Type-Driven Design with Swift

Alex Ozun 👋 🇺🇦
Staff iOS Engineer

AMERICAN EXPRESS

swiftology.io

Code Review

Version Control

Pair Programming

Documentation

Static Analisys

Type System

Automated Testing

Benchmarking

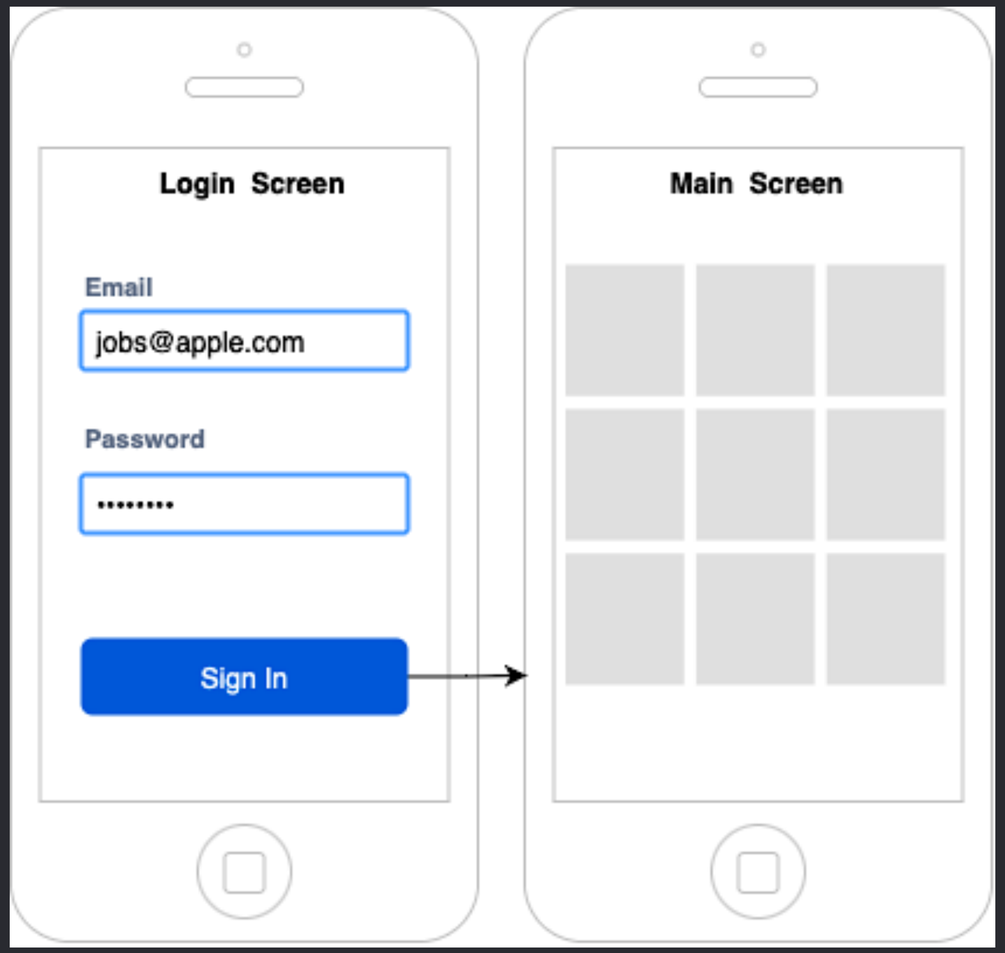
Manual Testing

Logging & Tracing

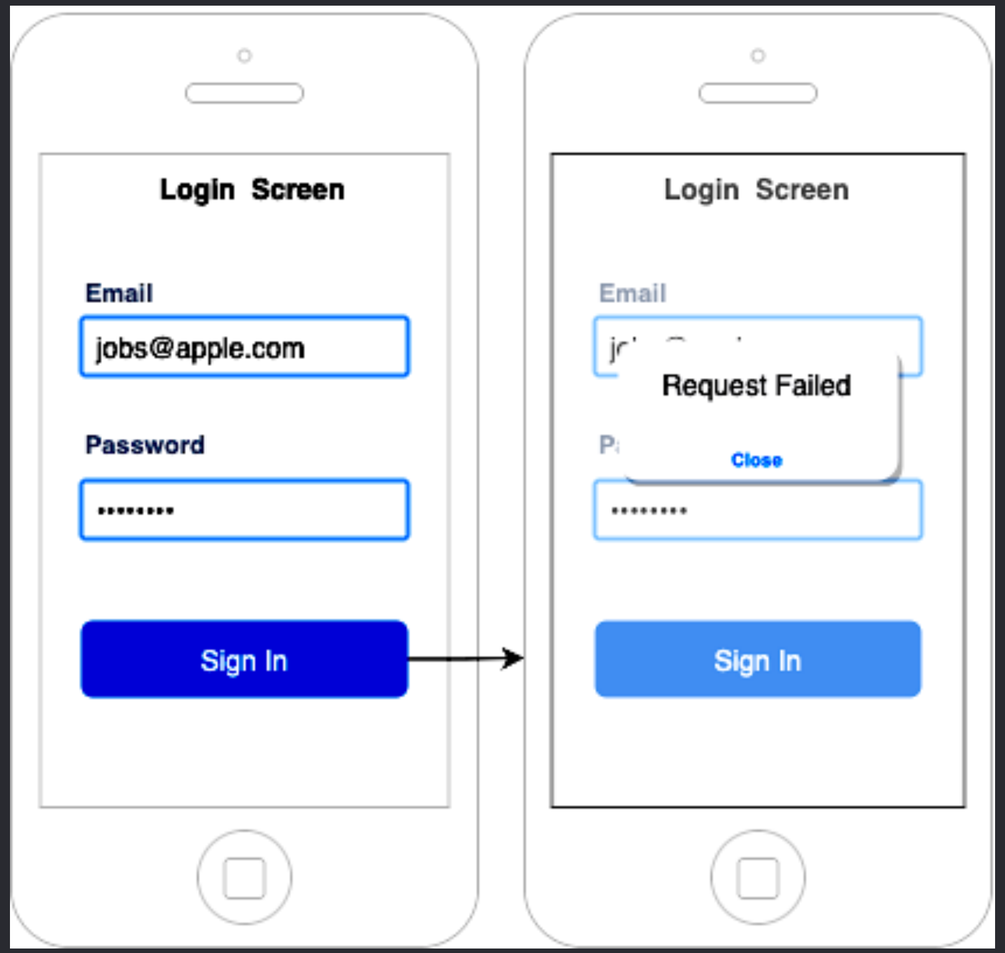
Topics

- ✦Parse, Don’t Validate.
- ✦Type, Define, Refine.
- ✦Type Safety Back and Forth.
- ✦Falling into the Pit of Success.

Example: Sign In Flow



Success



Failure

Requirements

SCENARIO: User successfully Signs into application.

GIVEN:

- User has entered email and password
- The email has valid format
- The password is at least 8 characters long

WHEN:

- User taps on the "Sign In" button
- The app sends a Sign in request
- The Sign in request is successful

THEN:

- The app navigates to the Main screen

```swift
let emailTextField: UITextField
let passwordTextField: UITextField

@objc func signInButtonTapped(_ button: UIButton) {
  // guard emailTextField.text.matches("##email_regex##")
  //  && passwordTextField.text.count > 8 else {
  //  return
  // }
🤔signIn(email: emailTextField.text, password: passwordTextField.text)
}
func signIn(email: String, password: String) {
  AuthService.signIn(email: email, password: password) { (isSuccess: Bool) in
    if isSuccess {
        showMainScreen()
    } else {
        showErrorMessage()
    }
  }
}
```


```swift
func showMainScreen() {
  let vc = MainScreenViewController()
  navigationController.push(vc)
}
```

```
let emailTextField: UITextField
let passwordTextField: UITextField

@objc func signInButtonTapped(_ button: UIButton) {
    showMainScreen()
    signIn(email: emailTextField.text, password: passwordTextField.text)
}
func signIn(email: String, password: String) {
    AuthService.signIn(email: email, password: password) { (isSuccess: Bool) in
        if isSuccess {
            showMainScreen()
        } else {
            showErrorMessage()
        }
    }
}
```

SPRAY AND PRAY VALIDATION

```
if emailTextField.text.matc
//  ...Send Sign in R...

if passwordTextField.text.count > 8
//  ...Show Main Screen...

if isSuccess
//  ...Show Error Message...
```

🙏 that creds have been validated before you send Sign in request

🙏 that the User has Signed in before you show the Main Screen

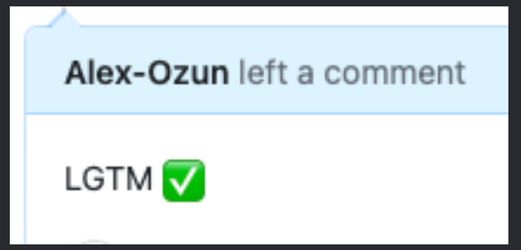
Linters can be silenced

```
// swiftlint:disable force_unwrapping
let url = URL(string: string)!
```

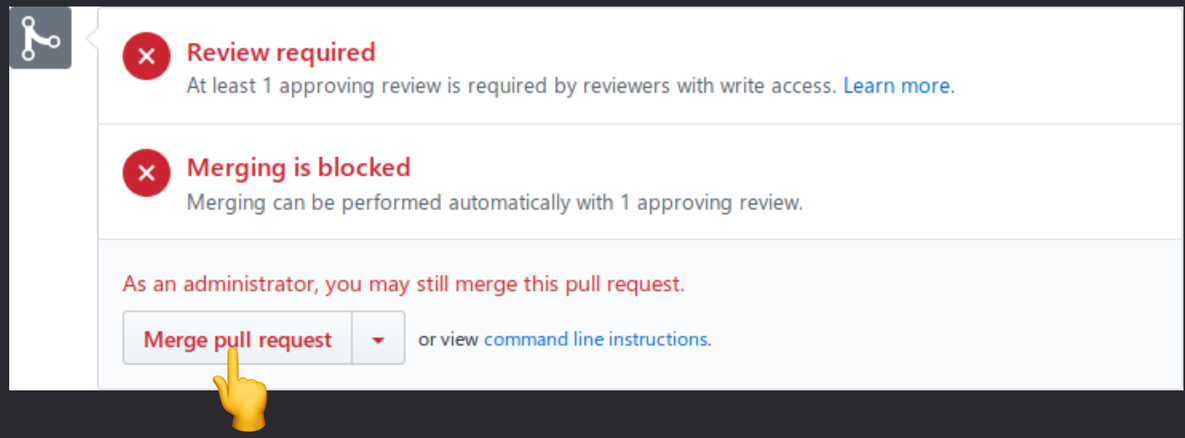
Tests can be disabled

```
  func DISABLED_testNonEmptySetWithTrivialValue() {
    let xs = NonEmptySet<TrivialHashable>(.init(value: 1), .init(value: 2))
    let ys = NonEmptySet<TrivialHashable>(.init(value: 2), .init(value: 1))

    XCTAssertEqual(xs, ys)
  }

//  func testMutableCollectionWithArraySlice() {
//    let numbers = Array(1...10)
//    var xs = NonEmpty(rawValue: numbers[5...])!
//    xs[6] = 43
//    XCTAssertEqual(43, xs[6])
//  }
```

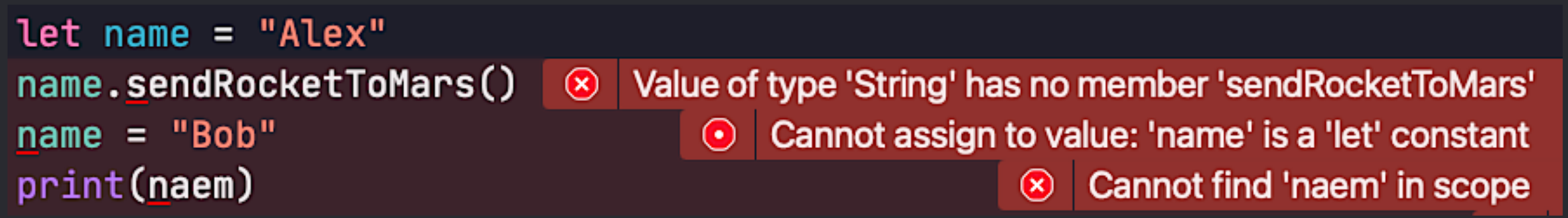
Pull Requests can be rubber-



or force-merged



Compilation Error
is the only *real* guarantee that
undesired change doesn’t get shipped


let name = "Alex"
name.sendRocketToMars()
Value of type 'String' has no member 'sendRocketToMars'
name = "Bob"
Cannot assign to value: 'name' is a 'let' constant
print(naem)
Cannot find 'naem' in scope

```
@objc func signInButtonTapped(_ button: UIButton) {
    signIn(email: emailTextField.text, password: passwordTextField.text)

    guard emailTextField.text.matches("##email_regex##")
        && passwordTextField.text.count > 8 else {
        return
    }
    …
}

func signIn(email: String, password: String) {

    showMainScreen()

    AuthService.signIn(email: email, password: password) { (isSuccess: Bool) in
        …
    }
}
```

Cannot Sign In before validating email and password

Cannot show Main Screen before Signing In

Type, Define, Refine.

