Vaccine Stock Tracking System: Design and Implementation

Ali Byott **Josh Cho** **Elijah Greisz** **Xuan You Lim** **Michael Wiem**
University of Washington
Seattle, WA, USA
[byotta, joshua97, egreisz, xylim98, mwe6453]@uw.edu

ABSTRACT
Recent events have shown an ever-present and increasingly urgent need to modernize and optimize vaccine information infrastructure, which many countries currently lack. This presents a need to develop a system that fulfills such a niche and that can be easily adapted to any country with similar requirements. We explore the development of such a system that makes few assumptions about the location of deployment. This requires a synchronizable and accessible mobile application for data entry and a web dashboard for visualization and aggregation of the data. Beyond this, we demonstrated ODK-X's effectiveness in building such a system and integrating our pieces with a backend web server. We intend for this prototype to be easily replicable to any other deployment context with minimal existing infrastructure and provide a design process that demonstrates one approach to take.

Author Keywords
ICTD; vaccine stock tracking; immunization campaigns; mHealth; ODK-X

INTRODUCTION
Vaccines – a biological invention that provides immunity to certain diseases typically caused by viruses – have been a long-standing tool used to fight and eradicate infectious diseases. Conceptually, a vaccine works by introducing an altered or selective form of a virus to a person, often through an injection, so that the body's immune system can recognize and acquire immunity toward the virus. Subsequently, if the same type of virus were to invade the person's body, the body's immune system will recognize and destroy the virus, preventing the person from getting ill.

Vaccines have been used as a prevention method for multiple infectious diseases such as polio, measles, and more all over the world. Vaccination – the administration of vaccines – has been effective at generating widespread immunity that has resulted in worldwide eradication of certain diseases such as smallpox[11]. Some vaccines are administered routinely for which people get at certain stages of their lives such as Hepatitis B which everyone gets the first dose at birth[6] or the flu vaccine which everyone can get on an annual basis.

On the other hand, some vaccines are developed and administered to people on an ad-hoc basis where mass vaccination campaigns are needed to vaccinate a large majority of people in a short period of time, typically in response to outbreaks of infectious, viral, and deadly diseases.

On March 11, 2020, the World Health Organization (WHO) declared the novel coronavirus outbreak, COVID-19, a pandemic. COVID-19 is a novel infectious disease that can cause lung complications such as pneumonia and result in various symptoms such as shortness of breath and fever. In some cases, COVID-19 can lead to death or long-term injuries[10].

Vaccination is then seen as the vehicle that can drive the world out of the pandemic, reducing infections and deaths, and loosening lockdowns and movement restrictions, all of which are consequences of the infectious COVID-19 disease.

In view of the current state of vaccination and the need to fight and eradicate COVID-19, we picked up the challenge of an implementation of a vaccine stock tracker. One important aspect for mass vaccination campaigns is tracking vaccine usage in different geographical locations to inform vaccine distribution and acquisition strategies. This is so that an effective vaccination campaign with minimum waste and maximum efficiency can be achieved.

Tracking vaccine stock usage for vaccine distribution and acquisition represents both a logistical and technical challenge. Logistically, vaccine doses need to be transported to multiple facilities at different geographical locations. These vaccine doses also have an expiry date or an efficacy period when the vaccine doses remain usable. So, delivering the right amount at the right time is a significant logistical challenge to allow for an effective vaccination campaign. Consequently, on the technical side, tracking vaccine usage at multiple facilities represent a technical challenge to accommodate for multiple clients or users interacting, inputting, and manipulating vaccine usage data, potentially in a concurrent manner. Furthermore, these vaccine usage data need to be aggregated and transformed to present useful information that can inform vaccine distribution and acquisition strategies.

We focused on the technical aspect of the challenge and attempted to design and develop an implementation of a vaccine stock tracker. A vaccine stock tracker allows tracking when the amount of vaccine doses is altered for differing reasons such as being acquired, administered, spoiled etc. Fundamentally, there are two aspects to a vaccine stock tracking system. First, tracking the changes to vaccine doses through data input. Second, aggregating and transforming the raw input data of vaccine doses changes into useful data that can be presented as information.

Our implementation of the vaccine stock tracking system consists of the following features: a mobile solution for inputting vaccine dose changes; a web service for data aggregation and transformation; an application programming interface (API) to distribute aggregated and transformed data; and a visual web-based dashboard for presentation and communication of up-to-date vaccine usage data and information.

This paper will detail the design processes of each component. Then, the paper describes the technical implementations of each component before finishing with a discussion on takeaways and future works.

