STATE MANAGEMENT & SESSION TRACKING

State Management

- What to do when server data gets updated?
- How to refresh our user Interface?
- What happens if the user refreshes the browser URL?
- Should we reload all the data?

State Management

- Common ways to handle state related questions.
- Defines data flow

STATE MANAGEMENT

- Server side scripting:

  client –server communication –http

  http-stateless

  unable to carry data

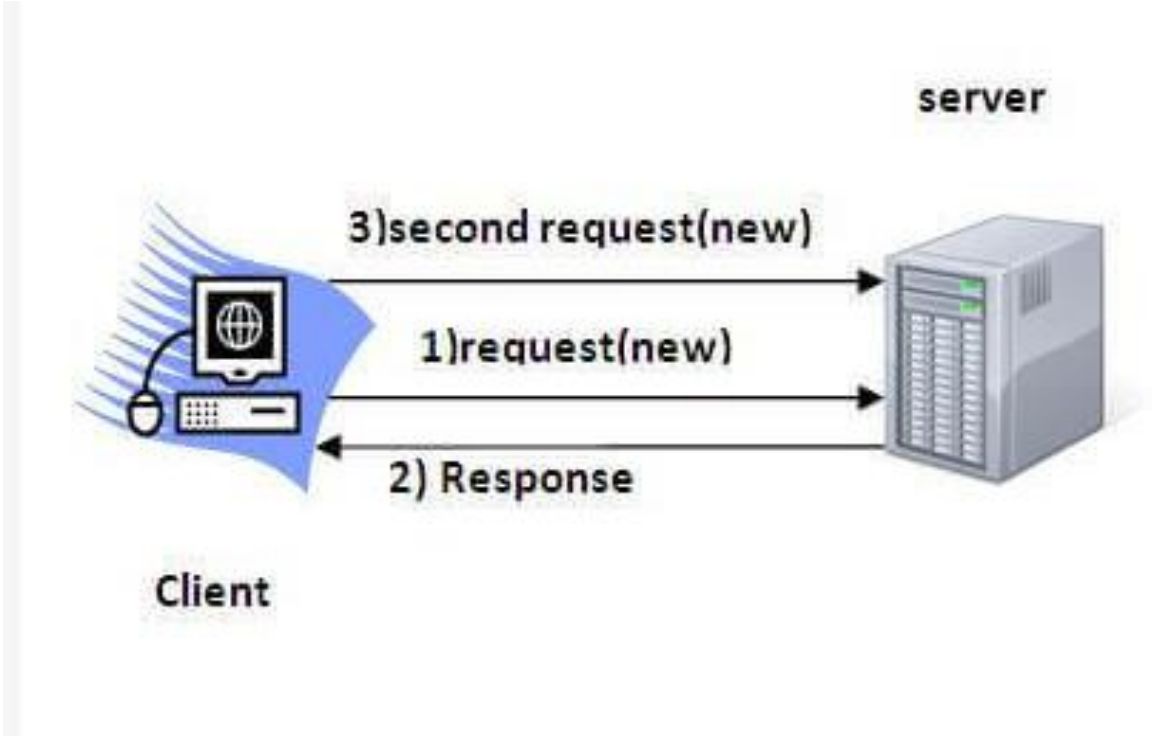
STATE MANAGEMENT

- Eg: when user enters data in webform and when it is submitted the http will not carry any data ,so to maintain those data we have a concept of state management.
- Two types: 1. client side :cookie
  2.Server side: sessions.

SESSION TRACKING

- **Session** simply means a particular interval of time.
- **Session Tracking** is a way to maintain state (data) of an user. It is also known as **session management** in servlet.
- HTTP is stateless that means each request is considered as the new request.

SESSION TRACKING



WHY SESSION?

- **To recognize the user .**
- When there is a need to maintain the conversational state, session tracking is needed.
- For example, in a shopping cart application a client keeps on adding items into his cart using multiple requests.
- When every request is made, the server should identify in which client's cart the item is to be added. So in this scenario, there is a certain need for session tracking.
- Solution is, when a client makes a request it should introduce itself by providing unique identifier every time.
- There are five different methods to achieve this.

SESSION TRACKING METHODS:

- User authorization
- Hidden fields
- URL rewriting
- Cookies
- Session tracking API

USER AUTHORIZATION

- Users can be authorized to use the web application in different ways. Basic concept is that the user will provide username and password to login to the application. Based on that the user can be identified and the session can be maintained.

HIDDEN FIELDS

- <INPUT TYPE=”hidden” NAME=”technology” VALUE=”servlet”>

- Hidden fields like the above can be inserted in the webpages and information can be sent to the server for session tracking.
- These fields are not visible directly to the user, but can be viewed using view source option from the browsers.
- This type doesn’t need any special configuration from the browser of server and by default available to use for session tracking.
- This cannot be used for session tracking when the conversation included static resources lik html pages.

URL REWRITING

- When a request is made, additional parameter is appended with the url.
- In general added additional parameter will be sessionid or sometimes the userid.
- It will suffice to track the session.
- This type of session tracking doesn't need any special support from the browser.
- Disadvantage is, implementing this type of session tracking is tedious. We need to keep track of the parameter as a chain link until the conversation completes and also should make sure that, the parameter doesn't clash with other application parameters.

URL REWRITING

- Original
  URL: http://server:port/servlet/ServletName
  Rewritten
  URL: http://server:port/servlet/ServletName
  ?sessionid=7456

COOKIES

- Cookies are the mostly used technology for session tracking.
- Cookie is a key value pair of information, sent by the server to the browser.
- This should be saved by the browser in its space in the client computer.
- Whenever the browser sends a request to that server it sends the cookie along with it.
- Then the server can identify the client using the cookie.

COOKIES

- Session tracking is easy to implement and maintain using the cookies.
- Disadvantage is that, the users can opt to disable cookies using their browser preferences. In such case, the browser will not save the cookie at client computer and session tracking fails.

SESSION TRACKING API

- Session tracking API is built on top of the first four methods.
- This is in order to help the developer to minimize the overhead of session tracking.

USING EXPRESS FRAMEWORK

session.js | sessionview.js | express2.js

```
1  var express= require('express');
2  var bodyParser = require('body-parser');
3  var cookieparser= require('cookie-parser');
4
5  var app= express();
6  app.use(cookieparser());
7  var urlencodedParser = bodyParser.urlencoded({ extended: false })
8  app.get('/', function (req,res) {
9      res.cookie('mycookie','hi this is my cookie');
10     res.end('hi welcome');
11 });
12
13 app.get('/remove', function(req,res)
14 {
15     res.clearCookie('mycookie');
16 });
17
18 app.listen(3000,'127.0.0.1');
19 console.log('you are listening to 3000');
20
```

MAXAGE-COOKIE

session.js | sessionview.js

```
1  var express = require('express');
2  var cookieParser = require('cookie-parser');
3  var session = require('express-session');
4
5  var app = express();
6
7  app.use(cookieParser());
8  app.use(session({secret: "secret!" ,saveUninitialized:'true',resave:'true',cookie:{maxAge
9
10 app.get('/', function(req, res){
11    if(req.session.page_views){
12       req.session.page_views++;
13       res.send("You visited this page " + req.session.page_views + " times");
14    } else {
15       req.session.page_views = 1;
16       res.send("Welcome to this page for the first time!");
17    }
18 });
19 app.listen(3000);
```