Session Management

Dr. Abdelkarim Erradi

CSE @ QU

Outline

1. Session Management
2. Cookies
3. HTML5 Local Storage

Session Management

Session is a mechanism used by Web Apps to **maintain state about a series of requests from the same user** (that is, requests originating from the same browser) within a period of time

Back

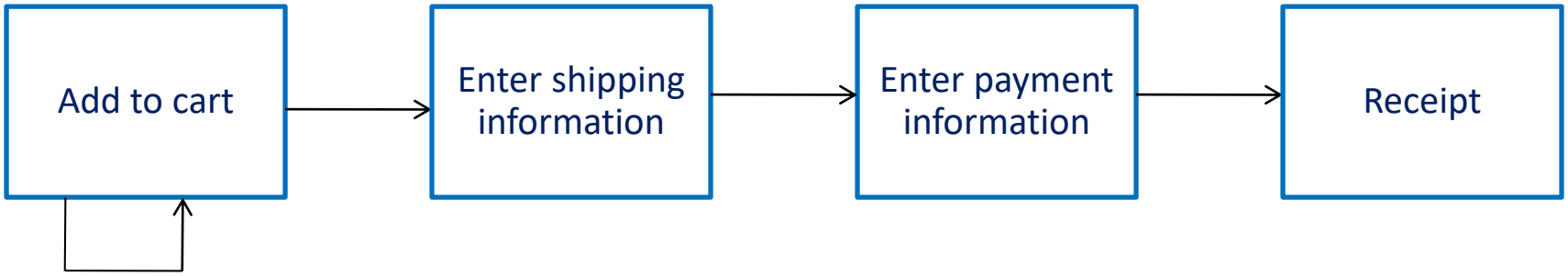
Need for Session Management

- HTTP is a "**stateless**" protocol
  - **Does not support conversations**
  - Has no easy way to distinguish between clients
  - This is good for **scalability** … but keeping state is needed for some scenarios

Issue

- Session: maintain **state** between set of interactions with a user to accomplish goal
  - e.g., shopping cart in online store
  - Server may have to simultaneously manage thousands of sessions

