Lab: State Management & Asynchronous Programming

This document defines the lab problems for the

I. State Management

1. Examine Your Cookies

Most cookies are stored in a RDBMS, usually SQLite. You can download SQLite browser from here.
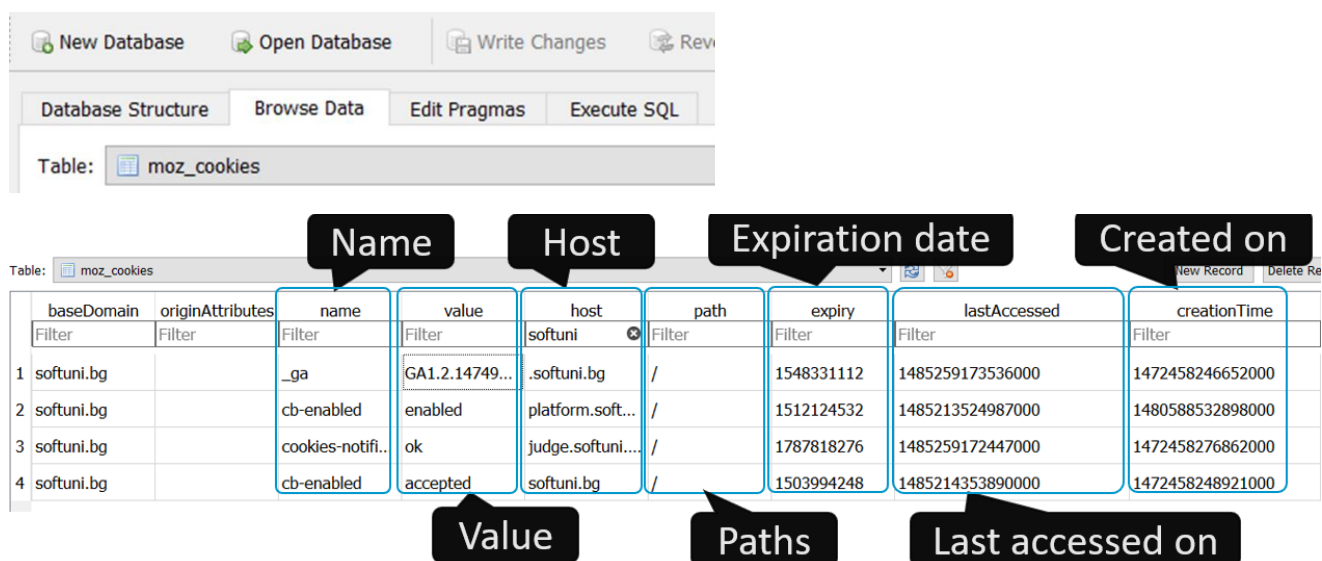The location of the Mozilla cookies is the following:

`C:\Users\{username}\AppData\Roaming\Mozilla\Firefox\Profiles\{name}.default\cookies.sqlite`

The location of the Chrome cookies is the following:

`C:\Users\{username}\AppData\Local\Google\Chrome\User Data\Default\Cookies`

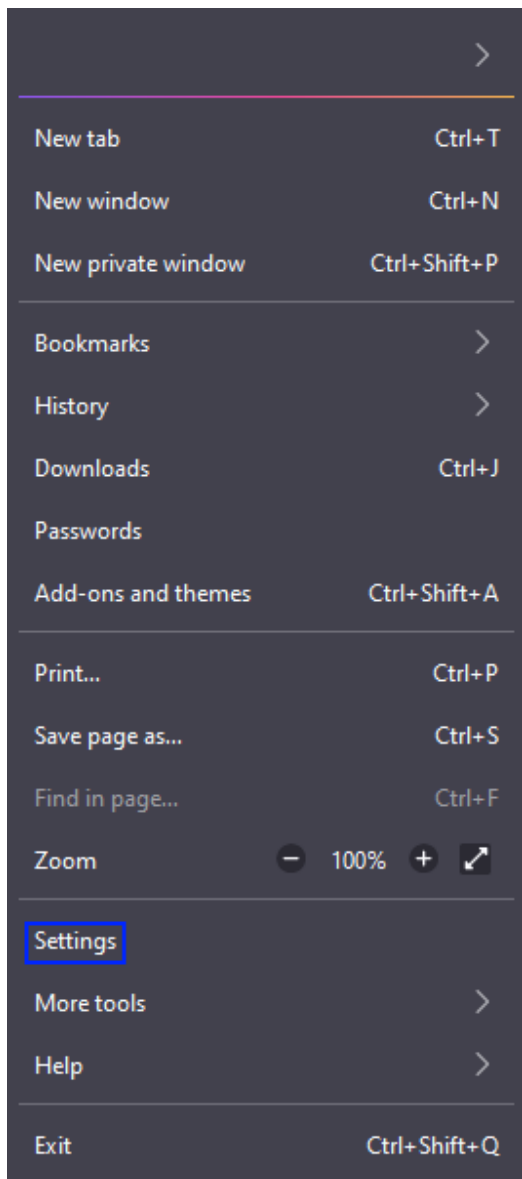Open the file with the **SQLite** browser and explore your cookies.

