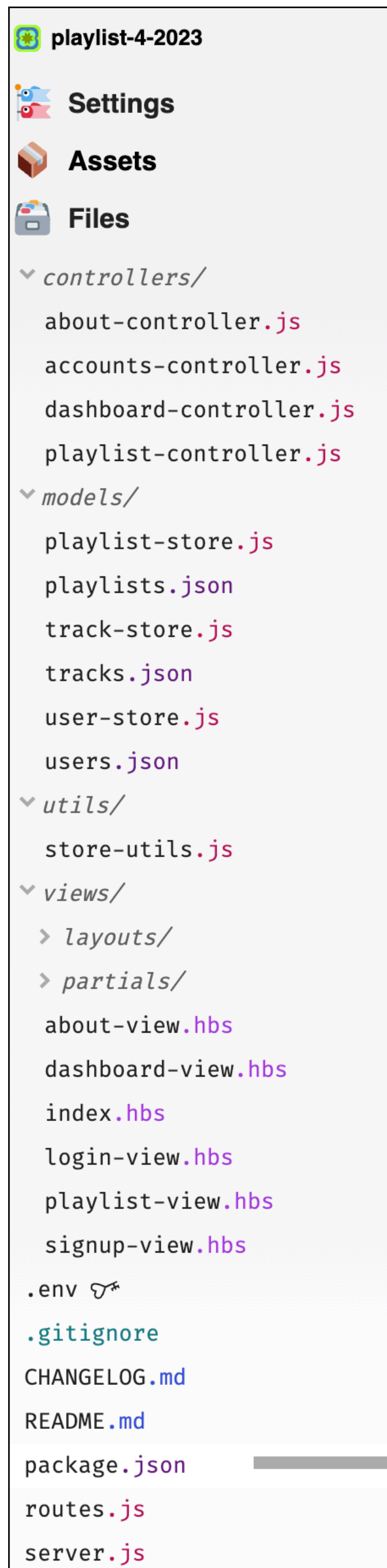


Creating Sessions

Creating Sessions

The diagram illustrates the relationship between cookies and session cookies. At the top, a horizontal flow shows 'client side' leading to a box labeled 'Cookies', which then leads to 'server side'. To the left of 'client side' is a box labeled 'ordinary cookies', and to the right of 'server side' is a box labeled 'Identified by client'. Below this, a circular diagram for 'Session Cookies' lists several characteristics: 'Used even when cookies disabled', 'Force explicit login', 'Distinguishes Guests', 'Deleted when browser is closed', and 'Can only be read by site that created it'. A curved arrow at the top of the circle states 'CMS uses session cookies for extra security'.

The API to create, access and destroy sessions.



```
{
  "name": "playlist",
  "version": "0.4.0",
  "description": "Simple Playlist Maker",
  "main": "server.js",
  "type": "module",
  "scripts": {
    "start": "node server.js"
  },
  "dependencies": {
    "body-parser": "^1.20.2",
    "cookie-parser": "^1.4.6",
    "express": "^4.18.2",
    "express-fileupload": "^1.4.0",
    "express-handlebars": "^6.0.7",
    "fs-extra": "^11.1.0",
    "lowdb": "^5.1.0",
    "uuid": "^9.0.0"
  },
  "repository": {
    "url": "https://github.com/wit-hdip-comp-sci-2023/playlist-4"
  },
  "license": "MIT",
  "keywords": [
    "node",
    "glitch",
    "express"
  ],
  "engines": {
    "node": "16.x.x"
  },
  "prettier": {
    "singleQuote": false,
    "printWidth": 120
  },
  "devDependencies": {
    "prettier": "^2.8.4"
  }
}
```

package.json

describes the
characteristics
+ dependencies
of the
application

dependencies

```
{
  ...
  "dependencies": {
    "body-parser": "^1.20.2",
    "cookie-parser": "^1.4.6",
    "express": "^4.18.2",
    "express-fileupload": "^1.4.0",
    "express-handlebars": "^6.0.7",
    "fs-extra": "^11.1.0",
    "lowdb": "^5.1.0",
    "uuid": "^9.0.0"
  },
  ...
}
```

