

MVI

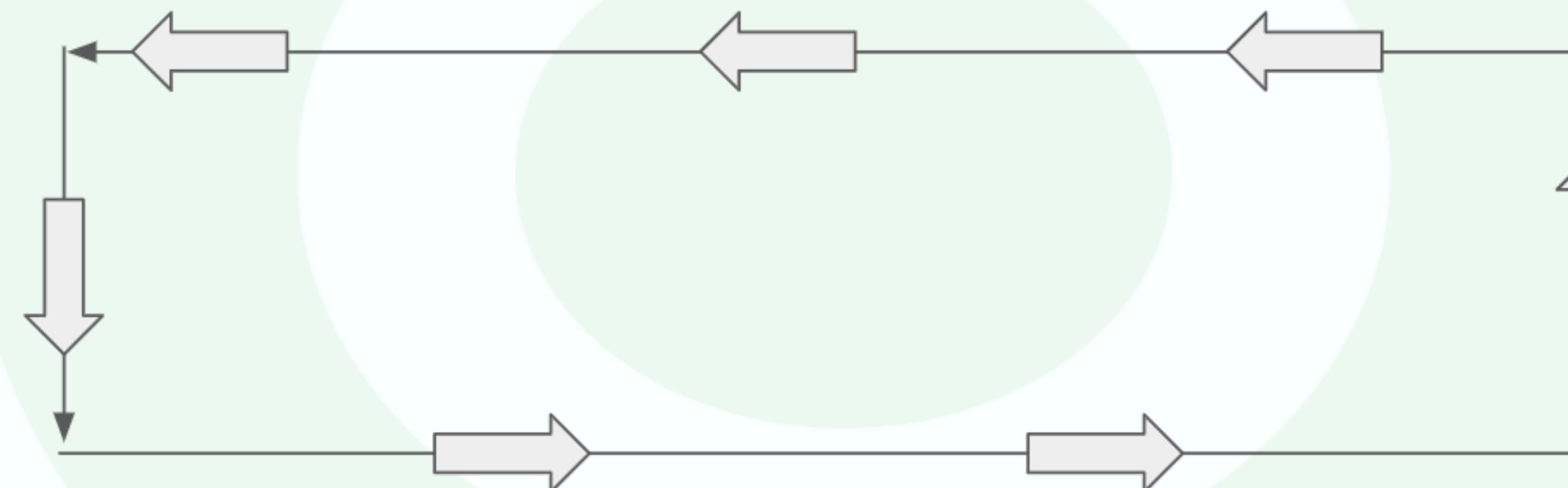
Model View Intent

Note (Intent = User Intention)

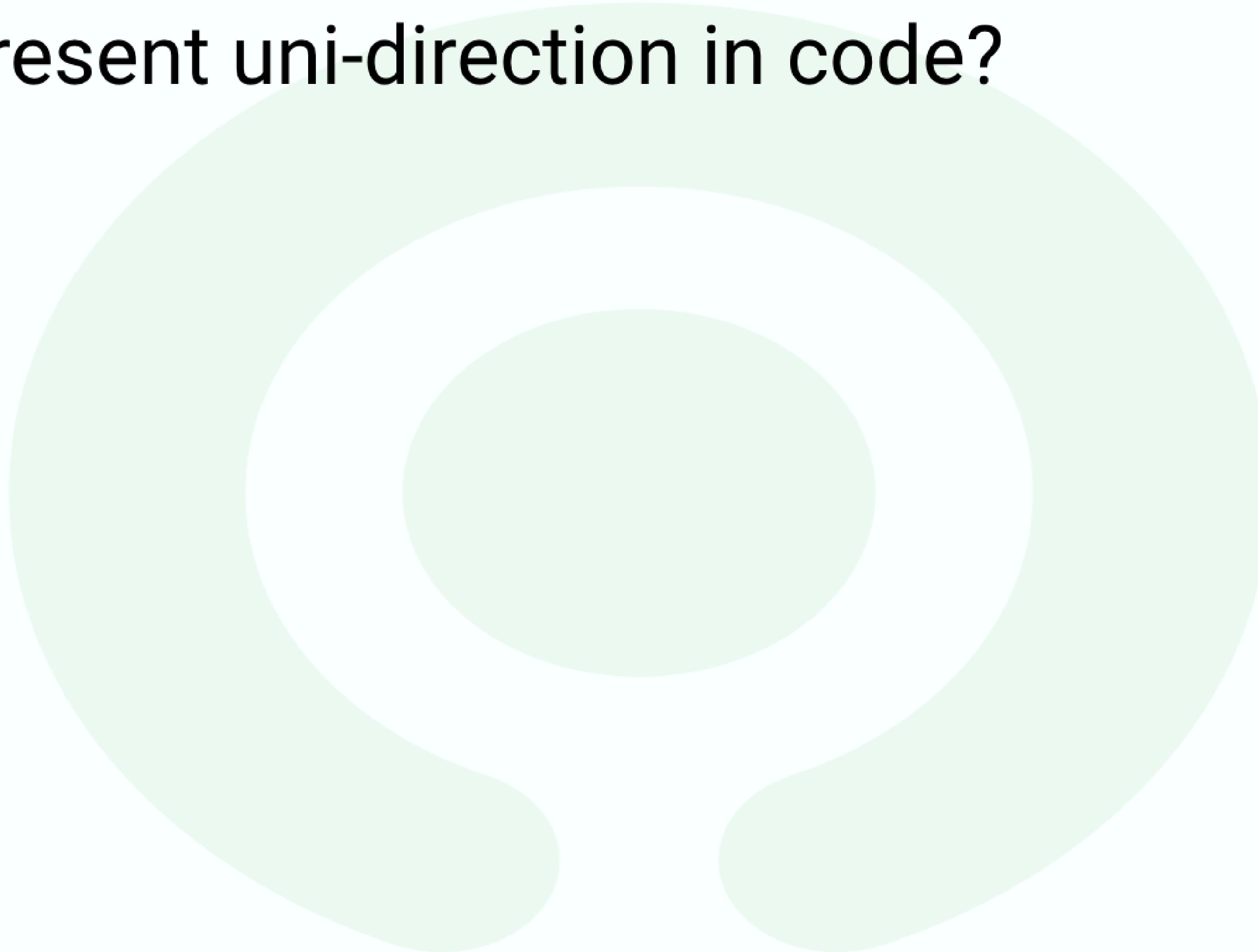
Goal of this talk

- What is MVI
- Why MVI
- Share knowledge in a way, so our team able to reduce learning curve time from two years to one week.

MVI is a unidirectional pattern (like Redux, cycle.js).



How represent uni-direction in code?



How represent uni-direction in code?

```
fun methodA(a: Int): String = "A"
```

How represent uni-direction in code?

```
fun methodA(@ Int): String = "A"  
fun methodB(@ String): Float = 1.1f
```

How represent uni-direction in code?

```
fun methodA(a: Int): String = "A"  
          ^  
          Output  
  
fun methodB(s: String): Float = 1.1f  
          ^  
          Input
```

How represent uni-direction in code?

```
fun methodA(a: Int): String = "A"
```

```
fun methodB(s: String): Float = 1.1f
```

```
fun methodC(f: Float): Int = 1
```

How represent uni-direction in code?

```
fun methodA(a: Int): String = "A"  
fun methodB(s: String): Float = 1.1f  
fun methodC(f: Float): Int = 1  
methodA( a: 3 )
```

How represent uni-direction in code?

```
fun methodA(a: Int): String = "A"
```

```
fun methodB(s: String): Float = 1.1f
```

```
fun methodC(f: Float): Int = 1
```

```
methodB(methodA( a: 3 ))
```

How represent uni-direction in code?

```
fun methodA(a: Int): String = "A"
```

```
fun methodB(s: String): Float = 1.1f
```

```
fun methodC(f: Float): Int = 1
```

```
methodA(methodC(methodB(methodA( a: 3 ))))|
```

How represent uni-direction in code?

```
fun methodA(a: Int): String = "A"
```

```
fun methodB(s: String): Float = 1.1f
```

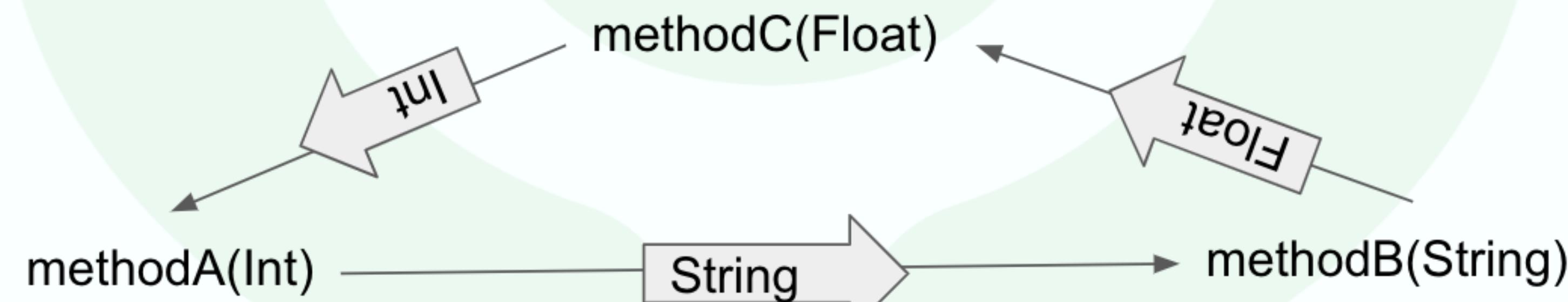
```
fun methodC(f: Float): Int = 1
```

```
methodB(methodA(methodC(methodB(methodA( a: 3)))))
```

Map code into drawing?

```
fun methodA(a: Int): String = "A"  
  
fun methodB(s: String): Float = 1.1f  
  
fun methodC(f: Float): Int = 1  
  
methodB(methodA(methodC(methodB(methodA( a: 3 )))))
```

readability



Tip to make our previous code readable using Kotlin

```
fun methodA(a: Int): String = "A"

fun methodB(s: String): Float = 1.1f

fun methodC(f: Float): Int = 1

methodA( a: 3)
    .let(::methodB)
    .let(::methodC)
    .let(::methodA)
    .let(::methodB)
    .let(::methodC)
```

```
fun methodA(a: Int): String = "A"

fun methodB(s: String): Float = 1.1f

fun methodC(f: Float): Int = 1

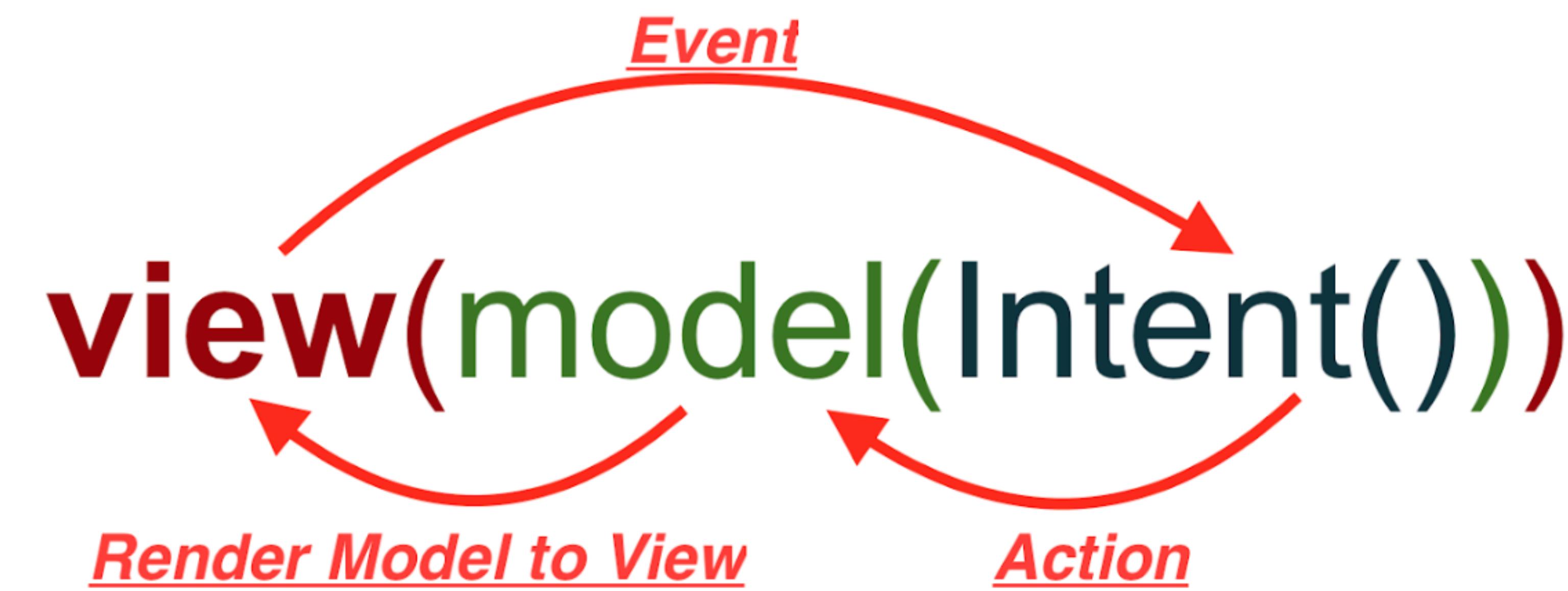
methodB(methodA(methodC(methodB(methodA( a: 3)))))
```

Until now, we know what uni-direction is and how to achieve uni-direction in code.

Back to MVI

MVI implementation be like VIM(View Intent Model)

view -> intent -> model

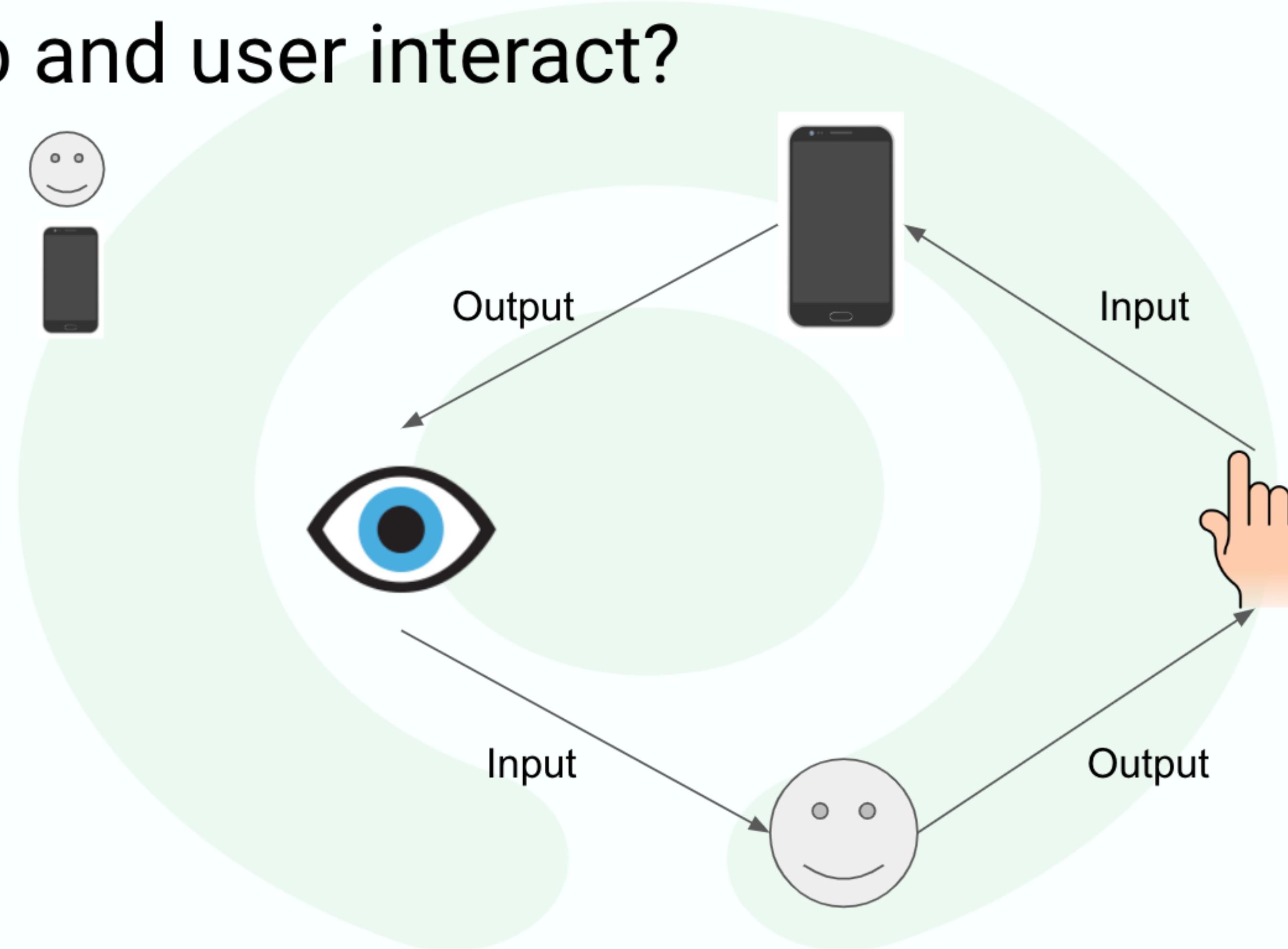


How app and user interact?

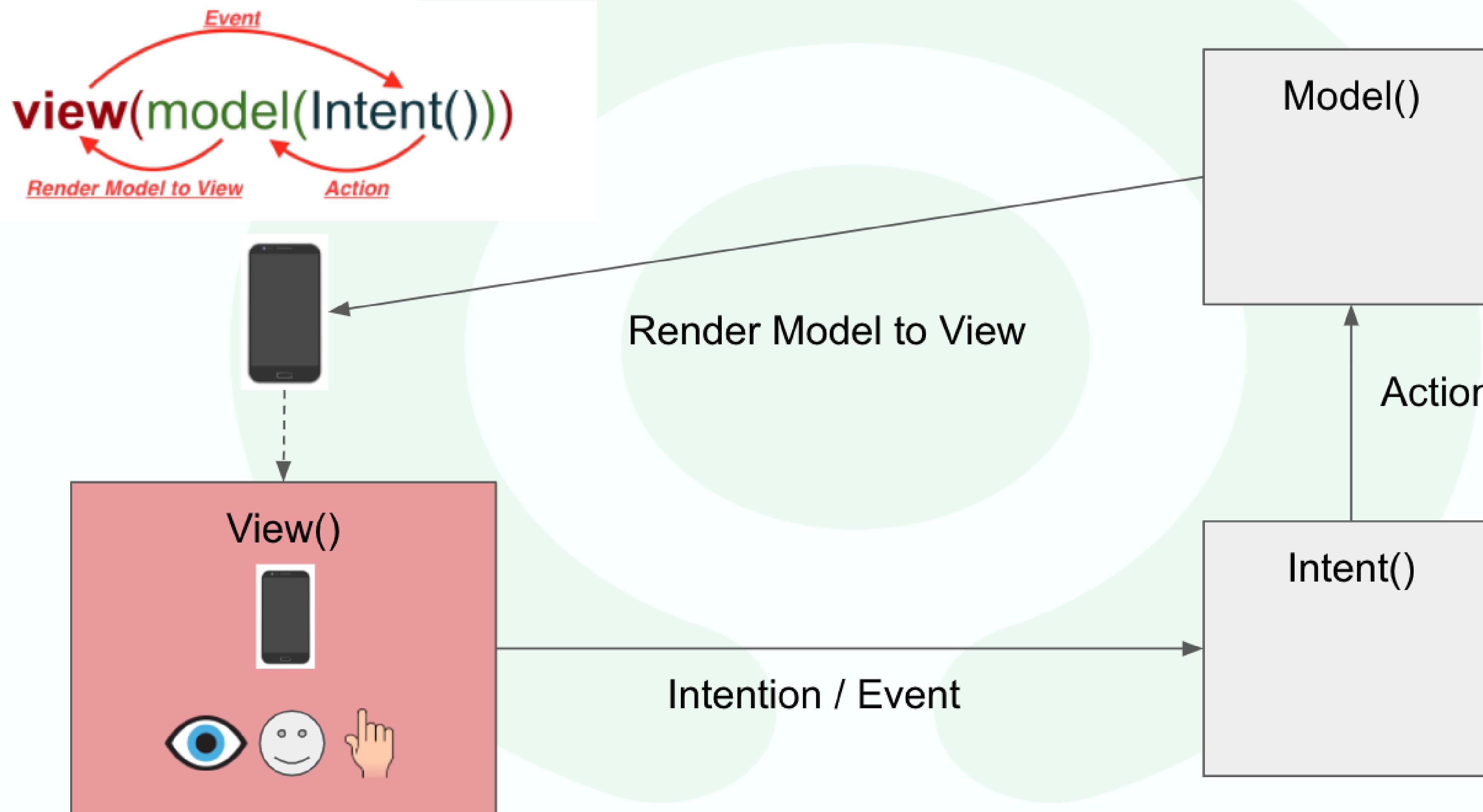
1. User



2. Device



Mapping User Interaction with MVI.



