

ERROR HANDLING

MADE EASY

Eleni Papanikolopoulou

iOS Developer @Workable

 @elenipapanikolo

Kostas Kremizas

iOS Developer @Workable

 @kremizask

Definitions

Error = "a value used to report that an error condition occurred and normal functionality was skipped"

Definitions

Error = "a value used to report that an error condition occurred and normal functionality was skipped"

Error Handling = "code that looks for errors and performs different actions based on the presence of those errors"

“Error handling is the art of
failing gracefully”

– *Swift Apprentice, Chapter 22 (Error Handling)*

Why bother?

More and more critical actions in apps

Why bother?

More and more critical actions in apps

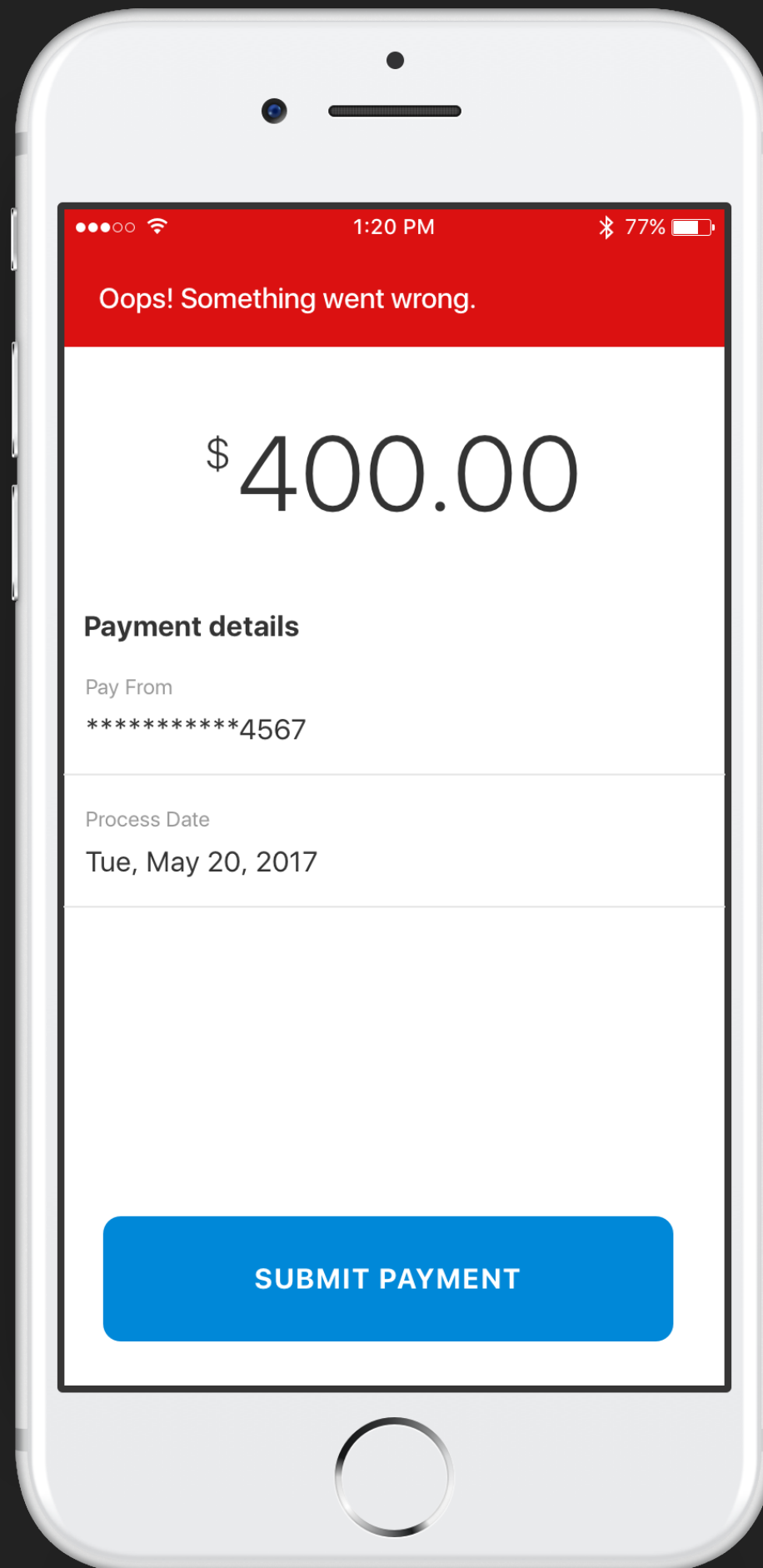
Apps often rely on unreliable sources

Why bother?

More and more critical actions in apps

Apps often rely on unreliable sources

Bad error handling => users don't trust our app



Oops! Something went wrong.

\$400.00

Payment details

Pay From

*****4567

Process Date

Tue, May 20, 2017

SUBMIT PAYMENT

Why so cryptic?

It's *just* copywriting and easy to fix

Right?

ONE DOES NOT SIMPLY

HANDLE ERRORS

Steps

For each method that can error out:

1. List possible errors

Steps

For each method that can error out:

1. List possible errors
2. Handle each error

1. List possible errors

Technical

- Network
- Disk
- Server unavailability

Business

- Authorization
- Validation
- Stale or bad data

2. Handle each error

Present a relevant error message

2. Handle each error

Present a relevant error message

Additional actions per case

2. Handle each error

Present a relevant error message

Additional actions per case

Log & report the error (always)

“In theory there is no difference
between theory and practice.
In practice there is.”

