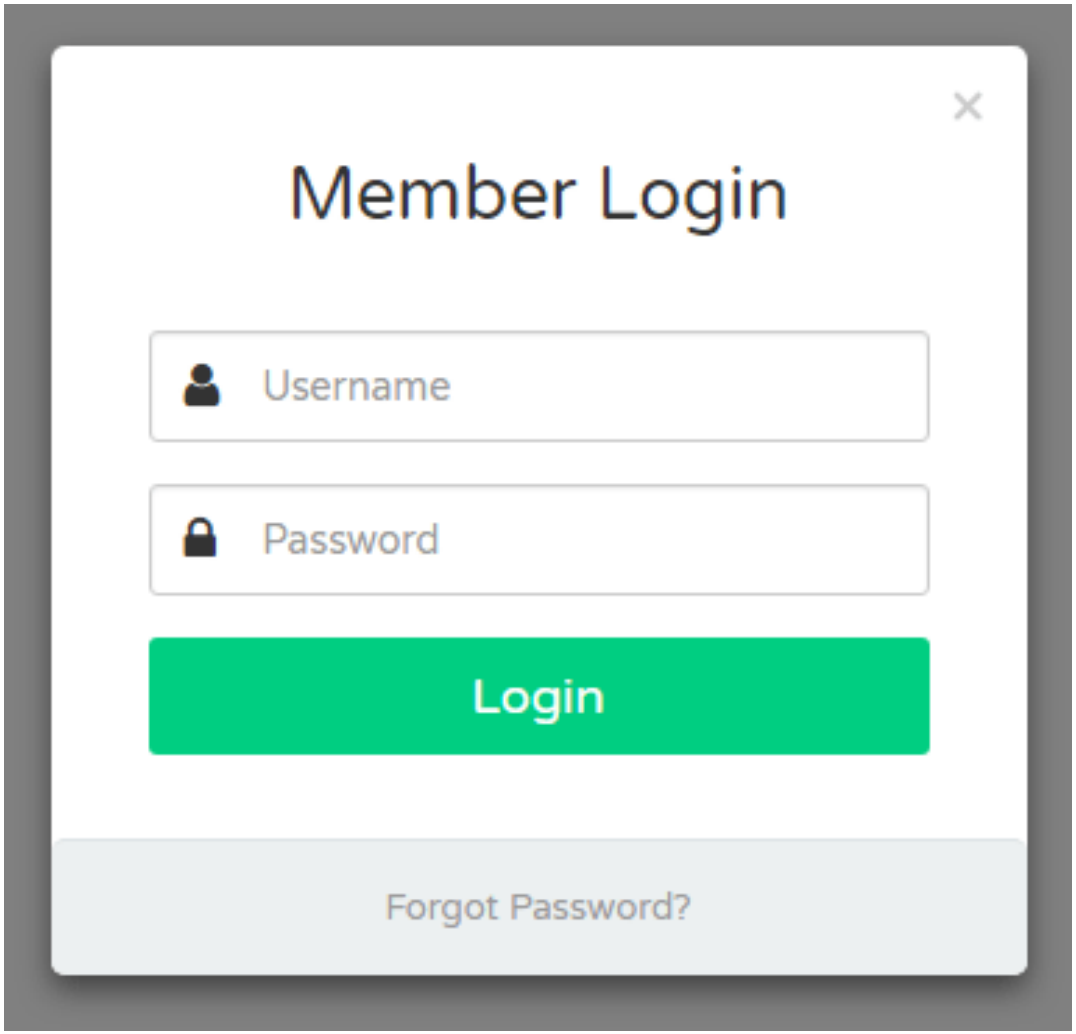
ASP.NET CORE

SESSIONS

issntt@nus.edu.sg

Task of the day

How to implement a Login system?



Objectives

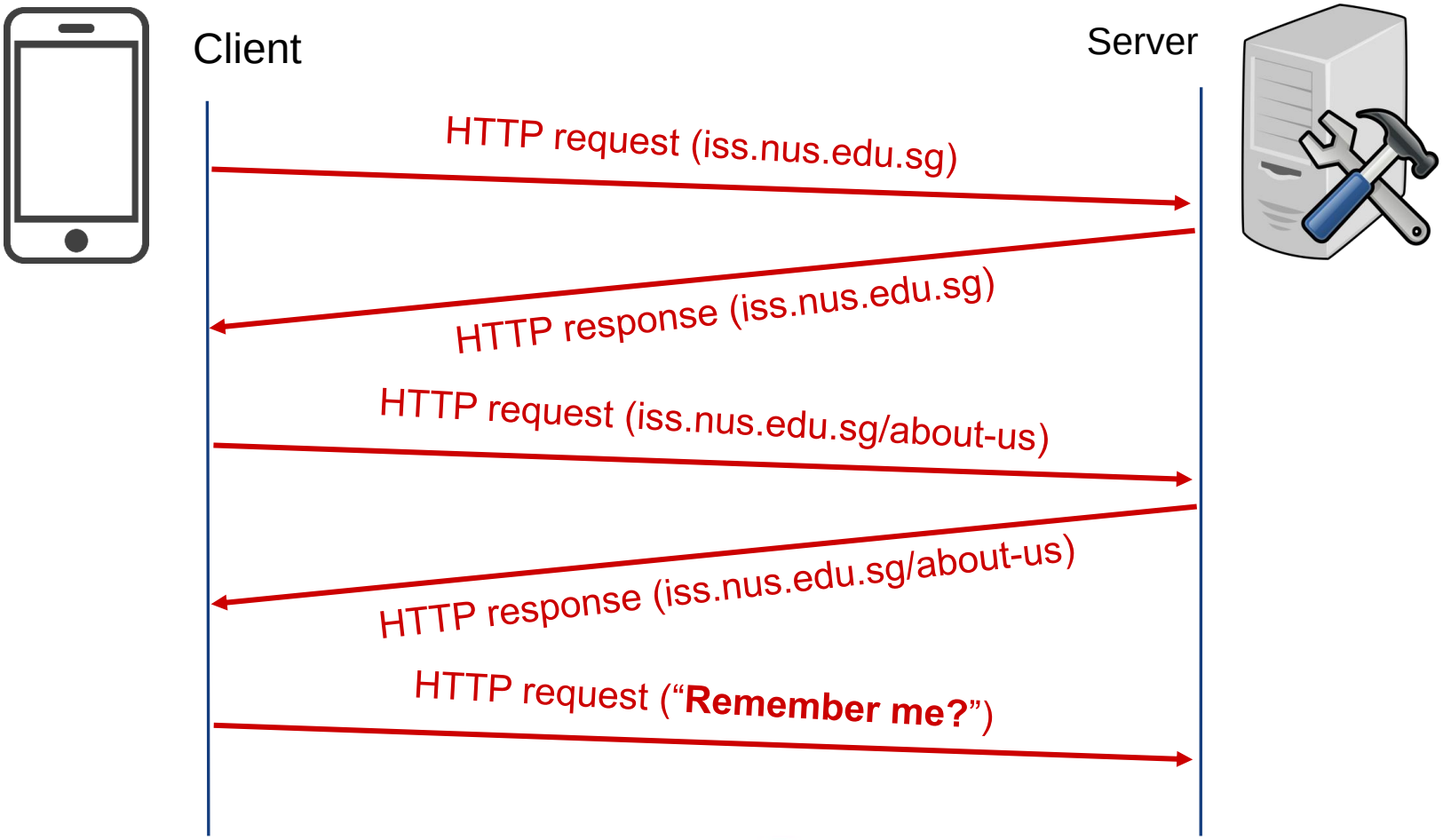
At the end of this lesson, students will be able to

- Explain why sessions are needed in the WWW
- Describe sessions and what are stored in sessions
- Describe when sessions start and end in different scenarios
- Describe why GUID is usually used to implement session IDs
- Describe common methods used to transfer session IDs between servers and clients
- Describe cookies and how sessions are implemented using cookies
- Describe hidden field inputs and how sessions are implemented using hidden field inputs
- Implement stateful web applications using ASP.NET Core Session State

Topics

- **Why are Sessions necessary?**
  - **HTTP is stateless**
- What are Sessions?
- How to implement Sessions?

Using HTTP



With the knowledge about HTTP **so far**, what should be the answer?

