Web Tech: Server Report

James Collerton

May 7, 2015

Packages Used

I used a number of packages in my server that will need to be installed using *npm*. I have tried to put them into *package.json* in the submission. Running the following should install all the necessary components if for whatever reason that doesn't work:

- *npm install nodemailer*
- *npm install git+https://github.com/orliesaurus/nodemailer-mailgun-transport.git*
- *npm install socket.io*
- *npm install sqlite3*
- *npm install validator*

New Files

To help with the marking I have included a list of the files with new content.

- css/forum_styles.css
- js/emailing_js.js
- js/facebook_share.js
- js/forum.js
- js/stats_gathering.js
- js/user_stats.js
- keys/cert.pem
- keys/key.pem
- server/server_email.js
- server/server_forum.js
- server/server_user_statistics.js
- forum.html
- user_stats.html
- server.js
- sql_testing.js

Introduction

I focussed on four main areas in my server assignment:

- **Automated email service:** The email modals I wrote now send emails using express and a MailGun client.
- **User statistics gathering:** The page that was previously set aside for the display of user statistics is now functional and gathers stats from the rest of the site.
- **Forum:** There is a page that works as a forum between separate instances of the site on the same machine.
- **Social Media Integration:** It is now possible to share the site over Facebook.

My aim was to include things that would add to my website's functionality, making it more polished and user friendly. Therefore the emphasis was less on technical elements and more biased towards the integration of open source libraries and packages to create a complete, professional feeling site.

I spent some time attempting to rewrite the server you provided, but found myself just writing the same thing out again. I decided that I would try adding components to the existing server rather than writing one from scratch.

I also looked at issues surrounding security, however there is little to protect on my site. I did, however, generate my own *SSL* key/ certificate using my *Cygwin* terminal.

The rest of the report will be broken down into:

- Setting up the server and server issues.
- Integration with a database and other technologies.
- Programming.
- General.

The general component will cover the features that were not relevant elsewhere.

Setting up the Server and Server Issues

As mentioned in the introduction, I spent time trying to rewrite what had been given to us, but felt as if I was just copying out what was already there. Because of this I have left a lot of the server as it was, but generated my own *SSL* key pair.

Something I found extremely useful was commenting and rearranging the existing *server.js* file. This meant at least I understood what was going on and, even though I hadn't wrote it, could have done so if necessary.

In terms of general server issues, one thing that I found very challenging was getting communication between the server and the client synchronous, so information was being relaid between the sides at

the correct times.

To solve this I used the *socket.io* package, which allowed for serialized, event-driven communication between the client and server. Setting up the package and learning how to use it was difficult to begin with, but once I was familiar with it *socket.io* proved very useful.

The other part that I found challenging was getting all of the packages and server functions to play nicely with each other. Even when I had different packages listening at different ports they could sometimes cause problems. Solving this took close examination of how all of the libraries were interacting, and careful coding surrounding this.

Integration with a Database and other Technologies

This is where the majority of the work was put in. As I mentioned in my client side report and in the introduction, my main focus was to create a polished site with an excellent user experience. In order to do this I used a number of different technologies including:

- **socket.io:** Controls the communication between the client and server. Used extensively throughout all of the project at any point when this is necessary.
- **validator:** Small but useful package used in the server side of the emailing client to validate that the email address is of the correct form.
- **sqlite3:** Library to control the interaction with the database of user statistics.
- **nodemailer:** This forms the basis of the email client. It is linked with a MailGun account that can be used to send emails using the email modal.
- **MailGun:** The MailGun plug-in is used with the *nodemailer* package in order to authorise requests to my MailGun account.
- **express:** Express is used to interact with a lot of these packages and integrate them into the server.

The easiest way to cover the use of these packages is to break down this section into the emailing client, the user statistics page and the forum. Then I will talk about how each of the different technologies were integrated into the separate stages.

Email Client

This was the first task that I attempted, due to the fact that it seemed the simplest way to get to grips with server side technologies. I carried out this part using only the documentation for the various packages. Setting up *socket.io* for this task took some time, but was well worth it.

One of the main problems was validating the email address the message is supposed to be from. I have one validation in the client side JavaScript, however after some research I read that it was also important to check this server side, as the client side JS could be switched off. In order to validate the email address on the server side it requires some to-and-fro between the client and server.

Primarily we send the email address to the server for checking using *validator*. The server then responds with the result of the validation. At this point the client either tells the user that the validation failed with a message or confirms the valid email address. After this the email address

and the body of the email are sent to *nodemailer* and sent away using *MailGun*.

Short bursts of several interactions where it is important that the responses remain serial were where I was very glad I used *socket.io*, as it forced the messages to remain chronological. You can see in the code the methods wait for the necessary result before composing a response.

nodemailer itself is quite intuitive to use, and comes with good documentation. It was a slight struggle to get it to function with MailGun, but I found a plug-in which eased the interaction of the two.

User Statistics Gathering

This was by far the most complex part of the server assignment that I attempted to implement, and is the part I am most proud of. The server gathers information from the client side, and creates a live feed back to a database. This database is then connected to a *D*3 visualisation, which can be used to view where people have spent most of their time on the site, and which parts they have interacted with most.

The reason I chose to try and implement this is because I hope to go into data science when I finish my course. I thought it would be a really interesting project to see if I could work an element of this into my server. I came up with the idea from scratch and used no tutorials or similar help, except for the *D*3 documentation.

