**DaBL Administrative Application Developer Guide**

11 December 2019

American University

|  |  |
| --- | --- |
| **Version** | **Date** |
| 1.0.0 | 12.9.2019 |

**TABLE OF CONTENTS**

**Electron Overview 4**

**DaBL Application overview 6**

**Code Layout 7**

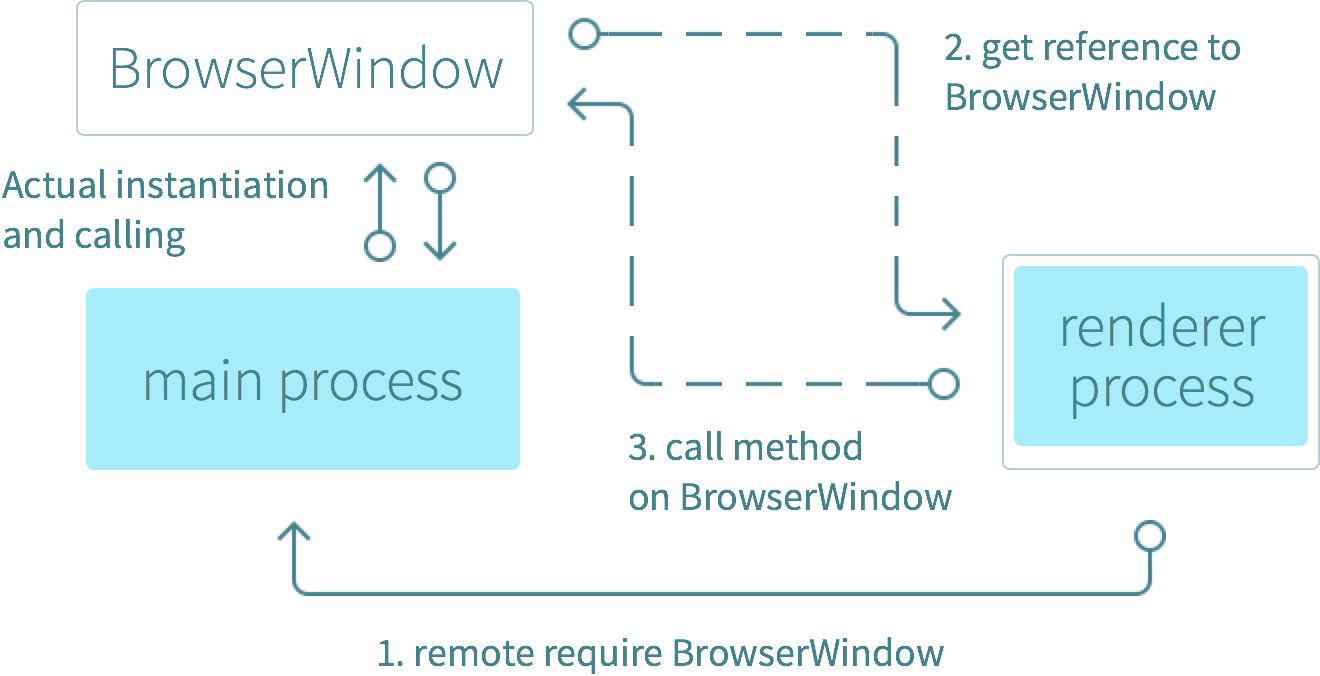
**Example Code Change 9**

**Summary 10**

# *Electron Overview*

The DaBL Administrative Application is an *Electron* application that is run locally. As described by *Wikipedia*:

*“Electron is an open-source framework developed and maintained by GitHub. Electron allows for the development of desktop GUI applications using web technologies: It combines the Chromium rendering engine and the Node.js runtime.”*

Because it is based on web-based technologies, it is a framework geared towards allowing web developers apply their skillset to create desktop applications. The primary programming languages employed are *HTML* and *javascript* for the front end, and *node.js* for the backend. The application communicates in the three different areas as show below. These are the *main process, renderer process,* and the *BrowserWindow*. 

