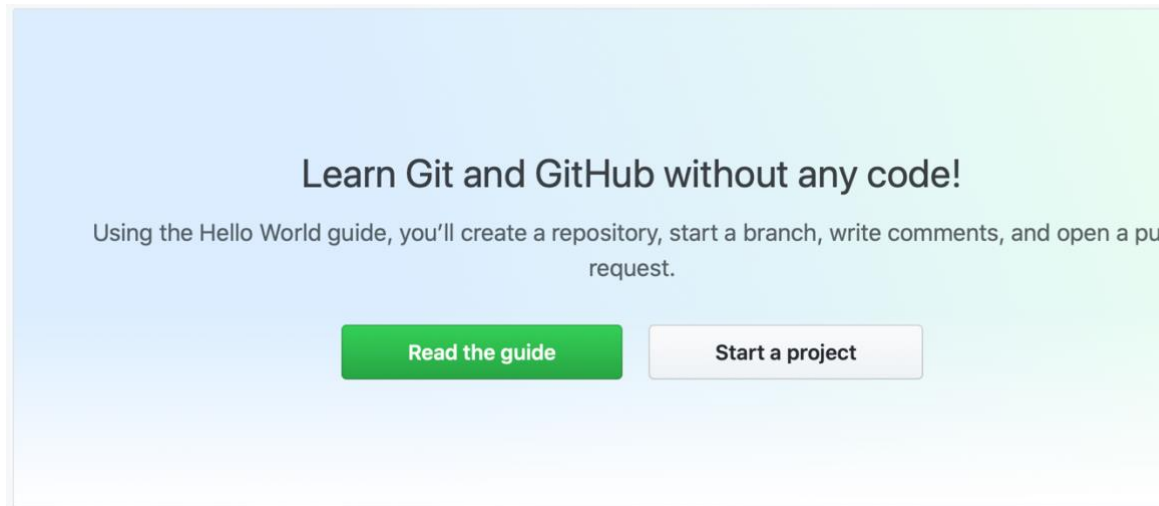


## Your First App - Follow this Step By Step

### Links:

1. Preface
2. Setting up your Environment
  - a. Github
    - i. Create an account
      1. <https://github.com/join>
    - ii. Create a repository



1. Follow steps in the new repo. You can make it private if you want no one to access it, however, many, if not most, repositories are

Public.

## Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository](#).

Owner

Repository name \*

 jaclary ▾ /

Great repository names are short and memorable. Need inspiration? How about **verbose-dollop**?

Description (optional)



**Public**

Anyone can see this repository. You choose who can commit.



**Private**

You choose who can see and commit to this repository.

Skip this step if you're importing an existing repository.



**Initialize this repository with a README**

This will let you immediately clone the repository to your computer.

Add .gitignore: **None** ▾

Add a license: **None** ▾



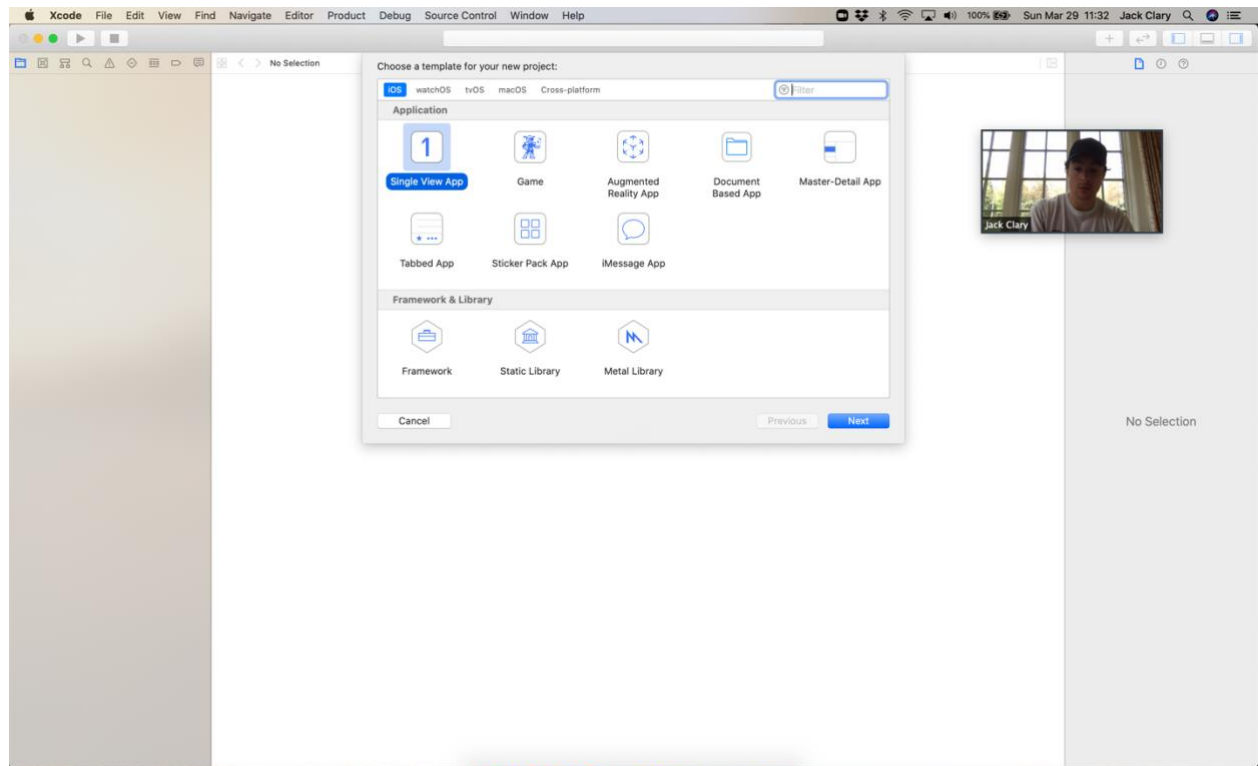
Create repository

a. Do not initialize a README. We will do this later via Terminal.

