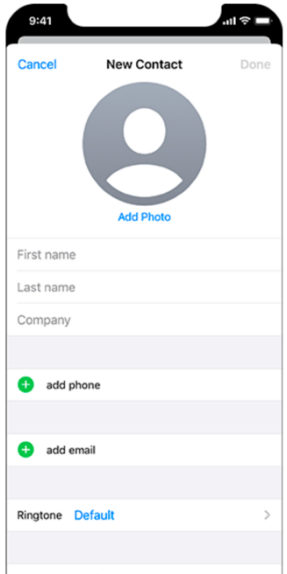
Alerts

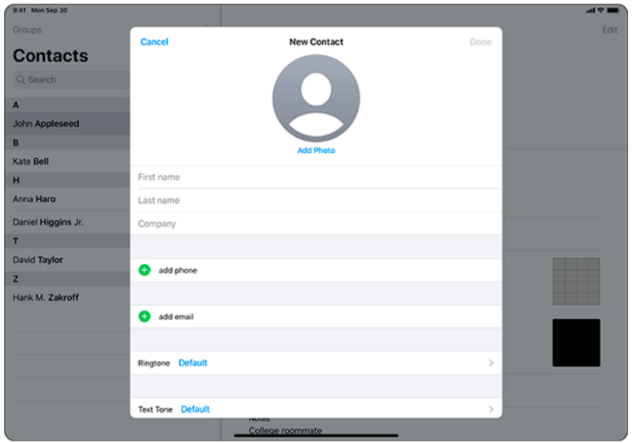
Presenting View Controllers

Presenting View Controllers

We are skipping Saving States (read Ch 13 on your own to follow the example).

iOS apps often present users with a view controller showing an action they must complete or dismiss. For example, when adding a new contact on iPhone, users are presented with a screen to fill out the contact's details. We call this kind of presentation modal, as the application is being put into a different "mode" where a set of actions become the focus. The user must interact with the modally presented view controller before proceeding.



