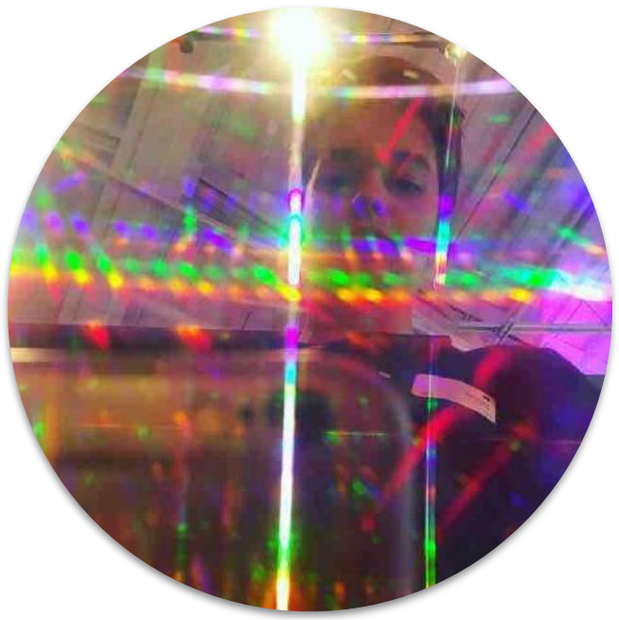
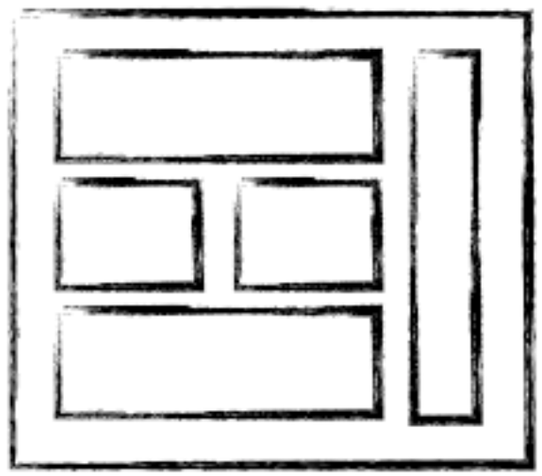
Microservices in Vapor