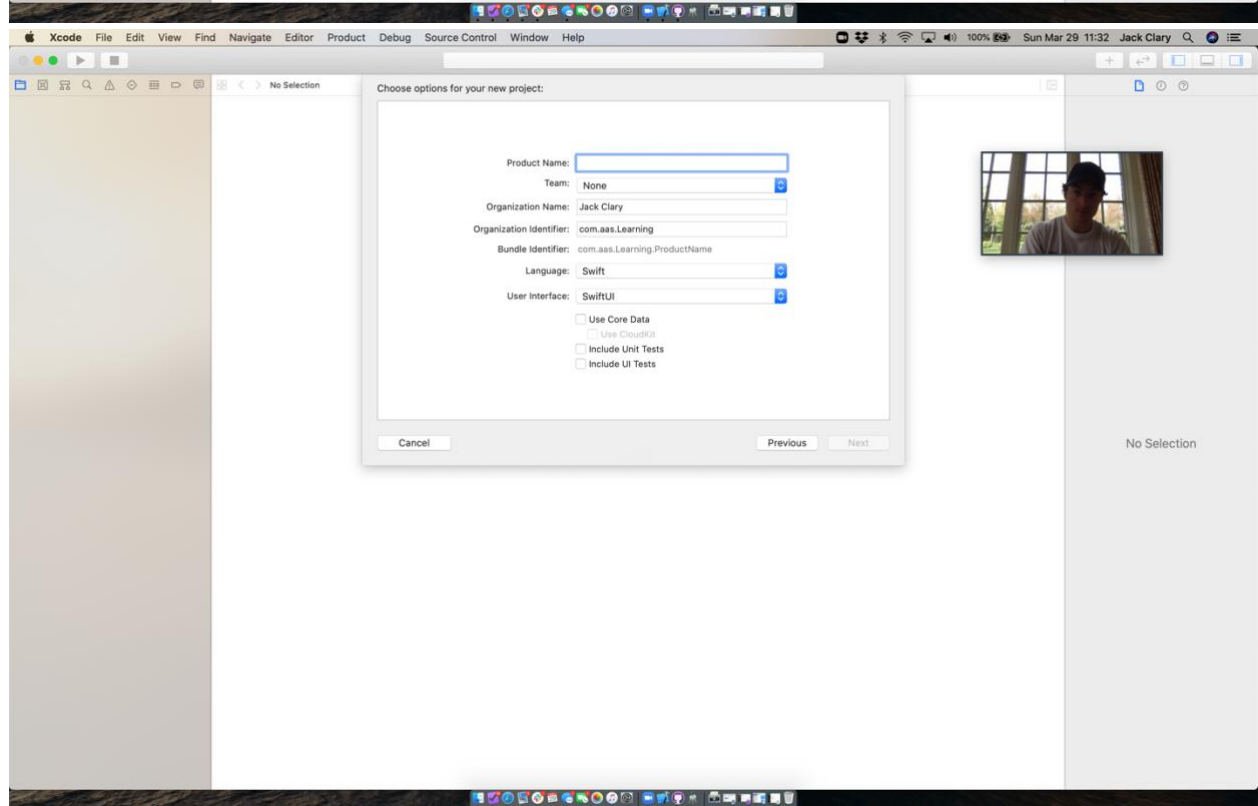
b. Now, go to Xcode

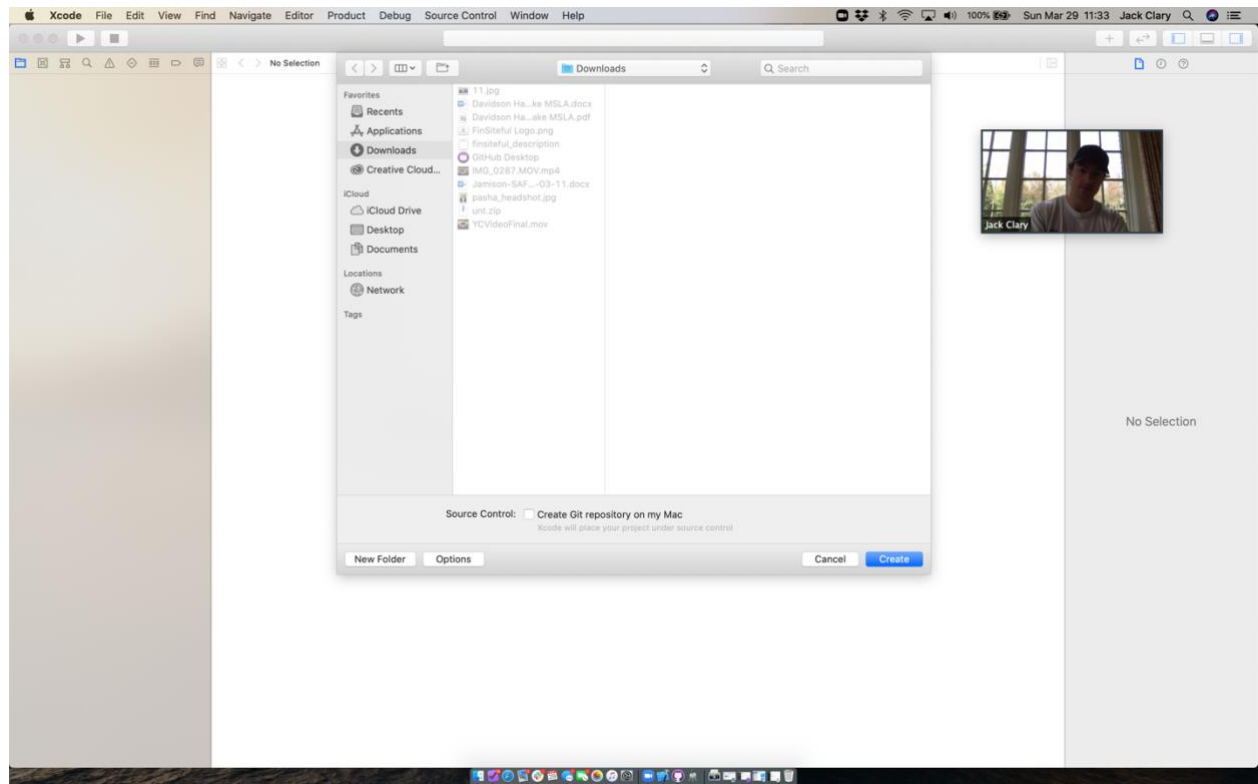
- i. Create project following screenshots



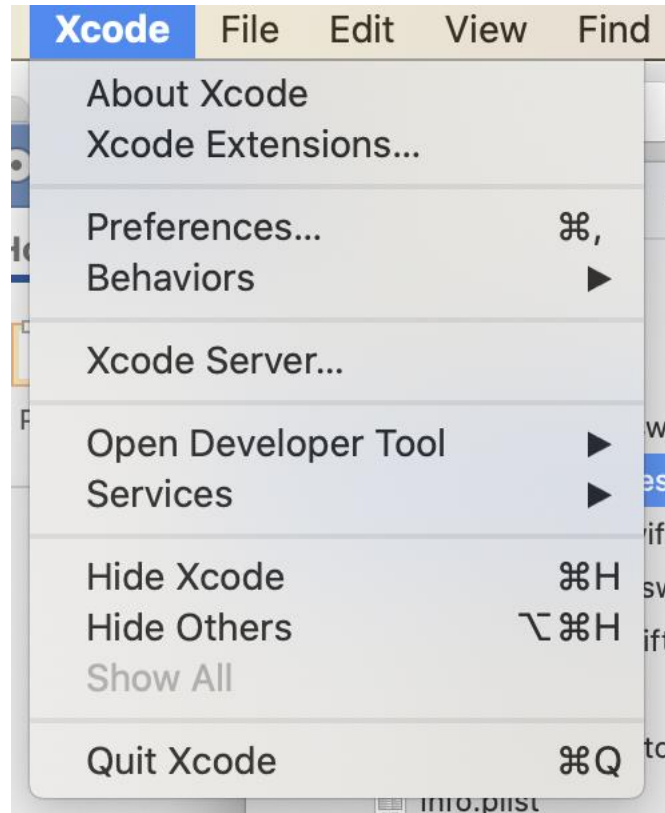


C.