– *Jan L. A. van de Snepscheut*



Adding error handling to a simple login screen

Carrier 6:03 PM

Welcome

LOGIN

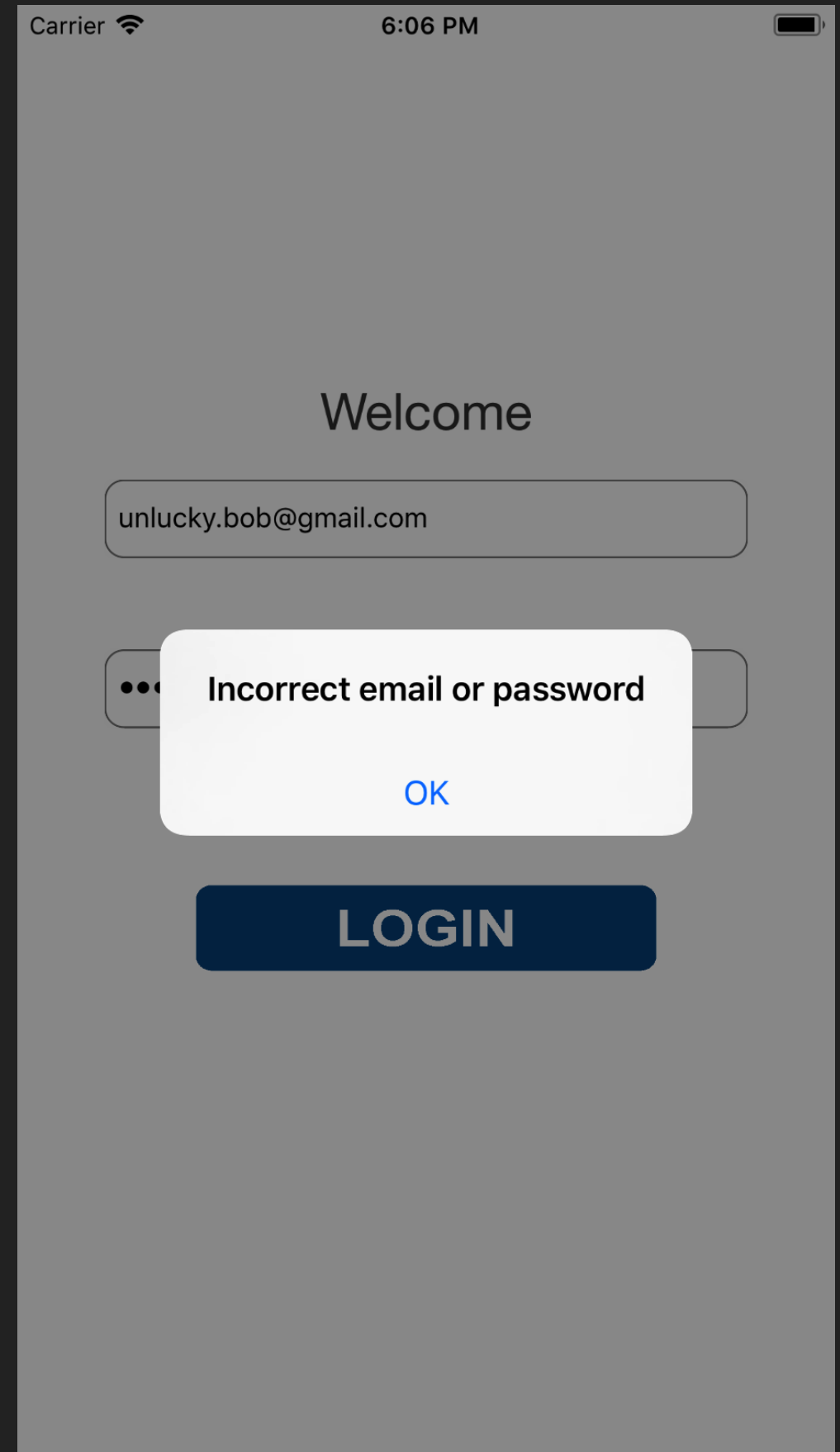
Adding error handling to a simple login screen

Carrier  6:03 PM 

Welcome

LOGIN

Adding error handling to a simple login screen



```
@IBAction func loginTapped(_ sender: Any) {  
    api.login(email, password) { (response, error) in  
        if let error = error {  
  
            switch error {  
            case let error as HttpError where error.status == 401:  
                showAlert(message: "Incorrect email or password")  

```

```
    }  
    }  
    }  
}
```

Carrier



6:02 PM



You appear to be offline. Please check
your connection

Welcome

LOGIN

```
@IBAction func loginTapped(_ sender: Any) {
    api.login(email, password) { (response, error) in
        if let error = error {

            switch error {
            case let error as HttpError where error.status == 401:
                showAlert(message: "Incorrect email or password")

            case let error as NSError
                where error.domain == NSURLErrorDomain
                && error.code == NSURLErrorNotConnectedToInternet:
                showWarning("You appear to be offline. Please check your connection.")

            }
        }
    }
}
```

Carrier



6:08 PM



Welcome

Invalid email

LOGIN


```
@IBAction func loginTapped(_ sender: Any) {
    api.login(email, password) { (response, error) in
        if let error = error {

            switch error {
            case let error as HttpError where error.status == 401:
                showAlert(message: "Incorrect email or password")

            case let error as NSError
                where error.domain == NSURLErrorDomain
                && error.code == NSURLErrorNotConnectedToInternet:
                showWarning("You appear to be offline. Please check your connection.")

            case let error as SignInError where error == .invalidEmail:
                showInvalidEmail()

            }
        }
    }
}
```

