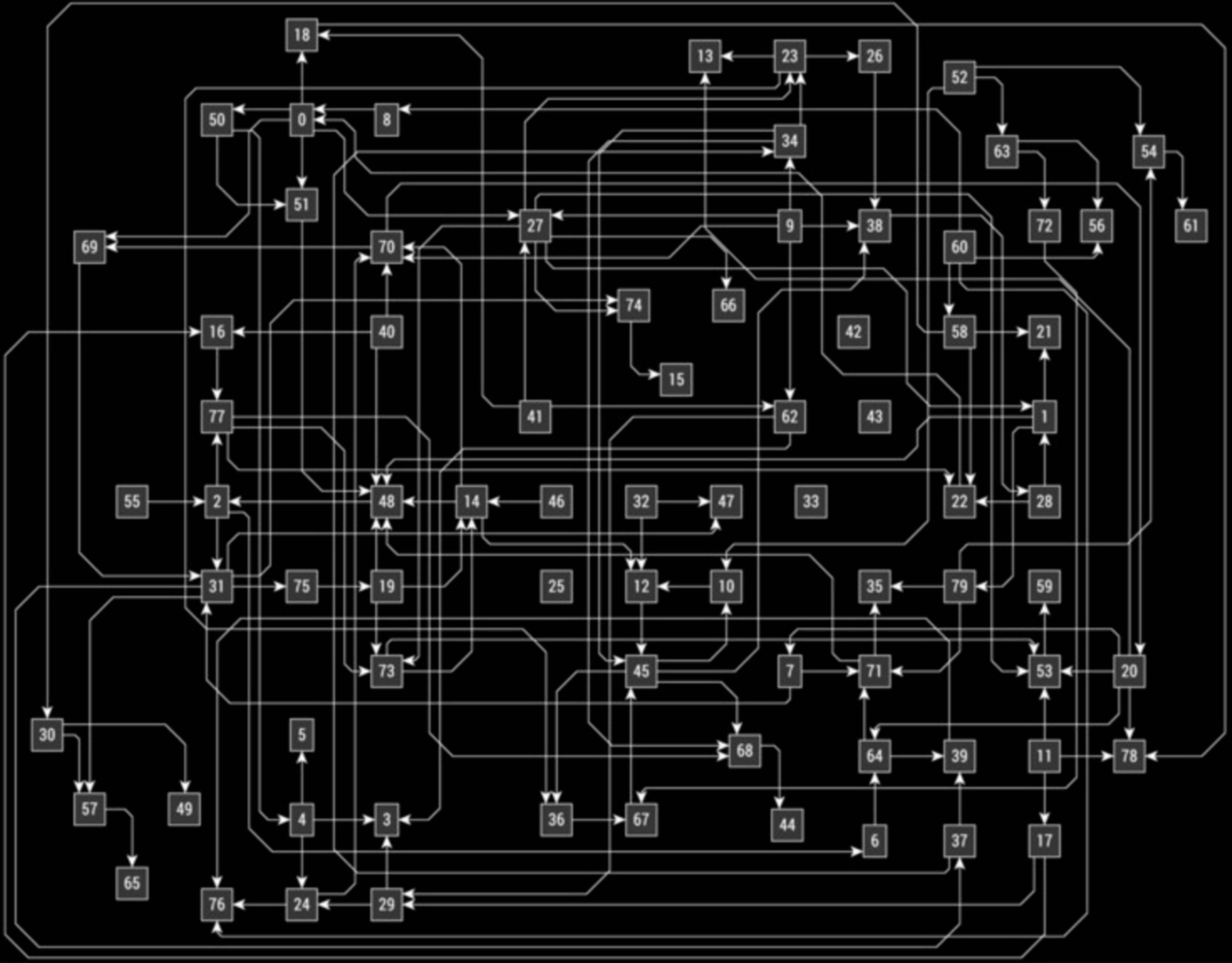
BEYOND DELEGATES: HOW TO BUILD AN APP WITH RXSWIFT

Vitor Makoto
@VitorMakoto
makoto@work.co

Why?


18
13
23
26
52
50
0
8
34
63
54
51
27
9
38
72
56
61
69
70
60
74
66
16
40
42
58
21
15
77
41
62
43
1
55
2
48
14
46
32
47
33
22
28
31
75
19
25
12
10
35
79
59
73
45
7
71
53
20
30
5
68
64
39
11
78
57
49
4
3
36
67
44
6
37
17
65
76
24
29

Reactive Programming

Sort of the Observer pattern done right.

