# Building highly customizable iOS applications on top of a core codebase

Supercharge

# Introduction - concept

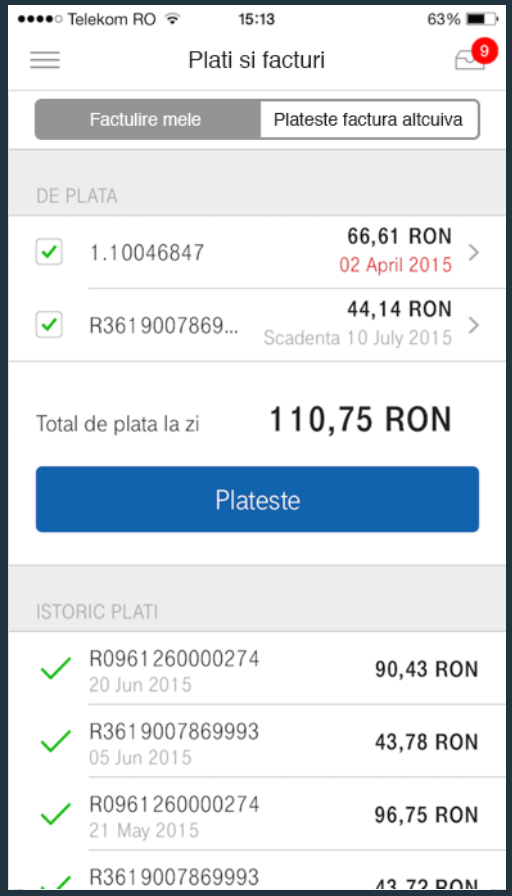

- State-of-the-art Telco self-care application
- Core application with a basic feature set
- Customized and distributed to multiple countries

# Agenda

- Introduction
- Challenges
- Solutions
- Tests, CI, Collaboration, Releases

# Core functionalities

- Login and authentication
- Dashboard
- Contract management
- Bill management
- Payment
- Store locator
- Etc …



# Differences between countries

- Backend system
- Login and authentication
- Payment
- Different type of contracts
- Localization
- Etc …

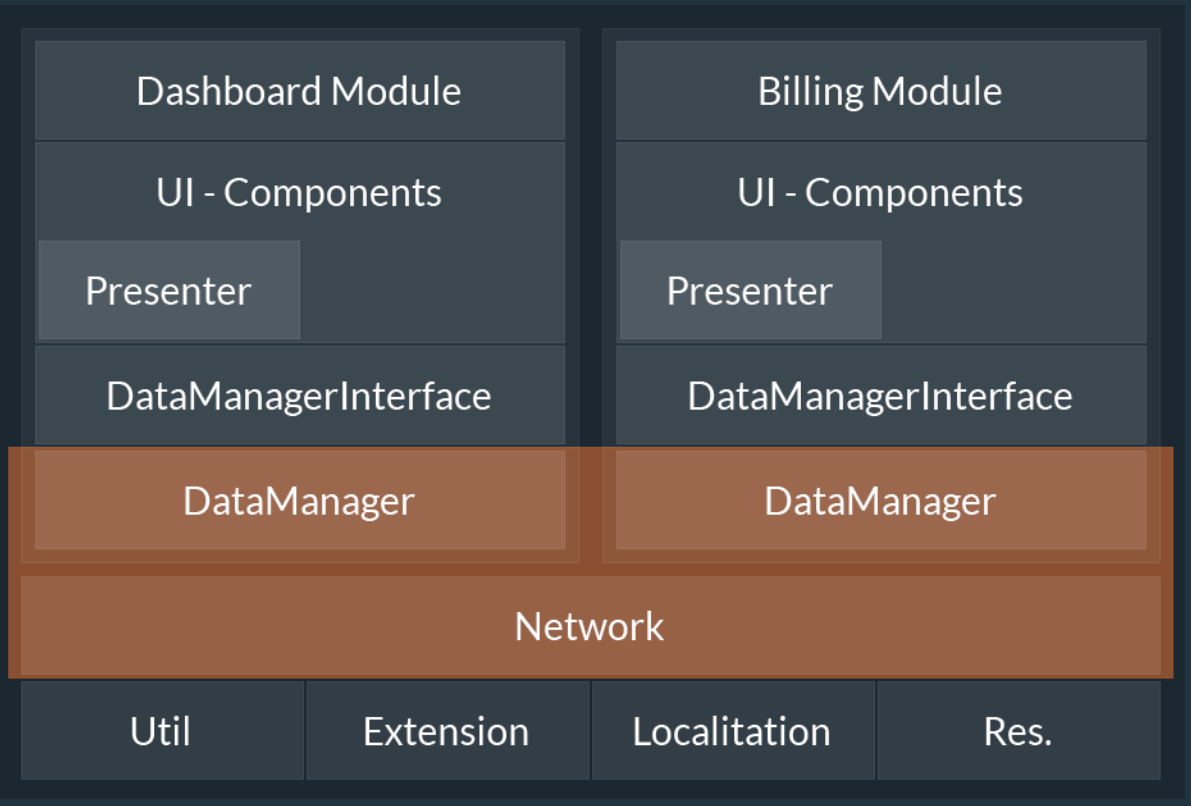
We need to handle any customization request