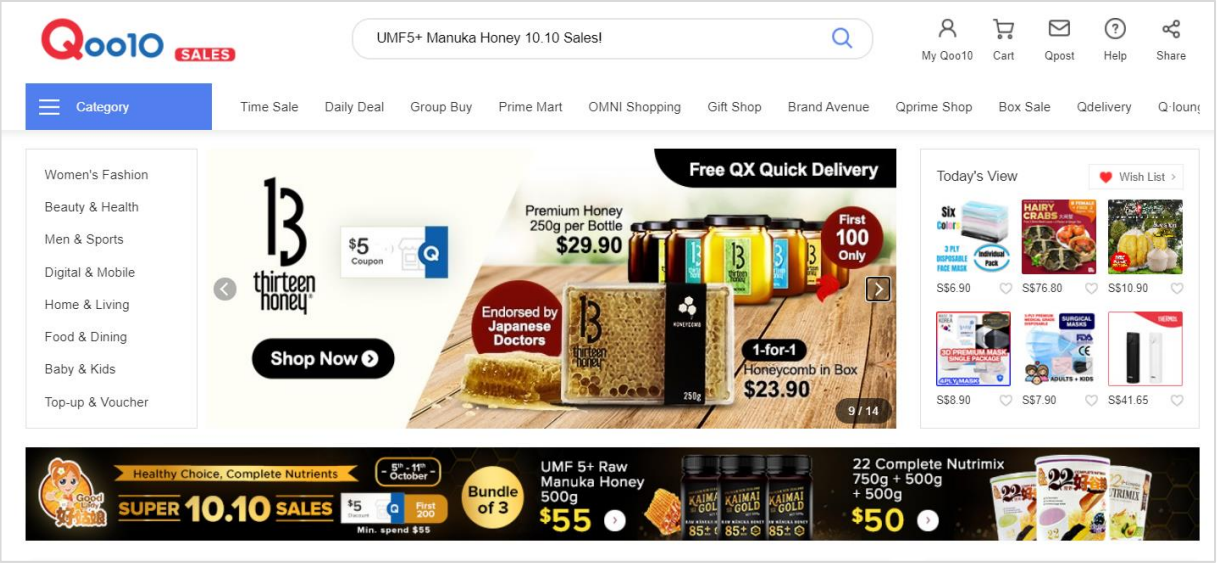
HTTP is Stateless

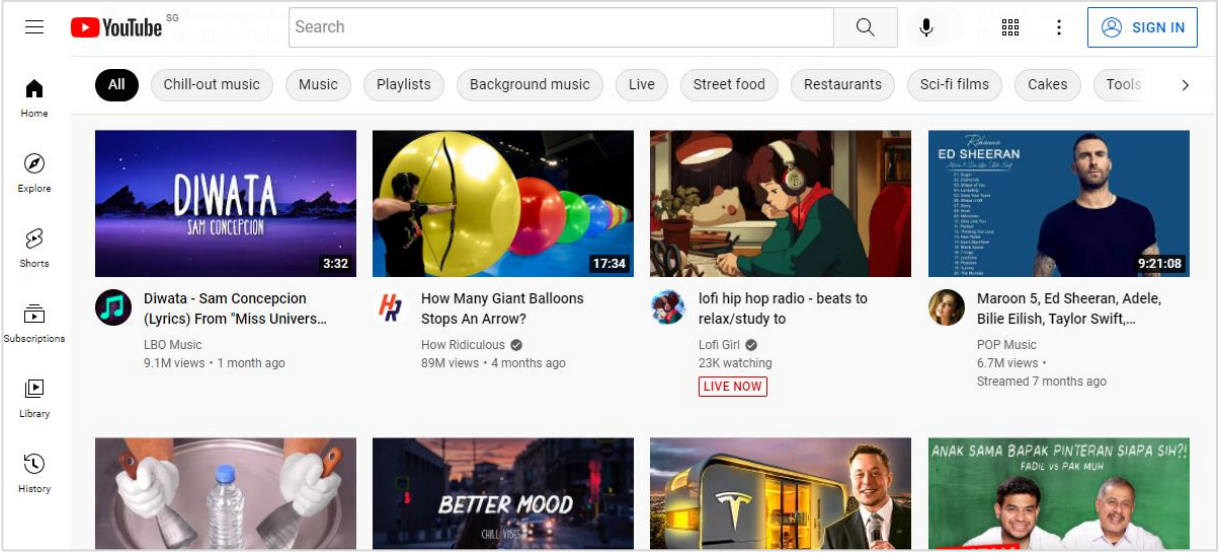
Each HTTP request is **independent** of earlier requests, so servers are **aware** of a **client** during a **current request only**

Image by Herr Dörr from Pixabay

Is it **really so?**

When you browse, do you feel that these webpages are **remembering** you?





Sessions are to **identify** users

To **identify** and interact with a **particular user** across those pages, **tag** him/her with an **unique session**

Image by bvick390 from Pixabay

Topics

- Why are Sessions necessary?
- **What are Sessions?**
  - **Session IDs and Session Data**
  - **Session Duration**
- How to implement Sessions?

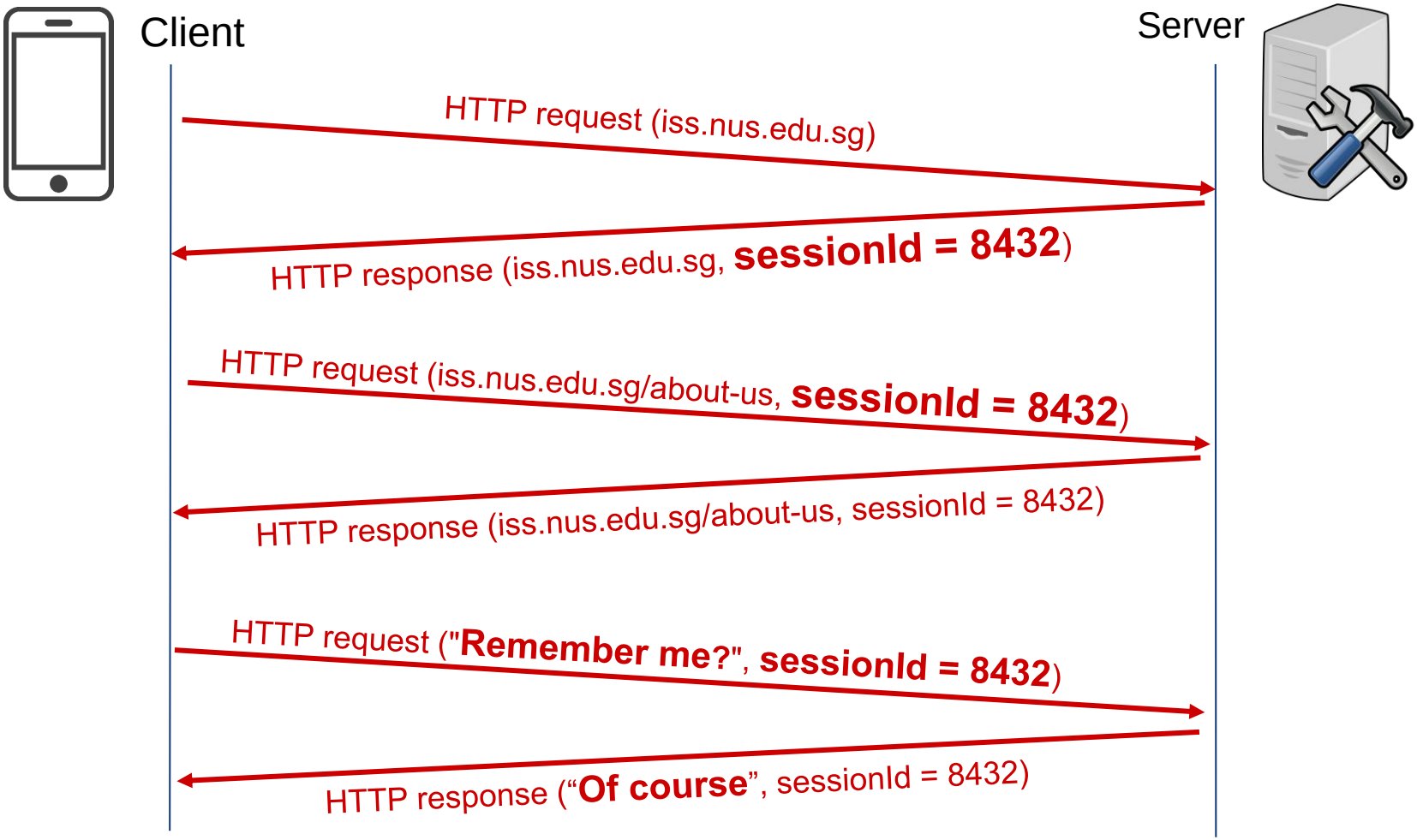
What are Sessions?

