NavigationStack

The Navigator You've Been Waiting For

Muralidharan Kathiresan

Murali|

muralidharan kathiresan • 1st • Senior iOS Develo...

👩🏼‍💻 Senior 🍏 at Ballys

🇮🇳 Living in London 🇬🇧

✍🏽 swiftpublished.com

What’s in for today?

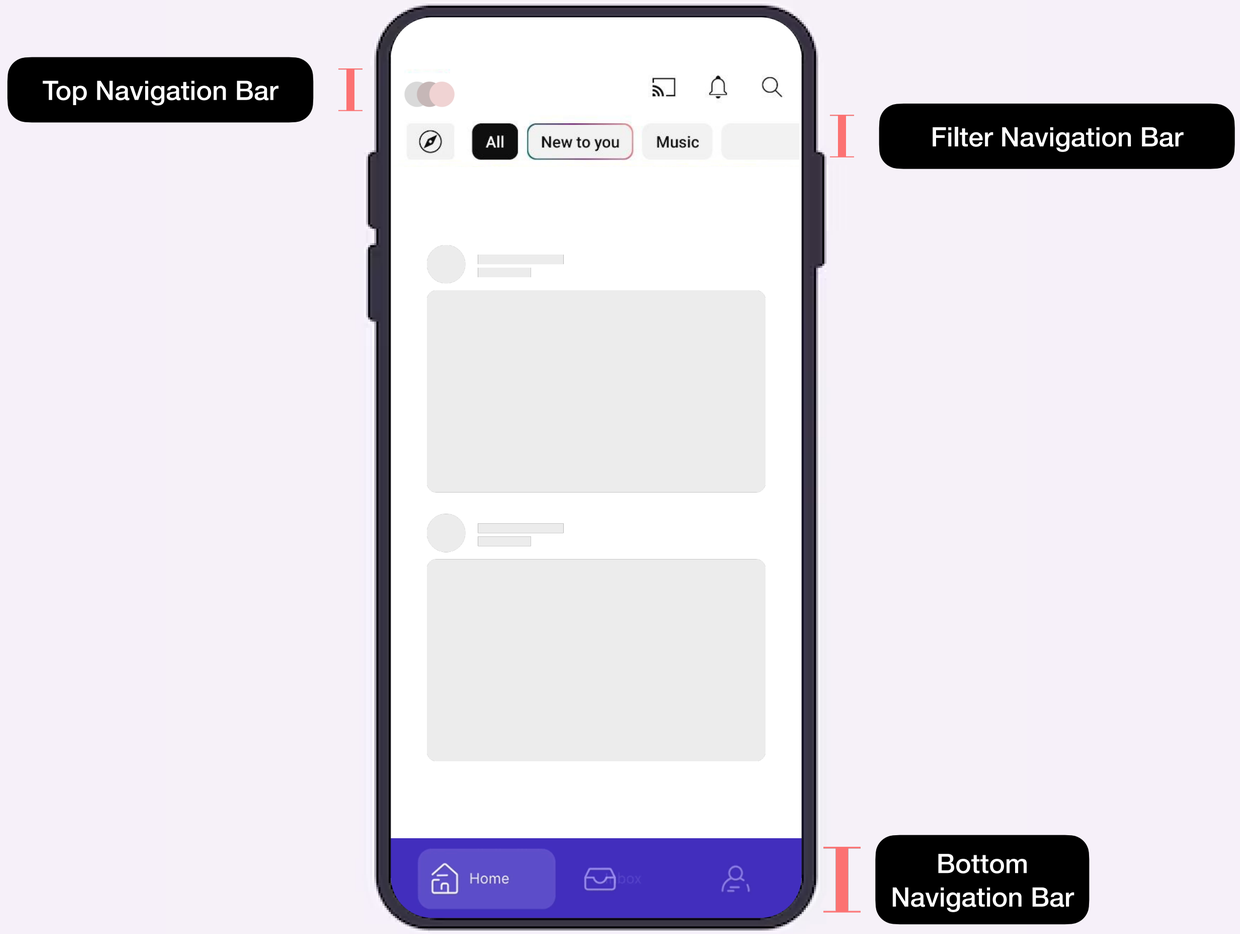
- What’s navigation
- NavigationView and its limitation
- How NavigationStack Becomes Our Saviour
- Overview of NavigationSplitView
- Best patterns for SwiftUI navigation
- Takeaways

What is Navigation?

