COMP 2406: Fundamentals of Web Applications

Winter 2014 Mid-Term Exam Solutions

1. **(true)** The Register button on "/" causes a form to be submitted to the server.
2. **(false)** In JavaScript, accessing object properties that have not been previously defined will generate a runtime error.
3. **(false)** To create new rooms in the game we must manually define new routes.
4. **(true)** Rooms can be stored in the "rooms" MongoDB collection that can never be accessed, even by directly typing in their name as part of a URL.
5. **(false)** Restarting the web application will cause all currently playing players to be redirected to the login/start screen.
6. **(true)** In general, HTML forms can be submitted successfully even if the browser has disabled JavaScript.
7. What HTTP command is normally used to submit a form? **A: POST**
8. (3) What MongoDB database does this application use? What collections? (Do not include any "system." collections that are used internally by MongoDB.)
   A: sessions, players, rooms
9. (2) Which node module(s) implement the methods `toArray()` and `findOne()`?
   A: mongodb
10. (14) How will program functionality change or what sort of error do you expect to see when the following lines are deleted from `small-adventure`?
    i. app.js, line 14: `app.use(express.urlencoded());`
    A: A: The body of HTTP POST requests won't be parsed, meaning `req.body` will not be defined. This will cause a reference error on playername the moment the user tries to log in or register (start() and register()).
    ii. app.js, line 24: `app.use(express.static(path.join(__dirname, public)));`
    A: The files in public/ will no longer be sent to the client in response to HTTP GET requests. This will prevent jQuery, Bootstrap, and our home.js files from being included in web pages. These omissions will cause the pages of the web app to revert to their default styling; also, because home.js cannot be loaded the Register button on / will now do nothing when pressed.
    iii. app.js, line 44: `var activeRooms = docs[0].activeRooms;`
    A: On application start-up we'll get a reference error on the next line with the call to activeRooms.forEach(); since activeRooms is now undefined, it has no properties to access (including no forEach()). The server will thus fail to start.
    iv. app.js, line 54: `routes.connectToDBs(createRooms);`
    A: We will get a TypeError in a crash page on clicking the start button (running start()) because playersCollection is undefined; thus the game cannot be played. Note that we also cannot navigate directly to any rooms because the routes for those rooms have not been defined.
    v. routes/index.js, line 12: `callback()`
    A: callback() is a call to createRooms(). If it is not called no rooms will be called. Thus the effect on the application is that we can log in but we'll get a message "cannot GET /bridge" for the first room or for any other room that we enter directly into the URL bar in the browser.
    vi. routes/index.js, line 23: `next()`
    A: The request will hang whenever we attempt to access / when not logged in (because next() is a call to routes.index()); thus, we can never log in. (If we are already logged in, the game will play normally.)
    vii. routes/index.js, line 97–103: `playersCollection.update()`
    A: (This should be for lines 96-102.) If we remove this call then the player's current state will never be saved. Thus they can play the game normally; however, when they quit and then return to the game, they'll be back in the room they first start in—the bridge—rather than being in the room they were in when they last stopped playing. (If you try deleting 97-103 in the source as listed, you'll get an error when rendering any room because we no longer are retrieving a room from roomsCollection).)

11. (2) In the `mongodb` module, `MongoClient()` should be originally assigned as a method of what object? Why?
A: MongoClient should have been assigned as a property of the `exports` object, because it is the exports object that is returned when a module is `require`'d. (Note: MongoClient is actually just an object, not a function. It has methods such as connect().)

12. (2) While developing this web application you manage to break the quit/logout functionality. How could you start a new game session (with a new player and password) using the same browser (i.e., without starting a new browser on the same or a different computer)?
A: You could clear the browser's cookies or just delete the cookies for your application in the browser.

13. (4) How many times is line 30 of app.js (`app.get('/', routes.redirectLoggedIn, routes.index);`) run? When are these invocation(s) of this line made, and who calls it? Explain briefly.
A: This line is run only once, when app.js is initially evaluated by running the command "node app.js". It is thus called by the node JavaScript runtime's file read-eval-print loop, the routine for executing all of the lines in a source file in the exact same fashion as if they were each typed at a prompt one at a time. (Note that this line registers a callback for the / URL; thus the two functions are potentially invoked for each browser request for this URL, with redirectLogin() being called every time and routes.index() only being called if the player isn't logged in in that session.)

14. (6) For each of the three functions defined inside of `makeRoomHandler()` on lines 92–118 of routes/index.js explain when each is called and what each does.
A: The three functions are:

    - **handler(): This function is called in response to a request for any room URL (e.g. /bridge). It sets the new rooms, updates the players collection to save the fact that the player has moved rooms, loads the room data from the rooms collections, and then uses this info to render the room page to be returned to the browser.**
    - **callback for playerCollection.update(): This callback is called once the call to playerCollection.update() has finished doing its work. It checks to see whether the update finished successfully and logs an error to the server's console if it did not.**
    - **callback for roomsCollection.findOne(): This callback is called once the query for the requested room document (from the rooms collection) has completed. If the query failed it throws an error (likely crashing the server); if it succeeded it runs res.render() to render the page using the retrieved room document/object.**

    (Note that this question was supposed to refer to lines 91–117; if you look at line 118 you have a fourth function that is being defined, getRooms(), which returns a cursor object for a query to retrieve the activeRooms document from the rooms MongoDB collection.)

