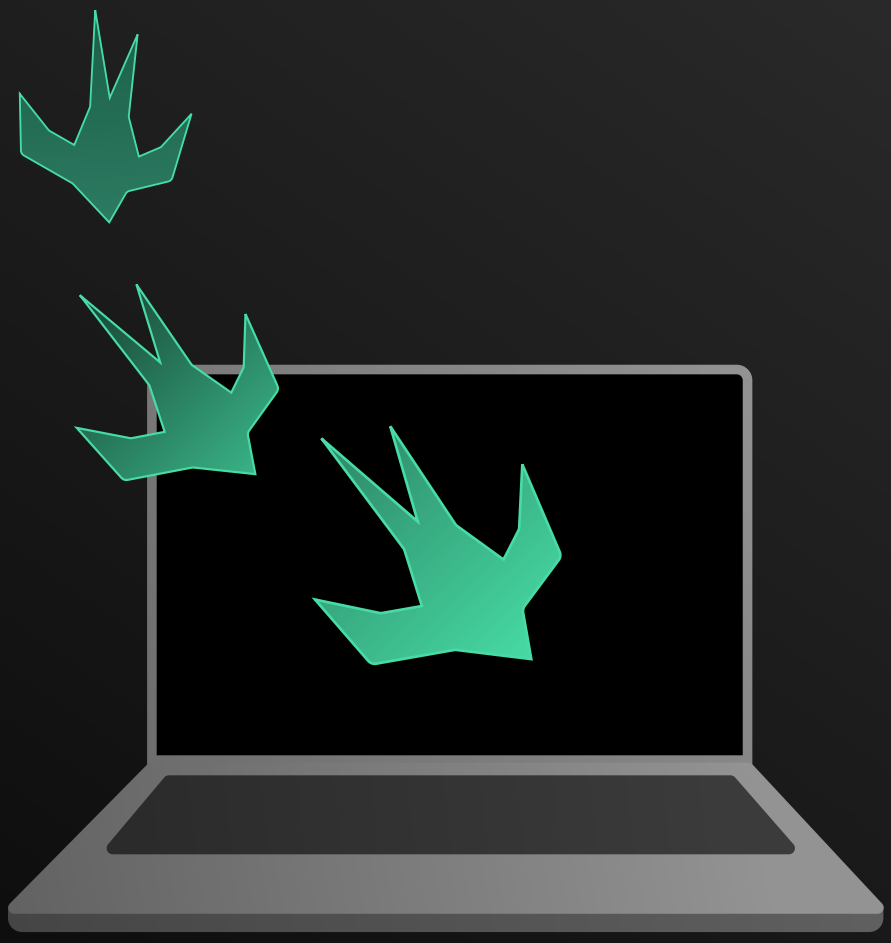
iOS Development Tips

for Junior Developers



Enid Hadaj

Thank you

for purchasing my eBook

DISCLAIMER

The information contained in this eBook is based on my personal experience and opinions. It is provided for general information purposes only and does not constitute professional advice. While I have made every effort to ensure the accuracy of the information at the time of the publication, I don’t assume any responsibility for errors, omissions, or contrary interpretations of the subject matter. You should take your own personal circumstances into consideration and, if necessary, seek professional advice before acting on any of the information contained in this eBook.

1. Developers are not geniuses (someone may be).

They are people like you who have learned programming over time.

2. Don’t compare yourself with others.

Please don’t make that mistake.
You cannot compare your '6 months' experience with someone else's '5 years' experience.

3. Every senior developer was once a junior developer.

Please remember this every time you feel discouraged.

4. Even senior developers search for the basic things on the web.

Yes, this is true.

5. Ignore those who say, “You can’t do it”.

They are talking about their limits, not yours.

6. Don’t try to learn everything at once.

Trying to learn everything at once is stressful and doesn’t help. You can’t do it, and no one can. Take your time :)

7. You should enjoy developing iOS apps.

If not, it’s impossible to succeed.

8. Submitting an app to the App Store is tempting.

I know that feeling, but you must be patient and learn the fundamentals first.
Remember that there are no shortcuts.
Learn the basics and the fundamentals, and gradually learn more advanced topics.
Day by day, it becomes even more enjoyable.

