A Cold Chain Data Visualization Tool

Melissa Winstanley

A senior thesis submitted in partial fulfillment of
the requirements for the degree of

Bachelor of Science
With Departmental Honors

Computer Science & Engineering
University of Washington

June 2012

Presented on June 1, 2012

Thesis and presentation approved by ________________________________

Date ______________________
CONTENTS

Abstract ................................................................................................................................. 3
1. Introduction ...................................................................................................................... 3
2. Visualization Background .............................................................................................. 3
3. System Features ............................................................................................................. 4
   3.1 Visualization Setup ..................................................................................................... 5
   3.2 Map Visualization ...................................................................................................... 7
4. System Implementation ................................................................................................. 8
   4.1 Infrastructure .............................................................................................................. 8
   4.2 Front End JavaScript ................................................................................................ 9
5. Development Process ................................................................................................. 10
   5.1 Stage 1: All JavaScript ............................................................................................. 11
   5.2 Stage 2: Google App Engine ..................................................................................... 11
   5.3 Stage 3: Ruby on Rails, Heroku ................................................................................ 11
6. Evaluation ..................................................................................................................... 12
   6.1 User Interface ........................................................................................................... 12
   6.2 Performance ............................................................................................................. 13
7. Conclusion ..................................................................................................................... 15
   Acknowledgements ........................................................................................................ 15
   References ..................................................................................................................... 15
ABSTRACT

As new vaccines are introduced in countries around the world, one important restriction on the success of immunization programs is the health of a country’s vaccine cold chain. A cold chain is the collection of refrigerators and cold rooms that keep vaccines cold from hubs to local facilities. To help address the issue of cold chain management, the organization Program for Appropriate Technology in Health (PATH) developed the Cold Chain Equipment Manager (CCEM) software application to manage an inventory of cold chain equipment. CCEM does not, however, contain any tool to visualize data geographically. To rectify this deficiency, we developed a CCEM Visualizer application to display cold chain data. The Visualizer is a Ruby on Rails web application that relies on the Google Maps API, allowing for custom mapping features. Initial user feedback and analysis confirms that the CCEM Visualizer would be useful for national health workers to manage vaccine cold chain vitality and coordinate responses.

1. INTRODUCTION

A cold chain is the collection of refrigerators and cold rooms necessary to keep vaccines cold from the central distribution hub to the individual local health facilities where they are used [1]. Without sufficient refrigeration at the proper temperatures, vaccines will spoil and become unusable. The management of the vaccine cold chain is important in many developing regions due to the increasing investment in vaccines and the limitations imposed by old, dysfunctional, inefficient or inadequately-sized refrigerators and cold rooms [18]. PATH first piloted the Cold Chain Equipment Manager (CCEM) in 2007 in Uganda, where it aided in inventory analysis and forecasting of future equipment needs [9]. Over the past five years, CCEM has been introduced in four other countries in Africa and one in Latin America.

CCEM is a desktop application built using Microsoft Access. Its data is drawn from national cold chain inventories that provide profile information about all health facilities and refrigeration units in the countries. Data is displayed in database format and is used to generate charts and reports; raw data can also be exported to files in various formats. The cold chain inventories also included geographic information system (GIS) coordinates among the data collected, although CCEM did not use it for visualization purposes. The CCEM Visualizer is intended to fill this void, making the facility and inventory data of CCEM more accessible to its users, who probably have limited experience with computers. Map displays help illuminate trends that may be present in data but hard to discern from tables of numbers [12]. Mapping is ideal for discovering problem hotspots and observing broad trends in future vaccine distribution schedules. In addition, a web-based application is more flexible than a desktop based one. We envision the CCEM Visualizer to be useful to managers both at the central hubs and in the field, where they may or may not have access to a desktop with CCEM installed but do have Internet access.

