causes first view to fade in and second view to fade out
    self.firstView.alpha = 1
    self.secondView.alpha = 0
})
```

5. Submitting the Project

Once you've completed the Tip Calculator application (**including adding a settings screen**), please push your app via Github. Also be sure to **include a README** containing a GIF walkthrough using LiceCap of the app demonstrating how it works with required user stories completed. Use this README template.

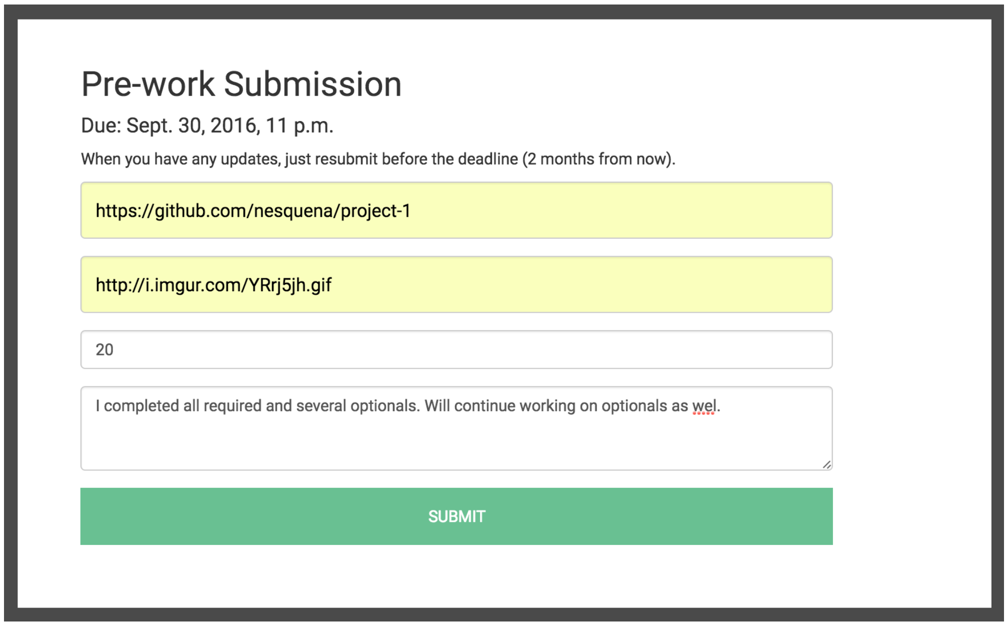
After you have **completed all required user stories and added a README as described above** then you are ready to notify us that you are ready for a pre-work review. To do this, go to the application status dashboard and then press the "SUBMIT" button in the pre-work section:


You should begin working on the pre-work project and submit to us below once completed.

Pre-work Status: Not Submitted

Due: Sept. 30, 2016, 11 p.m.

Make sure that this is the correct cohort before submitting. In the page that appears, enter the required information about your project:


Pre-work Submission

Due: Sept. 30, 2016, 11 p.m.

When you have any updates, just resubmit before the deadline (2 months from now).

https://github.com/nesquena/project-1

http://i.imgur.com/YRrj5jh.gif

20

I completed all required and several optionals. Will continue working on optionals as wel.

Then press "Submit" at the bottom to push your submission. Note that **you can always update your submission at any time** in case you want to re-upload the GIF or update the hours spent.

Submission Checklist

Please **review the following checklist** to ensure your submission is valid:

- Did you implement both the **TipViewController** and the **SettingsViewController**?
- Did you successfully **push your code to github**? Can you see the code on github?

- Did you **add a README.md** to the repo on github which includes a **GIF walkthrough** of the app's functionality?
- Did you visit your application and **submit using the pre-work form**?

6. Extend your app, improve UI, add features

After initial submission, you should iterate by adding several additional features to your app. Engineers that submit apps with extended functionality and improved UI are **significantly more likely to be accepted** into the bootcamp. Try your hand at implementing the following user stories and any other extensions of your own choosing:

- Remember the bill amount across app restarts. After an extended period of time, clear the state. This is a UI trick that Apple uses with the Spotlight app. If you return there a minute later, it will show your most recent search. if you return 10 minutes later, it defaults to blank. To implement this, plug into the application lifecycle and track time using NSDate.
- Use locale specific currency and currency thousands separator.
- Add a light/dark color theme to the settings view. In viewWillAppear, update views with the correct theme colors.
- Make sure the keyboard is always visible and the bill amount is always the first responder. This way the user doesn't have to tap anywhere to use this app. Just launch the app and start typing.
- Add animations to your UI
- The Tip calculator has a very primitive UI. Feel free to play with colors, layout, or even modify the UI to improve it.

Please **update your submission** after pushing new features. Be sure to include in the README an updated GIF walkthrough using LiceCap.

Quick Jump