Backbone of any app

The way your users will get from point A to point B

Refers to how users **move** through **different screens, views, or sections** within an app.



Why Navigation?

Enhancing Overall User Experience

Reduces confusion and enables to navigate users effortlessly

Facilitating Content Discovery

Guides users to various features and functionalities

Streamlining Task Completion

Allows users to complete tasks quickly and without unnecessary steps

Boosting Retention and Engagement

Prevents users from abandoning the app due to navigation struggles.

Timeline?

2007

UINavigation Controller

Introduced with the original iPhone, this hierarchical navigation stack is used for pushing and popping view controllers, commonly for drill-down navigation

2008

UITabBar Controller

Tab-based navigation switches between sections or features, with each tab representing a distinct view controller

2010

UISplitView Controller

Used on iPads for master-detail interfaces, featuring a split screen with a master list and a detail view. Common in mail apps and file explorers.

2019

Navigation View

SwiftUI's built-in navigation component. Hierarchical navigation similar to UINavigationController. Uses NavigationLink for pushing views

2021

Navigation Stack

Introduced in iOS 16.
A more flexible stack-based navigation system.
Allows dynamic addition and removal of views

2021

Navigation SplitView

Introduced in iOS 17.
Similar to UISplitView Controller. Supports master-detail interfaces on iPad

UINavigationController?

iOS 2.0+

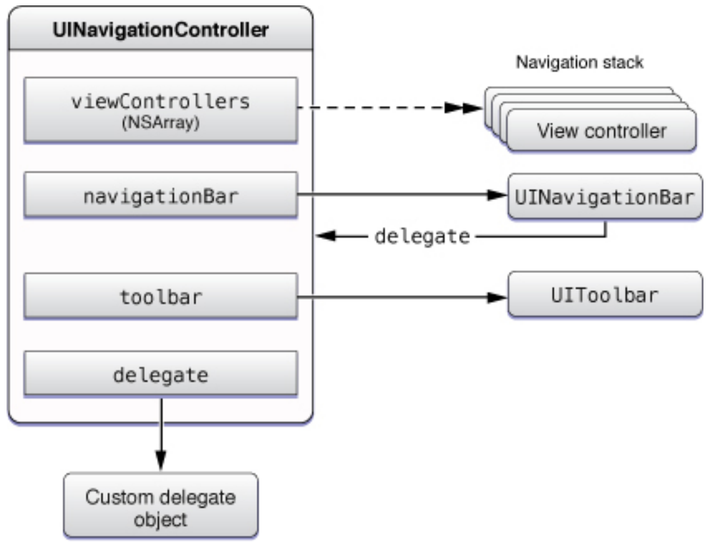
A container view controller

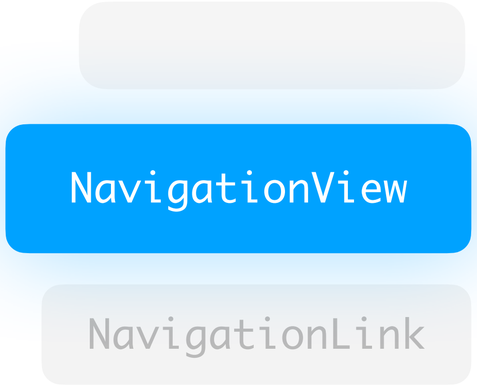
It defines a stack-based scheme for navigating hierarchical content

Use it when you have a stack of related view controllers that represent a navigation flow

Example: drill-down navigation

```
@MainActor
class UINavigationController : UIViewController
```





Introduced in iOS 13.0

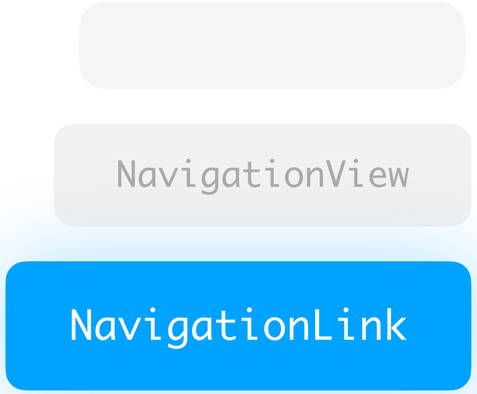
A container view

Presents a stack of views representing a visible path in a navigation hierarchy.

