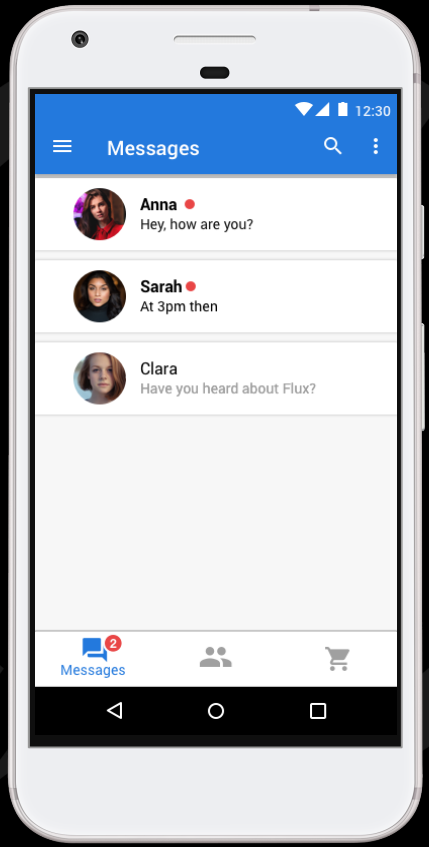
SUPER CHARGE

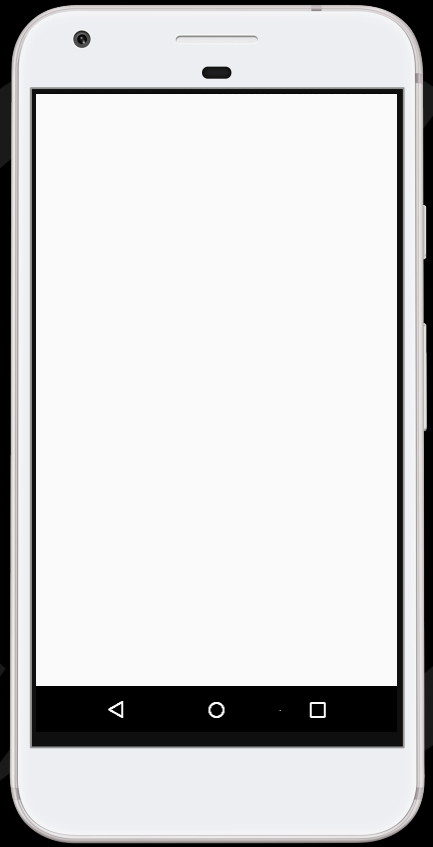
Android and Flux: It's a match!

Attila Polacsek
Senior Android Developer | Supercharge

SUPER CHARGE



SUPER CHARGE



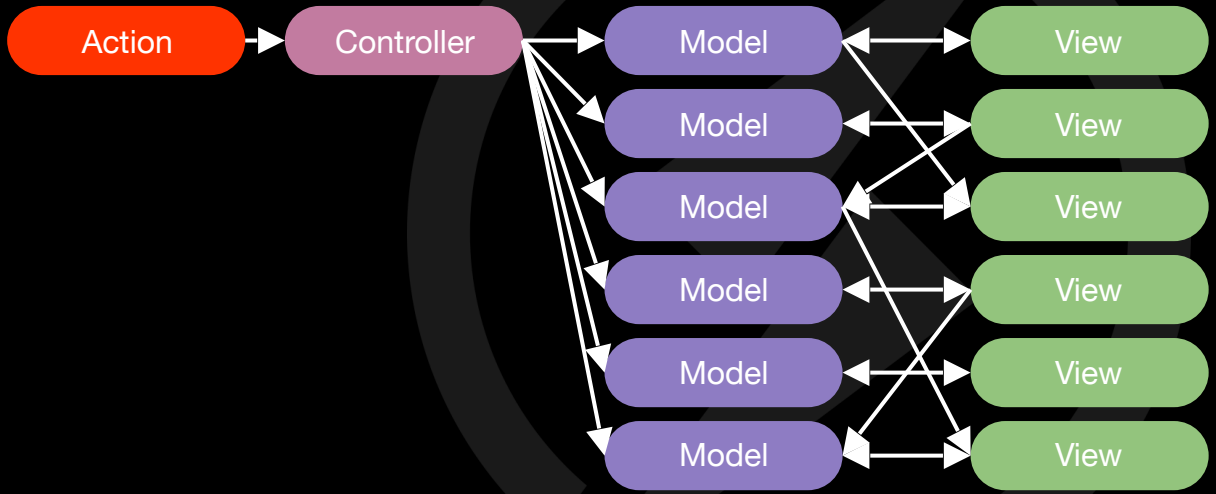
WHAT IS FLUX?

SUPER CHARGE

What is FLUX?

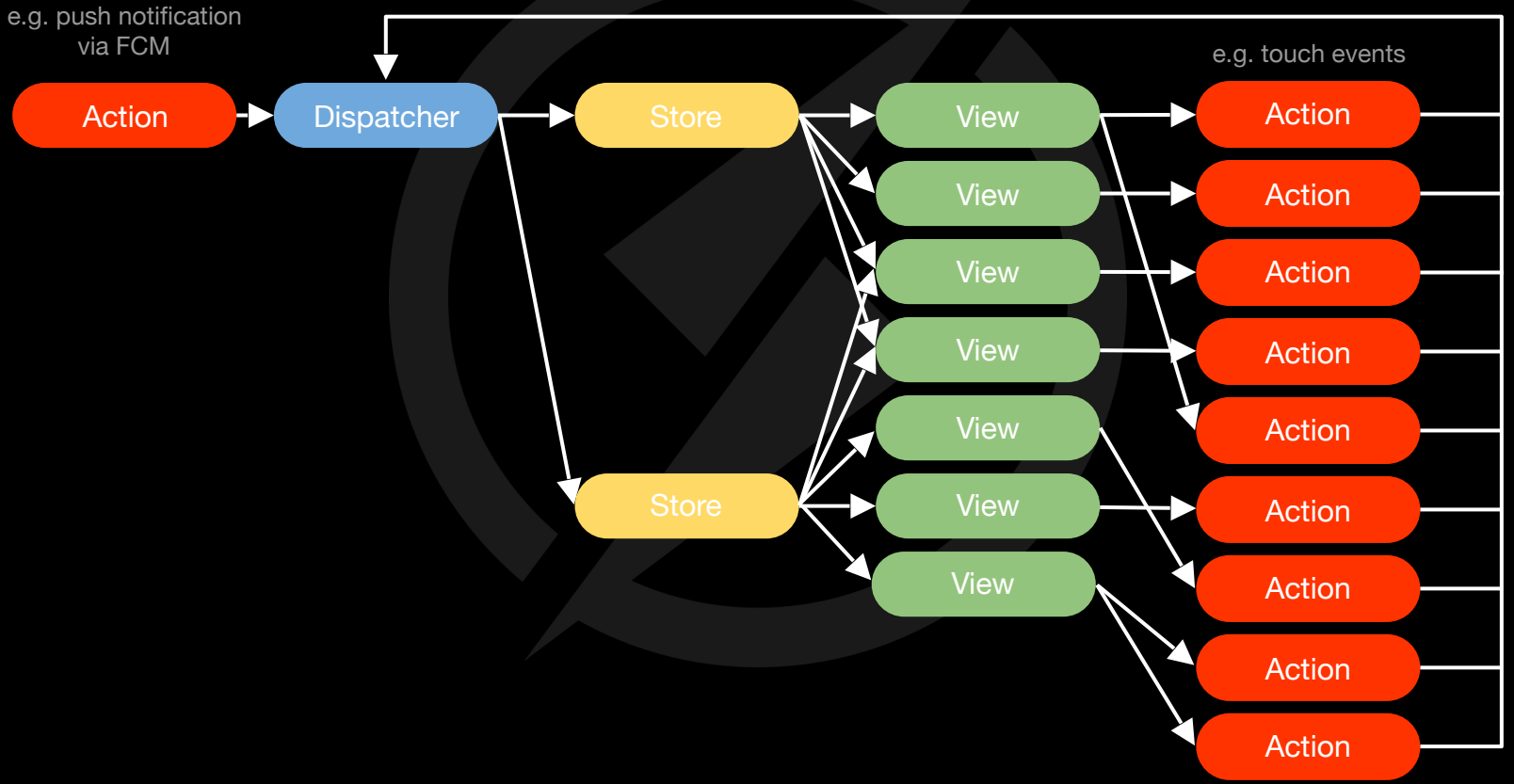
- Application architecture
- Pattern, not a framework
- Created by Facebook as a successor of client-side MVC
- Based on unidirectional data flow