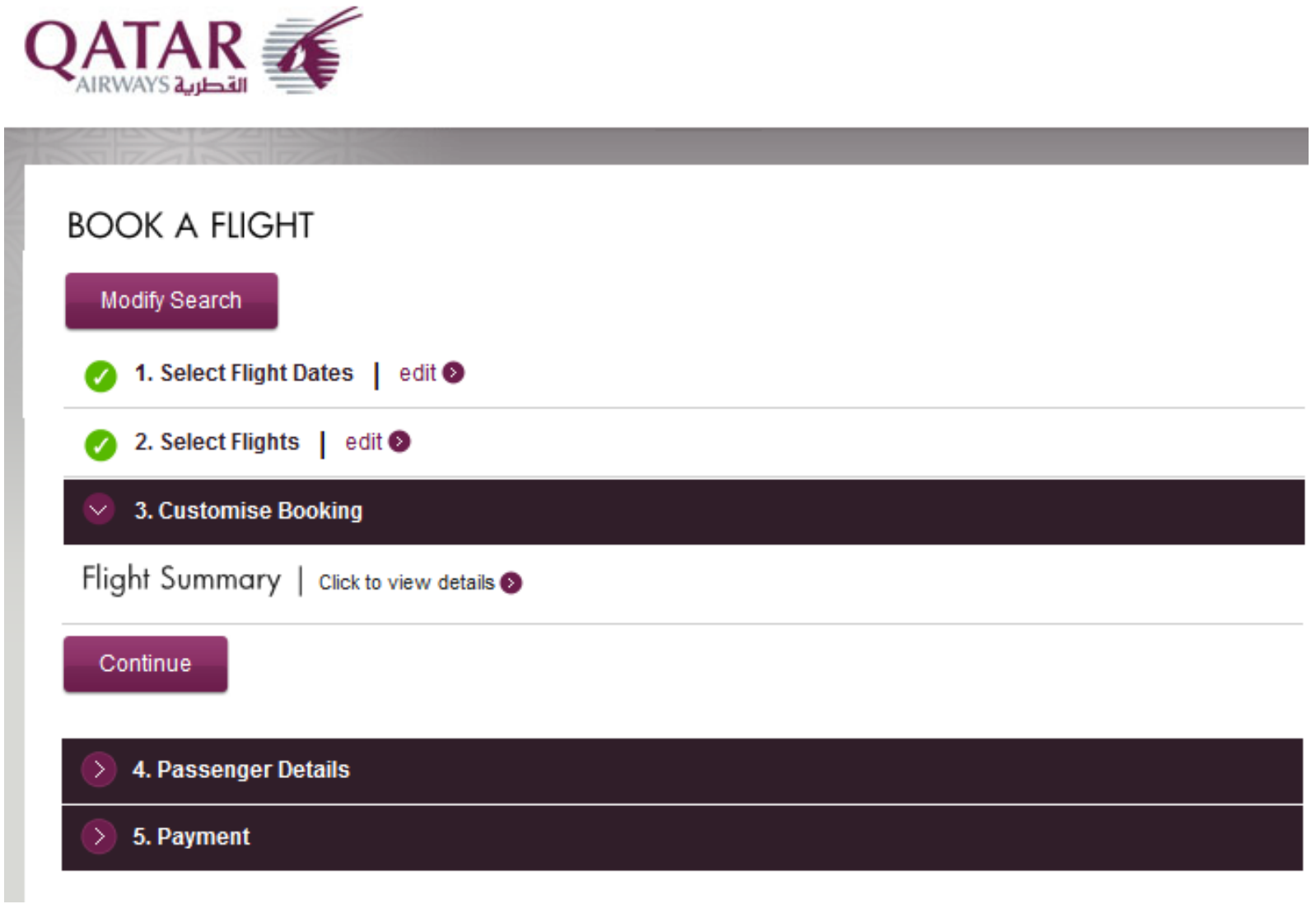
Solution



An example where maintaining State is needed

- Checkout Process

Stateful design use cases - Wizards & conversation-oriented web apps are good examples



Session Management Basics

- Session stores data objects that can be associated with a user
  - The objects exist only on the server memory

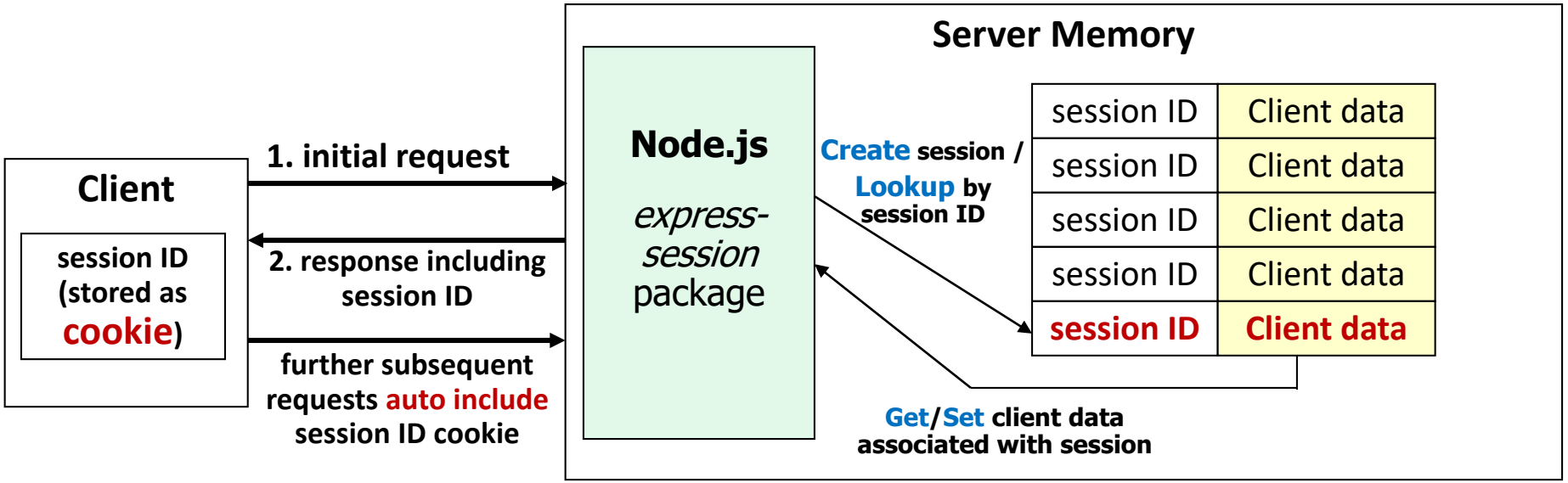
- Access the session object
  - Call **`req.session`** to get the Session object then read and write session data