Building Big Projects



@tanner0101

http://tanner.xyz

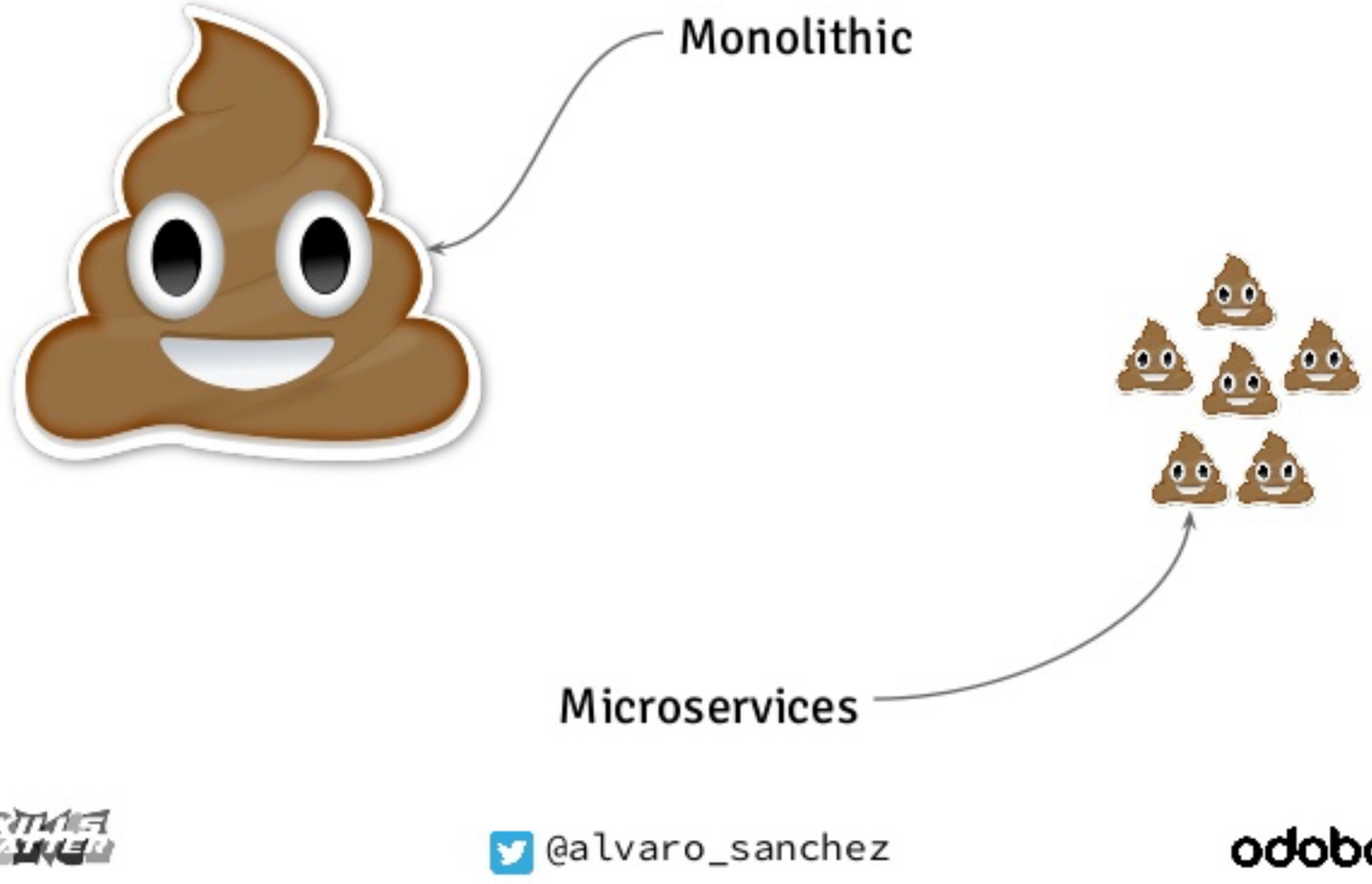


MONOLITHIC/LAYERED

MICRO SERVICES

*http://eugenedvorkin.com/seven-micro-services-architecture-advantages/

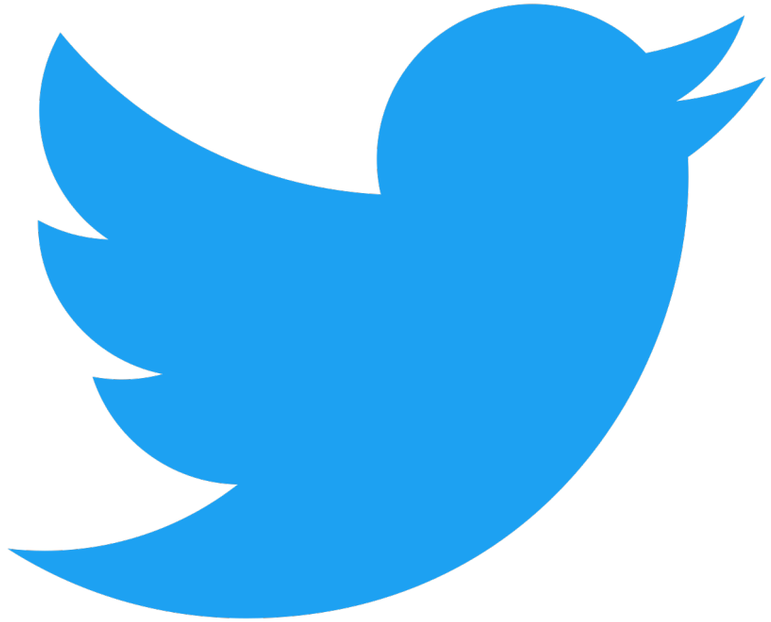
Monolithic vs Microservices

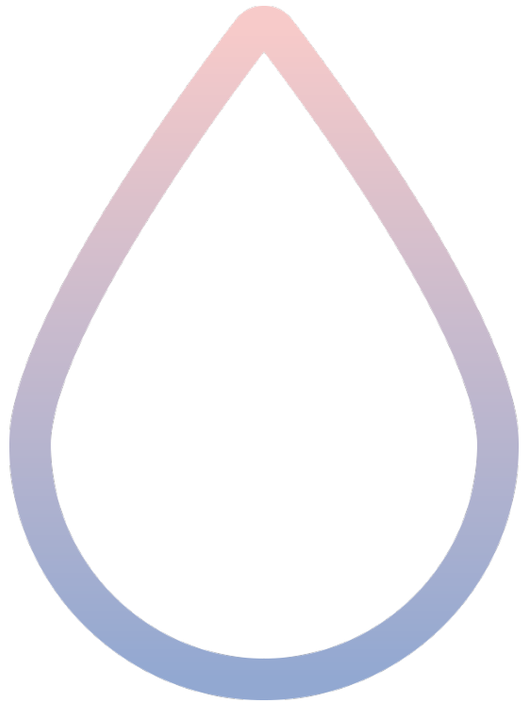


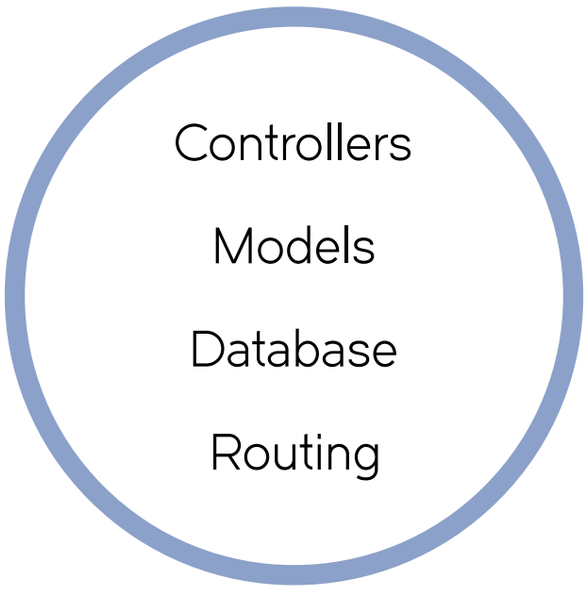
http://www.wwwlicious.com/2016/05/03/servicestack-microservices-1/

```swift
import Vapor

public class User: Model {
    ...
}
```

```swift
import Models

extension Models.User {
    func sayHello() {
        print("Hi, my name is \(self.name)")
    }
}
```







Tweets API

```
1   import Vapor
2
3   class Tweet: Model {
4       var userId: String
5       var message: String
6       ...
7   }
```

Tweets API

```swift
import Vapor

class TweetController {
    func store(_ req: Request) throws -> ResponseRepresentable {
        guard
            let message = req.data["message"]?.string,
            let userId = req.data["userId"]?.string
        else {
            throw Abort(.badRequest)
        }

        let tweet = Tweet(userId: userId, message: message)
        try tweet.save()
        return tweet
    }
}
```

Tweets API

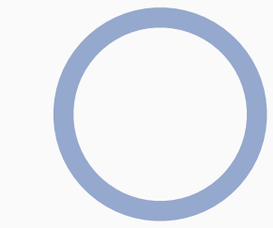
```
1   import Vapor
2
3   let drop = try Droplet()
4
5   let tweetController = TweetController()
6   drop.post("tweets", handler: tweetController.store)
7
8   try drop.run()
```

Tweets API

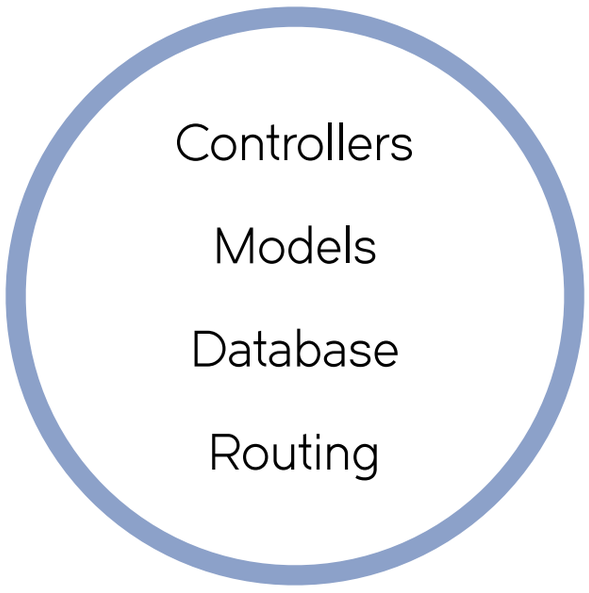
```swift
import Vapor

class TweetController {
    func store(_ req: Request) throws -> ResponseRepresentable {
        guard
            let message = req.data["message"]?.string,
            let userId = req.data["userId"]?.string
        else {
            throw Abort(.badRequest)
        }

        let tweet = Tweet(userId: userId, message: message)
        try tweet.save()
        return tweet
    }
}
```

Tweets API

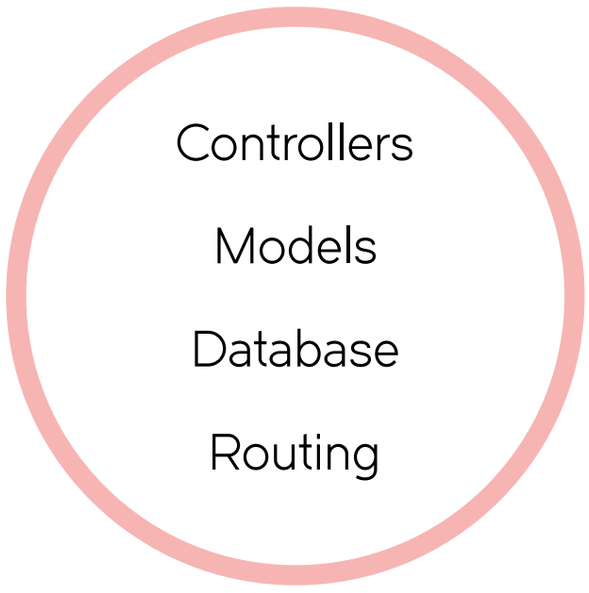
```
        let message = req.data["message"]?.string,
        let userId = req.data["userId"]?.string
    else {
        throw Abort(.badRequest)
    }

    let res = try drop.client.get("http://api.twitter.com/users/\(userId)")
    guard
        let isBanned = res.data["is_banned"]?.bool,
        res.status == .ok
    else {
        throw Abort(.internalServerError)
    }

    guard !isBanned else {
        throw Abort(.unauthorized)
    }

    let tweet = Tweet(userId: userId, message: message)
    try tweet.save()
    return tweet
  }
}
```