Recap

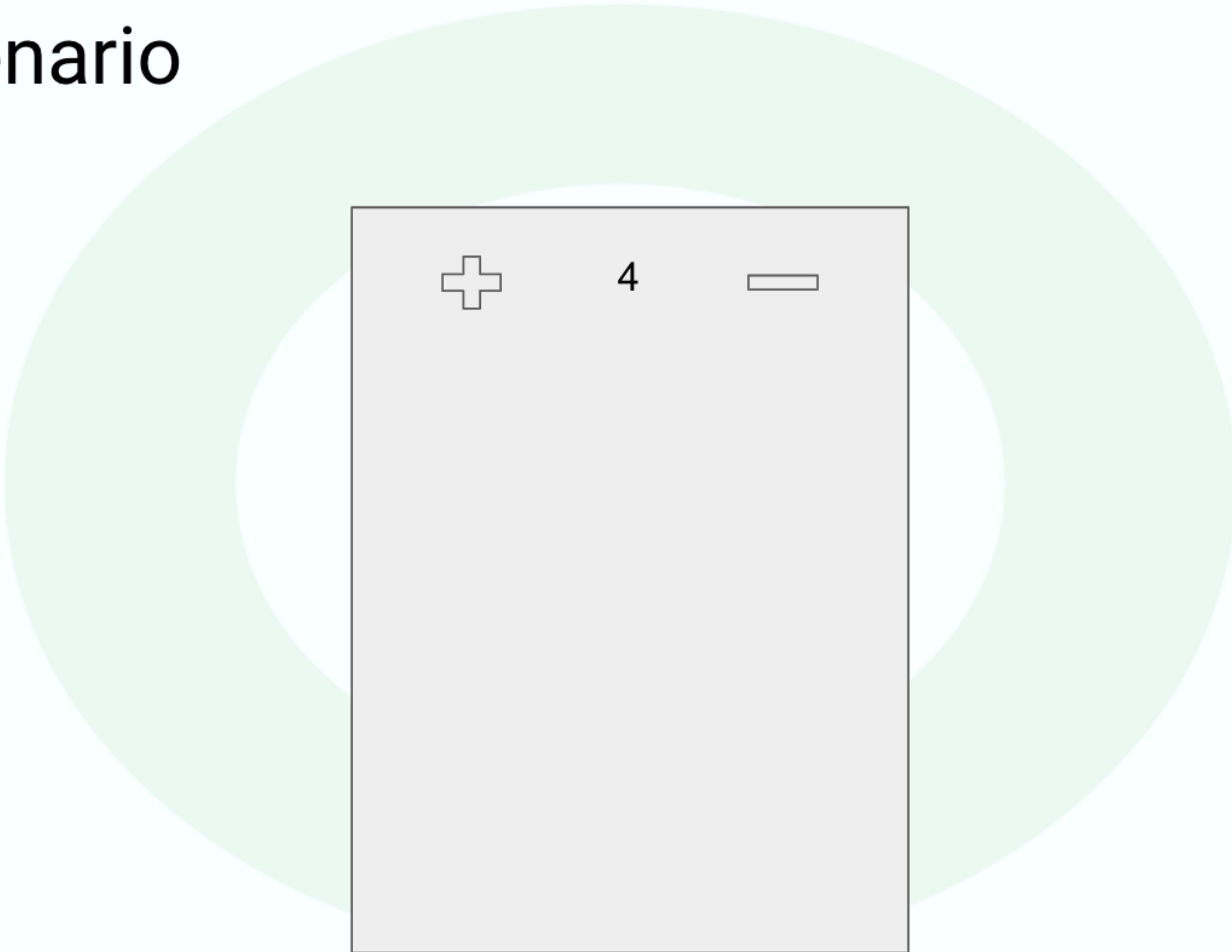
- Unidirectional
- View
- User and it's interaction
- Intent
- Action
- Model

It's time to see this pattern in code

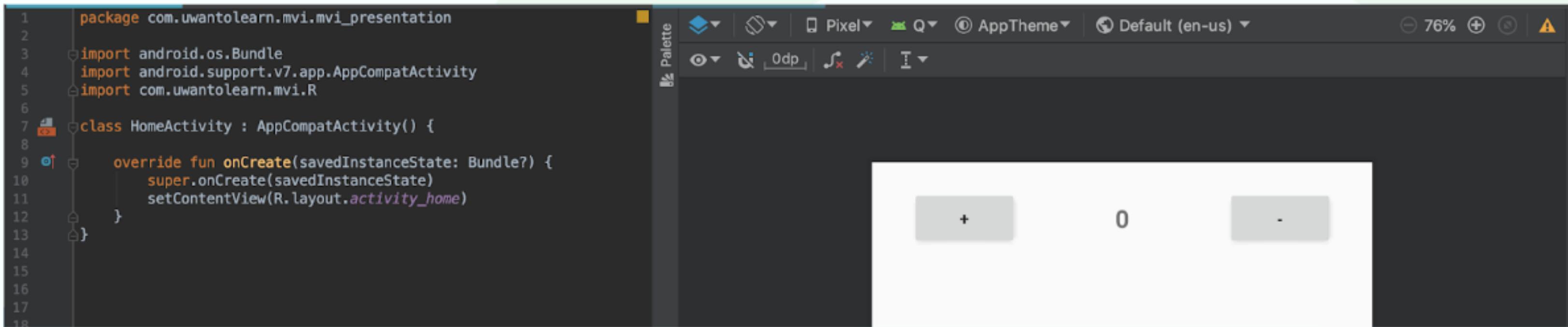
Language and libraries as Prerequisite (All are optional)

- Kotlin Language (Sealed classes, data classes, when exhaustive)
- RxJava/Kotlin (map, merge, scan)
- Rxbinding

Easy Scenario



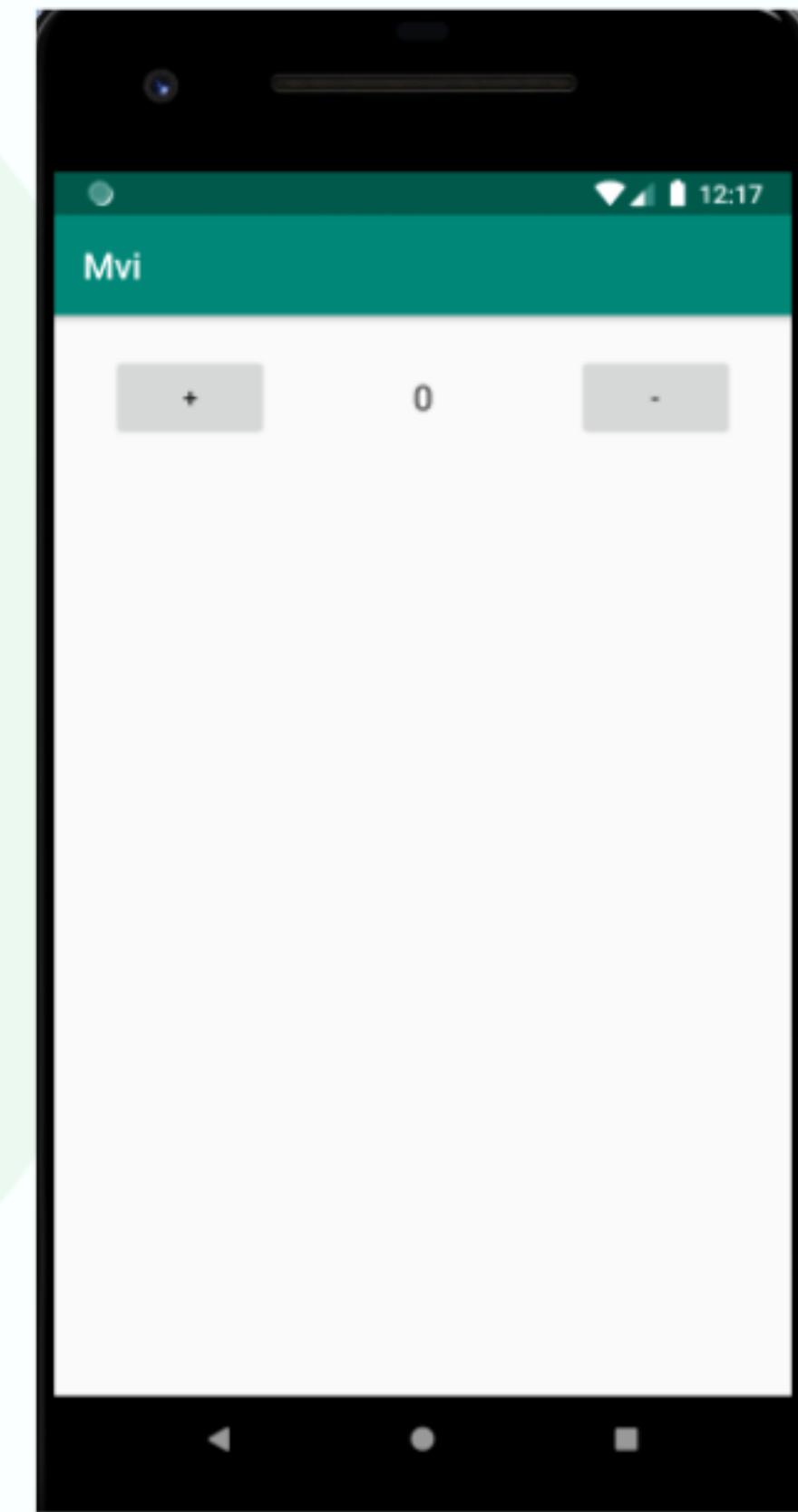
Android Ceremonies



A screenshot of an Android development environment. On the left is a code editor showing Java code for a HomeActivity:

```
1 package com.uwantolearn.mvi.mvi_presentation
2
3 import android.os.Bundle
4 import android.support.v7.app.AppCompatActivity
5 import com.uwantolearn.mvi.R
6
7 class HomeActivity : AppCompatActivity() {
8
9     override fun onCreate(savedInstanceState: Bundle?) {
10         super.onCreate(savedInstanceState)
11         setContentView(R.layout.activity_home)
12     }
13 }
14
15
16
17
18 }
```

The code editor has syntax highlighting and some annotations. To the right is a design preview window showing a layout with a central number '0' and two buttons labeled '+' and '-'. The top bar of the preview shows 'Pixel' and 'Default (en-us)'.



Implementation Strategy

There are a lot of strategies for implementation of this Pattern. In our case, we will go with the simplest one. We will do prepare independent concepts, and once they are ready, we will write glue code in between these concepts.

User Intentions or Intent

```
sealed class HomeIntent{  
    object IncrementIntent : HomeIntent()  
    object DecrementIntent : HomeIntent()  
}
```

Code how user will generate these intentions.

```
incrementButton.clicks().map { HomeIntent.IncrementIntent },  
decrementButton.clicks().map { HomeIntent.DecrementIntent }
```

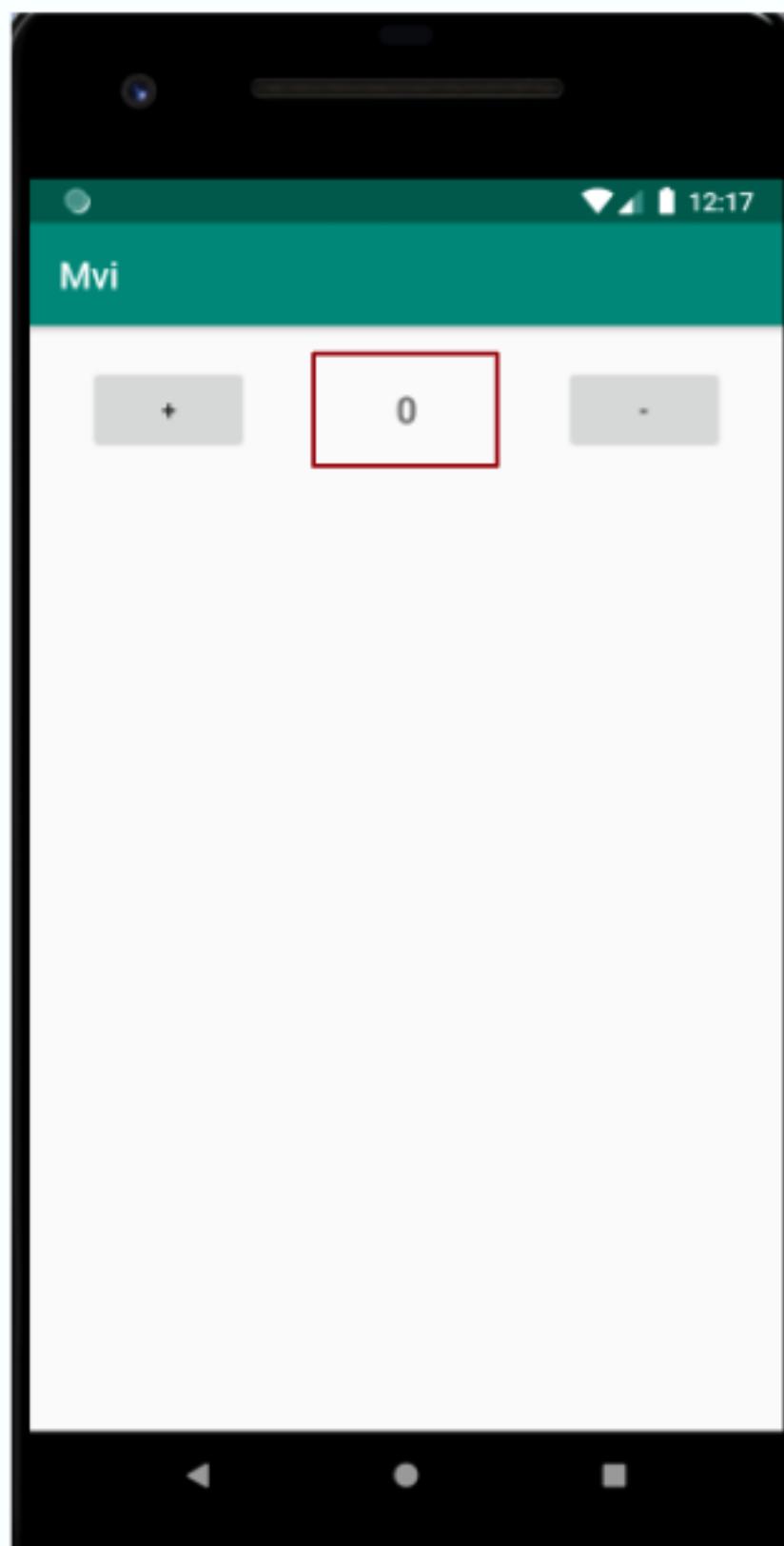
Merge All Intents

```
Observable.merge(  
    listOf<Observable<HomeIntent>>(  
        incrementButton.clicks().map { HomeIntent.IncrementIntent },  
        decrementButton.clicks().map { HomeIntent.DecrementIntent }  
    )  
)
```

View Will Provide all intents using `getIntents()`

```
fun getIntents(): Observable<HomeIntent> =  
    Observable.merge(  
        listOf<Observable<HomeIntent>>(  
            incrementButton.clicks().map { HomeIntent.IncrementIntent },  
            decrementButton.clicks().map { HomeIntent.DecrementIntent }  
        )  
)
```

ViewState (Model will be as view state)



```
data class HomeViewState(val counter: Int)
```

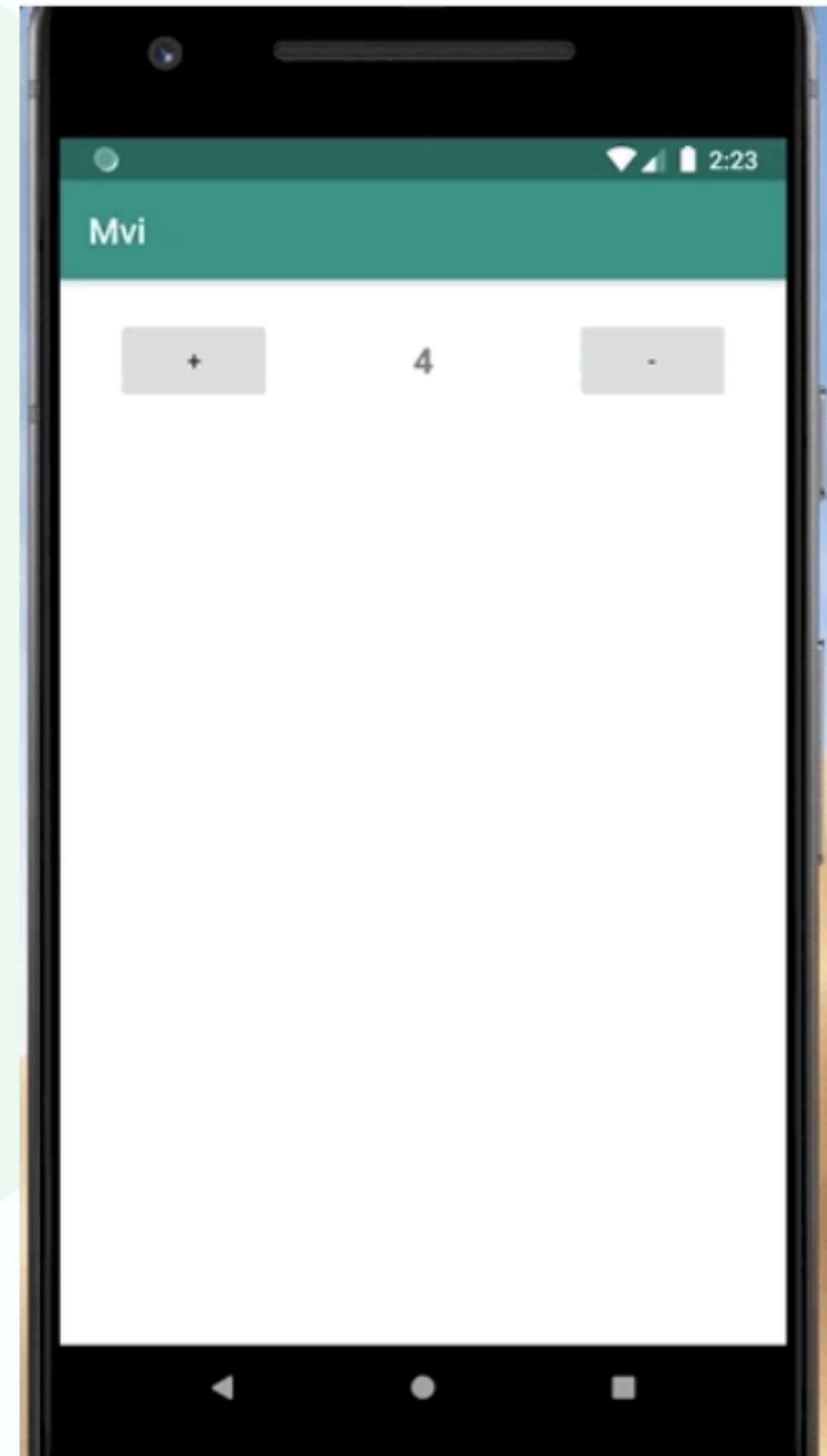
Intent Processing

```
private fun reduce(previousState: HomeViewState, intent: HomeIntent): HomeViewState =  
    when (intent) {  
        HomeIntent.IncrementIntent -> previousState.copy(counter = previousState.counter + 1)  
        HomeIntent.DecrementIntent -> previousState.copy(counter = previousState.counter - 1)  
    }
```

```
.scan(HomeViewState(counter: 0), ::reduce)
```

Glue code between Model View Intent

```
1 package com.uwantolearn.mvi.mvi_presentation
2
3 import ...
4
5 class HomeActivity : AppCompatActivity() {
6
7     private val compositeDisposable = CompositeDisposable()
8
9     override fun onCreate(savedInstanceState: Bundle?) {
10         super.onCreate(savedInstanceState)
11         setContentView(R.layout.activity_home)
12
13         getIntents()
14             .scan(HomeViewState(counter: 0), ::reduce)
15             .subscribe(::render)
16             .let(compositeDisposable::add)
17     }
18
19     private fun getIntents(): Observable<HomeIntent> =
20         Observable.merge(
21             listOf<Observable<HomeIntent>>(
22                 incrementButton.clicks().map { HomeIntent.IncrementIntent },
23                 decrementButton.clicks().map { HomeIntent.DecrementIntent }
24             )
25         )
26
27     private fun reduce(previousState: HomeViewState, intent: HomeIntent): HomeViewState =
28         when (intent) {
29             HomeIntent.IncrementIntent -> previousState.copy(counter = previousState.counter + 1)
30             HomeIntent.DecrementIntent -> previousState.copy(counter = previousState.counter - 1)
31         }
32
33     private fun render(state: HomeViewState) {
34         state.counter.toString()
35             .let(counterTextView::setText)
36     }
37
38     override fun onDestroy() {
39         compositeDisposable.clear()
40         super.onDestroy()
41     }
42
43 }
44
45 sealed class HomeIntent {
46     object IncrementIntent : HomeIntent()
47     object DecrementIntent : HomeIntent()
48 }
49
50
51
52
53 data class HomeViewState(val counter: Int)
54
55
56 }
```



Until Now

- We know what is Unidirectional flow and how we can achieve this in code.
- View, Model (as ViewState), Intents, Intent Processing (scan, reduce) are clear

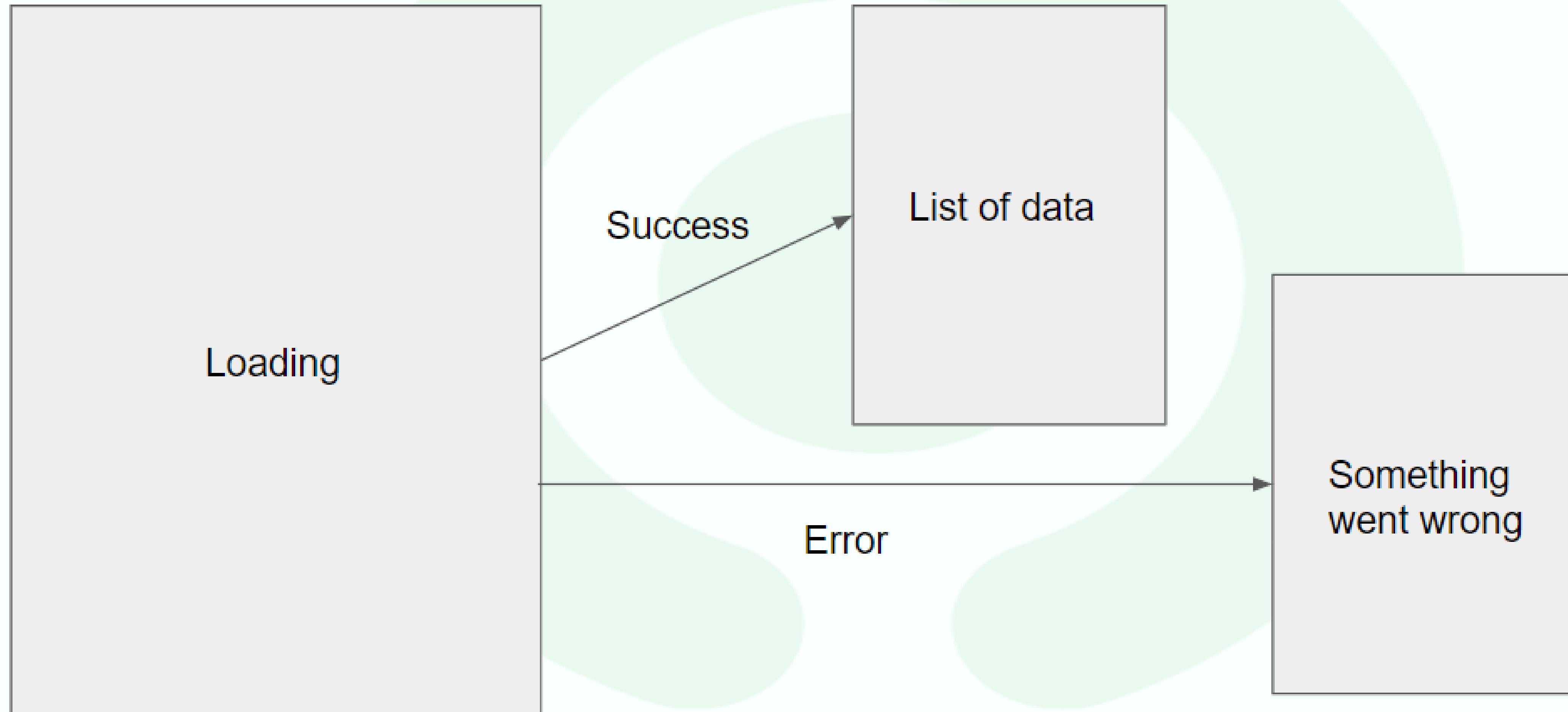
Code: <https://github.com/Hafiz-Waleed-Hussain/MviExample/tree/presentation-mvi-easy-implementation>

Any confusions or questions?