MVC Architecture



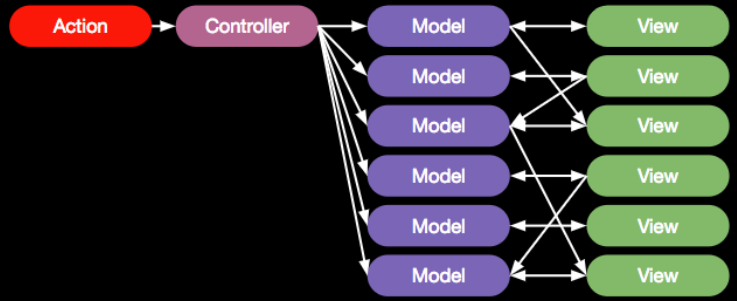
SUPER CHARGE

FLUX Architecture

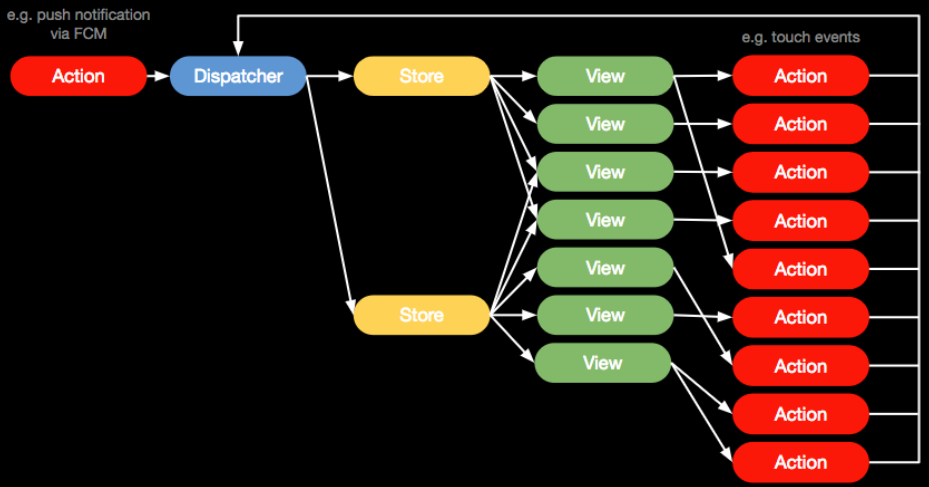


SUPER CHARGE

MVC Architecture



FLUX Architecture

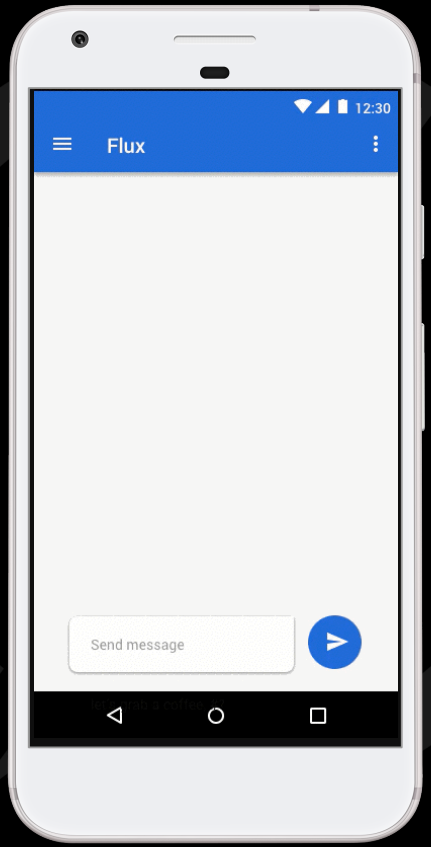


WHAT IS FLUX?

SUPER CHARGE

WHAT IS FLUX?

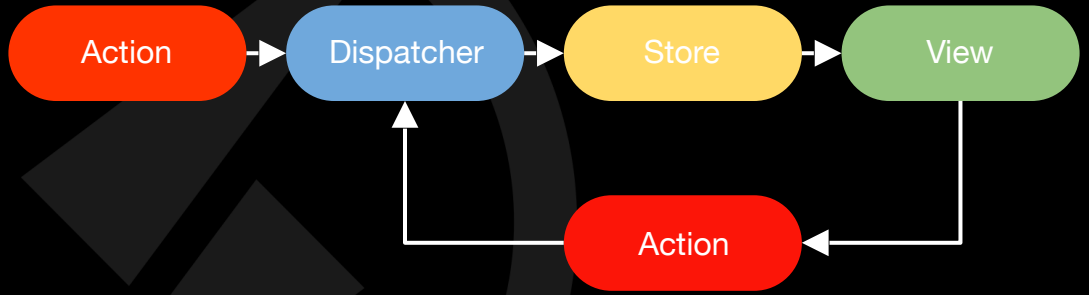
SUPER CHARGE



SUPER CHARGE

COMPONENTS

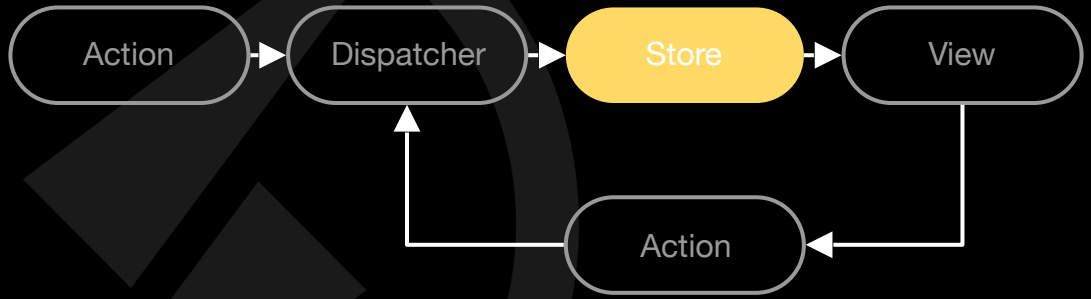
- Store
- Action
- Dispatcher
- View



SUPER CHARGE

STORE

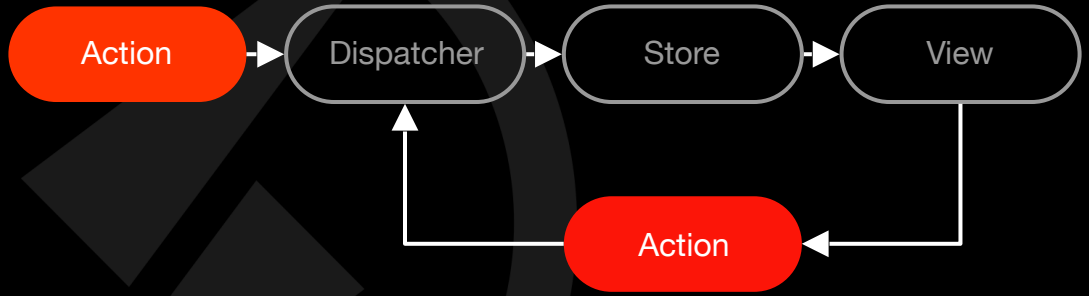
- Holds the application data
- Immutable from the outside
- Produces a change event