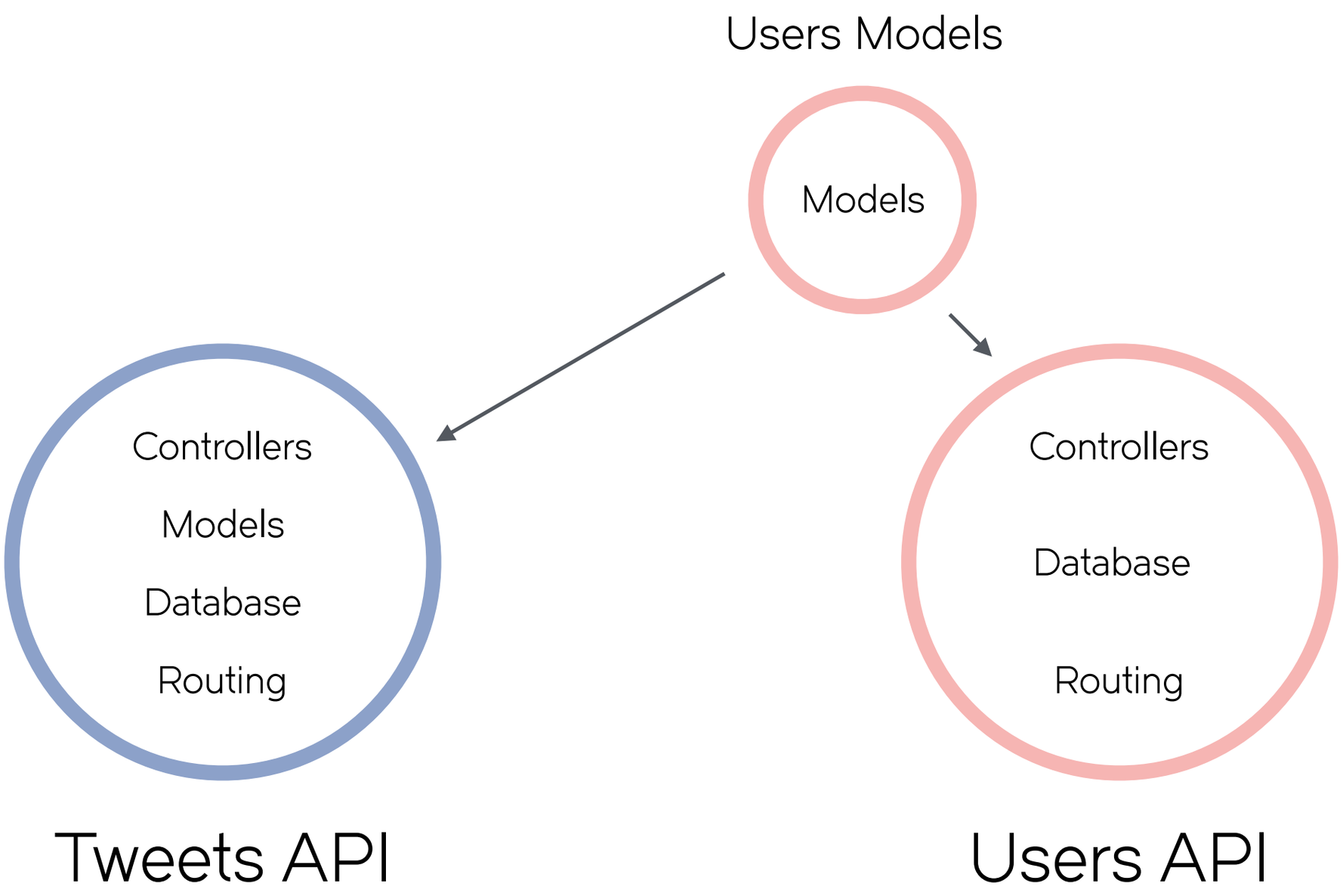
Tweets API



Tweets API



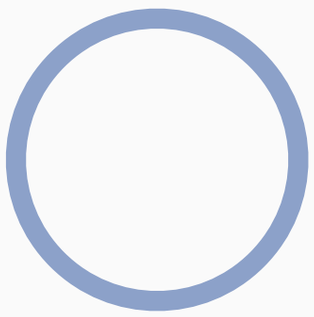
Users API


Users Models
Models
Controllers
Models
Database
Routing
Controllers
Database
Routing
Tweets API
Users API

Users Models

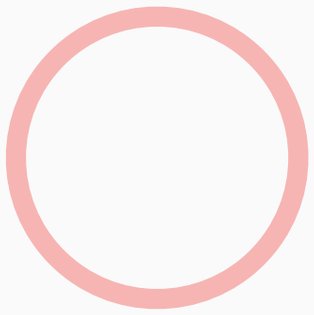
```
import PackageDescription

let package = Package(
    name: "UsersModels",
    dependencies: [
        .Package(url: "https://github.com/vapor/json.git", majorVersion: 2)
    ]
)
```

Tweets API



```
import PackageDescription

let package = Package(
    name: "TweetsAPI",
    dependencies: [
        .Package(url: "https://github.com/vapor/vapor.git", majorVersion: 2),
        .Package(url: "https://github.com/twitter/users-models.git", majorVersion: 1)
    ]
)
```

Users API



```
import PackageDescription

let package = Package(
    name: "UsersAPI",
    dependencies: [
        .Package(url: "https://github.com/vapor/vapor.git", majorVersion: 2),
        .Package(url: "https://github.com/twitter/users-models.git", majorVersion: 1)
    ]
)
```

```
1   public final class User {
2       public var id: String
3       public var name: String
4       public var isBanned: Bool
5   }
```

Users Models

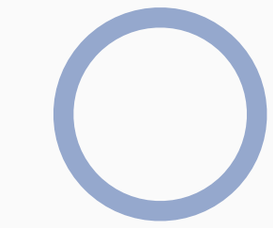
```swift
import JSON

extension User: JSONConvertible {
    public init(json: JSON) throws {
        id = try json.get("id")
        name = try json.get("name")
        isBanned = try json.get("isBanned")
    }

    public makeJSON() throws -> JSON {
        var json = JSON()
        try json.set("id", id)
        try json.set("name", name)
        try json.set("isBanned", isBanned)
        return json
    }
}
```

Users Models

```
{
    "id": "a8ccc26e-2c9d-4105-b69b-686f9b0b09f3",
    "name": "Bob",
    "isBanned": false
}
```

Users Models

```
        let message = req.data["message"]?.string,
        let userId = req.data["userId"]?.string
    else {
        throw Abort(.badRequest)
    }

    let res = try drop.client.get("http://api.twitter.com/users/\(userId)")
    guard
        let isBanned = res.data["is_banned"]?.bool,
        res.status == .ok
    else {
        throw Abort(.internalServerError)
    }

    guard !isBanned else {
        throw Abort(.unauthorized)
    }

    let tweet = Tweet(userId: userId, message: message)
    try tweet.save()
    return tweet
  }
}
```



Tweets API

```swift
    func store(_ req: Request) throws -> ResponseRepresentable {
        guard
            let message = req.data["message"]?.string,
            let userId = req.data["userId"]?.string
        else {
            throw Abort(.badRequest)
        }

        let res = try drop.client.get("http://api.twitter.com/users/\(userId)")
        let user = try User(json: res.json)

        guard !user.isBanned else {
            throw Abort(.unauthorized)
        }

        let tweet = Tweet(userId: userId, message: message)
        try tweet.save()
        return tweet
    }
}
```

