Tech Review

Mobile App for Forest Ecology Research

CS461 - First Term - Team Name: MAFE

Joseph Landreville

November 2019

Abstract

The purpose of this document is to examine the various technologies which may be used to implement the visualization of data on the Android devices our client will be using, and then make a decision on which one will be the best option to use for this project going forward. Rather than simply picking the first solution that comes to mind, we will do some preliminary research into which tools would best fit this project and provide the kind of visualizations required. At the end of the document I will examine some of the technologies which would enable us to realize our stretch goal of a form building application.

CONTENTS

1 INTRODUCTION

1.1 Overall Goal

The goal of our team's project is to modernize the electronic data collection system our client is using. Currently they are using nearly decade old handheld devices which suffer from poor performance and instability, and use proprietary software to manage the forms the field crews fill out to collect data. We will be replacing these with much newer Android tablets and will set them up to use the Open Data Kit X toolsuite for data collection. This should make collecting data much simpler and quicker while also eliminating the chance for data loss present in the old system.

1.2 My Role

My role within the project is data visualization. The Android devices our client will be using to record information from their research plots will have some part of the remote database stored locally in addition to the new data being collected while in the field. This data is made available for visualization by ODK Tables, which allows us to display data any way we want within the web framework it uses to render the interfaces. In addition to the data collected about the state of the trees we want to be able to be able to allow the field crews to easily be able to find the plot and tree they want to record data for. To accomplish this we would like to be able to display the locations of either all the trees stored on the device, or the plots which the trees are located in, on an interactive map which the user can scroll through.

For an additional stretch goal, we have also considered building a flow-chart style application which would enable more intuitive form creation. While the responsibilities for this application would be divided amongst our group should we have time to create it, I still think it is something worth doing some basic planning for. The current ODK-X workflow has you write the forms in an Excel spreadsheet, then run that file through a converter to create the ODK form definition file. This process seemed cumbersome to us and led us to consider developing a more intuitive method.

2 RESPONSIBILITIES

2.1 Visualizing Tree Data

One of the requirements for our application is to autofill the form for a specific tree with the information previously recorded for that tree. As such, each Android device must have some subsection of the information stored in the database so it can obtain these values. We will use this information to supply our visualizations with data. This data lends itself towards things like line charts which can show change over time rather than category.

ODK Tables has a built-in view for these kinds of visualizations called "Graph Views". The ODK-X documentation states that these graph views "[use] JavaScript libraries such as D3 to create visualizations of collected data on the device." [1] Most JS graphing libraries would work well in this situation since the only thing they need to query for is the data from the local database - no internet connection required. The example it gives, D3, calls itself "a JavaScript library for manipulating documents based on data," [2] and is closer to something

like jQuery than a bespoke graphing library. D3 allows for more freedom in implementation while still keeping the code required to create something nice relatively short. Creating visualizations for the tree data using this library would give us a significant amount of freedom to customize both the visuals and functionality to our client's preferences.

Another option would be a more purpose built graphing library such as Plotly's JavaScript library, which functions as another layer of abstraction on top of D3. Plotly.js describes itself as "a high-level, declarative charting library... [which] ships with 20 chart types, including 3D charts, statistical graphs, and SVG maps." [3] This library comes with way more built in functionality our client may want which we may not be able to implement within the time constraints of the project. On top of that, this kind of library greatly simplifies the work needed to create a graph, meaning we would probably be able to get much more working much faster than if we did everything manually with D3.

There are many more graphing libraries which fill the same role as Plotly.js. Chart.js is one I have worked with before and does a great job of creating things like line charts with little to no hassle. The reason I spoke mostly about Plotly.js was because it was based on D3 and it had support for map visualizations. While not as important for this section, the location data will need the map visualizations. Additionally, it would be good to stick to as few libraries as possible as to keep the size of the app just that little but lower for one, but more significantly, to keep the implementation cleaner. If every view uses a different library the code will become more confusing and harder to maintain.

2.2 Visualizing Location Data

There are a few ways we could go about implementing the map views for our project. The first is through ODK Tables' built in "Map View" [4], the second is through its "Navigate View" [5], the third is to use one of our decided upon js libraries in ODK Tables' "Custom View" [6].

