ChatGPT

macOS용 ZoomIt 유사 앱 개발 가이드

1. 개발 언어 및 프레임워크 선택

ZoomIt와 같은 고성능 화면 주석 앱을 개발하려면 **Swift 언어와 Cocoa 프레임워크(AppKit)**를 사용하는 것이 가장 권장됩니다. Swift는 macOS의 최신 API 접근에 유리하며 메모리 관리 등의 측면에서 안전성이 높습니다. 실제로 많은 macOS 개발자들이 "macOS용 간단한 유틸리티 프로그램을 만든다면 열 번 중 열 번은 C보다 Swift를 선택할 것"이라고 할 정도로 Swift를 선호합니다 [1]. Swift는 Objective-C보다 현대적인 언어이며, 특히 macOS 12 이후 도입된 새로운 프레임워크(예: ScreenCaptureKit 등)를 활용하기에 적합합니다. 또한 Swift는 C++에 필적하는 수준의 성능을 내면서도 더 간결하고 유지보수하기 쉽다는 평가를 받고 있습니다.

기존에 C++ 경험이 있다면, Swift와 Objective-C++를 연계하여 C++ 코드를 재사용할 수도 있습니다. 예를 들어, 성능-critical한 로직이나 재사용하려는 라이브러리가 C++로 구현되어 있다면, Objective-C++ 브리지 레이어를 만들어 Swift에서 호출하는 방식도 많이 활용됩니다 [2]. 이처럼 **Swift + (필요시) Objective-C++** 조합은 성능과 생산성을 모두 잡을 수 있는 접근입니다. SwiftUI와 AppKit 중에서는, **AppKit 기반**으로 시작할 것을 권장합니다. AppKit은 전통적인 Mac 데스크톱 UI 프레임워크로서, 사용자 지정 창(NSWindow) 제어나 저수준 이벤트 처리에 유연합니다. SwiftUI도 사용 가능하지만, 화면 오버레이처럼 세밀한 제어가 필요한 경우 **AppKit(NSWindow/UIResponder)**가 더 적합하며, SwiftUI를 쓰더라도 AppKit 윈도우를 직접 관리하는 혼합 방식이 필요합니다.

> **정리:** 단일 개발자로서 최신 macOS 기능과 성능 최적화를 모두 활용하려면 **Swift + AppKit** 조합이 이상적입니다. Swift는 빠르고 안전하며, Apple의 풍부한 문서와 커뮤니티 지원을 받을 수 있습니다. C++ 코드를 활용해야 한다면 Objective-C++ 브리지를 통해 통합할 수 있습니다. (참고: 대규모 C++ 기반 프로젝트에서도 Objective-C++로 래핑해 Swift에서 사용하는 패턴이 일반적입니다 [3].) 이러한 조합은 초기 개발은 무료 버전으로, 이후 유료 기능을 추가하는 전략에도 유연하게 대응할 수 있습니다.

2. 화면 캡처 및 오버레이 구현 기술

화면 캡처와 확대 기능은 macOS의 시스템 API를 통해 구현할 수 있습니다. 최신 macOS 환경에서는 **ScreenCaptureKit** 프레임워크를 사용하는 것이 권장됩니다. ScreenCaptureKit은 macOS 12.3부터 도입된 고성능 화면 캡처 API로, 실시간 화면 이미지 스트림을 얻을 수 있습니다 [4]. 이 방법을 사용하면 H.264/HEVC 인코딩된 프레임을 받을 수 있어 CPU 부하를 줄이고 App Sandbox에도 친화적입니다(샌드박스에서 허용되는 `com.apple.security.screen-capture` 엔토틀먼트를 사용) [4]. 과거 호환성을 위해 macOS 12 이전을 지원해야 한다면 `CGWindowListCreateImage`와 같은 CoreGraphics API로 전체 화면 스냅샷을 얻는 방법도 있습니다. 다만 이러한 레거시 API는 macOS 15부터 폐기 예정이며(ScreenCaptureKit으로 대체) [5], **화면 녹화 권한**이 없으면 바탕화면 이미지만 반환되는 제약이 있습니다. 따라서 대상 OS가 충분히 최신이라면 ScreenCaptureKit을 활용하고, 사용자의 **"화면 기록"(Screen Recording)** 권한을 획득하도록 구현하세요. 최초 실행 시 시스템이 자동으로 권한 승인 대화상자를 표시하며, 사용자가 허용하면 이후부터는 앱이 전체 화면 이미지를 캡처할 수 있게 됩니다 [6] [7]. (Xcode의 프로젝트 설정에서 "Screen Capture" 관련 Entitlement를 추가해야 하며, 사용자가 권한을 거부한 경우 이를 안내하는 로직도 필요합니다.)

