VIPER-B

Evaluating clean architecture hype in real app.

WARNING

Talk may carry **strong** over-architecting symptoms.

Please proceed with *caution*!

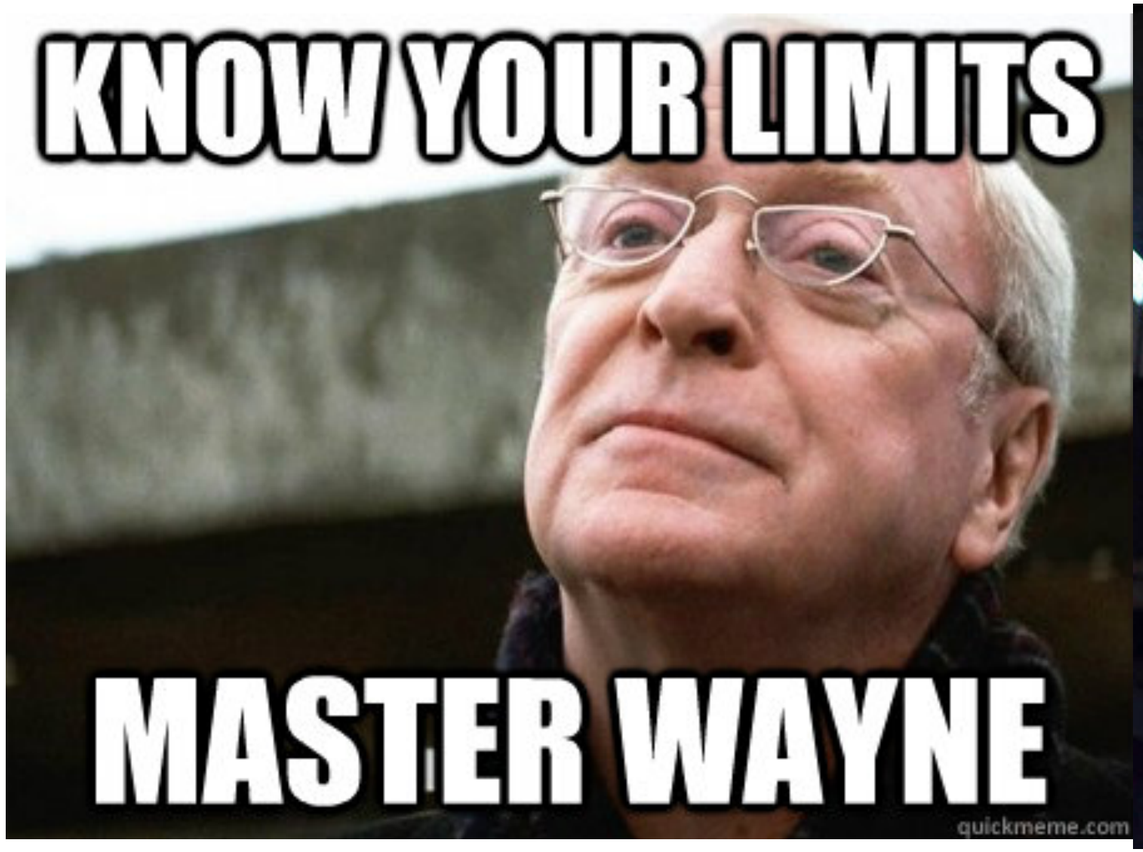
- Not necessary an expert view, just average dev tryout
- Just another Viper implementation agreements
- Consciously taken to the extreme
- Promised a friend to share an experience one day... (celeryman)

