Datava Packaging : Final Report

Client : Datava Software Solutions

Team Datava-1:
Caleb Bartel, William Culver, Nathan Keith, Jacob Vossen
June 10th, 2020

Introduction:

Client Description:

Datava is a company that develops cloud-based software solutions mainly for consulting firms and financial institutions. These solutions include tools to manage interactions with customers, predict what factors will improve sales and customer loyalty, train and certify employees on various policies and products, and much more. They value solutions that are highly customizable, stable, and easy to manage so that they can meet the needs of their clients. All of the tools they provide can currently be accessed through a web application that connects to different databases for each of their clients.

Product Vision:

Create a packaging system for the frontend and backend that creates a more robust, manageable, and simple deployment and upgrade process. The front end packaging software will take their existing web application and move to a build system that allows targeting iOS, Android, and desktop. The backend system will take the application layer and enable deployment via Docker to become a more scalable and reproducible build environment.

Requirements:

Functional requirements:

- The existing database system will be containerized.
- All features in the existing database system will still work.
- The user will be able to access both remote and local databases.
- The user will be able to run the front end program as a desktop and mobile application.
- The front end program will remotely communicate with the back end program.

Non-functional requirements:

- The front end program will not require multiple loading screens.
- The front end program must work the same way on all platforms (Desktop, Web, iOS, and Android).
- The back end program must work on different configurations and operating systems.
- Processes should be functional scripts, not instructions/pseudocode.
- JavaScript code should be reviewed before full implementation.
- Version control should be used at all times.
- Front end packaging should use Electron and Capacitor software.
- Back end system should use Docker software.

System Architecture:

Overall Architecture:

As shown in Figure 1, the user can connect to the database system through the desktop electron application, the web application, or the mobile application. They can access local databases on their machine, or connect to the back end Docker application to use remote databases on Datava servers. The front end applications are set up using Capacitor, and Docker was used to package the existing database system, both will be explained further in the following sections.

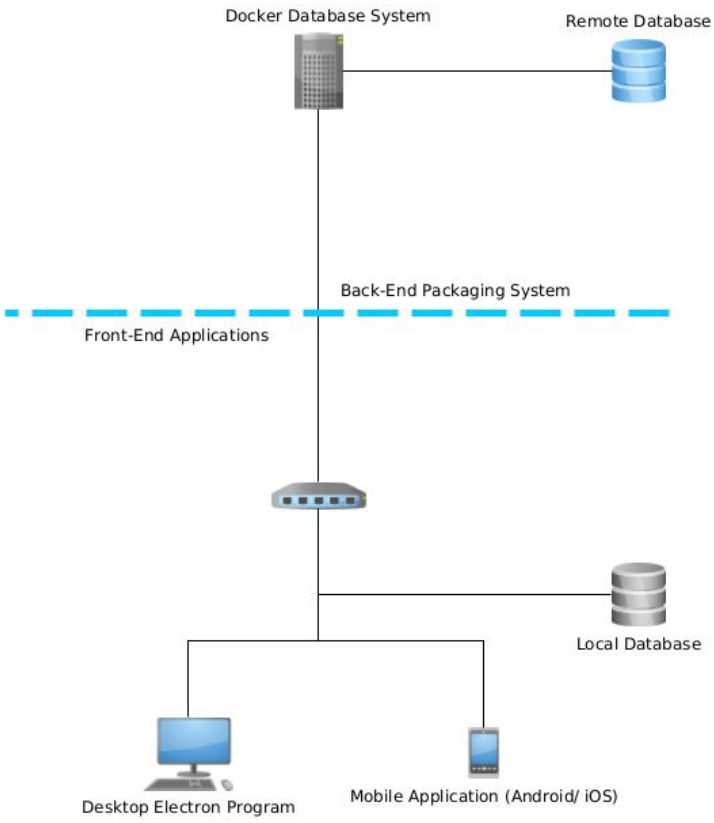

Figure 1: Overall Project Architecture

Front End Applications:

