Ribs, MVI, Compose — Buzzwords or an approach of the future?

Artur Badretdinov

Artur Badretdinov

— 3 years as a Digital Nomad[dn]

— 7 years as a Software Engineer

— Android Academy org [aa]

— GDG friend

Twitter: Arturstwit

Telegram: Gaket

[dn] "Как на удалёнке жить хорошо"

[aa] Android Fundamentals: free online course

9:41

Squire

Sign in

You are scheduled for a men's clipper cut in 4 days.

UPCOMING SEPT 30, 9AM

Fellow Barbershop with Alexia C.

Add to calendar

View more

OCT 4, 11:30AM

Fellow Barbershop

9:41

Choose a time

September 2019

S	M	T	W	T	F	S
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

Suggested for you

9:30am | 12:30pm | 12:30pm

1:00am | 2:00pm | 3:30pm

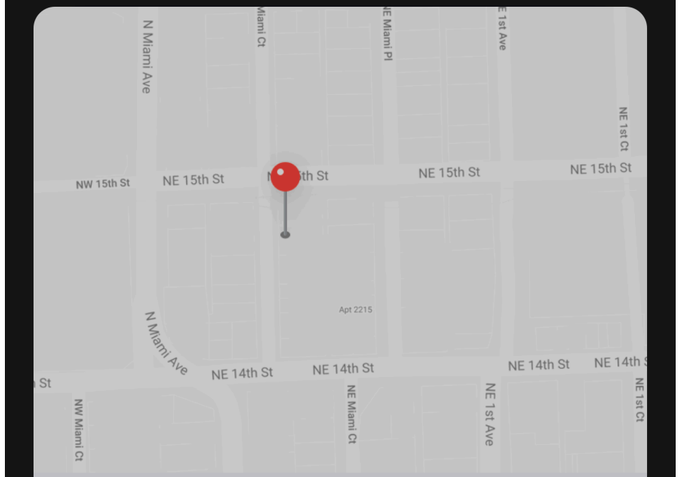
All Available Times

MORNING

8:30am | 9:30pm | 10:00am

11:00am

9:41



Fellow Barbershop

18 Fulton Street, New York, NY 10038
(347) 506-2115

Get directions

Barber Alexia

Add a tip 15% (~$10.00)

10% | 15% | 20%

25% | None | Custom

Add tip

Context

1. 2 devs on a project
2. 3 Months - from zero to MVP with 20 screens
3. 6 months - 30 screens + improvements
4. Near future - 3-5 devs per project, long-term support

MVP - Success!

What's next?

What are the main things composing an app?

App's "layers"

1. Visuals on a Screen
2. Data on a Screen
3. Navigation between Screens
4. App's data and logic

Let's experiment!

Goals

1. Less code[1]
2. Low coupling between the four layers
3. More tests

[1] Jonathan Blow about complexity: https://youtu.be/ZSRHeXYDLko

App's "layers"

1. Visuals on a Screen
2. Data on a Screen
3. Navigation between Screens
4. App's data and logic

Visuals

1. Android Views
2. Facebook Litho
3. Jetpack Compose
4. ~~Anko~~

Litho

1. Declarative
2. Extracts inflation to background[2]

[2] Sergey Ryabov about Litho (RU): https://habr.com/ru/company/jugru/blog/507130/

Jetpack Compose

Jetpack Compose is a modern toolkit designed to simplify UI development

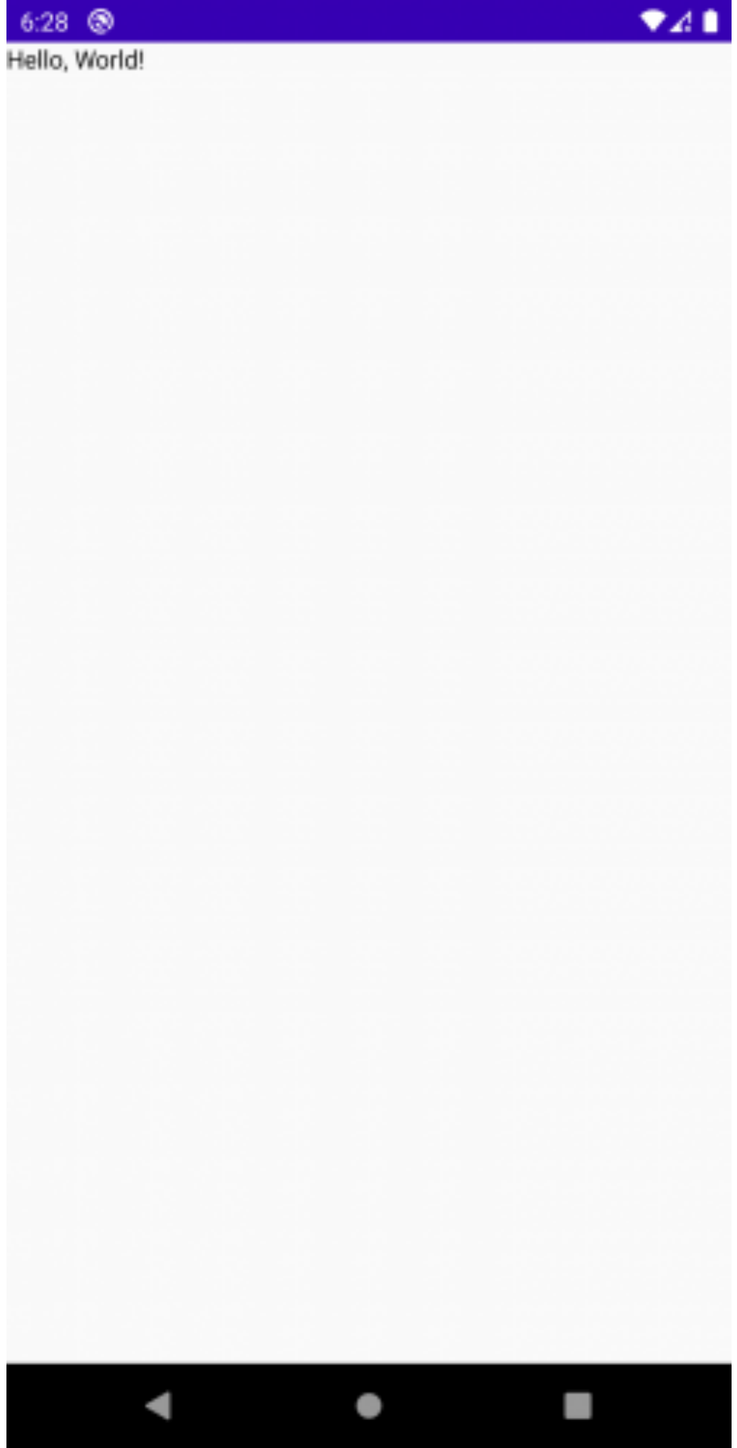
— Compose Codelab

Compose in a nutshell

1. Throw away layouts
2. Throw away resource files
3. Do everything in code

Hello world

```
// Old way
val textView = TextView(this)
textView.setText("Hello world")
setContentView(textView)

// New way
setContent { Text("Hello World") }
```



Goals

1. **Less code**
2. More tests
3. Low coupling between the four App's Components

Goals

1. ~~**Less code**~~
2. More tests
3. Low coupling between the four App's Components

Less code

1. No separation between creating a view and "binding" it

   — Saving Mind Fuel[16] [17]

2. Simpler Recycler Views[3]

[16] Джедайские техники

[17] Jedi Techniques

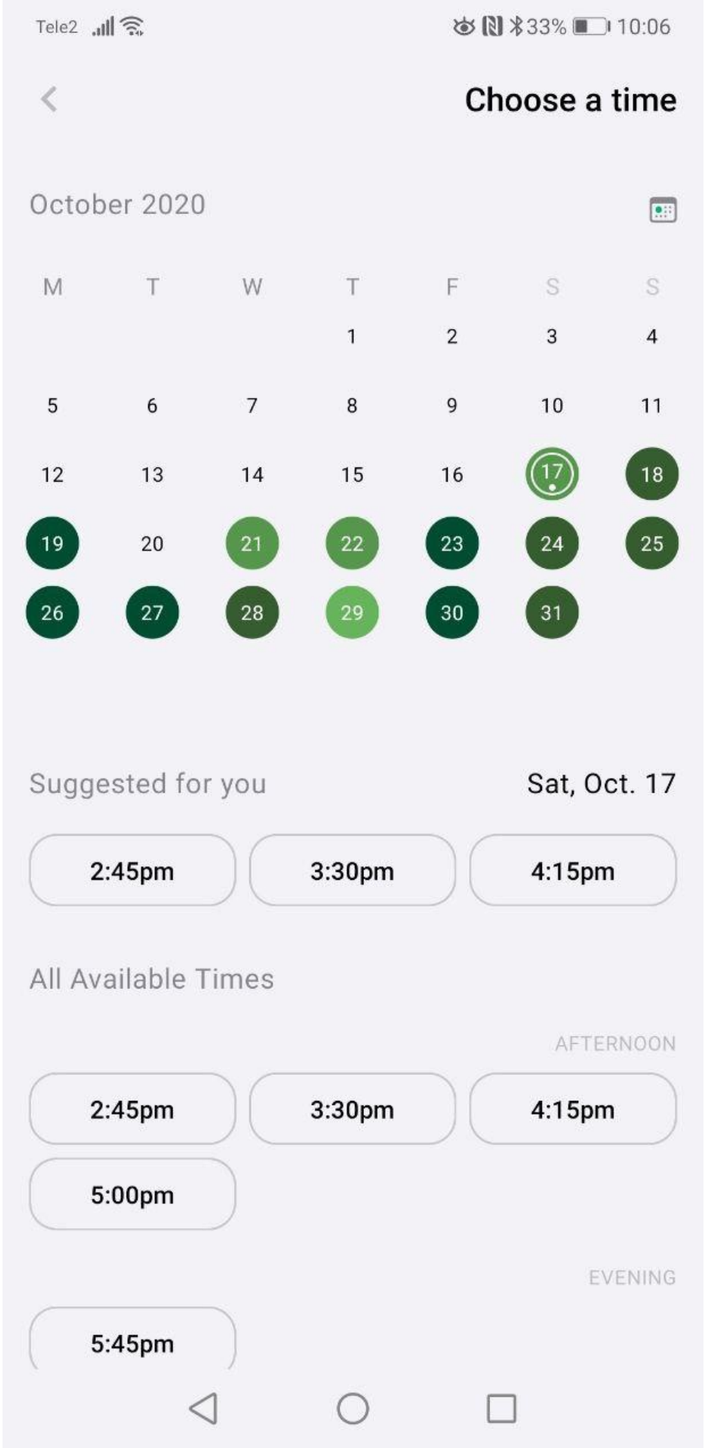
[3] RedMadRobot about lists: https://habr.com/ru/company/redmadrobot/blog/428525/

