Writing Testable ViewModels

Thomas Claudius Huber
MICROSOFT MVP (WINDOWS DEVELOPMENT)

@thomasclaudiush www.thomasclaudiushuber.com

Module Outline

Dependencies of a ViewModel

Abstract away dependencies with Interfaces

Mock dependencies in unit tests

Use a dependency injection framework

Dependencies of a ViewModel

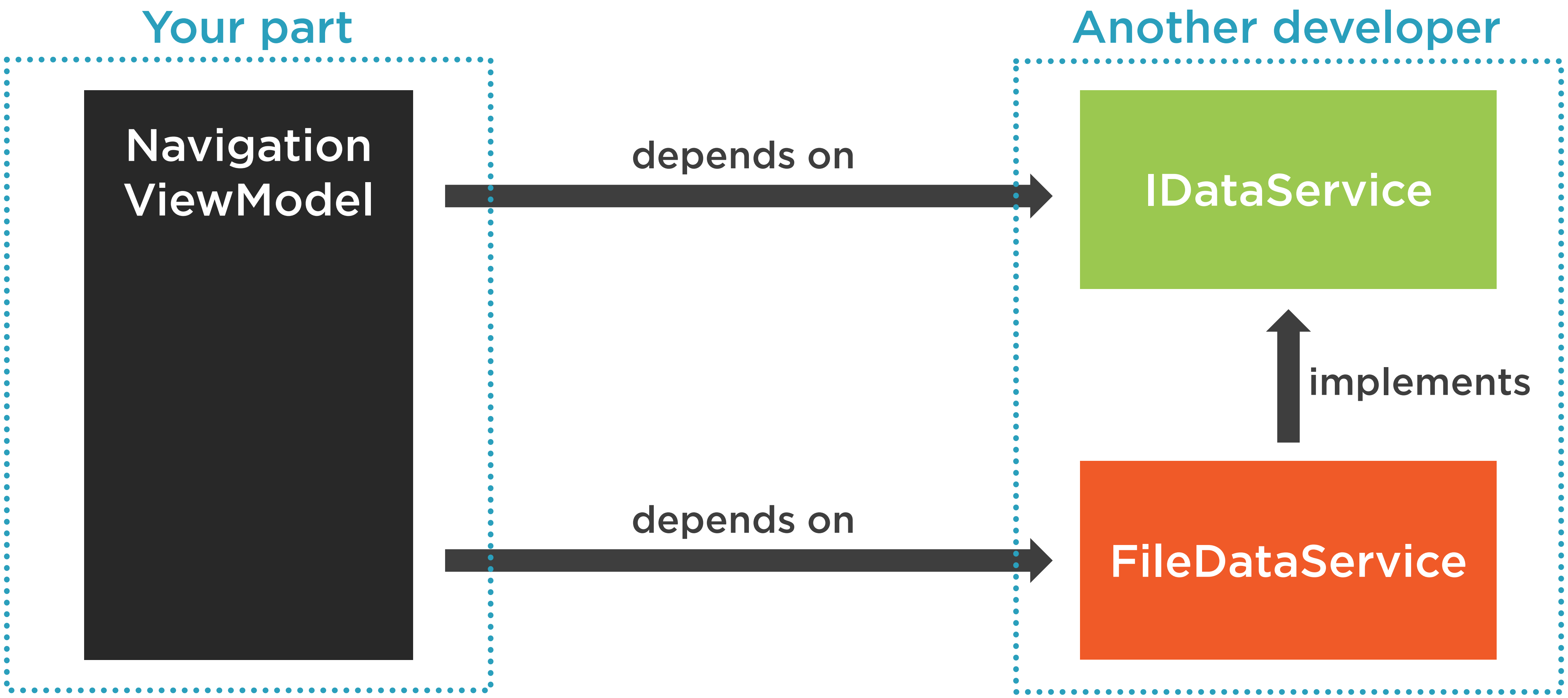
Data access