RELATED WORK

In the past half century, other researchers have found immunization information systems (IIS) extremely effective in streamlining the immunization process. While health care workers and administrators may be able to attest to anecdotal evidence that suggests so, Groom et al performed a systematic review of 240 articles and abstracts [4]. Through this collective body of evidence, they linked implementation and use of IISs to increased vaccination rates. More specifically, they examined common approaches by IIS-interventions in their attempt to increase vaccination rates. The first method was using IISs to remind people to get their vaccinations. The second was providing a comprehensive way to evaluate vaccination performance to provide feedback to healthcare providers. The systematic review also found that one prominent area of study within IISs is their effectiveness for vaccine ordering and inventory. This is the primary area in which we will focus our development.

In 2013, Quebec finished implementing a vaccine inventory management system. The system was comprehensive, encompassing stock management, monitoring of the cold chain, and inter-site transfers. It was expected to be used to minimize wasted vaccines, allow for distant managing of stock, and to generate reports. Gagnon et al. performed interviews during the implementation phase and conducted surveys among users during the deployment phase to extract insights regarding the process [3]. They found that stakeholders believed the key benefit to such a system would be the ability to track vaccine stock in real time. This was seen as the best way to improve future ordering of vaccines and prevent further wastage. Another belief was that the system could help health care workers more easily react to problems. Power might go out at a facility, leading to spoiled vaccines. Alternatively, there could be a bad batch, in which case it would be essential to quickly figure out where those vaccines had been distributed to take them out of circulation. Centralizing this information and moving away from non-uniform systems makes this process more efficient and less error-prone. However, they found that many workers found the system difficult to use. There is no elaboration on what made it that way, but streamlining the system and its use while still approaching the same, essential use cases is a primary goal of this project, and the experiences from Quebec provide some insight on the approach to take.

The concept of a specialized vaccine inventory system was extended further by Colburn's study that developed a system to manage Shingrix vaccine inventory for ambulatory clinics in New England [2]. The goal of this project was to trace the stock of a very limited and high-cost vaccine to minimize wastage, due to uneven supply and demand. In particular, they found that many vials would get held up in a refrigerator until their expiration date. This reveals a need to not only keep track of where vaccines are but to also maintain their expiration dates. Their findings indicated that the project saved money by making inventory management more efficient and likely led to less wastage of the vaccines.

However, both of these approaches developed a system with very specific assumptions in mind about the place of use. Another goal of our system is to make it semi-agnostic to the country in which it will be deployed. Some changes will need to be made based on the country's geography and political process, including the departments in charge of public health and immunizations, but the overall design of the system should not. In particular, our focus is to make it easily adaptable in lower-resource environments, such as low and middle-income countries (LMICs), as more developed areas often have pre-existing immunization systems and are less likely to need a new one. Mvundara et al. looked at two systems developed from 2013-2018 by the Better Immunization Data Initiative [7]. These systems were built to maintain registries for immunization in Tanzania and Zambia, two LMICs. Each health facility was provided with a tablet and barcode scanners to use. While purchasing these constituted most of the startup cost, a significant portion for each country was the software development process. Thus, a pre-developed system that can be generalized to different countries with little modification could potentially minimize these costs and make it an easier transition.

Werner et al. examined the same systems in Tanzania and Zambia from the health worker perspective [13]. They found that moving into the tablet-based system made workers more productive and made them spend less time on administrative tasks. Additionally, moving from paper to tablets made the system more accurate and more responsive, shortening the timeline necessary to retrieve useful information about immunization progress. Collectively, these reinforce the notion that electronic immunization information systems are useful in optimizing paper-based systems for countries that have yet to replace them.

As for the actual design, assumptions that can be made about Canada–such as the fact that hospitals have access to a strong connection–cannot always be made of other countries. Razaq et al. explored the design of an immunization information system in Pakistan [9]. While their system focuses on tracking the vaccination status of individual children, which is a decidedly different goal than our own, it provides some insights into the design process. In particular, they describe their approach of an Android mobile application for data entry in the field by vaccinators and a web dashboard for real-time visualization of data by supervisors and administrators. In use, they found that slow and unreliable internet connectivity could be a problem in uploading data in certain regions of Pakistan. Designing a system to accommodate an unreliable network must be another goal of our project.

A similar approach was taken by PATH, a global health organization, in implementing an electronic immunization registry in Vietnam [8]. Based on interviews with stakeholders within vaccine administration, they determined a technological foundation for which the application would need to be made. In particular, individual facilities often did not have access to computers while higher levels in the hierarchy–provinces and districts–did. However, cell phones were more universal, so the resulting system was developed on mobile and to work with normal cell phone bandwidth. They also found that the pre-existing paper-based system was error prone. While the introduction of errors is unavoidable, they should be dealt with and not persist. This is another important consideration in the development of a health system that is only as useful as the data is reliable.