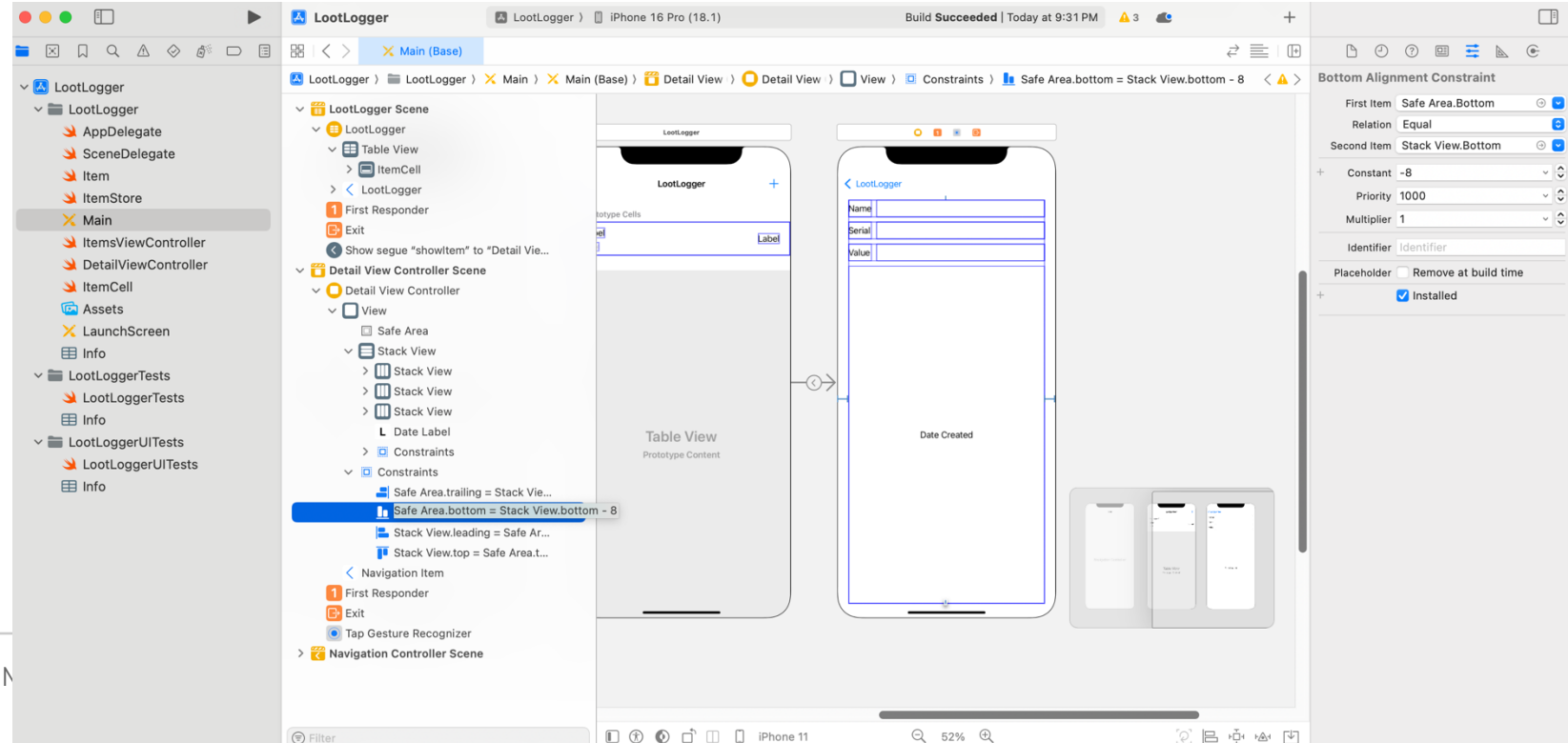


Presenting View Controllers

We will extend the LootLogger application to add the ability for users to associate a photo with each of their items. We will present the user with the option to select a photo from either the camera or the device's photo library.

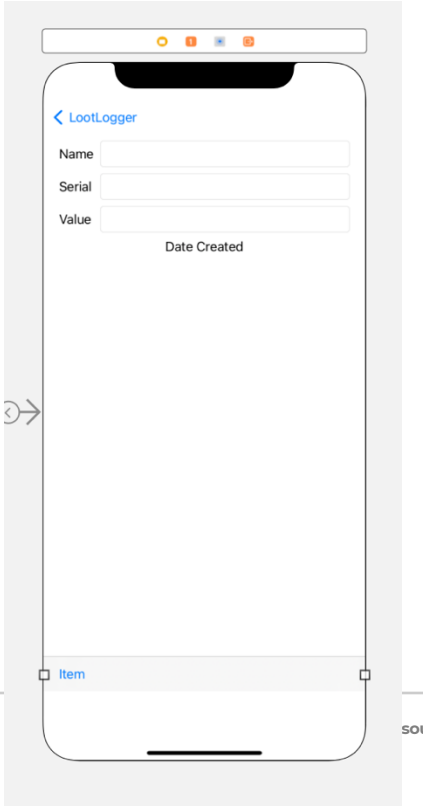
Presenting View Controllers

Open LootLogger.xcodeproj and navigate to Main.storyboard. In the detail view controller, select the bottom constraint for the outer stack view and press Delete to remove it. The stack view will resize itself, which will make some room for the toolbar at the bottom of the screen.