views/layout.jade:

```
doctype html
html
  head
    title= title
    script(src='/vendor/jquery/jquery.js')
    script('vendor/bootstrap/dist/js/bootstrap.js')
    link(rel='stylesheet', href='/vendor/bootstrap/dist/css/bootstrap.css')
    link(rel='stylesheet', href='/stylesheets/style.css')
    block header
  body
    block content
```

views/index.jade:

```
extends layout

block header
  script(src='/javascripts/home.js')

block content
  h1= title
  - if(error)
    div.alert-error #{error}
  p Please log in
  div
    form(action="/start", method="post")
        div.control-group.input-append
            input(type="text", name="playername")
            label.add-on(for="playername") Player Name
        div.control-group.input-append
            input(type="password", name="password")
            label.add-on(for="password") Password

        button(type="submit") Start
        button#register(type="button") Register
```

views/room.jade:

```
extends layout

block content
  h1= title
  p #{description}
  p Go to:
  ul
    each theExit in roomExits
      li
        a(href= theExit) #{theExit}
  form(action="/quit", method="post")
     button(type="submit") Quit
```

public/javascripts/home.js:

```
$(function(){
  $("#register").on("click",function(){
    var $form = $("form");
    $form.attr("action","/register");
    $form.submit();
  });
});
```

app.js:

```
var express = require('express');
var routes = require('./routes');
var path = require('path');
var http = require('http');
var MongoStore = require('connect-mongo')(express);

var app = express();

app.set('port', process.env.PORT || 3000);
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'jade');
app.use(express.favicon());
app.use(express.logger('dev'));
app.use(express.urlencoded());

app.use(express.cookieParser());
app.use(express.session({
    cookie: {maxAge: 60000 * 20} // 20 minutes
    , secret: "Shh... I'm a secret"
    , store: new MongoStore({db: "adventure-demo"})
}));

app.use(app.router);
app.use(express.static(path.join(__dirname, 'public')));

if ('development' == app.get('env')) {
  app.use(express.errorHandler());
}

app.get('/', routes.redirectLoggedIn, routes.index);
app.post("/register", routes.register);
app.post("/start", routes.start);
app.post("/quit", routes.quit);

var createRooms = function() {
    var i, theRoom;

    routes.getRooms().toArray(
        function(err, docs) {
            if (err) {
                throw "Couldn't find active room list";
            }

            var activeRooms = docs[0].activeRooms;

            activeRooms.forEach(function(roomName) {
                console.log('Creating room: ' + roomName);
                app.get('/' + roomName,
                        routes.makeRoomHandler(roomName));
            });
        }
    );
}
routes.connectToDBs(createRooms);

http.createServer(app).listen(app.get('port'), function(){
  console.log('Express server listening on port ' + app.get('port') +
              ' in ' + app.get('env') + ' mode.');
});
```

routes/index.js:

```
var mc = require('mongodb').MongoClient;
var playersCollection, roomsCollection;
exports.connectToDBs = function(callback) {
    mc.connect('mongodb://localhost/adventure-demo', function(err, db) {
        if (err) {
            throw err;
        }

        playersCollection = db.collection('players');
        roomsCollection = db.collection('rooms');

        callback();
    });
}
exports.index = function(req, res){
    res.render('index', { title: 'COMP 2406 Small Adventure Demo',
                          error: req.query.error });
}
exports.redirectLoggedIn = function(req, res, next) {
    if (req.session.player) {
        res.redirect("/" + req.session.player.room);
    } else {
        next();
    }
}
exports.redirectNotLoggedIn = function(req, res, next) {
    if (req.session.player) {
        next();
    } else {
        res.redirect("/");
    }
}
exports.register = function(req, res) {
    var playername = req.body.playername;
    var password = req.body.password;

    var addPlayer = function(err, players) {
        if(players.length!=0){
            res.redirect("/?error=player already exists");
            return;
        }

        var newPlayer = {
            playername: playername,
            password: password,
            room: "bridge"
        };

        playersCollection.insert(newPlayer, function(err, newPlayers){
            if (err) {
                throw err;
            } else {
                res.send('Successfully registered ' + newPlayers[0].playername);
            }
        });
    }

    playersCollection.find({playername: playername}).toArray(addPlayer);
}
exports.start = function(req, res){
    var playername = req.body.playername;
    var password = req.body.password;

    playersCollection.findOne({playername: playername}, function(err, player) {
        if (err || !player){
            req.session.destroy(function(err) {
                res.redirect("/?error=invalid playername or password");
            });
            return;
        }
```

```
71
72          if (password === player.password) {
73              req.session.player = player;
74              delete req.session.player._id;
75              res.redirect("/" + player.room);
76          } else {
77              req.session.destroy(function(err) {
78                  res.redirect("/?error=invalid playername or password");
79              });
80          }
81      });
82  }
83  exports.quit = function(req, res){
84      req.session.destroy(function(err){
85          if(err){
86              console.log("Error: %s", err);
87          }
88          res.redirect("/");
89      });
90  }
91  exports.makeRoomHandler = function(roomName) {
92      var handler = function(req, res) {
93          if (req.session.player) {
94              var player = req.session.player;
95              player.room = roomName;
96              playersCollection.update({"playername": player.playername},
97                                       player, function(err, count) {
98                                           if (err) {
99                                               console.log(
100                                                  "Couldn't save player state");
101                                          }
102                                      });
103             roomsCollection.findOne(
104                 {name: roomName},
105                 function(err, room) {
106                     if (err || !room) {
107                         throw "Couldn't find room " + roomName;
108                     }
109                     res.render("room.jade", room);
110                 }
111             );
112         } else {
113             res.redirect("/");
114         }
115     }
116     return handler;
117 }
118 exports.getRooms = function() {
119     return roomsCollection.find({name: "roomList"});
120 }
```