CocoaPods
A *dependency manager* is a tool that makes it easy to add, remove, update, and manage third-party dependencies used by your app.

- For example, instead of reinventing your own networking library, you could simply pull in AlamoFire using a dependency manager.
  - You can specify the exact version you want to use, or even a range of acceptable versions.
  - Even if AlamoFire gets updated, your app can continue to use the older version until / unless you’re ready to update it.
CocoaPods

CocoaPods is a dependency manager designed specifically for Swift and Objective-C projects.

- Contains over 30,000 libraries
- Used in over 2,000,000 apps
- Built on top of Ruby, which ships with all recent versions of Mac OS X
- “pods” are libraries or frameworks added to your project via CocoaPods.
Prior to iOS 8, CocoaPods were created as “fat” static libraries. iOS 8 introduced dynamic frameworks, which allow code, images, and other assets to be bundled together.

- “fat” means they contained several different code instruction sets (i386 for the simulator, armv7 for devices, etc.) but static libraries were not allowed to contain resources like images and assets.

- Dynamic frameworks have namespace classes, and static libraries don’t. If you had two classes with the same name in different static libraries, Xcode would be perfectly happy building a project that has two classes with the same name.
Installing CocoaPods

- **RubyGems** is a packaging system written in the language Ruby. A **gem** is a library packaged in the correct format for **RubyGems** to install it.

- The command `gem install cocoapods` says to install the package called “cocoapods”.

- **sudo** is the Linux command for running a program under superuser privileges. (Enter your root password when prompted.)

So this command simply uses superuser privileges to install CocoaPods.
Installing CocoaPods

This clones the CocoaPods repository onto your computer.

- This will take a few minutes as it does the unpacking and copying. The `verbose` option is nice because it displays the task’s progress, which is reassuring since otherwise you might think it hung.

- The repository will be installed in `~/.cocoapods`.

- [https://guides.cocoapods.org/using/getting-started.html#sudo-less-installation](https://guides.cocoapods.org/using/getting-started.html#sudo-less-installation)
Firebase
What is Firebase?

Firebase is a “Backend-as-a-Service” (BaaS). It is a mobile and web application development platform that manages servers for you so you can focus on your app.

- **Real-time database:** you can create a database on a server and access it through a Web socket, which is much faster than HTTP.

- **File storage:** you can save binary files (especially images) securely on Google Cloud Storage.

- **Authentication:** Firebase auth has a built-in email/password authentication system you can use for your app.
What is Firebase? (cont.)

Lots of other stuff, too. . .

- Hosting
- Analytics
- Cloud Messaging
- A collection of other Google products, including
  - Remote config
  - Test Lab
  - Crash
  - Notifications
  - Dynamic Links
  - AdMob
Setting up Firebase

1. Create a Google account, if you don’t already have one. (You can use the same one you have for Gmail, Google Drive, etc.)
2. Go to https://console.firebase.google.com

3. Add Firebase to your project by clicking on “Add project”
Setting up Firebase (cont.)

4. Enter your project name. Accept the T&Cs and click “Create Project”. Wait for it to complete.
Setting up Firebase (cont.)

This takes you to the “Get started” screen.
Setting up Firebase (cont.)

If you scroll to the bottom, you have links to help you manage Authentication, Storage, and any other services you want to include in your project.
Setting up Firebase (cont.)

Back at the top, click on the big "iOS" circle labeled, “Add Firebase to your iOS app”.
Setting up Firebase (cont.)

Enter the Bundle ID for your app. (Remember how to find this?) Click on “Register app”.

![Add Firebase to your iOS app](https://console.firebase.google.com/project/groc-rf16d/overview)
This will create a **GoogleService-Info.plist** file. Follow the instructions, and move it to your project in Xcode. (Be sure to check “Copy Items if needed” in Xcode.)
Setting up Firebase (cont.)

Clicking “Next” will give you the instructions on how to add the Firebase SDK to your project. This is the pod stuff we talked about earlier.

- Download config file
- Add Firebase SDK
  
  Google services use [CocoaPods](https://cocoapods.org) to install and manage dependencies. Open a terminal window and navigate to the location of the Xcode project for your app.

  Create a Podfile if you don't have one:
  
  ```sh
  $ pod init
  ```

  Open your Podfile and add:
  
  ```sh
  pod 'Firebase/Core'
  ```

  Includes Analytics by default

  Save the file and run:
  
  ```sh
  $ pod install
  ```

  This creates an `.xcworkspace` file for your app. Use this file for all future development on your application.

- Add initialization code
- Run your app to verify installation
Clicking “Next” from here will show you how to connect Firebase when your app starts.

To connect Firebase when your app starts up, add the initialization code below to your main AppDelegate class.

```swift
import UIKit
import Firebase

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

    var window: UIWindow?

    func application(_ application: UIApplication,
        didFinishLaunchingWithOptions launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool {
        FirebaseApp.configure()
        return true
    }
}
```
Setting up Firebase (cont.)