Tweets API

Users Models

Models

Controllers

Database

Routing

Controllers

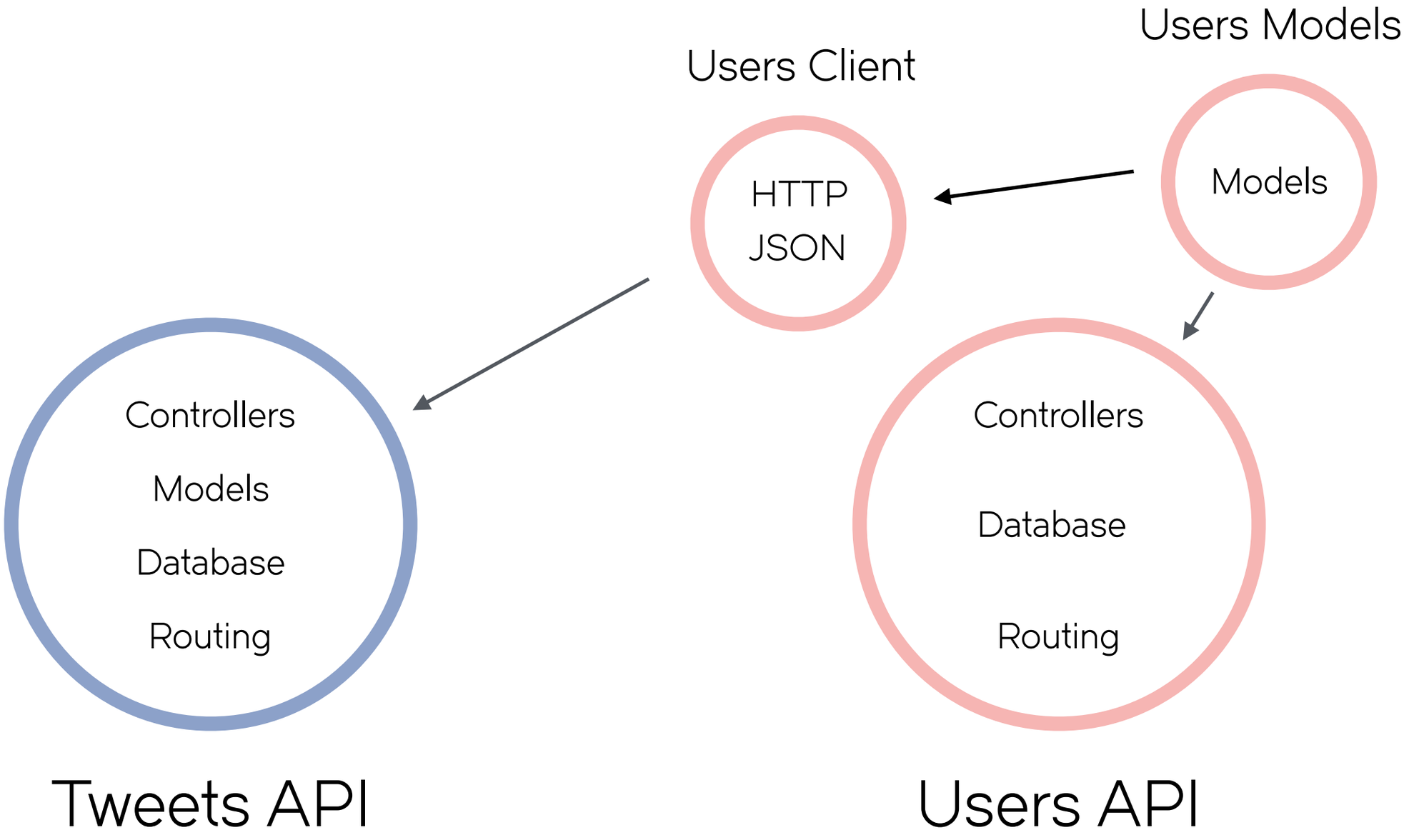
Models

Database

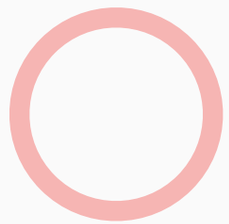
Routing

Tweets API

Users API


Users Models
Users Client
HTTP
JSON
Models
Controllers
Models
Database
Routing
Controllers
Database
Routing
Tweets API
Users API

```swift
public final class UserClient {
    ...

    public static func getUser(withId id: String) throws -> User {
        let res = client.get("/users/\(id)")
        return try User(json: res.json)
    }

    ...
}
```



Users Client

```swift
    func store(_ req: Request) throws -> ResponseRepresentable {
        guard
            let message = req.data["message"]?.string,
            let userId = req.data["userId"]?.string
        else {
            throw Abort(.badRequest)
        }

        let res = try drop.client.get("http://api.twitter.com/users/\(userId)")
        let user = try User(json: req.json)

        guard !user.isBanned else {
            throw Abort(.unauthorized)
        }

        let tweet = Tweet(userId: userId, message: message)
        try tweet.save()
        return tweet
    }
}
```

Tweets API

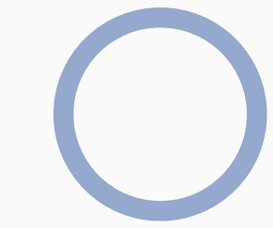
```
5           guard
6               let message = req.data["message"]?.string,
7               let userId = req.data["userId"]?.string
8           else {
9               throw Abort(.badRequest)
10          }
11
12          let user = try UserClient.getUser(withId: userId)
13          guard !user.isBanned else {
14              throw Abort(.unauthorized)
15          }
16
17          let tweet = Tweet(userId: userId, message: message)
18          try tweet.save()
19          return tweet
20      }
21  }
```

Tweets API

```
3   class TweetController {
4       func store(_ req: Request) throws -> ResponseRepresentable
5           guard
6               let message = req.data["message"]?.string,
7               let userId = req.data["userId"]?.string
8           else {
9               throw Abort(.badRequest)
10          }
11
12          try UserClient.assertUserNotBanned(withId: userId)
13
14          let tweet = Tweet(userId: userId, message: message)
15          try tweet.save()
16          return tweet
17      }
18  }
```

