**Mobile Computing – iOS Fall’23**

**Assignment04**

**30 Points**

**Please follow the following instructions to complete this assignment.**

1. Open Xcode from the launchpad of your Mac. Clone your private GitHub repository.
2. In that repo, create a new Xcode project. Select the iOS platform and click on the App template.
3. Click on next which will prompt you to choose options for the project.
4. Provide product name as **LastnameShoppingApp**, “**edu.nwmissouri.fall23.cs44643**” for organization identifier, “**Storyboard**” as interface and “**Swift**” as the language.
5. Click on next and select an appropriate location to save your app and click on Create. A project directory will be loaded.
6. From the project navigator click on “Main.storyboard” file, a blank mobile screen will be loaded, where the required fields for an app need to be added.

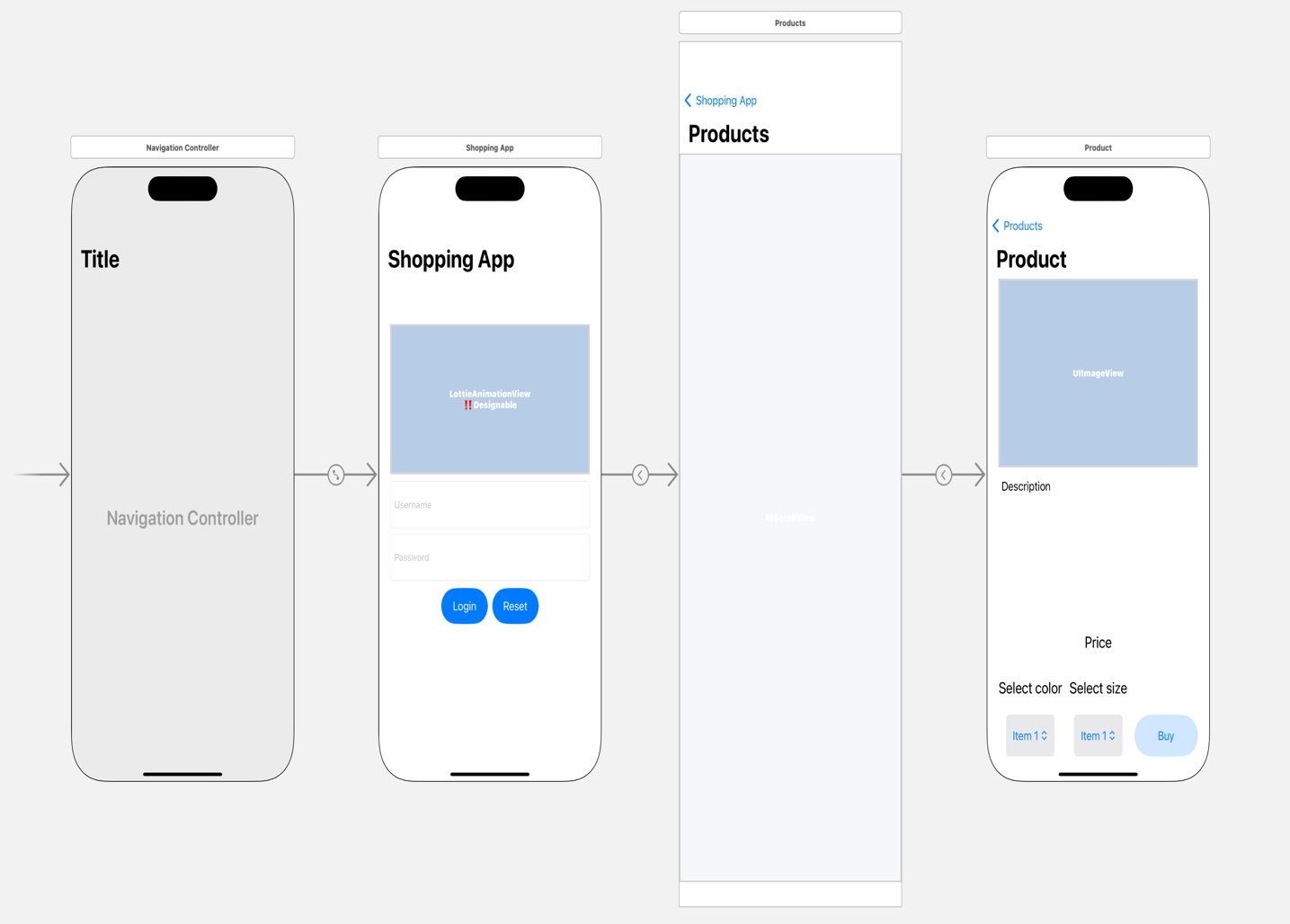
**The View**

1. At first, you will see a single view (first view) in the Main.storyboard file. Select that view and embed it in a navigation controller.
   1. Editor > Embed In > Navigation Controller
2. Check Prefers Large Title property for the navigation bar.
3. Create a Cocoa Touch Class “LoginVC” that is a sub class of UIViewController and assign it as class to the first view that is present in the Main.storyboard file. It will be the root view controller of the navigation controller.
4. Add two more view controllers to the storyboard for product view and product description view, respectively.

Table 1 The View Controllers

|  |  |
| --- | --- |
| **Screen** | **View Controller** |
| 1 | LoginVC |
| 2 | ProductsVC |
| 3 | ProductDescriptionVC |

1. Refer to Table 1 to create required classes and assign them to proper view controllers in the storyboard.
   1. To do it, repeat Step 3 for ProductsVC and ProductDescriptionVC.



