COMP 2406: Fundamentals of Web Applications

Winter 2014 Final Exam Solutions
Instructor: Anil Somayaji
April 24, 2014

This exam has 5 pages (including this one). There are 20 questions. Each question is worth one point except for those marked otherwise; all together there are 70 points. Answer all questions in an exam booklet. This exam is closed: no books, notes, or electronic aids.

Many of the questions refer to the code at the end of this midterm for `simple-adventure`. When analyzing the behavior of the code, assume that MongoDB has been appropriately initialized by our standard `storeRooms.js` or equivalent.

You have 2.5 hours. Good luck!

1. **(true)** The Register button on "/" causes a form to be submitted to the server.
2. **(true)** Cookies are sent from the browser to the server as part of each HTTP request (assuming the site has set cookies in the browser).
3. **(true)** In JavaScript, accessing object properties that have not been previously defined will return *undefined*.
4. **(false)** To create new rooms in `simple-adventure` we must manually define new routes.
5. **(false)** Rooms can be stored in the "rooms" MongoDB collection that can never be accessed, even through a custom POST request.
6. **(false)** Restarting the web application will cause all currently playing players to be redirected to the login/start screen.
7. **(false)** HTML forms do not work when JavaScript has been disabled in the browser.
8. What is the HTTP return status code that means "page not found"?
   A: 404
9. What HTTP command is normally used to submit a request for a page?
   A: GET
10. (3) What MongoDB collections are used by `simple-adventure`? this application? (Do not include any "system." collections that are used internally by MongoDB.)
    A: players, rooms, sessions
11. (2) Which node module(s) must we require in order to define the methods `join()` and `createServer()`?
    A: path (for join) and http (for createServer)
12. (16) How will program functionality change or what sort of error do you expect to see when the following lines are deleted from `simple-adventure`?
    i. views/room.jade, line 4: `script(src=/javascripts/adventure.js)`
       A: We'll just see the default HTML produced from room.jade, with things like the room title being "Room Name". There will be no working exit buttons and indeed the game will be non-functional.
    ii. public/javascripts/adventure.js, line 6: `$(#description).html(room.description);`
       A: The room description will never get updated, leaving it to be the default value of "Room Description." Otherwise the game will work fine.
    iii. public/javascripts/adventure.js, line 12: `$(.exits).remove();`
       A: The old exits will never be deleted. So we'll start with the "Exits" button and then for every room they visit (starting with the first) all of the exits will be appended to the end of this button list. Except for the first, all of the exit buttons will work.
    iv. public/javascripts/adventure.js, line 19–20: `$.post("doAction", ...)`
       A: Exit buttons will do nothing. The player will stay in the room they are currently in.
    v. app.js, line 25: `app.use(express.static(path.join(__dirname, public)));`
       A: Files in public will no longer be served to the client. All pages will have their default style as bootstrap won't be loaded. The initial login form will continue to work for regular logins; registering a new user won't work as that depends upon home.js being loaded. The game itself will be non-functional as adventure.js won't be loaded by the client.

vi. app.js, line 26: `app.use(express.errorHandler());`
A: When the server encounters an internal error in handling a page it won't send a backtrace in the response error page; otherwise the app will behave normally.

vii. routes/index.js, line 12: `roomsCollection = db.collection(rooms);`
A: roomsCollection will be undefined; thus any accesses to methods of this object will generate an error. getContents() will now generate an error when calling roomsCollection.findOne (as undefined is not a function); thus the player will end up in the default room, the Void, once they start playing.

viii. routes/index.js, lines 42–43: `res.render('registered', ...);`
A: After clicking on the register button, the browser will hang attempting to load the response page. register() will still have done its job though, so if the player cancels the page load and goes back to the login screen they'll be able to log in with their newly registered credentials normally.

13. (2) Briefly, why are functions assigned as methods of the `exports` object in routes/index.js?
A: Functions are assigned to the exports object because require('./routes') returns this exports object and in app.js this return value is assigned to routes. Thus methods of exports in routes/index.js are methods of the routes object in app.js.