- A combination of the best ideas from the Observer pattern and Iterator pattern.

RxSwift

An implementation of ReactiveX

- Observables
- Observers
- Subject.
- Combinators

Observable

It emits a stream of values over time

```
class ViewModel {
    var testTextWithDelay: Observable<String> {
        return Observable.just("test")
            .delay(2, scheduler: MainScheduler.instance)
    }
}
```

Observer

It subscribes to an observable

```swift
class ViewController: UIViewController {
    let viewModel = ViewModel()
    let disposeBag = DisposeBag()

    override func viewDidLoad() {
        viewModel.testTextWithDelay
            .subscribe(onNext: { text in
                print(text)
            }).addDisposableTo(disposeBag)
    }
}
```

Observer

It subscribes to an observable

```swift
class ViewController: UIViewController {
    let viewModel = ViewModel()
    let button = UIButton()
    let disposeBag = DisposeBag()

    override func viewDidLoad() {
        viewModel.testTextWithDelay
            .bindTo(button.rx.title(for: .normal))
            .addDisposableTo(disposeBag)
    }
}
```

Subject

Both Observable and Observer

- BehaviorSubject
- PublishSubject
- ReplaySubject

```
class ViewModel {
    let emailText = BehaviorSubject<String>(value: "")

    let disposeBag = DisposeBag()

    init() {
       emailText.asObservable()
            .subscribe(onNext: { text in
                print(text)
        }).addDisposableTo(disposeBag)
    }
}
```

Subject

Both Observable and Observer

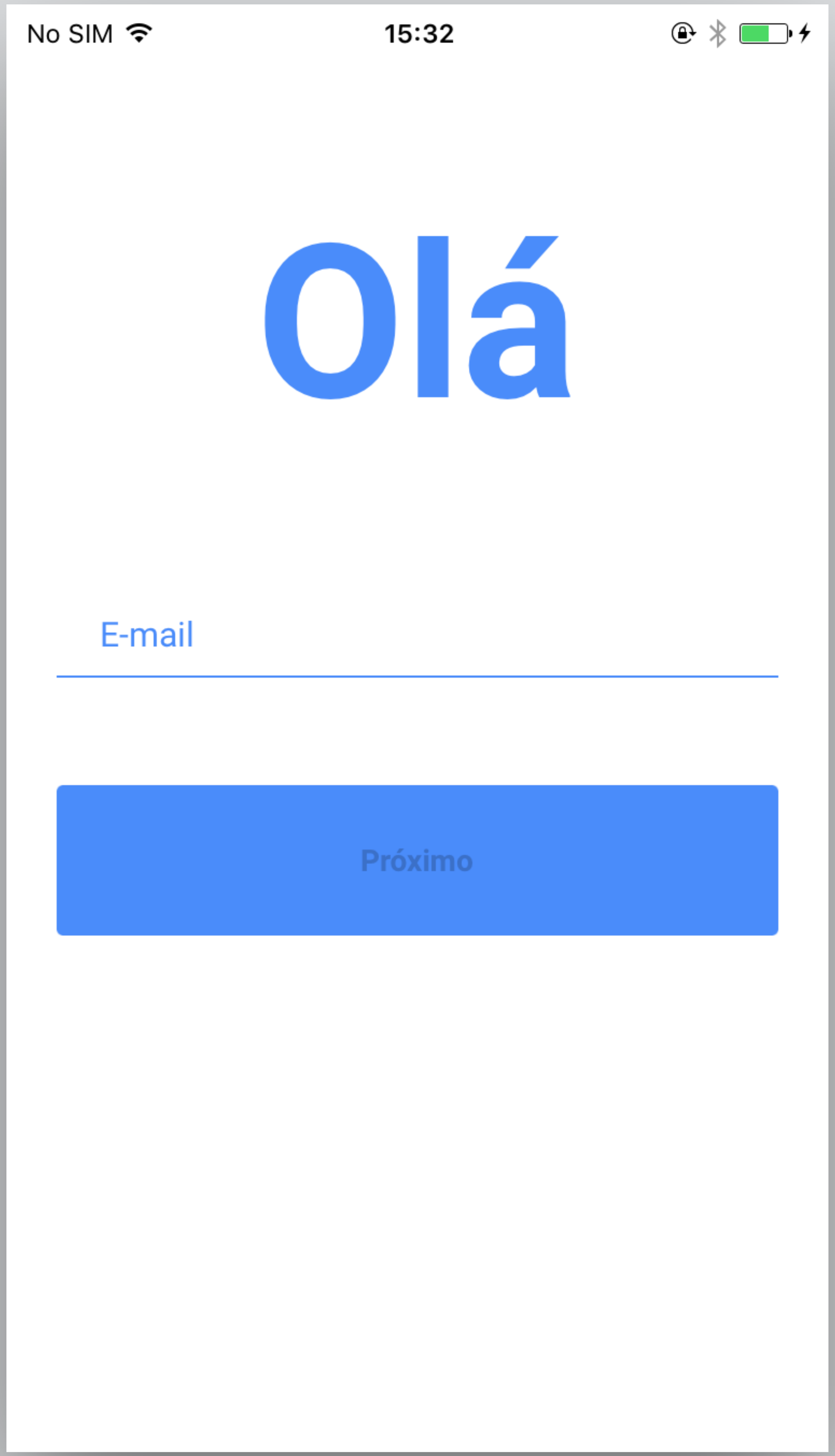
- BehaviorSubject
- PublishSubject
- ReplaySubject