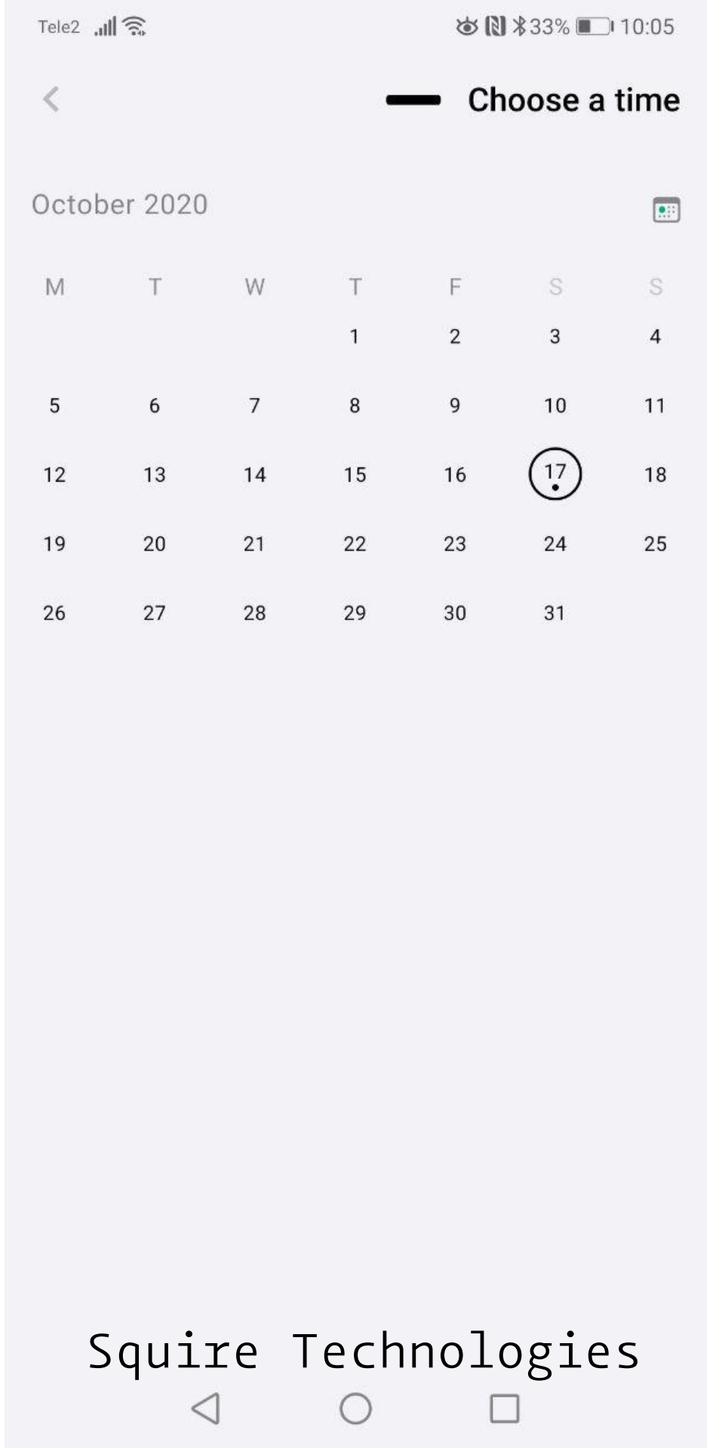
A typical screen

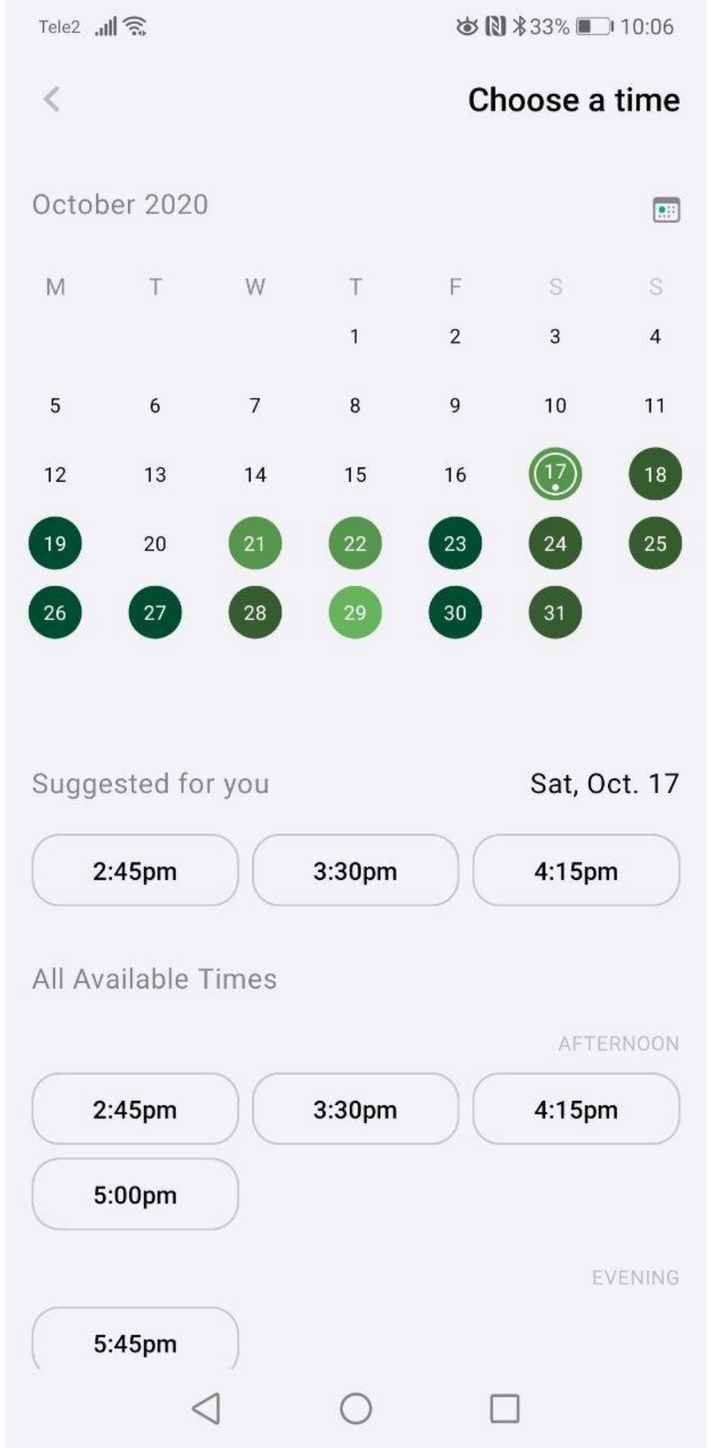
A typical screen

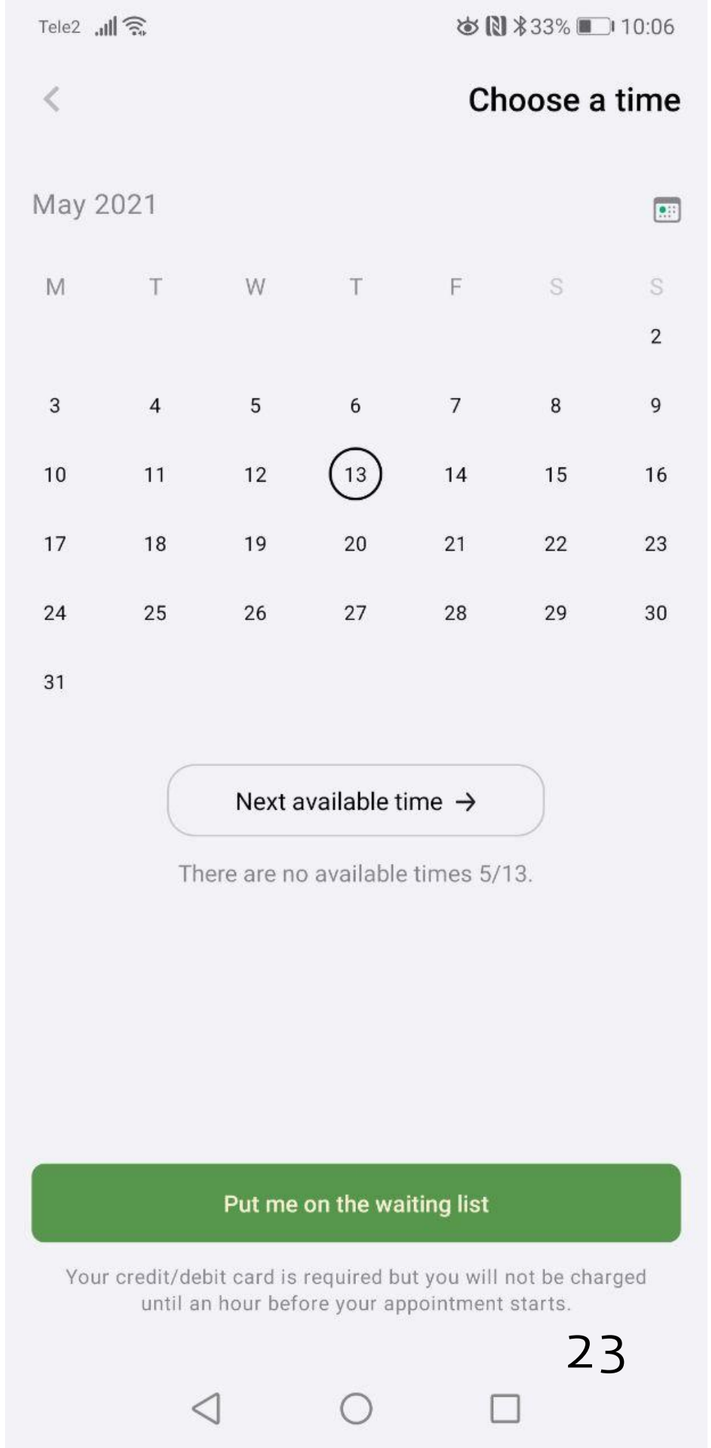
```
val materialBlue700 = Color(0xFF1976D2)
Scaffold(
    scaffoldState = ScaffoldState(DrawerState.Opened),
    topBar = {
        TopAppBar(title = { Text("TopAppBar") },
            backgroundColor = materialBlue700)
    },
    floatingActionButton = { FloatingActionButton() },
    drawerContent = { Drawer() },
    bodyContent = { Content() },
    bottomBar = { BottomAppBar() }
)
```

Different states




Tele2
33% 10:05
Choose a time
October 2020
M T W T F S S
1 2 3 4
5 6 7 8 9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31


Tele2
33% 10:06
Choose a time
October 2020
M T W T F S S
1 2 3 4
5 6 7 8 9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31
Suggested for you
Sat, Oct. 17
2:45pm
3:30pm
4:15pm
All Available Times
AFTERNOON
2:45pm
3:30pm
4:15pm
5:00pm
EVENING
5:45pm


Tele2
33% 10:06
Choose a time
May 2021
M T W T F S S
2
3 4 5 6 7 8 9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
31
Next available time →
There are no available times 5/13.
Put me on the waiting list
Your credit/debit card is required but you will not be charged until an hour before your appointment starts.

```
fun onCreate() {
    setContent {
        ChooseTimeScreen(initialState)
    }
}

@Composable
fun ChooseTimeScreen(state: ScreenState) {
  when (state) {
    is Loading -> ChooseTimeLoading()
    is Success -> ChooseTime(state.event)