A session identifies requests from the **same** client/user, and stores **stateful information** about its/their **past actions**

Image by PublicDomainPictures from Pixabay

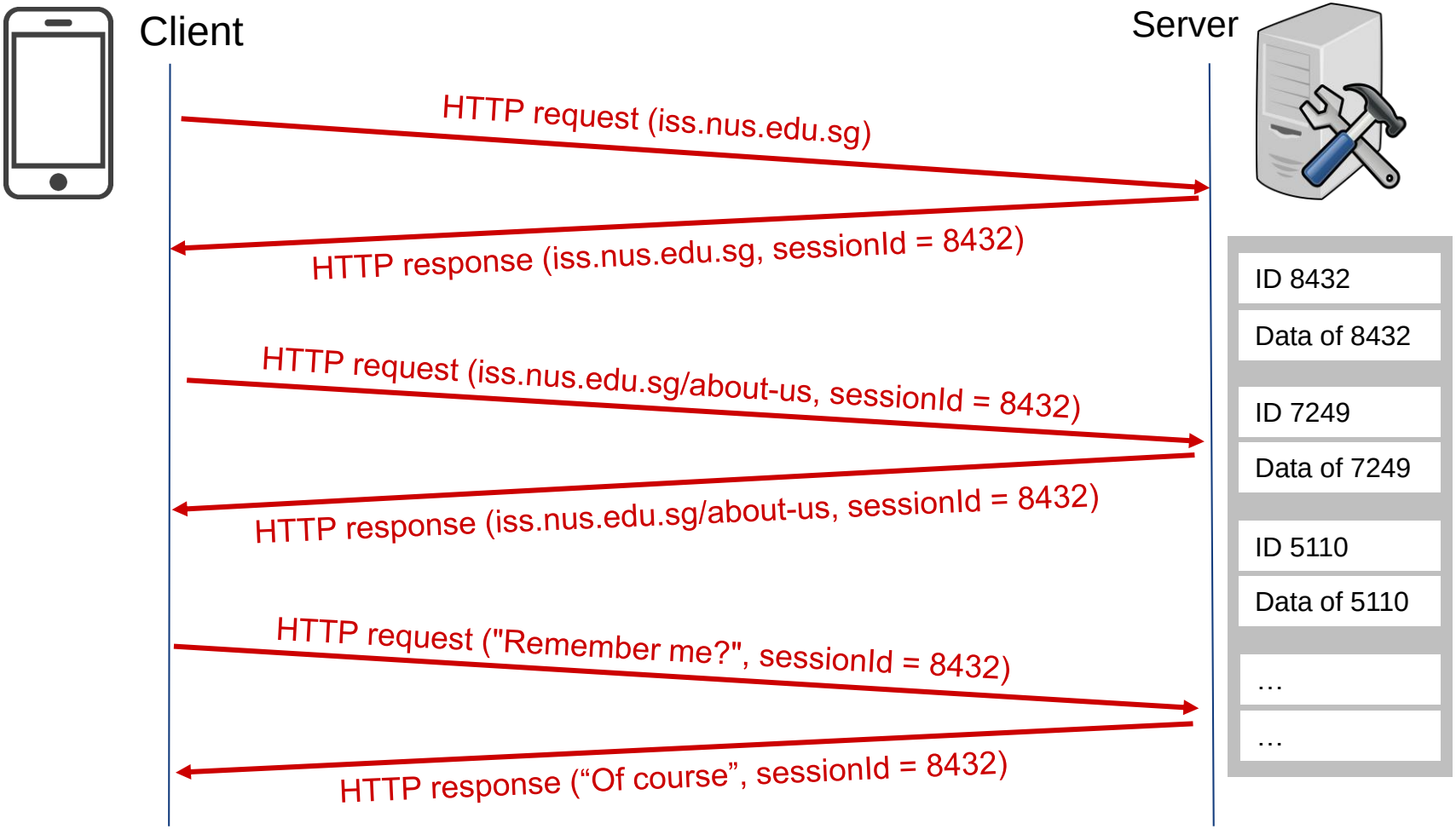
What are Sessions?

To identify a client/user, server **generates** a **session ID**, which will be **embedded** into the HTTP **Requests** and **Responses**



What are Sessions?

Server stores the **stateful information**, called **session data**, and **associated** them with the **session ID**



Summary

A session

- identifies requests from the **same** client or user, and
- stores **stateful information** about **past actions** of the client or user

A session usually includes

- **Session ID**: **unique**, generated by **server**, then transferred among server and client
- **Session data**: stateful information about past actions**.** For example:
  - Whether a user has already logged in
  - The list of items on his/her shopping cart

Session Duration

When does a session **start**?

- **As soon as** the user **hits the landing page**, OR
- **Only after** the user has **logged in**

How long does a session **last**?

- **Long-lived**, or
- **Short-lived**, where the user has to login again after a period of **inactivity**

Image by annca from Pixabay

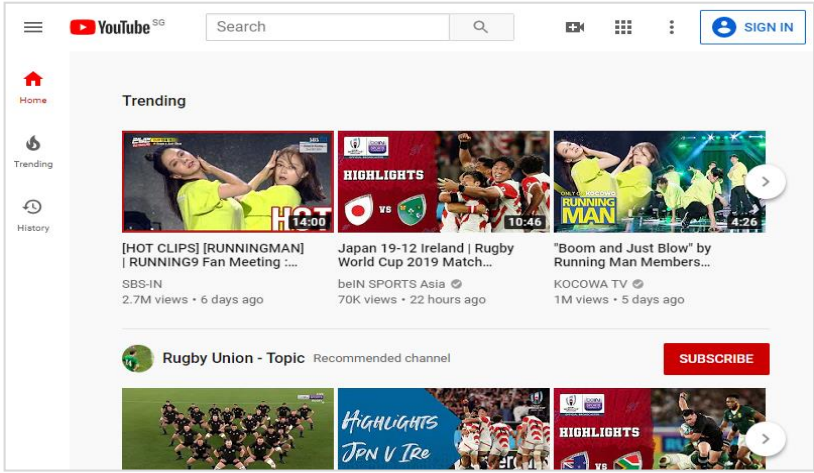
Session duration **depends on situations**

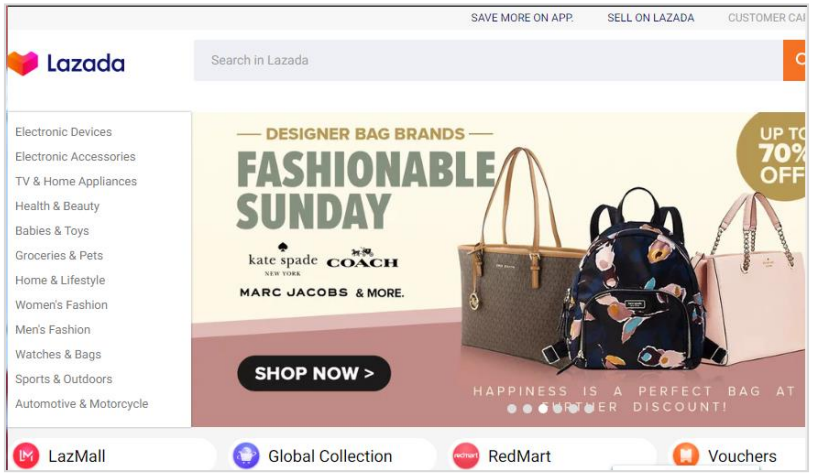
Questions

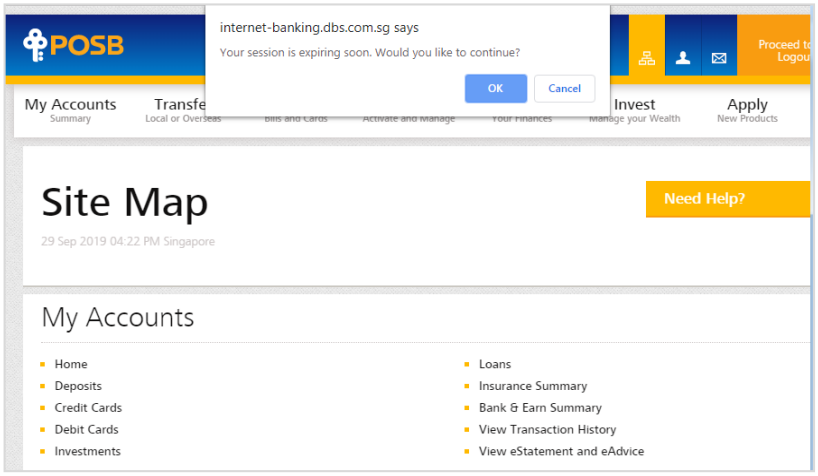
For each of following web apps, **how sessions** may be **likely implemented**:

1. As soon as the user hits landing pages, OR only after the user has logged in?
2. Long-lived or short-lived?

Questions







Topics

- Why are Sessions necessary?
- What are Sessions?
- **How to implement Sessions?**
  - **Using ASP.NET Core Session State**
  - Manually