Edwin Brady, Type-Driven Development with Idris

Revisiting Requirements

Step 1: Create a Type

```
// GIVEN:
// 1. User has entered email and password
// String and String

// 2. The email is valid
struct Email {}

// 3. The password is at least 8 characters long
struct Password {}

// WHEN
// 4. User taps on the "Sign In" button
func signInButtonTapped(_ button: UIButton) {}

// 5. The app sends an authentication request
func signIn(email: Email, password: Password) {}

// 6. The authentication request is successful | (OR failed)
class AuthClient {
  static func signIn(email: String, password: String, @escaping completion: (Bool) -> Void) {}
}

// THEN
// 7. The app navigates to the Main screen | (OR presents error)
func showMainScreen() {}
func showErrorMessage() {}
```

```
@objc func signInButtonTapped(_ button: UIButton) {
  guard emailTextField.text.matches("##email_regex##")
    && passwordTextField.text.count > 8 else {
    return
  }

  signIn(email: emailTextField.text, password: passwordTextField.text)
}
```

Cannot convert value of type 'String' to expected argument type 'Email'

Cannot convert value of type 'String' to expected argument type 'Password'

```
func signIn(email: Email, password: Password) {
  AuthService.signIn(email: email, password: password) { … }
}
```

Cannot convert value of type 'Email' to expected argument type 'String'

Cannot convert value of type 'Password' to expected argument type 'String'

Step 2: Define Implementations

```
@objc func signInButtonTapped(_ button: UIButton) {
  signIn(email: ???, password: ???)
}
```

```
func signIn(email: Email, password: Password) {
  AuthService.signIn(email: ???, password: ???) { … }
}
```

```
struct Email {
  let wrappedString: String
}
```

```
struct Password {
  let wrappedString: String
}
```

```swift
@objc func signInButtonTapped(_ button: UIButton) {
  signIn(email: ???, password: ???)
}

func signIn(email: Email, password: Password) {
  AuthService.signIn(
    email: email.wrappedString,
    password: password.wrappedString
  ) { … }
}

struct Email {
  let wrappedString: String
}

struct Password {
  let wrappedString: String
}
```

Parse, Don’t Validate

Alexis King

Validation:

Simply checks if data is valid, True or False.

Then discards this information.

```
func isValidEmail(string: String) → Bool
if isValidEmail("jobs@apple.com") {

}
```

Parsing:

Transforms less structured data into a more structured data, or fails.

```
if let email = Email("jobs@apple.com") {
  email
}
```

The difference between validation and parsing is in how information is preserved.

```swift
@objc func signInButtonTapped(_ button: UIButton) {
  signIn(email: ???, password: ???)
}

func signIn(email: Email, password: Password) {
  AuthService.signIn(
    email: email.wrappedString,
    password: password.wrappedString
  ) { … }
}

struct Email {
  let wrappedString: String

  init?(_ rawString: String) {
    guard rawString.matches("##regex##")
     else { return nil }

    wrappedString = rawString
  }
}

struct Password {
  let wrappedString: String

  init?(_ rawString: String) {
    guard rawString.count > 8
     else { return nil }

    wrappedString = rawString
  }
}
```

```swift
@objc func signInButtonTapped(_ button: UIButton) {
  guard let email = Email(emailTextField.text),
    let password = Password(passwordTextField.text) else { return }

  signIn(email: email, password: password)
}

func signIn(email: Email, password: Password) {
  AuthService.signIn(
    email: email.wrappedString,
    password: password.wrappedString
  ) { … }
}

struct Email {
  let wrappedString: String
  init?(_ rawString: String) {
    // parsing
  }
}

struct Password {
  let wrappedString: String
  init?(_ rawString: String) {
    // parsing
  }
}
```

```
@objc func signInButtonTapped(_ button: UIButton) {
  guard let email = Email(emailTextField.text),
    let password = Password(passwordTextField.text) else { return }

  signIn(email: email, password: password)
}

func signIn(email: Email, password: Password) {
  AuthService.signIn(
    email: email.wrappedString,
    password: password.wrappedString
  ) { … }
}
```

```
class AuthClient {
  static func signIn(email: String, password: String, completion: …) {}
}
```

```
@objc func signInButtonTapped(_ button: UIButton) {
  guard let email = Email(emailTextField.text),
    let password = Password(passwordTextField.text) else { return }

  signIn(email: email, password: password)
}

func signIn(email: Email, password: Password) {
  AuthService.signIn(
    email: email.wrappedString,
    password: password.wrappedString
  ) { … }
}
```

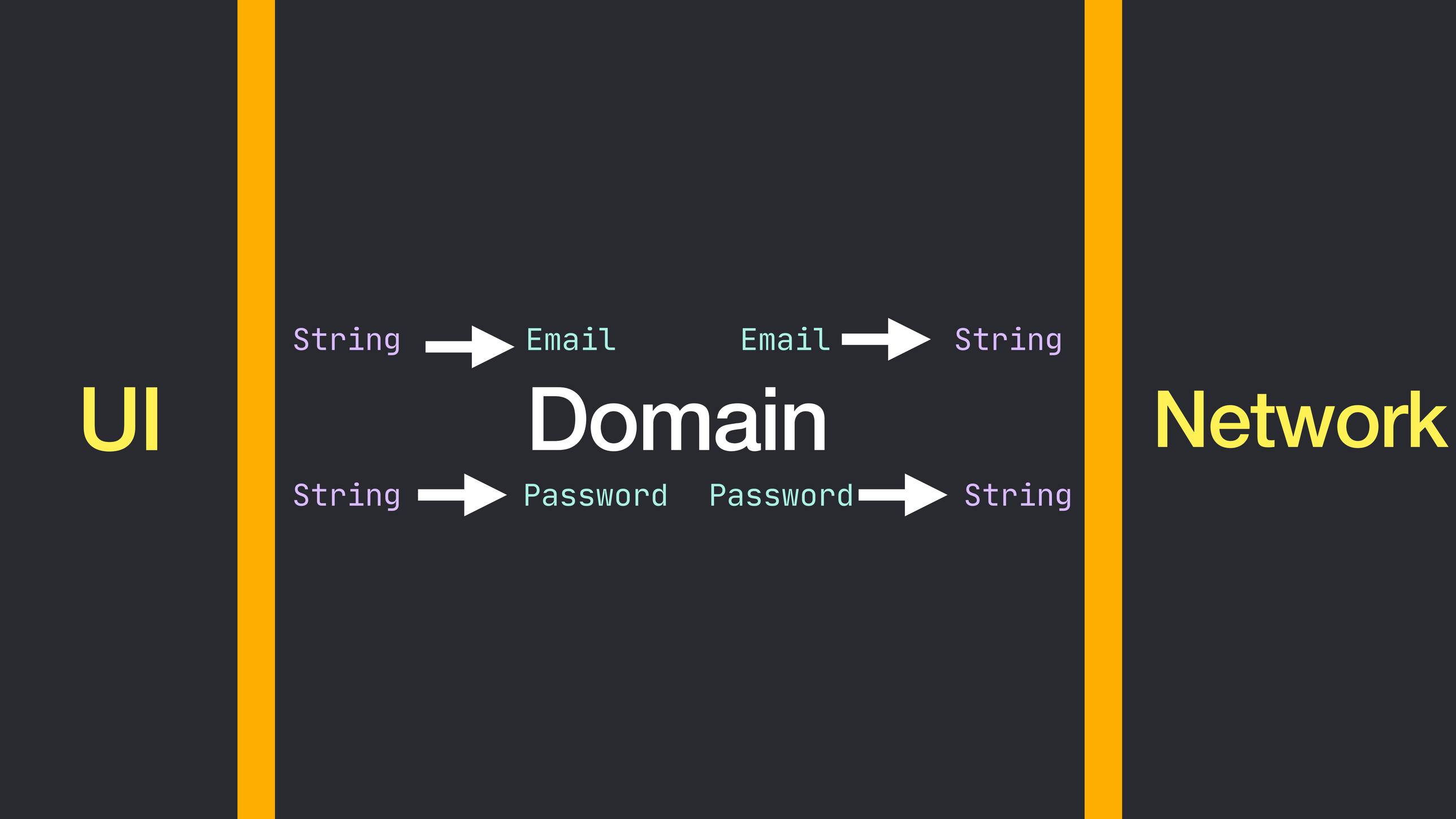
Step 3: Refine Types

```
class AuthClient {
  static func signIn(email: Email, password: Password, completion: …) {}
}
```

```swift
@objc func signInButtonTapped(_ button: UIButton) {
  guard let email = Email(emailTextField.text),
    let password = Password(passwordTextField.text) else { return }

  signIn(email: email, password: password)
}

func signIn(email: Email, password: Password) {
  AuthService.signIn(email: email, password: password) { … }
}
```

```swift
class AuthClient {
  static func signIn(email: Email, password: Password, completion: …) {}
}
```

```
@objc func signInButtonTapped(_ button: UIButton) {
  guard let email = Email(emailTextField.text),
    let password = Password(passwordTextField.text) else { return }

  signIn(email: email, password: password)
}

func signIn(email: Email, password: Password) {
  AuthService.signIn(email: email, password: password) { … }
}

class AuthClient {
  static func signIn(email: Email, password: Password, completion: …) {
    let body = [
      "email": email.wrappedString,
      "password": password.wrappedString
    ]
    // JSON encoding and URLSession boilerplate
  }
}
```


UI
String → Email
Email → String
Domain
String → Password
Password → String
Network

```swift
@objc func signInButtonTapped(_ button: UIButton) {
  guard let email = Email(emailTextField.text),
    let password = Password(passwordTextField.text) else { return }

  signIn(email: email, password: password)
}

func signIn(email: Email, password: Password) {
  AuthService.signIn(email: email, password: password) { (isSuccess: Bool) in
    if isSuccess {
      showMainScreen()
    } else {
      showErrorMessage()
    }
  }
}
```

🤔

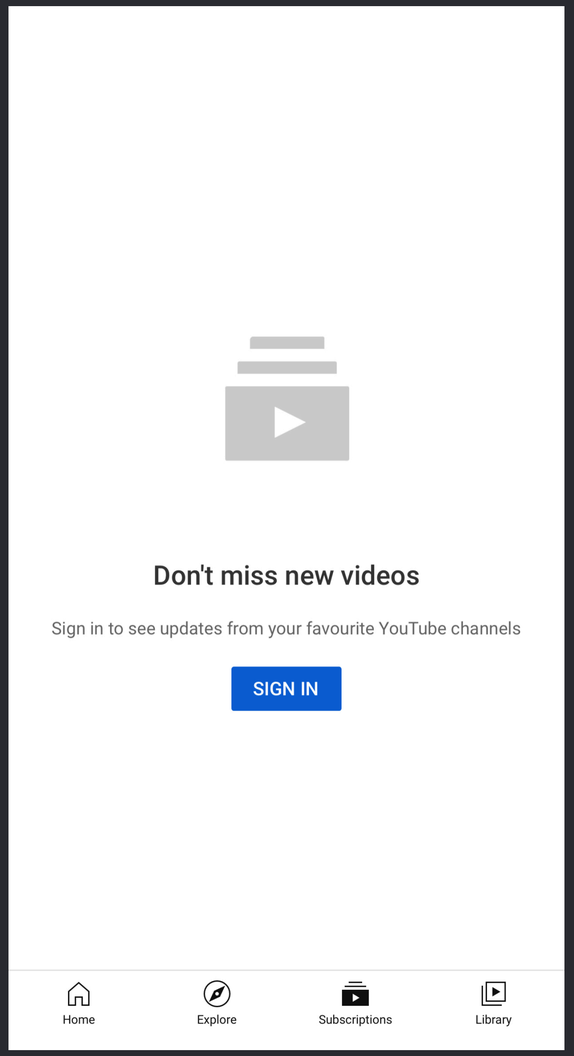
Boolean Blindness

```swift
@objc func signInButtonTapped(_ button: UIButton) {
  guard let email = Email(emailTextField.text),
    let password = Password(passwordTextField.text) else { return }

  signIn(email: email, password: password)
}

func signIn(email: Email, password: Password) {
  AuthService.signIn(email: email, password: password) { (isFailure: Bool) in
    if isFailure {
      showMainScreen()
    } else {
      showErrorMessage()
    }
  }
}
```

