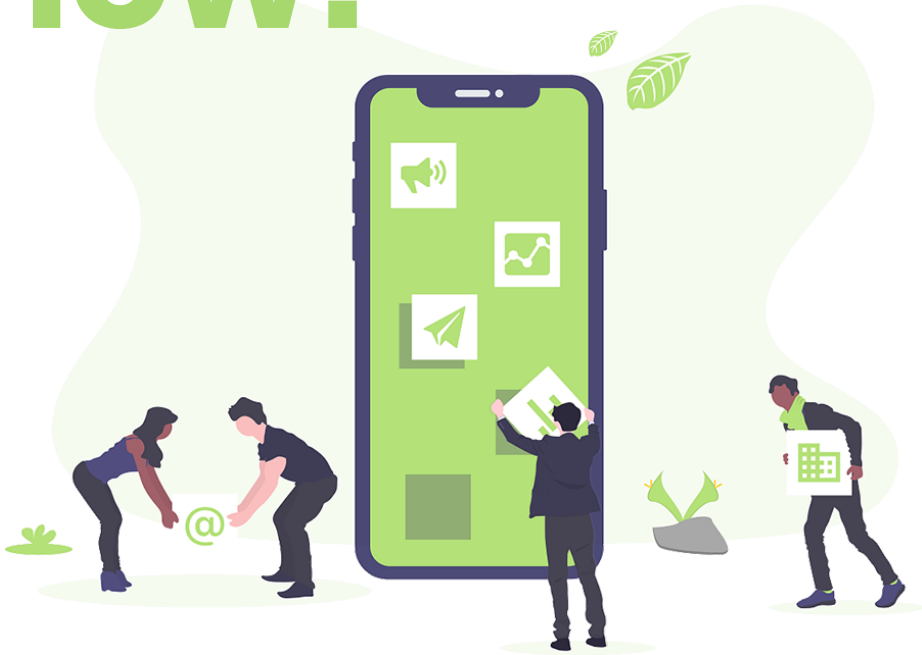
TDD in iOS, why and how?

- Test First
- Best Practices
- Testable Code



MOBILEDAY

Hello!

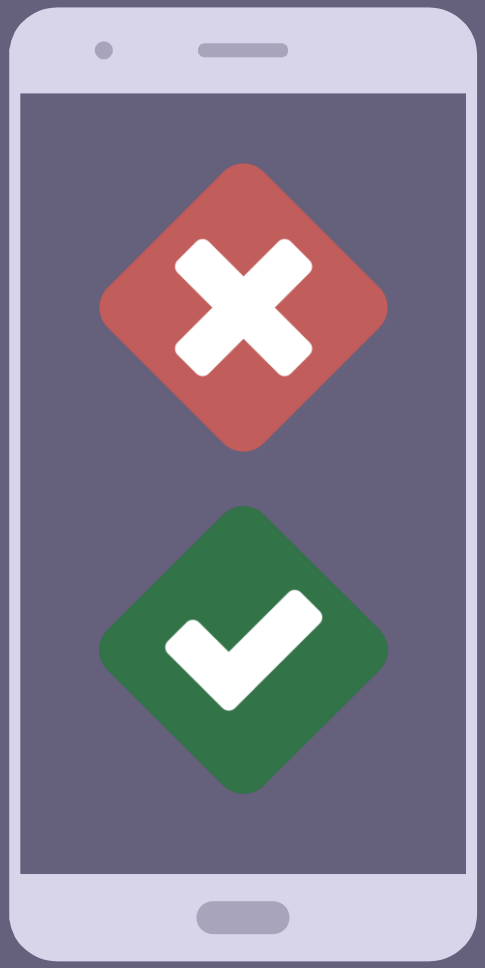
I am **Tai Le** @levantAJ

Mobile Software Engineer at **ShopBack**

Test-Driven Development

How to write test first?

Test First



Why?

