TESTING

Martin Vidovic, iOS Engineer at STRV

STRV

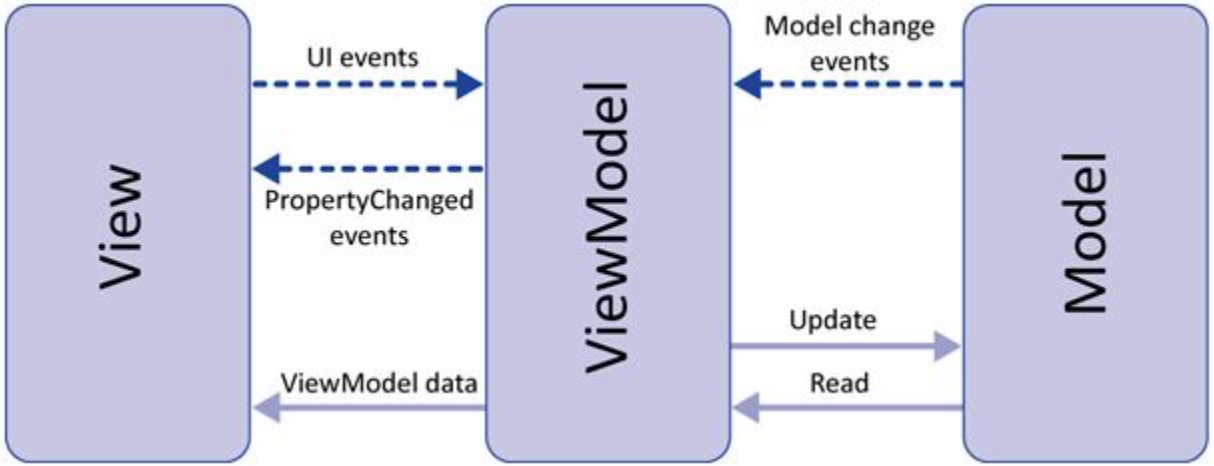
AGENDA

- Architecture
- DI
- Mocking
- Tests

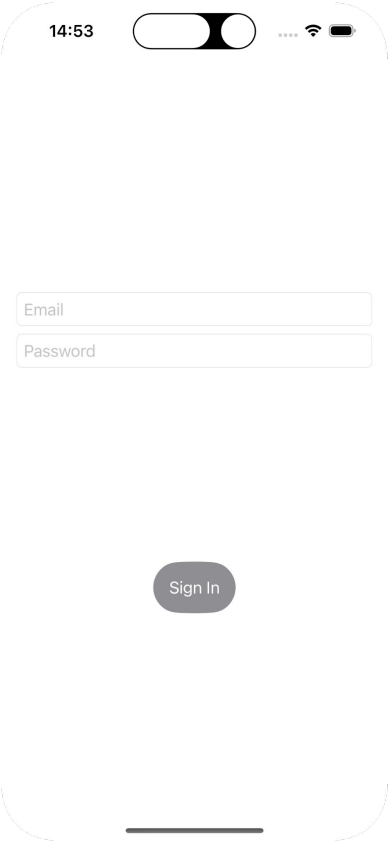
MVVM

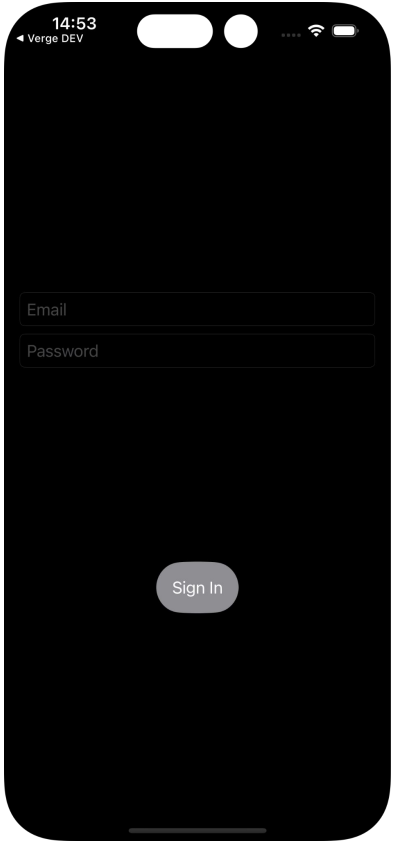
01

MVVM - Model View ViewModel



MVVM - Model View ViewModel





DEPENDENCY INJECTION

02

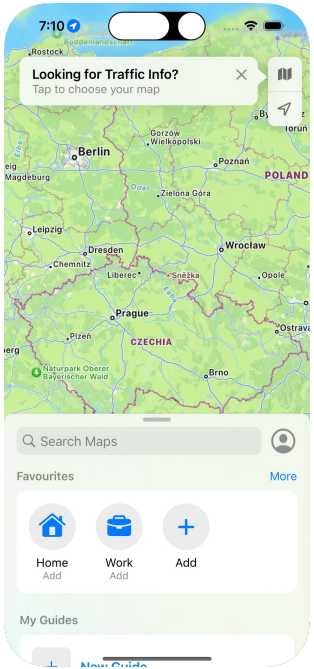
DEPENDENCY INJECTION

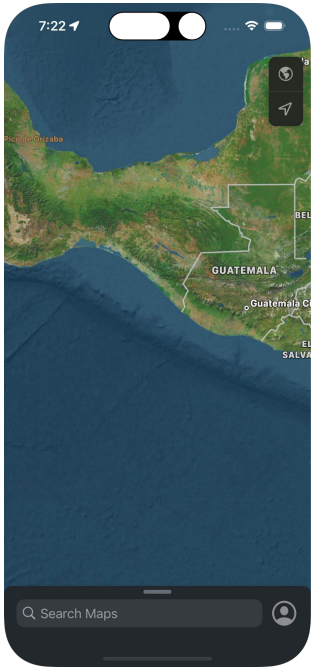
Passing a concrete dependency (e.g. view model or service) from outside of the object to avoid repeated code by extracting initialization

- Reduces repeated code
- Removed knowledge about implementation details where it is not needed
- Helps with interchangeability
- Allows testability

DI EXAMPLE

- Location Manager





TYPES OF DEPENDENCY INJECTION

- Initializer Injection
- Property Injection
- Method Injection
- Property Wrappers
- Singletons 😃

MOCKING

03

MOCKING

- The purpose of mocking is to isolate and focus on the code being tested
- An object under test may have dependencies on other (complex) objects. We want to eliminate those complex dependencies / objects.
- Options
  - Fake - fixed values
  - Stub - different values based on inputs
  - Mock - different values based on inputs + business logic