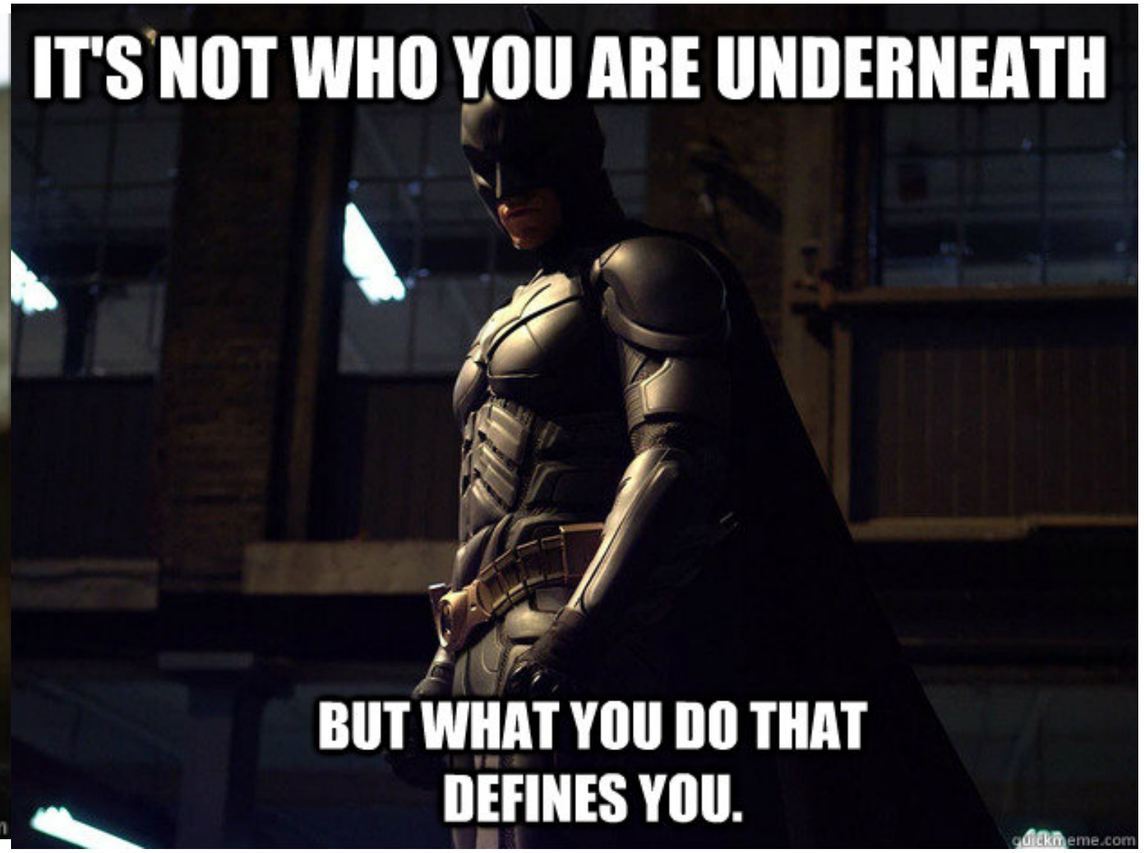
Genesis

- New, big and well project when Swift 3.0 was in beta
- Disappointed in MVC, MVVM and MVVM-C a bit
- Let's try something new...

Inspired by...

BATMAN


KNOW YOUR LIMITS
MASTER WAYNE
quickmeme.com


IT'S NOT WHO YOU ARE UNDERNEATH
BUT WHAT YOU DO THAT
DEFINES YOU.