3

2

1

Segue2

Segue1

Figure 1 The View

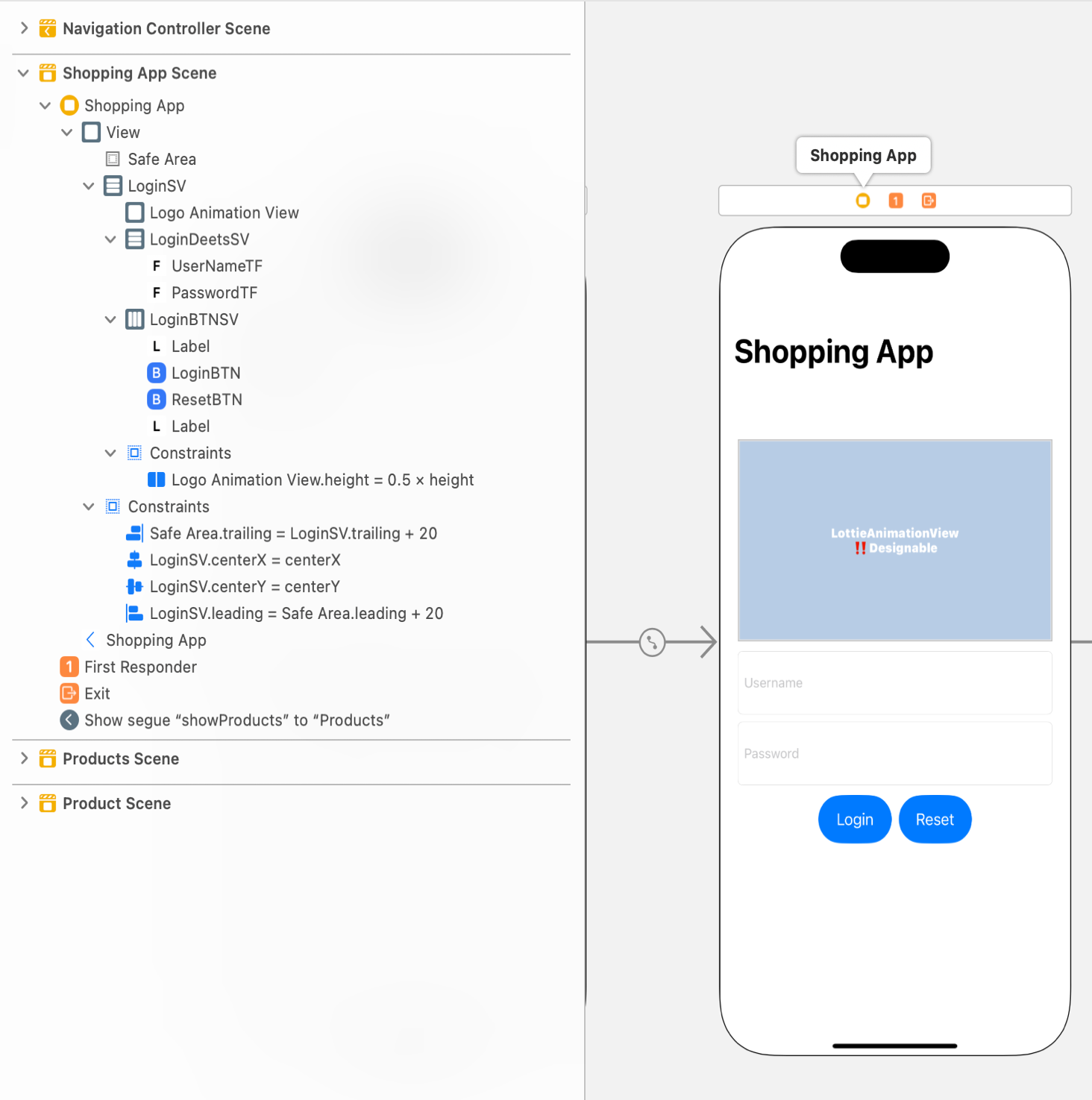


Figure 2 Auto Layout for LoginVC

Table 2 LoginVC's properties for stack views

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Stack View** | **Axis** | **Alignment** | **Distribution** | **Spacing** |
| LoginSV | Vertical | Fill | Fill | Standard |
| LoginDeetsSV | Vertical | Fill | Fill Equally | Standard |
| LoginBTNSV | Horizontal | Fill | Fill Equally | Standard |

