Mobile Programming (Part 2)

Copyright © 2022 by
Robert M. Dondero, Ph.D.
Princeton University

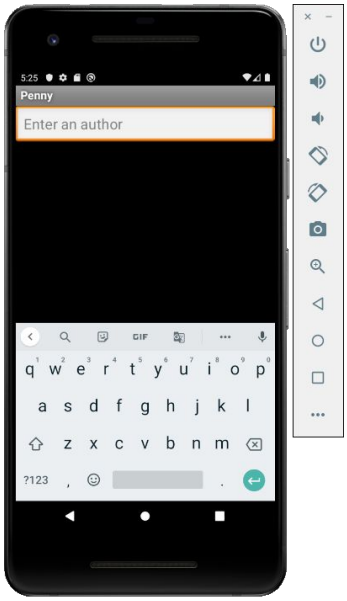
Objectives

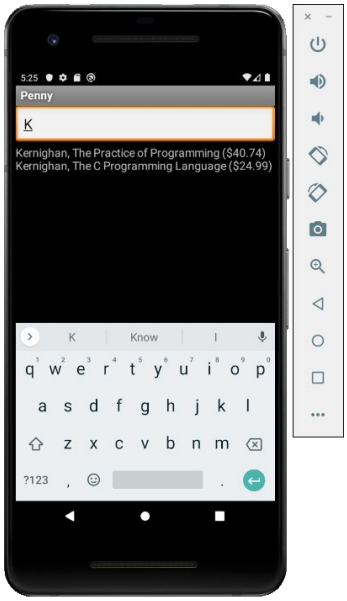
- We will cover:
  - Mobile programming
  - Android mobile programming

PennyAndroid App

- PennyAndroid app
  - The job:
    - One screen
    - User enters author name prefix
    - App displays books by authors whose names have that prefix
    - App responds to each keystroke

PennyAndroid App





Agenda

- **PennyAndroidServer**
- PennyAndroid

PennyAndroidServer: Defining

- PennyAndroid must communicate with PennyAndroidServer
- Where should PennyAndroidServer run?
  - courselab? **No**
  - Local computer? **Maybe**
  - Cloud? **Yes**!

6

PennyAndroidServer App

Where should PennyAndroid server run?
- courselab?
  - **No**; firewall
- Local computer?
  - PennyAndroid client & PennyAndroid server use same router?
    - **Yes**
    - But router could change server computer's IP address
  - PennyAndroid client & PennyAndroid server do not use same router?
    - **Yes**, but difficult
    - Must pay ISP for static IP address
    - Must configure local router to allow public access
- Cloud? **Yes**!

PennyAndroidServer: Defining

- Which **cloud service** for PennyAndroidServer?
  - I chose ***Heroku***

PennyAndroidServer App

Which **cloud service** for PennyAndroidServer?
I chose ***Heroku***

PennyAndroidServer: Defining

- What comm protocol?
  - **Observations**
    - Any would work
  - **Decisions**
    - Use HTTP
    - Design PennyAndroidServer as Python/Flask web app

8

PennyAndroid App: Server

What comm protocol?

Observations

PennyAndroid apps could communicate with PennyAndroidServer using any protocol

For Android apps, common to use HTTP

Using HTTP would allow PennyAndroidServer implementation via Python/Flask

Decisions

Use HTTP

Design PennyAndroidServer as Python/Flask web app

PennyAndroidServer: Defining

- What comm protocol? (cont.)
  - **Observations**
    - Book list could be expressed as XML, JSON, plain text, (HTML), …
  - **Decision**
    - Express book list as plain text

PennyAndroid App: Server

What comm protocol? (cont.)

Observations

PennyAndroid server must read author name, write book list

Book list could be expressed as XML, JSON, plain text, (HTML), …

Decisions

Keep it simple

Express book list as plain text

PennyAndroidServer: Defining

- See **PennyAndroidServer** app
  - penny.sql, penny.sqlite
  - book.py, database.py
  - **penny.py**
  - **Procfile**
  - **runtime.txt**
  - **requirements.txt**
    - `python -m pip freeze > requirements.txt`

10

PennyAndroidServer: Defining

[see slide]

Code notes: penny.py

Same as previous versions, except…
The only valid requests are …/ and …/searchresults
Delivers plain text, not HTML

Code notes: Procfile

Tells Heroku the app's process type; it's a web app
Tells Heroku the command to start the process
Heroku disallows test HTTP/WSGI server bundled with Python, and HTTP/WSGI server bundled with Flask
Heroku allows *gunicorn*

Code notes: runtime.txt

Tells Heroku which version of Python it should use

Code notes: requirements.txt

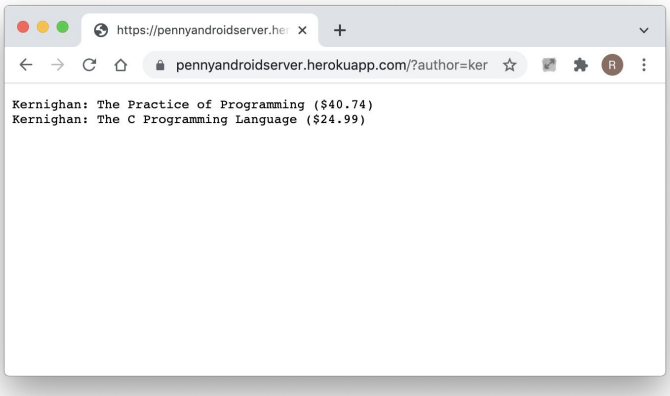
Tells Heroku what additional modules the app uses
Create manually, or…
Issue command `python -m pip freeze > requirements.txt`