# What do we need to customize?

- User Interface
  - Whole screen-flow
  - One screen
  - Part of a screen
- Integration
  - Network communication
  - Business logic
- Localization
  - Language
  - Date formatting
  - Unit formatting

What architecture should we choose?

# Architecture




## Core

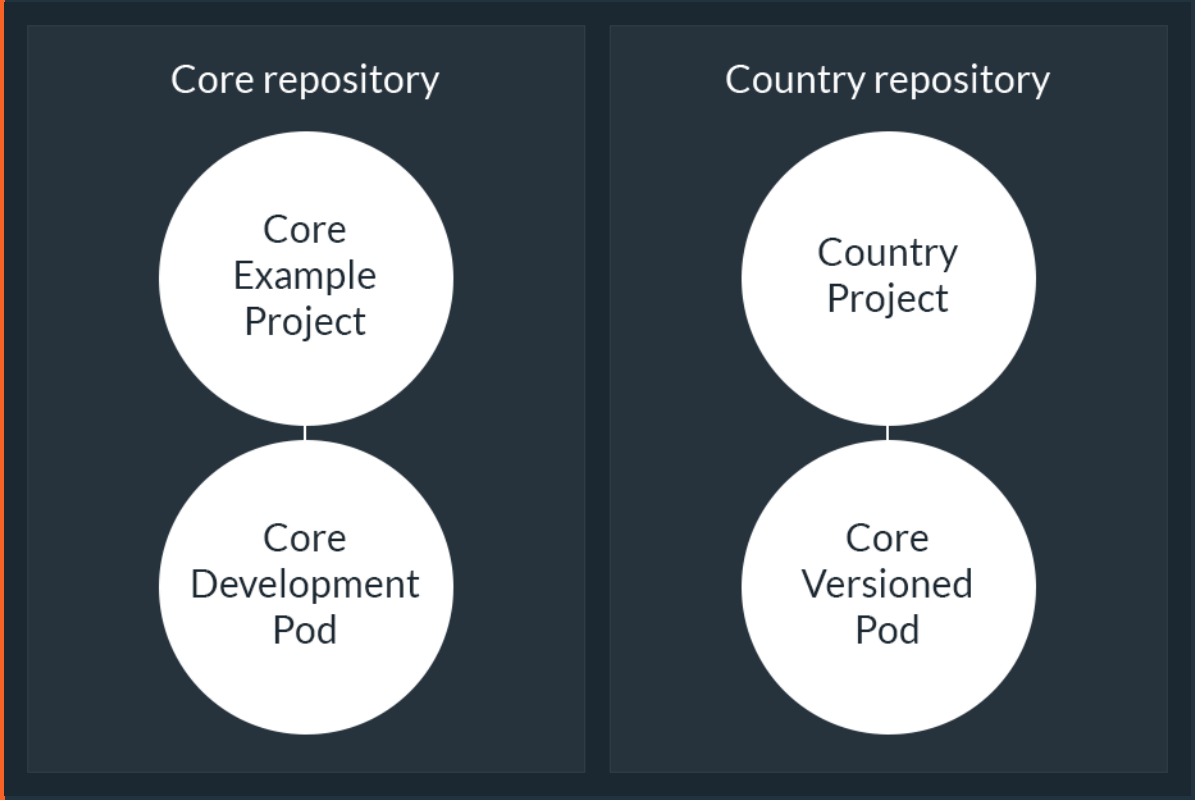
- UI – screens, components
- Presenter – view models
- Data manager Interface – business models
- Utils, Extensions, Localization, Resources

## Country specific

- Data managers
- Networking - API Models
- Customized UI, presenters, etc …

How should we structure our projects?