2. VISUALIZATION BACKGROUND

There are a wide variety of visualization tools available for both geographic and non-geographic visualizations through commercial releases and conventional research avenues. For example, IBM’s Many Eyes project – begun in 2007 by IBM Research – allows users to upload data sets
and create visualizations to share with other users, including graphs, charts, and maps of all kinds [8]. All uploaded data is publicly available and can then be edited or reused by other Many Eyes users, rendering this style of visualization unsuitable for a use in which cold chain data is proprietary to the countries involved. Tableau Software has also developed a variety of visualization products for use on the desktop; created interactive visualizations can then be published to the web [10]. More expensive versions of their products allow for use of proprietary data. While visualizations cannot be customized on the web itself, they could be customized on a separate page for each country that is using the CCEM Visualizer.

One of the most promising visualization technologies is the Data-Driven Documents (D3) framework for JavaScript [3]. D3 works by manipulating the document object model (DOM), improving transparency of representation and performance of interactive and animating components. Visualizations created with D3 include standard graphs and bar charts; line-and-hub charts that interact with the user; various circle charts; map-based overlays with markers or colored districts; and an infinite set of other possibilities due to the customizable nature of the framework. Although we were not aware of D3 at the time of initial work on the CCEM Visualizer, we will consider D3 in future iterations of the application.

Geographic visualizations, in particular, have been extremely useful in a number of situations. In the public health sphere, geographic analyses aid in analysis of cancer trends [2, 4], particularly at the county level in the United States. Map visualizations have also included analysis of other health trends, such as accidental poisoning [15], access to health services [11], and tuberculosis incidence rates [13]. The public health needs for visualization have led researchers to develop the Geovisual EXplication (G-EX) Portal for workers in public health and epidemiology to visualize their data geographically [16]. In particular, researchers have determined that interaction is invaluable for augmenting an observer’s understanding of the data [19]. As a consequence of the extensive use of geographical visualization in public health for geospatial decision-making and the importance of interactivity in aiding users with potentially low computer skills, we believe that adding an interactive, web-based geographic visualization component to CCEM will be a valuable addition to the features of the application. Decisions about inventory purchases and relocations may be influenced by geographic proximity and degree of scarcity. Since geography is an implicit factor in these decisions, CCEM should address it explicitly.

There are many commercial geographic visualization packages. One of the most well-known and popular is ArcGIS, produced by Esri [6]. ArcGIS is available for the desktop, server, mobile device, or Internet. While the Internet version of ArcGIS (ArcGIS Online) has standard features available for free, the software is not powerful enough for the customization necessary in the CCEM Visualizer. ArcGIS Online does have a JavaScript developer’s API, but at a cost. Unfortunately, the prices of ArcGIS and other commercial GIS software are rather high for use in the developing world.

3. SYSTEM FEATURES

The CCEM Visualizer is a web application, serving content from a server and rendering content in a browser. It consists of two user-visible components: a map setup page and a map display page. The flow of control is illustrated in Figure 1. First, a CCEM administrator loads all desired
data files describing health facilities, refrigerators and cold rooms, and vaccine schedule requirements onto the server. Data files are stored in comma-separated values (CSV) format and are exported from the CCEM application itself. After the data has been loaded, users may interact with the application through a browser, specifying their desired data files and visualization settings. The browser submits these settings to the server, which then saves them with a user identification number. The server can subsequently render the map visualization for the browser on demand. The visualization displays health facilities as markers on the map. The most common use case for the CCEM Visualizer is the following situation: a CCEM administrator contacts health managers in the target country or region and aggregates a set of factors and issues of interest. The administrator then loads CCEM data files for that country and completes the setup for the visualization, adding mappings and filters to match the factors of interest. The health manager may then follow a provided link to view and interact with the map itself.

![CCEM Visualizer flow of control.](image)

3.1 Visualization Setup
The admin user begins interacting with the application at the setup page. The page consists of eight tabs which, when navigated and completed sequentially, allow the user to fully customize a CCEM map visualization (Figure 2).