화면을 **확대(줌)**하는 기능은 캡처한 이미지를 활용하여 구현합니다. 일반적인 접근은 현재 화면을 이미지로 포착하여 확대 표시하는 것입니다. 예를 들어 사용자가 특정 단축키를 누르면: **1)** `CGImage`로 전체 화면을 캡처하고, **2)** 새 창에 해당 이미지를 배경으로 띄운 뒤, **3)** 이미지의 일부를 스케일 업하여 보여줍니다. 확대 비율은 고정 배율(예: 2배, 4배)로 하거나, 마우스 휠 등의 입력으로 조절할 수 있습니다. 성능을 위해 `CGImage`를 `CALayer`나 Metal 텍스처에 올려 GPU에서 스케일링하는 방법을 사용할 수 있지만, 간단한 구현으로는 NSImageView/UIImageView에 `contentsGravity = .resizeAspectFill` 설정 후 프레임을 확대해 보여주는 방식도 충분합니다. **실시간 줌**

(사용자가 화면 움직임을 지속적으로 확대하여 보는 기능)이 필요하다면, `CGDisplayStream` 또는 ScreenCaptureKit의 프레임 스트림을 사용해 주기적으로 갱신되는 이미지를 확대 표시할 수 있습니다. 하지만 이러한 실시간 처리에서는 성능과 지연에 신경써야 하므로, 우선은 **일시 정지된 화면을 확대하여 주석**하는 기능으로 시작하는 것이 안전합니다.

화면 오버레이 및 주석 그리기는 투명 창과 사용자 입력 이벤트 처리로 구현합니다. 화면을 캡처한 후 **전체 화면 크기의 투명 오버레이 창(NSWindow)**을 생성하여 최상위에 배치합니다. 이 창은 프레임리스(borderless) 스타일로 만들고, **항상 다른 창보다 위에 있도록** 윈도우 레벨을 설정합니다. 예를 들어 `window.level = .statusBar`로 지정하고 `orderFrontRegardless()`를 호출하면 어떤 앱 위에서도 최상위로 표시됩니다[8]. 또한 `window.collectionBehavior = [.canJoinAllSpaces]`로 설정하여 **모든 데스크탑 Spaces에서 보이도록** 합니다[9]. 이렇게 하면 사용자가 Mission Control이나 전체화면 앱 전환을 하더라도 주석 오버레이가 지속적으로 표시됩니다.

오버레이 창은 투명 배경을 가지며, 그 위에 사용자의 **그리기와 텍스트 주석**을 표현하는 커스텀 뷰(NSView)를 올립니다. 이 뷰 클래스에서는 `mouseDown(_:)/mouseDragged(_:)/mouseUp(_)` 등을 오버라이드하여 사용자의 펜 입력(마우스 드래그)를 캡처하고, 그 궤적을 화면에 그립니다. 간단한 구현으로는 드래그 동안 좌표들을 배열에 저장하고 `draw(_:)` 메서드에서 NSBezierPath로 해당 좌표를 이은 선을 그릴 수 있습니다. 이렇게 하면 사용자가 마우스로 자유롭게 선을 그어 화면에 **라이브 주석**을 달 수 있습니다. 선의 색상이나 두께는 미리 정해두거나, 사용자 인터페이스를 통해 선택할 수 있게 추가 구현할 수도 있습니다.

텍스트 입력 주석은 일반적으로 특정 키 입력 시 텍스트 모드로 전환되어 구현됩니다. 예를 들어 ZoomIt(Windows 버전)의 경우, 주석 모드에서 키보드를 치면 화면 중앙에 텍스트를 입력할 수 있었는데, 비슷하게 사용자가 `T` 키를 누르면 텍스트 입력 상태로 전환되도록 만들 수 있습니다. 구현상으로는 오버레이 뷰에 투명한 NSTextField를 하나 두고, 해당 모드일 때 첫 마우스 클릭 위치에 텍스트 필드를 배치하여 사용자가 입력하도록 하는 방법이 있습니다. 사용자가 입력을 완료하고 Enter를 누르거나 다른 작업을 하면, 해당 텍스트를 렌더링(예: NSAttributedString의 `draw(in:)`으로 화상에 그리기)하고 텍스트 필드는 제거합니다. 이렇게 하면 **텍스트 주석이 최종 이미지의 일부로 그려져 남게** 됩니다.

