Getting started with Vapor 3

By Joannis Orlandos

Joannis Orlandos

- The "Optimization guy"
- Code Idealist/Perfectionist
- MongoDB enthusiast

What is Vapor?

- 💧 Swift Web Framework/Ecosystem
- 👨‍👩‍👧‍👦 Community
- ☁️ Cloud hosting

Community

Get in on the conversation at

http://vapor.team

Philosophy

- 🚀 High performance implementations
- 💖 Beautiful high level APIs
- 🧠 100% Swift

Simple to start

```
router.get("hello/world") { request in
    return "Hello, world!"
}
```

Simple to use

```
router.get("hello", String.parameter) { request in
    let name = try request.parameter(String.self)

    return "Hello, \(name)!"
}
```

```swift
struct JSONResponse: Content {
    var message: String
    var name: String
}
router.get("hello", String.parameter) { request in
    let name = try request.parameter(String.self)

    return JSONResponse(message: "Hello!", name: name)
}
```

GET /hello/Joannis

json

```json
{
    "message": "Hello!",
    "name": "Joannis"
}
```

Fast for simple applications

```
Running 10s test @ http://localhost:8080/plaintext
  4 threads and 100 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency     2.00ms    4.09ms 109.74ms   94.39%
    Req/Sec    19.76k     2.33k   25.69k    74.25%
  786409 requests in 10.00s, 69.75MB read
Requests/sec:  78631.72
Transfer/sec:      6.97MB
```

Stays fast for heavy applications

```
➜  ~ wrk -d 10s -t 4 -c 100 http://localhost:8080/redirect
Running 10s test @ http://localhost:8080/redirect
  4 threads and 100 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency     2.42ms    8.09ms 161.38ms   98.71%
    Req/Sec    15.28k     2.20k   17.76k    88.25%
  608604 requests in 10.02s, 42.95MB read
Requests/sec:  60712.03
Transfer/sec:      4.28MB
```

Proxies the HTTP requests to http://google.com

(yes, before HTTPS redirect)

Before we start

- Xcode 9
- Swift 4.1 (snapshot)
- Vapor Toolbox

Break #1

- ✅ Set up your environment
- ☕️ Have a drink and a talk

Ecosystem overview

- (High Level) Frameworks
- Database drivers
- Networking
- "Web" Services
- Utility

Frameworks

- Vapor
- Fluent

Web Services (Vapor)

- Engine
    - HTTP
    - WebSocket
    - Multipart forms
- Routing

Database Drivers (Fluent)

- MySQL
- PostgreSQL
- Redis
- SQLite
- MongoDB

Networking

- TLS
- TCP

Utility

- Async
- Services
- Crypto

WIP features

- HTTP/2
- Kafka
  Open to suggestions ✅

Questions?

Setting up your application 🎉

```
# Navigate to my project folder
cd /path/to/my/projects/
# Create a new API project
vapor new todo --api --branch=beta
# Enter the project
cd ./todo
# Fetch dependencies and open in Xcode
vapor xcode -y
```

Project structure

```
.
├── Public
├── Sources
│   ├── App
│   │   ├── Controllers
│   │   ├── Models
│   │   ├── boot.swift
│   │   ├── configure.swift
│   │   └── routes.swift
│   └── Run
│       └── main.swift
├── Tests
│   └── AppTests
└── Package.swift
```

Routes.swift

```swift
// Basic "Hello, world!" example
router.get("hello") { req in
    return "Hello, world!"
}
```

GET /hello/<name>

```
router.get("hello", String.parameter) { request in
    let name = try request.parameter(String.self)

    return "Hello, \(name)!"
}
```

Models/Todo.swift

- Basic TODO SQLite model
- Schema gets created automatically
- SQLite flushes every application restart

```swift
/// A single entry of a Todo list.
final class Todo: SQLiteModel {
    /// The unique identifier for this `Todo`.
    var id: Int?
    /// A title describing what this `Todo` entails.
    var title: String
    /// Creates a new `Todo`.
    init(id: Int? = nil, title: String) {
        self.id = id
        self.title = title
    }
}
/// Allows `Todo` to be used as a dynamic migration.
extension Todo: Migration { }
/// Allows `Todo` to be encoded to and decoded from HTTP messages.
extension Todo: Content { }
/// Allows `Todo` to be used as a dynamic parameter in route definitions.
extension Todo: Parameter { }
```

Controllers/TodoController.swift

- Basic APIs
    - List
    - Create
    - Delete

GET /todos

```
return Todo.query(on: req).all()
```

POST /todos

```
return try req.content.decode(Todo.self).flatMap(to: Todo.self) { todo in
    return todo.save(on: req)
}
```

DELETE /todos/<todoID>

```
return try req.parameter(Todo.self).flatMap(to: Todo.self) { todo in
    return todo.delete(on: req)
}.transform(to: .ok)
```

configure.swift

- Registers services
  - Router
  - Database
  - Migrations

```
migrations.add(model: Todo.self, database: .sqlite)
```

Task #1

- Add a "message" to the TODO
- Add an "update" PUT route to the API
- Test the API manually using a REST client
- Accept form input alongside JSON

Break #2

Any questions?

Working with Leaf

Package.swift

```swift
// Add this to your dependencies
.package(url: "https://github.com/vapor/leaf.git", from: "3.0.0-rc"),

// Update your Target to depend on Leaf
.target(name: "App", dependencies: ["FluentSQLite", "Vapor", "Leaf"]),
```

Shell

```
# Update dependencies
vapor update
# Rebuild Xcode project
vapor xcode -y
```

Project structure

```
.
├── Public
├── Resources
│   └── Views      <----
├── Sources
│   ├── App
│   │   ├── Controllers
│   │   ├── Models
│   │   ├── boot.swift
│   │   ├── configure.swift
│   │   └── routes.swift
│   └── Run
│       └── main.swift
├── Tests
│   └── AppTests
└── Package.swift
```

configure.swift

```swift
import Leaf
try services.register(LeafProvider())
```

Resources/View/example.leaf

```
#for(product in list) {
    <p>
        <h1>#(product.name) - #(product.price) euro</h1>
        #if(product.awesome) {
            <h2>recommended</h2><br />
        }
        <span>#(product.description)</span>
    </p>
}
```

routes.swift

```swift
router.get("example") { request -> Future<View> in
    struct Product: Codable {
        var name: String
        var price: Double
        var awesome: Bool
        var description: String
    }

    let productA = Product(name: "cheese", price: 1.40, awesome: true, description: "Yummy")
    let productB = Product(name: "milk", price: 1.80, awesome: false, description: "Fluid-y")
    let productC = Product(name: "yoghurt", price: 2.20, awesome: true, description: "Something else-y")

    return try request.view().render("example", [
        "list": [
            productA,
            productB,
            productC
        ]
    ])
}
```

Accepted Leaf data

- `LeafData`
- `Encodable`
- `Future where T : Encodable`
- `OutputStream where Output : Encodable` (only in loops)

Task #2

- Display all todo items
- Create HTML buttons for removing todos
- Create new TODOs using a form

Questions? Let me know!