Session

HTTP sessions provide a way to store information about the user across multiple requests, which is particularly useful for MVC applications.

Use with Express (default)

First install the required package (and its types for TypeScript users):

```
$ npm i express-session
$ npm i -D @types/express-session
```

Once the installation is complete, apply the `express-session` middleware as global middleware (for example, in your `main.ts` file).

```
import * as session from 'express-session';
// somewhere in your initialization file
app.use(
  session({
    secret: 'my-secret',
    resave: false,
    saveUninitialized: false,
  }),
);
```

> warning **Notice** The default server-side session storage is purposely not designed for a production environment. It will leak memory under most conditions, does not scale past a single process, and is meant for debugging and developing. Read more in the official repository.

The `secret` is used to sign the session ID cookie. This can be either a string for a single secret, or an array of multiple secrets. If an array of secrets is provided, only the first element will be used to sign the session ID cookie, while all the elements will be considered when verifying the signature in requests. The secret itself should be not easily parsed by a human and would best be a random set of characters.

Enabling the `resave` option forces the session to be saved back to the session store, even if the session was never modified during the request. The default value is `true`, but using the default has been deprecated, as the default will change in the future.

Likewise, enabling the `saveUninitialized` option Forces a session that is "uninitialized" to be saved to the store. A session is uninitialized when it is new but not modified. Choosing `false` is useful for implementing login sessions, reducing server storage usage, or complying with laws that require permission before setting a cookie. Choosing `false` will also help with race conditions where a client makes multiple parallel requests without a session (source).

You can pass several other options to the `session` middleware, read more about them in the API documentation.

> info **Hint** Please note that `secure: true` is a recommended option. However, it requires an https-enabled website, i.e., HTTPS is necessary for secure cookies. If secure is set, and you access your site over HTTP, the cookie will not be set. If you have your node.js behind a proxy and are using `secure: true`, you need to set `"trust proxy"` in express.

With this in place, you can now set and read session values from within the route handlers, as follows:

```
@Get()
findAll(@Req() request: Request) {
  request.session.visits = request.session.visits ? request.session.visits
+ 1 : 1;
}
```

> info **Hint** The `@Req()` decorator is imported from the `@nestjs/common`, while `Request` from the `express` package.

Alternatively, you can use the `@Session()` decorator to extract a session object from the request, as follows:

```
@Get()
findAll(@Session() session: Record<string, any>) {
  session.visits = session.visits ? session.visits + 1 : 1;
}
```

> info **Hint** The `@Session()` decorator is imported from the `@nestjs/common` package.

Use with Fastify

First install the required package:

```
$ npm i @fastify/secure-session
```

Once the installation is complete, register the `fastify-secure-session` plugin:

```
import secureSession from '@fastify/secure-session';

// somewhere in your initialization file
const app = await NestFactory.create<NestFastifyApplication>(
  AppModule,
  new FastifyAdapter(),
);
await app.register(secureSession, {
  secret: 'averylogphrasebiggerthanthirtytwochars',
  salt: 'mq9hDxBVDbspDR6n',
});
```

> info **Hint** You can also pregenerate a key (see instructions) or use keys rotation.

Read more about the available options in the official repository.

With this in place, you can now set and read session values from within the route handlers, as follows:

```
@Get()
findAll(@Req() request: FastifyRequest) {
  const visits = request.session.get('visits');
  request.session.set('visits', visits ? visits + 1 : 1);
}
```

Alternatively, you can use the `@Session()` decorator to extract a session object from the request, as follows:

```
@Get()
findAll(@Session() session: secureSession.Session) {
  const visits = session.get('visits');
  session.set('visits', visits ? visits + 1 : 1);
}
```

> info **Hint** The `@Session()` decorator is imported from the `@nestjs/common`, while `secureSession.Session` from the `@fastify/secure-session` package (import statement: `import * as secureSession from '@fastify/secure-session'`).