Building robust UI tests

Supercharge

Why should we test?

- Maintain development / release speed
- Reduce number of bugs
- Catch bugs before production
- Better code structure and quality
- Makes refactoring easier
- Confidence for the team

Then why are we not testing?

Because this is how our CI runs automated tests

Why are we not testing?

- It has a learning curve
- Needs infrastructure
- Test fragility
- Extra development effort
- Maintenance costs
- Changing requirements

What should we test?
TDD is not a must

What should we test?

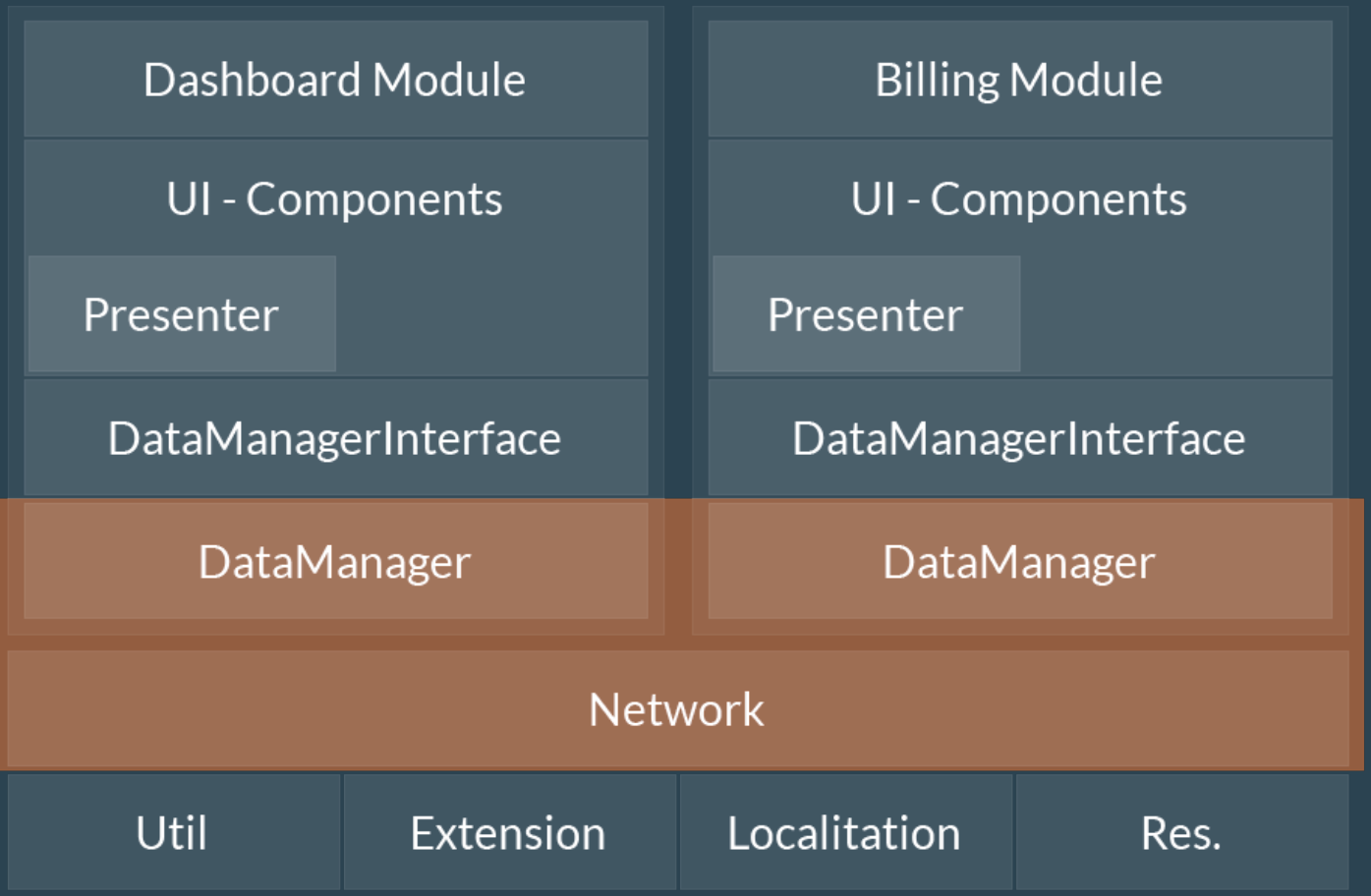
- Complex business logic
- Integration
- UI Components
- User journeys

How should we prepare our code for testing?

Code for testing

- Clean code
- Dependency injection
- Different environments - MOCK
- Test features – accessibility label

Code for testing



How should we start testing?