A Capacitor framework is used to deploy the Datava web application in conjunction with Android Studio to download the system as an Android Application, XCode to download the system as an iOS application, and we compiled the Javascript code with Electron to create a desktop application. The Capacitor framework and file structure was set up using command line scripts and the mobile applications were finalized and simulated by Android Studio and XCode. A software framework is essentially a toolkit that can be used to develop programs in different applications. Capacitor works as a mobile development framework in this situation. Mobile development frameworks basically convert code from common web languages like HTML and Javascript into apps that can run on Android and iOS. Datava already had a website, so creating a mobile app is as simple as dropping the existing source code into a mobile development framework like Capacitor. We tailored the application to meet our Client's specifications by editing the configuration files for Capacitor as well as some files in their code base. Some of these changes were purely decorative, such as the splash screens, while some were to fix issues, such as the status bar on iOS devices not being accounted for. Each application and what is used to create it can be seen below in Figure 2, and the application running on the three different platforms can be seen in Appendix 2.

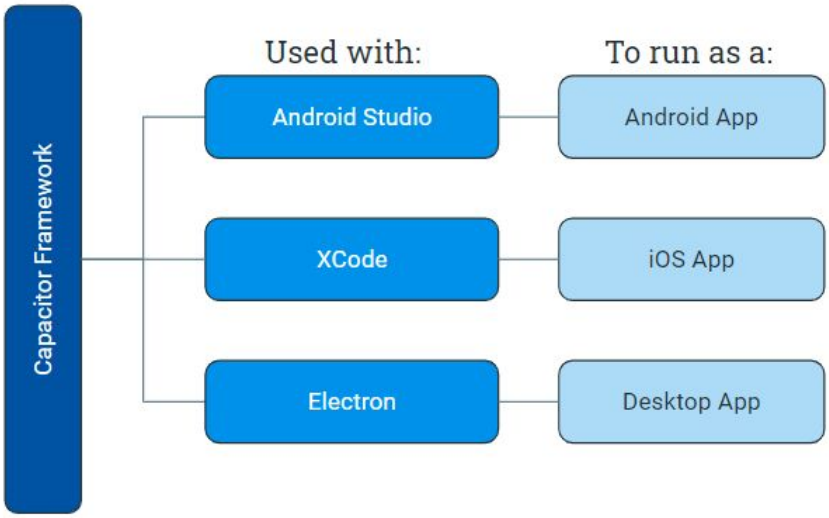

Figure 2: Applications and Programs to Create Them

Back End System:

Datava chose to migrate to Docker for the back end system because Docker allows applications to be packaged into containers that can communicate with each other. Each main process should have its own container so that updating a single process is possible. Each container has a Dockerfile that acts as a template for a process. Once this file is built, it becomes an image which is the packaged process but in a non running state. The image can then be run and it then becomes a container which is the process running. Networks in Docker allow for containers to communicate with the user and with each other in a specified way so that information is not shared to users or containers unintentionally. Docker volumes are just a way to store information for a container across system restarts since they map a local drive location to a location in the Docker container.

The main Docker containers for this project are Apache, PHP, and MySQL. The Apache container is the central container and it depends on the PHP and MySQL containers for system files and database information. The PHP and MySQL containers are part of the backend network, while Apache is part of the both networks. This allows users to connect to the Apache container but not directly access sensitive information from the PHP and MySQL containers. The PHP container has three volumes that link the root of the project to the Html container volume and other necessary system files to their correct locations in the Docker container. The MySQL container has a custom created volume called data that links to the MySQL system volume. These volumes allow data to be stored across system restarts. All of the components and relations can be seen in Figure 3.