14. (2) When programming AJAX-style applications in Node as we have in class, how are functions/route handlers that return AJAX responses (such as `getContents()`) programmed differently from ones that return standard responses (such as `index()`)? Why the difference? Explain briefly.
A: Handlers that return AJAX responses use res.send() to return JSON objects, because the client expects to receive JSON-encoded data that is fed into client-side JavaScript to render an interface for the user in the browser. In contrast, standard responses return HTML pages which, in our app, are generated from Jade templates by calls to res.render(). HTML encodes entire new pages that potentially use no client-side JavaScript at all. In other words, when an entire page is to be returned, we use res.render(); when data is being returned to JavaScript code running in the browser, we use res.send().

15. (4) If you are playing `simple-adventure` and then decide to delete all of your browser's cookies, will you be able to continue your game? Explain briefly. (You need to be clear on what you mean by "continue the game" and realize that this phrase may have multiple meanings.)
A: By deleting your browser's cookies you lose your current session. /getContents won't return a room anymore (because cookies are needed for the server to set the correct req.session). Thus /game won't be updated, and so you won't be able to play anymore. However, if you reload the page you'll be redirected to /. There you can log in again and continue the game as before.

16. (4) How many times is line 28 of app.js (`app.get('/', routes.index);`) run? When are these invocation(s) of this line made, and who calls it? Explain briefly.
A: Line 28 runs once, when app.js is evaluated when the server first starts (i.e., when node app.js is run). Because this line is executed at server start, the function routes.index is called every time the server receives a GET request for /.

17. (4) `simple-adventure` is very sparse on error checking. What are two places in the code where it would be good to add error handling? What kinds of error(s) could you catch with each handler?
A: There are many errors that are not caught. For example:

- **connectToDBs() does not check to see whether we successfully connected to the database. Adding an error handler that checked err on line 11 of routes/index.js would be that you could catch if MongoDB was not running or was incorrectly configured.**
- **There is no check in register() that playername and password have actually been set; because of this it is possible to set a player name or password to "undefined".**
- **Another error—or more an attack—that you could catch would be to check in doAction() whether the requested room is actually an exit of the current room. This check would prevent users from jumping to random rooms simply by forging a POST request.**
- **In adventure.js, we don't check whether the /getContents or /doAction AJAX calls fail. If we checked for this we could put up a message saying we are having problems connecting to the server while retrying. Note to check for failure you'll have to chain a .fail() call to register an additional callback; see the jQuery documentation for $.getJSON().**

18. (4) If you visit `http://localhost:3000/getContents` in your web browser with `simple-adventure` running on your local machine, what will the browser display? Explain briefly.
A: If you are not logged in, you'll get a page with just one string in its contents, "Error: NotLoggedIn". If you are logged in you'll see the returned room object (corresponding to the player's current room) in JSON notation, i.e., as a text document.

19. (10) For a room document stored in the rooms collection, specify for each property its name, the type of its value, and whether it is mandatory or optional for the proper functioning of `simple-adventure`. (You only need to specify properties that the application explicitly accesses.)
A: The following properties of the room object are used:
    - **name (string): mandatory, in that we can't find rooms without this property.**
    - **title (string): mandatory, in that it is always output and is never checked whether it is undefined.**
    - **description (string): mandatory, in that it is always output and is never checked whether it is undefined.**
    - **activity (string): optional, in that the code checks if there is no activity and outputs an empty string if so.**
    - **exits (array of strings): mandatory, although the code functions properly if it is a zero-length array—the room just won't have any exits. If it isn't an array the code will generate an error as the forEach method won't be defined in adventure.js line 13.**