ASP.NET Core Session State

Session State is a **library** that helps developers by **automatically**

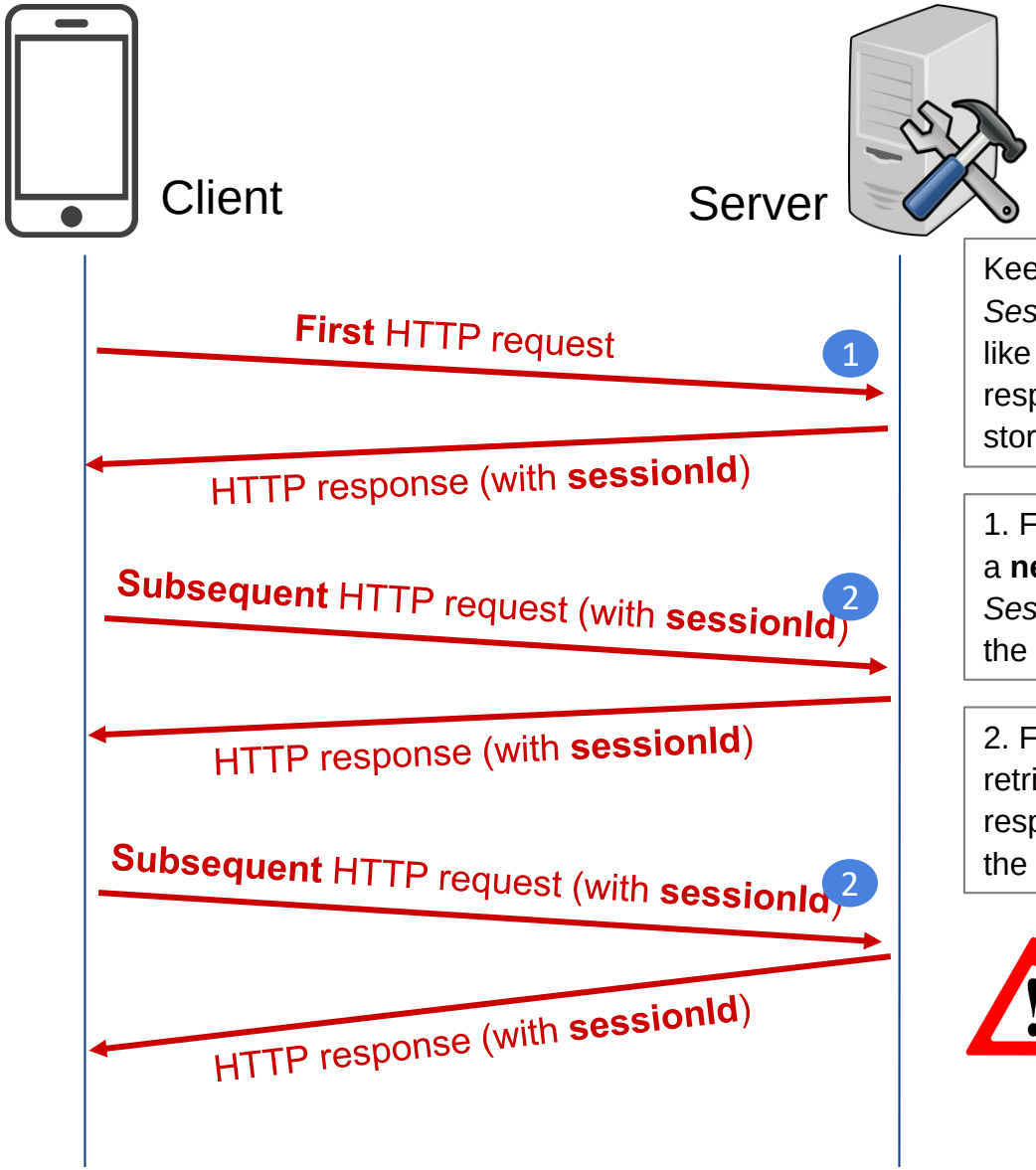
1. **Generating** Session IDs
2. **Transferring** Session IDs between Server and Client
3. **Storing** Session ID and Session Data

Image by OpenClipart-Vectors from Pixabay

Roughly, how does Session State do that?

Generating, Storing IDs and Data

Explore by yourself



Keep a storage that maps *Session IDs* to *Session objects* (we can imagine something like a Dictionary). Given a *Session ID*, the respective *Session object* can be then used to store and retrieve data for that session

1. For the **first** HTTP request, server creates a **new** *Session object* and generates a **new** *Session ID*. Then server **maps** them inside the storage

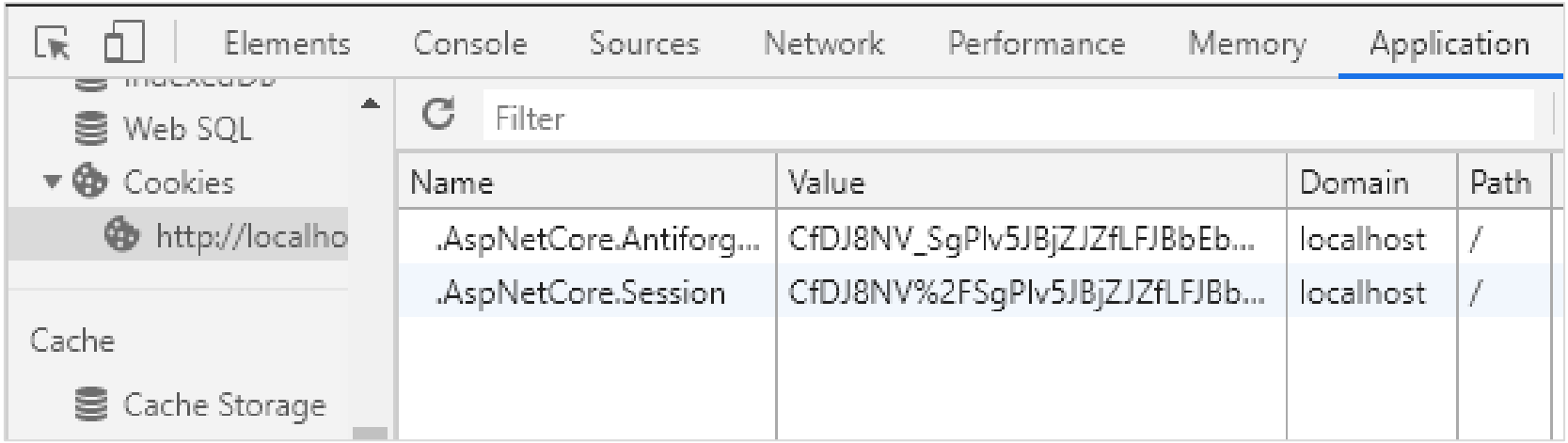
2. For each **subsequent request**, given the retrieved *Session ID*, server will get the respective *Session object* and retrieve/store the respective session data

Session State does not persist data, it **loses all** its **data** when the web app restarts

Transferring Session IDs

Explore by yourself

By default, **Session IDs** is named .AspNetCore.Session and are sent in **cookies**



Using Session State, do **servers** need to call method **`Response.Cookies.Append()`** to send the `.AspNetCore.Session` cookie?

Question

So, Session State takes care many things **automatically**, **how** can we the developers **implement sessions** with it?

Image by mohamed Hassan from Pixabay

1. Enabling Session State

To implement a session, **enable** Session State by **adding** Session Middleware **into** our **pipeline**

```
// Add services to the container.
builder.Services.AddControllersWithViews();
builder.Services.AddSession();
…
```

Program.cs

```
…
app.UseAuthorization();

app.UseSession();

app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");
…
```

Program.cs

2. Working with Session Data

After Session State is **enabled**, this library **takes care of** the session and **gives developers** Session Object

```
public IActionResult Login(string username) {
  ISession sessionObj = HttpContext.Session;

  sessionObj.SetString("username", username);
  sessionObj.SetString("run", "");
  sessionObj.SetInt32("number", 1);

  return RedirectToAction("Track", "Home");
}
```

```
public ActionResult Track() {
  ISession sessionObj = HttpContext.Session;

  string? usernameInSession =
         sessionObj.GetString("username");
  if (usernameInSession == null)
      return RedirectToAction("Login", "Home");

  return View();
}
```

2. Working with Session Data

Developers can **use** the given **Session Object**, a **simple dictionary-like ADT**, **to store** any Session **Data**

```
public IActionResult Login(string username) {
  ISession sessionObj = HttpContext.Session;

  sessionObj.SetString("username", username);
  sessionObj.SetString("run", "");
  sessionObj.SetInt32("number", 1);

  return RedirectToAction("Track", "Home");
}
```

For all methods, read https://learn.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.http.isession?view=aspnetcore-6.0

2. Working with Session Data

Developers can later **retrieve** and **clear data**

```
public ActionResult Track() {
  ISession sessionObj = HttpContext.Session;

  string? usernameInSession =
        sessionObj.GetString("username");
  if (usernameInSession == null)
      return RedirectToAction("Login", "Home");

  return View();
}
```