타이머 기능은 프레젠테이션 중 휴식 시간 표시 등에 활용될 수 있는 유용한 부가 기능입니다. ZoomIt의 타이머처럼, 특정 단축키(예: ⌘+Shift+T 등)를 누르면 화면에 커다란 숫자 타이머를 표시하도록 구현할 수 있습니다. 이때 화면을 어둡게 덮는 오버레이를 띄운 후, 남은 시간을 대형 텍스트로 중앙에 그려주면 됩니다. NSTimer(또는 GCD의 DispatchSourceTimer)를 사용해 1초 주기로 감소하는 **카운트다운을 업데이트**하고, `NSView`의 `setNeedsDisplay`를 호출하여 매 초 숫자가 갱신되도록 합니다. 또는 NSTextField/UILabel로 큰 폰트의 라벨을 만들어 1초마다 텍스트를 갱신하는 방식도 가능합니다. 타이머 완료 시 알림음 재생이나 화면 색 변화 같은 피드백을 줄 수 있으며, 사용자가 ESC를 누르면 타이머를 취소하고 주석 모드를 종료하도록 처리해야 합니다.

마지막으로 **전역 단축키 처리**입니다. 프레젠테이션 도중 언제든지 주석 모드로 진입하려면 글로벌 핫키를 등록해야 합니다. 샌드박스 환경의 Mac 앱에서는 Carbon API인 `RegisterEventHotKey`를 이용해 전역 단축키를 등록하는 방법이 일반적으로 쓰입니다[10]. 이 API를 통해 앱이 백그라운드에 있어도 특정 키 조합을 누르면 지정한 콜백이 실행되어, 곧바로 화면 캡처 및 오버레이 표시 루틴을 시작할 수 있습니다. 예를 들어 "Ctrl+1"을 줌 모드 토글 키로 지정하는 식입니다. `RegisterEventHotKey`는 샌드박스에서도 동작하며, sandboxed 앱이 전역 단축키를 사용할 수 있는 **유일한 공식 API**입니다[10]. (참고로 Shift+Option 같이 일부 조합은 macOS 15에서 동작 문제가 발견되어 Apple이 수정 중이므로, 일반적인 조합을 사용하는 것이 좋습니다.) 전역 단축키는 가급적 다른 시스템 또는 앱 단축키와 충돌하지 않도록 선택해야 하며, 사용자에게 단축키를 커스터마이즈할 수 있는 UI를 제공하면 더욱 좋습니다.

> **요약 구현 흐름:** 사용자가 설정한 전역 단축키 입력 → 앱이 현재 화면을 캡처 (ScreenCaptureKit 활용) → 전체화면 투명 오버레이 창 생성 (모든 Spaces에 표시) → 캡처 이미지를 확대 표시 및 주석용 커스텀 뷰 활성화 → 사용자의 펜/텍스트 입력을 실시간으로 그림 → (옵션) 타이머 모드 전환 시 화면 중앙에 카운트다운 표시 → 사용자가 ESC 또는 특정 키 입력 시 오버레이 창 닫고 원래 화면으로 복귀.

이 일련의 사이클에서 **macOS의 그래픽/이벤트 API(NSWindow, NSView, CoreGraphics 등)**를 적극 활용하고, 가능한 GPU 가속을 활용하여 부드러운 확대와 필기 선描画를 구현합니다.

3. App Store 배포 절차와 무료→유료 전환 전략

개발된 앱을 **Mac App Store에 배포**하려면 몇 가지 절차를 따라야 합니다. 아래는 **배포 단계 요약**입니다:

1. **애플 개발자 계정 가입:** 유료 Apple Developer Program에 가입하고(macOS 및 iOS 배포를 위해 연 $99), App Store에 앱을 올릴 자격을 획득합니다. 개발자 계정으로 Apple의 개발자 인증서와 Provisioning Profile을 발급받아야 합니다.

2. **프로젝트 설정 및 코드 서명:** Xcode에서 배포 대상(Target)을 Mac App Store로 설정하고, “Automatically manage signing”을 활성화하여 **Mac App Store 배포용 인증서(Apple Distribution)**로 코드 서명하도록 설정합니다. 이때 App Sandbox를 활성화하고 필요한 Entitlement(`com.apple.security.screen-capture` 등)을 추가해야 합니다 (자세한 보안 사항은 아래 4번 섹션 참고). 코드 서명이 올바르게 되어 있어야 앱이 Gatekeeper를 통과합니다 (개발 중에는 개발자 코드 서명, 배포 시에는 배포용 코드 서명).

3. **앱 빌드 및 아카이브:** Xcode에서 **Archive**를 수행하여 배포용 .app 파일을 생성합니다. 아카이브된 빌드를 Xcode Organizer에서 **“Validate App”**으로 점검한 뒤 **“Upload to App Store”**를 통해 Apple App Store Connect에 업로드합니다. (이 과정에서 자동으로 bitcode 처리 및 필수적인 **notarization** 검증이 이뤄집니다. Mac App Store에 올리는 앱은 따로 공증 요청을 할 필요가 없으며, 업로드 시 Apple의 자동 검사에 통과하면 곧바로 TestFlight 또는 출시 준비 상태가 됩니다 [11].)