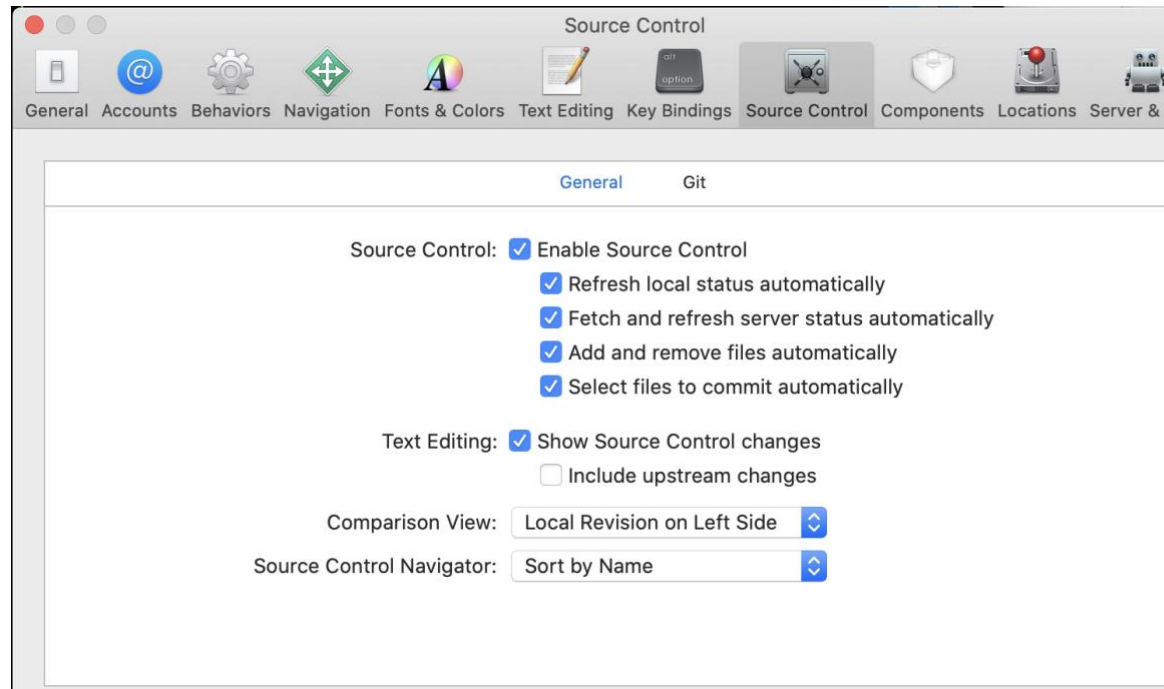




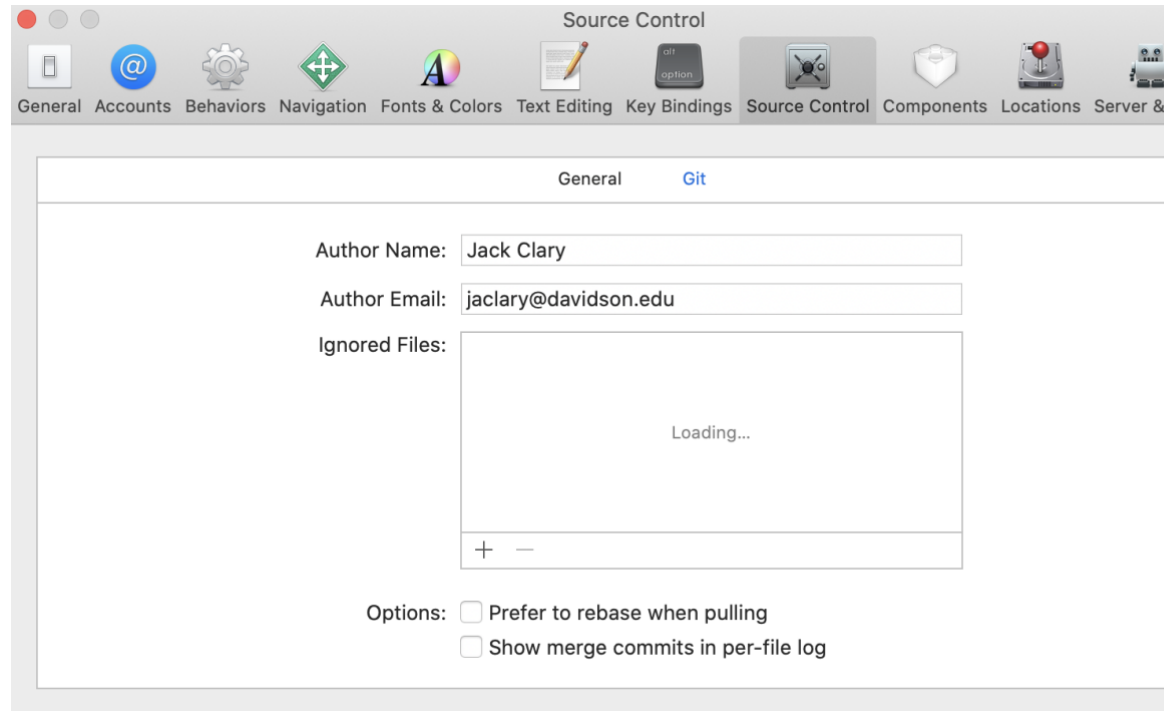
- i. Be sure to note what file location you saved it in. You need to be able to locate it via “Terminal” or “the command line” in the next step
- ii. Now you need to set up your Github in Xcode. (Xcode calls it source control sometimes). First, click on **Preferences** in the dropdown in the upper left of your screen.



- iii. Now, click “Source Control” in the upper center and make sure you have these boxes checked.