Includes Multiplatform support

Deprecated in 16.0

```
struct NavigationView<Content> where Content : View
```



A SwiftUI component that controls a navigation presentation

This modifier is used within NavigationView to create links that lead to other views within the navigation hierarchy.

Navigates to a destination view by selecting a NavigationLink

On iPadOS and macOS, the destination content appears in the next column.

Other platforms push a new view onto the stack

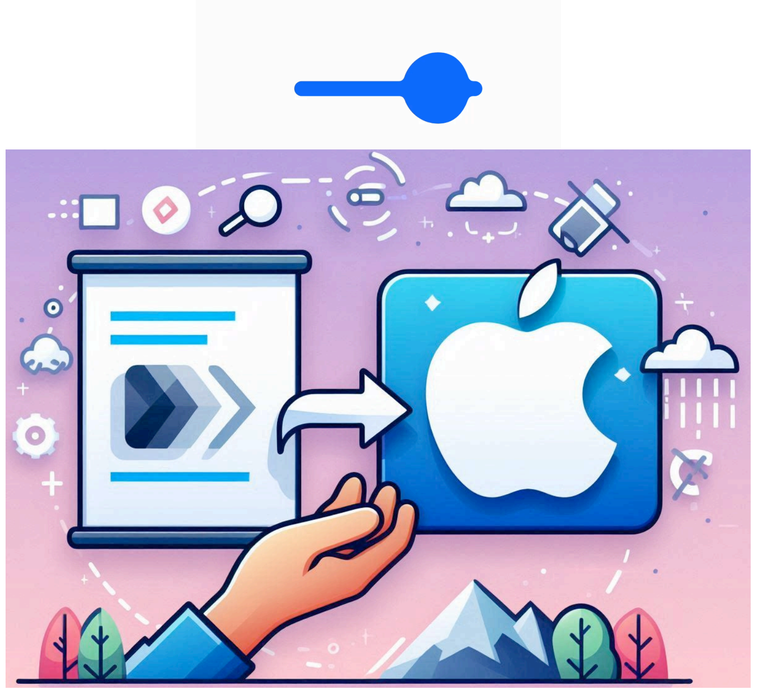
```
struct NavigationLink<Label, Destination> where
Label: View, Destination: noteView
        NavigationLink(note.title, destination:
NoteEditor(id: note.id))
    }
    Text("Select a Note")
}
```

Why NavigationView was the most hated API?

- Limited Customisation
- Inconsistent Behaviour Across Platforms
- Complex Patterns
- Hard State Restoration
- Performance Issues

```
NavigationView {
    List(model.notes) { note in
        NavigationLink(note.title, destination:
NoteEditor(id: note.id))
    }
    Text("Select a Note")
}
```

Switch to Xcode



NavigationStack

NavigationPath

Navigation Destination

Introduced with iOS 16 and macOS 13

A container view that has and displays a root view

Enables you to present additional views over the root view

Here, you can pop to root or pop one step back without passing the push reference of the previous view

Data/State-based

```
struct NavigationView<Content> where
Content : View
```

```
struct NavigationStack<Data, Root> where
Root : View
```

NavigationStack

NavigationPath

Navigation Destination

```
struct NavigationStack<Data, Root> where
Root : View
```

Introduced with iOS 16 and macOS 13

A container view that has and displays a root view

Enables you to present additional views over the root view

Here, you can pop to root or pop one step back without passing the push reference of the previous view

Data/State-based

```
NavigationView { // This is
deprecated.
    /* content */
}
.navigationViewStyle(.stack)
```

```
NavigationStack {
    /* content */
}
```

NavigationStack

NavigationPath

Navigation Destination

Type-erased list of data representing the content of a navigation stack.

Each element in the path represents a specific view in the navigation hierarchy.

Manages the state of a NavigationStack by initialising with a binding to a collection of data.

Conforms to Hashable, enabling efficient storage and equality checks.