4. **App Store Connect 메타데이터 작성:** App Store에 표시될 **앱 정보(메타데이터)**를 작성합니다. 앱 이름, 설명, 카테고리, 키워드, 지원 OS 버전, 스크린샷, 아이콘 등을 등록해야 합니다. 특히 **스크린샷**은 주석 기능, 확대 기능 등을 잘 보여줄 수 있도록 캡처해서 올리세요. 또한 화면 녹화 권한 사용 이유를 설명하는 문구를 Review Notes나 사용자 설명에 명시하면 심사에 도움이 될 수 있습니다. (예: "본 앱은 프레젠테이션 시 화면에 주석을 그리기 위해 시스템의 화면 기록 권한을 사용합니다.")

5. **심사 및 출시:** 메타데이터와 빌드가 준비되면 **App Store 심사(Review)**에 제출합니다. Apple 심사 팀이 앱을 검토하며, 주로 샌드박스 준수 여부, 권한 요청의 합리성, UI/콘텐츠의 App Store 가이드라인 적합성 등을 확인합니다. 화면에 임의의 오버레이를 그리는 앱이라도, 명확한 사용자 기능(프레젠테이션 보조)으로서 권한을 요청하고 있으면 승인에 문제없을 것입니다. 심사를 통과하면 App Store에 앱이 **출시(Release)**되며, 처음에는 **무료(free)**로 배포됩니다.

6. **무료에서 유료로 전환:** 초기 사용자 확보나 테스트를 위해 무료로 배포한 후, 추후 **유료 앱으로 전환**할 수 있습니다. App Store Connect의 **Pricing** 섹션에서 가격을 $0.00 (무료)에서 원하는 유료 가격 티어로 변경할 수 있습니다. Apple은 개발자가 언제든지 앱의 가격을 조정하는 것을 허용하며, 많은 앱들이 무료→유료 가격 변경이나 한시적 무료 행사를 진행합니다 [12] [13]. 다만 **주의사항**: 기존에 무료로 다운로드한 사용자들은 앱 가격이 나중에 유료로 바뀌더라도 추가 비용 없이 계속 앱 업데이트를 받을 수 있습니다 [13]. 가격 변경은 **신규 다운로드**에만 적용되므로, 한번 무료로 배포된 앱은 기존 유저에게는 영구히 무료인 셈입니다. 따라서 수익화를 위해서는 새로운 SKU로 앱을 출시하거나 인앱 구매(In-App Purchase)를 도입하는 방법을 고려해야 합니다. 일반적으로는 **프리미엄 기능을 인앱 구매로 잠금 해제**하는 형태의 Freemium 모델이 권장됩니다 [14]. 예를 들어, 기본적인 주석 기능은 무료로 제공하고, 타이머 커스터마이즈나 고급 펜 도구 등을 유료 **인앱 구매**로 판매할 수 있습니다. 이렇게 하면 기존 무료 사용자도 앱 내에서 유료 기능을 구매하도록 유도할 수 있어 수익화에 효과적입니다.

7. **유료 전환 옵션 구현:** 만약 추후 앱 자체를 유료로 전환하기보다는, 일정 기간 후 **신규 사용자는 유료로 받게 하고 싶다면**, 앞서 언급한 가격 변경 기능을 활용하면 됩니다. 가격을 유료로 올릴 시 별도의 앱 재심사가 필요하지는 않지만(가격 변경은 개발자 재량), 업데이트와 함께 공지하는 것이 좋습니다. 다른 대안으로 **새로운 유료**

버전 앱을 별도 출시(Pro 버전 등)하는 방법도 있습니다. 그러나 이 경우 기존 사용자 데이터 이관, 별도 앱 관리 등의 부담이 있으므로, **동일 앱에서 IAP로 업그레이드**하는 방안이 개발 및 사용자 경험 측면에서 선호됩니다.

8. **버전 업데이트 및 코드 서명 유지:** 무료든 유료든, 업데이트를 배포할 때마다 Xcode에서 아카이브하여 **새 버전을 App Store Connect에 올리고 심사** 받아야 합니다. 이때도 코드 서명과 샌드박스 설정이 일관되게 유지되어야 하며, 번들 ID를 변경하지 않는 한 기존 사용자들은 계속 업데이트를 받을 수 있습니다.

   팁: 개인 개발자로서 처음 앱을 배포한다면, **TestFlight**를 통해 베타 테스트를 진행해 보는 것도 권장합니다. App Store Connect에서 TestFlight 베타그룹을 만들어 지인이나 커뮤니티 사용자들에게 미리 앱을 사용해보게 하고 피드백을 받을 수 있습니다. 특히 화면 주석 앱은 다양한 환경에서 테스트가 중요하므로, macOS 권한 설정 단계나 단축키 충돌 여부 등을 베타 테스터로부터 점검받으면 정식 출시 전에 품질을 높일 수 있습니다.

