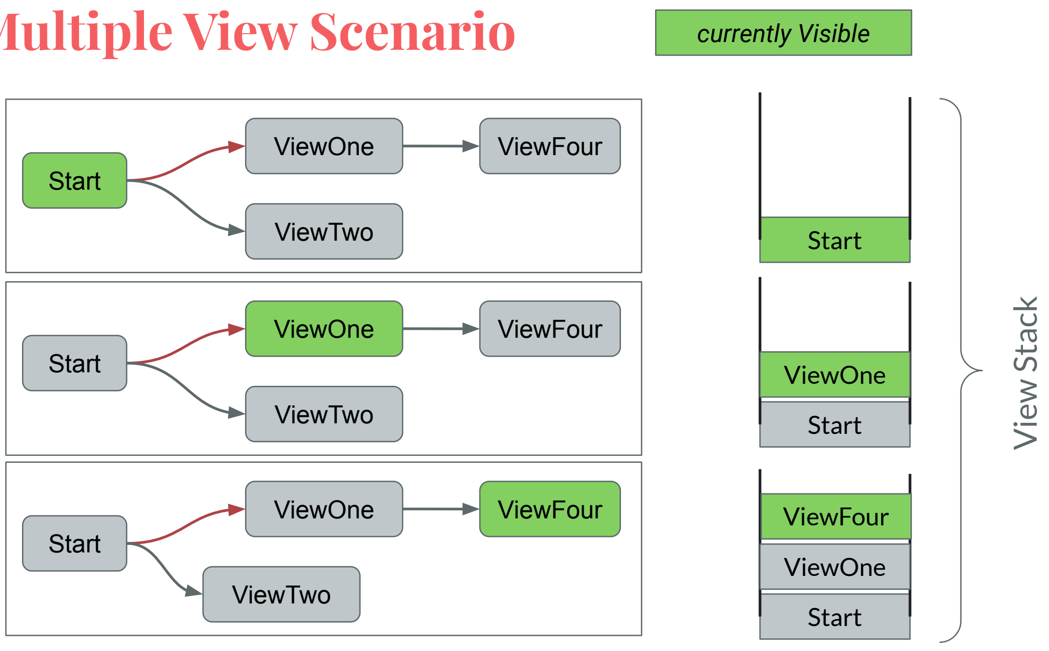
SwiftUI
View Navigation

Screen/View/Page Navigation

Common navigation techniques used by Web Browsers, Mobile Apps, Web Apps

- A “destination” is the next screen/view/page presented to the user
- Navigating to a new destination is accomplished by pushing a new destination on top of the previous one
- They use a stack to manage these destinations. This stack is called
  - Activity Stack (in Android)
  - Navigation Stack (in SwiftUI)
  - History Stack (in Browser)

Multiple View Scenario



Navigation in Jetpack Compose & SwiftUI

Elements needed:

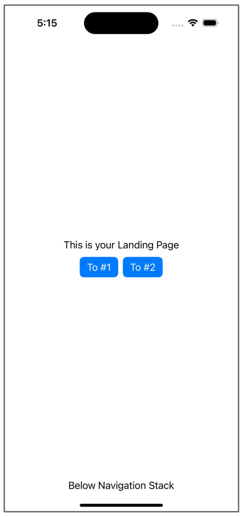
- Screen area where destination view/screen will appear/disappear
- A Stack
- A stack manipulator

	Jetpack Compose	**SwiftUI**
Area for destinations	NavHost	NavigationStack
Stack	*implied*	NavigationPath
Stack manipulator	NavHostController	N/A
What gets pushed to stack	Tokens associated with view/@Composable	

NavigationStack & NavigationLink

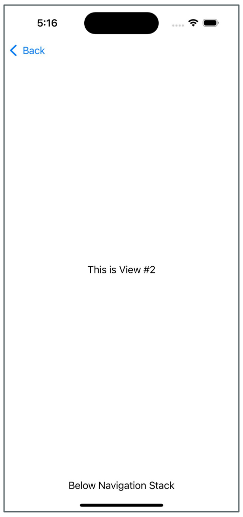
Navigate Forward with Navigation Links

```
const ViewStart: View {
  var body: some View {
     NavigationStack {
       Text("Your Landing Page")
       // Your other widgets here
       HStack {
         NavigationLink("To #1") {
           ViewOne()
         }
         NavigationLink("To #2") {
           ViewTwo()
         }
       }
     }
     .buttonStyle(.borderedProminent)
  }
}
```


