A DEEP DIVE INTO TESTING YOUR VAPOR APPLICATIONS

TIM CONDON

BBC
Broken Hands

INTRODUCTION

- Varied background, currently at the BBC
- @0xTim - Twitter/Slack/Github
- @brokenhandsio - Twitter/Github
- Created SteamPress, Vapor Security Headers, Vapor OAuth

AGENDA

- Testing Recap
- Testing and Design Strategies
- Testing Vapor
- Testing Views
- Testing Authentication

CAVEATS!

- From my experience
- Borrow and adapt for your use cases
- A lot of my code (SteamPress especially) doesn't follow everything I talk about today!
- (The views are my own!)

TESTING RECAP

WHY UNIT TEST

- It makes sure things work!
- Helps you design your code
- Confidence in refactoring
- Write a test for every bug - no more regressions!
- Isn’t it a waste of time?
- Speed!
- Swift on Linux

TESTING ON SWIFT - LINUX RECAP

- No Objective-C Runtime
- Defines Test Cases in `LinuxMain.swift`
- Each `XCTestCase` class needs an `allTests` dictionary
- See templates/SteamPress/Broken Hands packages for examples
- Run tests on Docker

TESTING IN SWIFT - LINUX HELPER

```
// Courtesy of https://oleb.net/blog/2017/03/keeping-xctest-in-sync/
func testLinuxTestSuiteIncludesAllTests() {
    #if os(macOS) || os(iOS) || os(tvOS) || os(watchOS)
        let thisClass = type(of: self)
        let linuxCount = thisClass.allTests.count
        let darwinCount = Int(thisClass.defaultTestSuite().testCaseCount)
        XCTAssertEqual(linuxCount, darwinCount, "\(darwinCount - linuxCount)
tests are missing from allTests")
    #endif
}
```

TESTING SWIFT - TESTING ON LINUX

- Use Docker either manually and part of your CI

```
FROM swift:3.1

WORKDIR /package

COPY . ./

RUN swift package --enable-prefetching fetch
RUN swift package clean
CMD swift test
```