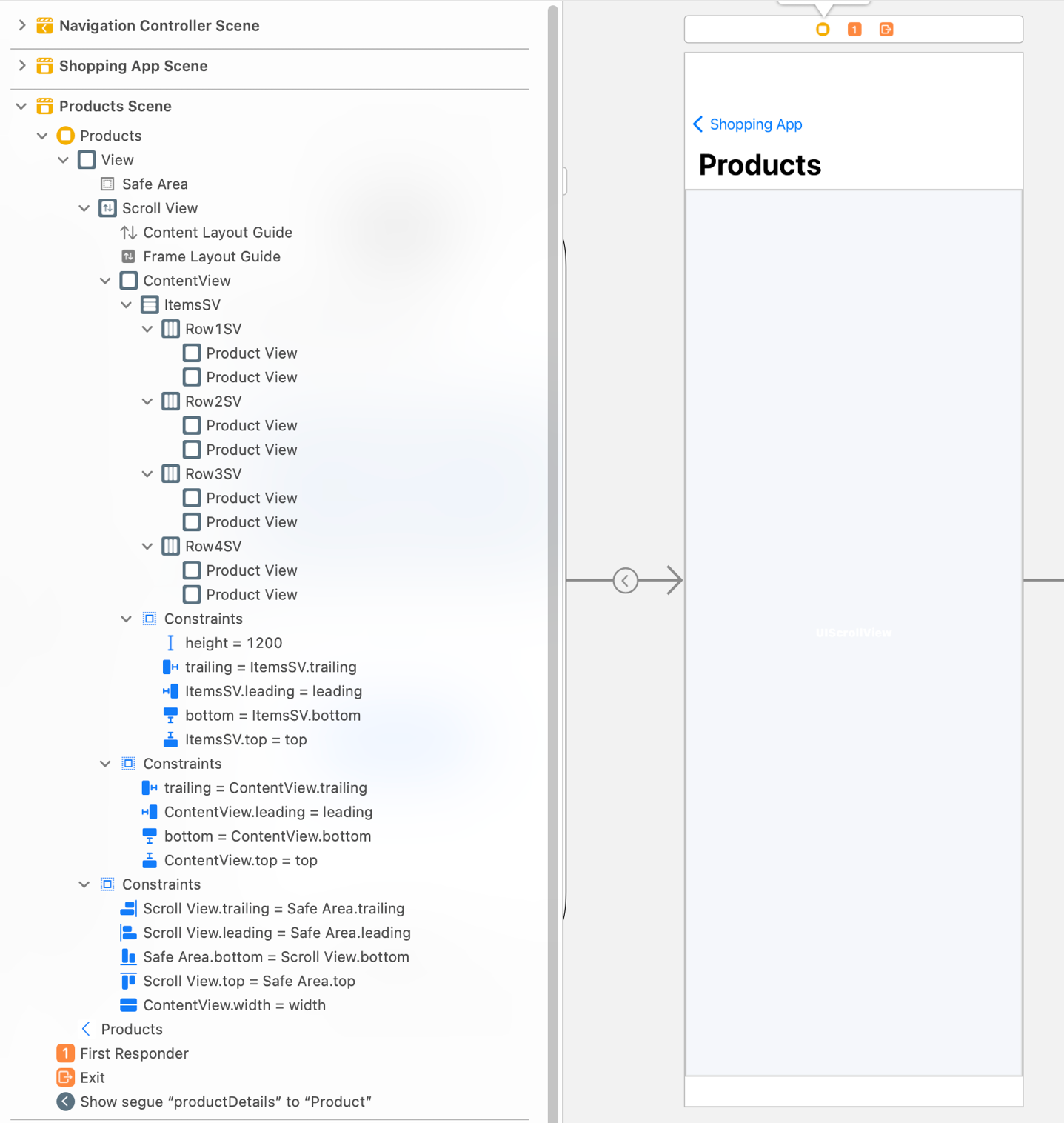


Figure 3 Auto Layout for ProductsVC

Table 3 ProductsVC’s properties for stack views

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Stack View** | **Axis** | **Alignment** | **Distribution** | **Spacing** |
| ItemsSV | Vertical | Fill | Fill Equally | Standard |
| Row1SV, Row2SV, Row3SV, Row4SV | Horizontal | Fill | Fill Equally | Standard |

