Curtin University

System Controllers in Apps & User Input

In this lesson, we'll learn to...

- Display alerts, share content and send messages from within Apps;
- Access the camera and photo library on a device;
- Build custom forms for creating new object models;
- Get complex user input through forms, data collection and dynamic table views.

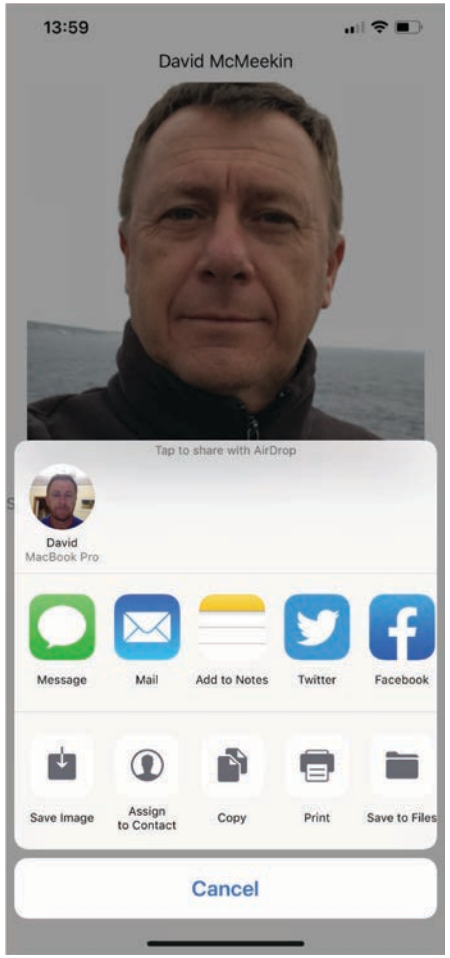
Curtin University

System View Controllers

```
var myVariable = 42
myVariable = 50
let myConstant = 42
```

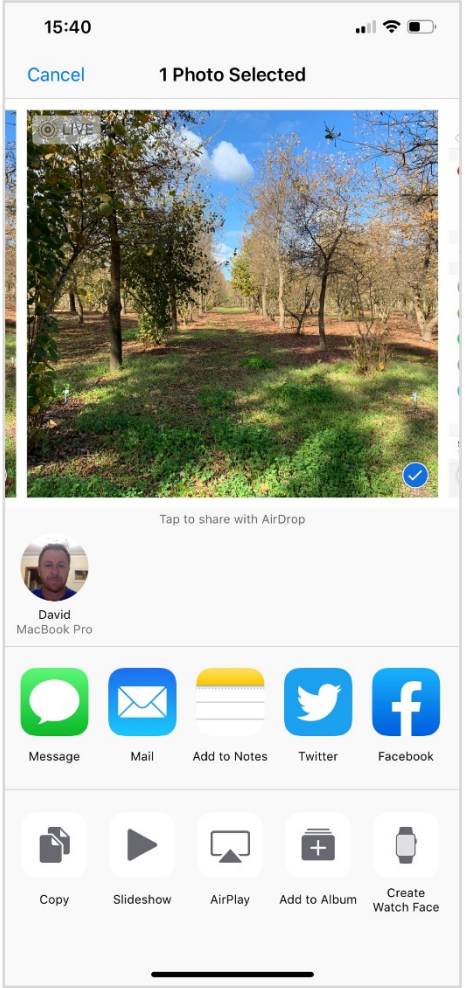
System View Controllers

- Provide a way to:
  - Display alerts;
  - Share content;
  - Send messages;
  - Access photo library, camera etc.



Different Controllers

- UIActivityController: share with other apps;
- SFSafariViewController: display things from the web;
- UIAlertController: present information & options;
- UIImagePickerViewController: camera or photolibrary;
- MFMailComposeViewController: send email.