- iv. Lastly click the little “Git” and type in your information



- v.
- vi. You will have to type your Github account information later, but it can/will happen a few different times so just have it at the ready.
- d. Now, open Terminal
  - i. “ls” shows current directory location and its contents

```
jackclary@Hurc-III ~ % ls
Applications      Documents          Movies
Creative Cloud Files  Downloads          Music
Desktop           Dropbox            Pictures
Developer          Library            Public
jackclary@Hurc-III ~ %
```

- ii. “cd ‘filename’” moves you to said file or directory

```
jackclary@Hurc-III ~ % cd Developer
jackclary@Hurc-III Developer %
```

1. As you can see, my “directory” changed from “~” to “Developer”
  2. If you type in “**cd ..**” it will bring you back to the previous directory.
  3. If you type in “**cd**” it will bring you back to the “~” directory
- iii. So, cd to the directory you have your Xcode project saved in

```
jackclary@Hurc-III ~ % cd Documents/GitHub/swift_repo/Your\ First\
jackclary@Hurc-III Your First App % |
```

1. If the title of the directory is unique, you can press the **tab key** after you’ve typed in sufficient letters to auto-type the rest. This is useful when there are spaces like in “Your First ...” because they require “\” in terminal.
- iv. Use the following commands in terminal when you are in your directory. You know you are in your directory when it says the correct name to the left of the “%” (E.g. “Your First App” in my case)

**...or create a new repository on the command line**

```
echo "# test" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/jaclary/test.git
git push -u origin master
```

1. The link, i.e. <https://github.com/jaclary/test.git> will be different for you based on your github username and the name of the repository. This link can be found if you look at your browser’s search bar when you are on your repository page which will load after you create it in step 2.a.ii.1
3. Your First App Intro - Users can sign-up for your service
- a. After you start the project + initializing your repository
    - i. Creating a .gitignore
      1. “**cd**” to the repository location in terminal
      2. “**touch .gitignore**”
      3. “**open .gitignore**”
      4. This is basically a text file where each line represents a file or files you don’t want being put in your repository.



ii. What to type in it?



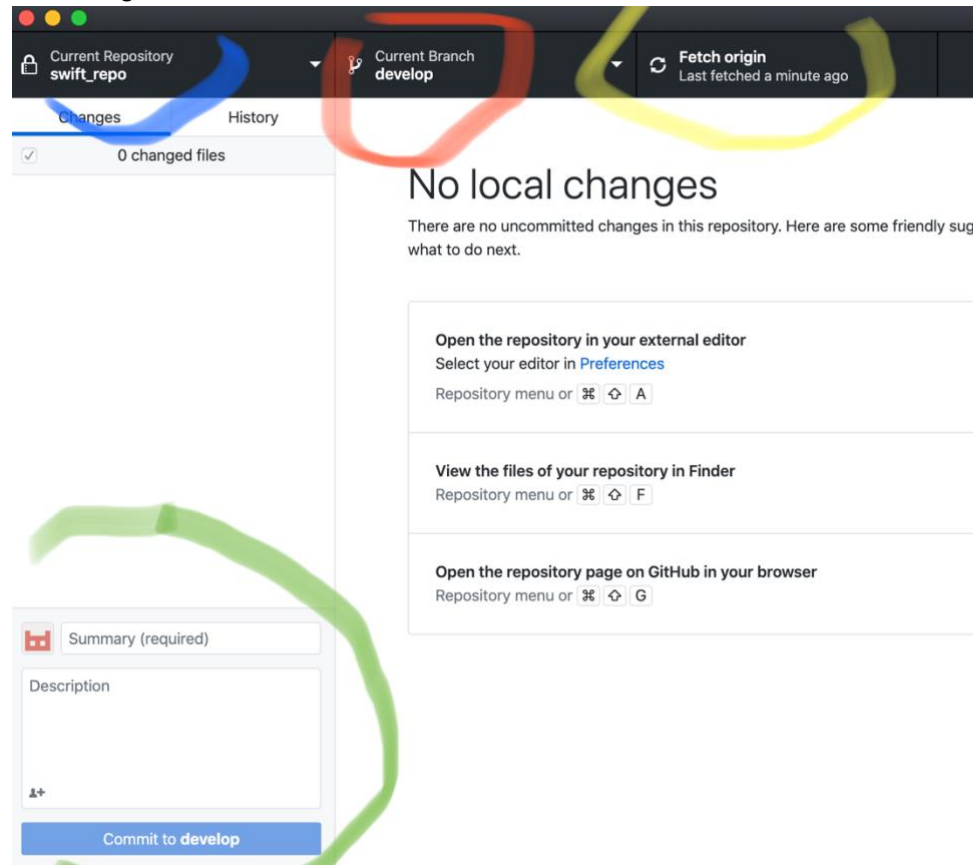
```
.gitignore
*.xcuserstate
*.xcuserdata
*.DS_Store
```

1. \*.xcuserstate
2. \*.xcuserdata
3. \*.DS\_Store
4. The “\*” at the beginning means all files that contain the specified string.

iii. How to github?

1. Read this: <http://think-like-a-git.net>
2. Terminal vs. Xcode vs. Other Applications
  - a. Terminal is more difficult, but more powerful. Look here for a tutorial: <https://rogerdudler.github.io/git-guide/>
  - b. Xcode is more simple, but prone to bugs/errors.
  - c. Other applications like Github desktop and GitKraken are great, simple ways to deal with source control. For simplicities sake, download Github Desktop and use it.
  - d. <https://desktop.github.com>
3. GitHub Desktop
  - a. Follow the instructions when you first download it. Very simple.

b. For using it,



c. The blue represents the current repository.

d. The red is the current branch (

#### 4. Pull Requests

a. FLESH OUT

b. How do you Xcode?

- i. “cmd+r” runs your app in a virtual apple device
- ii. “cmd+b” builds the app, but doesn’t run it.
- iii. Preview
- iv. FLESH OUT

c. What are these initial files?

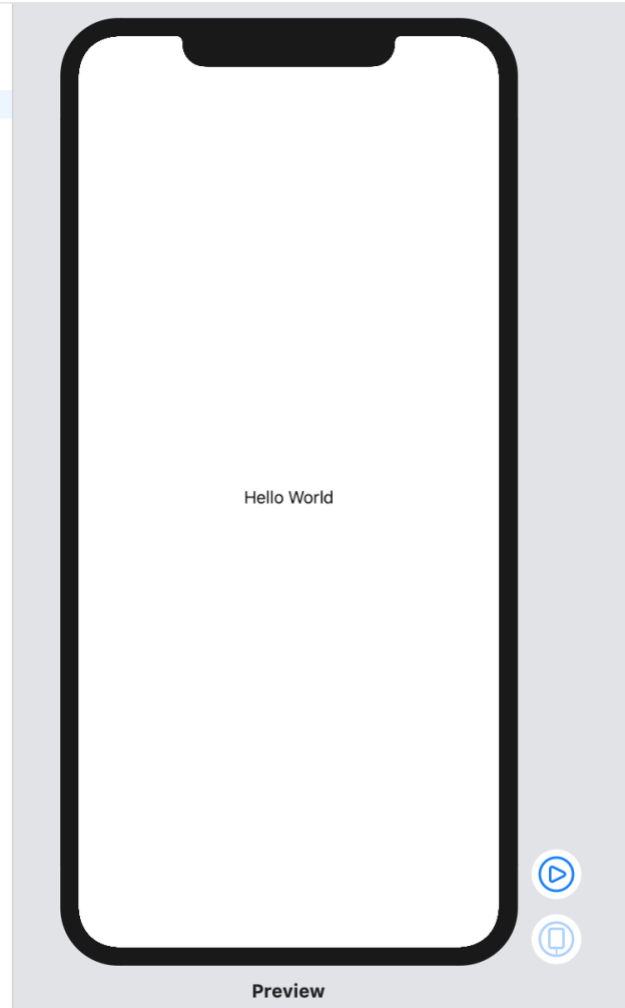
- i. AppDelegate
  1. Runs when your app starts, specifically the first function of the class runs (func application(\_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool)
  2. More information: <https://dev.to/theobendixson/comment/4g8j>
- ii. SceneDelegate
  1. Read here (also gives good information on AppDelegate): <https://www.donnywals.com/understanding-the-ios-13-scene-delegate/>
- iii. ContentView