PennyAndroidServer: Deploying

- Deploying the app
  - Deploy to Heroku, in the usual way
    - Local git → Github → Heroku
  - See Appendix for shortcut
    - Local git → Heroku

PennyAndroidServer: Running

- Run the app
  - Browse to:
    https://pennyandroidserver.herokuapp.com/?author=ker



Agenda

- PennyAndroidServer
- **PennyAndroid**

PennyAndroid: Defining

- **Android design constraint 1**
  - Main/GUI thread is not allowed to do networking
- **Implications**
  - Main/GUI thread must spawn a child/worker thread
  - Child/worker thread must comm with PennyAndroidServer

PennyAndroid: Defining

- **Android design constraint 2**
  - Main/GUI thread must remain responsive
  - Main/GUI thread laggy => typing/tapping fast generates “App is unresponsive” messages
- **Implications**
  - Main/GUI thread cannot wait for child/worker thread to finish
  - Main/GUI thread and child/worker thread must run concurrently

PennyAndroid: Defining

- **Android design constraint 3**
  - Child/worker thread is not allowed to update GUI
- **Implications**
  - Child/worker thread must send changes to main/GUI thread
  - Option 1: ordinary Java threads, shared queue, queue polling
  - Option 2: **AsyncTask**

PennyAndroid: Defining

- Class ***AsyncTask***
  - Android-specific Java class
  - "AsyncTask enables proper and easy use of the UI thread. This class allows you to perform background operations and publish results on the UI thread without having to manipulate threads and/or handlers."
  - Internally uses an ordinary Java thread to do network comm
  - "On the way out" updates the GUI

https://developer.android.com/

PennyAndroid: Defining

```
PennyAndroid/
   AndroidManifest.xml
   res/
      layout/
         activity_main.xml
      values/
         values.xml
   src/
      edu/
         princeton/
            penny/
               MainActivity.java
```

18

PennyAndroid: Defining

Code notes: values.xml

Defines string resources

Code notes: activity_main.xml

The MainActivity page consists of a vertical LinearLayout containing a EditText object and a TextView object

EditText object

"A user interface element for entering and modifying text."

TextView object (as in HelloAndroid app)

"A user interface element that displays text to the user."

Code notes: AndroidManifest.xml

The app's name is Penny

The app consists of one Activity

The Activity is named edu.princeton.penny.MainActivity

The app will use networking

Code notes: MainActivity.java

See following slides

PennyAndroid: Defining

- **MainActivity.java**
  - Static overview:

```
class MainActivity extends Activity
   class AuthorSearchTask extends AsyncTask
      doInBackground()
      onPostExecute()
      onCancelled()
   class MyTextWatcher implements TextWatcher
      afterTextChanged()
   onCreate()
```


Try to see the commonalities between this Penny Android app and the Penny client-server app

PennyAndroid: Defining

- **MainActivity.java**
  - Dynamic overview:
    - `onCreate()` instantiates `MyTextWatcher` object, and installs it as the listener for the `EditText` object
      - When user enters text into `EditText` object, Android calls `afterTextChanged()` method in `MyTextWatcher` object

PennyAndroid: Defining

- **MainActivity.java**
  - Dynamic overview:
    - `afterTextChanged()` method instantiates `AuthorSearchTask` object and sends `execute()` message; creates child/worker thread; main/GUI thread proceeds
    - Child/worker thread `doInBackground()` method sends author to PennyAndroidServer, receives book list from PennyAndroidServer
    - Child/worker thread ends; executes `onPostExecute()` on the way out
    - `onPostExecute()` updates GUI

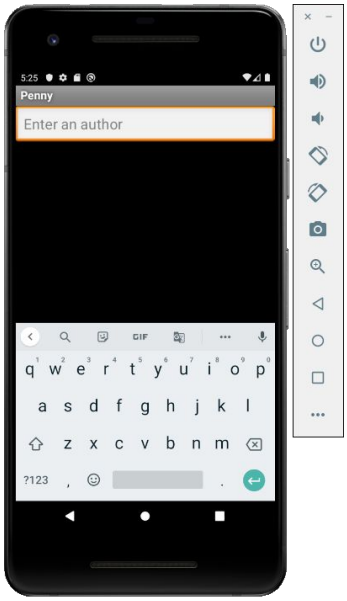
PennyAndroid: Building

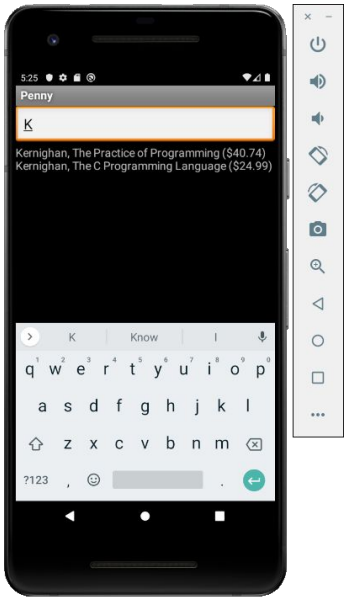
- See **AndroidBuild**

PennyAndroid: Installing

- See **AndroidInstall**

PennyAndroid App: Result





Summary

- We have covered:
  - Mobile programming
  - Android mobile programming
- See also:
  - **Appendix 1**: Shortcut Heroku Deployment
  - **Appendix 2**: iOS Development
    - Written by **Christine Sun**

Appendix 1:
Shortcut Heroku Deployment

Shortcut Heroku Deployment

- How to deploy to Heroku?
  - Production quality approach
    - Local computer → GitHub → Heroku
    - Described in the optional *Server-Side Web Programming: Python (Part 3)* lecture
  - Shortcut approach
    - Local computer → Heroku
    - Described here…

