State Management

- HTTP is stateless
- In order to associate a request to any other request, you need a way to store user data between HTTP requests
- Cookies are used to transport data between the client and the server
- Sessions allow you to store information associated with the client on the server

Cookies	Session
Stored on the client side	Stored on the server side
Can only store strings	Can store objects
Can be set to a long lifespan	When users close their browser, they also lose the session

Cookies

- Cookies are simple, small files/data that are stored on the client side
- Every time the user loads the website back, this cookie is sent with the request
- This helps us keep track of the user's actions

cookie-parser

- To use cookies with Express, you can use the cookie-parser middleware
- To install it, type the following command:

```
C:\NodeJS\cookies-demo>npm install cookie-parser
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN cookies-demo@1.0.0 No description
npm WARN cookies-demo@1.0.0 No repository field.

+ cookie-parser@1.4.3
added 3 packages in 1.227s
```

- The cookie-parser is used the same way as other middlewares:

```
const express = require('express');
const cookieParser = require('cookie-parser');

const app = express();
app.use(cookieParser());
```

Setting New Cookies

- To set a new cookie, use **res.cookie(name, value [, options])**
- The value parameter may be a string or object converted to JSON
- Example:

```
res.cookie('name', 'Roi');
```

- You can pass an object as the value parameter - it is then serialized as JSON and parsed by bodyParser() when received in the request

```
res.cookie('cart', { items: [1,2,3] });
```

Cookies with Expiration Time

- The options object allows you to set expiration time to the cookie:

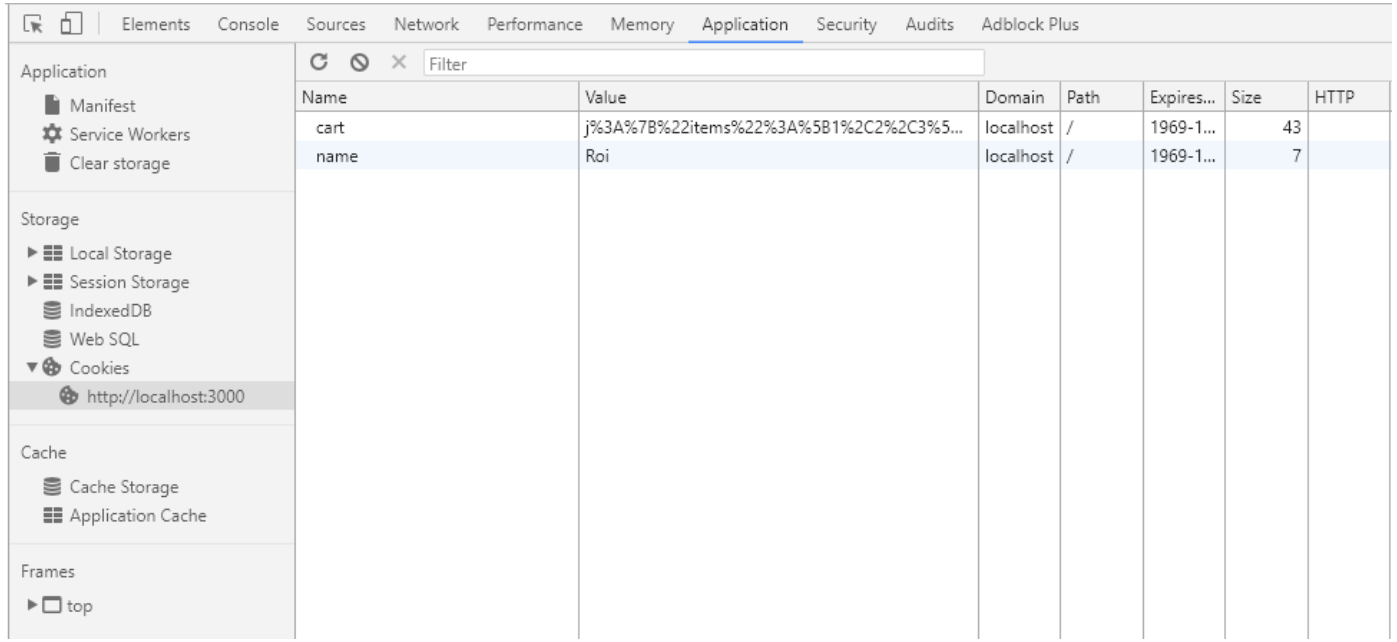
```
// Expires after 600000 ms (10 min) from the time it is set
res.cookie('rememberme', 1, {expire: Date.now() + 600000});
```

- If expiration time is not specified or set to 0, then it creates a session cookie
  - i.e., a cookie that is erased when the user closes the browser
- Another way to set expiration time is using the maxAge property
  - Using this property, we can specify expiration time which is relative to the current time (in milliseconds) instead of absolute time
  - The following is equivalent to the example above:

```
res.cookie('rememberme', 1, {maxAge: 600000});
```

Inspecting Cookies in Chrome Developer Tools

- In the Google Chrome developer tools, you can view the cookies sent to the browser under the Application tab:



Reading Cookies

- The browser sends back the cookies every time it queries the server
- cookie-parser parses the Cookie header and populates req.cookies with an object keyed by the cookie names
- If the request contains no cookies, it defaults to {}

```
app.get('/show_cookies', (req, res) => {
    res.write('name: ' + req.cookies.name + '<br/>');
    res.write('remember me: ' + req.cookies.rememberme);
    res.end();
});
```

- To delete a cookie, use the clearCookie() function

```
res.cookie('name', 'Roi');
res.clearCookie('name');
```

Signed Cookies

- You can sign your cookies, so it can be detected if the client modified the cookie
- When the cookie gets read, it recalculates the signature and makes sure that it matches the signature attached to it
- If it does not match, then it will give an error
- To create a signed cookie you would use

```
res.cookie('name', 'value', {signed: true})
```

- And to access a signed cookie use the signedCookies object of req:

```
req.signedCookies['name']
```

Exercise (9)

- Create a form that will enable the user to choose his favorite color
- After choosing a color, the background color of the page should change to that color
- The next time the user visits the page, his last chosen color should be displayed as the background color of the page