1. ContentView is your first View, the technical word for a screen on your app. It is initialized, or started, as an empty view with the words "Hello World" in the center of the screen.
- iv. Assets.xcassets
  1. This is a folder that will hold assets like images. We won't go into detail here.
- v. LaunchScreen.storyboard
  1. This is the "splash screen" or the window that briefly appears as the app loads.
- vi. Info.plist
  1. This topic has a lot involved. We will discuss this when we add dependencies to the app.
- d. SignUpView
  - i. What are views in SwiftUI?
    1. <https://developer.apple.com/documentation/swiftui/view>
    2. VStack
    3. HStack
    4. Group
    5. Text
    6. TextField
  - ii. @State
    1. Storing user data
    2. Create a user class? (maybe throw all signed up users in a list that displays after they sign)
  - iii. NavigationView?
- e. SignUpViewModel

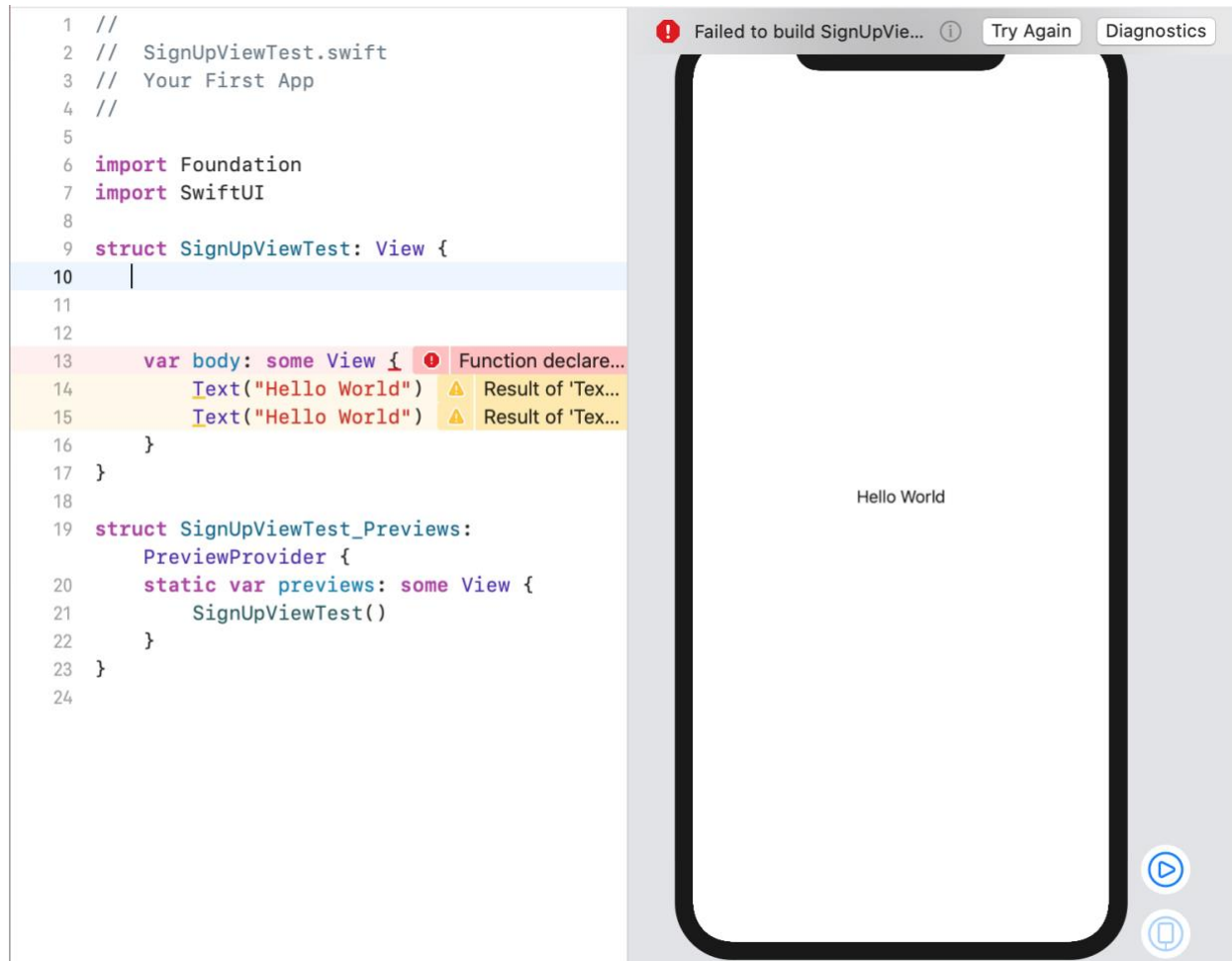
4. Let's code!

- a. Start by opening the SignUpViewTest.swift. You should see the below.

```
1 //  
2 // SignUpViewTest.swift  
3 // Your First App  
4 //  
5  
6 import Foundation  
7 import SwiftUI  
8  
9 struct SignUpViewTest: View {  
10  
11  
12  
13     var body: some View {  
14         Text("Hello World")  
15     }  
16 }  
17  
18 struct SignUpViewTest_Previews:  
19     PreviewProvider {  
20     static var previews: some View {  
21         SignUpViewTest()  
22     }  
23 }
```



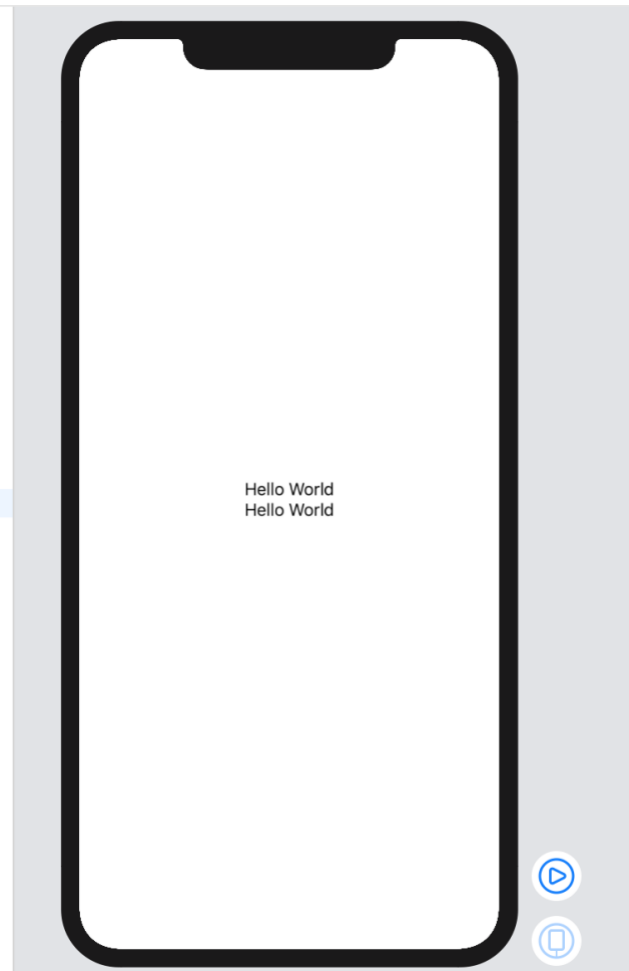
- b. To showcase how the “body” section of the page works, try duplicating line 14. Do you get the following error?