Issues or Problems Until Now

- ViewState, Intents, processing are coupled
- Everything is Synchronized (No Async calls)
- Config changes are not handle (Will lose data on rotation)
- Android Process Death not handle

Medium Scenario



New Intentions

```
sealed class HomeIntent {  
    object IncrementIntent : HomeIntent()  
    object DecrementIntent : HomeIntent()  
    object LoadDataIntent : HomeIntent() // New Intent which we need in new requirement  
}
```

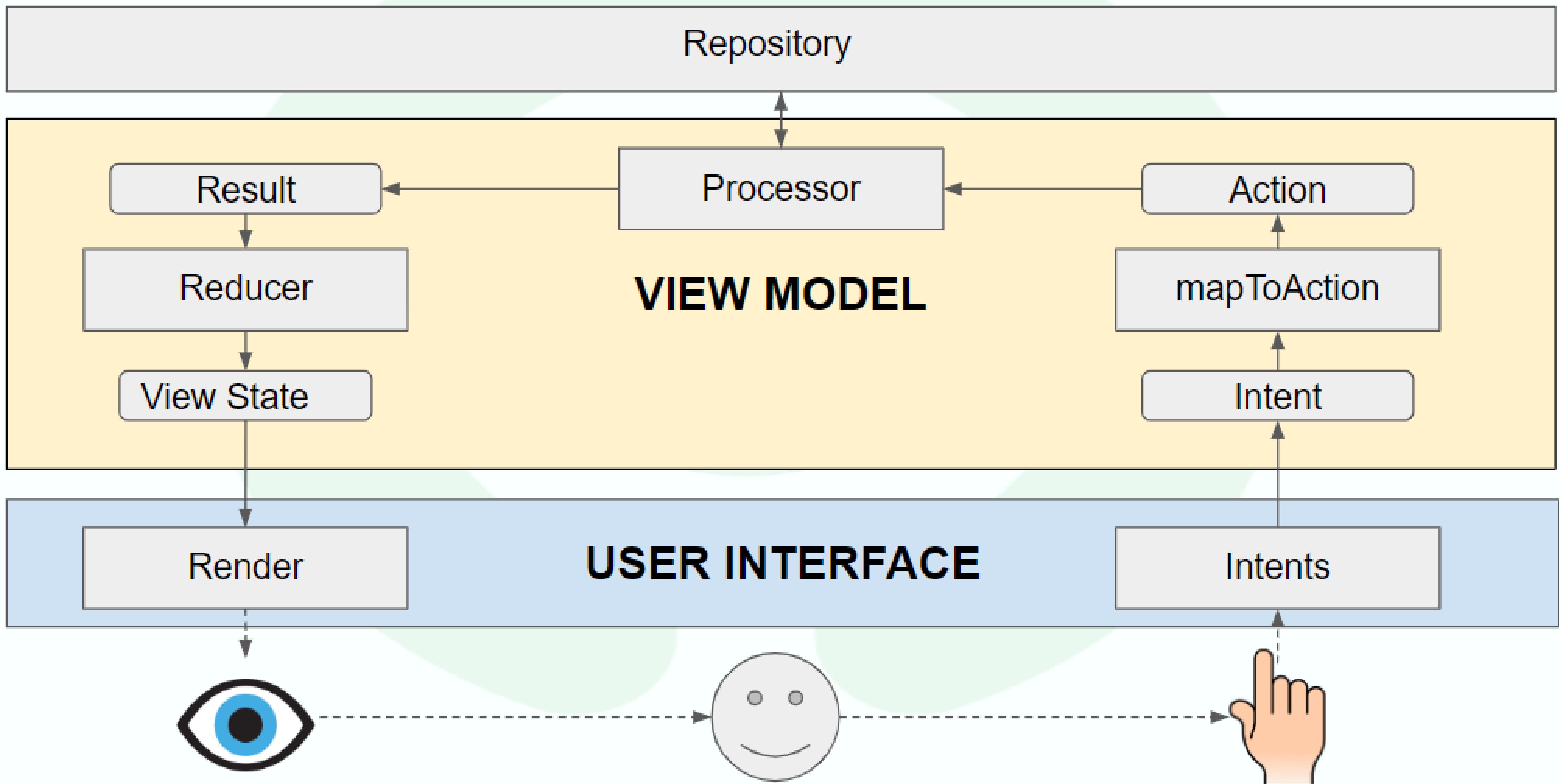
ViewState

```
data class HomeViewState(  
    val counter: Int,  
    val inProgress: Boolean,  
    val data: List<String>,  
    val isError: Boolean  
)
```

Here, I change ViewState data class to sealed class. We can use both.

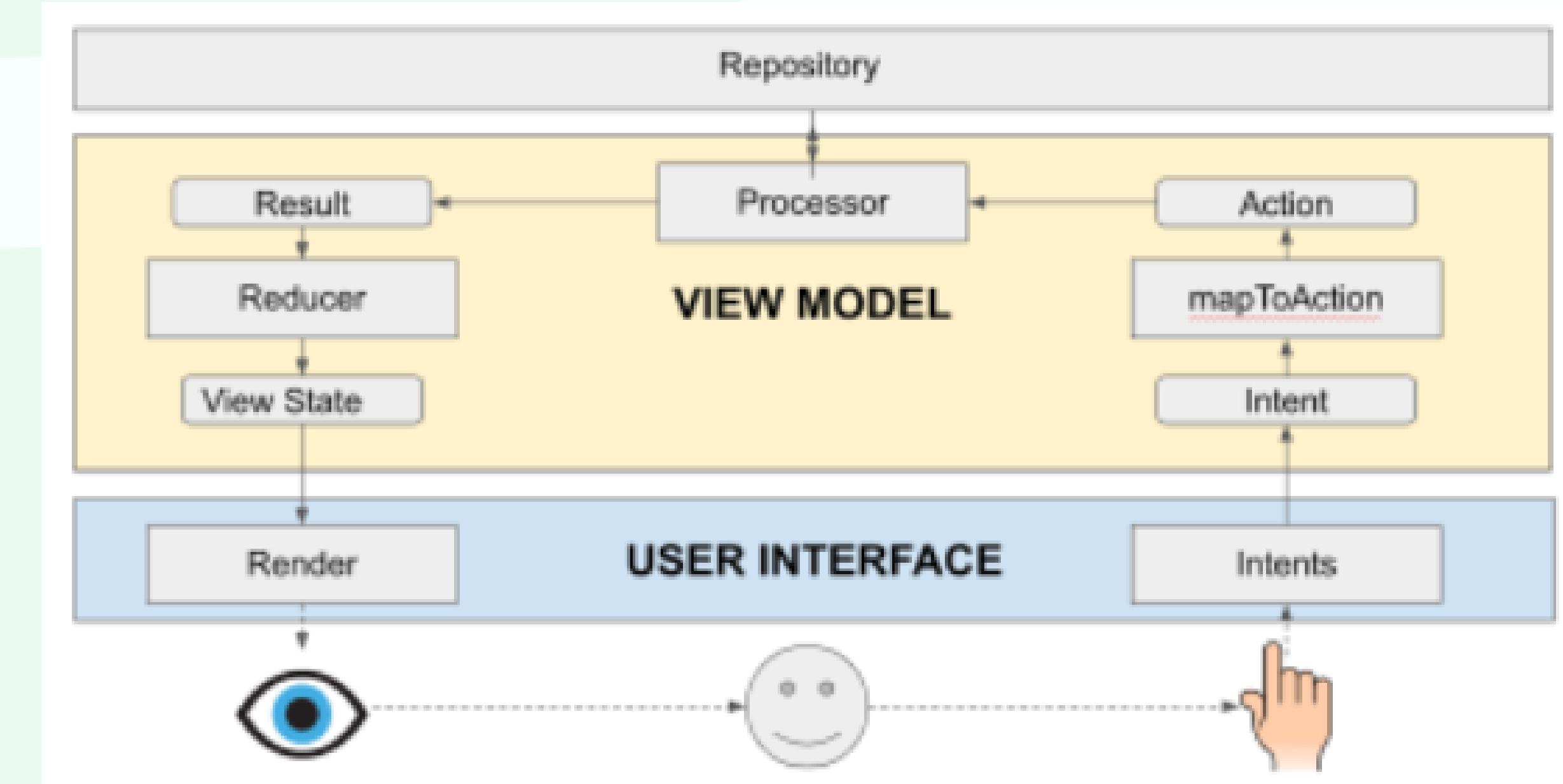
```
sealed class HomeViewState {  
    object ProgressViewState : HomeViewState()  
    object FailureViewState : HomeViewState()  
    data class DataViewState(val list: List<String>) : HomeViewState()  
    data class CounterViewState(val counter: Int) : HomeViewState()  
}
```

Fix of ViewState, Intents and Processing Coupling

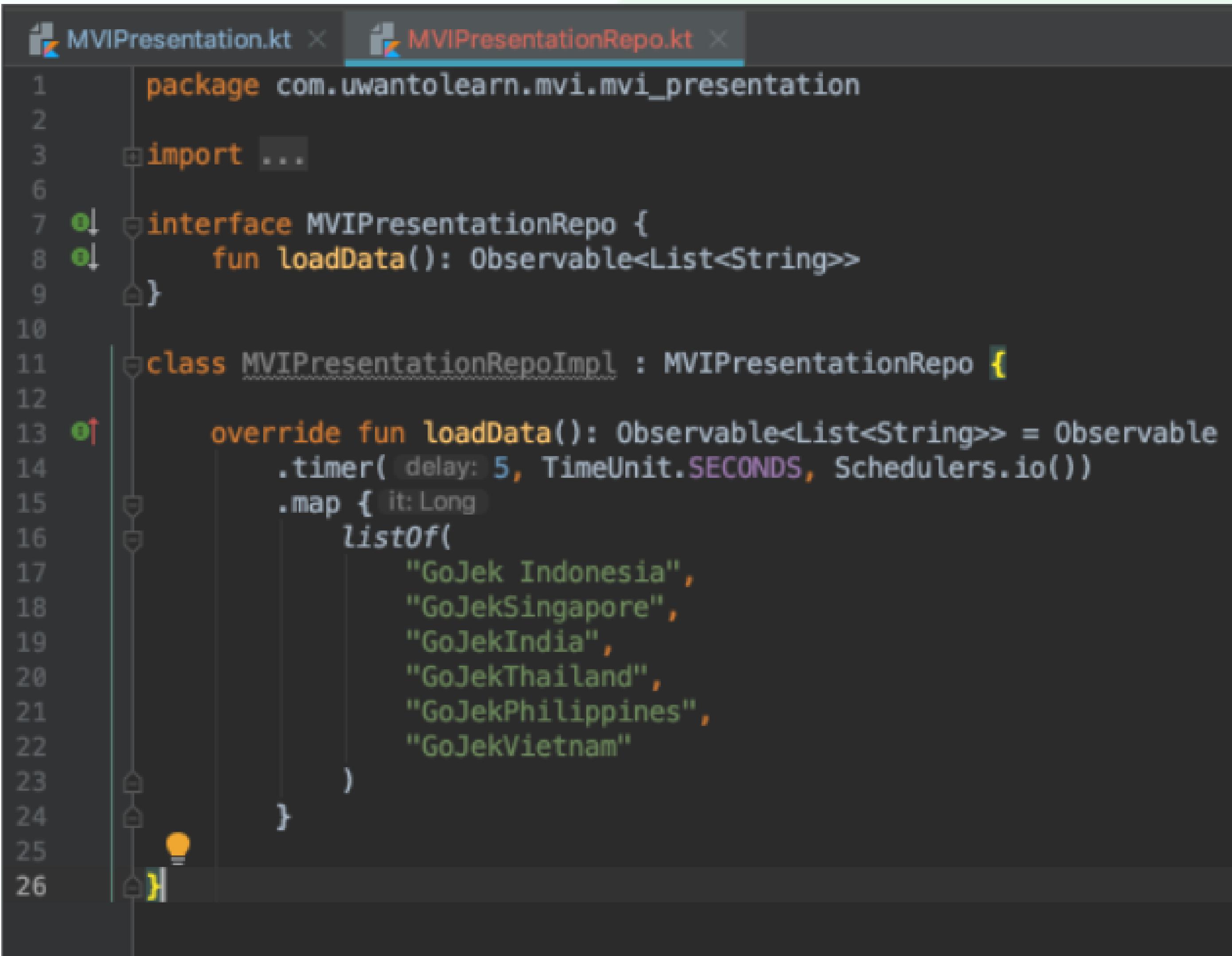


User Interface

```
1 package com.wanted.learn.mvi.mvi_presentation
2
3 import ...
4
5 class HomeActivity : AppCompatActivity() {
6
7     private val compositeDisposable = CompositeDisposable()
8
9     override fun onCreate(savedInstanceState: Bundle?) {
10         super.onCreate(savedInstanceState)
11         setContentView(R.layout.activity_main)
12     }
13
14     override fun onDestroy() {
15         compositeDisposable.clear()
16         super.onDestroy()
17     }
18
19     fun intents(): Observable<Intent> =
20         Observable.merge(
21             Observable.of<Intent>(Intent.ACTION_SEARCH),
22             Observable.just(<@name Intent.LaunchIntent>)
23         )
24
25     fun renderList(data: List<String>): Unit = when (state) {
26         is HomeViewState.ProgressViewState -> renderLoadingState()
27         is HomeViewState.FailureViewState -> renderFailureState()
28         is HomeViewState.DataViewState -> renderDataState(state.list)
29     }
30
31     private fun renderLoadingState() {
32         dataTextView.visibility = View.GONE
33         progressbar.visibility = View.VISIBLE
34     }
35
36     private fun renderFailureState() {
37         progressbar.visibility = View.GONE
38
39         dataTextView.visibility = View.VISIBLE
40         dataTextView.text = getString(R.string.something_went_wrong)
41     }
42
43     private fun renderDataState(data: List<String>) {
44         progressbar.visibility = View.GONE
45
46         dataTextView.visibility = View.VISIBLE
47         dataTextView.text = data.reduce { acc, s -> "$acc, $s" }
48             .let { dataTextView.setText(it) }
49     }
50
51     sealed class HomeIntent {
52         object LoadDataIntent : HomeIntent()
53     }
54
55     sealed class HomeViewState {
56         object ProgressViewState : HomeViewState()
57         object FailureViewState : HomeViewState()
58         data class DataViewState(val list: List<String>) : HomeViewState()
59     }
60 }
```



Repository



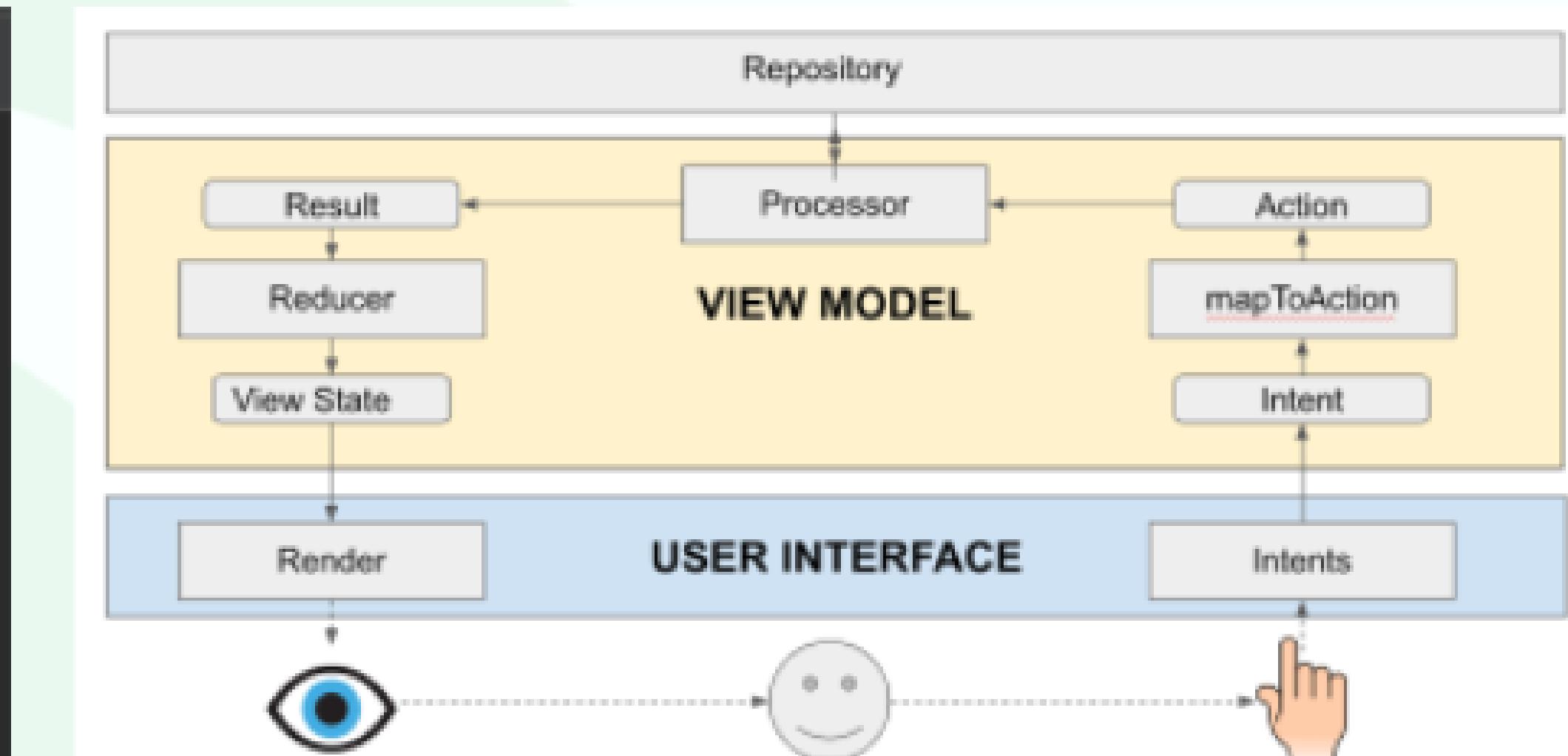
```
MVIPresentation.kt x MVIPresentationRepo.kt x
package com.uwantolearn.mvi.mvi_presentation

import ...

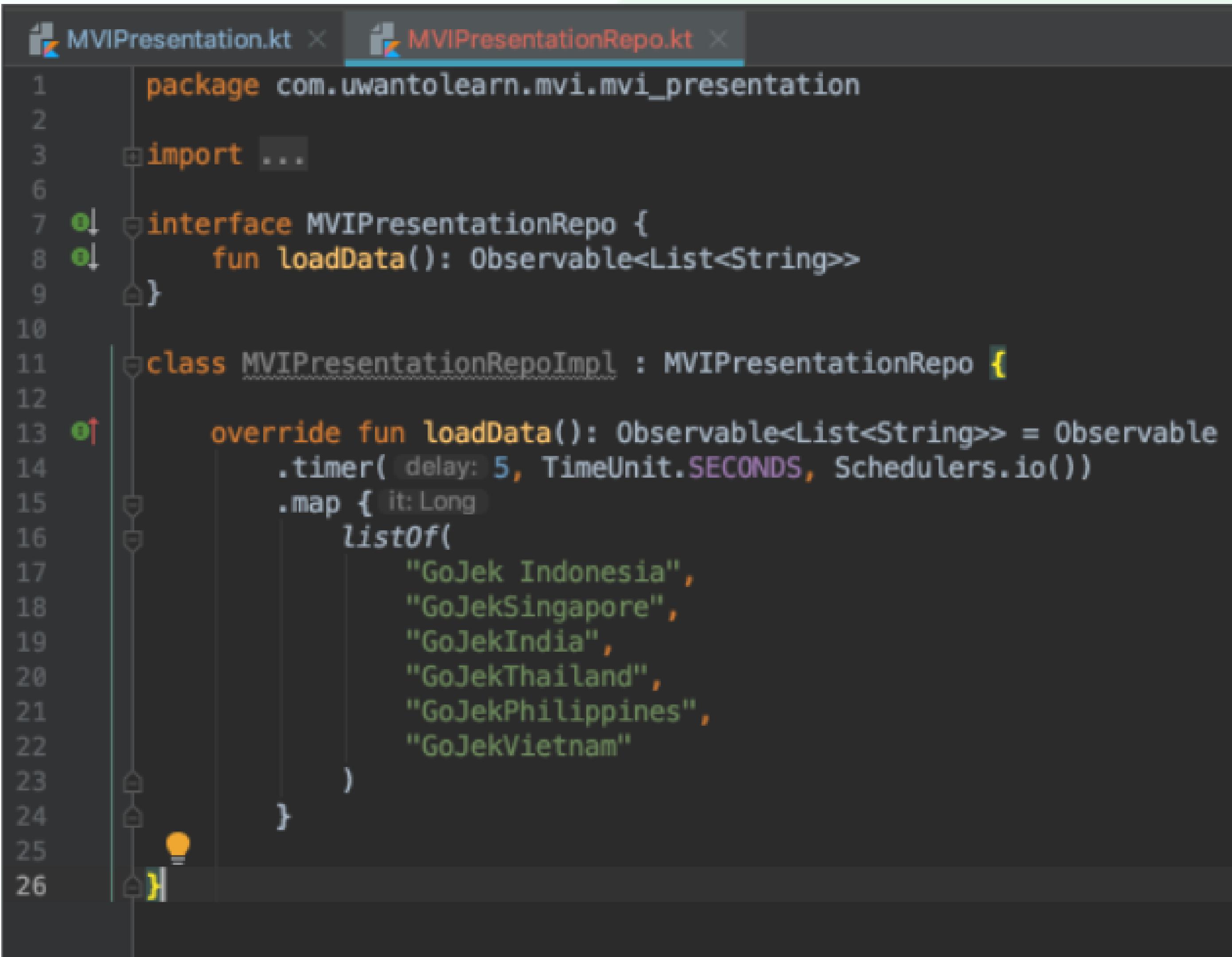
interface MVIPresentationRepo {
    fun loadData(): Observable<List<String>>
}

class MVIPresentationRepoImpl : MVIPresentationRepo {

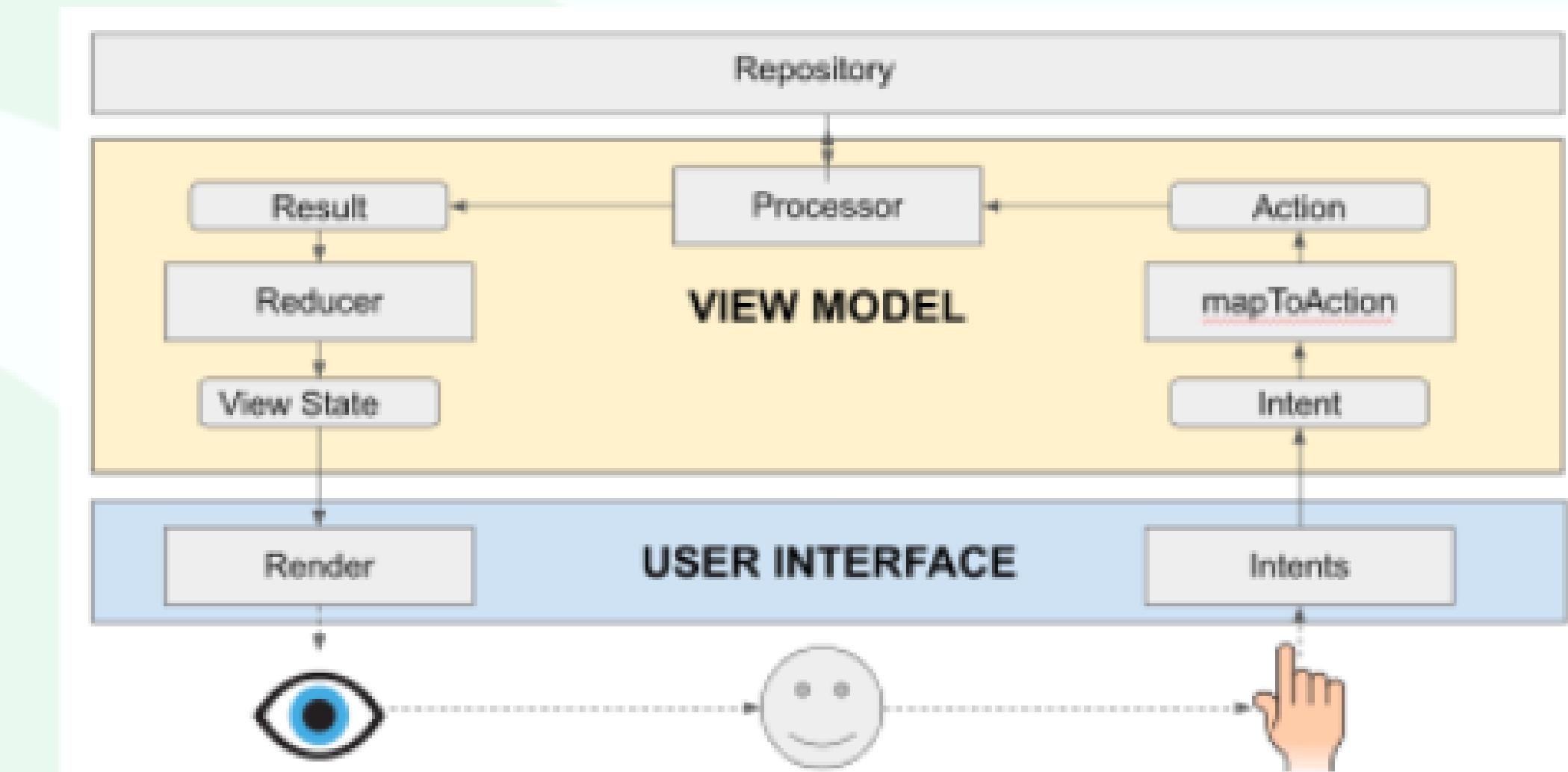
    override fun loadData(): Observable<List<String>> = Observable
        .timer( delay: 5, TimeUnit.SECONDS, Schedulers.io())
        .map { it: Long
            listOf(
                "GoJek Indonesia",
                "GoJekSingapore",
                "GoJekIndia",
                "GoJekThailand",
                "GoJekPhilippines",
                "GoJekVietnam"
            )
        }
}
```