Best practice

- Stability over everything
- Built into the CI loop
- Easy to write
  - Stay close to the original dev environment
  - Choose a stable test framework
- When things constantly change: don’t test them
- Write environment independent tests
- STAY GREEN

Testing on iOS

Unit testing business logic and integration

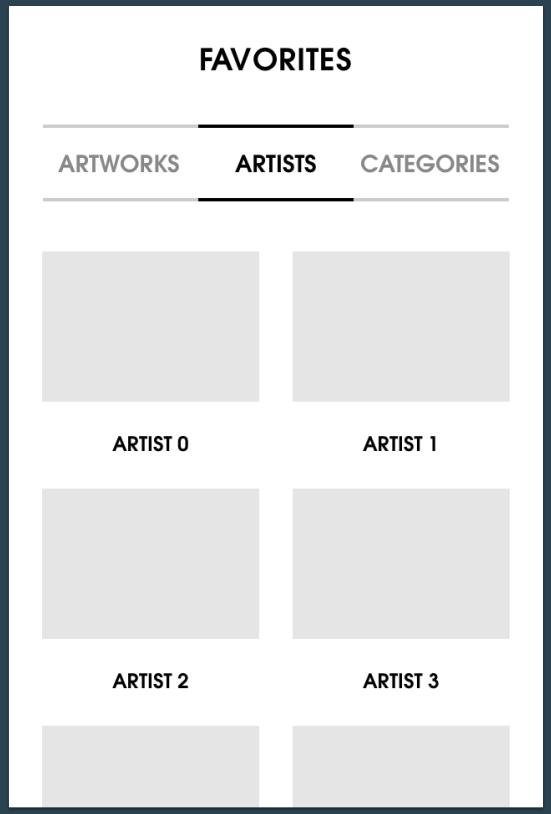
```
beforeAll(^{
    [Expecta setAsynchronousTestTimeout:100];
    NSError *error;
    [NSURLConnectionVCR startVCRWithPath:[NSProcessInfo processInfo].environment[@"VCR_REFERENCE_CASSETTES_DIR"] error:&error];
});
afterAll(^{
    [NSURLConnectionVCR stopVCRWithError:nil];
});

describe(@"ULTIntegrationTest", ^{
    ULTServiceDataManager *serviceDataManager = [ULTServiceDataManager new];

    it(@"checks category integration", ^{
        __block bool returned = false;

        [serviceDataManager servicesCompletionHandler:^(NSArray *categories) {
            expect(categories.count).equal(6);
            returned = true;
        } error:^(NSError *error) {
            expect(false).equal(true);
            returned = true;
        }];

        expect(returned).will.equal(true);
    });
});
```

Unit testing business logic and integration

- Mock network communication (Nocilla, NSURLConnectionVCR)
- Mock application state (OCMock)
- Assert for data that will be displayed on UI
- PRO:
  - Stable, Fast
  - You can test specific, complicated logics
- CON:
  - Can be really hard to mock the environment
  - Time consuming

