Express Sessions

about session and using them with express

Lesson Plan

- The problems sessions are solving
- What are sessions and how they work
- Session store options
- express-session middleware
- logging the user with session
- CSRF
- redis store
- db store

Lesson URL:

https://www.nerdeez.com/articles/express/express-sessions

The problem

- http is a request response protocol built on TCP/IP
- http is stateless
- web applications are stateful, we need to know the user logged in then wants to go to his wall and then to the settings page and we need to know the logged in user when moving from page to page
- We are going to use cookies to solve this problem

what are Cookies

- key value data stored in the client
- that data usually set by the server with the **Set-Cookie** headers
  - the data can have optional expiration
  - the data can have optional domains to be sent
- the data is sent from the client back to the server
  - if not expired
  - if the domain match
- data can be encrypted and read only be the server
- data is limited to 4k
- data is always sent in the headers which slows communication

Sessions - solving the problem

- using cookies we can make http from stateless to stateful
- server can send an identifier to the client as cookie
- the identifier will be associated with data for that user
- each request the identifier will be sent and we will know the user that sends the request
- we can save in the session if the user is logged in

Where session data is stores

- memory
  - good for development
  - not used in production
  - can cause memory leakage
  - on crash session data is lost
- cookie - data is not saved rather we keep adding data to the cookie
  - make requests and response larger
  - size of data is limited to 4k
  - client can read the data (data can be encrypted)
- db - data can be saved in the database
  - slows down request/response
  - more work on db
- in memory db - like redis, memcached
  - considered to be the best solution
  - need to create high availability server more maintenance and work

express-session middleware

- express has a popular middleware which takes care of sessions
- can be configured to work with different stores
- by default will store session data in memory
- after installing the middleware will add an object **req.session** where you can add key value data to that object to set session data
- you can delete keys with the **delete** keyword
- when using the middleware you can add configuration object
  - **secret -** this is required
  - **resave -** set to false
- lets create an express app and install the middleware
- lets create a login which sets a session key when login is success and redirect to a welcome page
- the welcome page will redirect to the login page is the user did not log in.

postgres store

- express-session is modular and you can connect different stores for the session data
- to store the data in postgres database we will need the store **connect-pg-simple**
- the above will return a class factory the factory gets the session middleware and returns a class
- you create a new instance of that class where the constructor gets an object with the options
    - **conString** - will hold the connection string to the database
- this will create a database called **session** with the identifier, data, and expiration

Redis Store

- Install Redis
  - brew install redis
- Activate redis server
  - /usr/local/Cellar/redis/4.0.11/bin/redis-server
- Install the redis store
  - **npm install connect-redis**
- the package is a class factory which gets the session middleware
- you attach to the store option of the session middleware a new instance of that class
- the constructor of the class will get a **url** option with the url of the redis server
- Lets try and connect the previous example to redis

CSRF

- a security vulnerability usually exploited by phishing sites
- since cookies of domain A will automatically sent when domain A request is sent, I can send a request from domain B to domain A and the same cookies will be sent
- I can send a request from domain B that the client will not be aware of.
- This session exploit is called CSRF

Solving CSRF

- domain B can send a request in behalf of domain A and the cookies will be sent
- What domain B cannot do is read the content of the cookies
- The idea is to send a token in the cookies and the one who can read the cookies (domain A) is in charge to pass a token in the cookies to a special csrf header
- the server will read the csrf header and will know if its a valid one, meaning is the request sent from domain A

CSURF

- there is an express middleware that helps us solve the CSRF vulnerability
- you can install it with npm
  - **npm install csurf**
- Install it as a middleware
- after you install it you can use
  - **req.csrfToken()**
- you will need to pass it along in the forms you submit, so lets modify the login form
- add a hidden input with value of the token created and name **_csrf**

Summary

- you starting to see the advantage of middlewares in express and the large community and many packages
- session are easily connected by a middleware and so are the different way to store them

EX.

- connect you login app to a session
- connect it to a db store or redis store
- pass data in the session that will be printed in the welcome page