

VIPER

A radical iOS architecture

About me

Apostolis, or just Tolis

- Background in linguistics and computational linguistics
- Python -> Android -> iOS
- I like to go to fan conventions with my two sons
- KnowTrail, PokeCalc @ App Store
- Jovian Outpost @ Spotify

About
eTraveli Group

Brands:

mytr!p

GO TO GATE

✓ flight network

flygresor.se

TripStack

PRECISION

75

Markets

35

Languages

30^M

Travelers

2500+

Employees

9

Main offices

8

Departments

Partnerships with:

Booking.com

Google Flights

Skyscanner

KAYAK

Stockholm, Uppsala & Gothenburg

Athens Toronto Krakow

Pune & Mumbai Montevideo

Way of Working

A robust company culture, evidenced
by **86%** of our staff affirming that they
embody our company values daily.



About the eTraveli apps

Gotogate, Mytrip, Flight Network

- Native apps on iOS & Android
- The iOS apps are written in Swift / SwiftUI
- They use VIPER
- 3 iOS developers
- 100% test coverage crucial due to many & complex product specs

What will follow

- VIPER basics
- VIPER in action
- Thoughts about VIPER
- Discussion

Traditional architectures

- The UIKit Apple-endorsed architecture is **MVC** (Model-View-Controller)
 - Model (manages data and business logic)
 - View (presents the user interface)
 - Controller (mediates between Model and View).
- There is also **MVVM** with an extra layer between code and view, the ViewModel

But...

- **Massive View Controllers** accumulating too much logic making it difficult to manage and test.
- **Unclear Separation of Concerns** with overlapping of responsibilities can lead to code that's hard to maintain.
- **Problematic test coverage** in MVC because of ViewController & in MVVM because of either data binding in the ViewModel or navigations still handled by the View.

Enter VIPER

The history

- It all started with the article *Architecting iOS Apps with VIPER* ↓ by *Mutual Mobile* (2014)



- *VIPER* is an acronym

What does V stand for?

V for View (Controller)

- The ViewController is stripped away of any logic. It gets downgraded to an as dumb as possible View
- The View does not "know" what it presents. It trusts blindly what is given to it.
- The goal is to never have to write tests about the View.

What does I stand for?

I for Interactor

- Most of business logic is housed here.
- It is independent of the View
- 100% test coverage is crucial

What does P stand for?

P for Presenter

- Connecting the Interactor with the View.
- Converts raw data to a format understood by the View
- 100% test coverage is crucial

What does E stand for?

E for Entity

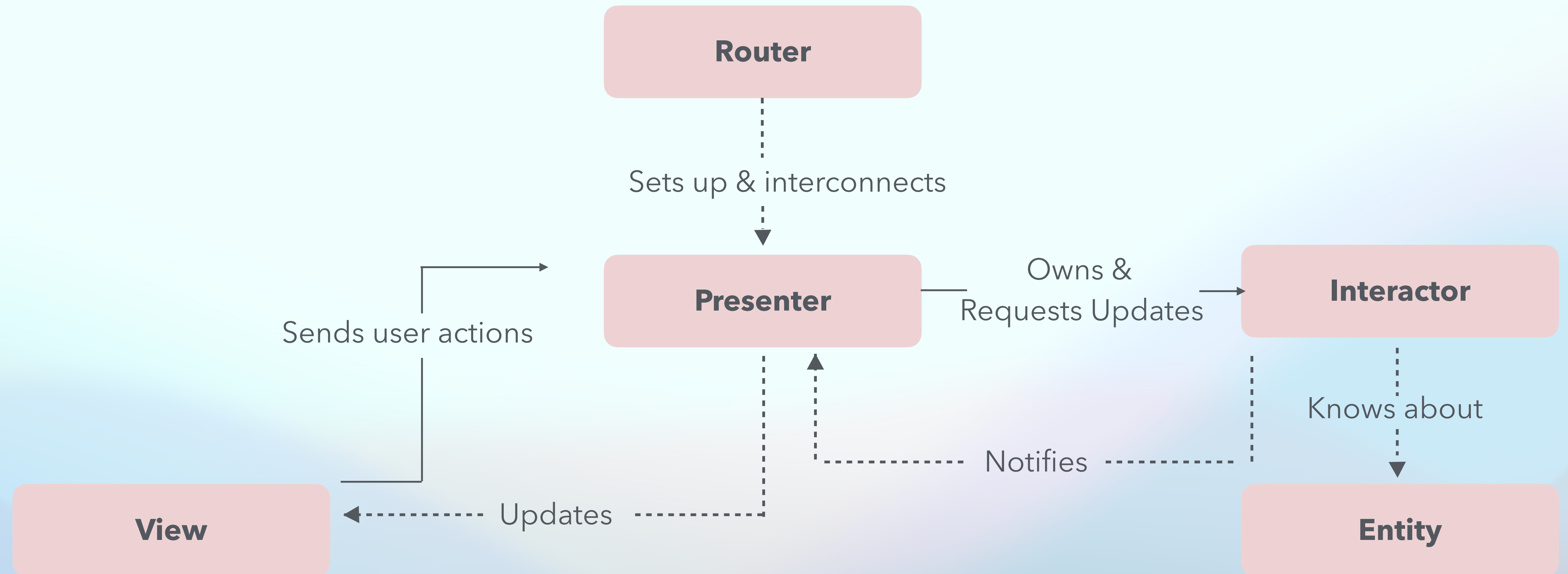
- Minimal logic
- Intended to be used only by Interactors
- Is not bound to a specific screen but can be shared by many

