# Exercises: State Management

Problems for exercises and homework for the "Web Dev Basics" course from the official "Applied Programmer" curriculum. In these exercises we will create a very **simple HTTP Server**. We will extend it every time and design it to mimic Microsoft’s IIS.

## Implement Cookie Functionality

In this task we will see how to add a **cookie functionality** to our server. The **cookies** will appear as part of the **responses** to the browser and we will be able to see them in the DevTools:

Graphical user interface, application

Description automatically generated

### Step 1: Create Cookies Classes

For our cookies we need to create a Cookie and a CookieCollection classes. Let's start with the **cookie class** first.

Create the Cookie **class** in the "HTTP" **folder** of the "BasicWebServer.Server" **project**:

Graphical user interface, text, application, chat or text message

Description automatically generated

Then, **write the class** itself (it is pretty **similar** to the Header **class**). As the **cookie is a header**, it should contain **properties** for the **cookie name** and **value**, accepted through a **constructor**. Also, it should **override** the ToString() **method** to **return the cookie in a correct format**. The Cookie **class** should look like this:

Graphical user interface, text, application

Description automatically generated

Create the CookieCollection **class**, as well. It is pretty similar to the HeaderCollection **class**, as it **stores the cookies**. There should be a **field for the cookies**, which is a **dictionary of a** string (the **cookie name**) and the **cookie** itself. This dictionary should be **initialized in the constructor**. Also, **add a method** for **adding a cookie to the collection** and **make the collection enumerable**.

The CookieCollection **class** is the following:

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Use these classes to **implement the cookie functionality** in the following steps.

### Step 2: Parse Cookies in the Request

As you know, cookies are **key-value pairs**, part of the **headers** of the **HTTP requests** from the browser to our server. We will need to **get the cookies separately** from the **headers** to use them.

To do this, let's first **define the cookie headers** we will need as **constants** in the Header **class**. Add the following lines:

Text

Description automatically generated

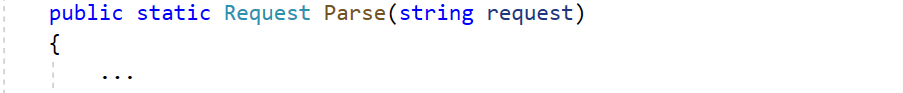
Now let's **modify** the Request **class** to **read the cookies**. Start by **creating a field for the cookies**:

Graphical user interface, text, application

Description automatically generated

Then, in the Parse(string request) **method** of the Request **class** you should **invoke a method for parsing the cookies** and **add the cookies** to the **request**:

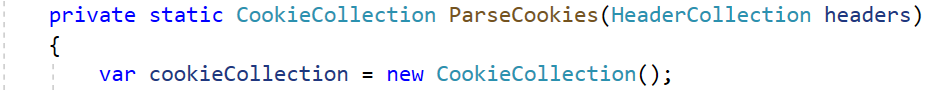
A picture containing text

Description automatically generatedGraphical user interface, text, application

Description automatically generated

**Implement** the ParseCookies(HeaderCollection headers) **method**, which should **return a** CookieCollection. **Initialize the collection** in the method:

A picture containing text

Description automatically generated

Now we should **get the header** with **name** "Cookie", **separate the name and value** of each cookie, and **add cookies to** **the collection**. Do it like this:

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

Finally, **return** the CookieCollection to the **method**:

A picture containing text

Description automatically generated

Now we have **parsed** and **separated the cookies** from the HTTP request. Now let's **add them to the response**.

### Step 3: Add Cookies to the Response

Our task is to **modify our response** and **add** "Set-Cookie" **headers** to it. This way, we will **send the cookies** to the browser.