- Now build and run your app.
- Once it’s running, click on the last “Next”, and the Firebase console will test to see if it can successfully connect to your app.
- If it does, you’re done!
Using Firebase in your code

All data stored in Firebase in formatted in JSON. Suppose you created a JSON structure in the database like this:

```json
{
  "books": {
    "LOTR": {
      "title": "Lord of the Rings",
      "author": "Tolkien"
    },
    "Potter": {
      "title": "Harry Potter and the Sorcerer’s Stone",
      "author": "Rowling"
    }
  }
}
```
Using Firebase in your code

All Firebase keys map to *paths*.

That means you can refer to all of the books using the name

`books`

and you can refer to the second book this way:

`books/Potter`

Note that this means you have to choose these tokens wisely. (Avoid spaces, special characters, etc.)
Using Firebase in your code

To access a Firebase database in your code, add the following to a ViewController:

```swift
let ref = Database.database().reference(withPath: "books")
```

In Firebase terminology, properties are referred to as references. The idea is that they refer to a location in your Firebase database.
To add something to your database:

```swift
// Create a new book as a dictionary object
let newItem = [
    "title": "Horton Hears a Who",
    "author": "Seuss"
]

// Create a new child reference with key "Fish"
let newItemRef = self.ref.child("Horton")

// Use setValue to save it to the database
newItemRef.setValue(newItem)
```
Accessing your database

If you try to run your app now, it won’t save anything because you haven’t given yourself write access to the database.

From the web console for your app, in the list of services at the bottom of the screen, find “Database” and click on “Get started”.
Accessing your database (cont.)

Find ”Realtime Database” and click on ”Get Started”.

Cloud Firestore Beta
The next generation of the Realtime Database with more powerful queries and automatic scaling
Learn more

Get started

Realtime Database
Store and sync data in realtime across all connected clients
Learn more

Get started

Not sure which to choose? View comparison
Accessing your database (cont.)

Select “Start in test mode” to give all users who reference your database read and write access.
Select the tab “Rules”. You’ll see that the default is to set both read and write to true.

If you rebuild and run your code, you can see it is writing to the database.
Add a call to `ref.observe` to `viewDidLoad()` (or wherever you want the app to update itself):

```swift
ref.observe(.value, with: { snapshot in
    print(snapshot.value as Any)
})
```

`ref.observe` takes two parameters:

- **An instance of** `DataEventType`. `.value` **means any event**, including adds, deletes, and changes.

- **A closure that dictates what you want to do when the event occurs.** `snapshot` **is just a snapshot of the data at that specific moment in time**. The above just prints all of the data; you have to write code to parse the JSON.
Authentication by email address

Find “Authentication” and either click on “Set up sign-in method” or click on the “Sign-in method” tab.
Authentication by email address (cont.)

Select “Email/Password”.

![Firebase Authentication](https://console.firebase.google.com/project/grocr-6b2ef/authentication/providers)

<table>
<thead>
<tr>
<th>Provider</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Email/Password</td>
<td>Disabled</td>
</tr>
<tr>
<td>Phone</td>
<td>Disabled</td>
</tr>
<tr>
<td>Google</td>
<td>Disabled</td>
</tr>
<tr>
<td>Play Games</td>
<td>Disabled</td>
</tr>
<tr>
<td>Facebook</td>
<td>Disabled</td>
</tr>
<tr>
<td>Twitter</td>
<td>Disabled</td>
</tr>
<tr>
<td>GitHub</td>
<td>Disabled</td>
</tr>
<tr>
<td>Anonymous</td>
<td>Disabled</td>
</tr>
</tbody>
</table>
Authentication by email address (cont.)

Select “Enable” and then “Save”.

Allow users to sign up using their email address and password. Our SDKs also provide email address verification, password recovery, and email address change primitives. Learn more.
Authentication by email address (cont.)

Since Firebase stores credentials in the keychain, in Xcode, go to your project properties, and under the Capabilities tab, toggle “Keychain Sharing” on.

Now you can authenticate users for your app using their email and password.
Use `Auth.auth().createUser` to register a new user into the authentication database. It takes three arguments:

- `withEmail`: a text string, ideally the `text` parameter from a `textField` object.
- `password`: a similar text string.
- a closure specifying what to do once the user is created.
If there are no errors with the email address or password, use `Auth.auth().signIn` to authenticate an email address / password pair. It takes two arguments:

- **withEmail**: a text string, ideally the `text` parameter from a `textField` object.
- **password**: a similar text string.
If we want to immediately log the user in once the account is created, we put the call to `signIn()` in the closure of `createUser()`:

```swift
// Assume uidField and pwdField are outlets of text fields

Auth.auth().createUser(
    withEmail: uidField.text!,
    password: pwdField.text!
) { user, error in

    if error == nil {
        Auth.auth().signIn(
            withEmail: self.textFieldLoginEmail.text!,
            password: self.textFieldLoginPassword.text!
        )
    }
}
```