The system works by each of the selected pages including the *stats_gathering.js* script, which collects the numbers of clicks, time spent, hits, and author contacts per page. To get a cumulative count of page views *etc.* the number of interactions per second are counted, and then added onto the existing amounts in the database.

I spent a lot of time making the statistics gathering flexible. As long as you attach the *stats_gathering.js* script, the server should automatically create space in the database for that page. The benefit of this is that you can reset the database values, and as you visit the pages of the website it will automatically create all of the necessary parts of the database to hold the user statistics. This is achieved by, from the *socket.io* given data, checking the database to see if there is already an entry for the considered page. If so it adds the new data onto the existing data, if not it creates an entry for that page and adds the data there.

Updating values in the database was incredibly hard to implement due to the fact that all steps needed to be kept serial. As I was incrementing numbers stored in the database, I had to pull the numbers from the database first, then add on the increments, then store the new values, all in order.

This was challenging as functions were asynchronous. I had problems with the server storing values before they were incremented, not pulling the old numbers in time to be incremented, and not storing the new numbers properly.

The solution I found was to call each function at the end of the one before it. This ensured that each step was completed before the next one was started.

To aid marking I have left in the ability to watch the stats being collected in real time through the node terminal.

```
function run_update(data){

    var sql_cmmd = "update stats_tab set " +
                   "Time = " + new_time +
                   ", Num_Clicks = " + new_clicks +
                   ", Hits =" + new_hits +
                   ", Contacts =" + new_contacts +
                   " where Page = '" + String(data['new_stats']['page']) + "'";

    db.run(sql_cmmd, err);

    // Uncomment this line to see the stats!
    // console.log("\n\n");
    // db.each("select * from stats_tab", show);
}
```

Figure 1: Uncomment the two lines to see the stats gathering in real time.

Finally, in order to supply data to the *D*3 visualisation, the user statistics page formats a request for the data, sends it using *socket.io*, and then has a response taken from the database and fed back to it. The core problem for this process was formatting the response so it can be used in *D*3. However, after some experimentation this was solved.

Forum

The forum was another idea I had for the server side. I created the forum as I was keen to use a little bit more of the functionality of *socket.io* and to use the backend to dynamically generate HTML content. I created the forum with the help of the *socket.io* documentation.

This was a lot more simple to implement than the user statistics gathering as there is no permanent storage. Messages are sent from the client side using *socket.io* and JavaScript and then broadcast to all online users. More JavaScript is then used to turn this information back to HTML and to display it on the page.

The best way to test this is to open up two tabs and start a conversation between them, watching for the messages being translated between the two.

If I had had a little more time it would have been really interesting to try and allow people to upload their own icons, send private messages and have a *user is typing* function. However, I didn't have time to implement this.

Programming

This section will highlight the programming I did in order to create my server side work. I will cover any components that I thought were particularly interesting and refer to the code, which you can look over in the submission.

All of the functions are short and broken up as much as is sensible. I also tried to move the server JavaScript into separate files depending on what it was related to (the email client is in one file, the user statistics in another, the forum in another), and to keep code as DRY and flexible as possible.

Another thing I tried to do was to incorporate all of the packages I chose to use neatly into the code. As each package is associated with one particular part of the server, they are only included in the relevant file. Therefore there exists minimal confusion about which libraries are used where.

General

Something I would like to cover in this section is all of the work I have had to do on the rest of the site in order to make all of the server work fit in.

Included in this was a revision of the email modal, so that it displayed messages telling you when mail had been sent correctly or not. As well as this a massive revision of the D3 graph was needed. This was a large undertaking as it involved learning about ordinal scales and similar. However, it did prove what I said in my client report about understanding the library and being able to use it well.

There was also the addition of a whole new page with all the necessary HTML and CSS to house the forum. Included in this was all of the JavaScript to make the entries to the forum as they were posted by the server.

The final piece of work I did on the server assignment also falls in this category. I was very keen to integrate social media into the project, and so looked to include a Facebook share button. Initially I tried implementing this with the *passport* library, however found that this overcomplicated things. Instead, I used the *SDK* provided by Facebook.

This required the creation of a Facebook app, and learning about the *SDK* from the Facebook website, and has added an extra dimension to the site. To aid marking I have included a screenshot of the app in action in case you do not have a Facebook account.

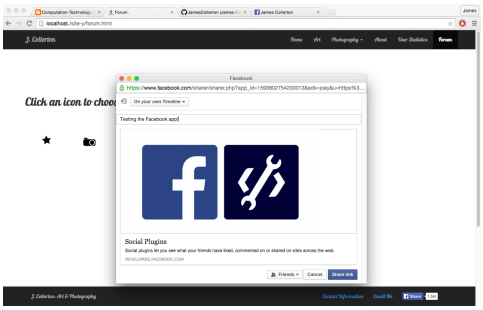

Figure 2: Link for sharing a placeholder site open on my website.

Figure 3: Post after it appears on my Facebook page.

Conclusion

The focus of my server side work was to add more, relevant functionality to my website, that would aid in the user experience. I feel I achieved this well, whilst investigating a wide range of open source options and implementing them effectively. Each of the parts required learning how to use packages and how to best integrate them with the rest of the website.

In conclusion, I feel I achieved my aim of creating a polished site with a range of functionality.