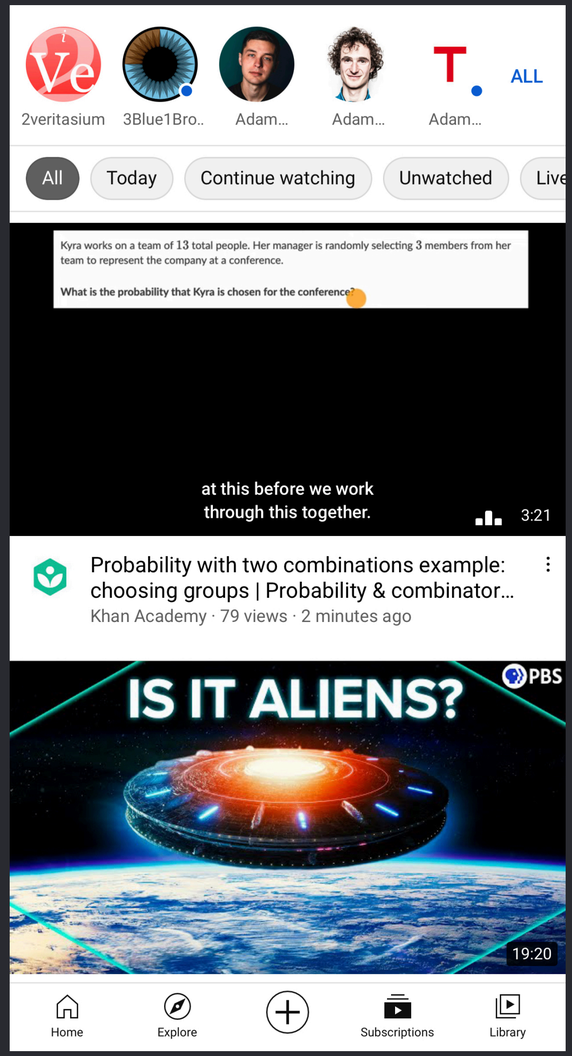
```swift
@objc func signInButtonTapped(_ button: UIButton) {
  guard let email = Email(emailTextField.text),
    let password = Password(passwordTextField.text) else { return }

  signIn(email: email, password: password)
}

func signIn(email: Email, password: Password) {
  AuthService.signIn(email: email, password: password) { (user: User?) in
    if user != nil {
      showMainScreen()
    } else {
      showErrorMessage()
    }
  }
}
```

```swift
@objc func signInButtonTapped(_ button: UIButton) {
  guard let email = Email(emailTextField.text),
    let password = Password(passwordTextField.text) else { return }

  signIn(email: email, password: password)
}

func signIn(email: Email, password: Password) {
  AuthService.signIn(email: email, password: password) { (user: User?) in
    if user ≠ nil {
      showMainScreen()
    } else {
      showErrorMessage()
    }
  }
}
```

```swift
var user: User?
```



Anonymous User



Signed In User

```swift
struct User {
  let userID: String
  let username: String?
  let sessionToken: String?


  var isAnonymous: Bool {
    username == nil && sessionToken == nil
  }
}
```

```swift
struct User {
  let userID: String
  let username: String?
  let sessionToken: String?

  var isAnonymous: Bool {
    username == nil && sessionToken == nil
  }
}
```

```swift
struct SignedInUser {
  let userID: String


}
```

```swift
struct User {
  let userID: String


  var isAnonymous: Bool {
    username == nil && sessionToken == nil
  }
}
```

```swift
struct SignedInUser {
  let userID: String
  let username: String?
  let sessionToken: String?
}
```

```swift
struct AnonymousUser {
  let userID: String
}
```

```swift
struct SignedInUser {
  let userID: String
  let username: String
  let sessionToken: String
}
```

```swift
enum User {
  case anonymous(AnonymousUser)
  case signedIn(SignedInUser)
}
```

```swift
struct AnonymousUser {
  let userID: String
}
```

```swift
struct SignedInUser {
  let userID: String
  let username: String
  let sessionToken: String
}
```

```swift
enum User {
  case anonymous(AnonymousUser)
  case signedIn(SignedInUser)
}
```

```swift
var user: User = .anonymous(AnonymousUser(userID: "#random_string#"))
```

```swift
struct AnonymousUser {
  let userID: String
}
```

```swift
struct SignedInUser: Decodable {
  let userID: String
  let username: String
  let sessionToken: String

  init(from d: Decoder) throws {
    …
  }
}
```

```swift
enum User {
  case anonymous(AnonymousUser)
  case signedIn(SignedInUser)
}
```

```swift
var user: User = .anonymous(AnonymousUser(userID: "#random_string#"))
```

```swift
struct AnonymousUser {
  let userID: String
}
```

```swift
enum User {
  case anonymous(AnonymousUser)
  case signedIn(SignedInUser)
}
```

```swift
struct SignedInUser: Decodable {
  let userID: String
  let username: String
  let sessionToken: String

  init(from d: Decoder) throws {
    …
  }
}
```

```swift
var user: User = .anonymous(AnonymousUser(userID: "#random_string#"))
user = .signedIn(SignedInUser(???))
```

```swift
@objc func signInButtonTapped(_ button: UIButton) {
  guard let email = Email(emailTextField.text),
    let password = Password(passwordTextField.text) else { return }

  signIn(email: email, password: password)
}

func signIn(email: Email, password: Password) {
  AuthService.signIn(email: email, password: password) { (user: SignedInUser?) in
    if user =≠ nil {
      showMainScreen()
    } else {
      showErrorMessage()
    }
  }
}
```

```swift
enum User {
  case anonymous(AnonymousUser)
  case signedIn(SignedInUser)
}

var user: User = .anonymous(…)
```

```
@objc func signInButtonTapped(_ button: UIButton) {
  guard let email = Email(emailTextField.text),
    let password = Password(passwordTextField.text) else { return }

  signIn(email: email, password: password)
}

func signIn(email: Email, password: Password) {
  AuthService.signIn(email: email, password: password) {
  (result: Result<SignedInUser, Error>) in

  if user ≠ nil {
      showMainScreen()
    } else {
      showErrorMessage()
    }
  }
}
```

```
@objc func signInButtonTapped(_ button: UIButton) {
  guard let email = Email(emailTextField.text),
    let password = Password(passwordTextField.text) else { return }

  signIn(email: email, password: password)
}

func signIn(email: Email, password: Password) {
  AuthService.signIn(email: email, password: password) {
  (result: Result<SignedInUser, Error>) in

  switch result {
  case let .success(signedInUser):
    showMainScreen()

  case let .failure(error)
    showErrorMessage()
  }
}
```

```
@objc func signInButtonTapped(_ button: UIButton) {
  guard let email = Email(emailTextField.text),
    let password = Password(passwordTextField.text) else { return }

  signIn(email: email, password: password)
}

func signIn(email: Email, password: Password) {
  AuthService.signIn(email: email, password: password) {
  (result: Result<SignedInUser, Error>) in

  switch result {
  case let .success(signedInUser):
    showMainScreen(signedInUser)

  case let .failure(error)
    showErrorMessage(error)
  }
}
```

```swift
@objc func signInButtonTapped(_ button: UIButton) {
  guard let email = Email(emailTextField.text),
    let password = Password(passwordTextField.text) else { return }

  signIn(email: email, password: password)
}

func signIn(email: Email, password: Password) {
  AuthService.signIn(email: email, password: password) {
  (result: Result<SignedInUser, Error>) in

  switch result {
  case let .success(signedInUser)
    showMainScreen(signedInUser)

  case let .failure(error)
    showErrorMessage(error)
  }
}
```

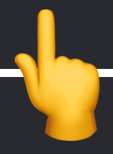
```swift
func showMainScreen(_ user: SignedInUser) {
  let vc = MainScreenViewController()
  navigationController.push(vc)
}
```

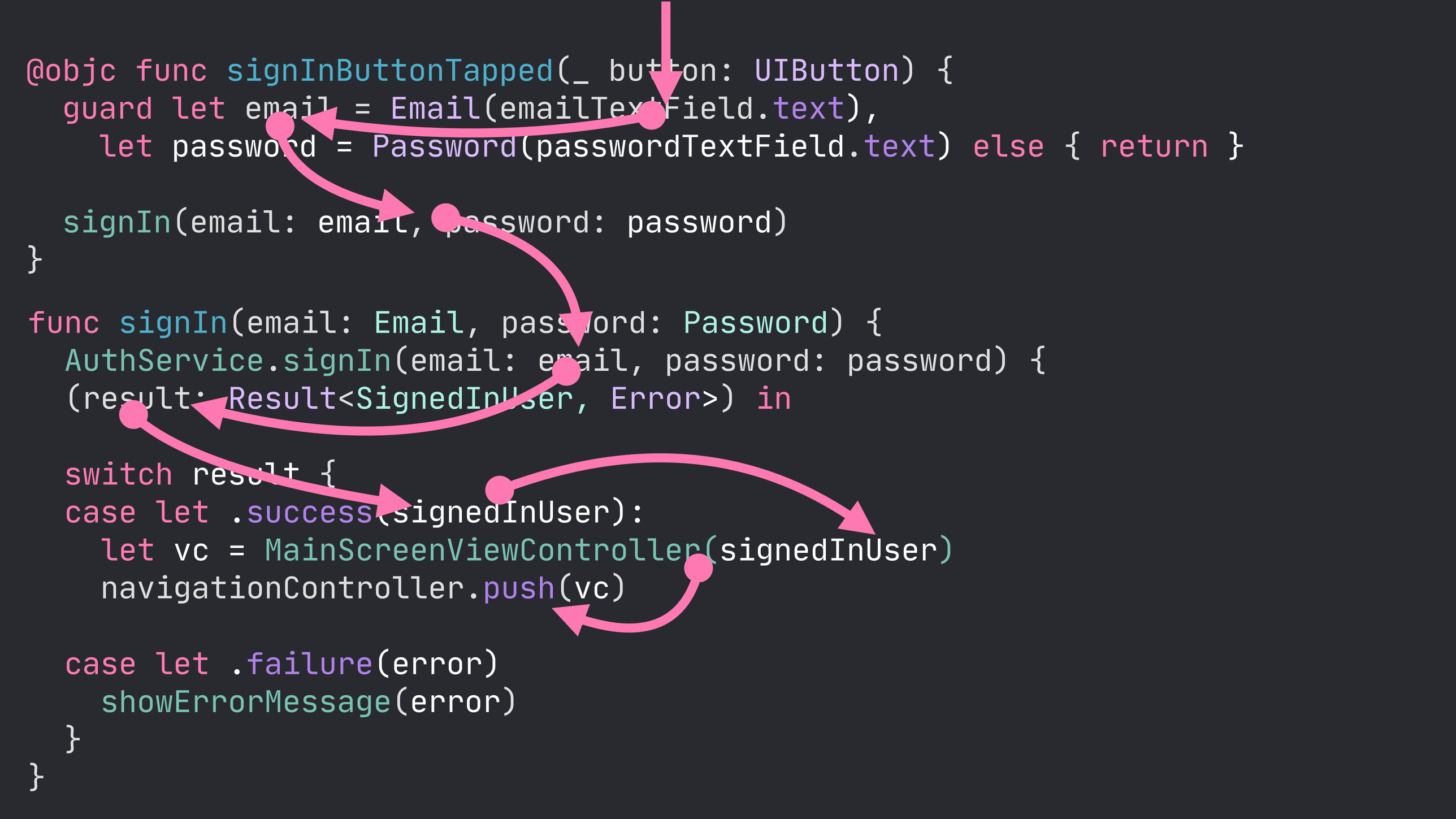
```swift
@objc func signInButtonTapped(_ button: UIButton) {
  guard let email = Email(emailTextField.text),
    let password = Password(passwordTextField.text) else { return }

  signIn(email: email, password: password)
}

func signIn(email: Email, password: Password) {
  AuthService.signIn(email: email, password: password) {
  (result: Result<SignedInUser, Error>) in

  switch result {
  case let .success(signedInUser)
    showMainScreen(signedInUser)

  case let .failure(error)
    showErrorMessage(error)
  }
}
```

```swift
func showMainScreen(_ user: SignedInUser) {
  let vc = MainScreenViewController(user)
  navigationController.push(vc)
}
```