Enumerates the various
modules our application
requires

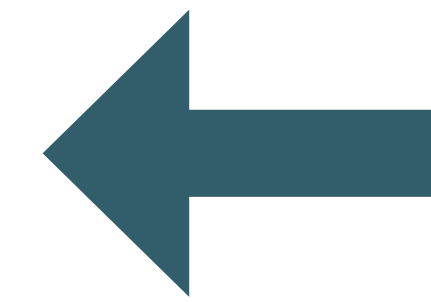
dependencies

```
{  
  ...  
  "dependencies": {  
    "body-parser": "^1.20.2",  
    "cookie-parser": "^1.4.6",  
    "express": "^4.18.2",  
    "express-fileupload": "^1.4.0",  
    "express-handlebars": "^6.0.7",  
    "fs-extra": "^11.1.0",  
    "lowdb": "^5.1.0",  
    "uuid": "^9.0.0"  
  },  
  ...  
}
```

Express
framework

lowdb
database

uuid ID
generation



Each of these an
independent
library, with its
own
documentation,
examples and
community of
users

Express 4.18.1

Express Framework

Fast, unopinionated, minimalist
web framework for [Node.js](#)

```
$ npm install express --save
```

 **Express 5.0 beta documentation is now available.**

The beta [API documentation](#) is a work in progress. For information on what's in the release, see the Express [release history](#).

Web Applications

Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.

APIs

With a myriad of HTTP utility methods and middleware at your disposal, creating a robust API is quick and easy.

Performance

Express provides a thin layer of fundamental web application features, without obscuring Node.js features that you know and love.

Frameworks


Many [popular frameworks](#) are based on Express.

<https://expressjs.com/>

lowdb 

6.0.1 • Public • Published 20 days ago

Lowdb Database

 Readme

 Code Beta

 1 Dependency

lowdb downloads 2.1M/month Node.js CI passing

Simple to use local JSON database. Use native JavaScript API to query. Written in TypeScript. 🦉

```
// Edit db.json content using plain JavaScript
db.data.posts.push({ id: 1, title: 'lowdb is awesome' })


// Save to file
db.write()
```

```
// db.json
{
  "posts": [
    { "id": 1, "title": "lowdb is awesome" }
  ]
}
```

<https://github.com/typicode/lowdb#readme> ⁶



uuid 

9.0.0 • Public • Published 8 months ago

 Readme

 Code Beta

 0 Dependencies

uuid  CI passing  Browser passing

For the creation of **RFC4122** UUIDs

- **Complete** - Support for RFC4122 version 1, 3, 4, and 5 UUIDs
- **Cross-platform** - Support for ...
 - CommonJS, **ECMAScript Modules** and **CDN builds**
 - Node 12, 14, 16, 18
 - Chrome, Safari, Firefox, Edge browsers
 - Webpack and rollup.js module bundlers
 - **React Native / Expo**
- **Secure** - Cryptographically-strong random values
- **Small** - Zero-dependency, small footprint, plays nice with "tree shaking" packagers
- **CLI** - Includes the **uuid** **command line** utility

Upgrading from **uuid@3** ? Your code is probably okay, but check out **Upgrading From uuid@3** for details.