4. macOS 보안 및 앱 승인 관련 고려사항 (코드 서명, 샌드박스 등)

macOS 앱을 배포할 때는 **코드 서명**과 **샌드박싱(App Sandbox)**, 그리고 **앱 공증/인증(Notarization)** 등 보안 요건을 충족해야 합니다.

- **코드 서명(Code Signing):** 모든 Mac 앱은 개발자의 인증서로 서명되어야 하며, 특히 App Store에 올리는 앱은 Apple이 발행한 **배포용 인증서(Apple Distribution)**로 서명되어야 합니다. 코드 서명이란 해당 앱이 개발자 본인이 만든 것임을 증명하고, 배포 후 무결성이 유지되었음을 보장하는 기술입니다 [15]. Xcode에서 자동 서명을 사용하면 적절한 인증서로 앱 번들과 그 내부의 실행 파일들(.app 내부의 바이너리, 프레임워크, XPC 서비스 등)이 모두 서명됩니다. 코드 서명이 제대로 되어야 **Gatekeeper**가 앱을 신뢰하여 사용자 시스템에서 정상 실행을 허용합니다. App Store를 통해 배포하는 경우, Apple이 추가로 DRM 서명 및 암호화를 더한 후 배포하므로 개발자는 Xcode 설정만 올바르게 하면 됩니다. (만약 App Store 외부에 배포한다면 개발자 계정의 **Developer ID Application** 인증서로 서명하고 **Notarization(공증)**을 거쳐야 하지만, App Store 배포의 경우 Apple 심사 과정에서 유사한 자동 검증을 하므로 별도 공증 절차는 필요 없습니다 [11].)

- **앱 샌드박스(App Sandbox):** Mac App Store에 등록하는 모든 앱은 **샌드박스 활성화**가 필수입니다. 샌드박스는 앱이 자기 자신의 컨테이너 외부 자원에 함부로 접근하지 못하게 제한하여 시스템 보안을 높이는 기능입니다 [16]. Xcode의 Signing & Capabilities 탭에서 “App Sandbox”를 켜면 자동으로 entitlements 파일에 `"com.apple.security.app-sandbox"` 키가 추가되고 빌드 시 샌드박스가 적용됩니다 [17]. 샌드박스 환경에서는 파일 접근, 네트워크, 하드웨어 접근 등이 기본적으로 차단되며, 필요한 항목만 **Entitlement(권한)** 형태로 허용할 수 있습니다 [18]. 예를 들어 화면 캡처를 위해서는 앞서 언급한 `"com.apple.security.screen-capture"` 권한이 필요하고, 만약 마이크 소리를 함께 녹음한다면 `"com.apple.security.device.audio-input"` 권한, 네트워크 통신이 필요하면 `"com.apple.security.network.client"` 등의 권한을 추가로 부여해야 합니다. 이번 앱의 기능 범위에서는 **Screen Capture 권한**이 주요하며, 그 외에 특별한 파일 접근이나 시스템 제어가 없도록 설계하는 것이 좋습니다. (만약 전역 단축키로 **Accessibility API**를 사용하게 된다면 `"com.apple.security.accessibility"` 권한이나 사용자의 손쉬운 사용 권한 허용이 필요할 수 있지만, `RegisterEventHotKey`는 그런 권한 없이 동작하므로 피하는 편이 낫습니다.)

- **사용자 프라이버시 권한:** macOS은 민감한 기능에 대해 사용자 **프라이버시 허가**를 요구합니다. 이 앱의 경우 **화면 기록(Screen Recording)** 권한이 이에 해당됩니다. 앱이 ScreenCaptureKit이나 CGWindowList API로 다른 윈도우의 픽셀을 캡처하려 할 때 시스템이 한 번 사용자에게 "{앱 이름}에게 화면을 기록할 권한을 허용하시겠습니까?"라는 대화상자를 띄웁니다. 사용자가 이를 승인해야만 화면 캡처가 정상 동작하므로, 권한 요청 시 적절한 안내를 제공하세요. Info.plist에 카메라나 마이크처럼 사용자 설명 문자열을 넣는 항목은 **화면 기록의 경우 존재하지 않으므로** 시스템 기본 문구로 표시됩니다 [19]. 따라서 초기 앱 실행 시 튜토리얼 등에 "화면 녹화 권한을 허용해주세요"라는 가이드를 포함하면 좋습니다. (참고로 한 번 권한을 거부한 사용자는 시스템

환경설정 > 보안 및 개인정보 보호 > 화면 기록에서 수동으로 앱을 활성화해야 하므로, 이에 대한 안내도 필요할 수 있습니다.)