- i. You get this error because the body needs to return 1 thing (it does so implicitly, hence no “return Text(“Hello World”)”), and now it is returning 2 things. But what if you want it to say Hello World twice? Stacks!

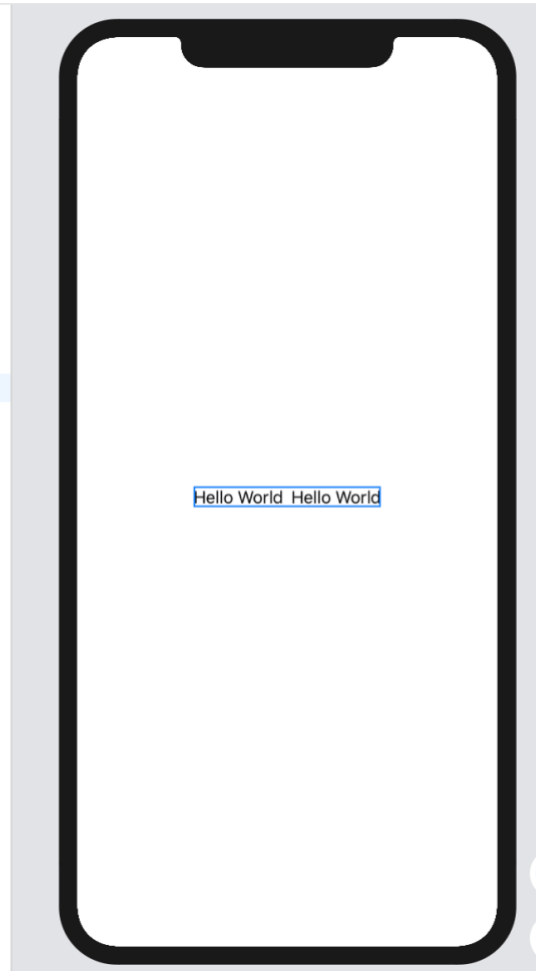
- c. If you wrap the two `Text()` in a stack, like vertical stack, or `VStack`, you should get the following

```
1  //
2  //  SignUpViewTest.swift
3  //  Your First App
4  //
5
6  import Foundation
7  import SwiftUI
8
9  struct SignUpViewTest: View {
10
11
12
13      var body: some View {
14          VStack {
15              Text("Hello World")
16              Text("Hello World")
17          }
18      }
19  }
20
21  struct SignUpViewTest_Previews:
22      PreviewProvider {
23      static var previews: some View {
24          SignUpViewTest()
25      }
26  }
```



- i. See how they're vertically aligned? If you wanted them horizontal, you can use a horizontal stack or HStack.

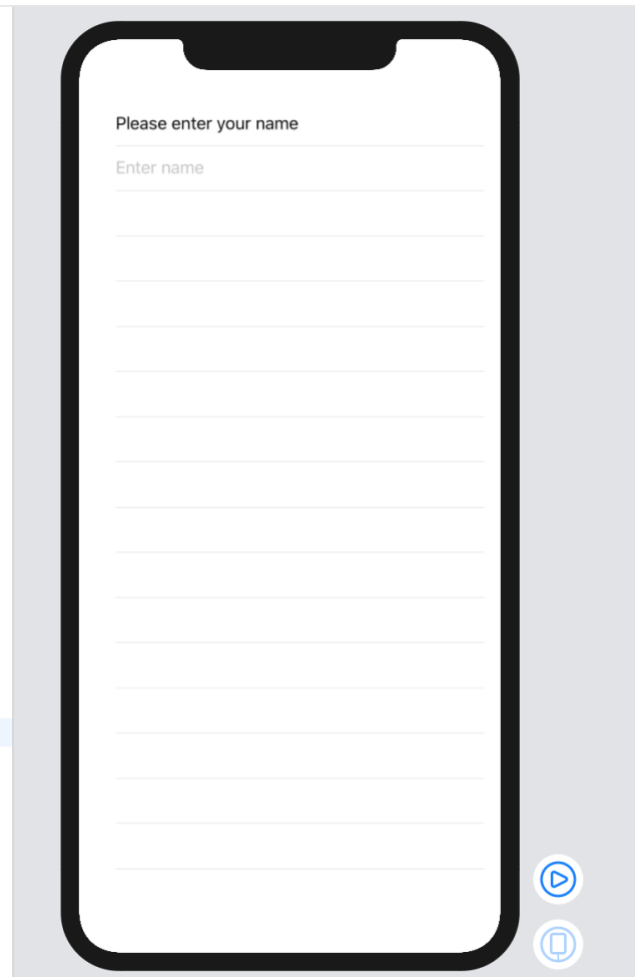
```
1 //
2 //  SignUpViewTest.swift
3 //  Your First App
4 //
5
6 import Foundation
7 import SwiftUI
8
9 struct SignUpViewTest: View {
10
11
12
13     var body: some View {
14         HStack {
15             Text("Hello World")
16             Text("Hello World")
17         }
18     }
19 }
20
21 struct SignUpViewTest_Previews:
22     PreviewProvider {
23     static var previews: some View {
24         SignUpViewTest()
25     }
26 }
```



- 1. As a side note, if you are running the preview (option + cmd + return and then option + cmd + p) and you click on a section of the code, like the HStack here, the preview will highlight that portion of the screen.
- d. Now let's begin the real fun. We like to define features as User Stories because they allow us to be flexible in the way we implement our solutions. For this story, let's define it like this:
  - i. Our user opens the app and can type in their name.

