# Overview

- HTTP is a stateless protocol.
- Need some way to tie multiple HTTP request/response pairs together: **State**.
- State can be maintained within a client, or the server or a combination of both.
- State maintained on the server: **session**.
- Client needs to provide some kind of **session-id** to reference state maintained on server.
- Will use a shopping cart as example of state.

# Client State

- The state is maintained entirely on the client.
- Example: authentication information for a particular authentication realm for `Basic` authentication. Browser caches user credentials (state) and resubmits whenever it receives a 403 `UNAUTHORIZED` challenge from the server.
- Modern browsers also provide *session storage*, *local storage* and `indexedDB`.
- Client needs to transmit state to the server on each request. Puts a limit on the size of the state. Transmitting a shopping cart back-and-forth may be ok WRT size.
- Easy to tamper with the state by user or by other programs (a security problem if other programs not authorized by user). With shopping cart, based entirely on client state, user could tamper with shopping cart to change prices!

# Client State Continued

- If state is critical, then it needs to be cryptographically sealed so as to be tamper-proof. If shopping cart is sealed, then user cannot tamper with it.
- User preferences would be an example of non-critical state which can be stored non-securely on a client.

# Server State

- **Persistent state** is state which is maintained in persistent storage like a database or filesystem.
- An advantage of persistent state is that it is durable; databases make strong guarantees about not loosing data.
- A disadvantage of persistent state is that it is slow; a database usually involves some form of *inter-process communication* (IPC) often across a network.
- **In-memory state** is maintained within the memory of the server.
- An advantage of in-memory state is that it is fast.
- A serious disadvantage of in-memory state is that it is not durable; will not survive server restarts or crashes.
- A compromise is to use in-memory state purely as a cache for persistent state. Decide on *write-back* vs *write-through* cache semantics.

# Server APIs

Many web server APIs usually support 4 objects:

**Application Context** The overall server state shared across multiple requests/responses for all users.

**Session Context** Session state shared across multiple requests/responses for a session usually corresponding to a single user.

**Request** The request object. Used for a single request-response cycle. Will contain details of the URL, query parameters, request headers and body. May also allow the application to cache information which is needed for only a single request-response cycle.

**Response** The response object. Used for a single request-response cycle. Will contain details of the response status, response headers and will allow building up of the response body.

# Session Ids

- If state is maintained on the server, the client needs to send some kind of session-id with each request to identify the state on the server.
- Session-id is basically client-side state which serves as a proxy for server-side state.
- Since session-id is client-side state, usual problems with tampering.
- Early versions of tomcat generated sequential session ids!!
- Minimally, session ids should contain some random component.
- Will look at methods of having client maintain session id state.

# Maintaining Session Ids Via URLs

- Every request contains session id as part of URL; either as part of URL path component or as a query parameter.
- Every response from the server contains session id in within any links in returned content.
- Can be problematic when linking to shared content.
- Can prevent shared caching.
- RESTful in that session id being part of the URL means that state is a resource.
- Example: Id of cart as part of URL.

# Cookies

- Cookies are merely some limited size content.
- Cookies are sent from server to client using `Set-Cookie` header.
- Cookies are sent from client to server using the `Cookie` header.
- Cookies can be used for session management, user info, shopping carts, personalization, tracking.
- All cookies currently accessible can be read using `document.cookie`; returns semicolon-separated *name=value* pairs. Can parse easily using regex but usually better to use a library.
- Can create/update a single cookie by assigning to `document.cookie` a cookie *name=value* followed by optional cookie attributes following semicolon.

# Set-Cookie Header

`Set-Cookie` response header contains following semi-colon separated fields:

*name=value* This is required. The *name* is the name of the cookie and *value* is its value. Example: `cart_id=a23b-c3f4-a83e`.

`Expires=`*DateTime* or `Max-Age=`*relTime* The time at which the cookie expires. If neither specified, or time is in the past, then cookie is a **session cookie** and expires when browser is closed. `Expires` is supported across all browsers, `Max-Age` on modern browsers.

`Secure` If set, then the cookie is sent only on https connections. Some modern browsers require that only https responses set this cookie.

`HttpOnly` If set, then the cookies cannot be accessed via JavaScript API's within the browser. Prevents *cross-site scripting* (XSS) attacks.

# Set-Cookie Header Continued

`Domain=`*domain* The domain to which the cookie should be sent. For example, if *domain* is specified as `binghamton.edu`, then cookie will be sent to both `binghamton.edu` and `cs.binghamton.edu`.

`Path=`*path* The path and subpaths for which the cookie should be sent. For example, if *path* is `/cgi-bin`, then cookie will be sent for `/cgi-bin` as well as `/cgi-bin/carts`.

# Cookie Header

`Cookie:` *name*=*value*; *name*=*value* ..

- Sent by client to server.
- Contains previously received *name* / *value* pairs for cookies which apply to request URL (domain and path).