This then extends into different technological approaches to these problems. As mentioned, the system designed in Quebec used computers. The other systems used mobile devices for data entry and a web dashboard for visualization of that data. In considering lower-resource environments, the second approach of using mobile phones seems better suited and more widely used. Back in 2008, Heeks described the "next" phase of Information and Computing Technology for Development as one concentrated around the use of mobile technology due to a much higher percentage of people having access to mobile phones–and mobile data–compared to personal computers [5]. This is still true of healthcare professionals. Olok et al. studied doctors' use of technology in Northern Uganda [12]. Scoring their aggregated level of technology use, they found that doctors have a significantly higher score for mobile phones than they do computers. In other words, healthcare professionals are more comfortable and have greater access to mobile technology. This is an important element in designing a system that can be widely used in countries where this disparity exists.

The use of mobile technology in a generalizable health information system was further explored by Brunette et al [1]. They developed a cold chain information system that provides relevant information about health facilities and their refrigerators. A core piece of this was ensuring that the system could be easily replicated in at least 70 countries. The data management was done via Open Data Kit's tool, ODK-X. This is a framework that renders a mobile application that interacts with a master database containing the synchronized data from all health facilities. Then, there is a dashboard on the web that visualizes that data, much like Razaq et al.'s system in Pakistan. However, this application focuses solely on the cold chain infrastructure and not vaccine stock or immunization registries. The goal of our project is to adapt this overarching design and model into working for a vaccine stock system that is likewise replicable in many different countries. Thus, there is a lot to learn about the process this team used, ranging from the specific technologies, such as ODK-X for the mobile application and JS and D3 for the dashboard, to broader lessons about general use cases.

DESIGN PROCESS

Designing from the perspectives of users, the vaccine stock tracking system should accommodate multiple use cases derived from different stakeholders. We designed the system with country-level implementation of mass vaccination campaigns in mind and simulated our system to work in Uganda. As such, the relevant stakeholders are country-level health ministers, facility-level administrators, and researchers.

A health minister or someone in charge of the mass vaccination campaign of the country makes vaccine distribution and acquisition decisions and conducts certain strategies to ensure a successful vaccination campaign. To do so, they will require vaccine usage data such as the overall remaining vaccine doses and vaccine usage trend in the country and in districts or facilities. Another stakeholder of the system is a facility-level or district-level administrator. They require specific facility-level or district-level data and information such as vaccine doses changes i.e. new delivery of vaccines or vaccine doses administered at the site, to make logistical decisions. The web-based visual dashboard of our system, particularly, is designed to accommodate these stakeholders and use cases.

A third stakeholder of the system is a group of relevant researchers or system developers who are keen to extract useful vaccine usage data from the system. These can be researchers who are studying vaccination campaign efficacy and impact, or system developers who are designing other related systems such as a vaccine availability notification tool which will require vaccine usage and stock data. Our web service that serves these data and information via an API accommodates these stakeholders and use cases.

A fourth stakeholder of the system is a group of users who need to input data such as changes to the amount of vaccine doses into the system. This group of users are likely to be facility-administrators or healthcare workers who administer the vaccines in various facilities in multiple geographical locations. Considering the different user locations and uncertain availability of computing equipment at every location, a mobile solution for data input is more apt as the availability and usage of mobile phones are widespread. Additionally, we sought to build on and adapt the overarching model of the cold chain information system developed by Brunette et al. [1]. As a result, we found that adapting and utilizing the ODK-X framework to build the mobile solution helped us abstract away some technical challenges such as synchronization of data changes from multiple concurrent mobile clients and mobile application development.

In this section, we will discuss the design processes and decisions of each system components: mobile solution for data input, web service for data aggregation and transformation, and web-based visual dashboard.

Mobile Application

The first part of the design was conceptualizing the mobile application. As mentioned previously, we came to the determination to use ODK-X. Beyond its purpose in structuring our data, this already functionally scopes our app. ODK-X serves as a rendering engine for the mobile application given some configuration files. This also abstracts away all direct interaction with the database server and handles authentication of users.

Thus, the design for the mobile app mainly came down to choices about what should be included, which includes the overall schema of our ODK-X database. The core use case is the entry of stock data. While it might be enough to make the user simply update the quantity of stock present in the facility on some regular interval, this provides little actionable information. For one, we want to be able to separate this out into different types of vaccines, which is a necessary use case. An example is the United States' COVID-19 vaccination campaign, which utilized three different vaccines–Pfizer, Moderna, and Johnson & Johnson. Additionally, the same system should be used for COVID-19 campaign immunization as routine polio immunization. Thus, the app includes a form to add new vaccine types, which allows the user to specify the types of vaccines that the country is currently administering.

As mentioned previously, vaccine types are not uniform. Vaccines of a certain type produced can be broken down into "batches," which is the set of vaccines produced together. Each batch could have different expiration dates, and some batches may be faulty. These should be captured by our system, so we included the ability to enter vaccine batches. Necessarily, these include an identification number that is unique to that vaccine type (a different vaccine type might have a batch with the same identifier) and an expiration date. This allows facilities to digitally keep track of the amount of vaccines belonging to each batch for each vaccine type.