    is Error -> ChooseTimeError(state.error)
  }
}
```

```
fun onCreate() {
    setContent {
        ChooseTimeScreen(initialState)
    }
}

@Composable
fun ChooseTimeScreen(state: ScreenState) {
  when (state) {
    is Loading -> ChooseTimeLoading()
    is Success -> ChooseTime(state.event)
    is Error -> ChooseTimeError(state.error)
  }
}
```

Goals. Final version for Compose

1. ~~Less code~~
2. **Low coupling between the four App's Components**
3. More tests

Goals. Final version for Compose

1. ~~Less code~~
2. **~~Low coupling between the four App's Components~~**
3. More tests?

```
fun onCreate() {
    setContent {
        ChooseTimeScreen(initialState)
    }
}

@Composable
fun ChooseTimeScreen(state: ScreenState) {
  when (state) {
    is Loading -> ChooseTimeLoading()
    is Success -> ChooseTime(state.event)
    is Error -> ChooseTimeError(state.error)
  }
}
```

Where is my State?

App's "layers"

1. Visuals on a Screen
2. Data on a Screen
3. Navigation between Screens
4. App's data and logic

MVP

```
intervace View {
    fun showLoader()
    fun hideLoader()
    fun showError(msg: String)
    fun hideError()
    fun showContent(events: List<Event>)
}
```

MVVM

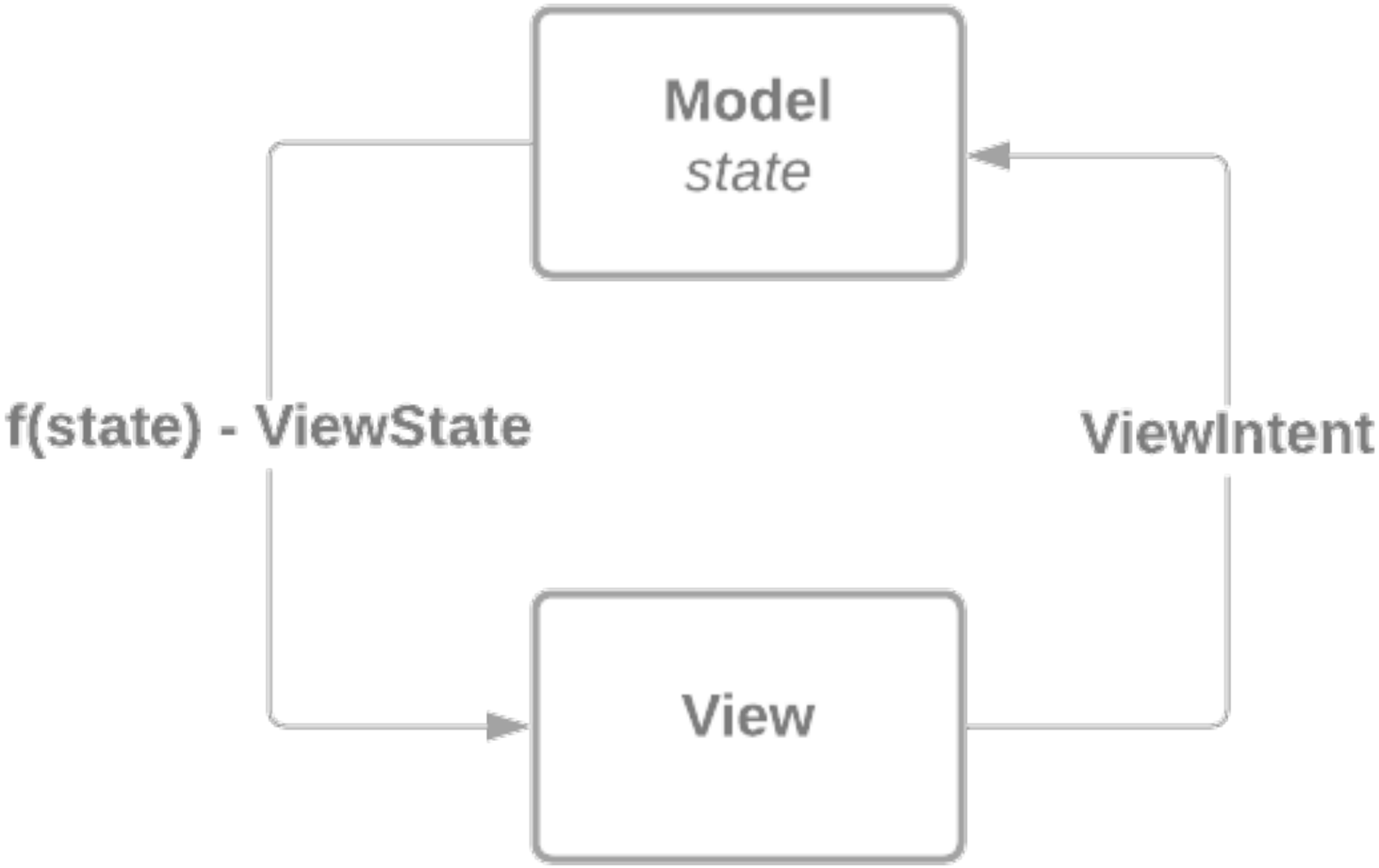
```
class View {
  init {
    loaderStream.observe { handleLoader(it) }
    errorStream.observe { handleError(it) }
    contentStream.observe { handleContent(it) }
  }
}
```

MVI

```
class View {
  init {
    uiStateSteam.observe { renderUi(it) }
  }
}
```