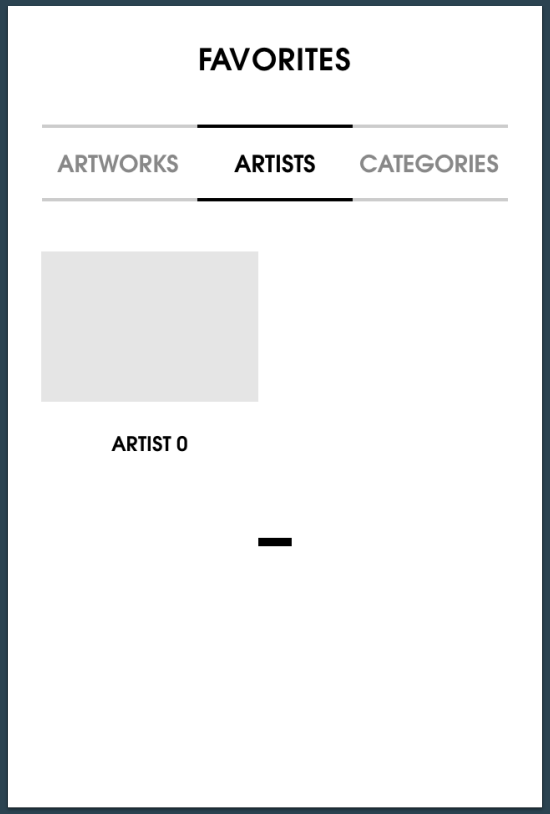
Snapshot testing UI components

Reference Snapshot



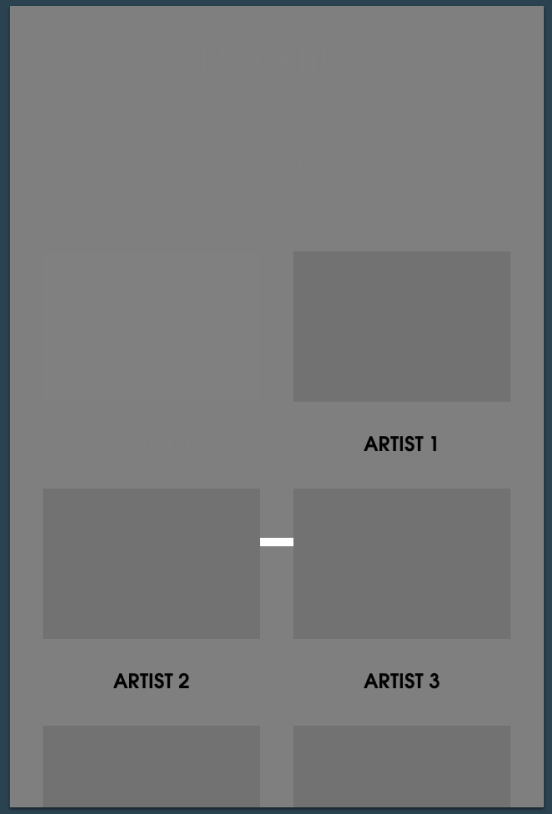
reference_iphone_artists_with_artists_looks_correct@2x.png

Runtime Generated Snapshot



failed_iphone_artists_with_artists_looks_correct@2x.png

Visual Difference of changes

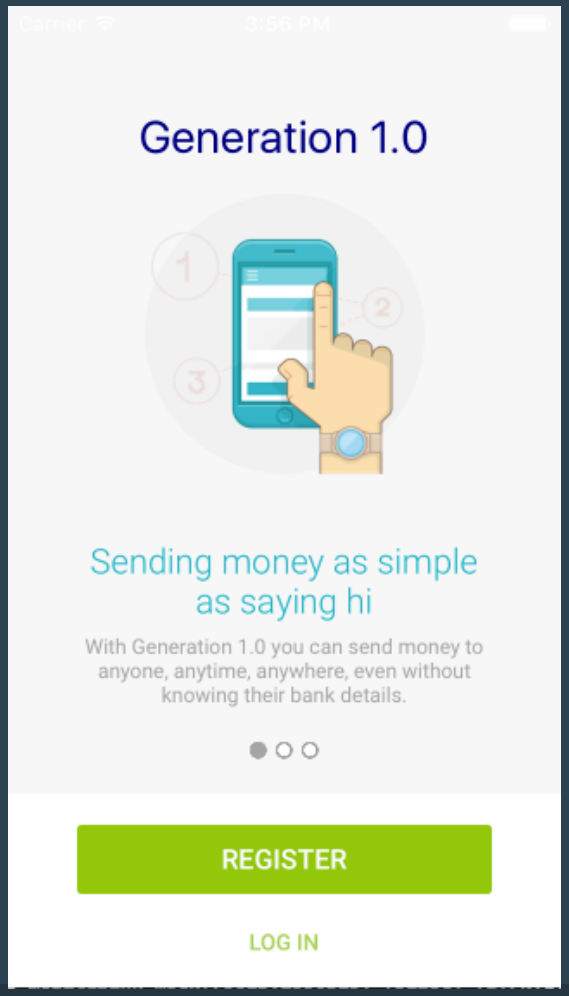


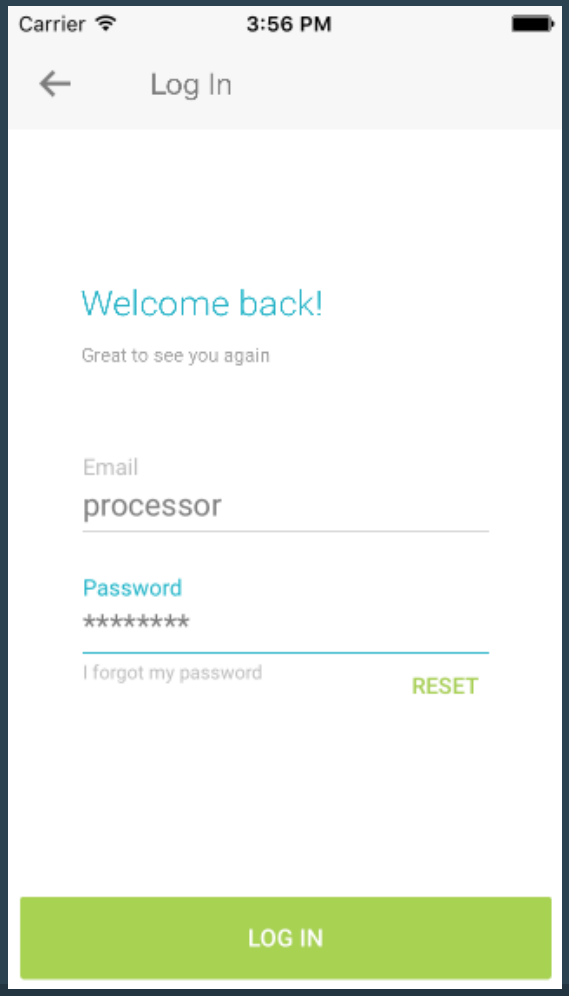
diff_iphone_artists_with_artists_looks_correct@2x.png