Tweets API

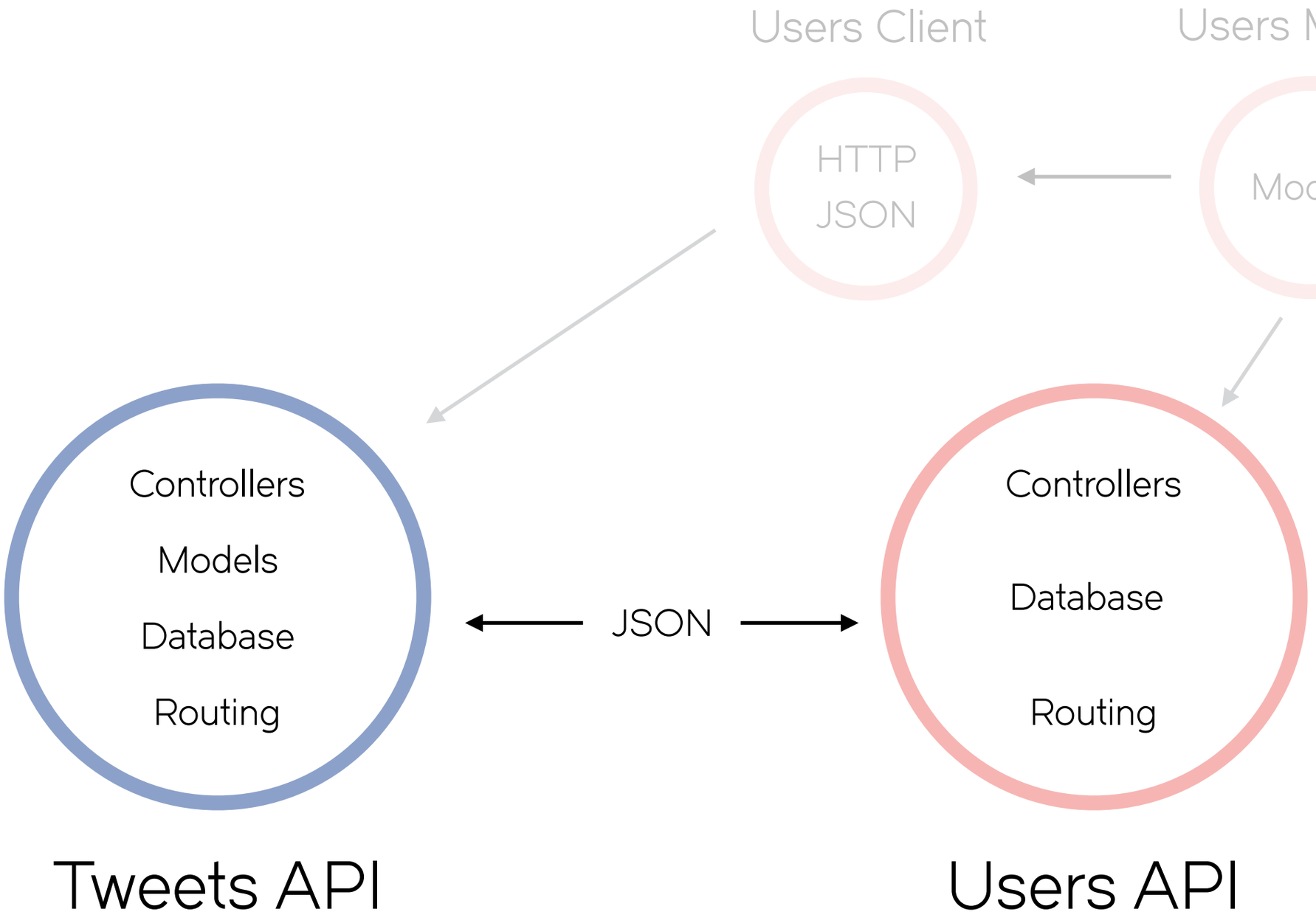
```
        let message = req.data["message"]?.string,
        let userId = req.data["userId"]?.string
    else {
        throw Abort(.badRequest)
    }

    let res = try drop.client.get("http://api.twitter.com/users/\(userId)")
    guard
        let isBanned = res.data["is_banned"]?.bool,
        res.status == .ok
    else {
        throw Abort(.internalServerError)
    }

    guard !isBanned else {
        throw Abort(.unauthorized)
    }

    let tweet = Tweet(userId: userId, message: message)
    try tweet.save()
    return tweet
  }
}
```