... and Uncle Bob

Nicola Zaghini

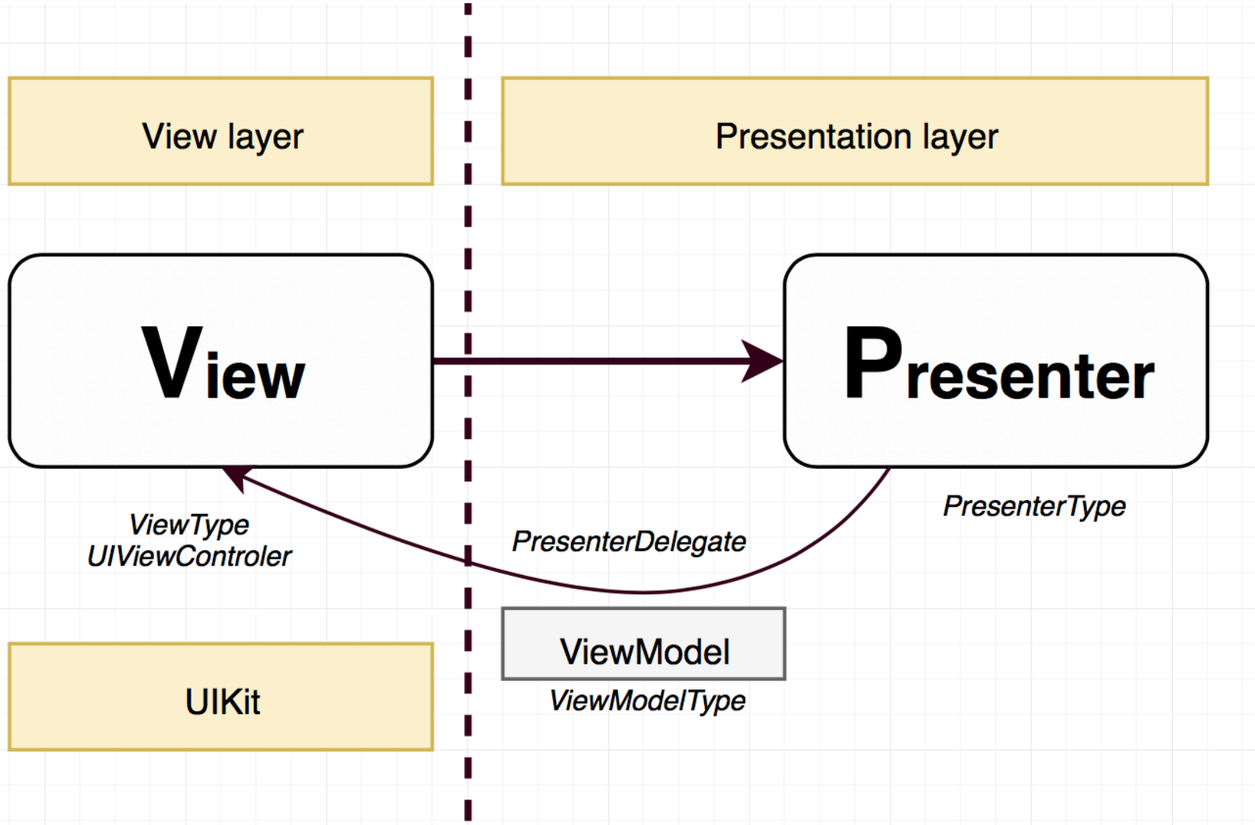
!!! Must watch video:

Mastering reuse: A journey into application modularization with VIPER

Why Viper-B?

- Describes components well
- Well defined application layers: view, presentation, business, enterprise, routing
- Clear responsibilities and boundaries
- Small components, resembling units