Additionally, we want our system to capture changes in vaccine count. We label each change in stock a "transaction." Transactions can have various reasons. A district store responsible for storing vaccines and shipping them to other facilities might have a net-negative transaction that sends vaccines away. Another facility might have a net-negative transaction that results from administering vaccines. When vaccines spoil or are lost, that is another way to have a net-negative transaction. To gain vaccines, they must be delivered from some other facility, resulting in a net-positive transaction. Categorizing each transaction into these buckets allows administrators to easily access information about what is happening at each facility. We also made the choice to make these transaction periods be of an arbitrary length. Different countries and different regions within them will likely have different reporting periods. We allow the user to select the start and end date for the transaction, providing maximal flexibility.

During much of the development process, there was the intention to include a facility stock table within the app. This table would show the amount of each batch at a specific facility. Because all of our stock calculations are happening on the backend server (including some additional level of validation), this would require a two-way flow of data between the ODK-X database and the backend server. This was not built into our model of the system, so this has instead been relegated to future work, which will be discussed later.

Additionally, we need to be concerned about user mistakes in data entry. We categorized these into two major categories. The first is when a single transaction is significantly off. This could happen if a user types 5000 instead of 500, for example. The other is when small errors build up into a sizable difference. Thus, we added another transaction type for a stock correction. A facility administrator can use this transaction type to update the reported stock value if it does not match the actual value. This can be positive or negative and depends on the difference between the two.

Backend

The backend is focused around connecting the ODK-X database to the frontend. To do this, the backend needed to retrieve data from the ODK-X database, process and cache the data, and transfer it to the frontend. Here, the underlying principle is being robust; the backend must represent the newest version of the ODK-X database, while being accurate and fast.

Retrieving data from the ODK-X database was a design challenge. At first, a direct connection to the ODK-X database was attempted. Instead of caching the data elsewhere and transferring it later, we would access the data directly from the web client. However, any connection attempt we threw at the ODK-X database was simply rejected. We soon realized that the ODK-X database is protected from direct connections; a design choice to protect the data and minimize possible errors in the database. Next, we tried the REST API for ODK-X. This did let us access data directly, but the API calls were complex, and the retrieved data needed a lot of handling on the client side. The REST API for ODK-X was surely viable, but it would not be robust enough for our purposes.

At this point, we needed to reconsider our approach. Integrating the backend to the frontend will minimize the number of technical components we need to develop. However, this approach won't allow us to cache the data and the workload on the client side will be overwhelming. Also, we wanted our backend to be expandable to other projects, but this will be impossible with the integration approach. So, it was only reasonable to completely separate the backend from the frontend.

After deciding to separate the backend from the frontend, we stumbled upon ODK Sync Client. ODK Sync Client is a Java library that allows quick and reliable access to the rows in the tables on the ODK-X database.

Since the ODK Sync Client uses Java, we began investigating for frameworks that would abstract away the setting up of endpoints. For this, we found the Spring framework. We decided on a few endpoints, their parameters, and what the expected output would be. After so, we needed to decide on where to host the backend. To keep with the robust theme, the backend should sync with the ODK-X database periodically, be accessible from different locations, and remain fast. Our local machine worked as a testing ground, but it was not accessible from different locations. Thus, we designed the backend to sync with the ODK-X database once every minute and hosted it on Azure with the "always on" feature turned on. This allowed the backend to be up-to-date, and the frontend would be able to communicate with the backend anytime, anywhere, and with a response time less than 500 milliseconds.

After some trial and error, the backend was successfully retrieving data from the ODK-X database and communicating with the frontend. The next big question was how to handle the read data from the ODK-X database. The main data we were looking at was the transaction data. Since the project is about tracking the stock number of vaccines, any information about usage or current stock can be deduced by traversing the transactions. However, some information in the transaction data was encoded, meaning the actual data was stored in a different table and a reference key of sorts was left in its place. To fully understand what the transaction is about, we needed to reference these keys back to their corresponding tables to extract the actual data. Also, the backend was intended to be flexible and able to return any data the frontend might request. With these constraints in mind, our first attempt at the backend was made: creating a unique data structure which chronologically organized the transactions in a nested map structure. We handled the reference problem by dereferencing all the keys inside the transaction data before they were placed inside our data structure. The chronological organization of data seemed like a reasonable approach since we expected most requests from the frontend would be constrained on a specific timeframe. To keep things flexible, we created methods in a fragmented and minimal manner. For example, if you wanted to filter the database with three different constraints, you would need to use three different methods in combination that matched the constraints respectively. This seemed like a solid design but was quickly proven inadequate for our purposes.

The flexibility of our initial backend didn't last long, as the code became unreadable from all the fragmentation. We would have to use multiple loops and dozens of different methods to return what the frontend wanted. On top of that, what the frontend expected from the backend became more specific as the design progressed, and thus the backend needed to expand. The designed flexibility covered some of these expansions, but when the expansion reached the outer limit of our backend scope, we had no choice but to add new data structures and methods. In the end, the backend was functional, but the design was an entangled mess: inflexible, unreadable, and disorganized.

