Documentation

Hummingbird Document… / Sessions

Article

Sessions

Session based authentication

Overview

Sessions allow you to persist state eg user authentication status between multiple requests to the server. They work by creating a temporary session object that is stored in a key/value store. The key or session id is returned in the response. Subsequent requests can then access the session object by supplying the session id in their request. This object can then be used to authenicate the user. Normally the session id is stored in a cookie.

SessionMiddleware

The `SessionMiddleware` is used to extract and save session state from the RequestContext. To use it, your `RequestContext` must conform to `SessionRequestContext`. Adding the `SessionMiddleware` to your middleware stack will mean any middleware or routes after will have read/write access to session state via the member `sessions`.

The `SessionMiddleware` needs a persist key value store to save its state. You can find out more about the persist framework here Persistent data. In the example below we are using an in memory key value store, but `FluentPersistDriver` and `RedisPersistDriver` provide solutions that stores the session data in a database or redis database respectively.

```
router.add(
    middleware: SessionMiddleware(
```

```
        storage: MemoryPersistDriver()
    )
)
```

By default sessions store the session id in a `SESSION_ID` cookie and the default session expiration is 12 hours. At initialization it is possible to set these up differently.

```
router.add(
    middleware: SessionMiddleware(
        storage: MemoryPersistDriver(),
        sessionCookie: "MY_SESSION_ID",
        defaultSessionExpiration: .seconds(60 * 60)
    )
)
```

SessionRequestContext

The `SessionRequestContext` protocol requires you include a member `sessions`. This is a `SessionContext` type which holds the session data for the current request and includes a generic parameter defining what type this session data is.

```
struct MyRequestContext: SessionRequestContext {
    /// core context
    public var coreContext: CoreRequestContextStorage
    /// session context with UUID as the session object
    public let sessions: SessionContext<UUID>
}
```

Saving a session

Once a user is authenticated you need to save a session for the user.

```
func login(_ request: Request, context: MyRequestContext) async throws -> HT
    // get authenticated user
    let user = try context.requireIdentity()
```

```
    // create session lasting 1 hour
    context.sessions.setSession(user.id, expiresIn: .seconds(600))
    return .ok
}
```

In this example `user.id` is saved with the session id. The data we save in `setSession` is saved to storage when we return to the `SessionMiddleware`. If your route throws an error then the session data is not updated.

Sessions Authentication

To authenticate a user using a session id you need to add a `SessionAuthenticator` middleware to the router. This uses the session stored in the request context and converts it into the authenticated user using the closure or `UserSessionRepository` provided. The session authenticator requires your `RequestContext` conforms to both `SessionRequestContext` and `AuthRequestContext`.

```
router.addMiddleware {
    SessionMiddleware(storage: MemoryPersistDriver())
    SessionAuthenticator { session, context in
        try await getUser(from: session)
    }
}
router.get("session") { request, context -> HTTPResponse.Status in
    _ = try context.requireIdentity()
    return .ok
}
```

See Also

Related Documentation

`struct SessionAuthenticator`

Session authenticator

`protocol` SessionRequestContext

Protocol for RequestContext that stores session data

Authentication

- Authenticator Middleware

  Request authentication middleware

- One Time Passwords

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