App design and UI

Choose a programming approach to build your app, create your app's interface, and implement the fundamental behaviors that your app requires.

At the start of every new project, you need to choose an app-builder technology to use for your initial code. App-builder technologies define the programming approach you take for your app's interface, event-handling code, and other behaviors. You can choose one of these programming approaches for your app, or combine the approaches.

Each platform defines the overall look for views and controls, and your app-builder technology determines how you create and manage your interface. Build your interface with standard views and controls, a mixture of standard and custom views, or entirely custom content.

SwiftUI apps

SwiftUI is the best option when you're learning to program for Apple platforms, or when you want to create a new app. With SwiftUI, you build your app's interface and content using a declarative programming model. With this model, you describe the behaviors and appearance you want, and SwiftUI creates and manages the interface for you. Changes are data driven, so when you update variables that affect the state of a view, SwiftUI refreshes your interface for you.

Use SwiftUI to build apps for iOS, iPadOS, macOS, tvOS, visionOS, and watchOS and the Swift programming language.

- Build apps and widgets using a declarative programming model and data-driven changes.
- Build your interface, and incorporate features like custom drawing and text editing.
- See live previews of your interface as you write the code for your views.
- Incorporate existing UIKit or AppKit views and view controllers into your interface.

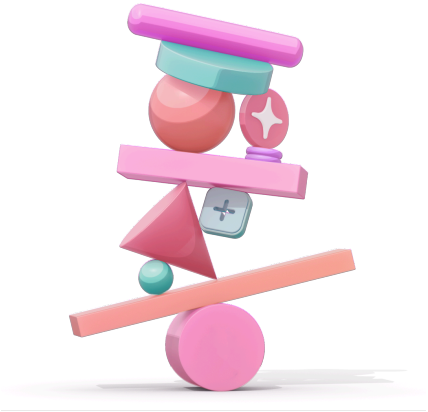
To learn more, read SwiftUI apps.

UIKit and AppKit apps

UIKit and AppKit offer a more traditional, object-oriented approach to building apps. These frameworks provide a library of objects that you assemble and customize to achieve the behavior you want. For example, you assemble your interface from standard and custom views and place the logic for managing view interactions in custom controller objects. Each object manages its own behavior, and your custom code defines the overall behavior of your app.

Use UIKit to build apps for iOS, iPadOS, tvOS, visionOS, and Mac Catalyst. Use AppKit to build apps for macOS. Build your app using either Swift or the Objective-C programming language.

- Build apps using a library of objects and a model-view-controller architecture.
- Build your interface, and incorporate features like custom drawing and rich-text editing.
- Assemble your app's view hierarchies using Xcode's visual editor.
- Adopt SwiftUI views incrementally in your view hierarchies.

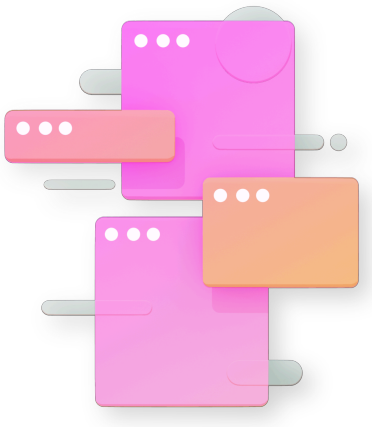


To learn more, read UIKit and AppKit apps.

Interface fundamentals

No matter which app-builder technology you choose, most of the components you use to build your interface are the same. Before you build your interface, learn about the different components available to you, and learn how different platforms use those components. You can also learn about other technologies that impact the design of your interface and how you display content.

- Learn about the windows, views, and other visual elements available to you.
- Explore the design approaches for each platform, and learn how to make your app stand out.
- Manage app-related assets, and learn how to load them locally or from a remote server.
- Support common features like internationalization, accessibility, undo and redo, and the pasteboard.



To learn more, read Interface fundamentals.

Liquid Glass

Interfaces across Apple platforms feature a new dynamic material called Liquid Glass, which combines the optical properties of glass with a sense of fluidity. Learn how to leverage Liquid Glass to make sure your interface looks right at home on Apple platforms.

- Embrace the visual refresh for materials, controls, and app icons.
- Provide a universal navigation and search experience across platforms.
- Ensure your interface's organization and layout looks consistent with other apps and system experiences.
- Adopt best practices for windows, modals, menus, and toolbars.
- Test your app to provide a great experience across platforms.

To learn more, read Liquid Glass.