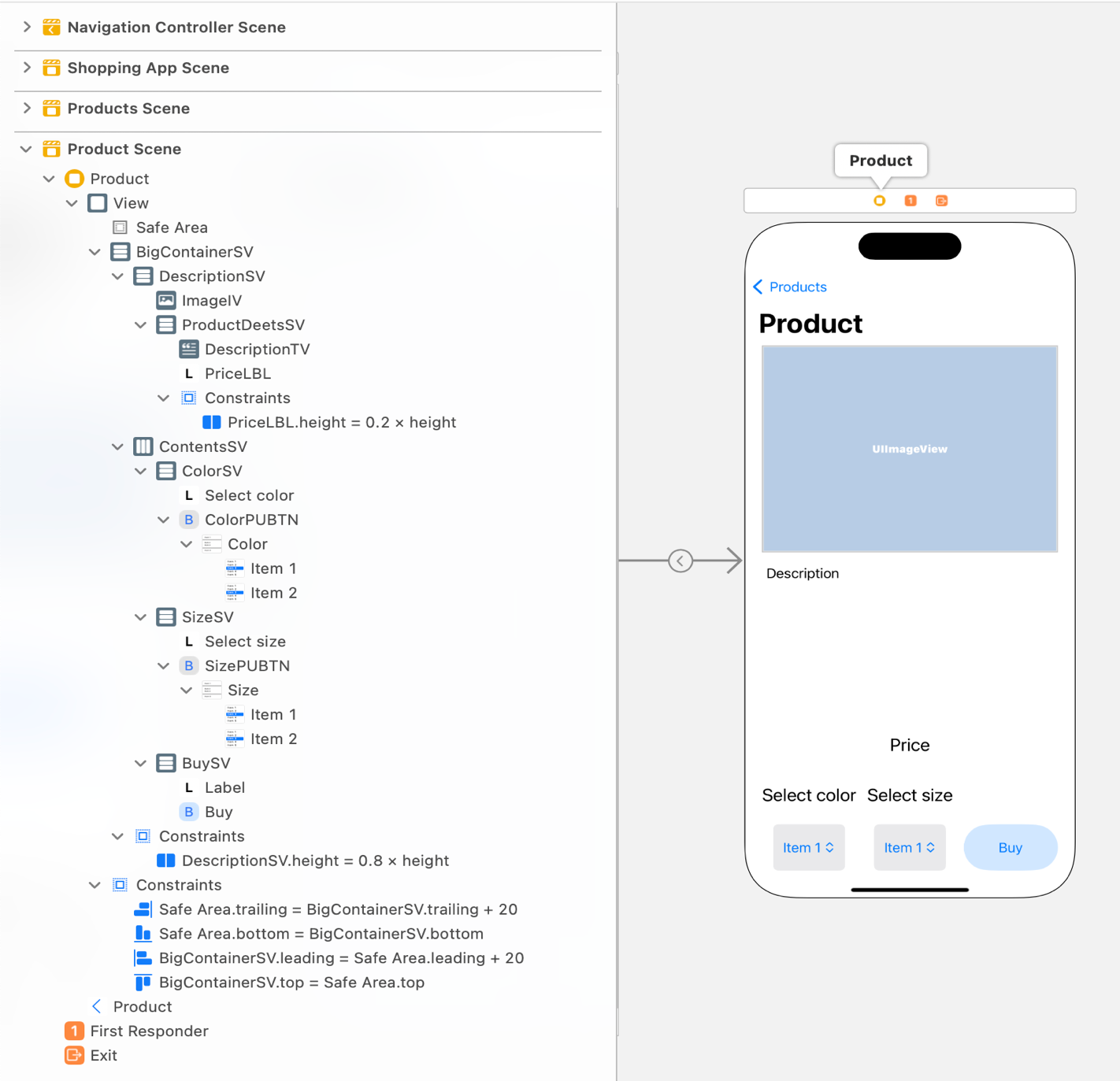


Figure 4 Auto Layout for ProductDescriptionVC

Table 4 ProductDescriptionVC’s properties for stack views

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Stack View** | **Axis** | **Alignment** | **Distribution** | **Spacing** |
| BigContainerSV, ProductDeetsSV | Vertical | Fill | Fill | Standard |
| DescriptionSV, BuySV | Vertical | Fill | Fill Equally | Standard |
| ContentsSV | Horizontal | Fill | Fill Equally | Standard |
| ColorSV, SizeSV | Vertical | Center | Fill Equally | Standard |

A screenshot of a computer

Description automatically generated

Figure 5 Product View

Table 5 UI elements configuration for ProductView

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Stack View** | **Axis** | **Alignment** | **Distribution** | **Spacing** |
| BigContainerSV, RecommendLikeSV | Vertical | Fill | Fill | Standard |
| NamePriceSV | Vertical | Center | Fill Equally | Standard |

1. Design the views as per the Figures 1, 2, 3, 4, and 5.

Table 6: UI elements configuration for LoginVC

|  |  |  |
| --- | --- | --- |
| **UI element** | **Purpose** | **Outlet/action name** |
| 2 text fields | To enter username | usernameTF |
| To enter password | passwordTF |
| 2 UI Button | To verify and login the user | loginBTN  login: |
| To reset the text fields | resetBTN  reset: |
| 1 LottieAnimationView | To display an animation in login screen | logoAnimationView |

Table 7 UI elements configuration for ProductsVC

|  |  |  |
| --- | --- | --- |
| **UI element** | **Purpose** | **Outlet/action name** |
| 1 scroll view | To configure it | scrollView |
| 1 normal view | For zooming in and out | contentView |
| 1 outlet collection for Custom View | To group all the product views | products |

Table 8 UI elements configuration for ProductDescriptionVC

|  |  |  |
| --- | --- | --- |
| **UI element** | **Purpose** | **Outlet/action name** |
| 1 image view | To display an image | productIV |
| 1 label | To display the price of the product | priceLBL |
| 2 pop up buttons | To show colors menu | colorPUBTN |
| To show size menu | sizePUBTN |
| 1 text view | To display text related to the product | descriptionTV |
| 1 button | To perform buy action | buy: |

