# Basic Exercises Part 3.7. UIImagePickerController

## UIImagePickerController

* The UIImagePickerController class is a container view controller that manages navigation between pages of content, where each page is managed by a child view controller.
* UIImagePickerController provides an almost out of the box solution to allow the user to select an image from their device or take a picture with the camera and then present that image. By conforming to the UIImagePickerControllerDelegate, you can create logic that specifies in your app how to present the image and what to do with it (using didFinishPickingMediaWithInfo) and also what to do if the user declines to select an image or take a picture (using imagePickerControllerDidCancel).

### **1.1 Apple documentation**

For information about basic behaviors, visit:

<https://developer.apple.com/documentation/uikit/uiimagepickercontroller>

More resources available:

<https://developer.apple.com/documentation/photokit>

### **1.2 Create a new project**

Create a basic Single View application.

### **1.3 Setup the UIImagePickerController**

Create the controller, set the delegate, and conform to the protocol.

//Swift

class ImageUploadViewController: UIViewController, UIImagePickerControllerDelegate, UINavigationControllerDelegate {

let imagePickerController = UIImagePickerController()

override func viewDidLoad() {

super.viewDidLoad()

imagePickerController.delegate = self

} }

//Objective-C

@interface ImageUploadViewController : UIViewController <UIImagePickerControllerDelegate,UINavigationControllerDelegate> {

UIImagePickerController \*imagePickerController;

}

@end  
@implementation ImageUploadViewController

- (void)viewDidLoad {

[super viewDidLoad];

imagePickerController.delegate = self; }

@end

note: We actually will not implement anything defined in UINavigationControllerDelegate, but UIImagePickerController inherits from UINavigationController and changes the behavior of UINavigationController. Therefore, we still need to say our view controller conforms to UINavigationControllerDelegate.

### **1.4 Mandatory**

The role and appearance of an image picker controller depend on the source type you assign to it before you present it.

Whenever you need to show UIImagePickerController, you need to implement part of the next code:

//Swift  
self.imagePickerController.sourceType = .Camera // options: .Camera , .PhotoLibrary , .SavedPhotosAlbum  
self.presentViewController(self.imagePickerController, animated: true, completion: nil)

//Objective-C

imagePickerController.sourceType = UIImagePickerControllerSourceTypeCamera; // options: UIImagePickerControllerSourceTypeCamera, UIImagePickerControllerSourceTypePhotoLibrary, UIImagePickerControllerSourceTypeSavedPhotosAlbum  
[self presentViewController:imagePickerController animated:YES completion:nil];

### **1.5 Implement the delegate methods.**

To use an image picker controller, you must provide a delegate that conforms to the UIImagePickerControllerDelegate protocol. Starting in iOS 4.1, you can use the delegate to save still-image metadata to the Camera Roll along with the image. See UIImagePickerControllerDelegate.

//Swift

func imagePickerController(picker: UIImagePickerController, didFinishPickingMediaWithInfo info: [String : AnyObject]) {

if let pickedImage = info[UIImagePickerControllerOriginalImage] as? UIImage { // Your have pickedImage now, do your logic here

}

self.dismissViewControllerAnimated(true, completion: nil) }

func imagePickerControllerDidCancel(picker: UIImagePickerController) { self.dismissViewControllerAnimated(true, completion: nil)

}

//Objective-C

- (void)imagePickerController:(UIImagePickerController \*)picker didFinishPickingMediaWithInfo:(NSDictionary \*)info {

UIImage \*pickedImage = info[UIImagePickerControllerOriginalImage]; if (pickedImage) {

//You have pickedImage now, do your logic here

}

[self dismissViewControllerAnimated:YES completion:nil];

}

- (void)imagePickerControllerDidCancel:(UIImagePickerController \*)picker { [self dismissViewControllerAnimated:YES completion:nil];

}

### **1.6 Asking for permission**

The app must ask for permission from the user before accessing the camera.

A screenshot of a cell phone

Description automatically generated

When it does that it will display a message to the user explaining why it needs to use the camera.