After the final demo, with the final versions of the frontend and the ODK-X database, we decided to refactor the entire backend. This time, the focus was on organization and flexibility. We would read from the ODK-X database and dereference the keys like the last time. However, we designed it so that the backend would be able to expand this process to read more, or less, data from the ODK-X database without worrying about key dereferencing. Also, the chronological organization just added confusion and complexity, so we simplified the data structure (to be discussed in the system architecture section), which made it flexible for change and expansion. The fragmented functions seemed redundant; all of these functions filtered the database in their own way, but filtering is still filtering. With the simplified data structure, a single method with loosely typed parameters was able to handle all filtering actions. The refactored backend ended up being shorter, flexible, organized, documented, and thus robust.

Frontend

The visual dashboard is a key aspect of the vaccine stock tracking system. It is the place where all the components come together, to facilitate important public health decisions, such as the decision to order more vaccines or redistribute vaccines between districts/facilities.

To begin with our design choices, we must first look at what functions the visual dashboard needs to be able to provide: *a*) the ability to get an overall picture of current vaccine inventory in each facility and district, *b*) the ability to see vaccine stock change over time, and *c*) the ability to see a granular, chronological view of vaccine transactions as they come in. Each of these was derived from the insights gained by the related work.

To build the frontend, we were drawn to using React over using plain JavaScript. React offers a robust, highly industry-standard method of building front-end applications. However, the library we used to generate the choropleth map (Leaflet) did not have official documentation explaining its usage with React. As such, it was difficult for us to quickly get both React and Leaflet to work together.

After deciding not to pursue the usage of React for our project, coding the dashboard in plain JavaScript was the primary focus. We decided to pursue each of the three functions separately.

For the function *a*, the healthcare administrator would need to be able to get an overall idea of how many vaccines are available for use currently, both by facility and by district. As we researched information systems, we decided an interactive map view would be the optimal way to display this information. We briefly considered a list view (where the number of vaccines in a facility/district were all laid out in numerical format) but decided against it, as this view would be less user-friendly. On the other hand, a map allows the user to interact with the different pieces of data, zooming into districts and seeing proximity of facilities to each other. This gives the healthcare administrator a better idea of what is happening at both the facility and district level. Furthermore, to add aggregating district-level information, we designed a choropleth layer to the map, which darkens the color of the district if more vaccines are present. We chose to do this to make the user be able to be read the map quickly and accessibly.

After deciding to pursue the creation of a choropleth map, we immediately began finding JavaScript libraries which could aid us in this. The first one we landed on was called Leaflet.js, a very lightweight library which is commonly used for displaying map data. Leaflet was clearly suitable for our purposes with robust documentation, including an example chloropleth map. Not only did Leaflet allow us to implement interactivity in the map (zoom in, zoom out, panning), it also allowed us to quickly load in map data from shapefiles, which had been provided to us. Furthermore, Leaflet provided a straight-forward way to add markers to the map, which we utilized to show the location of facilities.

As we created the map, we ran into the issue of having to display data for many different vaccine types. It would be inconvenient for the user to see all the vaccine types at once when viewing the map, furthermore, it would not make sense to color the map with respect to an arbitrary, random vaccine. Instead, we decided to implement a dropdown menu, in which the user could select the vaccine type they were interested in. Only then, would the facility and district data load, as they were with respect to the specific vaccine. As mentioned in the related work, workers found it valuable to get live data, so we added a "Refresh Data" button to fetch the data from our backend. We could have not included this button to make the website look more minimalistic, but then the user would have to refresh the entire dashboard. This did not align with our goals, as our dashboard has separate components and the user should only need to refresh the data that they want fresh to minimize bandwidth usage.

After the map was completed, we deemed that function *a* was now fulfilled. We moved onto function *b*, the ability to see vaccine usage over time. To fulfill this we decided to create two components, the line chart component, and the donut chart component.

The primary purpose of the line chart would be to see vaccine usage changes over a flexible period of time. We wanted the user to be able to select any time range, and show them how the vaccine usage changed. As such, we researched JavaScript libraries we could use that focused on data visualization. In the end, D3 seemed to be the most promising due its flexibility and rising industry prominence. Additionally, we wanted the user to have some choice in which lines were drawn. We provided the ability to select whether each line was a district or facility and a drop-down to filter the specific districts or facilities they wanted to include.

On the other hand, the donut chart component is responsible in presenting overall vaccine usage in the country. Aside from the line chart presenting vaccine usage over time, we decided to aggregate data points to summarize the amount of vaccines both administered and remaining at the country level. To provide stakeholders with more relevant information, we decided to only aggregate vaccines used in the past 30 days. To implement this, we again chose D3 as it is an industry standard visualization library.

