TODAY

- Navigation Controllers
- Firebase

NAVIGATION CONTROLLER

Pushes and pops off of a stack.

“Pushed” view controllers are always brand new, and are destroyed as they are popped off.
Your view controller has the following properties for children:

- var viewControllers: [UIViewController]? { get set }
- // in order (left-to-right (tab), master-detail (split), root etc. (nav)

References to enclosing view controllers

- var tabBarController:UITabBarController? { get }
- var splitViewController:UISplitViewController? { get }
- var navigationController:UINavigationController? { get }

Pushing/popping VC from a UINavigationController

- func pushViewController(_ vc: UIViewController, animated: Bool)
- func popViewController(animated: Bool)

But rarely done. Instead:

- segues
SEGUES

• For navigation and split-screen controllers, we use **segues**:  
  • *show* segue: overlays a new VC  
  • *show detail* segue: same, but shows detail  
  • modal segue: takes over whole screen, already shown  
  • popover segue: little window  

• Important: **segues**  
  • always create *new instances* of VC  
  • always thrown away when we go *Back* (not a segue)

---

SEGUES

• How do we use the segue identifier?  
  • with storyboard  
    • `ctrl-drag from VC to VC`  
  • programmatic segues (rare)  
    • perform the segue programmatically  
    • sender can be anything  

```swift
func performSegue(withIdentifier: String, sender: Any?) {
    . . .
}
```
**SEGUES**

- parent VC *prepares* child VC for segue, calls:
  ```swift
  func prepare(for segue: UIStoryboardSegue, sender: Any?) {
    if let identifier = segue.identifier {
      switch identifier {
      case "moon":
        if let segueVC = segue.destination as? PicController {
          segueVC.this = "blather"
          segueVC.this = 42
        }
      case "orion":
        . . .
      default: break
      }
    }
  }
  ```

  - Destination is always brand-new VC, *outlets not yet set*
  - *sender* is button or control that caused the segue

**SPLITVIEWCONTROLLER**

- Side-by-side VCs:
SPLITVIEWCONTROLLER

- Just show detail in portrait, but
  - can pull out the master for side-to-side
- None of this works on smaller phones, even Xr/X
  - so we wrap the master in nav controller
  - and then split-view does it’s thing on iPad or iPhone
SEGUES

- Can prevent segue from happening:

  ```swift
  func shouldPerformSegue(withIdentifier identifier: String?,
                          sender: Any?) -> Bool {
        return false
  }
  ```

NAV CNTR COMMUNICATION

- To summarize communication between view controllers through a navigation controller:
  - parent sets up child controller through:
    - a `prepare(... for segue ...)` call
  - child responds to parent controller through:
    - a `delegate` protocol (parent is the delegate)
FIREBASE OVERVIEW

- NoSQL
  - no rigid schemas
  - no SQL
- Based on JSON object tree
  - all queries performed on sub-trees
- Concurrency control
  - atomic multi-path puts
  - transactions

FIREBASE DATABASES

<table>
<thead>
<tr>
<th>Realtime Database</th>
<th>Cloud Firestore</th>
</tr>
</thead>
<tbody>
<tr>
<td>one large JSON tree, very simple</td>
<td>collections of documents w/ sub-collections, much complexity</td>
</tr>
<tr>
<td>simple data model, complex data hard to model</td>
<td>less flattening because of collections/documents/sub-collections</td>
</tr>
<tr>
<td>queries can do either filtering or sorting</td>
<td>queries can do both, and with more precision</td>
</tr>
<tr>
<td>single geo-region</td>
<td>multiple geo-regions</td>
</tr>
</tbody>
</table>
COCOAPODS

- dependency manager built on ruby
- Podfile explicitly specifies versions of external frameworks

```ruby
target 'fire' do
  platform :ios, '12.2'

  pod 'Firebase'
  pod 'Firebase/Core'
  pod 'Firebase/Database'
  pod 'Firebase/Auth'
end
```

- "pod install",
- **must** open the `.xcworkspace` instead of `.xcodeproj`

CONFIGURING FIREBASE

1. bring up firebase console in browser w/ google account
2. “New Project”, give it a name, accept terms, continue
3. Specify bundle from the app
4. Click “register app” and download Googleservice-info.plist
   a. put in app on same level as source files
5. In main console click sign-in method, email/password, save.
6. Click on database, select “Create Database” in realtime section, start in “test mode”.
DATABASE AUTH RULES

- “testing” rules:
  
  ```json
  "rules": {
    ".read": "auth != null",
    ".write": "auth != null"
  }
  ```

- regular rules:
  
  ```json
  "rules": {
    "users": {
      "$uid": {
        ".read": "auth != null && auth.uid == $uid",
        ".write": "auth != null && auth.uid == $uid",
        "items": {
          "$item_id": {
            "title": {
              ".validate": "newData.isString() && newData.val().length > 0"
            }
          }
        }
      }
    }
  }
  ```

JSON OBJECT TREE

```json
{
  "users": {
    "alovelace": {
      "name": "Ada Lovelace",
      "contacts": { "ghopper": true },
    },
    "ghopper": { ... },
    "eclarke": { ... }
  }
}
```
**AppDelegate**

- import Firebase
- add FirebaseApp.configure() in first method
  - or put in new `override init()` if crash

**REFERENCES AND WRITES**

- basically paths
  ```swift
  let rootRef = Database.database().reference()
  let itemsRef = rootRef.child("items")
  ```

- leaves are dictionaries
  ```swift
  let firstValues: [String: Any] = [
  "name": "Good Dog",
  "added by": "pete",
  "done": false
  ]
  let firstRef = itemsRef.child("first")
  firstRef.setValue(firstValues)
  ```

- nothing actually written until `setValue`

- arrays -ish
  ```swift
  let key = ref.childByAutoID()
  key.setValue("a string")
  ```
READING (OBSERVERS)

- asynchronous

- possibly repeated
  ```swift
  itemsRef.observe(.value, with: { (snapshot) in
    print(snapshot)
  })
  ```

- sometimes not
  ```swift
  itemsRef.observeSingleEvent(of: .value, with: { (snap) in
    for child in snap.children {
      if let snapshot = child as? DataSnapshot {
        print("\(snapshot.ref): \(snapshot.key) VALS \(snapshot.value)"")
      }
    }
  })
  ```

ALSO

- updates
- observing different things
  - childAdded
  - childRemoved
  - childChanged
  - childMoved
  - value
- want tree to be shallow
  - queries return entire subtrees
• **Huge subject**
  
  • User accounts
  
  • Authentication
  
  • iOS Keychain
  
  • Read up