### **1.6 Info.plist setting**

You can set this message by setting the

You can set this message by setting the NSPhotoLibraryUsageDescription key in the Info.plist list of your app.

If you don’t add the NSPhotoLibraryUsageDescription key you will get an error that looks like this if you try to run that code:

Error: [access] This app has crashed because it attempted to access privacy-sensitive data without a usage description. The app's Info.plist must contain an NSCameraUsageDescription key with a string value explaining to the user how the app uses this data.

Open Info.plist and click the + on the root node of the plist:

A screenshot of a cell phone

Description automatically generated

### **1.7 This is not working!! Funny face with no fill**

Not on the iOS simulator because it does not have a camera. Try in a real device. Also, it’s a good idea to check if the camera is available, you can use the UIImagePickerController.isSourceTypeAvailable(\_:) class method for this

class ViewController: UIViewController {

...

func openCamera() {

guard UIImagePickerController.isSourceTypeAvailable(.camera) else {

print("camera not supported by this device")

return

}

imagePicker.sourceType = .camera

imagePicker.delegate = self

present(imagePicker, animated: true)

}

}

### **1.8 Getting the image form the info dictionary**

UIImagePickerController give a dictionary with information about the selected media. You can find the UIImageinstance representing the selected image on the UIImagePickerControllerOriginalImage key:

func imagePickerController(\_ picker: UIImagePickerController, didFinishPickingMediaWithInfo info: [String : Any]) {

...