View & Presentation layer

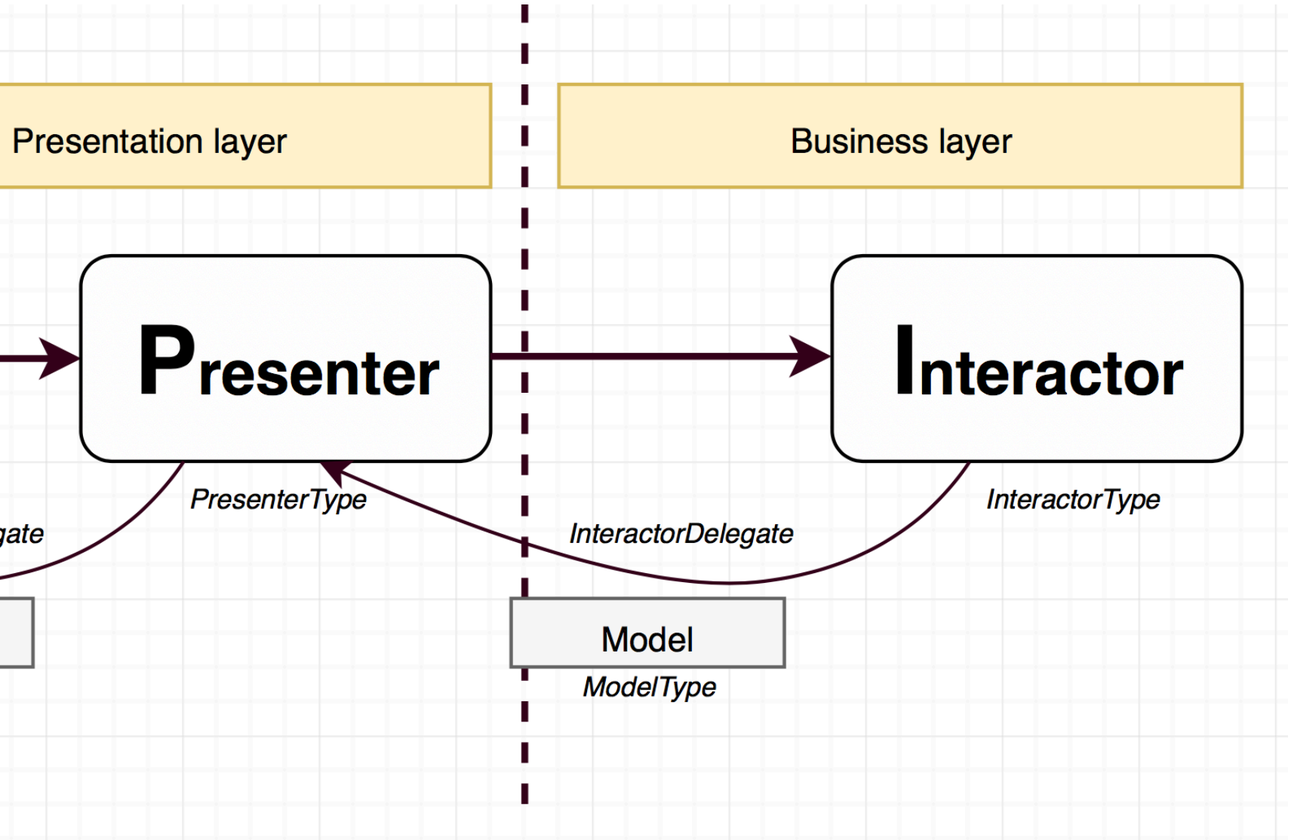


View:
- Usually UIViewController subclass
- Simplest `UIKit` code (similar to other patterns, like MVVM)
- Displays view model delivered by presenter (weak delegate)
- Delegates all UI events to presenter

Presenter:
- Delivers **ViewModel** to the **VIEW**
- Handles events delivered from the **VIEW**
- An adapter between business logic **INTERACTOR** and UI (**VIEW**)
- Converts **Models** to something displayable (**ViewModel**) and delivers it to the delegate (**VIEW**)
- May call **ROUTER** when decides that view events need to be handled externally
- No `UIKit`

View model:
- Data model generated by **PRESENTER**
- Describing UI state for the **VIEW**
- Simple property types (`String`, `Bool`, `enum`)
- No `UIKit`

