Data Verity Final Report

Team Data Verity #1

Griffin Ciluffo, Daniel Herman, Stephen Tracy

Client: David Flammer

21 June 2016

Contents

1 Introduction

Data Verity builds customized data analysis graphs and tables for its clients, allowing them to see important data and trends. The data is gathered, analyzed, and displayed in a useful way which allows the customer to view all important information through a desktop application. The purpose of the project was to work with an existing code base for a mobile version of the desktop application to make it contain all the same functionality as the desktop application. The first task was to update the JavaScript web framework used by the app, ExtJS, which is created by the company Sencha. After that, the display of charts and graphs had to be working, and then other features such as forms could be implemented. Throughout the process, it was important to ensure that the mobile application's layout and styling was appealing and matched that of mobile applications currently on the market.

2 Project Requirements

2.1 Functional Requirements

The functional requirements of the project were to upgrade the JavaScript framework of the original mobile application, update the mobile version to include the same functionality as the desktop version, and add features that would serve to make the app feel like a mobile app as opposed to a desktop app being displayed on a mobile device. Before features could be added, the JavaScript framework had to be updated from Sencha Touch 2.3/ExtJS 2.3.1 to ExtJS 6+. After the framework upgrade was completed, desktop features were implemented on the mobile version. One necessary feature was ensuring charts, graphs, tables, and reports display properly on mobile devices and are easy to view, i.e., has touch-screen zoom functionality. Other features included implementing communications-related forms for the user. The desktop version has forms that allow the user to send communications and a table that tracks communications within their company; these features had to be implemented on the

mobile version. To keep the mobile app feeling like a mobile application, zoom and swipe functionality also had to be implemented. Finally, once the framework was updated and the features implemented, the application had to be deployed to iOS and Android app stores using Apache Cordova and a document needed to be produced outlining the deployment process. The production of the document is necessary so that Data Verity can follow the same process in the future when they make further upgrades to the mobile application.

2.2 Non-functional Requirements

Non-functional requirements included preserving desktop functionality, using Subversion for version control, and updating the user interface for the mobile version. While upgrading the mobile version and adding features, it was occasionally necessary to modify files that both the mobile and desktop versions use. However, the modifications were not allowed to disrupt the functionality of the desktop version of the app. Additionally, Subversion had to be used for version control since Data Verity uses subversion and the code base was given to us as three separate branches from their Subversion version control. The final non-functional requirement was updating the appearance of the mobile version to make it more comparable to mobile apps currently on the market. For example, the user interface had to be mobile-friendly in appearance. The layout of the app also needed to be changed from displaying large buttons to displaying icons in a grid, similar to many current mobile apps. Finally, charts and graphs needed to look good on mobile devices and support zoom functionality.

3 System Architecture

The project's architecture has a heavy back end created in PHP with a modular front end made in JavaScript utilizing the ExtJS framework. The user logs in to their account on the mobile version and

sees every chart, graph, report, and form that exists on the desktop version. This is due to the fact that the mobile version and desktop version use the same back end and gather data from the same server. Due to the desktop version using the same back end as the mobile version, the back end was treated as a black box that was not allowed to be changed. The application uses AJAX to make server requests. The server responses are returned as JSON strings which had to be parsed, analyzed, and occasionally changed in order to obtain the desired functionality on the mobile version. The user is also able to edit the data shown by adding, deleting, or modifying the existing data. A flowchart of the interactions between the application and the server can be seen in Figure 1 below.

