10 Ways to Build Mac Apps in 2021

Yan Li

Disclaimer

These are mostly personal opinions.

It's about as serious as a YouTube video.

Why Mac Apps in 2021 - Ergonomics

- For content consumption apps (e.g. streaming, games)
    - More screen options
    - More headroom for performance intensive contents
- For content creation apps (e.g. media editing, developer tools)
    - macOS is designed for mouse+keyboard input
    - Full system resource access
    - Better multi-tasking support
    - Better interoperability among apps

Why Mac Apps in 2021 - Business

- Many use cases are simply not possible on iOS/iPadOS
  - Multiple windows for a single process
  - Interprocess communication
  - Filesystem monitoring
  - Supporting USB/PCIe devices
- macOS users are willing to pay more per app
- Expanding audience for existing iOS/iPadOS apps

The Mac App Store - Pros

- Apple handles payments and authroization
    - App purchase and in-app purchases
- Apple handles downloads
    - Installation and updates
    - (New in 2021) TestFlight for macOS
- MAS featuring and search results may bring some traffic
- Making better use of the US$99 annual fee
    - It's required for codesigning and notarization already

The Mac App Store - Cons

- Apple takes 30% of the sales
    - Or 15% if you are a small business
- App Store business model limitations
    - E.g. no paid upgrades
- App Sandboxing is mandatory
- App Store Review

Core Business Logic

- Objective-C
- Swift
- Any language with support of:
    - Compiling into Mach-O binaries for x86_64 and arm64
    - C ABI
    - E.g.: C, C++, Rust

User Interface

Guessing game: Which toolkit does this app use?

Home

Find My

Mac App Store

Alacritty

Discord

Serato Studio

Sublime Text

1Password

Things

Google Chrome `about:tracing`

Google Chrome

egui

https://emilk.github.io/egui/index.html

End of guessing game

AppKit

(The "native" macOS toolkit.)

(macOS. Only.)

- Pros
  - Authentic Mac OS X experience since 10.0
  - "Free" access to Apple API goodies
- Cons
  - The app will only be able to run on macOS
  - Many concepts are no longer relevent (e.g. NSCell)
  - Many components are hard to customize/unoptimized (e.g. NSButton, NSCollectionView)
  - Forced to update your app when AppKit updates

Catalyst

(An iPad app adapted to AppKit.)

(macOS/iPadOS)

- Pros
  - Can share most of the code with the iPad app
  - Can be (has to be) tweaked to feel more like an AppKit app
- Cons
  - macOS 10.15+ only
  - Bugs in the framework
  - Forced to update your app when Catalyst updates

iOS/iPadOS app running on Apple Silicon

(As the title says.)

(macOS/iPadOS/iOS)

- Pros
  - "With a click of a button" (it's actually an opt-out-only feature in App Store)
- Cons
  - macOS 11+ only
  - Apple Silicon Macs only
  - Touch UX sometimes doesn't translate well to mouse+keyboard
  - Forced to update your app when Apple updates the UIKit

SwiftUI

(The future awaits.)

(macOS/iPadOS/iOS/tvOS/watchOS)

- Pros
  - Designed to work on all Apple OSes from the start
  - Modern API design
- Cons
  - Swift only
    - Limits app core to Obj-C/Swift/C-ABI
    - E.g. A C++ app core would be challenging
  - macOS 10.15+ only
  - Still in development (ready for production use?)

You are about to depart from the realm of Apple.

(Abandon Apple look and feel ye who enter here.)

Qt.Quick

(Not the best choice, but the only choice.)

(macOS/Windows/Linux(KDE/GNOME))

- Pros
  - Cross-platform
  - Fully GPU accelerated rendering (Metal/DX12/Vulkan/OpenGL)
  - QML: Declarative and reactive UI
  - Open Source
- Cons
  - Bloated
  - Not the best code quality
  - Some high level GUI components are poorly written (e.g. TreeView)

Flutter

(Google's SwiftUI/QML.)

(macOS/Windows/Linux/Android/iOS)

- Pros
  - Cross-platform
  - Declarative and reactive UI
  - Based on the Skia 2D graphics library
  - Official component registry
- Neutral
  - Main language: Dart
  - Emphasise heavelily on Material Design
- Cons
  - Desktop support is still in its early days

Immediate mode GUI

(When the G needs a UI.)

(macOS/Windows/Linux)

- Pros
  - Cross-platform
  - State-less UI
  - Simple to setup (if you already have a graphical context)
  - Highly performant
- Cons
  - Designed for very specific use cases
  - Look and feel customization options are limited

Custom GUI on top of an OS abstraction layer

(The OS is just a Window surface.)

(macOS/Windows/Linux)

- Pros
    - Cross-platform
    - You are in total control (make your own trade-offs)
    - Great for learning (if you can't build it, then you don't understand it)
- Cons
    - A lot of work
        - Graphics rendering
        - Layout
        - High level components
    - Even harder to generalize

Web

(All you need is a Web browser (or the core of it, says Electron).)

(Any platform with a modern Web browser)

- Pros
  - The app can be used without installation
- Cons
  - Built on top of extremly complex systems
    - A web browser is an extreamly generalised UI toolkit
  - Requires internet access

WebAssembly and WebGL

(The Web browser is just an ArrayBuffer and a WebGL surface.)

(Any platform with a modern Web browser)

- Pros
    - Usually an extra flavour of the native app
- Cons
    - New techonologies, may be unstable

Honourable Mention: CLI and TUI

(GUIs are for newbie.)

(macOS/Windows/Linux)

Thank you!

Hope this will help you next time when you want to build a Mac app.