SUPER CHARGE

ACTION

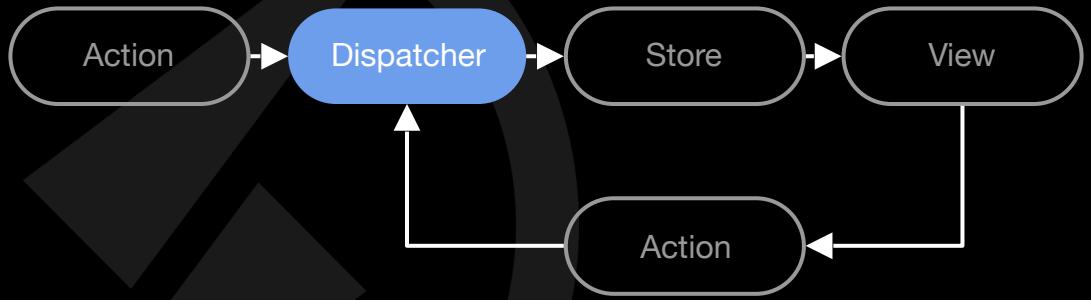
- Actions data objects
- They define the internal API
- Action Creators are methods
- Send action to dispatcher



SUPER CHARGE

DISPATCHER

- Receives all actions
- Broadcast them to stores
- Invoke callbacks in specific order



SUPER CHARGE

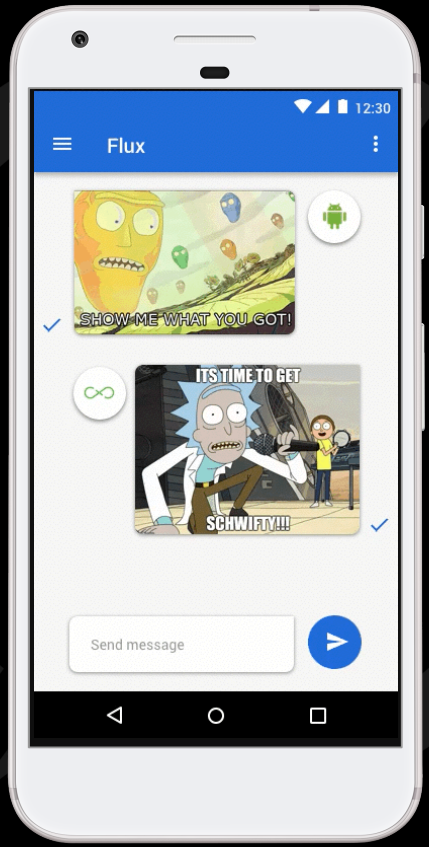
```
// Facebook's own example for waitFor:
CityStore.dispatchToken = flightDispatcher.register(function(payload) {
    if (payload.actionType === 'country-update') {
        // `CountryStore.country` may not be updated.
        flightDispatcher.waitFor([CountryStore.dispatchToken]);
        // `CountryStore.country` is now guaranteed to be updated.

        // Select the default city for the new country
        CityStore.city = getDefaultCityForCountry(CountryStore.country);
    }
});
```

CONS

- All UI state is handled in stores
- Stores are too dependent
- Dispatcher is everywhere
- Not cancelable

WHAT IS FLUX?

SUPER CHARGE



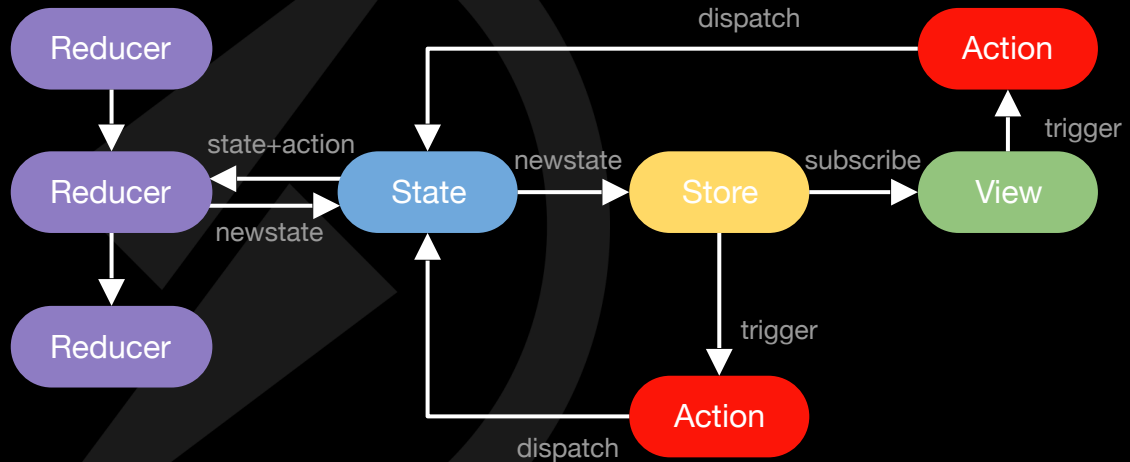
HOW WE IMPLEMENTED IT

On top of RxJava2

SUPER CHARGE

SCFlux

- Implemented on top of RxJava2
- Dependencies between stores are looser
- State persistence is extracted from the stores
- Reducers functions mutate the state



SUPER CHARGE

STORE

- Minor differences
- Returns a Flowable instead of callbacks
- State persistence is extracted

```
class ConversationStoreImpl
@Inject constructor(repository: MessageRepository) : ConversationStore {
    private val listFlowable: Flowable<List<Message>>

    init {
        listFlowable = Flowable.fromCallable(repository::findAll)
            .repeatWhen { completed ->
                completed.zipWith(
                    repository.dataUpdatedEvents(),
                    BiFunction<Any, Any, Any> { item, _ -> item }
                )
            }
            .share()
    }

    override fun getList(): Flowable<List<Message>> {
        return listFlowable
    }
}
```

SUPER CHARGE

ACTION

- Type comes from the class
- Payload is optional

ACTION CREATORS

- Describes the action pipeline
- Composes API calls and actions
- Returns an Rx Comletable

```
data class CreateMessage(
    val msg: String,
    val status: MessageStatus) : Action()

data class UpdateMessage(
    val id: String,
    val status: MessageStatus) : Action()



interface MessageActions {
   fun sendMessage(message: String): Completable

   fun updateMessage(id: String,
                     status: MessageStatus): Completable
}
```

SUPER CHARGE

DISPATCHER

- Without waitFor it was just a dumb pipe
- Completely removed
- Event pipes are declared with Rx operators
- State is modified directly with actions and a reducers

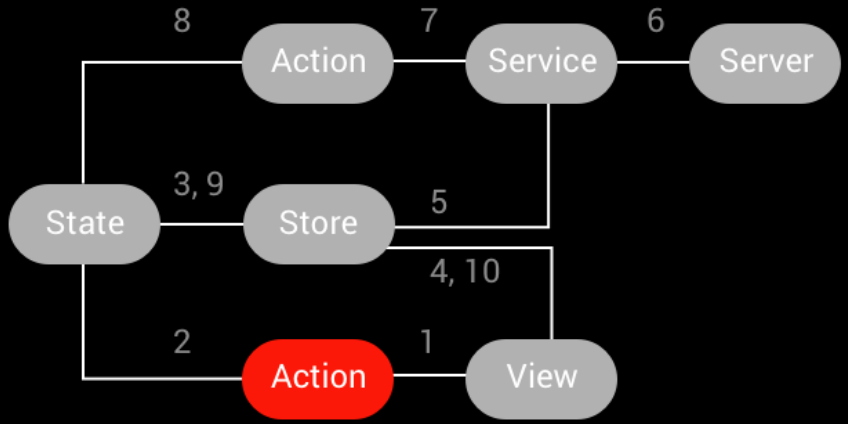
```
fun changeMessageStatus(messages: List<Message>,
                        action: UpdateMessage): List<Message> {
    return messages.map { message ->
        if (message.id == action.id) {
            message.copy(status = action.status)
        } else {
            message
        }
    }
}
```