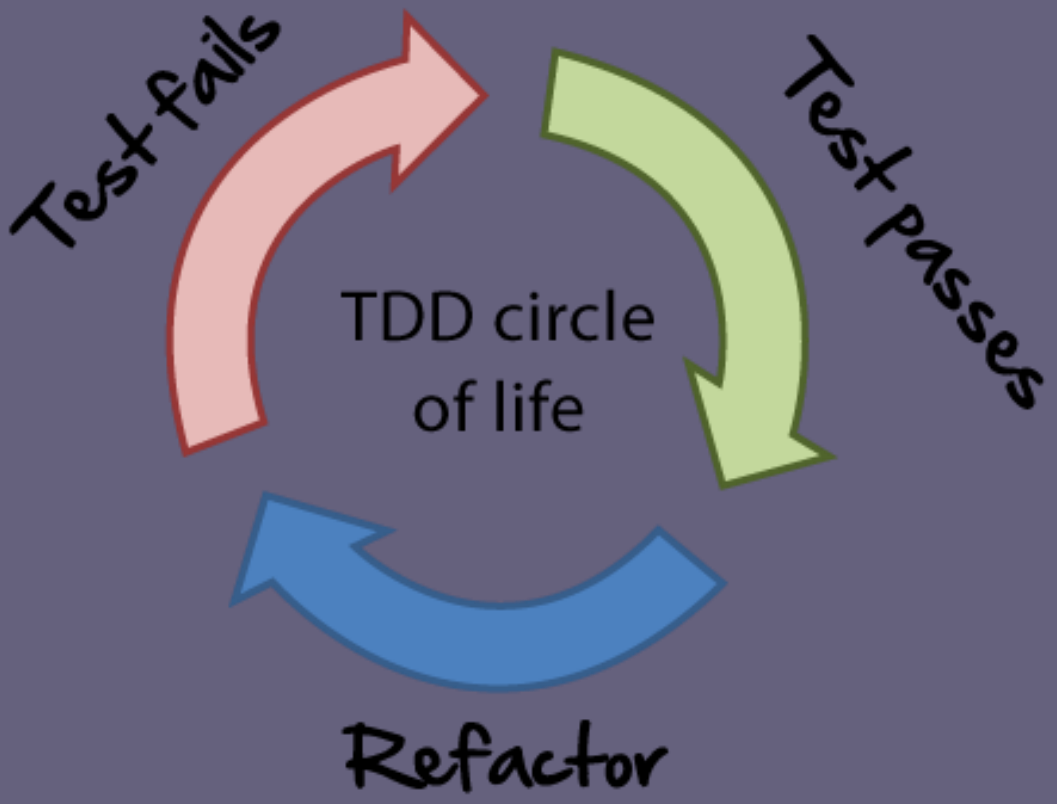
- Project build-outs to take up to 30% longer with TDD.
- TDD reduces production bug density 40%-80%

Why?

- Make product more stable
- Easy to refactor
- Tests are documentation
- Better code quality: Readability, maintain, modify...

How?

- Failing tests (RED)
- Minimum amount of code to pass the tests (GREEN)
- Refactor


Test fails
Test passes
TDD circle
of life
Refactor

Unit Test in Best Practices

How to write the tests more effective?

Unit Tests are

FIRST

Unit Tests are FIRST

- Fast
- Isolated (Independent)
- Repeatable
- Self-Verifying
- Timely

System Under Test

“ Given – When – Then

(Behavior-Driven Development)

```
func test<#method#>(){
    //Given:
    <#given#>
    //When:
    <#when#>
    //Then:
    <#then#>
}
```