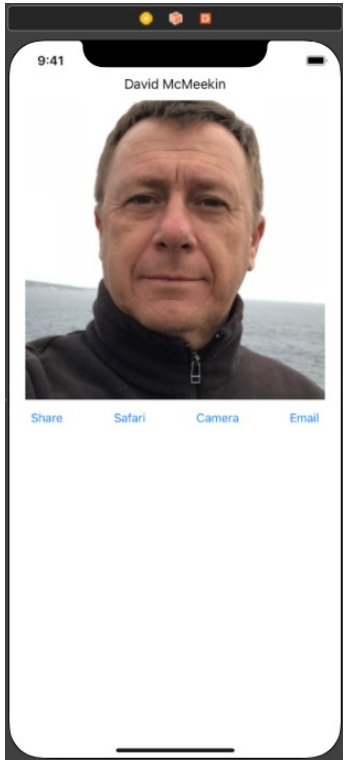
Curtin University

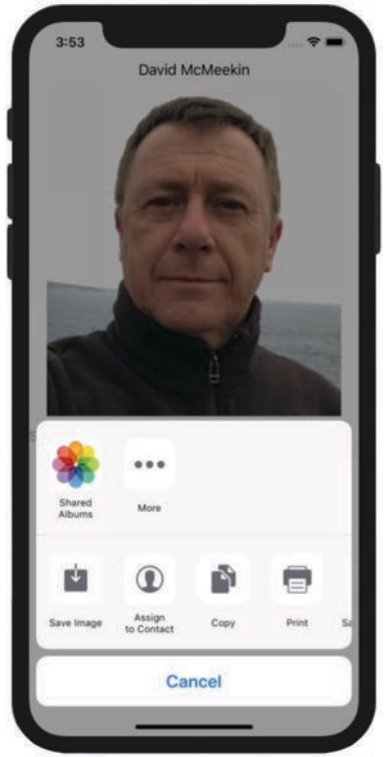
UIActivityViewController

```
var myVariable = 42
myVariable = 50
let myConstant = 42
```

UIActivityViewController

- Tapping Share button creates a UIActivityViewController instance;
- Grab the image from the UIImageView;
- UIActivityViewController's initializer takes a parameter activityItems;
- activityItems is an array of type Any;
- Add the image to the array;
- Present the activity controller.

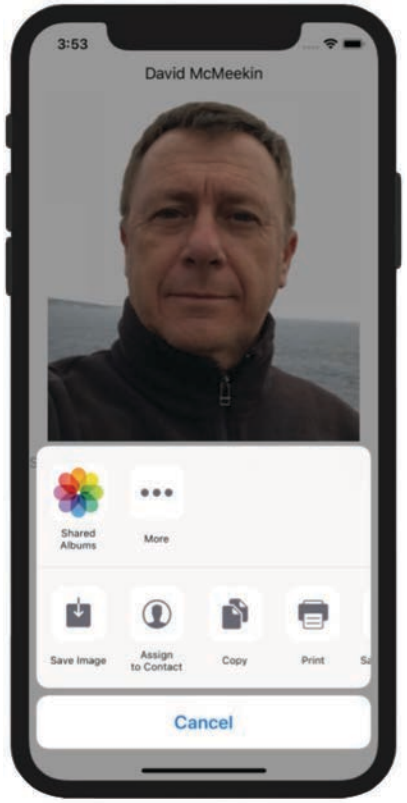




The Code

```
@IBAction func shareButtonTapped(_ sender: UIButton) {
    guard let image = imageView.image else {return}
    let activityController = UIActivityViewController(activityItems: [image],
                                                      applicationActivities: nil)
    activityController.popoverPresentationController?.sourceView = sender
    present(activityController, animated: true, completion: nil)
}
```

- An image variable is created from UIImageView;
- image is added to the activity controller's array parameter;
- We won't use applicationActivities;
- popoverPresentationController from where the view controller is presented to the user.



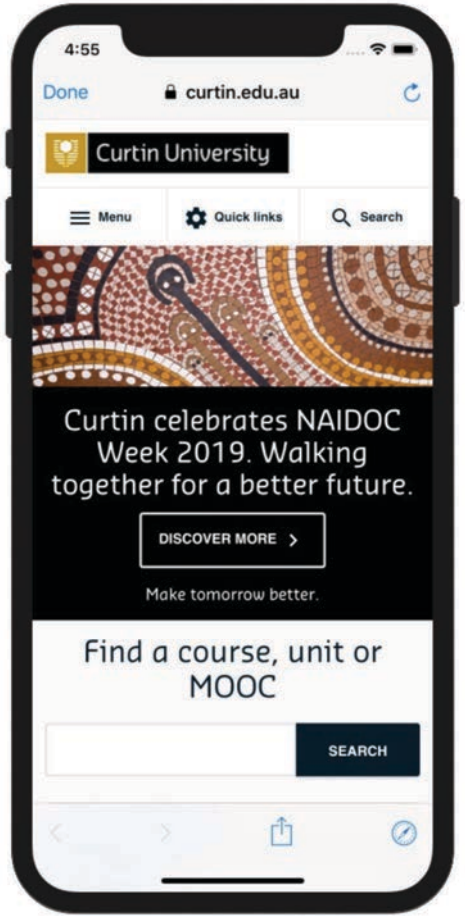
Curtin University

SFSafariViewController

```
var myVariable = 42
myVariable = 50
let myConstant = 42
```