Carrier



6:06 PM



Sorry, there was a problem. Please try again

Welcome

unlucky.bob@gmail.com

••••••••

LOGIN

```
@IBAction func loginTapped(_ sender: Any) {
    api.login(email, password) { (response, error) in
        if let error = error {

            switch error {
            case let error as HttpError where error.status == 401:
                showAlert(message: "Incorrect email or password")

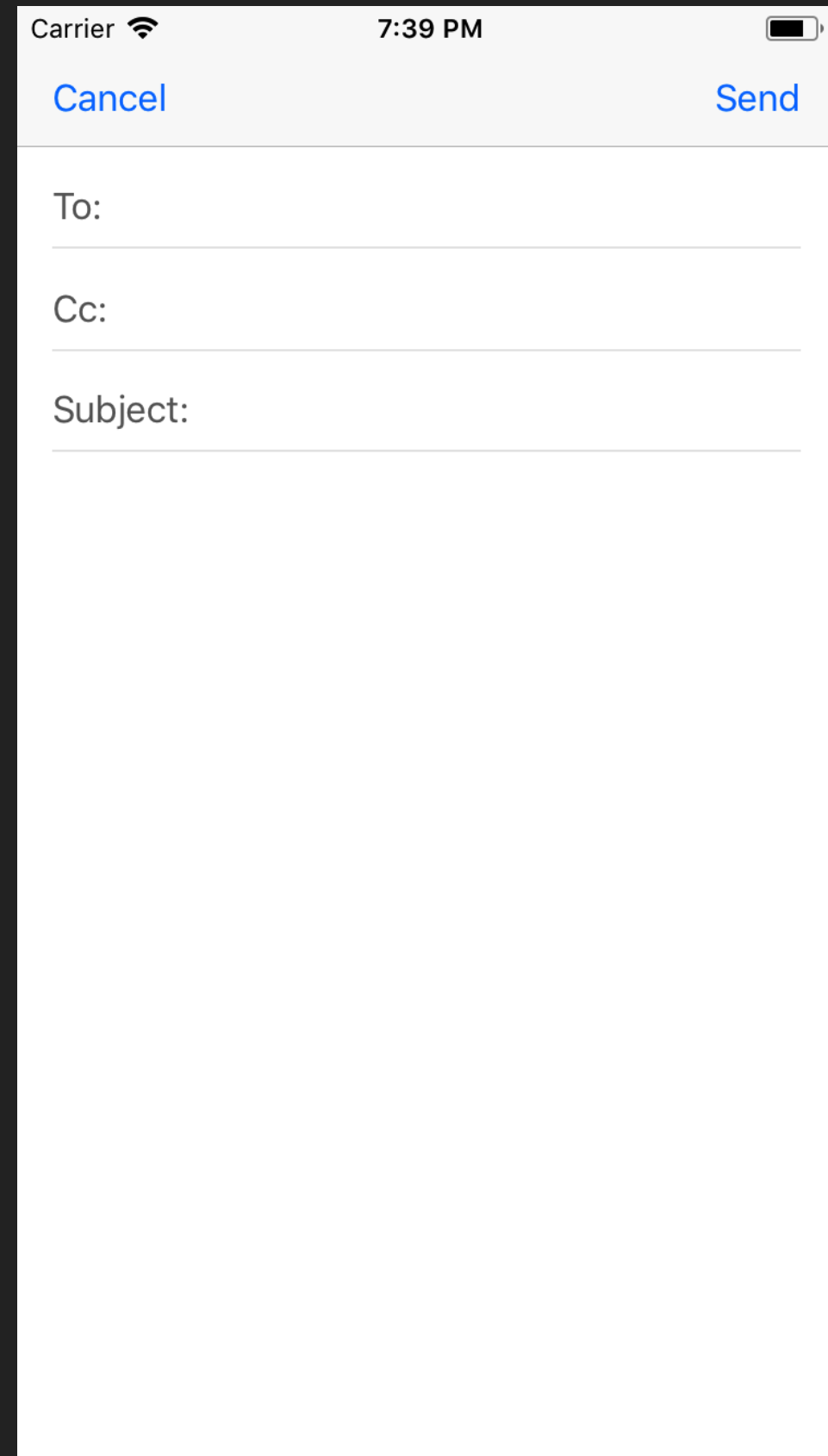
            case let error as NSError
                where error.domain == NSURLErrorDomain
                && error.code == NSURLErrorNotConnectedToInternet:
                showWarning("You appear to be offline. Please check your connection.")

            case let error as SignInError where error == .invalidEmail:
                showInvalidEmail()

            default:
                showError(message: "Sorry, there was a problem. Please try again")
            }
        }
    }
}
```



```
@IBAction func loginTapped(_ sender: Any) {
    api.login(email, password) { (response, error) in
        if let error = error {
            Logger.log(error)
            switch error {
            case let error as HttpError where error.status == 401:
                showAlert(message: "Incorrect email or password")
            case let error as NSError
                where error.domain == NSURLErrorDomain
                && error.code == NSURLErrorNotConnectedToInternet:
                showWarning("You appear to be offline. Please check your connection.")
            case let error as SignInError where error == .invalidEmail:
                showInvalidEmail()
            default:
                showError(message: "Sorry, there was a problem. Please try again")
            }
        }
    }
}
```

Add error handling
to a Compose
email screen



The image shows a screenshot of an iOS email compose screen. At the top, the status bar displays 'Carrier' with a signal icon, the time '7:39 PM', and a battery icon. Below the status bar, there is a header bar with a light gray background. On the left side of this bar is a blue 'Cancel' button, and on the right side is a blue 'Send' button. The main area of the screen is white and contains three text input fields. The first field is labeled 'To:' and has a horizontal line below it. The second field is labeled 'Cc:' and also has a horizontal line below it. The third field is labeled 'Subject:' and has a horizontal line below it. The rest of the screen is empty, suggesting a large text area for the email body is located below these fields.