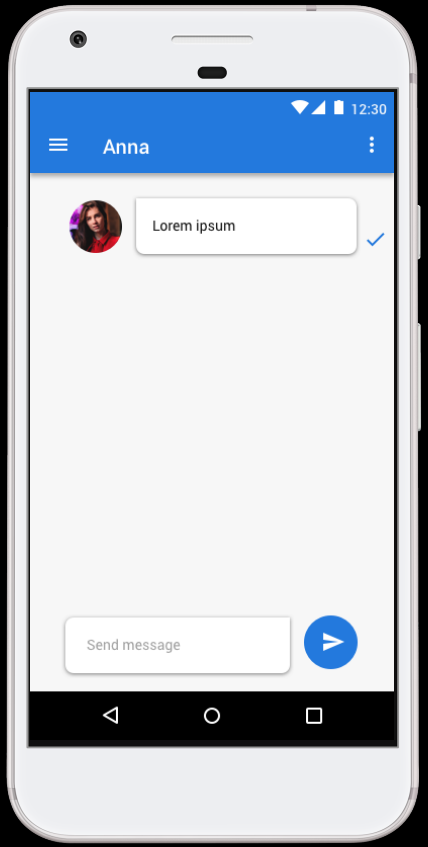
HOW WE IMPLEMENTED IT

SUPER CHARGE

HOW WE IMPLEMENTED IT

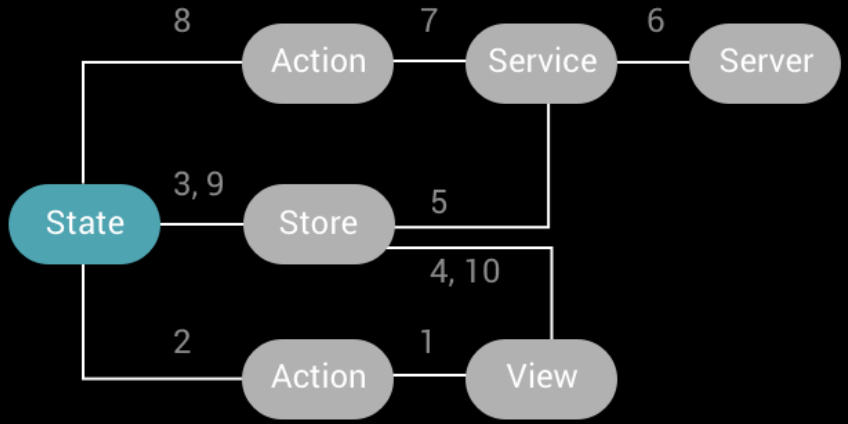
SUPER CHARGE

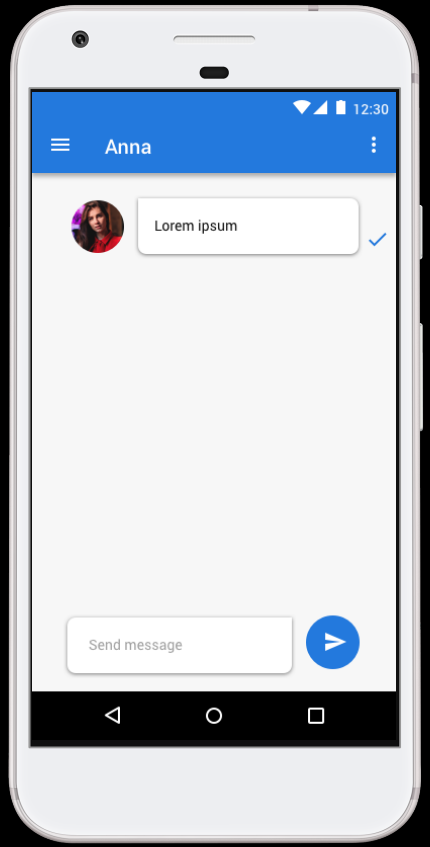




HOW WE IMPLEMENTED IT

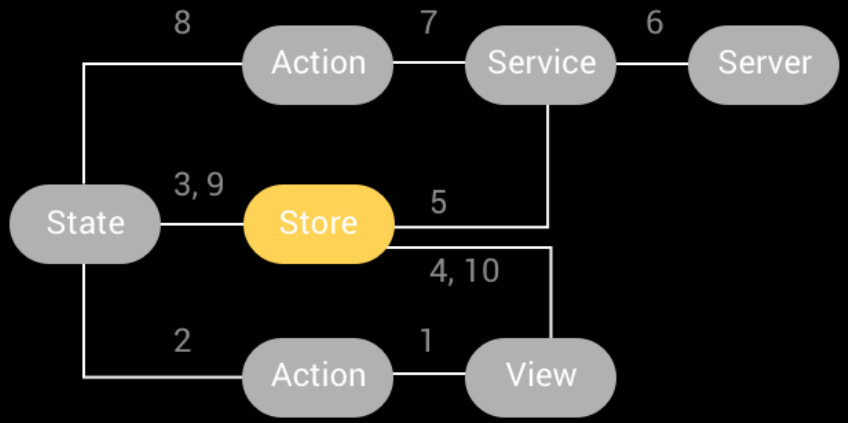
SUPER CHARGE

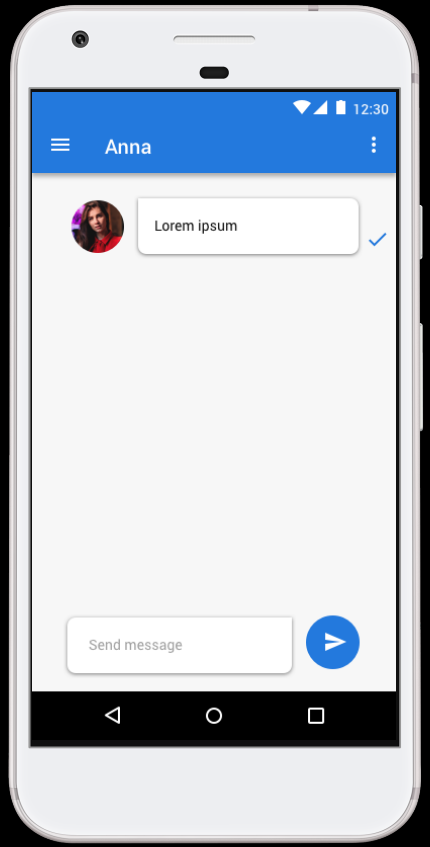




HOW WE IMPLEMENTED IT

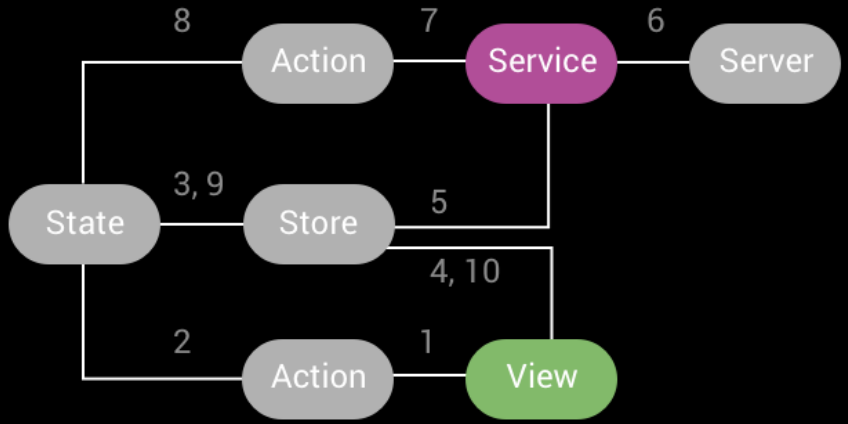
SUPER CHARGE

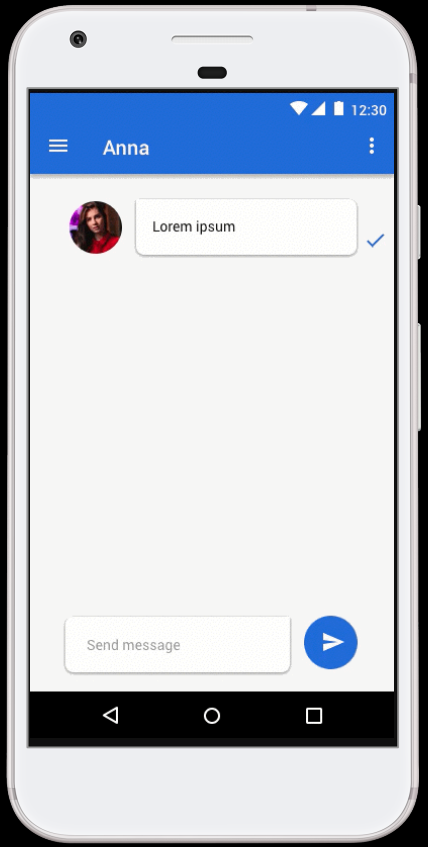




