Garantindo qualidade no App do Nubank

Victor Maraccini
@vmaraccini

Francesco Perrotti-Garcia
@fpg1503

Show of hands

Show of hands

Quem aqui:

Show of hands

Quem aqui:

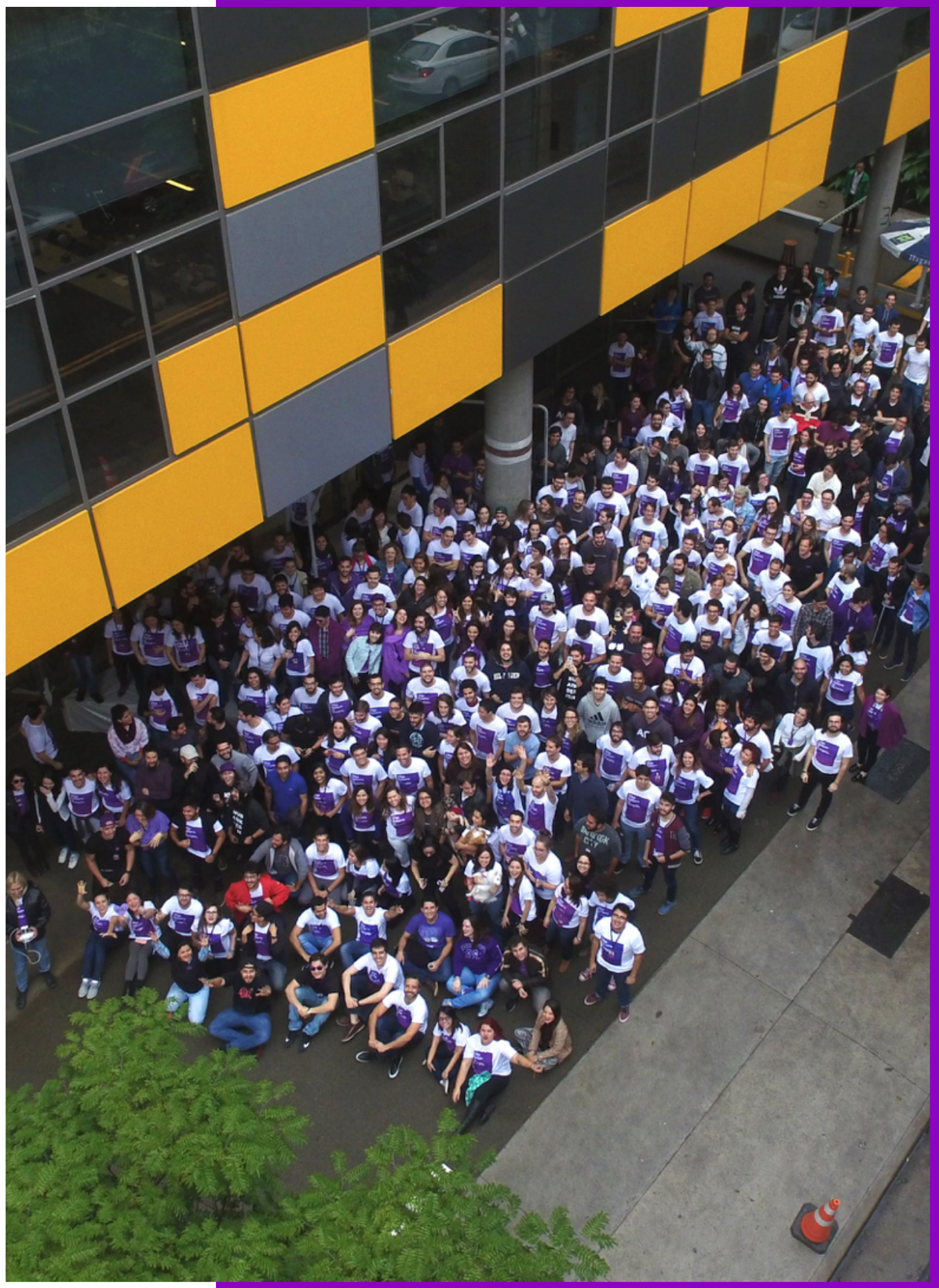
- É cliente do Nubank

O que é Nubank?

O que é Nubank?

O que é Nubank?

- Mais de 1200 funcionários



O que é Nubank?

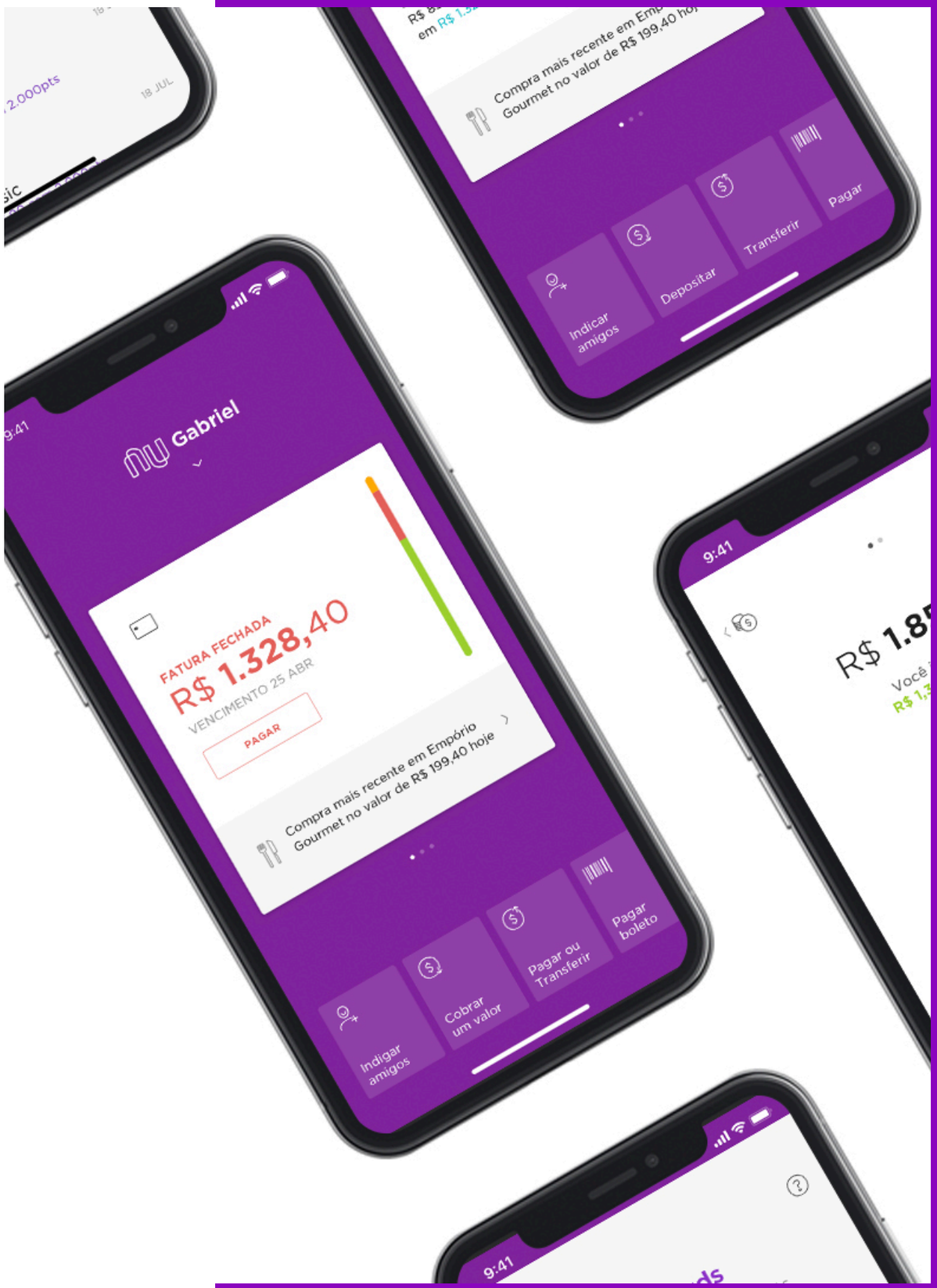
- Mais de 1200 funcionários
- Diversidade

O que é Nubank?

- Mais de 1200 funcionários
- Diversidade
- Mais de 5MM de clientes

O que é Nubank?

- Mais de 1200 funcionários
- Diversidade
- Mais de 5MM de clientes
- Clientes fazem tudo pelo app



O que é Nubank?

- Mais de 1200 funcionários
- Diversidade
- Mais de 5MM de clientes
- Clientes fazem tudo pelo app
- Cultura de qualidade e testes

Show of hands

Show of hands

Quem aqui:

- Acredita que testes melhoram a qualidade do código

Show of hands

Quem aqui:

- Acredita que testes melhoram a qualidade do código

- Mexeu com Mobile

Por que testamos?

Por que testamos?

- Evitar regressão
    - Mais velocidade para refatorar e fazer mudanças
    - Novos engenheiros tem confiança nas entregas

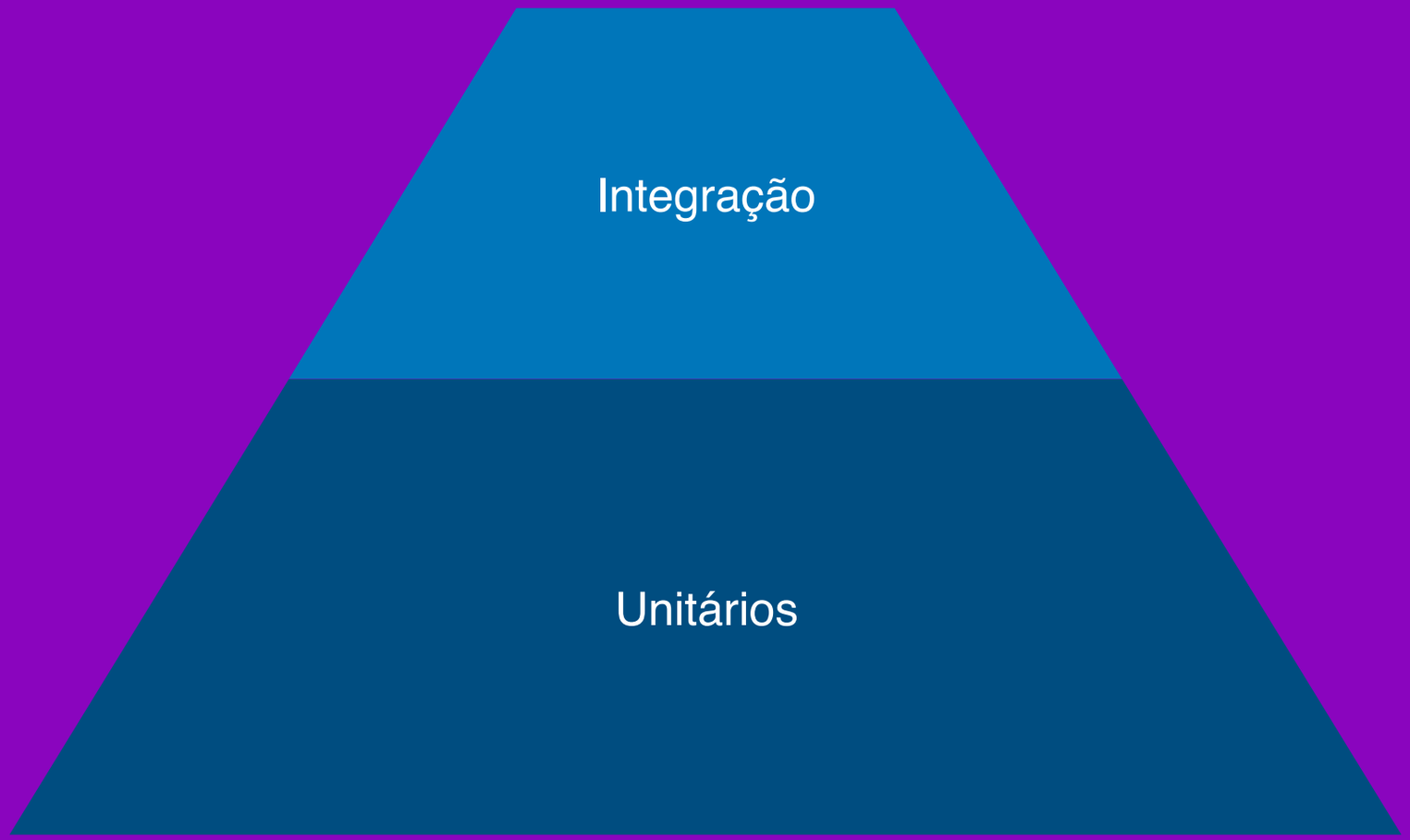
Por que testamos?

- Evitar regressão
    - Mais velocidade para refatorar e fazer mudanças
    - Novos engenheiros tem confiança nas entregas
- Garantir comportamentos existentes
    - De forma automatizada
    - Em diferentes condições de uso