Shortcut Heroku Deployment

- Get a Heroku account
  - Browse to https://signup.heroku.com/dc
  - Choose Python as primary dev lang
  - Enter your email addr
  - Choose a password

Shortcut Heroku Deployment

- Install the Heroku Command Line Interface (CLI)
  - Browse to https://devcenter.heroku.com/articles/heroku-cli
    - Follow the instructions

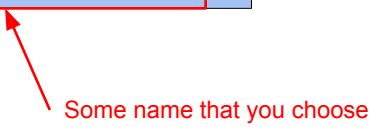
Shortcut Heroku Deployment

- Log into heroku:

```
$ heroku login
```

- Create a Git repo on Heroku:

```
$ heroku create pennyandroidserver
```



Shortcut Heroku Deployment

- Create a local Git repo

```
$ cd PennyAndroidServer  # if necessary

$ git init
$ git add .
$ git commit -m "initial load"
```

Shortcut Heroku Deployment

- Specify your Heroku Git repo as the remote of your local Git repo:

```
$ cd PennyAndroidServer  # if necessary
$ heroku login           # if necessary

$ heroku git:remote -a pennyandroid
```

- (Optional) Confirm that you set the remote repo:

```
$ git remote -v
```

Shortcut Heroku Deployment

- Deploy the app to Heroku:

```
$ cd PennyAndroidServer   # if necessary
$ heroku login            # if necessary

$ git push heroku master
```

You may need to enter `main` instead of `master`

  - Git uploads the app to Heroku repo
  - Heroku builds and deploys the app

Appendix 2:
iOS Development

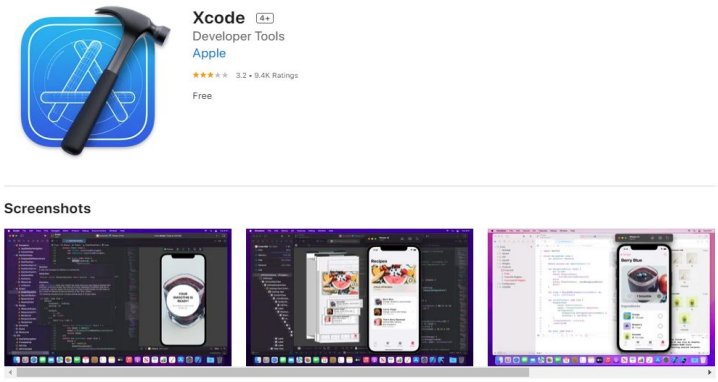
Written by
Christine Sun ‘24

Goal

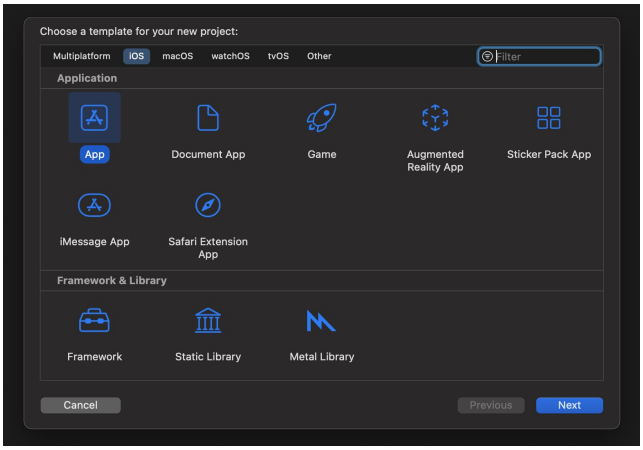
- Compose a **PennySwift** application
  - **Server**: PennyAndroidServer
    - As described previously in this lecture
  - **Client**: an iOS application
    - Written using the Swift programming language

Setting up XCode

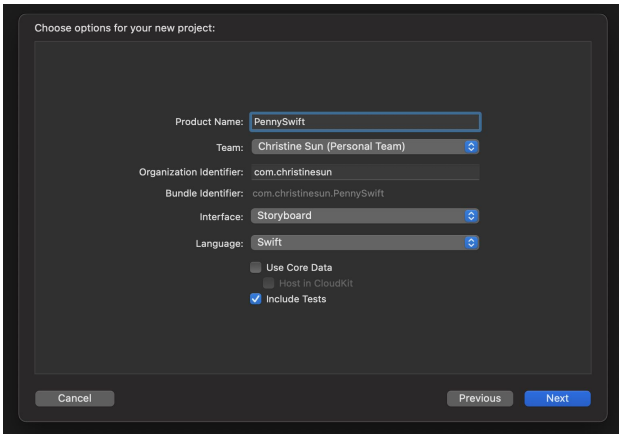
XCode is the IDE used to develop software for iOS apps. You can find and download it in the Mac App Store.