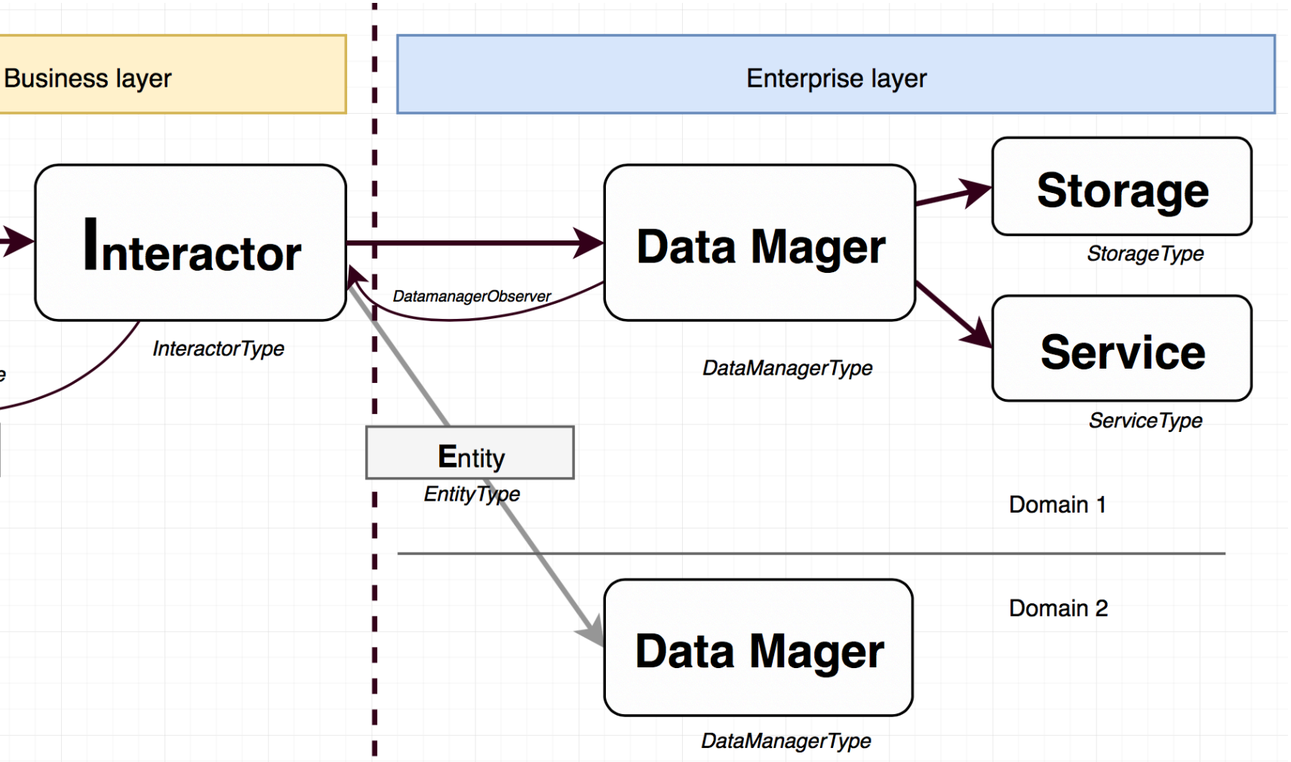
Business layer



Model
- Encapsulates data model of results coming from **INTERACTOR**.
- Contains raw properties that are interesting for the use case.
- Not suitable for display (presenter's job).

Interactor
- A PONSO encapsulating business rules for specific use case.
- Transforms entities into models.
- Knows what tasks needs to be carried out before returning stuff to the presenter.
- Does not care where data comes from, uses data manager(s) for this.
- Does not care about presentation.
- Can subscribe to events from a data managers
- Api should have very specific methods to meet business tasks.