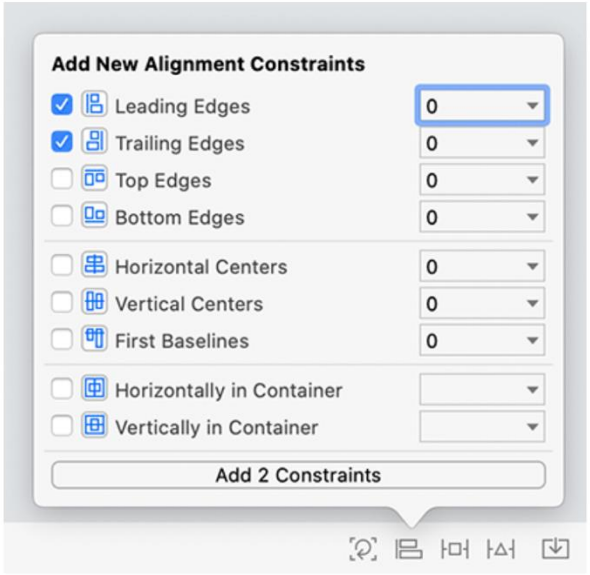
Presenting View Controllers

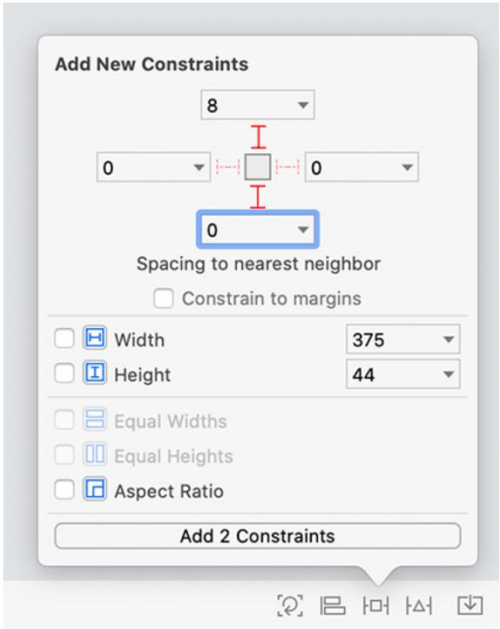
Now, drag a toolbar from the library and place it near the bottom of the view.
Make sure it is above the Home indicator (the black bar along the bottom of the screen).



Presenting View Controllers

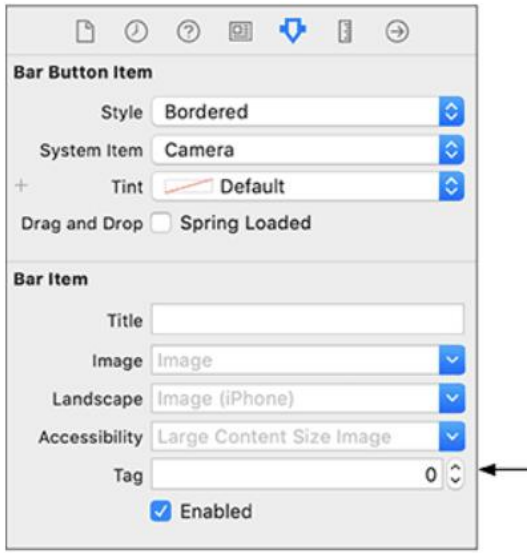
You want the toolbar to extend from the superview's leading edge to its trailing edge, independent of the safe area. To do this, select both the toolbar and the superview and open the Auto Layout Align menu. Configure the constraints as shown on the left:
Select only the toolbar this time and open the Auto Layout Add New Constraints menu. Configure the top and bottom constraints as shown on the right:

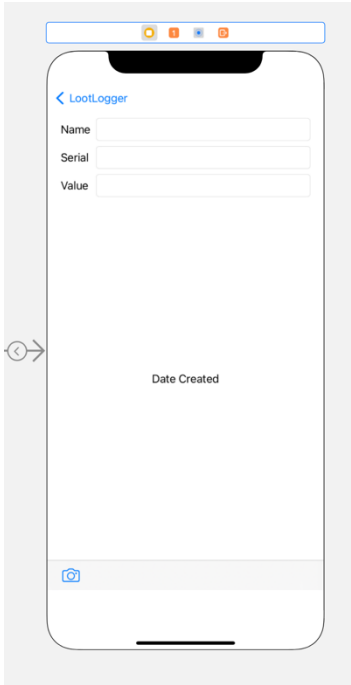




Presenting View Controllers

By default, a new instance of UIToolbar that is created in an interface file comes with one UIBarButtonItem. Select this bar button item and open the attributes inspector. Change the System Item to Camera, and the item will show a camera icon.