The admin user begins on the “Files” tab by submitting the names of the data files that he or she is interested in, classifying each as either a main facility data file, a cold chain inventory file, or a vaccine schedule file. After selecting the data files, the user proceeds to the “Fields” tab to select which fields – corresponding to columns in the original CCEM database – are of interest. Each field is given a human-readable name and is classified as either a discrete, continuous, unique, or
string field. Subsequently, on the “Values” tab, the admin user may optionally enter the possible values taken by the discrete fields and the corresponding human-readable interpretations of the values. Values given here will be incorporated into the display of the information box.

The remaining five tabs may be completed in any order. The “GPS” tab is the most essential and must be submitted before the map can be displayed at all. The user chooses which of the fields of the main facility file provide the latitude and longitude of the facilities. In addition, the user can indicate the coordinates are given in Universal Transverse Mercator (UTM) format instead of traditional latitude/longitude, giving the UTM zone and southern-hemisphere designation of the coordinates. Finally, the user must indicate where he or she would like the map visualization centered by giving a latitude/longitude coordinate pair. Under “Info Box” tab, the user may choose which fields to include in a pop-up information box that appears when a facility marker is clicked on. The first field given is the title of the information box.

The “Mappings,” “Filters,” and “Pie markers” tabs provide interfaces for specifying different types of visualizations. A mapping colors the marker for each facility according to a condition on the facility’s values. A filter generates a set of options to display only a subset of facilities based on a condition on the facilities’ values. A pie marker, similar to a mapping, determines the look of a facility’s marker. However, instead of just being a single color, the pie has various border colors and three different interior colors; the proportion of the pie occupied by each of the interior colors is determined by a specific condition on the facility’s values.

Conditions on values are provided using a basic predefined grammar. Fundamental elements of the grammar are given in Table 1 and should always be separated by spaces on either side. In addition to the default grammar elements, the user can include the variable “x” in any conditions (or additional variables for a pie marker), signifying the data to map. The variables themselves are defined using the same grammar combined with specifications of fields in the format “{<field-type>::<field-name>}”, where <field-type> is replaced by either “facility,” “fridge,” or “schedule” and <field-name> is replaced by the name of the column. Any spaces in the field name should be replaced by a double dash – “--” – to prevent the spaces from distinguishing elements. An example specification of a filter is given in Figure 3.
Table 1. Condition grammar elements.

<table>
<thead>
<tr>
<th>Operators</th>
<th>Conditions</th>
<th>Logical</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ (</td>
<td>= lt</td>
<td>AND</td>
<td>COUNT ( … )</td>
</tr>
<tr>
<td>- )</td>
<td>!= lte</td>
<td>OR</td>
<td>SUM ( … )</td>
</tr>
<tr>
<td>/</td>
<td>? gt</td>
<td>NOT</td>
<td>AVERAGE ( … )</td>
</tr>
<tr>
<td>*</td>
<td>: gte</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 3. Example filtering on facility type.

To revisit a setup page and revise the user options, the admin user navigates to the setup URL and append as a query parameter the ID of the desired user options (for example, “?id=3”).

3.2 Map Visualization

After fully customizing the visualization settings, the admin user clicks “Map” to proceed to the map display or sends the corresponding link to a health manager for viewing. After letting the facilities, fridges, and schedules load, the user (admin or health manager) will hit the “Display” button to load the actual map visualization (Figure 4). A marker determined by the currently selected “Category” – either a mapping or a pie marker – represents each health facility. Clicking on a particular marker brings up an information box describing the corresponding facility. The “Key” given at the bottom of the page explains the meaning of each marker type. The “Setup” button the upper right-hand corner provides easy access to the setup page in the event that the user decides to change his or her settings.