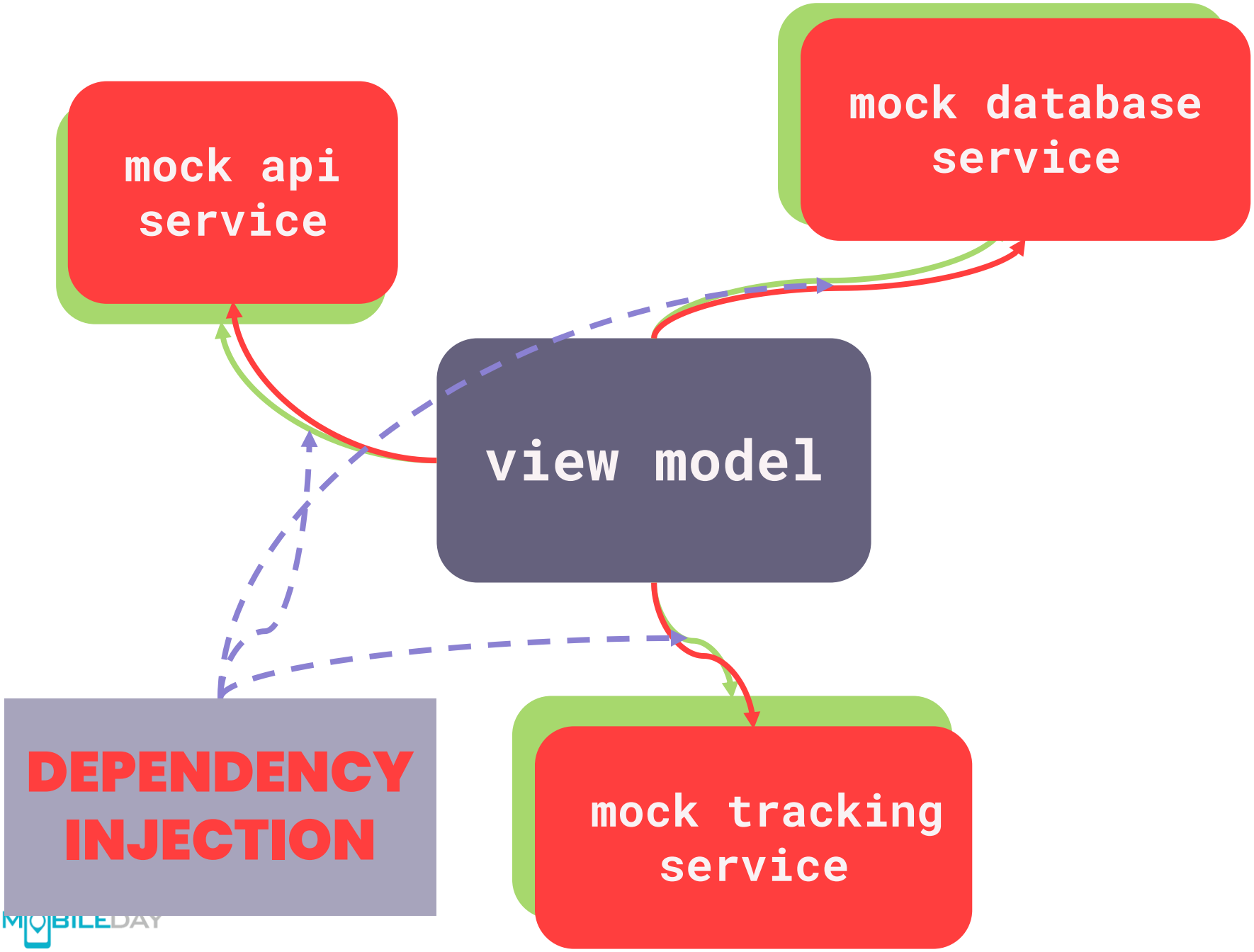
Dependency Injection & Mocking

How to write testable code?

Interaction Test

Dependency Injection

- Passing dependency to other objects or framework.
- **Testing easier!**


mock api service
mock database service
view model
DEPENDENCY INJECTION
mock tracking service

Forms of Dependency Injection

1. Constructor Injection
2. Property Injection
3. Method Injection
4. Extract and Override Call
5. Ambient Context

Mocking

- Creating objects that simulate the behavior of real objects.

Basic type of Mocking

- Inheritance Mocks
- Protocol Mocks

Inheritance Mocks

```
// Override the real class
class MockApiService: ApiService {
    var didCallFetchData = false

    override func fetchData(completion: (Data?, Error?) -> Void) {
        didCallFetchData = true
    }
}
```

Protocol Mocks

```swift
//1: Extract to a protocol
protocol ApiServiceProtocol {
    func fetchData(completion: (Data?, Error?) -> Void)
}

//2: Confirm real class to the protocol
class ApiService: ApiServiceProtocol {}

//3: Create mock class from the protocol
class MockApiService: ApiServiceProtocol {
    var didCallFetchData = false

    func fetchData(completion: (Data?, Error?) -> Void) {
        didCallFetchData = true
    }
}
```

Frameworks/Libs

- Cuckoo (Swift)
- OCMock (Objective-C)

Conclusion

- No silver bullet
- Take time
- Maintenance

Thanks!

Any questions?

You can find me at:

- @levantAJ
- tai.le@shopback.com

“Where to go from here?

- https://geek-is-stupid.github.io/2019-04-03-mocking-for-test-in-ios-development/
- https://geek-is-stupid.github.io/2019-04-20-way-to-approach-unit-tests-in-iOS-developmemt/
- https://pragprog.com/magazines/2012-01/unit-tests-are-first
- https://www.objc.io/issues/15-testing/dependency-injection/
- https://www.artima.com/lejava/articles/designprinciples.html