Dependency Management
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 third-party 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

CocoaPods is still a very useful tool for adding and managing dependencies in a project, but there is another option that is starting to replace it.
Swift Package Manager

The *Swift Package Manager* (SwiftPM) has become the easier option for iOS dependency management.

- With it, you no longer have to rely on a third party dependency manager such as CocoaPods.
- It's been around since Swift 3.0, but only available for a limited collection of packages.
- Apple significantly improved support for it in Swift 5 / Xcode 11.
- A *package* is simply a collection of Swift source code files, plus a metadata ("manifest") file called Package.swift that defines various properties about the package, such as the name, code dependencies, etc.
Adding a package using SwiftPM

To add a package to your project:

1. Select your project
2. Select the "Package Dependencies" tab
3. Click the "+" button
Adding a package using SwiftPM (cont.)

This pops up a window where you can either use a recently used package, or enter a URL where the package can be found.
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 though 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.)
Setting up Firebase (cont.)

2. Go to https://console.firebase.google.com

3. Add Firebase to your project by clicking on “Create project”
3. Enter your project name.

Let's start with a name for your project

Project name
CS371L Homework

Continue
Setting up Firebase (cont.)

4. It will offer to set up Google Analytics. Either accept (and complete an additional screen) or decline and hit “Create Project”. Wait.
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 "iOS" circle.
Enter the Bundle ID for your app. (Remember how to find this?) Click on “Register app”.

1. Register app
   - iOS bundle ID
     - com.company.appname
   - App nickname (optional)
     - My iOS App
   - App Store ID (optional)
     - 123456789

2. Download config file
3. Add Firebase SDK
4. Add initialization code
5. Read the Get Started Guide for iOS
Setting up Firebase (cont.)

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.)
Clicking “Next” will give you the instructions on how to add the Firebase SDK to your project. By default, it will show you the instructions for Swift Package Manager, but you can switch it to CocoaPods if you need to.

1. In Xcode, with your app project open, navigate to File > Add Packages
2. When prompted, enter the Firebase iOS SDK repository URL:
   https://github.com/firebase/firebase-ios-sdk
3. Select the SDK version that you want to use.
   We recommend using the default (latest) SDK version, but you can use an older version, if needed.
4. Choose the Firebase libraries that you want to use.

After you click Finish, Xcode will automatically begin resolving and downloading your dependencies in the background.
Clicking “Next” from here will show you how to connect Firebase when your app starts. Click “Continue to console”. Build and run your app.
All data stored in Firebase is 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:

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

// Create a new child reference with key "Horton"
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.)

Click on "Cloud Firestore", and then "Create Database".
Accessing your database (cont.)

Select “Start in test mode” to give all users who reference your database read and write access.
Retrieving data

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 screen showing email/password sign-in method enabled](image-url)
Authentication by email address (cont.)

Select “Enable” and then “Save”.
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.

![Capabilities settings in Xcode](image)

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.

Note: Firebase expects both the userid and password and be at least 7 characters long!
Authenticating by email address (cont.)

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: uidField.text!,
            password: pwdField.text!)
    }
}
```