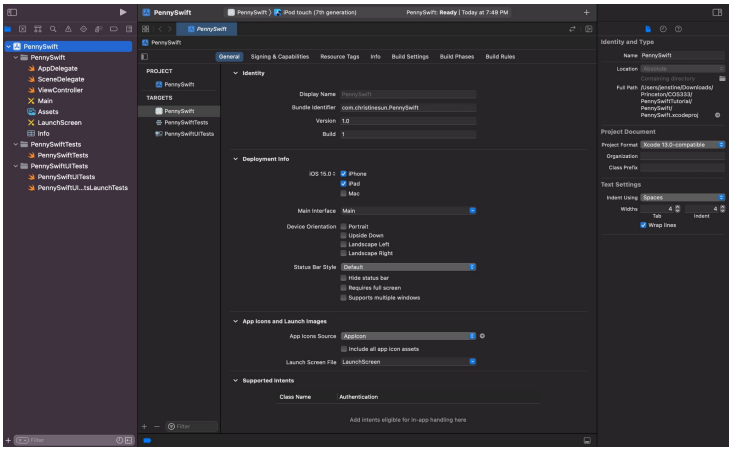
- Open XCode
- Click “Create a new XCode project”



- Select iOS App
- Click Next

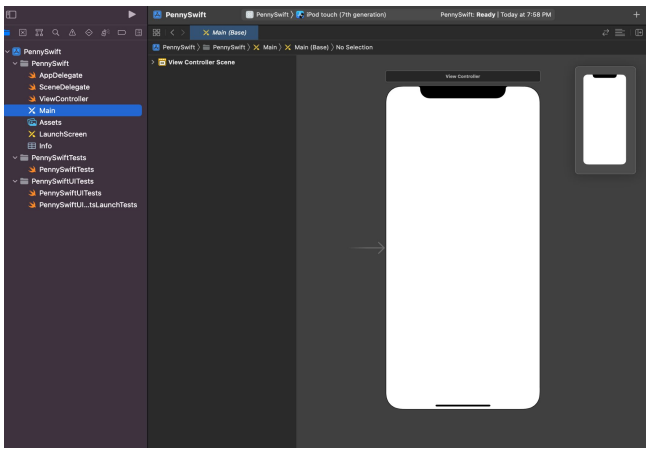


- In the Product Name, type "**PennySwift**"
- Select the team you're creating this app for
  - typically "**<Your Name> (Personal Team)**"
- For Organization Identifier put "**com.<yourname>**"
- For Interface, select **Storyboard**
- For Language, select **Swift**
- Click Next



Keep clicking next, choosing a location to save your project, until XCode opens up and looks like this

Building the user interface

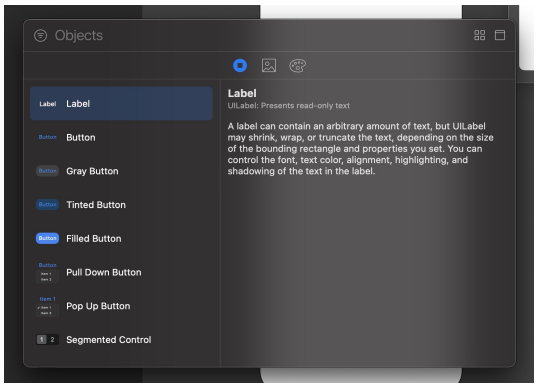


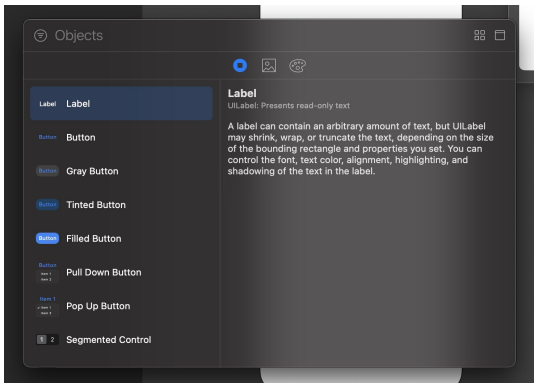
We build the user interface in a storyboard file called "**Main**"

Here you'll see "screens" (view controllers) that look like iPhones

There is a XML representation that you can view by right clicking Main on the left hierarchy and selecting "Open As > Source Code"

Click on the + on the top right to open up the toolbox

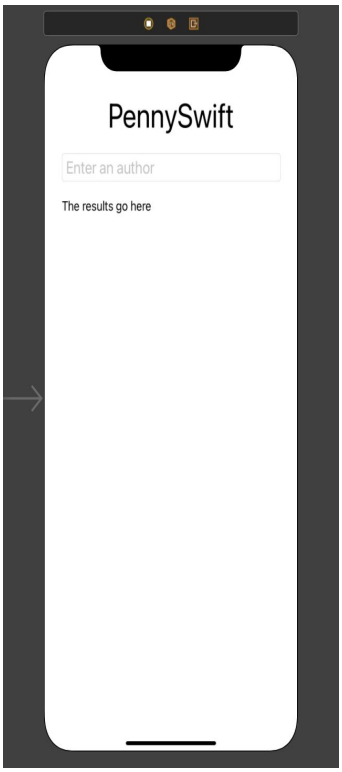




The toolbox contains different types of “views,” elements on the screen.