// get the image

guard let image = info[UIImagePickerControllerOriginalImage] as? UIImage else {

return

}

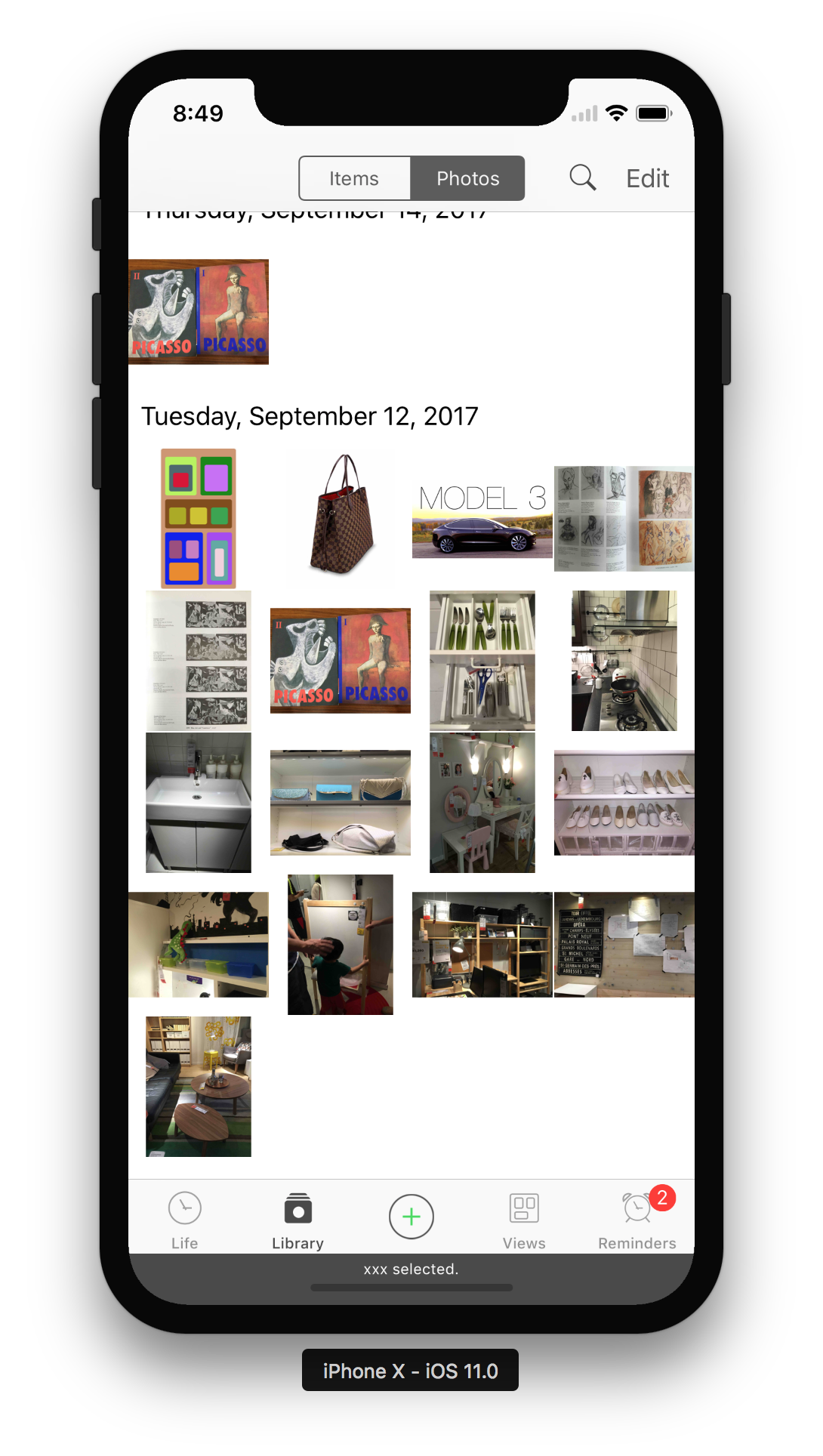
// do something with it

imageView.image = image

}

### **1.9 Photo Library**

Open the Library



To let the user pick an image he previously took, you have to change the sourceType of the image picker to .photoLibrary:

func openPhotoLibrary() {

guard UIImagePickerController.isSourceTypeAvailable(.photoLibrary) else {

print("can't open photo library")

return

}

imagePicker.sourceType = .photoLibrary

imagePicker.delegate = self

present(imagePicker, animated: true)

}

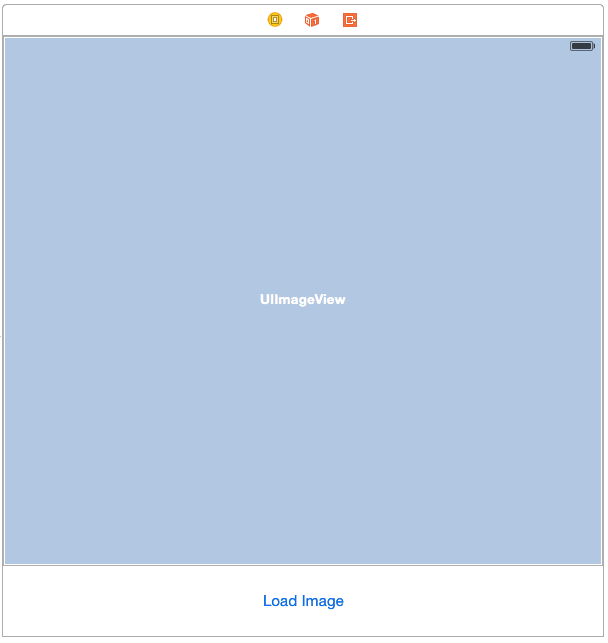
You have to set the NSPhotoLibraryUsageDescription key in Info.plist, same as for the camera. If you don’t you’ll get this error:

Error: [access] This app has crashed because it attempted to access privacy-sensitive data without a usage description. The app's Info.plist must contain an NSCameraUsageDescription key with a string value explaining to the user how the app uses this data.

Since we are using UIImagePickerController for this task as well, we have to photo selection or canceling with the same delegate methods.

### **1.9 Homework!**

1. Make an app that gets an image from UIImagePickerController and show it on the screen (see example below).



2. Make an app that load two images one on top each other and make the one on top 50% transparent in order to combine them (free layout)

### **1.9 SetUp same project**

Create the same project using Objective C.