20. (10) For each of the five functions defined in public/javascripts/adventure.js, explain when each is called and what each does.
A: The functions are:
    - **Line 1, callback for $: Called when the page has finished loading (and just exists in order to do this deferred execution). It also provides a local scope for the rest of the code.**
    - **Line 2, getRoom(): Called when the previous function is called (the invocation at the end) and every time the user enters a new room (callback for the doAction POST). This function contains the code for getting the code for getting the player's current room and updating the page accordingly. (Much of the key functionality is implemented in the following functions.)**
    - **Line 3, callback for getJSON(): This function is called once getJSON has gotten a JSON response for "GET /getContents". It uses the returned room object to update the DOM with the current room's state.**
    - **Line 13, callback for forEach(): This function is called once per element in the room.roomExits array. It creates an exit button for the given room exit.**
    - **Line 18, callback for .on('click'): This is the callback function that is run every time a user clicks on an exit button. It sends a background (AJAX) POST /doAction in order to move the player to a new room. (The $.post() calls getRoom() once the post has completed.) Note "theExit" gets its value from the enclosing forEach callback instance.**

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 header
  script(src='/javascripts/adventure.js')

block content
  h1(id="title") Room Name
  p(id="description") Room Description
  p(id="activity") Room Activity
  p Go to:
  div(id="exitList").btn-group
    button(type="button").btn.btn-default.exits Exits
  p
  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();
  });
});
```

public/javascripts/adventure.js:

```
$(function() {
    var getRoom = function() {
        $.getJSON("/getContents", function(room) {
            var exits;
            $('#title').html(room.title);
            $('#description').html(room.description);
            if (room.activity) {
                $('#activity').html(room.activity);
            } else {
                $('#activity').html("");
            }
            $('.exits').remove();
            room.roomExits.forEach(function(theExit) {
                $('#exitList').append(
                    '<button type="button" id="' + theExit +
                        '" class="btn btn-default exits">'
                        + theExit + '</button>');
                $('#'+theExit).on('click', function() {
                    $.post("/doAction", {action: "move",
                                         room: theExit}, getRoom);
                });
            });
        });
    }
    getRoom();
});
```

app.js:

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

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.logger('dev'));
app.use(express.json());
app.use(express.urlencoded());
app.use(express.methodOverride());

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

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

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

routes.connectToDBs();
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;
var defaultRoom = {   name: "theVoid",
                      title: "The Void",
                      description: "You are in the Void.  " +
                      "How did you get here?  There are no exits.",
                      roomExits: [’theVoid’]
                  };
exports.connectToDBs = function() {
    mc.connect(’mongodb://localhost/adventure-demo’, function(err, db) {
        playersCollection = db.collection(’players’);
        roomsCollection = db.collection(’rooms’);
    });
}
var savePlayer = function(player) {
    var errorHandler = function(err, count) {
        if (err) {
            console.log("Couldn’t save player state");
        }
    }
    playersCollection.update({"playername": player.playername},
                             player, errorHandler);
}
exports.index = function(req, res) {
    res.render(’index’, { title: ’COMP 2406 Simple Adventure’,
                          error: req.query.error });
}
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){
            res.render(’registered’,
                       { playername: 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 || password !== player.password){
            req.session.destroy(function(err) {
                res.redirect("/?error=invalid playername or password");
            });
            return;
        } else {

            req.session.player = player;
            delete req.session.player._id;
            res.redirect("/game");
        }
    });
}
exports.quit = function(req, res){
    req.session.destroy(function(err){
        if (err) {
            console.log("Error: %s", err);
        }
        res.redirect("/");
    });
}
exports.game = function(req, res) {
    if (req.session.player) {
        res.render("room.jade", {title: "AJAX Adventure Demo"});
    } else {
        res.redirect("/");
    }
}
exports.doAction = function(req, res) {
    var action = req.body.action;
    var room = req.body.room;

    if (!req.session.player) {
        res.send("Error: NotLoggedIn");
        return;
    }

    if (action === "move") {
        req.session.player.room = room;
        savePlayer(req.session.player);
        res.send("Success");
    } else {
        res.send("Error: InvalidAction");
    }
}
exports.getContents = function(req, res) {
    if (!req.session.player) {
        res.send("Error: NotLoggedIn");
        return;
    }

    roomsCollection.findOne(
        {name: req.session.player.room},
        function(err, room) {
            if (err || !room) {
                room = defaultRoom;
            }
            res.send(room);
        }
    );
}
```