Web Programming 2 (C#)

Week 2

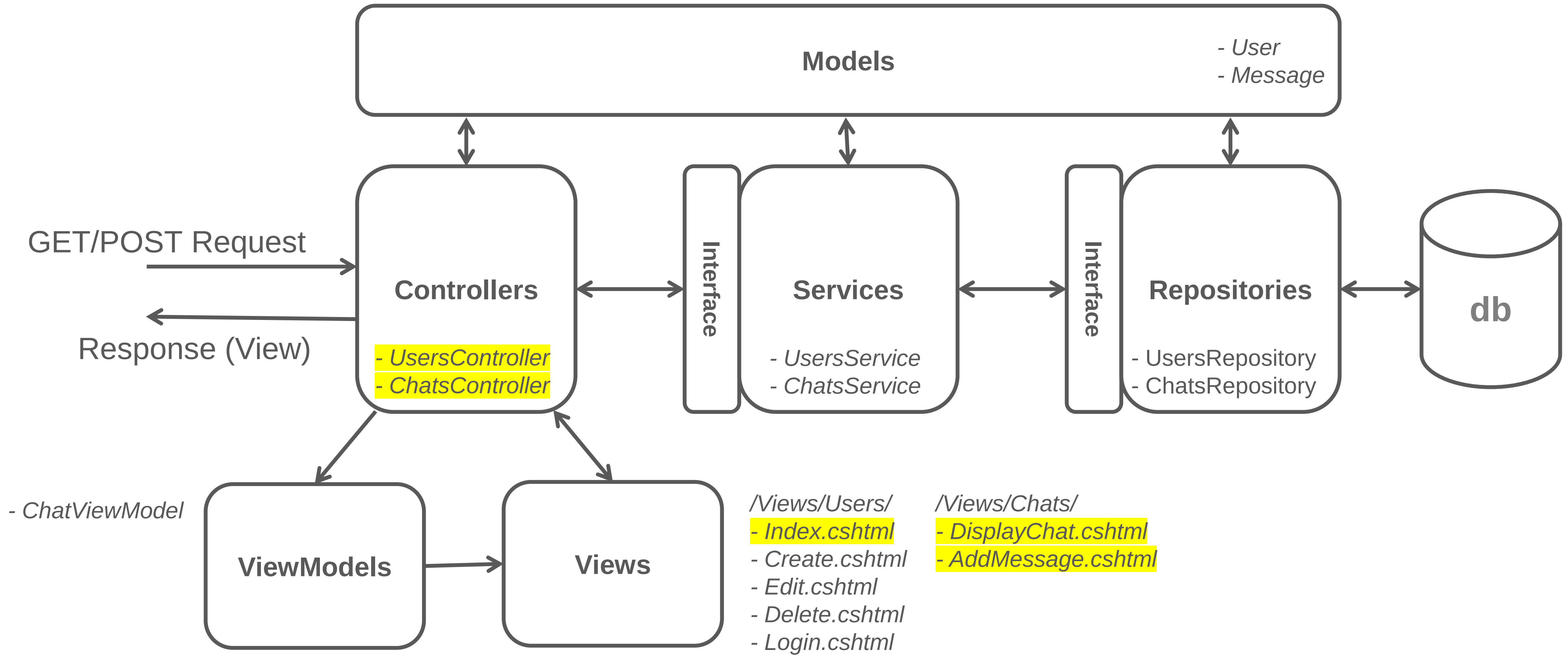
Web Programming 2 program

01 (wk-16)	ViewModels / Passing data to a View
02 (wk-17)	**State Management**
03 (wk-18)	*break*
04 (wk-19)	Services
05 (wk-20)	Partial views / View Components
06 (wk-21)	…
07 (wk-22)	…
08 (wk-23)	repetition

----

09 (wk-24)	exams term 4
10 (wk-25)	retake exams term 3
11 (wk-26)	retake exams term 4

Architecture WhatsUp application



State Management / Sessions

State Management

- The HTTP protocol is **stateless**
- This means that for the web application every request is independent, once the response is sent back to the client, the server "forgets" everything about the request

- Most web applications needs to maintain some sort of state about the clients, like a shopping cart, preferences, …
- In the previous term (Web Programming 1) we have used a cookie to remember the logged in user
- Remember that a cookie is **transferred back and forth** between browser and web server: browser → web server (in the HTTP request), browser ← web server (in the HTTP response)
- For more complex data this sending back and forth is not appropriate

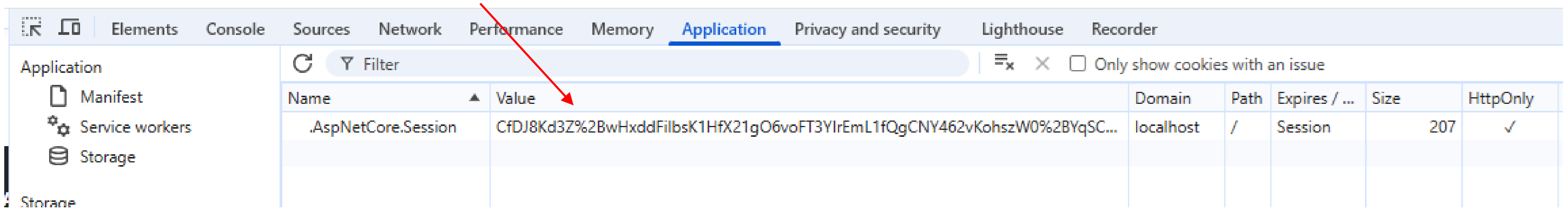
Sessions

– A common technique for implementing state management in web applications is a **Session**

– A Session can be seen as a period of time during which a user interacts with a website or web application

– A Session typically starts when the user logs in or accesses the site, and ends when the user logs out or closes the browser, or due to session expiration

– Every Session has a unique **Session ID** (a random string) that is commonly stored in a cookie

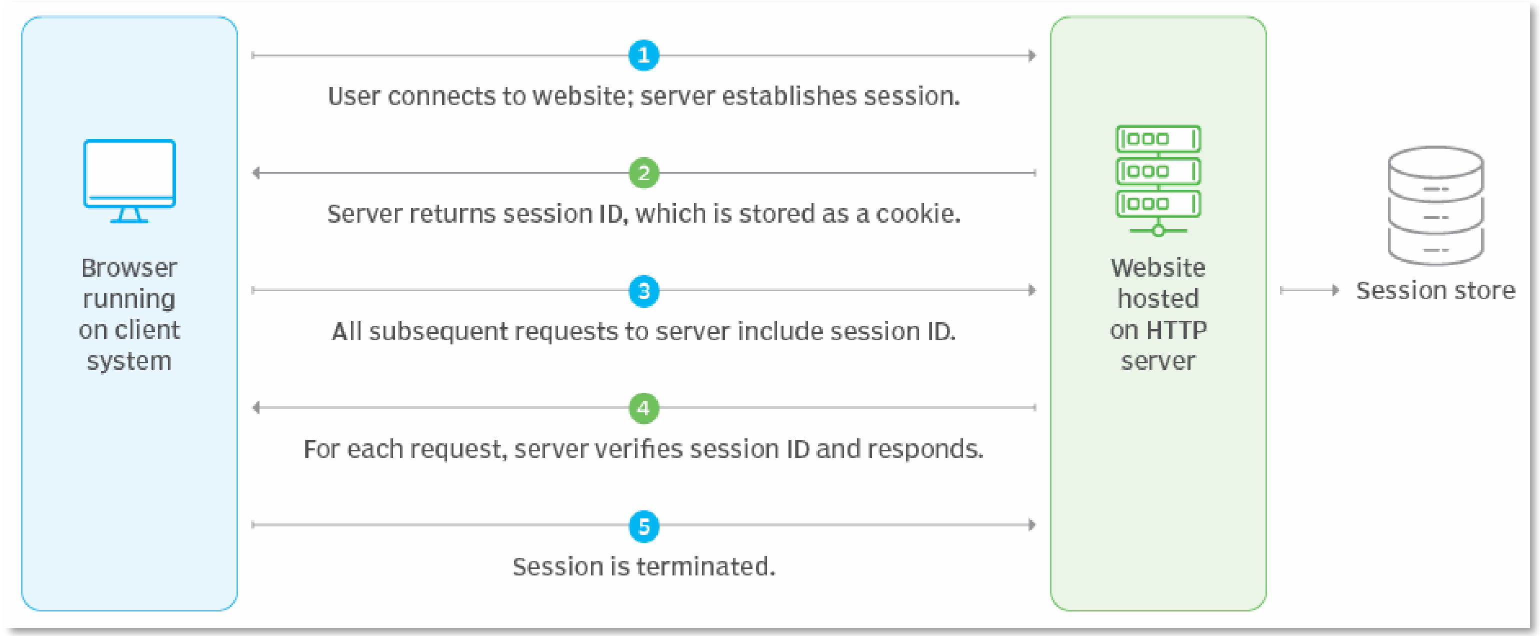


Name	Value	Domain	Path	Expires / ...	Size	HttpOnly
.AspNetCore.Session	CfDJ8Kd3Z%2BwHxddFilbsK1HfX21gO6voFT3YIrEmL1fQgCNY462vKohszW0%2BYqSC...	localhost	/	Session	207	✓