Add error handling to a Compose email screen

Carrier  7:39 PM 

Cancel Send

To: natalie.sung@gmail.com

Cc:

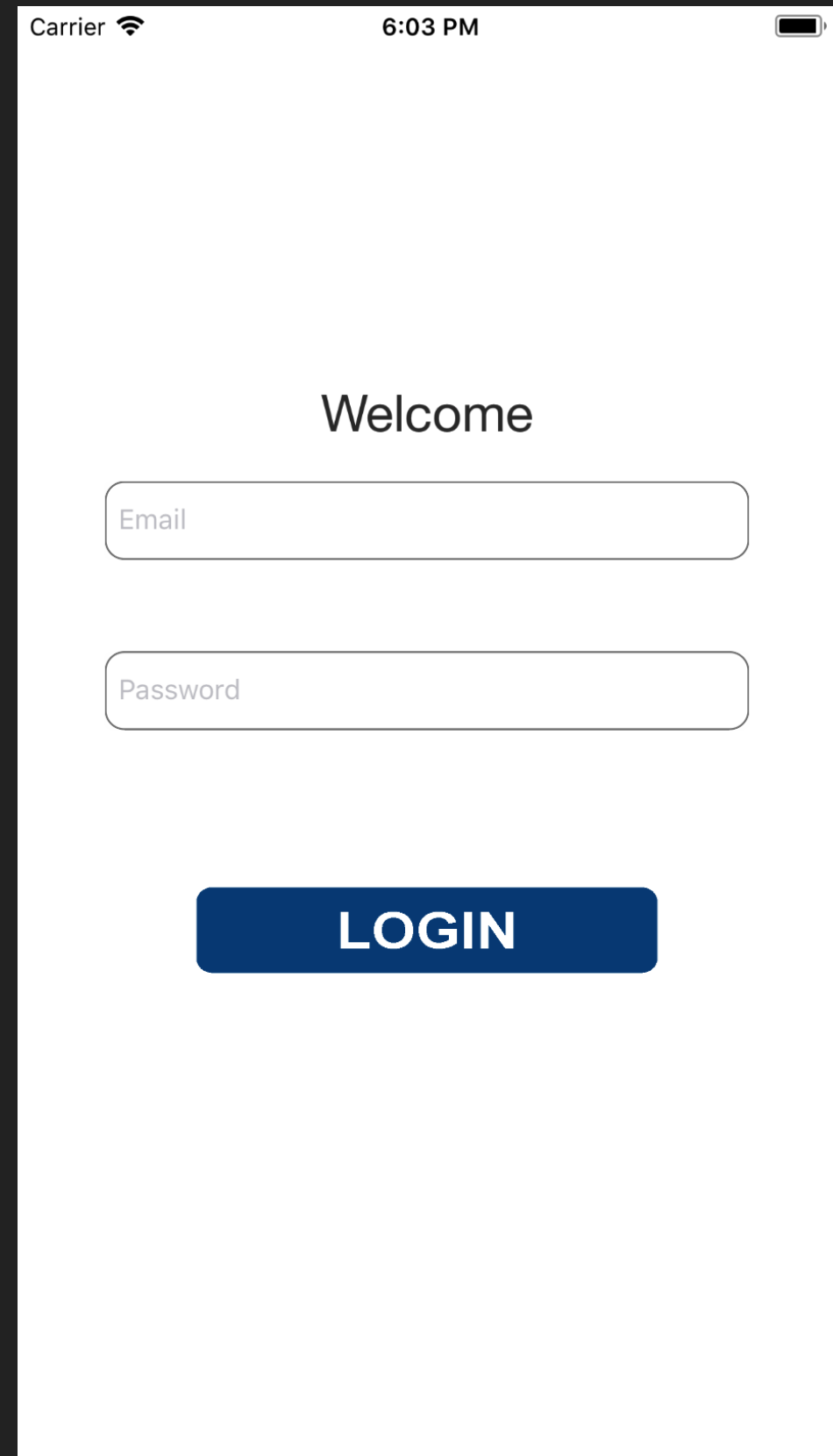
Subject: Issue

Dear Natalie,

I would like to ask you about a recent purchase of your product. I've having issues with...

Thanks,
Bob

Add error handling to a Compose email screen



A mobile app login screen mockup. At the top, the status bar shows 'Carrier' with a signal icon, '6:03 PM', and a battery icon. The main content area is white. It features the word 'Welcome' in a dark gray font. Below it are two rounded rectangular input fields: the first is labeled 'Email' and the second is labeled 'Password'. At the bottom is a dark blue button with the word 'LOGIN' in white capital letters.

Carrier 6:03 PM

Welcome

Email

Password

LOGIN

```
@IBAction func sendTapped(_ sender: Any) {  
    api.send(emailData) { (response, error) in  
        if let error = error {  
  
            switch error {  
            case let error as HttpError where error.status == 401:  
                showSignInScreen()  

```

```
    }  
    }  
    }  
}
```


Carrier 

6:59 PM



Cancel

Send

You appear to be offline. Please check your connection.

Cc:

Subject: Issue

Dear Natalie,

I would like to ask you about a recent purchase of your product. I've having issues with...

Thanks,
Bob

```
@IBAction func sendTapped(_ sender: Any) {
    api.send(emailData) { (response, error) in
        if let error = error {

            switch error {
            case let error as HttpError where error.status == 401:
                showSignInScreen()
            case let error as NSError
                where error.domain == NSURLErrorDomain
                && error.code == NSURLErrorNotConnectedToInternet:
                showWarning("You appear to be offline. Please check your connection.")
            }
        }
    }
}
```

Carrier



7:06 PM



Cancel

Send

Sorry, there was a problem. Please try again

Cc:

Subject: Issue

Dear Natalie,

I would like to ask you about a recent purchase of your product. I've having issues with...

Thanks,
Bob

```
@IBAction func sendTapped(_ sender: Any) {
    api.send(emailData) { (response, error) in
        if let error = error {

            switch error {
            case let error as HttpError where error.status == 401:
                showSignInScreen()
            case let error as NSError
                where error.domain == NSURLErrorDomain
                && error.code == NSURLErrorNotConnectedToInternet:
                showWarning("You appear to be offline. Please check your connection.")
            default:
                showError(message: "Sorry, there was a problem. Please try again")
            }
        }
    }
}
```

Carrier 

6:58 PM



Cancel

Send

To: natalie.sung@gmail.com

Cc:

Subject: Issue

Don't worry, a draft has been saved.

Dear Natalie,

I would like to ask you about a recent purchase of
your product. I've having issues with...