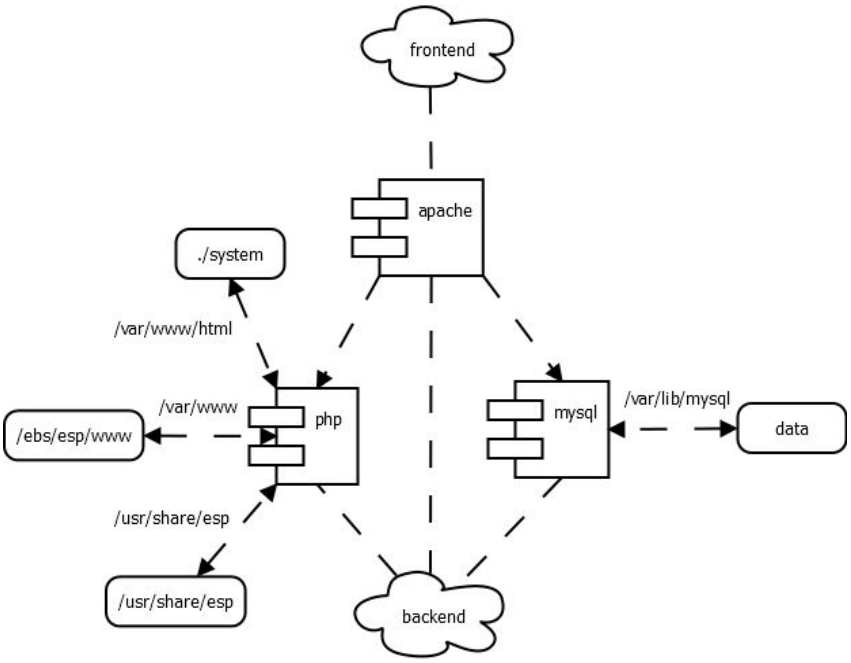

Figure 3: Docker Architecture

Technical Design:

Analysis of Capacitor:

Datava was originally using Apache Cordova as a mobile framework for their iOS and Android applications. Cordova is still a powerful framework, but Capacitor can do more and is a much more modern solution. As shown in Figure 4, Capacitor is not just a mobile framework. It can also create Electron and Web applications. Capacitor uses platform specific IDEs like Android Studio and Xcode to run apps rather than running them through the command line. Furthermore, it uses newer APIs than Cordova and runs more smoothly since Capacitor embraces native functionality rather than relying on abstraction. However, Capacitor cannot do everything since it is still relatively new. The Capacitor developers are still finalizing many common plugins and their relations to the different platforms, such as GeoLocation. Since Capacitor can be seen as unfinished, it could be argued that it is a risk switching to Capacitor so soon. However, Capacitor has already, in many ways, surpassed Cordova's abilities, and continues to receive updates to ensure stability and expand on its capabilities.

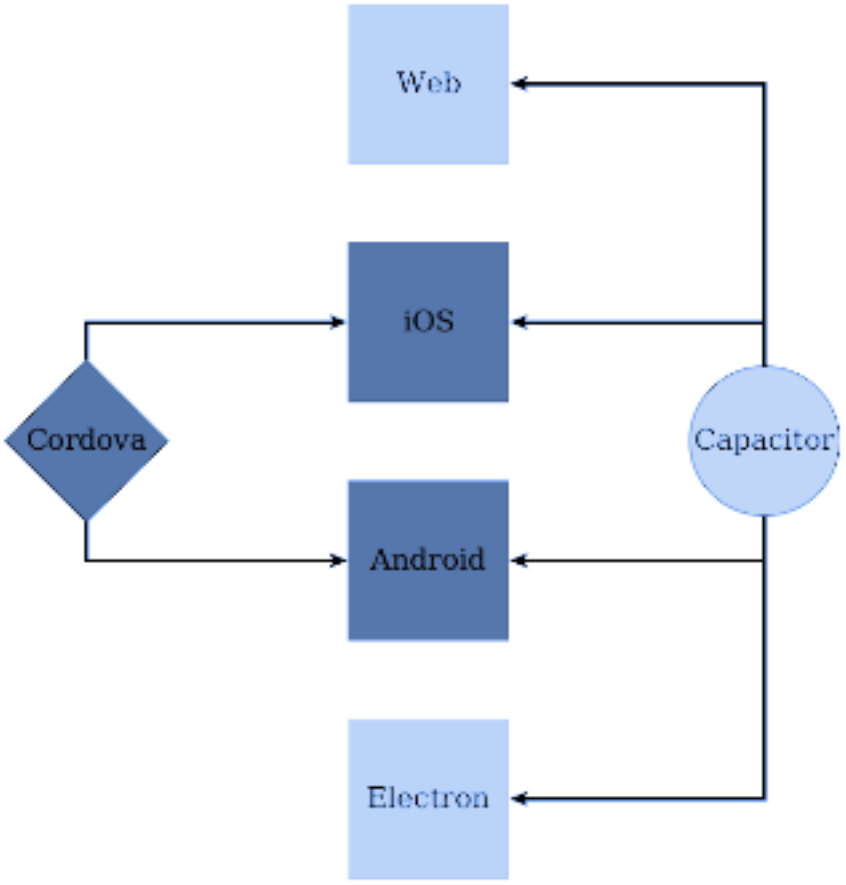

Figure 4: Cordova and Capacitor Capabilities

Analysis of Docker:

Before Docker, the Datava system had several install locations and systems. There were files in /var/www, /var/esp, /usr/share/esp and more. This meant that the installation and configuration on a machine were suboptimal. Using docker has enabled us to streamline this process into just one command to go from an SVN pull to a running system. Running the command “docker-compose up” creates three containers, two networks, and four volumes as seen in Figure 3 that hold all of the information and storage for the Datava system.

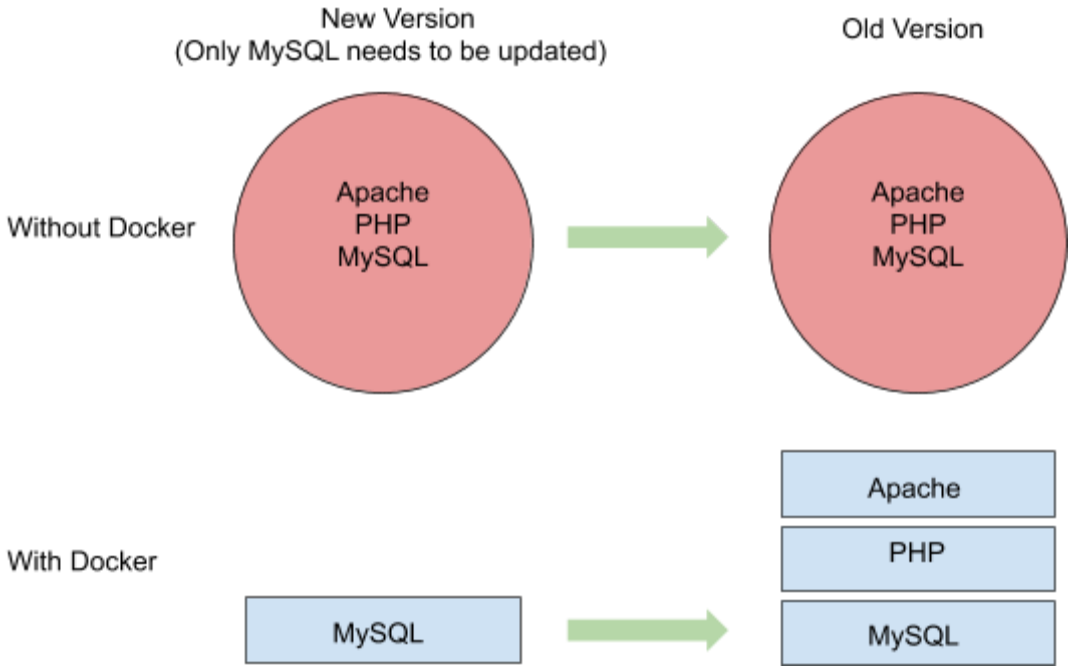

Figure 5: System Upgrade Process With and Without Docker

The previous system had the same components that it does now but they were all intertwined with little separation of components. This made it difficult to run the system and upgrade individual parts. But now with Docker, each component can be updated to a new version just by editing the Dockerfile specific to that container instead of having to push out a new version for the entire system at once. This change in the upgrade process is depicted in Figure 5, using the example of updating to a new version of MySQL.

Quality Assurance:

To ensure the quality of our final project, we have completed several activities between team members and with our client. A wide variety of tests were performed to ensure that each aspect of the requirements were fully accounted for and that we met all of our client-specified requirements.

Activities List:

- **Code Review:** All code written was reviewed by team members and some code was also reviewed by Datava employees to ensure completion, consistency, and cohesiveness with the current code base.
- **Documentation:** Detailed documentation was created for any processes that were unable to be scripted for future reference and debugging and to ensure the product is completed.
- **Run docker-compose up**: This command runs the entire system through Docker and tests that the system functions the same as it did without Docker to ensure the process runs correctly and efficiently.
- **Integration Testing**: All previously used plugins for Cordova were tested in the new Capacitor system to ensure that the equivalent plugins were properly integrated into the system.
- **Acceptance Testing:** All features were approved by the client before full integration into the code-base to ensure all client requirements and standards are met.
- **Platform Testing:** We ensured many features work and are consistent between the 4 platforms, web, desktop, Android, and iOS.