- e. To do this we are going to embed or wrap our text, that tells the user what to do, with a `TextField`, that allows the user to type in information. FLESH OUT

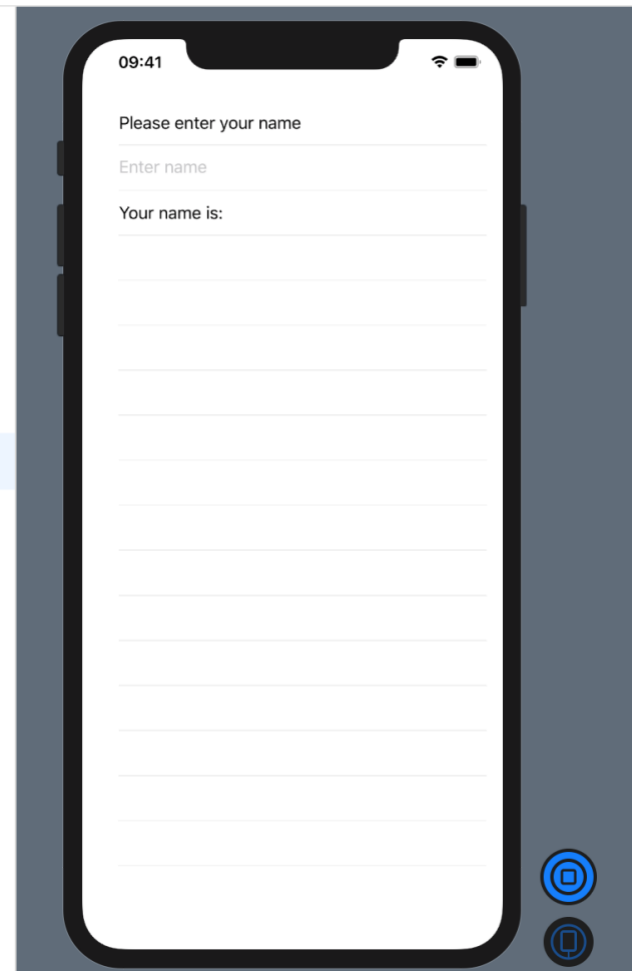
```
1 //
2 //  SignUpViewTest.swift
3 //  Your First App
4 //
5
6 import Foundation
7 import SwiftUI
8
9 struct SignUpViewTest: View {
10     @State var name: String = ""
11
12
13     var body: some View {
14         List {
15             Text("Please enter your name")
16             TextField("Enter name", text:
17                 $name)
18             Spacer()
19         }.padding()
20     }
21
22
23
24
25 |
26 struct SignUpViewTest_Previews:
27     PreviewProvider {
28         static var previews: some View {
29             SignUpViewTest()
30         }
31     }
```





- f. Now that we can type in a name, let's present that name back to the user. Since we stored the name in the variable aptly titled "name" we can display it like this:

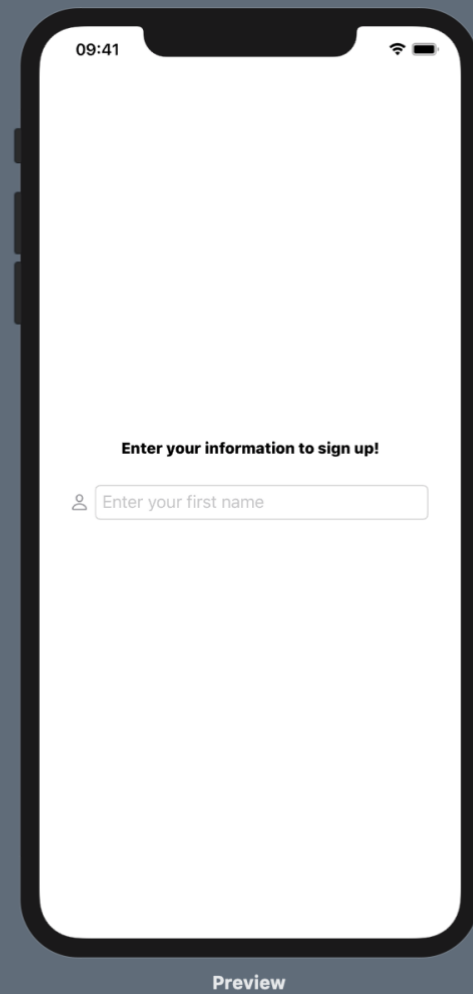
```
1 //
2 // SignUpViewTest.swift
3 // Your First App
4 //
5
6 import Foundation
7 import SwiftUI
8
9 struct SignUpViewTest: View {
10     @State var name: String = ""
11
12     var body: some View {
13         List {
14             Text("Please enter your name")
15             TextField("Enter name", text:
16                 $name)
17             Text("Your name is: \(name)")
18             Spacer()
19         }.padding()
20     }
21 }
22
23
24
25
26
27 struct SignUpViewTest_Previews:
28     PreviewProvider {
29     static var previews: some View {
30         SignUpViewTest()
31     }
32 }
```



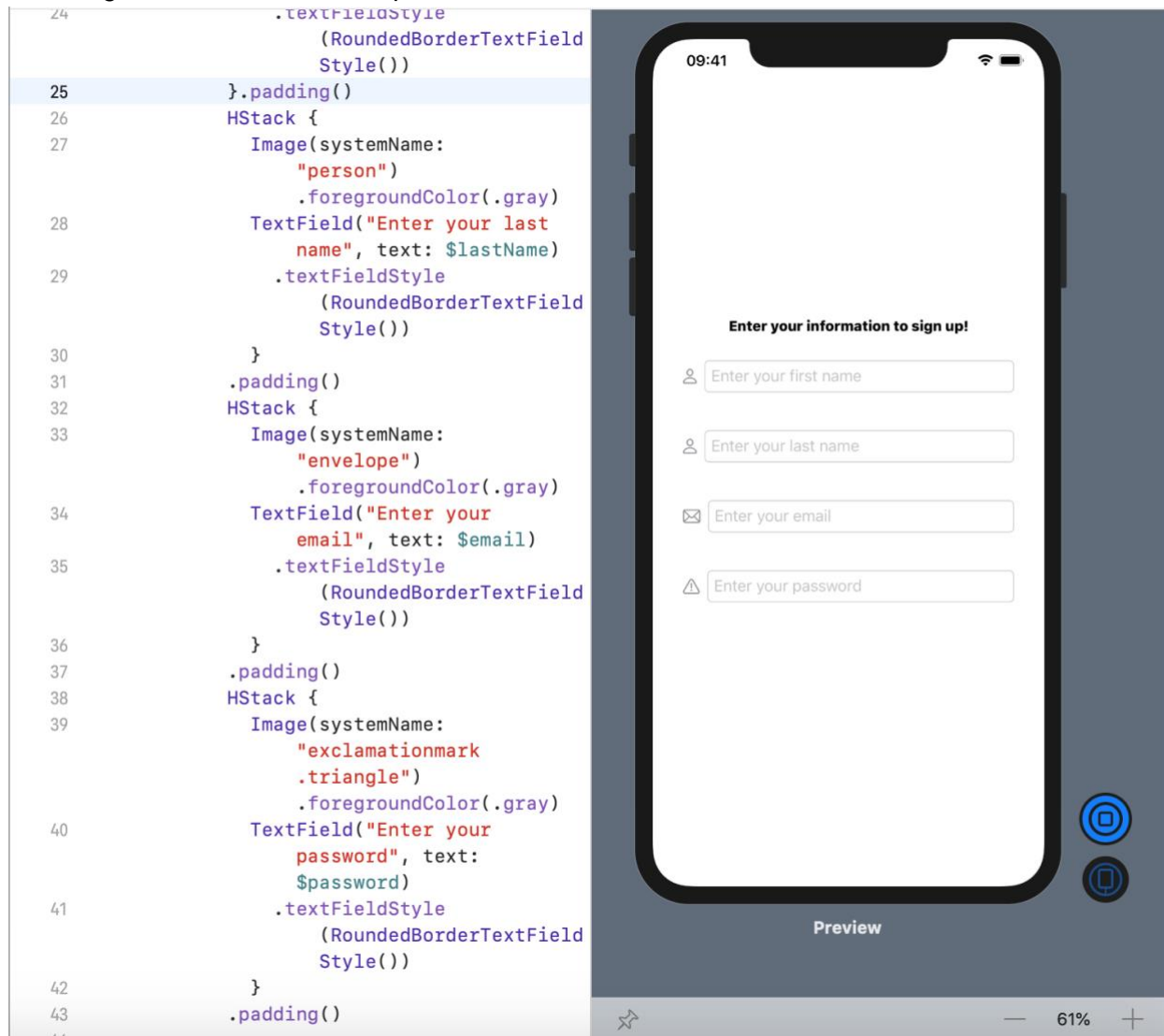
- g. However, this looks terrible. To begin adding some design let's add a few more variables, things like email, and some styling.
- The styling we will add simply showcases some simple tools you have at your disposal like the alignment parameter in the VStack (and in the other