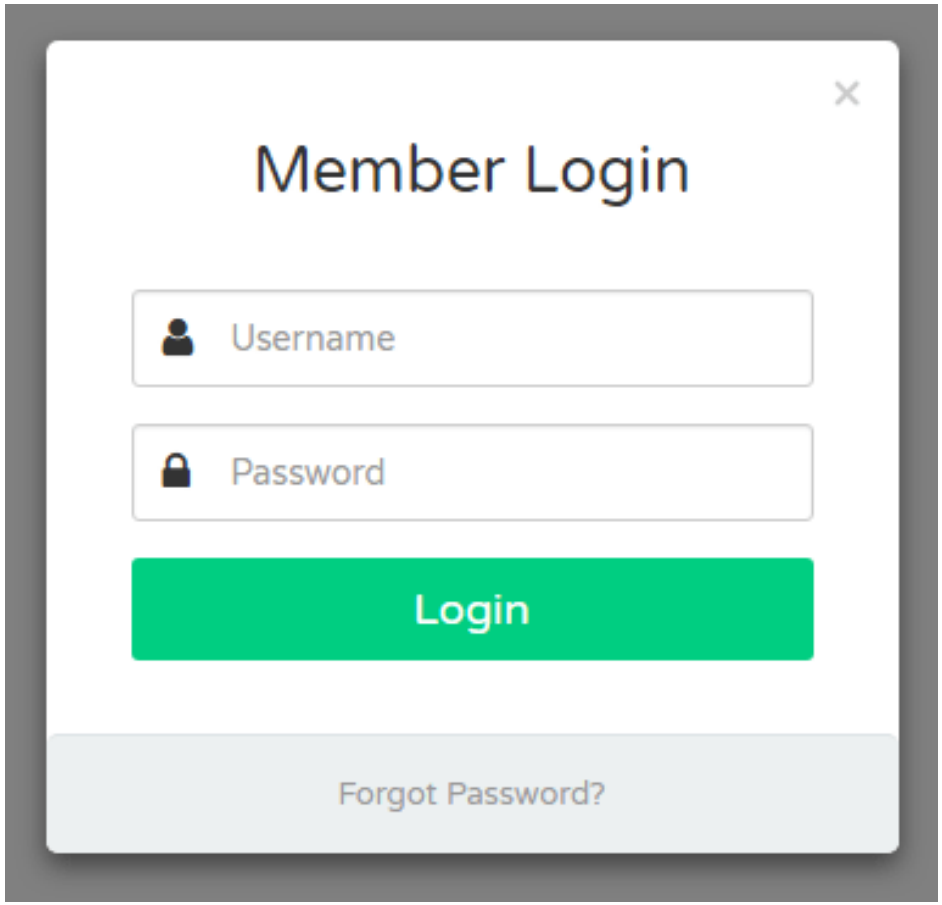
```
public IActionResult Logout() {
   ISession sessionObj = HttpContext.Session;

   sessionObj.Clear();
   // or sessionObj.Remove("username");
   return RedirectToAction("Login", "Home");
}
```

Clear() = Remove() everything

For all methods, read https://learn.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.http.isession?view=aspnetcore-6.0

Practice

Using Session State, implement the Login feature?



An Interview Question

Image by OpenClipart-Vectors from Pixabay

If we **don't use** any **library** like ASP.NET Core Session State, how can we **implement Sessions manually**?

Topics

- Why are Sessions necessary?
- What are Sessions?
- **How to implement Sessions?**
  - Using ASP.NET Core Session State
  - **Manually**

The 3 Problems

Generating Session IDs

- What can be used as IDs?
- How are they used in servers and clients?

Transferring Session ID

- How are Session IDs transferred between servers and clients?

Storing Session IDs and Data

- How do servers store Session IDs and Session Data?

Image by Gordon Johnson from Pixabay

Topics

- Why are Sessions necessary?
- What are Sessions?
- **How to implement Sessions?**
  - Using ASP.NET Core Session State
  - **Manually**
    - **Generating Session IDs**
    - Transferring Session IDs
    - Storing Session IDs and Data

Generating Session IDs

A developer uses **running numbers** to implement Session IDs

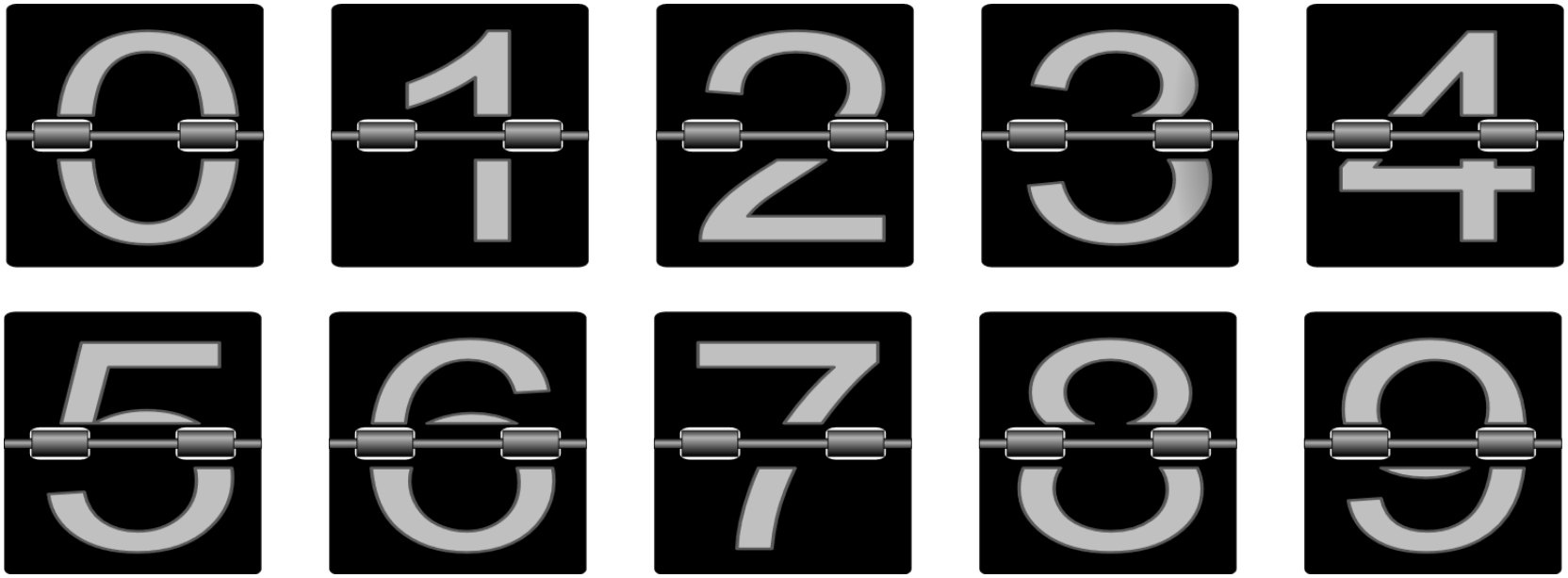


Image by OpenClipart-Vectors from Pixabay

Is it a good option?

UUIDs (aka GUIDs)

An UUID is a 128-bit string and there are $\mathbf{2^{128}}$ **possible** UUIDs

Are UUIDs and the like **good enough** to be used to **implement** Session IDs?