- `docker build --tag vapor-oauth . && docker run --rm vapor-oauth`

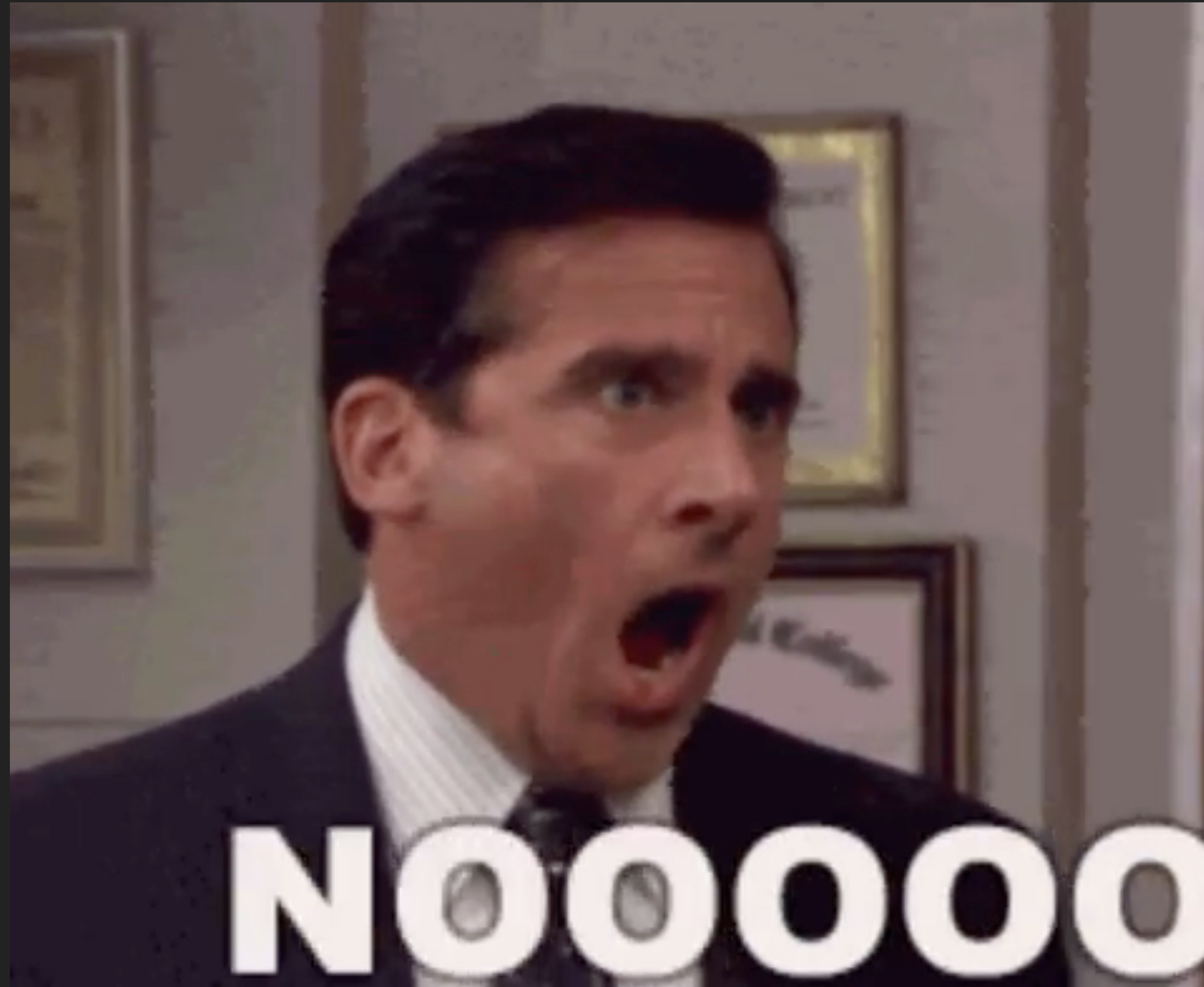
TESTING AND DESIGN STRATEGIES

WHAT IS A UNIT TEST

- Used to be thought of as a test of a method
- That's brittle
- Instead, tests should describe the behaviour of the system e.g. given I submit a POST with missing values, I get a 400 Bad Request response
- Self documenting
- Don't be afraid of big test names!

OCMOCK EXAMPLE

```
GCKRemoteMediaClient *sut = [[GCKRemoteMediaClient alloc] init];
self.mockRemoteClient = [OCMockObject partialMockForObject:sut];
```


NOOOOO

THE PROBLEM WITH MOCKING FRAMEWORKS

- Reliance on frameworks
- May not be testing what you think
- Encourages bad behaviours
- If you can only test by using a mock class, you're doing it wrong!

TESTING IN SWIFT

- Embrace the protocol!
- Protocol Orientated Programming
- https://developer.apple.com/videos/play/wwdc2015/408/
- No more subclassing!

TESTING IN SWIFT

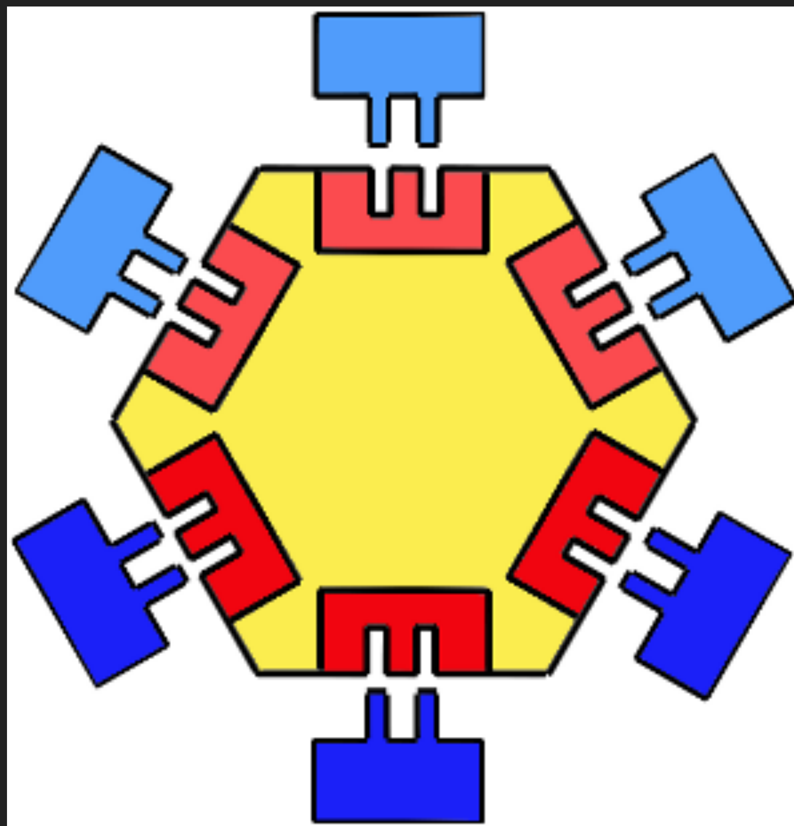
```
class CapturingClient: ClientProtocol {

    init() {}
    required init(hostname: String, port: Sockets.Port,
securityLayer: SecurityLayer, proxy: Proxy?) throws {}

    private(set) var capturedRequest: Request?
    func respond(to request: Request) throws -> Response {
        capturedRequest = request
        return "Test".makeResponse()
    }
}
```