<https://github.com/uuidjs/uuid#readme>

Sessions Support - Cookie Parser

Express

[Home](#) [Getting started](#) [Guide](#) [API reference](#) [Advanced topics](#) [Resources](#)

- [body-parser](#)
- [compression](#)
- [connect-redis](#)
- [cookie-parser](#)
- [cookie-session](#)
- [cors](#)
- [csrf](#)
- [errorhandler](#)
- [method-override](#)
- [morgan](#)
- [multer](#)
- [response-time](#)
- [serve-favicon](#)
- [serve-index](#)
- [serve-static](#)

cookie-parser

npm **v1.4.3** downloads **2M/month** node **>= 0.8.0** build **passing** coverage **100%**

Parse `Cookie` header and populate `req.cookies` with an object keyed by the cookie names. Optionally you may enable signed cookie support by passing a `secret` string, which assigns `req.secret` so it may be used by other middleware.

Installation

```
$ npm install cookie-parser
```

API

```
var express = require('express')
var cookieParser = require('cookie-parser')

var app = express()
app.use(cookieParser())
```

cookieParser(secret, options)


- `secret` a string or array used for signing cookies. This is optional and if not specified, will not parse signed cookies. If a string is provided, this is used as the secret. If an array is provided, an attempt will be made to unsign the cookie with each secret in order.
- `options` an object that is passed to `cookie.parse` as the second option. See [cookie](#) for more information.
 - `decode` a function to decode the value of the cookie

```
{
  ...
  "dependencies": {
    "body-parser": "^1.20.2",
    "cookie-parser": "^1.4.6",
    "express": "^4.18.2",
    "express-fileupload": "^1.4.0",
    "express-handlebars": "^6.0.7",
    "fs-extra": "^11.1.0",
    "lowdb": "^5.1.0",
    "uuid": "^9.0.0"
  },
  ...
}
```

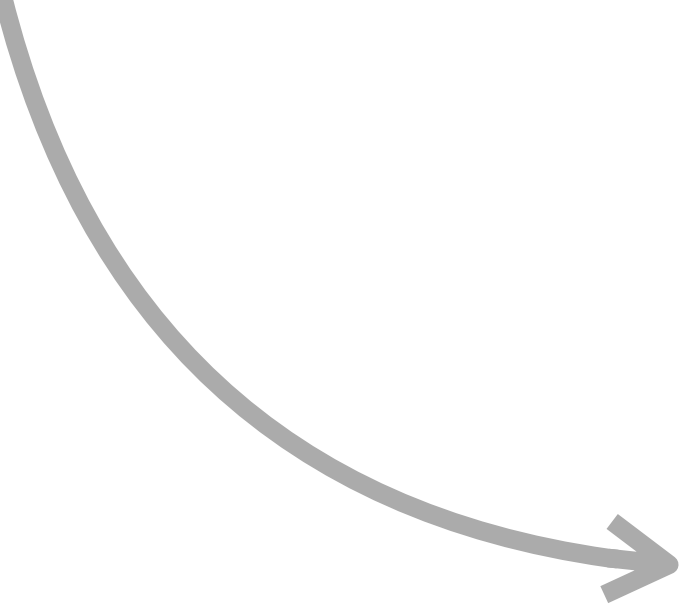

Routes

routes.js

```
router.post("/register", accountsController.register);  
router.post("/authenticate", accountsController.authenticate);
```



Register a new
User with the
application



Check to see if a
given email/
password is
known to the
application

Routes

routes.js

```
router.post("/register", accountsController.register);  
router.post("/authenticate", accountsController.authenticate);
```

Create a new user
data store object

Check dates store
for given user
-> Create Session
Object if user
found

```

import { v4 } from "uuid";
import { initStore } from "../utils/store-utils.js";

const db = initStore("users");

export const userStore = {
  async getAllUsers() {
    await db.read();
    return db.data.users;
  },

  async addUser(user) {
    await db.read();
    user._id = v4();
    db.data.users.push(user);
    await db.write();
    return user;
  },

  async getUserById(id) {
    await db.read();
    return db.data.users.find((user) => user._id === id);
  },

  async getUserByEmail(email) {
    await db.read();
    return db.data.users.find((user) => user.email === email);
  },

  async deleteUserById(id) {
    await db.read();
    const index = db.data.users.findIndex((user) => user._id === id);
    db.data.users.splice(index, 1);
    await db.write();
  },

  async deleteAll() {
    db.data.users = [];
    await db.write();
  },
};