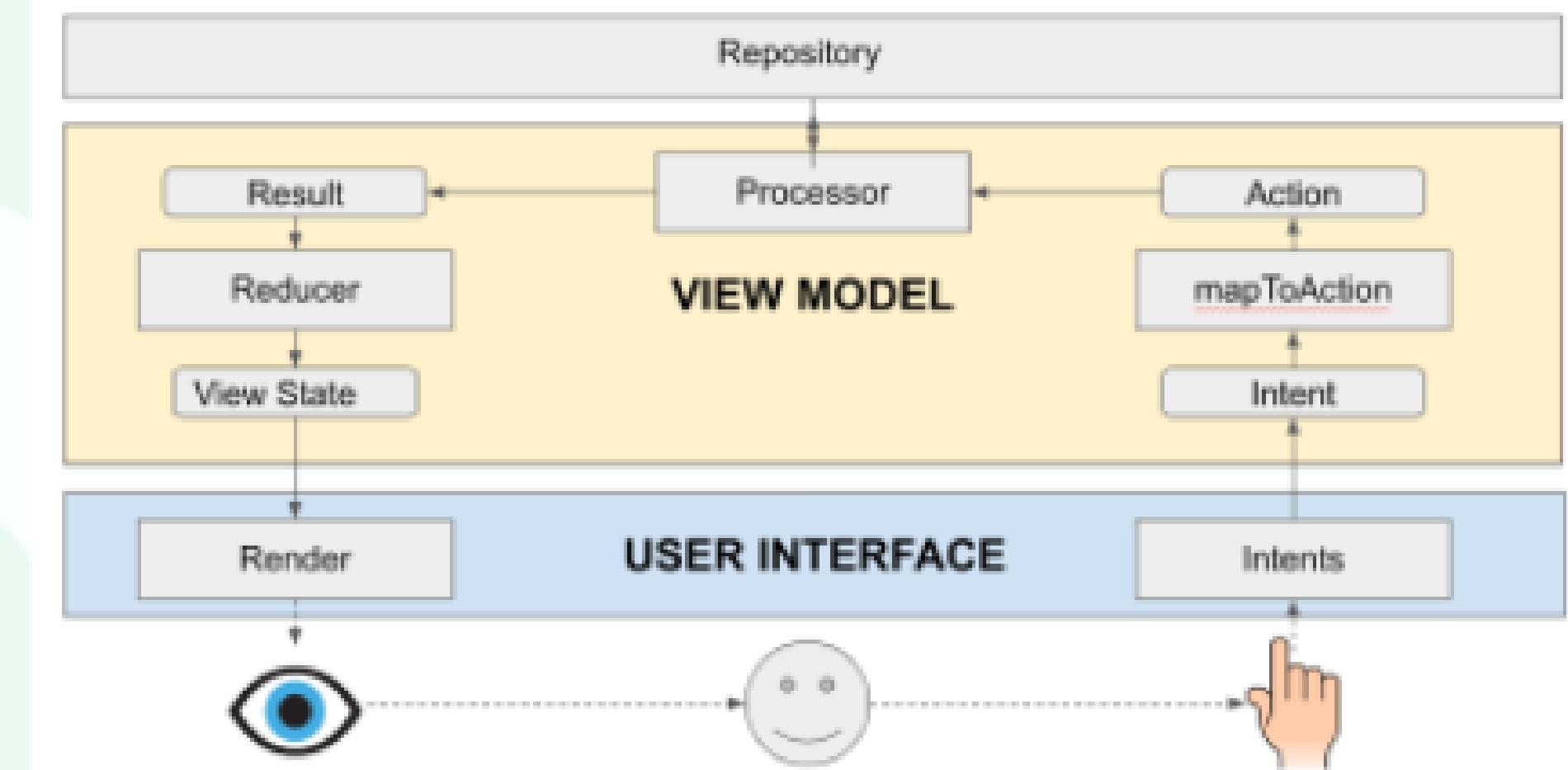
Repository



```
1 package com.uwantolearn.mvi.mvi_presentation
2
3 import ...
4
5
6 interface MVIPresentationRepo {
7     fun loadData(): Observable<List<String>>
8 }
9
10
11 class MVIPresentationRepoImpl : MVIPresentationRepo {
12
13     override fun loadData(): Observable<List<String>> = Observable
14         .timer( delay: 5, TimeUnit.SECONDS, Schedulers.io())
15         .map { it: Long
16             listOf(
17                 "GoJek Indonesia",
18                 "GoJekSingapore",
19                 "GoJekIndia",
20                 "GoJekThailand",
21                 "GoJekPhilippines",
22                 "GoJekVietnam"
23             )
24         }
25
26 }
```



ViewModel

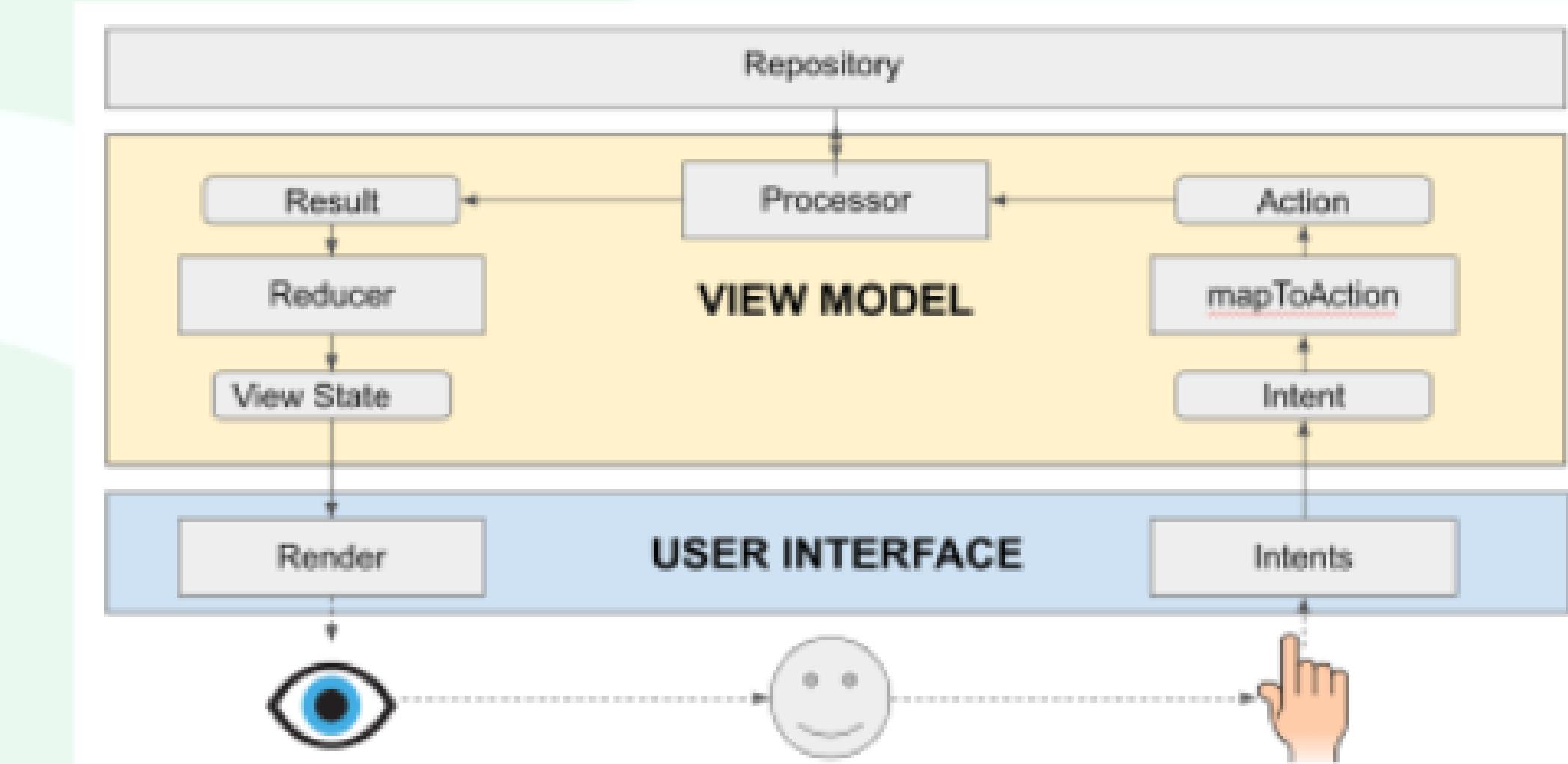


```
sealed class HomeActivityAction {  
    object LoadDataAction : HomeActivityAction()  
}
```

```
private fun mapToActions(intent: HomeIntent): HomeActivityAction = when (intent) {  
    HomeIntent.LoadDataIntent -> HomeActivityAction.LoadDataAction  
}
```

ViewModel

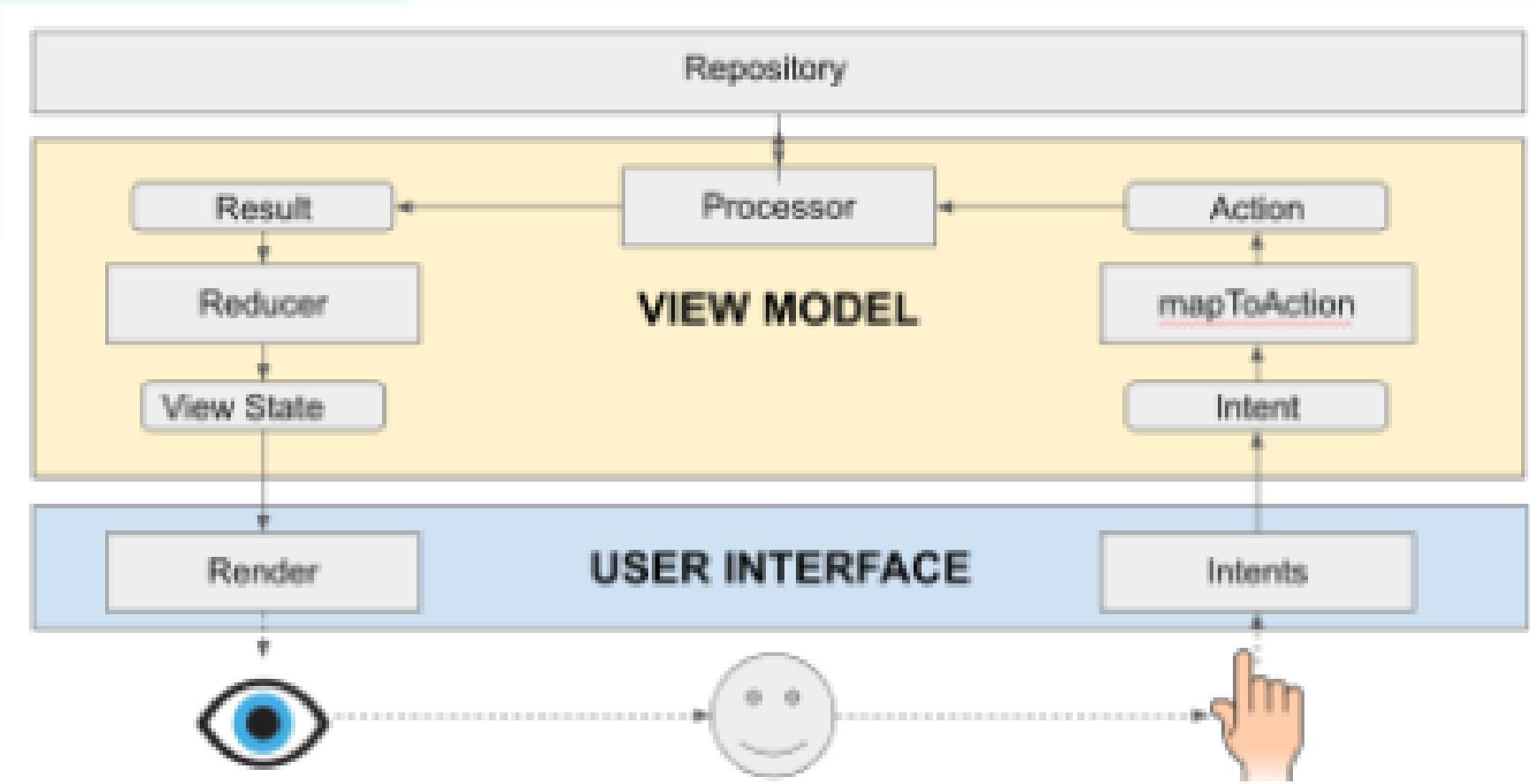
```
sealed class HomeActivityResult {  
    data class DataResult(val data: List<String>) : HomeActivityResult()  
    object FailureResult : HomeActivityResult()  
    object LoadingResult : HomeActivityResult()  
}
```



```
private fun reduce(previousState: HomeViewState, result: HomeActivityResult): HomeViewState =  
    when (result) {  
        is HomeActivityResult.DataResult -> HomeViewState.DataViewState(result.data)  
        HomeActivityResult.FailureResult -> HomeViewState.FailureViewState  
        HomeActivityResult.LoadingResult -> HomeViewState.ProgressViewState  
    }
```

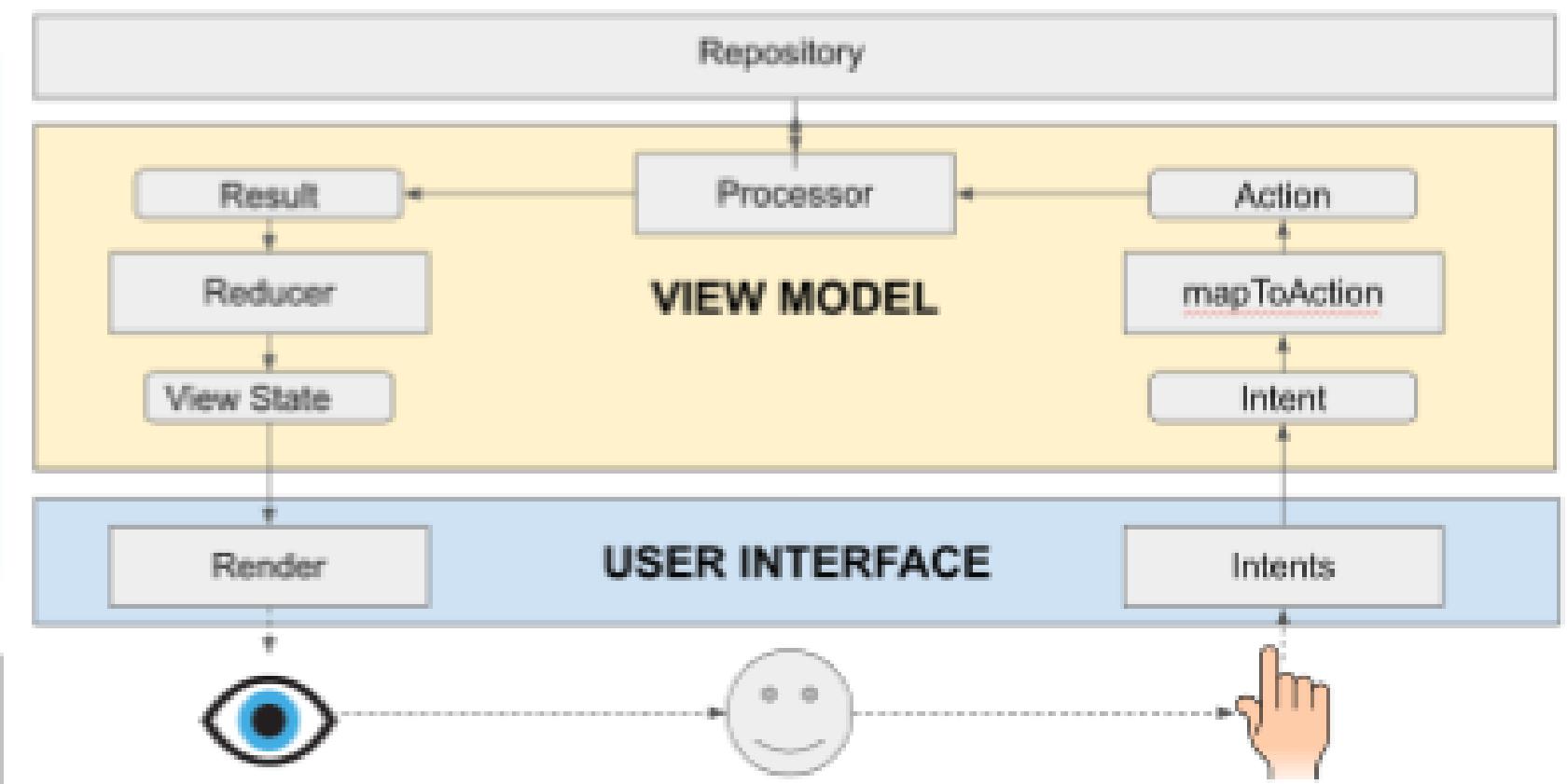
Until Now View Model

```
MVIPresentation.kt x MVIPresentationViewModel.kt x
1 package com.uwantolearn.mvi.mvi_presentation
2
3 import io.reactivex.ObservableTransformer
4
5 class MVIPresentationViewModel(val repo: MVIPresentationRepo) {
6
7     private fun mapToActions(intent: HomeIntent): HomeActivityAction = when (intent) {
8         HomeIntent.LoadDataIntent -> HomeActivityAction.LoadDataAction
9     }
10
11     private fun reduce(previousState: HomeViewState, result: HomeActivityResult): HomeViewState =
12         when (result) {
13             is HomeActivityResult.DataResult -> HomeViewState.DataViewState(result.data)
14             HomeActivityResult.FailureResult -> HomeViewState.FailureViewState
15             HomeActivityResult.LoadingResult -> HomeViewState.ProgressViewState
16         }
17
18     sealed class HomeActivityAction {
19         object LoadDataAction : HomeActivityAction()
20     }
21
22     sealed class HomeActivityResult {
23         data class DataResult(val data: List<String>) : HomeActivityResult()
24         object FailureResult : HomeActivityResult()
25         object LoadingResult : HomeActivityResult()
26     }
27 }
28
```



ViewModel

```
class MVIPresentationActionProcessor(private val repo: MVIPresentationRepo) {  
  
    private val loadDataActionProcessor =  
        ObservableTransformer<HomeActivityAction.LoadDataAction, HomeActivityResult> { action ->  
            action.flatMap { it: HomeActivityAction.LoadDataAction  
                repo.loadData()  
                    .map(HomeActivityResult::DataResult)  
                    .cast(HomeActivityResult::class.java)  
                    .onErrorReturn { HomeActivityResult.FailureResult }  
                    .startWith(HomeActivityResult.LoadingResult)  
            }  
        }  
  
    val actionProcessor = ObservableTransformer<HomeActivityAction, HomeActivityResult> { action ->  
        action.publish { actionSource ->  
            actionSource.ofType(HomeActivityAction.LoadDataAction::class.java)  
                .compose(loadDataActionProcessor)  
        }  
    }  
}
```