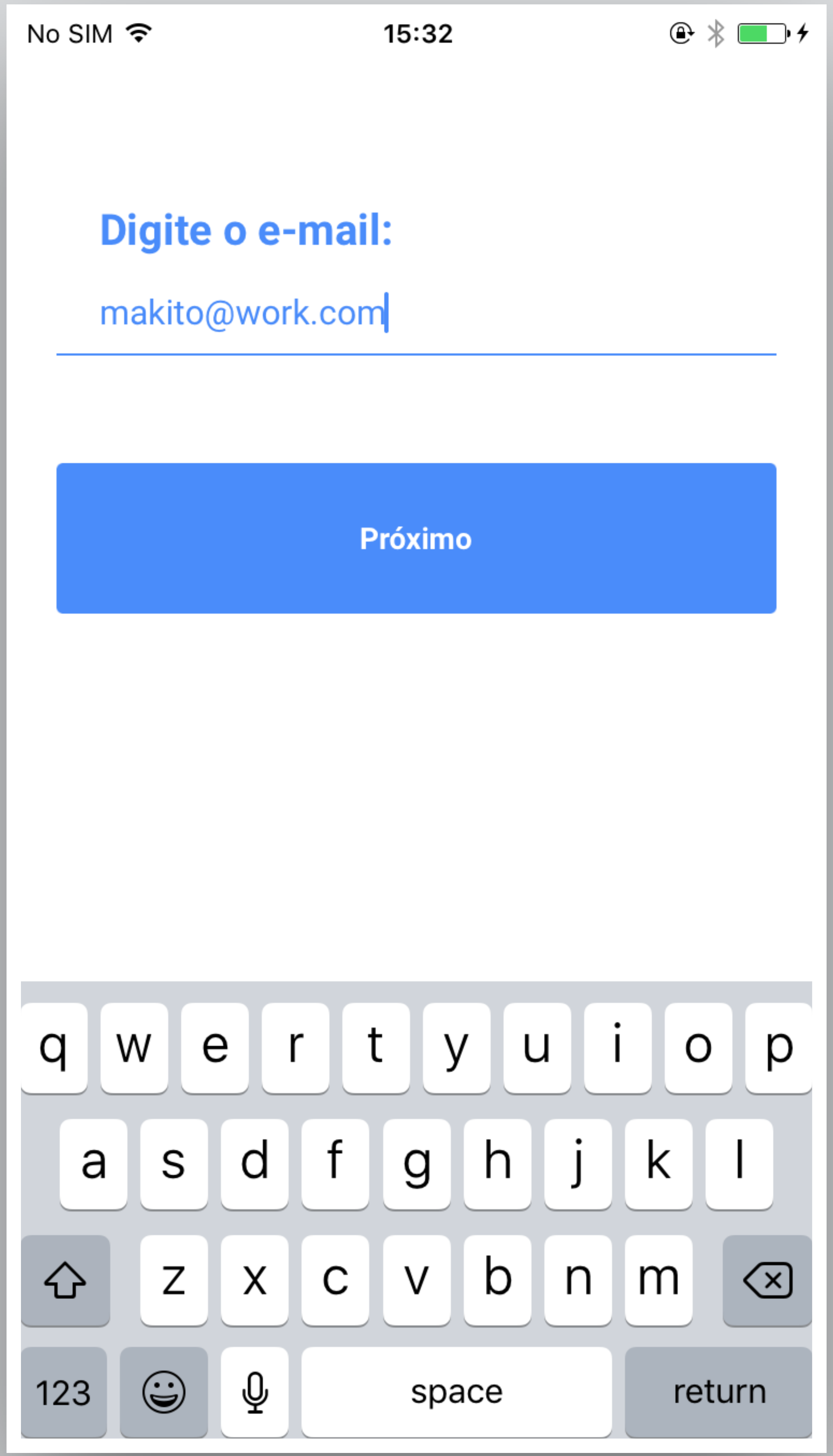
```
class ViewController: UIViewController {
    let viewModel = ViewModel()

    let emailTextfield = UITextField()

    let disposeBag = DisposeBag()

    override func viewDidLoad() {
        emailTextfield.rx.text.orEmpty
            .bindTo(viewModel.emailText)
            .addDisposableTo(disposeBag)
    }
}
```