The control panel on the left-hand side provides options to change the current mapping/pie display. Only one mapping or pie display may be shown at one time because only one marker
can be used per facility. A second pane of drop-down options enables filtering of the displayed set of facilities. More than one filter may be applied at once. A third pane switches between vaccine schedules, allowing for planning and manipulation of vaccine requirements. Finally, the “Ignore/consider population” button turns on and off the variation of marker size based on the total population serviced by each health facility.

![CCEM Geographic Visualization](image)

**Figure 4. Map display.**

4. SYSTEM IMPLEMENTATION

4.1 Infrastructure

The server-side infrastructure of the CCEM Visualizer is implemented using Ruby on Rails and deployed on Heroku [7]. Rails is an object-oriented language that utilizes the model-view-controller (MVC) design pattern to organize classes. The models (black) and controllers (red with dashed borders) are illustrated in Figure 5. The infrastructure is backed by a PostgreSQL database, as preferred by Heroku.

Each model represents a table in the database. The first major type of data in the database relates to the original CSV data files. Facilities, Fridges, and Schedules each represent a row of their corresponding data files and each belongs to the VaccineFile representation of that file. Each file also has a number of Fields, corresponding to the column names of the data file. The other type of data stored in the database pertains to user-specified display options. The UserOptions class stores these options in a hierarchy: in addition to InfoBox options (which fields should be displayed in the pop-up information box) and Displays (mappings, filters, and pie displays), a UserOptions object contains InfoOptions for customizing which data files are to be used and
what human-readable names are to be used for field names (stored in FieldOptions) and value names of discrete fields (stored in ValueOptions objects).

Figure 5. UML class diagram, back-end infrastructure

The two controllers of the application facilitate client interaction with the models. They receive and process HTTP requests. The PagesController responds to GET requests for the setup page and map page, rendering the appropriate view and providing needed data from the models (for example, existing UserOptions data for a particular options ID, or the desired map display). It also renders facility, fridge, and schedule inventory data upon requests for the appropriate URLs on the map display page. The UserOptionsController deals with PUT and POST requests that update a user’s display options, parsing parameters and redirecting any new data to the corresponding UserOptions update method.

The two major views correspond to the two pages of the application. Views are a combination of HTML, CSS, and embedded Ruby code. The embedded Ruby allows for creating more sophisticated views using Ruby constructs – for example, rendering a sub-view for each of an array of objects, such as the Fields of a particular VaccineFile. The views also link to JavaScript functions that create interactivity.

4.2 Front End JavaScript
The JavaScript of the CCEM Visualizer fulfills three distinct roles. On the setup page, it is responsible for form interaction: dynamically adding and removing rows of the various input tables, as well as submitting updates to the current user options. Updates are implemented as
AJAX requests to the server, with the page being reloaded on each successful update. This allows changes to propagate easily through all of the tabs.

The JavaScript on the map page is more sophisticated. It performs requests for facility, fridge, and schedule data and then renders the display. Displaying involves reading the user options object and extracting the conditions for mappings, filters, and pie displays. Once the appropriate display type has been extracted, the JavaScript iterates through all facilities and calculates the appropriate value (mapping color, pie proportions, filter values) for each facility. This involves parsing the conditions and substituting facility (or fridge/schedule) values into the condition before evaluating it.

Parsing translates a condition into an Expression, a sequence of valid operators, conditions, and logical operators (Table 1), as well as Funcs (functions listed in Table 1) and FieldVars (variables specified in curly braces). Functions recursively contain a sub-expression. Once the conditions are parsed into valid expressions, we iterate over each facility, substituting that facility’s values for the FieldVars in the Expression. We then evaluate the Expression by concatenating all components of the Expression into a string and then calling JavaScript’s built-in eval() function. Although eval() is considered an unsafe operation when applied to unknown user-given data, we carefully white-list all appropriate components of the expression to prevent unsafe code from being evaluated. The evaluated expressions give the corresponding color or filter value for the display. These values/colors are cached for future viewing of the same display.