HOW WE IMPLEMENTED IT

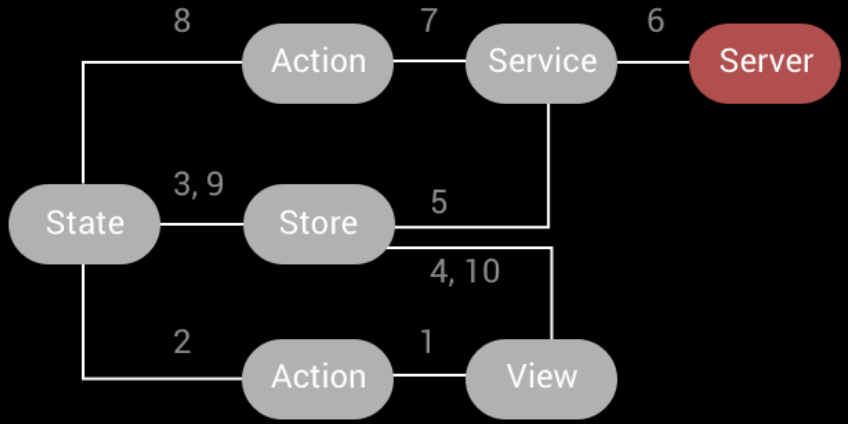
SUPER CHARGE

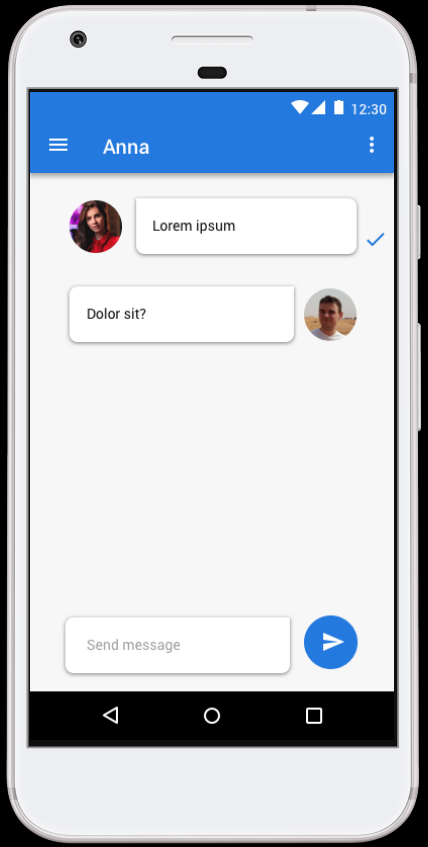




HOW WE IMPLEMENTED IT

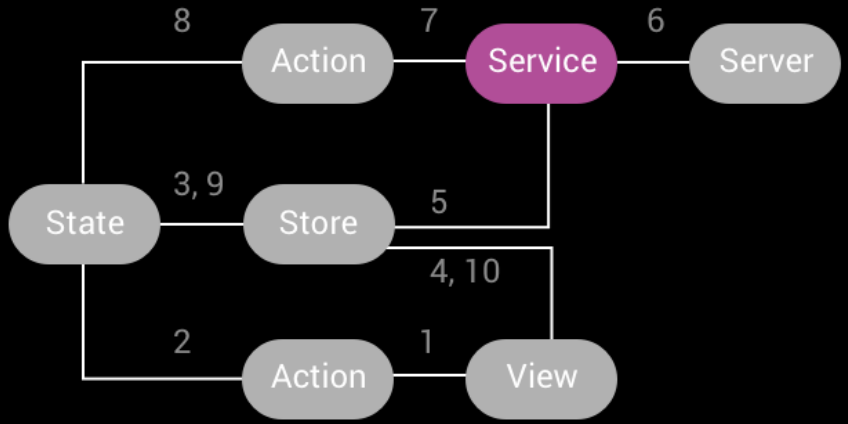
SUPER CHARGE

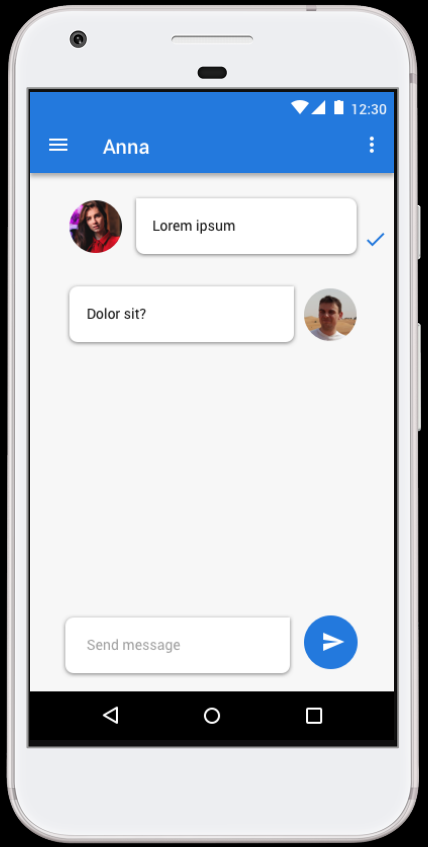




HOW WE IMPLEMENTED IT

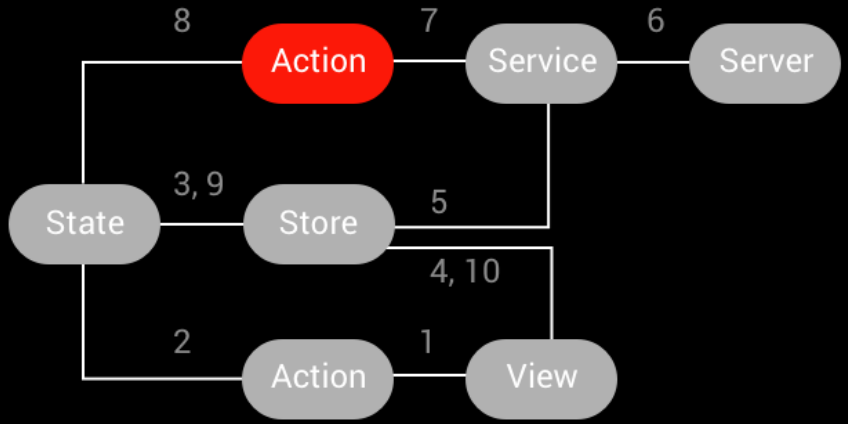
SUPER CHARGE

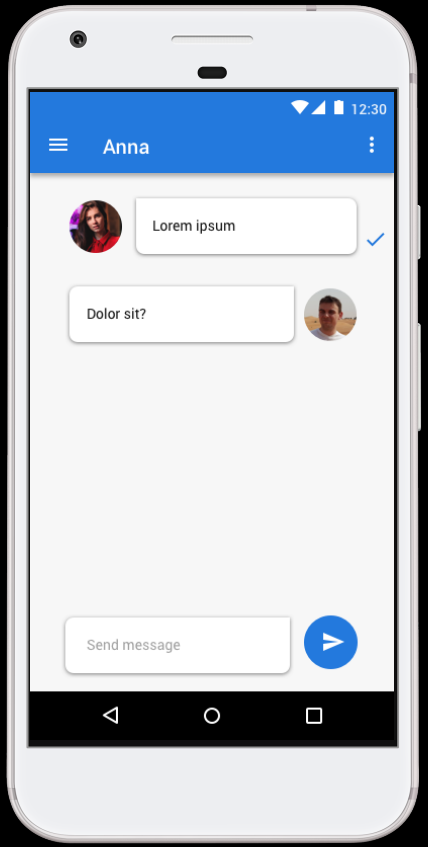




HOW WE IMPLEMENTED IT

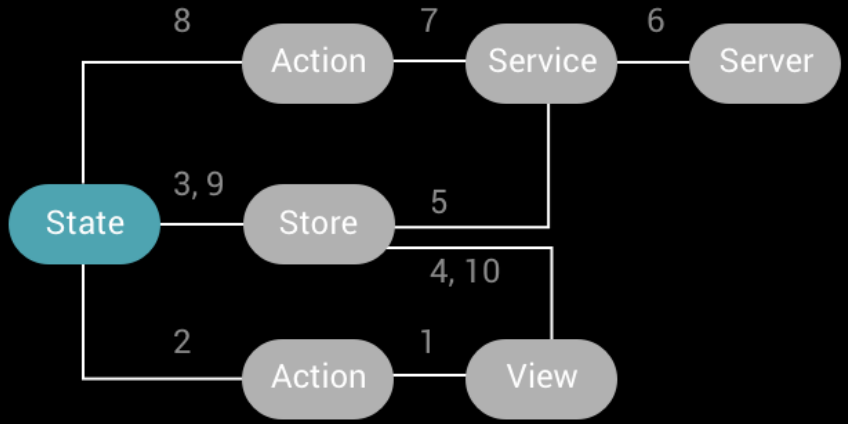
SUPER CHARGE

