NODE WEB PROTOTYPING

(NWP)

Adam Lloyd – R00117318

Advanced Web Publishing Apps – COMP8003

https://github.com/Radicis/npm-html-prototype

https://www.npmjs.com/package/nwp

HIGH LEVEL CONCEPT

For this project, I decided to explore the possibility of creating a standalone, native desktop application using JavaScript, which can be distributed via NPM.

JavaScript is conventionally a client side technology but, with the emergence of node.js, it is gaining popularity on the server side also. Node is a JavaScript runtime environment which, along with providing server side capabilities, allows it to access the host file system via the fs library. This is the core functionality that this project focused on as it overcomes one of the main disadvantages of using JavaScript since it has no access to the filesystem.

Given that JavaScript now has access to the host filesystem, and after learning that other applications such as the ATOM text editor were built using JavaScript, the idea of making a desktop application was born. After researching the problem, it became clear that I would need to leverage the power of the Electron library for node.js.

Electron is a framework that creates native, cross-platform applications with web technologies. It handles the communication between the web application and the host operating system by providing an API for interacting with the host system. The node web application is built into a standalone native application using Chromium instead of a conventional web browser as the client.

THE IDEA

With the above concept, I decided to create an application that mimicked a useful web application but provided extra functionality that the original was incapable of by leveraging Electron and node.js.

I decided to recreate a web prototyping application similar to CodePen, JsFiddle and Plunkr as it is something I used regularly and was relevant to my field of study.

I replicated the basic functionality of these applications, providing a clean interface for quickly prototyping web applications and testing scripts. Then I analysed the existing solutions for features that they lacked due to the limitations of being web applications.

While these applications are well fleshed out, none of them were able to scaffold a web application once prototyping was complete. Many had the option to download the source but it was zipped up and contained only minimal files without the directory structure one would expect from even a basic web site.

As such, this was the core focus of my project. To scaffold the directory structure of a web site/application containing all of the work that has been completed in the prototyping interface. This results in a project on the host filesystem that work can continue on without the need for manual steps to create the structure.

The ability to run the application from the command line after being added to the PATH and the speed of the application resultant from running locally were all improvements over the web applications that inspired it. The ability to run the application along with the ability for the user to add and save their own code snippets are also improvements over the web based alternatives.

TECHNOLOGIES

Node.js is the platform which the entire project is based on. It provides the runtime environment and the means to add Electron and other packages to the project.

Angular.js provides the business logic for the application and renders the interface. It allows for modular and extensible code via controllers and services.

Bootstrap is the UI framework used to leverage predefined UI elements which speed up development and increase consistency.

Electron builds the application and provides the API for interacting with the host system. It also wraps the application in Chromium to present the interface to the user.

JSON to persist user information (Snippets) a local flat file JSON database was used to maintain simplicity and portability without the need to set up servers.

CodeMirror provides the functionality to turn conventional text areas into rich code editors with syntax highlighting and line numbers.

Bower manages the client side packages allowing for separation between these and the core node packages.

FEATURES

Native Cross Platform implementation runs on anything!

Fast execution as there is no reliance on web servers. It uses the full resources of the host system.

Modular code with Node and angular modules being separated using bower and npm and functionality separated by using angular best practices.

User managed snippets using a JSON database making for extreme portability with no data source dependencies.

Stylish, **windowless interface** created by removing the native window manager and replacing it with pure HTML/CSS controls makes for as modern and clean look.

External library options enabling users to toggle on/off jQuery Bootstrap and more and add and remove with minor code edits.

Project Scaffolding which outputs the full project structure when finished prototyping and saves directly to filesystem

Packaged with NPM making it easy to install with npm and run from the command line.

Cool things like the Dialog service, Status service giving appropriate feedback to the user.

COMPLEXITY

Packaging proved difficult in that it had to be a consideration at all stages of development. The application needed to work as a standalone without the need for any setup other than the prerequisite node.js install. Paths to node and bower packages along with file system operations needed to be relative to the application install path. Furthermore, the standard Electron package could not be used to distribute as a standalone application. I had to import a non-standard package and create a **build** script which runs on the target system when it is installed.

Learning and using **Electron** was challenging as it is quite new and lacking in online documentation other than their own API reference which did not apply directly to my implementation. I had to create and modify other people's libraries to integrate Electron and angular. Giving the conventionally client side library, access to node and native operations was difficult to get my heard around initially.

Persistence using JSON, while a good idea to keep the dependencies light (no need for a database server), meant that the data needed to be manually extracted and formatted. The angular service to save and load the JSON snippets got a bit messy and I was going down the road of making my own DBMS before I had to reign it in and cut back the functionality (Save/Load).

Scaffolding proved difficult in that files needed to be checked for and created. The user input from the CodeMirror panels needed to be extracted and turned into a string which could be written to a file. The interaction with the host system via the directory browse dialog was also a challenged as the API reference provided by electron did not apply directly to my angular implementation so had to be experimented with the get it working as expected.

FUTURE WORK

I initially expected to have the possibility to save and load workspaces, however, the work required to provide this functionality far outweighed the value of the feature. This would be nice to have and would add value in the future though but would require a much more complex data storage solution other than the basic JSON that is currently used.

Integration with Gist and other online code sharing platforms or social media would also be valuable.

Overall I think I will develop this further in the future and perhaps make it into something I use regularly.

KNOWN ISSUES

Global npm installs on OSX fail to add nwp to the path. Appears to be an OSX issue unrelated to program code but rather host system node setup.

INSTALLATION

Install globally with npm using this command:

npm install -g nwp

Note: *node.js installation is a prerequisite*

RUN

After being globally installed, if environment variables are set up correctly, the application will be accessible simply by typing the following command:

nwp