- To discard session data use destroy method

```
req.session.destroy();
```

Session Management

- Server **creates** a session object for new client at start of the session (i.e., the first time a **req.session** is used in the app)
  - Each session has a **unique Session ID**
  - can store data associated with session ID
  - can look up data associated with session ID
- Server **Passes** the **session ID** as a cookie to client as part of the response
- Client **Stores** the Session ID as a **cookie**
- Client **Passes** the Session ID back to server with subsequent requests



Cookies

"Set-Cookie" is just another header sent in the response.

Here's your cookie with the session ID inside...

HTTP Response

OK, here's the cookie with my request

"Cookie" is another header sent in the request.

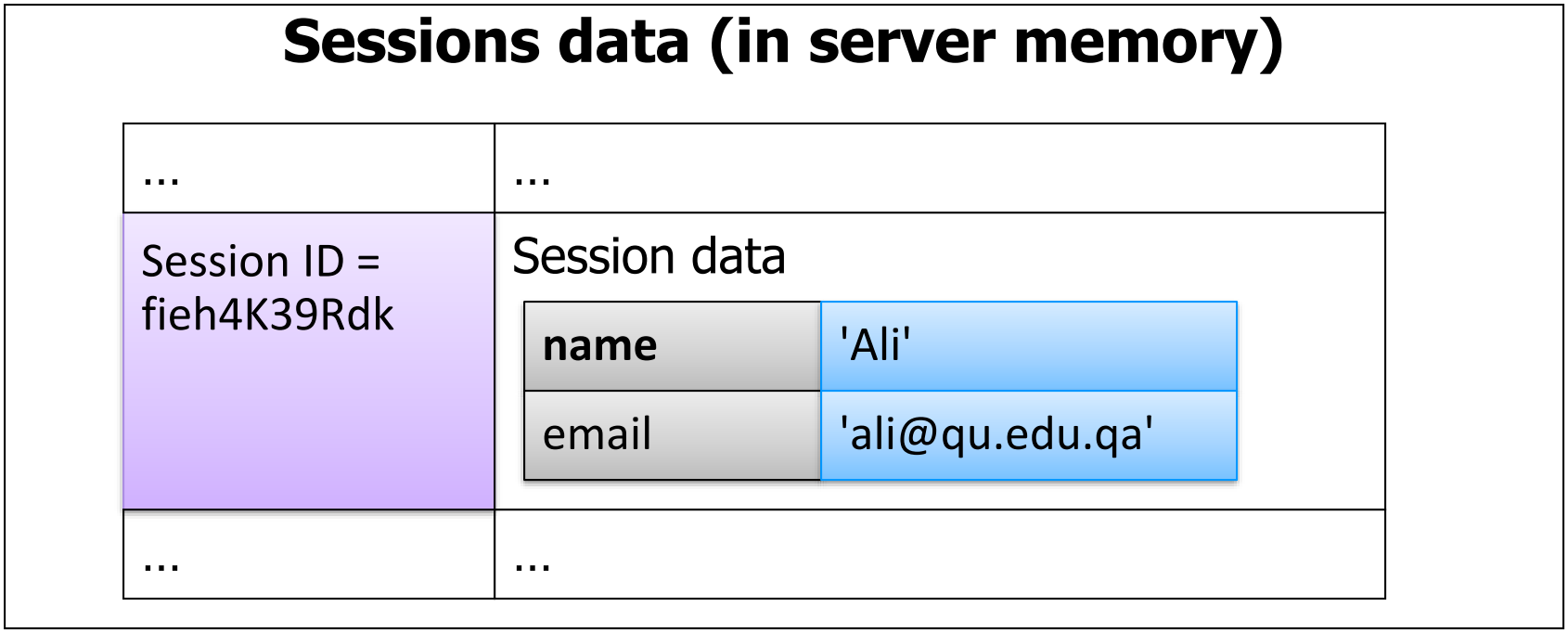
HTTP Request

Get/Set Session Data

- **Syntax:**

```
req.session.name = value

let myVar = session.name
```

- Session data stored as name/value pairs