- **노터라이제이션(Notarization)과 인증:** 앞서 설명한 대로 Mac App Store에 올리는 앱은 별도의 notarization 요청이 필요 없지만, **개발 단계에서 테스트용으로 앱을 직접 배포하거나** App Store 외 채널로 배포할 가능성이 있다면 notarization 절차를 알아두어야 합니다. notarization은 Apple 서버에서 해당 앱에 악성코드가 없는지 자동 검사 후 **공증 티켓**을 발급해주는 과정입니다 [20]. Developer ID로 서명한 앱을 배포할 때는 이 공증을 거쳐야 macOS Catalina(10.15) 이상에서 정상 실행됩니다. 일반 사용자는 App Store 버전을 설치하므로 이 과정은 신경 쓸 필요 없지만, TestFlight 외에 직접 .app이나 .dmg로 배포하는 상황이 생긴다면 Xcode나 `xcrun altool` 명령을 통해 notarize를 해야 합니다.

- **기타 보안 고려사항:** 앱 번들 내에 만약 자체 업데이터나 확장 등이 있다면, 샌드박스 규칙을 준수해야 합니다 (예: Sparkle 같은 자체 업데이트는 Mac App Store에서는 금지). 그리고 앱과 함께 번들되는 프레임워크, dylib 등도 모두 동일한 인증서로 서명되어야 합니다. Xcode 빌드를 활용하면 이 부분은 자동 처리됩니다. **네트워크 사용** 계획이 있다면 ATS(App Transport Security) 규정도 준수해야 하며, 필요 시 Info.plist에 예외를 설정합니다. 마지막으로, **앱 아이덴티티(번들 ID)**가 한번 정해지면 변경하면 안 됩니다. 번들 ID는 코드 서명과 연계되어 App Store에 등록되므로, 추후 SKU 변경 없이 가격 전환이나 업그레이드를 위해서는 번들 ID를 유지하면서 동일 앱으로 업데이트하는 것이 중요합니다.

요약하면, **Mac App Store용 앱 개발에서는 보안 요건을 철저히 준수**해야 합니다. Xcode의 자동 서명과 샌드박스 설정을 활용하고, 필요한 권한만 최소한으로 부여하세요. 이러한 절차를 따르면 Apple의 앱 심사 및 사용자 시스템의 Gatekeeper 통과에 문제가 없을 것입니다.

5. AI 에이전트 활용 및 프롬프트 엔지니어링 전략

혼자 앱을 개발할 때 **AI 코딩 도우미**(예: ChatGPT 등)를 적절히 활용하면 생산성을 크게 높일 수 있습니다. 효과적으로 AI의 도움을 받기 위해서는 **프롬프트 엔지니어링** 기법을 사용하여 원하는 정보를 정확히 얻어내는 것이 중요합니다. 프롬프트 엔지니어링이란 “AI 모델에게 특정 상황과 요구사항을 잘 지시해 기대하는 결과물을 얻도록 프롬프트(질문)을 설계하고 최적화하는 기술”을 뜻합니다 [21]. 아래에 개발 업무에 AI를 활용할 때 유용한 전략들을 정리합니다:

- **질문을 명확하고 구체적으로 작성:** 원하는 답변의 형태와 내용을 프롬프트에 분명히 명시하세요 [22]. 예를 들어 "macOS에서 NSWindow로 전체화면 투명 창을 만드는 Swift 코드 예시를 보여줘"처럼 구체적인 요청이 막연한 질문보다 좋은 답을 얻습니다. 요구 사항(예: "타이머 기능 구현 예시 코드를 Swift 5.7 문법으로 보여주세요")이나 출력 형식(예: 코드, 목록, 단계별 설명 등)을 함께 알려주면 AI가 방향을 잡기 쉽습니다 [22].

- **맥락과 배경 정보 제공:** AI에게 현재 직면한 문제의 맥락을 충분히 알려주는 것이 중요합니다 [23]. 예를 들어 오류 해결을 물을 때는 에러 메시지와 관련 코드 조각을 함께 제공하고, "Xcode 14, macOS Ventura 환경에서 발생한 문제" 등 환경 정보도 덧붙입니다. AI는 주어진 정보를 토대로 답변하므로, **맥락을 자세히 줄수록** 정확한 해결책이나 코드를 제시할 확률이 높아집니다.

- **단계를 나누어 질문:** 복잡한 문제는 한 번에 모두 물어보기보다 **단계적으로 쪼개서 질문**하는 것이 효과적입니다. 예를 들어 화면 캡처 → 오버레이 생성 → 펜 그리기 기능 구현을 한꺼번에 묻기보다, 먼저 "Swift에서 화면 캡처 API 사용 방법?"을 묻고, 답을 얻은 뒤 이어서 "캡처한 이미지를 화면에 표시하는 방법?"을 묻는 식입니다. 이렇게 하면 AI가 각 부분에 집중하여 상세한 답변을 제공하며, 개발자는 부분적 이해를 쌓아 최종적으로 전체 기능을 완성할 수 있습니다.