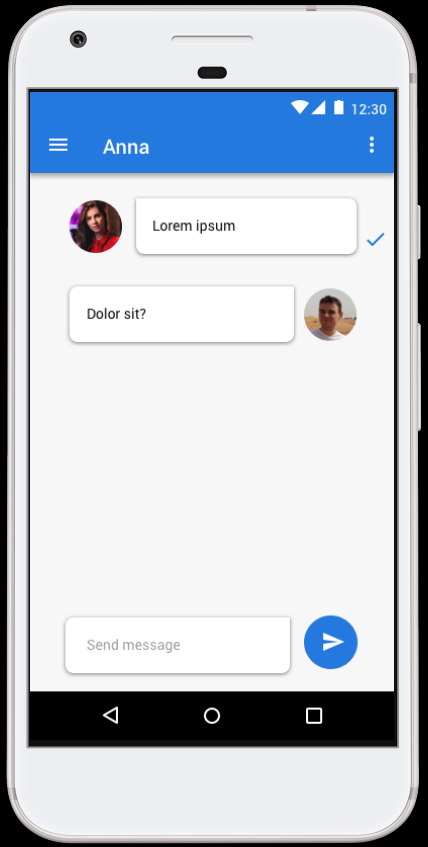




HOW WE IMPLEMENTED IT

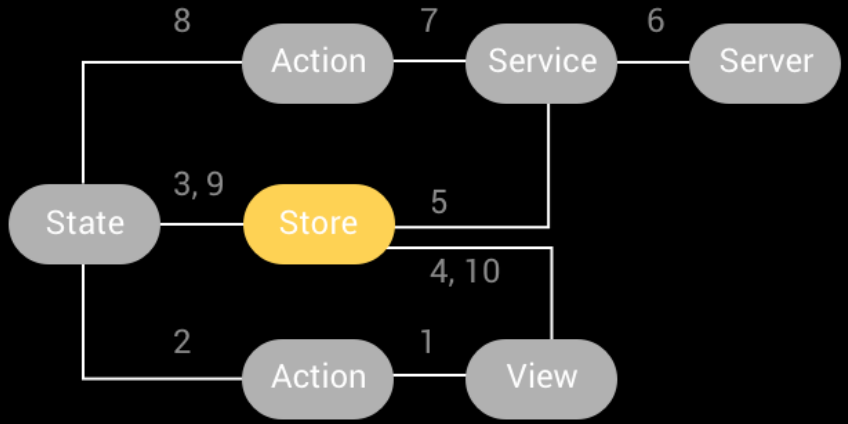
SUPER CHARGE

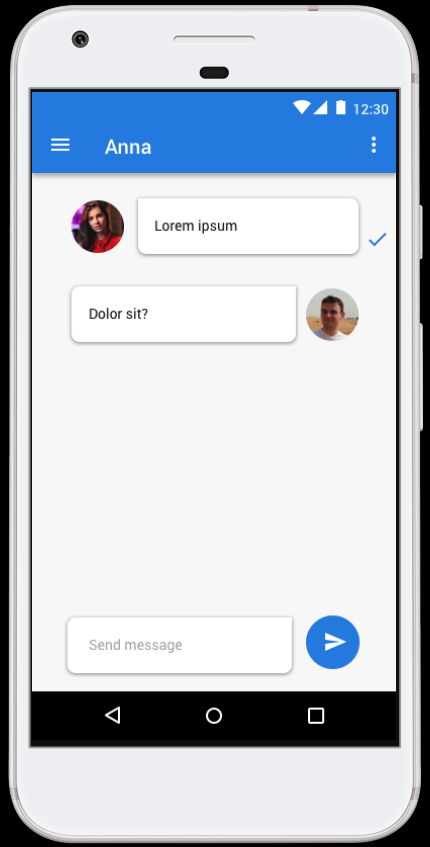




HOW WE IMPLEMENTED IT

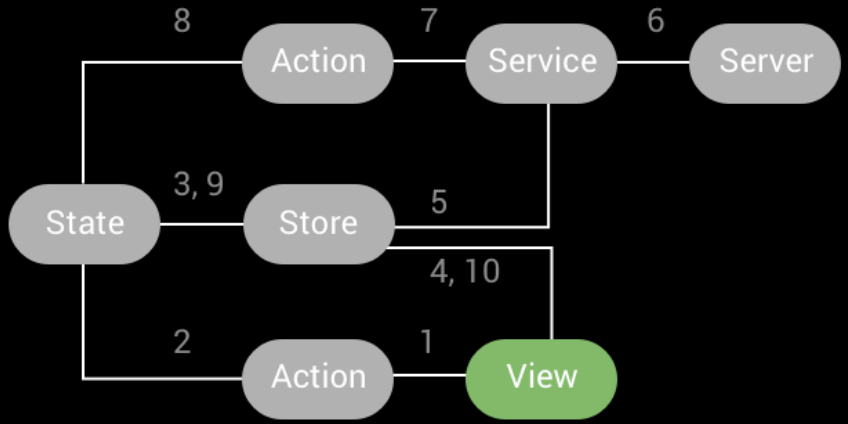
SUPER CHARGE

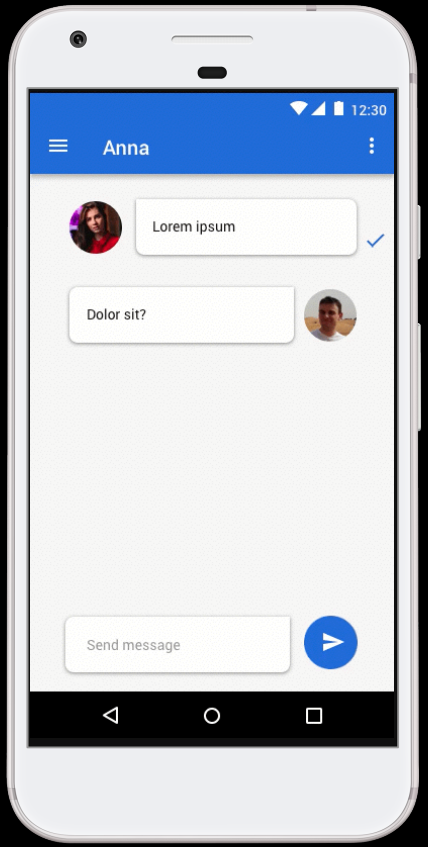




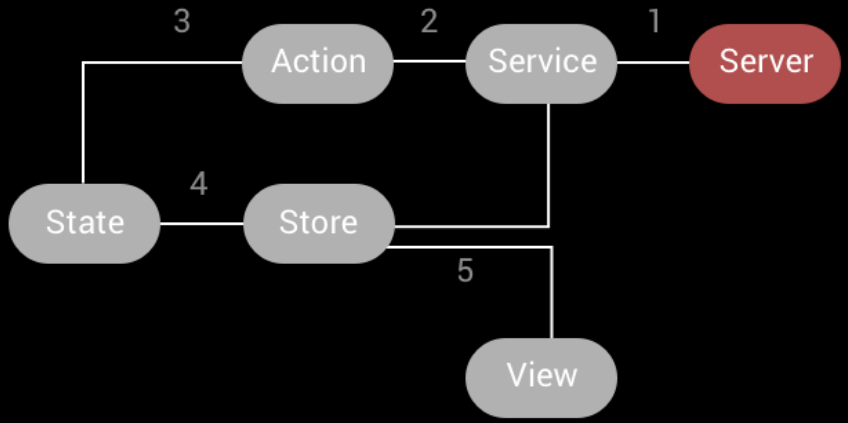
HOW WE IMPLEMENTED IT

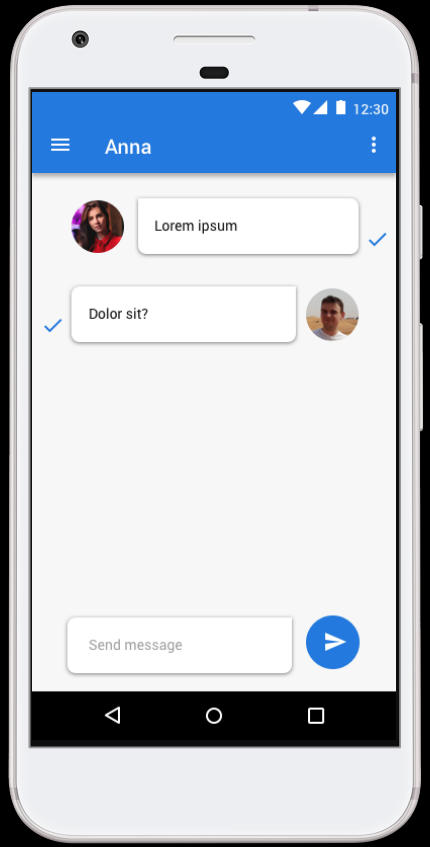
SUPER CHARGE



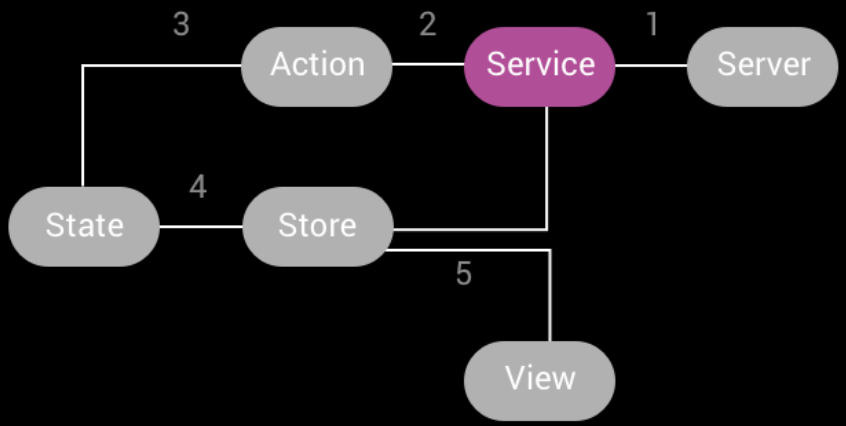