9. Practice what you learn.

You are doing it wrong if you’re watching some tutorials without practicing them.
You cannot learn just by watching or reading.
Practice is the key.

10. Train your ‘problem-solving’ skills.

In programming, ‘problem-solving’ skills are a must.
Every developer faces challenges with their codebase.

If you are stuck on something:

- Try to overcome it using your knowledge first.
- If you need help to solve it, try to search on the web.
- If you cannot find answers online, don’t hesitate to ask for help.

11. Don’t just copy & paste.

Don’t copy and paste directly the code you find on the web (Stack Overflow, ChatGPT, etc.) without first understanding it, even if it works. It’s important to know what you add to your code. Otherwise, you will find it challenging to debug it later on.

Also, you will learn something new.

12. Learn from video tutorials first.

Learning from video tutorials is more helpful than learning from books when you are a beginner. Of course, books are helpful, but not in this phase.

13. Don’t start learning about AI, AR, etc., without knowing the variables first.

Start from the basics.

14. Don’t try to memorize code.

You are a human, not a robot.

15. Learn Swift programming language.

Please don’t try to build an app without knowing the basics of the Swift programming language first.
Learn its syntax. Swift is a beautiful modern language.

16. Skip Objective-C.

Swift is the present and future, and learning and writing in Swift is easier than Objective-C.
Most apps are written in Swift, and only some old codebases are written in Objective-C.

17. Be curious about Apple platforms and technologies.

18. Follow Apple design guidelines.

The design is crucial because it “talks” to the user.

19. Stack Overflow is your best friend.

Stack Overflow has almost all the answers to many iOS development questions.

20. ChatGPT may be your friend too.

21. Don’t be tempted by new frameworks and technologies.

Wait until they are mature enough so you can start implementing them gradually in your apps.

22. “Play” on Playground.

Playground in Xcode is a great feature for learning and practicing Swift programming language.

23. Use Camel case.

Camel case

```
var isAvailable: Bool = true
```

Snake case

```
var is_available: Bool = true
```

Kebab case

```
var is-available: Bool = true
```

Pascal case

```
var IsAvailable: Bool = true
```

24. Choose ‘let’ over ‘var’.

Choose constants over variables if the value doesn’t change. This will make your code even safer and easier to understand.

For example, the name doesn’t change. Therefore, it doesn’t make sense to declare it as a variable. On the contrary, the age will change, so it makes sense to declare it as a variable.

```
var name: String = “John”
var age: Int = 20
```

```
let name: String = “John”
var age: Int = 20
```

25. Be careful with force unwrapping optionals (!).

Force unwrap an optional only if you are sure that it contains a value. However, even in that case, try to avoid it to prevent an unexpected crash.

26. Choose ‘switch’ over ‘if-else’ statement.

If the value compares against three or more possible matching patterns, consider using the ‘switch’ statement to make your code more readable.

27. Choose ‘struct’ over ‘class’.

Choose ‘struct’ over ‘class’ where possible. Structures make it easier to “control” your code without considering the whole state of the app. This is because, unlike classes, structures are value types, not reference types.

28. Say “Hi” to enumerations.

Enums in Swift are very powerful. They make your code type-safe, error-free, and more readable.

```
enum
```

29. Don’t skip Closures.

I know closures are hard to learn, especially when you are new to iOS development, but closures are present everywhere (in SwiftUI, too). After learning them, you will find them very useful.

30. Learn Generics.

Generics may seem abstract initially, but they help make your code reusable.

31. Learn Protocols.

Learn protocols and the delegate pattern. The delegate pattern will help pass data in your app, especially when working with UIKit.

```
protocol
```

32. Learn about Functional Programming.

Consider making use of Functional Programming, but don’t overdo it. Remember to keep it simple.

33. Try to avoid global variables.

Using global variables to communicate data in your app is not considered safe because they are accessed from different places in your code, and you don't have complete control of them. Also, using global variables makes the code hard to debug.

```
import Foundation

var isAvailable = true
```

34. Code clean.

Removing empty/unnecessary spaces makes your code look clean.

```
func doSomething(){

        print(“Hello”)
}
```

```
func doSomething() {
    print(“Hello”)
}
```

35. ‘^ + i’ to re-indent the code.