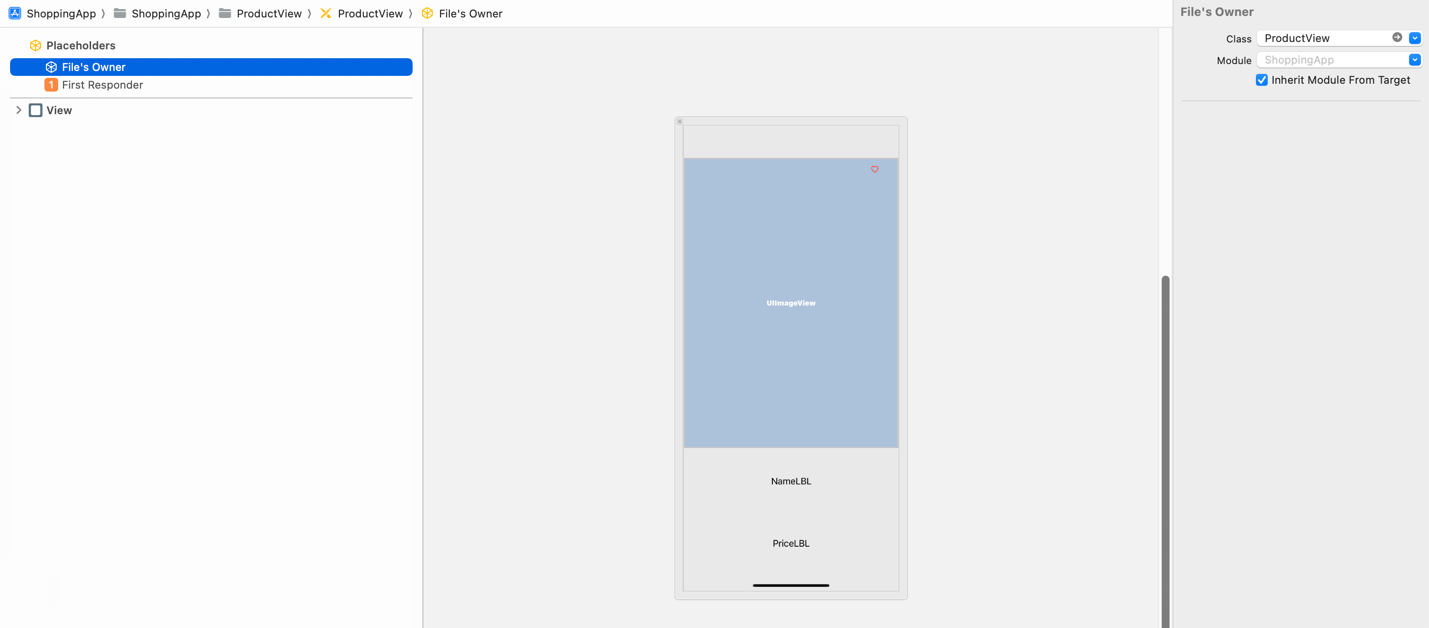
Table 9 UI elements configuration for ProductView

|  |  |  |
| --- | --- | --- |
| **UI element** | **Purpose** | **Outlet/action name** |
| 1 image view | To display an image | productIMG |
| 5 labels | To display a like symbol (♡) | likeLBL |
| To display the name of the product | productNameLBL |
| To display the price of the product | productPriceLBL |
| To display whether the product is recommended or not | recommendedLBL |

*Note: Define appropriate methods wherever required to achieve the required functionality via @IBActions and @Objc functions.*

**A custom UIView**

1. Create a custom view to configure the look of each product in the shopping app.
2. First create a class ProductView that is a subclass of UIView. Next, create a user interface view file and name it as ProductView as well, but the file type will be a .xib.



Set this to ProductView.

Figure 6 The ProductView’s .xib file

1. Design the UI and apply auto layout constraints as shown in the Figure 5.

**The Model**

1. Create a Swift file called Model.swift that has below structs.



1. Property items is an array of Item elements. Each item contains an image (that you provide in the app Assets folder), and its details. (Choose at least 8 products of your choice.)
   1. Note: For each image in the Assets folder, provide @2x and @3x versions.
2. When creating a new product only choose the values for price, season, and material from the keys present in the dictionaries priceTypes, seasonTypes, and materialTypes, respectively.
   * 1. Note: Values are case sensitive, for instance, a price tag low is not same as Low.