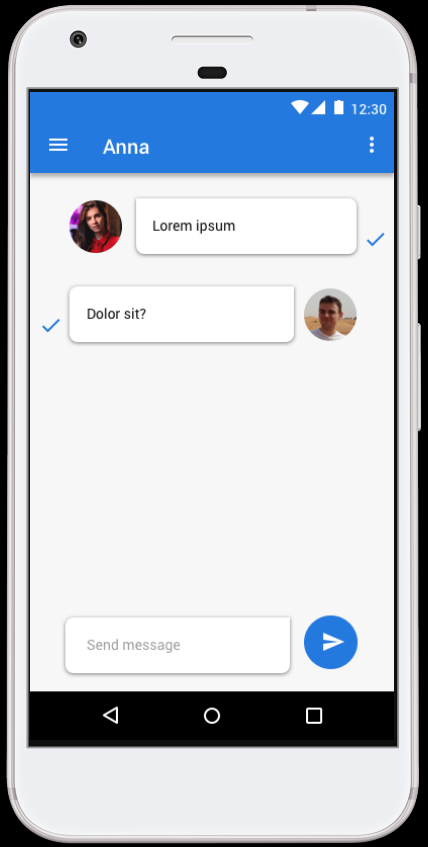
HOW WE IMPLEMENTED IT



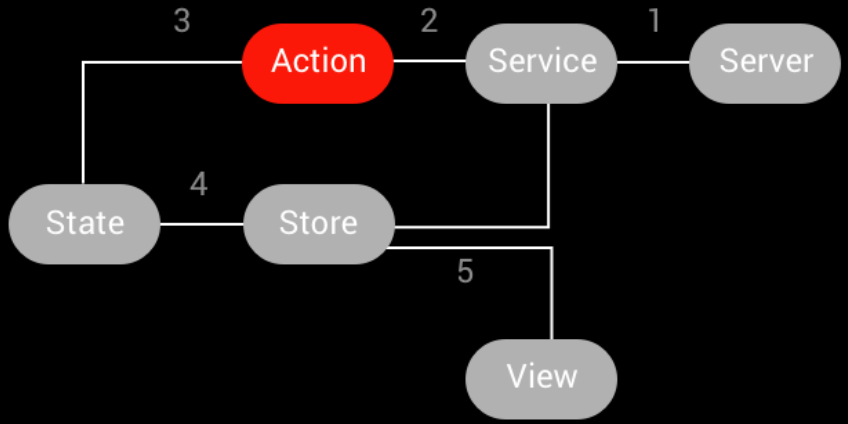


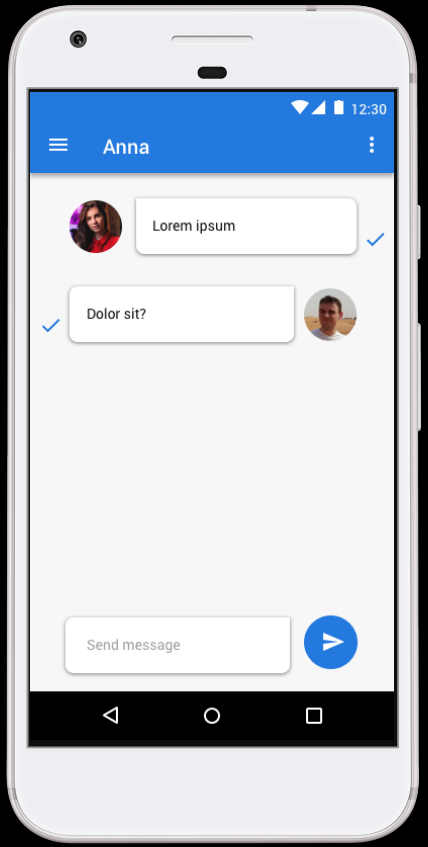
HOW WE IMPLEMENTED IT



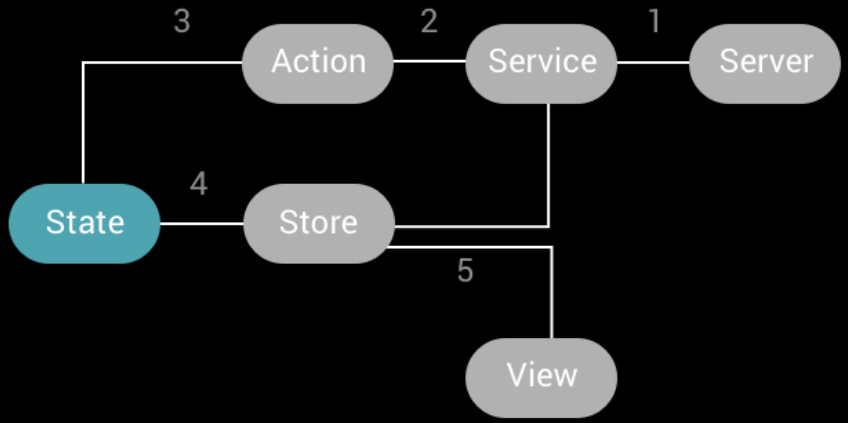


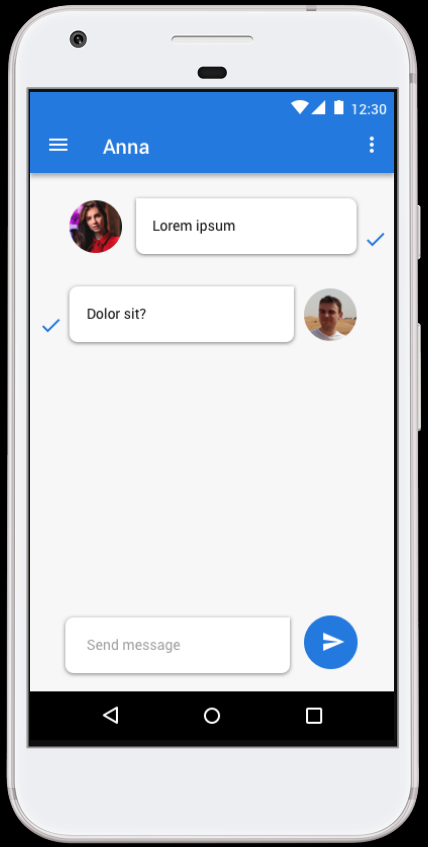
HOW WE IMPLEMENTED IT



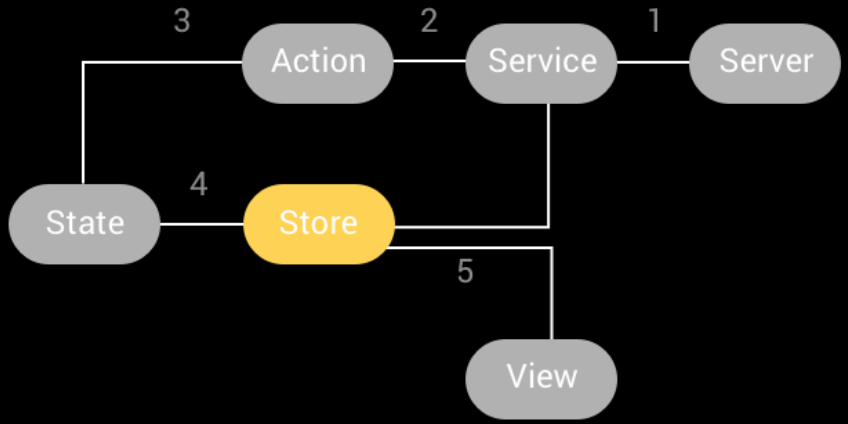


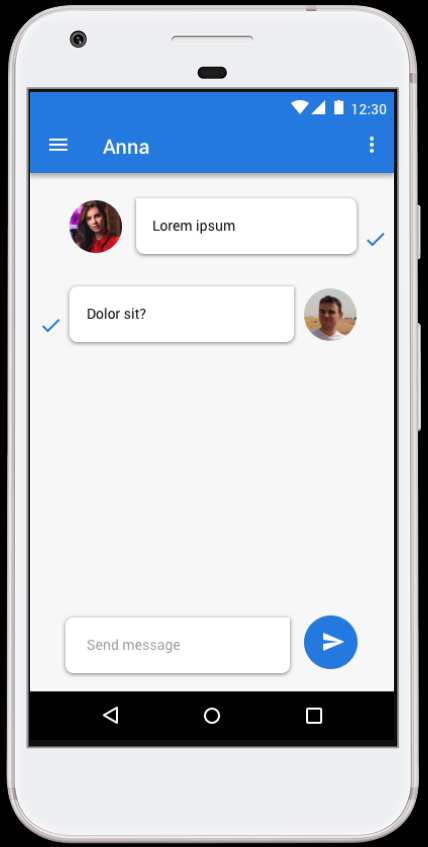
HOW WE IMPLEMENTED IT





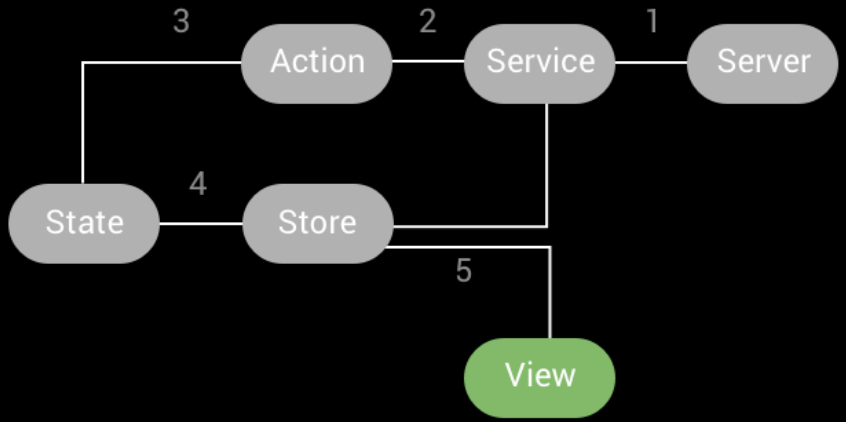
HOW WE IMPLEMENTED IT