```

userStore


Manage database of users, supporting:

- Create
- Read
- Update
- Delete

C.R.U.D.

users.json

```
{
  "file": "./models/users.json",
  "users": [
    {
      "firstName": "Homer",
      "lastName": "Simpson",
      "email": "homer@simpson.com",
      "password": "secret",
      "_id": "538a527f-e42d-47db-b3fe-cdb2da9548c0"
    },
    {
      "firstName": "Marge",
      "lastName": "Simpson",
      "email": "marge@simpson.com",
      "password": "secret",
      "_id": "288f6deb-90d9-49e1-b855-9222dfe19a85"
    }
  ]
}
```


 Playlist 4.0

Log inSign up

Sign up

Name

Email

Password

Submit

accountsController.register()

Create a new user
object based on
form data
+ Add to user-
store

```
...  
  
  async register(request, response) {  
    const user = request.body;  
    await userStore.addUser(user);  
    console.log(`registering ${user.email}`);  
    response.redirect("/");  
  },  
  
  ...
```



Log in

Email

Password

Submit

accountsController.authenticate()

Check if user exists

If user known, create
cookie called
'playlist' containing
users email, then
switch to dashboard

Otherwise, ask user
to try to log in again

```
...
  async authenticate(request, response) {

    const user = await userStore.getUserByEmail(request.body.email);

    if (user) {

      response.cookie("playlist", user.email);
      console.log(`logging in ${user.email}`);
      response.redirect("/dashboard");

    } else {
      response.redirect("/login");
    }
  },
  ...
```

accountsController.getLoggedInUser method

Utility method to see if session exists
and which user 'owns' the session.

```
...  
  async getLoggedInUser(request) {  
    const userEmail = request.cookies.playlist;  
    return await userStore.getUserByEmail(userEmail);  
  },  
  ...
```

Return a valid user object if session exists
Otherwise return null

dashboardController.index()

Discover
which user
is currently
logged in

```
export const dashboardController = {  
  async index(request, response) {  
    const loggedInUser = await accountsController.getLoggedInUser(request);  
    const viewData = {  
      title: "Playlist Dashboard",  
      playlists: await playlistStore.getPlaylistsByUserId(loggedInUser._id),  
    };  
    console.log("dashboard rendering");  
    response.render("dashboard-view", viewData);  
  },  
  ...  
}
```

Retrieve only those
playlists associated with
the logged in user

dashboardController.addPlaylist()

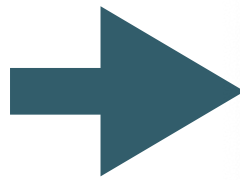
Discover
which user
is currently
logged in

```
export const dashboardController = {  
  ...  
  async addPlaylist(request, response) {  
    const loggedInUser = await accountsController.getLoggedInUser(request);  
    const newPlayList = {  
      title: request.body.title,  
      userid: loggedInUser._id,  
    };  
    console.log(`adding playlist ${newPlayList.title}`);  
    await playlistStore.addPlaylist(newPlayList);  
    response.redirect("/dashboard");  
  },  
  ...  
}
```

In the new playlist,
include the id of the
currently logged in user

Browser Cookies

Session
Cookies can
be inspected
in Browser



Playlist 4.0

Dashboard

About

Logout

Play List One

Play List Two

Application

Manifest

Service Workers

Storage

Storage

Local Storage

Session Storage

IndexedDB

Web SQL

Cookies

http://localhost:4000

Private State Tokens

Interest Groups

Shared Storage

Cache Storage

Background Services

Back/forward cache

Background Fetch

Background Sync

Notifications

Payment Handler

Periodic Background Sync

Filter

Name

Value

Domain

Path

Expire...

Size

HttpO...

Secure

Same...

