Session Management

Table of Contents

1. Introduction

In this tutorial, we'll be creating a web application with the following characteristics:

- web server using Node.js and Express
- session management using express-session as middleware
- session storage using Redis

2. Basic Web Server

To run this tutorial script, just execute `sh webserver.sh` in this directory. The code has been produced from the current file.

2.1. Environment

We need to set up the server environment first. Make sure that `node` is installed and working.

```
echo -n "Testing node: "
if type -P node > /dev/null; then
    echo "Node.js seems to be correctly installed!"
else
    echo "Node.js command not found, please ensure it is installed and in your path!
    exit 1
fi
```

Next, we set up `package.json` via `npm`. This file tracks metadata like the name of the application, version, git upstream URL, and other stuff like that. Most importantly, it tracks the nodejs packages. We don't really care about the metadata, so let's just use the default values.

```
echo -n "Initializing project: "
npm init -y >/dev/null
if [ -f "package.json" ]; then
    echo "Found package.json!"
fi
```

2.2. Web Server with Express

`Express` is a Node.js library to manage web servers. A good tutorial starting guide can be found here: "Getting Started".

First, we need to install it.

```
npm install --save express >/dev/null

echo -n "Installing express: "
if [ "$?" -ne 0 ]; then
    echo "npm install failed, please check logs!"
    exit 1
else
    echo "npm install finished successfully!"
fi
```

A simple "Hello World" application is quite easy to set up.

```
const express = require('express')
const app = express()
const port = 3000

app.get('/', (req, res) => {
    res.send('Hello World!')
})

app.listen(port, () => {
    console.log(`Hello World app listening at http://localhost:${port}`)
})
```

The most important part of that snippet is the `app.get` function call. In general, assuming `app` is the variable referring to an `express` object, our basic primitive is an `app.METHOD(PATH, HANDLER)` function, where:

- `METHOD` is an HTTP request method, like GET, POST, etc., in lowercase
- `PATH` is the virtual path on the server (like: `example.com/PATH`)
- `HANDLER` is a function taking in request and response objects

We can start this webserver as follows:

```
node hello_world/index.js
```

In order to run this in the background, we'll need to wrap this up a bit.

```
echo -n "Starting server: "
node hello_world/index.js & &>/dev/null
node_pid="$!"
echo "Started node with PID $node_pid! Please open localhost:3000 in your browser."
```

This will start a server listening at port 3000. You can check it is running correctly with:

```
echo -n "Testing server (sleep 5 seconds): " && sleep 5

if [ "$(curl localhost:3000 2>/dev/null)" = "Hello World!" ]; then
    echo "Server is running correctly!"
else
    echo "Server is not running!"
fi
```

After we've done, we can quit the node server.

```
if ! [ -z "$node_pid" ]; then
    echo -n "Exit server: Press <enter> to exit... "
    read
    kill $node_pid
fi
```

More examples can be found on Github.

3. Sessions

By session, we mean assigning every user connecting to server with a certain session ID which determines their session. Every session can be handled differently and can have different data.

The basic motivation for sessions is: HTTP is a stateless protocol, and in order to have state (for example: user accounts, logged in users), we need to maintain sessions and associate each request with a session ID.

The main way to do that is by creating a cookie with a session ID key-value pair. In the express framework, we'll use the `express-session` middleware library.

To run this tutorial, execute `sh webserver_sessions.sh` in this directory.

3.1. Environment

First, we install the required libraries.

```
npm install --save express-session > /dev/null

echo -n "Installing express-session: "
if [ "$?" -ne 0 ]; then
    echo "npm install failed, please check logs!"
    exit 1
else
    echo "npm install finished successfully!"
fi
```

3.2. Basic Sessions

Let's try to make a simple application which uses sessions.

First, the basic constants. Note that we now have `session` as well.

```
const express = require('express')
const session = require('express-session')

const app = express()
const port = 3000
```

Now we set some properties for `express-session`. The main property is the `secret` string, which will be used to sign session IDs. Use a strong key and do not leak it!

The other properties are as follows:

- maxAge of cookie: how long a session lasts on inactivity
- resave: force update of session data on all requests, not needed
- saveUnitialized: save sessions even if no data is stored with it, not needed, but unused here

- rolling: force update of session cookie and expiration on all requests, set to true

The `resave` and `rolling` properties can look very similar, but be careful. The `rolling` property makes sure the client-side expiry is also kept up-to-date, and the `resave` forces an update of the server-side expiry. The server-side expiry is always greater or equal to the client-side expiry, but not necessarily equal. Whenever you modify the `req.session` variable in any sense, the client-side expiry will also be updated.

```
var session_options = {
    secret: "some_randomly_generated_string_here!",
    cookie: {
        secure: "auto",
        maxAge: 10000
    },
    resave: false,
    saveUninitialized: false,
    rolling: false,
}

app.use(session(session_options))
```

Now we can start returning some information about the session.

```
app.get('/', (req, res) => {
    if (!req.session.initialized) {
        req.session.initialized = 0
        res.write('New session_id created!\n')
    }

    // if you don't do this, client-side cookies will expire too soon!
    // basically, you need any change to req.session to refresh cookies client-side.
    req.session.initialized++

    res.write('session_id: ' + req.session.id + '\n')
    res.write('expires in: ' + (req.session.cookie.maxAge / 1000) + 's\n')
    res.write('expires at: ' + (req.session.cookie.expires) + '\n')

    res.end()
})

app.listen(port, () => {
    console.log(`Static Sessions app listening at http://localhost:${port}`)
})
```

Let's start the server.

```
echo -n "Starting server: "

node sessions/01-static.js & &>/dev/null
node_pid="$!"

if ! [ -z "$node_pid" ]; then
    echo "Started node with PID $node_pid! Please open localhost:3000 in your browse
    echo -n "Exit server: Press <enter> to exit... "
    read
    kill $node_pid
else
    echo "Could not start server!"
    exit 1
fi
```

3.3. Dynamic Sessions

Let's try and make the session management a little more dynamic.

This is all basic stuff:

```
const express = require('express')
const session = require('express-session')

const app = express()
const port = 3000

var session_options = {
    secret: "some_randomly_generated_string_here!",
    cookie: {
        secure: "auto",
        maxAge: 15000
    },
    resave: false,
    saveUninitialized: false,
    rolling: false,
}

app.use(session(session_options))
```

The GET response is a bit more involved now, but not too much. We check whether we've initialized the session or not, and depending on that we show two different views.

If it's uninitialized, we have a button to "log in". If we're initialized, we show session ID and seconds to expiry, and buttons to "refresh expiry" and "log out".

```
app.get('/', (req, res) => {
    res.setHeader('Content-Type', 'text/html')
    var response_html =
        "<h1>Hello World!</h1>" +
        "<hr>"
    if (!req.session.initialized) {
        response_html +=
            "<p>You are logged out!</p>" +
            "<form action='/' method='post'>" +
            "<input type='submit' name='login' value='LOG IN'/>" +
            "</form>"
    } else {
        var ms_to_expiry = req.session.original_expiry - Date.now()
        response_html +=
            "<p>You are logged in!</p>" +
            "<ul>" +
            "<li>session_id: " + req.session.id + "</li>" +
            "<li>expiry: " + ms_to_expiry/1000 + "s (" + req.session.original_expiry
            "</ul>" +
            "<form action='/' method='post'>" +
            "<input type='submit' value='REFRESH PAGE'/></br>" +
            "<input type='submit' name='refresh' value='REFRESH EXPIRY'/></br></br><
            "<input type='submit' name='logout' value='LOG OUT'/>" +
            "</form>"
    }

    res.write(response_html)
    res.end()
})
```

This is where we actually handle the POST request. We need certain middleware to parse POST bodies, which is in the first two lines.

Depending on the POST data – which is determined by the button `name` above – we have some logic to handle log in and log out, and refreshing expiry time. Remember that client-side cookie and expiry time would be updated only when the session value (aka hash) changes.

Since server-side expiry time is always updated after requests, we store the original expiry time in the session data. Date handling is ugly.

Finally, we redirect back. This is a Post/Redirect/Get pattern.

```
app.use(express.json()) // for parsing application/json
app.use(express.urlencoded({ extended: true })) // for parsing application/x-www-for

app.post('/', (req, res) => {
    if (req.body.login) {
        if (req.session.initialized) {
            console.log("ERROR: login after being initialized!")
        }
        req.session.initialized = true
        req.session.original_expiry = Date.parse(req.session.cookie.expires)
        req.session.original_expiry_str = new Date(req.session.original_expiry).toLo
        req.session.refresh_flag = false
    } else if (req.body.logout) {
        req.session.regenerate((err) => { } )
    } else if (req.body.refresh) {
        req.session.refresh_flag = !req.session.refresh_flag
        req.session.original_expiry = Date.now() + req.session.cookie.originalMaxAge
        req.session.original_expiry_str = new Date(req.session.original_expiry).toLo
    }

    res.redirect('/')
    res.end()
})

app.listen(port, () => {
    console.log(`Dynamic Sessions app listening at http://localhost:${port}`)
})
```

Again, let's start the server.

```
echo -n "Starting server: "

node sessions/02-dynamic.js & &>/dev/null
node_pid="$!"

if ! [ -z "$node_pid" ]; then
    echo "Started node with PID $node_pid! Please open localhost:3000 in your browse
    echo -n "Exit server: Press <enter> to exit... "
    read
    kill $node_pid
else
    echo "Could not start server!"
    exit 1
fi
```

3.4. Session Storage with Redis

TODO…

Date: today
Author: whatsmyname
Created: 2022-01-13 Thu 06:09
Validate