

# Creating Sessions

# package.json

back-end +

 .env

**.gitignore** ▼

.jscsrc

controllers/about.js

controllers/accounts.js

controllers/dashboard.js

controllers/playlist.js

controllers/start.js

models/json-store.js

models/playlist-store.js

models/playlist-store.json

models/user-store.js

models/user-store.json

package.json →


README.md

routes.js

**server.js** ▼

utils/logger.js

front-end +

 assets

views/about.hbs

views/dashboard.hbs

views/index.hbs

views/layouts/main.hbs

views/login.hbs

views/partials/addplaylist.hbs

views/partials/addsong.hbs

views/partials/listplaylists.hbs

views/partials/listsongs.hbs

views/partials/menu.hbs

views/partials/welcomemenu.hbs

views/playlist.hbs

views/signup.hbs

views/start.hbs

```
{
  "name": "gomix-playlist-4",
  "version": "0.0.1",
  "description": "Playlist application",
  "main": "server.js",
  "scripts": {
    "start": "node server.js"
  },
  "dependencies": {
    "body-parser": "^1.15.2",
    "express": "^4.14.0",
    "express-handlebars": "^3.0.0",
    "lodash": "^4.17.3",
    "lowdb": "^0.14.0",
    "uuid": "^3.0.1",
    "winston": "^2.3.0"
  },
  "engines": {
    "node": "6.9.1"
  },
  "repository": {
    "url": "https://github.com/edeleastar/gomix-playlist-4.git"
  },
  "keywords": [
    "node",
    "gomix",
    "express"
  ]
}
```

describes the characteristics  
+ dependencies of the  
application

# dependencies

```
{  
  ...  
  "dependencies": {  
    "body-parser": "^1.15.2",  
    "express": "^4.14.0",  
    "express-handlebars": "^3.0.0",  
    "lodash": "^4.17.3",  
    "lowdb": "^0.14.0",  
    "uuid": "^3.0.1",  
    "winston": "^2.3.0"  
  },  
  ...  
}
```

Enumerates the various  
modules our application  
requires

# dependencies

```
{  
...  
  "dependencies": {
```

```
    "body-parser": "^1.15.2",  
    "express": "^4.14.0",  
    "express-handlebars": "^3.0.0",
```

*Express Framework*

```
    "lodash": "^4.17.3",  
    "lowdb": "^0.14.0",
```

*lowdb Database*

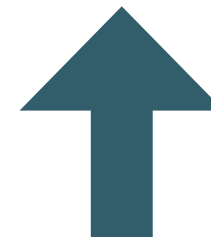
```
    "uuid": "^3.0.1",
```

*uuid ID generation library*

```
    "winston": "^2.3.0"
```

*winston logging*

```
  },  
...  
}
```



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

# Express Framework

Express

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

Express 4.15.2

Fast, unopinionated,  
minimalist web  
framework for Node.js

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



## 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 Database

**Lowdb**

npm package 0.16.0

build passing

A small local database powered by lodash API

```
const db = low('db.json')

// Set some defaults if your JSON file is empty
db.defaults({ posts: [], user: {} })
  .write()

// Add a post
db.get('posts')
  .push({ id: 1, title: 'lowdb is awesome'})
  .write()

// Set a user
db.set('user.name', 'typicode')
  .value()
```

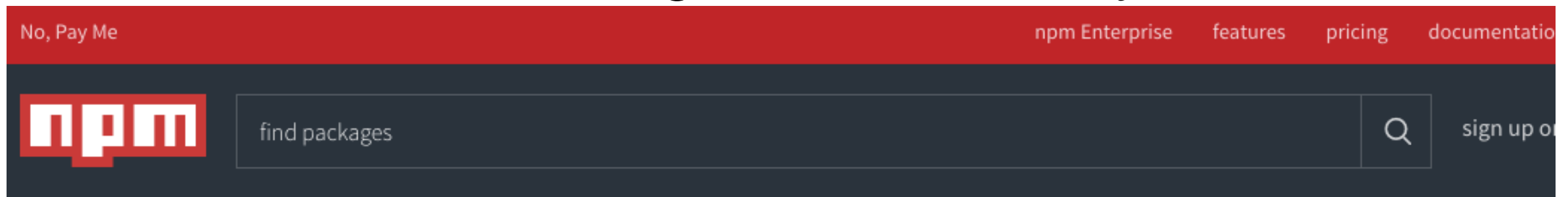
Data is saved to db.json

```
{
  "posts": [
    { "id": 1, "title": "lowdb is awesome" }
  ],
  "user": {
    "name": "typicode"
  }
}
```

<https://github.com/typicode/lowdb>

You can use any [lodash](#) function like `_.get` and `_.find` with shorthand syntax.

# uuid ID generation library



uuid public



Simple, fast generation of **RFC4122** UUIDs.