MOCKING PREVIEWS

- Enables to see SwiftUI Preview / Canvas

```
57  #Preview {
58      RelationshipGoalsView(
59          store: RelationshipGoalsStore.PreviewStore()
60      )
61  }
```

MOCKING STORE (VIEWMODEL)

- Provide default values & empty functions
- Useless mock, but needed for SwiftUI Preview

```
// MARK: - Mock
#if DEBUG
extension RelationshipGoalsStore {
    class PreviewStore: RelationshipGoalsStoring {
        var state: RelationshipGoalsState = .initial

        func send(action _: RelationshipGoalsAction) {}
    }
}
#endif
```

MOCKING MODELS

- Replacing Classes / Structs Models
- Usage:
  - Networking
  - Secrets
  - Simulator values
  - Location

```
struct User: Decodable, Equatable {
    let userId: Int
    let id: Int
    let title: String
    let completed: Bool
}
```

MOCKING MODELS

- Examples

```swift
// MARK: Testing
extension User {
    static let mock: User = User(
        userId: 1,
        id: 1,
        title: "delectus aut autem"
        completed: false
    )
}
```

```swift
// MARK: Testing
extension User {
    static let mockJSONString = """
    {
        "userId": 1,
        "id": 1,
        "title": "delectus aut autem",
        "completed": false
    }
    """
}
```