Partiti...

Priority

playlist

homer%40simpson.com

localh...

/

Session

27

Medium

Cookie Value

☐ Show URL-decoded

homer%40simpson.com

18

Password Check?

```
...
  async authenticate(request, response) {

    const user = await userStore.getUserByEmail(request.body.email);

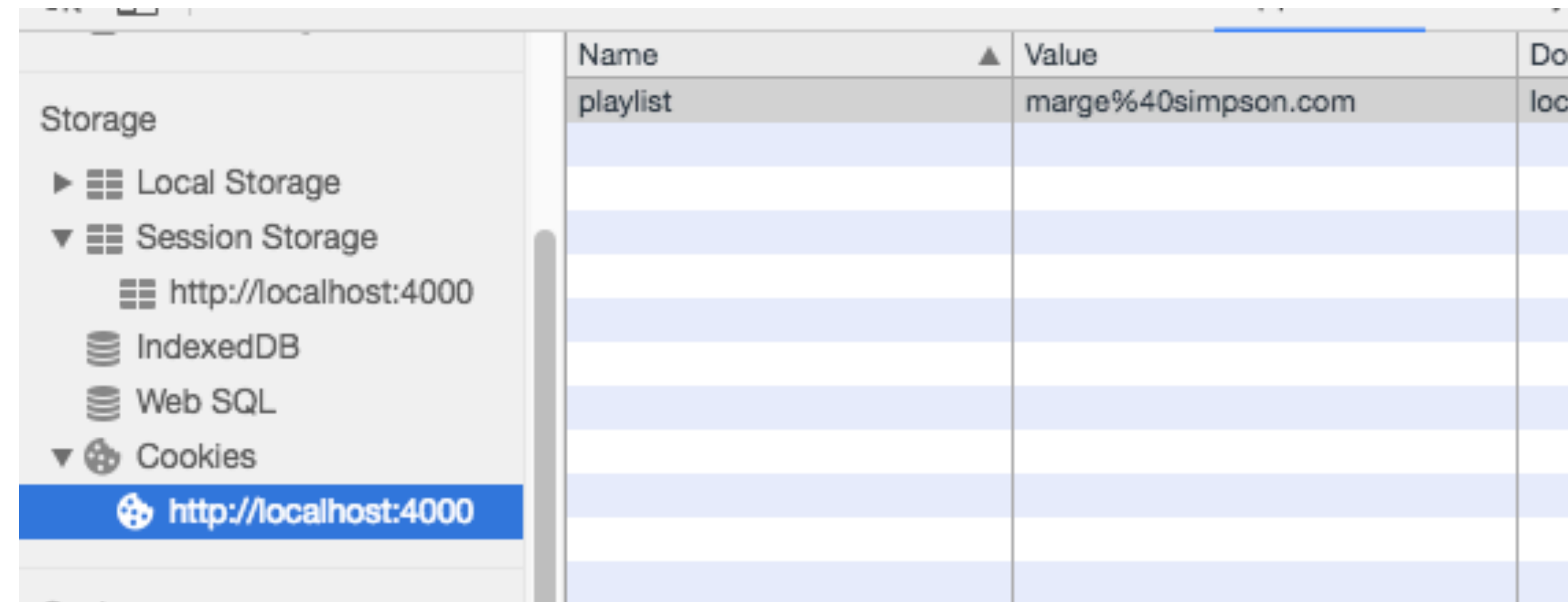
    if (user) {

      response.cookie("playlist", user.email);
      console.log(`logging in ${user.email}`);
      response.redirect("/dashboard");

    } else {
      response.redirect("/login");
    }
  },
...