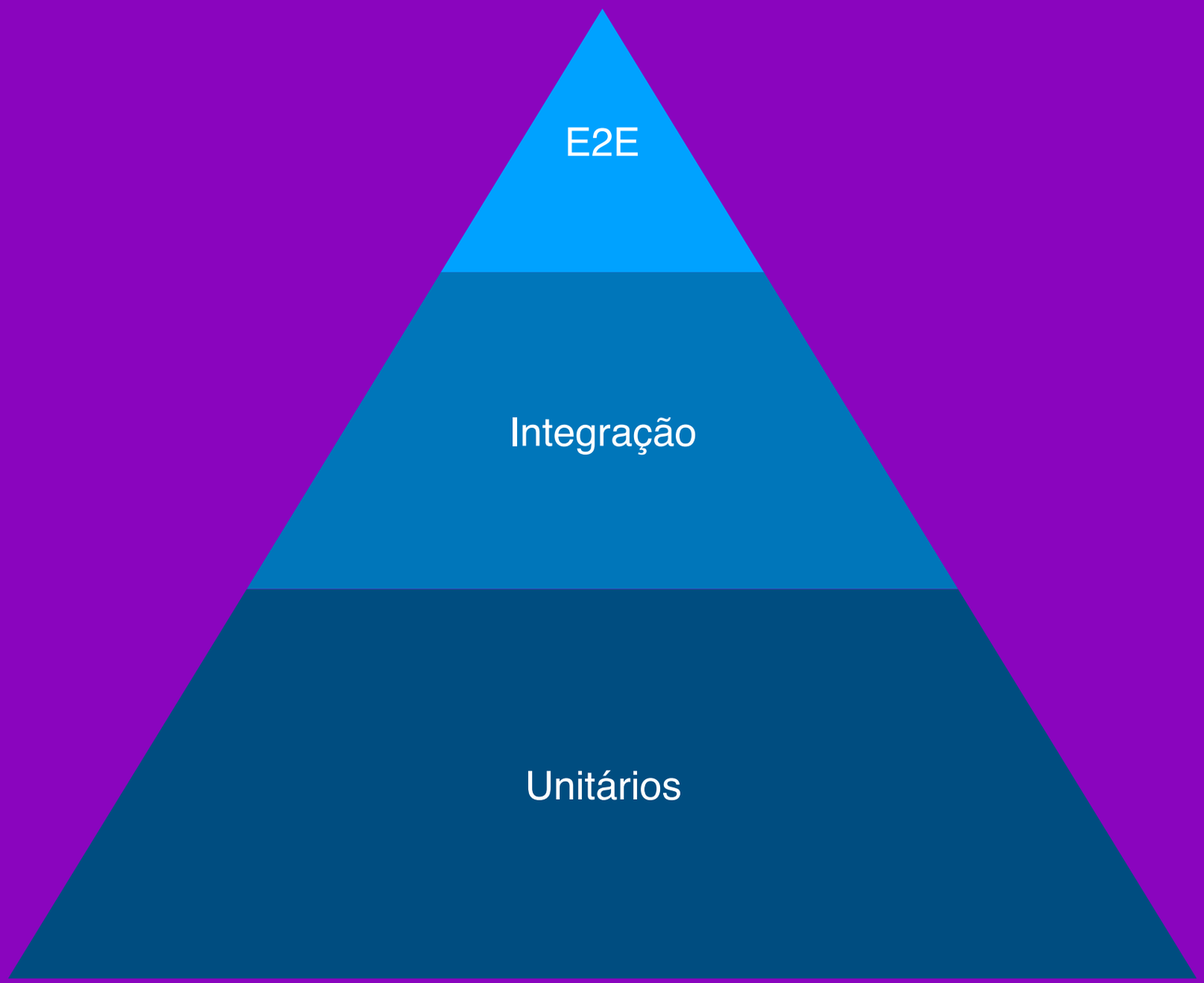
Pirâmide de testes

Pirâmide de testes

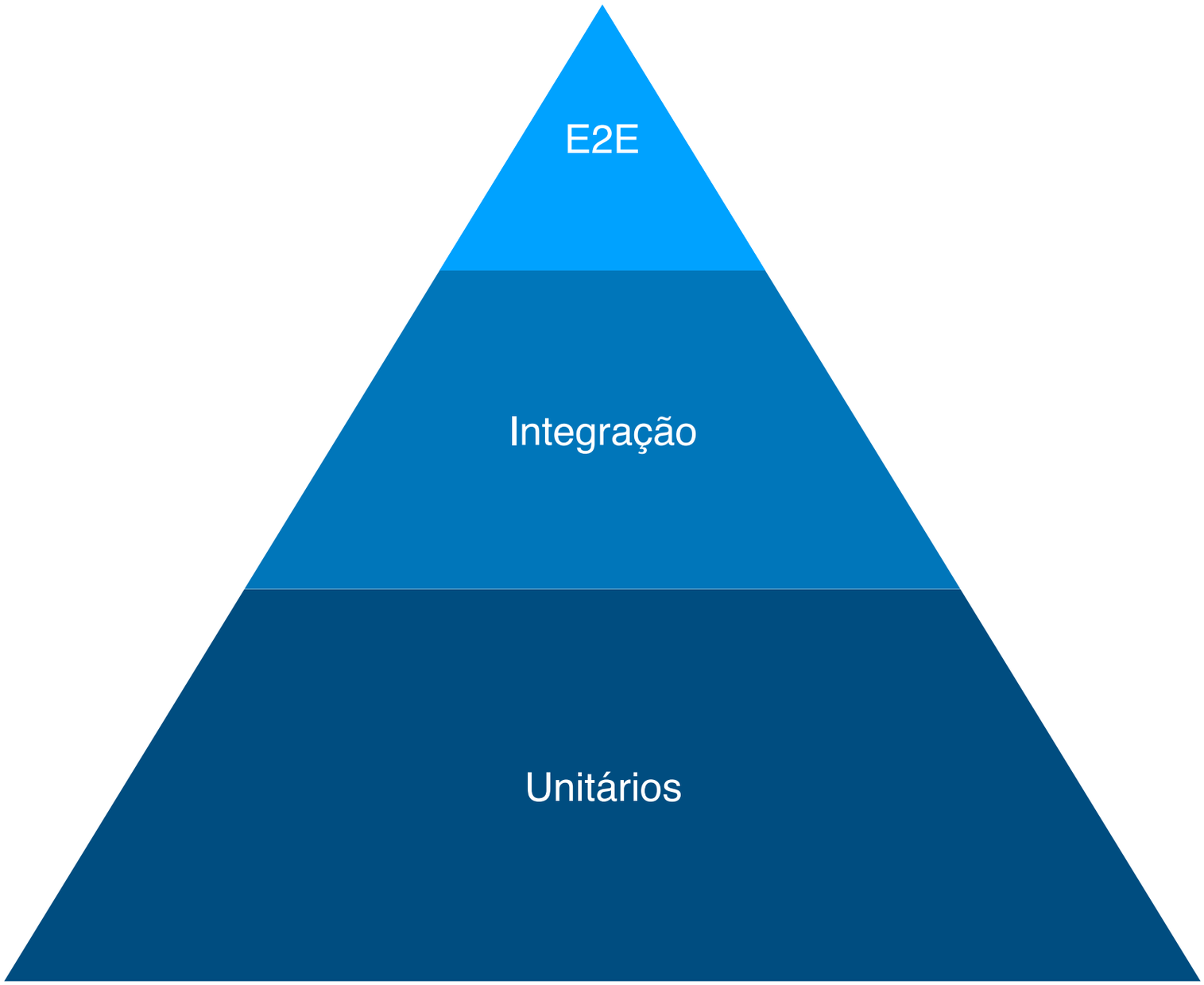
Pirâmide de testes



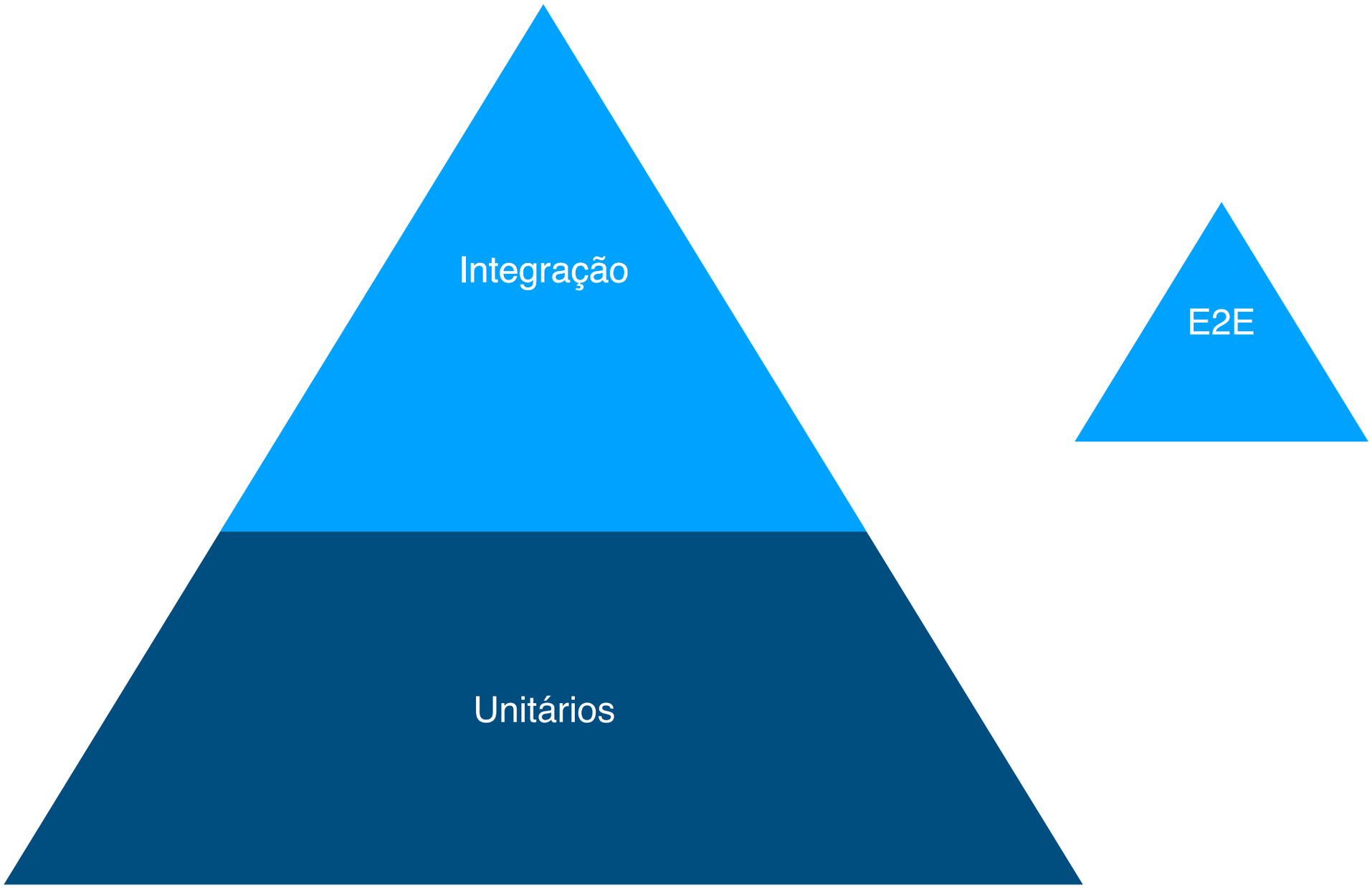
Pirâmide de testes



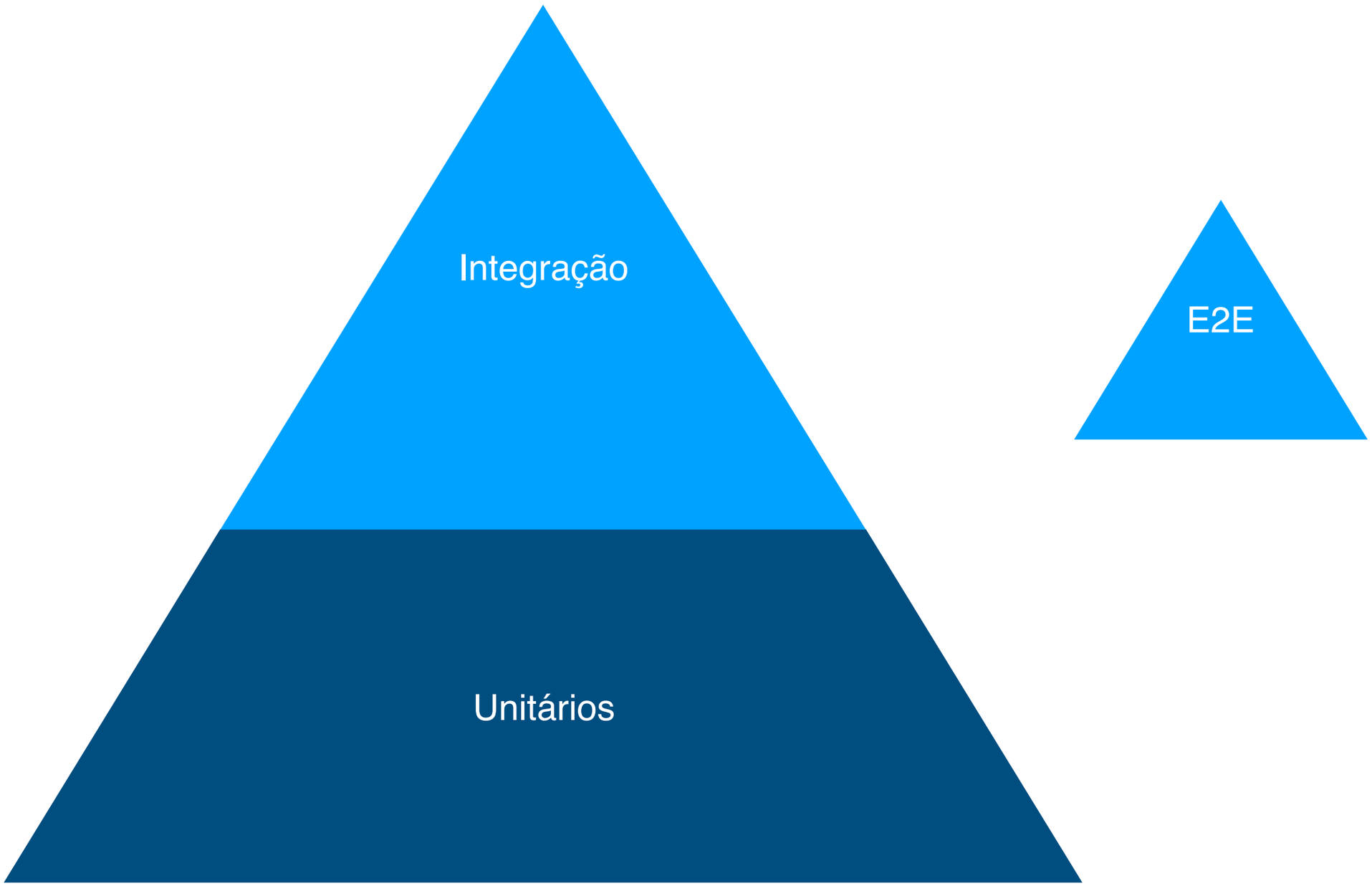
Nossa pirâmide de testes em Mobile



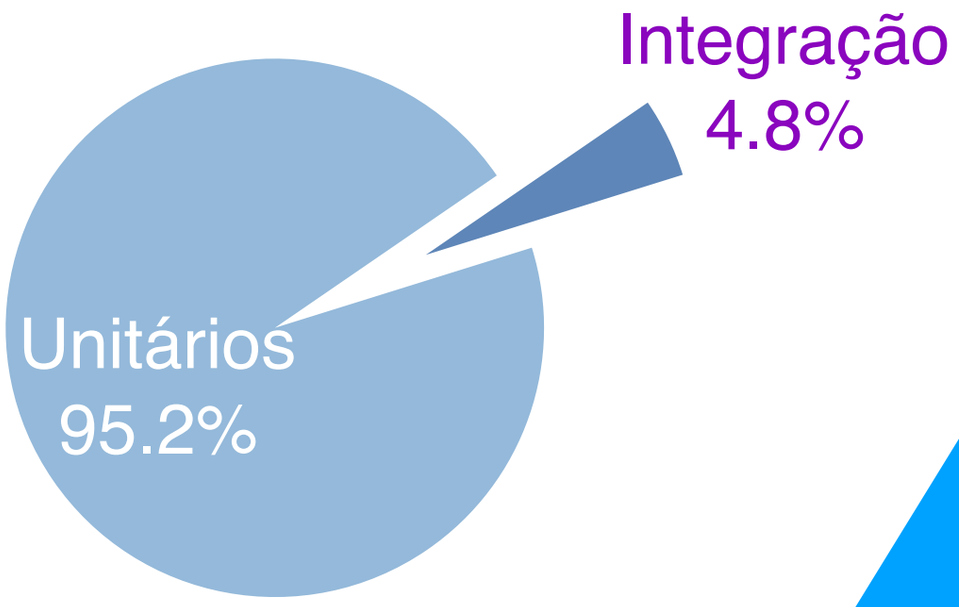
Nossa pirâmide de testes em Mobile

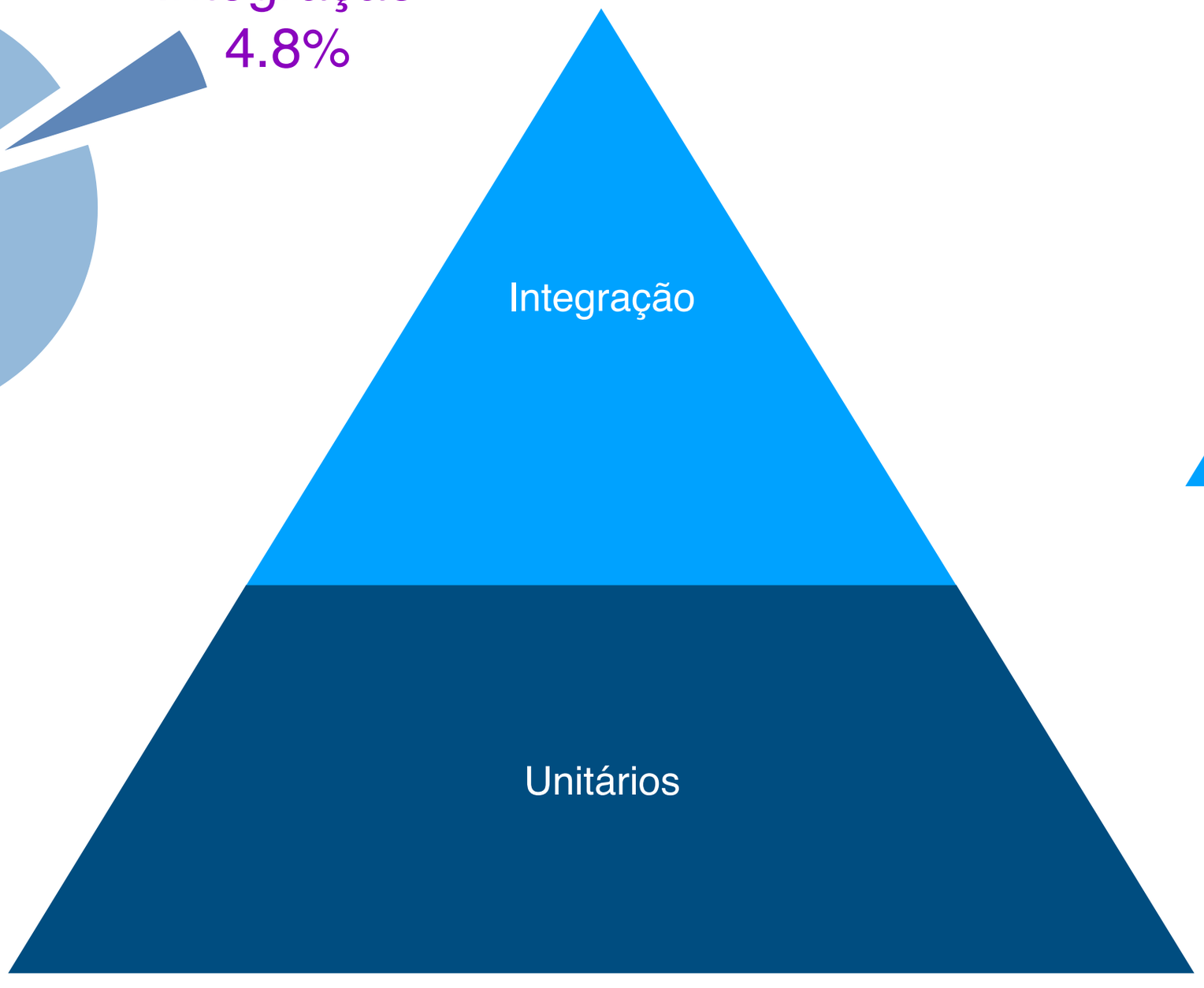


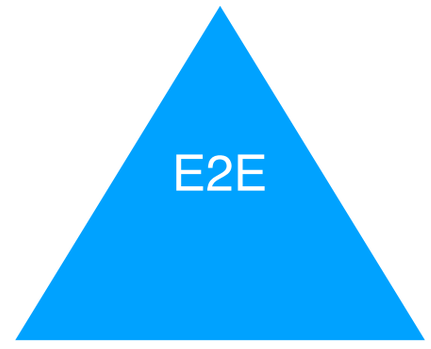
Nossa pirâmide de testes em Mobile



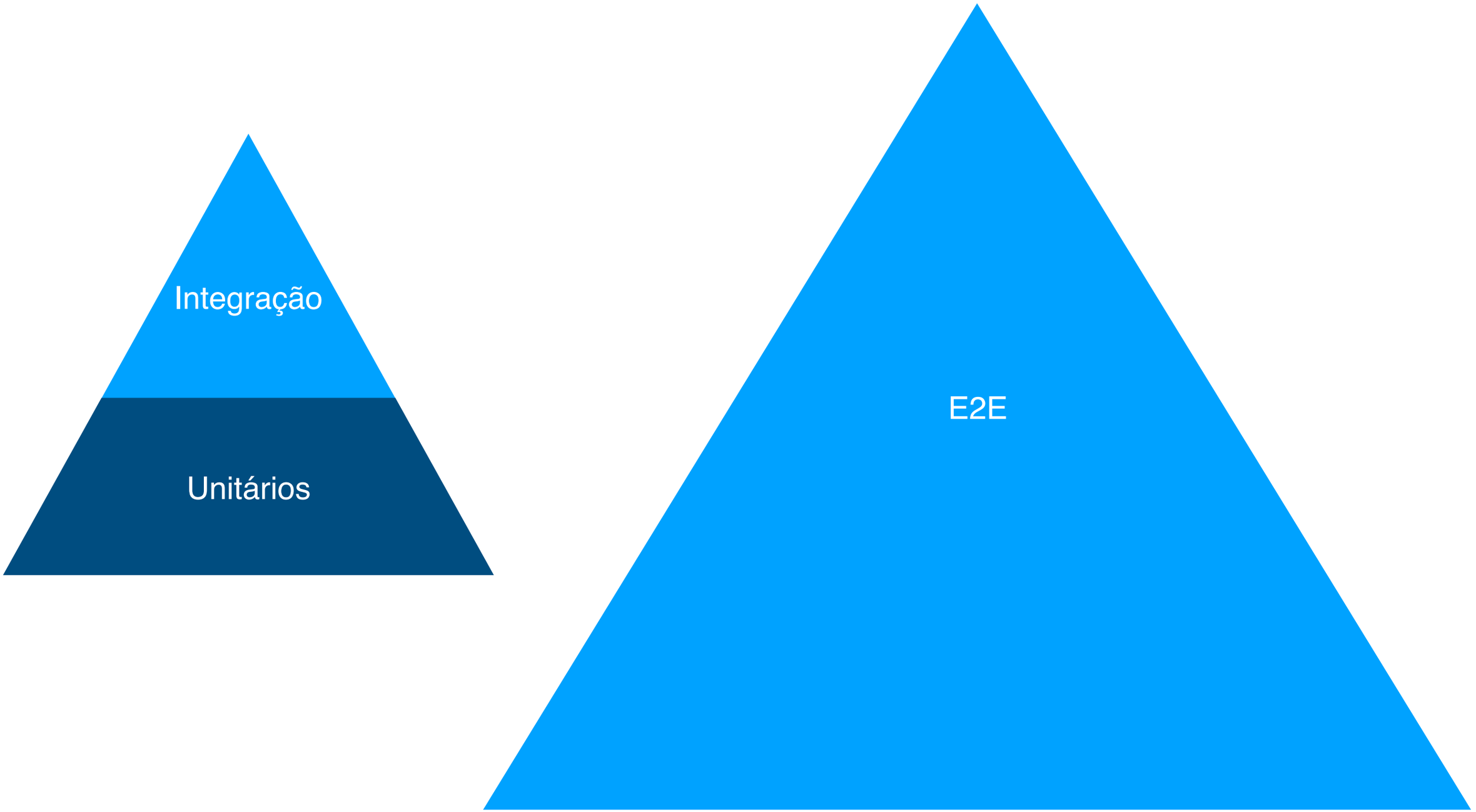
Nossa pirâmide de testes em Mobile