‘^ + i’ to re-indent your code and make it look cleaner.

```
func doSomething() {
        print(“Hello”)
}
```

```
func doSomething() {
    print(“Hello”)
}
```

36. Code now, refactor later.

Yeah, this is true. Don’t be obsessed with the “perfect” code.
Make your app run successfully first, then try to refactor the code.

37. Don’t use User Defaults to store everything.

User Defaults is here to help you store some small data (for example, user preferences) and not to replace a database.

38. Start learning SwiftData.

It’s a good idea to start learning SwiftData gradually. SwiftData is a new framework but is not mature enough and will not replace Core Data anytime soon.

39. Don’t avoid Core Data.

Core Data is a powerful framework that helps you store data in your app and more.
Many apps use Core Data, so don’t try to avoid it.

40. Choose UIKit over SwiftUI.

Yes, I know SwiftUI is the future.
However, if you want to get a job as an iOS developer anytime soon, you must first learn UIKit, as most apps are developed with UIKit.
UIKit is a ten-year-old framework and more mature than SwiftUI.
You can learn SwiftUI after learning UIKit.

```
import UIKit
```

```
import SwiftUI
```

41. Go with SwiftUI if developing your own apps.

If you are developing your own apps (or don’t plan to search for a job anytime soon), then choosing SwiftUI would be wise.
For most of the development part, SwiftUI will do the job.

42. UIKit + Swift is powerful.

Making use of both frameworks in a project means even more benefits.

For example, if you have developed an app in UIKit, you can use SwiftUI to create secondary views.

43. Programmatic UI is better than Storyboards.

If you are using UIKit, start with storyboards, but gradually start learning the programmatic way after some time.
With the programmatic approach, you will have more control over your code.

44. Learn about Networking.

As an iOS developer, you should learn about Networking to communicate data with a server. At least learn how to make API calls to get and send data.

45. URLSession is powerful.

Try to avoid third-party dependencies.
URLSession is powerful and flexible. You can create your custom network layer with it.

`URLSession`

46. Learn some Backend stuff.

This is a good advantage. You will be even more confident when collaborating with backend developers.

47. Learn about Multithreading.

Multithreading is inevitable, especially when making API calls.
Learn some topics about Multithreading, Concurrency, Synchronous/Asynchronous programming, and GCD.

Most of the time, the two lines below are sufficient. :D

```
DispatchQueue.main.async {}
```

```
DispatchQueue.global().async {}
```

48. UI code on the main thread.

Always run the code responsible for the UI on the main thread.

```
DispatchQueue.main.async {
    // UI code
}
```

49. Don’t execute heavy tasks on the main thread.

Don’t forget to execute time-consuming tasks (data processing, downloading, etc.) on a background thread so you don’t block the main thread, which is responsible for the user interface.

50. Learn Swift concurrency.

Learn Swift concurrency to write asynchronous code in your projects easily.

```
async    await
```

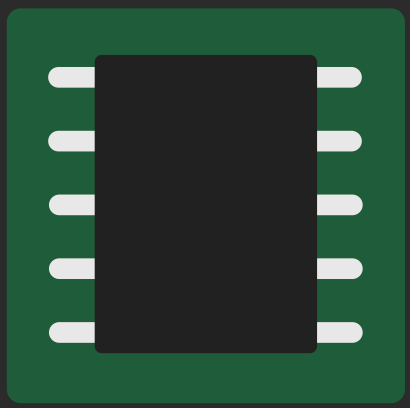
51. Don’t use ‘self’ everywhere.

Don’t use ‘self’ if not needed.
Use it only if required from the code or logic.
Remember to make your code clean.
The best code is no code at all.

```
self
```

52. Learn about Memory Management.

This is a must in iOS development. Don’t forget to check for memory leaks in your code.



53. [weak self] > [unowned self]

If you’re unsure which one to choose, go for [weak self].
At least your code will not crash.

`[weak self]` 

54. ‘deinit {}’ is your friend.

This method is called when a class instance is deallocated.
It’s a good idea to use it to check for memory leaks in your code
(especially when working with UIKit).

```
deinit {
    print(“DEBUG: AccountVC deallocated”)
}
```

55. Learn about Encryption.

Protecting sensitive data is very crucial, so learn how to encrypt data.

