502070

WEB APPLICATION DEVELOPMENT USING NODEJS

LESSON 05 – Cookies & session

Content

- Cookie in Express
- Sessions in Express
- Using Sessions to Implement Flash Messages

Cookie in Express

- HTTP is a *stateless* protocol
- We need something to know who visiting our site.
- Cookie is some information which server need to save at browser.
- Cookie will be sent with requests

Cookie in Express

- *Cookies are not secret from the user*
- *The user can delete or disallow cookies*
- *Regular cookies can be tampered with* ➔ use signed cookies
- *Cookies can be used for attacks*
- *Users will notice if you abuse cookies*
- *Prefer sessions over cookies*

Signed cookies

- To make cookies secure, a *cookie secret* is necessary. The cookie secret is a string that's known to the server and used to encrypt secure cookies before they're sent to the client.
- Example: *credentials.js*

```
module.exports = {
    cookieSecret: 'your cookie secret goes here',
};
```

- make sure we don't accidentally add this file to our repository, add *creden- tials.js* to your *.gitignore* file.

Create cookies

- To import your credentials into your application, all you need to do is:

```
var credentials = require('./credentials.js');
```

- First, *npm install --save cookie-parser*, then:

```
app.use(require('cookie-parser')(credentials.cookieSecret));
```

- You can set a cookie or a signed cookie anywhere you have access to a response object:

```
res.cookie('monster', 'nom nom');
res.cookie('signed_monster', 'nom nom', { signed: true });
```

Read cookies

- To retrieve the value of a cookie (if any) sent from the client, just access the cookie or signedCookie properties of the request object:

```
var monster = req.cookies.monster;
var signedMonster = req.signedCookies.monster;
```

- To delete a cookie, use res.clearCookie:

```
res.clearCookie('monster');
```

Cookie options

- domain
- path
- maxAge
- secure
- httpOnly
- signed

Sessions in Express

- To implement sessions, *something* has to be stored on the client to know which session of which client.
  - Cookie
  - local storage
- There are two ways to implement sessions:
  - Cookie-based sessions (cookie-session middleware)
  - Store only a unique identifier in the cookie and everything else on the server

Memory Stores

- First, install express-session (*npm install --save express-session*); then, after linking in the cookie parser, link in express-session:

```
app.use(require('cookie-parser')(credentials.cookieSecret));
app.use(require('express-session')());
```

- The express-session middleware accepts a configuration object with the following options:
  - key: The name of the cookie that will store the unique session identifier. Defaults to “connect.sid”.
  - store: An instance of a session store. Defaults to an instance of **MemoryStore**
  - cookie: Cookie settings for the session cookie (path, domain, secure, etc.).

Using Sessions

- Once you've set up sessions, using them couldn't be simpler: just use properties of the request object's session variable:

```
req.session.userName = 'Anonymous';
var colorScheme = req.session.colorScheme || 'dark';
```

- To delete a session, you can use JavaScript's delete operator:

```
req.session.userName = null;              // this sets 'userName' to null,
                                          // but doesn't remove it

delete req.session.colorScheme;           // this removes 'colorScheme'
```

Using Sessions to Implement Flash Messages

- “Flash” messages are simply a way to provide feedback to users in a way that’s not disruptive to their navigation.
- The easiest way to implement flash messages is to use sessions

```
{{#if flash}}
        <div class="alert alert-dismissible alert-{{flash.type}}">
                <button type="button" class="close"
                        data-dismiss="alert" aria-hidden="true">&times;<button>
                <strong>{{flash.intro}}</strong> {{{flash.message}}}
        </div>
{{/if}}
```

Using Sessions to Implement Flash Messages

- Now let's add some middleware to add the flash object to the context if there's one in the session

```
app.use(function(req, res, next){
        // if there's a flash message, transfer
        // it to the context, then clear it
        res.locals.flash = req.session.flash;
        delete req.session.flash;
        next();
});
```

Example Flash Messages

- Example: We're signing up users for a newsletter, and we want to redirect them to the newsletter archive after they sign up.

```
app.post('/newsletter', function(req, res){
    var name = req.body.name || '', email = req.body.email || '';
    // input validation
    if(!email.match(VALID_EMAIL_REGEX)) {
        if(req.xhr) return res.json({ error: 'Invalid name email address.' });
        req.session.flash = {
            type: 'danger',
            intro: 'Validation error!',
            message: 'The email address you entered was  not valid.',
        };
        return res.redirect(303, '/newsletter/archive');
    }
```

```
    new NewsletterSignup({ name: name, email: email }).save(function(err){
        if(err) {
            if(req.xhr) return res.json({ error: 'Database error.' });
            req.session.flash = {
                type: 'danger',
                intro: 'Database error!',
                message: 'There was a database error; please try again later.',
            }
            return res.redirect(303, '/newsletter/archive');
        }
        if(req.xhr) return res.json({ success: true });
        req.session.flash = {
            type: 'success',
            intro: 'Thank you!',
            message: 'You have now been signed up for the newsletter.',
        };
        return res.redirect(303, '/newsletter/archive');
    });
});
```