Complete View Model

```
# MVVPresentationViewModel.kt
package com.quentinlears.mvi.mvi_presentation

import ...

class MVIViewPresentationView(val repo: MVIPresentationRepo) {

    private val actionProcessor = MVIPresentationActionProcessor(repo)

    fun bindIntents(observedIntent: Observable<Intent>): Observable<HomeViewState> {
        val intents = listOf(
            mapToAction,
            composeActions,
            scanHomeViewState,
            ProgressViewState :: reduce,
            subscribeSchedulers,
            observeSchedulers,
            loadThread
        )
        return observedIntent.flatMap { intent ->
            actionProcessor.mapToAction(intent)
        }.map { actionProcessor.mapAction(intent, action) }
    }

    private fun mapActions(intent: HomeIntent): HomeActivityAction = when (intent) {
        HomeIntent.LoadViewState -> HomeActivityAction.LoadViewState
    }

    private fun reduce(previousState: HomeViewState, result: HomeActivityResult): HomeViewState =
        when (result) {
            is HomeActivityResult.DataResult -> HomeViewState.dataViewState(result.data)
            HomeActivityResult.FailureResult -> HomeViewState.failureViewState
            HomeActivityResult.LoadingResult -> HomeViewState.progressViewState
        }
    }

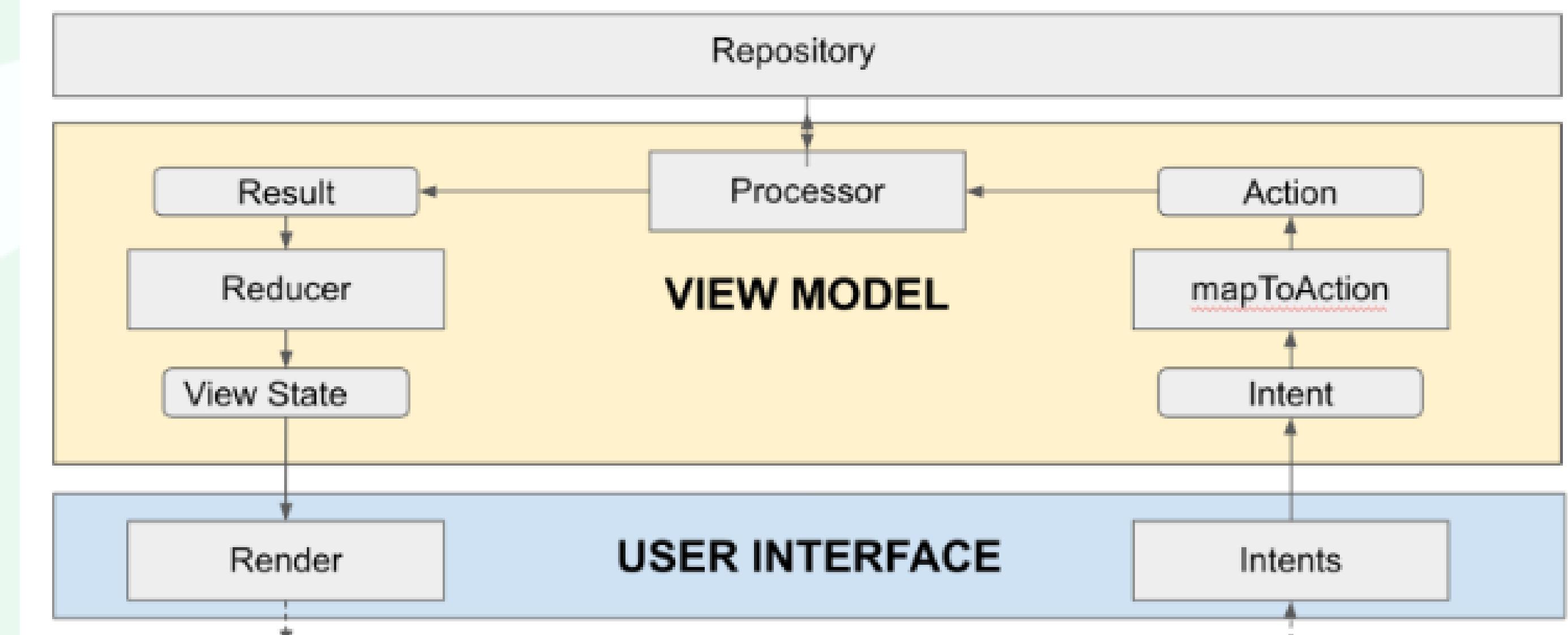
    sealed class HomeActivityAction {
        object LoadViewState : HomeActivityAction()
    }

    sealed class HomeActivityResult {
        data class DataResult(val data: List<String>) : HomeActivityResult()
        object FailureResult : HomeActivityResult()
        object LoadingResult : HomeActivityResult()
    }

    class MVIPresentationActionProcessor(private val repo: MVIPresentationRepo) {

        private val loadViewStateProcessor = ObservableTransformer<HomeActivityAction, HomeViewState> { action ->
            ObservableTransformer<HomeActivityAction, HomeActivityResult> { action ->
                action.flatMap { it: HomeActivityAction.LoadViewState ->
                    repo.loadData()
                        .map(HomeActivityResult::DataResult)
                        .cast(HomeActivityResult::class.java)
                        .onErrorReturn { HomeActivityResult.FailureResult }
                        .startWith(HomeActivityResult.LoadingResult)
                }
            }
        }

        val processActions = ObservableTransformer<HomeActivityAction, HomeViewState> { action ->
            action.publish { actionSource ->
                actionSource.ofType(HomeActivityAction.LoadViewState).map { javaClass }
                    .compose(loadViewStateProcessor)
            }
        }
    }
}
```

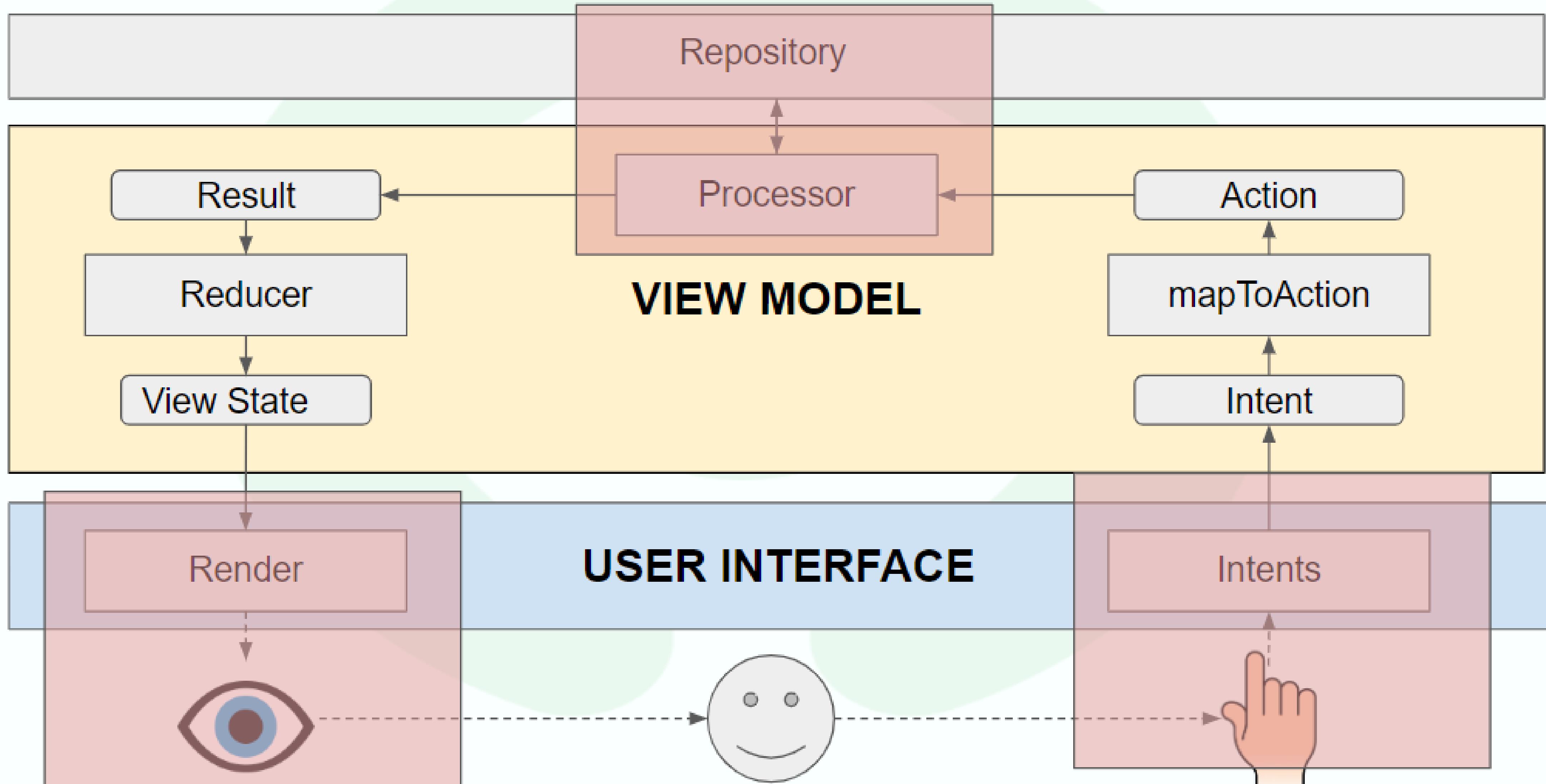


Glue Code

```

MVPresentationKit
1 package com.uwntolearn.mvi.mvi_presentation
2
3 import ...
4
5 class HomeActivity : AppCompatActivity() {
6
7     private val compositeDisposable = CompositeDisposable()
8     private val viewModel: MVPresentationViewModel =
9         MVPresentationViewModel(MVPresentationRepoImpl())
10
11    override fun onCreate(savedInstanceState: Bundle?) {
12        super.onCreate(savedInstanceState)
13        setContentView(R.layout.activity_home)
14        intents()
15        ,let(viewModel::bind)
16        ,subscribe()
17        ,let{compositeDisposable::add}
18    }
19
20    override fun onDestroy() {
21        compositeDisposable.clear()
22        super.onDestroy()
23    }
24
25    private fun intents(): Observable<HomeIntent> =
26        Observable.merge(
27            Observable<HomeIntent>(
28                Observable.just(HomeIntent.LoadDataIntent))
29        )
30
31    private fun render(state: HomeViewState): Unit = when (state) {
32        HomeViewState.ProgressViewState -> renderLoadingState()
33        HomeViewState.FailureViewState -> renderFailureState()
34        is HomeViewState.DataViewState -> renderDataState(state.list)
35    }
36
37    private fun renderLoadingState() {
38        dataTextView.visibility = View.GONE
39        progressBar.visibility = View.VISIBLE
40    }
41
42    private fun renderFailureState() {
43        progressBar.visibility = View.GONE
44        dataTextView.visibility = View.VISIBLE
45        dataTextView.text = "Something went wrong"
46    }
47
48    private fun renderDataState(data: List<String>) {
49        progressBar.visibility = View.GONE
50        dataTextView.visibility = View.VISIBLE
51        data.reduce { acc, s -> "$acc $s" }.let(dataTextView::setText)
52    }
53
54
55
56
57
58
59
60
61    sealed class HomeIntent {
62        object LoadDataIntent : HomeIntent()
63    }
64
65    sealed class HomeViewState {
66        object ProgressViewState : HomeViewState()
67        object FailureViewState : HomeViewState()
68        data class DataViewState(val list: List<String>) : HomeViewState()
69    }
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
617
618
619
619
620
621
622
623
624
625
626
627
627
628
629
629
630
631
632
633
634
635
635
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
```

Side Effects



Until Now

- We know what is Unidirectional flow and how we can achieve this in code.
- View, Model (as ViewState) , Intents, Intent Processing (scan, reduce) are clear
- Action Processor, Actions and Results
- How to manage Async Code

Code:

<https://github.com/Hafiz-Waleed-Hussain/MviExample/tree/presentation-mvi-medium-implementation-pre-refresh-intent>

Any confusions or questions?

Live Demo So we can see some benefits

- Add a Refresh Intent with Exception Mock
- Add a Button to get Random Number

Code:

<https://github.com/Hafiz-Waleed-Hussain/MviExample/tree/presentation-mvi-medium-implementation-complete>

Difficult Scenario

- Manage Config Changes (For now if your app is only portrait mode so this is optional step)
- Refactor to a small framework

Config Changes by using AndroidViewModel

```
class MVIPresentationViewModel(repo: MVIPresentationRepo) : ViewModel() {
```

```
class MVIPresentationViewModelFactory : ViewModelProvider.Factory {  
    override fun <T : ViewModel?> create(modelClass: Class<T>): T = MVIPresentationViewModel(  
        MVIPresentationRepoImpl()  
    ) as T  
}
```

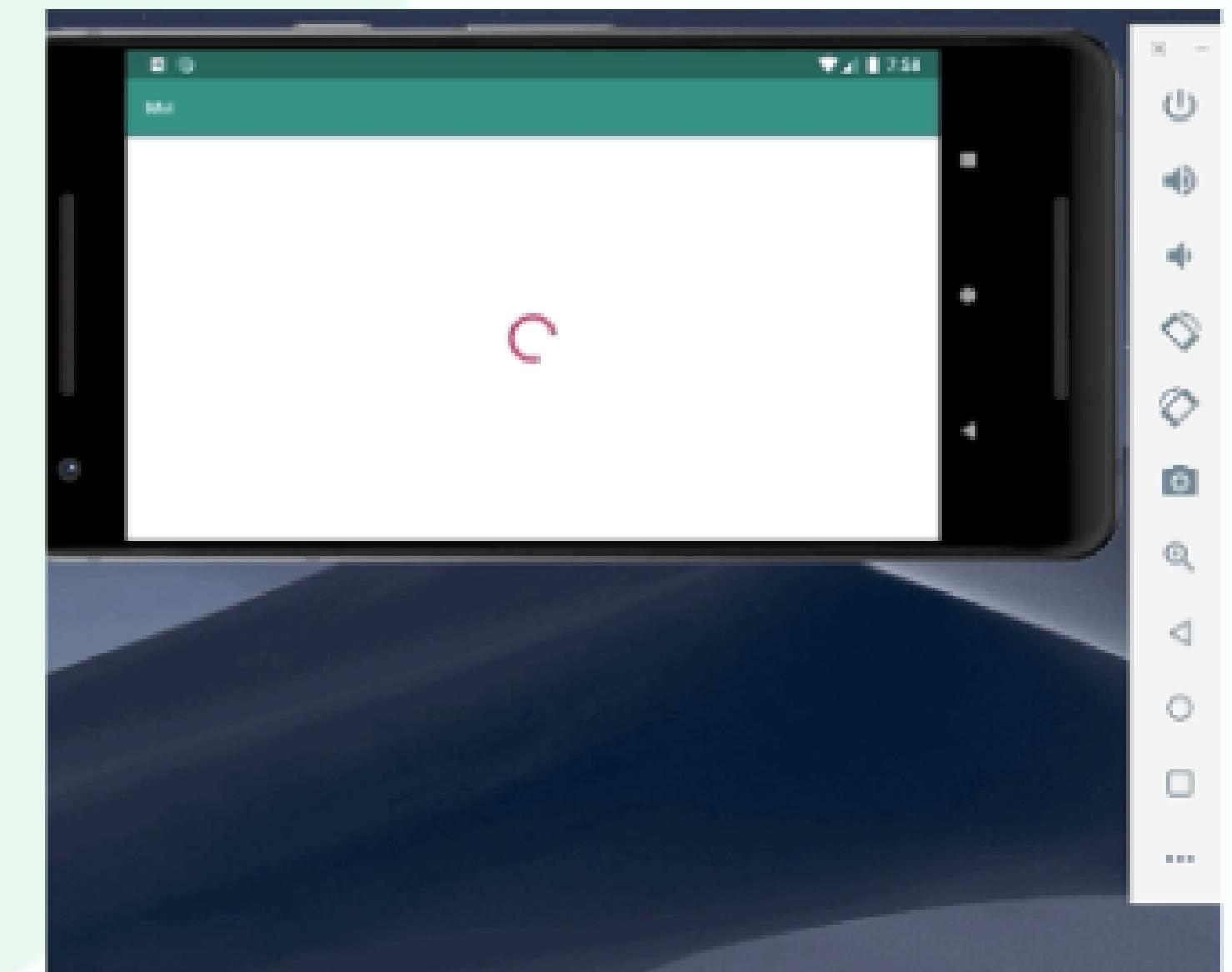
```
class HomeActivity : AppCompatActivity() {  
  
    private val compositeDisposable = CompositeDisposable()  
    private lateinit var viewModel: MVIPresentationViewModel  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_home)  
        viewModel =  
            ViewModelProviders.of(  
                activity: this,  
                MVIPresentationViewModelFactory()  
            )[MVIPresentationViewModel::class.java]  
  
        intents()  
            .let(viewModel::bind)  
            .subscribe(::render) { e -> Log.d(this.javaClass.simpleName, "msg: " + e.message) }  
            .let(compositeDisposable::add)  
    }  
}
```

Is ViewModel Survive Config Changes?

```
setContentview(R.layout.activity_name)
viewModel = viewModel: MVIPresentationViewModel@5356
ViewModelProviders.of(
    activity: this,
    MVIPresentationViewModelFactory()
) [MVIPresentationViewModel::class.java]
```

After Rotation

```
setContentview(R.layout.activity_name)
viewModel = viewModel: MVIPresentationViewModel@5356
ViewModelProviders.of(
    activity: this,
    MVIPresentationViewModelFactory()
) [MVIPresentationViewModel::class.java]
```



What Happened with data :(

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_home)
    viewModel =
        ViewModelProviders.of(
            activity: this,
            MVIPresentationViewModelFactory()
        )[MVPresentationViewModel::class.java]

    intents()
        .let(viewModel::bind)
        .subscribe(::render) { e -> Log.d(this.javaClass.simpleName, msg: "Error" + e.message) }
        .let(compositeDisposable::add)
}

override fun onDestroy() {
    compositeDisposable.clear()
    super.onDestroy()
}
```

We lose our stream.

Make ViewModel Stream independent from lifecycle.

```
class MVIPresentationViewModel(repo: MVIPresentationRepo) : ViewModel() {

    private val actionProcessor = MVIPresentationActionProcessor(repo)
    private val intentsSubject = PublishSubject.create<HomeIntent>()
    private val states = PublishSubject.create<HomeViewState>()

    init {
        intentsSubject
            .map(::mapToActions)
            .compose(actionProcessor.processActions)
            .scan(HomeViewState.ProgressViewState, ::reduce)
            .subscribeOn(Schedulers.io())
            .observeOn(AndroidSchedulers.mainThread())
            .subscribe(states)
    }

    fun processIntents(intents: Observable<HomeIntent>): Disposable = intents
        .subscribe(intentsSubject::onNext)

    fun state(): Observable<HomeViewState> = states.hide()

    fun mapToActions(intent: HomeIntent): HomeActivityAction = when (intent) {
        HomeIntent.LoadDataIntent, HomeIntent.RefreshIntent -> HomeActivityAction.LoadDataAction
        HomeIntent.GetRandomNumberIntent -> HomeActivityAction.GetRandomNumberAction
    }