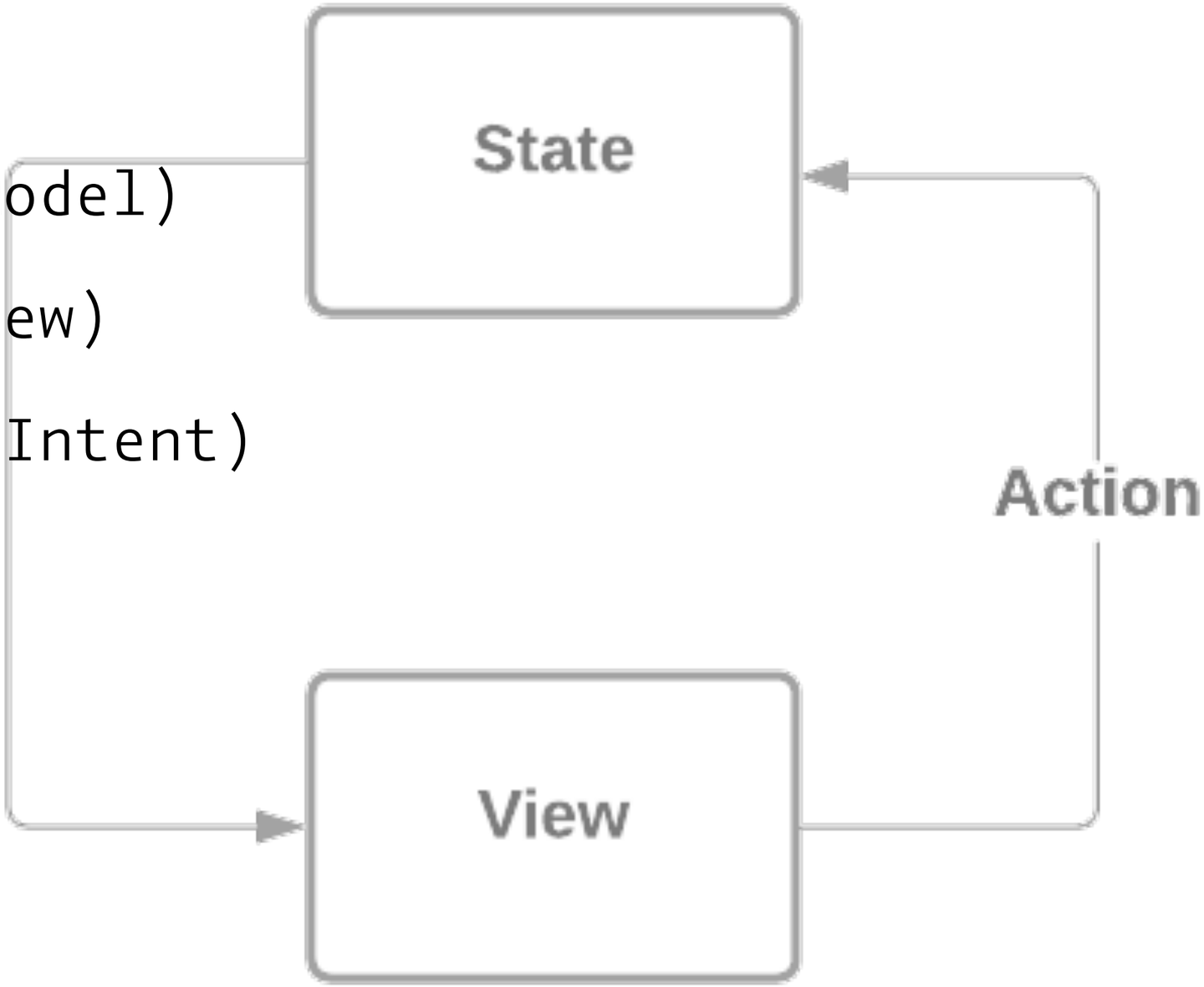
```
@Composable
fun ChooseTimeScreen(state: ScreenState) {}

class View {
  init {
    uiStateSteam.observe { state -> ChooseTimeScreen(state) }
  }
}
```