Dialogs

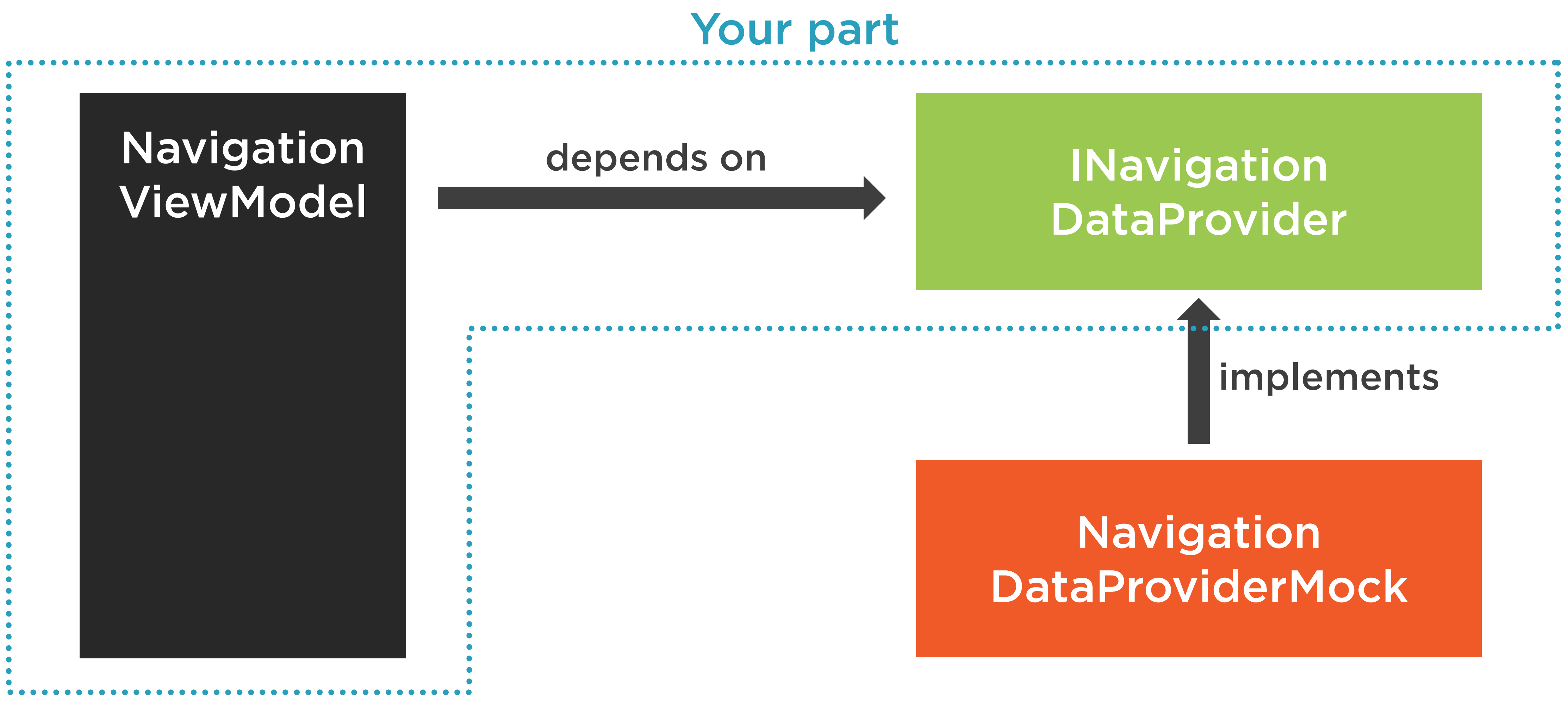
Event aggregator

Other ViewModels

Abstract Away Dependencies with Interfaces



Abstract Away Dependencies with Interfaces