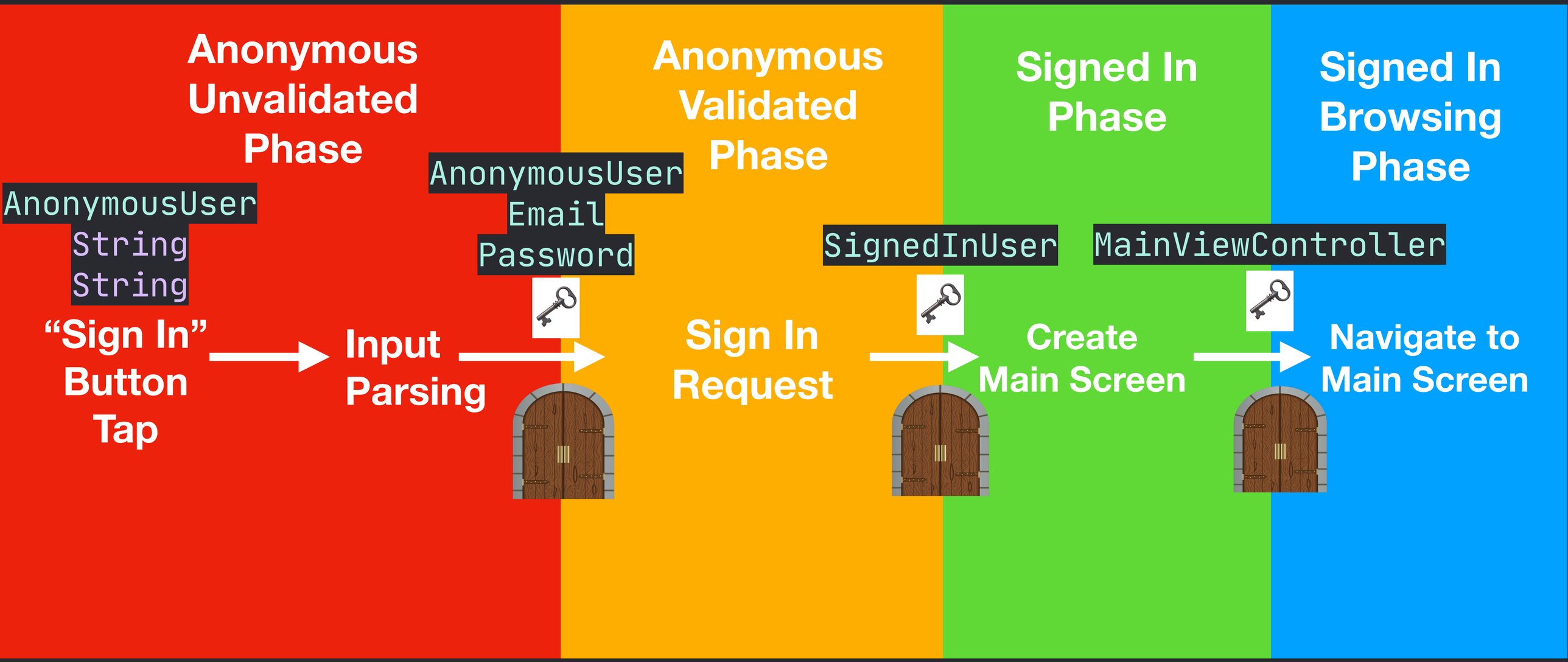
```swift
@objc func signInButtonTapped(_ button: UIButton) {
  guard let email = Email(emailTextField.text),
    let password = Password(passwordTextField.text) else { return }

  signIn(email: email, password: password)
}

func signIn(email: Email, password: Password) {
  AuthService.signIn(email: email, password: password) {
  (result: Result<SignedInUser, Error>) in

  switch result {
  case let .success(signedInUser)
    showMainScreen(signedInUser)

  case let .failure(error)
    showErrorMessage(error)
  }
}
```

```swift
func showMainScreen(_ user: SignedInUser) {
  let vc = MainScreenViewController(user)
  navigationController.push(vc)
}

class MainScreenViewController: UIViewCo..{
  init(_: SignedInUser) { … }
}
```

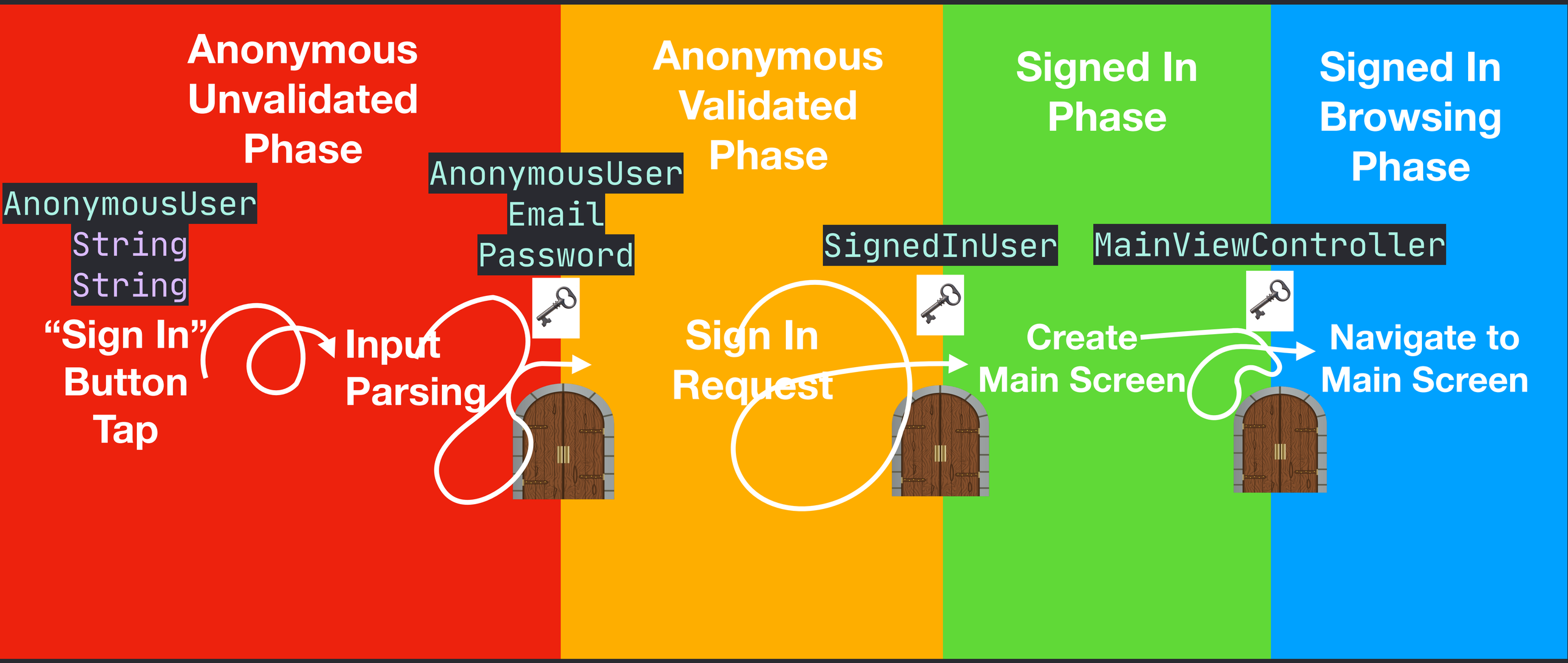


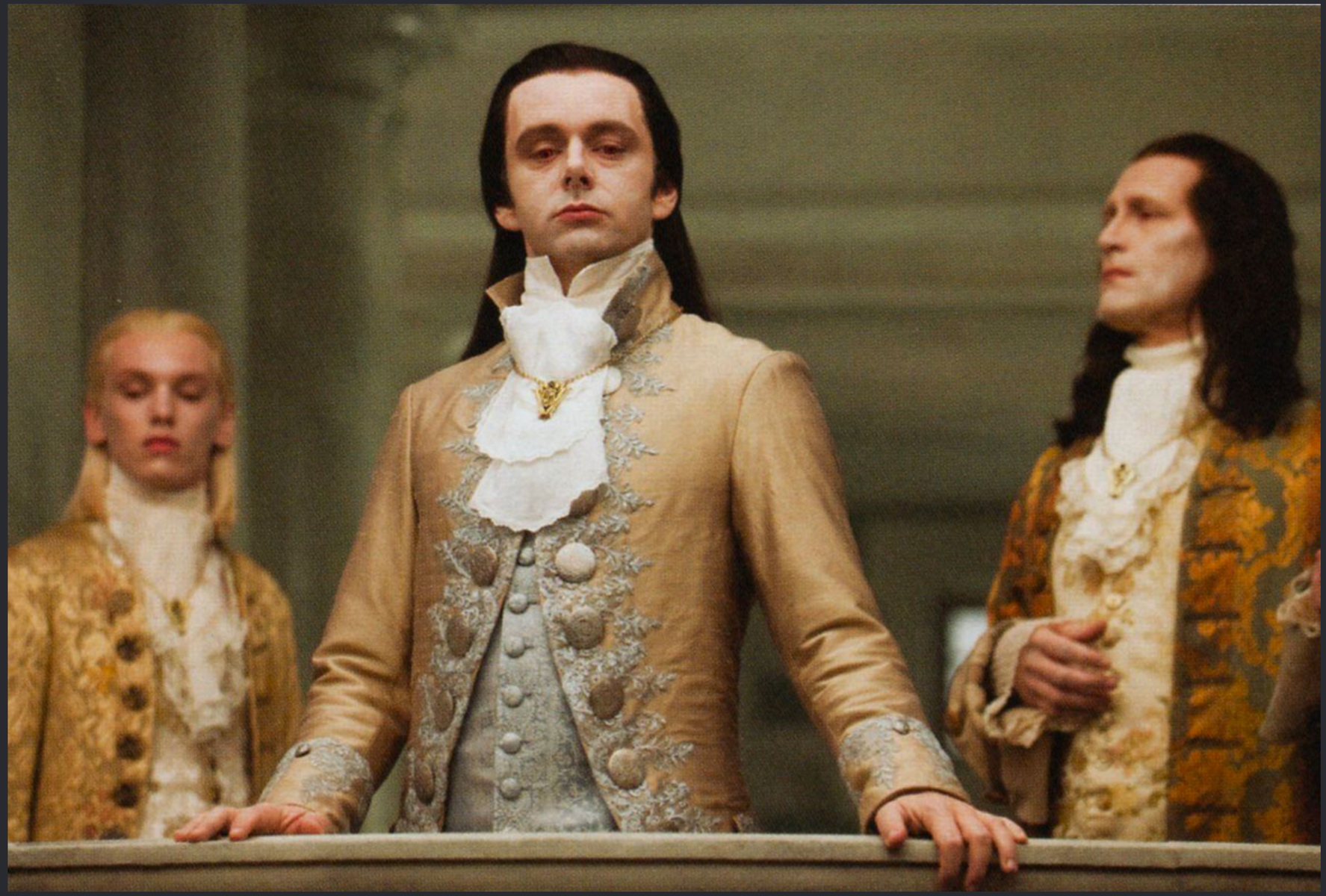

@objc func signInButtonTapped(_ button: UIButton) {
guard let email = Email(emailTextField.text),
let password = Password(passwordTextField.text) else { return }
signIn(email: email, password: password)
}
func signIn(email: Email, password: Password) {
AuthService.signIn(email: email, password: password) {
(result: Result<SignedInUser, Error>) in
switch result {
case let .success(signedInUser):
let vc = MainScreenViewController(signedInUser)
navigationController.push(vc)
case let .failure(error)
showErrorMessage(error)
}
}