    private fun reduce(previousState: HomeViewState, result: HomeActivityResult): HomeViewState =
        when (result) {
            is HomeActivityResult.DataResult -> HomeViewState.DataViewState(result.data)
            HomeActivityResult.FailureResult -> HomeViewState.FailureViewState
            HomeActivityResult.LoadingResult -> HomeViewState.ProgressViewState
            is HomeActivityResult.RandomNumber -> HomeViewState.RandomNumberState(result.randomNumber)
        }
}
```

Diagram Changes

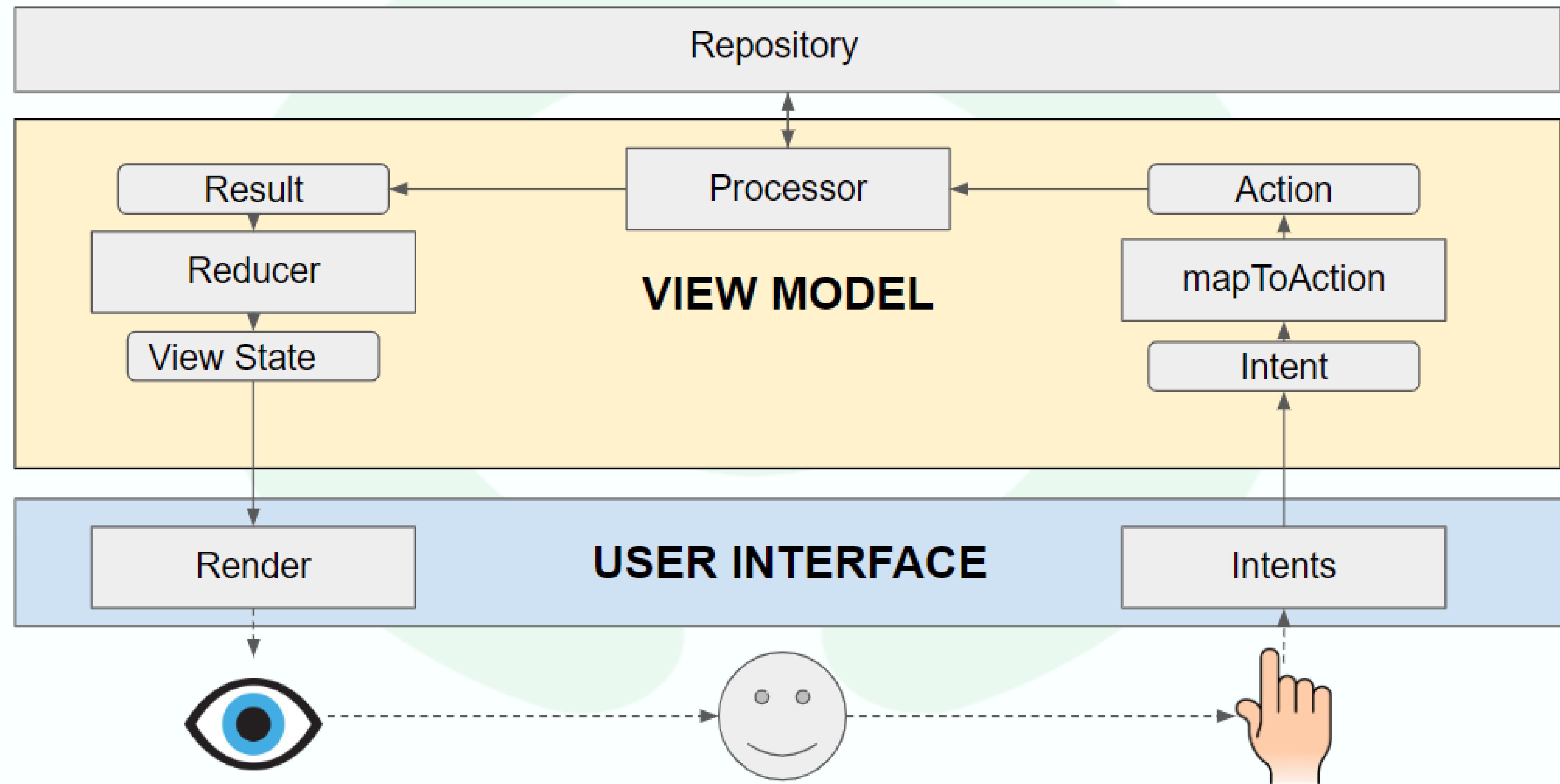


Diagram Changes

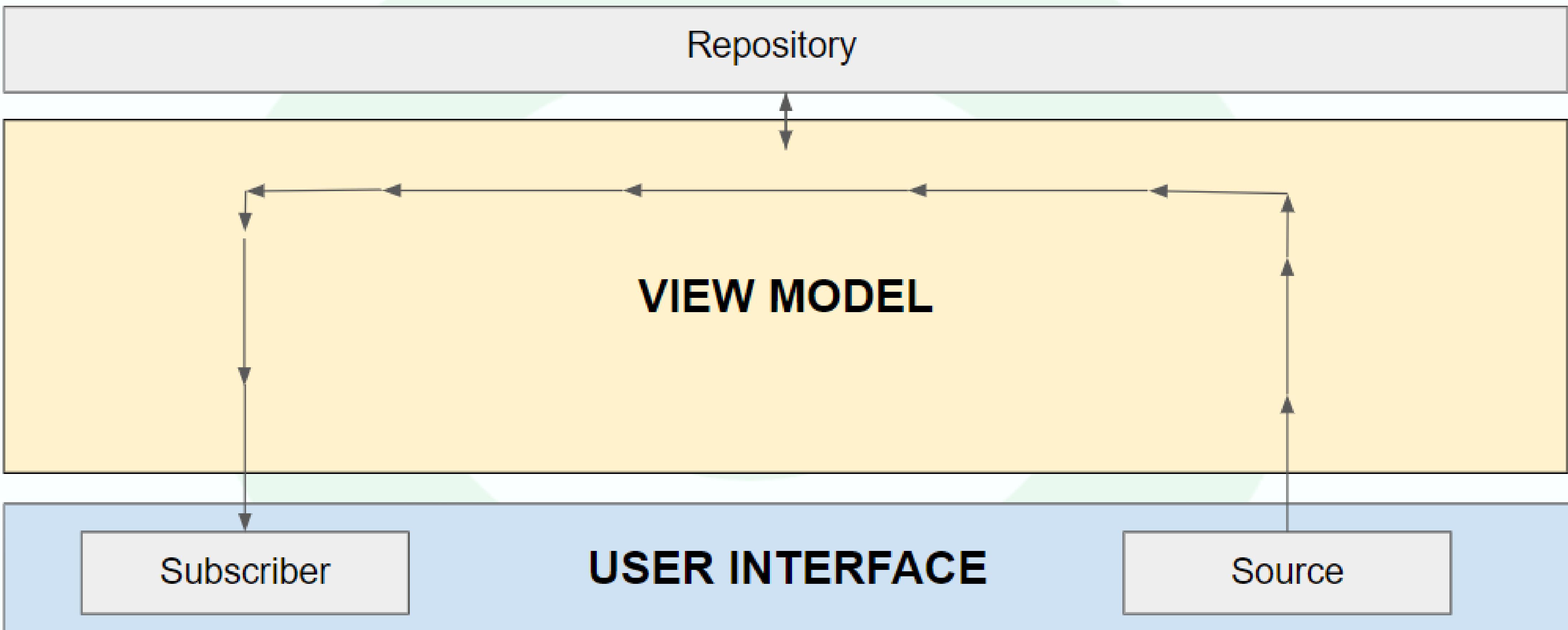


Diagram Changes

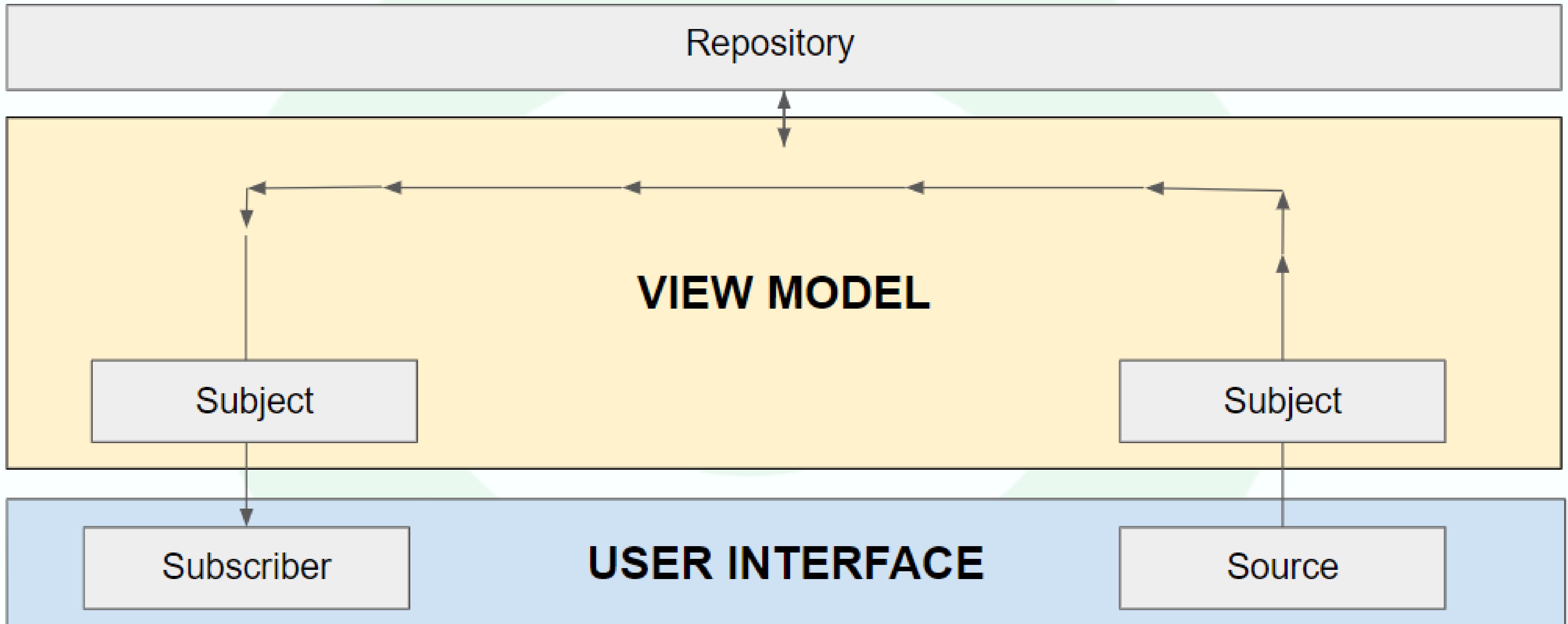


Diagram Changes (Config Change)

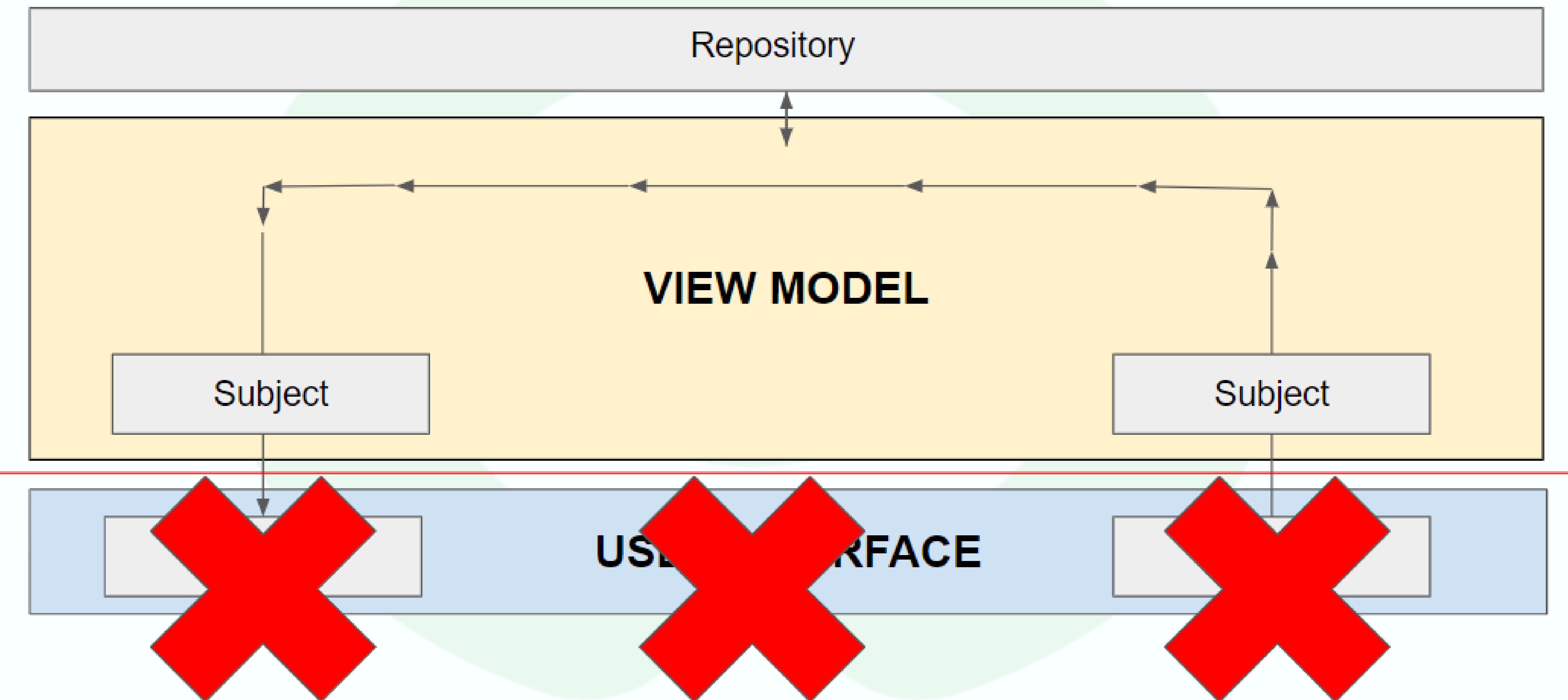


Diagram Changes (Config Change)

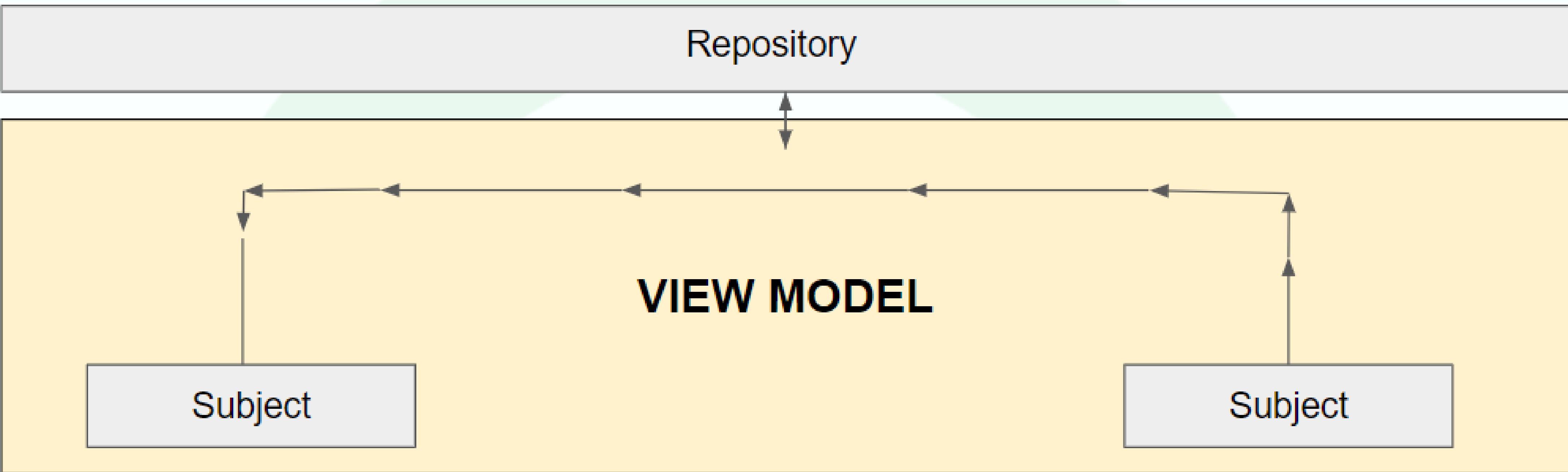
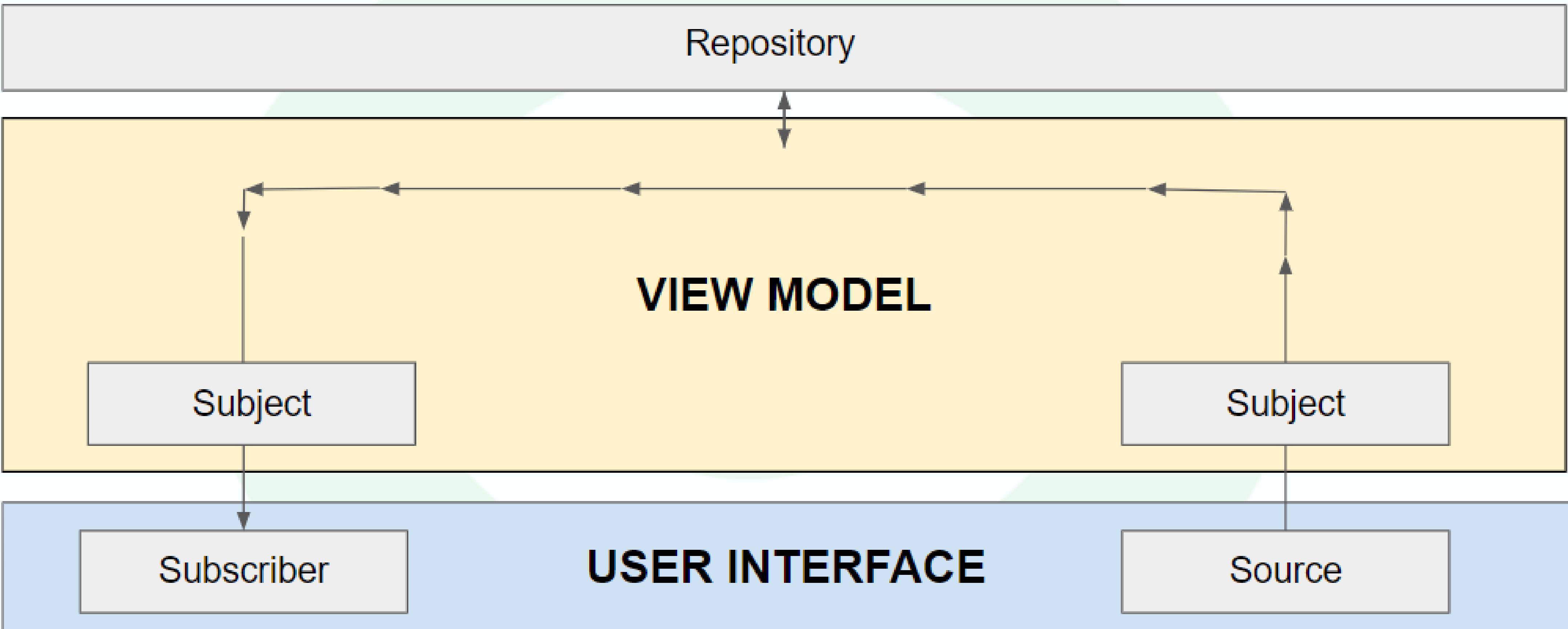


Diagram Changes



After New Code Changes

```
MVIPresentation.kt
```

```
import android.arch.lifecycle.ViewModelProvider
import android.arch.lifecycle.ViewModelProviders
import android.os.Bundle
import android.support.v7.app.AppCompatActivity
import android.util.Log
import android.view.View
import com.jakewharton.rxbinding3.view.clicks
import com.uwanteam.mvi.R
import com.uwanteam.mvi.base.MviIntent
import io.reactivex.Observable
import io.reactivex.disposables.CompositeDisposable
import kotlin.android.synthetic.main.activity_home.*
```

```
class MVIPresentationViewModelFactory : ViewModelProvider.Factory {
    override fun <T : ViewModel> create(modelClass: Class<T>): T = MVIPresentationViewModel()
    } as T
}
```

```
class HomeActivity : AppCompatActivity() {
    private val compositeDisposable = CompositeDisposable()
    private lateinit var viewModel: MVIPresentationViewModel
```

```
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_home)
        viewModel =
            ViewModelProviders.of(
                activity,
                MVIPresentationViewModelFactory()
            )(MVIPresentationViewModel::class.java)
```

```

        viewModel.state()
            .subscribe(:render) { e -> Log.d(this, javaClass.simpleName, "Error" + e.message) }
        intents()
            .let(viewModel::processIntents)
            .let(compositeDisposable::add)
    }
}
```

```
    override fun onDestroy() {
        compositeDisposable.clear()
        super.onDestroy()
    }

    private fun intents(): Observable<HomeIntent> =
        Observable.merge(
            Observable.just(HomeIntent.LoadDataIntent),
            refreshButton.clicks().map { HomeIntent.RefreshIntent },
            randomNumberClick.clicks().map { HomeIntent.GetRandomNumberIntent })
}
```

```
MVIPresentationViewModel.kt
```

```
package com.uwanteam.mvi.mvi_presentation
import ...
class MVIPresentationViewModel(repo: MVIPresentationRepo) : ViewModel() {
    private val actionProcessor = MVIPresentationActionProcessor(repo)
    private val intentsSubject = PublishSubject.create<HomeIntent>()
    private val states = PublishSubject.create<HomeViewState>()

    init {
        intentsSubject
            .map(intentFilter)
            .map(mapToActions)
            .compose(actionProcessor.processActions)
            .scan(HomeViewState.ProgressViewState, ::reduce)
            .observeOn(AndroidSchedulers.mainThread())
            .subscribe(states)
    }

    fun processIntents(intents: Observable<HomeIntent>): Disposable = intents
        .subscribe(intentsSubject::onNext)

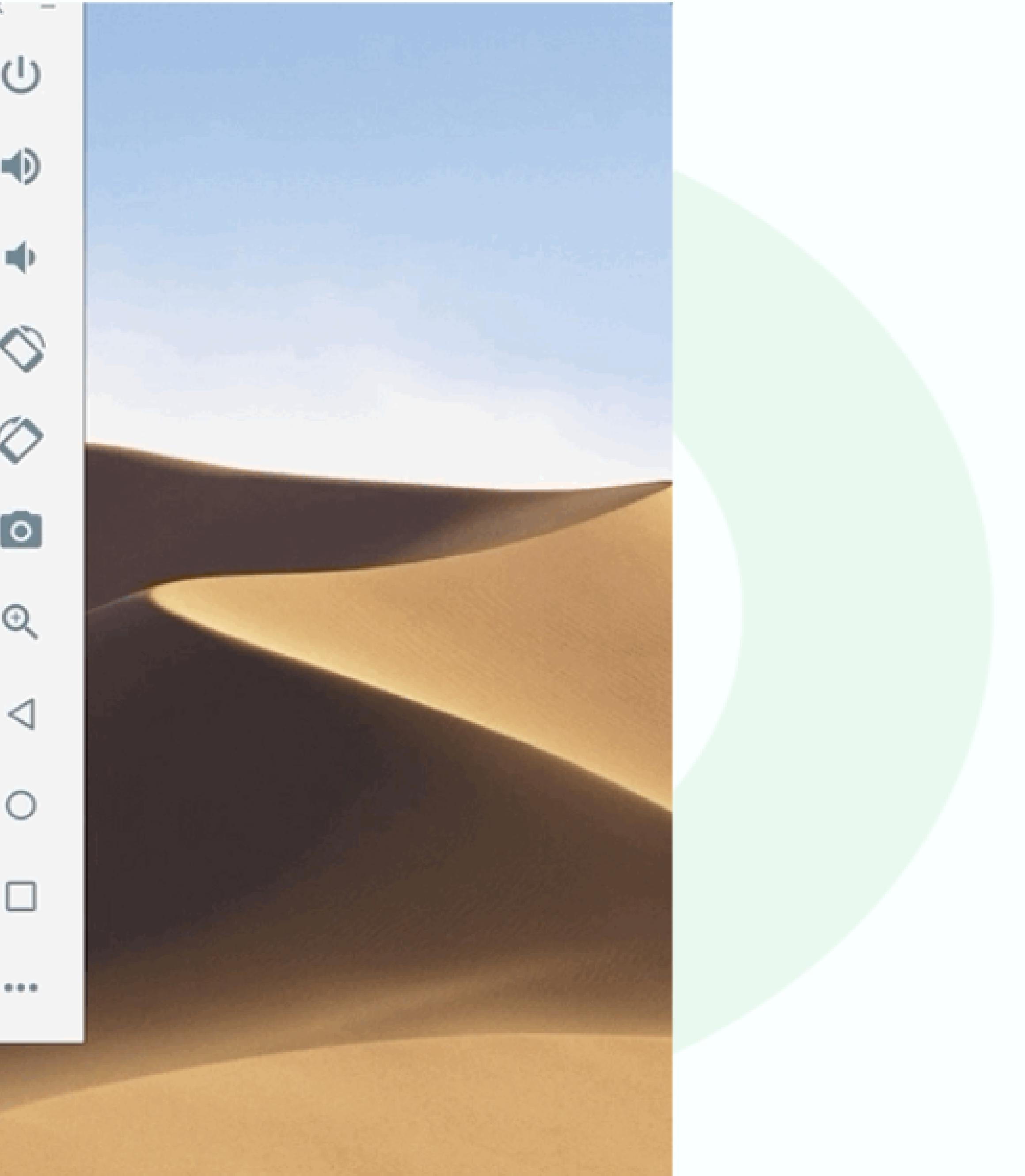
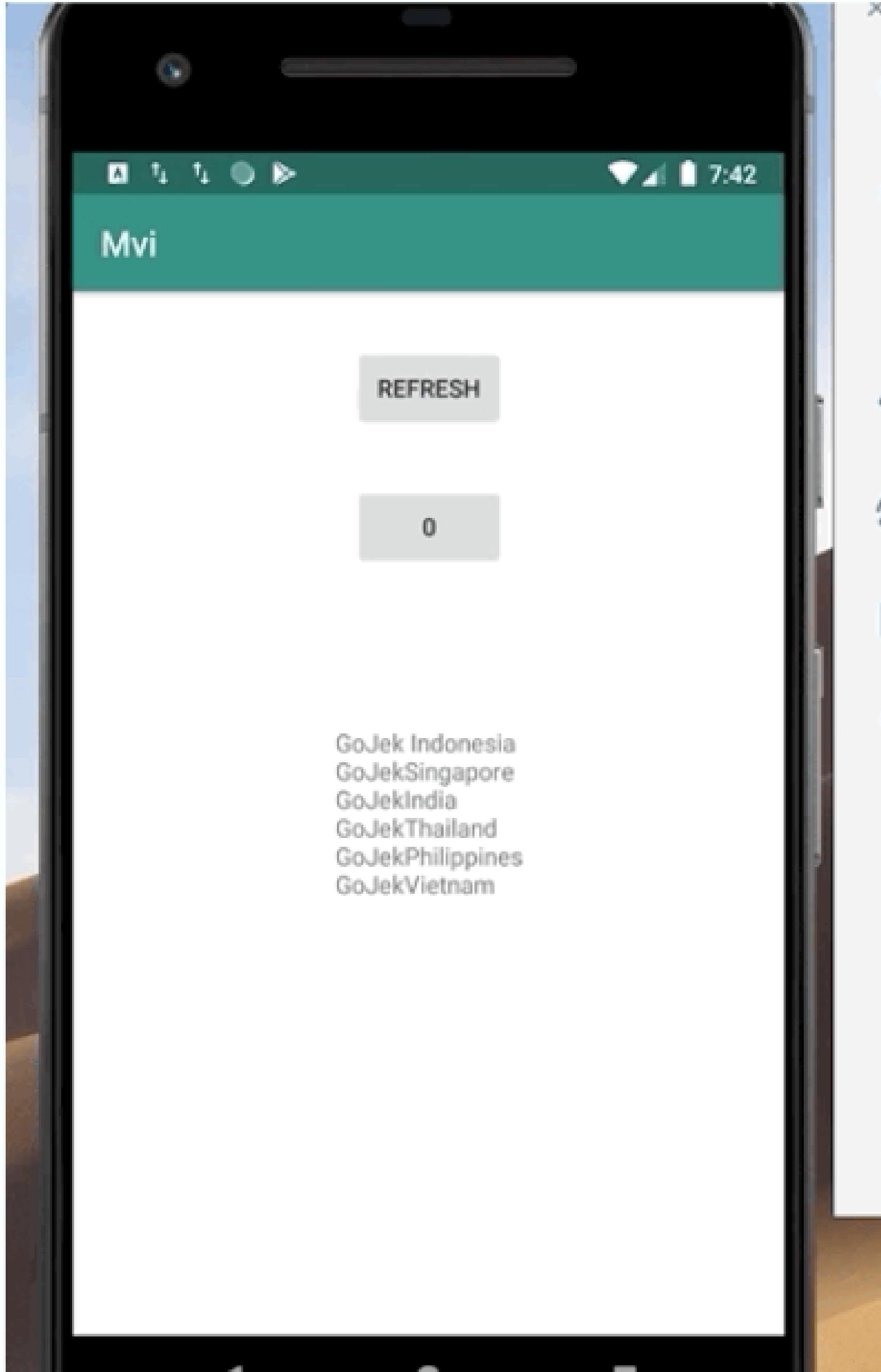
    fun state(): Observable<HomeViewState> = states.hide()