Storage for Session data

– For each Session, the server can **store information on the server** (e.g. a shopping cart with items in it), like a Dictionary with the Session ID as the key

– For every request with a Session ID (cookie), the server restores the state (e.g. a shopping cart) and use it


Browser running on client system
1
User connects to website; server establishes session.
2
Server returns session ID, which is stored as a cookie.
3
All subsequent requests to server include session ID.
4
For each request, server verifies session ID and responds.
5
Session is terminated.
Website hosted on HTTP server
Session store

Enabling Sessions

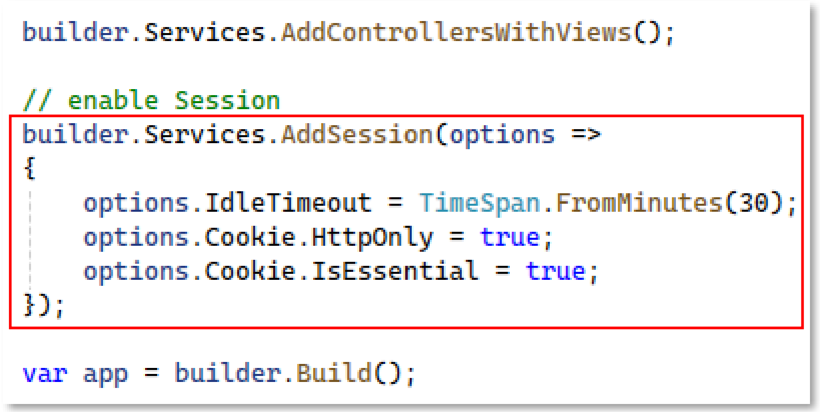
– Before Sessions can be used in an ASP.NET MVC web application, we have to enable it

– A few lines need to be added to **Program.cs**

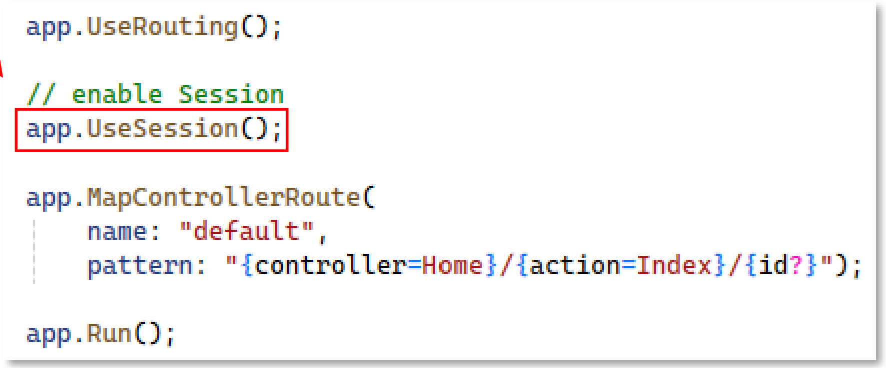
– The **IdleTimeOut** indicates that the Session ends when the user has no communication with the web application for 30 minutes *(in this example)*

– **HttpOnly** is set to true for security reasons
(the cookie can not be accessed with JavaScript)

– The cookie for a Session is marked as **IsEssential**, so no user consent is required



```
builder.Services.AddControllersWithViews();

// enable Session
builder.Services.AddSession(options =>
{
    options.IdleTimeout = TimeSpan.FromMinutes(30);
    options.Cookie.HttpOnly = true;
    options.Cookie.IsEssential = true;
});

var app = builder.Build();
```



```
app.UseRouting();

// enable Session
app.UseSession();

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

app.Run();
```

UsersController
(using Session)

Storing the logged in user in a Session

- Once Sessions are enabled in the WhatsUp application, we can change the storage of the user id from a cookie to a Session

- Instead of storing the user id in a Session, the **complete user object** will be stored
- Storing the complete user object in a Session has benefits: we can display the name of the user (e.g., in the chat overview) or as the person that is logged in (e.g., in the header)

- We need to change some code in the **UsersController**: a cookie is used in action Index (reading the cookie) and in action Login (creating the cookie) → this needs to changed to a Session

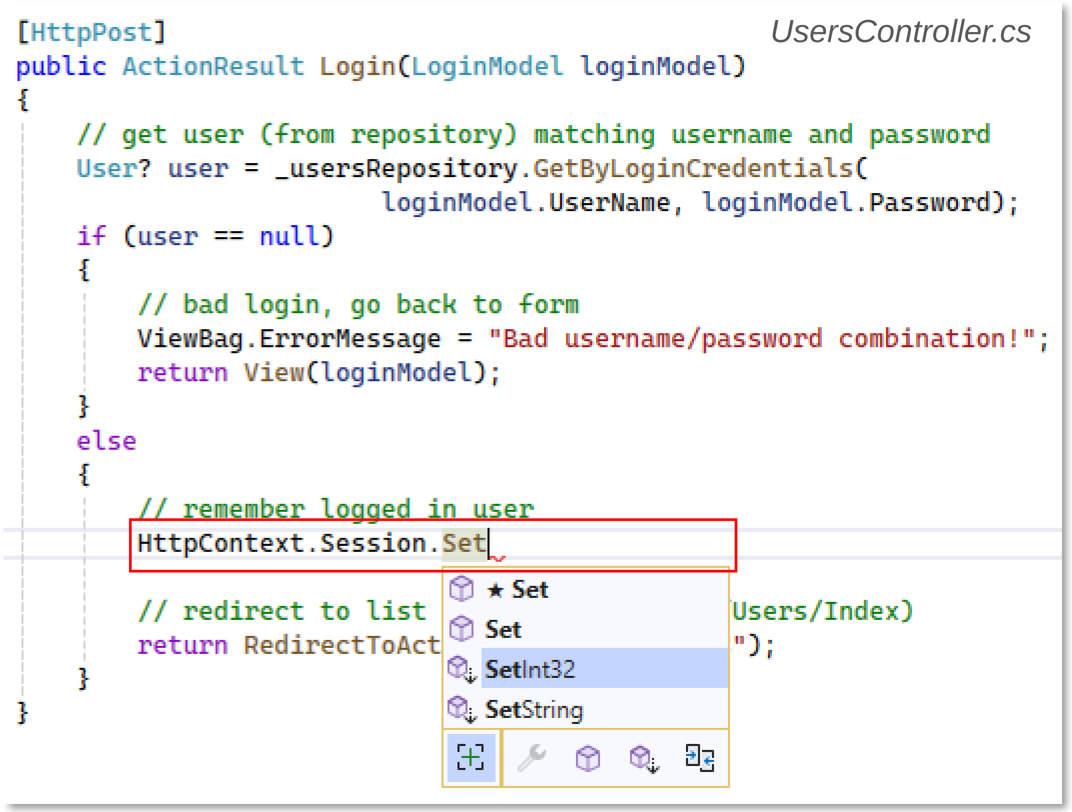
UsersController

– After successfully log in (method **Login**), we need to store the user in the Session

– The Session can simply be accessed with:
  HttpContext.Session

– As can be seen in the screenshot, we can only store an Integer or a String

– So, how can we store a complete user object?!