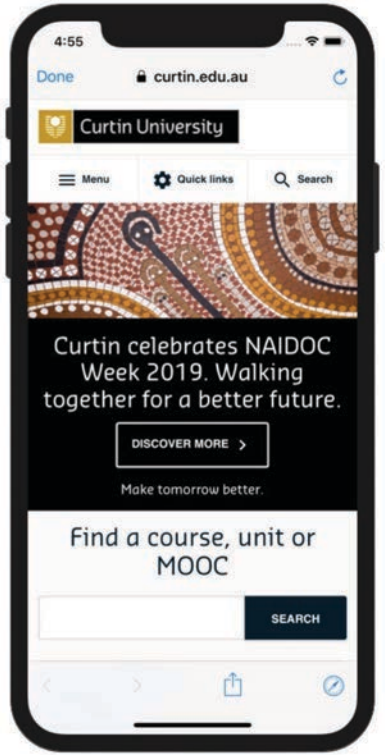
SFSafariViewController

- Opens a Safari web browser within our App;
- SafariServices framework must be imported;
- Three things are required:
  1. Create a URL from a string, a web address;
  2. Create an instance of SFSafariViewController with the URL;
  3. Present the view to the user.



The Code

```
@IBAction func safariButtonTapped(_ sender: UIButton) {
    if let url = URL(string: "https://www.curtin.edu.au") {
        let safariViewController = SFSafariViewController(url: url)
        present(safariViewController, animated: true, completion: nil)
    }
}
```



- A URL is created;
- An instance of SFSafariViewController is created;
- The view controller is presented to the user.

Curtin University

UIAlertController

```
var myVariable = 42
myVariable = 50
let myConstant = 42
```

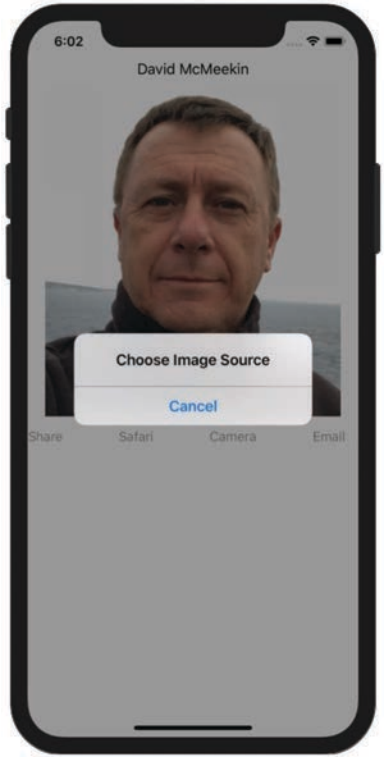
UIAlertController

- Used to allow users to interact with your App;
- Get user's attention, present options, get choice;
- Things that need to be done:
  - Specify the Alert title;
  - Create the message;
  - Decide how to present it to the user:
    - .alert at centre of screen, .actionSheet at bottom of screen.

The Code

```
@IBAction func photoButtonTapped(_ sender: UIButton) {
    let alertController = UIAlertController(title: "Choose Image Source",
                                            message: nil, preferredStyle: .alert)

    let cancelAction = UIAlertAction(title: "Cancel", style: .cancel, handler: nil)
    alertController.addAction(cancelAction)
    alertController.popoverPresentationController?.sourceView = sender
    present(alertController, animated: true, completion: nil)
}
```



- A UIAlertController is created with a title & style is set to .alert;
- The UIAlertAction is created to respond to the user's choice, with title, style & handler;
- It is added and the UIAlertController is presented.

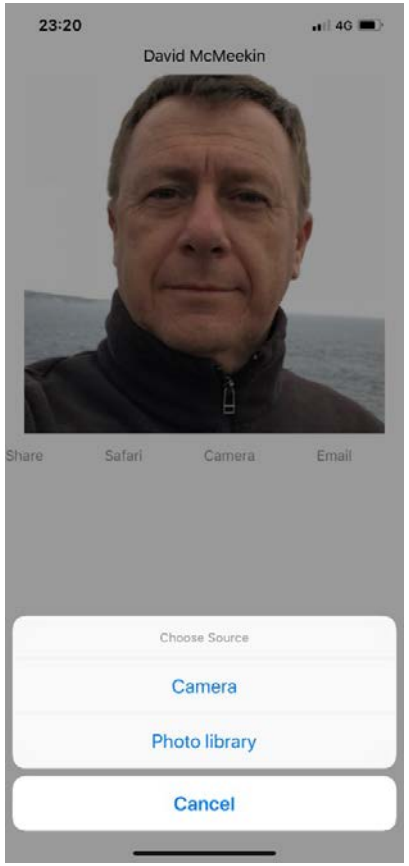
Curtin University

UIImagePickerController

```
var myVariable = 42
myVariable = 50
let myConstant = 42
```

UIImagePickerController

- UIImagePickerController provides access to the camera & the photo library;
- Two protocols must be adopted:
  1. UIImagePickerControllerDelegate;
  2. UINavigationControllerDelegate.



- Create an instance of UIImagePickerController;
- Set the view controller as the delegate.