We then moved onto function *c*, the ability to display transaction data. We wanted this transaction data to be in table format, as there are many pieces of information a transaction includes, such as facility, date, stock change amount, and more. It thus made sense to organize them similar to a SQL table or Excel table. As such, we began examining different JavaScript libraries that effectively display tabular data. An important feature we wanted present in our table would be the ability for the user to quickly sort and filter by any column of the table. As such, the library we settled on was AG Grid.

We decided to keep a very basic design for the transaction table. AG Grid came default with a simple white theme, which we used to display the data after fetching the transactions from the server. One thing to note about the transaction table, is that by default it shows a "past 7 days view." This is because this is the view we thought the healthcare administrator would most likely want to see. However, the table does support the viewing of "all transactions" as well as "past month transactions." These different views are supported through making a different type of request to the back-end server. As for the sorting and filtering feature, our table supports sorting, by simply clicking on the header of a column. We used this method of sorting, as it is most intuitive and user-friendly. Furthermore, clicking on the right side of a column header allows the user to filter a column by a specific keyword. This allows quick removal of non-desired values in the column. For example, if the healthcare administrator was only interested in transactions regarding the "Masindi" district, they can quickly filter the "district" column in that fashion.

After completing the transaction table, we decided to add one more feature which would give the healthcare administrator another view of the vaccine stock data. This view is the "report" view, in which a user can see a facility monthly report, facility weekly report, an error report (which can report any errors recorded by healthcare workers at the facilities), and a "total" report. These reports are convenient for the user, as they can download the CSV file to their system, and share it amongst other relevant parties. This is particularly useful because many preexisting information systems rely on Excel and CSV files. This feature fulfilled both function *a* and function *b*. Within each report, there is vaccine information for each district, such as current stock, spoiled stock, and used stock. As for the front-end design choices, we decided to make these four reports each their own button, and place them at the bottom of the page. Simplicity was the goal here, as we wanted the user to be able to quickly generate these reports without much trouble.

One more thing to note is that we chose to use Bootstrap to generate styles for the buttons on the page, as well as the loading spinner. The reason we chose Bootstrap is that it is a quick and easy way to apply aesthetic styles to commonly used components such as the buttons.

In the end, we designed the front-end with a focus on the three functions mentioned earlier, and we kept it a priority to always consider the user first, and make sure to display vaccine stock information to them in a manner which we thought would suit their needs best. We believe these components of the webpage successfully encapsulate the basic needs of the healthcare administrator, yet we have many ideas for things we could include if given more time (discussed in future work.)

SYSTEM ARCHITECTURE

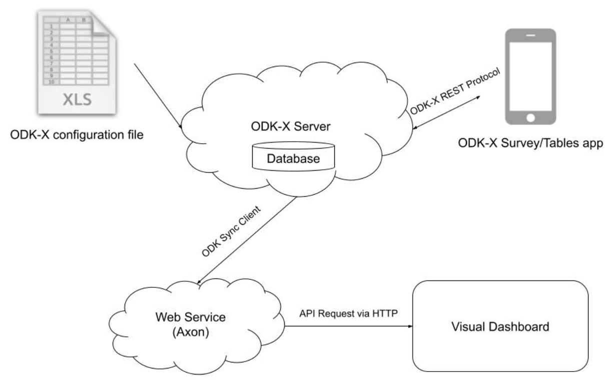


Figure 1. Overall Architecture

The vaccine stock tracking system consists of multiple components. Firstly, to support ODK-X Survey or ODK-X Tables mobile application for data input, an ODK-X server is set up. The ODK-X server is configured using spreadsheets that define the table schemas to be set up in the database of the server.

In order to transmit the data and information from the ODK-X server, we set up a web service server that retrieves data from the ODK-X server using the ODK Sync Client library. The web service server is named after axon, a nerve fiber that transmits information to neurons and muscles. The web service server aggregates, transforms, and stores the transformed data. Then, the web-based visual dashboard retrieves the transformed data via HTTP requests, processes the returned JSON data objects, and visualizes the information.

In this section, we detail the technical implementations of the components: web service for data aggregation and transformation, and web-based visual dashboard.