56. Don’t store sensitive data in User Defaults.

Storing sensitive data in User Defaults is not a good practice because User Defaults itself is not secure.
Instead, storing them in Keychain is a good choice because Keychain is encrypted and offers high security.

User Defaults 

Keychain 

57. Be careful with the ‘print()’ method.

The ‘print()’ method is a quick and easy debugging tool, but don’t forget to remove it from the release versions, especially if you are printing sensitive data.
Try to use breakpoints and other debugging methods as well.

```
print(“Yesssssssssss”)
```

58. ‘⌘ R’ to run the build.

Use the ‘⌘ R’ keyboard shortcut to run the build.

59. ‘⌘ .’ to stop the build.

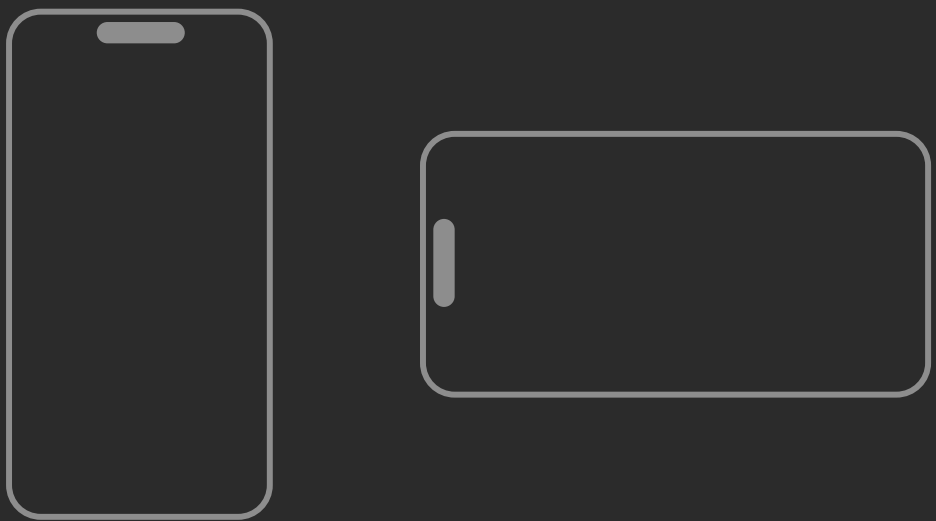
Use the ‘⌘ .’ keyboard shortcut to stop the build.

60. ‘⇧ ⌘ K’ to clean the project.

Use the ‘⇧ ⌘ K’ keyboard shortcut to clean the project.

61. ‘⌘ →’ or ‘⌘ ←’ to change the orientation.

Use the ‘⌘ →’ or ‘⌘ ←’ keyboard shortcuts to change the simulator orientation.

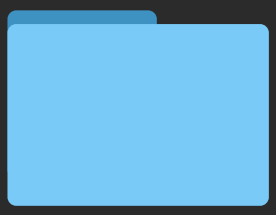


62. ‘⇧ ⌘ A’ to switch between light and dark mode.

Use the ‘⇧ ⌘ A’ keyboard shortcut to switch between light and dark mode.

63. Delete the derived data.

If Xcode is not behaving correctly, the first thing that comes to mind to do (after cleaning the project) is to delete the derived data and re-run the project again.



DerivedData

64. Learn some Terminal commands.

Learning some basic Terminal commands is a must for an iOS developer.

65. Learn about Version Control.

Version Control is a great way to back up your project files, manage changes to your code, and collaborate with other developers. Using Version Control is a wise approach. The most popular system is Git.

66. Limit the usage of third-party libraries.

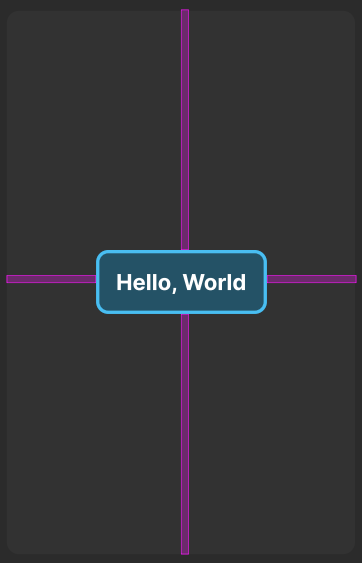
Too many dependencies in your project is not a good idea.
You will have little control over your codebase because you have other libraries implemented, which other authors maintain.
When a library is not maintained, you are forced to find another solution or library.
Try not to use third-party libraries for simple tasks or implementations.

