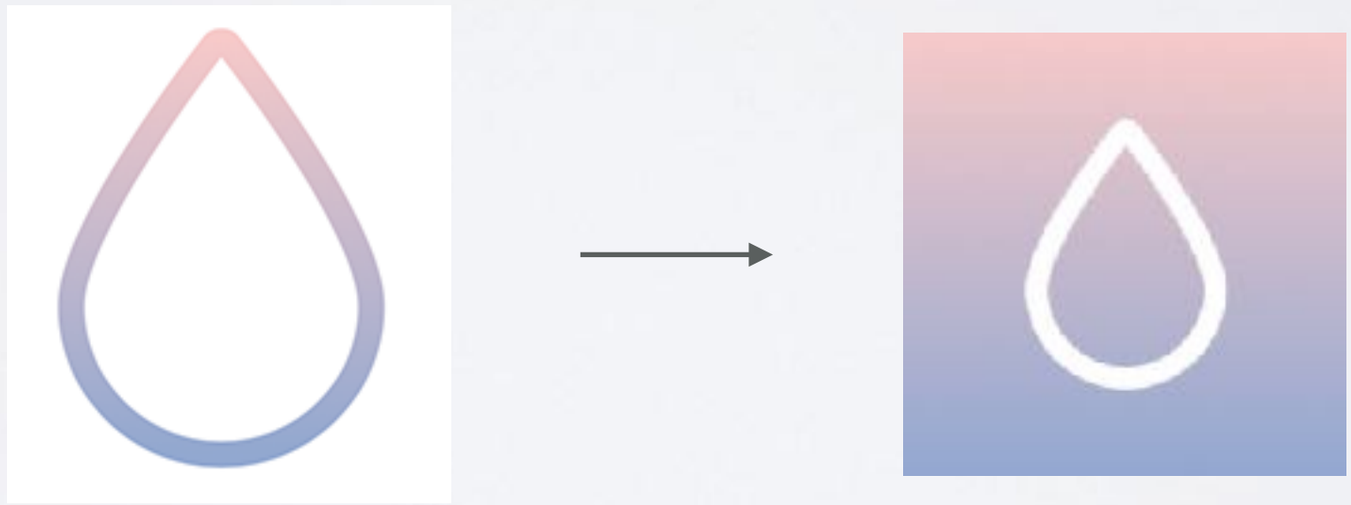
MIGRATING STEAMPRESS FROM VAPOR 1 TO VAPOR 2



INTRODUCTION

- Varied background - currently at BBC as a Mobile Developer
- Twitter, Github, Slack - @0xTim
- Twitter, Github - @brokenhandsio

WHY STEAMPRESS

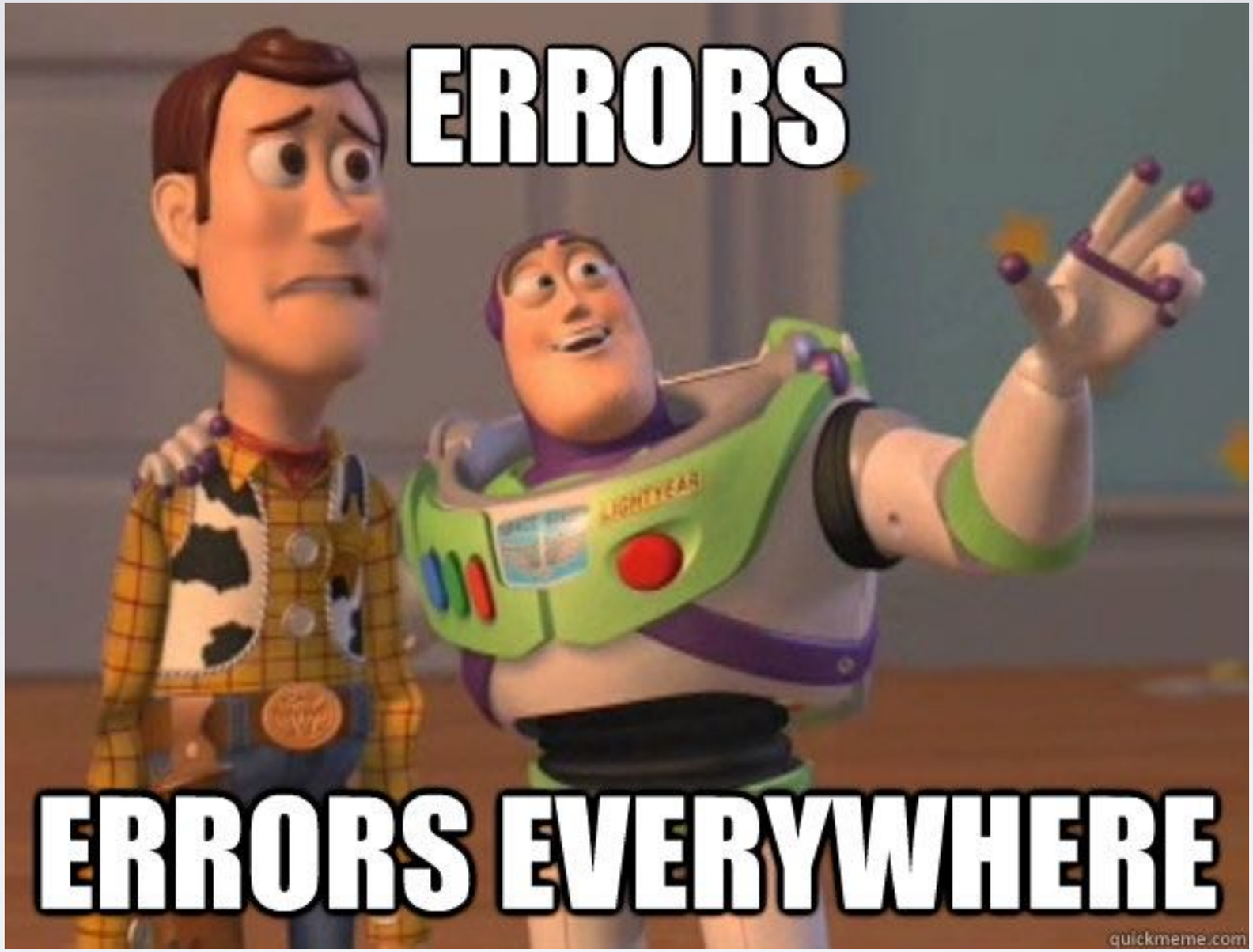
- Decided on Vapor at Christmas
- Started to port website
- Wanted a blog!

MIGRATING TO VAPOR 2

1. Getting Started
2. Configuration
3. Fluent Models
4. Authentication
5. Routing
6. Leaf and Views
7. Testing

GETTING STARTED

- You will need Swift 3.1
- Download the latest Vapor toolbox:
  `brew tap vapor/homebrew-tap`
  `brew install vapor`
- Migrate dependencies
- Edit Package.swift
  `.Package(url: "https://github.com/vapor/vapor.git", majorVersion: 2)`
- `vapor update`


ERRORS
ERRORS EVERYWHERE
quickmeme.com

CONFIGURATION

- Set middleware, providers on `Configuration`

- Droplet's properties are all now `lets`

FROM THIS

```
drop.middleware.append(securityHeaders)
drop.middleware.insert(abortMiddleware, at: 0)
try drop.addProvider(SteamPress.Provider.self)
```

TO THIS

```
config.addConfigurable(middleware:
securityHeaders.builder(), name: "security-headers")
config.addConfigurable(middleware:
BlogErrorMiddleware.init, name: "blog-error")

try config.addProvider(SteamPress.Provider.self)
try config.addProvider(LeafProvider.Provider.self)
try config.addProvider(FluentProvider.Provider.self)
```

droplet.json

```
{
    "server": "engine",
    "client": "engine",
    "console": "terminal",
    "log": "console",
    "hash": "crypto",
    "cipher": "crypto",
    "view": "leaf",
    "middleware": [
        "security-headers",
        "blog-error",
        "date",
        "file",
        "steampress-sessions",
        "blog-persist"
    ],
    "commands": [
        "prepare"
    ]
}
```

FLUENT MODELS

- `Storage` replaces `id`, `exists` and takes all responsibility

- **`Row`** is now responsible for saving and getting from the database

- Relations are simplified

FROM THIS

```
required public init(node: Node, in context: Context) throws {
   id = try node.extract(“id")
   title = try node.extract(“title")
   contents = try node.extract("contents")
   author = try node.extract("bloguser_id")
   slugUrl = try node.extract("slug_url")
   published = try node.extract("published")
   let createdTime: Double = try node.extract("created")
   let lastEditedTime: Double? = try? node.extract("last_edited")

   created = Date(timeIntervalSince1970: createdTime)

   if let lastEditedTime = lastEditedTime {
      lastEdited = Date(timeIntervalSince1970: lastEditedTime)
   }
}
```

TO THIS

```
public required init(row: Row) throws {
   title = try row.get(Properties.title.rawValue)
   contents = try row.get(Properties.contents.rawValue)
   author = try row.get(BlogUser.foreignIdKey)
   slugUrl = try row.get(Properties.slugUrl.rawValue)
   published = try row.get(Properties.published.rawValue)
   let createdTime: Double = try
   row.get(Properties.created.rawValue)
   let lastEditedTime: Double? = try?
   row.get(Properties.lastEdited.rawValue)

   created = Date(timeIntervalSince1970: createdTime)

   if let lastEditedTime = lastEditedTime {
      lastEdited = Date(timeIntervalSince1970: lastEditedTime)
   }
}
```

FROM THIS

```
func getAuthor() throws -> BlogUser? {
   return try parent(author, "bloguser_id", BlogUser.self).get()
}

func tags() throws -> [BlogTag] {
   return try siblings().all()
}
```

TO THIS

```
var postAuthor: Parent<BlogPost, BlogUser> {
   return parent(id: author)
}

var tags: Siblings<BlogPost, BlogTag, Pivot<BlogPost, BlogTag>> {
   return siblings()
}
```

AUTHENTICATION

- Now a separate module
- Much simplified!
- Lot's of different `Authenticable` types

PASSWORD AUTHENTICABLE

- `authenticate()` is now implemented for you!
- Need to implemented at least:
  - `hashedPassword`
  - `passwordVerifier` - do not leave this empty!

BlogUser.swift

```
extension BlogUser: PasswordAuthenticatable {
   public static let usernameKey =
   Properties.username.rawValue
   public static let passwordVerifier: PasswordVerifier? =
   BlogUser.passwordHasher
   public var hashedPassword: String? {
      return password.makeString()
   }
   public static let passwordHasher = BCryptHasher(cost: 10)
}
```

SessionPersistable

- Just implement it!

```
extension BlogUser: SessionPersistable {}
```

USER REQUEST CONVENIENCE FUNCTION

- Makes getting the user from a request easier

```
extension Request {
   func user() throws -> BlogUser {
      return try auth.assertAuthenticated()
   }
}
```

LOGIN

```
guard let username = request.data["username"]?.string, let
password = request.data["password"]?.string else {
   throw Abort.badRequest
}

let passwordCredentials = Password(username: username, password:
password)

do {
   let user = try BlogUser.authenticate(passwordCredentials)
   request.auth.authenticate(user)
   return Response(redirect: pathCreator.createPath(for: "admin"))
} catch {
   let loginError = ["Your username or password was incorrect"]
   return try viewFactory.createLoginView(loginWarning: false,
   errors: loginError, username: username, password: "")
}
```

REGISTRATION

```
let hashedPassword = try
BlogUser.passwordHasher.make(password)
let newUser = BlogUser(name: name, username:
username.lowercased(), password: hashedPassword,
profilePicture: profilePicture, twitterHandle: twitterHandle,
biography: biography, tagline: tagline)

do {
   try newUser.save()
}
catch {
   // Deal with error
}
```

ROUTING

- No parameter restrictions!
- New `Parameterizable` protocol
- New way of extracting parameters from requests

Parameterizable

- For `Entity` types, just add the protocol conformance!

```
extension BlogUser: Parameterizable {}
```

Parameterizable

- For custom types, implement it like so:

```
extension MyCustomType: Parameterizable {
   static var uniqueSlug: String = "mycustomtype"

   static func make(for parameter: String) throws -> MyCustomType {
      guard let customTypeObject = try MyCustomType(id: parameter)
      else {
         throw RouterError.invalidParameter
      }
      return customTypeObject
   }
}
```

USING Parameterizable

```
// https://steampress.io/users/1/
drop.get("users", BlogUser.parameter) { request in
   let user = try request.parameters.next(BlogUser.self)
   ...
}
```

LEAF AND VIEWS

- Add Provider to Package.swift
- Add Provider to Config
- Add view to droplet.json

```
{
    ...
     "view": "leaf",
     ...
}
```

TESTING

- Much simplified due to Configuration changes
- Set up preparations, middleware, routes as you would your main application
- Just don't add `try drop.run()`

TIDBITS

- `makeNode()` *must* now take a `Context`
- Lots of things now decoupled from Vapor - Leaf, Fluent, Auth. You may need to include them

NEXT STEPS

- https://steampress.io - on Vapor 2
- https://github.com/brokenhandsio/SteamPress/issues/12
  https://github.com/brokenhandsio/SteamPressExample/issues/7
- https://docs.vapor.codes/2.0/

QUESTIONS

We're Hiring!

http://www.bbc.co.uk/careers/

tim.condon@bbc.co.uk