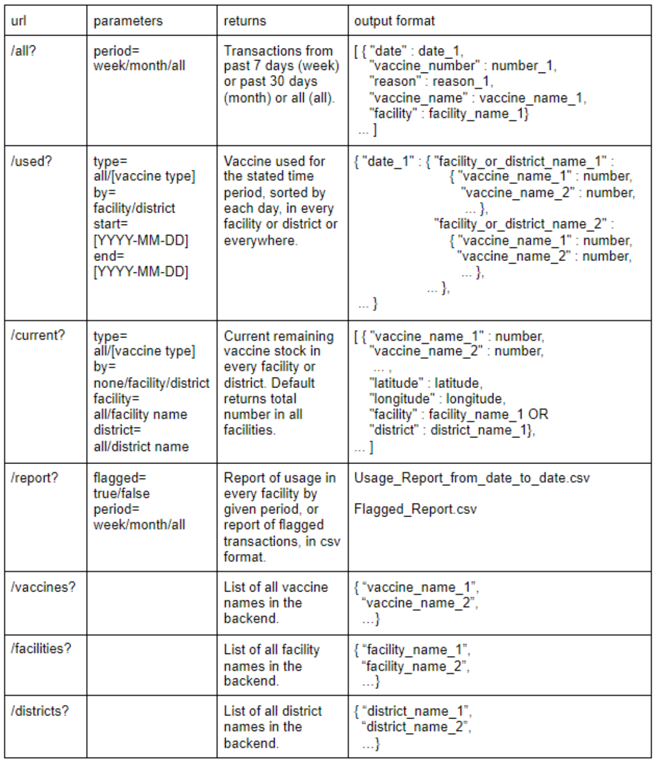
Backend

The backend follows three steps to connect the ODK-X database to the frontend dashboard. First, the backend reads data off of the ODK-X database. Second, the backend processes the read data into a meaningful format. Lastly, the backend transfers the processed data to the frontend in designated format.

ODK Sync Client is used to read from the ODK-X database. The Sync Client establishes a connection to the ODK-X database with given authentication. After the connection is verified, we can retrieve specific data. There are four tables within our ODK-X database, three of which the backend accesses. “facilities” table holds the district, name, and location information for each facility. “vaccine_type” table holds the name information for each vaccine. “stock_transactions” table holds the change in stock, reason for the change, batch ID, reference to a vaccine in “vaccine_type” table, reference to a facility in “facilities” table, and the time information for each transaction. To process the read data, we first read the “facilities” and “vaccine_type” tables to map the reference key to its contents. Then we read the “stock_transactions” table to extract the transaction data; for reference keys, we look at the map constructed before to locate the actual data. For each transaction within the “stock_transactions” table, we create a Transaction object from the read data and add it to our data structure.

Each Transaction object holds the following information: change in stock of the used vaccine, reason for the change, transaction ID, batch ID, vaccine ID, name of the vaccine used, facility and district of where this transaction happened, the location of this transaction, and the start and end date of this transaction. All Transaction objects have a method to create a JSONObject representing the current state, and a single get method to retrieve a field based on string input.

The backend stores the data from ODK-X database in our own data structure. The data structure is kept simple; a list of Transaction objects with ways to handle them. For each transaction added to the data structure, it first checks if the transaction is logically valid. For example, if the reason for the transaction is said to be “administered”, and the change in stock is a positive number, the transaction is not logically valid, thus the data structure will flag the transaction and exclude it. There are only two methods that belong exclusively to the data structure. First is a method to filter the given list of Transaction objects by checking if a specific field of the Transaction object matches the given object. Second is a method to filter the given list of Transaction objects by checking if the transaction happened within the given time period. Rest of the methods within the data structure are helper methods for the endpoints.



url	parameters	returns	output format
/all?	period= week/month/all	Transactions from past 7 days (week) or past 30 days (month) or all (all).	[{ "date" : date_1, "vaccine_number" : number_1, "reason" : reason_1, "vaccine_name" : vaccine_name_1, "facility" : facility_name_1} ...]
/used?	type= all/[vaccine type] by= facility/district start= [YYYY-MM-DD] end= [YYYY-MM-DD]	Vaccine used for the stated time period, sorted by each day, in every facility or district or everywhere.	{ "date_1" : { "facility_or_district_name_1" : { "vaccine_name_1" : number, "vaccine_name_2" : number, ... }, "facility_or_district_name_2" : { "vaccine_name_1" : number, "vaccine_name_2" : number, ... }, ... }, ... }
/current?	type= all/[vaccine type] by= none/facility/district facility= all/facility name district= all/district name	Current remaining vaccine stock in every facility or district. Default returns total number in all facilities.	[{ "vaccine_name_1" : number, "vaccine_name_2" : number, ... , "latitude" : latitude, "longitude" : longitude, "facility" : facility_name_1 OR "district" : district_name_1}, ...]
/report?	flagged= true/false period= week/month/all	Report of usage in every facility by given period, or report of flagged transactions, in csv format.	Usage_Report_from_date_to_date.csv Flagged_Report.csv
/vaccines?		List of all vaccine names in the backend.	{ “vaccine_name_1”, “vaccine_name_2”, ...}
/facilities?		List of all facility names in the backend.	{ “facility_name_1”, “facility_name_2”, ...}
/districts?		List of all district names in the backend.	{ “district_name_1”, “district_name_2”, ...}

Figure 2. Descriptions for our endpoints

We use endpoints to transmit data to the web-based visual dashboard. Most of the above endpoints are handled similarly. First step is to filter the list of Transaction objects within the data structure. The filtering process can be done multiple times, like filtering by facility name and then by given timeframe. Second step is taking the filtered list of Transaction objects and reading data off of each element within the list. The final step is to put the data into the designated format, in most cases, a JSON object or array.