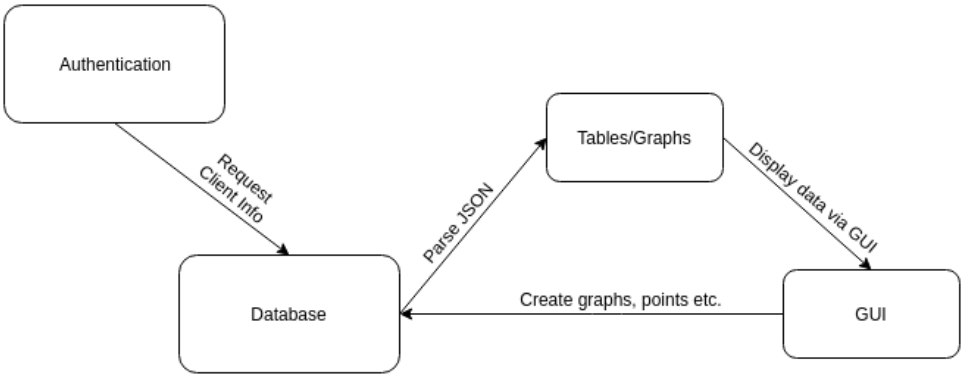

Figure 1: Flowchart of System Architecture

4 Technical Design

4.1 Upgrading to ExtJS 6

The very first task was to upgrade the old mobile application to the latest version of the ExtJS framework. The upgrade was necessary to allow the latest charting API to be implemented which is much more robust. Sencha, the creator of ExtJS, provides detailed documentation for upgrading via the tool Sencha Cmd. However, difficulty arose because the code base was not local; it is located on the clients server which had to be accessed via SSH. This introduced problems because the work was done

in the Alamode Lab at the Colorado School of Mines where students are not allowed to freely download and install software. After a week, most of which was spent attempting to learn the code base, the latest versions of ExtJS 6 and Sencha Cmd were found online and uploaded to the server. By placing the files inside of the app folder of the file structure and installing them to the server, the command sencha app upgrade automatically updated the relevant ExtJS architecture. It was necessary to ask the client to upgrade the servers version of the Java Development Kit, because the upgrade process was getting interrupted due to the JDK being out of date. After the app was upgraded, most features broke on the application. The client decided that the most important task was to get the existing code working, as well as import the new charting API. The upgrade introduced many app-breaking bugs which will not all be mentioned due to the fact that the bug-fixing process is similar for all bugs. Instead, three bugs will be mentioned in the following subsections. These bugs were chosen because they are representative of the type of bugs that were fixed throughout the project, as well as showing the sometimes complicated and sometimes frustratingly easy nature of the fixes.

4.2 Duplicate Entity Error

One error introduced by the upgrade involved the attempted defining of already-defined models. This error is an example of a simple fix because it could easily be isolated using the Chrome Developer Tools which allowed for a quick fix. Unlike the old version of ExtJS, the new version does not allow redefinition of models. This was occurring in several files: MobileActionMenu.js, MobileFieldEditor.js, and MobileAutoPageGrid.js. The error was fixed by checking for whether or not ExtJS recognized the class names as already being defined. An example of this fix is shown in Figure 2. It simply checks whether or not the model is defined before proceeding to define it.

```
// check if the model already exists, otherwise will result in duplicate entity error
if(!Ext.ClassManager.isCreated('ListItem')){
   // console.log("duplicate");

Ext.define('ListItem', {
    extend: 'Ext.data.Model',

    config: {
    fields: ['text',
         'itemId',
         'buttonType',
         'viewID',
         'scope',
         'fn',
         'action',
         'handler',
         'htmlPopup',
         'iconCls',
         'tooltip',
         'xtype'
        ]}
    });
    }
```

Figure 2: Example fix for the duplicate entity error.

4.3 Scrolling Bar

There are several table views within the app that display a grid of data. Many views have multiple pages which make a scrolling bar and next/previous page buttons necessary for navigation. The scrollbar at the bottom of the screen (see Figure 3) which should change the displayed page had several issues. The first issue was that the arrows to the left and right of the scrollbar were not properly graying out and being disabled when at the first or last page. The next issue was that once the last page was reached, page transitions broke entirely and the table had to be refreshed to navigate again. The last issue was that clicking or tapping on the scrollbar did not properly change the page, so a user would have to drag the knob to get to a certain page which can be difficult when there are 500+ pages of data.

The root of these issues was discovered in PagingToolbar.js where the functions were incomplete or using the wrong parameters which was due to the upgrade to ExtJS 6. One ExtJS function had incorrect logic, requiring that function to be overridden. The most interesting change came from the function updatePageButtons which was changed to get the current page, determine whether or not the

page is the first or last page and disable/enable the left/right arrow keys appropriately. The parameter changes were simply updating a line in several functions to use a different parameter in the function. In the previous version of ExtJS, the page number was passed in; however, in ExtJS 6 the page parameter was changed to an array of pages which is partially what broke the scrollbar. Figure 3 shows the final scrolling bar, with proper graying out, disabling of arrow keys, and knob position. This error was not as simple to fix because isolating it with the Chrome Developer Tools, given the complicated code base, was almost impossible. By researching online it was discovered that the file was a plugin. The PagingToolbar.js file was eventually found deep in the large file structure of the code base. From there, the Chrome Developer Tools were used to show what functions were called during the execution of certain actions and ultimately the functions were fixed.