- **예시와 기대 출력 제시:** 원하는 답변 형식이 있다면 예시를 보여주는 것도 좋습니다. 예를 들어 "코드 예시를 보여줘 (`swift ...`로 감싸서)" 또는 "...표 형태로 정리해줘"처럼 요구하거나, 이전에 받은 부실한 답변이 있다면 "위 답변에서는 X 부분이 부족한데, Y를 포함하여 다시 알려줘"라고 피드백을 주면 AI가 답변을 개선합니

다. 한 번에 완벽한 답변이 나오지 않더라도, **반복 피드백을 통해 답을 정제(refinement)**하는 대화를 이어나갈 수 있습니다 [24].

- **코드 생성 및 검증:** AI에게 코드를 생성하게 할 때는, **짧은 모듈별 코드**를 요청하는 것이 좋습니다. 예를 들어 "NSWindow 레벨을 설정하는 Swift 코드 예시?" 처럼 물어보면 필요한 부분만 정확히 얻기 쉽습니다. 이렇게 생성된 코드는 곧바로 프로젝트에 적용하기 전에 꼭 **검증 및 테스트**하세요. AI가 제공한 코드는 문법적으로 그럴듯해도 실제 환경에 맞지 않거나, 최신 API 변화로 맞지 않는 경우가 있습니다. 컴파일 에러가 나거나 예상과 다르게 동작하면, 그 오류 메시지나 문제 상황을 다시 AI에게 설명하며 디버깅 도움을 받을 수도 있습니다 (예: "이 코드를 컴파일하니 deprecated 경고가 발생하는데, 최신 방법으로 수정해줘."). AI는 그런 피드백을 토대로 대안을 제시해 줄 수 있습니다. 항상 **공식 문서**와 AI 답변을 교차검증하는 습관을 가지면 신뢰성을 높일 수 있습니다.

- **지식 학습 및 아이디어 브레인스토밍:** 혼자 개발하면서 막힐 때, AI를 **튜터처럼 활용**할 수도 있습니다. 예를 들어 "AppKit에서 커스텀 커서를 지정하는 방법을 설명해줘" 또는 "화면 주석 앱에서 메모리 누수를 방지하려면 어떤 패턴이 좋을까?" 등의 물음으로 개념을 학습할 수 있습니다. 또한 여러 가지 구현 방안이 고민될 때 아이디어를 나열해 달라고 할 수도 있습니다. AI는 방대한 지식을 바탕으로 다양한 접근법을 제시할 수 있으므로, 당신은 그중 현실적으로 맞는 것을 선택하면 됩니다. (**주의:** AI의 설명이나 코드가 항상 100% 정확한 것은 아니므로, 최종 결정은 개발자가 해야 합니다.)

- **프롬프트 실험과 개선:** 프롬프트 엔지니어링 자체도 창의적인 과정입니다. 원하는 결과를 얻지 못했다면, 프롬프트를 약간 바꾸어 재시도해보세요 [25] [26]. 같은 질문이라도 표현을 달리하거나 세부 조건을 추가/제거하면 응답 품질이 달라질 수 있습니다. 예를 들어 답변이 너무 일반적이면 "좀 더 상세히 설명해줘", 코드가 장황하면 "간결한 코드로 보여줘" 등 추가 지시를 곁들입니다. 또는 "이 역할을 네가 Mac 개발 전문 엔지니어라고 생각하고 대답해줘."처럼 AI에게 특정 역할을 부여하면 전문적인 어조의 답변을 얻는 데 도움이 됩니다. 이러한 **프롬프트 튜닝 과정**을 통해 AI의 출력을 점점 내가 원하는 방향으로 맞춰갈 수 있습니다 [24].

- **AI 도구의 다양성 활용:** 하나의 AI 모델뿐만 아니라, 경우에 따라 GitHub Copilot처럼 IDE에 통합된 보조 도구나, GPT-4 등의 더 성능 좋은 모델을 활용할 수도 있습니다. Copilot은 코딩 중 자동완성 제안을 주므로 루틴한 코드를 빨리 작성하게 해주고, ChatGPT 같은 대화형 모델은 개념 설명이나 디버깅에 유용합니다. 또한 필요한 경우 **공식 문서 검색 결과를 함께 검토**하도록 AI에게 자료를 주면서 질문할 수도 있습니다 (예: Apple Developer 문서 일부를 보여주며 이해가 안 되는 부분을 물어보기). 이렇듯 AI를 단순 질의응답뿐만 아니라 개발 전 과정에서 **다양한 역할**로 활용하면 혼자서도 풍부한 아이디어와 도움을 얻을 수 있습니다.