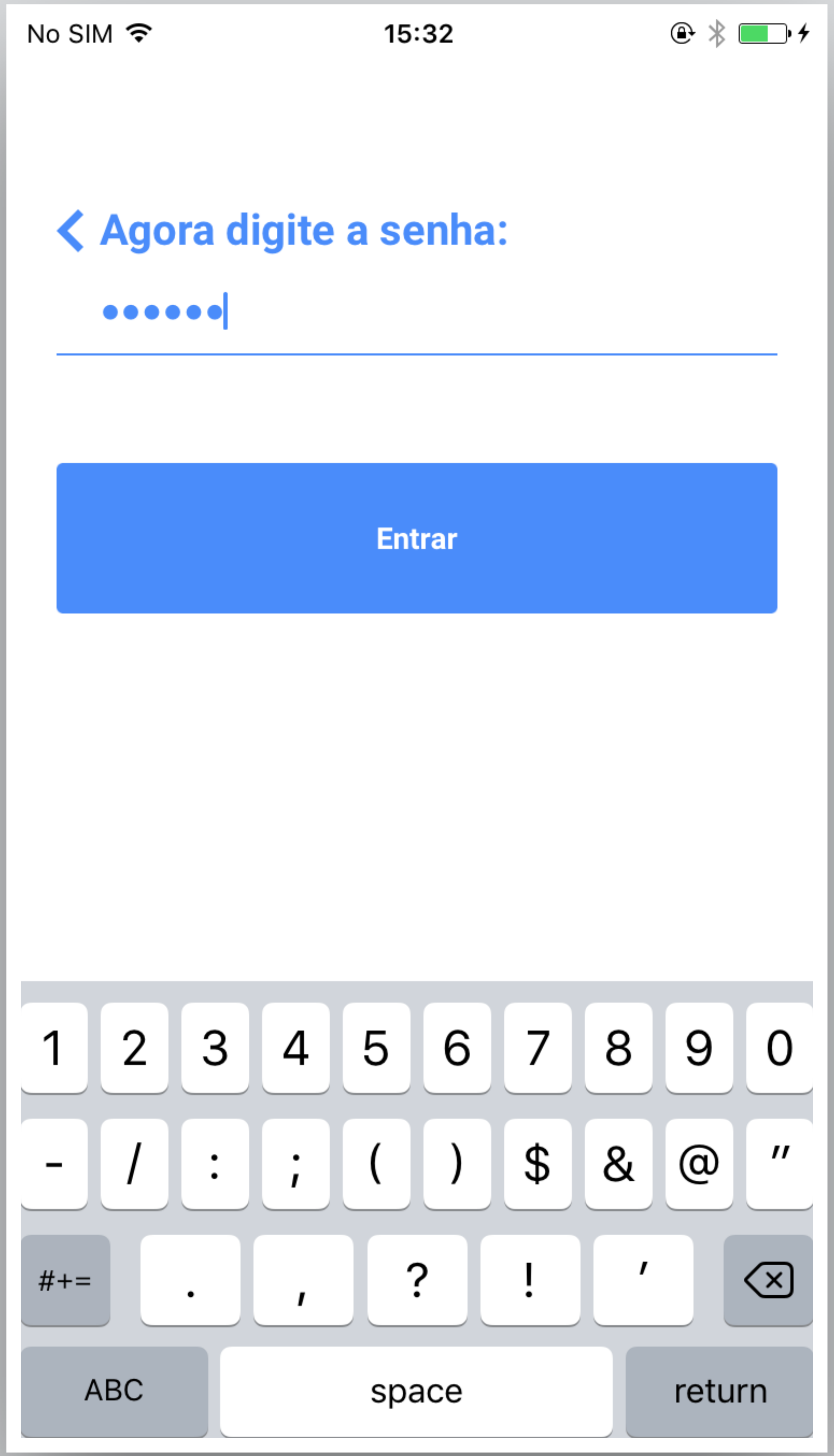
Operators

Transforms, combines and filters a stream of data

- Transforming
- Filtering
- Combining