Uses type erasure to handle a collection of heterogeneous elements.

```
@State private var path: [Item] = []
```



```
@State private var path = NavigationPath()
```

NavigationStack

NavigationPath

Navigation Destination

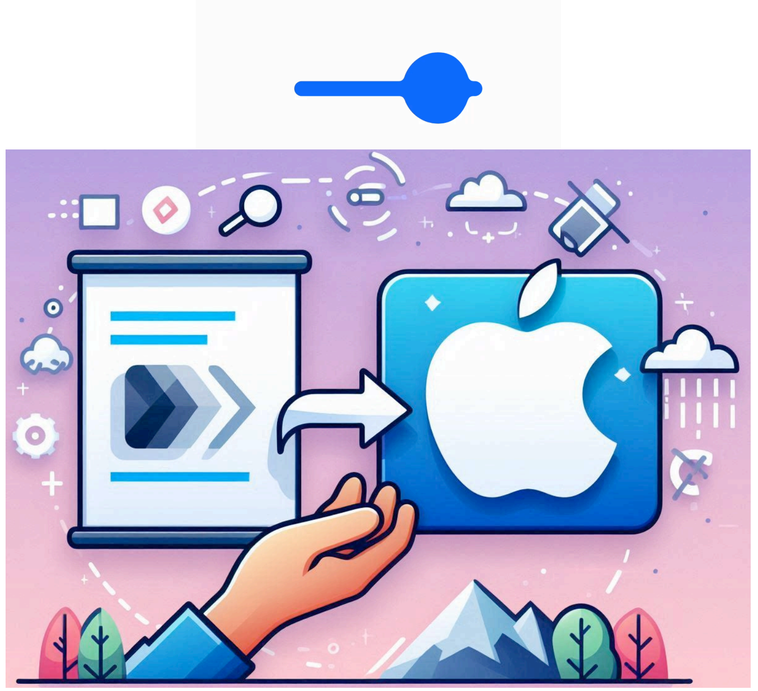
A modifier used within NavigationStack

Used to link path components with their corresponding destination views.

It helps to implement complex navigation flows while maintaining type safety and clarity

```
func navigationDestination<D, C>(
    for data: D.Type,
    @ViewBuilder destination: @escaping (D) -> C
) -> some View where D : Hashable, C : View
```

Switch to Xcode



NavigationSplitView

Another second container view type

Its a view that presents views in two or three columns

Is perfect for multicolumn apps

NavigationSplitView automatically adapts to a single-column stack on iPhone,

Slide Over on iPad, and even on Apple Watch and Apple TV.

NavigationSplitView has two sets of initialisers.

One set, like shown here, creates a two-column experience.

The other set of initialisers creates a three-column experience.

```
struct NavigationSplitView<Sidebar, Content,
Detail> where Sidebar : View, Content : View,
Detail : View
```

NavigationSplitView

Another second container view type

Its a view that presents views in two or three columns

Is perfect for multicolumn apps

NavigationSplitView automatically adapts to a single-column stack on iPhone,

Slide Over on iPad, and even on Apple Watch and Apple TV.

NavigationSplitView has two sets of initialisers.

One set, like shown here, creates a two-column experience.

The other set of initialisers creates a three-column experience.

```
NavigationView { // This is
deprecated.
    /* column 1 */
    /* column 2 */
}.navigationViewStyle(.columns)
```



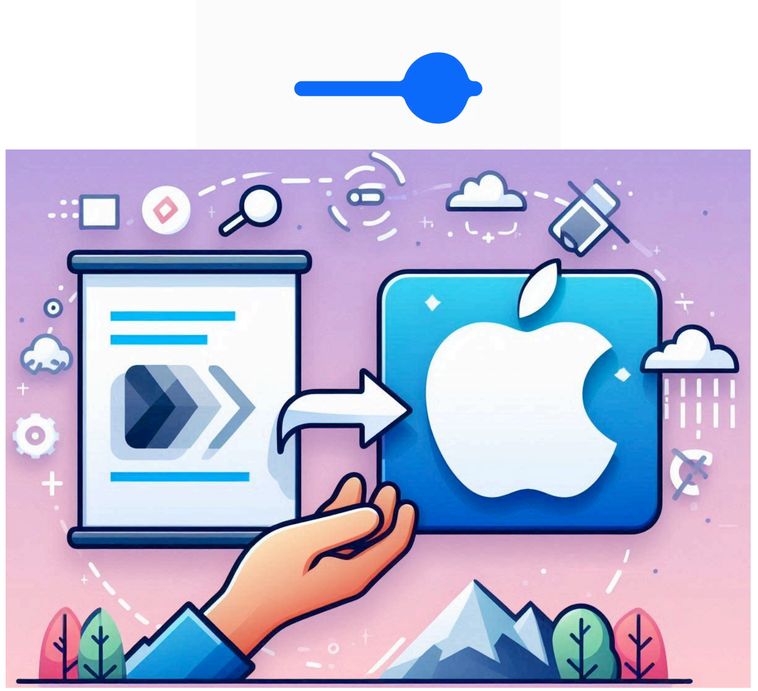
```
NavigationSplitView {
    /* column 1 */
} detail: {
    /* column 2 */
}
```

```
NavigationView { // This is
deprecated.
    /* column 1 */
    /* column 2 */
    /* column 3 */
}.navigationViewStyle(.columns)
```



```
NavigationSplitView {
    /* column 1 */
} content: {
    /* column 2 */
} detail: {
    /* column 3 */
}
```