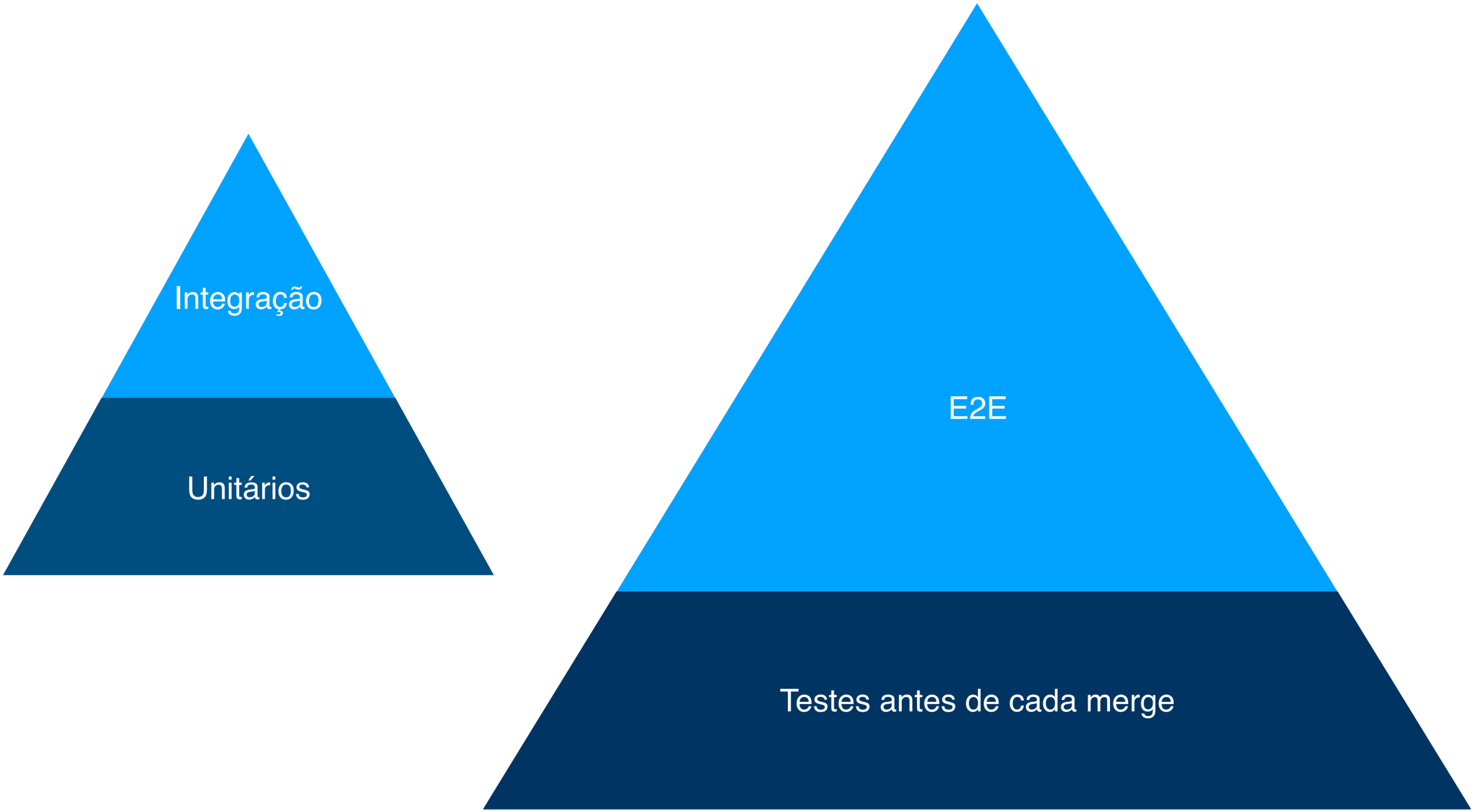




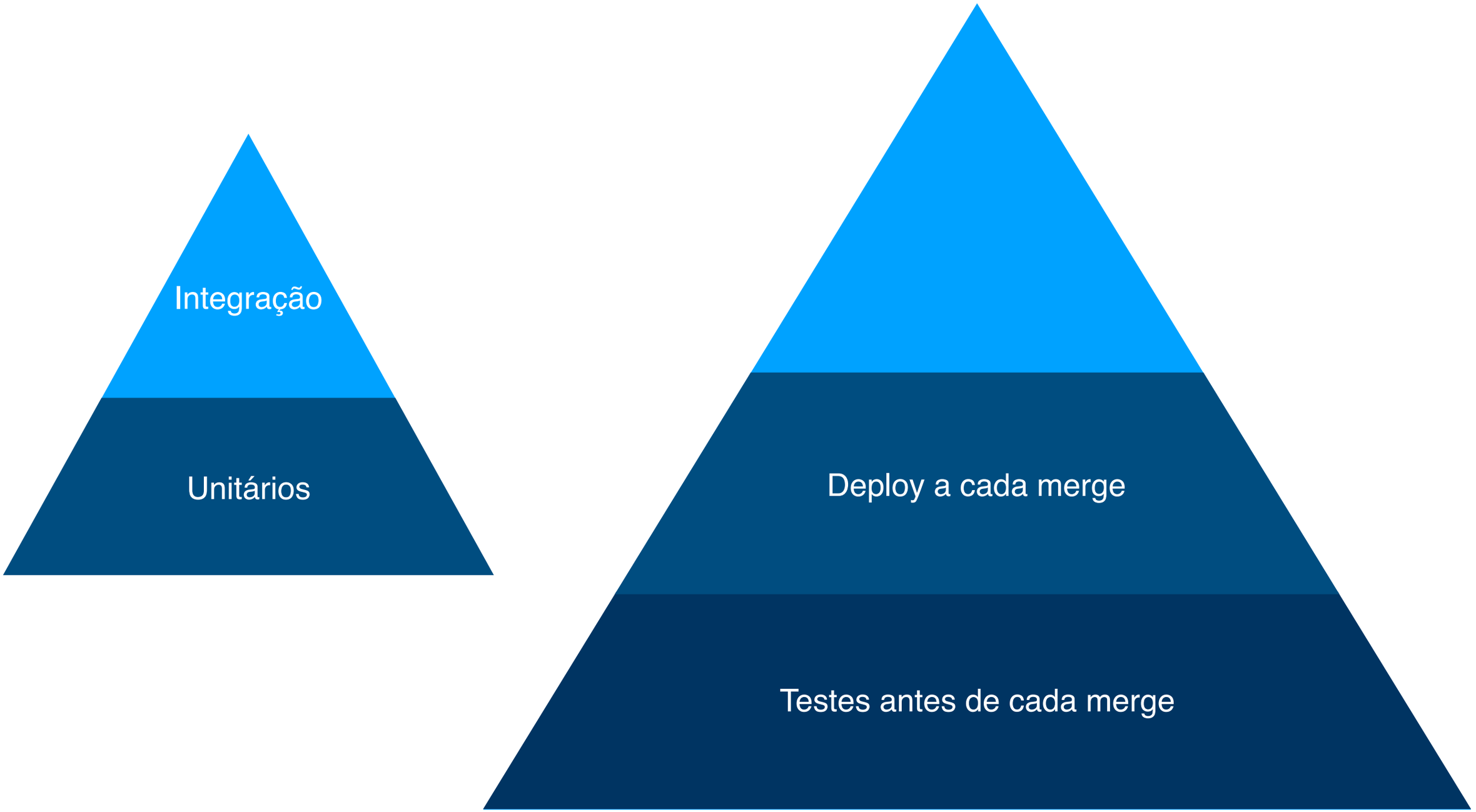
Nossa pirâmide de testes em Mobile



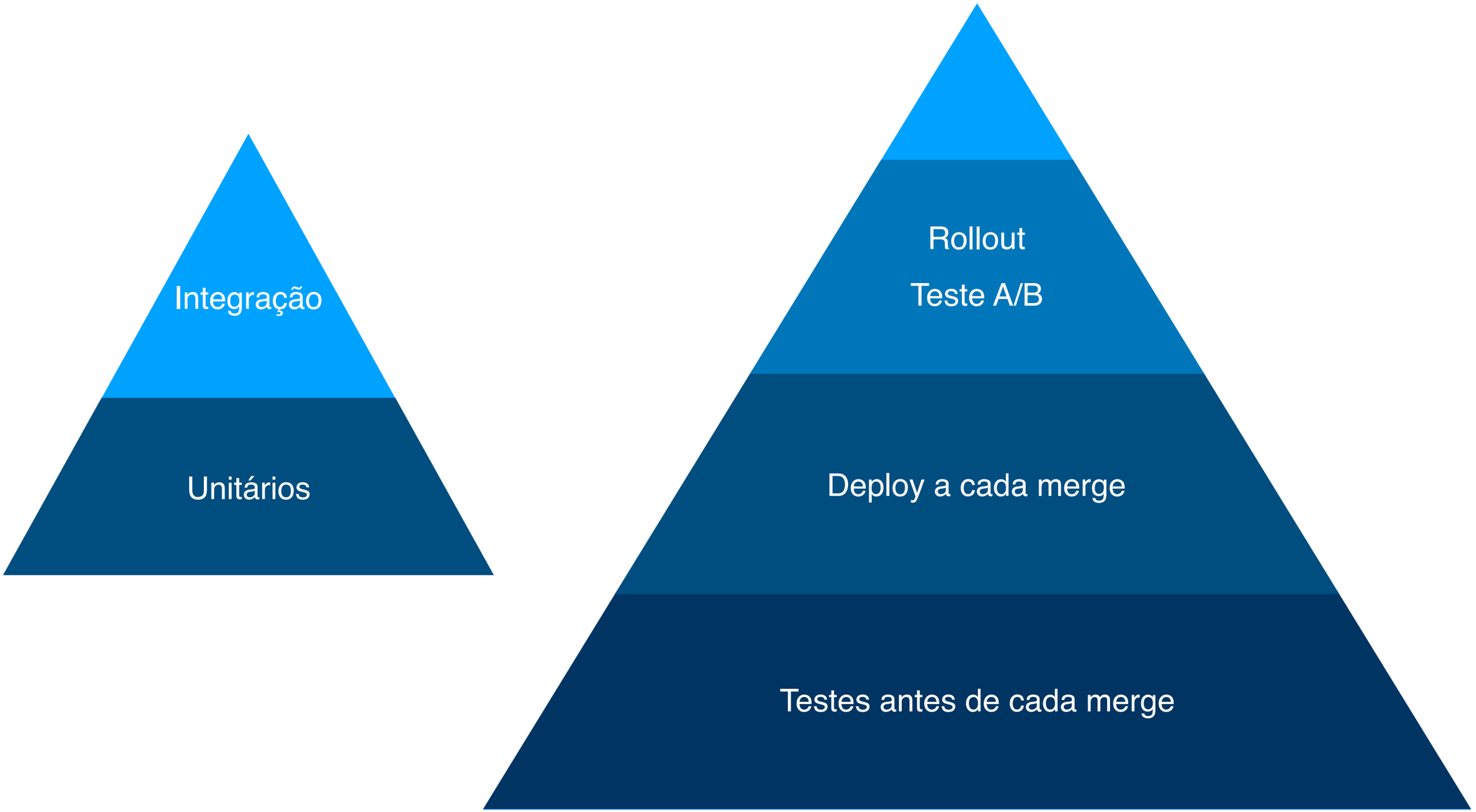
Nossa pirâmide de testes em Mobile



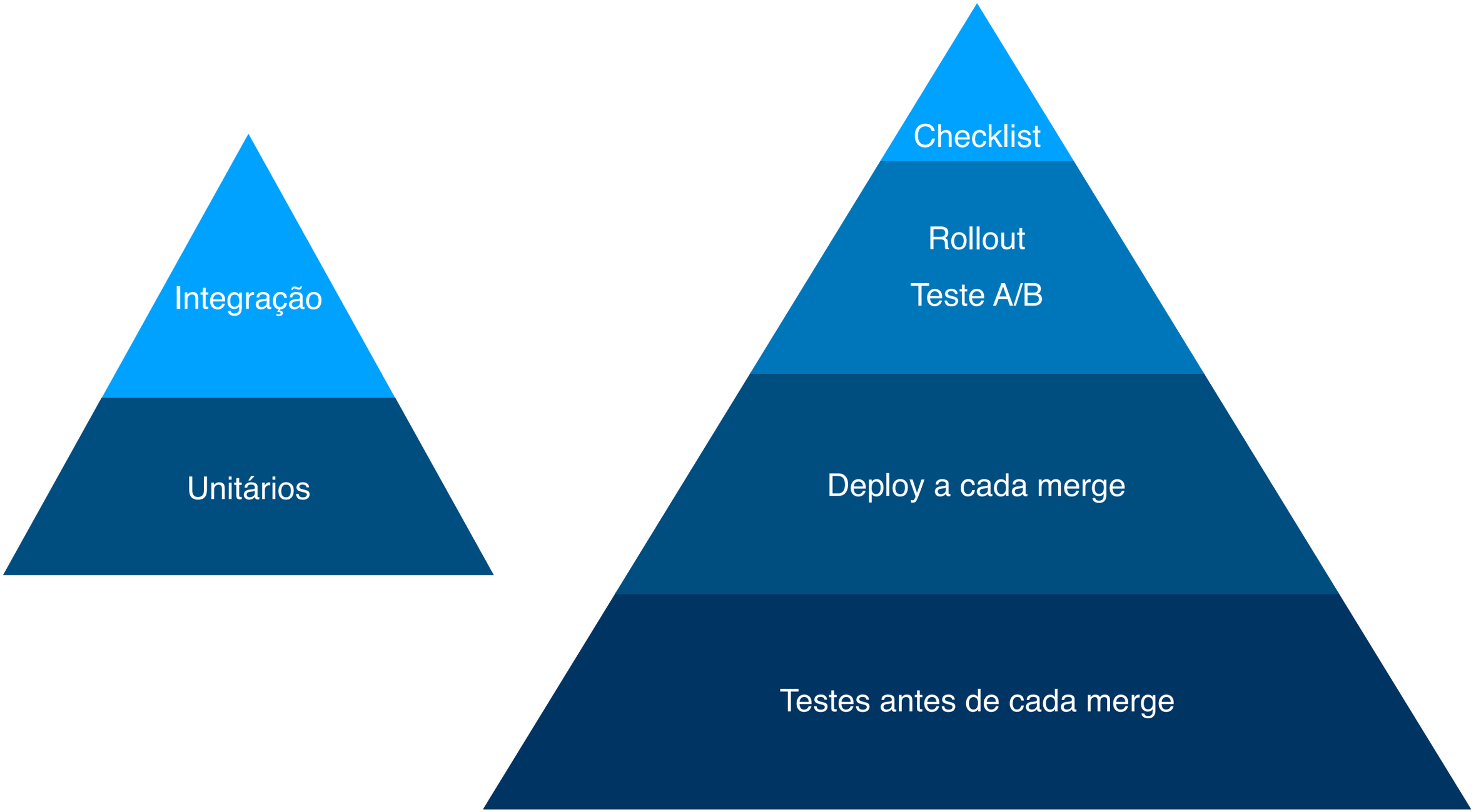
Nossa pirâmide de testes em Mobile



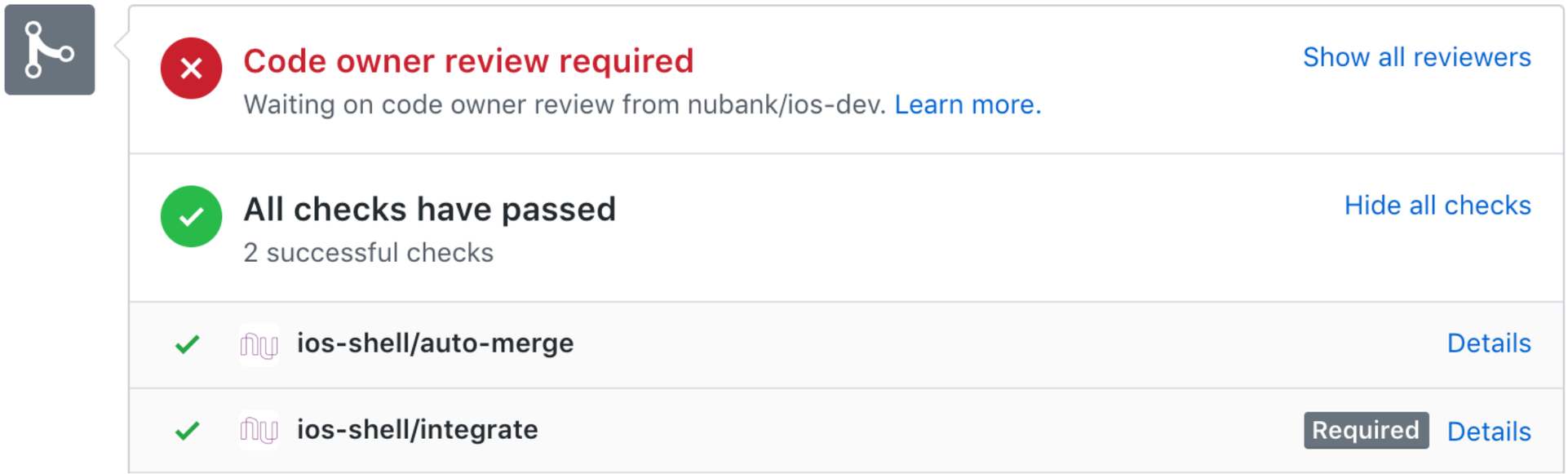
Nossa pirâmide de testes em Mobile



Nossa pirâmide de testes em Mobile

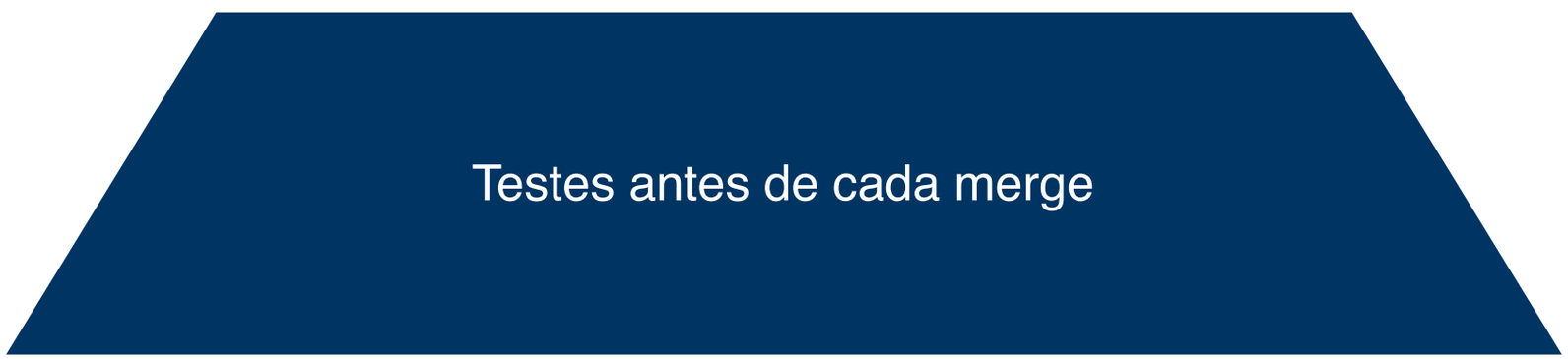


Integração contínua



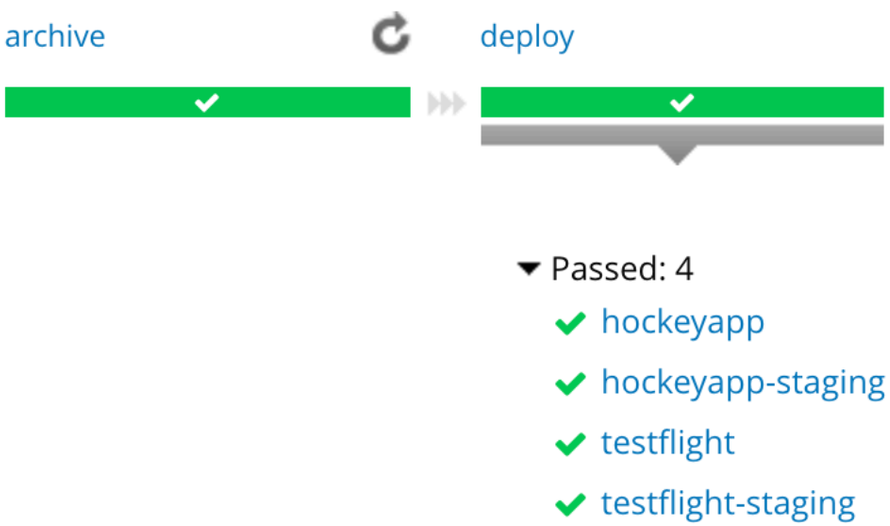
Code Review

Testes passando



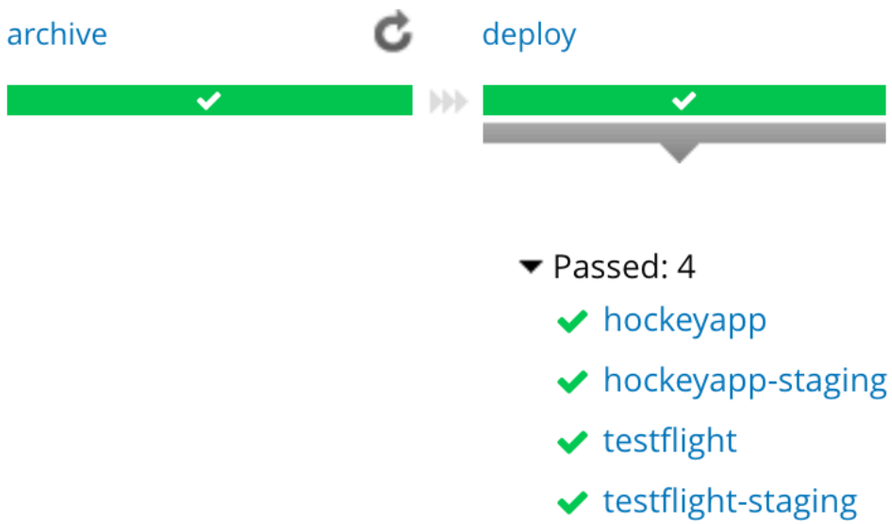
Entrega contínua

