EECS 581: Lab 03

Chris Hogan

25 September 2014

Project repository: https://github.com/woodenToaster/lab2

1. Modifications

The main modifications include non-interference, interaction, and styling. The original goal for non-interference was that each user could have his or her own universe to interact with that is completely separate from other users’ worlds. This requires that each browser is tied to a unique cookie, and that each user has a personal clone of the world. I used the cookie-parser and express-session modules to implement sessions. Cookie-parser checks the cookie secret on each request to prevent attacks, and express-session generates unique id’s for users and allows programmatic access. When a new request arrives from a client, the server checks its in-memory list of users (called agents in my code). If the cookie assigned to that particular browser is not in the agents object, it creates a world, initializes the agent’s fields, and adds it to the list of agents. If an agent it already in the list, then nothing happens. Then, in each route I define the campus and inventory in terms of a user’s personal world. That way, only that user’s details are updated on a request.

After I had each user completely isolated in non-interfering environments, I began to add some interaction. It made sense to leave the inventories and worlds separate because each person is on a mission. For example, if someone picks up the basketball and never puts it down, then no one else can ever “win.” I started by listing users that are in the same room. For example, if you are at Strong Hall, and someone else playing the game in a different browser is also at Strong Hall, the each person will see a notification that there are other people there. I listed the actual cookie of other users for testing purposes. Realizing that was a horrible security vulnerability, I decided to play on that by including a button to use that information for “evil.” When someone is in the same location as you and you click the red button, that person’s next move will send them to jail, remove their entire inventory, and they will not be able to do anything until the server restarts.

Finally, I added Twitter Bootstrap to improve the appearance of the site.

2. Flaws

The way I clone the world for each user is highly inefficient. The campus object gets created on every single request. I could easily move that logic into a branch that is only taken if the agent is not already defined in the agents object. I simply don’t have the time.

I don’t think I’m using cookie-parser and express-session correctly. It was confusing that you can use cookies exclusively to accomplish the same thing as sessions (storing all state in the cookie instead of on the server), and figuring out where the two are supposed to overlap was a bit difficult.

3. Modules

cookie-parser – parses the cookie secret to set up express-session.

Express – handles routing, file serving, and http requests and responses.

express-session – sets up unique identifiers for cookies and allows access to those session id’s.

node-uuid – generates a unique identifier for express-session to use.

4. Implementing Sessions

The basic implementation of sessions requires some way to distinguish among different browsers, or different client instances. The most common way to do that is with a cookie. When a client first visits a web service, it is assigned a unique identifier. Then, on each subsequent request, that client can be identified. State and data are stored on the server under that identifier. When a request comes in, the server checks the client’s cookie, and processes the request based on the stored data and state under that cookie identifier. When I log in to a web site, my information is retrieved and displayed, and when someone else logs in, their information is displayed.