The map view and navigate view are fairly similar. The bottom portion of both have Google maps up with pin points dropped on the various locations it grabbed from the database. At the top of the map view there is a list of all the various points found in the map at the bottom. When you click on one of the points it scrolls to that item in the list. At the top of the navigate view is a compass, heading/bearing data, and the distance between the device and the selected item on the map. Of the two, the navigate view is probably the more useful of the two for our project since the Android devices the field crews will be using are GPS enabled and being able to quickly find the tree represented on the map would be incredibly valuable. These views are somewhat limited compared to doing things with D3 for example, since D3 can load additional data on top of the map apart from just the location points.

Using either D3 or Plotly.js in a ODK Tables custom view here would allow us to create more customized maps with more data than just the points, however, getting it to display the current position of the user would probably be significantly more difficult since the JS probably can't talk to the Android hardware to get the GPS data.

One of the big considerations here is how well a solution works without an internet connection. D3 and Plotly.js make queries to external services to get the map images, however, they can account for this by also

rendering SVG outlines of the map on top of the images so that if it can't access the map data it can still display the outline from a file on the device. [7] This isn't a great solution since being able to see something like satellite imagery under the map points is very helpful for telling where those points actually are.

The ODK-X documentation doesn't really talk about how its map/navigate views work without internet connectivity, however, the normal Google Maps app is able to download portions of the world map to be used offline. The ODK map/navigate views are probably implemented using the actual Android map view, meaning it's embedding part of the Google Maps app's functionality within ODK Tables and can therefore also benefit from the offline map downloads.

2.3 Form Builder

The current ODK-X workflow involves converting Excel XLSX files into JSON form definitions which can then be imported into the ODK toolsuite and served to users. This struck us as a cumbersome method for a project like ODK which aims to be accessible to those without technical knowledge. (It's a good idea since most places will have Excel and users who have some degree of proficiency with it, but why not go a step further?) Forms with branching logic seemed like a good place to implement some kind of flowchart workflow, where users could drag and drop the form elements they wanted into place, then connect them to each other while setting the logic to decide which branch gets taken.

Part of the ODK-X family of applications includes the ODK Application Designer [8], which runs in the browser and houses the aforementioned XLSX converter among other things. This would be a good place to build the form builder. JS and JSON go hand in hand, and the drag-and-drop operations needed would be simple to implement either on our own or with some prebuilt library. Since ODK is an open source project, we could modify the ODK Application Designer to include our new functionality. The alternative is some standalone application which implements the same functions. This could be done in some kind of native .NET application [9] or perhaps even a webapp style program using electron.js [10], which would be almost identical to implementing it in the browser while still being its own independent application.

3 Conclusions

With respect to the visualizations I believe it would be best to try and implement things with just D3 at first. D3 affords us a great amount of freedom to implement things as we see fit while still being relatively quick to develop in. It also supports creating map views with geo-points as well as other data. The main drawback of D3 would be the potential lack of GPS data overlaid on the map. This could perhaps be remedied by having two map views: an Android native one with the GPS data, and another with the geo-points. This will likely be best decided through client/user feedback as development progresses to see which one meets their needs best.

As for the form builder, I think it would be best to go with modifying the ODK Application designer source code, since web development lends itself to working with JSON data as well as the kind of interactions which would allow for drag-and-drop operations.

4 REFERENCES

[1] "Using odk tables - graph view," *Open Data Kit Documentation*, 2017.

[2] "D3.js - data-driven documents," 2019.

[3] "Plotly javascript open source graphing library," 2019.

[4] "Using odk tables - map view," *Open Data Kit Documentation*, 2017.

[5] "Using odk tables - navigate view," *Open Data Kit Documentation*, 2017.

[6] "Using odk tables - custom view," *Open Data Kit Documentation*, 2017.

[7] M. Bostock, "Zoomable raster vector / d3," *Observable*, 2019.

[8] "Odk application designer," *Open Data Kit Documentation*, 2017.

[9] "What is .net?," *.NET Documentation*, 2019.

[10] "Electron.js," *Electron Home Page*, 2019.