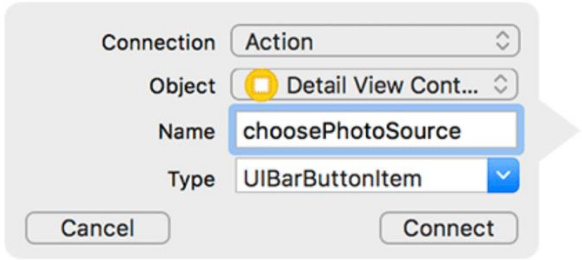
Presenting View Controllers

Build and run the application and navigate to an item's details to see the toolbar with its camera bar button item.

With **Main.storyboard still** open, **Option-click DetailViewController.swift** in the project navigator to open it in another editor.

In Main.storyboard, select the camera button in the document outline and Control-drag from the selected button to the DetailViewController.swift editor.

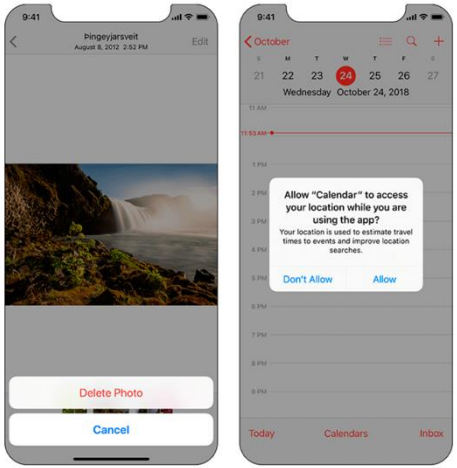
In the panel, select Action as the Connection,
name it choosePhotoSource,
select UIBarButtonItem as the Type, and click Connect



Alerts

To allow the user to choose a photo source, you will present an alert with the possible choices. Alerts are often used to display information the user must act on. When you want to display an alert, you create an instance of **UIAlertController** with a preferred style. The two available styles are UIAlertControllerStyle.actionSheet

The .actionSheet style is used to present the user with a list of actions to choose from. The .alert type is used to display critical information and requires the user to decide how to proceed. The distinction may seem subtle, but if the user can back out of a decision or if the action is not critical, then an .actionSheet is probably the best choice.



.actionSheet .alert

Alerts

In DetailViewController.swift, update choosePhotoSource(_:) to create an alert controller instance.

```
@IBAction func choosePhotoSource(_ sender: UIBarButtonItem) {
    let alertController = UIAlertController(title: nil,
                                            message: nil,
                                            preferredStyle: .actionSheet)
}
```

Alerts

After determining that the user wants to associate a photo with some item, you create an instance of UIAlertController. No title or message are needed for this action sheet since the purpose should be self-evident from the action the user took. Finally, you specify the .actionSheet style for the alert.

If the alert controller were presented with the current code, there would not be any actions for the user to choose from. You need to add actions to the alert controller, and these actions are instances of UIAlertAction. You can add multiple actions (regardless of the alert's style). They are added to the UIAlertController instance using the addAction(_:) method.

Alerts

```
IBAction func choosePhotoSource(_ sender: UIBarButtonItem) {
    let alertController = UIAlertController(title: nil,
                                            message: nil,
                                            preferredStyle: .actionSheet)
    let cameraAction = UIAlertAction(title: "Camera", style: .default) { _ in
        print("Present camera")
    }
    alertController.addAction(cameraAction)

    let photoLibraryAction
            = UIAlertAction(title: "Photo Library", style: .default) { _ in
        print("Present photo library")
    }
    alertController.addAction(photoLibraryAction)

    let cancelAction = UIAlertAction(title: "Cancel", style: .cancel, handler: nil)
    alertController.addAction(cancelAction)
}
```

Alerts

Each action is given a title, a style, and a closure to execute if that action is selected by the user. The different styles – .default, .cancel, and .destructive – influence the position and styling of the action within the action sheet. For example, .cancel actions show up at the bottom of the list, and .destructive actions use red font colors to emphasize the destructive nature of the action.