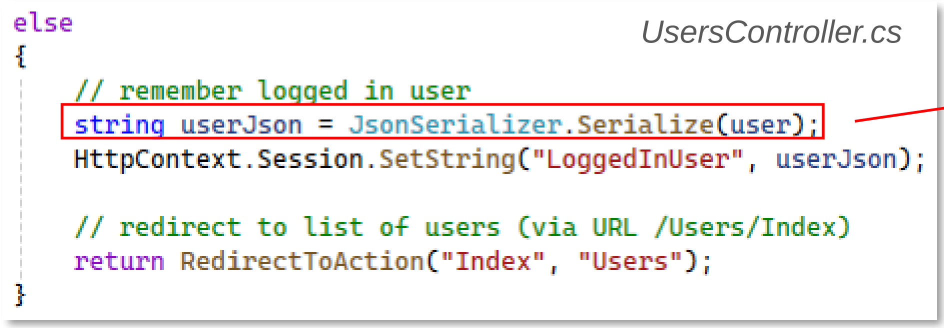


UsersController.cs

```
[HttpPost]
public ActionResult Login(LoginModel loginModel)
{
    // get user (from repository) matching username and password
    User? user = _usersRepository.GetByLoginCredentials(
                        loginModel.UserName, loginModel.Password);
    if (user == null)
    {
        // bad login, go back to form
        ViewBag.ErrorMessage = "Bad username/password combination!";
        return View(loginModel);
    }
    else
    {
        // remember logged in user
        HttpContext.Session.Set

        // redirect to list                   Users/Index)
        return RedirectToAct                  ");
    }
}
```

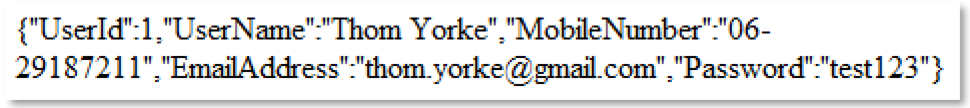
UsersController

- Solution: we have to convert the user object into a string, and store that string in the Session
- The conversion can be done with **JsonSerializer** (namespace **System.Text.Json**)
- Method **Serialize** converts the User object into a **JSON** string ("JavaScript Object Notation")
- JSON is a lightweight and human-readable format for storing and exchanging data, used a lot in web development



UsersController.cs

```
else
{
    // remember logged in user
    string userJson = JsonSerializer.Serialize(user);
    HttpContext.Session.SetString("LoggedInUser", userJson);

    // redirect to list of users (via URL /Users/Index)
    return RedirectToAction("Index", "Users");
}
```

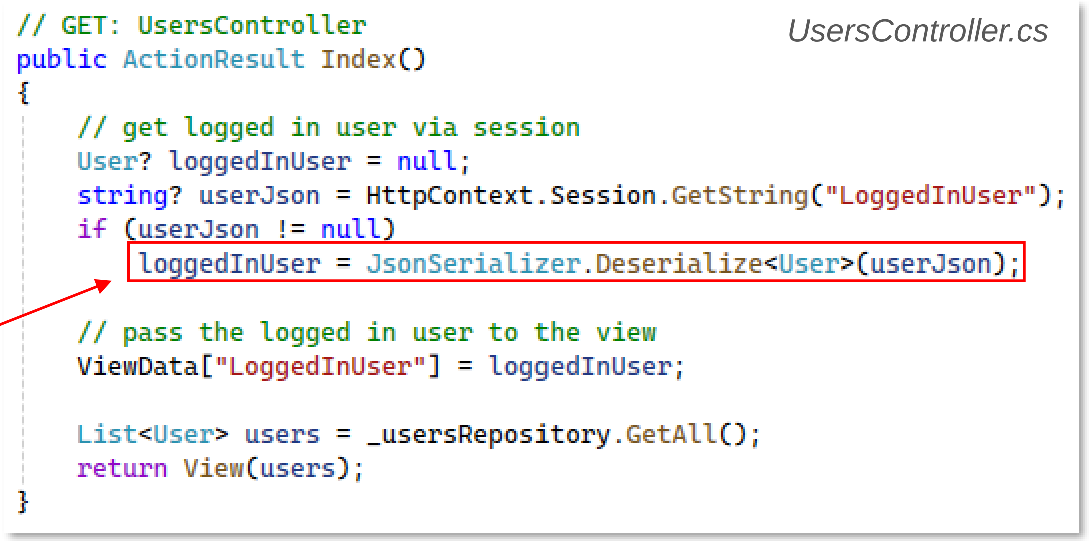


```
{"UserId":1,"UserName":"Thom Yorke","MobileNumber":"06-29187211","EmailAddress":"thom.yorke@gmail.com","Password":"test123"}
```

- The Json string (in variable "userJson") can now be stored in the Session with method **SetString**

UsersController

- We also need to change the **Index** method of the UsersController
- There, we have to read back the JSON data (representing the logged in user) from the Session, and convert that into a User object
- We use **GetString** to get the string from the Session
- If there was a string stored for key "LoggedInUser", then this string is used to **Deserialize** it (converting the JSON string into a User object)



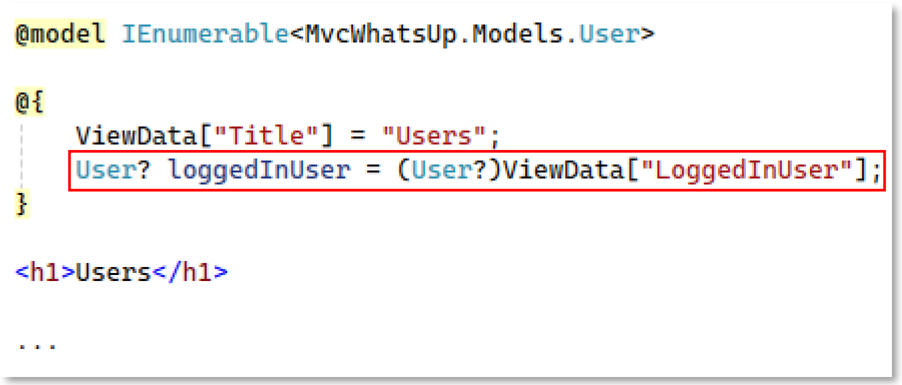
UsersController.cs

```
// GET: UsersController
public ActionResult Index()
{
    // get logged in user via session
    User? loggedInUser = null;
    string? userJson = HttpContext.Session.GetString("LoggedInUser");
    if (userJson != null)
        loggedInUser = JsonSerializer.Deserialize<User>(userJson);

    // pass the logged in user to the view
    ViewData["LoggedInUser"] = loggedInUser;

    List<User> users = _usersRepository.GetAll();
    return View(users);
}
```

- We can now pass the (logged in) **user object** to the Index view using ViewData

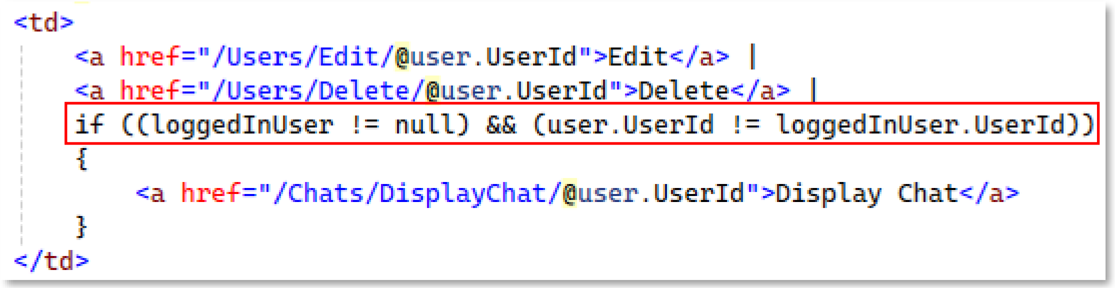
Index View

- The Index view of the UsersController first received the id of the logged in User, but now the whole user object is available (in ViewData)



```
@model IEnumerable<MvcWhatsUp.Models.User>

@{
    ViewData["Title"] = "Users";
    User? loggedInUser = (User?)ViewData["LoggedInUser"];
}

<h1>Users</h1>

...
```

- Also change the other parts in this View, to use the user object (instead of the user id)



```
<td>
    <a href="/Users/Edit/@user.UserId">Edit</a> |
    <a href="/Users/Delete/@user.UserId">Delete</a> |
    if ((loggedInUser != null) && (user.UserId != loggedInUser.UserId))
    {
        <a href="/Chats/DisplayChat/@user.UserId">Display Chat</a>
    }
</td>
```

Class exercise

- Change methods Login and Index in the UsersController to use a Session (instead of a cookie) to remember the logged in user
- Use JsonSerializer.Serialize to convert the user object to a string and use JsonSerializer.Deserialize to convert the (Session) string back to a user object

