# Tutorialspoint

## Session

<https://www.tutorialspoint.com/expressjs/expressjs_sessions.htm>

HTTP is stateless; in order to associate a request to any other request, you need a way to store user data between HTTP requests. Cookies and URL parameters are both suitable ways to transport data between the client and the server. But they are both readable and on the client side. Sessions solve exactly this problem. You assign the client an ID and it makes all further requests using that ID. Information associated with the client is stored on the server linked to this ID.

We will need the *Express-session*

npm install --save express-session

We will put the **session** and **cookie-parser** middleware in place. In this example, we will use the default store for storing sessions, i.e., MemoryStore. Never use this in production environments. The session middleware handles all things for us, i.e., creating the session, setting the session cookie and creating the session object in **req** object.

Whenever we make a request from the same client again, we will have their session information stored with us (given that the server was not restarted). We can add more properties to the session object. In the following example, we will create a view counter for a client.

var express = require('express');

var cookieParser = require('cookie-parser');

var session = require('express-session');

var app = express();

app.use(cookieParser());

app.use(session({secret: "Shh, its a secret!"}));

app.get('/', function(req, res){

if(req.session.page\_views){

req.session.page\_views++;

res.send("You visited this page " + req.session.page\_views + " times");

} else {

req.session.page\_views = 1;

res.send("Welcome to this page for the first time!");

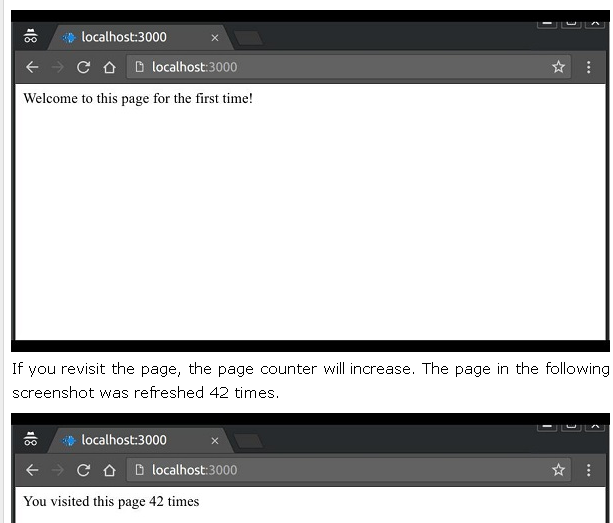
}

});

app.listen(3000);

What the above code does is, when a user visits the site, it creates a new session for the user and assigns them a cookie. Next time the user comes, the cookie is checked and the **page\_view** session variable is updated accordingly.

Now if you run the app and go to **localhost:3000**, the following output will be displayed.



# Express-session

**Reference – Read this**

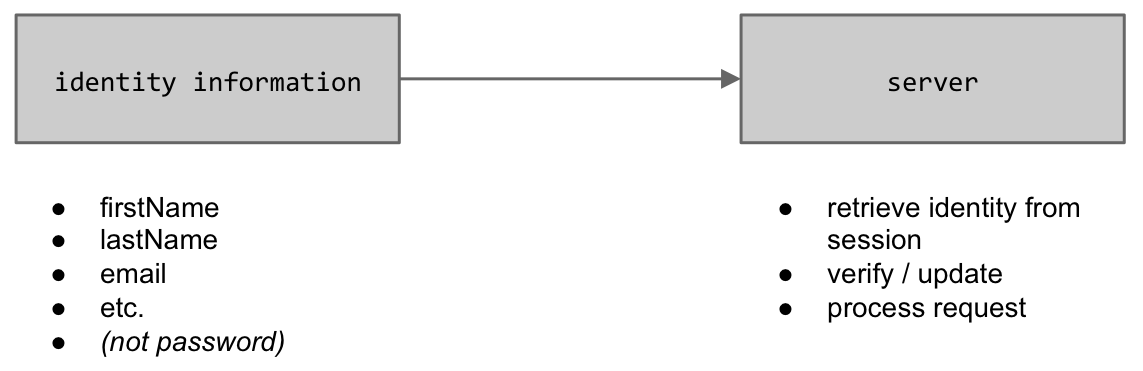
<https://expressjs.com/en/resources/middleware/session.html>

<https://expressjs.com/en/resources/middleware/cookie-session.html>

What’s In A Session Anyway?

When a user first logs in or registers for your site, you know who they are because they just submitted their information to your server. You can use that information to create a new record in your database, retrieve an existing one, or better yet – [use Stormpath](https://stormpath.com/). Simple!

But how do you keep them [authenticated](https://stormpath.com/product/authentication/) when they do something crazy like reload the page? Magic, that’s how! Also known as sessions.



A user session can be stored in two main ways with cookies: on the server or on the client. **cookie-session** module stores the session data on the client within a cookie, while a module like [**express-session**](https://www.npmjs.com/package/express-session) stores only a session identifier on the client within a cookie and stores the session data on the server, typically in a database.

The following points can help you choose which to use:

* cookie-session does not require any database / resources on the server side, though the total session data cannot exceed the browser’s max cookie size.
* cookie-session can simplify certain load-balanced scenarios.
* cookie-session can be used to store a “light” session and include an identifier to look up a database-backed secondary store to reduce database lookups.

# Example – DAPP Project

var session = require('express-session');

var mongoSessionStorage = require('connect-mongodb-session')(session);

here we have provided “api\_session” collection to strore Session at server side in MongoDB database

// -----------------------------------------------------------------

// SESSIONS: Initialize with Mongo as a storage

// -----------------------------------------------------------------

sessionStorage = new mongoSessionStorage({

uri: config.database,

collection: 'api\_session'

});

sessionStorage.on('error', function (error) {

console.log('Session init failed! ', error);

});

var sessioncfg = {

secret: config.api.session\_secret,

name: 'dapp.api.sid',

cookie: { maxAge: 1000 \* 60 \* config.api.session\_minutes },

store: sessionStorage,

proxy: true,

resave: true,

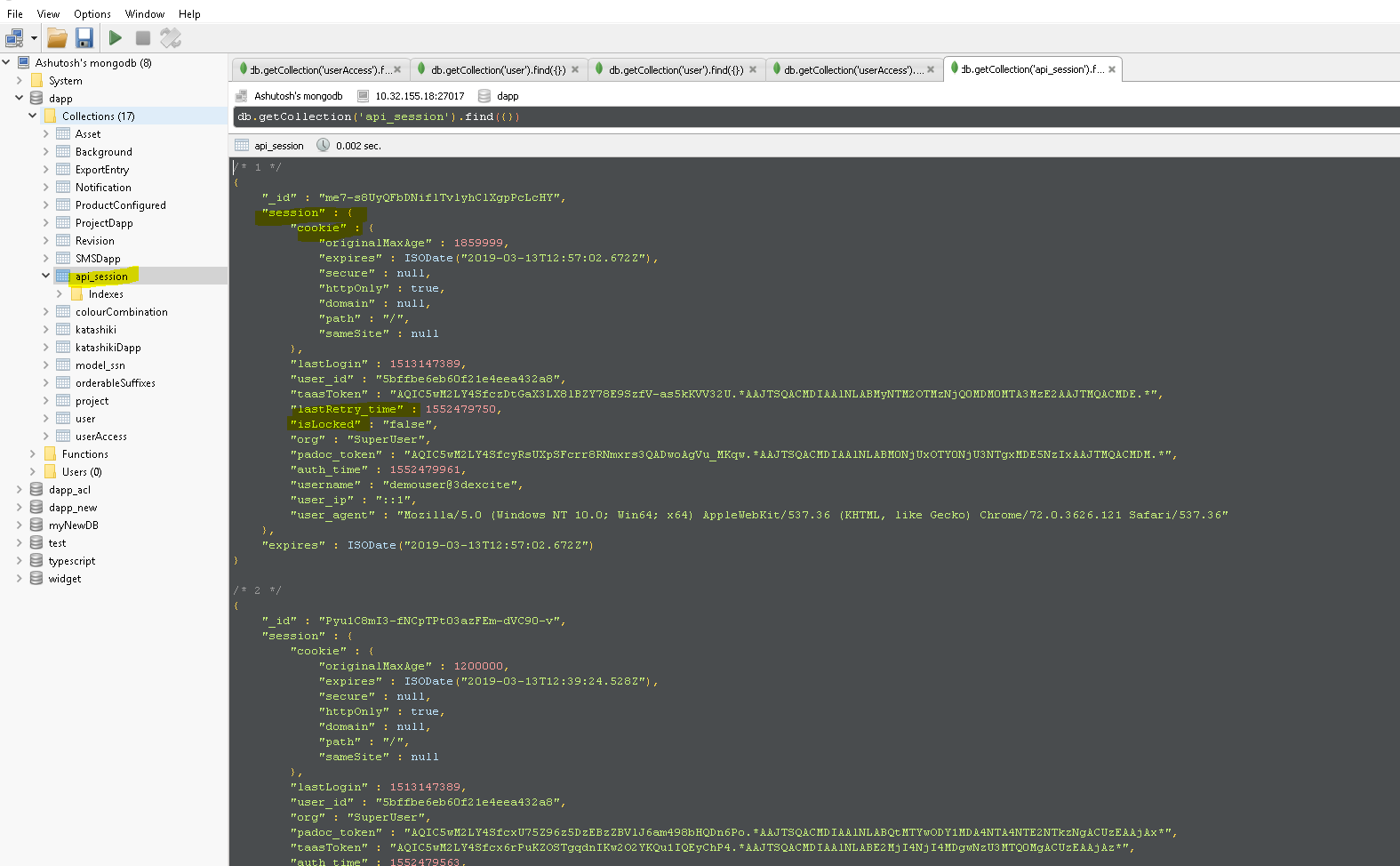
saveUninitialized: true

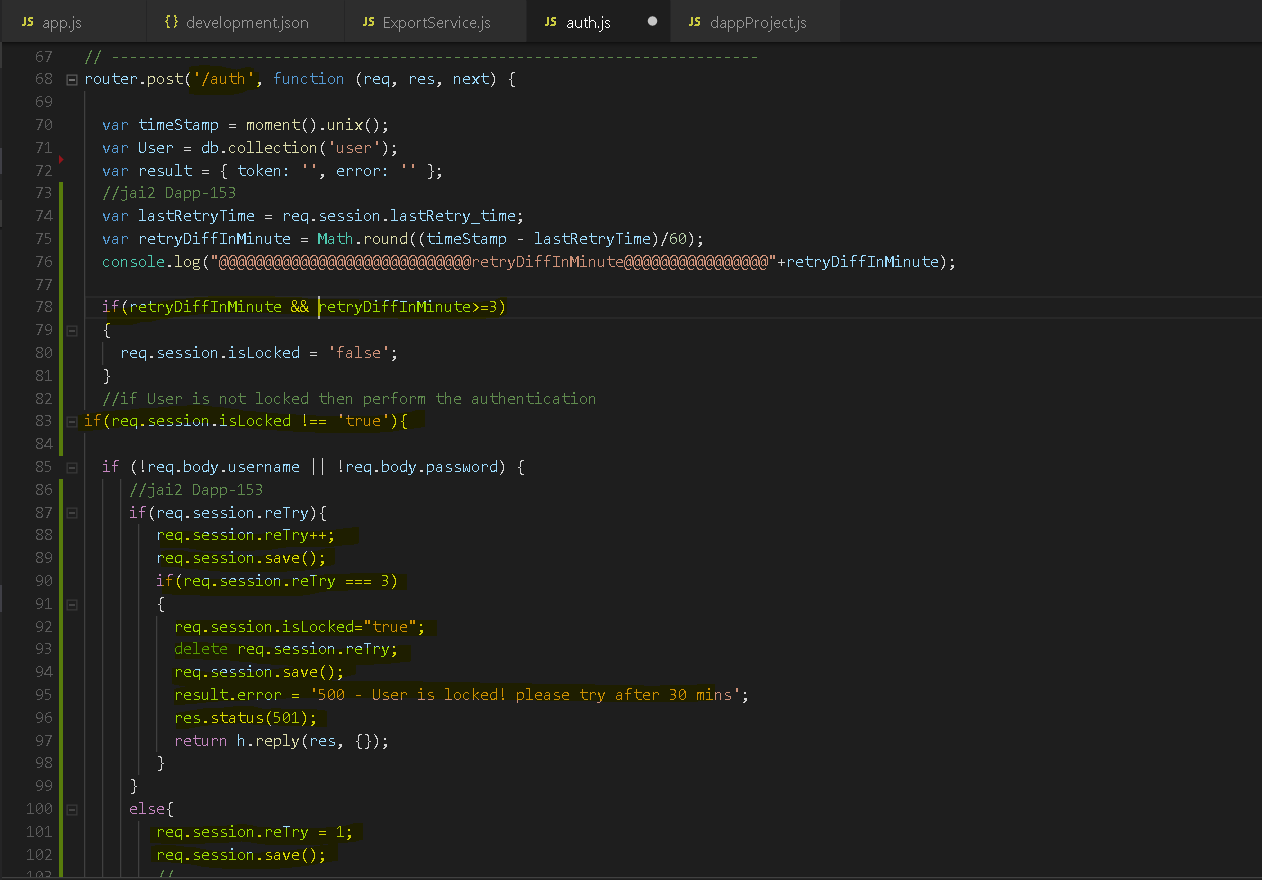
};

var sessionInstance = session(sessioncfg);

app.use(sessionInstance);

console.log('Session initialized.');



Here’s the code when user authenticate to server - 

And accordingly we send response code 200 or 500 according to user is locked or not.

User retryCount and userLocked information is saved in session storage at server side in mongodb in collection “api\_session”.

# Example cookie-session

var express = require('express');

var cookieParser = require('cookie-parser');

var bodyParser = require('body-parser');

var session = require('cookie-session');

var app = express();

// Cookie parsing needed for sessions

app.use(cookieParser('notsosecretkey'));

// Session framework

app.use(session(

{secret: 'notsosecretkey123',

name: 'session1',

// Cookie Options

maxAge: 0.5 \* 60 \* 60 \* 1000 // 0.5 hrs

}));

function getName(req, res) {

if (req.session.name) {

return res.json({ name: req.session.name , age : req.session.age});

}

else {

return res.json({ name: '' });

}

}

function setName(req, res) {

console.log("setname");

req.session.name = "cool man1";

req.session.age = 18;

return res.json({ name: req.session.name });

}

function logout(req, res) {

req.session = null;

return res.json({});

}

///////////////////////////////////////////////////////////////////////////////////////////// ROUTES

app.get('/getname', getName);

app.get('/setname', setName);

app.get('/logout', logout);

app.get('/',function(req,res){

console.log("yo");

// Update views

if(req.session.views){

req.session.views++;

}

else{

req.session.views = 1;

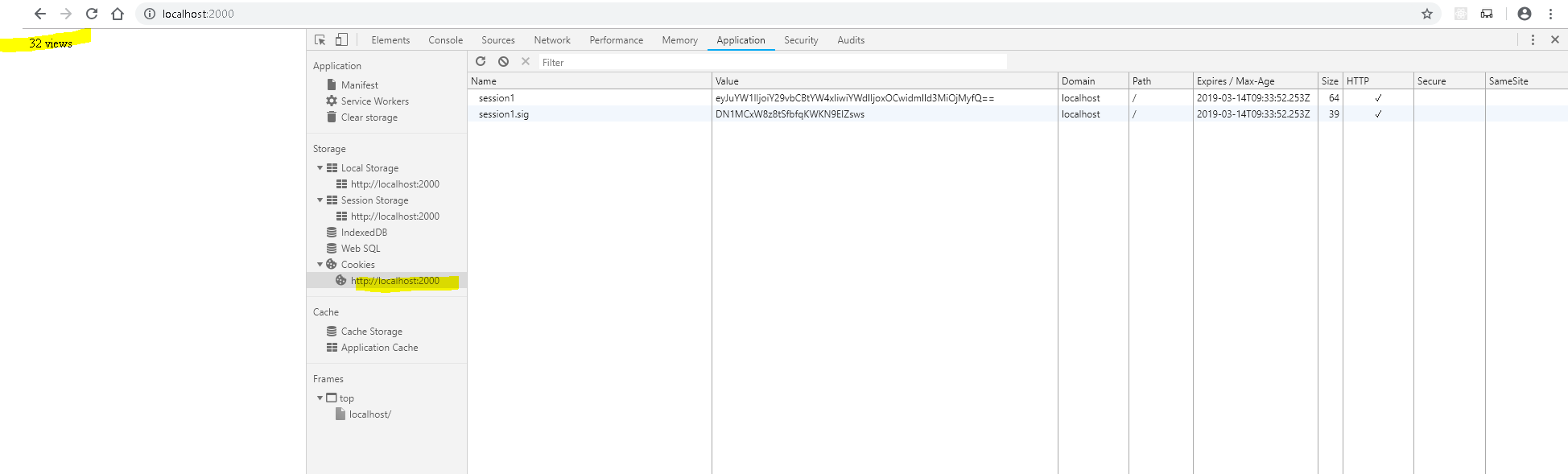
}

res.send(req.session.views + ' views');

});

var port = process.env.PORT || 2000;

app.listen(port, function () { console.log("Listening on port " + port); });

**Output -** 

# Use cookies securely

To ensure cookies don’t open your app to exploits, don’t use the default session cookie name and set cookie security options appropriately.

There are two main middleware cookie session modules:

* [express-session](https://www.npmjs.com/package/express-session) that replaces express.session middleware built-in to Express 3.x.
* [cookie-session](https://www.npmjs.com/package/cookie-session) that replaces express.cookieSession middleware built-in to Express 3.x.

The main difference between these two modules is how they save cookie session data. The [express-session](https://www.npmjs.com/package/express-session) middleware stores session data on the server; it only saves the session ID in the cookie itself, not session data. By default, it uses in-memory storage and is not designed for a production environment. In production, you’ll need to set up a scalable session-store; see the list of [compatible session stores](https://github.com/expressjs/session#compatible-session-stores).

In contrast, [cookie-session](https://www.npmjs.com/package/cookie-session) middleware implements cookie-backed storage: it serializes the entire session to the cookie, rather than just a session key. Only use it when session data is relatively small and easily encoded as primitive values (rather than objects). Although browsers are supposed to support at least 4096 bytes per cookie, to ensure you don’t exceed the limit, don’t exceed a size of 4093 bytes per domain. Also, be aware that the cookie data will be visible to the client, so if there is any reason to keep it secure or obscure, then express-session may be a better choice.

## Don’t use the default session cookie name

Using the default session cookie name can open your app to attacks. The security issue posed is similar to X-Powered-By: a potential attacker can use it to fingerprint the server and target attacks accordingly.

To avoid this problem, use generic cookie names; for example using [express-session](https://www.npmjs.com/package/express-session) middleware:

var session = require('express-session')

app.set('trust proxy', 1) // trust first proxy

app.use(session({

secret: 's3Cur3',

name: 'sessionId'

}))

## Set cookie security options

Set the following cookie options to enhance security:

* secure - Ensures the browser only sends the cookie over HTTPS.
* httpOnly - Ensures the cookie is sent only over HTTP(S), not client JavaScript, helping to protect against cross-site scripting attacks.
* domain - indicates the domain of the cookie; use it to compare against the domain of the server in which the URL is being requested. If they match, then check the path attribute next.
* path - indicates the path of the cookie; use it to compare against the request path. If this and domain match, then send the cookie in the request.
* expires - use to set expiration date for persistent cookies.

Here is an example using [cookie-session](https://www.npmjs.com/package/cookie-session) middleware:

var session = require('cookie-session')

var express = require('express')

var app = express()

var expiryDate = new Date(Date.now() + 60 \* 60 \* 1000) // 1 hour

app.use(session({

name: 'session',

keys: ['key1', 'key2'],

cookie: {

secure: true,

httpOnly: true,

domain: 'example.com',

path: 'foo/bar',

expires: expiryDate

}

}))

# Stackoverflow – Questions

For GET requests, the server assumes that you're going to be sending data back, and will automatically save session data once the route is fully processed.

For POST requests (what I'm using), however, the same assumption isn't made. Session states are only saved in one of two conditions - either when data is being sent out (through res.send, res.redirect, etc.), or if you manually call req.session.save(). I was already calling /login from an AJAX request, I just wasn't returning anything if certain conditions were met. Having the server respond to the client with some data after setting the session variable fixed this.

[**https://stackoverflow.com/questions/26531143/sessions-wont-save-in-node-js-without-req-session-save**](https://stackoverflow.com/questions/26531143/sessions-wont-save-in-node-js-without-req-session-save)