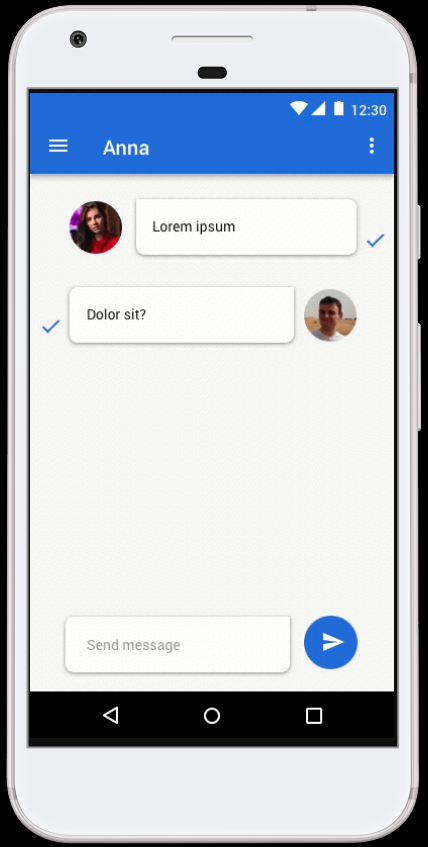




HOW WE IMPLEMENTED IT

SUPER CHARGE





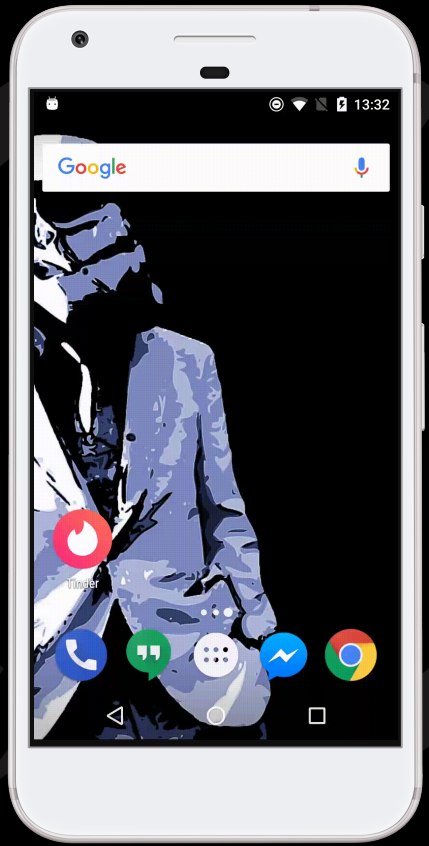
SUPER CHARGE

PROS

- Unidirectional data flow
- State mutation only via actions
- Single source of truth: store
- Automatic re-render after state update

HOW WE IMPLEMENTED IT

SUPER CHARGE



SUPER CHARGE

Thanks for your attention

Contact us!

Attila Polacsek
Senior Android Developer | Supercharge


SUPER
CHARGE