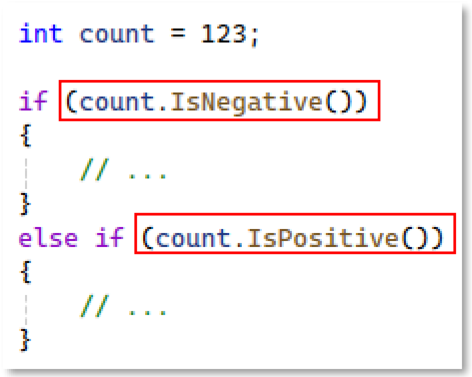
- Pass the user object from method Index to the View (using ViewData), and update the Index View accordingly

Extension methods

Extension methods

- Wouldn’t it be nice if **HttpContext.Session** not only had methods to set/get integers and strings, but also methods to set/get objects?
- It would make the code it bit cleaner, no need to convert the object into a string and back into an object
- In C#, methods can be added (injected) to existing data types/classes by so called **extension methods**, without changing these data types/classes or deriving from them

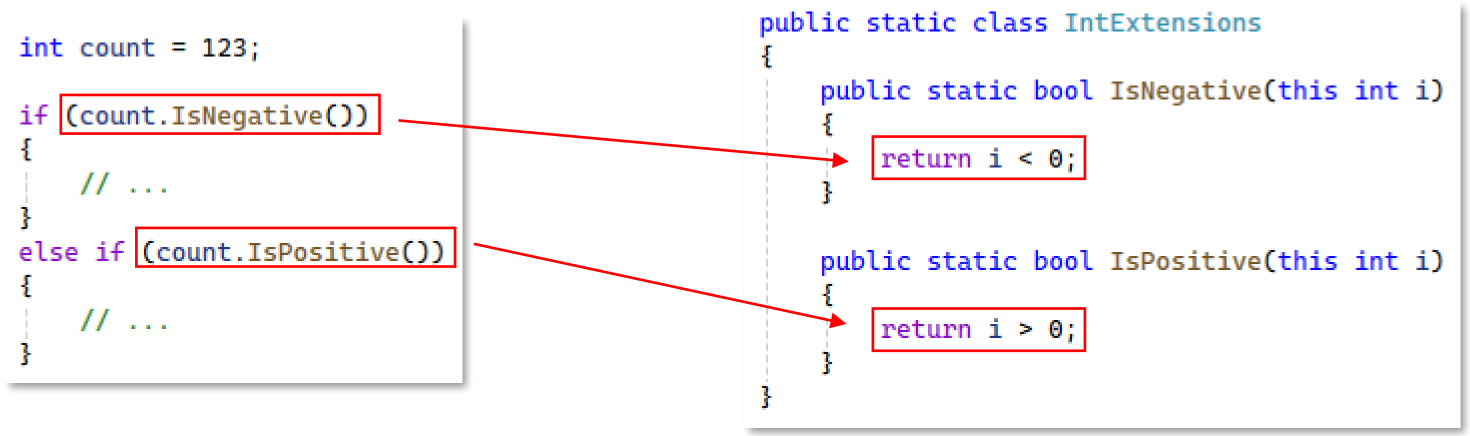
- In the example to the right, methods **IsNegative** and **IsPositive** can be used on an **int** variable, but these 2 methods are not available by default
- These 2 methods have been added with extension methods (see next slide)



```
int count = 123;

if (count.IsNegative())
{
    // ...
}
else if (count.IsPositive())
{
    // ...
}
```

Extension methods

- Extension methods are **static methods** defined in a **static class**
- As you can see in the screenshot, a static class IntExtensions is defined, containing 2 static methods: IsNegative and IsPositive
- The first parameter specifies which type the method operates on (preceded by the "this" keyword)
- These 2 static methods will make be it possible to use **IntExtensions.IsNegative(count)** and **IntExtensions.IsPositive(count)**, but also to use **count.IsNegative()** and **count.IsPositive()**, using your variable (count) as parameter (i) in these 2 methods

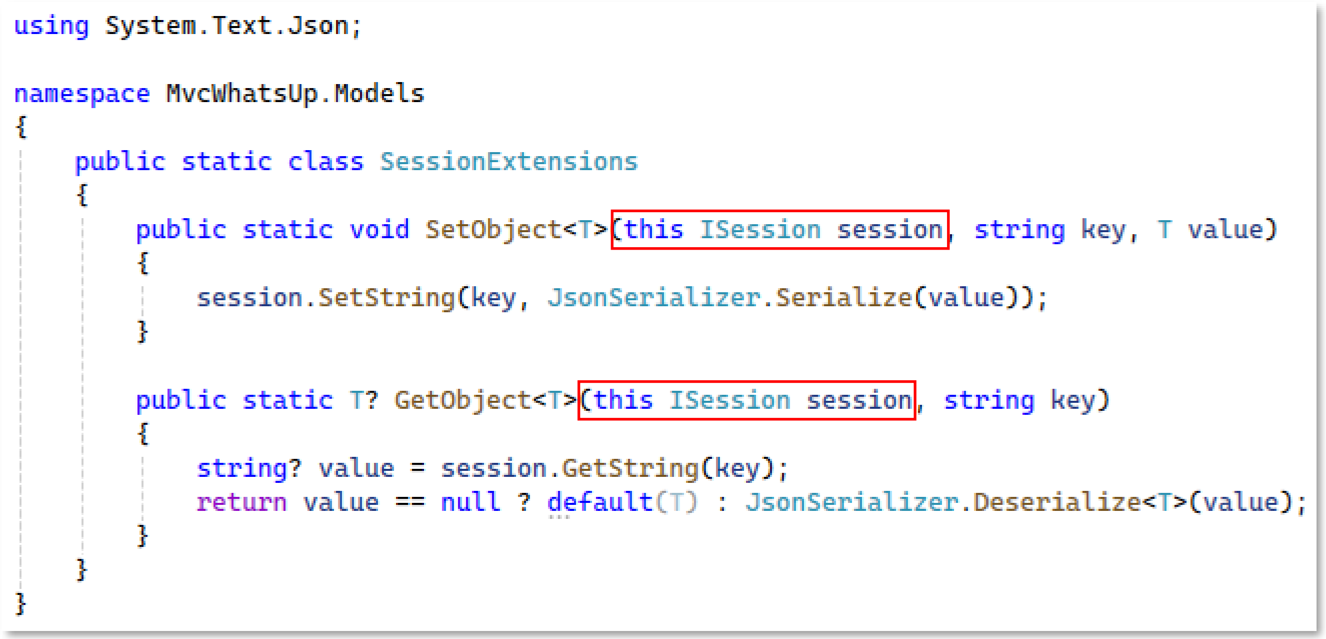


```
int count = 123;

if (count.IsNegative())
{
    // ...
}
else if (count.IsPositive())
{
    // ...
}
```

```
public static class IntExtensions
{
    public static bool IsNegative(this int i)
    {
        return i < 0;
    }

    public static bool IsPositive(this int i)
    {
        return i > 0;
    }
}
```

Extension methods

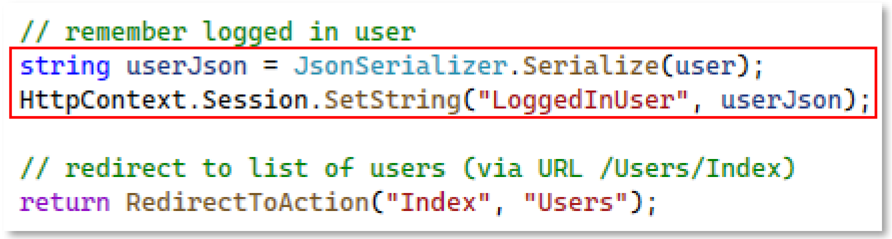
- Now let’s add methods **SetObject** and **GetOject** to HttpContext.Session
- Since HttpContext.Session is of type ISession, "this ISession" must be the first parameter
- The other 2 parameters are the key to be used and the object to be stored in the Session
- <T> is a generic type, which can be any class (like class User)
- It’s ok if you don’t fully understand the code, but with these 2 extension methods we can use Session.GetObject(…) and Session.SetObject(…)