Thanks,
Bob

```
@IBAction func sendTapped(_ sender: Any) {
    api.send(emailData) { (response, error) in
        if let error = error {

            showMessage("Don't worry, a draft has been saved.")
            switch error {
            case let error as HttpError where error.status == 401:
                showSignInScreen()
            case let error as NSError
                where error.domain == NSURLErrorDomain
                && error.code == NSURLErrorNotConnectedToInternet:
                showWarning("You appear to be offline. Please check your connection.")
            default:
                showError(message: "Sorry, there was a problem. Please try again")
            }
        }
    }
}
```

```
@IBAction func sendTapped(_ sender: Any) {
    api.send(emailData) { (response, error) in
        if let error = error {
            Logger.log(error)
            showMessage("Don't worry, a draft has been saved.")
            switch error {
            case let error as HttpError where error.status == 401:
                showSignInScreen()
            case let error as NSError
                where error.domain == NSURLErrorDomain
                && error.code == NSURLErrorNotConnectedToInternet:
                showWarning("You appear to be offline. Please check your connection.")
            default:
                showError(message: "Sorry, there was a problem. Please try again")
            }
        }
    }
}
```



Welcome

LOGIN



Cancel

Send

To: natalie.sung@gmail.com

Cc:

Subject: Issue

Dear Natalie,

I would like to ask you about a recent purchase of your product. I've having issues with...

Thanks,
Bob

Carrier

6:06 PM

Sorry, there was a problem. Please try again

Welcome

unlucky.bob@gmail.com

.....

Carrier

6:08 PM

Welcome

unlucky.bobgmail.com

Invalid email

.....

LOGIN

Carrier

6:06 PM

Welcome

unlucky.bob@gmail.com

Incorrect email or password

OK

LOGIN

Carrier

6:59 PM

Cancel

Send

You appear to be offline. Please check your connection.

Cc:

Subject: Issue

Dear Natalie,

I would like to ask you about a recent purchase of your product. I've having issues with...

Thanks, Bob

Carrier

6:19 PM

Cancel

Send

Sorry, there was a problem. Please try again

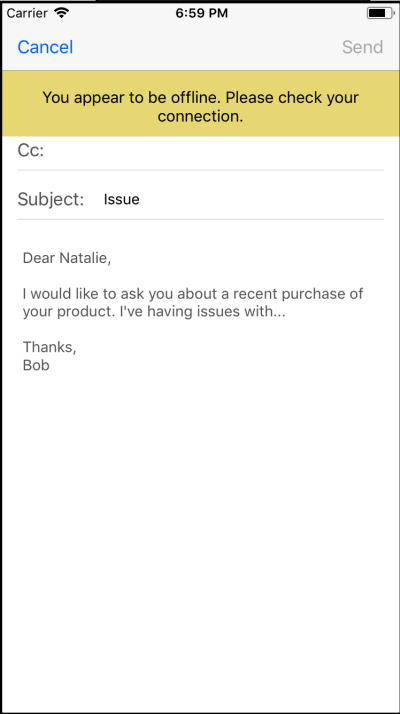
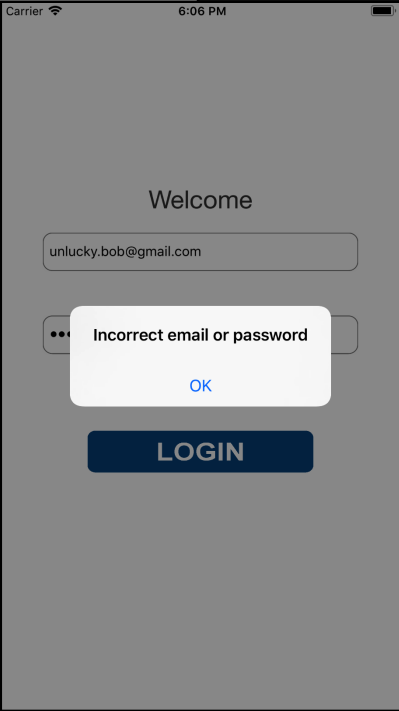
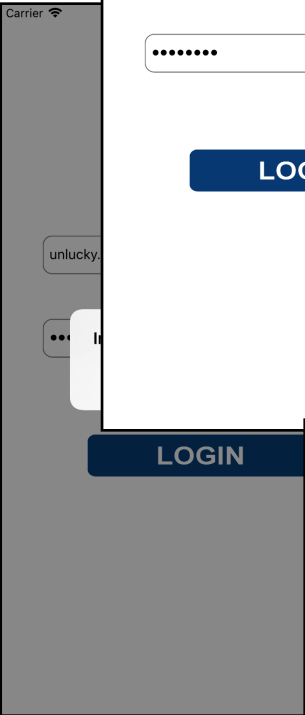
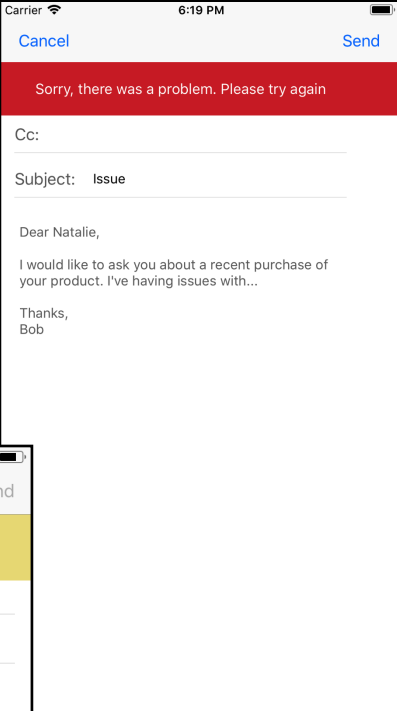
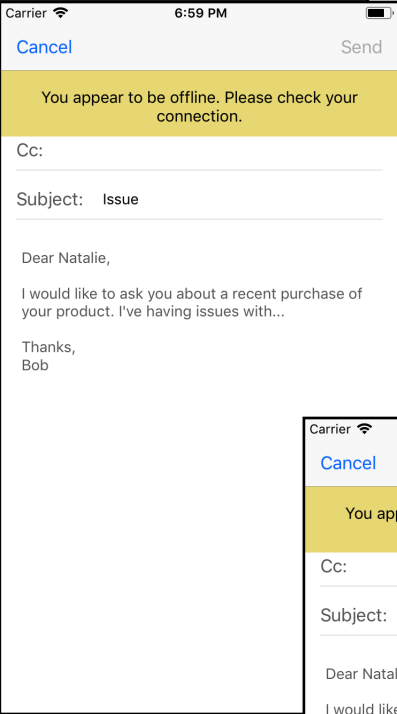
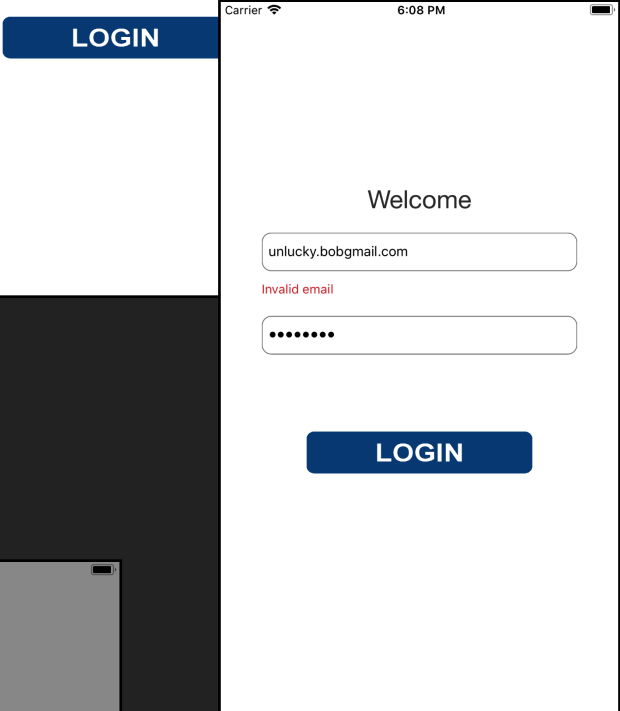
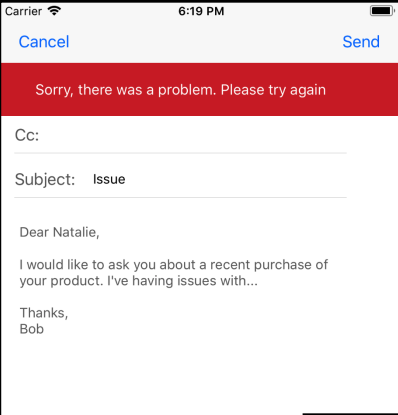
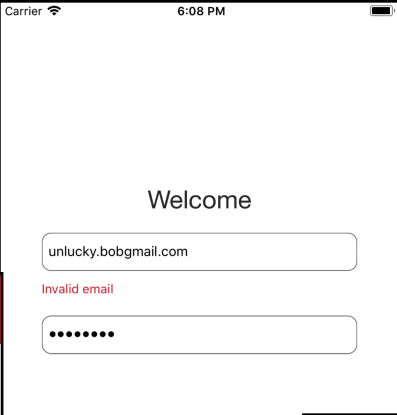
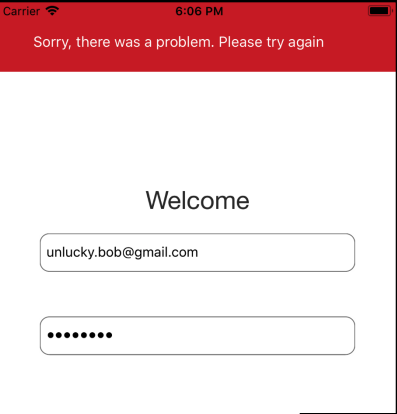
Cc:

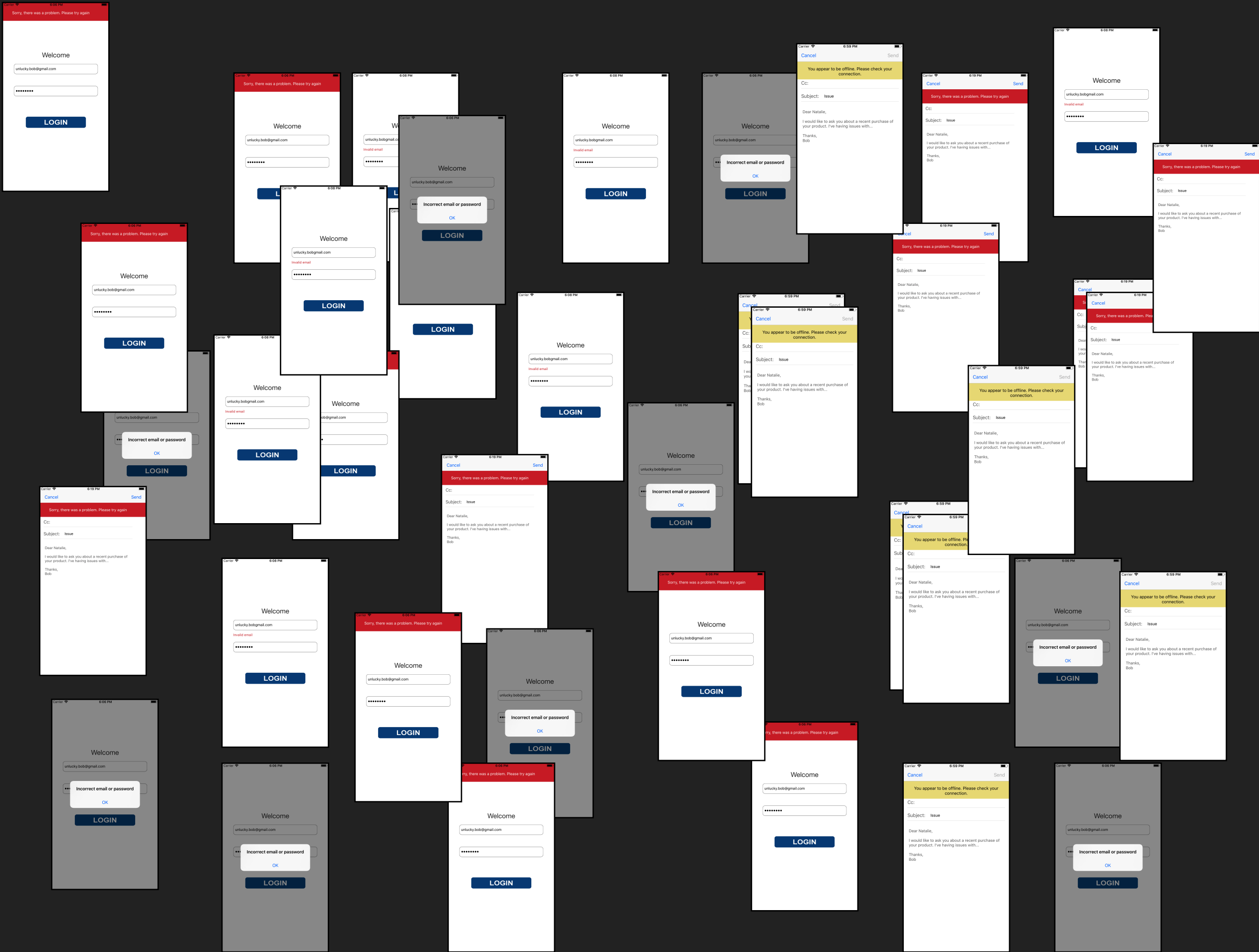
Subject: Issue

Dear Natalie,

I would like to ask you about a recent purchase of your product. I've having issues with...

Thanks, Bob







We end up with..

- ✗ Boilerplate
- ✗ Repetition
- ✗ No standard way of handling
- ✗ Cognitive overhead

Bad practices

Skip cases and use generic error messages

Bad practices

Skip cases and use generic error messages

Don't abstract and leave duplicate code everywhere

Bad practices

Skip cases and use generic error messages

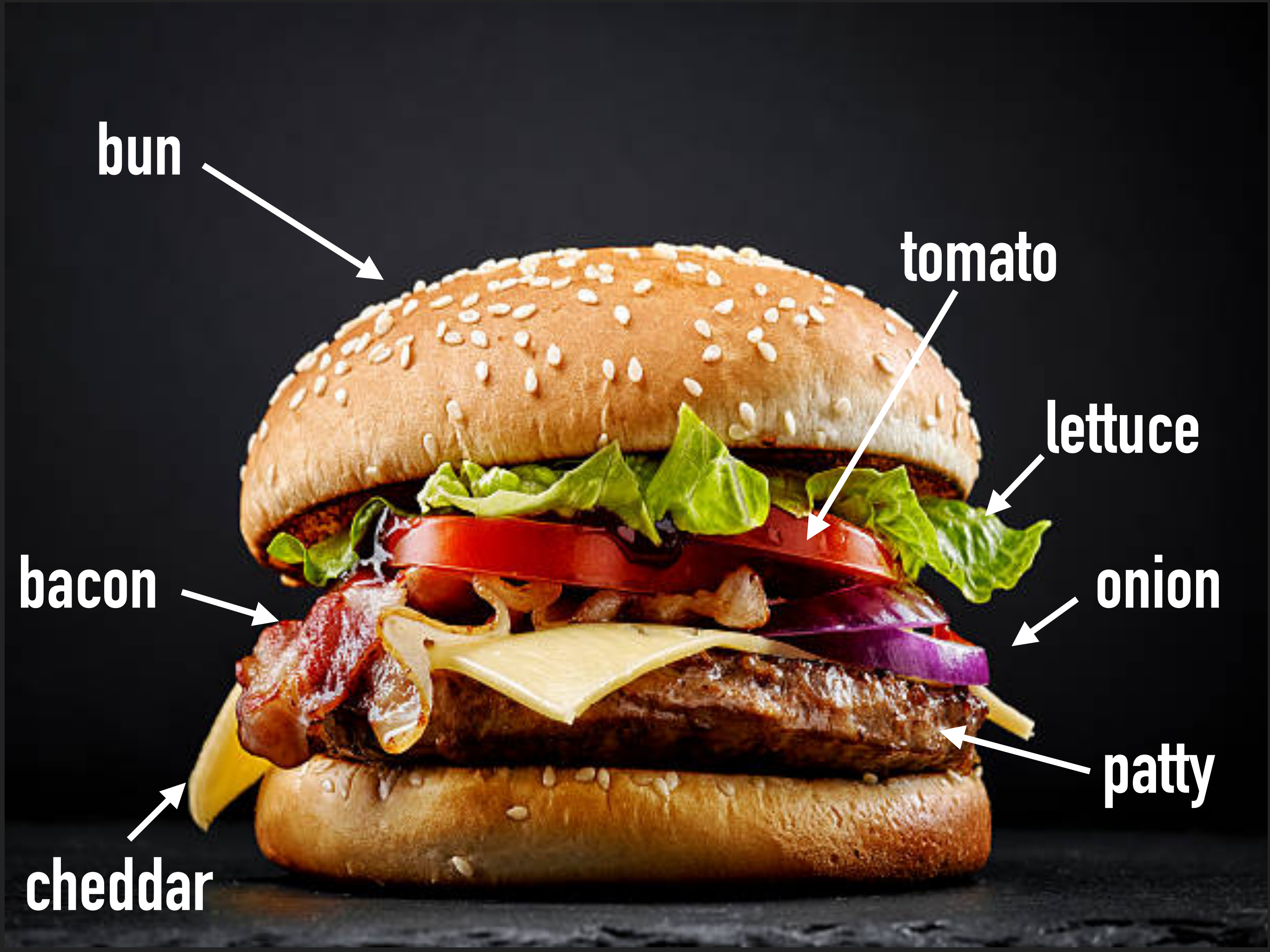
Don't abstract and leave duplicate code everywhere

Handle errors at the network layer 🤨

What would
we want ideally?