마지막으로, 프롬프트 엔지니어링의 핵심은 **개발자가 원하는 것을 정확히 알고, 이를 AI에게 효과적으로 전달하는 것**입니다. 초기에는 시간이 걸릴 수 있으나, 반복하다 보면 어떤 표현이 잘 통하는지 감을 잡게 되고 점차 **AI와 협업하는 능력**이 향상됩니다. 이는 혼자 개발할 때 생기는 막연함을 줄이고, 마치 페어 프로그래밍을 하듯 아이디어를 주고받는 든든한 조력자가 될 것입니다. 필요한 정보를 빠르게 얻고 시행착오를 줄여, 궁극적으로 개발 효율과 결과물의 완성도를 높일 수 있을 것입니다.

참고자료: 본 가이드에서는 Apple 공식 문서와 개발자 커뮤니티의 조언을 바탕으로 ZoomIt 유사 기능의 macOS 구현 전략을 다루었습니다. Swift 언어 선택의 이점 [1], C++ 연계 방안 [3], ScreenCaptureKit 및 샌드박스 권한 [4], 오버레이 윈도우 구현 기법 [8] [9], App Store 가격 정책 [13] [14] 등의 인용을 통해 신뢰성을 더했습니다. 이 정보를 바탕으로 효율적인 개발과 배포 준비를 하시기 바라며, 성공적인 macOS 앱 출시를 기원합니다.

---

[1] Thoughts on Swift and Objective-C (2022) | Hacker News
https://news.ycombinator.com/item?id=35914330

2 3 For industry iOS jobs, how much do you write Swift vs Obj-C? : r/iOSProgramming
https://www.reddit.com/r/iOSProgramming/comments/v7514i/for_industry_ios_jobs_how_much_do_you_write_swift/

4 App Development: ScreenCaptureKit or DriverKit Virtual Display? (for a Mac app with XREAL glasses) : r/Xreal
https://www.reddit.com/r/Xreal/comments/1kd6lyd/app_development_screencapturekit_or_driverkit/

5 CGWindowListCreateImage | Apple Developer Documentation
https://developer.apple.com/documentation/coregraphics/cgwindowlistcreateimage(_:_:_:_:)?language=objc

6 7 macos - Capture screen with CGWindowListCreateImage just return the wallpaper - Stack Overflow
https://stackoverflow.com/questions/58997582/capture-screen-with-cgwindowlistcreateimage-just-return-the-wallpaper

8 9 Create a Translucent Overlay Window on MacOS in Swift | by Adonis Gaitatzis | Apr, 2025 | Medium
https://gaitatzis.medium.com/create-a-translucent-overlay-window-on-macos-in-swift-67d5e000ce90

10 PSA: macOS 15 breaks Option-key hotkeys : r/macapps - Reddit
https://www.reddit.com/r/macapps/comments/1fjpiiw/psa_macos_15_breaks_optionkey_hotkeys/

11 15 20 Signing a macOS app | Electron Forge
https://www.electronforge.io/guides/code-signing/code-signing-macos

12 13 14 ios appstore - Pricing an app in the App Store - is free followed by paid strategy allowed? - Ask Different
https://apple.stackexchange.com/questions/247663/pricing-an-app-in-the-app-store-is-free-followed-by-paid-strategy-allowed

16 Accessing files from the macOS App Sandbox - Apple Developer
https://developer.apple.com/documentation/security/accessing-files-from-the-macos-app-sandbox

17 Configuring the macOS App Sandbox - Apple Developer
https://developer.apple.com/documentation/xcode/configuring-the-macos-app-sandbox

18 What's new in privacy | Documentation - WWDC Notes
https://wwdcnotes.com/documentation/wwdcnotes/wwdc24-10123-whats-new-in-privacy/

19 Is there an Info.plist key "Privacy - ... Usage Description" for screen recording in macOS? - Stack Overflow
https://stackoverflow.com/questions/62641652/is-there-an-info-plist-key-privacy-usage-description-for-screen-recordin

21 24 [AI 코딩] 개발자를 위한 프롬프트 엔지니어링 시작하기
https://goldenrabbit.co.kr/2024/05/07/ai-%EC%BD%94%EB%94%A9-%EA%B0%9C%EB%B0%9C%EC%9E%90%EB%A5%BC-%EC%9C%84%ED%95%9C-%ED%94%84%EB%A1%AC%ED%94%84%ED%8A%B8-%EC%97%94%EC%A7%80%EB%8B%88%EC%96%B4%EB%A7%81-%EC%8B%9C%EC%9E%91%ED%95%98%EA%B8%B0/

22 23 25 26 프롬프트 엔지니어링 팁
https://velog.io/@leesa1125/%ED%94%84%EB%A1%AC%ED%94%84%ED%8A%B8-%EC%97%94%EC%A7%80%EB%8B%88%EC%96%B4%EB%A7%81-%ED%8C%81