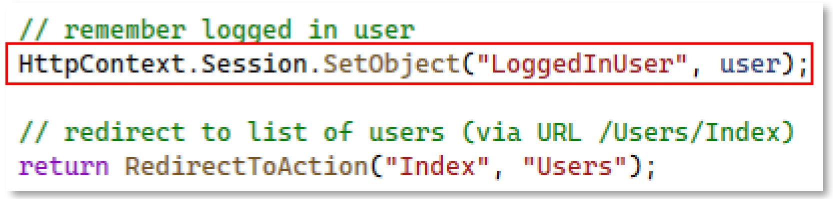
```
using System.Text.Json;

namespace MvcWhatsUp.Models
{
    public static class SessionExtensions
    {
        public static void SetObject<T>(this ISession session, string key, T value)
        {
            session.SetString(key, JsonSerializer.Serialize(value));
        }

        public static T? GetObject<T>(this ISession session, string key)
        {
            string? value = session.GetString(key);
            return value == null ? default(T) : JsonSerializer.Deserialize<T>(value);
        }
    }
}
```

Simplify actions in UsersController

– Method Login can now be simplified:

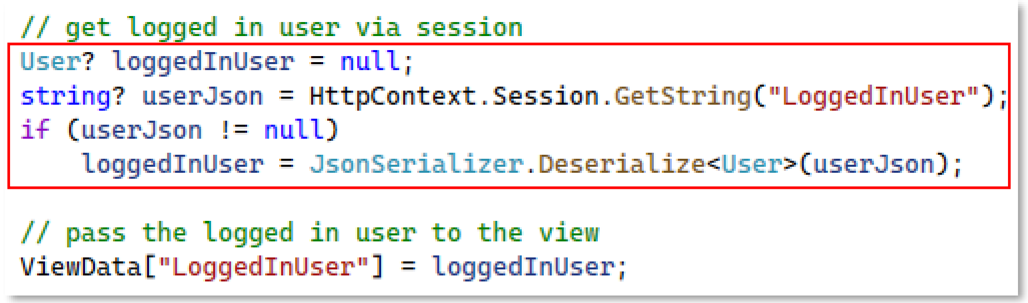


```
// remember logged in user
string userJson = JsonSerializer.Serialize(user);
HttpContext.Session.SetString("LoggedInUser", userJson);

// redirect to list of users (via URL /Users/Index)
return RedirectToAction("Index", "Users");
```

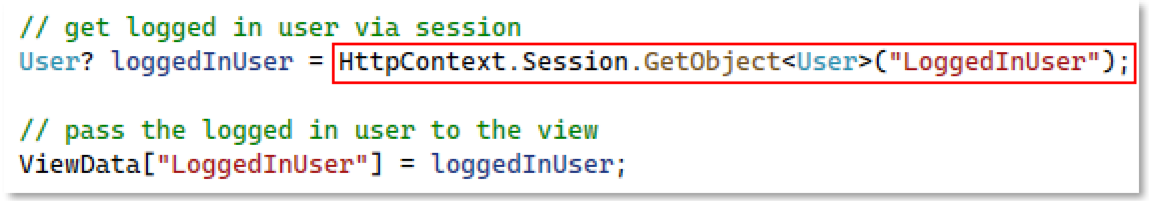


```
// remember logged in user
HttpContext.Session.SetObject("LoggedInUser", user);

// redirect to list of users (via URL /Users/Index)
return RedirectToAction("Index", "Users");
```

– and Method Index can now be simplified:



```
// get logged in user via session
User? loggedInUser = null;
string? userJson = HttpContext.Session.GetString("LoggedInUser");
if (userJson != null)
    loggedInUser = JsonSerializer.Deserialize<User>(userJson);

// pass the logged in user to the view
ViewData["LoggedInUser"] = loggedInUser;
```



```
// get logged in user via session
User? loggedInUser = HttpContext.Session.GetObject<User>("LoggedInUser");

// pass the logged in user to the view
ViewData["LoggedInUser"] = loggedInUser;
```

Class exercise

– Create the 2 Extension methods GetObject and SetObject for (interface) ISession, and use these 2 methods in the UsersController

ChatsController

(using Session)

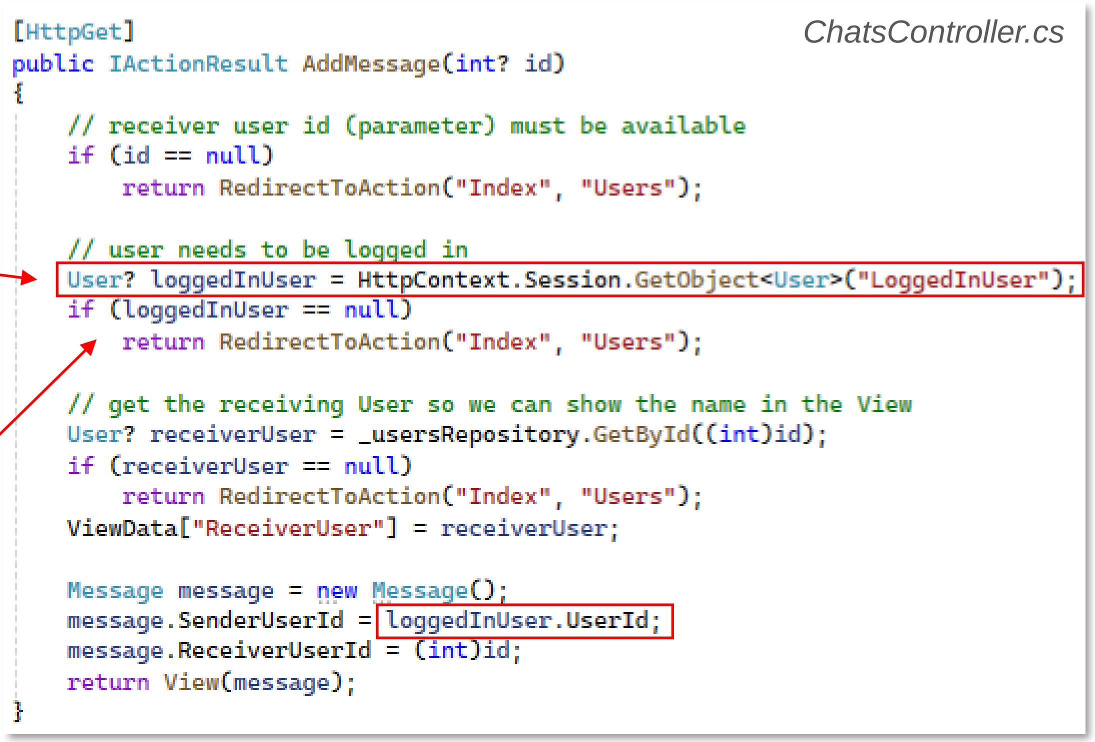
Updating the ChatsController - AddMessage

– We also need to change code in the **ChatsController**: a cookie is used in action AddMessage and in action DisplayChat, both reading the cookie → this needs to changed to a Session

– Let’s start with action **AddMessage**

– Instead of using a cookie to get the id of the logged in user, use the session to get the logged in User object *(using the GetObject method)*