Figure 3: Figure showing proper functioning of the scrollbars.

4.4 Mobile Dashboard

The mobile application has a dashboard at the top with tabs which can be selected and contain different menu items. In the desktop view of the app, the dashboard displayed correctly. However, viewing the app on a mobile device made these tabs disappear which is a significant functionality drop causing only the first tab menu items to ever be displayed. It took a large amount of time to determine how/where in the code these tabs were actually being created because there are so many different elements to item creation sprawling across many different files. Many hours were lost trying to alter the code in Dashboard.js where the tabs are created. However, the root issue was actually occurring in Main.js where the Dashboard is created. After reading ExtJS documentation and forums online, it was discovered that there is a bug within ExtJS that does not auto size the tabs correctly on mobile. Because of this, the height has to be set manually. This could be fixed as newer ExtJS versions come out, but for now the height is set manually to 60 pixels which gives a 10 pixel buffer to the top and bottom of the 40 pixel tall tabs.

Additionally, once the tabs were displayed they were cut off and reaching the far right tabs on mobile was not possible. This was fixed by adding a horizontal scrollable property during tab creation along with the height. Figure 4 shows the application with the tabs at the top not displaying versus correctly displaying.

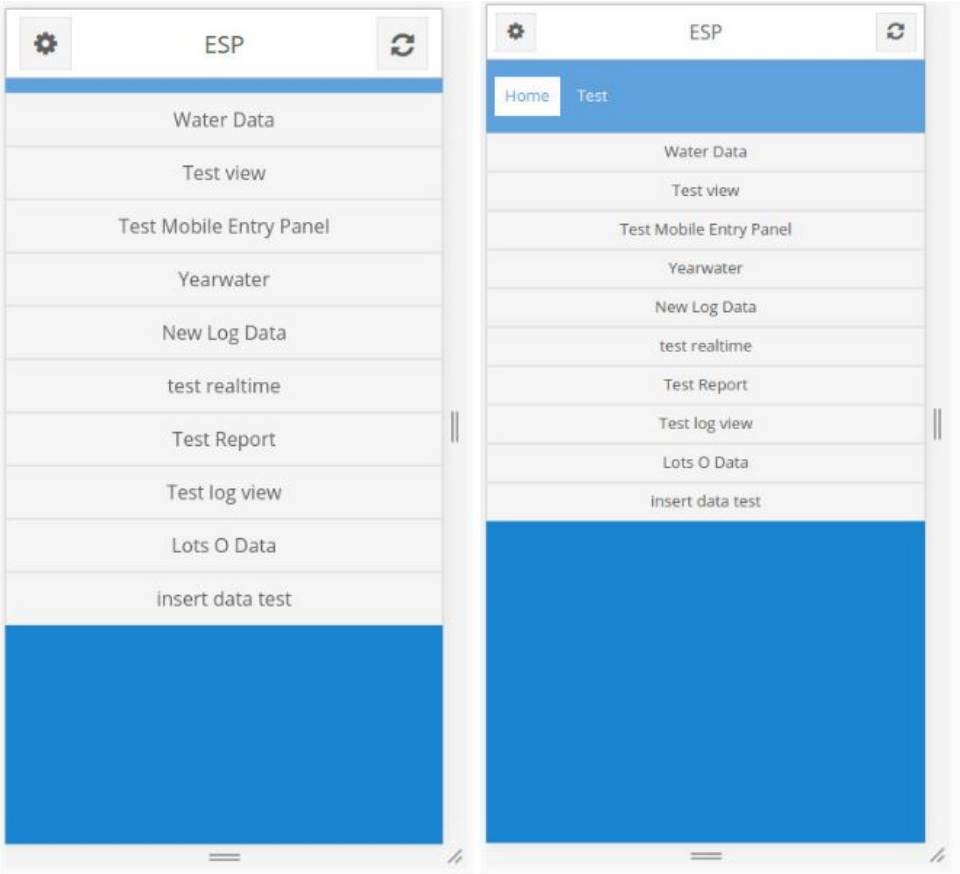

Figure 4: Shows the tabs hidden on the mobile view as well as what the app looks like after the fix.

