Web Programming

Lecture 19

Shamsa Abid

Agenda

- Session Management and Authentication
- Debugging

Authentication

- Authentication is for identifying users and provide different access rights and content depending on their id.
- In most cases the application provides a login form with certain credentials to verify a user.
- It's necessary to understand:
  - authentication as
  - authorization as
  - a session
  - cookies

Dependencies

- Following packages are used
  - body-parser (for parsing incoming requests)
  - express (to make the application run)
  - nodemon (restarting server when changes occur)
  - mongoose (object data modeling to simplify interactions with MongoDB)
  - bcrypt (for hashing and salting passwords)
  - express-session (to handle sessions)
  - connect-mongo (for storing sessions in MongoDB)

Stages

- User registration (setting up routes and database)
- Sessions and Cookies (connecting them to login routes)
- Creating custom middleware

User Registration

- UserSchema
  - the schema should describe the fields we have in our registration form and specify the data it can expect
  - Design the registration view pug/html
  - Create a POST route for sending the data to the server
  - Store the values of the filled out form and store the data in the db with the schema

Hashing and salting

- **Cryptographic** hash functions take a piece of information and return a string, representing this information.
- Hash values cannot easily be "unhashed" or decrypted and that's why they are a perfect fit for passwords.
- **Salt values** are random data that is included with the input for the hash function.

Bcrypt

- install the bcrypt package
- add a prehook to your mongoose schema.

```
//hashing a password before saving it to the database
UserSchema.pre('save', function (next) {
  var user = this;
  bcrypt.hash(user.password, 10, function (err, hash){
    if (err) {
      return next(err);
    }
    user.password = hash;
    next();
  })
});
```

Sessions

- HTTP is a stateless protocol, which means that web servers don't keep track of who is visiting a page.
- Displaying specific content to logged-in users require this tracking.
- Therefore **sessions** with a session ID are created.
- **Cookies** are key/value pairs managed by browsers. Those correspond with the sessions of the server.

Set up Sessions

- add the express-session package
- add the session middleware in your app

```
//use sessions for tracking logins
app.use(session({
  secret: 'work hard',
  resave: true,
  saveUninitialized: false
}));
```

Next Steps

- store the MongoDB userId (_id) in the req.session.userId
- setup the login route the same way you set up the register route (in the login you only have the username and password)
- authenticate the input against the data in the database in the user schema.

Authenticate

```
//authenticate input against database
UserSchema.statics.authenticate = function (email, password, callback) {
  User.findOne({ email: email })
    .exec(function (err, user) {
      if (err) {
        return callback(err)
      } else if (!user) {
        var err = new Error('User not found.');
        err.status = 401;
        return callback(err);
      }
      bcrypt.compare(password, user.password, function (err, result) {
        if (result === true) {
          return callback(null, user);
        } else {
          return callback();
        }
      })
    });
}
```

Refining the app

- make sure to adapt your layout accordingly to the sessions (hiding register fields and providing logout buttons)
- create a middleware to make user IDs available in HTML
- create a logout route that destroys the session id and redirects back to the home route.

Destroying Session

```
// GET /logout
router.get('/logout', function(req, res, next) {
  if (req.session) {
    // delete session object
    req.session.destroy(function(err) {
      if(err) {
        return next(err);
      } else {
        return res.redirect('/');
      }
    });
  }
});
```

Creating custom middleware

- Middleware runs after a request is received, but before a response is sent back.
- Middleware functions can be chained after each other and fit into the request/response cycle of the application.
- When writing custom middleware, next() always has to be called at the end of that middleware to move to the next one in the cycle.

Example

- Creating middleware that requires a login for certain pages.

```
function requiresLogin(req, res, next) {
  if (req.session && req.session.userId) {
    return next();
  } else {
    var err = new Error('You must be logged in to view this page.');
    err.status = 401;
    return next(err);
  }
}
```

```
router.get('/profile', mid.requiresLogin, function(req, res, next) {
  //...
});
```

A note on scalability with sessions

- Currently sessions are stored in RAM.
- To store have more size we can connect the session store to MongoDB.
- Use the connect-mongo package for that.
- When checking with the mongo shell you should see how the new collection "sessions" is created.
- When logging in or out the data in that collection changes accordingly.

```
//use sessions for tracking logins
app.use(session({
  secret: 'work hard',
  resave: true,
  saveUninitialized: false,
  store: new MongoStore({
    mongooseConnection: db
  })
}));
```

res.locals

- We will use this to store the logged in users id and name
- **res.locals** An object that contains response local variables scoped to the request, and therefore available only to the view(s) rendered during that request / response cycle (if any). Otherwise, this property is identical to app.locals.
- This property is useful for exposing request-level information such as the request path name, authenticated user, user settings, and so on.
- Source: http://expressjs.com/en/api.html#res.locals

Loc8r case study

- Making a Register page
- Making a Login Page
- Adding Login, Register and Logout options in layout
- When a user registers, redirect to Login Page
- Only allow logged in users to Add Review
- When a user logs in and adds a review, save the users id in the review, so update the review schema to include user id
- Allow logged in users to update/delete their own reviews
- Show edit delete options in the view against a users personal reviews only

Other ways to authenticate

- Session based authentication
- Token based authentication
  - https://scotch.io/@devGson/api-authentication-with-json-web-tokensjwt-and-passport
  - https://medium.com/front-end-hacking/learn-using-jwt-with-passport-authentication-9761539c4314

References

- https://github.com/Createdd/Writing/blob/master/2017/articles/AuthenticationIntro.md#set-up-sessions
- https://security.stackexchange.com/questions/81756/session-authentication-vs-token-authentication