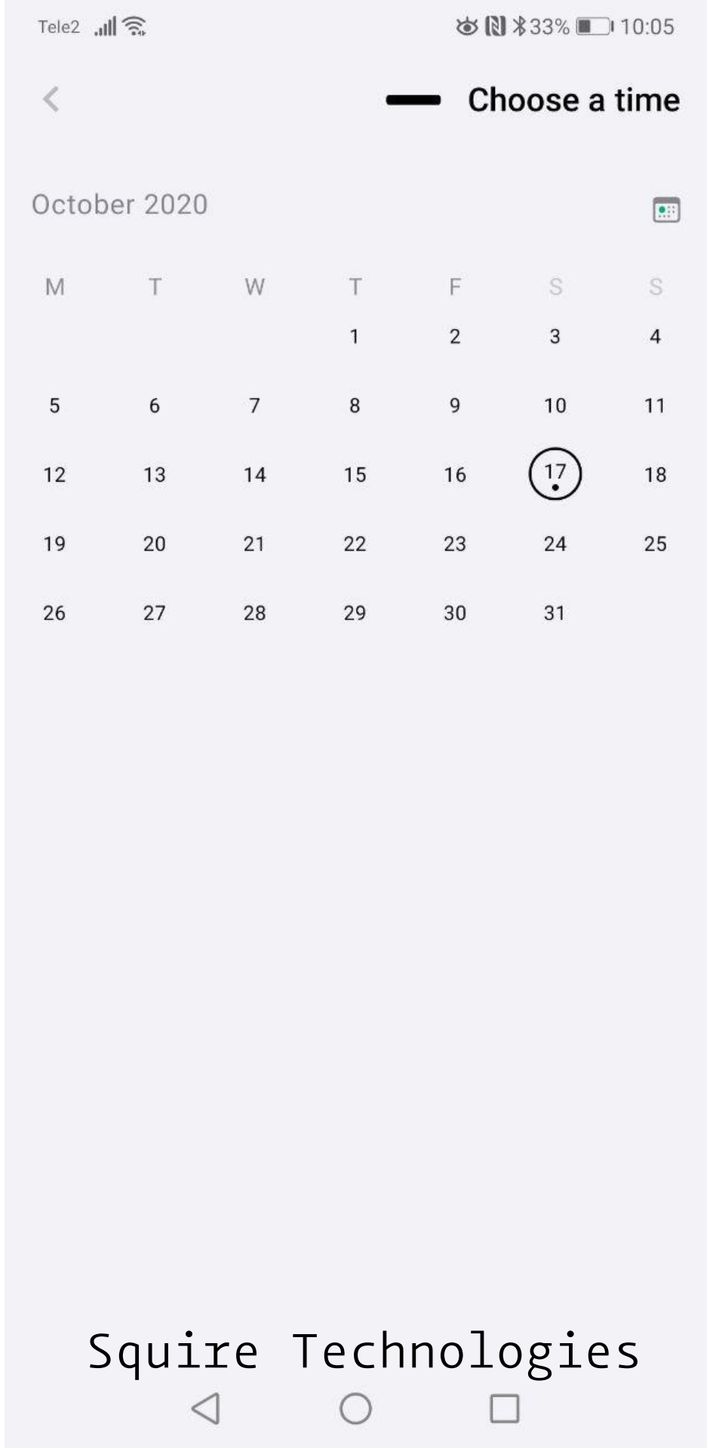
MVI

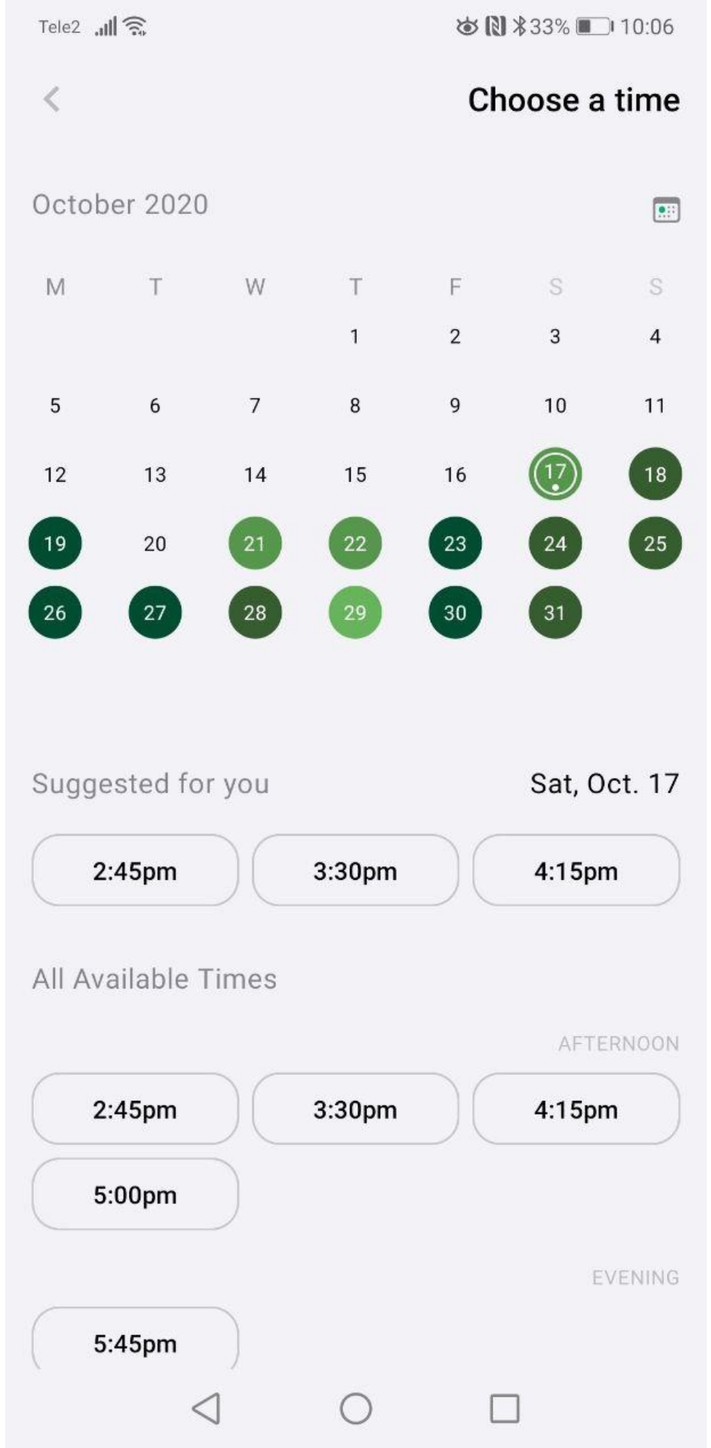


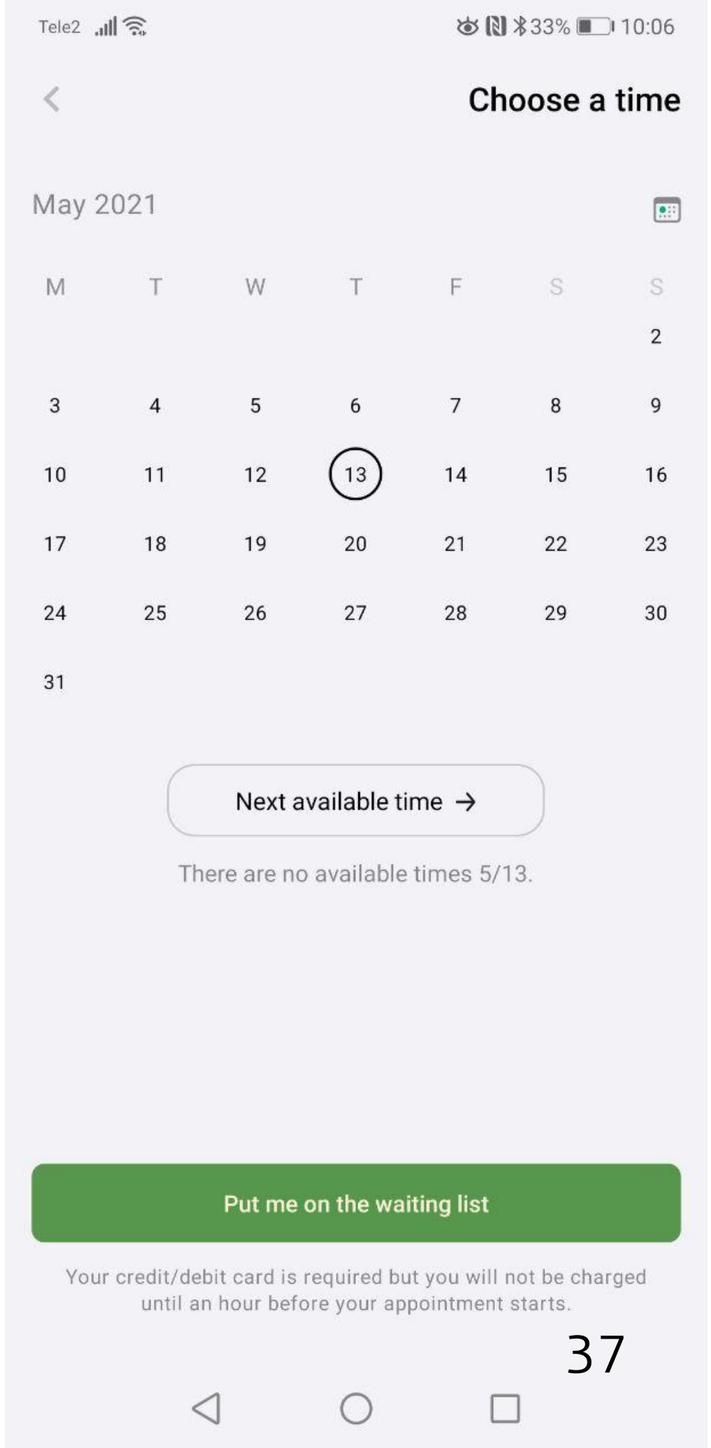
Redux

1. State (Model)
2. View (View)
3. Action (Intent)




Tele2
33% 10:05
Choose a time
October 2020
M T W T F S S
1 2 3 4
5 6 7 8 9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31


Tele2
33% 10:06
Choose a time
October 2020
M T W T F S S
1 2 3 4
5 6 7 8 9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31
Suggested for you
Sat, Oct. 17
2:45pm
3:30pm
4:15pm
All Available Times
AFTERNOON
2:45pm
3:30pm
4:15pm
5:00pm
EVENING
5:45pm

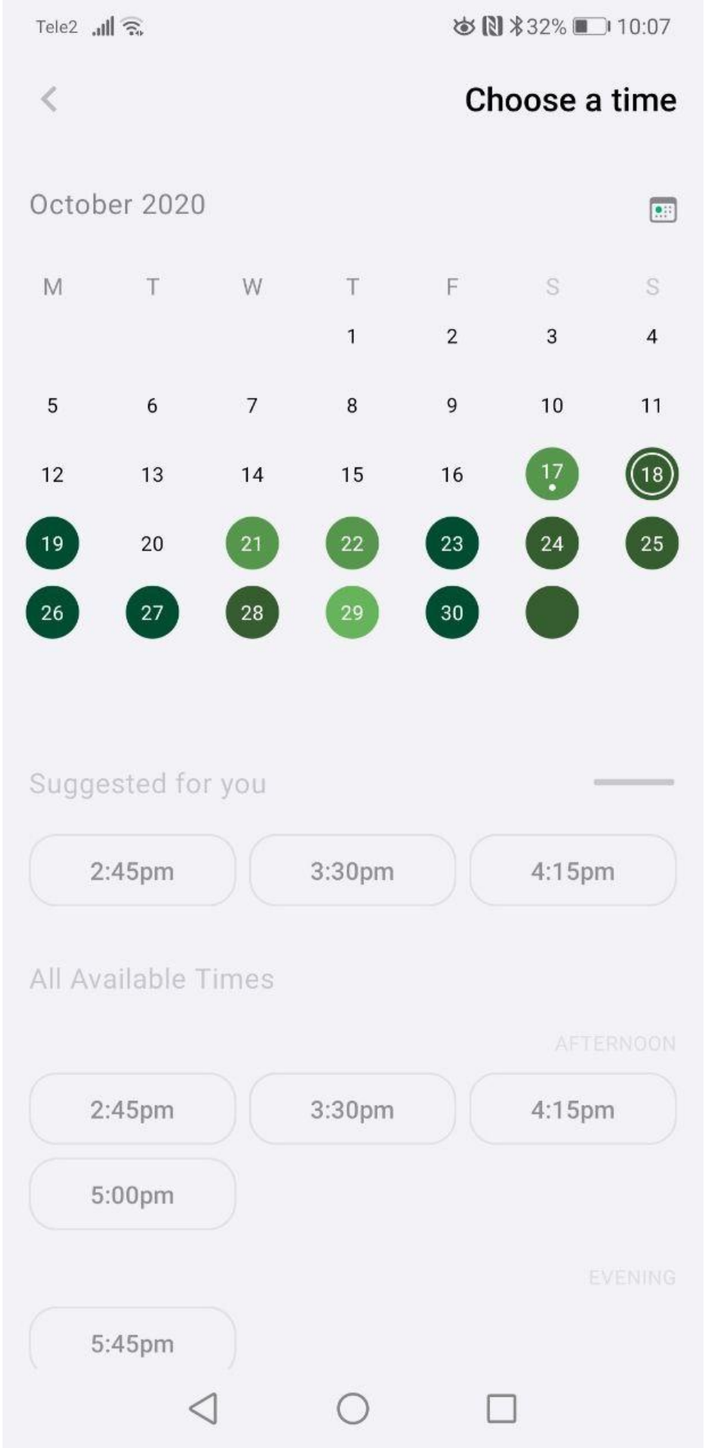

Tele2
33% 10:06
Choose a time
May 2021
M T W T F S S
2
3 4 5 6 7 8 9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
31
Next available time →
There are no available times 5/13.
Put me on the waiting list
Your credit/debit card is required but you will not be charged until an hour before your appointment starts.