4.5 Mobile Entry Panel

The majority of the project consisted of ensuring stability by fixing bugs introduced after upgrading to a new version of ExtJS as requested by the client. However, there was sufficient time at the end to add some functionality to the mobile version that was in the desktop version. One of these features was the Entry Panel used for communications. The desktop version can be seen in Figure 5.

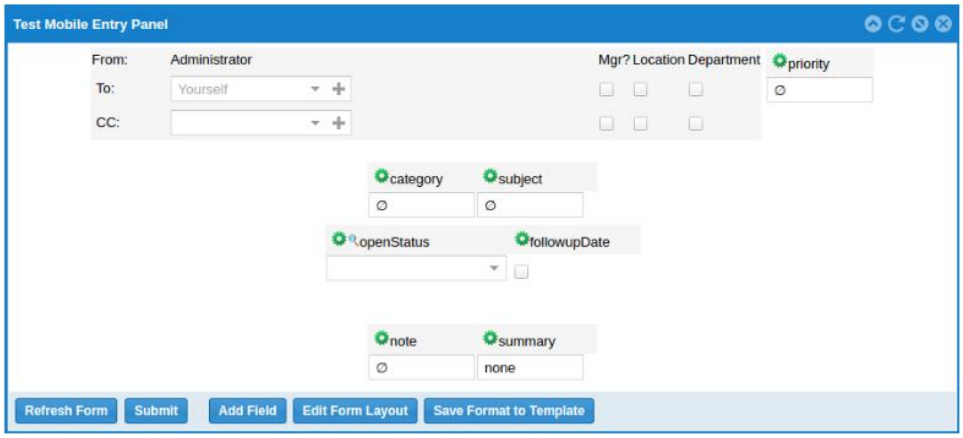

Figure 5: Desktop version of the entry panel.

The layout of the Entry Panel in Figure 5 would not work on a mobile device due to the screen size. Thus it was decided to translate this into a more mobile-friendly form with the entries stacked in a scrollable column. A significant portion of the desktop code could be used for the mobile part. Changing only a few places to access mobile classes instead of the desktop classes allowed the form to display properly. However, the mobile classes do not use containers in the same way as the desktop version, so none of the fields worked initially. The desktop version is able to access specific fields by referencing HTML Div Tags set by the container, but the mobile version needed to get the fields by name since it does not use the same container scheme. After stepping through the desktop version in the debugger, the name of each field was acquired and the various areas where fields were being accessed were changed to access fields by their name. The code controlling what happens when the various checkboxes are toggled had to be changed as well. These functions have a specific order for displaying other fields which can be seen in Figure 6.

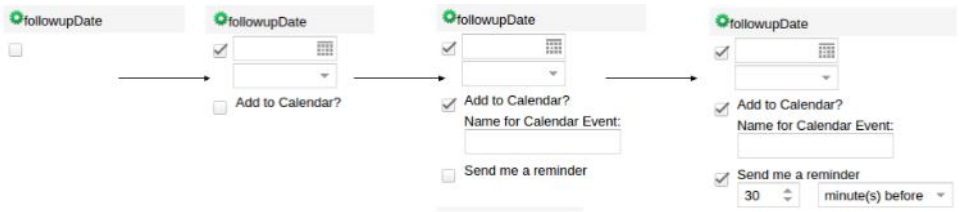

Figure 6: Shows the flow of displaying fields based on checkbox statuses.

For this flow to work on the mobile version, all of the hidden fields are created and then hidden. This is necessary because the desktop side hides fields based on a property that does not exist for the mobile version. The toggle functions for each checkbox call each successive toggle level each time a check box is checked. This allows for the application to keep track of which fields are 'displayed' even when the parent field is hidden. For example, the application will remember that the reminder field is being displayed (but will not actually display it) even if the calender field is hidden. There are checks within the functions for previous levels, so if all boxes are checked and then the first checkbox is unchecked, everything will hide.