Finally, some geographic coordinates provided for facilities were given in the UTM coordinate system rather than conventional latitude and longitude. In order to display these coordinates on Google Maps, we translated them from UTM to traditional latitude/longitude. The translation is performed in the JavaScript because we utilized pre-existing JavaScript code for performing the complicated translation process.

5. DEVELOPMENT PROCESS

The CCEM Visualizer went through several iterations, beginning with a prototype and ending with an evaluable product. The shortcomings of the first two attempts led us to decide on the final application construction, using Ruby on Rails (Figure 6).
5.1 Stage 1: All JavaScript

The first implementation of the application was a prototype that utilized JavaScript, HTML, and CSS exclusively. Data were served directly to the user as static CSV files that were then parsed by JavaScript in the user’s browser. JavaScript also did all the work of setting up the map display. In addition, all visualization options and data were personalized to the country of Malawi. We showed the prototype to personnel in Malawi associated with the Clinton Health Access Initiative (CHAI), who affirmed the potential usefulness of the prototype.

There were several issues with this scheme. Firstly, data could not be personalized. For example, all file and column names were hard-coded into the application, prohibiting any attempt to generalize the application to other locations or slightly different data sets. Secondly, data files were re-parsed each time to extract facility, fridge, and schedule information, resulting in duplicated work with each session. Thirdly, the CSV files were served to the browser statically. While the data in these files are not as confidential as patient records, they should nonetheless not be shared with the entire Internet. In the interest of security, we desired a less accessible way of serving the data.

5.2 Stage 2: Google App Engine

Following the success of the JavaScript prototype, we proceeded to rectify the issues we had identified as problematic. Generalizing data and display options to arbitrary countries required a more sophisticated means of saving user options and personalizing the CCEM Visualizer experience. Instead of simply serving static files, we determined that a client-server model of interaction was required. After consideration, we decided to write our application for Google App Engine. App Engine provides a convenient Java API for developers, abstracting system administration and server maintenance. It also features data storage through the App Engine Datastore, replacing the need to explicitly manage a database [5].

Unfortunately, App Engine’s abstractions ultimately proved too confining for the complexity of the CCEM Visualizer. In particular, the Datastore over-complicated and over-restricted our data manipulation. The annotation and method framework required to implement Datastore compatibility was not clearly documented and did not allow for complicated embedded data structures such as maps. In addition, the free App Engine instance we used allowed only a finite number of Datastore write operations per day. We found that the number of operations required to maintain the application often exceeded the number of allotted operations, requiring distribution of operations over multiple days. Although upgrading to a paid App Engine instance could have solved this problem, the combination of scalability and database storage issues encouraged us to find an alternative back-end for the CCEM Visualizer.

5.3 Stage 3: Ruby on Rails, Heroku

Ruby on Rails (or simply Rails) is an open-source framework for building web-based applications. It provides support for the full range of web-application activities, from database queries to server HTTP responses to front-end template rendering into HTML. Rails’ basic framework emphasizes the MVC architecture, enforcing good coding practice. We chose Rails for all of these features: although it provides many out-of-the-box tools for easy programming, it
also has significant power to customize. The ability to design a database exactly to our specification was also very desirable.

Like the Java back-end of the second implementation, the Ruby on Rails implementation has a convenient way to host the server. Heroku is a free cloud platform for web applications that helps minimize the time spent on server management [7]. It also allows the addition of add-ons, supplemental tools to enhance an application. In particular, the New Relic add-on is a useful performance measurement tool that helps us evaluate the reliability, scalability, and response times of the CCEM Visualizer around the world.

6. EVALUATION

6.1 User Interface

We solicited feedback about the CCEM Visualizer from representatives of PATH, the organization that developed the original CCEM software. These representatives work closely with public health managers in the countries that use CCEM and have a thorough understanding of the requirements and uses of software for effective cold chain management. They were able to confirm that this tool would be useful in a number of circumstances for diagnosing problem areas, coordinating responses, and submitting reports and budgets:

[T]he mapping enables one to answer quickly:
* is this a serious/widespread problem nationally?
* if yes, is it related to type of facility?
* and is the problem geographically localised, or general throughout the country?
* lets see some example facilities with the problem
* so which types of facility and in which zones should we concentrate response...