67. Don’t install iOS beta versions on your primary device.

Installing new iOS beta versions is tempting, but try not to install them on your primary device. Beta versions are buggy and unstable. Consider installing them on a secondary device you use for development to test your app on new iOS versions.

68. Consider learning some UI & UX skills.

Having some UI/UX skills will give you a significant advantage. Try to follow some courses on UI/UX to learn about layout, color combinations, typography, etc.



69. Implement some animations, but don’t overdo it.

An app without animations is boring.
An app with too many animations may be confusing.

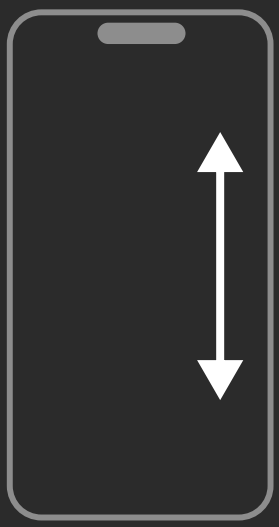
70. Make use of SF Symbols.

SF Symbols is created by Apple, it’s free to download and easy to use.
It contains a lot of symbols that you can use in your apps.
They are beneficial for the UI part, even if you don’t have design skills.

```
UIImage(systemName: “person.fill”)
```

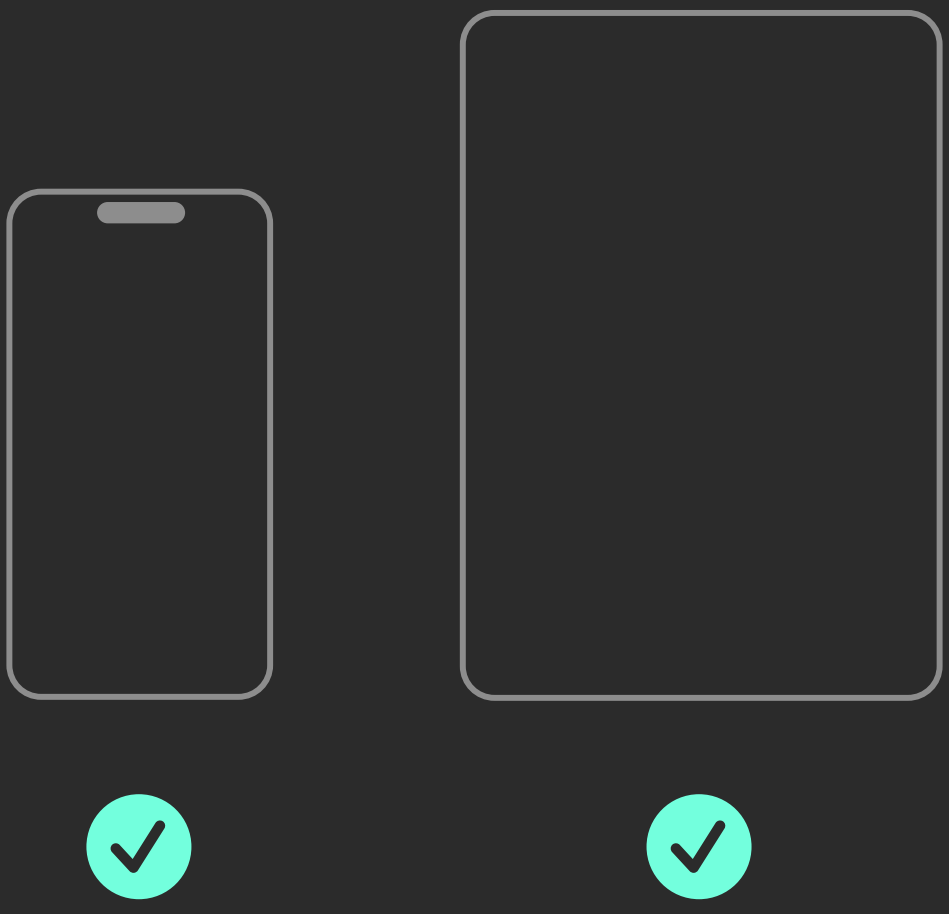
71. ‘Scroll View’ is the answer.