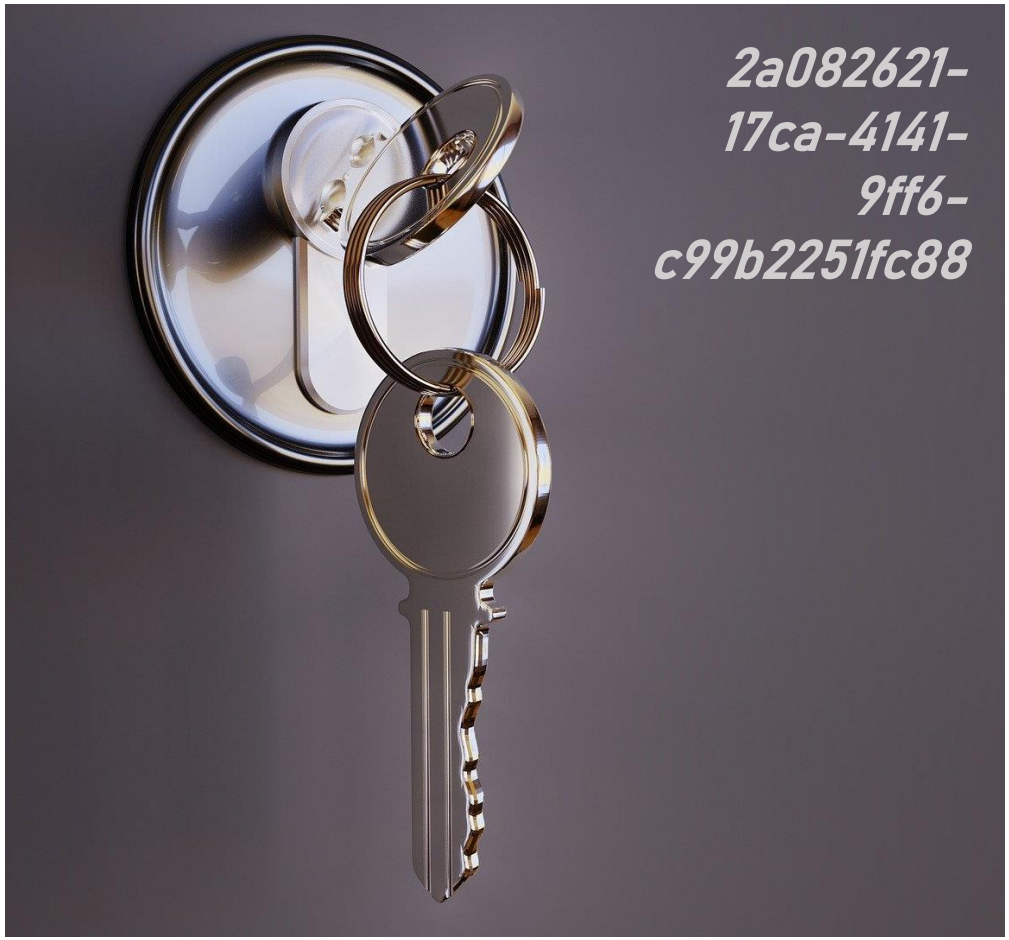


Image by MasterTux from Pixabay

Using UUIDs for Session IDs

Is an UUID **unique** across systems?

- **Practically unique**; not guaranteed unique
- **Collision** probability is **very low**

Can an UUID for a session be **easily guessed**?

- If our session's UUID is *10fc820f-fece-4f5b-bc5e-7c33438ea75c*
- Can we infer some other user' session IDs?

To lower the collision probability and make it more secure, many systems use even longer strings than UUIDs

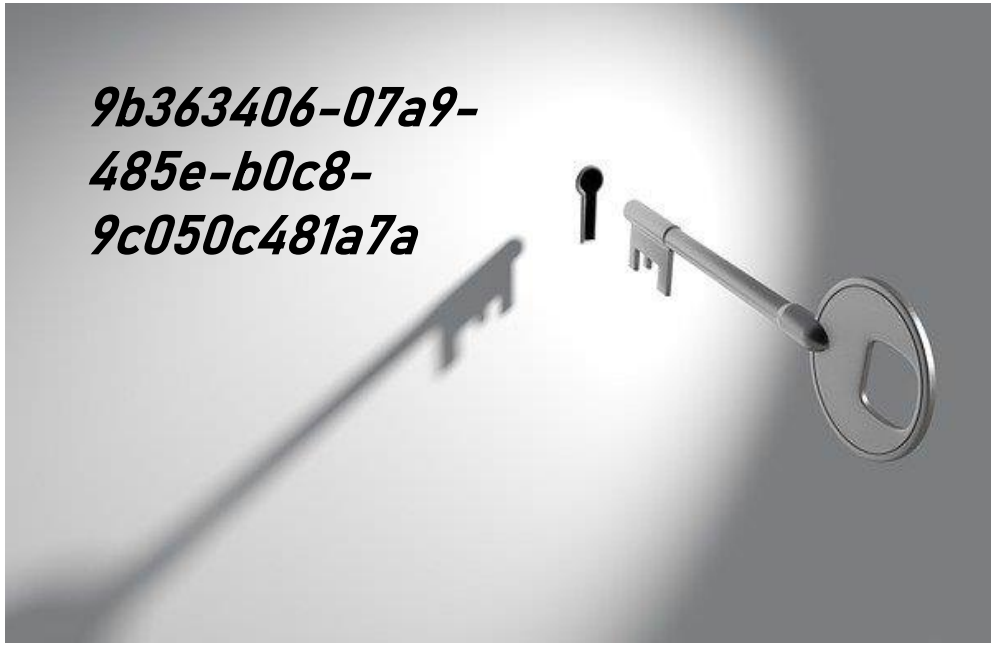


Image by Arek Socha from Pixabay

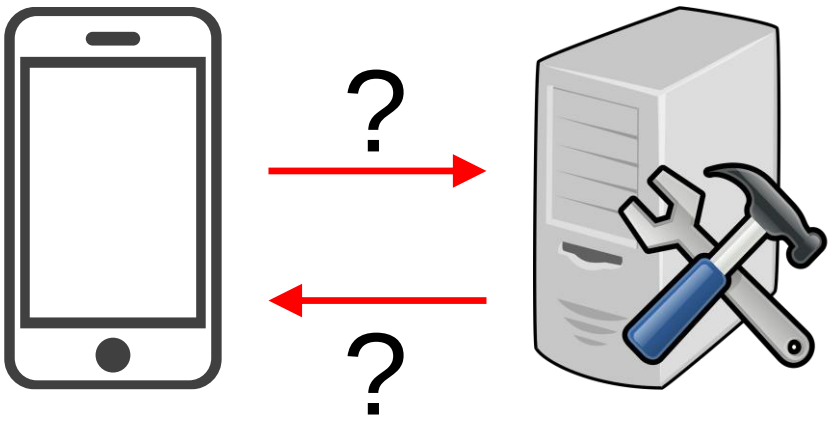
Topics

- Why are Sessions necessary?
- What are Sessions?
- **How to implement Sessions?**
  - Using ASP.NET Core Session State
  - **Manually**
    - Generating Session IDs
    - **Transferring Session IDs**
    - Storing Session IDs and Data

Review Question

A **client** needs to **send** a **session ID** to the **server**. Using HTTP Request, **how many options** can it use to send the data?

How about a **server** to **client**?



Some common methods

Cookies

Server asks the client to store Session ID as a cookie within the web browser

(Request Header and Response Header)

Query Strings

Session ID is embedded in every URL link that it creates in its View

(Request Query String and Response Body)

Hidden Fields

Session ID is embedded as a hidden value within a HTML Form

(Request form body and Response Body)

JSON Message

Session ID is embedded within a JSON message and send it using JavaScript

(Request body and Response Body)

In theory, any combination of Headers, Query Strings, Body… is fine. Here we focus on common methods

Topics

- Why are Sessions necessary?
- What are Sessions?
- **How to implement Sessions?**
  - Using ASP.NET Core Session State
  - **Manually**
    - Generating Session IDs
    - **Transferring Session IDs**
      - **HTTP Cookies**
      - Hidden Form Inputs
    - Storing Session IDs and Data

HTTP Responses (Revisit)

Following is a sample response when Browsers access *http://iss.nus.edu.sg*

status code status phrase

```
HTTP/1.1 200 OK
Content-Type: text/html
Server: Microsoft-IIS/10.0
Content-Length: 210
Set-Cookie: …

<html>
<head>
<META NAME="robots" CONTENT="noindex,nofollow">
...
```

headers

body

https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview#HTTP_Messages

HTTP Cookies

A cookie is a small piece of data that **servers send** to the **web browsers** using *`Set-Cookie`* header

```
HTTP/2.0 200 OK
Content-Type: text/html
Set-Cookie: yummy_cookie=choco
Set-Cookie: tasty_cookie=strawberry

[page content]
```

Each stores a key/value pair

headers

body

HTTP Response

https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies

HTTP Cookies

Then, browsers may **store** each cookie and **send** it **back** with **every subsequent** request to the servers

```
GET /sample_page.html HTTP/2.0
Host: www.example.org
Cookie: yummy_cookie=choco;
  tasty_cookie=strawberry
```

HTTP Request

The browsers send cookies back **automatically**

Unless users disable cookies

HTTP Cookies

The **lifetime** of a cookie can be **specified** with the *`Expires`* attribute, after which the cookie will be **deleted**

```
HTTP/2.0 200 OK
Content-Type: text/html
Set-Cookie: Set-Cookie: id=a3fWa;
  Expires=Wed, 31 Oct 2021 07:28:00
  GMT;

[page content]
```

HTTP Response

HTTP Cookies

The *`Path`* attribute defines the **scope** of the cookie: **what URLs** the cookie should be sent to

```
HTTP/2.0 200 OK
Content-Type: text/html
Set-Cookie: Set-Cookie: id=a3fWa;
  Path=/;

[page content]
```

HTTP Response

Slash / means sending to **all pages** in the domain

Implementing Sessions using Cookies

In servers, use *`Response.Cookies`* to **create** and **send Session IDs** to clients via cookies

```
public IActionResult StartSession() {
    string sessionId = 1 System.Guid.NewGuid().ToString();

 2 CookieOptions options = new CookieOptions();
   options.Expires = DateTime.Now.AddDays(10);
 3 Response.Cookies.Append("SessionId", sessionId, options);

 4 return RedirectToAction("Index");
}
```

```
HTTP/2.0 200 OK
Content-Type: text/html
Set-Cookie:
    SessionId=dbd9a209-de94-
    4b16-aeae-5816fa136f28;
    expires=Fri, 16 Oct 2021
    01:33:01 GMT; path=/

[page content]
```