```

logout

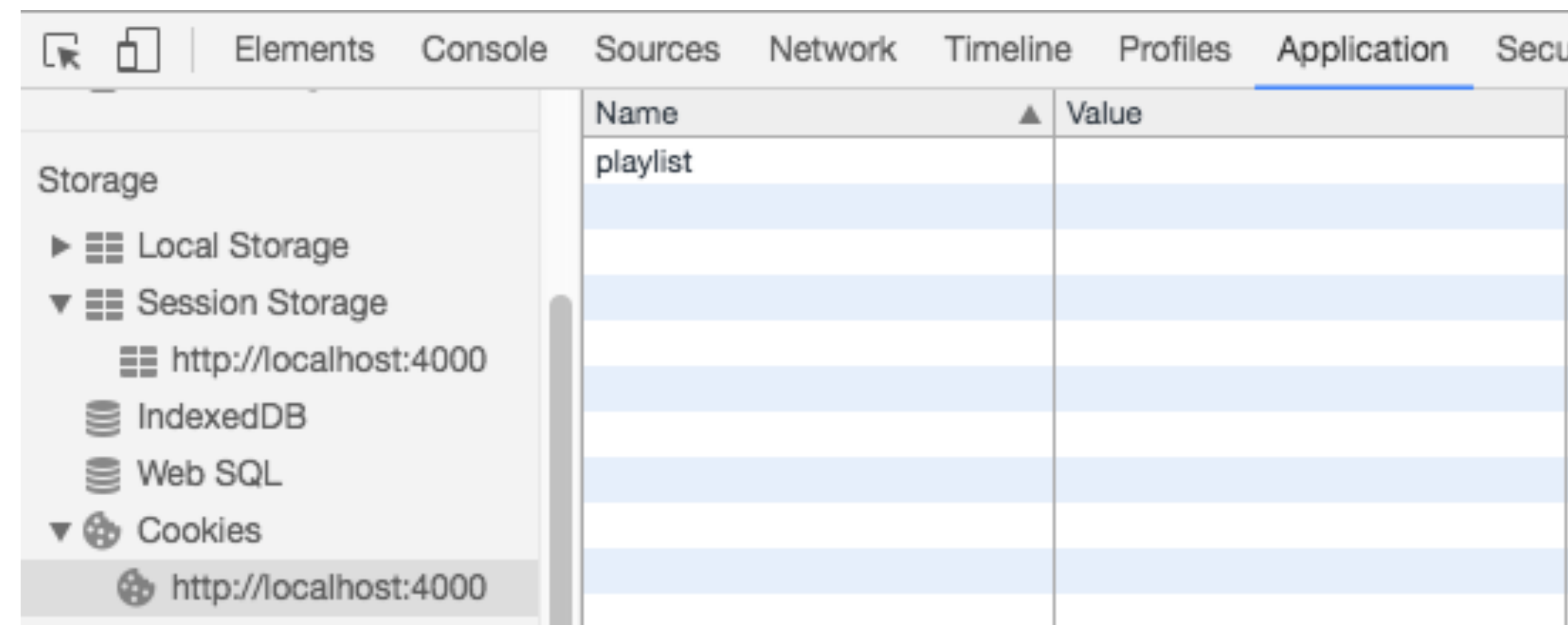


The screenshot shows the Chrome DevTools Storage tab. On the left, the 'Storage' sidebar is expanded to 'Cookies' for the domain 'http://localhost:4000'. The main pane displays a table of cookies:

Name	Value	Domain
playlist	marge%40simpson.com	localhost:4000

```
logout(request, response) {  
  response.cookie("playlist", "");  
  response.redirect("/");  
},
```

Clear the cookie



The screenshot shows the Chrome DevTools Storage tab after the logout function has been executed. The 'playlist' cookie is no longer present in the table:

Name	Value	Domain

Creating Sessions

Creating Sessions

The diagram illustrates the difference between Cookies and Session Cookies. Cookies are stored on the client side and are identified by the client. Session Cookies are stored on the server side and are used even when cookies are disabled, force explicit login, distinguish guests, are deleted when browsing, and can only be read by the site that created them.

The API to create, access and destroy sessions.