Enterprise layer



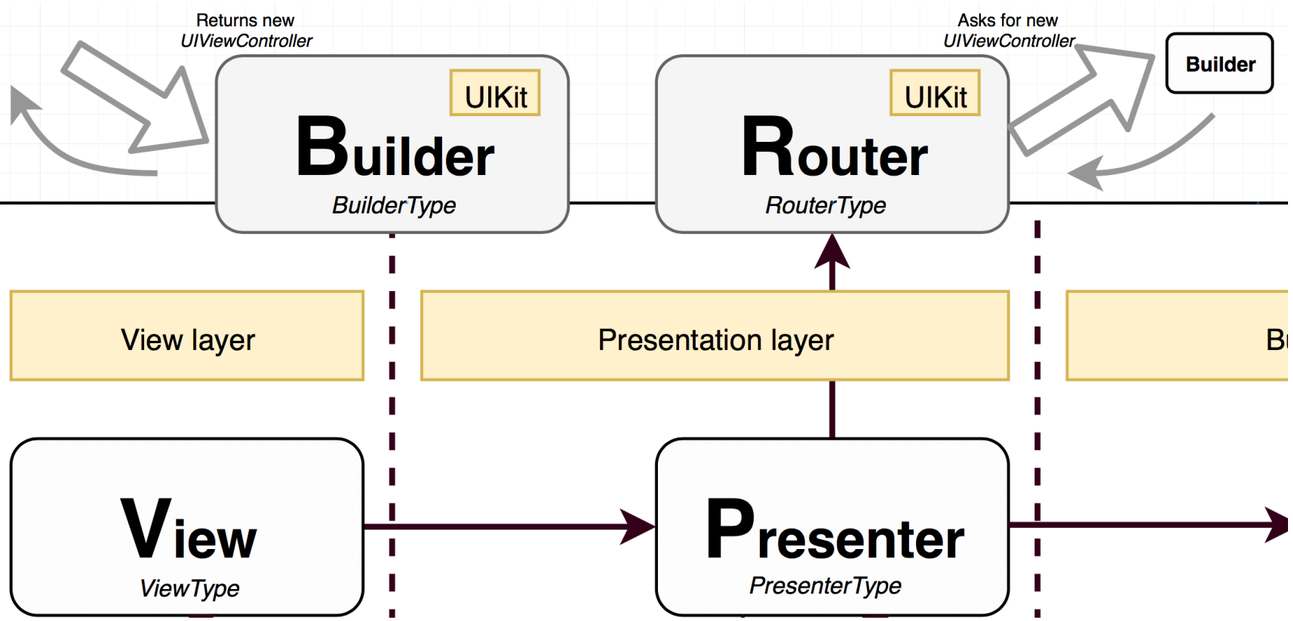
Entity
- Enterprise data model.
- Not application specific (e.g. coming from external database scheme or web service)
- Primitives / database-like properties (Int, String)

Data Manager (entity Gateway)
- Encapsulates an API to the enterprise layer (return entities).
- Grouped around domains, e.g. Hero, Image, User
- The data manager knows where to fetch data from and knows if something should be persisted
- Does not know the underlaying technologies used of storage or service.
- May have multiple subscribed interactors
- Longer life-span

Service
- Service to get data from slower resources (network)
- Operates on entities

Storage
- Faster, local storage or cache (e.g. NSUserDefaults, CoreData etc)
- Api should be simple and fast.

Builder & Router



Builder:

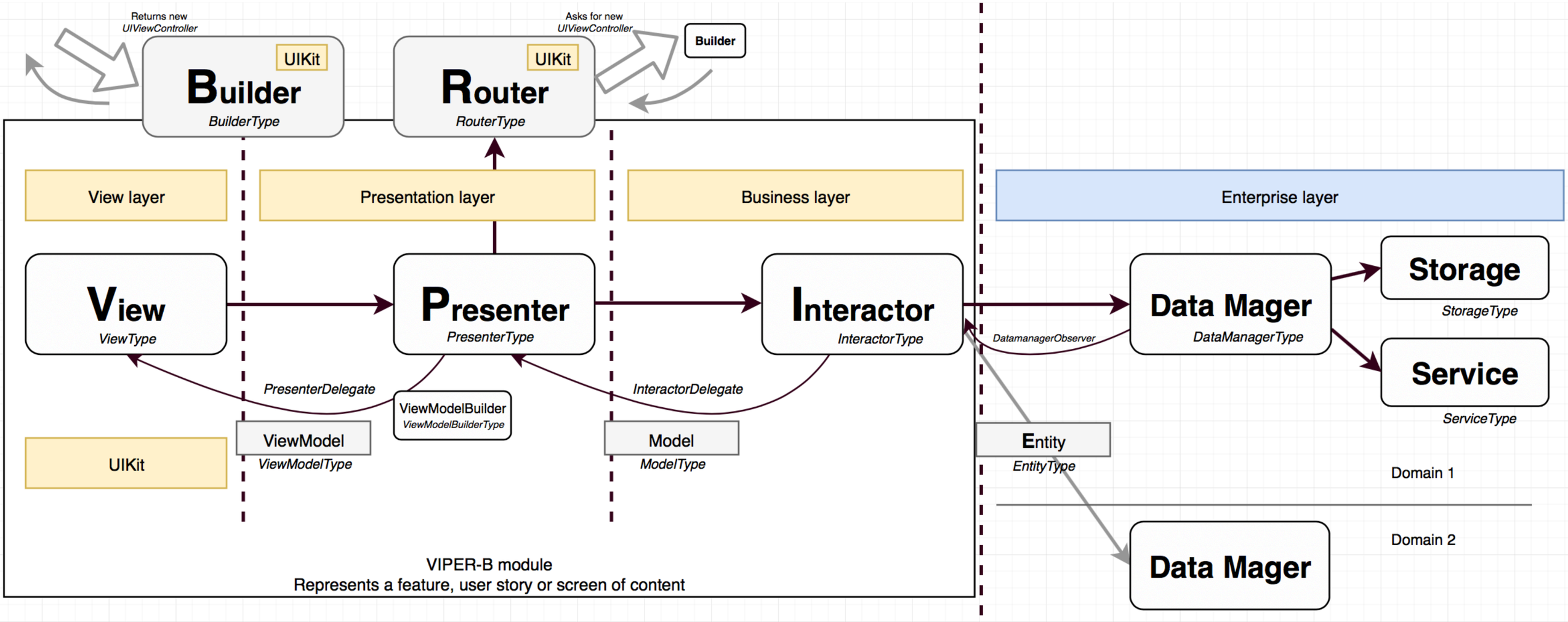
- An ENTRY point to the module.
- Responsible for creating and wiring up Viper module.
- Produces and returns UIViewController retaining the module.
- Must contain at least one building method.