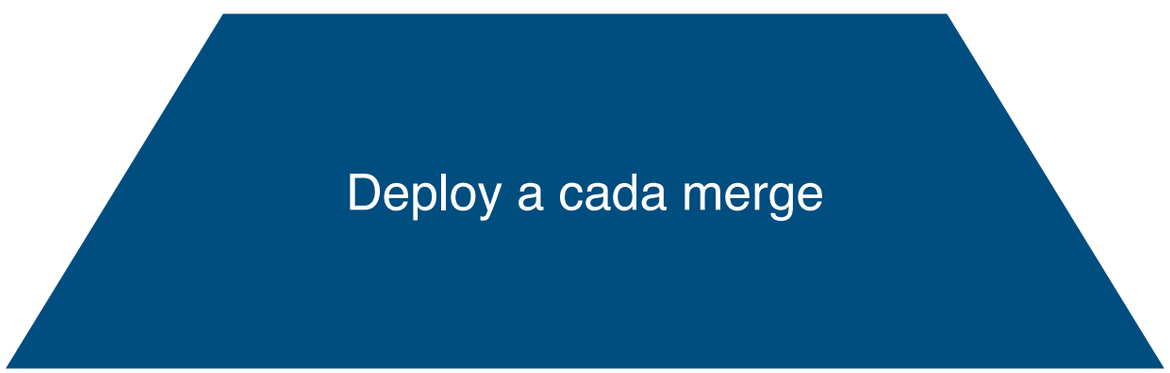
Entrega contínua



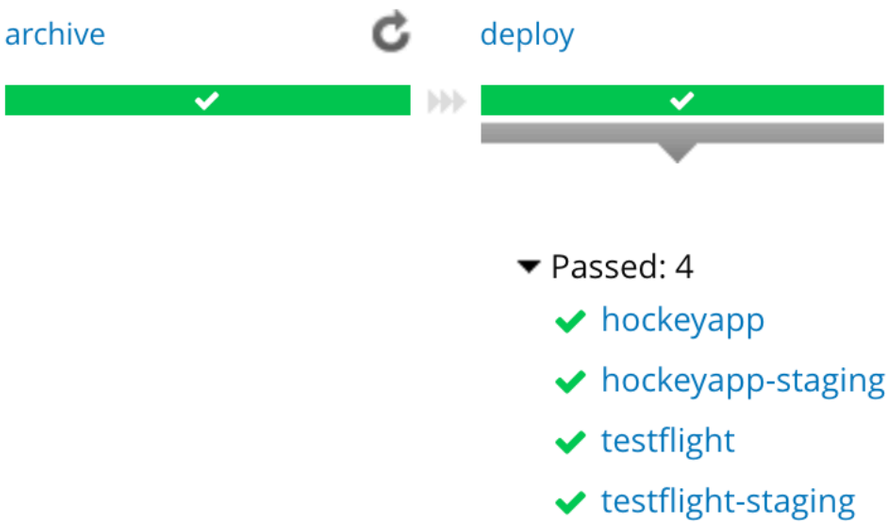
Deploy a cada merge

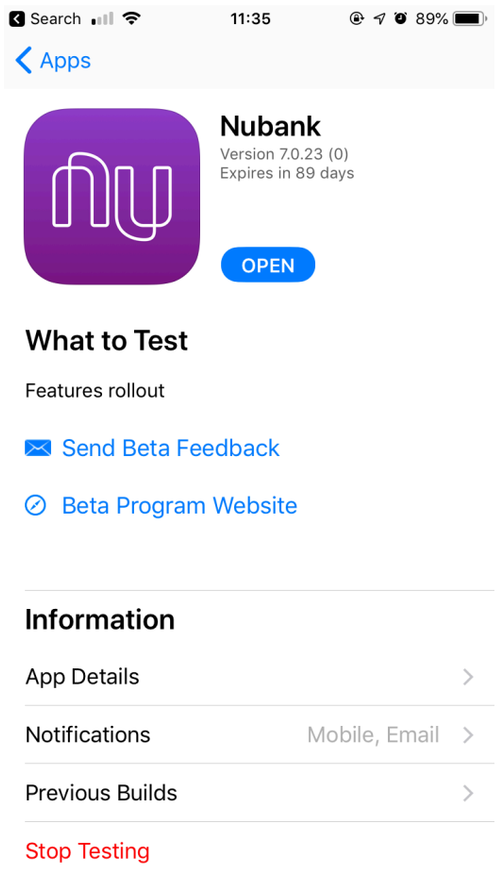
Entrega contínua





Entrega contínua





Deploy a cada merge

Testes unitários

Testes unitários

- Foco no comportamento individual de componentes

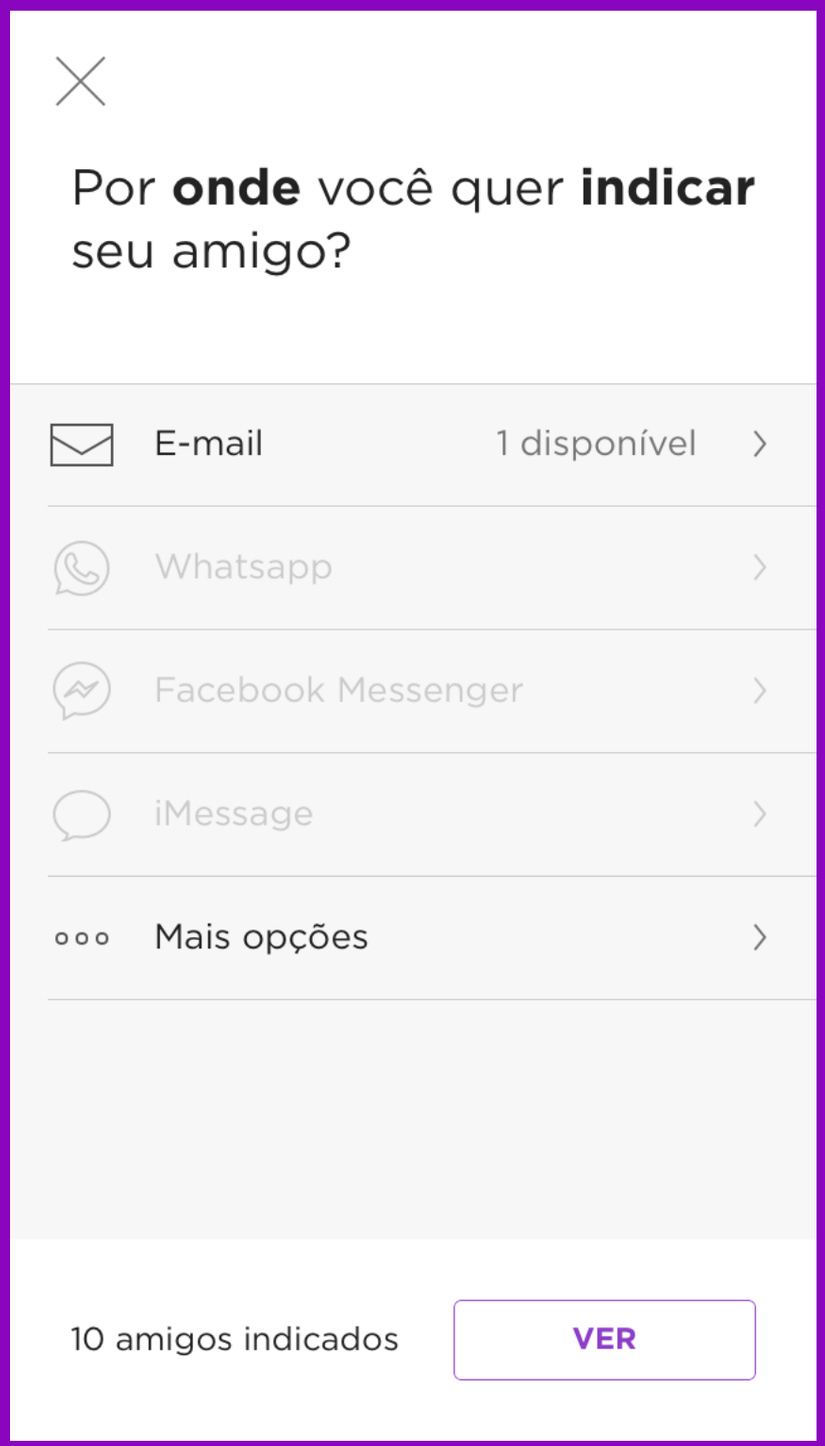
Testes unitários

- Foco no comportamento individual de componentes
- Desacoplado

Testes unitários

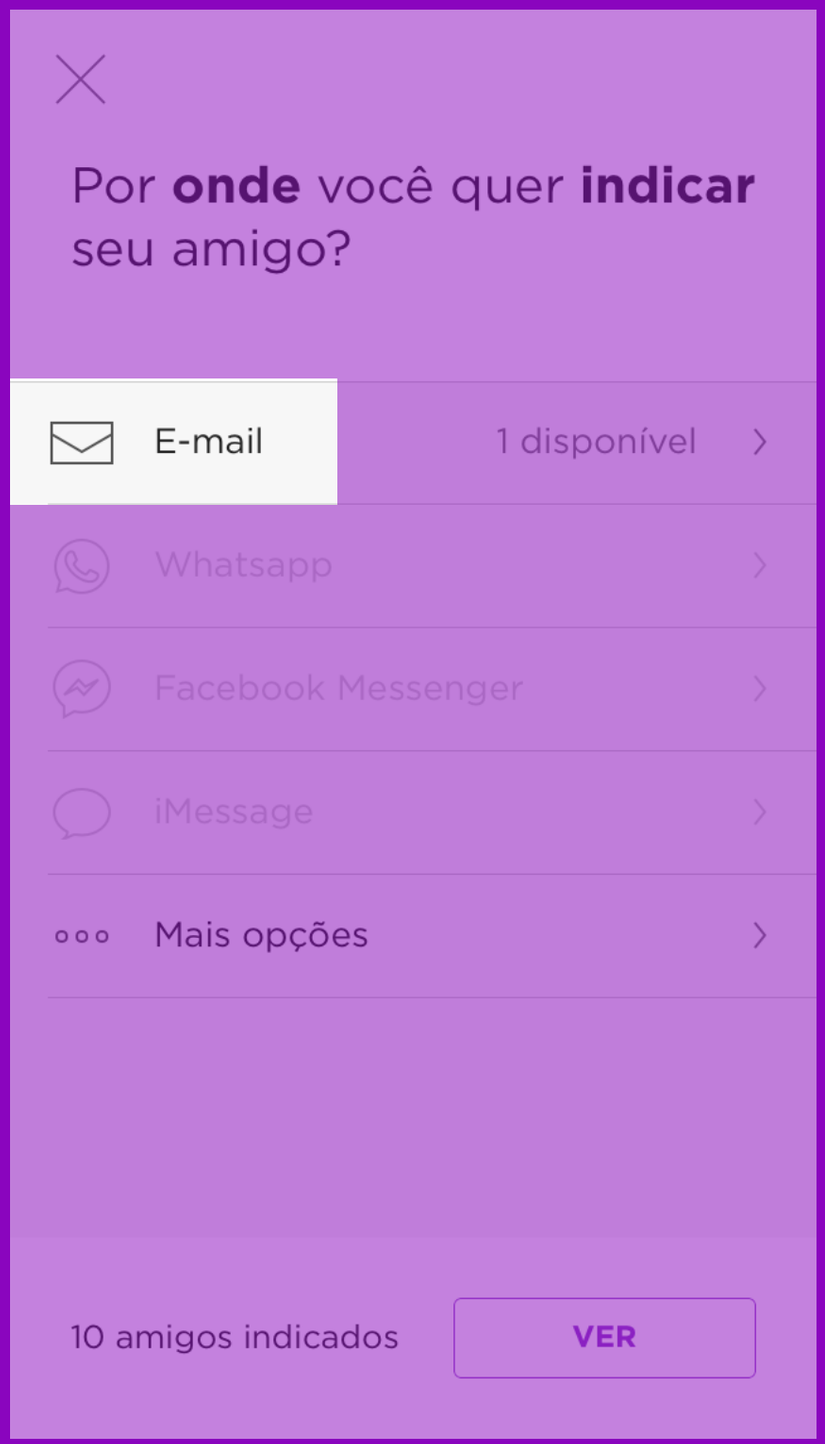
- Foco no comportamento individual de componentes
- Desacoplado
- Especializado