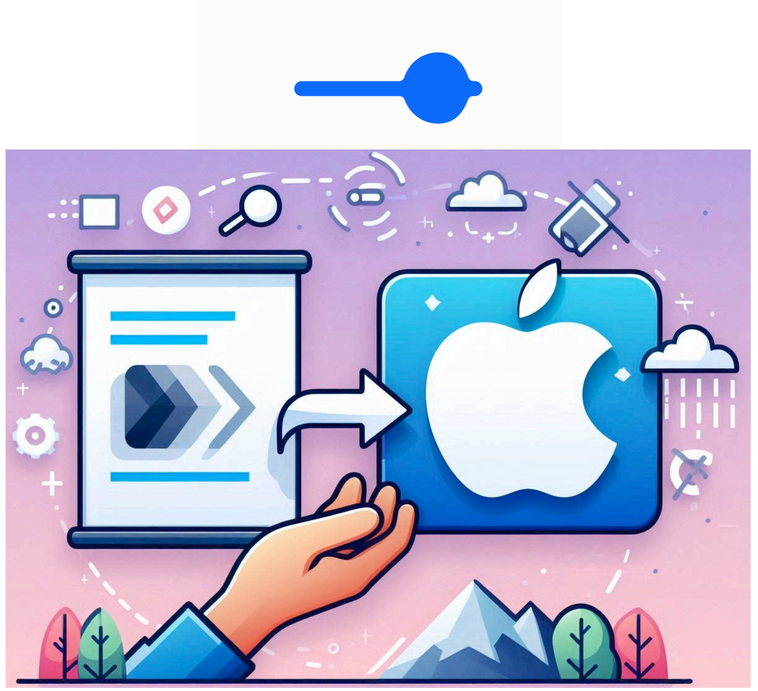
Switch to Xcode



Why Navigation Stack?

- Enables and Eases Programmatic Navigation
- Even with Programmatically from Outside of Views (eg: Deep linking)
- Automatic State Restoration

Switch to Xcode



Navigation patterns for SwiftUI

Limitations on NavigationStack

- RootView **is not part** of NavigationStack
- **Multiple operations** is not allowed
- **Limited Backward** Navigation
- No Direct **Support for TabView**
- No Built-in Support for **Modal Presentation**
- **Navigation bar customisation** is tricky
- **Terrible animations** whenever multiple views are popped off the stack.
- You can not control whether **push/pop operations** should be animated

UINavigationController vs NavigationStack

- UINav works from iOS 2.0+, which is **good for mixed UIKit/SwiftUI** apps,
- NavStack works from **iOS 16.0 only** for SwiftUI apps
- NavStack has **Multiplatform support** (iOS 16+, iPadOS 16+, macOS 13+, tvOS 16+)
- UINavigationController is **Imperative** & Push/pop view controller-based
- NavStack is a **Declarative** & **Path-based** state management
- UINav is Not inherently **type-safe** and NavStack is Type-safe via NavigationDestination
- NavStack has Built-in SwiftUI **accessibility features**

Take aways

- Value must conform to the **Hashable** protocol
- navigationDestination view modifier **should be inside** the NavigationStack
- The top-level navigationDestination view modifier **will always override** the lowest one for the same type
- **Don’t place** navigationDestination view modifier on the child of a lazy container (like List, ScrollView, LazyVStack, etc)
- **Never define the path** in the App protocol
- **Start with NavigationSplitView** if needed
- Separate **Navigation from Views**
- Adopt a **Navigation pattern** which suits your app
- Take advantage of SwiftUI’s Navigation default **accessibility support**

Thank You :)