Go to the Response **class** and add a **cookie collection** to store cookies like this:

Graphical user interface, text, application

Description automatically generated

**Modify** the ToString() **method**, as well. Each **header with a cookie** should be in a format "Set-Cookie: <cookie-name>=<cookie-value>". We should **add the cookies headers** to the response like this:

A picture containing text

Description automatically generatedGraphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

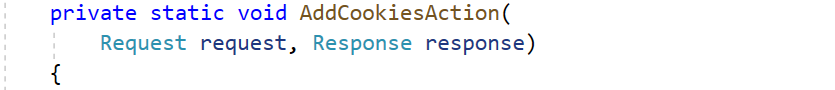
Note that the cookie uses its **overridden** ToString() **method** here to **return the name and value in a correct format**.

### Step 4: Modify the Startup Class

As we can now **read the cookies** from the **Http request** and **send them as a response**, let's **add cookies to the response** when the app is accessed for the first time.

To do this, go to the Startup **class** and **create the** AddCookiesAction(Request request, Response response) **method**, which should **accept a request and a response**:

A picture containing text

Description automatically generated

Then, check if the **request contains cookies** and create an **empty** **variable** for the **response body text**, which will be returned later:



If we have **any cookies from the response**, we should display them in **HTML format**. If there aren't any, show the "No cookies yet!" **message**. **Modify the response body** to change the response content:

A picture containing graphical user interface

Description automatically generated

At the end, **add cookies to the response** if there aren't any coming from the request:

Text

Description automatically generated with medium confidence

You now have an **action method** for **adding cookies**. Use it in the Main() **method**, so that **cookies are added to the** **response** when "/Cookies" is accessed:

A picture containing text

Description automatically generatedText

Description automatically generated

Text

Description automatically generated

To do this, you should **add a pre-render action parameter** to the HtmlResponse **class** like this:

Graphical user interface, text

Description automatically generated with medium confidence

### Step 5: Examine Cookies in the Browser

**Run** the "BasicWebServer.Demo" **project**, **open the browser** and **navigate** to "/Cookies". Look at the [Application] **tab** in the DevTools – our **cookies should be present**:

Graphical user interface, application

Description automatically generated

Note that the **correct message** is returned, as well. Now try to **delete the cookies** and **refresh the page** – they should be **created again**.

**Examine the request** and **response headers** in the [Network] **tab**. When cookies are created for the **first time**, the request has **no header** "Cookie" but the response has the "Set-Cookie" **headers**:

Graphical user interface, text, application, Word

Description automatically generated

If you already **have the cookies set** and **refresh the Web page**, you should see that the cookies are now part of the "Cookie" **request header**:

Graphical user interface, text, application

Description automatically generated

Note that the "Set-Cookie" **headers** are sometimes **present in the response**, even though the **browser doesn't need them** anymore (when **cookies are already created**). This happens, as the **headers** are added to the given **response** and the **same response** is returned until the **server is refreshed**.

However, if you **open another page** in the app while you have the "Cookies" **header** in the **request**, you will see that **they are not present**:

Graphical user interface, text, application, email

Description automatically generated

## Implement Session Functionality

As you know, the **session is actually stored as a cookie**, but it is the same for a given user, until the browser is closed. That's why the **session implementation** is pretty similar to the cookies' one, but it has its differences. Let's **implement a session**, which should appear like this:

Graphical user interface

Description automatically generated with medium confidence

### Step 1: Create Session Class

Our first task is to **create** the Session **class** in the "HTTP" **folder** of **our server project**. In this class, we will **create a** **constant field** for the **session cookie's name** and one for the **session key with the current date** (we'll need it later):

Graphical user interface, text

Description automatically generated

Graphical user interface, text

Description automatically generated

As you know, each **session** has a **session** ID and **session key-value data pairs**. The ID should be a **public property**, as we are going to use it outside of our class. The **data**, however, doesn't need to be accessed, so it will be a **private** **field**. The **constructor** of the class should **accept an** ID and **initialize the data collection**. Do it like this:

Graphical user interface, text, application

Description automatically generated

As we **cannot access the session data** from outside of the class, we will **add an indexer** for **getting** and **setting** it through the class:

Text

Description automatically generated

At the end, add a ContainsKey(string key) **method**, which should **check whether the session data dictionary** has the **given key** in a key-value pair:

Text

Description automatically generated with medium confidence

### Step 2: Get the Session from the Request

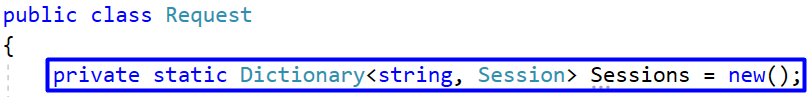
In order to work with the **session**, we need to **separate it from the HTTP request cookies**. Go to the Request **class** and **add a session property**:

Graphical user interface, text, application

Description automatically generatedGraphical user interface, text, application

Description automatically generated

Also, you could **add a collection of sessions** (for all users): these are all the sessions, which are **currently connected to the server**. To do this, you can create a SessionCollection **class** to **store the sessions**. We won't create a collection class but store them in a **private dictionary** like this:



Now **get the session** in the Parse(string request) **method** of the Request **class** by **invoking another method** and **adding the result to the returned request**:

A picture containing text

Description automatically generatedGraphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generatedGraphical user interface

Description automatically generated with medium confidence

Now **implement the** GetSession(CookieCollection cookies) **method**. In it, we should check whether there is **already a** **session** with **our session cookie name** and, if not, **create the session**.

To check whether **cookies contain a cookie with a given key** and then **get the value of the cookie**, we should **add an** **indexer** and a Contains(string name) **method** to the CookieCollection **class** like this:

Graphical user interface, text, application

Description automatically generated

Then go back to the Request **class** and **write** the GetSession(CookieCollection cookies) **method**. Check if the **session** **exists** and, if it does, get its ID. If it doesn't, then use the Guid **class** to **generate one**:

A picture containing text

Description automatically generatedText

Description automatically generated

On the next step, check if the **current sessions have a session** with this ID and **create a new session** if needed:

A picture containing text

Description automatically generated

Finally, **return the session**:

Text

Description automatically generated

Now our server is able to **get the session** from **HTTP requests**.

### Step 3: Modify the HTTP Server

At the end, let's **modify the server to add a session**, which has an ID and a **key-value paired data**. In the Start() **method** of the HttpServer **class** we will **invoke** **the** AddSession(Request request, Response response) **method**. We should **add it to the server class**, as the **session** should be part of **each response** to the browser:

A picture containing logo

Description automatically generated

A picture containing graphical user interface

Description automatically generatedGraphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

**Implement** the invoked method. Check if there is already a **session with the** **given key** (the **current date key**):

A picture containing logo

Description automatically generatedText

Description automatically generated with medium confidence

Finally, if the **session doesn't not exist**, **create one with the current date** and **add it to the request to save it**. Also, **add a cookie to the response with the session cookie name as name** and the **session** ID **as value**:

Graphical user interface, text, application

Description automatically generated

### Step 4: Examine Session in the Browser

Before you examine the result in the browser you should make a **small correction** in the Startup **class** of the "BasicWebServer.Demo" **project**. The AddCookiesAction(Request request, Response response) needs to be **modified** like this:

A picture containing text

Description automatically generatedA picture containing text

Description automatically generated

This is necessary because otherwise the **session cookie will prevent** the creation of **other cookies**.

**Run** the demo app, **open the browser** and **navigate to any page**. **Examine the request** and **response** **headers** in the [Network] **tab** of the DevTools. When the site is opened for the **first time**, the **session doesn't not exist** as part of the request but is **set as a cookie** in the **response headers**:

Graphical user interface, text, application, email

Description automatically generated

**Check the session cookie** in the [Application] **tab**. It should be **created** and should have a **session** ID:

Graphical user interface, text, application, email

Description automatically generated

Now **refresh the browser**. You will see that the **request now keeps the session**, which we created, and **sends it to the** **browser as a cookie**:

Graphical user interface, text, application, email

Description automatically generated

Go to the [Application] **tab** again. Notice that the **session** ID is **the same** from before- this means that we are **still** **connected to the same session** and a **new one was not created**:

Graphical user interface, text, application, email

Description automatically generated

Try to **close all browser windows** on your computer and **open the page** again- the **session should be a new one**:

Graphical user interface, text, application, email

Description automatically generated

### Step 5: Display Session Info on a Page

In this task, our aim is to **open** **the** "/Session" **page** and **obtain information** about the **date and time**, on which our **current session was created**. If the session is **just created** and it doesn't come from the request the "**Current** **date** **stored!**" **message** should be displayed. It looks like this:

Graphical user interface, text, application, chat or text message

Description automatically generated Graphical user interface, text, application, chat or text message

Description automatically generated

Start by creating an **action method** in the Startup **class** of the "BasicWebServer.Demo" **project**. The DisplaySessionInfoAction(Request request, Response response) **method** should **check whether a session with a key** is **created** and **display different text responses**, depending on this. Write the method like this:

A picture containing text

Description automatically generatedGraphical user interface, text, application, email

Description automatically generatedGraphical user interface, text, application, email

Description automatically generated

Then, **map** the "/Session" **path** to an TextResponse with the **action** in the Main() **method**:

A picture containing text

Description automatically generatedText

Description automatically generated

Text

Description automatically generated

Now **run the app** and **open** the "/Session" **page**. When the app is **accessed from the first time**, a **new session** should be created. The result is the following:

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

If you **refresh the page**, you will see that the **session is kept**. The page should display the **date and time**, on which the **session was created**:

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text

Description automatically generated

## Implement Login + Profile + Logout

In this task we shall implement:

* **Login form** (GET "/Login") + **login action** (POST "/Login")

Graphical user interface, application

Description automatically generatedGraphical user interface, text, application, chat or text message

Description automatically generated

* **"**UserProfile**"** page (GET "/UserProfile"), accessible only for logged-in users

Graphical user interface, text, application

Description automatically generatedGraphical user interface, text, application

Description automatically generated

* **Logout action** (GET "/Logout")

Graphical user interface, text, application, chat or text message

Description automatically generated

**Hardcode** the valid user + password: "user", "user123".

To implement the above **functionalities**, we will **create** and **delete session data**, in which we will keep track of the user. That's why we should first go to the Session **class** and **add a new property** with the **key** for the **user session**:

Text

Description automatically generated

### Implement "GET" to "/Login"

Let's start creating the **mappings** and the **methods** they need. Go to the Startup **class** and **create a mapping** for the "GET" **request** to "/Login" – a **login form** should be displayed.

First, create the **form** as a **field** in the Startup **class**, together with the **valid username** and **password**:

Graphical user interface, text, application

Description automatically generated

You can **copy the form** from here:

|  |
| --- |
| private const string LoginForm = @"<form action='/Login' method='POST'>  Username: <input type='text' name='Username'/>  Password: <input type='text' name='Password'/>  <input type='submit' value ='Log In' />  </form>"; |

The "GET" **request** to "/Login" should return the **login form** as **HTML**. Createthe **mapping** in the Main() **method** like this:

A picture containing text

Description automatically generatedText

Description automatically generated

Now go to "/Login" in the **browser** and you should see the **login form**. However, if you **submit** it, you should get an **error**, as we haven't implemented the "POST" for "/Login":

Graphical user interface, application

Description automatically generatedGraphical user interface, text, application, email

Description automatically generated

Note that a **session is created** as before, but it **doesn't contain the user key** yet. It only has the key-value pair for the **current date** (from the previous task):

Graphical user interface, text, application

Description automatically generated

### Implement "POST" to "/Login"

Let's implement the **mapping** and **method** for the "POST" **request** to "/Login". The response should be of type HtmlResponse, as if the **login is unsuccessful**, the **login** **form** should be **returned** again with an **invalid message**.