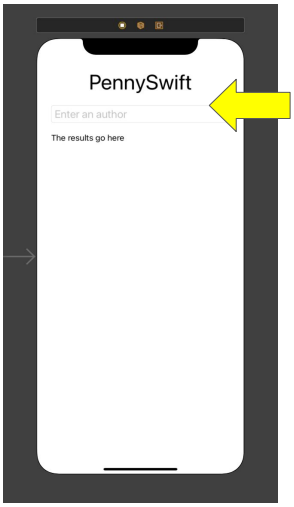
For PennySwift, we will be using

- **Label:** to display text
- **Text Field:** to receive user input

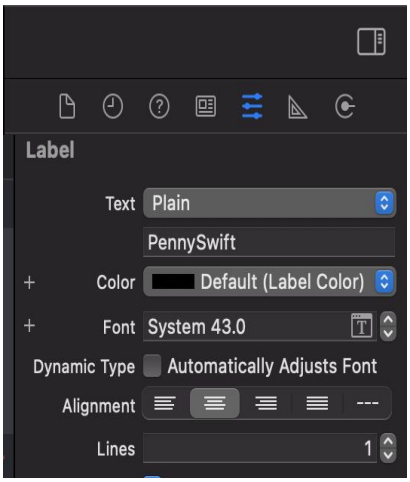


For reference, we want to build our final screen to look like this

Reference

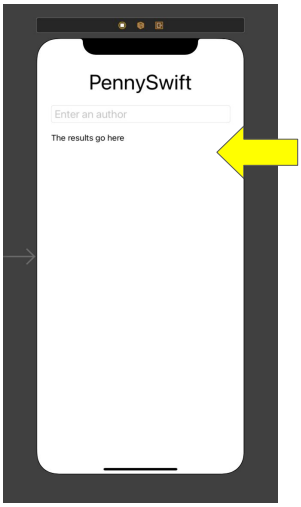


- Click and drag in a Label at the top center
- For the text put “PennySwift”
- Increase the font size (ex: use size 43)
- Center-align the text

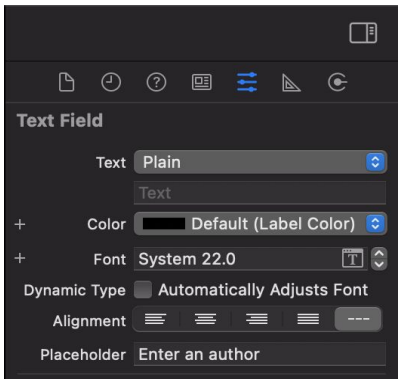


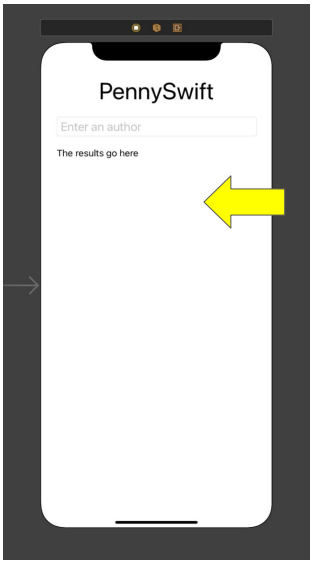
These are the “properties” of the label, which you’ll find by clicking on the label and then looking at the bar on the right side

Reference

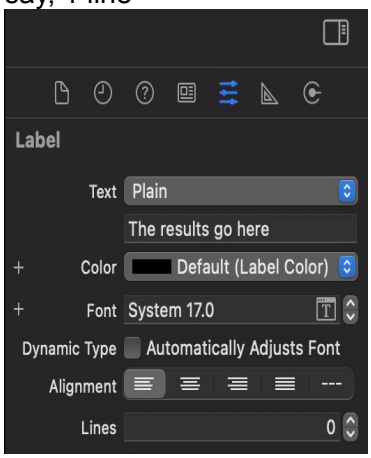


- Open the toolbox
- Click and drag in a Text Field
- For the placeholder text put “Enter an author”





- Open the toolbox
- Click and drag in a Label
- Put whatever you want for text. This is what will show when you load the app. For example, you could use “The results go here” or make the text empty
- Make the lines 0
    - This means the label will expand to fit however many lines the text needs instead of cutting off at, say, 1 line



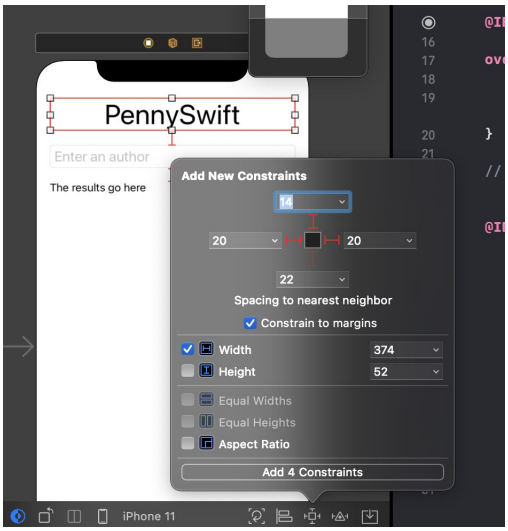
Adding constraints to our user interface

We add constraints so that

- Our app will look fine across different phone dimensions and orientations
- The views will be in the right location relative to each other

To add constraints, we

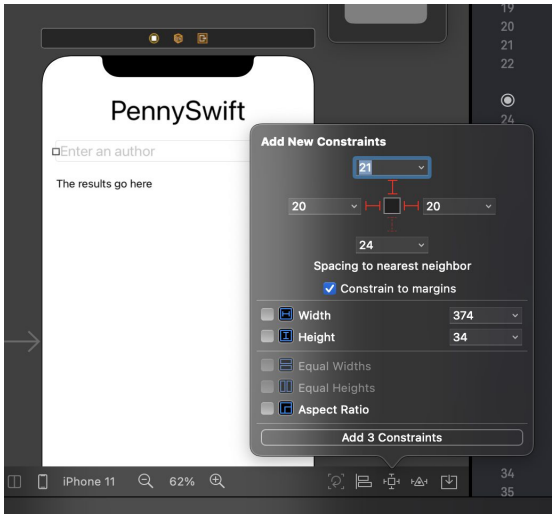
- Select a view
- Click on one of the options along the bottom right. For PennySwift, we will be using the option that looks like a box with T's around it



Select the PennySwift label

- Add constraints by clicking on the `I` and changing the number corresponding to
  - Above: 14
  - Left: 20
  - Right: 20
- Add a width constraint of 374