**The Controller**

1. Apply an animated gradient as background to all views in your app. It should work properly in all orientation modes. Pick variety of colors required for the animated gradient.
   1. Use the Swift package AnimatedGradientView available at <https://github.com/rwbutler/AnimatedGradientView.git>
2. Add Lottie package to the Project using the GitHub link <https://github.com/airbnb/lottie-ios.git>. The animation used in LoginVC are provided in the assignment zip folder.
3. ***ProductView***
   1. Define the required initializers for the ProductView.
   2. Set the following properties for the product image.
      1. Border color to white.
      2. Border width to 2.
      3. Corner radius to 20
   3. Add a single tap gesture to the likeLBL that shows or hides the like symbol ♥️.
      1. Play a system sound (1105) when the user taps the likeLBL.
   4. By default, set the likeLBL to empty heart “♡”.
4. ***LoginVC***
   1. Play the Lottie animation loginAnimation.json only once whenever the LoginVC is appeared.
   2. The passwordTF should be enabled only if username is “admin”.
   3. The resetBTN should be enabled only if username is not empty.
   4. The loginBTN should be enabled only if username is “admin” and password in “password”.
      1. To control the enabling of UI elements via text entered in the text fields use IBAction(s) on text fields for the event type Editing Changed.
   5. On tapping the loginBTN, perform a segue (i.e., Segue1 with identifier “showProducts”) to ProductsVC as shown in Figure 1.
   6. On tapping the resetBTN, the controller should get back to its initial state.
      1. Clear all text fields, and disable passwordTF, loginBTN, and resetBTN.
5. ***ProductsVC***
   1. Display the products in a 2-column style using the custom ProductView.
      1. If needed, check the Clips to Bounds property for the custom view.
   2. Add the DCBT500.mlmodel file to your Xcode project. Use the Core ML model to predict whether a product can be recommended to the user or not.
   3. ML model accepts input for a test sample DCBT500Input(price: Int64, rating: Double, season: Int64, material: Int64).
   4. Once all the products are loaded, for each of them, convert its price, season, and material values to Int64 type using the dictionaries priceTypes, seasonTypes, and materialTypes, respectively.
   5. Use the Core ML model to predict recommendation for products.
   6. Based on the prediction, using the recommendedLBL
      1. Display the text “Recommended 👍🏻” in green color if a product is recommended.
      2. Display the text “Not recommended 👎🏻” in green color if a product is recommended.
   7. For every product image in the products, add a long press gesture that performs a segue (i.e., Segue2 with identifier “productDetails”) to ProductDescriptionVC as shown in Figure 1. Moreover, play a system sound (1306) when long press is recognized.
      1. Make sure to enable user interaction for the product image.
      2. Set content mode of the product image to “Scale to Fill”
   8. While performing the segue, send the long pressed ProductView’s tag value to the ProductDescriptionVC to display the details of the product using the database UtilityConstants.items
   9. Enable zooming feature for the scroll view.
      1. Set scrollView minimum zoom scale to 1/25 and maximum zoom scale to 1.0
6. ***ProductDescriptionVC***
   1. Set the navigation area title to the product name.
   2. When the view is loaded, show the selected product’s image, description, and price.
   3. Add a double tap gesture to the product image to see the product image in full screen. Once it is in the full screen mode, a double tap on it will make product image exit from the full screen mode.
      1. Use animation with an option transitionCrossDissolve when entering and exiting full screen mode,
   4. Apply a corner radius of 20.0 to the product image view.
   5. Add 2 pop up buttons (use Xcode library) to select product color and size. Design the choices for each selection, as shown in the sample output.
   6. Display an alert on tapping on the buy button with the text “Thanks for purchasing the product. We will send you the shipping information soon.”

**Submission:** Push your entire Xcode project to your private GitHub repo and submit your private repository link on the Canvas via Text Entry.