Model (and State)

```
sealed class ScreenState<out T> {
  class Loading<out T> : ScreenState<T>()
  data class Error(val error: String) : ScreenState<Nothing>()
  data class Success<out T>(val data: T) : ScreenState<T>()
}
```

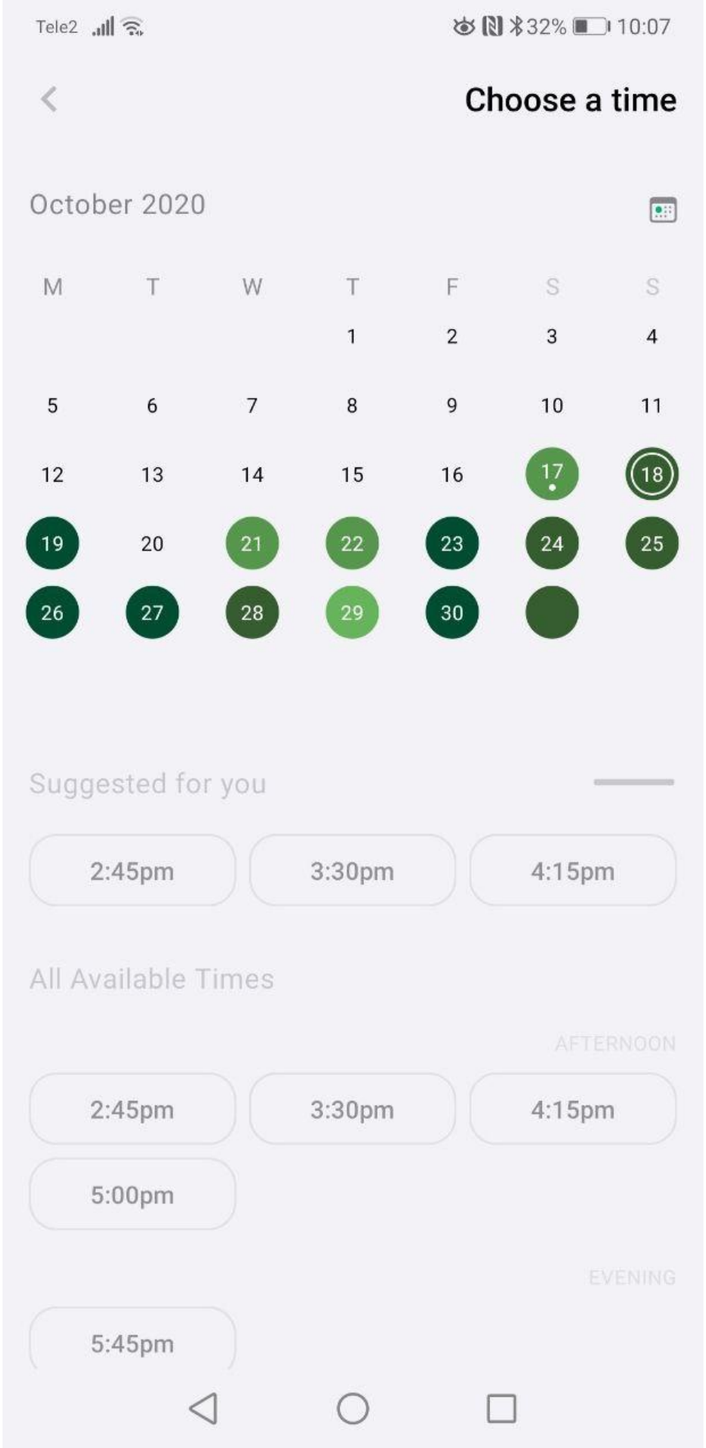
Model (and State)

```
sealed class ScreenState<out T> {
  class Loading<out T> : ScreenState<T>()
  data class Error(val error: String) : ScreenState<Nothing>()
  data class Success<out T>(val data: T) : ScreenState<T>()
}
```



Content and update

```
data class TimeState (
  val dateSlots: List<DateSlot>,
  val selectedDate: LocalDate,
  val selectedTime: LocalDateTime,
  val isLoading: Boolean,
  val isError: Boolean
)
```



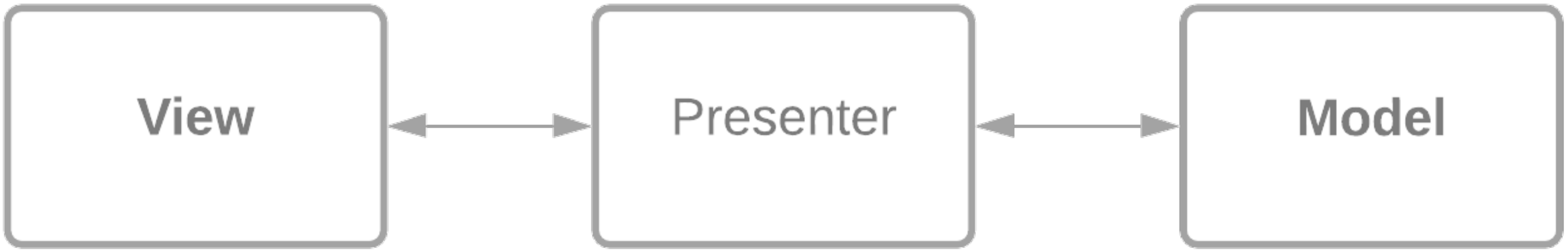
View

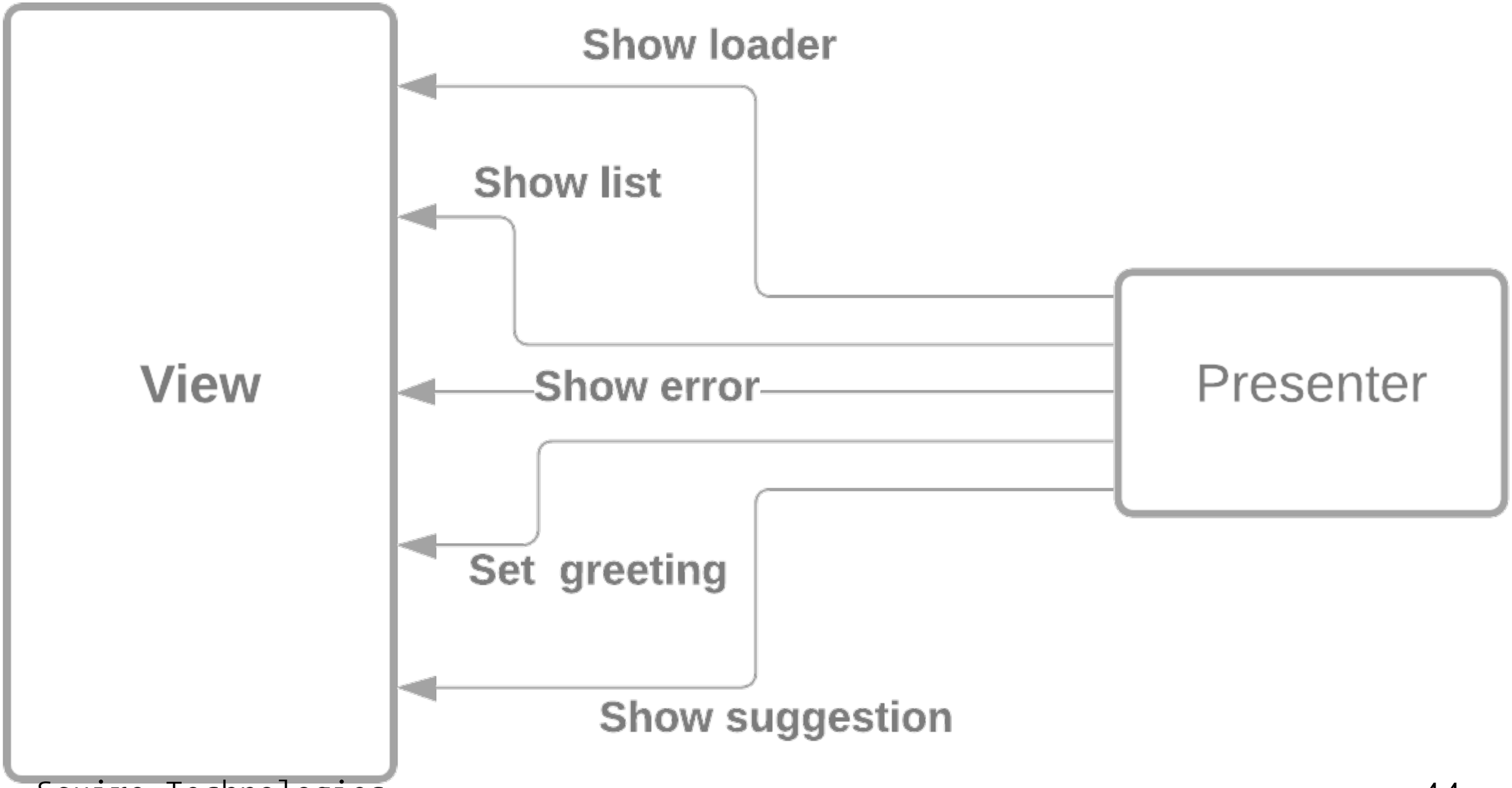
Anything from Part 1

Goals

1. **Low coupling between the four App's Components**
2. Less code
3. More tests

But... MVP?