Type-Driven Sign In Flow



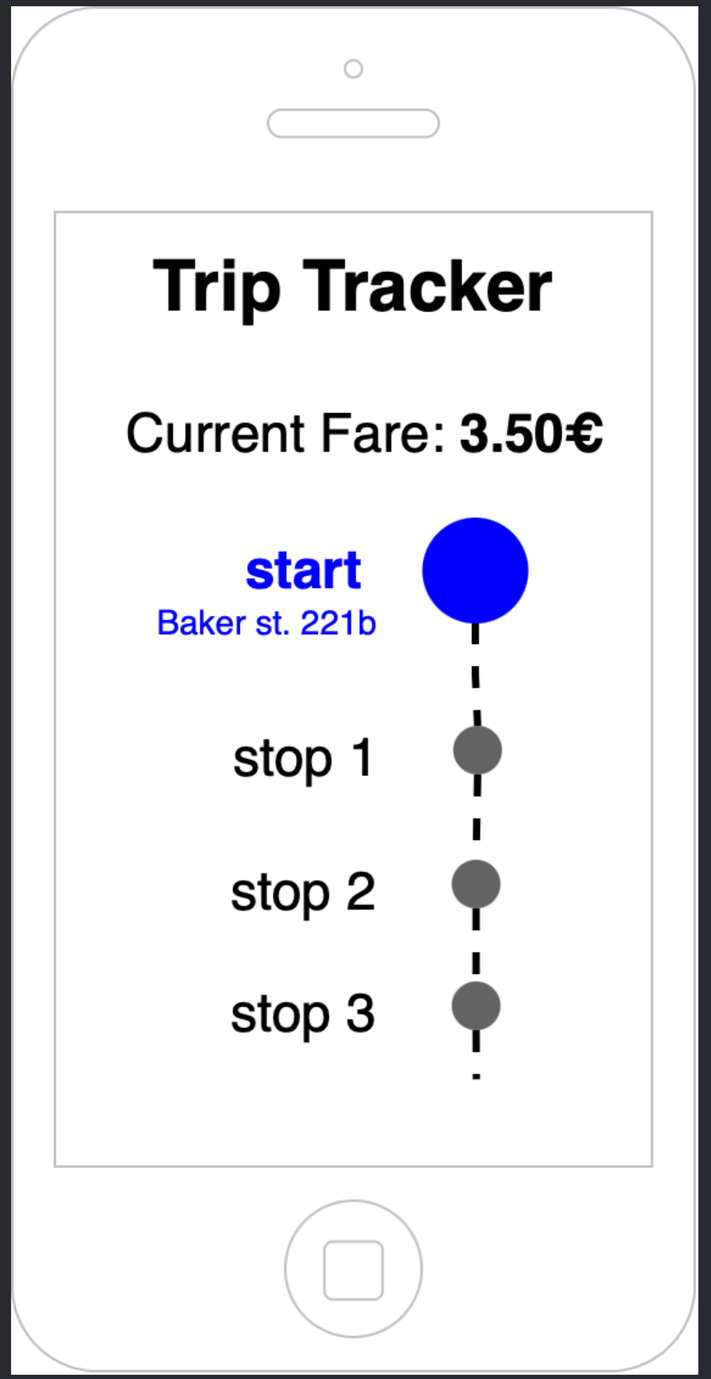
Type-Driven Sign In Flow



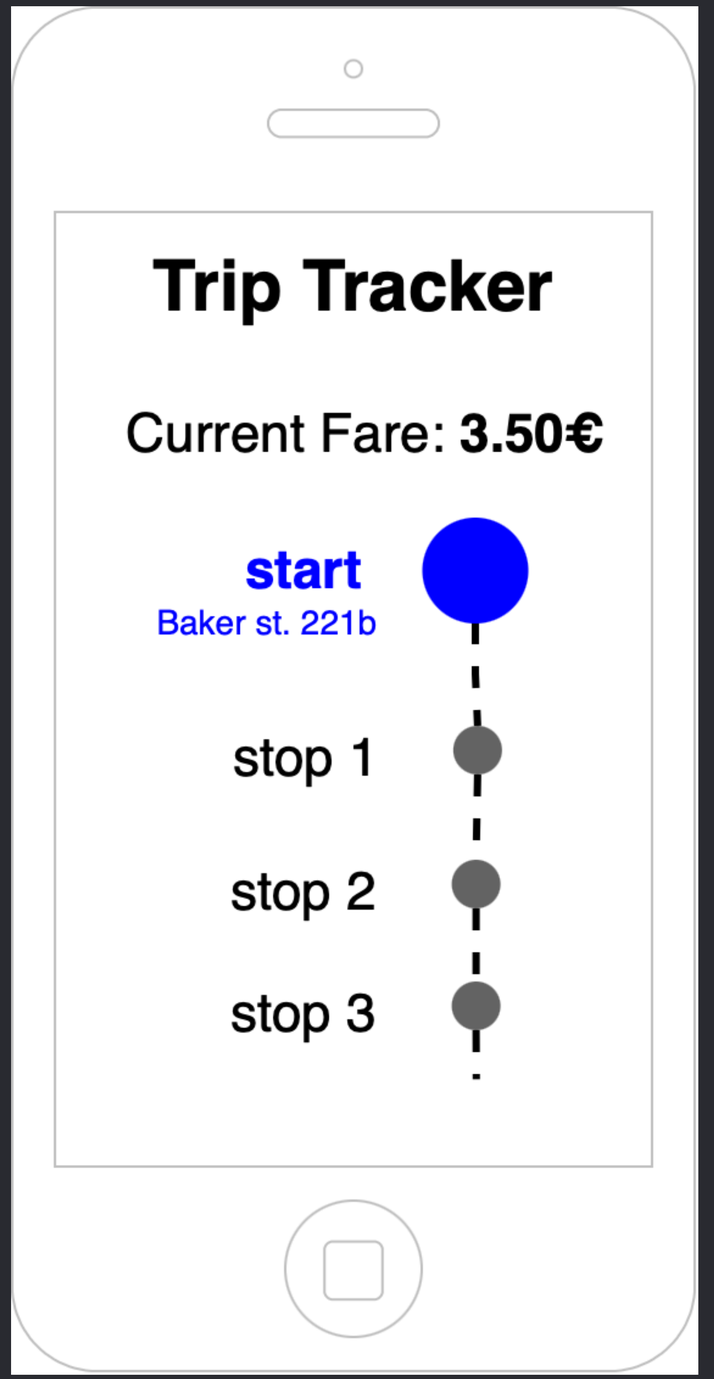


Type Safety Back and Forth

```
struct TripTrackerView: View {
  let stops: [Stop]

  var currentFare: Decimal {...}

  var body: some View {
     VStack {
       Text("Current Fare: \(currentFare)€")
       StartPointView()
       StopPointView()
       ...
       StopPointView()
     }
  }
}
```

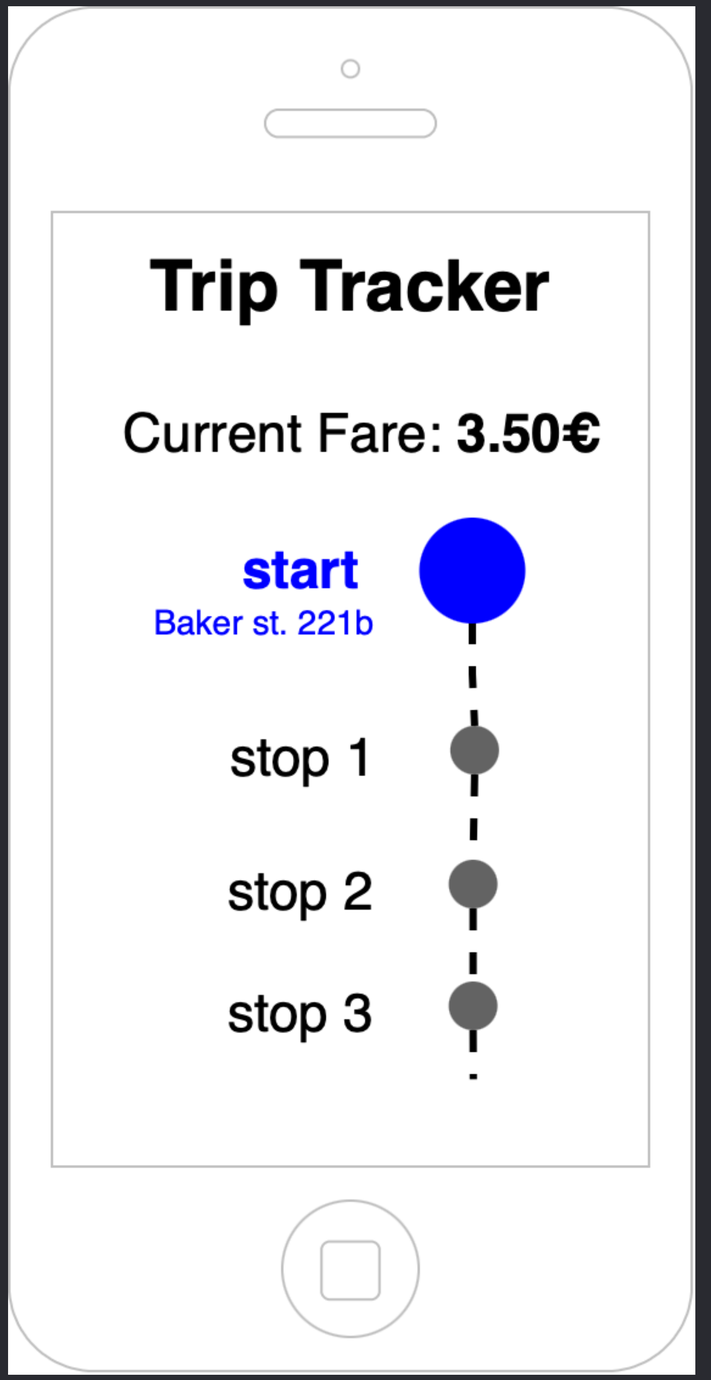


```swift
struct TripTrackerView: View {
  let stops: [Stop]

  var currentFare: Decimal {
    pickUpFare + (tripDistance * 1.5)
  }

  var pickUpFare: Decimal {...}

  var tripDistance: Decimal {...}

  var body: some View {
     VStack {
       Text("Current Fare: \(currentFare)€")
       StartPointView()
       StopPointView()
       ...
       StopPointView()
     }
  }
}
```

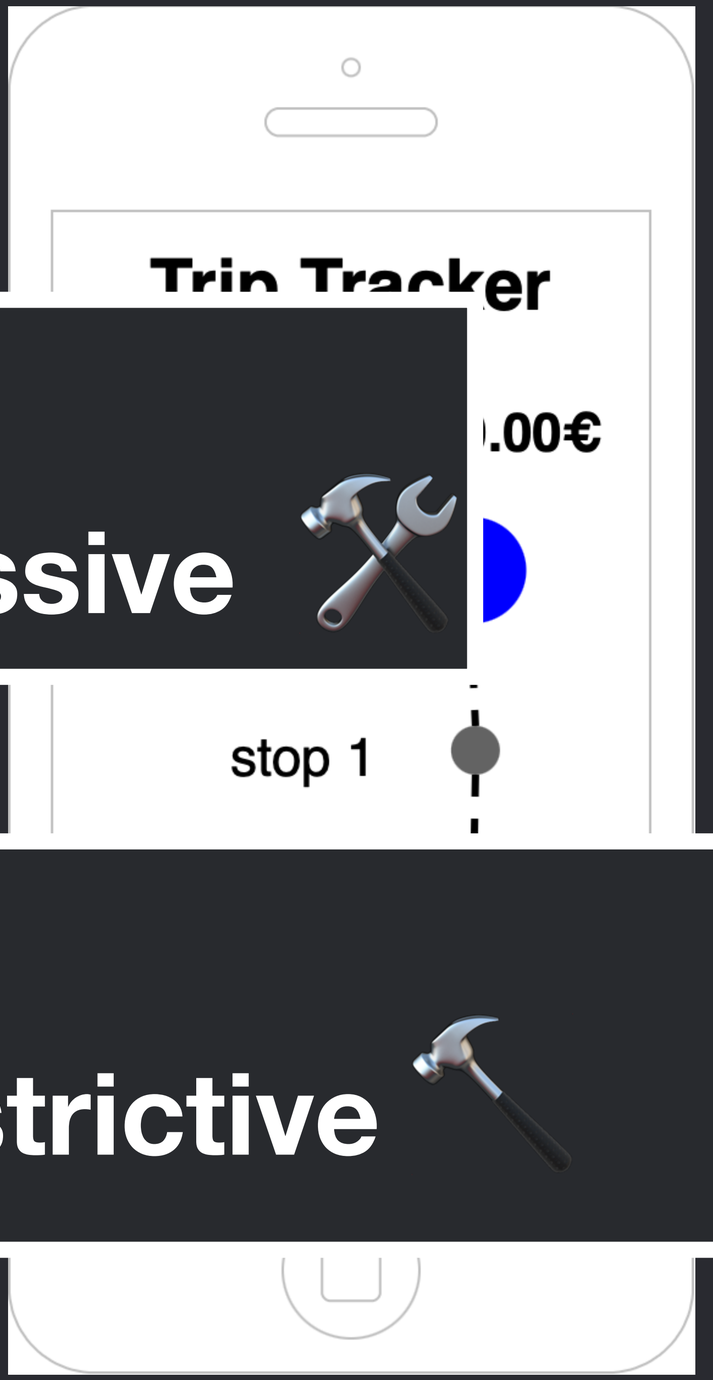


```
struct TripTrackerView: View {
  let stops: [Stop]
  var currentFare: Decimal {
    pickUpFare + (tripDistance * 1.5)
  }

  var pickUpFare: Decimal {
     if let start = stops.first {
        // calculating..
     } else {
        return ???
     }
  }

  var tripDistance: Decimal {...}

  var body: some View {
     VStack {
       Text("Current Fare: \(currentFare)€")
       ...
     }
  }
}
```

stops: []



```
struct TripTrackerView: View {
  let stops: [Stop]

  var currentFare: Decimal {
    pickUpFare + (tripDistance * 1.5)
  }
    }
  }
  }
}
```