Very useful indeed...

Similarly,

I was thinking that the tool/map would be used for: Annual planning by the national or regional managers...I think that Mr. Valle would want to hone in on where equipment purchase/replacement is needed and he would to use the map to support his budget request with a clear visual to convince the MoH [Ministry of Health] and MoF [Ministry of Finance] honchos to honor his request...The expected outputs would be to improve supportive supervision, activate repair services, request replacement parts/technicians/transport...

Apart from confirmation of the goals of the CCEM Visualizer, PATH provided other useful feedback about the uses for this geographic visualization of data. We had not previously considered either solar energy or equipment performance in our mappings and pie charts, so there are many additional visualization possibilities. Many of the issues brought up by the PATH representatives are not limitations of the application but rather are problems that can be addressed by modifying configuration parameters:
I think ‘Met by other refrigerators’ needs to be made clearer…Could I see facilities with electricity > 8 hrs a day that have absorption equipment?…What does electricity mean? > 8 hrs per day? Or > 16 hours per day?…Will countries with more facility types be able to have a drop down menu for all their types?…How about filtering by the ownership aspect (public v. private v. NGO v. ?)

Clarifying human-readable names, adding new filters and mappings, and customizing by country are all features that are already supported by the CCEM Visualizer. All that is required are a few edits of the particular map setup involved. Such comments reinforce, however, that the users of the final visualization will not be computer scientists and that created visualizations must be clear, uncomplicated, and well documented.

Input from PATH also uncovered complementary displays that would be extremely helpful in negotiating the complexity of national cold chain inventories. One representative brought up the benefit of an associated listing of facilities:

One mapping example that comes to mind is Redfin. I was able to look for houses with certain characteristics (floor space, lot size, garage, bedrooms) in a region (Magnolia and Ballard). I saw them on a map and in a line listing at the bottom of the map. I can sort the line listing by a characteristic (price), and then select/scroll through houses a little less and a little more expensive than my price point, seeing details on the right hand of the screen. This allowed me to get a feel for what was going on in the housing market. It could have some transferable value to the cold chain.

The combination of text and geographic displays, while not incorporated into the existing CCEM Visualizer beyond the pop-up information boxes, could provide an intermediate connection between the pure data of the CCEM Access Database and the map-type display of the CCEM Visualizer. Such a listing of matching facilities – for example, those that have a certain set of traits indicated by the currently-selected filters – would help users find facilities of concern more easily, and is a candidate for a future iteration of the Visualizer.

6.2 Performance
In addition to using input about the design of the interface, use cases, and additional desired features, we evaluated the CCEM Visualizer on its performance capabilities. The Visualizer is likely to be of most use in regions of the world where Internet speed is slower than in the United States, so response time is an issue. We did not, however, worry about Internet connectivity in general: the target users of the CCEM Visualizer are health managers who can depend on Internet access. Additionally, the spread of mobile technologies throughout the developing world has resulted in increasing availability of the Internet through mobile networks. Therefore, we chose to analyze the response time of the visualizer in reaction to various types of requests.

The results are summarized in Table 2. All server response times were gathered from Heroku using New Relic, a web application performance management tool. Times to switch between mappings and between pie charts was measured in the browser itself, as switching does not
require contacting the server. Heroku provided the current size of the database through the `heroku info` command.

While these response times are not incredibly fast, they seem to be acceptable for the use case of the application. The initial responses of the setup page and the map page are completed within 1 second, allowing the typical user’s flow of thought to stay uninterrupted [14]. The loading of the facilities, fridges, and schedules, however, does have significant delay, particularly as the size of the data set grows. For normal-sized maps, however, the response time is still well within the 10-second limit to keep the user’s attention [14]. To further mitigate the effects of delays, we ensure that the map page itself loads quickly and then asynchronously load the data set accompanied by a loading indicator.