Exercises

- Do login page: save user name in session then show in template

Execution Environments

- We can set execution environment by calling app.set('env', 'production')
- Or set NODE_ENV when execute nodejs then use app.get('env') to get its value.
- Example:

```
$ export NODE_ENV=production
$ node meadowlark.js
```

or

```
$ NODE_ENV=production node meadowlark.js
```

Database Persistence

- We use MongoDB to save data
- Before we get started, we'll need to install the Mongoose module:
  - npm install --save mongoose
- Then we'll add our database credentials to the *credentials.js* file:

```
mongo: {
    development: {
        connectionString: 'your_dev_connection_string',
    },
    production: {
        connectionString: 'your_production_connection_string',
    },
},
```

Database Connections with Mongoose

```
var mongoose = require('mongoose');
var opts = {
    server: {
       socketOptions: { keepAlive: 1 }
    }
};
switch(app.get('env')){
    case 'development':
        mongoose.connect(credentials.mongo.development.connectionString, opts);
        break;
    case 'production':
        mongoose.connect(credentials.mongo.production.connectionString, opts);
        break;
    default:
        throw new Error('Unknown execution environment: ' + app.get('env'));
}
```

Creating Schemas and Models

- To work with Mongo we need to define models
- *Example: models/vacation.js*

```
var vacationSchema = mongoose.Schema({
    name: String,
    slug: String,
    category: String,
    sku: String,
    description: String,
    priceInCents: Number,
    tags: [String],
    inSeason: Boolean,
    available: Boolean,
    requiresWaiver: Boolean,
    maximumGuests: Number,
    notes: String,
    packagesSold: Number,
});
vacationSchema.methods.getDisplayPrice = function(){
    return '$' + (this.priceInCents / 100).toFixed(2);
};
var Vacation = mongoose.model('Vacation', vacationSchema);
module.exports = Vacation;
```

Seeding Initial Data

var Vacation = require('./models/vacation.js');

```
Vacation.find(function(err, vacations){
    if(vacations.length) return;

    new Vacation({
        name: 'Hood River Day Trip',
        slug: 'hood-river-day-trip',
        category: 'Day Trip',
        sku: 'HR199',
        description: 'Spend a day sailing on the Columbia and ' +
            'enjoying craft beers in Hood River!',
        priceInCents: 9995,
        tags: ['day trip', 'hood river', 'sailing', 'windsurfing', 'breweries'],
        inSeason: true,
        maximumGuests: 16,
        available: true,
        packagesSold: 0,
    }).save();
```

Retrieving Data

- To display only vacations that are currently available create a view for the *vacations* page, *views/vacations.handlebars*:

```
<h1>Vacations</h1>
{{#each vacations}}
    <div class="vacation">
        <h3>{{name}}</h3>
        <p>{{description}}</p>
        {{#if inSeason}}
            <span class="price">{{price}}</span>
            <a href="/cart/add?sku={{sku}}" class="btn btn-default">Buy Now!</a>
        {{else}}
            <span class="outOfSeason">We're sorry, this vacation is currently
            not in season.
            {{! The "notify me when this vacation is in season"
                page will be our next task. }}
            <a href="/notify-me-when-in-season?sku={{sku}}">Notify me when
            this vacation is in season.</a>
        {{/if}}
    </div>
{{/each}}
```

Retrieving Data

- Now we can create route handlers that hook it all up:

```
app.get('/vacations', function(req, res){
    Vacation.find({ available: true }, function(err, vacations){
        var context = {
            vacations: vacations.map(function(vacation){
                return {
                    sku: vacation.sku,
                    name: vacation.name,
                    description: vacation.description,
                    price: vacation.getDisplayPrice(),
                    inSeason: vacation.inSeason,
                }
            })
        };
        res.render('vacations', context);
    });
});
```

Adding Data

- Create model object and call save method
- To update data, let check other example: we create the schema and model (*models/vacationInSeasonListener.js*):

```
var vacationInSeasonListenerSchema = mongoose.Schema({
    email: String,
    skus: [String],
});
var VacationInSeasonListener = mongoose.model('VacationInSeasonListener',
    vacationInSeasonListenerSchema);

module.exports = VacationInSeasonListener;
```

➔ new VacationInSeasonListener({email:”vdhong@...”, skus:”...”}).save();

Updating Data

```
var VacationInSeasonListener = require('./models/vacationInSeasonListener.js');
```

- In route handler we can call update() like:

```
VacationInSeasonListener.update(
    { email: req.body.email },
    { $push: { skus: req.body.sku } },
    { upsert: true },
    function(err){
        if(err) {
            console.error(err.stack);
            req.session.flash = {
                type: 'danger',
                intro: 'Ooops!',
                message: 'There was an error processing your request.',
            };
            return res.redirect(303, '/vacations');
        }
        req.session.flash = {
            type: 'success',
            intro: 'Thank you!',
            message: 'You will be notified when this vacation is in season.',
        };
        return res.redirect(303, '/vacations');
    }
);
```

Exercise

- Define student models and add some student data