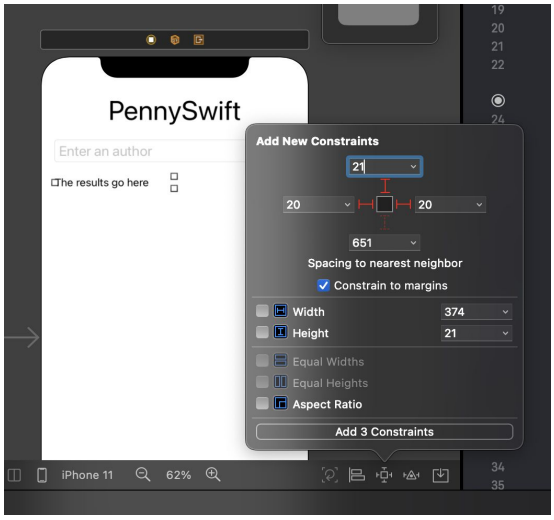
Note you can change the numbers according to how you want to design your app



Select the text field

- Add constraints for
  - Above: 21
  - Left: 20
  - Right: 20

Note: you can change the numbers according to how you want to design your app



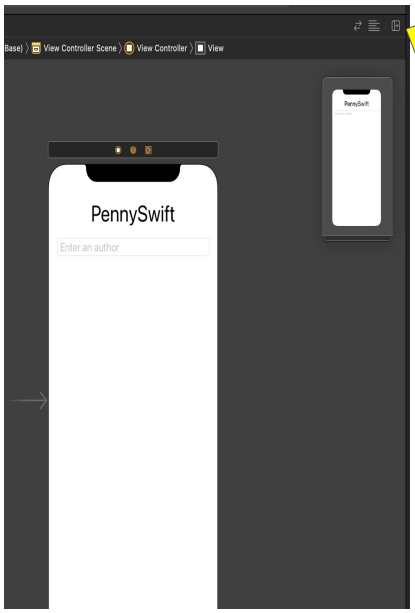
Select the results label

- Add constraints for
  - Above: 21
  - Left: 20
  - Right: 20

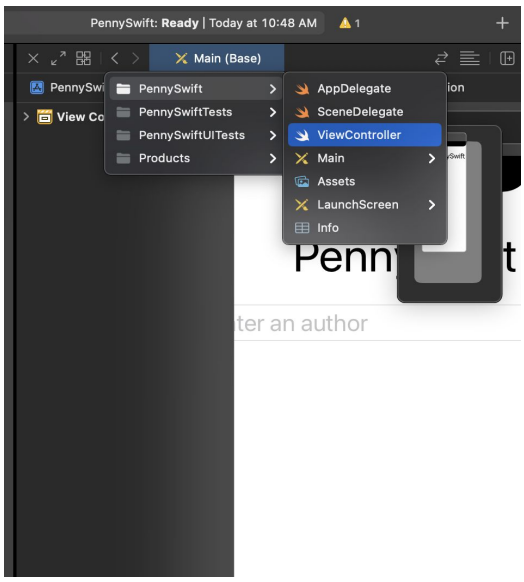
Note: you can change the numbers according to how you want to design your app

Connecting the user interface and code

We are going to open the Main storyboard and our code side-by-side to prepare to “connect” them



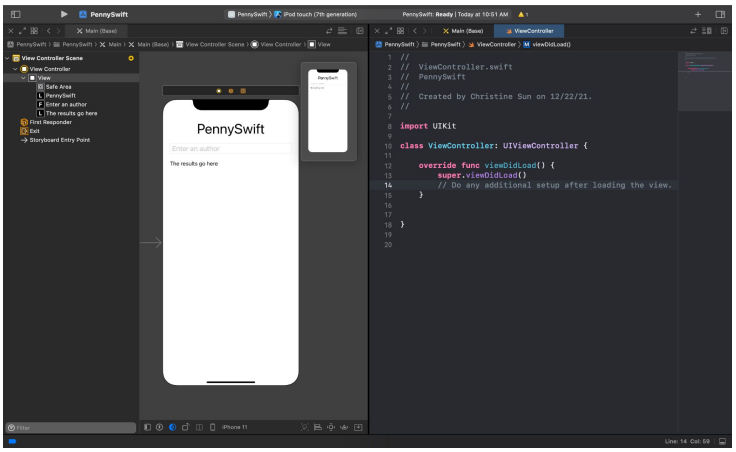
Click on this



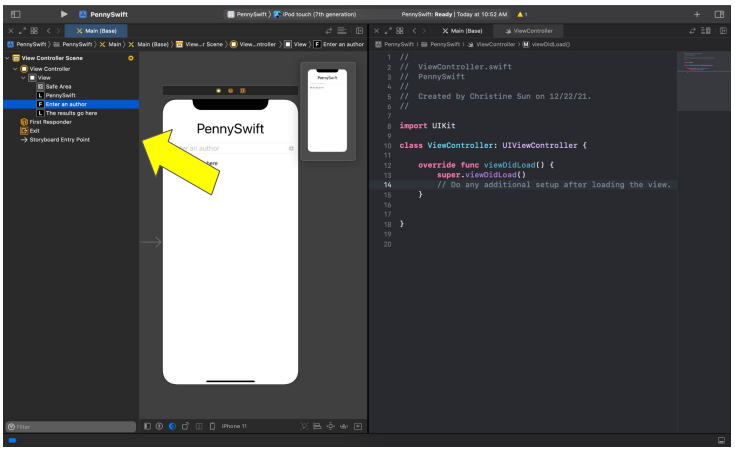
Use the navigator at the top of the section that displays to open the file “ViewController”