Contexto



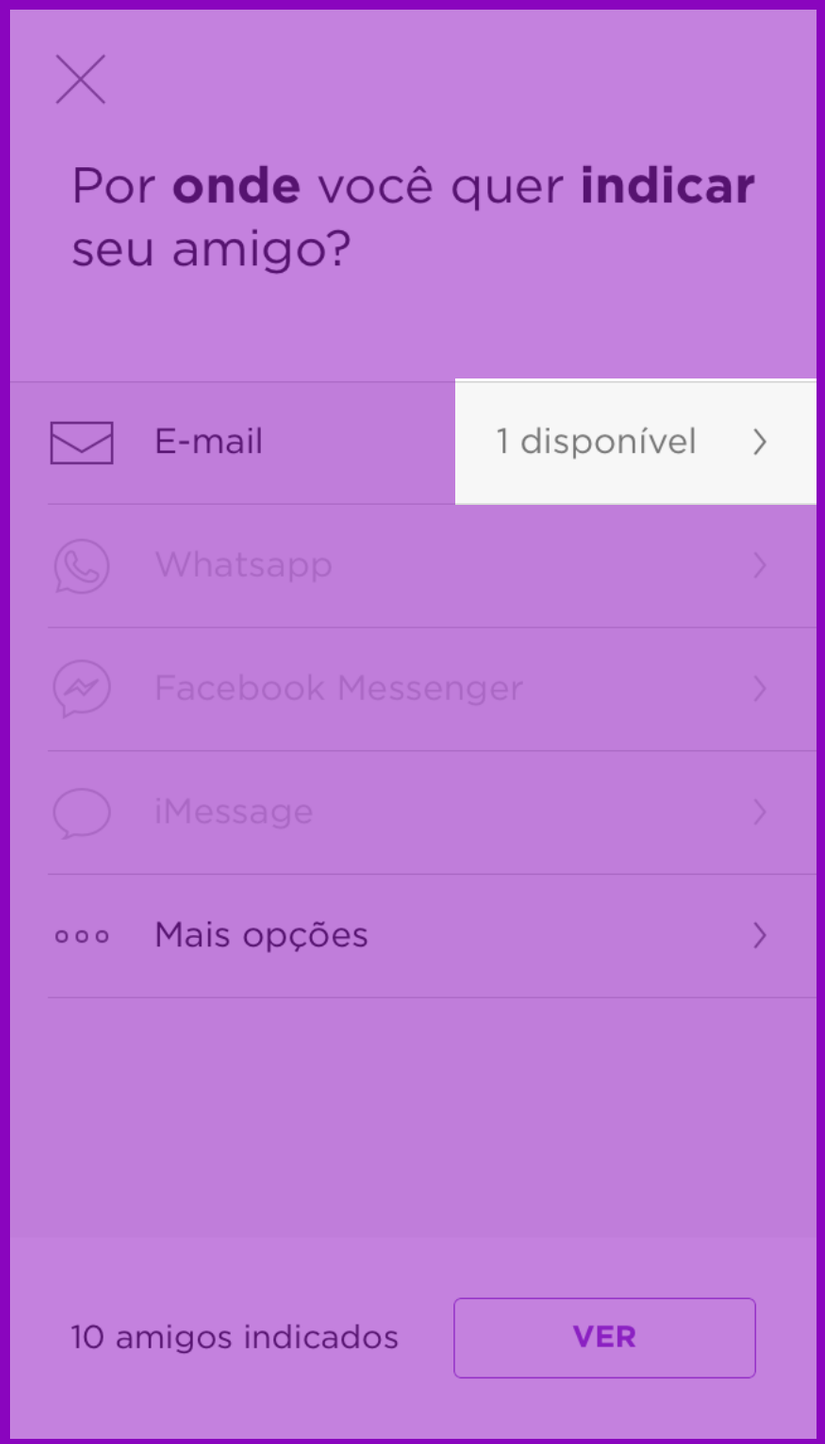

```
struct MGMChannelData {
    let type: MGMChannel
    let count: Int?
    let canOpen: Bool
}
```

Contexto



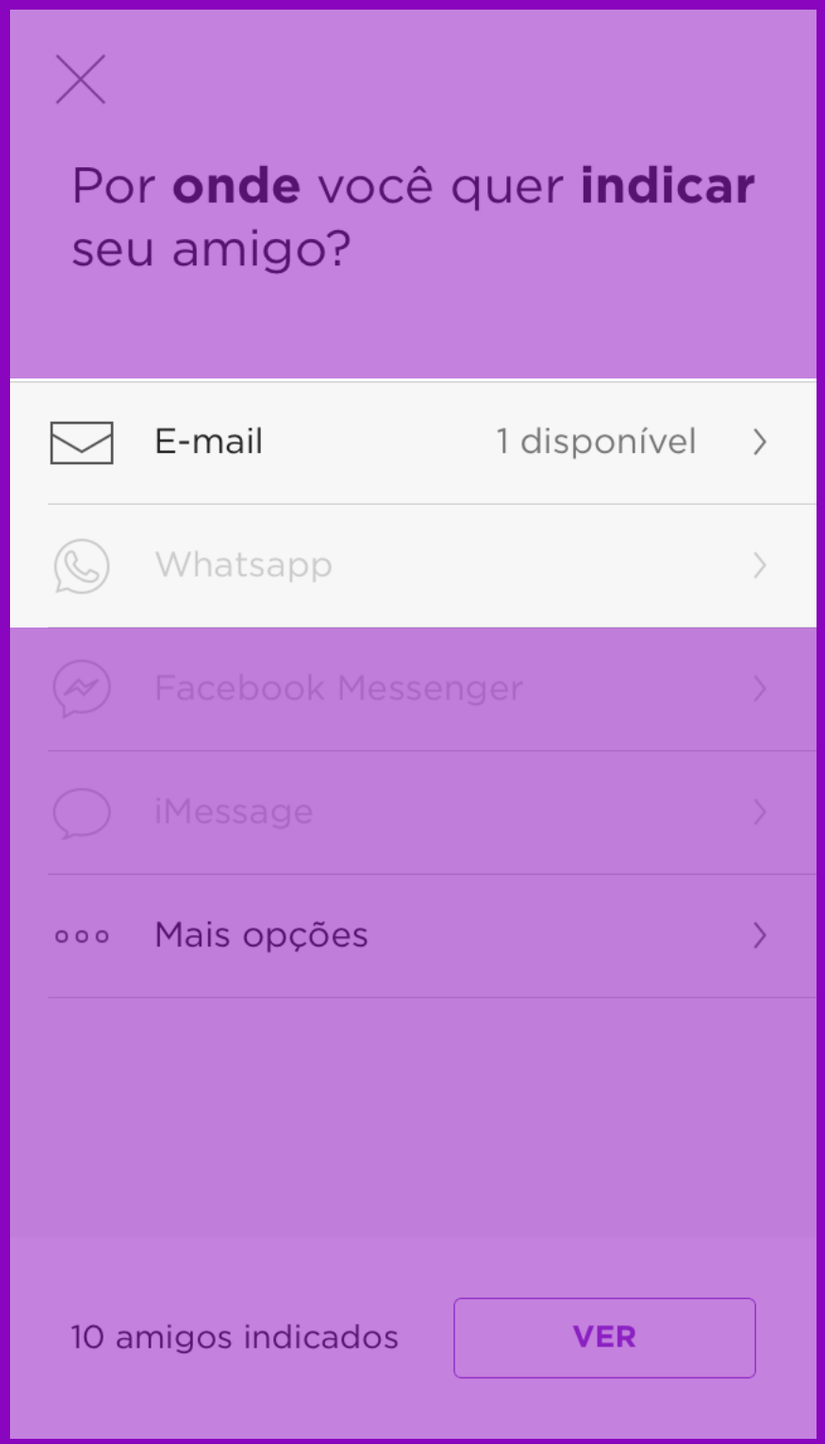

```
struct MGMChannelData {
    let type: MGMChannel
    let count: Int?
    let canOpen: Bool
}
```

Contexto




```
struct MGMChannelData {
        let type: MGMChannel
        let count: Int?
        let canOpen: Bool
}
```

Contexto




```
struct MGMChannelData {
        let type: MGMChannel
        let count: Int?
        let canOpen: Bool
}
```

Dependências implícitas

```
struct MGMChannelData {
    let type: MGMChannel
    let count: Int?
    let canOpen: Bool
}

class MGMInteractor {
    ...
    func createChannelData(invitations: Int,
                           channel: Customer.Channel) -> MGMChannelData {
        switch channel {
        case .whatsapp:
            let canOpenWhatsapp = UIApplication.shared
                                               .canOpenURL(Hypermedia.whatsApp)
            return MGMChannelData(type: .whatsapp,
                                  count: nil,
                                  canOpen: canOpenWhatsapp)
        ...
        }
    }
}
```

Dependências implícitas

```
struct MGMChannelData {
    let type: MGMChannel
    let count: Int?
    let canOpen: Bool
}

class MGMInteractor {
    ...
    func createChannelData(invitations: Int,
                           channel: Customer.Channel) -> MGMChannelData {
        switch channel {
        case .whatsapp:
            let canOpenWhatsapp = UIApplication.shared
                                               .canOpenURL(Hypermedia.whatsApp)
            return MGMChannelData(type: .whatsapp,
                                  count: nil,
                                  canOpen: canOpenWhatsapp)
        ...
        }
    }
}
```

Dependências implícitas

```
struct MGMChannelData {
    let type: MGMChannel
    let count: Int?
    let canOpen: Bool
}

protocol URLHandler {
    func canOpenURL(_ url: URL) -> Bool
}
```

```
class MGMInteractor {
    ...
    func createChannelData(invitations: Int,
                           channel: Customer.Channel) -> MGMChannelData {
        switch channel {
        case .whatsapp:
            let canOpenWhatsapp = UIApplication.shared
                                               .canOpenURL(Hypermedia.whatsApp)
            return MGMChannelData(type: .whatsapp,
                                  count: nil,
                                  canOpen: canOpenWhatsapp)

        ...
        }
    }
}
```

Funções puras

Funções puras

- Possuí **retorno** (não é `void`)

Funções puras

- Possuí **retorno** (não é `void`)
- Resultado determinístico

Funções puras

- Possuí **retorno** (não é `void`)
- Resultado determinístico
    - Dado um **input**, retorna sempre o mesmo **resultado**

Funções puras

- Possuí **retorno** (não é `void`)
- Resultado determinístico
  - Dado um **input**, retorna sempre o mesmo **resultado**
  - Como funções matemáticas: f(**x**) = **y**

Funções puras (ou não)

Funções puras (ou não)

```
func fibonacci(n: UInt) -> UInt {
    guard n > 1 else {
        return n
    }
    return fibonacci(n: n-1) + fibonacci(n: n-2)
}
```

Funções puras (ou não)

```
func fibonacci(n: UInt) -> UInt {
    guard n > 1 else {
        return n
    }
    return fibonacci(n: n-1) + fibonacci(n: n-2)
}
```

```
func save(identities: [User]) -> Int {
    let db = Database()
    return db.save(identities)
}
```

Injeção de dependências

```swift
struct MGMChannelData {
    let type: MGMChannel
    let count: Int?
    let canOpen: Bool
}

class MGMInteractor {
    ...
    func createChannelData(invitations: Int,
                           channel: Customer.Channel) -> MGMChannelData {
        switch channel {
        case .whatsapp:
            let canOpenWhatsapp = UIApplication.shared
                                               .canOpenURL(Hypermedia.whatsApp)
            return MGMChannelData(type: .whatsapp,
                                  count: nil,
                                  canOpen: canOpenWhatsapp)

        ...
        }
    }
}
```

```swift
protocol URLHandler {
    func canOpenURL(_ url: URL) -> Bool
}
```

Injeção de dependências

```swift
struct MGMChannelData {
    let type: MGMChannel
    let count: Int?
    let canOpen: Bool
}

class MGMInteractor {
    ...
    func createChannelData(urlHandler: URLHandler,
                           invitations: Int,
                           channel: Customer.Channel) -> MGMChannelData {
        switch channel {
        case .whatsapp:
            let canOpenWhatsapp = UIApplication.shared
                                               .canOpenURL(Hypermedia.whatsApp)
            return MGMChannelData(type: .whatsapp,
                                  count: nil,
                                  canOpen: canOpenWhatsapp)
        ...
        }
    }
}
```

```swift
protocol URLHandler {
    func canOpenURL(_ url: URL) -> Bool
}
```

Injeção de dependências

```swift
struct MGMChannelData {
    let type: MGMChannel
    let count: Int?
    let canOpen: Bool
}

class MGMInteractor {
    ...
    func createChannelData(urlHandler: URLHandler,
                           invitations: Int,
                           channel: Customer.Channel) -> MGMChannelData {
        switch channel {
        case .whatsapp:
            let canOpenWhatsapp = UIApplication.shared
                                               .canOpenURL(Hypermedia.whatsApp)
            return MGMChannelData(type: .whatsapp,
                                  count: nil,
                                  canOpen: canOpenWhatsapp)
        ...
        }
    }
}
```

```swift
protocol URLHandler {
    func canOpenURL(_ url: URL) -> Bool
}
```

Injeção de dependências

```
struct MGMChannelData {
    let type: MGMChannel
    let count: Int?
    let canOpen: Bool
}

class MGMInteractor {
    ...
    func createChannelData(urlHandler: URLHandler,
                           invitations: Int,
                           channel: Customer.Channel) -> MGMChannelData {
        switch channel {
        case .whatsapp:
            let canOpenWhatsapp = urlHandler
                                              .canOpenURL(Hypermedia.whatsApp)
            return MGMChannelData(type: .whatsapp,
                                  count: nil,
                                  canOpen: canOpenWhatsapp)

        ...
        }
    }
}
```

Injeção de dependências

```swift
struct MGMChannelData {
    let type: MGMChannel
    let count: Int?
    let canOpen: Bool
}

class MGMInteractor {
    ...
    func createChannelData(urlHandler: URLHandler,
                           invitations: Int,
                           channel: Customer.Channel) -> MGMChannelData {
        switch channel {
        case .whatsapp:
            let canOpenWhatsapp = urlHandler.canOpenURL(Hypermedia.whatsApp)

            return MGMChannelData(type: .whatsapp,
                                  count: nil,
                                  canOpen: canOpenWhatsapp)
        ...
        }
    }
}
```

```swift
protocol URLHandler {
    func canOpenURL(_ url: URL) -> Bool
}
```

Testes unitários

Exemplo: Funções “puras"

