Documentation

Article

Testing

Using the HummingbirdTesting framework to test your application

Overview

Writing tests for application APIs is an important part of the development process. They ensure everything works as you expected and that you don't break functionality with future changes. Hummingbird provides a framework for testing your application as if it is a running server and you are a client connecting to it.

Example

Lets create a simple application that says hello back to you. ie If your request is to `/hello/adam` it returns "Hello adam!".

```
let router = Router()
router.get("hello/{name}") { _,context in
    return try "Hello \(context.parameters.require("name"))!"
}
let app = Application(router: router)
```

Testing

We can test the application returns the correct text as follows

```
func testApplicationReturnsCorrectText() async throw {
    try await app.test(.router) { client in
        try await client.execute(
            uri: "/hello/john",
            method: .get,
            headers: [:],  // default value
            body: nil      // default value
        ) { response in
            #expect(response.status == .ok)
            #expect(String(buffer: response.body) == "Hello john!")
        }
    }
}
```

`Application.test`

The `test(_:_:)` function takes two parameters, first the test framework to use and then the closure to run with the framework client. The test framework defines how we are going to test our application. There are three possible frameworks

Router (.router)

The router test framework will send requests directly to the router. It does not need a running server to run tests. The main advantages of this is it is the quickest way to test your application but will not test anything outside of the router. In most cases you won't need more than this.

Live (.live)

The live framework uses a live server, with an HTTP client attached on a single connection.

AsyncHTTPClient (.ahc)

The AsyncHTTPClient framework is the same as the live framework except it uses AsyncHTTPClient from swift-server as its HTTPClient. You can use this to test TLS and HTTP2 connections.

Executing requests and testing the response

The function `execute(uri:method:headers:body:testCallback:)` sends a request to your application and provides the response in a closure. If you return something from the closure then this is returned by `execute`. In the following example we are testing whether a session cookie works.

```
@Test
func testApplicationReturnsCorrectText() async throw {
    try await app.test(.router) { client in
        // test login, returns a set-cookie header and extract
        let cookie = try await client.execute(
            uri: "/user/login",
            method: .post,
            headers: [.authorization: "Basic blahblah"]
        ) { response in
            #expect(response.status == .ok)
            return try #require(response.headers[.setCookie])
        }
        // check session cookie works
        try await client.execute(
            uri: "/user/is-authenticated",
            method: .get,
            headers: [.cookie: cookie]
        ) { response in
            #expect(response.status == .ok)
        }
    }
}
```

See Also

Related Documentation

```
func test<Value>(TestingSetup, (any TestClientProtocol) async throws
-> Value) async throws -> Value
```

Test `Application`

`protocol` `TestClientProtocol`

Protocol for client used by HummingbirdTesting

Hummingbird Server

- Router

  The router directs requests to their handlers based on the contents of their path.

- Request Decoding

  Decoding of Requests with JSON content and other formats.

- Response Encoding

  Writing Responses using JSON and other formats.

- Request Contexts

  Controlling contextual data provided to middleware and route handlers

- Middleware

  Processing requests and responses outside of request handlers.

- Error Handling

  How to build errors for the server to return.

- Logging, Metrics and Tracing

  Considered the three pillars of observability, logging, metrics and tracing provide different ways of viewing how your application is working.

- Result Builder Router

  Building your router using a result builder.

- Server protocol

  Support for TLS and HTTP2 upgrades

- Service Lifecycle

  Integration with Swift Service Lifecycle

- Persistent data

  How to persist data between requests to your server.

- Migrating to Hummingbird v2

  Migration guide for converting Hummingbird v1 applications to Hummingbird v2