# Project structure




## Core

- Source code
- Xibs
- Resources

## Country specific

- Customized source codes
- Configuration file
- Assembly definitions
- Language files
- Feature flags
- Resources

How to replace any object of an application for specific countries?

# Dependency Injection

- iOS: Typhoon
- Assemblies for each module
- Move every object initialization to assemblies
- Assemblies can be overridden in the country specific implementation project
- Automatic injection on code level

```
/**
 Base of billing components. It provides datamanager, assembly, etc
 */
@interface ULTBillingComponentBase : ULTComponentView

@property (weak, nonatomic) InjectedProtocol(ULTBillingDataManagerInterface) billingDataManager;
@property (weak, nonatomic) InjectedClass(ULTBillingAssembly) billingAssembly;
@property (weak, nonatomic) InjectedProtocol(ULTInputValidatorInterface) inputValidator;

@end
```

# How to customize different parts of a screen?

# View architecture


- Screens are built up by reusable, independent Components
- Components are placed inside ComponentCollections
- Aligned with constraints inside the collections
- Width is defined by the screen's width
- Height is defined by it's inner constraints

# Components

- A screen is built up by a collection of components
- Components can be inserted, deleted or replaced
- Independent from each other
- Communication through their data manager

```
@implementation ULTBillingLineChartViewController

- (void)viewDidLoad
{
    [super viewDidLoad];

    self.title = NSLocalizedString(@"billing.bill_details.bill_details.statistics", nil);
    self.componentCollection.backgroundColor = [UIColor primaryGray2];

    [self.componentCollection registerComponent:[self.billingAssembly billingLineChartComponent]];
    [self.componentCollection registerComponent:[self.billingAssembly billingLineChartCostsComponent]];
}

@end
```

# How to select core features for country specific applications?

# Business feature flags

- Defined in a property list
- Every core feature that needs to be turned off gets one
- Unlike simple Feature flags used for reducing deployment risk these are permanent in your code
- Keep feature flags on the UI to minimize conditional logic

```
if ([self.featureFlagger isFamilyNfriendsEnabled])
{
    [self.componentCollection registerComponent:[self.serviceAssembly familyNfriendsComponent]];
}
```

# Best practices

- Do not mark core code as private
  - All IBOutlet should be public
  - All Component attribute should be public
- Staging / Production environments with Build Schemes
- Configuration variables in a property list (SCConfiguration) for each environment

# Testing

- Snapshot tests for core components (FBSnapshotTestCase)
- Snapshot tests for feature flags!

```
it(@"change plan disabled, reconnect disabled", ^{
    [corePatcher patchDefinitionWithSelector:@selector(featureFlagger) withObject:^id{
        ULTFeatureFlagger* featureFlagger = OCMClassMock([ULTFeatureFlagger class]);

        OCMStub([featureFlagger isManageServicesChangePlanEnabled]).andReturn(false);
        OCMStub([featureFlagger isManageServicesReconnectEnabled]).andReturn(false);

        return featureFlagger;
    }];

    [coreAssembly attachPostProcessor:corePatcher];

    ULTComponentCollectionView *collection = [ULTComponentCollectionView new];
    collection.frame = CGRectMake(0, 0, 320.0, 568.0);
    ULTCustomerPlanDetailsComponent *view = [coreAssembly.serviceAssembly planDetailsComponent];

    [collection registerComponent:view];

    expect(collection).to.haveValidSnapshotNamed(@"ULTCustomerPlanDetailsComponent");
});
```

# Testing

- Snapshot tests for core components (FBSnapshotTestCase)
- Snapshot tests for feature flags!
- Unit tests for country specific integration (NSURLConnectionVCR or Nocilla)

```
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);
    });
});
```

# Continous Integration

- Create a MR
- Merge to the dev branch
- Run ocstyle, OCLint
- Run tests
- Release to Fabric
- Mark the MR as ready to merge

# Collaboration, Versioning, Release

- 2 teams are working on the core per mobile platform
- Communication via Slack
- Collaboration on core through Gitlab with MRs
- Changelog
- Documentation for core features
- Core is versioned with semantic versioning
- Production releases are handled separately by the teams

# Summary

- Simple, flexible architecture
- Flexible UI
- Continuous integration
- Communication is key

# 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
- http://semver.org/

# Thanks for your attention!

## Questions?


David Kovacs

Supercharge

CTO

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

# Supercharge