    override fun onCleared() {
        intentsSubject.onComplete()
        states.onComplete()
        super.onCleared()
    }

    private fun intentFilter(initialIntent: HomeIntent, newIntent: HomeIntent): HomeIntent =
        if (newIntent is HomeIntent.LoadDataIntent)
            HomeIntent.GetLastStateIntent
        else
            newIntent

    private fun mapToActions(intent: HomeIntent): HomeActivityAction = when (intent) {
        HomeIntent.LoadDataIntent, HomeIntent.RefreshIntent -> HomeActivityAction.LoadDataAction
        HomeIntent.GetRandomNumberIntent -> HomeActivityAction.GetRandomNumberAction
        HomeIntent.GetLastStateIntent -> HomeActivityAction.GetLastStateAction
    }

    private fun reduce(previousState: HomeViewState, result: HomeActivityResult): HomeViewState =
        when (result) {
            is HomeActivityResult.DataResult -> HomeViewState.DataViewState(result.data)
            HomeActivityResult.FailureResult -> HomeViewState.FailureViewState
            HomeActivityResult.LoadingResult -> HomeViewState.ProgressViewState
            is HomeActivityResult.RandomNumber -> HomeViewState.RandomNumberState(result.randomNumber)
            HomeActivityResult.GetLastState -> previousState
        }
    }
}
```

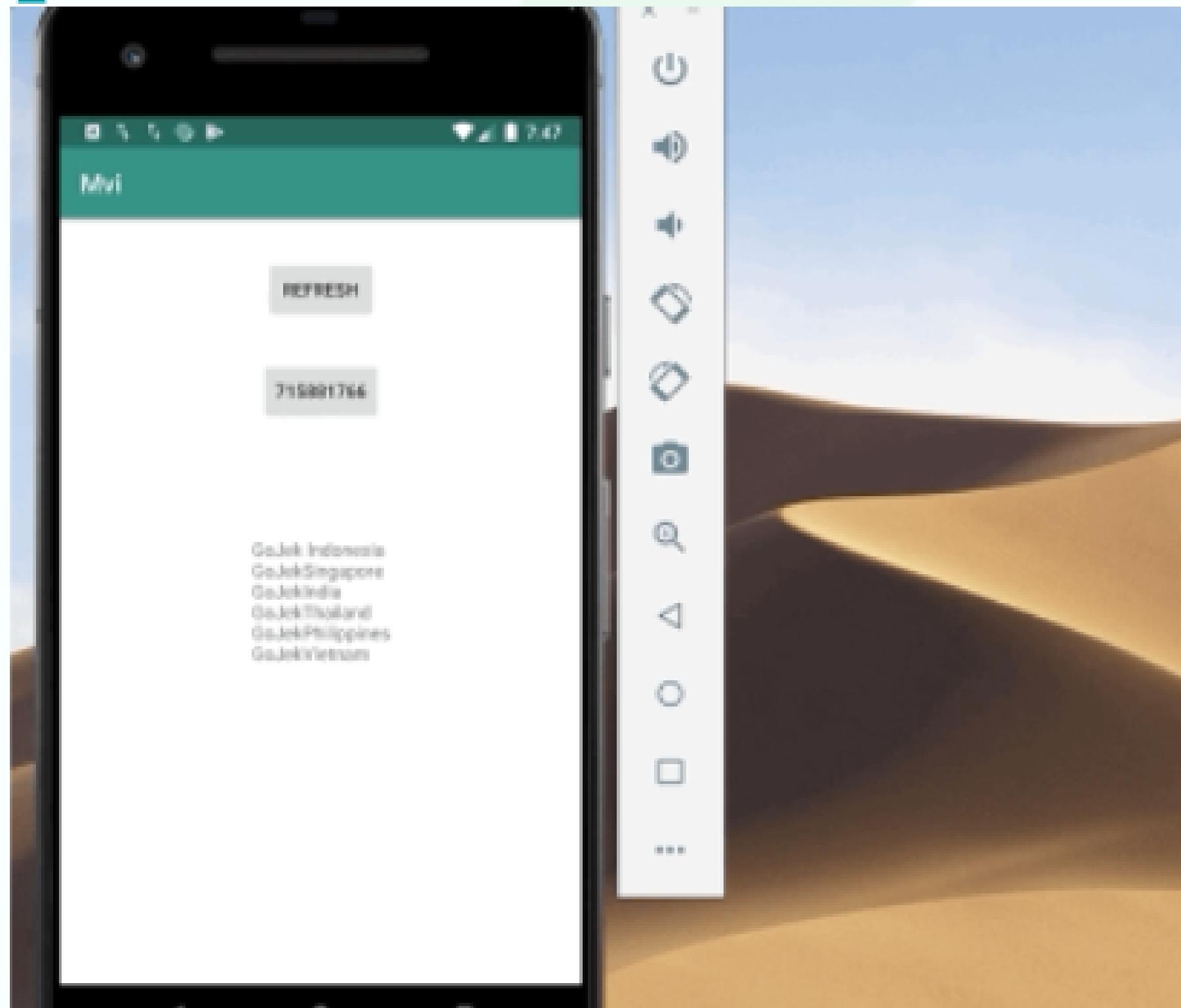


Until Now

- Config Changes are working but with one issue :(

Code:

<https://github.com/Hafiz-Waleed-Hussain/MviExample/tree/presentation-mvi-difficult-implementation-complete-with-one-issue>



Simple solution

```
sealed class HomeViewState {  
    object ProgressViewState : HomeViewState()  
    object FailureViewState : HomeViewState()  
    data class DataViewState(val list: List<String>) : HomeViewState()  
    data class RandomNumberState(val randomNumber: Int) : HomeViewState()  
}  
  
  
data class HomeViewState(  
    val inProgress: Boolean,  
    val isFail: Boolean,  
    val data: List<String>,  
    val randomNumber: Int  
)
```

HomeViewState as data class

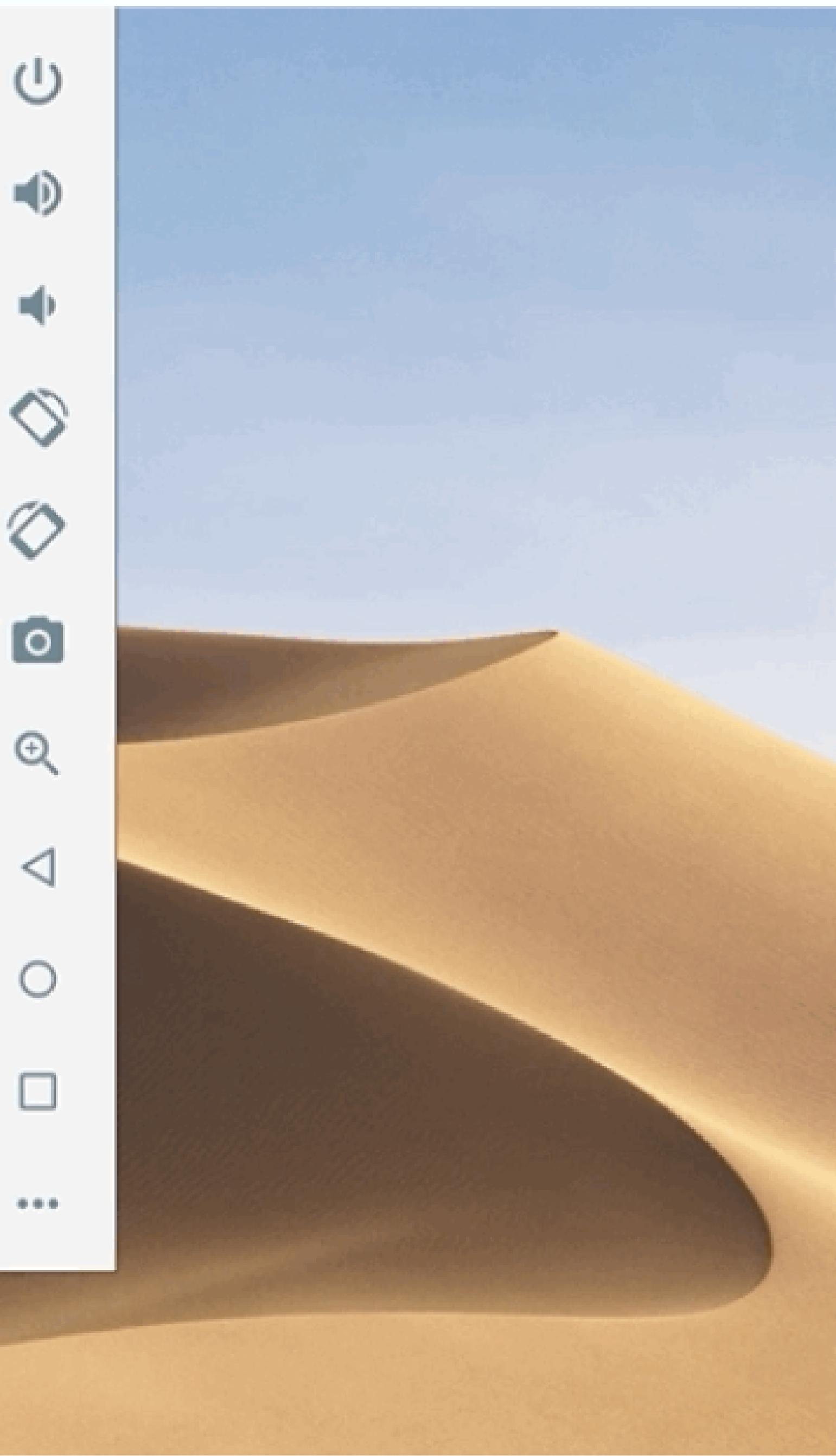
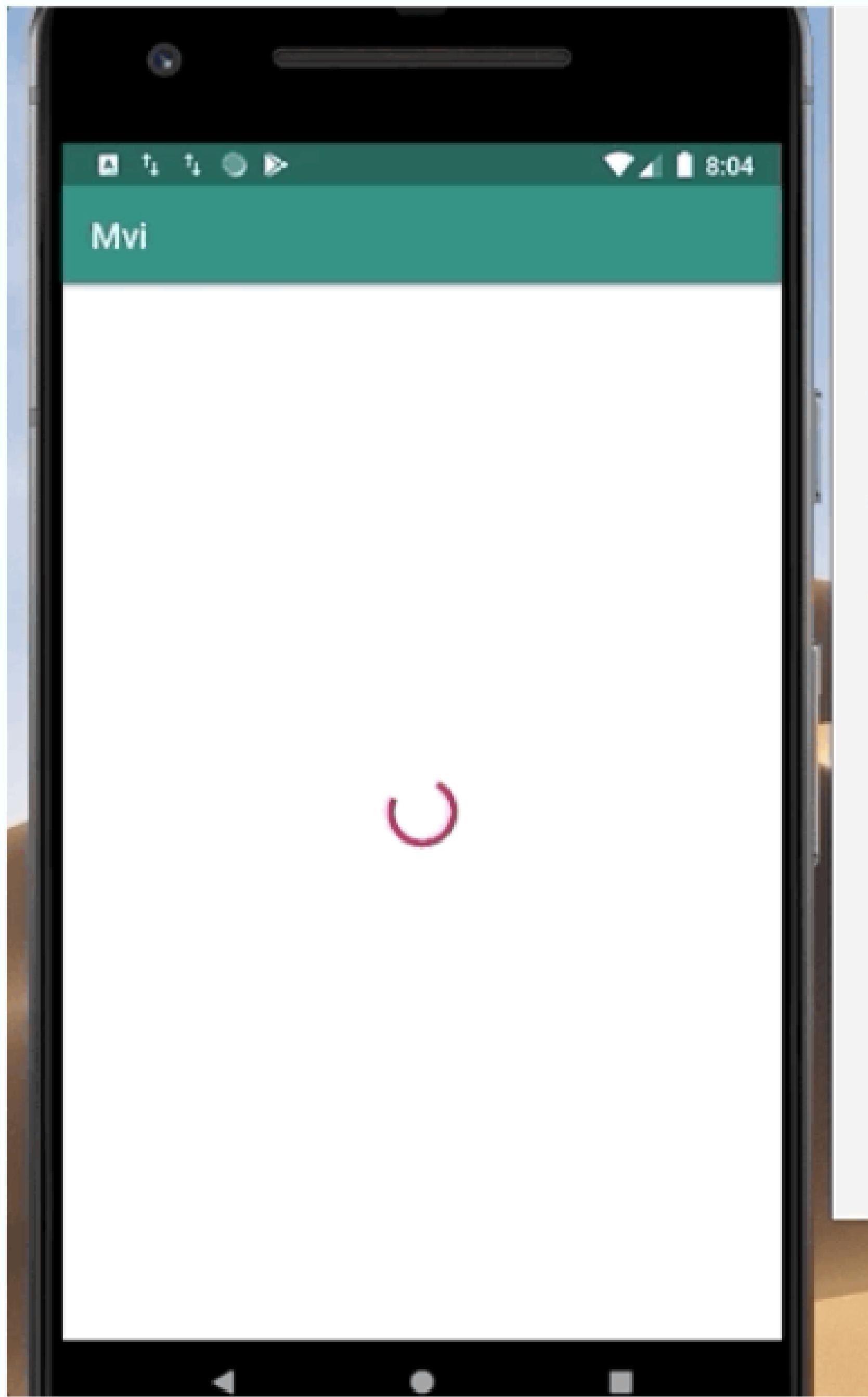
The image shows two code editors in Android Studio. The left editor contains `HomeActivity.kt` and the right editor contains `MVIPresentationViewModel.kt`. Both files are annotated with `com.ttt.ttt`.

HomeActivity.kt:

```
22 package com.ttt.ttt
23
24 class HomeActivity : AppCompatActivity() {
25
26     private val compositeDisposable = CompositeDisposable()
27     private lateinit var viewModel: MVIPresentationViewModel
28
29     override fun onCreate(savedInstanceState: Bundle?) {
30         super.onCreate(savedInstanceState)
31         setContentView(R.layout.activity_home)
32         viewModel =
33             ViewModelProviders.of(
34                 this,
35                 MVIPresentationViewModelFactory())
36             .get(MVIPresentationViewModel::class.java)
37
38         viewModel.state()
39             .subscribe(): render { e -> Log.d(this.javaClass.simpleName, "msg: " + e.message) }
40             .let{compositeDisposable.add()}
41
42         intents()
43             .let{viewModel::processIntents}
44             .let{compositeDisposable.add()}
45
46     override fun onDestroy() {
47         compositeDisposable.clear()
48         super.onDestroy()
49     }
50
51     private fun intents(): Observable<HomeIntent> =
52         Observable.merge(
53             listOf<Observable<HomeIntent>>(
54                 Observable.just(HomeIntent.LoadDataIntent),
55                 refreshButton.clicks().map { HomeIntent.RefreshIntent },
56                 randomNumberClick.clicks().map { HomeIntent.GetRandomNumberIntent })
57
58     private fun render(state: HomeViewState): Unit = when {
59         state.isLoading -> renderLoadingState()
60         state.isError -> renderFailureState(state)
61         else -> renderDataState(state)
62     }
63
64     private fun renderLoadingState(): Unit {
65         progressBar.visibility = View.GONE
66
67         randomNumberClick.visibility = View.VISIBLE
68         refreshButton.visibility = View.VISIBLE
69         dataTextView.visibility = View.VISIBLE
70         state.data.reduce { acc, s -> "acc($s)" }.let{dataTextView.setText(it)}
71         randomNumberClick.text = state.randomNumber.toString()
72     }
73
74     private fun renderFailureState(state: HomeViewState) {
75
76     }
77
78     private fun renderDataState(state: HomeViewState) {
79
80         progressBar.visibility = View.GONE
81
82         randomNumberClick.visibility = View.GONE
83         refreshButton.visibility = View.GONE
84         dataTextView.visibility = View.GONE
85         state.data.reduce { acc, s -> "acc($s)" }.let{dataTextView.setText(it)}
86         randomNumberClick.text = state.randomNumber.toString()
87     }
88
89 }
```

MVIPresentationViewModel.kt:

```
1 package com.ttt.ttt.mvi.mvi_presentation
2
3 import com.ttt.ttt.MVIPresentationActionProcessor
4 import com.ttt.ttt.MVIPresentationIntent
5 import com.ttt.ttt.MVIPresentationViewState
6 import io.reactivex.Observable
7 import io.reactivex.subjects.PublishSubject
8 import io.reactivex.subjects.Subject
9 import io.reactivex.schedulers.Schedulers
10
11 class MVIPresentationViewModel(repo: MVIPresentationRepo) : ViewModel() {
12
13     private val actionProcessor = MVIPresentationActionProcessor(repo)
14     private val intentsSubject = PublishSubject.create<HomeIntent>()
15     private val states = PublishSubject.create<HomeViewState>()
16
17     init {
18         intentsSubject
19             .scan(<:> IntentFilter())
20             .map{mapToActions}
21             .compose(actionProcessor.processActions)
22             .scan(<:> HomeViewState(), ::reduce)
23             .observeOn(Schedulers.mainThread())
24             .subscribe(states)
25     }
26
27
28     fun processIntents(intents: Observable<HomeIntent>): Disposable =
29         intents
30             .subscribe(intentsSubject::onNext)
31
32     fun state(): Observable<HomeViewState> = states.hide()
33
34     override fun onCleared() {
35         intentsSubject.onComplete()
36         states.onComplete()
37         super.onCleared()
38     }
39
40
41     private fun intentFilter(initialIntent: HomeIntent, newIntent: HomeIntent): HomeIntent =
42         if (newIntent is HomeIntent.LoadDataIntent)
43             HomeIntent.GetLastStateIntent
44         else
45             newIntent
46
47
48     private fun mapToActions(intent: HomeIntent): HomeActivityAction =
49         when (intent) {
50             HomeIntent.LoadDataIntent, HomeIntent.RefreshIntent -> HomeActivityAction.LoadDataAction
51             HomeIntent.GetRandomNumberIntent -> HomeActivityAction.GetRandomNumberAction
52             HomeIntent.GetLastStateIntent -> HomeActivityAction.GetLastStateAction
53         }
54
55
56     private fun reduce(previousState: HomeViewState, result: HomeActivityResult): HomeViewState =
57         when (result) {
58             is HomeActivityResult.DataResult -> previousState.copyWith(data = result.data, isLoading = false)
59             is HomeActivityResult.FailureResult -> previousState.copyWith(isError = true, isLoading = false)
60             is HomeActivityResult.LoadingResult -> previousState.copyWith(isLoading = true)
61             is HomeActivityResult.RandomNumber -> previousState.copyWith(randomNumber = result.randomNumber)
62             is HomeActivityResult.GetLastState -> previousState
63         }
64 }
```

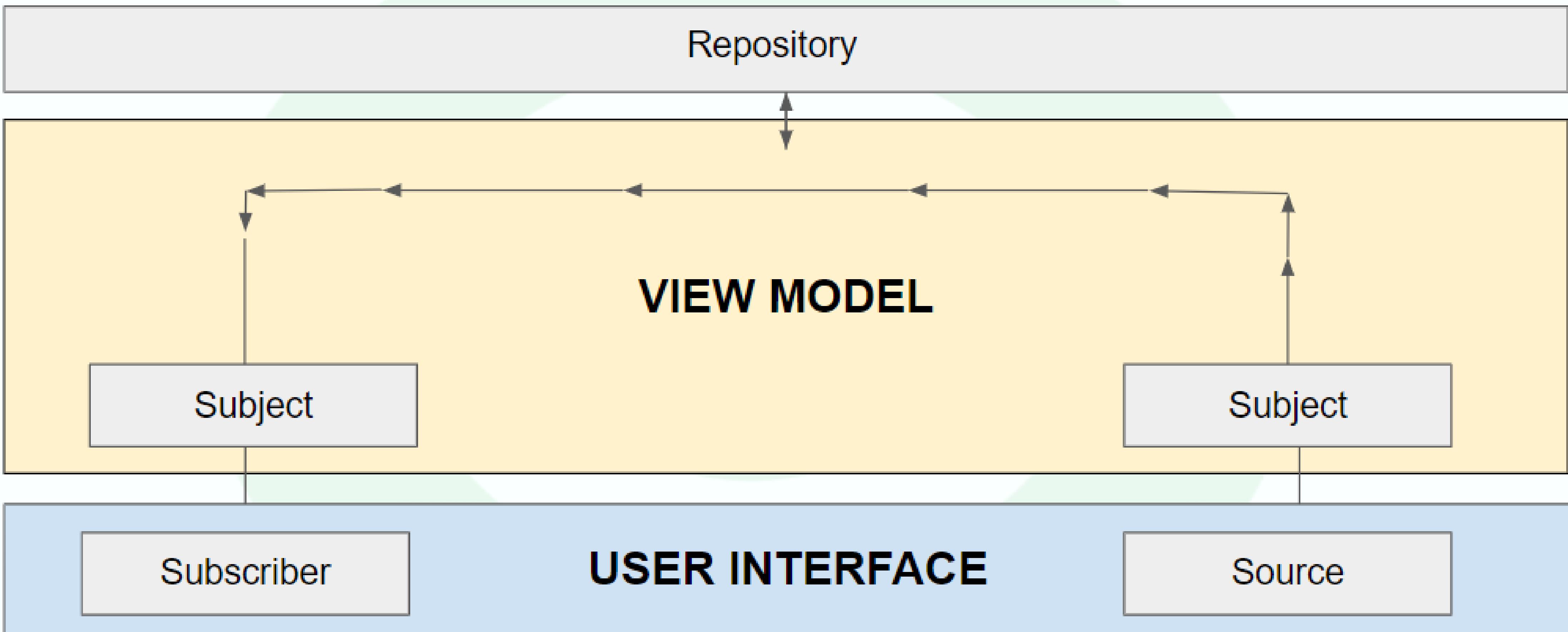


Until Now

- We achieved everything except proper framework api

Code: <https://github.com/Hafiz-Waleed-Hussain/MviExample/tree/presentation-mvi-difficult-implementation-complete>

Before Framework API what about testing