Results:

Final Project Description:

Front End: The goal of the front end of our project was to package the existing Datava web application into various platform applications that all had the same functionality. This was accomplished using Capacitor, replacing the Cordova system they had previously set up. Capacitor was used with other software, such as Electron, to ensure the existing code base was properly packaged.

Back End: The goal of the back end of the project was to take the existing server side stack of Apache, PHP, and MySQL and turn it into a containerized environment that could be deployed to multiple systems. This was accomplished using Docker, which enabled us to create a system where each of those technologies ran in an individual container, and a network defined connections between them. Docker was used as a cross platform technology that enables Datava to run this system on any target operating system.

Features Not Implemented:

- Integration: Progress was made on both sides of the project, but more still needs to be done to get the front-end program to use the back-end docker system.

Summary of Testing:

- Acceptance Testing: The product was constantly reviewed by our client for adherence to the requirements.
- Integration Testing: All plugins were fully integrated into the system except for geolocation which needs larger code-base changes.
- Platform Testing: All features work properly on all 4 versions of the application.
- Code Review: The code was commonly created and debugged with the help of several Datava employees.

Lessons Learned:

- Working on an unfamiliar system can be confusing and challenging: Both the back-end database and the front-end platform were mostly complete when we started on the project, so we had to learn how to decipher large sections of unfamiliar code and the basics of Javascript and Docker. Creating Docker containers on an existing system is difficult because the files need to be arranged in a specific way so they are in a container if they need to be accessed.
- Working from home brings many unique challenges: One of the core aspects of teamwork is communication, which was often difficult in a remote environment. Delegating which tasks would be completed individually and which tasks would be completed in groups was often difficult. It was also difficult to ask team members and the client quick questions, and likewise, give acceptable answers. Another challenge with working from home was needing to remote into virtual servers and desktops. Both the front and back-end teams were required to use unfamiliar, unpolished remote tools which led to small delays.

Appendix 1: Docker Setup Documentation

The file paths and steps taken listed in this document are specific to the current Datava system and would vary if implemented on another system.

Setup:

- Create a docker directory for the system files and Docker containers
- If copying the system files, make sure there are no symbolic links to files outside the docker directory context
    - system/js/ext and system/js/ext/ext-6.2.1 should be copied directly from /ebs/esp/www/ext/ext-6.2.1
    - system/js/tablesort.js should be copied directly from system/js/core/tablesort.js without link if needed
- Inside the docker directory create directories for the apache and php containers with a Dockerfile for each
- Optional: create a mysql directory for imported database files and configurations if there is a desire to create a separate mysql container
    - Export the external mysql database using the command: mysqldump -u username -p database > file.sql
    - Import the mysql file into the Docker container by logging into the container, creating a database, using the database, and running the command: source /path/to/file.sql
- Create a docker-compose file in the docker directory for each of the containers
- Copy necessary files outside the system directory into the docker container
    - Option 1: Copy files directly into the container and change paths in system/espaux/espConfig/config.cfg.php
    - Option 2: Use a volume to map /ebs/esp/www to /var/www
    - Option 3: Run a command in a Dockerfile to copy files from an svn or git repository upon building

Known Issues:

- The icons are missing from the web application when logging into the external demo database on the dev.datava.com server and the internal container database

Appendix 2: Application on All Platforms

The examples below show the front-end program running on the three main platforms we developed on.