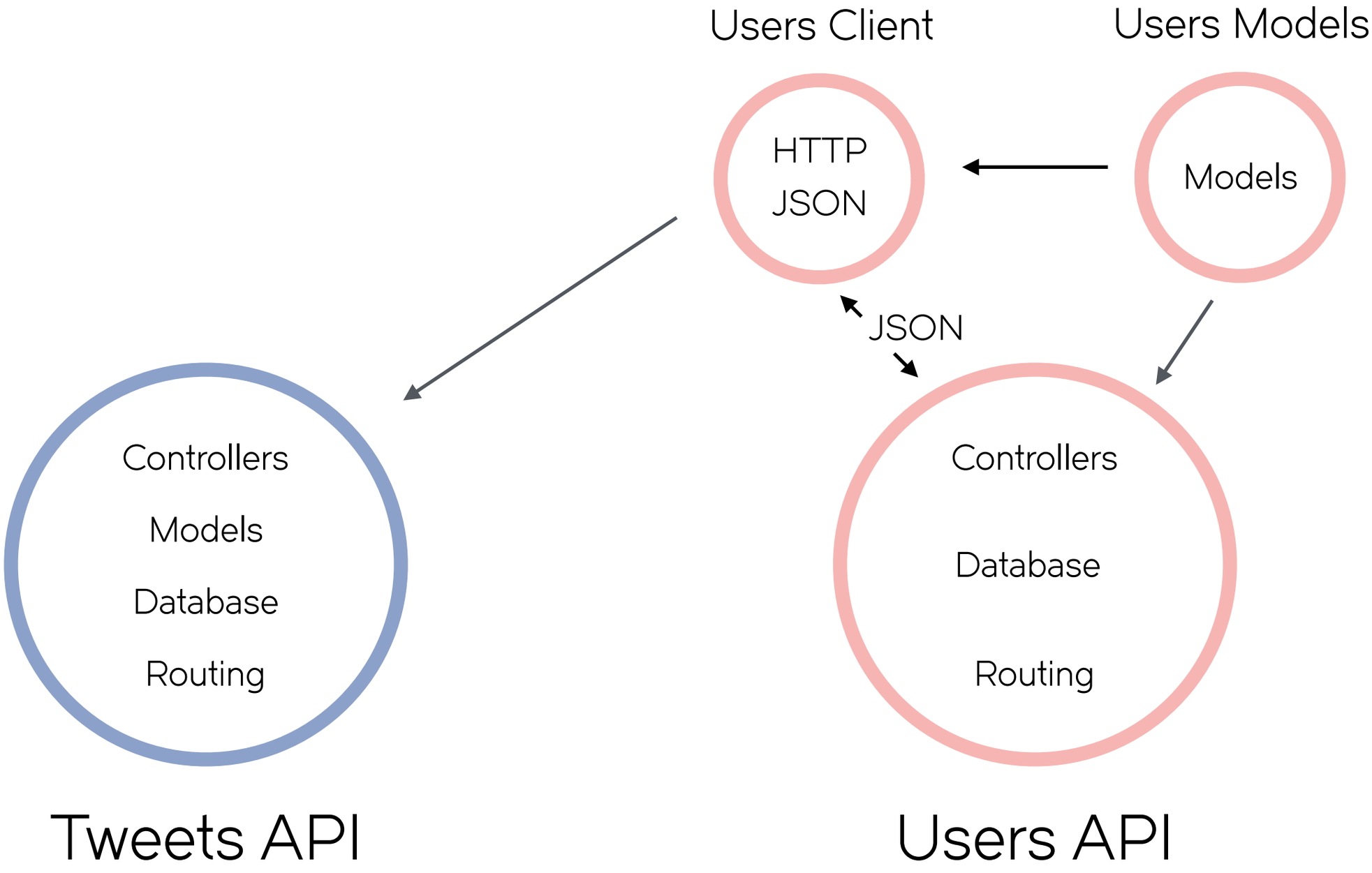


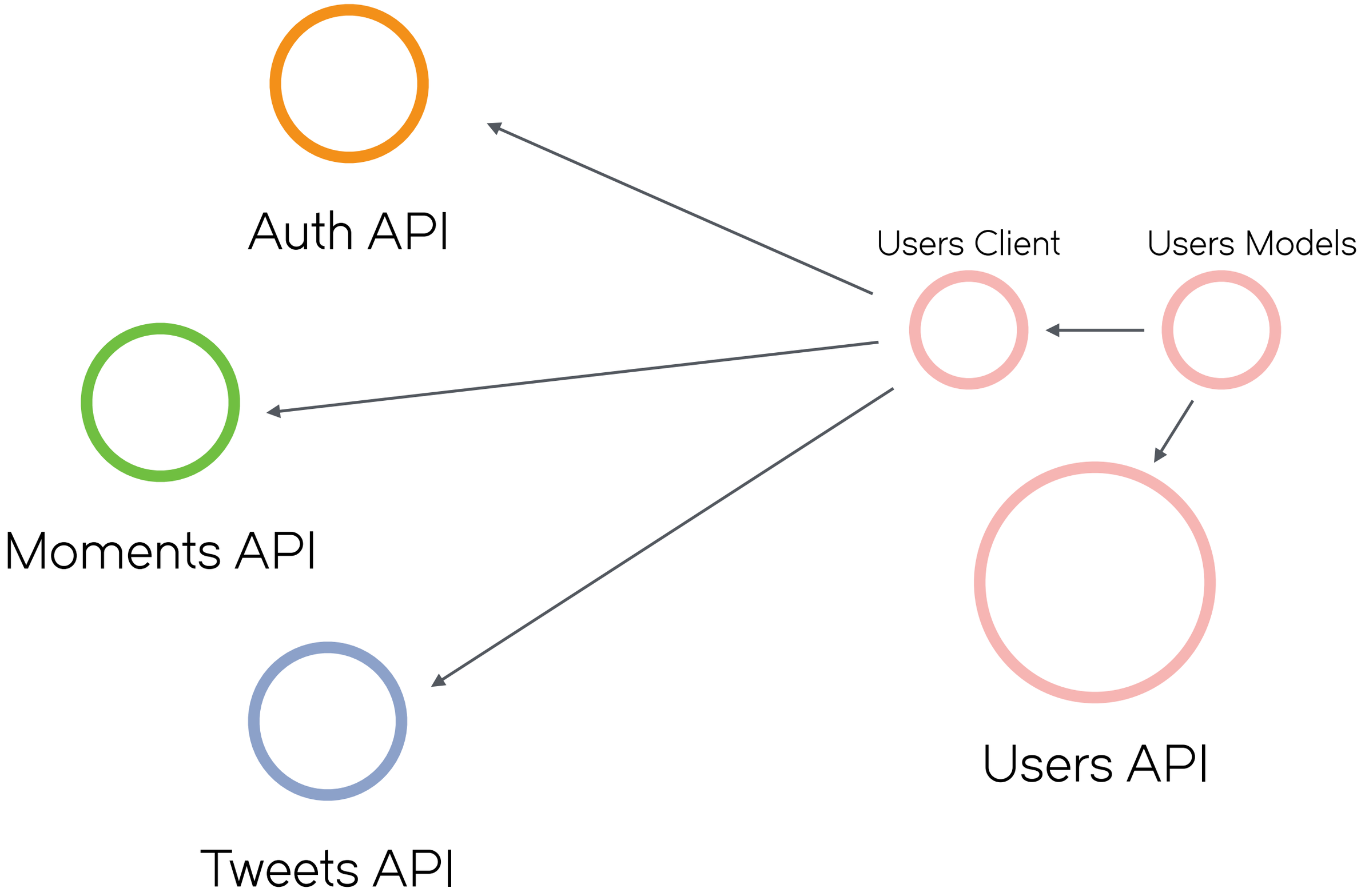
Tweets API

```
3   class TweetController {
4       func store(_ req: Request) throws -> ResponseRepresentable
5           guard
6               let message = req.data["message"]?.string,
7               let userId = req.data["userId"]?.string
8           else {
9               throw Abort(.badRequest)
10          }
11
12          try UserClient.assertUserNotBanned(withId: userId)
13
14          let tweet = Tweet(userId: userId, message: message)
15          try tweet.save()
16          return tweet
17      }
18  }
```

Tweets API


Users Client
Users Models
HTTP
JSON
Models
Controllers
Models
Database
Routing
JSON
Controllers
Database
Routing
Tweets API
Users API


Users Client
Users Models
HTTP
JSON
Models
JSON
Controllers
Models
Database
Routing
Controllers
Database
Routing
Tweets API
Users API


Auth API
Users Client
Users Models
Moments API
Users API
Tweets API

```swift
public final class UserClient {
    ...

    @available(*, deprecated: 2.0, message: "No longer necessary.")
    public static func assertUserNotBanned(withId id: String) throws {
        ...
    }

    ...
}
```

Users Client

Tw...PI 〉 My Mac | TweetsAPI | Build **Succeeded** ⚠ 1

TweetsAPI 〉 Sources 〉 App 〉 Controllers 〉 TweetController.swift 〉 M store(_:)

```
1  import Vapor
2  import HTTP
3
4  class TweetController {
5      func store(_ req: Request) throws -> ResponseRepresentable {
6          guard
7              let message = req.data["message"]?.string,
8              let userId = req.data["userId"]?.string
9          else {
10             throw Abort(.badRequest)
11         }
12
13         try UserClient.assertUserNotBanned(withId: userId)
14
15         let tweet = Tweet(userId: userId, message: message)
16         try tweet.save()
17         return tweet
18     }
19 }
20
21
22
```

⚠ 'assertUserNotBanned(withId:)' is deprecated: No longer necessary.

Tweets API

Building Big Projects with Vapor

DRY, use modules and extensions

DRY, use modules and extensions

```
1   drop.get("profile") {
2       drop.view.make("profile.leaf", [
3           "name": "\(user.firstName) \(user.lastName)"
4       ])
5   }
```

DRY, use modules and extensions

```
1   extension User {
2       var fullName: String {
3           return "\(user.firstName) \(user.lastName)"
4       }
5   }
6
7   drop.get("profile") {
8       drop.view.make("profile.leaf", [
9           "name": user.fullName
10      ])
11  }
```

Building Big Projects with Vapor

DRY, use modules and extensions

Avoid direct JSON / Strings

Avoid direct JSON / Strings

✅

```swift
let user = try User(json: json)
guard !user.isBanned else {
    throw Abort(.unauthorized)
}
```

❌

```swift
guard let isBanned = json["isBanned"]?.bool else {
    throw Abort(.internalServerError)
}

guard !isBanned else {
    throw Abort(.unauthorized)
}
```

Building Big Projects with Vapor

DRY, use modules and extensions

Avoid direct JSON / Strings

The compiler is your friend

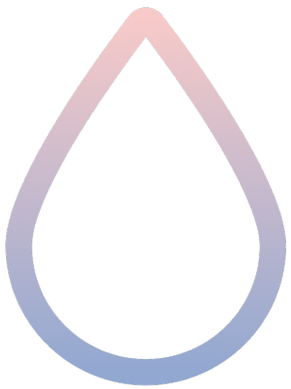
The compiler is your friend

✅

```
1   try UsersClient.assertUserNotBanned(withId: userId)
```

❌

```
1   let res = try drop.client.get("https://api.twitter.com/users/\(userId)")
2   let user = try User(json: res.json)
3   guard !user.isBanned else {
4       throw Abort(.unauthorized)
5   }
```

