Battelship Game State Persist
S19 CSCI470 Web Science

Assigned: 2019-02-08
Reference: JavaScript, CSS, HTML5, Ajax, JSON Due Date: 2019-02-16

Summary:

Abstract:	Battleship Gameboard State Persist
Objectives:	1. HTML5 Semantic Tags - Structural 2. Simple CSS3 for Presentation 3. JavaScript for Behaviors 4. IIFE and Scoping 5. JSON and AJAX 6. Node
Grading:	45 pts - A $\geq$ 41.85; A- $\geq$ 40.50; B+ $\geq$ 39.15; B $\geq$ 37.35; B- $\geq$ 36; C+ $\geq$ 34.65; C $\geq$ 32.85; C- $\geq$ 31.75; D+ $\geq$ 30.15; D $\geq$ 28.35; D- $\geq$ 27
Outcomes:	R2 (CAC-a,c,i, j, k; EAC-a, b, c, e, k, 1, 2); R5 (CAC-a, i; EAC-a, k) (see syllabus for description of course outcomes)

Project Description:

Be able to use Ajax calls to a node server to persist and retrieve persisted Battleship Game Board state from a JSON object. You will implement that [Save Game] and [Load Game] menu items to persist a Gameboard to the Node server and retrieve a previously persisted Gameboard from the Node server from which the Gameboard will be initialized.

Obtaining Project Files and Setting up the Initial Web App:

1. Logon to `gitlab.cs.mtech.edu` and locate the project `bsStatePersist` under the |S19 CSCI470| (sub)group, and then fork this project into your own account.
2. Navigate to your own account and locate the `bsStatePersist` project to just forked, and copy the project url
3. Next, logon to `csdept16.mtech.edu` using your Department username and password.
4. Execute the `mkdir ~/CSCI470/Projects/` command, which will create a projects folder if not already created.
5. Execute the `cd ~/CSCI470/Projects` command, which will change the current working directory to the specified parameter.
6. Issue the command `git clone <project_url>`, where `<project_url>` is the url you copied in the above step. This will create the project folder inside your `~/CSCI470/Projects` directory,
7. Execute the command `cd bsStatePersist` to enter the project directory.
8. Continue with the specific project activities below.

Project Activities:

Please perform the following activities in the completion of the lab assignment.

1. Create a basic node express application by executing

   ```
   express bsServer
   cd bsServer
   npm install
   ```

2. Edit your startup file – `bin/www` – for your server to make use of your unique port by editing the line

   ```
   var port = normalizePort(process.env.PORT || '3000');
   ```

   to read

   ```
   var port = normalizePort(process.env.EXPRESSPORT || '3000');
   ```

3. Start your server by executing the command `npm start` from the `bsServer` directory.
4. Navigate your browser to

   ```
   http://csdept16.mtech.edu:30120
   ```

   where `30120` is your port id, and your should see a welcome message from express.
5. Familiarize yourself with the contents of the `app.js` file and chapter-3 of your text book.
6. Write four routes in your `app.js` server
   (a) route: `bsState/[filename]` using a GET method
   (b) route: `bsState/[filename]` using a POST method
   (c) route: `bsStates` using the GET method
   (d) route: `bsState/[filename]` using a DELETE method
7. The first route should retrieve a bsState object returned in the response from the filename provided.
8. The second route should save a bsState object found in the request body to the filename provided.
9. The third route should retrieve a list of all the filenames of the bsState objects found on the server.
10. The fourth route should delete the bsState object on the server with the filename provided.
11. Make sure to store the files on the server in a folder not accessible from any other url or in the `public_html`.
12. You should develop a method whereby only your client can make these requests.
13. We will later implement a secure communications channel (HTTPS) between your node server and your client.

Figure 1: Programming Project Grading Rubric

Attribute (pts)	Exceptional (1)	Acceptable (0.8)	Amateur (0.7)	Unsatisfactory (0.6)
Specification (10)	The program works and meets all of the specifications.	The program works and produces correct results and displays them correctly. It also meets most of the other specifications.	The program produces correct results, but does not display them correctly.	The program produces incorrect results.
Readability (10)	The code is exceptionally well organized and very easy to follow.	The code is fairly easy to read.	The code is readable only by someone who knows what it is supposed to be doing.	The code is poorly organized and very difficult to read.
Reusability (10)	The code could be reused as a whole or each routine could be reused.	Most of the code could be reused in other programs.	Some parts of the code could be reused in other programs.	The code is not organized for reusability.
Documentation (10)	The documentation is well written and clearly explains what the code is accomplishing and how.	The documentation consists of embedded comments and some simple header documentation that is somewhat useful in understanding the code.	The documentation is simply comments embedded in the code with some simple header comments separating routines.	The documentation is simply comments embedded in the code and does not help the reader understand the code.
Efficiency (5)	The code is extremely efficient without sacrificing readability and understanding.	The code is fairly efficient without sacrificing readability and understanding.	The code is brute force and unnecessarily long.	The code is huge and appears to be patched together.
Delivery (total)	The program was delivered on-time.	The program was delivered within a week of the due date.	The program was delivered within 2-weeks of the due date.	The code was more than 2-weeks overdue.

The *delivery* attribute weights will be applied to the total score from the other attributes. That is, if a project scored 36 points total for the sum of *specification*, *readability*, *reusability*, *documentation* and *efficiency* attributes, but was turned in within 2-weeks of the due date, the project score would be $36 \cdot 0.7 = 25.2$.

Project Grading:

The project must compile without errors (ideally without warnings) and should not fault upon execution. All errors should be caught if thrown and handled in a rational manner. Grading will follow the *project grading rubric* shown in figure 1.

Collaboration Opportunities:

You may collaborate with up to one other person on this project - but you must cite, in the code (html, css, js) each students' contribution.