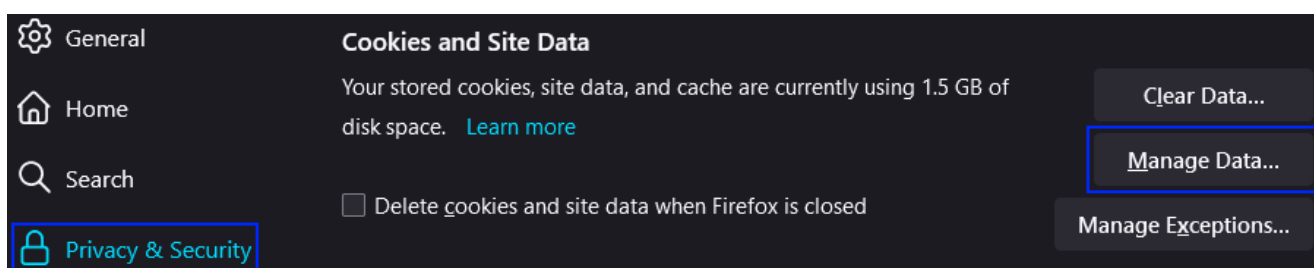### 2. Control Your Cookies (Mozilla)

Use Mozilla Browser to explore and control your cookies.

Open the hamburger menu button and select **Settings:**

Select **Privacy & Security** and click on the **[Manage Data]** button:

Type **softuni** in the **Search** bar and browse the cookies from the selected website.

You can also delete a particular cookie or all of the cookies.

Manage Cookies and Site Data

The following web sites store cookies and site data on your computer. Firefox keeps data from web sites with persistent storage until you delete it, and deletes data from web sites with non-persistent storage as space is needed.

🔍 softuni

Site	Cookies	Storage ▼	Last Used
softuni.bg	32	354 kB	3 hours ago
softuni.org	17	216 kB	yesterday

Remove Selected   Remove All Shown

Save Changes   Cancel

3. Control Your Cookies (Chrome)

Use Chrome Browser to explore and control your cookies.

Open the Kebab menu buttonand select `Settings`:
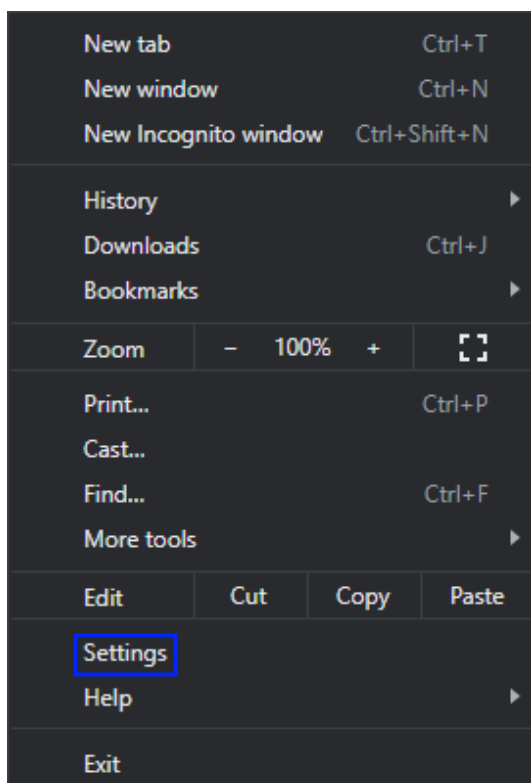
Go to `Privacy and security` and select `Cookies and other site data`:

Follow us:
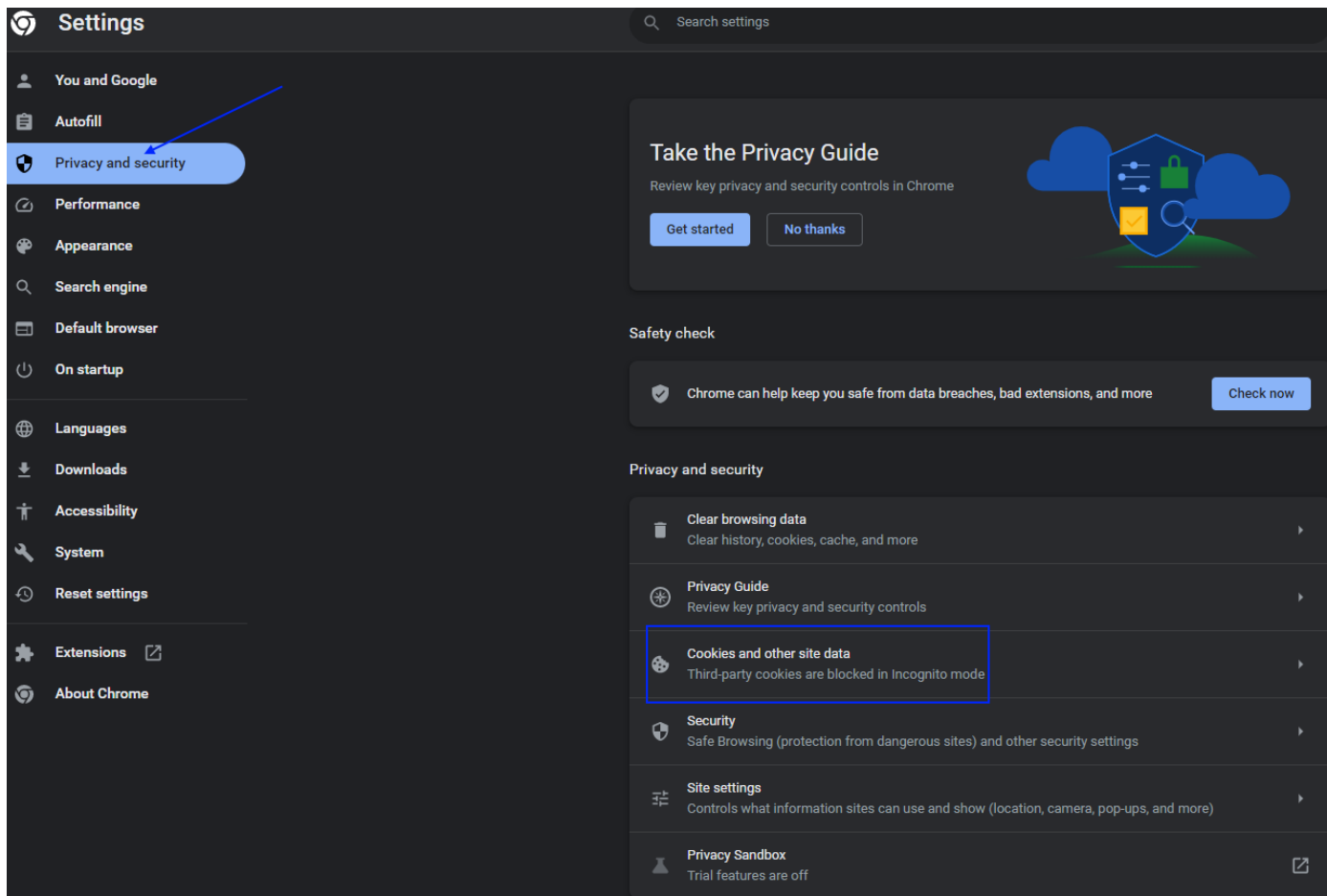
On the next window, click on **See all site data and permissions**. A new window opens with all of the websites that you have visited and their cookies, where you can browse the cookies.

You can also delete some of the cookies.

4. Change Cookie

Go to Amazon and look at the **menu language**.
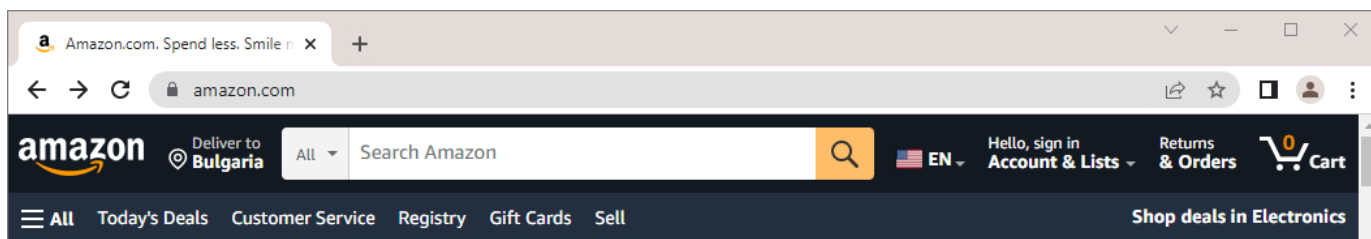
Open **DevTools** or press **F12 → [Application]** and search for the language cookie:

Follow us:

Change the cookie value to "**de_DE**" in order to change the language of the app:

Refresh the site – it should be in **German** now:

5. Authentication with Cookies

Go to the [SoftUni Judge](https://) site and log in. Search for the **authentication** cookie in the **Cookies** menu:

Delete the **authentication cookie. Refresh** Judge's site – you should **not** be logged-in **anymore**. The authentication cookie should be missing, too.

You can log in again – a new cookie should appear.

6. Session

Go to the GitHub site. Examine the **session cookie**, which holds the **current session id**.

Name	Value
_gh_sess	w5atA%2Fh3rM1NiGKjuYxMC3AR%2...

Close the browser and visit the site again. Notice that now there is a new session with a **different ID**.

Name ▲	Value
_gh_sess	t5N7PG%2FUtdtAAxICGWlsn4Ihllx05...

II. Asynchronous Processing

7. Even Numbers Thread

Print **all even** numbers in a given **range**. Printing should be executed on a **separate thread**. After all numbers are printed print "**Thread finished work**".

Example

Input	Output
1 10	2 4 6 8 10 Thread finished work

Hint

```
// TODO: Read the start and end number

Thread evens = new Thread(() => PrintEvenNumbers(start, end));
evens.Start();
evens.Join();
Console.WriteLine("Thread finished work");

// TODO: Implement the method PrintEvenNumbers
```

8. Sum Evens in Range

Sum all even numbers in given range [**1** to **1000**]. Read commands and print the result only on command "**show**".

Solution

```csharp
public static void Main()
{
    while (true)
    {
        var command = Console.ReadLine();
        if (command == "show")
        {
            var result = SumAsync();
            Console.WriteLine(result);
        }
    }
}

1 reference
private static long SumAsync()
{
    return Task.Run(() =>
    {
        long sum = 0;
        for (int i = 1; i < 10000; i++)
        {
            if (i % 2 == 0)
            {
                sum += i;
            }
        }
        return sum;
    }).Result;
}
```

9. Sum Evens in Background

This problem is similar to the previous one. You have to sum all even numbers in given range (**1** to **1000000000**), but this time leave the console interface unblocked while calculating the sum. Read commands and print the result only on command "**show**". Stop calculating on command "**exit**".

Solution

```csharp
public static void Main()
{
    long sum = 0;

    var task = Task.Run(() =>
    {
        for (long i = 0; i < 1000000000; i++)
        {
            if (i % 2 == 0)
            {
                sum += i;
            }
        }
    });

    while (true)
    {
        var line = Console.ReadLine();
        if (line == "exit")
        {
            return;
        }
        else if (line == "show")
        {
            Console.WriteLine(sum);
        }
    }
}
```

10. Chronometer

The **Chronometer** is one of the easiest examples of an **asynchronous processes**. Let's implement a simple Chronometer.

Create an **interface IChronometer** like this:

```csharp
public interface IChronometer
{
    0 references
    string GetTime { get; }

    0 references
    List<string> Laps { get; }

    0 references
    void Start();

    0 references
    void Stop();

    0 references
    string Lap();

    0 references
    void Reset();
}
```
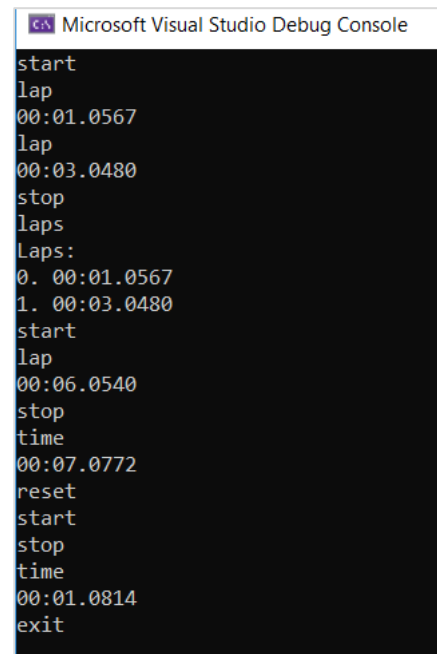
... and implement a class **Chronometer**, that implements it.