MOCKING BUSINESS LOGIC

- Faking the whole class
- What if protocol changes?

```
protocol NetworkingProtocol {
    func fetch() -> Int
}

class NetworkManager: NetworkingProtocol {
    func fetch() -> Int {•••}
}

class FakeNetworkManager: NetworkingProtocol {
    func fetch() -> Int {
        return 3
    }
}
```

MOCKING BUSINESS LOGIC

- Testing with unit tests

```
func testCorrectEmail() {
    let email = "abcd@efgh.ijkl"
    let result = validationManager.validateEmail(
        input: email
    )
    XCTAssertEqual(result, true)
}

func testWrongEmail() {
    let email = "Martin"
    let result = validationManager.validateEmail(
        input: email
    )
    XCTAssertEqual(result, false)
}
```

T□STING

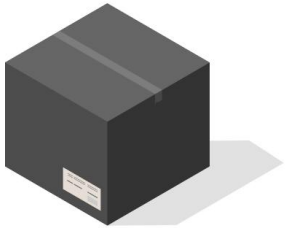
Martin VidovSyntaxError: illegal character

STRV

IMPORTANT QUESTIONS

- Do we have money? 💰
- Do we have time? ⏱️
- Do we have knowledge? 💪🤔
- What's the difference between manual and automatic testing?

QA testers



Black box - we do not know anything

Developers

White box - we know everything

WHY DO WE TEST??

- To be sure it WORKS!!!

WHAT TO TEST?

- All good scenarios
- All bad scenarios
- Bug fixes
- Refactored code

““A failing test is a good test.”

Me

TYPES OF TESTS

05

TYPES OF TESTS??

- Unit tests
- UI tests
- Integration tests
- Smoke tests
- Regression tests
- …
- Snapshot Tests 📷

UNIT TESTS

- Software testing of source code
- Individual single-purpose testing of each method

```
8   import XCTest
9   @testable import SignMeApp
10
    class SignMeAppTests: XCTestCase {
12      override func setUpWithError() throws {
13          // setup before test
14      }
15
16      override func tearDownWithError() throws {
17          // setup after test
18      }
19
        func testUnitTest() {
21          XCTAssertEqual(true, true)
22      }
23  }
```

UNIT TESTS

```
final class ValidationManagerTests: XCTestCase {
    var container: DIContainer!
    var validationManager: ValidationManaging!

    override func setUpWithError() throws {
        container = DIContainer()
        validationManager = container.validationManager
    }

    override func tearDownWithError() throws {
        container = nil
        validationManager = nil
    }

    func testCorrectEmail() {
        let email = "abcd@efgh.ijkl"
        let result = validationManager.validateEmail(
            input: email
        )
        XCTAssertEqual(result, true)
    }
```

UI TESTS

- User Interface Testing
- Tests the application's visual elements
- Checks functionality and expected performance

```
8   import XCTest
9
    class SignMeAppUITests: XCTestCase {
11
12      override func setUpWithError() throws {
13          // setup before test
14      }
15
16      override func tearDownWithError() throws {
17          // setup after test
18      }
19
        func testExample() throws {
21          let app = XCUIApplication()
22          app.launch()
23      }
24  }
```

UI TESTS

```
class SignInViewTests: XCTestCase {
    var app: XCUIApplication!
    var button: XCUIElement {
        app.buttons["signInButton"]
    }

    override func setUpWithError() throws {
        continueAfterFailure = false
        app = XCUIApplication()
        app.launch()
    }

    override func tearDownWithError() throws {
        takeScreenshotOfFailedTest()
        app = nil
    }

    func testEmail() {
        let textField = app.textFields["signInTextField"]
        let myEmail = "xxx.yyy@gmail.com"
        textField.tap()
        textField.typeText(myEmail)
        let typedText = textField.value as? String ?? ""
        XCTAssertEqual(typedText, myEmail)
        XCTAssertEqual(button.isEnabled, true)
    }
```