What does R stand for?

R for Router

- Sets up each component and its relation to the other components
- Replaces segues and wraps UIKit transitions
- Makes all transitions testable & scalable

Overview



Code Walkthrough

Are Presenters worth it?

Yes, mostly

- They increase testability
- Independent of the View structure but in reality, when something changes in the architecture of the View, the Presenter remains rarely unaffected
- Skip them totally when nothing is dynamic in the View

Are Interactors worth it?

Y-E-S

- Separation of concerns starts from here
- Enable 100% test coverage
- Countless cases where you are certain that after a “small” code change your app will work the same but the tests fail.

Are Routers worth it?

Surprisingly yes

- Screen transitions become flexible and 100% testable
- Separation of concerns becomes even more apparent. The implementation details of how the next screen will be displayed are hidden from the View
- Reduce Storyboard merge conflicts

When is VIPER not worth it?

It depends

- When you are sure that your app will not need to scale
- When there are tight time constraints
- When you love resolving merge conflicts in Storyboard code
- When you are the sole developer (?)
- When your app is SwiftUI-only (?)

What about performance?

Same - same

- **Runtime:** Insignificant difference for our apps and probably any app except graphic-heavy ones
- **Compile time:** A little more than MVC (according to research ↓) but is it enough as an argument against VIPER?



What about the learning curve?

Steep.

- Takes some time to get used to create all the components from scratch
- Separation of concerns is always a challenge
- Debugging can be a little slower in the beginning (but faster in the long-term)
- Requires a will to learn to implement workarounds where UIKit makes it hard for VIPER

HOW LONG CAN YOU WORK ON MAKING A ROUTINE TASK MORE
EFFICIENT BEFORE YOU'RE SPENDING MORE TIME THAN YOU SAVE?
(ACROSS FIVE YEARS)

		HOW OFTEN YOU DO THE TASK					
		50/DAY	5/DAY	DAILY	WEEKLY	MONTHLY	YEARLY
HOW MUCH TIME YOU SHAVE OFF	1 SECOND	 DAY	2 HOURS	30 MINUTES	4 MINUTES	1 MINUTE	5 SECONDS
	5 SECONDS	 DAYS	12 HOURS	2 HOURS	21 MINUTES	5 MINUTES	25 SECONDS
	30 SECONDS	 4 WEEKS	 DAYS	12 HOURS	2 HOURS	30 MINUTES	2 MINUTES
	1 MINUTE	 8 WEEKS	 DAYS	 DAY	4 HOURS	1 HOUR	5 MINUTES
	5 MINUTES	9 MONTHS	 4 WEEKS	 DAYS	21 HOURS	5 HOURS	25 MINUTES
	30 MINUTES		6 MONTHS	 5 WEEKS	 DAYS	 DAY	2 HOURS
	1 HOUR		10 MONTHS	2 MONTHS	 DAYS	 DAYS	5 HOURS
	6 HOURS				2 MONTHS	 2 WEEKS	 DAY
	 DAY					 8 WEEKS	 DAYS

Conclusion

Follow your heart

- Not so important to adopt every detail of VIPER
- More important to get into the mentality of separating concerns
- Depending on your project & your needs you decide if VIPER or parts of it are worth your time and effort

VIPER vs ...

Discussion