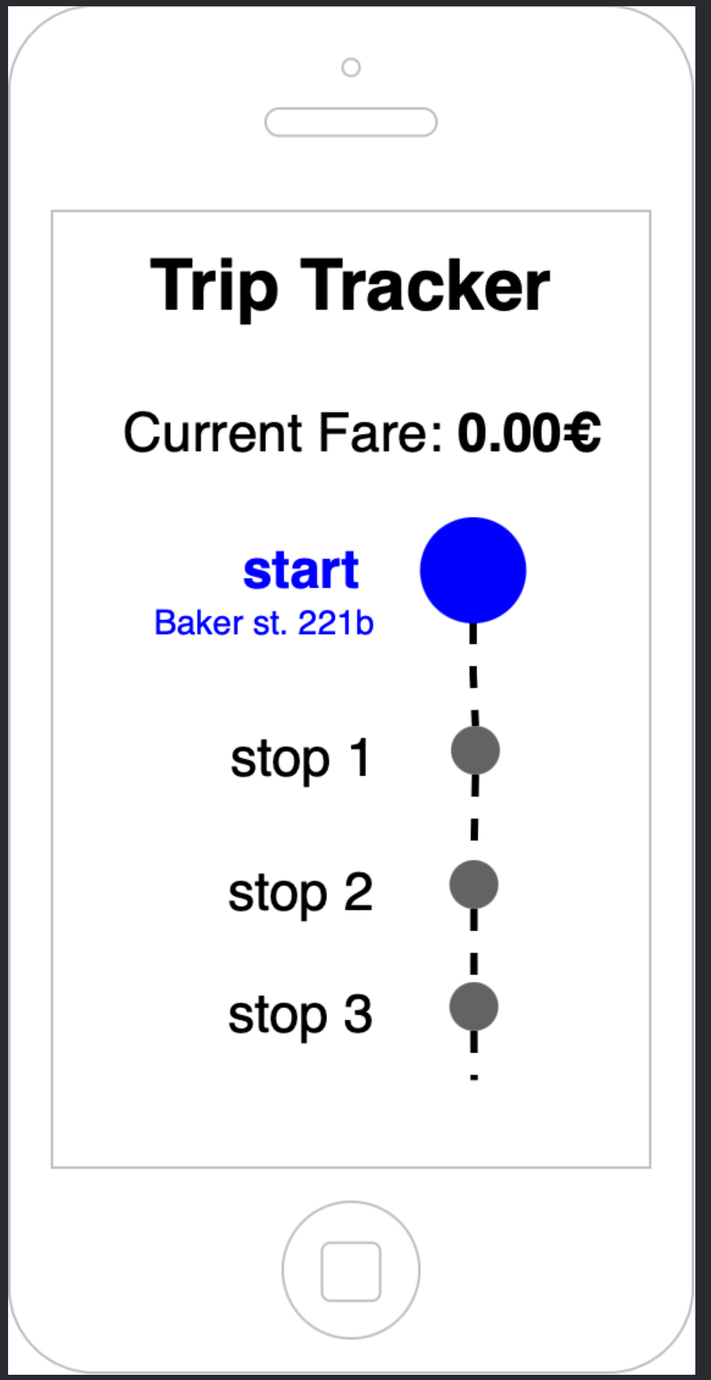
Choice 1

Make return type more permissive 🛠️

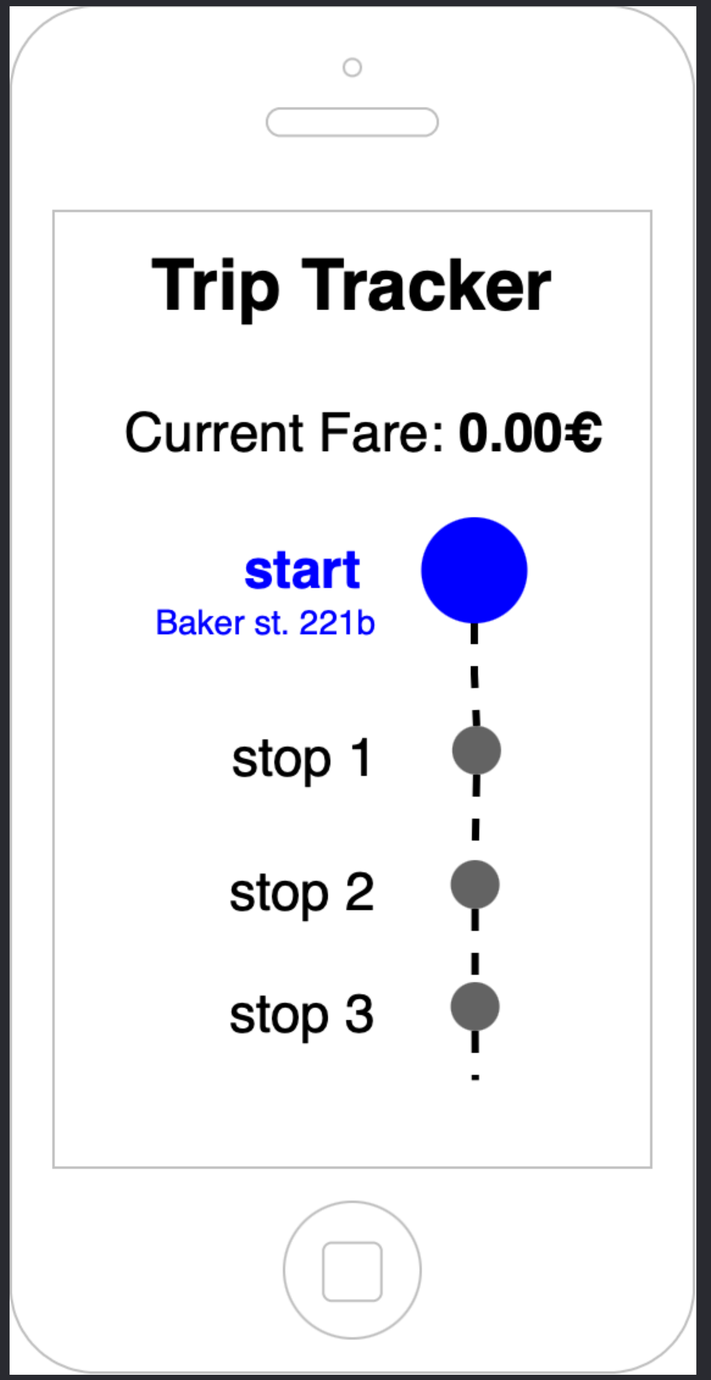
Choice 2

Make argument type more restrictive 🔨

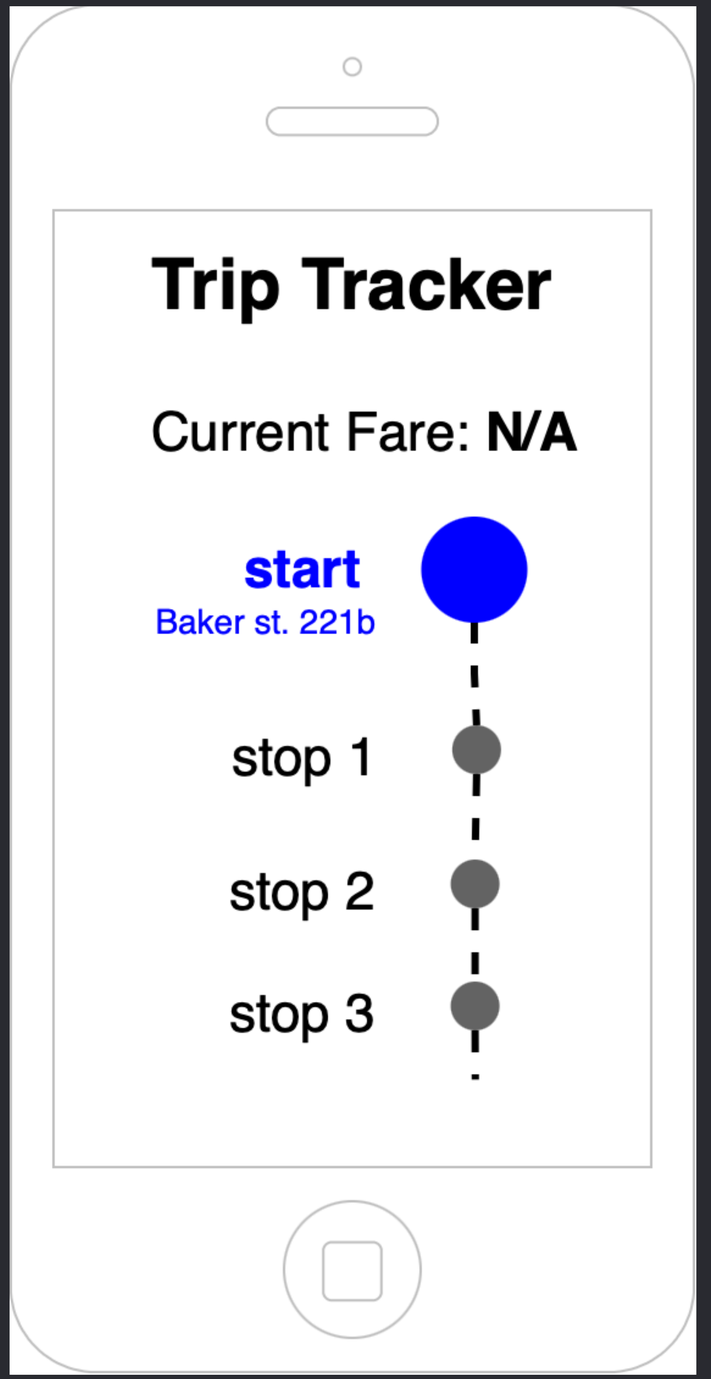
```swift
struct TripTrackerView: View {
  let stops: [Stop]

  var currentFare: Decimal {
    pickUpFare + (tripDistance * 1.5)
  }

  var pickUpFare: Decimal? {
    guard let start = stops.first else { return nil }
    // calculating..
  }



  var tripDistance: Decimal {...}

  var body: some View {
     VStack {
       Text("Current Fare: \(currentFare)€")
       ...
     }
  }
}
```



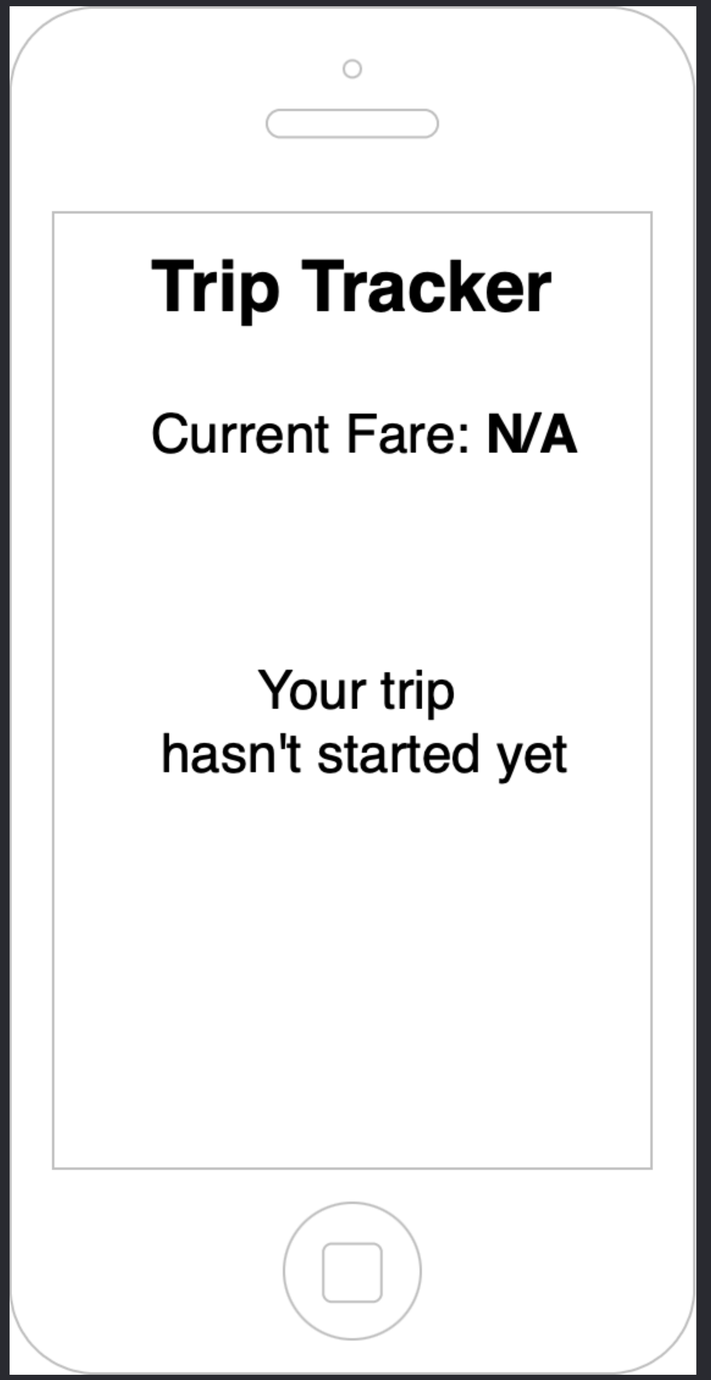
```swift
struct TripTrackerView: View {
  let stops: [Stop]

  var currentFare: Decimal? {
   guard let pickUpFare else { return nil }

    return pickUpFare + (tripDistance * 1.5)
  }

  var pickUpFare: Decimal? {
    guard let start = stops.first else { return nil }
    // calculating..
  }


  var tripDistance: Decimal {...}

  var body: some View {
     VStack {
       Text("Current Fare: \(currentFare)€")
       ...
     }
  }
}
```



```swift
struct TripTrackerView: View {
  let stops: [Stop]

  var currentFare: Decimal? {
   guard let pickUpFare else { return nil }

    return pickUpFare + (tripDistance * 1.5)
  }

  var pickUpFare: Decimal? {
    guard let start = stops.first else { return nil }
    // calculating..
  }


  var tripDistance: Decimal {...}

  var body: some View {
     VStack {
       Text("Current Fare: \(currentFare ?? "N/A")")
       ...
     }
  }
}
```