- Click “PennySwift” blue icon
- Click the “PennySwift” grey folder
- Click the “ViewController” file

The Main storyboard and ViewController file should now be side-by-side, like this



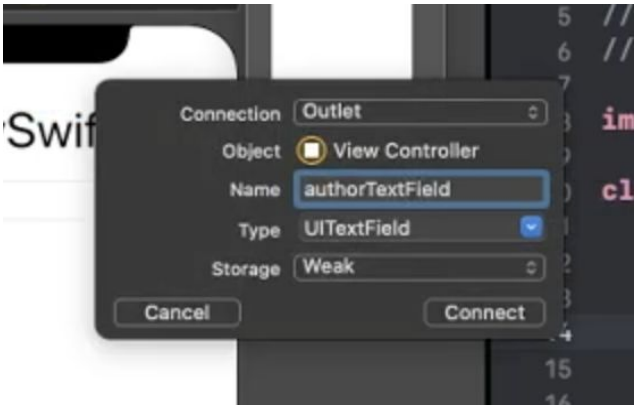
We are now going to connect the text field with our code so we can read the text that the user inputted



Click on the text field from the left hierarchy

- Hold the “Ctrl” button on your keyboard
- Drag from the text field on the left to the code on the right
- You should see a blue line. “Drop” the end of the blue line right above the `"override func viewDidLoad()"`

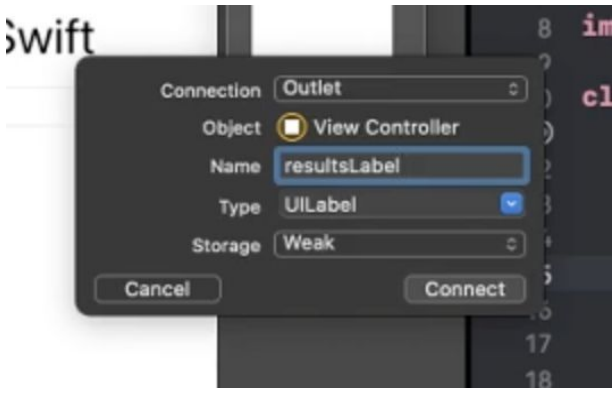
We just created an “outlet,” a connection from a view in our Main storyboard to a property in our code!



- Use a descriptive name for the text field, like “authorTextField”
- Click “Connect”

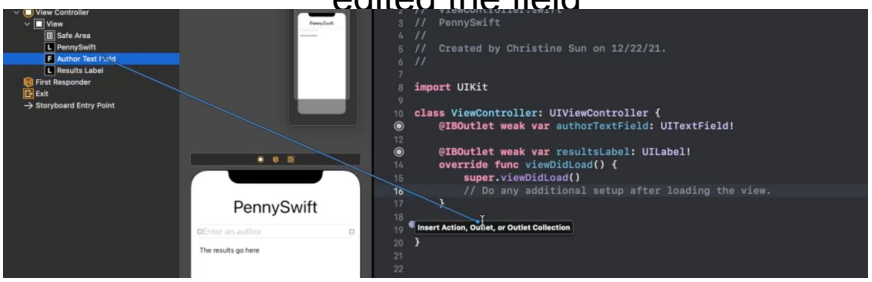
Repeat for the results label

- Click the results label on the left hierarchy
- Hold the “Ctrl” button on your keyboard
- Drag from the label on the left to the code on the right
- You should see a blue line. “Drop” the end of the blue line right above the `"override func viewDidLoad()"`



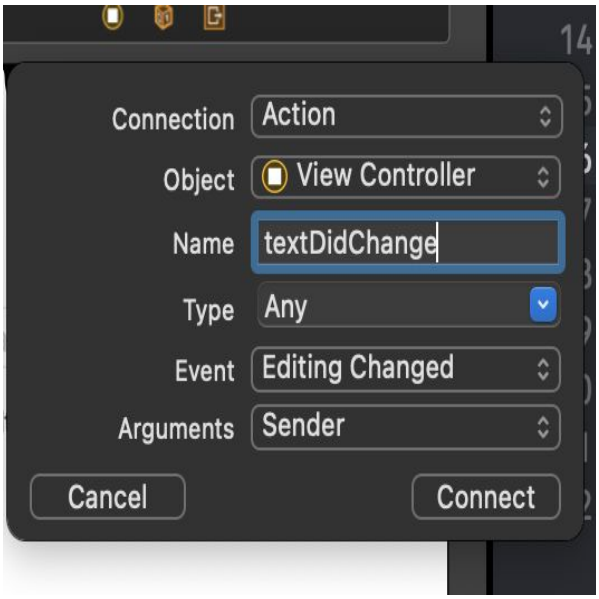
- Use a descriptive name for the label, like “resultsLabel”
- Click “Connect”

We are now going to connect the “Editing Changed” action of the text field with our code to detect whenever the user edited the field



- Click the text field on the left hierarchy
- Hold the “Ctrl” button on your keyboard
- Drag from the text field on the left to the code on the right
- You should see a blue line. “Drop” the end of the blue line under the function `"override func viewDidLoad()"`

We just created an “action,” a connection from an event to a function in our code!