5:15

This is your Landing Page

Below Navigation Stack


5:16

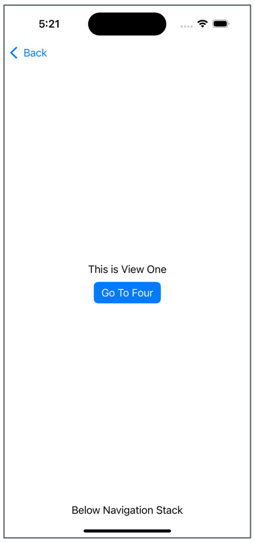
This is View #2

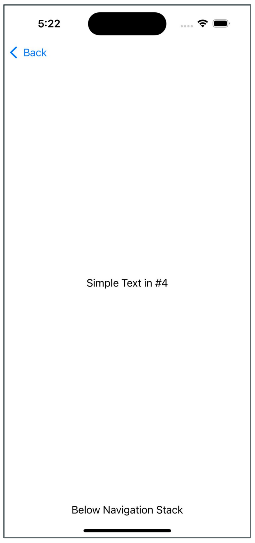
Below Navigation Stack

More NavigationLink

```
import SwiftUI
struct ViewOne: View {
    var body: some View {
        Text("This is View One")

        NavigationLink("Go To Four") {
            Text("Simple Text in #4")
        }
        .buttonStyle(.borderedProminent)
    }
}
```





Programmatic Navigation

Our Design Strategy

- SwiftUI provides only the stack (NavigationPath) but does not provide the stack manipulator
- Our approach
  - Create our own stack manipulator
  - Bundle the manipulator & the stack as an object
  - Inject the bundle as a "global" variable that is accessible through the View hierarchy (parent view, child view, grandchild view, grand grandchild view, ...)

Prerequisite: Swift Environment

- Swift Environment variables
  - The Swift runtime maintains a set of "global" variables in a dictionary/map
  - Each global variable is associated with a unique KeyPath (the key of the dictionary)
  - The keypath identity follows this syntax: `\.someNamePredefinedByApple`
- Property Wrappers
  - @Environment: allows a View to access (read) data which are stored in Swift environment variables
  - @EnvironmentObject: allows your SwiftUI code to inject value into Swift environment variables

Navigate Backward with dismiss

```
const ViewTwo: View {
  // Get the DismissAction to dismiss the current screen

  // Declare as a variable
  @Environment(\.dismiss) var goBackPlease

  var body: some View {
     VStack {
        // Your other widgets here
        Button("Back to Previous Screen") {
          goBackPlease()     // invoked as a function
        }
     }
  }
}
```

[Nav]Stack Manipulator

Step 1: MyNavigator

("Jetpack NavHostController")

MyNavigator.swift

```
import SwiftUI
enum Destination: Hashable {
    case MyFirstDestination  // One "token" for each destination View
    case MySecondDestination
    case MyFourthDestination
}

class MyNavigator: ObservableObject {
    @Published var navPath: NavigationPath = NavigationPath()

    func navigate(to dest: Destination) {
        navPath.append(dest)
    }
    func navigateBack() {
        navPath.removeLast()
    }
}
```

Step 2: Use NavigationStack

```
import SwiftUI

class ContentView: View {
    var body: some View {
        YourLandingViewHere()
    }
}
```

Before

```
import SwiftUI

class ContentView: View {
    @ObservedObject private var navCtrl = MyNavigator()
    var body: some View {
        NavigationStack(path: $navCtrl.navPath) {
            YourLandingViewHere()
        }
        .environmentObject(navCtrl)
    }
}
```

After

- Create MyNavigator at the common parent of all your destinations
- Use `.environmentObject()` to make it "globally" available throughout your view hierarchy