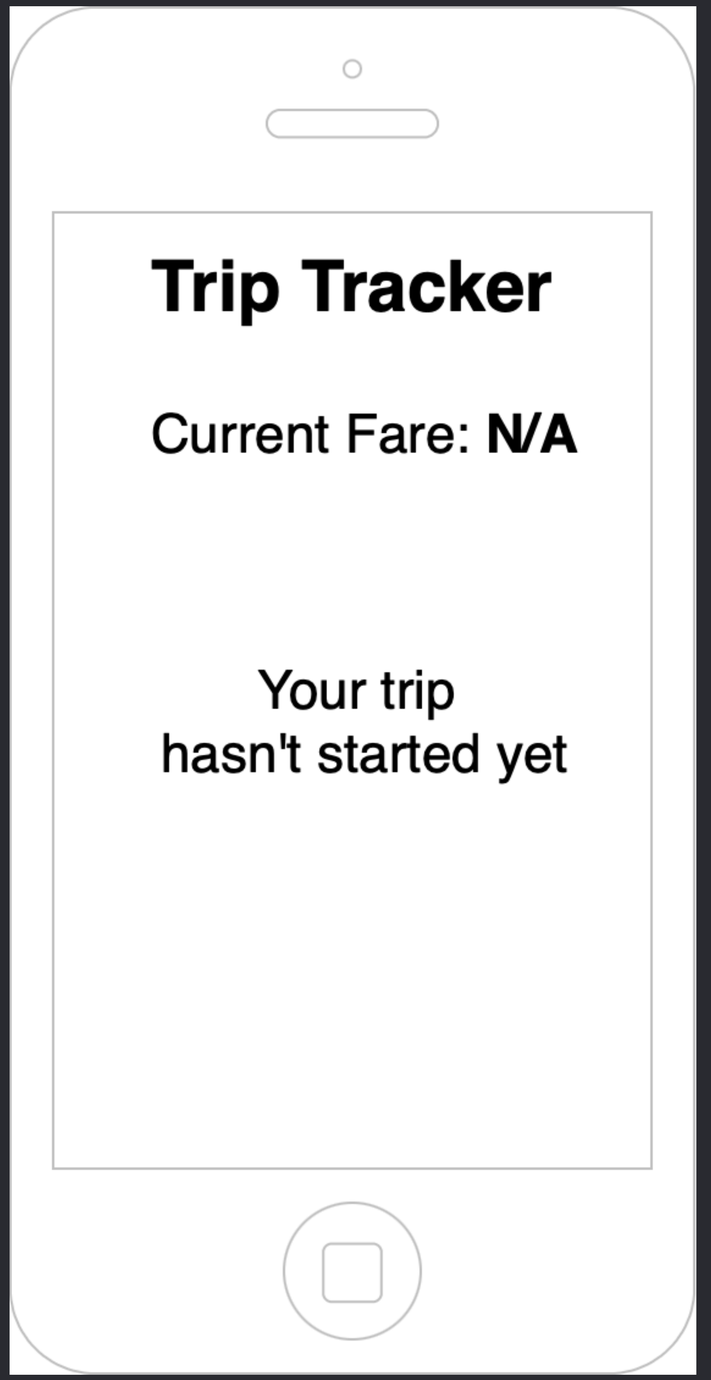


```swift
struct TripTrackerView: View {
  let stops: [Stop]

  var currentFare: Decimal? {
    guard let pickUpFare else { return nil }
    return pickUpFare + (tripDistance * 1.5)
  }
  var pickUpFare: Decimal? {
    guard let start = stops.first else { return nil }
    // calculating..
  }
  var body: some View {
     VStack {
       Text("Current Fare: \(currentFare ?? "N/A")")
       if let start = stops.first {
         StartPointView(start)

         ForEach(stops.dropFirst()) { stop in
           StopPointView(stop)
         }
       } else {
         Text("Your trip hasn’t started yet")
       }
     }
  }
}
```

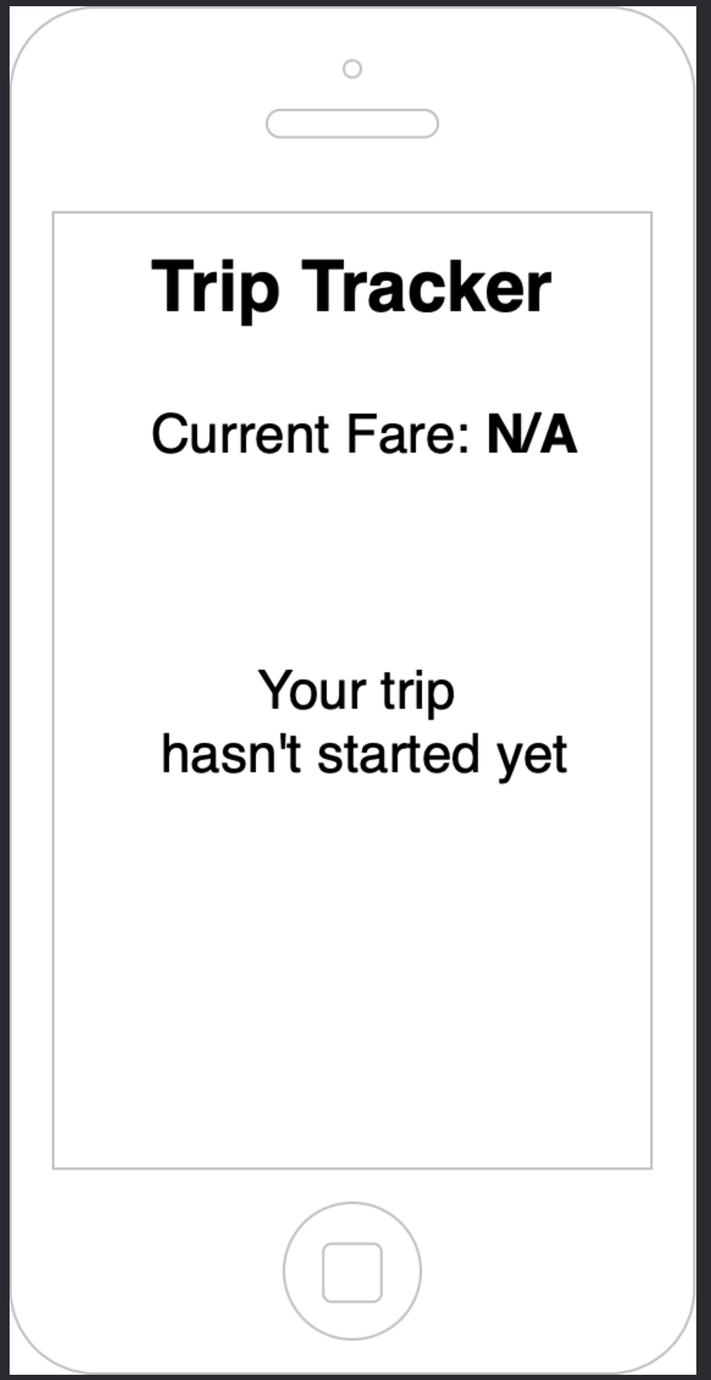


```
struct NonEmpty<Element> {
    let head: Element
    let tail: [Element]
}
```

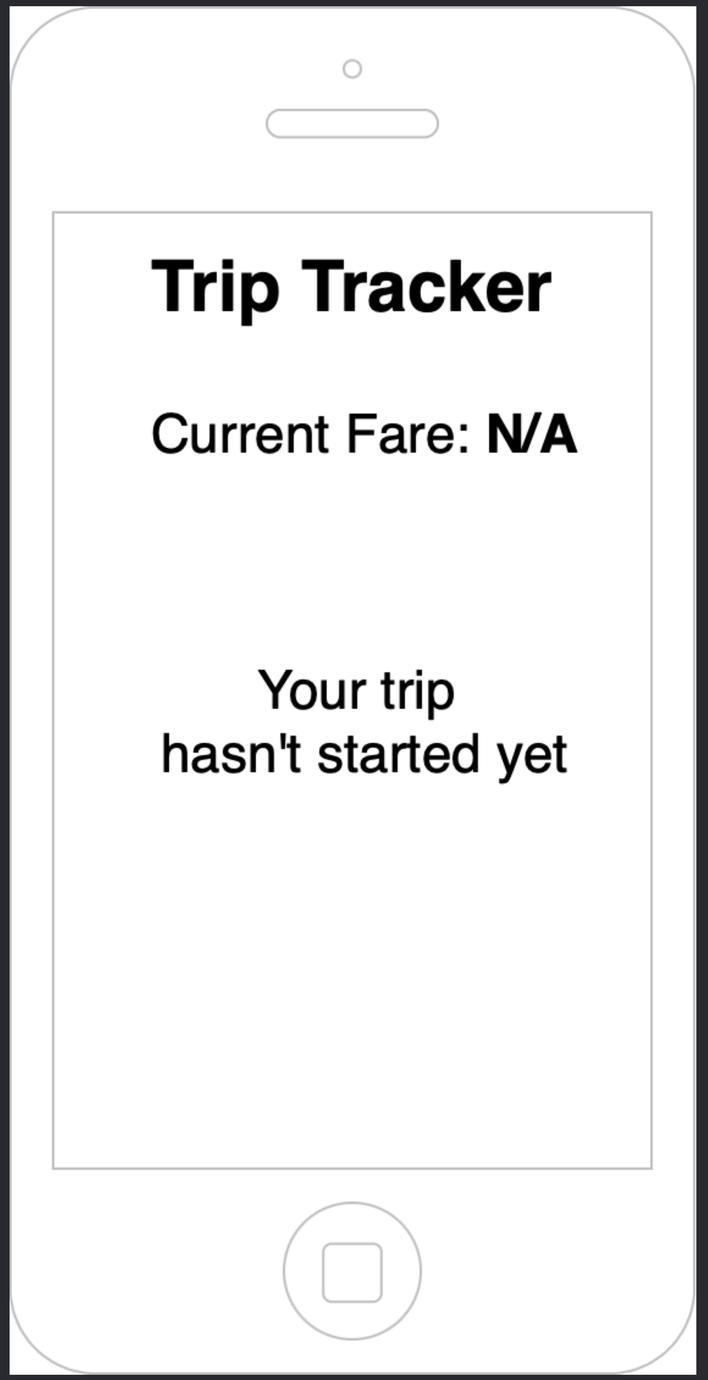
```
struct TripTrackerView: View {
  let stops: NonEmpty<Stop>

  var currentFare: Decimal? {
    guard let pickUpFare else { return nil }
    pickUpFare + (tripDistance * 1.5)
  }
  var pickUpFare: Decimal? {
    guard let start = stops.first else { return nil }
    // calculating..
  }
  var body: some View {
     VStack {
       Text("Current Fare: \(currentFare ?? "N/A")")
       if let start = stops.first {
         StartPointView(start)

         ForEach(stops.dropFirst()) { stop in
           StopPointView(stop)
         }
       } else {
         Text("Your trip hasn’t started yet")
       }
     }
  }
}
```



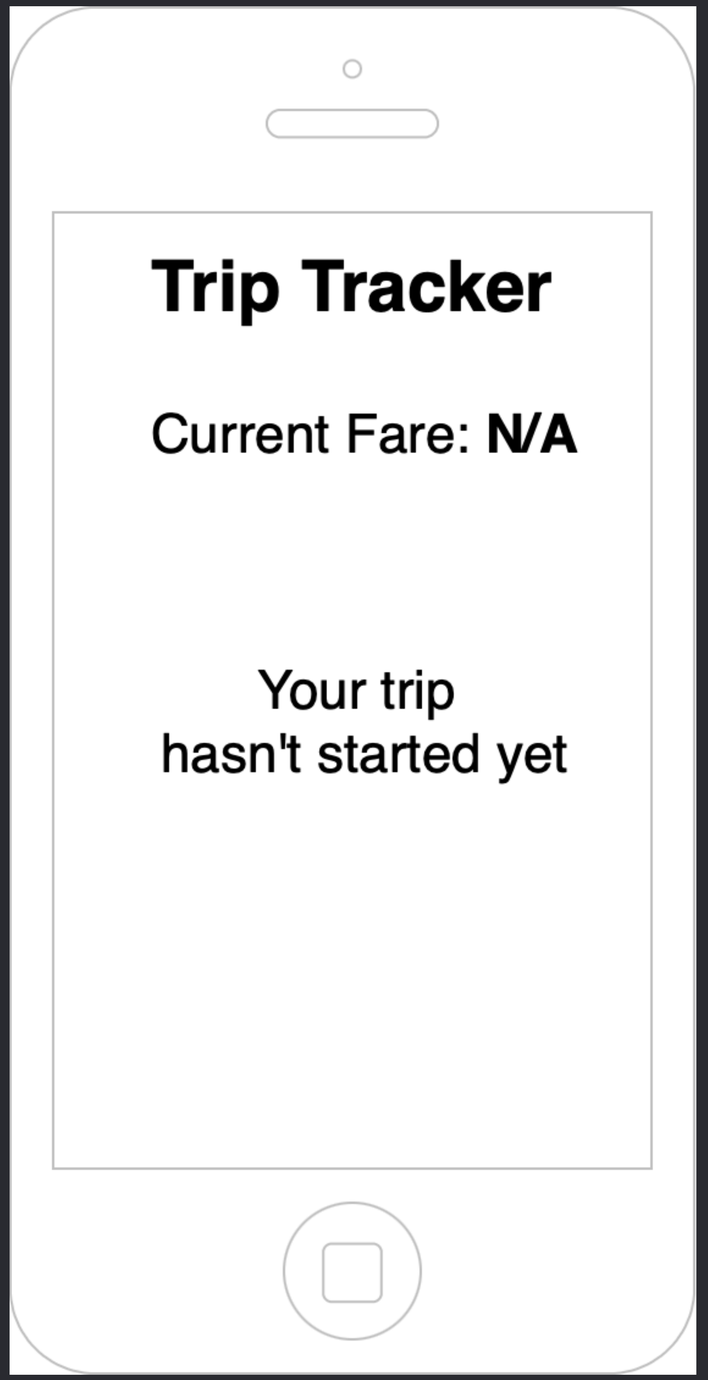
```swift
struct TripTrackerView: View {
  let stops: NonEmpty<Stop>

  var currentFare: Decimal  {

    pickUpFare + (tripDistance * 1.5)
  }
  var pickUpFare: Decimal  {
    let start = stops.head
    // calculating..
  }
  var body: some View {
     VStack {
       Text("Current Fare: \(currentFare ?? "N/A")")
       if let start = stops.first {
         StartPointView(start)

         ForEach(stops.dropFirst()) { stop in
           StopPointView(stop)
         }
       } else {
         Text("Your trip hasn’t started yet")
       }
     }
  }
}
```