UIImagePickerController

- The user is presented with choices;
- The UIAlertController will need to handle responses;
- The camera and/or photo library should only be presented if they are available;
- `UIImagePickerController.isSourceTypeAvailable(_:)` is used for this, returning a Bool;

The Code (1)

```
@IBAction func photoButtonTapped(_ sender: UIButton) {
    let imagePicker = UIImagePickerController()
    imagePicker.delegate = self

    let alertController = UIAlertController(title: "Choose Source", message: nil,
                                            preferredStyle: .actionSheet)

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

- Create the UIImagePickerController instance;
- Set the delegate to self;
- Create & configure the UIAlertControllers as we did earlier.

The Code (2)

```
if UIImagePickerController.isSourceTypeAvailable(.camera){
    let cameraAction = UIAlertAction(title: "Camera", style: .default, handler: {action in
        imagePicker.sourceType = .camera
        self.present(imagePicker, animated: true, completion: nil)})
    alertController.addAction(cameraAction)
}

if UIImagePickerController.isSourceTypeAvailable(.photoLibrary){
    let photoLibraryAction = UIAlertAction(title: "Photo library", style: .default, handler: {action in
        imagePicker.sourceType = .photoLibrary
        self.present(imagePicker, animated: true, completion: nil)})
    alertController.addAction(photoLibraryAction)
}
present(alertController, animated: true, completion: nil)
```

Changing the image

imagePickerController(_: didFinishPickingMediaWithInfo:) method is used;

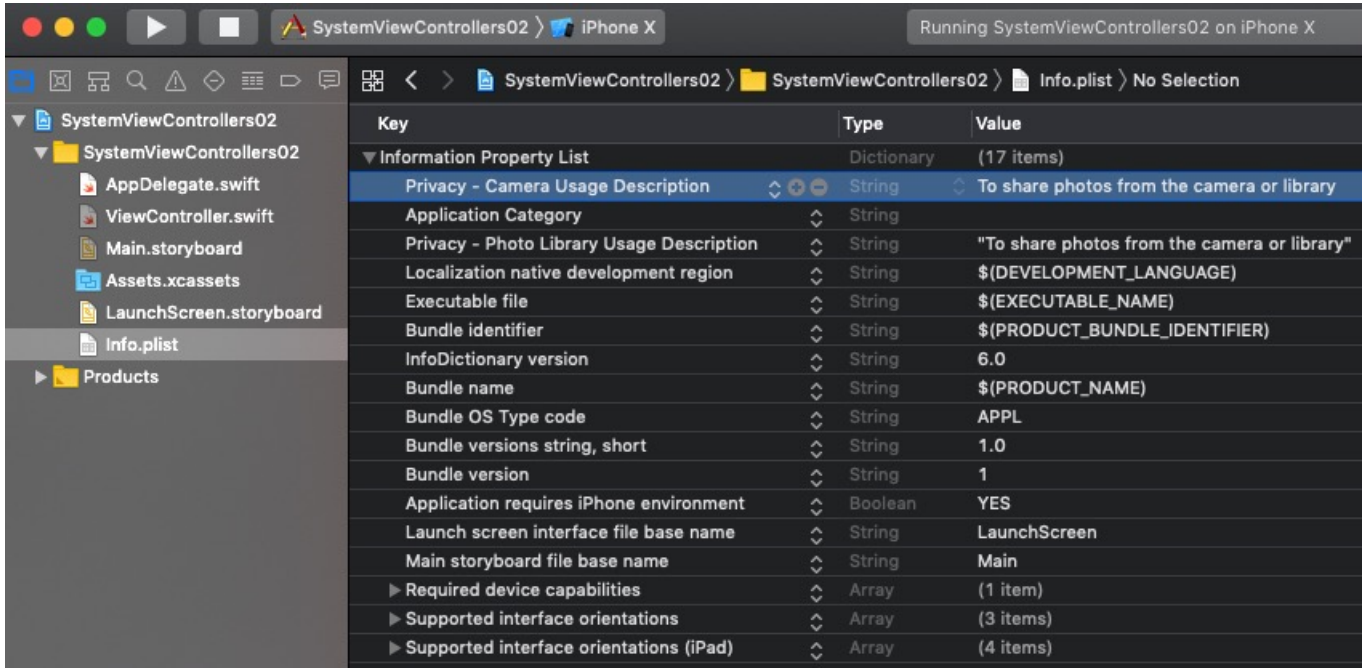
```
func imagePickerController(_ picker: UIImagePickerController, didFinishPickingMediaWithInfo info:
    [UIImagePickerController.InfoKey : Any]) {

    guard let selectedImage = info[.originalImage] as? UIImage
        else {return}

    imageView.image = selectedImage
    dismiss(animated: true, completion: nil)
}
```

Info.plist

- The App won't behave as we think, not quite yet;
- The project Info.plist needs to be adjusted giving access to the photo library and camera.



Curtin University

MFMailComposeViewController