Session Idle Timeout

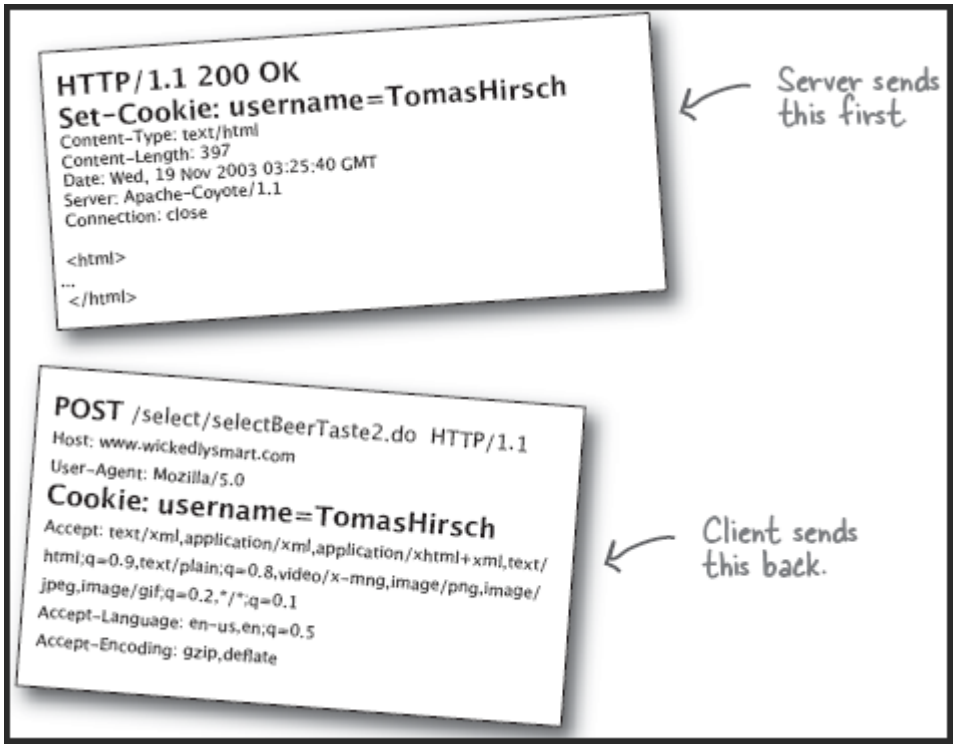
- Can session idle timeout using maxAge parameter
- Session expires if no request received within the specified **maxAge** time limit
  - Session id and all session data get destroyed upon expiry

```
...

//Session expires if no request received within the specified maxAge time limit

let idleTimeoutMilliseconds = 20 * 60 * 1000 //20 minutes
app.use( session ({ cookie: { maxAge: idleTimeoutMilliseconds } }) )

...
```

Cookies

Watch This Video!

https://www.youtube.com/watch?v=I01XMRo2ESg



Cookies

- Idea
  - Server sends a simple name and value pair to client.
  - Client returns same name and value when it connects to same site (or same domain, depending on cookie settings).
- Value limited to 4KB
- Has expiration date, and a server name (returned to same host and not to others)
- Cookie is sent in HTTP header of the response

```
Set-Cookie: name=value
```

```
res.cookie('varName', 'varValue');  // to send a cookie
```