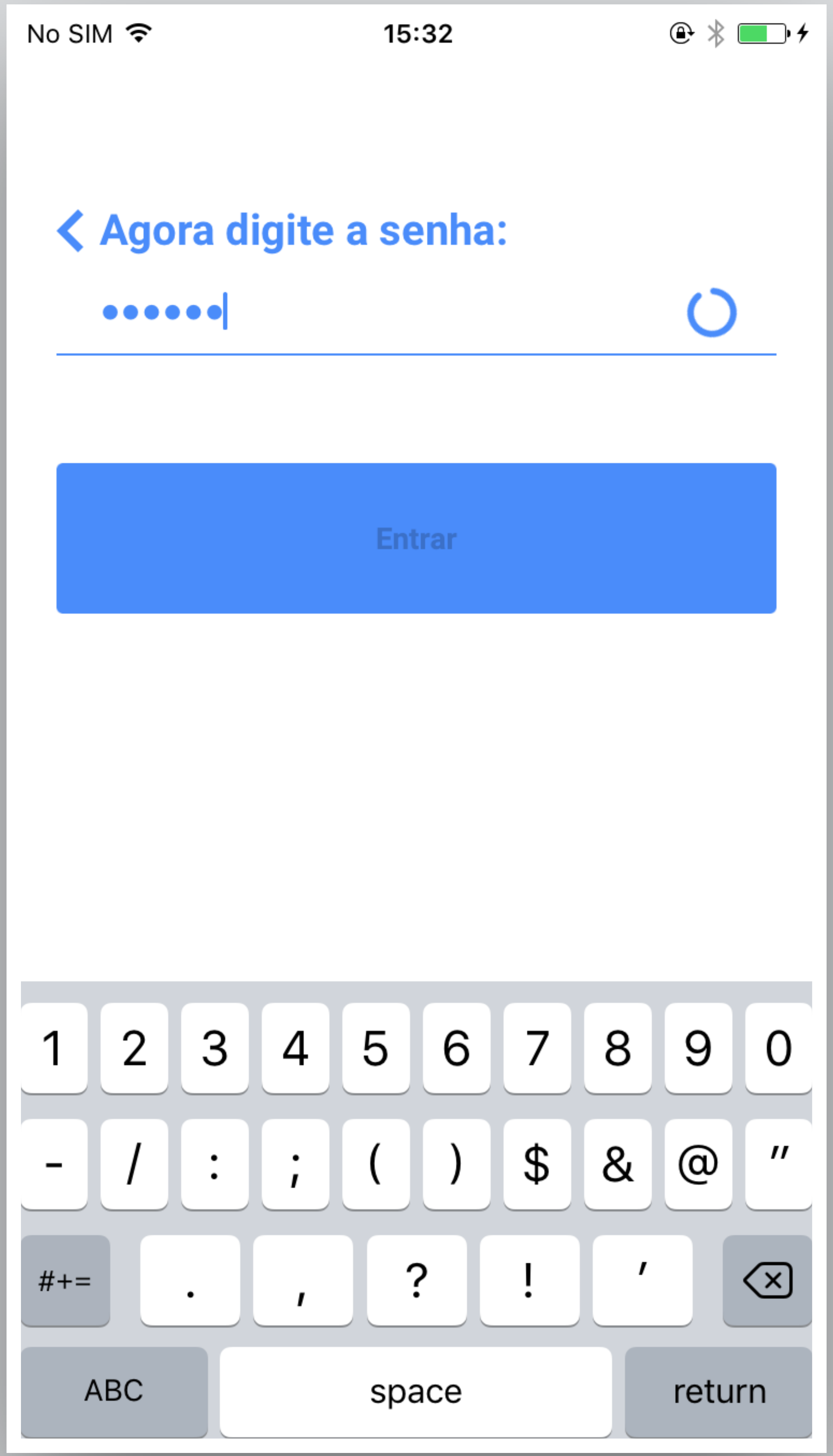

No SIM
15:32
Olá
E-mail
Próximo


No SIM
15:32
Digite o e-mail:
makito@work.com
Próximo