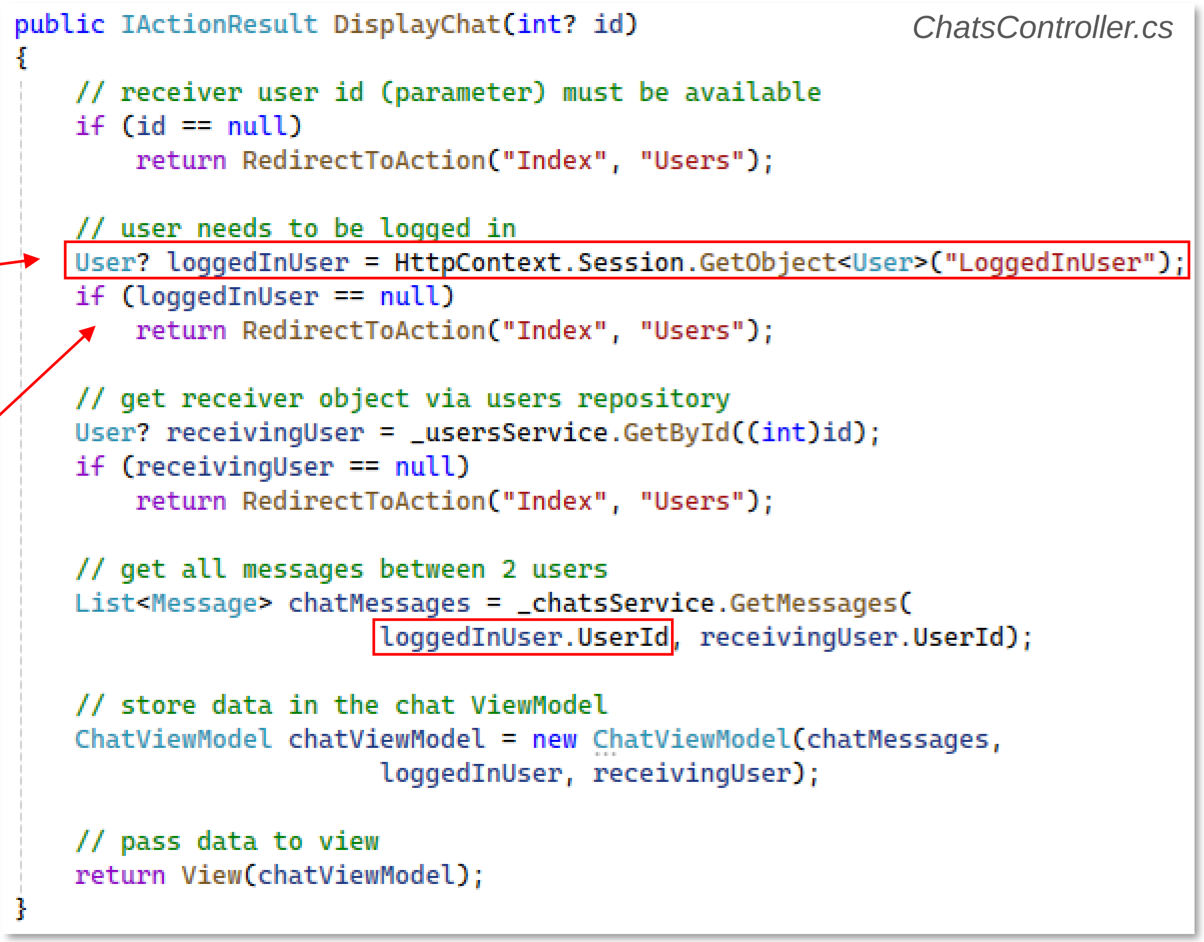
– If there is no logged in user, or no (valid) receiver user, redirect to URL /Users/Index


ChatsController.cs

```
[HttpGet]
public IActionResult AddMessage(int? id)
{
    // receiver user id (parameter) must be available
    if (id == null)
        return RedirectToAction("Index", "Users");

    // user needs to be logged in
    User? loggedInUser = HttpContext.Session.GetObject<User>("LoggedInUser");
    if (loggedInUser == null)
        return RedirectToAction("Index", "Users");

    // get the receiving User so we can show the name in the View
    User? receiverUser = _usersRepository.GetById((int)id);
    if (receiverUser == null)
        return RedirectToAction("Index", "Users");
    ViewData["ReceiverUser"] = receiverUser;

    Message message = new Message();
    message.SenderUserId = loggedInUser.UserId;
    message.ReceiverUserId = (int)id;
    return View(message);
}
```

Updating the ChatsController - DisplayChat

- And also update action **DisplayChat**
- Instead of using a cookie to get the id of the logged in user, use the session to get the logged in User object
  (using the GetObject method)
- If there is no logged in user, or no (valid) receiver user, redirect to URL /Users/Index


ChatsController.cs

```
public IActionResult DisplayChat(int? id)
{
    // receiver user id (parameter) must be available
    if (id == null)
        return RedirectToAction("Index", "Users");

    // user needs to be logged in
    User? loggedInUser = HttpContext.Session.GetObject<User>("LoggedInUser");
    if (loggedInUser == null)
        return RedirectToAction("Index", "Users");

    // get receiver object via users repository
    User? receivingUser = _usersService.GetById((int)id);
    if (receivingUser == null)
        return RedirectToAction("Index", "Users");

    // get all messages between 2 users
    List<Message> chatMessages = _chatsService.GetMessages(
                    loggedInUser.UserId, receivingUser.UserId);

    // store data in the chat ViewModel
    ChatViewModel chatViewModel = new ChatViewModel(chatMessages,
                    loggedInUser, receivingUser);

    // pass data to view
    return View(chatViewModel);
}
```

Class exercise

– Change methods AddMessage and DisplayChat in the ChatsController to use a Session (instead of a cookie) to get the logged in user

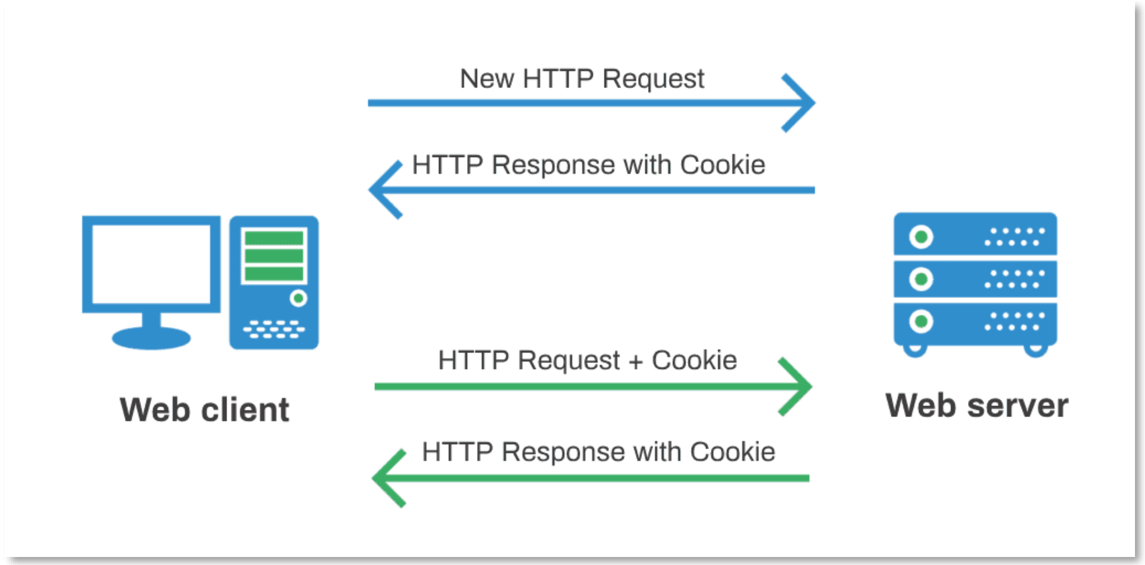
Cookies

(Session cookies and persistent cookies)

Cookies

- In the previous term (Web Programming 1) we already talked a bit about cookies
- A cookie is a small piece of text that is being send back and forth between a browser and a website

- Cookies are created by the web application and stored on the client side (user's computer) in the form of name-value pairs
- For subsequent requests to the same website, the browser automatically includes the cookie(s) in the request header



Cookies

– In this section we will discuss session cookies and persistent cookies, and do some exercises with them

– **Session cookies** (non-persistent cookies) are **temporary** cookies stored on the user’s computer while browsing a website. They are typically used to maintain a session state and are destroyed when the client closes their browser or navigates away from the web page

– **Persistent cookies** are **long-term** cookies stored on the user’s computer and used across multiple sessions (could be during days, months, years). They retain information such as preferred theme or language, allowing users to have a personalized experience when they revisit a website

Cookies

– A web application can create, read and delete cookies

- Creating and deleting cookies is done with the <u>Response</u> *(will be send back to the browser)*

Response.Cookies.Append("UserName", "SomeName");

→ an instruction to the browser to create the cookie

Response.Cookies.Delete("UserName");

→ an instruction to the browser to delete the cookie

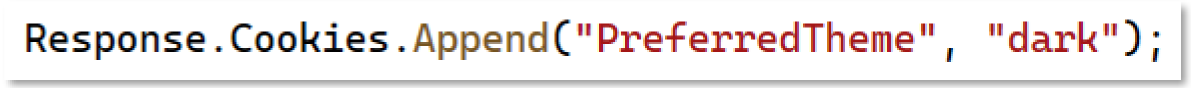
- Reading cookies is done with the <u>Request</u> *(received from the browser)*

string? userName = Request.Cookies["UserName"]

Creating a cookie

- A web application can create a cookie in order to remember certain settings for a user
- A cookie created in the response will be send back in the next request to the same website
- An example: the web application has an option to select a theme (dark, light, …) for the layout, the preferences for the current user is saved in a cookie

- Creating a cookie is done by using **Response.Cookies.Append** with key and value as parameters



```
Response.Cookies.Append("PreferredTheme", "dark");
```