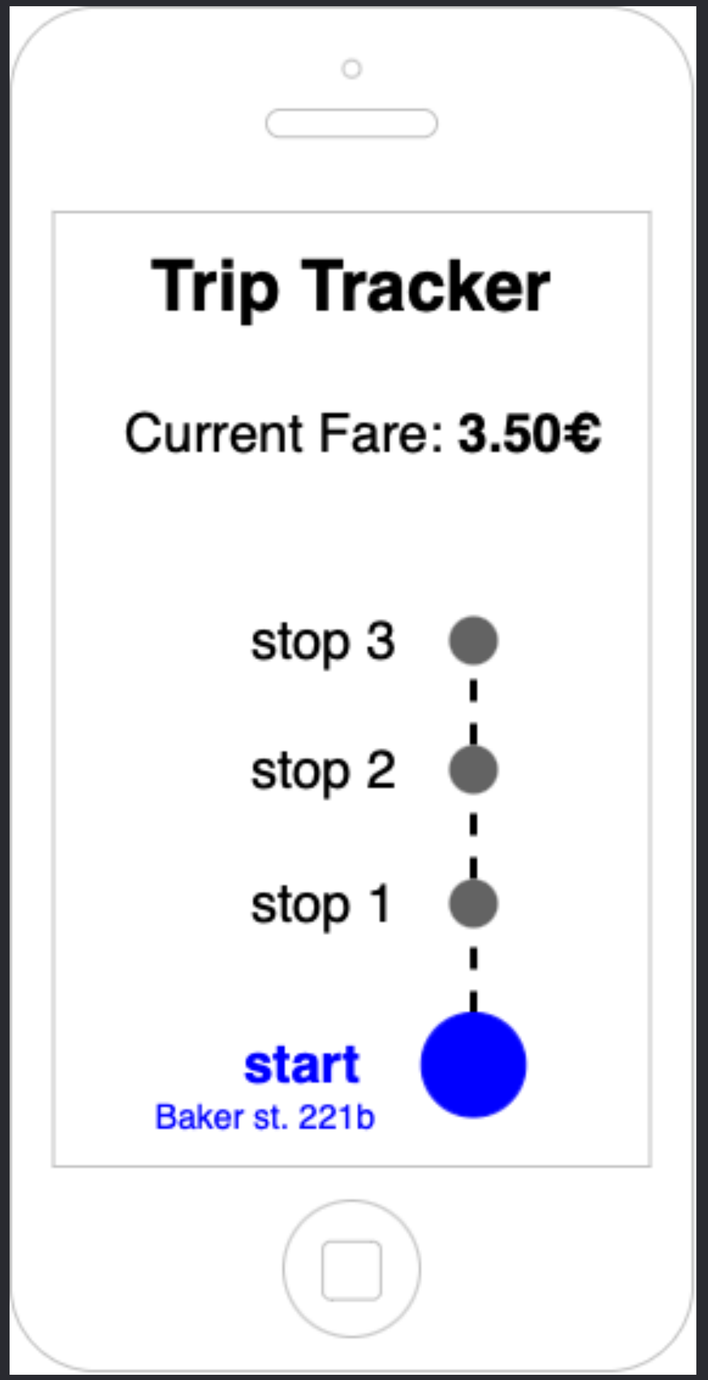
```
struct TripTrackerView: View {
  let stops: NonEmpty<Stop>

  var currentFare: Decimal  {

    pickUpFare + (tripDistance * 1.5)
  }
  var pickUpFare: Decimal  {
    let start = stops.head
    // calculating..
  }
  var body: some View {
     VStack {
       Text("Current Fare: \(currentFare)")
       if let start = stops.first {
         StartPointView(start)

         ForEach(stops.dropFirst()) { stop in
           StopPointView(stop)
         }
       } else {
         Text("Your trip hasn’t started yet")
       }
     }
  }
}
```

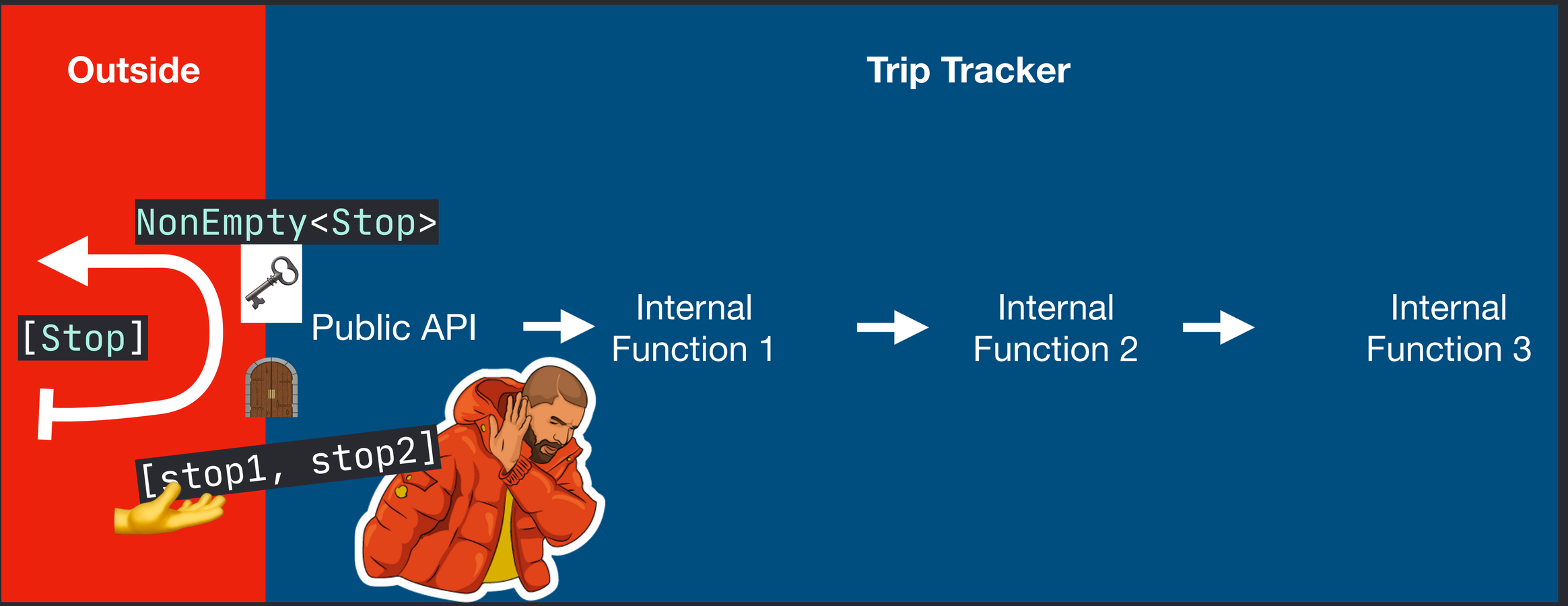


```
struct TripTrackerView: View {
  let stops: NonEmpty<Stop>

  var currentFare: Decimal  {

    pickUpFare + (tripDistance * 1.5)
  }
  var pickUpFare: Decimal  {
    let start = stops.head
    // calculating..
  }
  var body: some View {
     VStack {
       Text("Current Fare: \(currentFare)")

         StartPointView(stops.head)

         ForEach(stops.tail)) { stop in
           StopPointView(stop)
         }



     }
  }
}
```



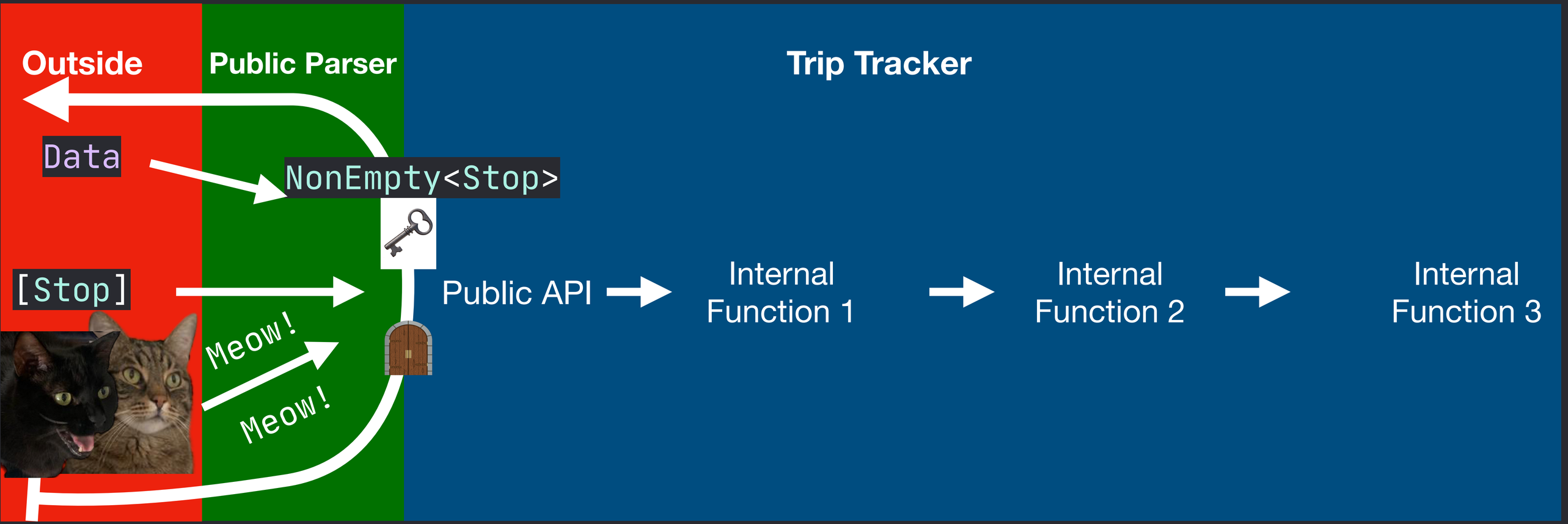
```swift
struct TripTrackerView: View {
  let stops: NonEmpty<Stop>

  var currentFare: Decimal  {
    pickUpFare + (tripDistance * 1.5)
  }
  var pickUpFare: Decimal  {
    let start = stops.head
    // calculating..
  }

  var body: some View {
     VStack {
       Text("Current Fare: \(currentFare)")

       StartPointView(stops.head)

       ForEach(stops.tail)) { stop in
         StopPointView(stop)
       }
     }
  }
}
```





Robustness principle (Postel's Law): 🤔

Allow non-conformant input as long as the meaning is clear




```
public init(stops: NonEmpty<Stop>) {…}
public init?(stops: [Stop]) {
    guard let nonEmptyStops = NonEmpty(stops) else { return nil }
    self.init(stops: nonEmptyStops)
}
public init?(stops: Data) {…}
public init?(stops: CatSpeech) {…}
```

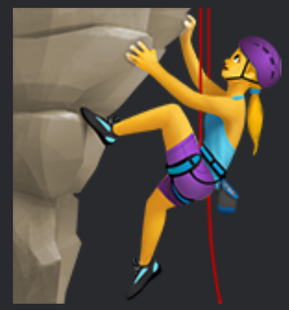
```
@objc func signInButtonTapped(_ button: UIButton) {
  showMainScreen()
}


class MainViewController: UIViewController {
  init(_: SignedInUser) { … }
}


struct SignedInUser: Decodable {
  init(from d: Decoder) throws { … }
}
```

```
@objc func signInButtonTapped(_ button: UIButton) {
  showMainScreen()
}
```

```
@objc func signInButtonTapped(_ button: UIB
  let data = """
  {
    "userID": "123",
    "username": "alex",
    "sessionToken": "1dfF7hlxhakhjx",
  }
  """.data(using: .utf8)
  let signedInUser = try! JSONDecoder().decode(SignedInUser.self, from: data)
  navigationController.push(MainScreenViewController(signedInUser))
}
```

Success

Mistakes



Falling into the Pit of Success

Jeff Atwood

✦Parse, Don’t Validate.

Prefer Parsing to Validation to retain information inside Types.

✦Type, Define, Refine.

Capture essential requirements with empty Types. Then fill them in. Then refine them as needed. Repeat.

✦Type Safety Back and Forth.

Push Type Safety to your public APIs. Keep your domain clean. Provide user-friendly APIs for non-conformant inputs.

✦Falling into the Pit of Success.

Great Type Safe API design allows users to naturally do the right thing. Doing the wrong thing becomes annoyingly hard.

Type-Driven Design with Swift

📖 Table of Contents: