# express-session

[NPM Version](https://npmjs.org/package/express-session) [NPM Downloads](https://nodejs.org/en/download) [Build Status](https://github.com/expressjs/session/actions?query=workflow%3Aci) [Test Coverage](https://coveralls.io/r/expressjs/session?branch=master)

## Installation

This is a [Node.js](https://nodejs.org/en/) module available through the [npm registry](https://www.npmjs.com/). Installation is done using the [npm install command](https://docs.npmjs.com/getting-started/installing-npm-packages-locally):

$ npm install express-session

## API

var session = require('express-session')

### session(options)

Create a session middleware with the given options.

**Note** Session data is *not* saved in the cookie itself, just the session ID. Session data is stored server-side.

**Note** Since version 1.5.0, the [cookie-parser middleware](https://www.npmjs.com/package/cookie-parser) no longer needs to be used for this module to work. This module now directly reads and writes cookies on req/res. Using cookie-parser may result in issues if the secret is not the same between this module and cookie-parser.

**Warning** The default server-side session storage, MemoryStore, 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.

For a list of stores, see [compatible session stores](#compatible-session-stores).

#### Options

express-session accepts these properties in the options object.

##### cookie

Settings object for the session ID cookie. The default value is { path: '/', httpOnly: true, secure: false, maxAge: null }.

The following are options that can be set in this object.

##### cookie.domain

Specifies the value for the Domain Set-Cookie attribute. By default, no domain is set, and most clients will consider the cookie to apply to only the current domain.

##### cookie.expires

Specifies the Date object to be the value for the Expires Set-Cookie attribute. By default, no expiration is set, and most clients will consider this a "non-persistent cookie" and will delete it on a condition like exiting a web browser application.

**Note** If both expires and maxAge are set in the options, then the last one defined in the object is what is used.

**Note** The expires option should not be set directly; instead only use the maxAge option.

##### cookie.httpOnly

Specifies the boolean value for the HttpOnly Set-Cookie attribute. When truthy, the HttpOnly attribute is set, otherwise it is not. By default, the HttpOnly attribute is set.

**Note** be careful when setting this to true, as compliant clients will not allow client-side JavaScript to see the cookie in document.cookie.

##### cookie.maxAge

Specifies the number (in milliseconds) to use when calculating the Expires Set-Cookie attribute. This is done by taking the current server time and adding maxAge milliseconds to the value to calculate an Expires datetime. By default, no maximum age is set.

**Note** If both expires and maxAge are set in the options, then the last one defined in the object is what is used.

##### cookie.partitioned

Specifies the boolean value for the [Partitioned Set-Cookie](http://rfc-cutler-httpbis-partitioned-cookies) attribute. When truthy, the Partitioned attribute is set, otherwise it is not. By default, the Partitioned attribute is not set.

**Note** This is an attribute that has not yet been fully standardized, and may change in the future. This also means many clients may ignore this attribute until they understand it.

More information about can be found in [the proposal](https://github.com/privacycg/CHIPS).

##### cookie.path

Specifies the value for the Path Set-Cookie. By default, this is set to '/', which is the root path of the domain.

##### cookie.priority

Specifies the string to be the value for the [Priority Set-Cookie attribute](https://tools.ietf.org/html/draft-west-cookie-priority-00#section-4.1).

* 'low' will set the Priority attribute to Low.
* 'medium' will set the Priority attribute to Medium, the default priority when not set.
* 'high' will set the Priority attribute to High.

More information about the different priority levels can be found in [the specification](https://tools.ietf.org/html/draft-west-cookie-priority-00#section-4.1).

**Note** This is an attribute that has not yet been fully standardized, and may change in the future. This also means many clients may ignore this attribute until they understand it.

##### cookie.sameSite

Specifies the boolean or string to be the value for the SameSite Set-Cookie attribute. By default, this is false.

* true will set the SameSite attribute to Strict for strict same site enforcement.
* false will not set the SameSite attribute.
* 'lax' will set the SameSite attribute to Lax for lax same site enforcement.
* 'none' will set the SameSite attribute to None for an explicit cross-site cookie.
* 'strict' will set the SameSite attribute to Strict for strict same site enforcement.

More information about the different enforcement levels can be found in [the specification](https://tools.ietf.org/html/draft-ietf-httpbis-rfc6265bis-03#section-4.1.2.7).

**Note** This is an attribute that has not yet been fully standardized, and may change in the future. This also means many clients may ignore this attribute until they understand it.

**Note** There is a [draft spec](https://tools.ietf.org/html/draft-west-cookie-incrementalism-01) that requires that the Secure attribute be set to true when the SameSite attribute has been set to 'none'. Some web browsers or other clients may be adopting this specification.

##### cookie.secure

Specifies the boolean value for the Secure Set-Cookie attribute. When truthy, the Secure attribute is set, otherwise it is not. By default, the Secure attribute is not set.

**Note** be careful when setting this to true, as compliant clients will not send the cookie back to the server in the future if the browser does not have an HTTPS connection.

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:

var app = express()

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

app.use(session({

secret: 'keyboard cat',

resave: false,

saveUninitialized: true,

cookie: { secure: true }

}))

For using secure cookies in production, but allowing for testing in development, the following is an example of enabling this setup based on NODE\_ENV in express:

var app = express()

var sess = {

secret: 'keyboard cat',

cookie: {}

}

if (app.get('env') === 'production') {

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

sess.cookie.secure = true // serve secure cookies

}

app.use(session(sess))

The cookie.secure option can also be set to the special value 'auto' to have this setting automatically match the determined security of the connection. Be careful when using this setting if the site is available both as HTTP and HTTPS, as once the cookie is set on HTTPS, it will no longer be visible over HTTP. This is useful when the Express "trust proxy" setting is properly setup to simplify development vs production configuration.

##### genid

Function to call to generate a new session ID. Provide a function that returns a string that will be used as a session ID. The function is given req as the first argument if you want to use some value attached to req when generating the ID.

The default value is a function which uses the uid-safe library to generate IDs.

**NOTE** be careful to generate unique IDs so your sessions do not conflict.

app.use(session({

genid: function(req) {

return genuuid() // use UUIDs for session IDs

},

secret: 'keyboard cat'

}))

##### name

The name of the session ID cookie to set in the response (and read from in the request).

The default value is 'connect.sid'.

**Note** if you have multiple apps running on the same hostname (this is just the name, i.e. localhost or 127.0.0.1; different schemes and ports do not name a different hostname), then you need to separate the session cookies from each other. The simplest method is to simply set different names per app.

##### proxy

Trust the reverse proxy when setting secure cookies (via the "X-Forwarded-Proto" header).

The default value is undefined.

* true The "X-Forwarded-Proto" header will be used.
* false All headers are ignored and the connection is considered secure only if there is a direct TLS/SSL connection.
* undefined Uses the "trust proxy" setting from express

##### resave

Forces the session to be saved back to the session store, even if the session was never modified during the request. Depending on your store this may be necessary, but it can also create race conditions where a client makes two parallel requests to your server and changes made to the session in one request may get overwritten when the other request ends, even if it made no changes (this behavior also depends on what store you're using).

The default value is true, but using the default has been deprecated, as the default will change in the future. Please research into this setting and choose what is appropriate to your use-case. Typically, you'll want false.

How do I know if this is necessary for my store? The best way to know is to check with your store if it implements the touch method. If it does, then you can safely set resave: false. If it does not implement the touch method and your store sets an expiration date on stored sessions, then you likely need resave: true.

##### rolling

Force the session identifier cookie to be set on every response. The expiration is reset to the original [maxAge](#cookiemaxage), resetting the expiration countdown.

The default value is false.

With this enabled, the session identifier cookie will expire in [maxAge](#cookiemaxage) since the last response was sent instead of in [maxAge](#cookiemaxage) since the session was last modified by the server.

This is typically used in conjuction with short, non-session-length [maxAge](#cookiemaxage) values to provide a quick timeout of the session data with reduced potential of it occurring during on going server interactions.

**Note** When this option is set to true but the saveUninitialized option is set to false, the cookie will not be set on a response with an uninitialized session. This option only modifies the behavior when an existing session was loaded for the request.

##### saveUninitialized

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.

The default value is true, but using the default has been deprecated, as the default will change in the future. Please research into this setting and choose what is appropriate to your use-case.

**Note** if you are using Session in conjunction with PassportJS, Passport will add an empty Passport object to the session for use after a user is authenticated, which will be treated as a modification to the session, causing it to be saved. *This has been fixed in PassportJS 0.3.0*

##### secret

**Required option**

This is the secret used to sign the session ID cookie. The secret can be any type of value that is supported by Node.js crypto.createHmac (like a string or a Buffer). This can be either 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. A best practice may include:

* The use of environment variables to store the secret, ensuring the secret itself does not exist in your repository.
* Periodic updates of the secret, while ensuring the previous secret is in the array.

Using a secret that cannot be guessed will reduce the ability to hijack a session to only guessing the session ID (as determined by the genid option).

Changing the secret value will invalidate all existing sessions. In order to rotate the secret without invalidating sessions, provide an array of secrets, with the new secret as first element of the array, and including previous secrets as the later elements.

**Note** HMAC-256 is used to sign the session ID. For this reason, the secret should contain at least 32 bytes of entropy.

##### store

The session store instance, defaults to a new MemoryStore instance.

##### unset

Control the result of unsetting req.session (through delete, setting to null, etc.).

The default value is 'keep'.

* 'destroy' The session will be destroyed (deleted) when the response ends.
* 'keep' The session in the store will be kept, but modifications made during the request are ignored and not saved.

### req.session

To store or access session data, simply use the request property req.session, which is (generally) serialized as JSON by the store, so nested objects are typically fine. For example below is a user-specific view counter:

// Use the session middleware

app.use(session({ secret: 'keyboard cat', cookie: { maxAge: 60000 }}))

// Access the session as req.session

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

if (req.session.views) {

req.session.views++

res.setHeader('Content-Type', 'text/html')

res.write('<p>views: ' + req.session.views + '</p>')

res.write('<p>expires in: ' + (req.session.cookie.maxAge / 1000) + 's</p>')

res.end()

} else {

req.session.views = 1

res.end('welcome to the session demo. refresh!')

}

})

#### Session.regenerate(callback)

To regenerate the session simply invoke the method. Once complete, a new SID and Session instance will be initialized at req.session and the callback will be invoked.

req.session.regenerate(function(err) {

// will have a new session here

})

#### Session.destroy(callback)

Destroys the session and will unset the req.session property. Once complete, the callback will be invoked.

req.session.destroy(function(err) {

// cannot access session here

})

#### Session.reload(callback)

Reloads the session data from the store and re-populates the req.session object. Once complete, the callback will be invoked.

req.session.reload(function(err) {

// session updated

})

#### Session.save(callback)

Save the session back to the store, replacing the contents on the store with the contents in memory (though a store may do something else--consult the store's documentation for exact behavior).

This method is automatically called at the end of the HTTP response if the session data has been altered (though this behavior can be altered with various options in the middleware constructor). Because of this, typically this method does not need to be called.

There are some cases where it is useful to call this method, for example, redirects, long-lived requests or in WebSockets.

req.session.save(function(err) {

// session saved

})

#### Session.touch()

Updates the .maxAge property. Typically this is not necessary to call, as the session middleware does this for you.

### req.session.id

Each session has a unique ID associated with it. This property is an alias of [req.sessionID](#reqsessionid-1) and cannot be modified. It has been added to make the session ID accessible from the session object.

### req.session.cookie

Each session has a unique cookie object accompany it. This allows you to alter the session cookie per visitor. For example we can set req.session.cookie.expires to false to enable the cookie to remain for only the duration of the user-agent.

#### Cookie.maxAge

Alternatively req.session.cookie.maxAge will return the time remaining in milliseconds, which we may also re-assign a new value to adjust the .expires property appropriately. The following are essentially equivalent

var hour = 3600000

req.session.cookie.expires = new Date(Date.now() + hour)

req.session.cookie.maxAge = hour

For example when maxAge is set to 60000 (one minute), and 30 seconds has elapsed it will return 30000 until the current request has completed, at which time req.session.touch() is called to reset req.session.cookie.maxAge to its original value.

req.session.cookie.maxAge // => 30000

#### Cookie.originalMaxAge

The req.session.cookie.originalMaxAge property returns the original maxAge (time-to-live), in milliseconds, of the session cookie.

### req.sessionID

To get the ID of the loaded session, access the request property req.sessionID. This is simply a read-only value set when a session is loaded/created.

## Session Store Implementation

Every session store *must* be an EventEmitter and implement specific methods. The following methods are the list of **required**, **recommended**, and **optional**.

* Required methods are ones that this module will always call on the store.
* Recommended methods are ones that this module will call on the store if available.
* Optional methods are ones this module does not call at all, but helps present uniform stores to users.

For an example implementation view the [connect-redis](http://github.com/visionmedia/connect-redis) repo.

### store.all(callback)

**Optional**

This optional method is used to get all sessions in the store as an array. The callback should be called as callback(error, sessions).

### store.destroy(sid, callback)

**Required**

This required method is used to destroy/delete a session from the store given a session ID (sid). The callback should be called as callback(error) once the session is destroyed.

### store.clear(callback)

**Optional**

This optional method is used to delete all sessions from the store. The callback should be called as callback(error) once the store is cleared.

### store.length(callback)

**Optional**

This optional method is used to get the count of all sessions in the store. The callback should be called as callback(error, len).

### store.get(sid, callback)

**Required**

This required method is used to get a session from the store given a session ID (sid). The callback should be called as callback(error, session).

The session argument should be a session if found, otherwise null or undefined if the session was not found (and there was no error). A special case is made when error.code === 'ENOENT' to act like callback(null, null).

### store.set(sid, session, callback)

**Required**

This required method is used to upsert a session into the store given a session ID (sid) and session (session) object. The callback should be called as callback(error) once the session has been set in the store.

### store.touch(sid, session, callback)

**Recommended**

This recommended method is used to "touch" a given session given a session ID (sid) and session (session) object. The callback should be called as callback(error) once the session has been touched.

This is primarily used when the store will automatically delete idle sessions and this method is used to signal to the store the given session is active, potentially resetting the idle timer.

## Compatible Session Stores

The following modules implement a session store that is compatible with this module. Please make a PR to add additional modules :)

[★ aerospike-session-store](https://www.npmjs.com/package/aerospike-session-store) A session store using [Aerospike](http://www.aerospike.com/).

[★ better-sqlite3-session-store](https://www.npmjs.com/package/better-sqlite3-session-store) A session store based on [better-sqlite3](https://github.com/JoshuaWise/better-sqlite3).

[★ cassandra-store](https://www.npmjs.com/package/cassandra-store) An Apache Cassandra-based session store.

[★ cluster-store](https://www.npmjs.com/package/cluster-store) A wrapper for using in-process / embedded stores - such as SQLite (via knex), leveldb, files, or memory - with node cluster (desirable for Raspberry Pi 2 and other multi-core embedded devices).

[★ connect-arango](https://www.npmjs.com/package/connect-arango) An ArangoDB-based session store.

[★ connect-azuretables](https://www.npmjs.com/package/connect-azuretables) An [Azure Table Storage](https://azure.microsoft.com/en-gb/services/storage/tables/)-based session store.

[★ connect-cloudant-store](https://www.npmjs.com/package/connect-cloudant-store) An [IBM Cloudant](https://cloudant.com/)-based session store.

[★ connect-cosmosdb](https://www.npmjs.com/package/connect-cosmosdb) An Azure [Cosmos DB](https://azure.microsoft.com/en-us/products/cosmos-db/)-based session store.

[★ connect-couchbase](https://www.npmjs.com/package/connect-couchbase) A [couchbase](http://www.couchbase.com/)-based session store.

[★ connect-datacache](https://www.npmjs.com/package/connect-datacache) An [IBM Bluemix Data Cache](http://www.ibm.com/cloud-computing/bluemix/)-based session store.

[★ @google-cloud/connect-datastore](https://www.npmjs.com/package/@google-cloud/connect-datastore) A [Google Cloud Datastore](https://cloud.google.com/datastore/docs/concepts/overview)-based session store.

[★ connect-db2](https://www.npmjs.com/package/connect-db2) An IBM DB2-based session store built using [ibm\_db](https://www.npmjs.com/package/ibm_db) module.

[★ connect-dynamodb](https://www.npmjs.com/package/connect-dynamodb) A DynamoDB-based session store.

[★ @google-cloud/connect-firestore](https://www.npmjs.com/package/@google-cloud/connect-firestore) A [Google Cloud Firestore](https://cloud.google.com/firestore/docs/overview)-based session store.

[★ connect-hazelcast](https://www.npmjs.com/package/connect-hazelcast) Hazelcast session store for Connect and Express.

[★ connect-loki](https://www.npmjs.com/package/connect-loki) A Loki.js-based session store.

[★ connect-lowdb](https://www.npmjs.com/package/connect-lowdb) A lowdb-based session store.

[★ connect-memcached](https://www.npmjs.com/package/connect-memcached) A memcached-based session store.

[★ connect-memjs](https://www.npmjs.com/package/connect-memjs) A memcached-based session store using [memjs](https://www.npmjs.com/package/memjs) as the memcached client.

[★ connect-ml](https://www.npmjs.com/package/connect-ml) A MarkLogic Server-based session store.

[★ connect-monetdb](https://www.npmjs.com/package/connect-monetdb) A MonetDB-based session store.

[★ connect-mongo](https://www.npmjs.com/package/connect-mongo) A MongoDB-based session store.

[★ connect-mongodb-session](https://www.npmjs.com/package/connect-mongodb-session) Lightweight MongoDB-based session store built and maintained by MongoDB.

[★ connect-mssql-v2](https://www.npmjs.com/package/connect-mssql-v2) A Microsoft SQL Server-based session store based on [connect-mssql](https://www.npmjs.com/package/connect-mssql).

[★ connect-neo4j](https://www.npmjs.com/package/connect-neo4j) A [Neo4j](https://neo4j.com)-based session store.

[★ connect-ottoman](https://www.npmjs.com/package/connect-ottoman) A [couchbase ottoman](http://www.couchbase.com/)-based session store.

[★ connect-pg-simple](https://www.npmjs.com/package/connect-pg-simple) A PostgreSQL-based session store.

[★ connect-redis](https://www.npmjs.com/package/connect-redis) A Redis-based session store.

[★ connect-session-firebase](https://www.npmjs.com/package/connect-session-firebase) A session store based on the [Firebase Realtime Database](https://firebase.google.com/docs/database/)

[★ connect-session-knex](https://www.npmjs.com/package/connect-session-knex) A session store using [Knex.js](http://knexjs.org/), which is a SQL query builder for PostgreSQL, MySQL, MariaDB, SQLite3, and Oracle.

[★ connect-session-sequelize](https://www.npmjs.com/package/connect-session-sequelize) A session store using [Sequelize.js](http://sequelizejs.com/), which is a Node.js / io.js ORM for PostgreSQL, MySQL, SQLite and MSSQL.

[★ connect-sqlite3](https://www.npmjs.com/package/connect-sqlite3) A [SQLite3](https://github.com/mapbox/node-sqlite3) session store modeled after the TJ's connect-redis store.

[★ connect-typeorm](https://www.npmjs.com/package/connect-typeorm) A [TypeORM](https://github.com/typeorm/typeorm)-based session store.

[★ couchdb-expression](https://www.npmjs.com/package/couchdb-expression) A [CouchDB](https://couchdb.apache.org/)-based session store.

[★ dynamodb-store](https://www.npmjs.com/package/dynamodb-store) A DynamoDB-based session store.

[★ dynamodb-store-v3](https://www.npmjs.com/package/dynamodb-store-v3) Implementation of a session store using DynamoDB backed by the [AWS SDK for JavaScript v3](https://github.com/aws/aws-sdk-js-v3).

[★ express-etcd](https://www.npmjs.com/package/express-etcd) An [etcd](https://github.com/stianeikeland/node-etcd) based session store.

[★ express-mysql-session](https://www.npmjs.com/package/express-mysql-session) A session store using native [MySQL](https://www.mysql.com/) via the [node-mysql](https://github.com/felixge/node-mysql) module.

[★ express-nedb-session](https://www.npmjs.com/package/express-nedb-session) A NeDB-based session store.

[★ express-oracle-session](https://www.npmjs.com/package/express-oracle-session) A session store using native [oracle](https://www.oracle.com/) via the [node-oracledb](https://www.npmjs.com/package/oracledb) module.

[★ express-session-cache-manager](https://www.npmjs.com/package/express-session-cache-manager) A store that implements [cache-manager](https://www.npmjs.com/package/cache-manager), which supports a [variety of storage types](https://www.npmjs.com/package/cache-manager#store-engines).

[★ express-session-etcd3](https://www.npmjs.com/package/express-session-etcd3) An [etcd3](https://github.com/mixer/etcd3) based session store.

[★ express-session-level](https://www.npmjs.com/package/express-session-level) A [LevelDB](https://github.com/Level/levelup) based session store.

[★ express-session-rsdb](https://www.npmjs.com/package/express-session-rsdb) Session store based on Rocket-Store: A very simple, super fast and yet powerfull, flat file database.

[★ express-sessions](https://www.npmjs.com/package/express-sessions) A session store supporting both MongoDB and Redis.

[★ firestore-store](https://www.npmjs.com/package/firestore-store) A [Firestore](https://github.com/hendrysadrak/firestore-store)-based session store.

[★ fortune-session](https://www.npmjs.com/package/fortune-session) A [Fortune.js](https://github.com/fortunejs/fortune) based session store. Supports all backends supported by Fortune (MongoDB, Redis, Postgres, NeDB).

[★ hazelcast-store](https://www.npmjs.com/package/hazelcast-store) A Hazelcast-based session store built on the [Hazelcast Node Client](https://www.npmjs.com/package/hazelcast-client).

[★ level-session-store](https://www.npmjs.com/package/level-session-store) A LevelDB-based session store.

[★ lowdb-session-store](https://www.npmjs.com/package/lowdb-session-store) A [lowdb](https://www.npmjs.com/package/lowdb)-based session store.

[★ medea-session-store](https://www.npmjs.com/package/medea-session-store) A Medea-based session store.

[★ memorystore](https://www.npmjs.com/package/memorystore) A memory session store made for production.

[★ mssql-session-store](https://www.npmjs.com/package/mssql-session-store) A SQL Server-based session store.

[★ nedb-session-store](https://www.npmjs.com/package/nedb-session-store) An alternate NeDB-based (either in-memory or file-persisted) session store.

[★ @quixo3/prisma-session-store](https://www.npmjs.com/package/@quixo3/prisma-session-store) A session store for the [Prisma Framework](https://www.prisma.io).

[★ restsession](https://www.npmjs.com/package/restsession) Store sessions utilizing a RESTful API

[★ sequelstore-connect](https://www.npmjs.com/package/sequelstore-connect) A session store using [Sequelize.js](http://sequelizejs.com/).

[★ session-file-store](https://www.npmjs.com/package/session-file-store) A file system-based session store.

[★ session-pouchdb-store](https://www.npmjs.com/package/session-pouchdb-store) Session store for PouchDB / CouchDB. Accepts embedded, custom, or remote PouchDB instance and realtime synchronization.

[★ @cyclic.sh/session-store](https://www.npmjs.com/package/@cyclic.sh/session-store) A DynamoDB-based session store for [Cyclic.sh](https://www.cyclic.sh/) apps.

[★ @databunker/session-store](https://www.npmjs.com/package/@databunker/session-store) A [Databunker](https://databunker.org/)-based encrypted session store.

[★ sessionstore](https://www.npmjs.com/package/sessionstore) A session store that works with various databases.

[★ tch-nedb-session](https://www.npmjs.com/package/tch-nedb-session) A file system session store based on NeDB.

## Examples

### View counter

A simple example using express-session to store page views for a user.

var express = require('express')

var parseurl = require('parseurl')

var session = require('express-session')

var app = express()

app.use(session({

secret: 'keyboard cat',

resave: false,

saveUninitialized: true

}))

app.use(function (req, res, next) {

if (!req.session.views) {

req.session.views = {}

}

// get the url pathname

var pathname = parseurl(req).pathname

// count the views

req.session.views[pathname] = (req.session.views[pathname] || 0) + 1

next()

})

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

res.send('you viewed this page ' + req.session.views['/foo'] + ' times')

})

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

res.send('you viewed this page ' + req.session.views['/bar'] + ' times')

})

app.listen(3000)

### User login

A simple example using express-session to keep a user log in session.

var escapeHtml = require('escape-html')

var express = require('express')

var session = require('express-session')

var app = express()

app.use(session({

secret: 'keyboard cat',

resave: false,

saveUninitialized: true

}))

// middleware to test if authenticated

function isAuthenticated (req, res, next) {

if (req.session.user) next()

else next('route')

}

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

// this is only called when there is an authentication user due to isAuthenticated

res.send('hello, ' + escapeHtml(req.session.user) + '!' +

' <a href="/logout">Logout</a>')

})

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

res.send('<form action="/login" method="post">' +

'Username: <input name="user"><br>' +

'Password: <input name="pass" type="password"><br>' +

'<input type="submit" text="Login"></form>')

})

app.post('/login', express.urlencoded({ extended: false }), function (req, res) {

// login logic to validate req.body.user and req.body.pass

// would be implemented here. for this example any combo works

// regenerate the session, which is good practice to help

// guard against forms of session fixation

req.session.regenerate(function (err) {

if (err) next(err)

// store user information in session, typically a user id

req.session.user = req.body.user

// save the session before redirection to ensure page

// load does not happen before session is saved

req.session.save(function (err) {

if (err) return next(err)

res.redirect('/')

})

})

})

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

// logout logic

// clear the user from the session object and save.

// this will ensure that re-using the old session id

// does not have a logged in user

req.session.user = null

req.session.save(function (err) {

if (err) next(err)

// regenerate the session, which is good practice to help

// guard against forms of session fixation

req.session.regenerate(function (err) {

if (err) next(err)

res.redirect('/')

})

})

})

app.listen(3000)

## Debugging

This module uses the [debug](https://www.npmjs.com/package/debug) module internally to log information about session operations.

To see all the internal logs, set the DEBUG environment variable to express-session when launching your app (npm start, in this example):

$ DEBUG=express-session npm start

On Windows, use the corresponding command;

> set DEBUG=express-session & npm start

## License

[MIT](http://license)