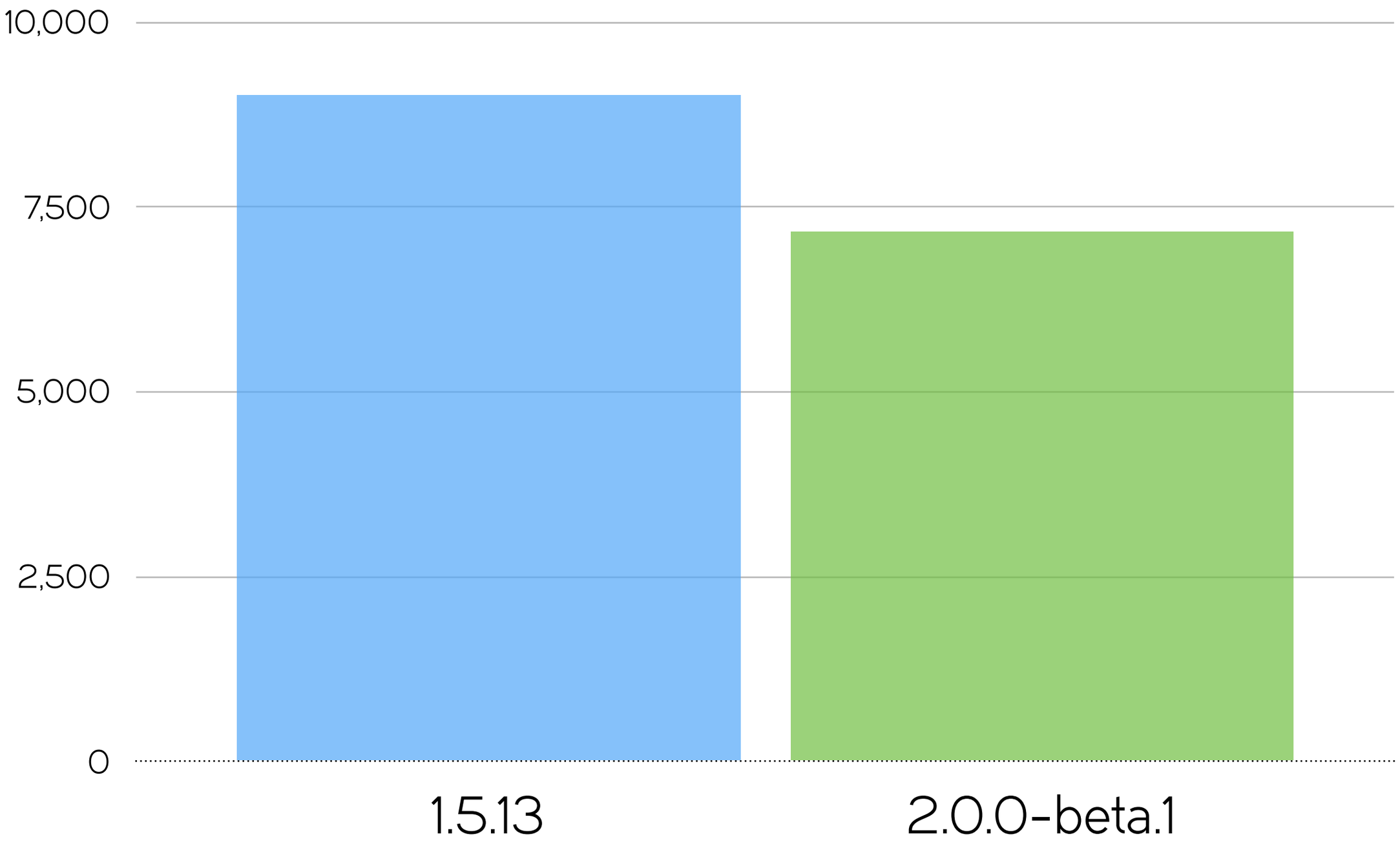


VAPOR

Vapor 2

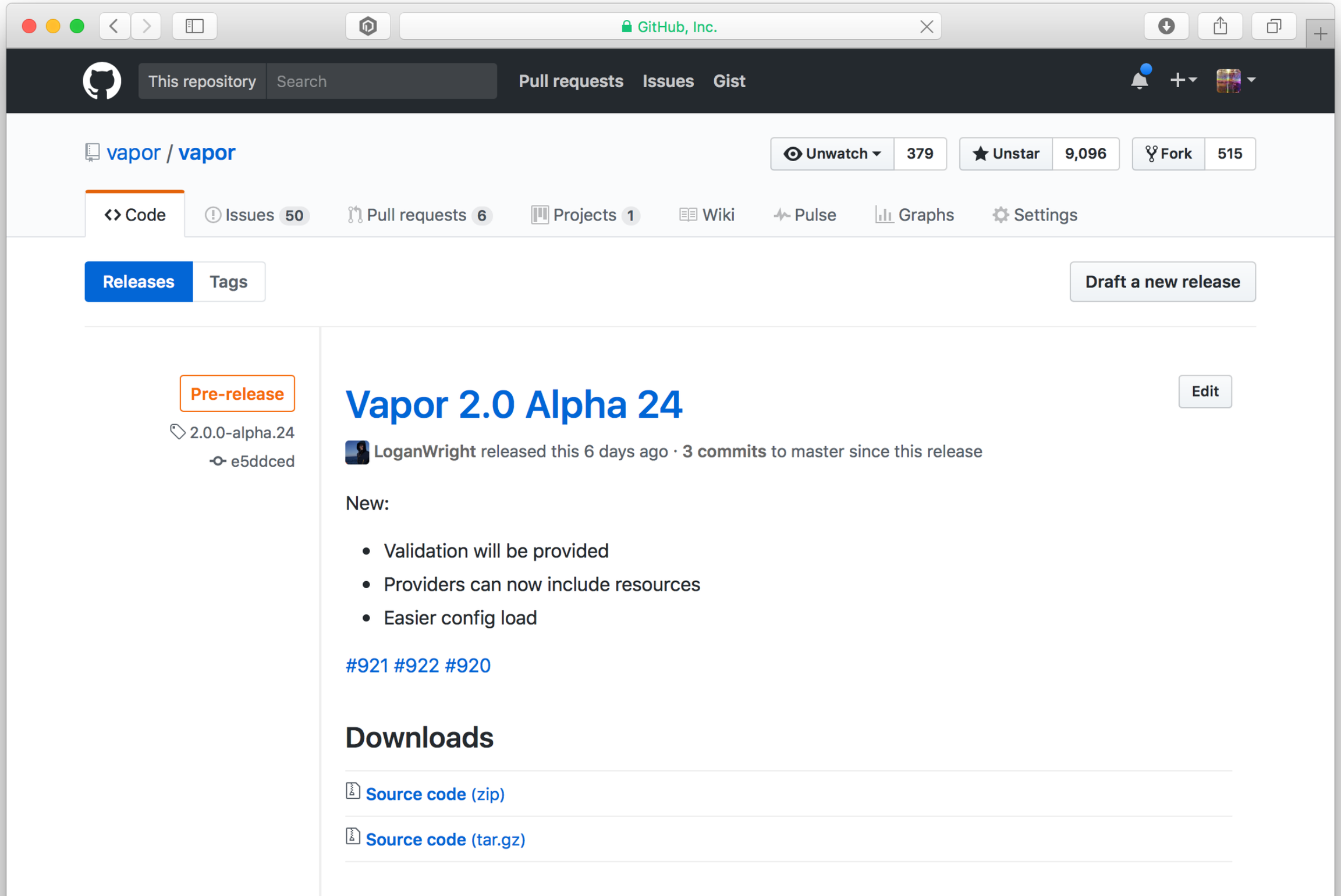
Less Code

Vapor 2 is lighter and more modular



More Features

>50 Alpha Iterations



Debuggable

Soft Delete

Mail

Generic Relations

Multiple Validation Errors

Raw

Swift 3.1

JWT

Authorization

Pagination

Improved Modularity

Fluent Storage

UUID

Row

BCrypt

Testing

Faster Compilation

_method

OpenSSL

5x Faster Redis

Cache Expiration

Custom ID Keys

Route List

Node

Double Pivot

SQLite Memory DB

Connections API

Timestamps

Beautiful New Docs

Vapor Docs
Overview
Getting started
Vapor
Folder Structure
Droplet
Views
Controllers
Middleware
Validation
Provider
Hash
Log
Commands
Settings
JSON
Routing
Fluent
Auth
Sessions
HTTP
Leaf
Advanced