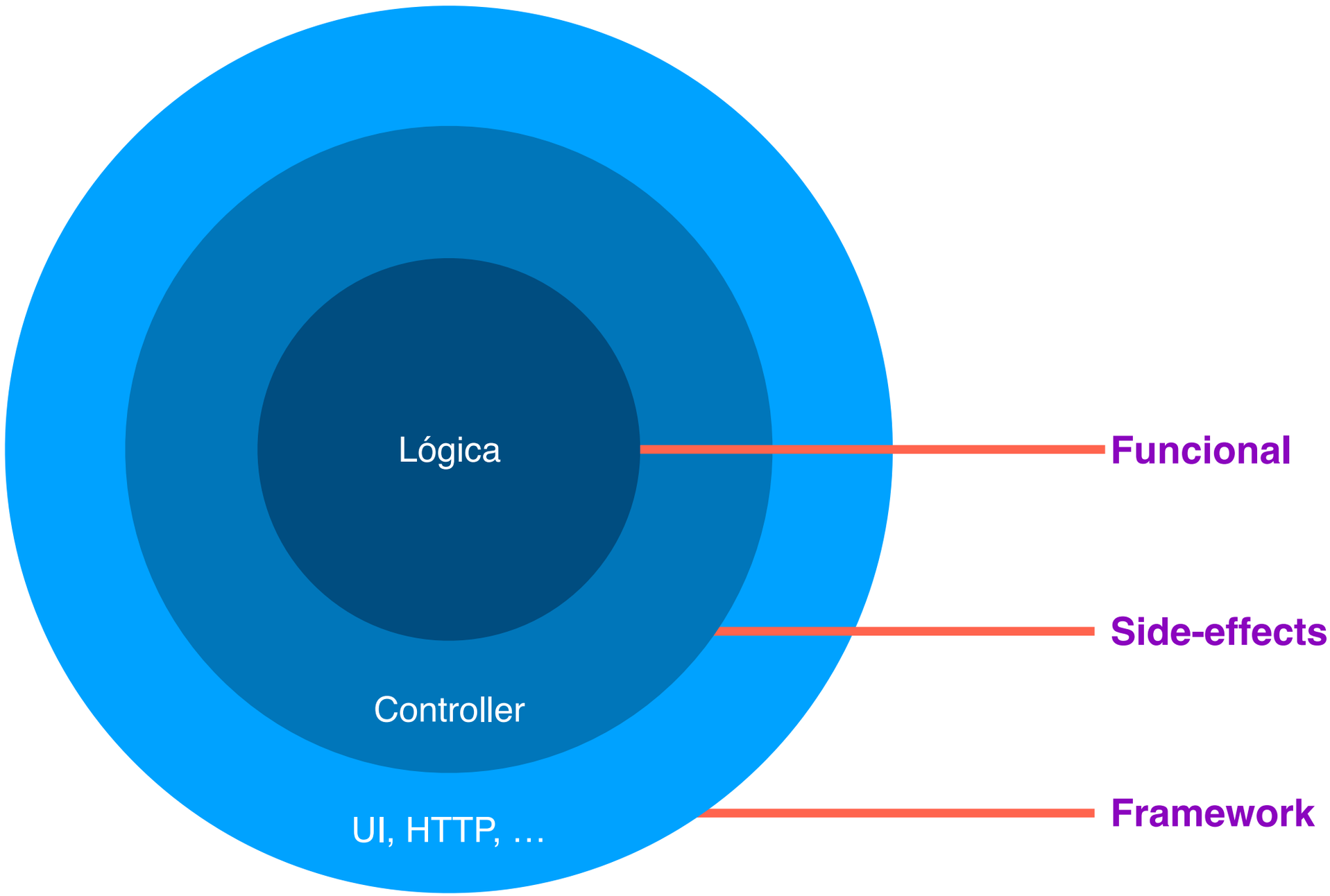
```
struct MGMChannelData {
    let type: MGMChannel
    let count: Int?
    let canOpen: Bool
}

class MGMInteractor {
    ...
    func createChannelData(urlHandler: URLHandler,
                           invitations: Int,
                           channel: Customer.Channel) -> MGMChannelData
}


func testChannelDataCreationWhatsappCannotOpen() {
    let urlHandler = URLHandlerMock(canOpenWhatsapp: false)
    let invites = Int.arbitrary.generate
    let channelData = MGMInteractor.createChannelData(invitations: invites,
                                                      channel: .whatsapp,
                                                      urlHandler: urlHandler)

    expect(channelData) == MGMChannelData(type: .whatsapp,
                                          count: nil,
                                          canOpen: false)
}
```

Testes unitários

Exemplo: Funções “puras"

```
struct MGMChannelData {
    let type: MGMChannel
    let count: Int?
    let canOpen: Bool
}

class MGMInteractor {
    ...
    func createChannelData(urlHandler: URLHandler,
                           invitations: Int,
                           channel: Customer.Channel) -> MGMChannelData
}


func testChannelDataCreationWhatsappCannotOpen() {
    let urlHandler = URLHandlerMock(canOpenWhatsapp: false)
    let invites = Int.arbitrary.generate
    let channelData = MGMInteractor.createChannelData(invitations: invites,
                                                      channel: .whatsapp,
                                                      urlHandler: urlHandler)

    expect(channelData) == MGMChannelData(type: .whatsapp,
                                          count: nil,
                                          canOpen: false)
}
```

Efeitos colaterais



Testes unitários

Exemplo: Efeitos colaterais

```
class Controller {
    init() {
        let viewController = ViewController()
        ...
        viewController.action
            .subscribe(onNext: {
                MGMManager().invite()
            })
    }
}
```

Testes unitários

Exemplo: Efeitos colaterais

```
class Controller {
    let viewController: ViewController
    let mgmManager: MGMManager

    init(mgmManager: MGMManager,
         viewController: ViewController) {
        self.mgmManager = mgmManager
        self.viewController = viewController

        viewController.action
            .subscribe(onNext: {
                mgmManager.invite()
            })
    }
}
```

Testes unitários

Exemplo: Efeitos colaterais

```
func testInviteMGM() {
    let mgmManager = StubMGMManager()
    let stubViewController = StubViewController()
    let controller = Controller(mgmManager: mgmManager,
                                viewController: stubViewController)

    //Given a simulated tap
    stubViewController.confirmSubject.onNext(())

    //Expect side-effects
    expect(mgmManager.inviteInvocationCount) == 1
}
```

Testes unitários

Exemplo: Efeitos colaterais

```
func testInviteMGM() {
    let mgmManager = StubMGMManager()
    let stubViewController = StubViewController()
    let controller = Controller(mgmManager: mgmManager,
                                viewController: stubViewController)

    //Given a simulated tap
    stubViewController.confirmSubject.onNext(())

    //Expect side-effects
    expect(mgmManager.inviteInvocationCount) == 1
}
```

Testes unitários

Exemplo: Efeitos colaterais

```
func testInviteMGM() {
    let mgmManager = StubMGMManager()
    let stubViewController = StubViewController()
    let controller = Controller(mgmManager: mgmManager,
                                viewController: stubViewController)

    //Given a simulated tap
    stubViewController.confirmSubject.onNext(())

    //Expect side-effects
    expect(mgmManager.inviteInvocationCount) == 1
}
```

Testes unitários de view

Testes unitários de view

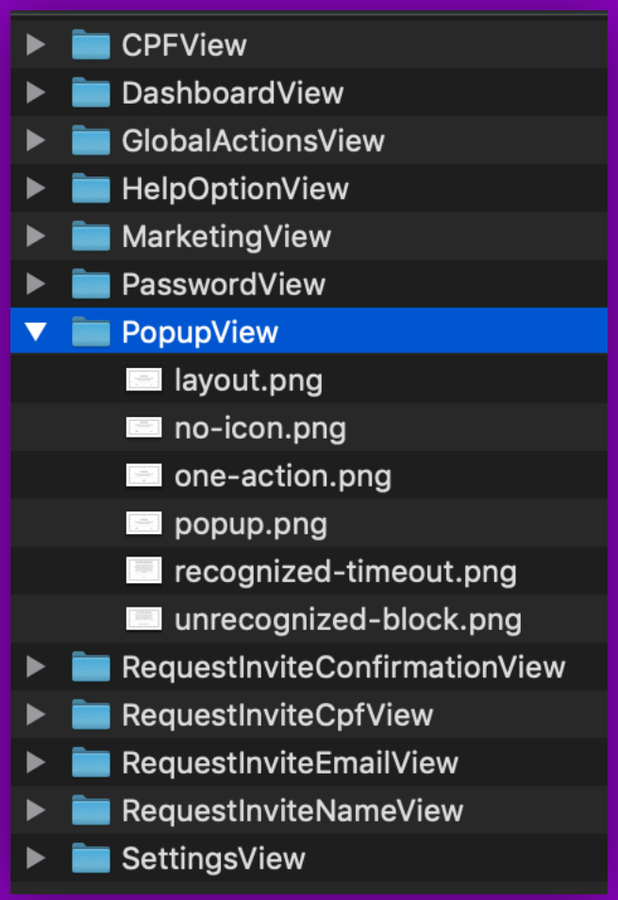
- Permite validar o layout em várias condições de uso

Testes unitários de view

- Permite validar o layout em várias condições de uso
  - Uso de screenshots

Testes unitários de view

- Permite validar o layout em várias condições de uso
  - Uso de screenshots



Testes unitários de view

- Permite validar o layout em várias condições de uso
    - Uso de screenshots
- Facilita a verificação com o Design

Testes unitários de view

- Permite validar o layout em várias condições de uso
  - Uso de screenshots
  - Facilita a verificação com o Design
- Evita modificações não intencionais no layout

Testes unitários de view

- Permite validar o layout em várias condições de uso
  - Uso de screenshots
  - Facilita a verificação com o Design
- Evita modificações não intencionais no layout
- Permite validar a acessibilidade das telas

Testes unitários de view

Testes unitários de view

- Como testar layout isolado?

Testes unitários de view

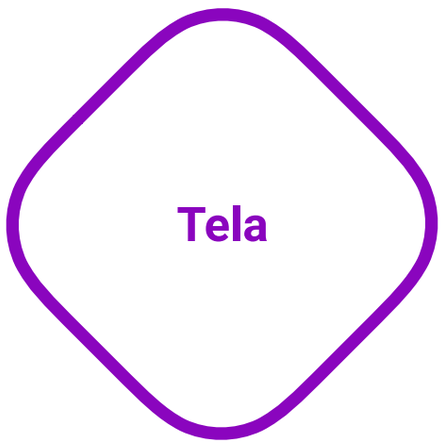
- Como testar layout isolado?
    - Dividir responsabilidades

Testes unitários de view

- Como testar layout isolado?
    - Dividir responsabilidades
    - Desacoplar comportamentos

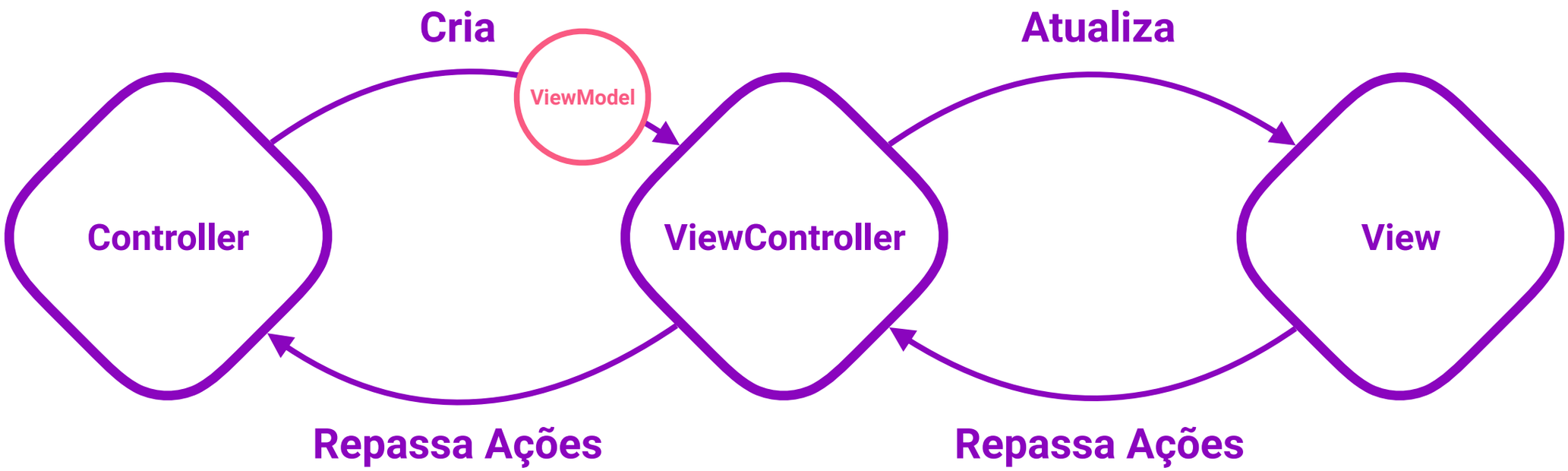
Testes unitários de view

- Como testar layout isolado?
    - Dividir responsabilidades
    - Desacoplar comportamentos



Testes unitários de view

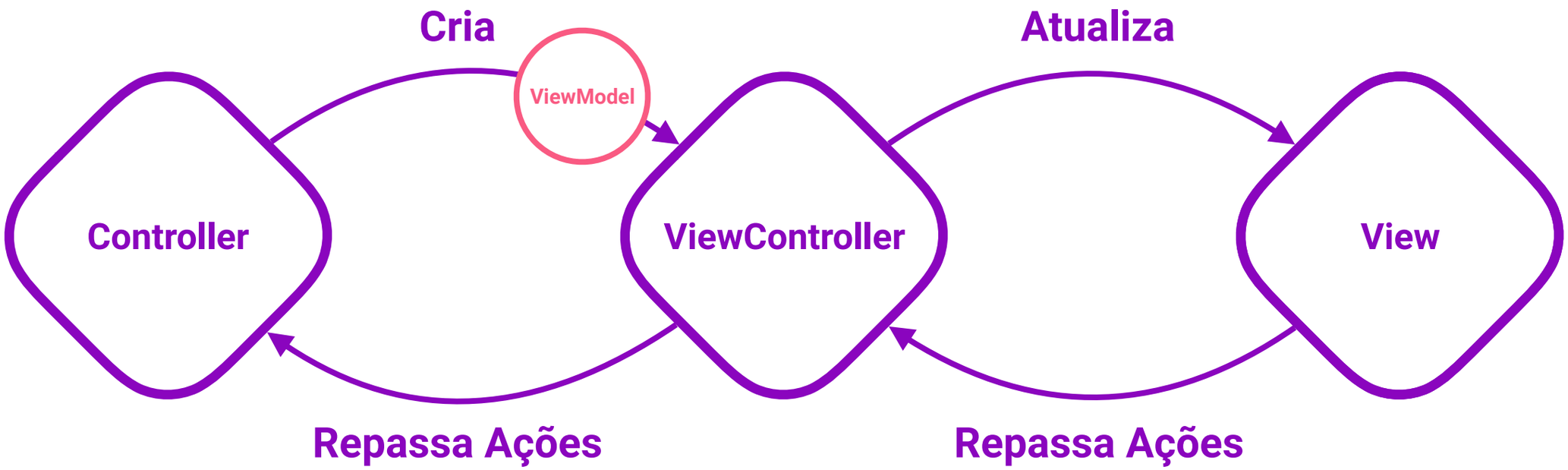
- Como testar layout isolado?
    - Dividir responsabilidades
    - Desacoplar comportamentos



Medium: Building Nubank - iOS App Architecture

Testes unitários de view

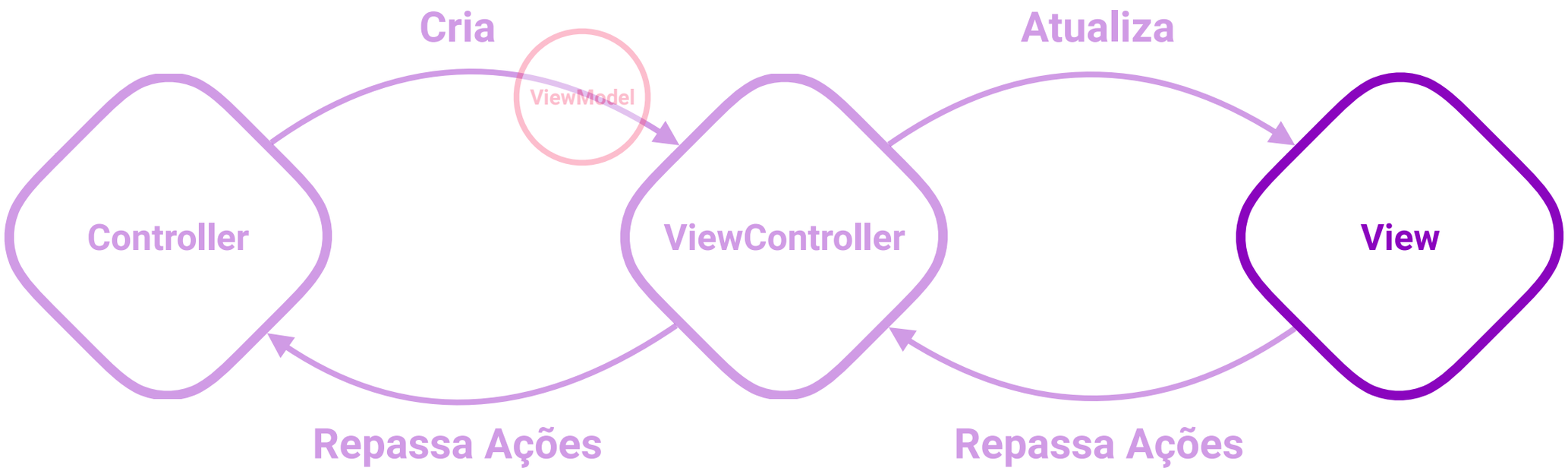
- Como testar layout isolado?
    - Dividir responsabilidades
    - Desacoplar comportamentos



Medium: Building Nubank - iOS App Architecture

Testes unitários de view

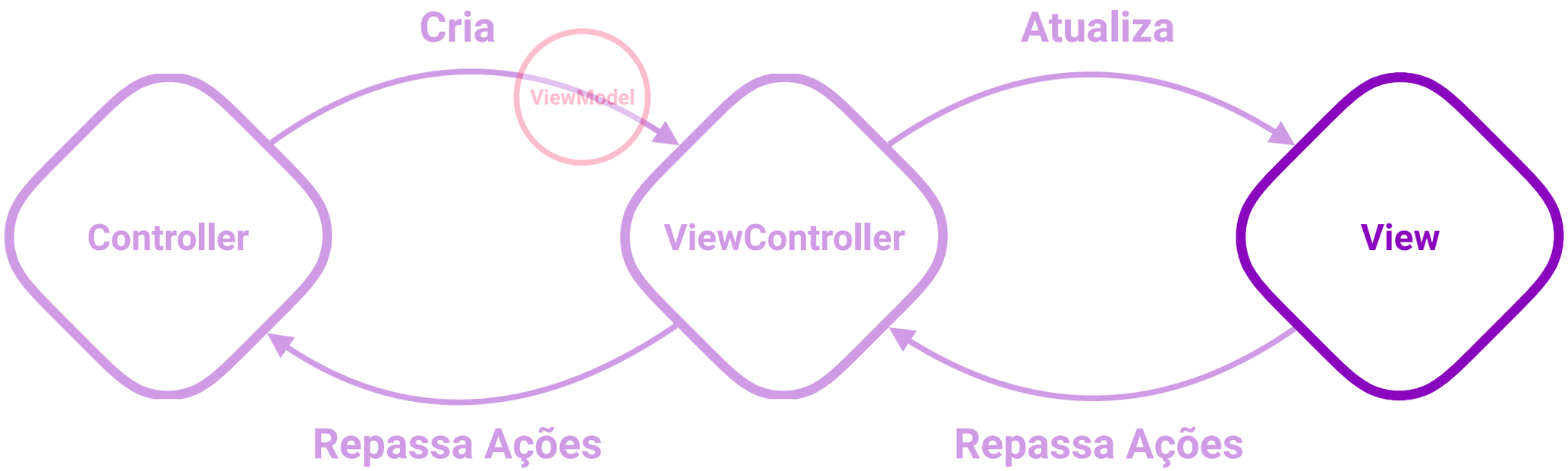
- Como testar layout isolado?
    - Dividir responsabilidades
    - Desacoplar comportamentos