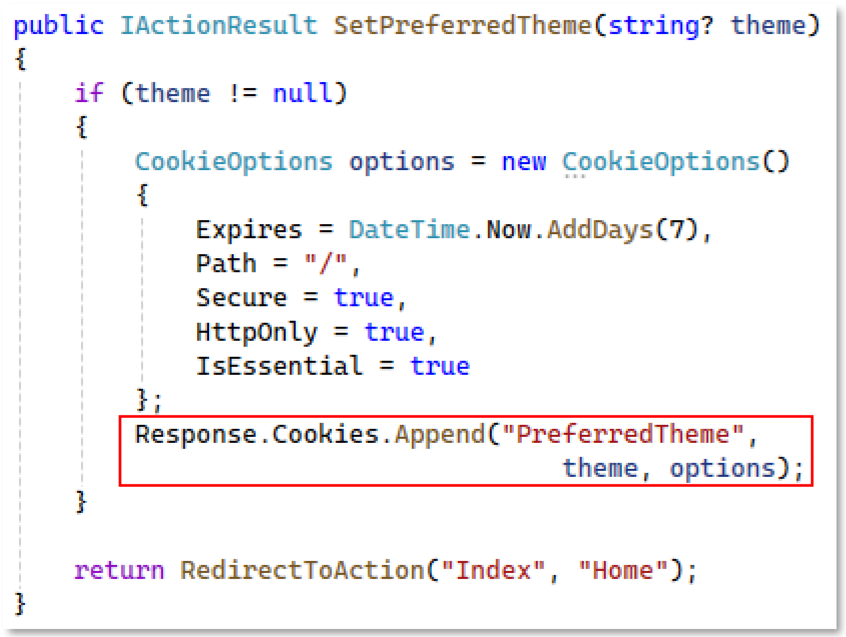
- If no options are given, the cookie is "Session bound“, meaning that it will be automatically deleted when the Session ends (by Session timeout or when the browser closes)

Creating a cookie (with options)

- When creating a cookie, cookie options can be used

- **Expires**: client's browser will delete the cookie after this date
- **Path** "/" → cookie is available within the entire application
- **Secure** true → cookie will only be sent over HTTPS
- **IsEssential** true → no consent needed (for essential cookies)
- **HttpOnly** true → prevent client-side scripts from accessing the cookie

- Setting the expire-date (to a future date) makes the cookie a **persistent cookie**
- Cookies created without an expire-date are **Session cookies**

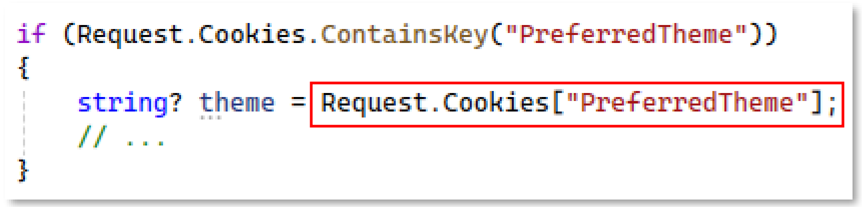


```
public IActionResult SetPreferredTheme(string? theme)
{
    if (theme != null)
    {
        CookieOptions options = new CookieOptions()
        {
            Expires = DateTime.Now.AddDays(7),
            Path = "/",
            Secure = true,
            HttpOnly = true,
            IsEssential = true
        };
        Response.Cookies.Append("PreferredTheme",
                                theme, options);
    }

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

Reading a cookie

- A web application can check if a cookie exists (from an earlier request), read it and act on it
- An example: the web application can set the layout according to a cookie (dark mode, light mode)

- Reading a cookie is done by using **Request.Cookies["key"]** with the cookie name as key
- Good practice to check whether the cookie exists before reading it

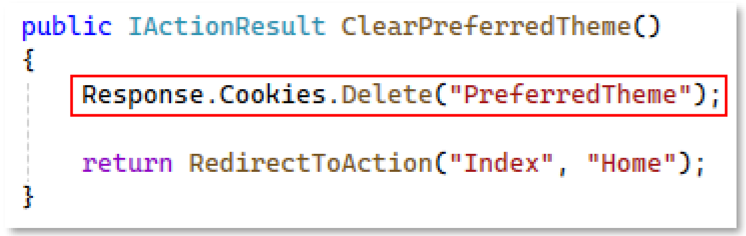


```
if (Request.Cookies.ContainsKey("PreferredTheme"))
{
    string? theme = Request.Cookies["PreferredTheme"];
    // ...
}
```

Deleting a cookie

- A web application can decide to delete an existing cookie
- An example: the user wants to go back to the default theme (*"forget my preferences"*)

- Deleting a cookie is done by using **Response.Cookies.Delete** with key as parameter
  → This is in fact an instruction for the web browser to delete the cookie



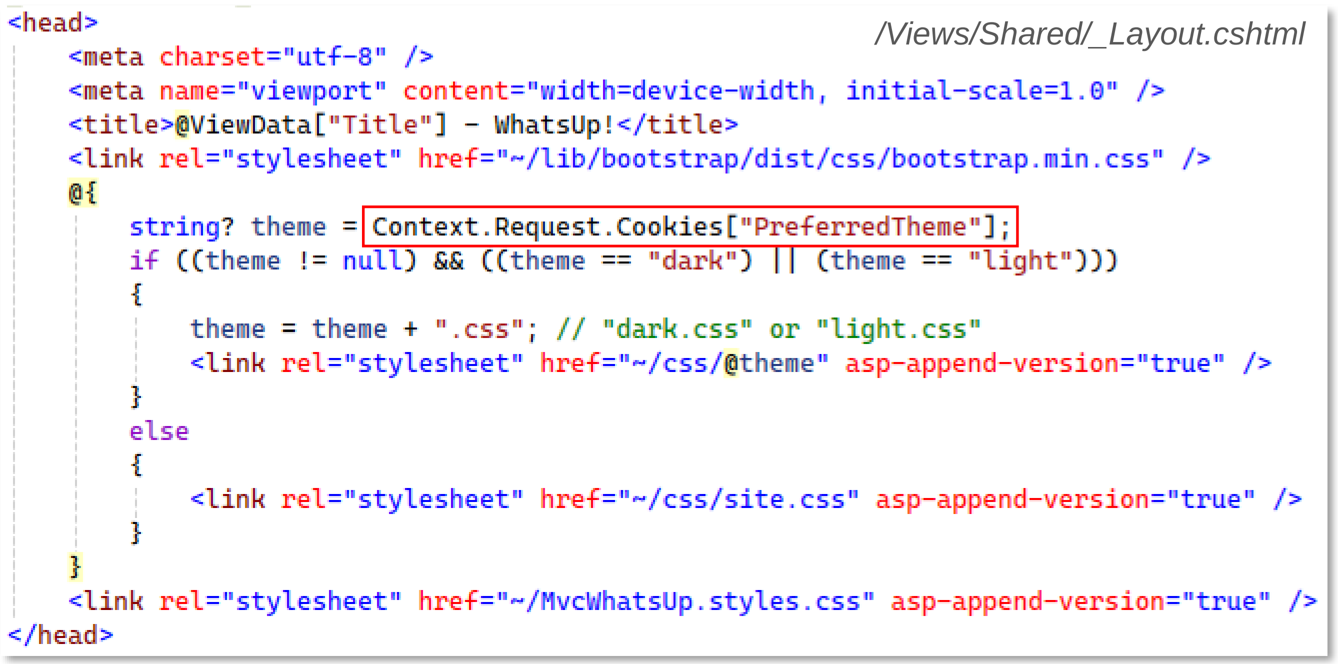
```
public IActionResult ClearPreferredTheme()
{
    Response.Cookies.Delete("PreferredTheme");

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

Activating a layout theme based on user preference

- To activate a certain layout based on user preferences, you can read a cookie in the default layout file to select a stylesheet

- In the example based on cookie "PreferredTheme" (dark or light), the corresponding stylesheet is used