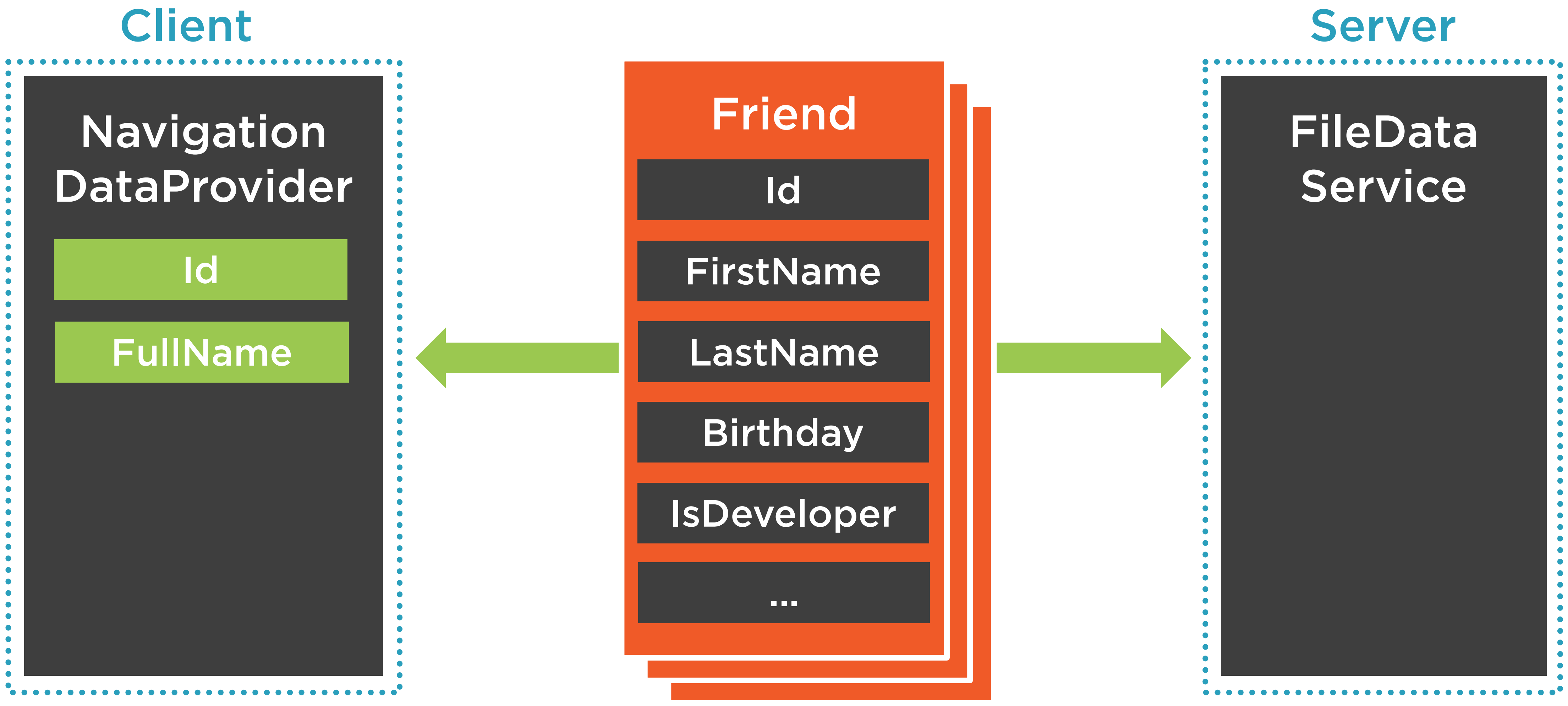
Demo

Introduce the INavigationDataProvider-interface

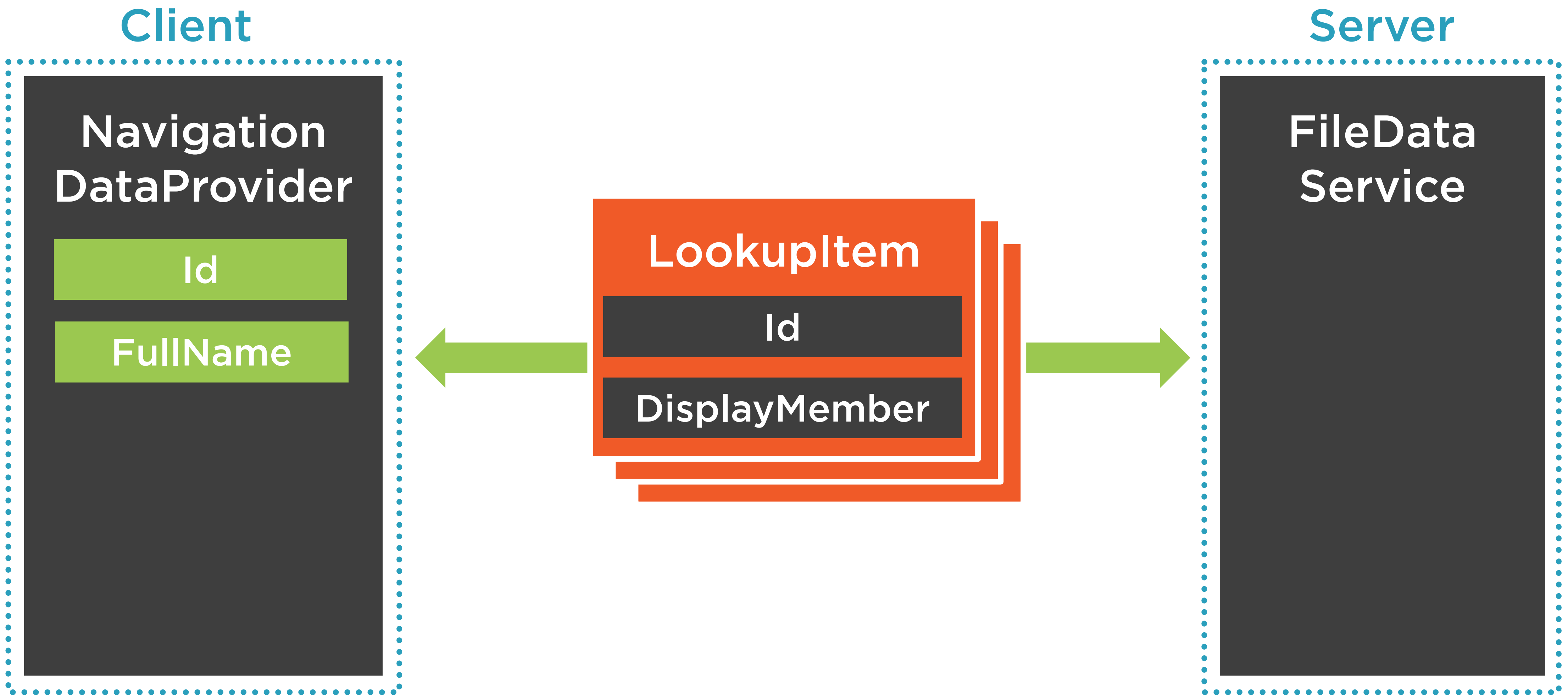
Write unit tests with a NavigationDataProviderMock