View
Show loader
Show list
Show error
Set greeting
Show suggestion
Presenter

Goals

1. ~~**Low coupling between the four App's Components**~~
2. Less code
3. More tests

Intent

```
sealed class ChooseTimeViewIntent {
  data class SelectDay(val day:  LocalDate) : ChooseTimeViewIntent()
  data class SelectTime(val time:  LocalDateTime) : ChooseTimeViewIntent()
  object Proceed : ChooseTimeViewIntent()
}
```

MVI. High-level example

```
// create a model with an initial state
val model = Model<SomeState>(SomeState(selected = null, isLoading = false))

// subscribe view to state's change
model.stateStream.subscribe { state -> view.renderUi(state) }

// start sending events from view to model
view.eventsStream.subscribe { oldState, event -> model.produceNewState(oldState, event) }
```

Reducer

```
class Model {

    val stateStream: PublishSubject <State>

    // Potentially, lots of code

    val reduce(oldState: State, intent: Intent) {
        when(intent) {
             is IntentA -> stateStream.onNext(state.copy(aChecked = true))
             is IntentB -> stateStream.onNext(state.copy(bChecked = true))
        }
    }

    // Potentially, lots of code
}
```

Testing

```
@Test
fun `on shop selection a shop gets selected and focused`() {
  sut.accept(Wish.SelectShop(testShop))
  assertThat(sut.state.selectedShop).isEqualTo(testShop)
  assertThat(sut.state.focusedShop).isEqualTo(testShop)
}

@Test
fun `on nearby action get sorted shops`() {
  sut.accept(Wish.ToggleNearby)
  assertThat(sut.state.sortedShops).isEqualTo(testLocations)
}
```

MVI. Outcomes of the pattern

1. Pushes you to care about the whole state
2. Things like State Logging and Time Machine
3. Highly testable

Goals. Final thoughts on MVI

1. ~~Low coupling between the four App's Components~~
2. **More tests**
3. Less code

Goals. Final thoughts on MVI

1. ~~Low coupling between the four App's Components~~
2. ~~**More tests**~~
3. Less code

Implementing MVI

Implementing for MVI

1. Build it manually
2. MVICore
3. MVIKotlin
4. Roxie
5. RxRedux
6. Mobius
7. Mosby

Code examples

What to pay attention at?

1. Language - Java 7/8 or Kotlin
2. Process death / Configuration change
3. Plugin - based architecture
4. Ability to select a reactive framework
5. Overall simplicity of code

MVICore

1. Great docs, including reasonings
2. Very powerful tool
3. Great level of decoupling
4. Lots of classes
5. Bound to RxJava 2

Handles

- lifecycle handling
- middlewares
- time travel

Classes

1. Feature
    1. State
    2. Wish
    3. Effect
    4. News
    5. Bootstrapper
    6. Actor
    7. Reducer
    8. Post Processor
1. View
    1. ViewModel
    2. ViewEvent
2. Binder
3. Middleware
4. Mappers
    1. StateToViewModel
    2. ViewEventToWish
    3. ViewEventToAnalytics

Classes

```
data class OnItemSelected (val item: Item) : ViewEvent()
data class SelectItem (val item: Item) : Wish()
data class SelectItem (val item: Item) : Effect()
data class ItemSelected (val item: Item) : News()

// RIBs
data class ItemSelected (val item: Item) : Output()
```

MVIKotlin

1. Successor of MVICore
2. Kotlin Multiplatform
3. Choice of reactive frameworks
4. State preservation
5. Lifecycle handling
6. Middlewares
7. Time travel

Roxie

1. Based on AAC ViewModels
2. Relatively simple
3. Heavy usage of RxJava 2
4. Doesn't have advanced stuff, time travel
5. Simple switchMap

Over board

1. RxRedux[5]
    1. From Hannes Dorfman
    2. Kotlin
2. Mobius (Spotify)[7]
    1. Java
3. Mosby[6]

[5] https://freeletics.engineering/2018/08/16/rxredux.html

[7] https://www.facebook.com/atscaleevents/videos/2025571921049235/

[6] http://hannesdorfmann.com/android/mosby3-mvi-1

Our experience

1. Kotlin + Big team -> MVICore
2. Kotlin Multiplatform / flexible settings -> MVIKotlin
3. Lightweight solution - Roxie
4. Own bicycle - not so bad :)

App's "layers"

1. Visuals on a Screen
2. Data on a Screen
3. Navigation between Screens
4. App's data and logic

Navigation and DI

Goals

1. Being able to go back and forth (backstack)
2. Convenient DI
3. Composable
4. Lifecycle. Ideally, a simple one

Options

1. Fragments + Fragment Manager, Cicerone / Nav Components
2. Badoo RIBs
3. Decompose
4. Conductor
5. Square Workflow
6. Navigation Feature in MainActivity
7. ~~Flow~~
8. ~~Activities~~

Badoo RIBs

1. Proper DI
2. Simpler lifecycle
3. Clear Input - Output contracts

Goals

1. Low coupling between the four App's Components
2. Less code
3. More tests

Goals

1. ~~Low coupling between the four App's Components~~
2. Less code
3. More tests

Goals

1. ~~Low coupling between the four App's Components~~
2. ~~Less code~~* (at least not more)
3. More tests

Goals

1. ~~Low coupling between the four App's Components~~
2. ~~Less code*~~ (at least not more)
3. ~~More tests~~

Decompose

1. Successor of Badoo RIBs
2. Kotlin Multiplatform
3. State restoration

Navigation Feature

1. Your navigation state is... a state!
2. No new dependencies
3. Or... ComposeRouter

Over board

1. Conductor
2. Square Workflow
3. ~~Flow~~
4. ~~Activities~~

Our thoughts

1. Common solution - Fragments
2. Cleaner contracts + better DI - Badoo RIBs
3. Feeling Hipster - Jetpack-only navigation / ComposerRouter

Back to the beginning

Our current stack

1. Android Views
2. MVICore
3. RIBs

Learnings

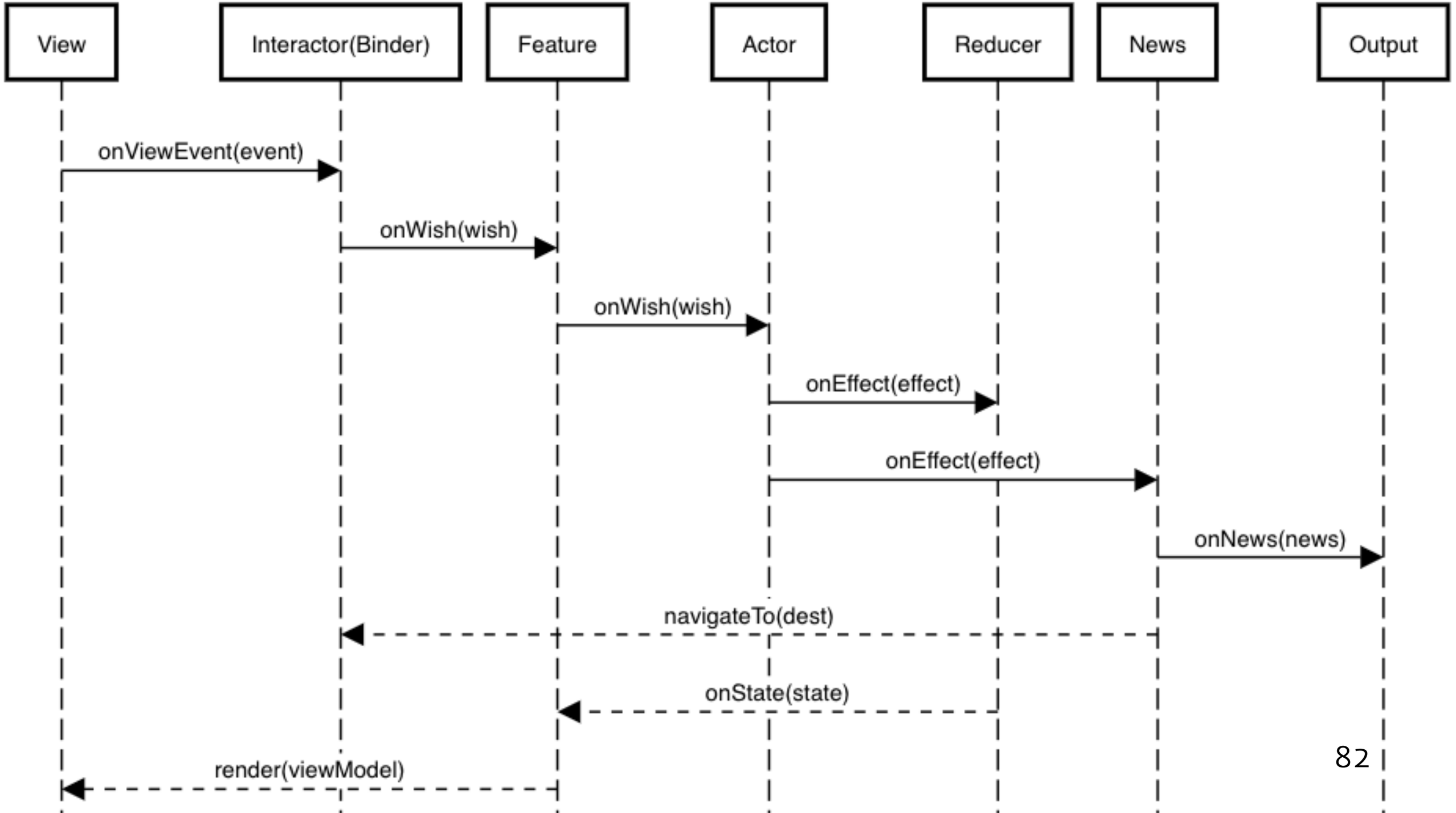
- Great separation of concerns
- Easier debug and testing
- Clean interfaces

  But:

- A LOT of classes
- Ambiguity in terms of logic separation
- Overhead when creating simple things

Examples

1. Interactor and Feature - business logic
    1. Feature - anything changing state
    2. Interactor - working with inputs / outputs of other ribs
    3. Interactor - navigation
2. Navigation from News or from ViewEvents
    1. Feature state change can lead to navigation event
    2. A click on a View can lead to navigation event without state change
3. 5 data classes with the same name (SelectSmth)


View
Interactor(Binder)
Feature
Actor
Reducer
News
Output
onViewEvent(event)
onWish(wish)
onWish(wish)
onEffect(effect)
onEffect(effect)
onNews(news)
navigateTo(dest)
onState(state)
render(viewModel)

Our plans [WIP]

1. Jetpack Compose
2. Custom MVI / Roxie
3. RIBs

Goals

1. ~~Low coupling between the four App's Components~~
2. ~~Less code~~
3. ~~More tests~~

Cons

1. Learning curve
2. Need to mature
3. Third-party dependency
4. Binding to RxJava 2

There is life behind MVP!