Table 2. CCEM Visualizer performance.

<table>
<thead>
<tr>
<th>Action</th>
<th>Performance¹</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serve setup page</td>
<td>0.97s</td>
</tr>
<tr>
<td>Serve map page</td>
<td>0.33s</td>
</tr>
<tr>
<td>Serve facilities</td>
<td>0.67s / 6.41s²</td>
</tr>
<tr>
<td>Serve fridges</td>
<td>0.84s / 4.61s</td>
</tr>
<tr>
<td>Serve schedules</td>
<td>0.65s³</td>
</tr>
<tr>
<td>Switch mappings</td>
<td>3.11s</td>
</tr>
<tr>
<td>Switch pie charts</td>
<td>3.33s</td>
</tr>
<tr>
<td>Database size</td>
<td>1 MB⁴</td>
</tr>
</tbody>
</table>

The time to switch between mappings and schedules is a more significant influence on overall application performance because users are likely to spend far more time examining and manipulating the map than refreshing the page in their browser. Pie chart in general take slightly longer to render than mappings, probably due to the larger amount of information required to determine a complete pie chart (proportions of inner colors and outer color, not just a single coloring). In both cases, the given averages consider both first-time rendering of the markers and subsequent renderings of the markers. The first rendering takes longer to complete because each facility must be evaluated to determine its marker. However, any ensuing re-rendering of the same mapping or pie chart utilizes cached computations, speeding up the process. The use of caching aides the performance of the standard use case, in which a health manager is likely to be switching back and forth between mappings to find cold chain issues and patterns. In addition, users are unlikely to be viewing all facilities at once (the vast quantity of data is overwhelming – filters facilitate focusing), speeding up the rendering as fewer facilities must be displayed.

¹ Given response times are the average of 10 trials.
² Pairs of numbers represent response times for small and large data sets, respectively. The small data set has 827 facilities, 1423 refrigerators, and 2481 schedules, while the large data set has 5305 facilities and 4946 refrigerators.
³ Schedules for a large data set were not available at the time of testing.
⁴ Database size includes both the large and small data sets.
7. CONCLUSION

The Cold Chain Equipment Manager (CCEM) software developed by PATH addresses a present need in the developing world: an ever-increasing number of vaccines are being made available, but without adequate refrigeration from central hub to remote facility, they will go bad before being used. A national-level inventory of the country’s cold chain – refrigerators and cold rooms – establishes a baseline, and CCEM facilitates centralized analysis of the cold chain. The CCEM Visualizer adds the new feature of interactive geographic visualization using Google Maps to the management software. Geographic visualization has been extremely useful for analysis of public health trends, and interactivity is known to improve user understanding and involvement in the data. The Visualizer consists of two major components: a means to specify data and customize the map display to show desired measures of cold chain vitality, and the map display itself, allowing the user to filter the data and change the marker display of each facility. The degree of customizability is an essential feature of the Visualizer, as revealed by feedback from representatives at PATH.

The CCEM Visualizer is built as a Ruby on Rails application deployed to Heroku. Although we attempted two other versions of the application, this final setup is the most customizable and maintainable of the iterations. The back-end server application stores the database and serves data back to the client’s browser. Although there are performance bottlenecks – the amount of data to transmit to the browser and the speed at which Google Maps can render markers on the map – the application is definitely usable for foreseeable uses, and we will continue to work to improve the performance of the CCEM Visualizer. Feedback from PATH has confirmed the usefulness of this geographic visualization for decision-making, report generation, and response coordination.

ACKNOWLEDGMENTS

Thank you to Ruth Anderson and Richard Anderson for their advice and guidance over the past year, and to Sophie Newland and John Lloyd at PATH for their valuable feedback on the project.

REFERENCES