Make error handling
as easy as ordering a burger





bun

tomato

lettuce

onion

patty

bacon

cheddar

*"Can I have a burger with bun,
bacon, cheddar, tomato, lettuce,
onion and patty please?"*

- noone ever

Instead we say...

"The usual..."

or

"Cowboy burger with extra
bacon without the onions"

What we actually want

A set of default actions for common errors

What we actually want

A set of default actions for common errors

An easy way to customise these defaults

- Add new cases
- Override existing ones
- Add actions for unknown errors
- Add actions for all errors (logging)

ErrorHandler

A library that provides a
declarative fluent API for flexible
error handling

ErrorHandler

Basic API

```
public func on(matches: Error -> Bool, do action: @escaping  
ErrorAction) -> ErrorHandler
```

```
public func always(do action: @escaping ErrorAction) ->  
ErrorHandler
```

```
public func onNoMatch(do action: @escaping ErrorAction) ->  
ErrorHandler
```

```
public func handle(_ error: Error)
```

Basic API

```
public func on(matches: Error -> Bool, do action: @escaping  
ErrorAction) -> ErrorHandler
```

```
public func on(error: Error & Equatable, do action:  
@escaping ErrorAction) -> ErrorHandler
```

```
public func always(do action: @escaping ErrorAction) ->  
ErrorHandler
```

```
public func onNoMatch(do action: @escaping ErrorAction) ->  
ErrorHandler
```

```
public func handle(_ error: Error)
```

```
ErrorHandler()  
  .on(error1, do: actionA)  
  .on(error2, do: actionB)  
  .onNoMatch(do: actionC)  
  .always(do: log)  
  .handle(error)
```

Our strategy

Setup a default ErrorHandler once

Our strategy

Setup a default ErrorHandler once

Customize the default ErrorHandler when
needed based on the context

In many cases all we will need is...

```
if let error = error {  
    ErrorHandler.defaultHandler.handle(error)  
    return  
}
```



Back to our example


```
extension ErrorHandler {
  class var defaultHandler: ErrorHandler {
    return ErrorHandler()
  }
  .on(NSError(domain:NSURLErrorDomain, code: NSErrorNotConnectedToInternet),
    do: { (error) -> MatchingPolicy in
      showWarning("You appear to be offline. Please check your connection.")
      return .continueMatching
    })
  .on(httpStatus: 401, do: { (error) -> MatchingPolicy in
    showSignInScreen()
    return .continueMatching
  })
  .always(do: { (error) -> MatchingPolicy in
    Logger.log(error)
    return .continueMatching
  })
  .onNoMatch(do: { (error) -> MatchingPolicy in
    showError(message: "Sorry, there was a problem. Please try again")
    return .continueMatching
  })
}
}
```

Carrier



6:03 PM



Welcome

LOGIN

```
@IBAction func loginTapped(_ sender: Any) {
    api.login(email, password) { (response, error) in
        if let error = error {
            ErrorHandler.defaultHandler
                .on(httpStatus: 401, do: { (_) -> MatchingPolicy in
                    showAlert(message: "Incorrect email or password")
                    return .stopMatching
                })
                .on(SignInError.invalidEmail, do: { [weak self] (_) -> MatchingPolicy in
                    self?.showInvalidEmail()
                    return .stopMatching
                })
                .handle(error)
        }
    }
}
```

Carrier



7:39 PM



Cancel

Send

To:

Cc:

Subject:

```
@IBAction func sendTapped(_ sender: Any) {
    api.send(emailData) { (response, error) in
        if let error = error {
            ErrorHandler.defaultHandler
                .always(do: { (error) -> MatchingPolicy in
                    self.showMessage("Don't worry, a draft has been saved.")
                    return .continueMatching
                })
                .handle(error)
        }
    }
}
```

We get to..

- ✓ Avoid the boilerplate
- ✓ Avoid the repetition
- ✓ Have a standard way of handling
- ✓ Eliminate cognitive overhead



Additional features

Error Matchers

ErrorMatcher

```
public protocol ErrorMatcher {  
    func matches(_ error: Error) -> Bool  
}
```

```
handler  
    .on(matcher) { (error) -> MatchingPolicy in  
        showAlert(message: "It's a match!")  
        return .stopMatching  
    }.handle(error)
```

ErrorMatcher && ErrorMatcher
 ||

```
let notConnectedMatcher = NSErrorMatcher(domain:  
NSURLErrorDomain, code: NSErrorNotConnectedToInternet)
```

```
let connectionLostMatcher = NSErrorMatcher(domain:  
NSURLErrorDomain, code: NSErrorNetworkConnectionLost)
```

```
let offlineMatcher = notConnectedMatcher || connectionLostMatcher
```

Additional features

Error Matchers

Tags

Tag

```
// At the setup point of our default handler
return ErrorHandler()
    // other setup code
    .tag(notConnectedMatcher, with: "offline")
    .tag(connectionLostMatcher, with: "offline")
```

```
// At the point where we handle an error e.g. any controller
ErrorHandler()
    .on(tag: "offline", do: { (error) -> MatchingPolicy in
        showError("You appear to be offline. Please check
        you connection.")
        return .continueMatching
    }).handle(error)
```

Additional features

Error Matchers

Tags

Extension for http status error handling
(Alamofire out of the box)

Conclusions

Error handling is an integral part of a good UX

It can get cumbersome if you want to do it right

You can minimize the friction with the right abstractions

Even if there are many error cases,
it's nothing you can't `.handle()`

Thanks!

<https://github.com/Workable/swift-error-handler.git>

<https://github.com/Workable/java-error-handler.git>