```
var myVariable = 42
myVariable = 50
let myConstant = 42
```

MFMailComposeViewController

- Allows email to be sent from within your App;
- The MessageUI framework must be imported;
- The mailComposeDelegate is responsible to dismiss the mail compose view controller;
- The ViewController class must adopt the protocol: MFMailComposeViewControllerDelegate;

MFMailComposeViewController

- Check if mail services are available: `.canSendMail()`
- Create an instance of `MFMailComposeViewController`;
- Set the `.mailComposeDelegate` to self;
- Configure different aspects of the mail message;
- Present and then dismiss the view controller.

The Code

```
@IBAction func emailButtonTapped(_ sender: UIButton) {
    guard !MFMailComposeViewController.canSendMail() else {
        print("Can not send mail")
        return
    }
    let mailComposer = MFMailComposeViewController()
    mailComposer.mailComposeDelegate = self
    mailComposer.setToRecipients(["Taylor@swift.com"])
    mailComposer.setSubject("Testing for you Taylor Swift")
    mailComposer.setMessageBody("Hello, Taylor!", isHTML: false)
    present(mailComposer, animated: true, completion: nil)
}
```

- Checked if mail can be sent;
- Created the MFMailComposeViewController instance;
- Set mailComposeDelegate to self;
- Presented it to the user.

mailComposeController

- Use a delegate method to dismiss the the view controller;

```
func mailComposeController(_ controller: MFMailComposeViewController,
                           didFinishWith result: MFMailComposeResult, error: Error?) {