q w e r t y u i o p
a s d f g h j k l
z x c v b n m
123
space
return


No SIM 15:32

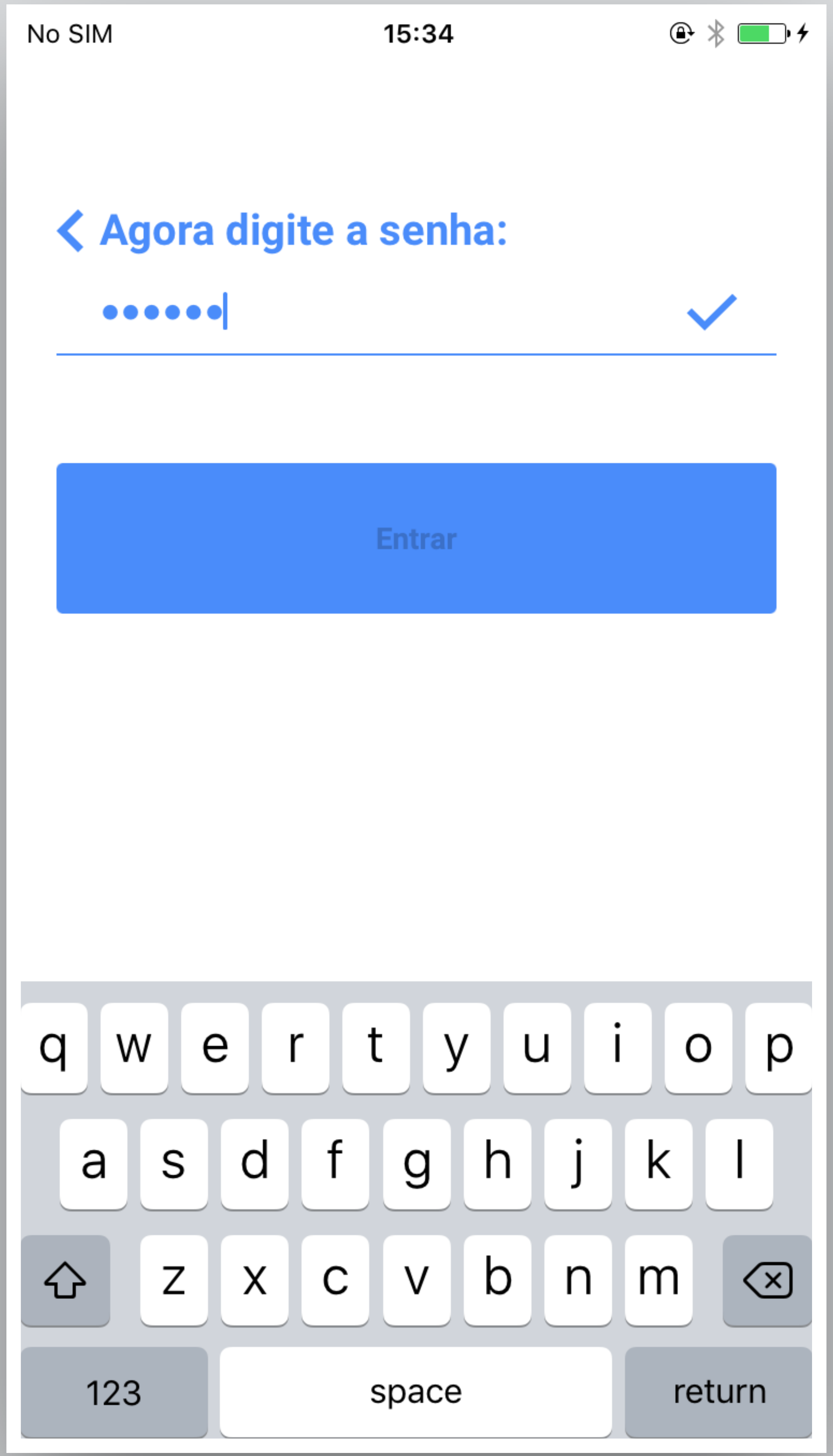
Agora digite a senha:

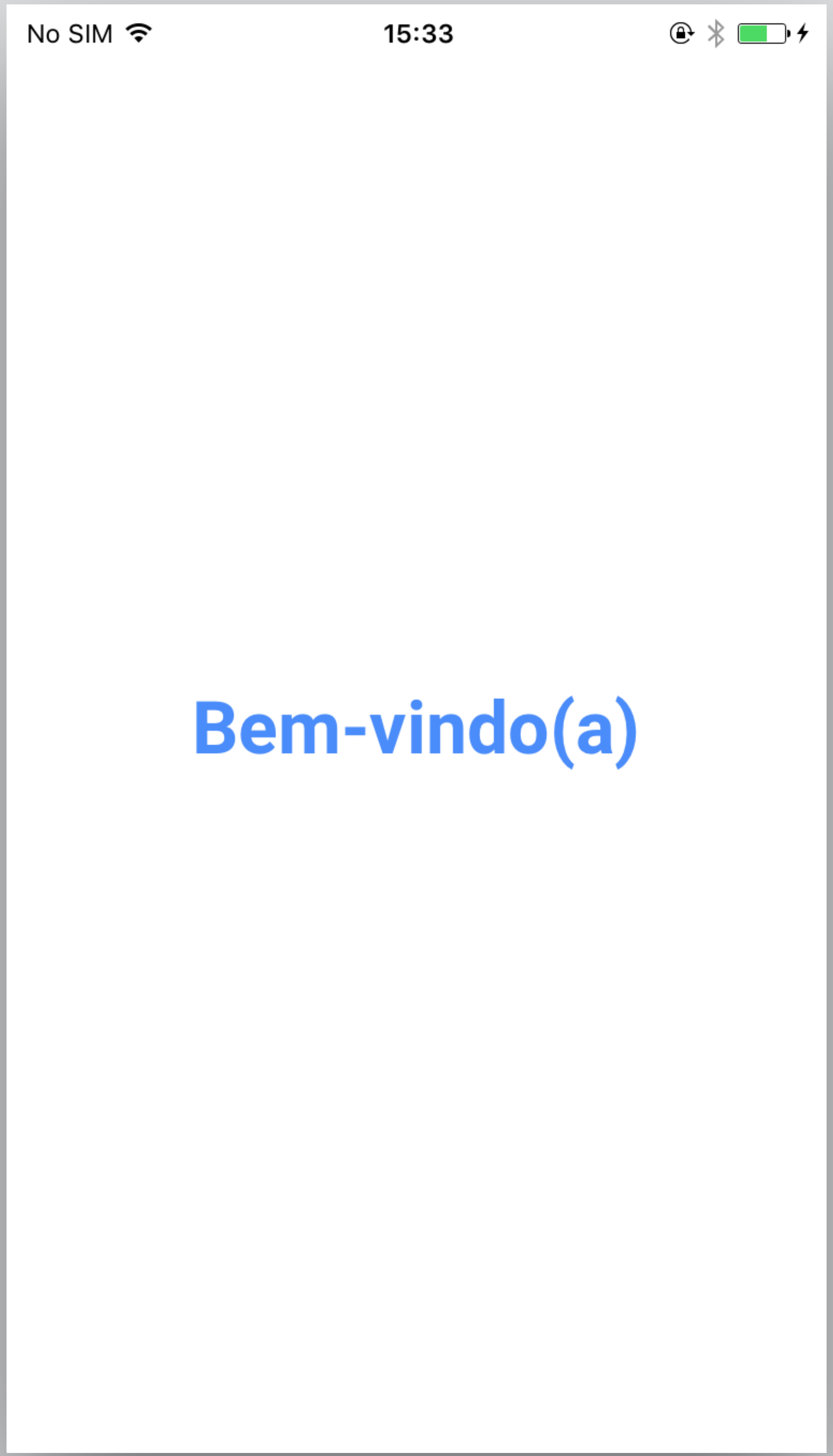
••••••


No SIM 15:32

Agora digite a senha:

••••••


No SIM
15:34
Agora digite a senha:
••••••
Entrar
q w e r t y u i o p
a s d f g h j k l
z x c v b n m
123
space
return


No SIM
15:33
Bem-vindo(a)

App State

```
enum LoginState {
    case emailState
    case passwordState
    case signingInState
    case signedInState
    case welcomeState
}

class LoginViewModel {
    private let currentState = BehaviorSubject<LoginState>(value: .emailState)
}
```

Enter Button Title

```swift
class LoginViewModel {
    private let currentState = BehaviorSubject<LoginState>(value: .emailState)

    let enterButtonTextObservable: Observable<String>

    init() {
        enterButtonTextObservable = currentState.map { state in
            switch state {
            case .emailState: return "Próximo"
            case .passwordState, .signingInState,
                 .signedInState, .welcomeState:
                return "Entrar"
            }
        }
    }
}
```

Click On Enter Button

```
class LoginViewModel {
    private let currentState = BehaviorSubject<LoginState>(value: .emailState)
    let didTapEnterButton = PublishSubject<Void>()

    init() {
        didTapEnterButton.withLatestFrom(currentState) { $1 }
            .subscribe(onNext: { [weak self] state in
                switch state {
                case .emailState:
                    self?.currentState.onNext(.passwordState)
                case .passwordState:
                    self?.currentState.onNext(.signingInState)
                case .signingInState, .signedInState, .welcomeState: break
                }
            }).addDisposableTo(disposeBag)
    }
}
```

Collapse Header

```
class ViewController {
    private func configureFocusObservers() {
        let didFocusEmail = emailTextfield
            .textField.rx.controlEvent(.editingDidBegin).do(onNext: { _ in
                self.headerContainer.showEmailTitle()
            })
        let didFocusPassword = passwordTextfield
            .textField.rx.controlEvent(.editingDidBegin).asObservable()
        Observable.of(
            didFocusEmail,
            didFocusPassword)
                .merge()
                .subscribe(onNext: { [weak self] in
                    self?.collapseHeader()
                }).addDisposableTo(disposeBag)
    }
}
```

Enter Button
Validation Status

```swift
class LoginViewModel {
    private let currentState = BehaviorSubject<LoginState>(value: .emailState)
    let emailText = BehaviorSubject<String>(value: "")
    let passwordText = BehaviorSubject<String>(value: "")

    init() {
        isEnterButtonValid = Observable
            .combineLatest(emailText, passwordText, currentState) { $0 }
            .map { emailText, passwordText, state in
                switch state {
                case .emailState:
                    return StringValidationHelper.isValidEmail(emailText)
                case .passwordState:
                    return passwordText.characters.count >= 6
                case .signingInState, .signedInState, .welcomeState:
                    return false
                }
            }
    }
}
```

Network Request

```swift
class LoginViewModel {
    //...
    private func configureSigningInObserver() {
        let minimumFetchingTime = Observable.just(Void.self)
            .delay(2, scheduler: MainScheduler.instance)
        let requestParameters = Observable.combineLatest(
            emailText.asObservable(),
            passwordText.asObservable()) { $0 }

        currentState.asObservable()
            .filter { $0 == .signingInState }
            .withLatestFrom(requestParameters)
            .flatMap { email, password in
                return Observable.combineLatest(
                    minimumFetchingTime,
                    LoginAPI.loginWithEmail(
                        email: email, password: password)) { $1 }
            }
            .subscribe(onNext: { [weak self] result in
                switch result {
                case .success:
                    self?.currentState.onNext(.signedInState)
                case .failure(let error):
                    self?.currentState.onNext(.passwordState)
                    self?.errorDidOccur.onNext(error)
                }
            }).addDisposableTo(disposeBag)
    }
    //...
}
```

Q & A

WORK
&CO

www.work.co