Implement the production NavigationDataProvider

Optimize the Code for Performance



Optimize the Code for Performance



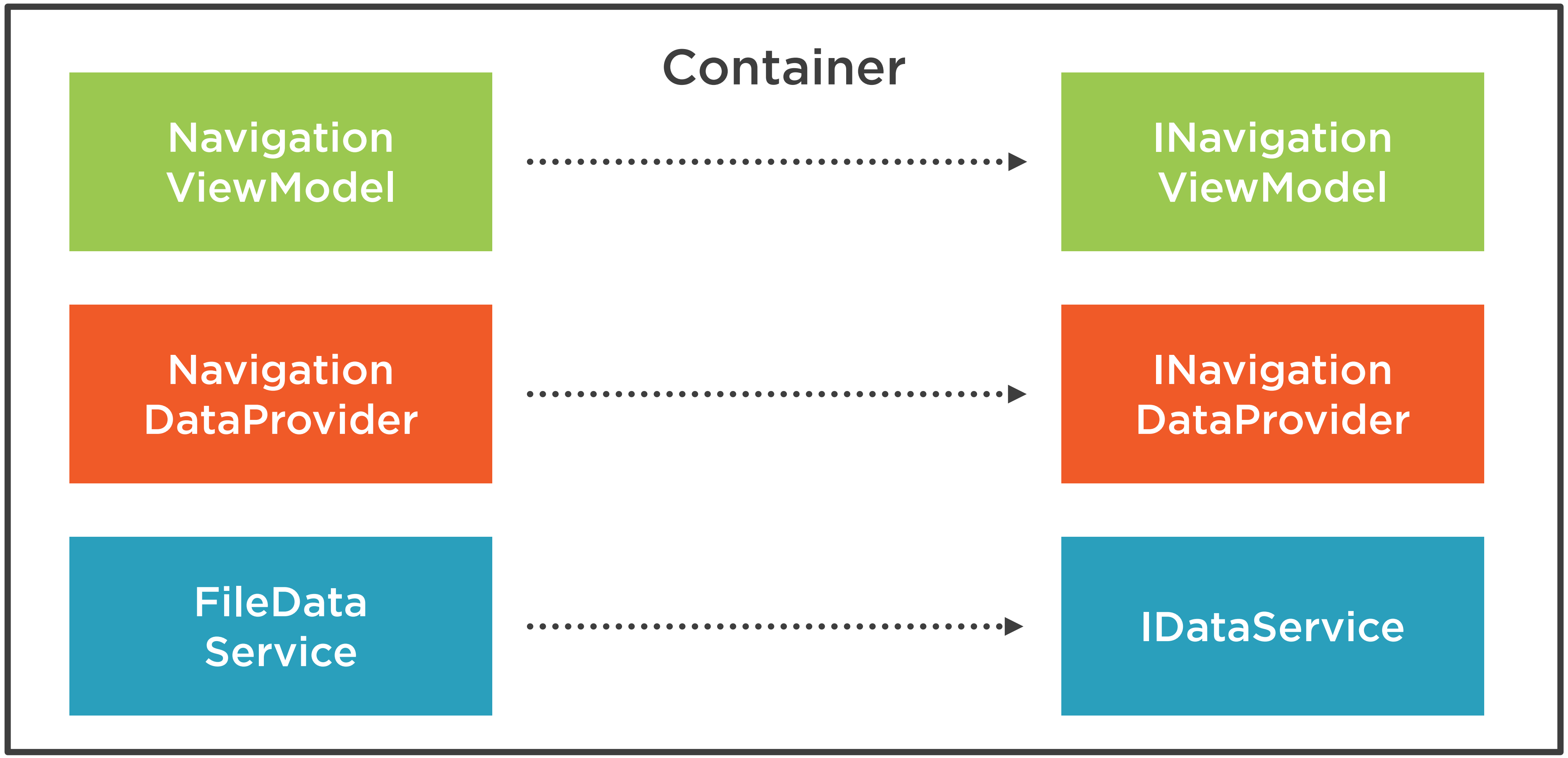
Demo

Test the MainViewModel’s Load-method

Dependency Injection Basics

```
var mainWindow = new MainWindow(
  new MainViewModel(
    new NavigationViewModel(
      new NavigationDataProvider(
        () => new FileDataService()))));
```

Dependency Injection Basics



Dependency Injection Basics

```
var mainWindow = new MainWindow(
  new MainViewModel(
    new NavigationViewModel(
      new NavigationDataProvider(
        () => new FileDataService()))));
```

Dependency Injection Basics

```
var mainWindow =
```

Dependency Injection Basics

```
var mainWindow = container.Resolve<MainWindow>();
```

Popular Dependency Injection Frameworks

Unity	Castle.Windsor	StructureMap
Spring.NET	Ninject	Autofac

Summary

Writing testable ViewModels means abstracting away dependencies

A dependency injection framework can resolve dependencies for you

Creating a class for each required Mock-object is not really fun