Features:

- Generate RFC4122 version 1 or version 4 UUIDs
- Runs in node.js and browsers
- Cryptographically strong random number generation on supporting platforms
- Small footprint (Want something smaller? [Check this out!](#))

## Quickstart - CommonJS (Recommended)

```
npm install uuid
```

```
// Generate a v1 UUID (time-based)
const uuidV1 = require('uuid/v1');
uuidV1(); // -> '6c84fb90-12c4-11e1-840d-7b25c5ee775a'

// Generate a v4 UUID (random)
const uuidV4 = require('uuid/v4');
uuidV4(); // -> '110ec58a-a0f2-4ac4-8393-c866d813b8d1'
```

Manage permissions for the whole team

Manage developer teams with varying permissions and multiple projects. [Learn more about Private Packages and Organizations...](#)

 npm install uuid

[how? learn more](#)

 [defunctzombie](#) published 3 month...

**3.0.1** is the latest of 11 releases

[github.com/kelektiv/node-uuid](https://github.com/kelektiv/node-uuid)

MIT 

Collaborators [list](#)

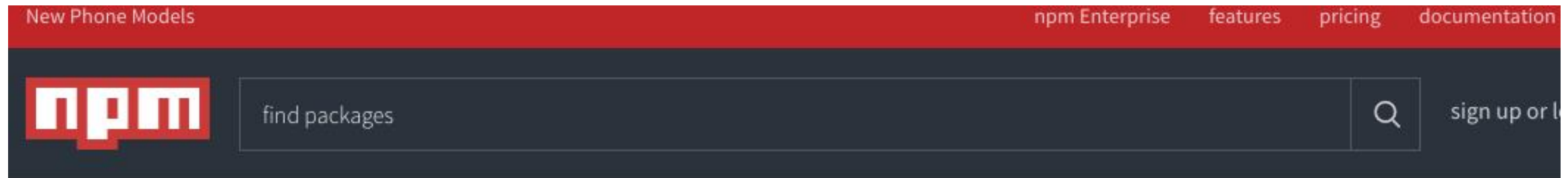


Stats

<https://www.npmjs.com/package/uuid>



# winston logging



winston

public

gitter join chat

npm v2.3.1 downloads 6M/month build passing dependencies out-of-date



npm install winston

6 dependencies version 2.3.1  
3,101 dependents updated 2 months ago  
5,909,418 downloads in the last month  
download rank: 361st of 380,500 packages

A multi-transport async logging library for node.js. "CHILL WINSTON! ... I put it in the logs."

## Motivation

Winston is designed to be a simple and universal logging library with support for multiple transports. A transport is essentially a storage device for your logs. Each instance of a winston logger can have multiple transports configured at different levels. For example, one may want error logs to be stored in a persistent remote location (like a database), but all logs output to the console or a local file.

There also seemed to be a lot of logging libraries out there that coupled their implementation of logging (i.e. how the logs are stored / indexed) to the API that they exposed to the programmer. This library aims to decouple those parts of the process to make it more flexible and extensible.

Manage permissions for the whole team

Manage developer teams with varying permissions and multiple projects. [Learn more about Private Packages and Organizations...](#)

npm install winston

[how? learn more](#)

indexzero published 2 months ago

2.3.1 is the latest of 42 releases

[github.com/winstonjs/winston](https://github.com/winstonjs/winston)

MIT

Collaborators [list](#)



<https://www.npmjs.com/package/winston>



# Sessions Support - Cookie Parser

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

## 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')

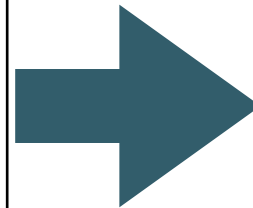
var app = express()
app.use(cookieParser())
```

#### `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

# Include Cookie Parser Module

```
{
  ...
  "dependencies": {
    "body-parser": "^1.15.2",
    "express": "^4.14.0",
    "express-handlebars": "^3.0.0",
    "lodash": "^4.17.3",
    "lowdb": "^0.14.0",
    "uuid": "^3.0.1",
    "winston": "^2.3.0"
  },
  ...
}
```



```
{
  ...
  "dependencies": {
    "body-parser": "^1.15.2",
    "express": "^4.14.0"
    "cookie-parser": "^1.4.3",
    "express-handlebars": "^3.0.0",
    "lodash": "^4.17.3",
    "lowdb": "^0.14.0",
    "uuid": "^3.0.1",
    "winston": "^2.3.0"
  },
  ...
}
```

New entry include  
cookie module in  
application

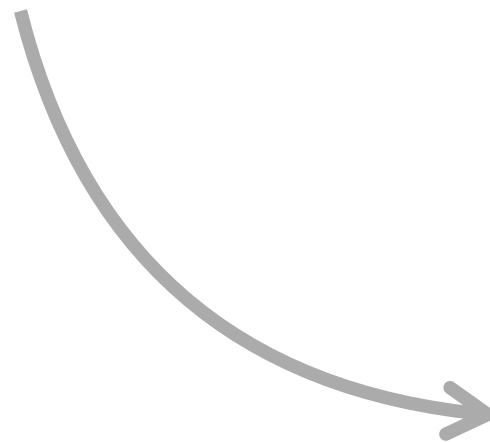
# Routes

Register a new  
User with the  
application

routes.js