Generated Response

1. Create a new session ID by generating a new UUID (aka GUID)

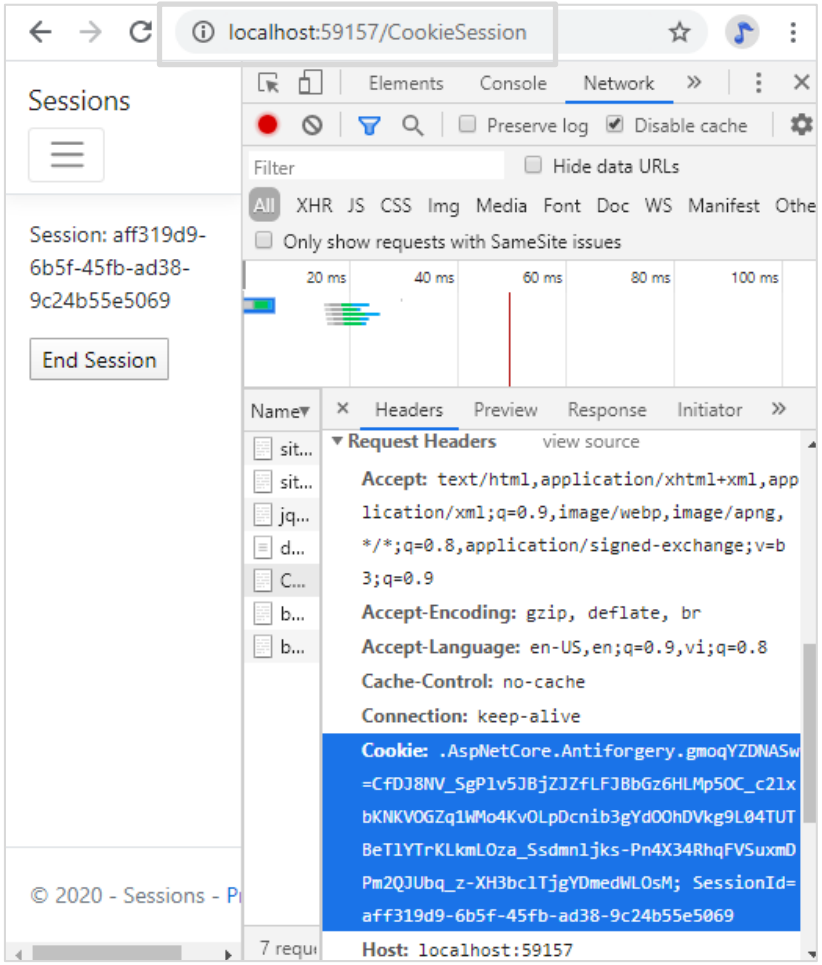
2. Options (scope, expiry…) can be put to cookies. Here, set cookie expiry to the next 10 days and scope to default - root domain

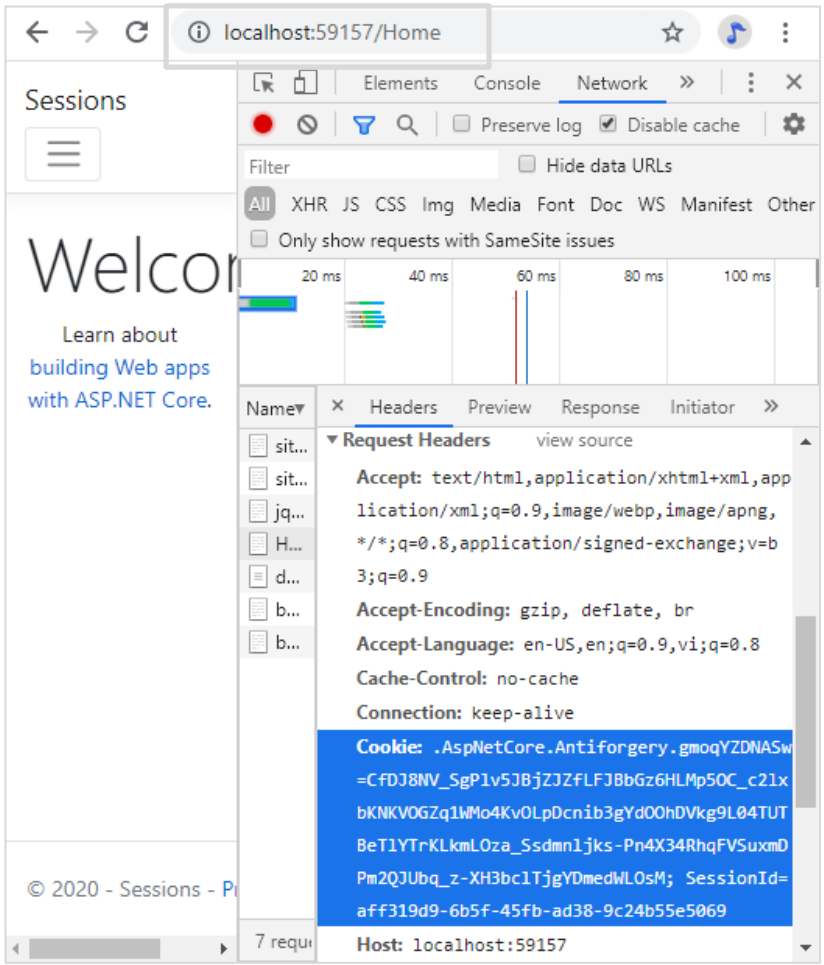
3. Add the new created cookie to the HTTP Response, given it a key, in this case, *SessionId*.

4. Then, other steps are as normal. The cookie will be sent automatically together with the HTTP response.

Implementing Sessions using Cookies

Then, **Browsers send** the cookies **back automatically** in any **subsequent requests** to the Servers





Implementing Sessions using Cookies

The Servers then **retrieves** and **uses** the cookie from the Request objects

```
public IActionResult Index()
{
   string sessionId = Request.Cookies["SessionId"];

   ViewData["session"] = sessionId;
   return View();
}
```

Implementing Sessions using Cookies

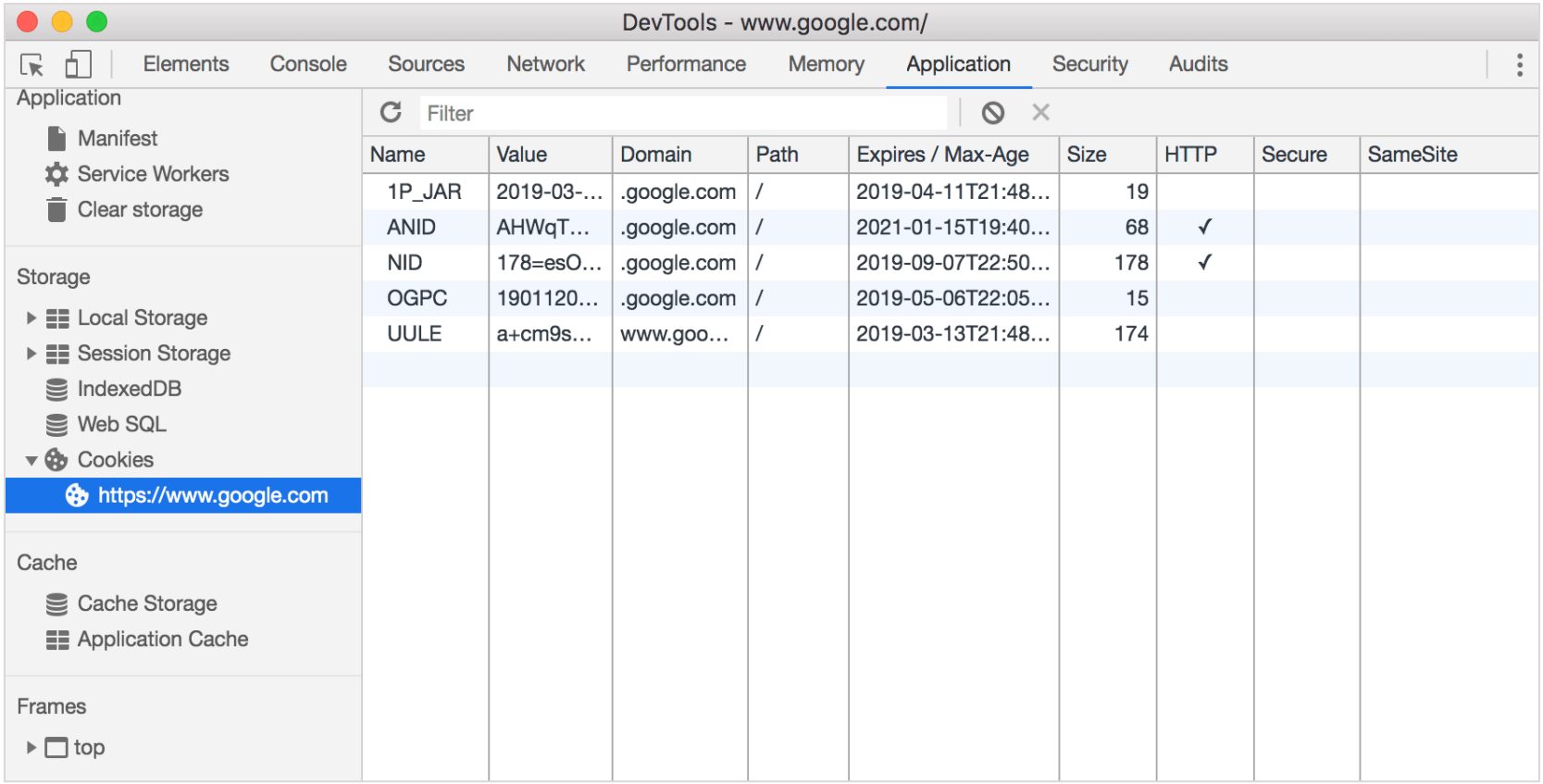
Eventually, servers can also **delete** the cookies, which sets them expired and therefore the **clients** will **stop sending** them back

```
public IActionResult EndSession()
{
   Response.Cookies.Delete("SessionId");
   return RedirectToAction("Index");
}
```

```
HTTP/2.0 302 Found
Location: /CookieSession
Set-Cookie: SessionId=;
   expires=Thu, 01 Jan 1970
   00:00:00 GMT; path=/

[page content]
```