Router:

- An EXIT point from the module.
- Takes care of routing from one module to another.

Ready for the full picture?

...you've been warned!


Returns new
UIViewController
Builder
UIKit
BuilderType
Asks for new
UIViewController
Builder
UIKit
Router
RouterType
View layer
Presentation layer
Business layer
Enterprise layer
View
ViewType
Presenter
PresenterType
Interactor
InteractorType
Data Mager
DataManagerType
Storage
StorageType
Service
ServiceType
DatamanagerObserver
PresenterDelegate
InteractorDelegate
ViewModelBuilder
ViewModelBuilderType
ViewModel
ViewModelType
Model
ModelType
Entity
EntityType
UIKit
Domain 1
Domain 2
Data Mager
VIPER-B module
Represents a feature, user story or screen of content

I know...

Example app

Pros:

- Testable
- Small and specialised classes
- Clear boundaries between layers and data types
- Represents data flow steps and transformation very well
- Promotes composition & protocol oriented design
- Promotes structured code
- Easy to work in teams (not many reasons to change one file)

Pros cd...

- Forces you take care about naming conventions (so many files and types ;-)
- Teaches discipline (every time I broke pattern, I had issues with testing)
- Teaches being careful about importing 3rd party libraries
- Gets somehow natural in a while
- Fun (if you like LEGO blocks)
- No singletons!

Cons:

- Ceremony: lots of files and boiler-plate code
- Overkill for some projects (prototyping, etc..)
- Requires well defined requirements up-front.
- Mentally challenging to grasp whole module and track data flow
- Lots of data model conversions
- Difficult to stay clean and reuse the code at the same time

Cons cd...

- Pain to refactor
- Pain when developing against iOS frameworks (UIKit, Core Data)
- Interactors turned out to be not that reusable after all
- Sometimes feels like a dog chasing its tail....
- Many files
- Changes you...

Subjective conclusions:

- A bazooka!
- Definitely worth trying but use with caution
- Don't do it alone!
- Don't get pulled too much into this, use as a guide
- Consider cutting corners (compromise on layer separation)
- Just be SOLID
- Try functional approach?

Technical issues:

- Swift generics: circular reference issue, syntax, concrete types requirements
- Equatable / Self requirement protocol issues (hacked by type erasure).
- Swift not that'protocol-oriented' after all

References

Example app GitHub repo:
https://github.com/unwire/viper-demo-ios.git

Mastering reuse: A journey into application modularization with VIPER
The Principles of Clean Architecture by Uncle Bob Martin
Brigade’s Experience Using an MVC Alternative
Architecting Apps with (B)VIPER Modules
Architecting iOS Apps with VIPER
iOS Architecture Patterns
#8 VIPER to be or not to be?