Project write-up CS554

Paromita Data

Based on the Gutenberg project (<https://www.gutenberg.org/wiki/Main_Page>) this application aims to create an Electron app using the feed of free books from Gutenberg. The catalog of books is available here: <https://www.gutenberg.org/wiki/Gutenberg:Feeds#The_Complete_Project_Gutenberg_Catalog>.

The application consists of two parts: The server side code (service-app) and an electron based front end UI (electron-app). Communication between the two modules happens via REST calls.

The service-app is responsible for:

1. Getting the catalog from Gutenberg server, extracting and parsing it and storing the list of books and the book’s html version URL in a database\*
2. Provide the list of books to the UI module. Also cache the frequently accessed pages
3. Fetch the content of the book as HTML when requested by UI and save a local copy
4. Maintain a library of books fetched by the UI

The electron-app is responsible for:

1. Providing a Single Page Application (SPA) for the Gutenberg catalog
2. The catalog consists of pages which contain names of books and links to see the contents
3. A library section which lists books that have been downloaded
4. Once the book hyperlink is clicked (from catalog or from library), the contents is displayed on the UI
5. The SPA is packaged inside Electron, and hence can be accessed without using a browser

\*The service-app exposes multiple GET requests for the electron-app to fetch catalog and library contents. However it also exposes one GET request (<http://localhost:3001/initialize>) which can be called to setup/reset the server. This flushes out the cache and database and performs step#1 as listed above. Ideally we can create a separate ADMIN page from which this REST call can be made to setup/reset the library. However for this project we simply make the call from a browser before we should start using the UI app.

The service-app uses mongodb as database, redis for caching the page information. For initialization, a separate worker is spawned using node-redis-pubsub. The rdf files from Gutenberg are parsed using the rdflib npm module. The archive is extracted using the unzip module. REST calls are exposed to the client using express server.

The electron-app uses react for the SPA pages. Browserify and babelify are used to create the equivalent javascript file which can then be packaged into the electron app.

To start the service-app make sure mongodb is running (from CLI):

mongod --dbpath=C:\work\mongoData

And then run *npm install* followed by *npm start* from the service-app workspace. Hit <http://localhost:3001/initialize> to initialize the server side data.

During development of the UI module, the following commands were used to create the skeleton of the react module:

npm install -g create-react-app

create-react-app electron-app

For testing the UI module initially, on the web browser, the package.json has the following entry:

"test": "react-scripts start"

And *npm run test* would be executed to test React in browser.

Once the development of the UI module was complete, to make sure it is working fine in the electron component, the package.json was updated with:

"start": "electron .",

"compile": "browserify -t [ babelify --presets [ react es2015 ] ] src/index.js -o public/js/app.js"

And *npm run compile* followed by *npm start* executed to test the packaged app.

Finally, to package the UI module as a native application (in my case specific to Windows OS), the following was executed:

electron-packager ./ gutenbergSPA --out=c:\work\workspace\_js --arch=all --ignore="(node\_modules|src)"

## Technical Issues and limitations:

I would have wanted to create and store the cache on the client end. However I was not able to incorporate redis inside React on the client end, without using npm packages that rely on Python. With limited time and working alone on the project, I didn’t have the option to do much research on this. The caching was hence implemented on the server-app.

The .tar.zip file downloaded from Gutenberg site extracts successfully to a .tar file using node. However the tar is not getting extracted due to invalid checksum error. I have tried various npm packages but all yield the same result. The .tar can however be extracted successfully with winrar tool manually. Also the contents of the tar are pretty huge. Based on these scenarios, I created my own .tar.zip file with 1000 rdf files taken from the Gutenberg archive. The code is written to download the original file from Gutenberg site and then extract to .tar. However the further extraction of .tar is done on my custom file rather than what was downloaded.

Also, since the rdf file contains links for books in various formats (epub, html, text etc). I coded only to extract the html version and show it on the UI.

I had considered handle cross-site scripting. However since the URL is not exposed (as we are not using a browser but electron), there was no real need to incorporate this.

I have attached the source code for service-app (along with my custom .tar.zip file), electron-app and the electron-packager output in 3 different zip files (excluding node\_modules).