Version Middleware

As an example, let's create a middleware that will add the version of our API to each response. The middleware would look something like this:

```
final class VersionMiddleware: Middleware {
    func respond(to request: Request, chainingTo next: Responder) throws -> Respo
        let response = try next.respond(to: request)

        response.headers["Version"] = "API v1.0"

        return response
    }
}
```

We then supply this middleware to our `Droplet`.

```
let drop = Droplet()
drop.middleware.append(VersionMiddleware())
```

You can imagine our `VersionMiddleware` sitting in the middle of a chain that connects the client and our server. Every request and response that hits our server must go through this chain of middleware.

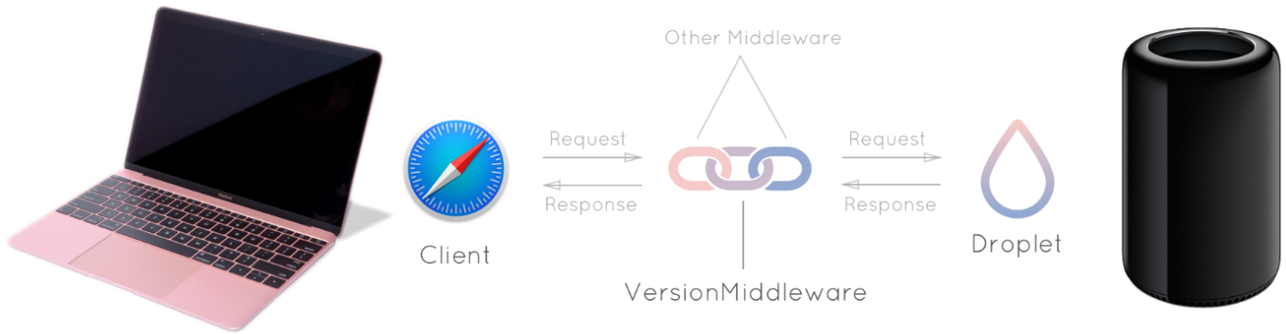
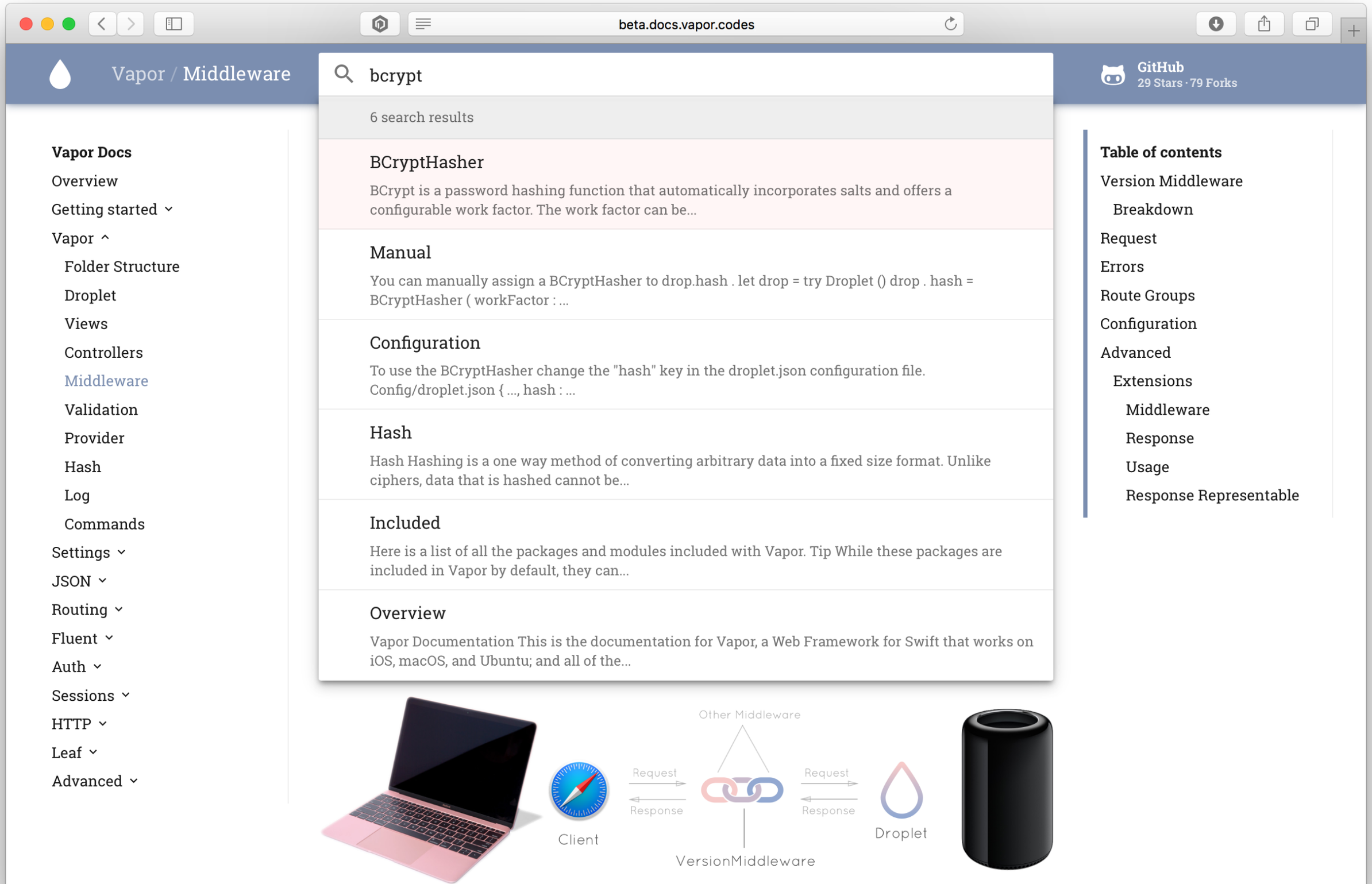


Table of contents
Version Middleware
Breakdown
Request
Errors
Route Groups
Configuration
Advanced
Extensions
Middleware
Response
Usage
Response Representable

Beautiful New Docs



Vapor 2

#vapor2

BETA Available Today

beta.docs.vapor.codes

#beta

vapor.codes

vapor.team

@codevapor
#vaporswift

docs.vapor.codes

Questions?

vapor.codes

vapor.team

@codevapor
#vaporswift

docs.vapor.codes