Adapting your app layout for multiple screen sizes is challenging for most developers, especially when designing for smaller screens like iPhone 8.
In this case, embed your UI elements in a scroll view.



72. Don’t forget to test your app on an iPad.

iPads have different screen sizes and some different APIs, so make sure to test your app on an iPad as well.
You can directly use the iPad simulator.



73. Test your app on different iOS versions.

If your app supports several iOS versions, remember to test it on those versions before launching it on the App Store.
This is a safe way to catch a bug before the launch.

74. Learn about UI and Unit testing.

UI and Unit testing have a lot of benefits, such as faster debugging, fewer bugs, etc.

75. Use Instruments.

Instruments in Xcode is a powerful tool for testing your app, finding memory issues, and more.

76. Test the app before sending it to the QA team.

It’s always a good idea to test the app before sending it to the QA team.

77. Understanding the life cycle of the app.

For example, learn how to handle the logic when your app is in the foreground or background, etc.

78. Reduce the app size.

Try to reduce the app size by removing unnecessary code, libraries, assets, etc.

79. Don’t forget to take a break.

If you’re facing a challenge solving a task, consider quitting Xcode and try again later.

80. Stop saying, “But it works on my machine.”

Please don’t say this. Don’t make that mistake. I have done it before. Please remember that there is no perfect code, and always try to test several use cases.

81. Ask for code reviews.

Don’t be afraid of asking for code reviews.
Asking for code review will help you become a better developer and improve your code quality.

82. Watch the WWDC videos.

Following the WWDC videos is an excellent way to increase your knowledge as an iOS developer, learn new APIs, etc.

83. Learn how to submit an app to the App Store.

As an iOS developer, you should learn how to submit an app to the App Store. Learn about certificates, provisioning profiles, etc.

84. Be familiar with App Store Connect.

Being familiar with App Store Connect is a good advantage.
Learn how to add and configure new apps, view analytics, add roles, etc.

85. Share your experience with another iOS developer.

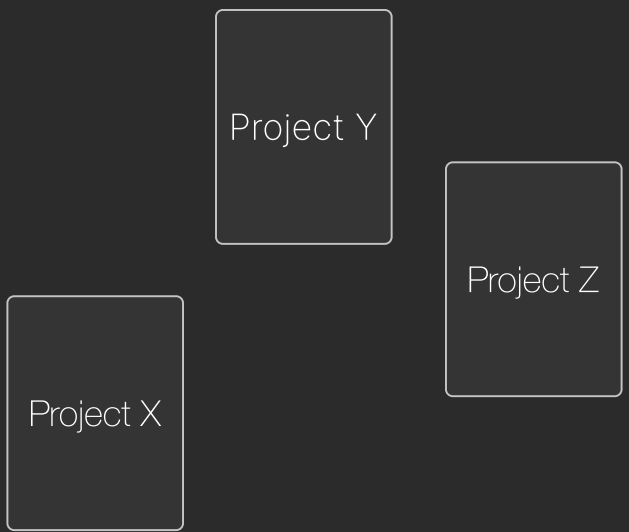
You will learn something new from each other.

86. Collaborate with your team.

It’s a good idea to share your experience with other developers of different fields (Android, Backend, etc.)

87. Learn from other projects.

Always try to learn from other projects developed by other developers.
Doing so will help you learn more about app architectures, coding styles, new APIs, etc.



88. Dual monitors.

In general, developers prefer dual monitors for their daily work. Utilizing extra space makes you more productive.

89. Create a GitHub account.

Having your portfolio on GitHub is also a great way to promote your work when looking for a job.

90. Create an account on X (formerly Twitter).

The iOS developer community on X is fantastic and very supportive, so consider joining (if you still need to) and interacting with other folks.

91. Create a LinkedIn account.

LinkedIn is a great place to search for a job.

92. Consider creating your own app.

You will learn more by creating an app from scratch than jumping directly to an existing project.

93. Learn some marketing skills.

Why not make money from your apps? :D

App → Money

94. Enjoy your journey.

Remember that the most important thing is to enjoy what you are doing. :)

Thanks for reading :)

iOS Development Tips for Junior Developers

V3

COPYRIGHT