SiM v2.0.0

Lewis Carson

March 23, 2018

1 Abstract

A node.js webapp based on saving water. It includes a login/register function and a stats page.

2 Technologies Used

We have used a number of technologies including but not limited to:

- node.js (primarily)
- nodemailer
- express
- socket.io
- materialize
- chart.js

Every technology used in this project is F/LOSS

3 functions

3.1 register

When you first hit register, you generate a UID which will be the identity of your account for the rest of the forseeable future. In theory, it might be possible to force a specific user idenity which may cause duping problems. Function illustrated in extract below.

```
function guid() {
    function s4() {
        return Math.floor((1 + Math.random()) * 0x10000)
            .toString(16)
            .substring(1);
    }
    return s4() + s4() + s4() + s4() + s4() + s4() + s4();
}
```

Next it checks for duplicate passwords. If it detects that they are not the same, it displays an error message and exits the function.

```
if(readbox('password') != readbox('confirmpassword')){
    errorm('Passwords do not match')
    return
}
```

Finally it emits a json document to the server containing info about the user. The server puts this directly into the database and calls 'syncdbwrite'

3.2 main function

When a user starts their shower, no data is emited to the server. Instead, the client waits until the user has finished their shower before telling the server the score. The server handles this by doing a couple of checks to prevent cheating, then it processes achivements and appends it to the users shower history. At the end of this it calls 'syncdbwrite' and gracefully exits this function.

3.3 stats

A simple array containing dates/scored of a user's shower history is kept. When a user navigates onto the stats page, it asks the server for a copy of the user's profile which the client then parses into two values, a graph (using chart.js) and a list of achievements previously detailed in 'main function'.

3.4 db (database)

The database uses simple JSON, and accounts are found using the previously generated uid in the register step. I use one function for this, 'syncdbwrite'. This writes the json loaded into memory onto the disk (a file called accounts.json which is present in the .gitignore file and which can be found in the root directory of the repo.)

4 Conclusion

This was great fun to work on, and I feel sorry for any future maintainers who might (??) be forced to work on this.