Create the mapping with **an empty response content** and with an **action**, which will contain the **login functionality** **logic**:

A picture containing text

Description automatically generatedText

Description automatically generated

Text

Description automatically generated

The LoginAction(…) method should **check whether the username and password are valid** and **create a session** if they are. If not, it should **return the login form**.

First, we will **clear the session data**, so that we have a **session only when the** **user is authenticated**.

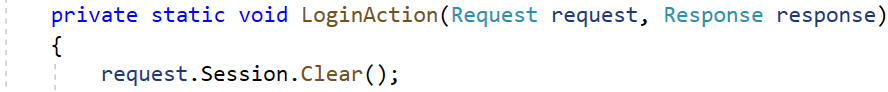
To do this, however, you should first go to the Session **class** and **add a method** for **removing all session key-value** **pairs** like this:

A picture containing text

Description automatically generated

Go back to the Startup **class** and **clear the current user session** in the LoginAction(…) **method**:

A picture containing text

Description automatically generated 

Then, create a **variable** for the **response text**. Add **variables** to check whether the submitted **username** and **password** in the form are the **valid ones**, as well:

Text

Description automatically generated with low confidence

If the **username and password are valid**, a **success message** should be returned. Also, a **new session** should be created and a **cookie** with it should be added to the **response**. In this way, if a **user is logged-in** but the **session is deleted**, the **user will stay logged-in** and a **new session** will be created.

Do it like this:

Graphical user interface, text

Description automatically generated

Otherwise, if the **username or password are wrong**, we should only **return the login form** to be submitted again:

Logo, company name

Description automatically generated

At the end, set the bodyText **variable** **content** as a **response body content**:

Text

Description automatically generated with low confidence

Now try out the **login functionality** in the browser. If you enter **valid credentials** the result should be the following:

Graphical user interface, application

Description automatically generated  Graphical user interface, text, application, email

Description automatically generated

Note that the **old session is not deleted** and its id is the same. However, the **session user data is cleared** and **added** **again when login is successful** and you can see this if you **debug your code**:

Graphical user interface, text, application, email

Description automatically generatedGraphical user interface, text, application

Description automatically generated

If the **login is not successful**, the **login form** will be displayed again:

Graphical user interface, application

Description automatically generated Graphical user interface, application

Description automatically generated

### Implement "GET" for "/Logout"

Implement the **logout functionality**, which should **remove the user session key-value pairs** and **return a success message**.

Map the "GET" **request** to "/Logout" to an **empty HTML response** and an **action**:

A picture containing text

Description automatically generatedText

Description automatically generated

The LogoutAction(…) **method** should look like this:

A picture containing text

Description automatically generated Text

Description automatically generated

**Try it out** in the browser by accessing "/Logout":

Graphical user interface, text, application, chat or text message

Description automatically generated

**Debug** and see that only the **user session key and value are deleted** and **not the whole session**. Note that you should be **logged-in** or the user session key and value won't be part of the request:

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

### Implement "GET" for "/UserProfile"

The "User Profile" **page** should **display the username of the currently logged-in user**. If the user is **not logged-in**, a **message** with a link to "/Login" should be displayed.

Create the **mapping** for "GET" to "/UserProfile":

A picture containing text

Description automatically generatedText

Description automatically generated

Text

Description automatically generated

Implement the GetUserDataAction(…) **method** – it should **set the response body message**, depending on whether the **session for the user exists or not**:

A picture containing text

Description automatically generatedGraphical user interface, text, application

Description automatically generated

Examine the **functionality** in the browser on "/UserProfile" when user is **authenticated** and when **not**:

Graphical user interface, text, application

Description automatically generatedGraphical user interface, text, application

Description automatically generated

Now **test the three functionalities together** and make sure everything works **correctly** – log in with valid and invalid user credentials, log out, look at the "User Profile" page, delete the session, open pages in multiple browser tabs, etc.