The final technical challenge of implementing the Entry Panel on the mobile side was loading the store of names that the very first field pulls from (see Figure 7). On the desktop side, the information is pulled only when the dropdown field is opened for selection. The purpose of this is to save resources when first loading the page. However, the mobile checkboxes for ExtJS do not allow interaction if there is no initial store of data, so clicking on the field to pull information is not a feasible option. Thus the code had to be altered in one of the store loading files to check if the field was mobile and, if it is, set the store to be loaded upon entering the panel. This slows down the mobile application slightly, but currently there is no workaround due to the nature of the checkboxes for the mobile version.

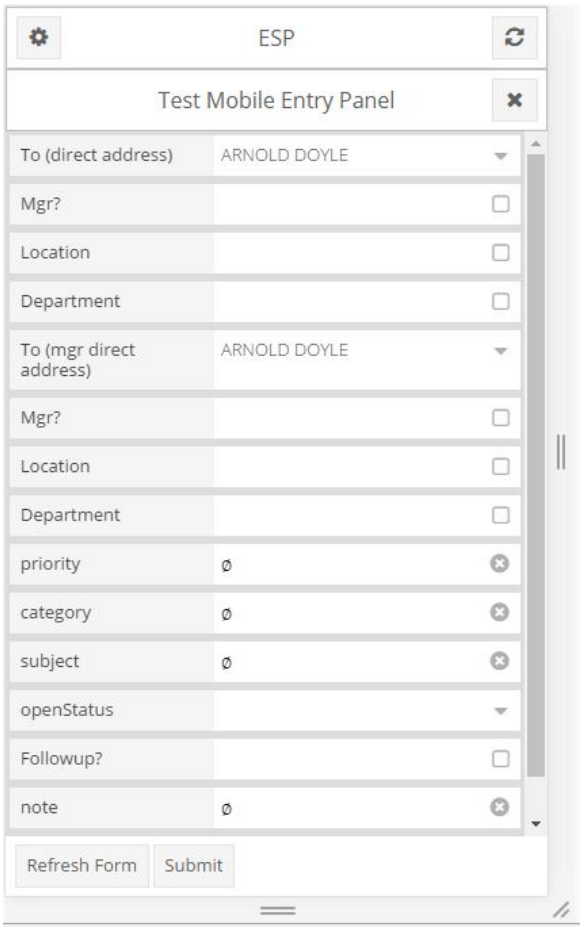

Figure 7: Final version of the Mobile Entry Panel.

5 Design Decisions

Several important design decisions were made based on various factors. These decisions include the form layout, dealing with a text field not displaying properly, opening the default number pad on a phone, zoom functionality, and opening reports in a separate window. The first major design decision was deciding not to change the layout of forms from the default layout. This decision was made because the default layout works for mobile so there isnt a convincing reason to change it. Additionally, updating the forms so they match or are similar to the layout of forms on the desktop version would appear packed on the small screen of a mobile device.

While the packed screen could be fixed by implementing zoom functionality, it was decided to not implement mobile pinch-zoom functionality. This decision was made primarily due to the fact that the ExtJS 6 framework does not natively allow for this particular feature, which would mean the only way to get the zoom functionality working would be to open the forms, charts, etc. in a new window. This option was decided against because the client did not want a multitude of new windows being opened for so many features. Opening many windows would lead to frustrating clutter the user would have to deal with. The feature of the app that displays reports to the user implements the method of opening a new window which does produce the desired zoom functionality. Given more time, there would have been a greater focus on implementing zoom functionality in some way other than opening new windows since it is so important to mobile applications.

Another design decision was changing the background color of a popup that appears when the user is locked out of the app. The background color change was necessary because the password text field did not display a border, making it impossible to tell there was a text field there without randomly tapping on the screen. Due to what is most likely a bug within the ExtJS framework itself, the border could not be displayed for a password field, regardless of setting the necessary properties; changing the color of the background was the only feasible way to make the text field obvious.

With the upgrade to ExtJS, the theme for the mobile app was updated which helped make the app more visually appealing. Due to the updated theme, time constraints, and unforeseen complications with updating the layout, the current app layout was determined to be sufficient. This resulted in the decision to not update the layout of the app to be more similar to current mobile apps on the market.

