Documentation

Hummingbird Document… / Authenticator Middleware

Article

Authenticator Middleware

Request authentication middleware

Overview

Authenticators are middleware that are used to check if a request is authenticated and then pass authentication data to functions further down the callstack via the request context. Authenticators should conform to protocol `AuthenticatorMiddleware`. This requires you implement the function `authenticate(request:context:)` that returns a value conforming to `Sendable`.

To use an authenticator it is required that your request context conform to `AuthRequestContext`. When you return valid authentication data from your `authenticate` function it is recorded in the `identity` member of your request context.

Usage

A simple username, password authenticator could be implemented as follows. If the authenticator is successful it returns a `User` struct, otherwise it returns `nil`.

```
struct BasicAuthenticator: AuthenticatorMiddleware {
    func authenticate<Context: AuthRequestContext>(request: Request, context
        // Basic authentication info in the "Authorization" header, is acces
        // via request.headers.basic
        guard let basic = request.headers.basic else { return nil }
        // check if user exists in the database and then verify the entered
        // against the one stored in the database. If it is correct then log
```

```
        let user = try await database.getUserWithUsername(basic.username)
        // did we find a user
        guard let user = user else { return nil }
        // verify password against password hash stored in database. If vali
        // return the user. HummingbirdAuth provides an implementation of Bc
        // This should be run on the thread pool as it is a long process.
        return try await NIOThreadPool.singleton.runIfActive {
            if Bcrypt.verify(basic.password, hash: user.passwordHash) {
                return user
            }
            return nil
        }
    }
}
```

An authenticator is middleware so can be added to your application like any other middleware

```
router.add(middleware: BasicAuthenticator())
```

Then in your request handler you can access your authentication data with `context.identity`.

```
/// Get current logged in user
func current(_ request: Request, context: MyContext) throws -> User {
    // get authentication data for user. If it doesnt exist then throw unaut
    let user = context.requireIdentity()
    return user
}
```

You can require that that authentication was successful and authentication data is available by adding the middleware `IsAuthenticatedMiddleware` after your authentication middleware

```
router.addMiddleware {
    BasicAuthenticator()
    IsAuthenticatedMiddleware()
```

```
}
```

See Also

Related Documentation

`protocol` `AuthenticatorMiddleware`

Protocol for a middleware that checks if a request is authenticated.

`protocol` `AuthRequestContext`

Protocol that all request contexts should conform to if they want to support authentication middleware

`struct` `IsAuthenticatedMiddleware`

Middleware returning 401 for unauthenticated requests

Authentication

Sessions

Session based authentication

One Time Passwords

A one time password (OTP) valid for only one login session.