INTEGRATION TESTS

- Integration Testing is defined as a type of testing where software modules are integrated logically and tested as a group.
  - Unit test + Unit test = Integration test
  - UI test tests logic + elements = Integration test

```
8  import Foundation
9  @testable import SignMeApp
10 import XCTest
11
13 final class SignInStoreTests: XCTestCase {
14     override func setUpWithError() throws {
15         container = DIContainer()
16         signInStore = container.signInStore
17     }
18
19     override func tearDownWithError() throws {
20         container = nil
21         signInStore = nil
22     }
24
25     func testInit() {
26         XCTAssertEqual(signInStore.emailText, "")
27         XCTAssertEqual(signInStore.passwordText, "")
28         XCTAssertEqual(signInStore.buttonDisabled, true)
27     }
28 }
```

SMOKE TESTS

- Testing basic functionality
  - Such as if the application launches
- Testing to reveal simple failures
- If a piece of software passes a smoke test, quality assurance (QA) teams then proceed with further testing.

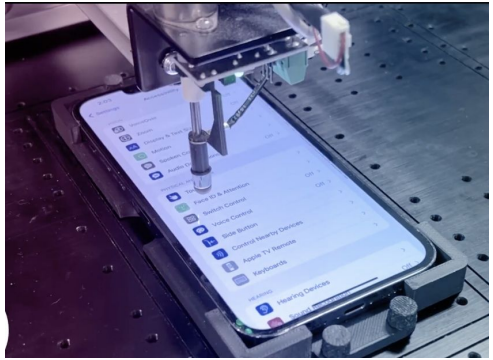
```
    func testLaunchPerformance() throws {
        if #available(macOS 10.15, iOS 13.0, tvOS 13.0, watchOS 7.0, *) {
            // This measures how long it takes to launch your application.
            measure(metrics: [XCTApplicationLaunchMetric()]) {   2   Duration (AppLaunch): 0.954 s
                XCUIApplication().launch()
            }
        }
    }
}
```

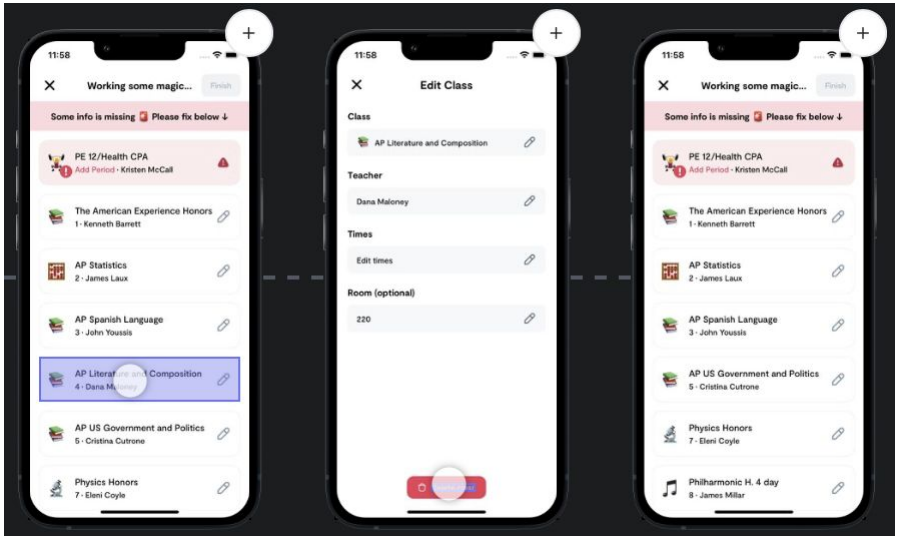
REGRESSION TESTS

- Testing before each release
- Running functional and non-functional tests to ensure that previously developed and tested software still performs after a change
- Done by QA Tester
- There is an output file
  - passed tickets
  - suggestions to future
  - found bugs
- Finishes with approval for release ✌️🍀 or not ❌

OTHER TYPES OF TESTS

- Snapshot Tests
- Performance Tests
- Benchmark Tests
- 3rd party services:
  - Waldo
  - Mobot





QUESTIONS

STRV

THANK YOU!

Martin Vidovic / martin.vidovic@strv.com

STRV