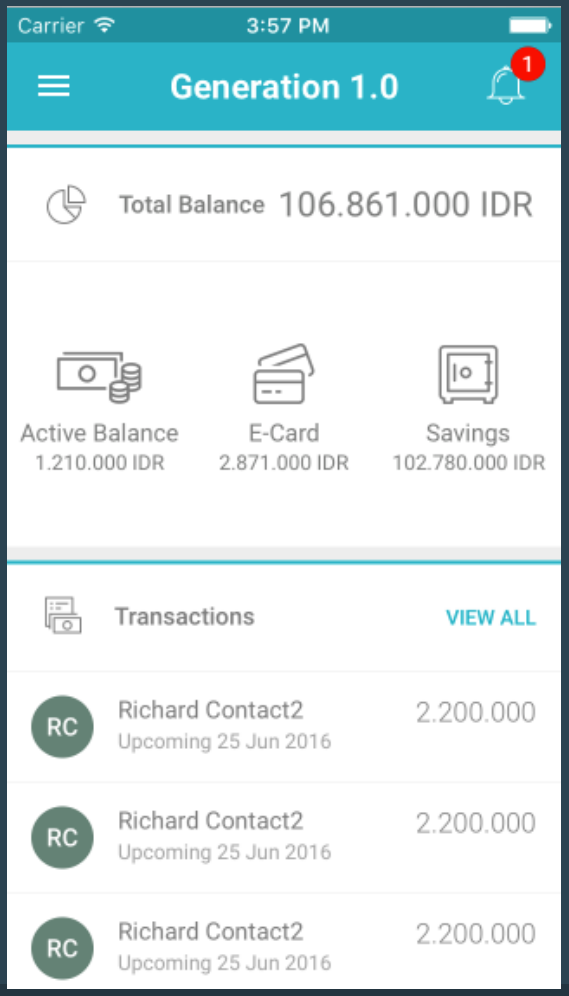
Snapshot testing UI components

- Mock displayed data
- Take a screenshot of the view and compare it to previous reference screenshot
- PRO:
  - Runs stable
  - You can test, complicated UIs
- CON:
  - Fragile: small change can break many test cases
  - Challenging to recapture / maintain screenshots

UI testing user journeys







UI testing user journeys

- UI testing framework (KIF, EarlGrey, Xcode UI tests, etc..)
- Have a mock environment
- Write test cases that goes through user journeys
- PRO:
  - Tests the UI in motion
  - Easy to write
  - Good effort / result ratio
- CON:
  - Can be unstable, Slow
  - Needs continuous infrastructure maintenance

Acceptance testing with integration

- UI testing framework (KIF, EarlGrey, Xcode UI tests, etc..)
- Have a test server to communicate with or record network communication
- Write test cases that goes through user journeys
- PRO:
  - Tests the whole app in motion
- CON:
  - Unstable, Slow
  - Needs continuous infrastructure maintenance

How to stabilize tests?

- Fix environment
  - Mock data
  - Recorded network communication
- Environment independent test cases
- Use a stable framework: KIF
- Experiment with frameworks / test runners
  - Newest KIF, xctool seems to be solid

How to fasten up UI tests? Parallelize

- We can run multiple simulators and attach the test runner
- Reduces stability
- 4-5x speed improvement

Takeaways – Start testing!

- Write UI tests in mock environment
- Integrate it to CI
- Experiment with tools
- Continuously improve test stability and speed

References

- http://typhoonframework.org/
- http://martinfowler.com/bliki/FeatureToggle.html
- https://github.com/team-supercharge/SCConfiguration
- http://ocmock.org/reference/
- https://github.com/specta/expecta/
- https://github.com/facebook/ios-snapshot-test-case
- https://github.com/dstnbrkr/VCRURLConnection
- https://github.com/luisobo/Nocilla
- https://wiki.jenkins-ci.org/display/JENKINS/Home
- http://oclint.org/
- https://github.com/Cue/ocstyle
- https://github.com/google/EarlGrey

Thanks for your attention!

Questions?

David Kovacs
Supercharge
CTO
@davidkovaccs

david.kovacs@supercharge.io // www.supercharge.io

Supercharge