Step 3: Add Navigation Destination

```
import SwiftUI

class ContentView: View {
    @ObservedObject private var navCtrl = MyNavigator()
    var body: some View {
        NavigationStack(path: $navCtrl.navPath) {

              YourLandingViewHere()
               .navigationDestinatin(for: Destination.self) { dest in
                  switch(dest) {
                      case .MyFirstDestination: ViewOne()
                      case .MySecondDestination: ViewTwo()
                      case .MyFourthDestination: ViewFour()

                  }
            }
        }
        .environmentObject(navCtrl)  // Inject the navigator bundle as a “global” variable
    }
}
```

Default destination View

Destination View

Step 4: Use MyNavigator

(in each Destination view)

```
import SwiftUI
struct StartView: View {
  @EnvironmentObject var navCtrl: MyNavigator

  var body: some View {
    Text("This is the start view")

    HStack {
      Button("Go to #1 via NavCtrl") {
        navCtrl.navigate(to: .MyFirstDestination)
      }
      Button("Go to #2") {
        navCtrl.navigate(to: .MySecondDestination)
      }
    }
    .buttonStyle(.borderedProminent)
  }
}
```

```
import SwiftUI
struct ViewOne: View {
    @EnvironmentObject var navCtrl: MyNavigator
    var body: some View {
       // UI Widgets here
    }
}
```

```
import SwiftUI
struct ViewTwo: View {
    @EnvironmentObject var navCtrl: MyNavigator
    var body: some View {
       // UI Widgets here
       Button(“Done”) {
          navCtrl.navigateBack()
       }
    }
}
```

Passing Data Into Destination View

Step A: Modified Destination Enum

```
// Without Associated Value
import SwiftUI
enum Destination: Hashable {
    case MyFirstDestination
    case MySecondDestination
    case MyFourthDestination
}
```

```
// Without Associated Value
import SwiftUI
enum Destination: Hashable {
    case MyFirstDestination
    case MySecondDestination
    case MyFourthDestination
    case BuyStockDestination(String, Int)
}
```

Assuming the view associated with `.BuyStockDestination` now accepts two parameters:

- a String ("stock name to buy")
- an Int ("number of stocks to buy")

Step B: Add Relevant Properties to View

```
import SwiftUI                    // BEFORE
struct BuyStockView: View {
  @EnvironmentObject var navCtrl: MyNavigator
  var body: some View {
    // Your UI here
  }
}
```

```
import SwiftUI                    // AFTER
struct BuyStockView: View {
  @EnvironmentObject var navCtrl: MyNavigator
  /* @State */ var stockName: String
  @State var buyUnit: Int // Use @State when the property is also used in the UI
  var body: some View {
    // Your UI here
    // Use
  }
}
```

Step C: Include Data When Navigate Into Dest

```
import SwiftUI                    // BEFORE
struct SomeView: View {
  @EnvironmentObject var navCtrl: MyNavigator
  var body: some View {
    Button(“Go to #1”) {
      navCtrl.navigate(.MyFirstDestination)
    }
  }
}
```

```
import SwiftUI                    // AFTER
struct SomeView: View {
  @EnvironmentObject var navCtrl: MyNavigator
  var body: some View {
    Button(“Buy Stock”) {
      navCtrl.navigate(.BuyStockDestination(“NVDA”, 75))
    }
  }
}
```

Step D: Unpack and Pass Data To Dest. View

```
NavigationStack(path: $navCtrl.navPath) {             // BEFORE
    YourLandingViewHere()
      .navigationDestination(for: Destination.self) { dest in
          switch(dest) {
              case .MyFirstDestination: ViewOne()
              case .MySecondDestination: ViewTwo()
              case .MyFourthDestination: ViewFour()

          }
       }
}
```

```
NavigationStack(path: $navCtrl.navPath) {                      // AFTER
    YourLandingViewHere()
      .navigationDestination(for: Destination.self) { dest in
          switch(dest) {
              case .MyFirstDestination: ViewOne()
              case .MySecondDestination: ViewTwo()
              case .MyFourthDestination: ViewFour()
              case .BuyStockDestination(let whichStock, let howMany):
                   BuyStockView(whichStock, howMany)
          }
       }
}
```