    dismiss(animated: true, completion: nil)
}
```

Curtin University

Input Screens

```
var myVariable = 42
myVariable = 50
let myConstant = 42
```

User Input

- Not all Apps are for entertainment, some are for productivity;
- Apps are used for business, research, data collection;
- These Apps can require complex user input;
- Multiple control types and views need to be used;
- More common than you might think.

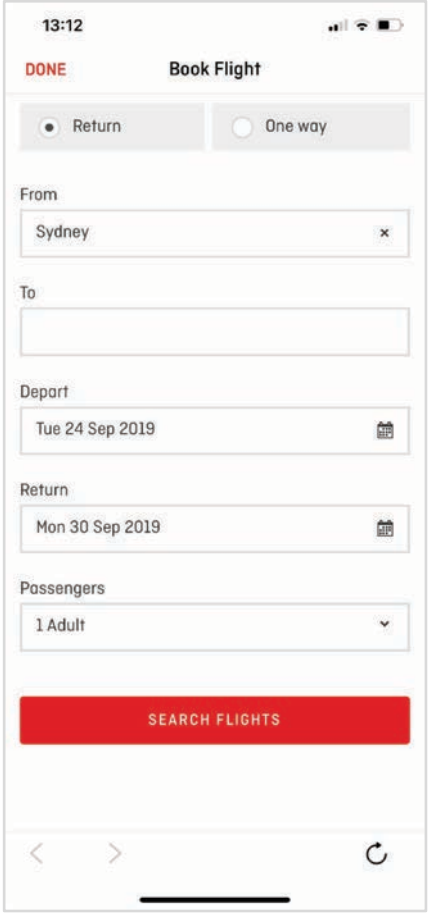


Image source: QANTAS app available from the App Store

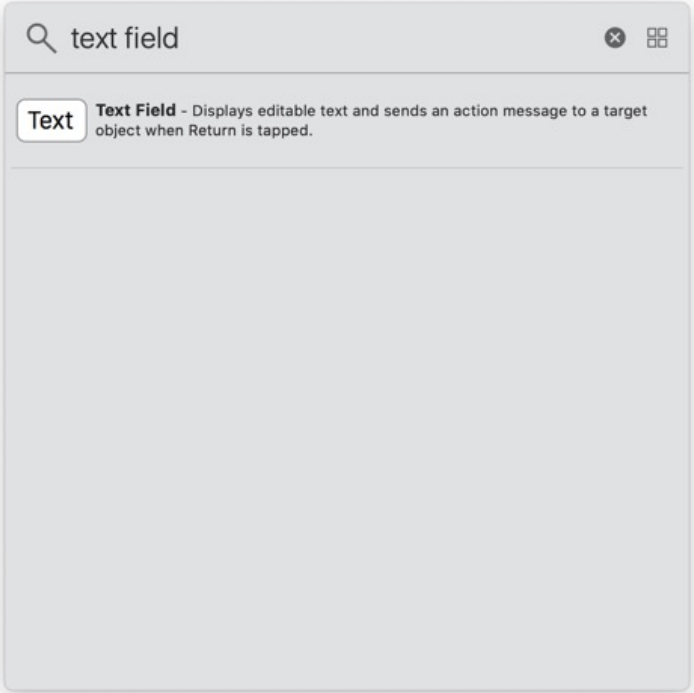
Data Model

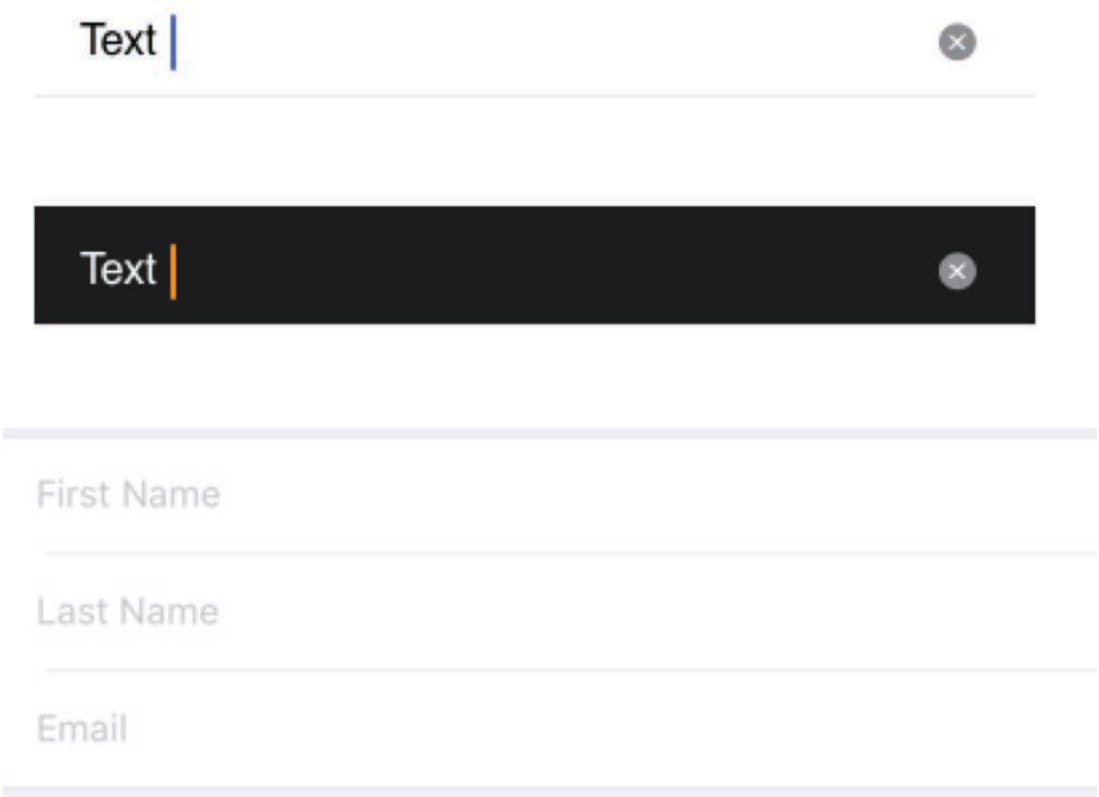
- What data are you collecting from the user?
- What type of data is it?
- What will you do with that data?
- How will you use the data?
- Plan the data model accordingly.

Collecting the Data

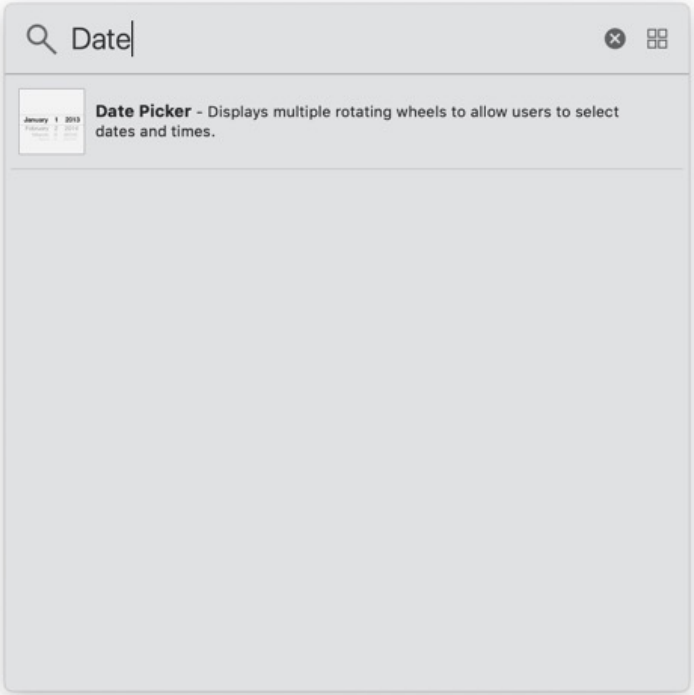
- What views are best for your data collection?
- Prevent user input errors;
- Dates: what kind of data format?
- Numbers: integers, doubles?
- Addresses, City names, Airport codes etc.

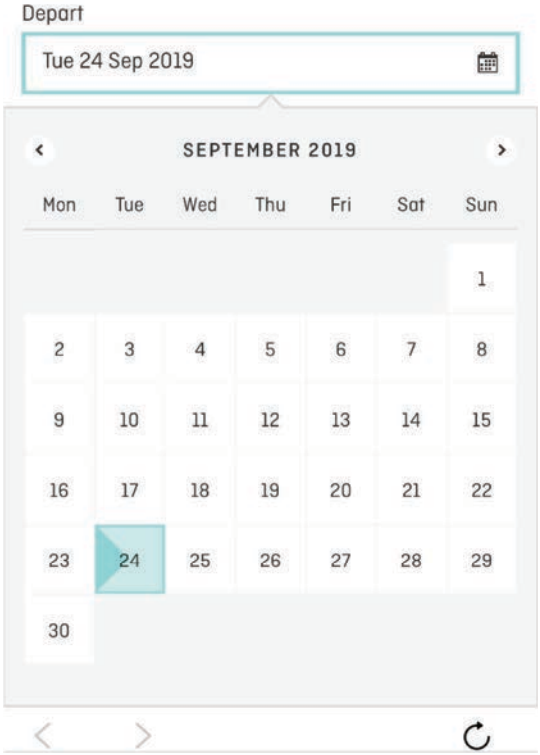
Collecting String Input

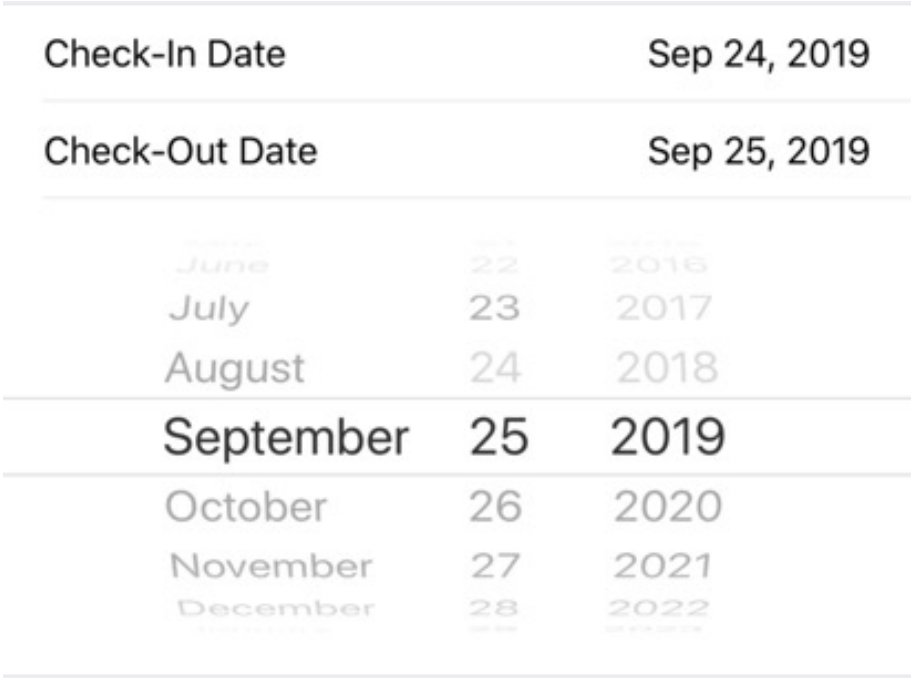




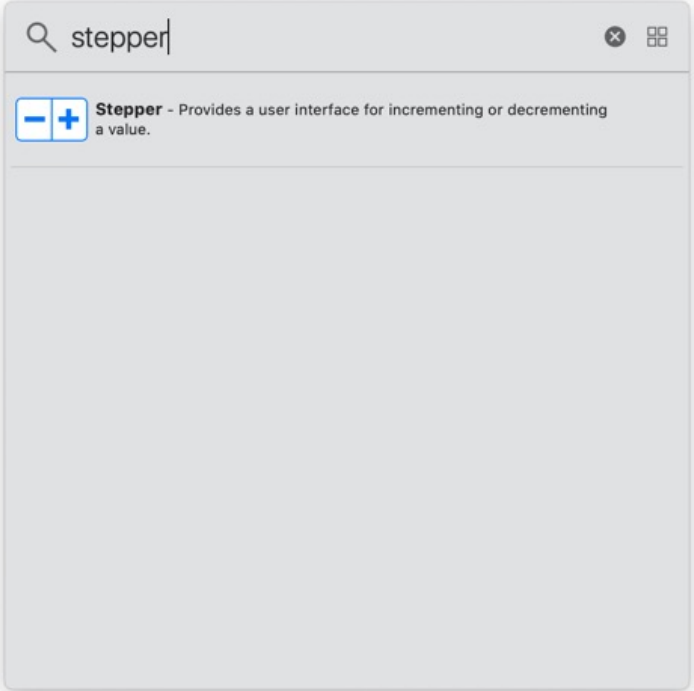
Collecting Date Input