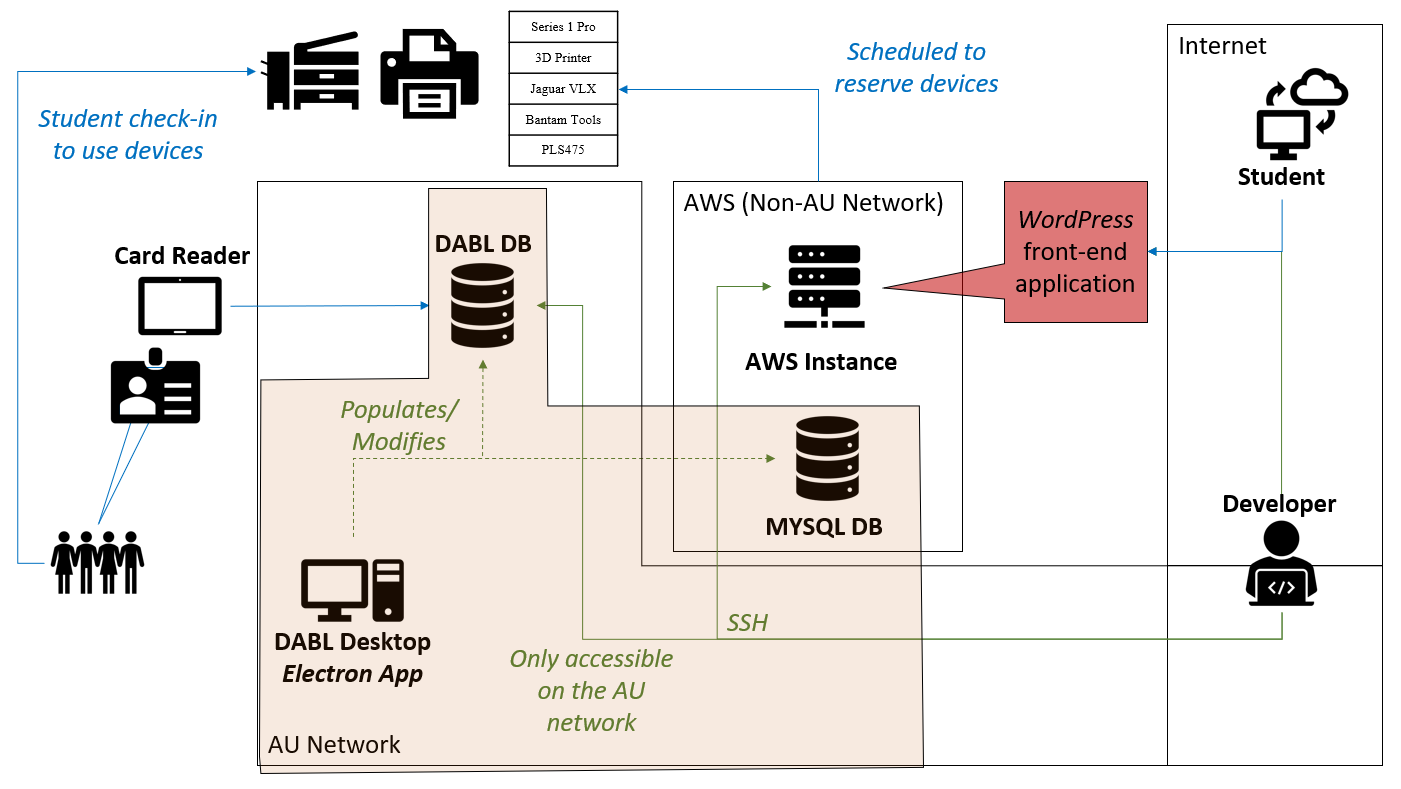
Generally, this can be viewed that the *main process* kicks off the *backend*. This should be viewed as the area where the *Electron* application can talk to the native system to access files and interact with the computer at a higher privilege. This process is the entrance to the computer system. Conversely, the front-end is also kicked off by the *main process* but operates at a lower privilege - like a normal browser-enclosed web application would. *Electron* renders HTML files for view in its *chromium* browser application. The *renderer process* coordinates and communicates between the front and backend. So here is an example:

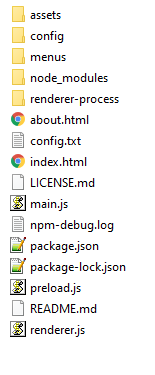
* *Front-end*: User selects an option that calls a file located on the local desktop, and clicks **submit**. This button is linked to a function in the *renderer process*.
* *Renderer Process*: Handles - using *Electron* *node.js* libraries - the submit features selection. Communicates the feature to the *main process* providing the request to the main process where higher privileged tasks can be carried out.
* *Main Process*: Receives the request and opens a text file on the Desktop, reading in the contents. The main process then sends the content back to the renderer process to display in the front-end browser.