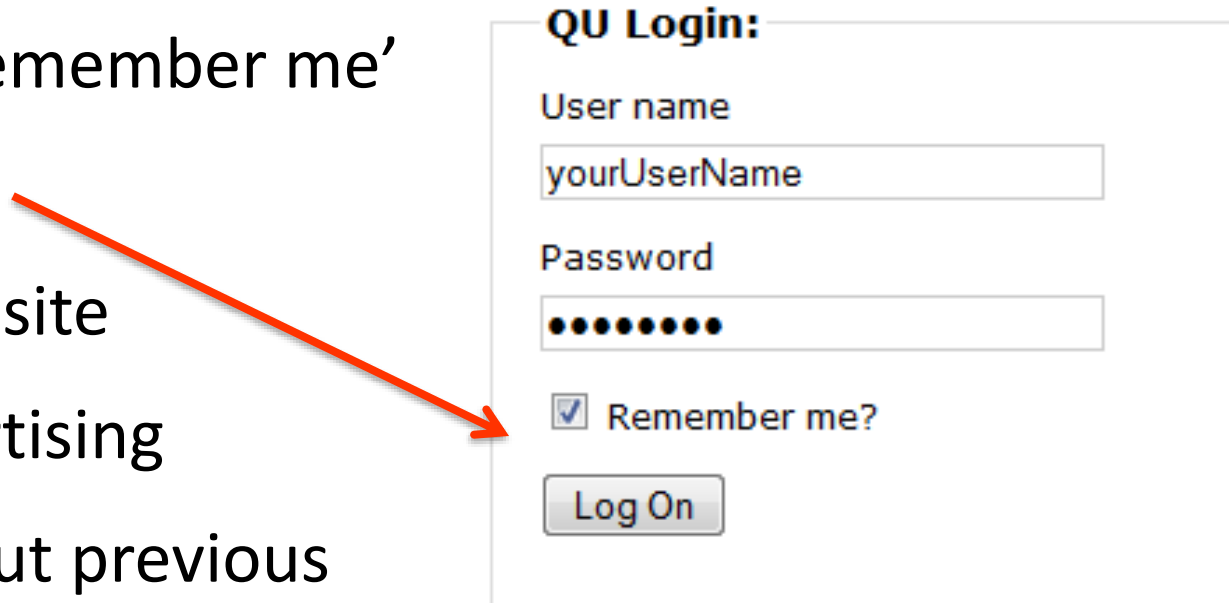
- Cookie is returned to server in HTTP header of subsequent request

```
Cookie: name=value;
```

```
cookies = req.cookies;   //to get cookies
```

Usage of Cookies

- Typical Uses of Cookies
  - Identifying a user during a session
  - Implement 'Remember me' during Login
  - Customizing a site
  - Focused advertising
  - Store info about previous visit and perhaps what the visitor did



Cookies and Focused Advertising

Amazon.com home page for new visitor or visitor with cookies disabled.

Amazon.com home page for repeat visitor. Books shown are based on prior history.

Some Problems with Cookies

- The problem is privacy and security risks
  - Servers can remember your previous actions
  - Servers can share cookie information through use of a cooperating third party like *doubleclick.net*
  - Hacker can steel your cookies and hijack your session or get access to sites under your name, and essentially be logged in as the user associated with it!
    - It is frightening thing if a malicious individual finds out the value of your cookie!

=> Don’t put sensitive info in cookies

Summary - Cookies

- Basic functionality
  - Cookies are name/value pairs sent from server to browser and *automatically* attached to subsequent requests (to the same site or domain)
- Cookies let you
  - Establish sessions
  - Permit users to avoid logging in (when rememberMe is ticked)
  - Customize sites for different users
  - Focus content or advertising
- Setting cookies
  - `res.cookie('varName', 'varValue'); // to send a cookie`
- Reading cookies
  - `cookies = req.cookies;  //to get cookies`

HTML5 Local Storage

HTML5 Local Storage

- Cookies are no longer the only way to store data on the client machine.
- HTML5 introduces local storage to store set of name value pairs directly accessible with **client-side** JavaScript
- Data placed in **local storage** is per origin (the combination of protocol, hostname, and port number) and persists after the browser is closed
  - the data is available to all scripts loaded from pages from the same origin that previously stored the data
- **Session storage** is per-origin-per-tab and data are available until the user closes the tab/browser

Simple API

- **Store**

```
localStorage.lastname = "Smith";
```

- **Retrieve**

```
Console.log(localStorage.lastname)
```

- **Remove**

```
localStorage.removeItem("lastname");
```

- **Remove all saved data**

```
localStorage.clear();
```

Cookies vs. Local Storage

- Cookies are auto-included with every subsequent HTTP request
- Cookies are limited to about 4 KB

- Data in local/session storage are **NOT** auto-included with subsequent HTTP requests
- Storage limited to about 5 MB

- Both cookies and browser storage can be cleared by the user and should not be completely relied upon for client-side storage