TESTING IN SWIFT

```
func testEmailSentWhenUserSuccessfullyRegistered() throws {
    let emailAddress = "han.solo@therebelalliance.com"
    let registrationRequest = Request(method: .post, uri: "/users/registration")
    var registrationJSON = JSON()
    try registrationJSON.set("first_name", "Han")
    try registrationJSON.set("last_name", "Solo")
    try registrationJSON.set("email", emailAddress)
    registrationRequest.json = registrationJSON

    _ = try drop.respond(to: registrationRequest)

    guard let json = capturingClient.capturedRequest?.json else {
        XCTFail()
        return
    }

    XCTAssertEqual(capturingClient.capturedRequest?.uri.description, "https://
notifications.api.brokenhands.io")
    XCTAssertEqual(json["email"]?.string, emailAddress)
    XCTAssertEqual(json["notification_type"]?.string, "registration_email")
}
```

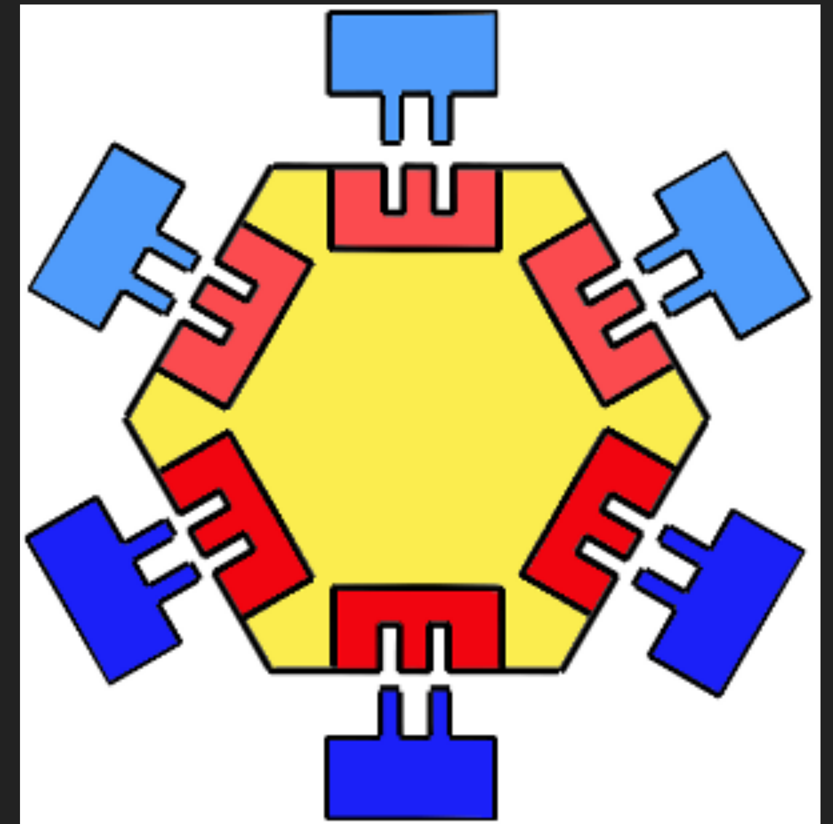
HEXAGONAL ARCHITECTURE

- Otherwise known as Ports and Adapters
- http://alistair.cockburn.us/Hexagonal+architecture



PORTS AND ADAPTERS

- Components are swappable - e.g. database, client
- Interfaces are optional - e.g. JSON, web, CLI
- Business logic is your core
- Use dependency injection and protocols



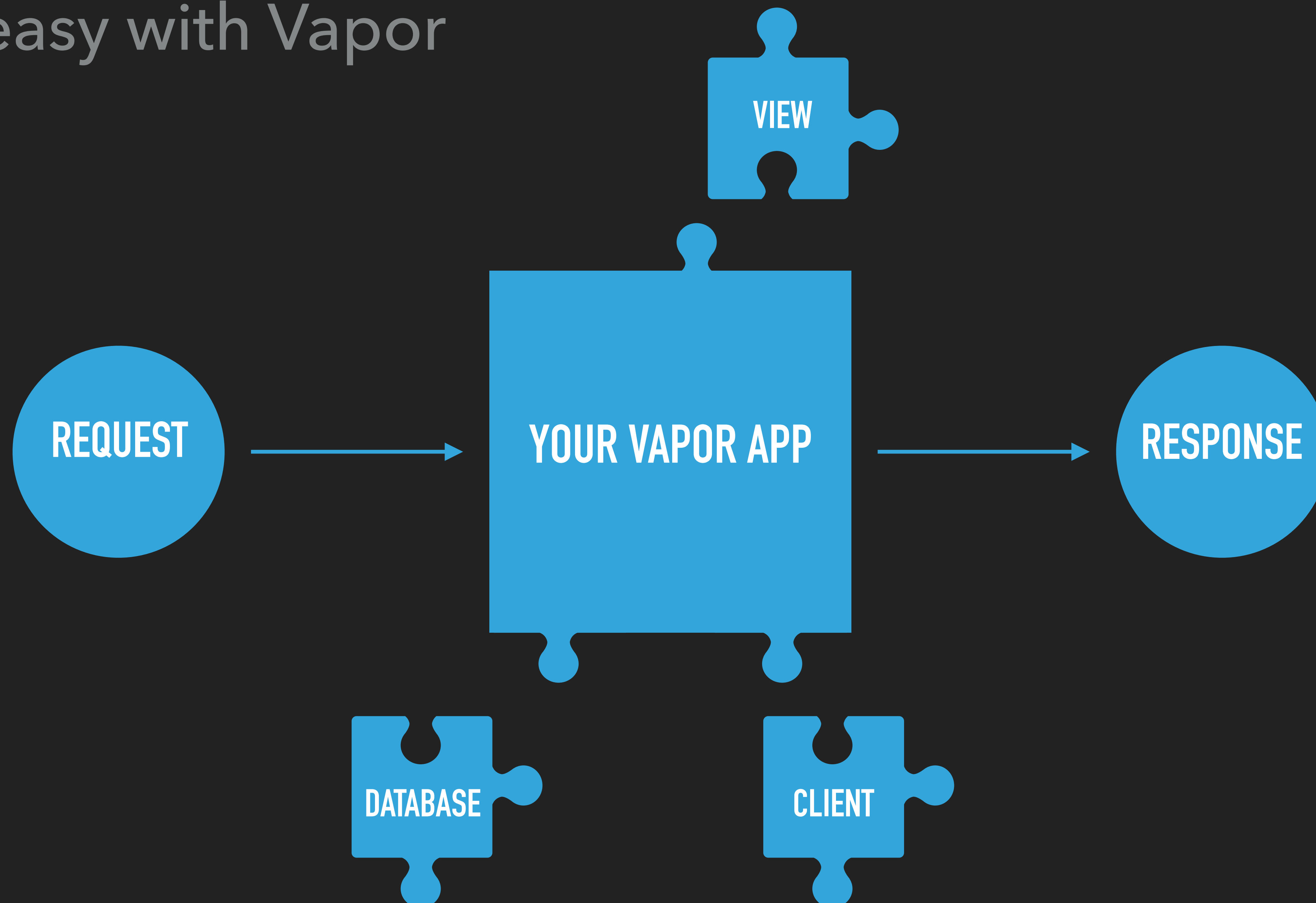
TEST DRIVEN DEVELOPMENT

- Origins in XP
- Tests design the system
- Helps break down code
- If you can’t test your code easily then it isn’t designed well
- Only do the minimum to make the test pass
- If changing a line doesn’t fail a test, it is deleted
- Red-Green-Refactor

TESTING VAPOR

PUTTING IT ALL TOGETHER

- Very easy with Vapor



VAPOR OAUTH EXAMPLE - FIRST TEST

```
func testThatAuthorizationCodeRequestRedirectsToLoginPage() throws {
    let config = Config([:])
    try config.addProvider(OAuth.Provider.self)
    let drop = try Droplet(config)

    let requestQuery =
"response_type=code&client_id=1234567890&redirect_uri=https://api.whats.io/
callback&scope=create+view&state=xcoivjuywkdkhvusuye3kch"
    let codeRequest = Request(method: .get, uri: "/auth?\(requestQuery)")
    let codeResponse = try drop.respond(to: codeRequest)

    XCTAssertEqual(codeResponse.status, .seeOther)
    XCTAssertEqual(codeResponse.headers[.location], "login/")
}
```

VAPOR OAUTH EXAMPLE - FIRST CODE

```
struct OAuth2Provider {
    func addRoutes(to router: RouteBuilder) {
        router.get("auth", handler: authHandler)
    }

    func authHandler(request: Request) throws -> ResponseRepresentable {
        return Response(redirect: "login/")
    }
}
```

TESTING ROUTES

- Different options:
  - Test methods individually
  - Test handlers individually
  - Test app end-to-end

TESTING END-TO-END

1. Set up environment (database, cookies, sessions) and request
2. Get response
3. Assert on response

Also known as Arrange, Act, Assert

SETTING UP YOUR TESTS - SETUP METHOD

```
var drop: Droplet!
let fakeClientGetter = FakeClientGetter()
let fakeUserManager = FakeUserManager()
let fakeTokenManager = FakeTokenManager()
let scopeRetriever = FakeScopeRetriever()
let capturingLogger = CapturingLogger()
let testClientID = "ABCDEF"
let testClientSecret = "01234567890"
let testUsername = "testUser"
let testPassword = "testPassword"
let testUserID = "ABCD-FJUH-31232"
let accessToken = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
let refreshToken = "ABCDEFGHIJLMNOP1234567890"
let scope1 = "email"
let scope2 = "create"
let scope3 = "edit"

override func setUp() {
    drop = try! TestDataBuilder.getOAuthDroplet(tokenManager: fakeTokenManager, clientRetriever: fakeClientGetter,
userManager: fakeUserManager, scopeRetriever: scopeRetriever, log: capturingLogger)

    let testClient = OAuthClient(clientID: testClientID, redirectURIs: [], clientSecret: testClientSecret,
validScopes: [scope1, scope2], firstParty: true)
    fakeClientGetter.validClients[testClientID] = testClient
    fakeUserManager.testUser = testUsername
    fakeUserManager.testPassword = testPassword
    fakeUserManager.testUserID = testUserID
    fakeTokenManager.accessTokenToReturn = accessToken
    fakeTokenManager.refreshTokenToReturn = refreshToken
    scopeRetriever.scopes = [scope1, scope2, scope3]
}
```

SETTING UP YOUR TESTS - HELPER FUNCTIONS

```
static func getOAuthDroplet(codeManager: CodeManager = StubCodeManager(), tokenManager: TokenManager
= StubTokenManager(), clientRetriever: ClientRetriever = FakeClientGetter(), userManager:
UserManager = StubUserManager(), authorizeHandler: AuthorizeHandler = CapturingAuthoriseHandler(),
scopeRetriever: ScopeRetriever = FakeScopeRetriever(), environment: Environment? = nil, log:
CapturingLogger? = nil) throws -> Droplet {
    var config = Config([:])

    if let environment = environment {
        config.environment = environment
    }

    if let log = log {
        config.addConfigurable(log: { (_) -> (CapturingLogger) in
            return log
        }, name: "capturing-log")
        try config.set("droplet.log", "capturing-log")
    }

    let provider = OAuth.Provider(codeManager: codeManager, tokenManager: tokenManager,
clientRetriever: clientRetriever, authorizeHandler: authorizeHandler, userManager: userManager,
scopeRetriever: scopeRetriever)

    try config.addProvider(provider)
    return try Droplet(config)
}
```

SETTING UP YOUR TESTS - TEST SPECIFICS

```
func testCorrectErrorWhenClientDoesNotAuthenticate() throws {
    let clientID = "ABCDEF"
    let clientWithSecret = OAuthClient(clientID: clientID, redirectURIs:
["https://api.brokenhands.io/callback"], clientSecret: "1234567890ABCD")
    fakeClientGetter.validClients[clientID] = clientWithSecret

    ...
}
```

GET RESPONSE

```
func testCorrectErrorWhenClientDoesNotAuthenticate() throws {
    ...

    let response = try getPasswordResponse(clientID: clientID, clientSecret: 
"incorrectPassword")

    ...
}
```

GET RESPONSE - HELPERS

```
func getPasswordResponse(grantType: String? = "password", username: String? = "testUser", password: String? =
"testPassword", clientID: String? = "ABCDEF", clientSecret: String? = "01234567890", scope: String? = nil) throws
-> Response {
    return try TestDataBuilder.getTokenRequestResponse(with: drop, grantType: grantType, clientID: clientID,
clientSecret: clientSecret, scope: scope, username: username, password: password)
}

static func getTokenRequestResponse(with drop: Droplet, grantType: String?, clientID: String?, clientSecret:
String?, redirectURI: String? = nil, code: String? = nil, scope: String? = nil, username: String? = nil, password:
String? = nil, refreshToken: String? = nil) throws -> Response {
    let request = Request(method: .post, uri: "/oauth/token/")

    var requestData = Node([:], in: nil)

    if let grantType = grantType {
        try requestData.set("grant_type", grantType)
    }

    ...

    if let refreshToken = refreshToken {
        try requestData.set("refresh_token", refreshToken)
    }

    request.formURLEncoded = requestData

    let response = try drop.respond(to: request)

    return response
}
```

ASSERT ON RESPONSE

```
func testCorrectErrorWhenClientDoesNotAuthenticate() throws {

    ...

    guard let responseJSON = response.json else {
        XCTFail()
        return
    }

    XCTAssertEqual(response.status, .unauthorized)
    XCTAssertEqual(responseJSON["error"]?.string, "invalid_client")
    XCTAssertEqual(responseJSON["error_description"], "Request had invalid
client credentials")
    XCTAssertEqual(response.headers[.cacheControl], "no-store")
    XCTAssertEqual(response.headers[.pragma], "no-cache")
}
```

ALTERNATIVES

```
func testHello() throws {
    try drop
        .testResponse(to: .get, at: "hello")
        .assertStatus(is: .ok)
        .assertJSON("hello", equals: "world")
}
```

- Requires @testable import

TESTING VIEWS

TESTING VIEWS

- JSON/APIs are easy
- Views change regularly
- Think about how difficult UI Testing is
- Introduce Presenters

PRESENTERS

- From Model View Presenter design pattern
- Split out difficult to test views and business logic
- Abstraction layer to allow testing
- Thin view with no real logic in (Leaf)
- Presenter cares about displaying data:
  - Order of lists
  - What data to display for a model
- Business logic stays in core app

PRESENTER

VIEW LAYER

PRESENTER LAYER

CORE APP

TESTING VIEW

- Controller takes the presenter as a parameter
- Protocol with all the different views to display and what data they need
- In SteamPress this is the `ViewFactory`
- Use a `CapturingViewFactory` for test
- (I know this isn't strict Presenterisation!)

TEST UP TO PRESENTER LAYER

```
func testBlogPostRetrievedCorrectlyFromSlugUrl() throws {
    try setupDrop()
    _ = try drop.respond(to: blogPostRequest)

    XCTAssertEqual(viewFactory.blogPost?.title, post.title)
    XCTAssertEqual(viewFactory.blogPost?.contents, post.contents)
    XCTAssertEqual(viewFactory.blogPostAuthor?.name, user.name)
    XCTAssertEqual(viewFactory.blogPostAuthor?.username, user.username)
}
```

TESTING THE PRESENTER

```
class CapturingViewRenderer: ViewRenderer {
    var shouldCache = false

    private(set) var capturedContext: Node? = nil
    private(set) var leafPath: String? = nil
    func make(_ path: String, _ context: Node) throws -> View {
        self.capturedContext = context
        self.leafPath = path
        return View(data: "Test".makeBytes())
    }
}

func testParametersAreSetCorrectlyOnAllAuthorsPage() throws {
    let user1 = TestDataBuilder.anyUser()
    try user1.save()
    let user2 = TestDataBuilder.anyUser(name: "Han", username: "han")
    try user2.save()
    let authors = [user1, user2]
    _ = try viewFactory.allAuthorsView(uri: authorsURI, allAuthors: authors, user: user1)

    XCTAssertEqual(viewRenderer.capturedContext?["authors"]?.array?.count, 2)
    XCTAssertEqual((viewRenderer.capturedContext?["authors"]?.array?.first)?["name"], "Luke")
    XCTAssertEqual((viewRenderer.capturedContext?["authors"]?.array?[1])?["name"], "Han")
    XCTAssertEqual(viewRenderer.capturedContext?["uri"]?.string, "https://test.com:443/authors/")
    XCTAssertEqual(viewRenderer.capturedContext?["site_twitter_handle"]?.string, siteTwitterHandle)
    XCTAssertEqual(viewRenderer.capturedContext?["disqus_name"]?.string, disqusName)
    XCTAssertEqual(viewRenderer.capturedContext?["user"]?["name"]?.string, "Luke")
    XCTAssertEqual(viewRenderer.leafPath, "blog/authors")
}
```

TESTING AUTHENTICATION

TESTING AUTHENTICATION

- Want to ensure protected routes can't be accessed by anyone
- Make sure our login works!
- Don't want to log in every request for a test, grab cookie, save it then send with each request
  - Hashing is slow
- Lots of different approaches

TEST REAL LOGIN

- Make sure an actual login works
  - Cookies
  - BCrypt
  - Sessions
  - Logout

TEST REAL LOGIN

```
func testLogin() throws {
    let hashedPassword = try BlogUser.passwordHasher.make("password")
    let newUser = TestDataBuilder.anyUser()
    newUser.password = hashedPassword
    try newUser.save()

    let loginJson = JSON(try Node(node: [
            "inputUsername": newUser.username,
            "inputPassword": "password"
        ]))
    let loginRequest = Request(method: .post, uri: "/blog/admin/login/")
    loginRequest.json = loginJson
    let loginResponse = try drop.respond(to: loginRequest)

    XCTAssertEqual(loginResponse.status, .seeOther)
    XCTAssertEqual(loginResponse.headers[HeaderKey.location], "/blog/admin/")
    XCTAssertNotNil(loginResponse.headers[HeaderKey.setCookie])

    let rawCookie = loginResponse.headers[HeaderKey.setCookie]
    let sessionCookie = try Cookie(bytes: rawCookie?.bytes ?? [])

    let adminRequest = Request(method: .get, uri: "/blog/admin/")
    adminRequest.cookies.insert(sessionCookie)
    let adminResponse = try drop.respond(to: adminRequest)

    XCTAssertEqual(adminResponse.status, .ok)

    let logoutRequest = Request(method: .get, uri: "/blog/admin/logout/")
    logoutRequest.cookies.insert(sessionCookie)
    let logoutResponse = try drop.respond(to: logoutRequest)

    XCTAssertEqual(logoutResponse.status, .seeOther)
    XCTAssertEqual(logoutResponse.headers[HeaderKey.location], "/blog/")

    let secondAdminRequest = Request(method: .get, uri: "/blog/admin/")
    secondAdminRequest.cookies.insert(sessionCookie)
    let loggedOutAdminResponse = try drop.respond(to: secondAdminRequest)

    XCTAssertEqual(loggedOutAdminResponse.status, .seeOther)
    XCTAssertEqual(loggedOutAdminResponse.headers[HeaderKey.location], "/blog/admin/login/?loginRequired")
}
```

FAKING A LOGIN

- Lots of different options:
  - Inject into storage
  - Inject into sessions
  - Full workflow
- I’m still undecided!

FAKING A LOGIN

```
private func createLoggedInRequest(method: HTTP.Method, path: String, for user: BlogUser? = nil)
throws -> Request {
    let uri = "/blog/admin/\(path)/"
    let request = Request(method: method, uri: uri)

    let authAuthenticatedKey = "auth-authenticated"

    if let user = user {
        request.storage[authAuthenticatedKey] = user
    }
    else {
        let testUser = TestDataBuilder.anyUser()
        try testUser.save()
        request.storage[authAuthenticatedKey] = testUser
    }

    return request
}
```

FAKE THE HASHER

- Hashing takes time (it should!)
- Tests should be quick
- Switch the hashing algorithm for tests
- Still test the real one!
- Can either use config or manually set it

QUESTIONS