# *DaBL Application Overview*

The DaBL Administrative Application is intended for easily and quickly populating the DaBL database. Moreover, it is intended to populate and modify the local DaBL database as well as the sanitized AWS DaBL database which supports the DaBL Event Calendar on the DaBL website. This allows users to select and schedule DaBL tools and devices ahead of time.

It also authorizes the user based on two factors. The first is that they have been placed into the DaBL user database (both locally and on AWS), as well as that they have an American University (AU) gmail account. The AWS user database provides the Event Calendar application the ability to check what certifications the user has in order to authorize them to schedule certain devices. The second benefit refers to AU gmail authentication is that it affiliates the user with AU and automates the linking of the calendar to their gmail. Ideally, the administrative application will populate both databases, abstracting the concept of dual interaction for the user, and make it easier for the person tasked with inputting new users.



***Code Layout*** 

The following sections describe the main files you will interact with and their purposes separated between each of the separate components of the application.

**Front-end:**

* **index.html**
  + The main page where all sub-pages are called, and all *assets* are imported.
* **about.html**
  + The front-page where the *Enter* button brings the user into the application. Also lists the *DaBL* site information
* **menus/menus**
  + The drop-down menu for *Creating New Users.*
* **menus/modify**
  + The drop-down menu for *Modifying Users.*
* **menus/config**
  + Informational area that displays the *MIT license*, and would be a good spot for any database or system configurations needed.
* **assets/\***
  + These are all associated *css* and images that may appear in the application

**Renderer Process:**

* **renderer-process/\***
  + These are the *javascript* and *node.js* files that correspond to each *HTML* file.
  + One of the primary classes to further research is the *interprocess communication* of *Electron* apps. These are proprietary libraries that perform *renderer-main process* communication.

**Backend:**

* **main.js**
  + This is where the *Electron* application is started and instantiated.
  + The lower half of the file consists of the MySQL code where the *renderer process* has sent a request to the *main process* to query a MySQL database. Would recommend separating this portion out to its own file later.
  + The MySQL connection configuration is hardcoded into this file. Recommend identifying a configuration strategy to separate it from the process.
* **package.json**
  + This file also supports the instantiation of the *Electron* application and contains dependencies as well.

***Example Code Change***

The most likely scenario will be the addition of a new tool to the lab. The DaBL tool is added into the MySQL database, where it will no longer function with the Administrative Application. The steps you would take to fix it are as follows:

* Add the new tool name to the database of choice
* Add the tool name as a checkbox in the **menus/menus.html**
  + Make sure to add a unique ID that will be referenced by the DOM:
    - <label><input type="checkbox" id="newtool-createnew" value="">New Tool</label>
* In **renderer process/context\_menus.js**, add the new tool to the bottom of the list where variables are being referenced:
  + const newtool = document.getElementById('newtool-createnew')
  + Follow the **\*.js** file down and confirm that the variable does not need to be appended or added where similar variables are acted upon prior to moving on.
* In the **main.js**, assign the variable to the arrays that are associated with the *interprocess communication* libraries in the ipc class. The ipc communication receives the query request and facilitates the MySQL query. Once a new variable is assigned, follow the programming thread to identify where it needs to be again implemented:
  + var newtool = new\_user\_array[15]
* Notice that the MySQL query strings are hard-coded, this means that you will have to add the database name to the query string:
  + var q\_string = 'INSERT INTO local\_db (name, UID, UIDprev, membertype, email, membersince, lastaccess, waiver, series1pro, thr3dprinter, jaguarvlx, bantamtools, pls475, sewing, test, **newtool**) VALUES (?)';
* This must be checked for all MySQL functions that perform queries through the **main.js**.

***Executable Creation***

Download the *node.js* library Electron-packager. Next, in the command line, run:

electron-packager <target directory> <app name> --platform=win32 --arch=x64

Of note, this tool will deposit the executable in the directory you ran this command from.

***Summary***

Hopefully that provides some perspective on the application and the distinct interactive components that make function. It is a somewhat cumbersome process making sure all components communicate in order to perform a system-level task, but it brings the benefits of browser visualization.

For more information, check out the *Electron* website. And please feel free to reach out to the team with any questions or support that you need. Thanks again - and enjoy!