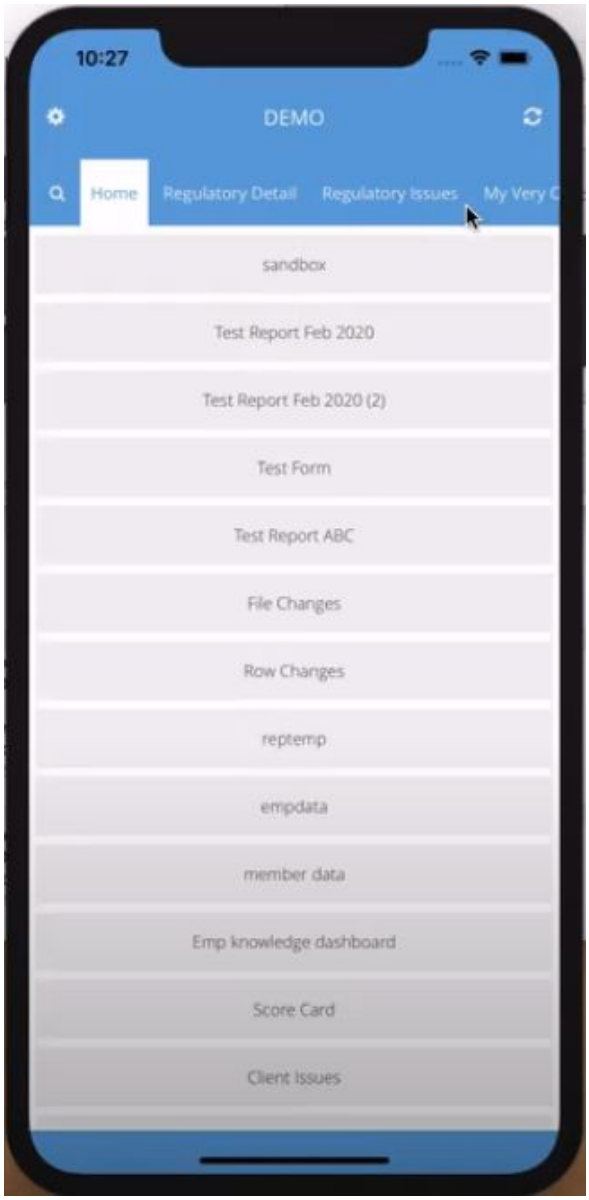

Figure 6: Screenshot of iOS Application

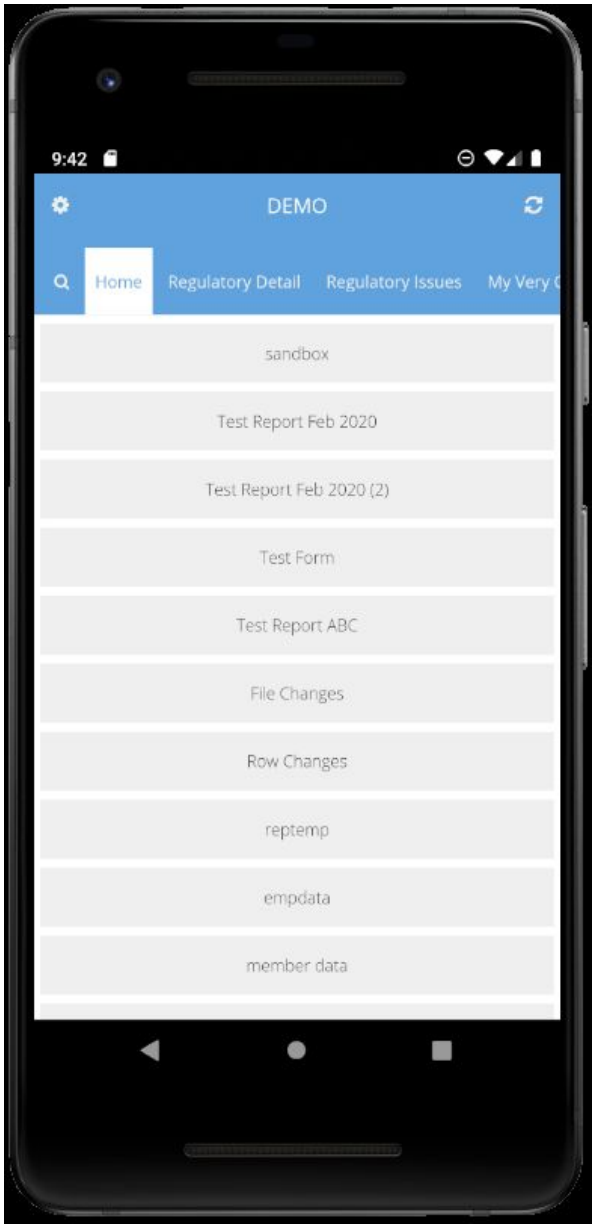