stacks), `fontWeight`, `font`, `foregroundColor`, `textFieldStyle`, and `padding`.

```
1 //
2 // SignUpViewTest.swift
3 // Your First App
4 //
5
6 import Foundation
7 import SwiftUI
8
9 struct SignUpViewTest: View {
10     @State var firstName: String = ""
11     @State var lastName: String = ""
12     @State var email: String = ""
13     @State var password: String = ""
14
15
16
17
18     var body: some View {
19         VStack(alignment: .center) {
20             Text("Enter your information
21                 to sign up!").fontWeight(
22                     .heavy).font(.system(size:
23                         15))
24             HStack {
25                 Image(systemName:
26                     "person")
27                     .foregroundColor(.gray)
28                 TextField("Enter your first
29                     name", text: $firstName)
30                     .textFieldStyle
31                     (RoundedBorderTextField
32                     Style())
33             }
34             .padding()
35         }
36         .padding()
37     }
38 }
```



- h. Now that we've done this for our first name, lets do it for all the new variables including last name, email, and password.



- i. As you can see, we HStack each textField so that the image appears in-line, or horizontally, with the textfield.
- j. Now that we can enter our information, let's make it so that after we do, it appears at the bottom of the screen. To do this, we need to create a variable to tells us whether or not we should show that part of the screen.

```
@State var showScreen = false
```

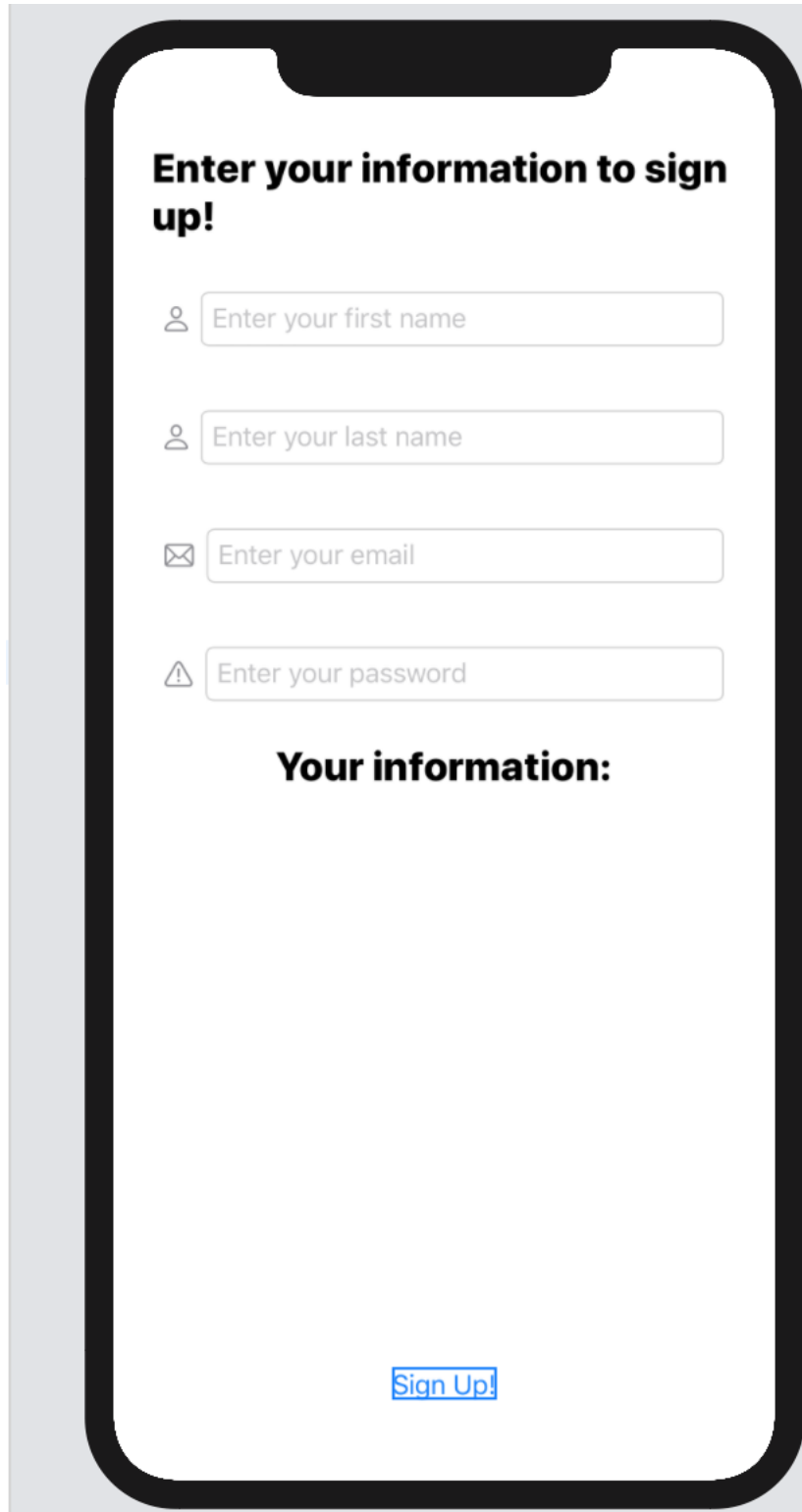
- k. Now, let's attach that variable to a Button, so that when we press it, it shows the

```
Button(action: {  
    self.showScreen.toggle()  
}) {  
    Text("Sign Up!")  
}
```

screen.

- i. Essentially what this does: creates a button where the action of pressing on it causes the showScreen variable to "toggle()" or switch from it's current state of false to true (or vice versa).

- I. By pressing it the your information section will appear.



**Enter your information to sign up!**

**Your information:**

[Sign Up!](#)