```
router.post('/register', accounts.register);  
router.post('/authenticate', accounts.authenticate);
```


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



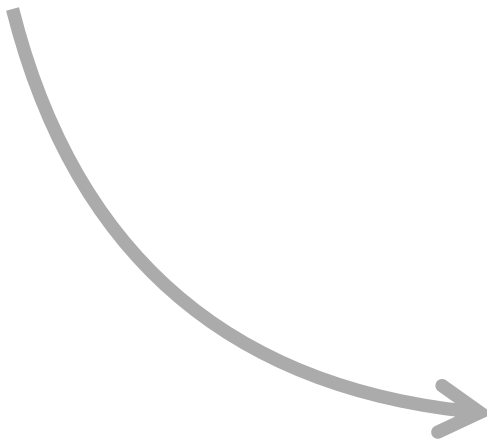
# Routes

routes.js

```
router.post('/register', accounts.register);  
router.post('/authenticate', accounts.authenticate);
```



Create a new user  
database object



Check Database  
for given user  
-> Create Session  
Object if user  
found

# userStore object

```
'use strict';

const _ = require('lodash');
const JsonStore = require('./json-store');

const userStore = {

  store: new JsonStore('./models/user-store.json', {users: []}),
  collection: 'users',

  getAllUsers() {
    return this.store.findAll(this.collection);
  },

  addUser(user) {
    this.store.add(this.collection, user);
  },

  getUserById(id) {
    return this.store.findOneBy(this.collection, { id: id });
  },

  getUserByEmail(email) {
    return this.store.findOneBy(this.collection, { email: email });
  },
}

module.exports = userStore;
```

Manage  
database of  
users,  
supporting:

- create
- getAll
- getById
- getByEmail

# user-store.json

```
{
  "users": [
    {
      "firstName": "homer",
      "lastName": "simpson",
      "email": "homer@simpson.com",
      "password": "secret",
      "id": "3ad52697-6d98-4d80-8273-084de55a86c0"
    },
    {
      "firstName": "marge",
      "lastName": "simpson",
      "email": "marge@simpson.com",
      "password": "secret",
      "id": "2b6f0989-7b7f-4a38-ad26-aa06b922d751"
    }
  ]
}
```



# account.register method

[Playlist 4](#)

SignupLogin

### Register

First Name

Homer

Last Name

Simpson

Email

homer@simpson.com

Password

.....

Submit

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

```
...
register(request, response) {
  const user = request.body;
  user.id = uuid();
  userstore.addUser(user);
  logger.info(`registering ${user.email}`);
  response.redirect('/');
},
...
```

# accounts.authenticate method

Playlist 4

Log-in

Email

homer@simpson.com

Password

.....

Login

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

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

  const user = userstore.getUserByEmail(request.body.email);

  if (user) {

    response.cookie('playlist', user.email);
    logger.info(`logging in ${user.email}`);
    response.redirect('/start');

  } else {

    response.redirect('/login');

  }

},
...
```

## accounts.getCurrentUser method

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

```
...  
getCurrentUser (request) {  
  const userEmail = request.cookies.playlist;  
  return userstore.getUserByEmail(userEmail);  
}  
...
```

Return a valid user object if session exists  
Otherwise return null

# dashboard: index

Discover  
which user  
is currently  
logged in

```
const dashboard = {  
  ...  
  index(request, response) {  
    logger.info('dashboard rendering');  
  
    const loggedInUser = accounts.getCurrentUser(request);  
  
    const viewData = {  
      title: 'Playlist Dashboard',  
      playlists: playlistStore.getUserPlaylists(loggedInUser.id),  
      ...  
    };  
  
    logger.info('about to render', playlistStore.getAllPlaylists());  
    response.render('dashboard', viewData);  
  },  
  ...  
}
```