Finally, the mobile app prior to the ExtJS 6 framework update opened a number pad for entering numbers into a number field. The code for the number pad was third-party code that did not work with ExtJS 6. Rather than attempting to implement an updated version of the number pad, the decision was made to simply change the type of the field to open the mobile devices default keyboard for entering numbers. This decision was partly due to the fact that the best fix to get a number pad

opened up the telephone keyboard on mobile phones, which did not have a decimal point key that could be used for entering decimals.

6 Results and Lessons Learned

6.1 Results

After the upgrade to ExtJS 6, the mobile application runs faster and smoother. The user interface is more responsive. Additionally, the upgrade changed the theme of the app, making it more visually appealing. The loading icon was also changed to an icon which places less of a load on the system, further speeding up the loading process of the home page. While no in-depth testing was done of the application due to time constraints and the complexity of the application, results from performing certain actions on the mobile app were compared to results from performing the same actions on the desktop version to ensure functionality was the same. Throughout the process, the client also gave feedback to help keep the project on track and make sure all features were working as intended.
Future work that there was not enough time to do includes adding pinch-zoom functionality for mobile devices, updating the layout of the app itself to be more like a typical mobile application, and updating the layout of entry forms to be more visually interesting and user-friendly as opposed to the current layout which can be seen in Figure 7.

6.2 Lessons Learned

This project taught some valuable skills that can be applied to any web programming project. The major skill developed was using the Chrome Developer Tools. Without these tools, it would be almost impossible to find where code is breaking in a language like JavaScript that does not follow linear code execution. Utilizing breakpoints to step through code and see the current state of function calls and the

contents of variables was necessary, as well as the various listeners available, e.g., mouse click listeners. The HTML element selector helped with debugging UI issues by displaying the HTML code for any element the mouse cursor hovered over. The tools network activity panel helped with debugging back end issues such as server requests and empty responses by comparing the broken communications with working ones to identify issues.

Learning to use the ExtJS documentation was immensely important in finding what changed after the upgrade of the framework. Navigating documentation and applying what was found to observed behavior is a skill that can be used with any framework.

Becoming acquainted with JavaScript object declarations and how they are used in the code base was difficult but important due to how different JavaScript is from most languages. Additionally, since this application has a heavy back end, learning to modify and parse JSON responses was crucial to understanding what was going on with server requests and locating specific lines of code where things on the front end were breaking. Lastly, being patient is a good skill for debugging any application. Bug fixing is not always a smooth process, especially if you are dealing with a large code base that is unfamiliar. Its good to take breaks often and not be so serious otherwise debugging will be a very frustrating process.

7 Conclusion

Overall, the project was successful and the client was satisfied with the end result. It was not expected that the upgrade to the framework would break a large chunk of the code base, but adapting to such conditions is important in being a professional programmer. After discussing what the upgrade did to the system, the client expressed the most interest in correcting what was broken rather than implementing new features right away. Finding bugs and fixing them was a fairly slow process due to the unfamiliar code base and not knowing the framework or Chrome Developer Tools, but the team

grew very comfortable with the file structure toward the end of the project. The upgrade did allow the new charting API to be imported, which is a vast improvement to the existing charts. The experience gained from this project is invaluable to the careers of each team member.

8 Appendix

8.1 Apache Cordova Process

Note that the deployment process is only from experience deploying the app to Android using Apache Cordova due to software limitations on the Mac that was used, making it impossible to attempt deploying the app to iOS as well.

1. Ensure the latest version of Cordova is installed.
2. Run the command 'cordova create test' from the mobile app's root directory (e.g., mobiledev/ for this project), where test is a directory name. The name does not matter.
3. Change into the directory created via the above command.
4. Run the command 'cordova platform add android –save' and 'cordova platform add ios –save' to add iOS and Android platforms.
5. Run the command 'cordova requirements' and ensure all requirements are met for each platform. Cordova lists the steps to be followed. For Android, go to the Android SDK Manager and make sure that all the necessary API's are installed. Cordova will tell which API's are needed.
6. Run the command 'cordova build'.
7. Ensure that an AVD is installed for emulating the Android application. It may be necessary to set one up using the AVD Manager.
8. Run the command 'cordova emulate android' to open the emulation for the Android application.
9. Once the 'cordova emulate android' command has run, it is necessary to run 'cordova emulate' in order to emulate the app itself.