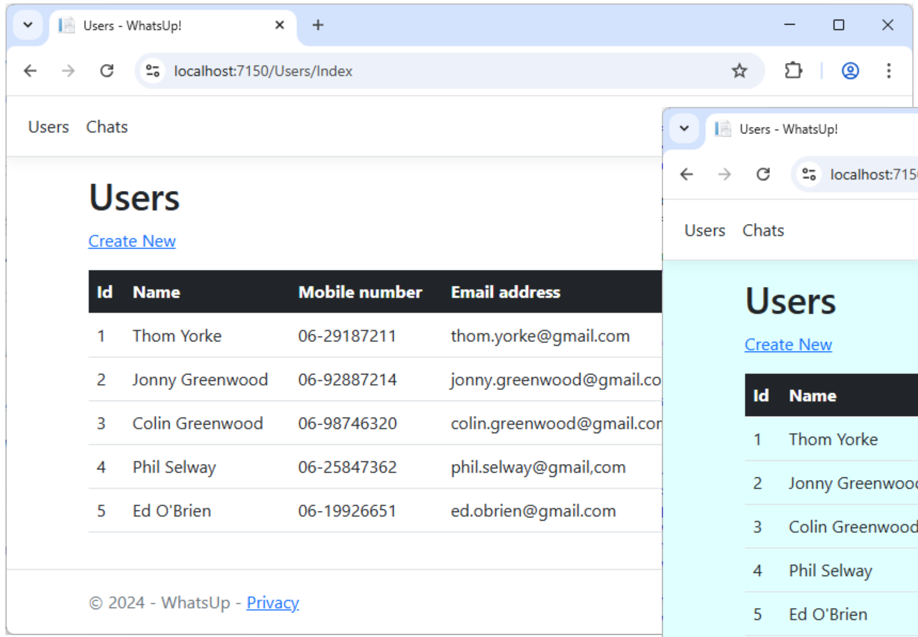
- If no cookie is set, then the default stylesheet is used (site.css)



/Views/Shared/_Layout.cshtml

```
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>@ViewData["Title"] - WhatsUp!</title>
    <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />
    @{
        string? theme = Context.Request.Cookies["PreferredTheme"];
        if ((theme != null) && ((theme == "dark") || (theme == "light")))
        {
            theme = theme + ".css"; // "dark.css" or "light.css"
            <link rel="stylesheet" href="~/css/@theme" asp-append-version="true" />
        }
        else
        {
            <link rel="stylesheet" href="~/css/site.css" asp-append-version="true" />
        }
    }
    <link rel="stylesheet" href="~/MvcWhatsUp.styles.css" asp-append-version="true" />
</head>
```

Activating a layout theme based on user preference



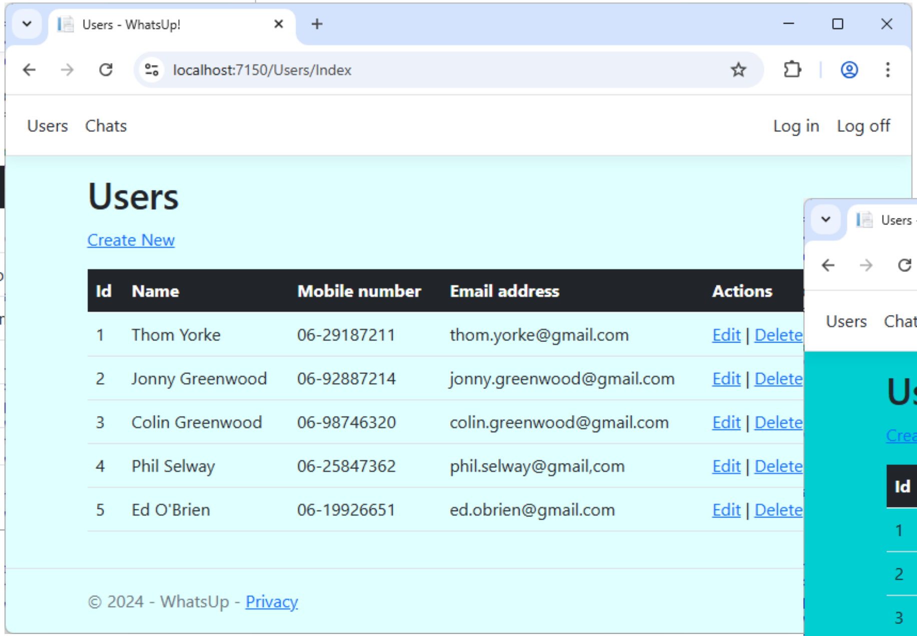
Users

Create New

Id	Name	Mobile number	Email address
1	Thom Yorke	06-29187211	thom.yorke@gmail.com
2	Jonny Greenwood	06-92887214	jonny.greenwood@gmail.co
3	Colin Greenwood	06-98746320	colin.greenwood@gmail.cor
4	Phil Selway	06-25847362	phil.selway@gmail,com
5	Ed O'Brien	06-19926651	ed.obrien@gmail.com

© 2024 - WhatsUp - Privacy

no cookie "PreferredTheme", using site.css



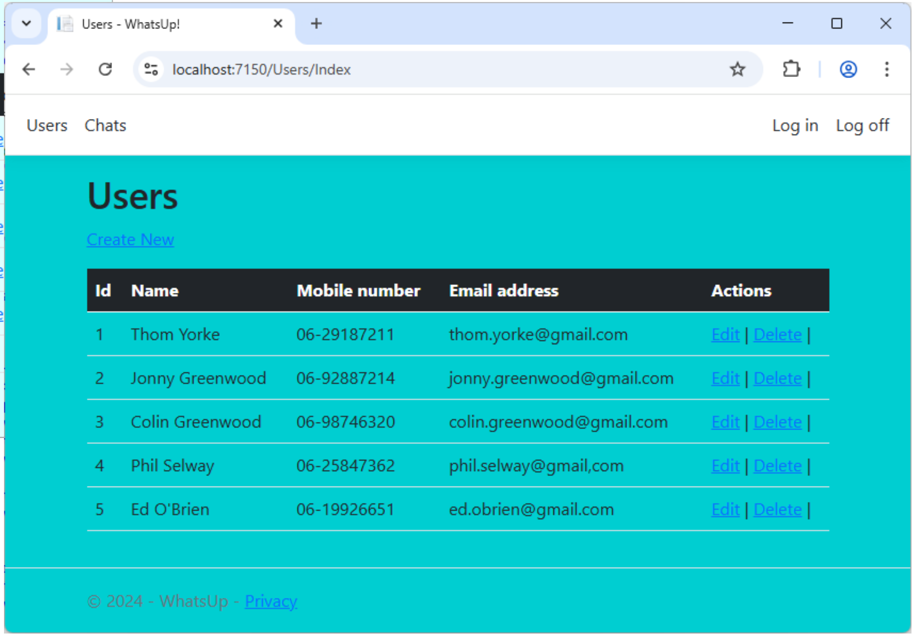
Users

Create New

Id	Name	Mobile number	Email address	Actions
1	Thom Yorke	06-29187211	thom.yorke@gmail.com	Edit | Delete
2	Jonny Greenwood	06-92887214	jonny.greenwood@gmail.com	Edit | Delete
3	Colin Greenwood	06-98746320	colin.greenwood@gmail.com	Edit | Delete
4	Phil Selway	06-25847362	phil.selway@gmail,com	Edit | Delete
5	Ed O'Brien	06-19926651	ed.obrien@gmail.com	Edit | Delete

© 2024 - WhatsUp - Privacy

cookie "PreferredTheme" set to "light", using light.css



Users

Create New

Id	Name	Mobile number	Email address	Actions
1	Thom Yorke	06-29187211	thom.yorke@gmail.com	Edit | Delete |
2	Jonny Greenwood	06-92887214	jonny.greenwood@gmail.com	Edit | Delete |
3	Colin Greenwood	06-98746320	colin.greenwood@gmail.com	Edit | Delete |
4	Phil Selway	06-25847362	phil.selway@gmail,com	Edit | Delete |
5	Ed O'Brien	06-19926651	ed.obrien@gmail.com	Edit | Delete |

© 2024 - WhatsUp - Privacy

cookie "PreferredTheme" set to "dark", using dark.css

Class exercise

– Add a Controller action to set a "theme" cookie, and a Controller action to delete a "theme" cookie
– Use a persistent cookie to set the theme (dark or light) of the WhatsUp application
– Change _Layout.cshtml in order to select the appropriate stylesheet (based on the "theme" cookie)

Homework – week 2

- Create the 2 extension methods GetObject and SetObject for (interface) ISession

- Change methods Login and Index in the **UsersController** to use a Session (instead of a cookie) to remember the logged in user → use the GetObject and SetObject (extension) methods
- Pass the user object from method Index to the View (using ViewData), and update the Index View accordingly

- Change methods AddMessage and DisplayChat in the **ChatsController** to use a Session (instead of a cookie) to get the logged in user → use the GetObject (extension) method

- Use a persistent cookie to set the theme (dark or light) of the WhatsUp application

inholland
hogeschool