Artur Badretdinov

Squire Technologies - hiring remotely

Android Academy - looking for mentors and students for a free course

Thanks to

1. Tim Plotnikov https://twitter.com/timofeipl
2. Arkadii Ivanov https://twitter.com/arkann1985
3. Mikhail Levchenko https://twitter.com/TheMishkun
4. Me :) https://twitter.com/ArtursTwit

Links

Test Events App repo:
https://github.com/Gaket/OhMyEvent
Android Academy:
https://habr.com/ru/news/t/522972/

Backup slides

Ekaterina Petrova
@KathrinPetrova

В ответ @Arturstwit

Работу с источниками данных точно стоит изолировать, чтобы потом взять Kotlin Multiplatform Mobile, пошарить эту часть с другой платформой и всех победить 😏

9:01 AM · 4 окт. 2020 г. из Санкт-Петербург, Россия · Twitter for iPhone

1 ретвит **20** отметок «Нравится»

Jetpack Compose Promises

— Declarative UI

— Rethinking the legacy

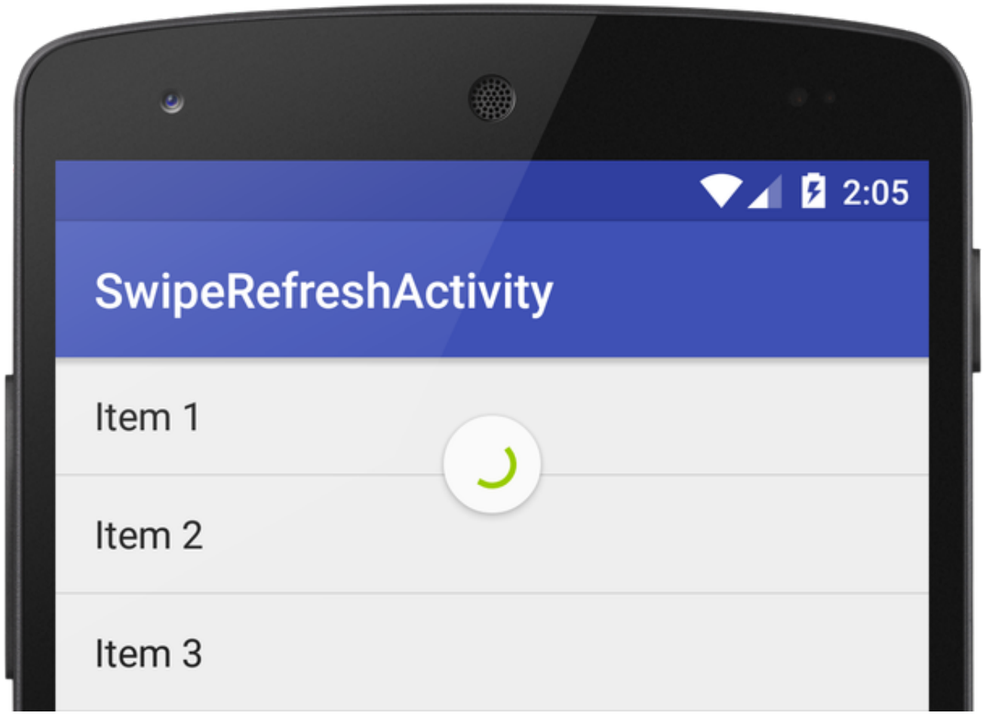
— Independent from Android Version

— Compatible with Views

A problem of state

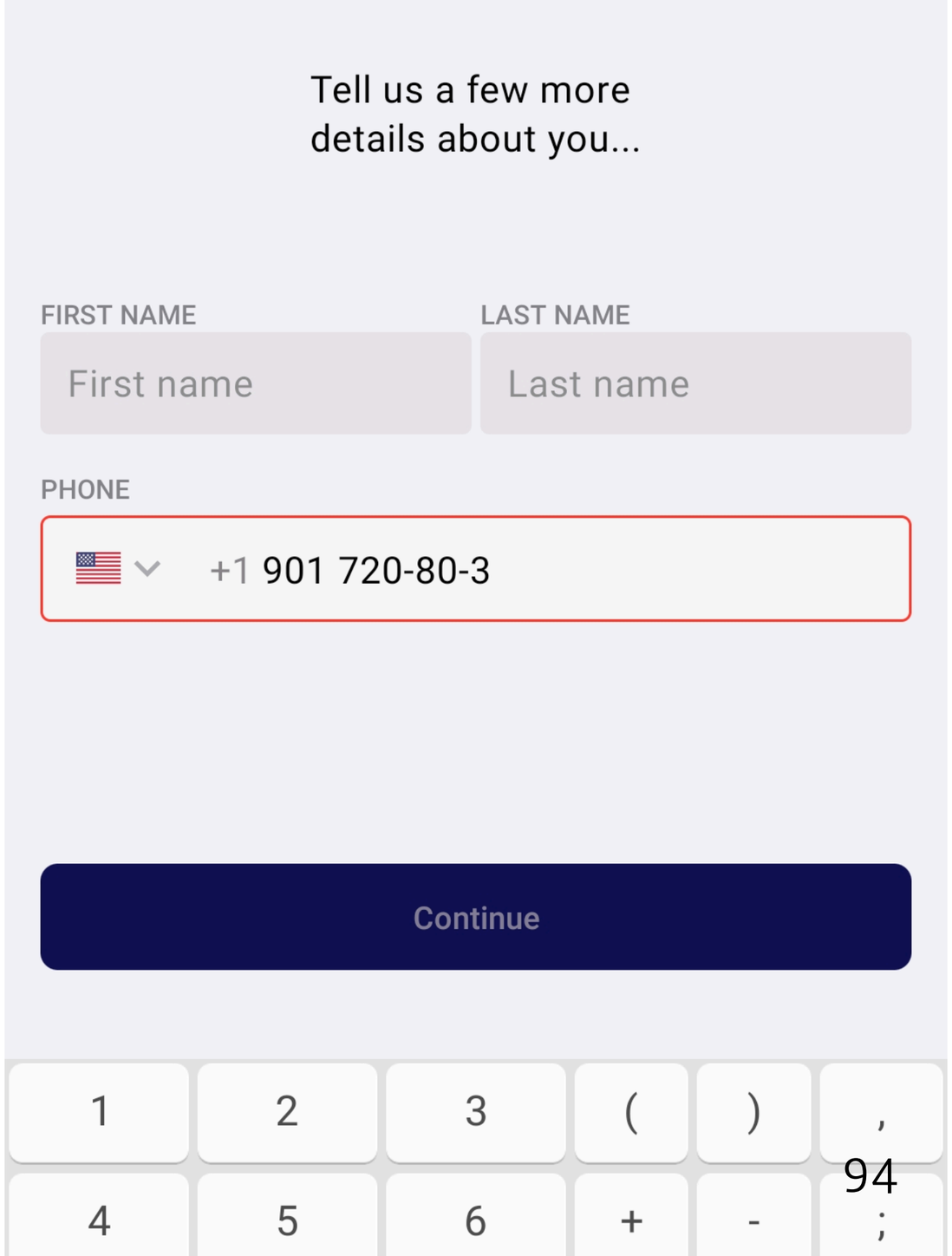
Views with intrinsic state:

- SwipeRefreshLayout
- Spinner
- EditText



EditText with PhoneWatcher

```
phoneView.addTextChangedListener(phoneFormatter)
phoneView.addTextChangedListener(viewIntentSender)
// ...
fun update(vm : viewStateModel) {
  phoneView.setText(vm.phone)
}
```



EditText in Compose

```
@Composable
fun PhoneTextField() {
  val state = state { TextFieldValue("") }
  TextField(
            value = state.value,
            onValueChange = { state.value = it }
  )
}
```

EditText in Compose

```
@Composable
fun PhoneTextField(val text : String,
                   val inputListener : Listener ) {
  TextField(
            value = text,
            onValueChange = { listener.textUpdated(it) }
  )
}
```

https://sequencediagram.org/