Inspect Cookies in Browsers

Common Browsers, such as Chrome, provides tools to **view**, **modify** and **delete** cookies



https://developers.google.com/web/tools/chrome-devtools/storage/cookies

Topics

- Why are Sessions necessary?
- What are Sessions?
- **How to implement Sessions?**
  - Using ASP.NET Core Session State
  - **Manually**
    - Generating Session IDs
    - **Transferring Session IDs**
      - HTTP Cookies
      - **Hidden Form Inputs (explore-by-yourself)**
    - Storing Session IDs and Data

HTML Input Hidden Fields

Explore by yourself

A hidden field includes **data** that **cannot be seen** or modified **by users** when a form is submitted

```
<form>
   <p>Some text: <input type="text" name="sometext"></p>
   <p>
      <input type="hidden" name="location"
                                   value="hidden-page">
      <input type="submit" value="Send data">
   </p>
</form>
```

Some text:

Send data

https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input/hidden

Implementing Sessions using Hidden Fields

Explore by yourself

In servers, **generate** and **embed** the session ID **as** a HTML input **hidden field**

```
[HttpGet]
public IActionResult Add()
{
1 string sessionId =
      System.Guid.NewGuid().
      ToString();

   ViewData["sessionId"] =
                        sessionId;
   return View();
}
```

```
@{
  ViewData["Title"] = "Example";
2 string sessionId = (string)
      ViewData["sessionId"];
}
<h3>Add course</h3>
<form method="post">
  <div>
    <label for="code">Course Code</label>
    <input type="text" name="code" value="" />
  </div>
  <div>
    <label for="course">
              Course Description</label>
    <input type="text"
              name="description" value=""/>
  </div>
  <input 3 type="hidden"
           name="sessionId"
           value="@sessionId"/>
  <button type="submit">Add</button>
</form>
```

```
4 <input type="hidden"
    name="sessionId"
    value="c23fd76d-42da-
4efc-9be2-7b0c75fe47f8"/>
```

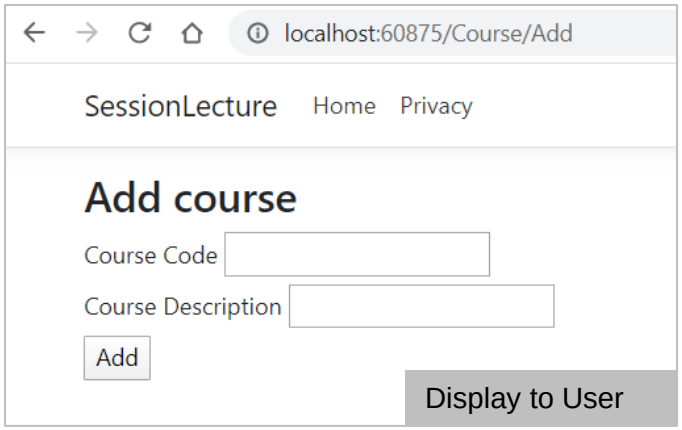
Implementing Sessions using Hidden Fields

Explore by yourself

The generated HTML is sent to clients, in which the **session ID** is **there** but **not shown** to users

```
<h3>Add course</h3>

<form method="post">
   <div>
      <label for="code">Course Code</label>
      <input type="text"
               name="code" value="" />
   </div>
   <div>
      <label for="course">
          Course Description</label>
      <input type="text"
               name="description" value=""/>
   </div>
   <input type="hidden"
        name="sessionId"
        value="c23fd76d-42da-4efc-
9be2-7b0c75fe47f8 "/>
   <button type="submit">Add</button>
</form>
```

Generated HTML



Display to User

Implementing Sessions using Hidden Fields

Explore by yourself

Although **not shown** to users, **session IDs** are **still inside** HTTP Requests to servers during form submission

```
POST /HiddenFieldSession/Add HTTP/1.1
Host: localhost:59157
User-Agent: Firefox/5.0
Content-Type: application/x-www-form-urlencoded
…

code=OOPCS&description=Object+Oriented&sessionId=2
   3fd76d-42da-4efc-9be2-7b0c75fe47f8
```

Implementing Sessions using Hidden Fields

Explore by yourself

Then of course, servers can then **bind data** and proceed

```
[HttpPost]
public IActionResult Add(string code,
                         string description,
                         string sessionId)
{
   // Receiving the sessionId, server needs to verify
   bool res = IsValid(sessionId);

   // Proceed with the code and description if session is valid
}
```

Further Question for Thought

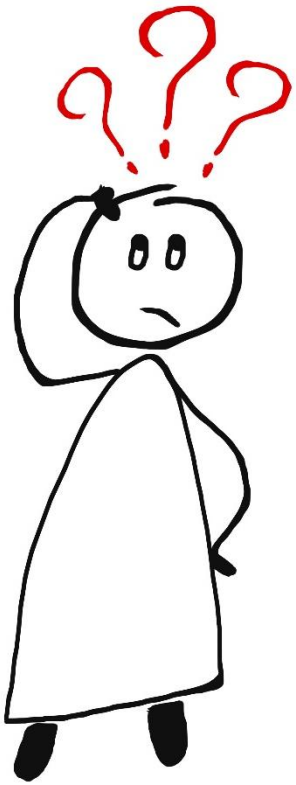


Image by ElisaRiva from Pixabay

Wait! It means **every single** action **method** must **deal with** the ***`sessionId`*** parameter, **no matter how** it is sent, right?

Is there a **better** way to **handle** the ***`sessionId`***?

Hint: cross-cutting concern

Topics

- Why are Sessions necessary?
- What are Sessions?
- **How to implement Sessions?**
  - Using ASP.NET Core Session State
  - **Manually**
    - Session IDs
    - How are Session IDs sent?
    - **How do Servers store Session IDs and Data?**

Storing Session IDs and Data

In general, two options

App Storage

- Store in the app itself
- One way is to use **ASP.NET Core Session State** Session Object

Database

- Store in DB
- A must for **critical data**

The two options can be **combined**, for example:

- Session State stores a **Session ID** and its respective **User ID**
- Database stores **past information** of the user, which **links** to the **User ID**

An Interview Question

- You are the Software Architect of a company. As your team has released new code over the weekend, you decided to see the changes.
- Once you logged in, you were brought to a list of services. You clicked on one of them.
- Instead of bringing you to the selected service, it brought you to the Login page?!
- Puzzled, you logged in again and selected another service. It brought you to the Login page again?!

Photo by Van Tay Media on Unsplash.

What bugs could they be?

Readings

- Session and State Management with ASP.NET Core https://docs.microsoft.com/en-us/aspnet/core/fundamentals/app-state?view=aspnetcore-6.0
- ASP.NET Session State Overview https://docs.microsoft.com/en-us/aspnet/core/fundamentals/app-state?#session-state
- ASP.NET Core Session Overview https://andrewlock.net/an-introduction-to-session-storage-in-asp-net-core/
- Session State in ASP.NET Core https://www.c-sharpcorner.com/article/session-state-in-asp-net-core/
- State Management in ASP.NET Core https://code-maze.com/state-management-in-asp-net-core-mvc/