Medium: Building Nubank - iOS App Architecture

Testes unitários de view

- Como testar layout isolado?
    - Dividir responsabilidades
    - Desacoplar comportamentos



Medium: Building Nubank - iOS App Architecture

Testes unitários de view

Exercitar todos os **tamanhos** da tela ou componente

* Telas não necessariamente verdadeiras

Testes unitários de view

Exercitar todos os **tamanhos** da tela ou componente

4S

5/5S/5C/SE

6/6S/7/8

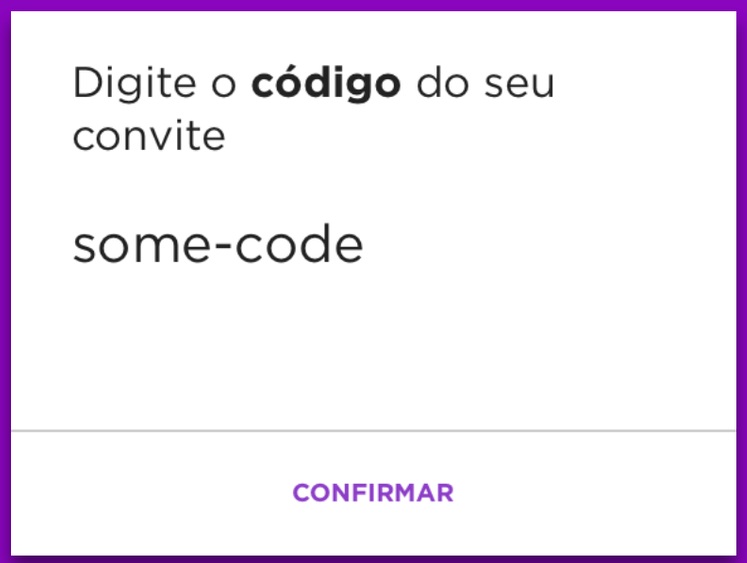
6P/7P/8P

X

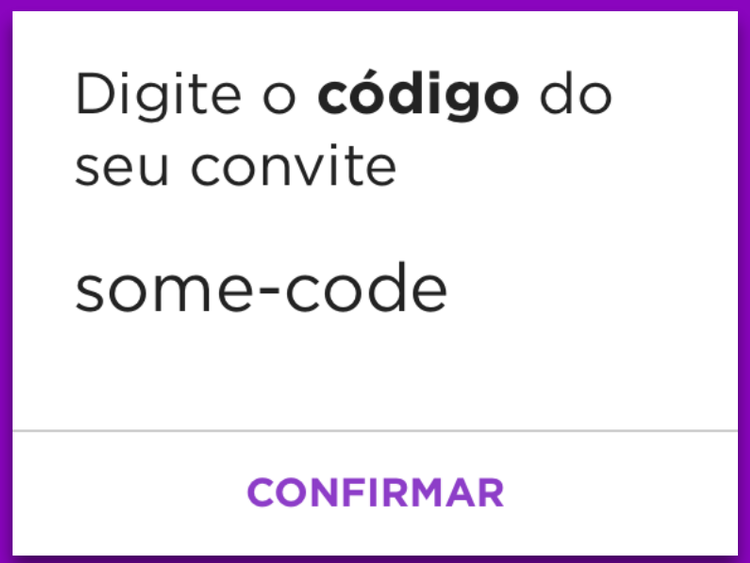
* Telas não necessariamente verdadeiras

Testes unitários de view

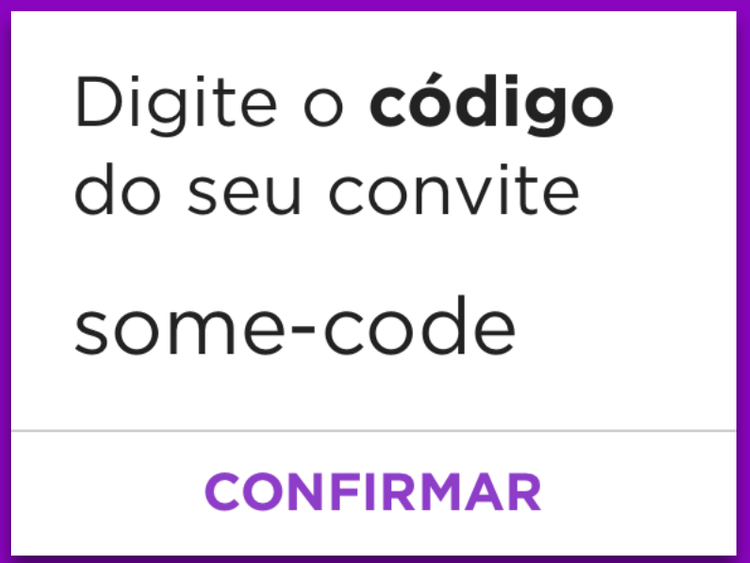
Exercitar todos os **tamanhos de fonte** na tela ou componente



CategoryXS



CategoryXXXL



CategoryAccessibilityXXXL

* Telas não necessariamente verdadeiras

Testes unitários de view

Exercitar todos os **estados** da tela ou componente

* Telas não necessariamente verdadeiras

Testes unitários de view

Exercitar todos os **estados** da tela ou componente

Secure code

Realmente enviar R$ 15,00 para Frutas Frescas Ltda?

NÃO

Secure code

Realmente enviar R$ 15,00 para Frutas Frescas Ltda?

NÃO | SIM

Seu cartão foi bloqueado

Achamos que alguém pode ter tido acesso aos dados dele. Efetue o cancelamento do seu cartão atual e solicite um novo.

Emitir novo cartão

Seu cartão foi desbloqueado

Como você confirmou que realmente tentou realizar aquela transação, já desbloqueamos o seu cartão e está tudo certo para você continuar usando seu roxinho.

Ok!

Secure code

Realmente enviar R$ 15,00 para Frutas Frescas Ltda?

NÃO | SIM

Secure code

Realmente enviar R$ 15,00 para Frutas Frescas Ltda?

NÃO | SIM

* Telas não necessariamente verdadeiras

Testes unitários de view

Exemplo

```
func testLayout() throws {
    let permutations = [
        ("popup", PopupViewModel(popup: somePopup)),
        ("one-action", PopupViewModel(popup: oneActionPopup)),
        ("no-icon", PopupViewModel(popup: noIconPopup)),
        ("recognized-timeout", .recognizedTimeout(url: .empty())),
        ("unrecognized-block", .unrecognizedBlock(url: .empty()))
    ]

    try permutations.forEach { identifier, viewModel in
        let viewController = PopupViewController()
        viewController.bind(viewModel)

        try assertView(viewController.view,
                       width: 350,
                       identifier: identifier,
                       screenshotService: service)
    }
}
```

Testes unitários de view

Exemplo

```
func testLayout() throws {
    let permutations = [
        ("popup", PopupViewModel(popup: somePopup)),
        ("one-action", PopupViewModel(popup: oneActionPopup)),
        ("no-icon", PopupViewModel(popup: noIconPopup)),
        ("recognized-timeout", .recognizedTimeout(url: .empty())),
        ("unrecognized-block", .unrecognizedBlock(url: .empty()))
    ]

    try permutations.forEach { identifier, viewModel in
        let viewController = PopupViewController()
        viewController.bind(viewModel)

        try assertView(viewController.view,
                       width: 350,
                       identifier: identifier,
                       screenshotService: service)
    }
}
```

Testes unitários de view

Exemplo

```
func testLayout() throws {
    let permutations = [
        ("popup", PopupViewModel(popup: somePopup)),
        ("one-action", PopupViewModel(popup: oneActionPopup)),
        ("no-icon", PopupViewModel(popup: noIconPopup)),
        ("recognized-timeout", .recognizedTimeout(url: .empty())),
        ("unrecognized-block", .unrecognizedBlock(url: .empty()))
    ]

    try permutations.forEach { identifier, viewModel in
        let viewController = PopupViewController()
        viewController.bind(viewModel)

        try assertView(viewController.view,
                       width: 350,
                       identifier: identifier,
                       screenshotService: service)
    }
}
```

Testes unitários de view

Exemplo

```
func testLayout() throws {
    let permutations = [
        ("popup", PopupViewModel(popup: somePopup)),
        ("one-action", PopupViewModel(popup: oneActionPopup)),
        ("no-icon", PopupViewModel(popup: noIconPopup)),
        ("recognized-timeout", .recognizedTimeout(url: .empty())),
        ("unrecognized-block", .unrecognizedBlock(url: .empty()))
    ]

    try permutations.forEach { identifier, viewModel in
        let viewController = PopupViewController()
        viewController.bind(viewModel)

        try assertView(viewController.view,
                       width: 350,
                       identifier: identifier,
                       screenshotService: service)
    }
}
```

Testes de integração

Testes de integração

- Foco no funcionamento dos componentes *em conjunto*

Testes de integração

- Foco no funcionamento dos componentes *em conjunto*
- Fluxos reais do app com poucas modificações

Testes de integração

- Foco no funcionamento dos componentes *em conjunto*
- Fluxos reais do app com poucas modificações
- Requests *mockados* internamente no teste

Testes de integração

Testes de integração

KIF

Input do Usuário

Testes de integração

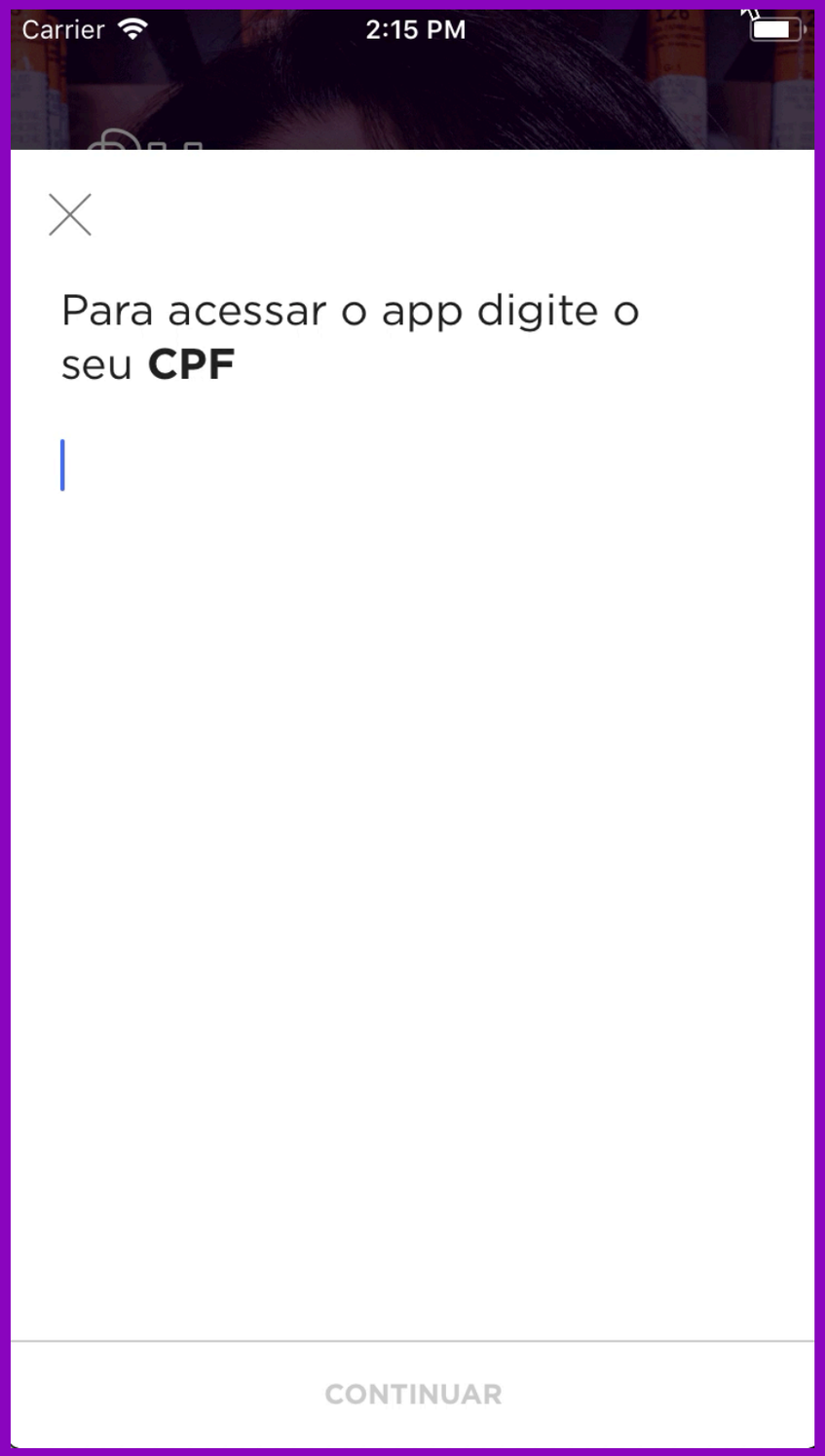
KIF

Input do Usuário

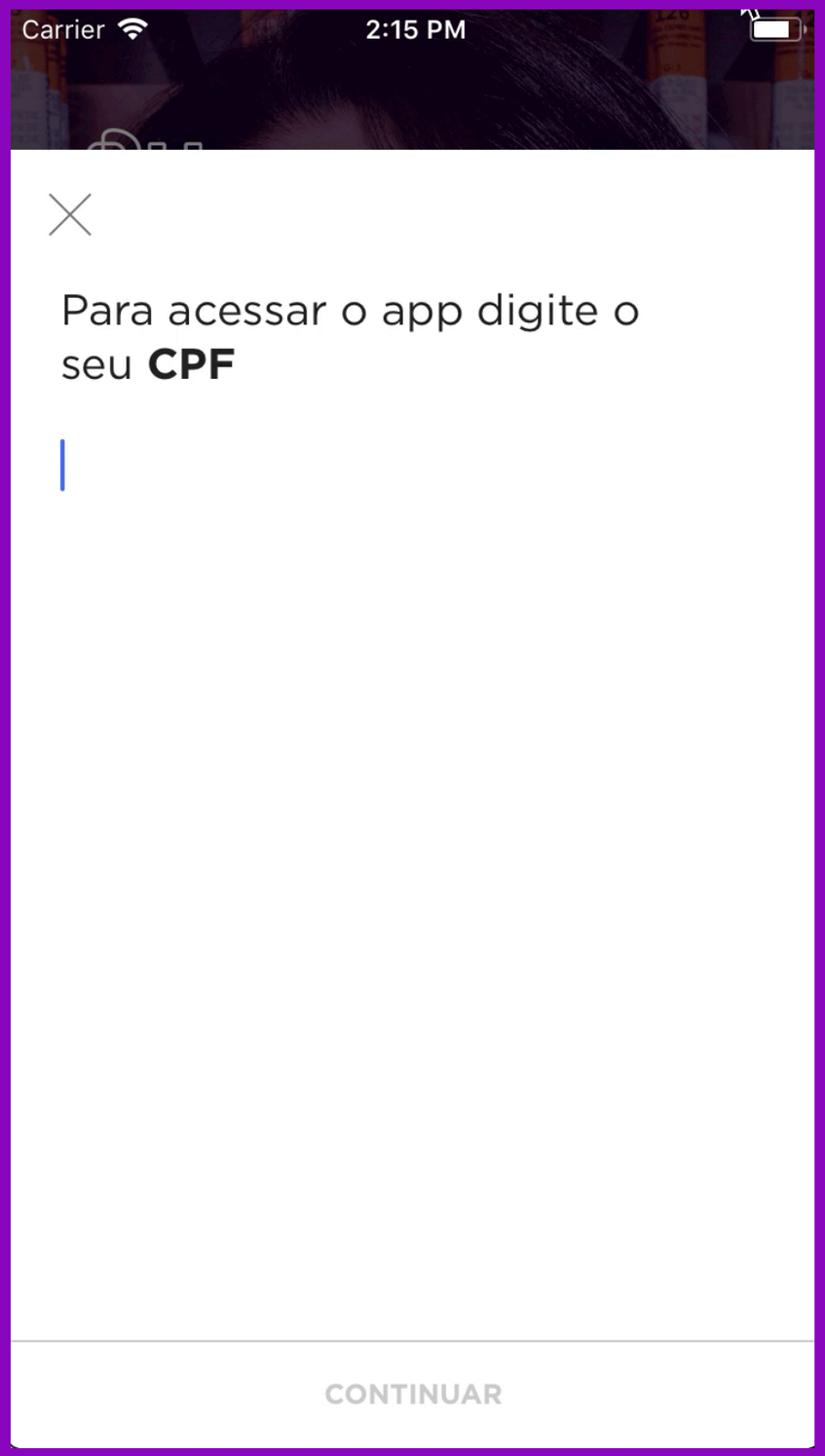
OHHTTPStubs

Mocks de requests

Testes de integração



Testes de integração



Testes de integração

Exemplo

```
func login() {
    tester().tapView(withAccessibilityLabel: "LOGIN")

    tester().waitForSoftwareKeyboard()
    tester().enterText(intoCurrentFirstResponder: validCPF)
    tester().tapView(withAccessibilityLabel: "CONTINUAR")

    tester().enterText(intoCurrentFirstResponder: validPassword)
    tester().tapView(withAccessibilityLabel: "ENTRAR")
}


func testReorderFlow() throws {
    tester().swipeView(withAccessibilityIdentifier: "global-actions",
                       inDirection: .left)

    tester().moveRow(at: IndexPath(row: 0, section: 0),
                     to: IndexPath(row: 1, section: 0),
                     in: tester().firstTableView())

    tester().waitForAnimationsToFinish()

    tester().tapView(withAccessibilityLabel: "Voltar")
    tester().swipeView(withAccessibilityIdentifier: "global-actions",
                       inDirection: .right)
    tester().waitForAnimationsToFinish()
}
```

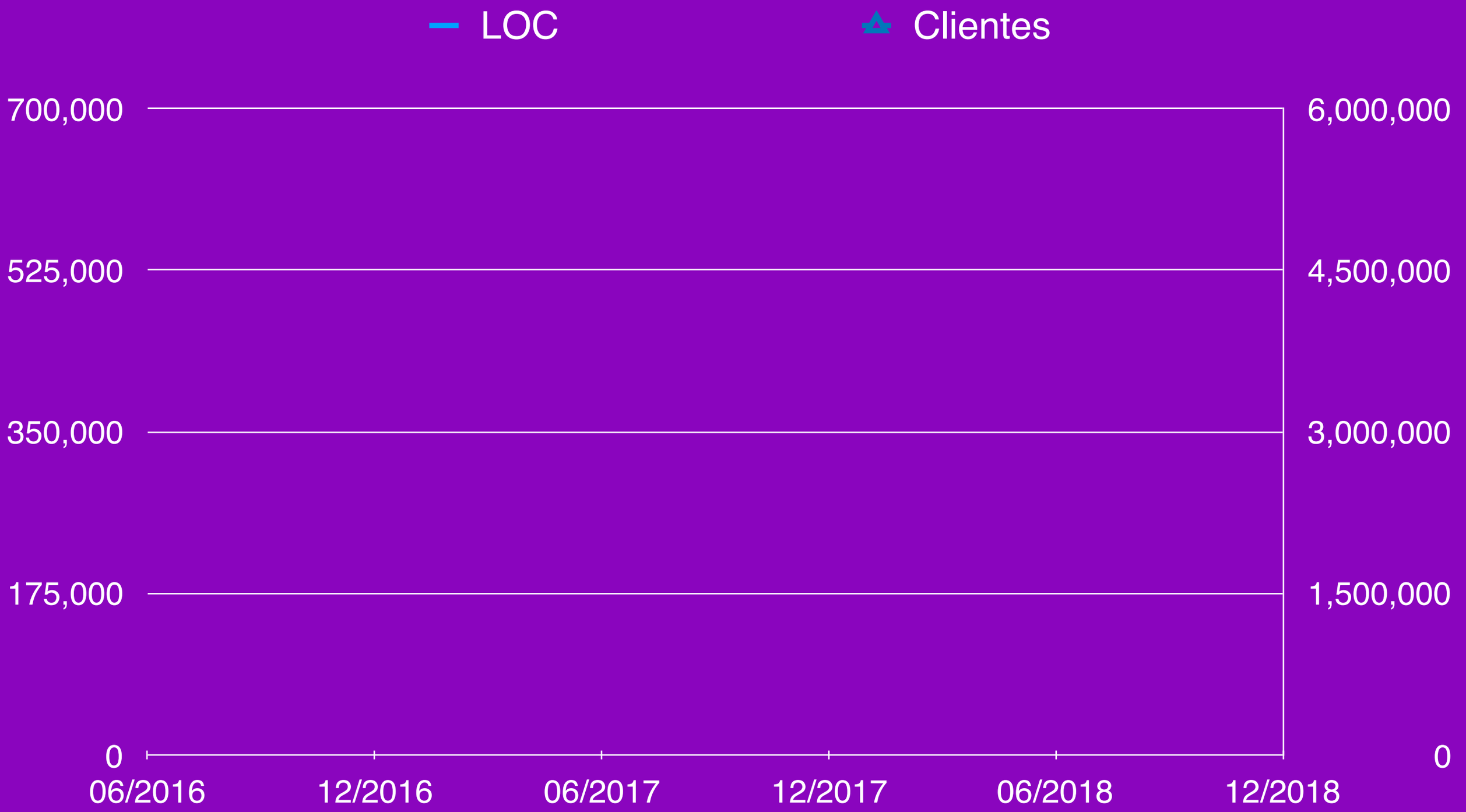
Testes de integração

Exemplo

```
func login() {
    tester().tapView(withAccessibilityLabel: "LOGIN")

    tester().waitForSoftwareKeyboard()
    tester().enterText(intoCurrentFirstResponder: validCPF)
    tester().tapView(withAccessibilityLabel: "CONTINUAR")

    tester().enterText(intoCurrentFirstResponder: validPassword)
    tester().tapView(withAccessibilityLabel: "ENTRAR")
}


func testReorderFlow() throws {
    tester().swipeView(withAccessibilityIdentifier: "global-actions",
                       inDirection: .left)

    tester().moveRow(at: IndexPath(row: 0, section: 0),
                     to: IndexPath(row: 1, section: 0),
                     in: tester().firstTableView())

    tester().waitForAnimationsToFinish()

    tester().tapView(withAccessibilityLabel: "Voltar")
    tester().swipeView(withAccessibilityIdentifier: "global-actions",
                       inDirection: .right)
    tester().waitForAnimationsToFinish()
}
```

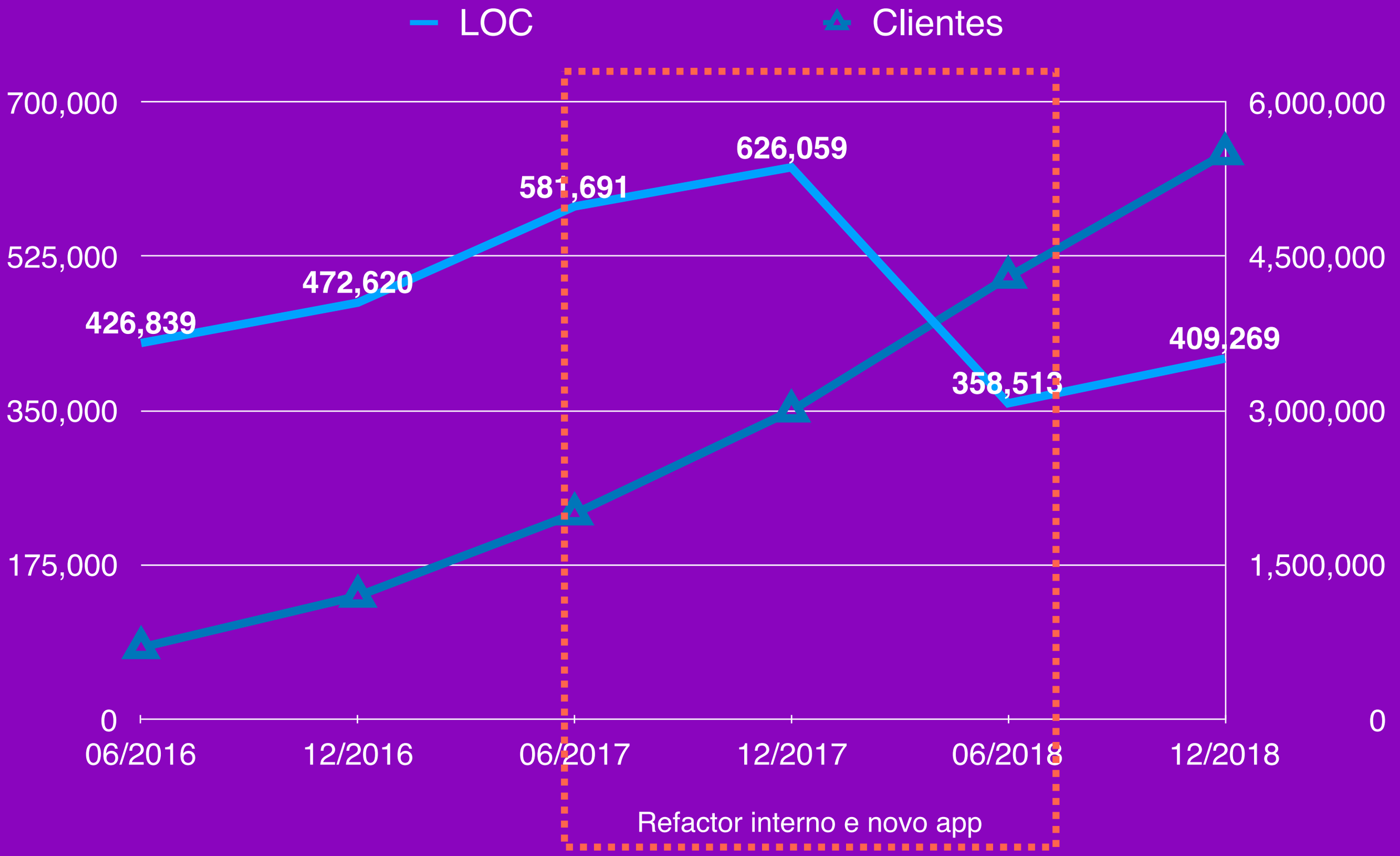
Estatísticas no App Nativo iOS

Estatísticas no App Nativo iOS



Estatísticas no App Nativo iOS

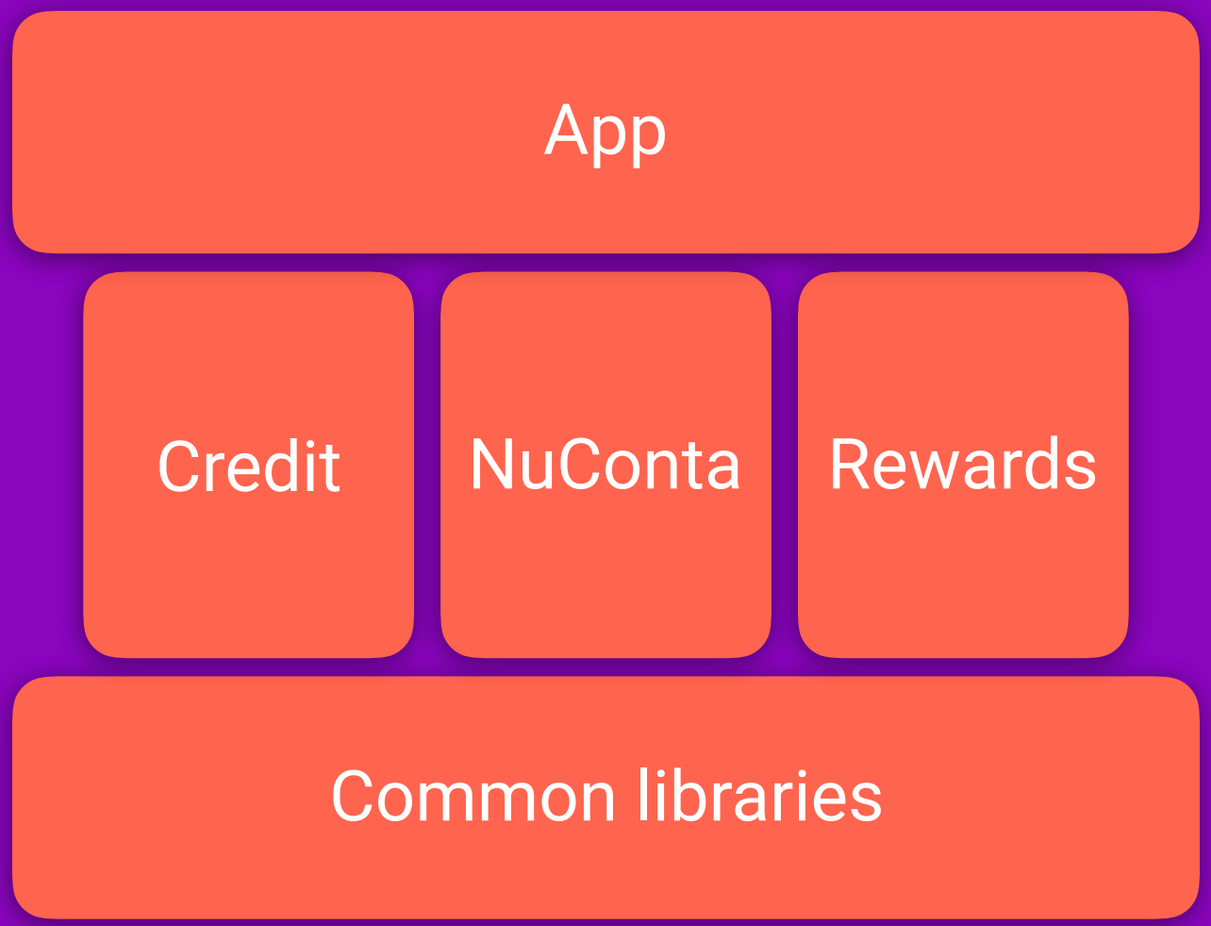
Estatísticas no App Nativo iOS



Estatísticas no App Nativo iOS

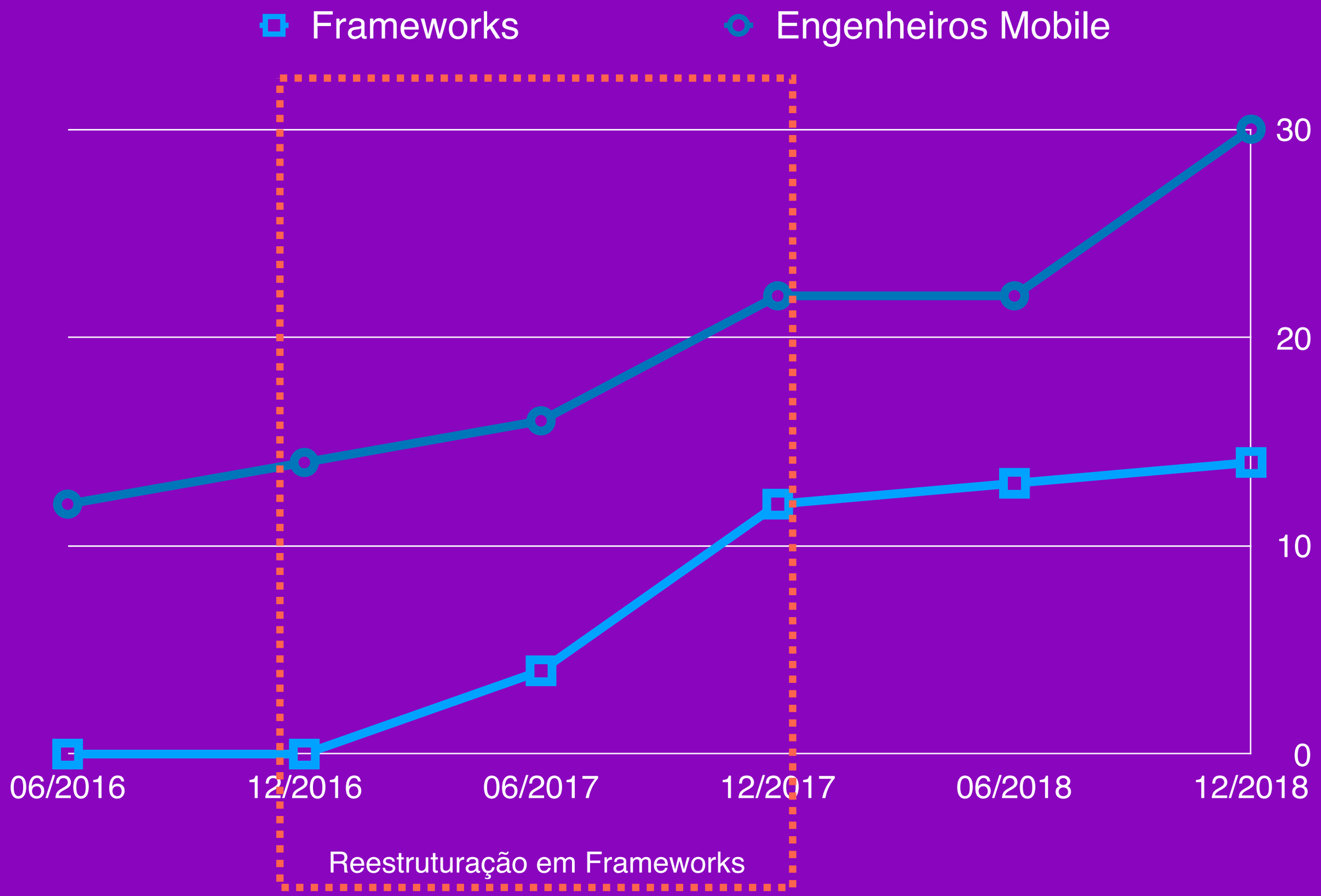
Estrutura do App

Estrutura do App



Estatísticas

Estatísticas



Obrigado!

Victor Maraccini

@vmaraccini | victor.maraccini@nubank.com.br

Francesco Perrotti-Garcia

@fpg1503 | francesco.garcia@nubank.com.br

Obrigado!

sou.nu/jobs-at-nubank

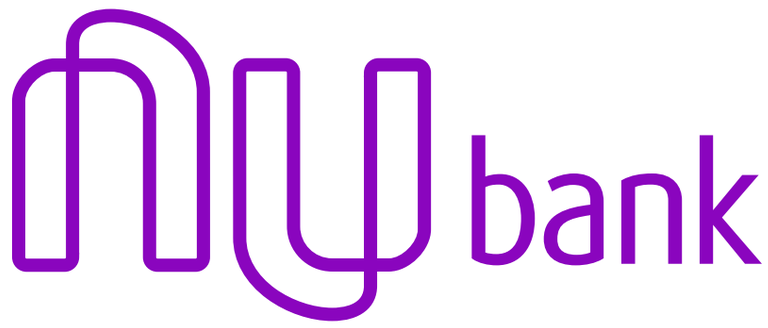
TestFlight (Beta)

Victor Maraccini

@vmaraccini | victor.maraccini@nubank.com.br

Francesco Perrotti-Garcia

@fpg1503 | francesco.garcia@nubank.com.br


bank