Implement a program which provides a **Chronometer functionality**, that responds to several commands from the user input:

- **start** – starts counting time in milliseconds, seconds and minutes
- **stop** – stops the process of counting time, but the counted time remains
- **lap** – creates a lap at the current time
- **laps** – returns all of the currently recorded laps
- **time** – returns the currently recorded time
- **reset** – stops the Chronometer, resets the currently recorded time and deletes all of the currently recoded laps
- **exit** – stops and exits the program

Here is an example screenshot of the functionality:

The time is outputted in the following format: "**{minutes}:{seconds}:{milliseconds}**", each of them should be **padded** with **zeros**.

Upon **making** a **lap** you should print the **time** at which it was made.

Requesting **all laps** should print them in the following format:

```
Laps:
0. {lap1}
1. {lap2}
...
```

In case there are no laps, you should print "**Laps: no laps**".

Hints

Let's start implementing our **asynchronous chronometer**. First, we need to create the **Chronometer class**, which implements the **IChronometer interface**:

```
public class Chronometer : IChronometer
```

Use the **Stopwatch C# class**, which provides a **set of methods and properties** that you can use to accurately measure elapsed **time**. Create a **field** for the stopwatch. Also, create a **collection for the laps**. Initialize the **fields** in the **constructor** like this:

```
public class Chronometer : IChronometer
{
    private Stopwatch _stopWatch;
    private List<string> _laps;

    1 reference
    public Chronometer()
    {
        _stopWatch = new Stopwatch();
        _laps = new List<string>();
    }
```

We have the **GetTime property**, which should return the **currently recorded time** since the start of the chronometer counter. Use the **Elapsed property** of the **Stopwatch class** to **get the total elapsed time**. This property returns the time as a **TimeSpan**, so you should convert it to **string** in the **correct format**. Do it like this:

```csharp
public string GetTime => _stopWatch.Elapsed.ToString(@"mm\:ss\.ffff");
```

The other property we have is the **Laps property**. It should just **return the current laps collection**:

```csharp
public List<string> Laps => _laps;
```

Next, we should **implement** the **Start()** and **Stop()** methods of the **Chronometer class**. The **Stopwatch class** has its own **methods for starting and stopping** – use them as shown below:

```csharp
public void Start()
{
    _stopWatch.Start();
}

public void Stop()
{
    _stopWatch.Stop();
}
```

The **Lap() method** returns the **current elapsed time** as a **string** and **adds it to a collection of laps**. It uses the **GetTime property**:

```csharp
public string Lap()
{
    string result = GetTime;
    _laps.Add(result);
    return result;
}
```

Finally, we have the **Reset() method**, which should **invoke the Reset() method** of the **Stopwatch class** and **clear the laps collection**. Do it like this:

```csharp
public void Reset()
{
    _stopWatch.Reset();
    _laps.Clear();
}
```

As we already have the **Chronometer class** let's use it and invoke its methods depending on **commands** from the console. **Instantiate the chronometer** in the **Main() method** of a **class** called **StartUp**:

```csharp
public class Startup
{
    0 references
    static void Main()
    {
        var chronometer = new Chronometer();
```

Then, we will **read a command** from the console, until the "**exit**" **command**.

```
        string line;

        while ((line = Console.ReadLine()) != "exit")
        {
```

In the **while loop**, work with the chronometer depending on the **read command**. Don't forget that the **Start()** command of the **Chronometer class** should be **run as a task to be asynchronous**. After the **while loop** you should **stop the chronometer**. Complete the **Startup class** like this:

```
            if (line == "start")
            {
                Task.Run(() =>
                {
                    chronometer.Start();
                });
            }
            else if (line == "stop")
            {
                chronometer.Stop();
            }
            else if (line == "lap")
            {
                Console.WriteLine(chronometer.Lap());
            }
            else if (line == "laps")
            {

                if (chronometer.Laps.Count == 0)
                {
                    Console.WriteLine("Laps: no laps");
                    continue;
                }
                Console.WriteLine("Laps: ");
                for (int i = 0; i < chronometer.Laps.Count; i++)
                {
                    Console.WriteLine($"{i} {chronometer.Laps[i]}");
                }
            }
            else if (line == "reset")
            {
                chronometer.Reset();
            }
            else if (line == "time")
            {
                Console.WriteLine(chronometer.GetTime());
            }
        }
        chronometer.Stop();
    }
}
```