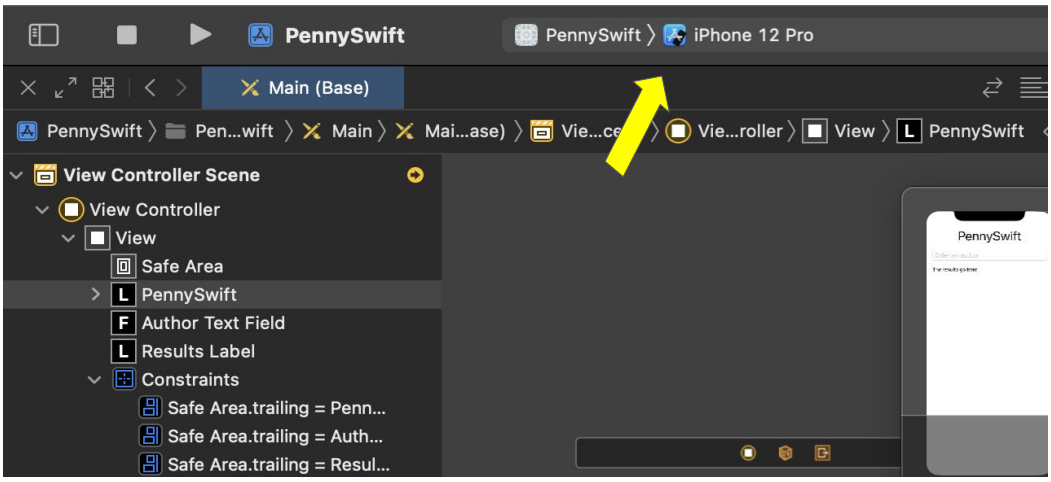
- Use a descriptive name for the action, like “textDidChange”
- Choose the Event “Editing Changed”
- Click “Connect”

```swift
class ViewController: UIViewController {
    // The text field users edit to search for an author
    @IBOutlet weak var authorTextField: UITextField!

    // The label that displays the results of the search
    @IBOutlet weak var resultsLabel: UILabel!

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view.
    }

    // This function is called every time the text inside authorTextField changes
    @IBAction func textDidChange(_ sender: Any) {

        // Get the text in authorTextField
        var author:String = self.authorTextField.text!

        // URL encode the author text input
        author = author.addingPercentEncoding(withAllowedCharacters: .urlQueryAllowed)!

        // Instantiate an object of the standard class URL that will be used to send to the server an
            HTTP request for the specified author
        let url =
            URL(string:"https://pennyandroidserver.herokuapp.com/searchresults?author=\(author)")!
        let task = URLSession.shared.dataTask(with: url) {(data, response, error) in
            if let data = data {
                // data is NOT nil
                DispatchQueue.main.async {
                    self.resultsLabel.text = String(data: data, encoding: .utf8)!
                }
            } else {
                // data IS nil
                DispatchQueue.main.async {
                    self.resultsLabel.text = error!.localizedDescription
                }
            }
        }
        task.resume()
    }

}
```

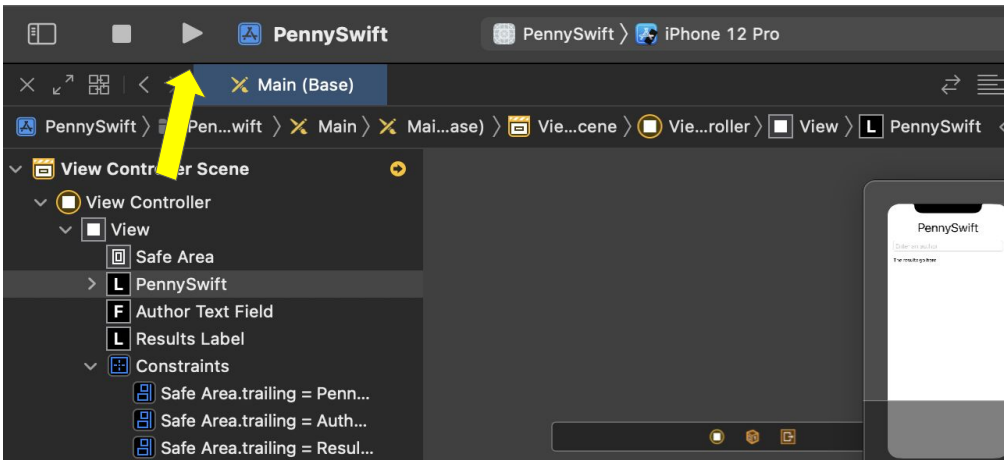
- Type the following code in the “textDidChange” function
  - This code is executed every time the user changes the text field

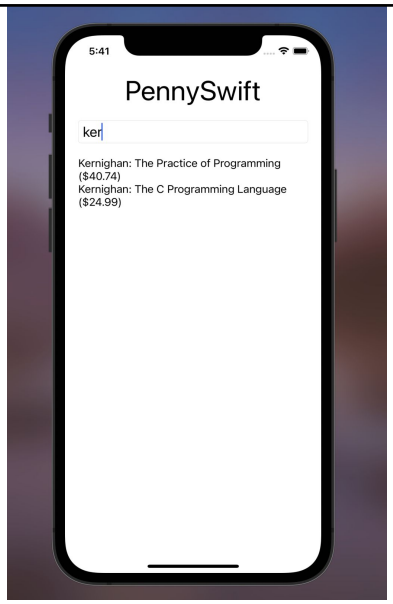
Running the app on the simulator

Select the deployment target



Click the play button to run the simulator





The simulator will open up in a new tab

Congratulations! You have finished your own PennySwift iOS app :)

Additional Resources

- Swift Basics Guide
  - https://guides.codepath.com/ios/Swift-Basics
- Uploading to the iOS App Store
  - https://christinesun.notion.site/christinesun/How-to-get-your-app-onto-the-iOS-app-store-018bcd0de6434878acb81c527f2efcb7

Our thanks go to Christine Sun for contributing this appendix to COS 333