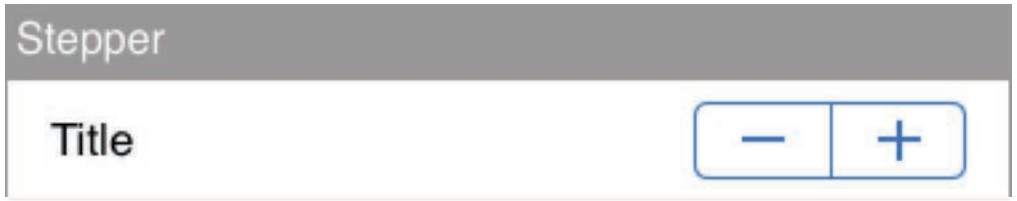




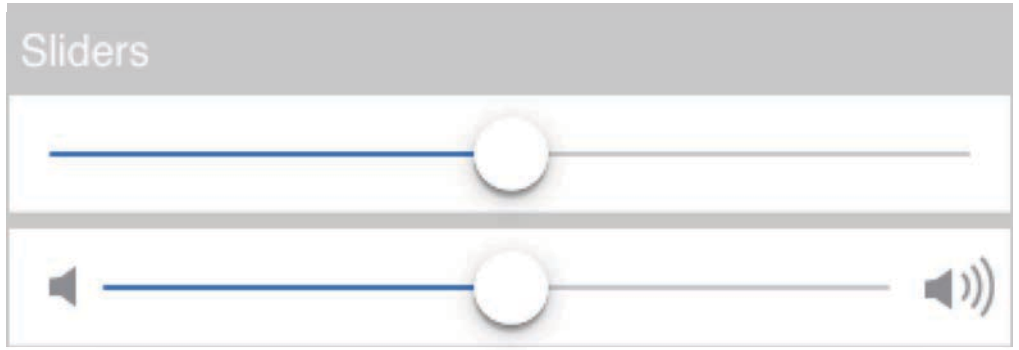


Collecting Integer Input

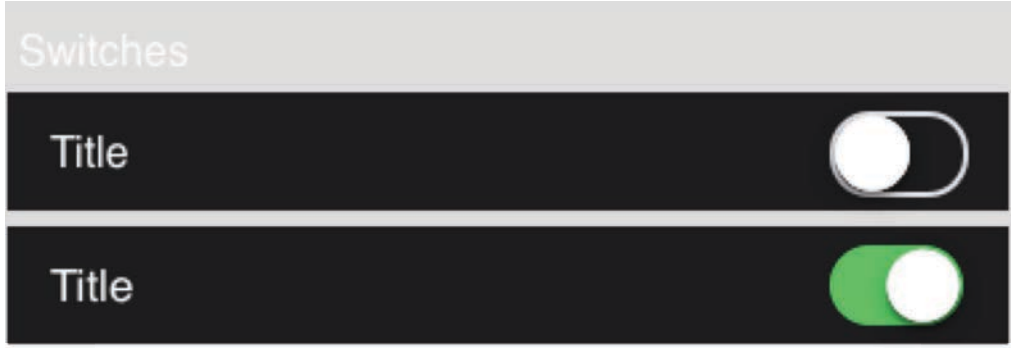


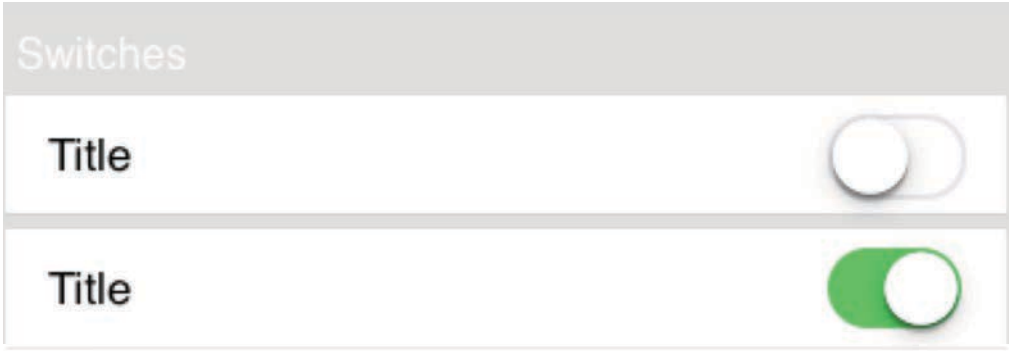


Sliding Scale Input



Collecting Binary Input





Curtin University

Wrapping Up

```
var myVariable = 42
myVariable = 50
let myConstant = 42
```

What we learned in this lesson:

- Displaying alerts, content sharing and message sending from within Apps;
- Camera and photo library access on the device;
- To build custom forms and create new object models;
- Ways to get complex user input and to collect data from the user.