Retrieve only those  
playlists associated with  
the logged in user

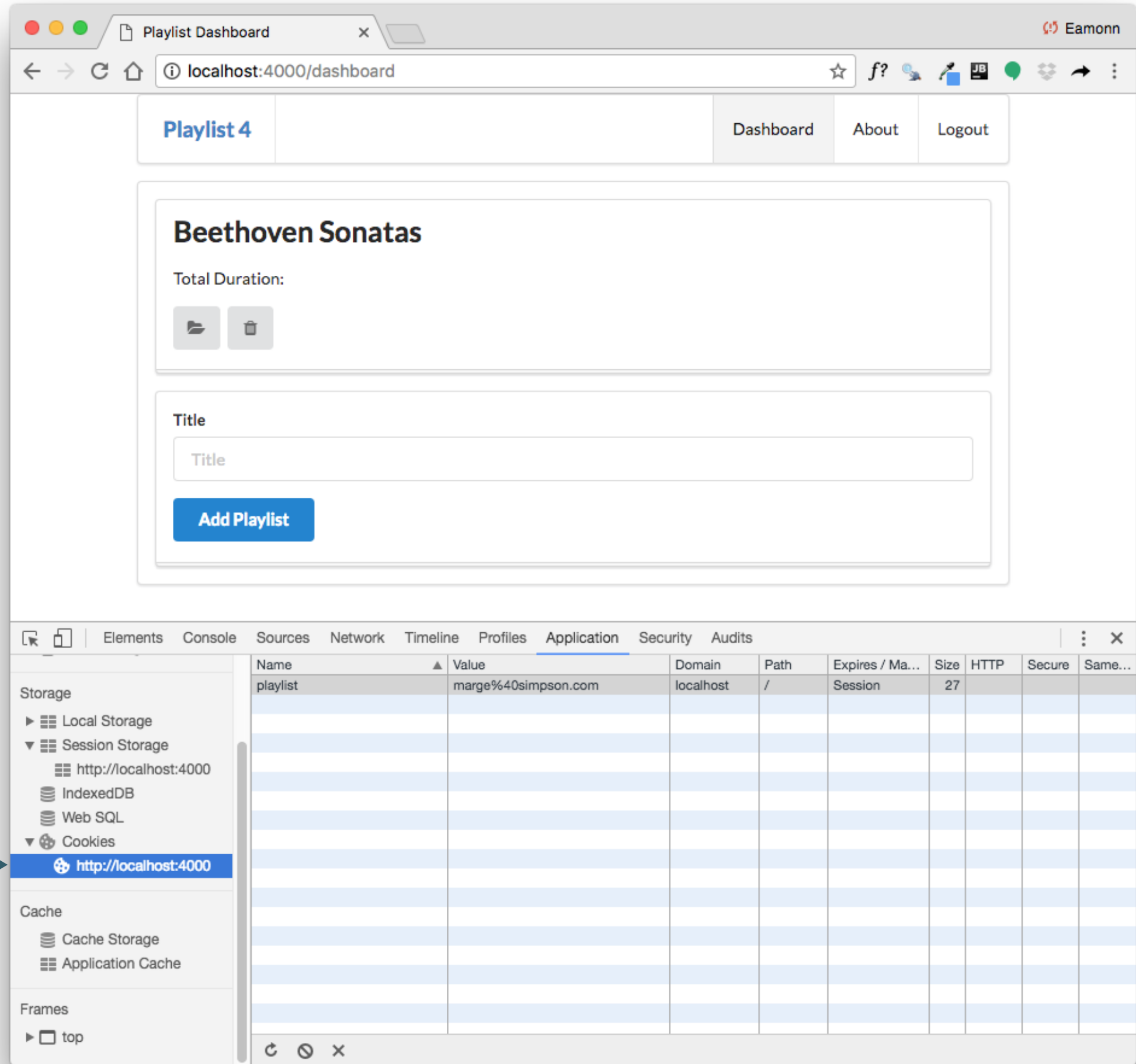
# dashboard: addPlaylist

Discover  
which user  
is currently  
logged in

```
const dashboard = {  
  ...  
  addPlaylist(request, response) {  
    const loggedInUser = accounts.getCurrentUser(request);  
  
    const newPlayList = {  
      id: uuid(),  
      userid: loggedInUser.id,  
      title: request.body.title,  
      songs: [],  
    };  
  
    logger.debug('Creating a new Playlist', newPlayList);  
    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



# Password Check?

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

  const user = userstore.getUserByEmail(request.body.email);
  if (user) {
    response.cookie('playlist', user.email);
    logger.info(`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' panel is expanded, showing 'Local Storage' selected. Under 'Local Storage', the URL 'http://localhost:4000' is listed. The main pane displays a table of stored items:

Name	Value	Domain
playlist	marge%40simpson.com	localhost

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

# Clear the cookie

The screenshot shows the Chrome DevTools interface with the 'Application' tab selected. In the left sidebar, under 'Storage', 'Local Storage' is expanded, showing a single entry for 'http://localhost:4000'. The main panel displays a table with two columns: 'Name' and 'Value'. The first row has 'playlist' in the 'Name' column and its corresponding value in the 'Value' column.

Name	Value
playlist	