```
class MVIPresentationViewModelTest {
    @Mock
    lateinit var repo: MVIPresentationRepo
    lateinit var actionProcessor: MVIPresentationActionProcessor
    lateinit var viewModel: MVIPresentationViewModel
    lateinit var testObserver: TestObserver<HomeViewState>

    @Test
    fun `when LoadDataIntent successful should return data with`() {
        repo.loadData()
            .let(Mockito::`when`)
            .thenReturn(just(mockData))

        HomeIntent.LoadDataIntent
            .let(::just)
            .cast(HomeIntent::class.java)
            .let(viewModel)::processIntents

        testObserver.values()
            .let(::println)
        testObserver.assertValueAt( index: 0, HomeViewState::inProgress)
        testObserver.assertValueAt( index: 1) { !it.inProgress }
        testObserver.assertValueAt( index: 1) { it.data == mockData }
    }

    @Test
    fun `when LoadDataIntent fail should return failure state`() {
        repo.loadData()
            .let(Mockito::`when`)
            .thenReturn(Observable.error(Exception()))

        HomeIntent.LoadDataIntent
            .let(::just)
            .cast(HomeIntent::class.java)
            .let(viewModel)::processIntents

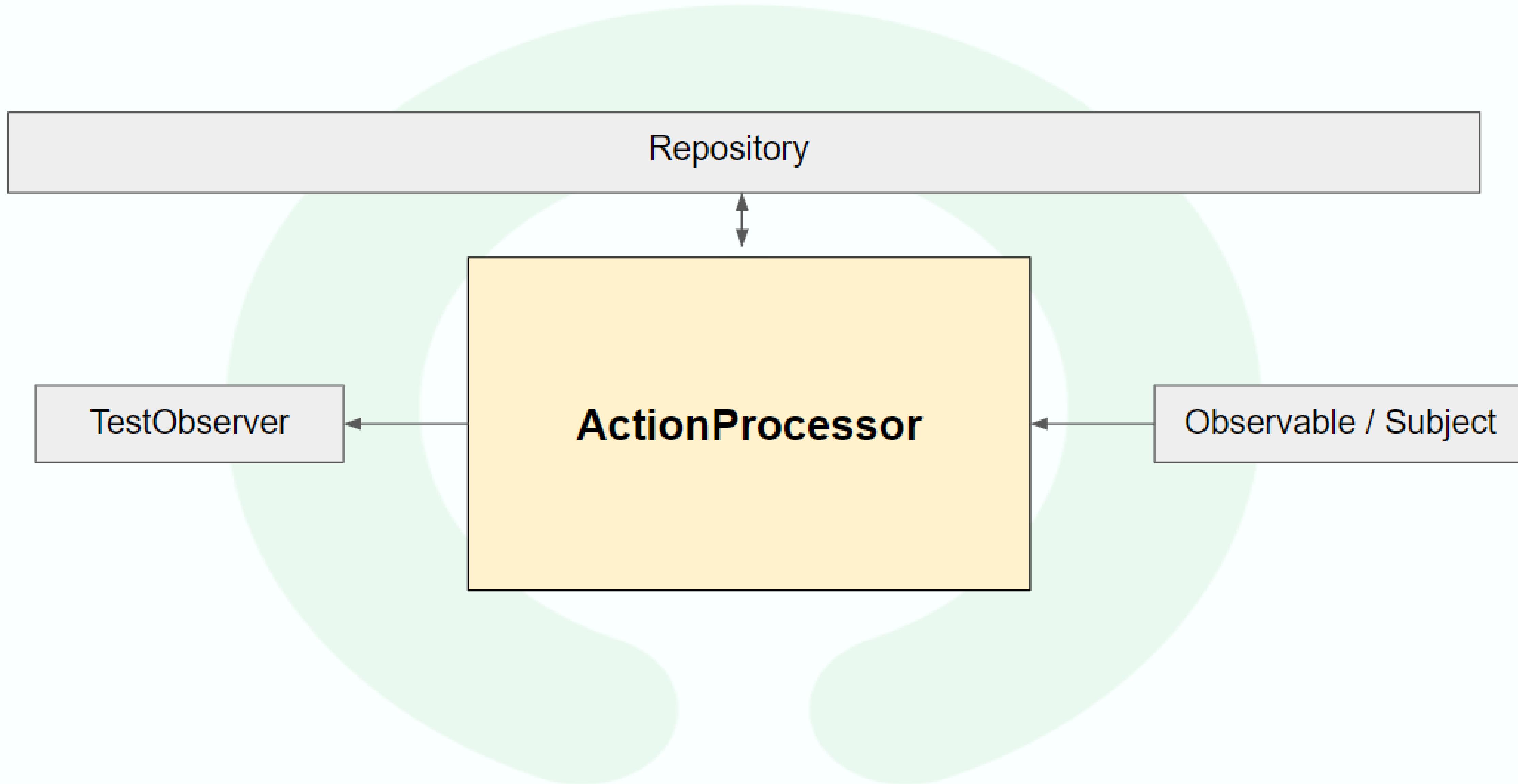
        testObserver.values()
            .let(::println)

        testObserver.assertValueAt( index: 0, HomeViewState::inProgress)
        testObserver.assertValueAt( index: 1) { !it.inProgress }
        testObserver.assertValueAt( index: 1, HomeViewState::isFail)
    }

    @Test
    fun `when RandomNumber click should return value`() {
        HomeIntent.GetRandomNumberIntent
            .let(::just)
            .cast(HomeIntent::class.java)
            .let(viewModel)::processIntents

        testObserver.values()
            .let(::println)

        testObserver.assertValueAt( index: 0) { it.randomNumber != 0 }
    }
}
```



```
class MVIPresentationActionProcessorTest {

    @Mock
    lateinit var repo: MVIPresentationRepo
    lateinit var actionProcessor: MVIPresentationActionProcessor
    private val mockData = listOf(...)

    companion object {...}

    @Before
    fun setUp() {...}

    @Test
    fun `when LoadDataAction success should return DataResult`() {
        repo.loadData()
            .let(Mockito::`when`)
            .thenReturn(mockData.let(::just))

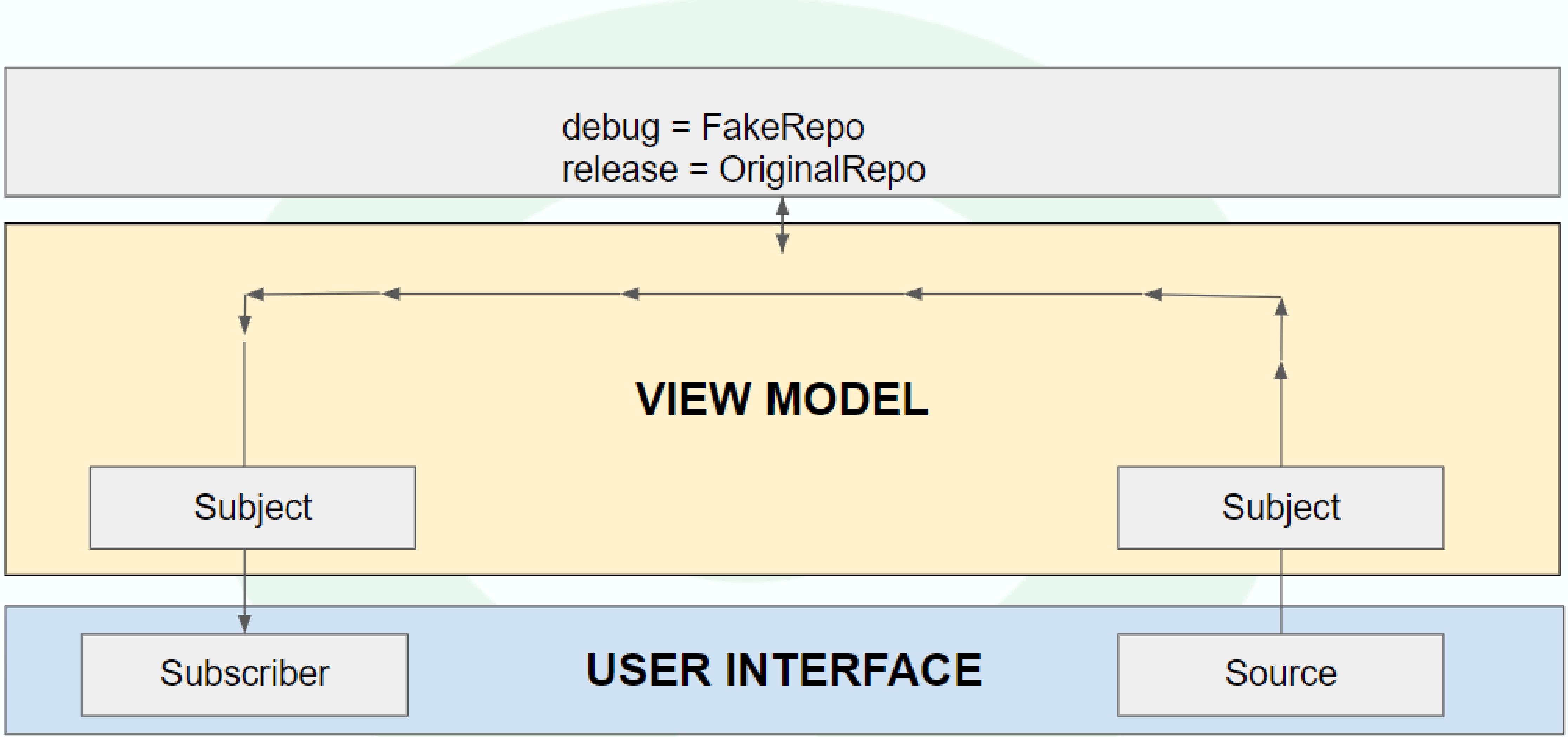
        val testObserver = HomeActivityAction.LoadDataAction.let(::just)
            .compose(actionProcessor.processActions)
            .test()

        testObserver.values()
            .let(::println)
        testObserver.assertValueAt( index: 0, HomeActivityResult.LoadingResult)
        testObserver.assertValueAt( index: 1) { it == HomeActivityResult.DataResult(mockData) }
    }

    @Test
    fun `when LoadDataAction failed should return FailedResult`() {
        repo.loadData()
            .let(Mockito::`when`)
            .thenReturn(error(Exception()))

        val testObserver = HomeActivityAction.LoadDataAction.let(::just)
            .compose(actionProcessor.processActions)
            .test()

        testObserver.values()
            .let(::println)
        testObserver.assertValueAt( index: 0, HomeActivityResult.LoadingResult)
        testObserver.assertValueAt( index: 1, HomeActivityResult.FailureResult)
    }
}
```



```
@RunWith(AndroidJUnit4::class)
class HomeActivityTest {

    @get:Rule
    var homeActivityTestRule = ActivityTestRule(
        HomeActivity::class.java,
        initialTouchMode: true,
        launchActivity: true
    )

    @Test
    fun whenLaunchShouldShowProgressBar() {
        onView(withId(R.id.progressBar))
            .check(matches(isDisplayed()))

       (withId(R.id.progressBar)
            .let(::onView)
            .check(matches(isDisplayed()))
    }

    @Test
    fun whenRandomButtonClickShouldChangeButtonValue() {

       (withId(R.id.progressBar)
            .let(::onView)
            .check(matches(isDisplayed())))
        Thread.sleep( millis: 4000)

       (withId(R.id.randomNumberClick)
            .let(::onView)
            .check(matches(withText( text: "0"))))

       (withId(R.id.randomNumberClick)
            .let(::onView)
            .perform(ViewActions.click()))

       (withId(R.id.randomNumberClick)
            .let(::onView)
            .check(matches(not(withText( text: "0")))))

    }
}
```

Framework API

```
MviIntent.kt ×  
1 package com.uwantolearn.mvi.base  
2  
3 interface MviIntent  
4
```

```
MviAction.kt ×  
Close. Alt-click to close others n.mvi.base  
1 interface MviAction  
2
```

```
MviResult.kt ×  
1 package com.uwantolearn.mvi.base  
2  
3 interface MviResult  
4
```

```
MviViewState.kt ×  
1 package com.uwantolearn.mvi.base  
2  
3 interface MviViewState  
4
```

```
MviView.kt ×  
1 package com.uwantolearn.mvi.base  
2  
3 import io.reactivex.Observable  
4  
5  
6 interface MviView<I : MviIntent, S : MviViewState> {  
7     fun intents(): Observable<I>  
8     fun render(state: S)  
9 }
```

```
MviViewModel.kt ×  
1 package com.uwantolearn.mvi.base  
2  
3 import ...  
4  
5  
6 interface MviViewModel<I : MviIntent, S : MviViewState> {  
7     fun processIntents(intents: Observable<I>): Disposable  
8     fun state(): Observable<S>  
9 }
```

After applying API's

```
sealed class HomeIntent : MviIntent {
    object LoadDataIntent : HomeIntent()
    object RefreshIntent : HomeIntent()
    object GetRandomNumberIntent : HomeIntent()
    object GetLastStateIntent : HomeIntent()
```

```
data class HomeViewState(
    val inProgress: Boolean = true,
    val isFail: Boolean = false,
    val data: List<String> = listOf(),
    val randomNumber: Int = 0
) : MviViewState
```

```
sealed class HomeActivityResult : MviResult {
    data class DataResult(val data: List<String>) : HomeActivityResult()
    data class RandomNumber(val randomNumber: Int) : HomeActivityResult()
    object GetLastState : HomeActivityResult()
    object FailureResult : HomeActivityResult()
    object LoadingResult : HomeActivityResult()
```

```
sealed class HomeActivityAction : MviAction {
    object LoadDataAction : HomeActivityAction()
    object GetRandomNumberAction : HomeActivityAction()
    object GetLastStateAction : HomeActivityAction()
}
```

After applying API's

```
class HomeActivity : MviView<HomeIntent, HomeViewState>, AppCompatActivity() {  
  
    private val compositeDisposable = CompositeDisposable()  
    private lateinit var viewModel: MVIPresentationViewModel  
  
    override fun onCreate(savedInstanceState: Bundle?) {...}  
  
    override fun onDestroy() {...}  
  
    override fun intents(): Observable<HomeIntent> =  
        Observable.merge(  
            listOf<Observable<HomeIntent>>(  
                Observable.just(HomeIntent.LoadDataIntent),  
                refreshButton.clicks().map { HomeIntent.RefreshIntent },  
                randomNumberClick.clicks().map { HomeIntent.GetRandomNumberIntent }  
            )  
  
    override fun render(state: HomeViewState): Unit = when {  
        state.inProgress -> renderLoadingState()  
        state.isFail -> renderFailureState(state)  
        else -> renderDataState(state)  
    }  
  
    private fun renderLoadingState() {...}  
    private fun renderFailureState(state: HomeViewState) {...}  
    private fun renderDataState(state: HomeViewState) {...}  
}
```

```
class MVIPresentationViewModel(repo: MVIPresentationRepo) : ViewModel(),  
    MviViewModel<HomeIntent, HomeViewState> {  
  
    private val actionProcessor = MVIPresentationActionProcessor(repo)  
    private val intentsSubject = PublishSubject.create<HomeIntent>()  
    private val states = PublishSubject.create<HomeViewState>()  
  
    init {...}  
  
    override fun processIntents(intents: Observable<HomeIntent>): Disposable = intents  
        .subscribe(intentsSubject::onNext)  
  
    override fun states(): Observable<HomeViewState> = states.hide()  
  
    override fun onCleared() {...}  
  
    private fun intentFilter(initialIntent: HomeIntent, newIntent: HomeIntent): HomeIntent =  
        if (newIntent is HomeIntent.LoadDataIntent)  
            HomeIntent.GetLastStateIntent  
        else  
            newIntent  
  
    private fun mapToActions(intent: HomeIntent): HomeActivityAction = when (intent) {  
        HomeIntent.LoadDataIntent, HomeIntent.RefreshIntent -> HomeActivityAction.LoadDataAction  
        HomeIntent.GetRandomNumberIntent -> HomeActivityAction.GetRandomNumberAction  
        HomeIntent.GetLastStateIntent -> HomeActivityAction.GetLastStateAction  
    }  
  
    private fun reduce(previousState: HomeViewState, result: HomeActivityResult): HomeViewState =  
        when (result) {  
            is HomeActivityResult.DataResult -> previousState.copy(  
                data = result.data,  
                inProgress = false  
            )  
            HomeActivityResult.FailureResult -> previousState.copy(  
                isFail = true,  
                inProgress = false  
            )  
            HomeActivityResult.LoadingResult -> previousState.copy(inProgress = true)  
            is HomeActivityResult.RandomNumber -> previousState.copy(randomNumber = result.randomNumber)  
            HomeActivityResult.GetLastState -> previousState  
        }  
}
```

Until Now

- Everything is perfect.

Code:

<https://github.com/Hafiz-Waleed-Hussain/MviExample/tree/presentation-mvi-with-framework>

Improve API's (For now Activity Support only)

```
abstract class MviBaseView<I : MviIntent, A : MviAction, S : MviViewState, R : MviResult> :  
    MviView<I, S>, AppCompatActivity() {  
  
    private val compositeDisposable = CompositeDisposable()  
    abstract val viewModel: MviBaseViewModel<I, A, S, R>  
  
    abstract val layoutId: Int  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(layoutId)  
    }  
  
    override fun onStart() {  
        super.onStart()  
        viewModel.state()  
            .subscribe(::render) { e ->  
                Log.d(  
                    this.javaClass.getSimpleName,  
                    msg: "Error" + e.message  
                )  
            }  
            .let(compositeDisposable::add)  
    intents()  
        .let(viewModel::processIntents)  
        .let(compositeDisposable::add)  
    }  
  
    override fun onStop() {  
        compositeDisposable.clear()  
        super.onStop()  
    }  
}
```

```
abstract class MviBaseViewModel<I : MviIntent, A : MviAction, S : MviViewState, R : MviResult>{  
    actionProcessor: MviBaseActionProcessor<A, R>,  
    initialState: S  
} :  
ViewModel(), MviViewModel<I, S> {  
  
private val intentsSubject = PublishSubject.create<I>()  
private val states = PublishSubject.create<S>()  
  
init {  
    intentsSubject  
        .scan(::intentFilter)  
        .map(::mapToActions)  
        .compose(actionProcessor.processActions)  
        .scan(initialState, ::reduce)  
        .observeOn(AndroidSchedulers.mainThread())  
        .subscribe(states)  
}  
  
override fun processIntents(intents: Observable<I>): Disposable = intents  
    .subscribe(intentsSubject::onNext)  
  
override fun state(): Observable<S> = states.hide()  
  
override fun onCleared() {  
    intentsSubject.onComplete()  
    states.onComplete()  
    super.onCleared()  
}  
  
abstract fun intentFilter(initialIntent: I, newIntent: I): I  
abstract fun mapToActions(intent: I): A  
abstract fun reduce(previousState: S, result: R): S  
}  
  
abstract class MviBaseActionProcessor<A : MviAction, R : MviResult> {  
    abstract val processActions: ObservableTransformer<A, R>  
}
```

```
sealed class OrderIntent : MviIntent
sealed class OrderAction : MviAction
sealed class OrderResult : MviResult
data class OrderState(val orderId: Int = 0) : MviViewState
```

Implementation of a new feature by using framework api

```
class OrderActivity : MviBaseView<OrderIntent, OrderAction, OrderState, OrderResult>() {

    override val viewModel: MviBaseViewModel<OrderIntent, OrderAction, OrderState, OrderResult>
        get() = OrderViewModel()

    override
    val layoutId: Int
        get() = R.layout.activity_home

    override fun intents(): Observable<OrderIntent> {
    }

    override fun render(state: OrderState) {
    }

}
```

```
class OrderActionProcessor : MviBaseActionProcessor<OrderAction, OrderResult>() {

    override val processActions: ObservableTransformer<OrderAction, OrderResult>
        get() = TODO()

}
```

```
class OrderViewModel() : MviBaseViewModel<OrderIntent, OrderAction, OrderState, OrderResult>(
    OrderActionProcessor(),
    OrderState()
) {
    override fun intentFilter(initialIntent: OrderIntent, newIntent: OrderIntent): OrderIntent {
    }

    override fun mapToActions(intent: OrderIntent): OrderAction {
    }

    override fun reduce(previousState: OrderState, result: OrderResult): OrderState {
    }
}
```

Only write the code in given methods. The framework will take care for everything. (Not production-ready but maybe we can use :))

Benefits

- Predictable
- Debuggable (Instead QA give us steps how to fix not how to reproduce)
- Crash reports are more readable
- Onboarding is easy
- No need for fancy frameworks to unit test
- Code reviews are easy
- Design pattern with Kotlin is self-sufficient to take care issues at dev time
(Live Demo)

Code:

<https://github.com/Hafiz-Waleed-Hussain/MviExample/tree/presentation-mvi-with-framework-complete-with-tests>