Figure 7: Screenshot of Android Application

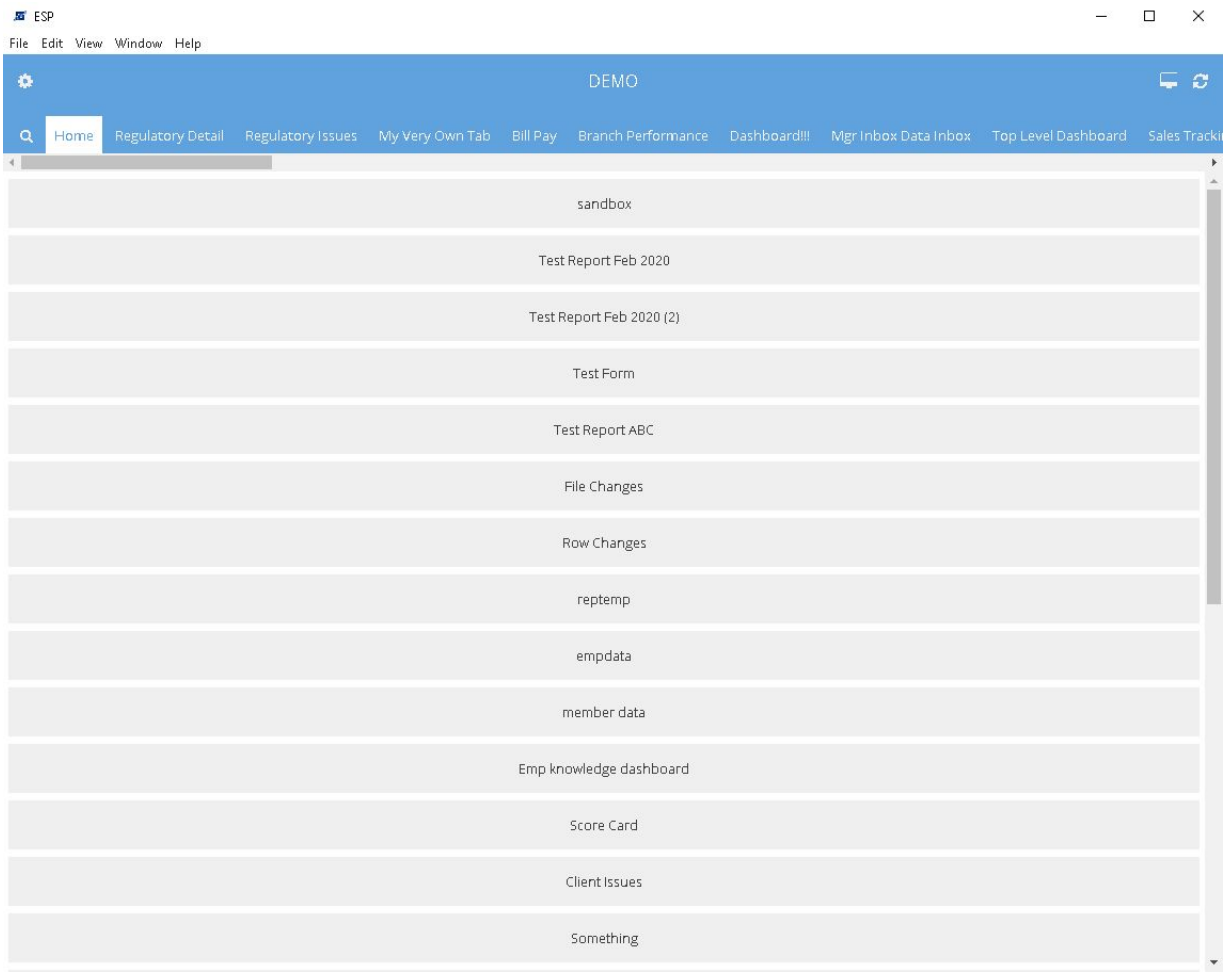

Figure 8: Screenshot of Electron Application