Now that the action sheet has been configured, you need a way to present it to the user. To present a view controller modally, you call present(_:animated:completion:) on the initiating view controller, passing in the view controller to present as the first argument.

Alerts

Update choosePhotoSource(_:) to present the alert controller modally:

```
@IBAction func choosePhotoSource(_ sender: UIBarButtonItem) {
    let alertController = UIAlertController(title: nil,
                                            message: nil,
                                            preferredStyle: .actionSheet).....
.............
.............
.............

alertController.addAction(photoLibraryAction)

    let cancelAction = UIAlertAction(title: "Cancel", style: .cancel, handler: nil)
    alertController.addAction(cancelAction)

    present(alertController, animated: true, completion: nil)
}
```

Alerts

The present(_:animated:completion) method takes in a view controller to present, a Bool indicating whether that presentation should be animated, and an optional closure to call once the presentation is completed. Generally, you will want the presentation to be animated, as this provides context to the user about what is happening.

Build and run the application. Tap the camera button and watch the action sheet slide up. Finally, tap one of the actions. If you tap either the Camera or Photo Library action, you will see a message logged to the console indicating which was tapped. Regardless of which action you tap, you will notice that the action sheet is automatically dismissed.

Presentations Styles of View Controllers

.automatic
Presents the view controller using a style chosen by the system. Typically this results in a .formSheet presentation. This is the default presentation style.

.formSheet
Presents the view controller centered on top of the existing content

.fullScreen
Presents the view controller over the entire application.

.overFullScreen
Similar to .fullScreen except the view underneath the presented view controller stays visible. Use this style if the presented view controller has transparency

.popover
Presents the view controller in a popover view on iPad. (On iPhone, using this style falls back to a form sheet presentation style due to space constraints)

Presentations Styles of View Controllers

Action sheets should be presented using the popover style.
on iPad this produces a popover interface with a “pointer” connecting it to the element that triggered it.
On iPhone, because of the smaller window size, .popover falls back to .automatic and allows the system to choose the best style.

This is what you want for your alert controller. On iPad, you want it to appear in a popover pointing at the camera bar button. On iPhone, you want the system to select the best style for the screen size (which will be the .formSheet style you just saw in action).

Presentations Styles of View Controllers

Update choosePhotoSource(_:) to tell the alert controller to use the popover presentation style.

```
@IBAction func choosePhotoSource(_ sender: UIBarButtonItem) {
    let alertController = UIAlertController(title: nil,
                                            message: nil,
                                            preferredStyle: .actionSheet)

    alertController.modalPresentationStyle = .popover
```

Presentations Styles of View Controllers

To indicate where the popover should point, you can specify a frame or a bar button item for it to point to. Since you already have a bar button item, that is the better choice here.

In choosePhotoSource(_:), specify the bar button item that the popover should point at.

```
@IBAction func choosePhotoSource(_ sender: UIBarButtonItem) {
  let alertController = UIAlertController(title: nil,
                                          message: nil,
                                          preferredStyle: .actionSheet)

  alertController.modalPresentationStyle = .popover
  alertController.popoverPresentationController?.barButtonItem = sender
```

Presentations Styles of View Controllers

Every view controller has a popoverPresentationController, which is an instance of UIPopoverPresentationController. The popover presentation controller is responsible for managing the appearance of the popover. One of its properties is barButtonItem, which tells the popover to point at the provided bar button item. Alternatively, you can specify a sourceView and a sourceRect if the popover is not presented from a bar button item.

Build and run the application in simulator, navigate to an item's details, and tap the camera button. The action sheet is presented in a popover pointing at the camera button