/vaccines, */facilities*, and */districts* endpoints return a list of all vaccine, facility names, or district names within the data structure. The list is constructed from the transactions, thus facilities or districts without any transaction will be omitted from the list.

/report endpoint returns a report in CSV format. There are two types of report. The first type is a regular report. The regular report will show how a vaccine stock was used in a facility, for each vaccine type, and for each facility. The regular report is constructed based off of the given period, which can be past 7 days, past 30 days, or to date. The second type of report is flagged report. The flagged report will show all transactions that were malformed with their corresponding transaction ID. This report can be used to track down malformed transactions and fix them with stock correction. The resulting report will be sent over HTTP to the client browser to be downloaded as an attachment.

Frontend

As for the frontend architecture, each component of the frontend is separated into its own Javascript file. Therefore, we have map.js, donut.js, lineChart.js, and transactions.js. These files are called upon in the index.html file. Each component will be explained separately.

First we have the map component, written in map.js. The code begins with setting up a Leaflet map object, with a default view/zoom of Uganda's coordinates. We then set up some global level variables, which we will explain more in-depth later. The file consists of two main functions: getVaccineChoice() and getData(). getVaccineChoice() is a simple function which is called when the user selects a vaccine from the vaccine dropdown menu. Once the function is called, it sets the global variable, selectedVaccine, to that choice, so we can use that specific vaccine when we make requests to the back-end. The getData() function is called when the user makes a vaccine choice, or when they click the "Refresh Data" button. After, we make a request to the back-end server, to get the current stock of vaccines. We populate a dictionary (one of the global variables) where each key is a district, and each key's value is the amount of available vaccines in that district. After that, we edit our GeoJSON object (which includes the map boundary data for Uganda) to include the vaccine amounts within it. When we make a call to our leaflet-choropleth library shortly after, it can easily read the vaccine amounts straight from the GeoJSON. Now that the district data has been covered, we move onto the facility data. We first make a fetch request to the web service server to retrieve the facility data. The web service returns an array of objects, where each object includes the latitude, longitude, facility name, and amount of a given vaccine present at that facility. We add these markers to the leaflet map so the user can see them. Furthermore, at the beginning of the code, we had a global variable, markers, which kept track of the facility marker references, so that we can clear them after each refresh call. This helps to overwrite the data to present the up-to-date version.

The next part of the dashboard is the line chart. It has a few ways to customize the data displayed. Much like the map, it is using the vaccine type selected by the user. Then, there is a "start date" and "end date" drop-down. These are simple HTML select elements that allow the user to select the time period for which the line chart will display data. After this, the user can choose to visualize the lines as districts or facilities. The district mode is likely useful to regional administrators viewing how different districts are performing whereas the facility mode is more useful to district administrators trying to manage the facilities within their district. From here, they can then select the facilities or districts that they want to display. The quantity of districts or facilities in a country is likely too numerous to be drawn on a single line chart. Thus, the user should filter these to only show certain districts or certain facilities. They can then click the button to refresh the line chart to finish the visualization. This will use a HTTP request to fetch data from the web service, receiving the vaccines administered for each line. The line chart itself is drawn with D3, a Javascript visualization library. We simply reformat the data to work with D3 and draw lines and circles accordingly. The circles will be at each discrete date and the lines are in between.

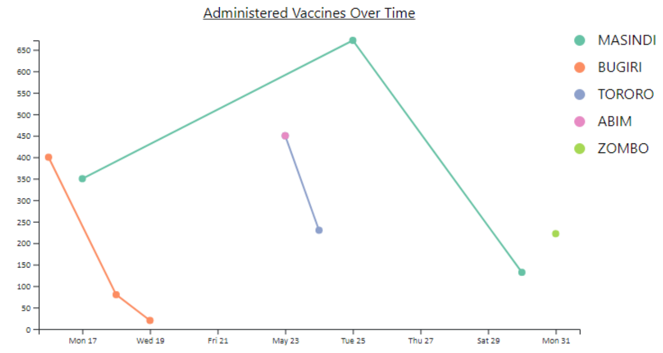


Figure 3. Displayed Line Chart for Specific Districts

As can be seen in the above figure, it's possible that some "lines" only have a single point of data, in which case they just have a circle and no line. If no data exists in the time period (i.e. empty data is returned from the backend), we simply add some text indicating so.

The next part of the dashboard is the donut chart, which is contained in donut.js. This uses the */used* endpoint to get all the vaccines used in the past 30 days and sums them up. It then uses the */current* endpoint to get the current vaccine count for each district and sums those up. D3 is then used to visualize this information, much like the line chart. The result is a pie chart that displays the total vaccines remaining across the country and the number used in the past 30 days.

We then move onto the transaction table, which is included in transactions.js. We first start by defining an array of objects, which contain the names of the columns for the transactions table. This will be passed into the AG Grid object constructor, which abstracts away the table construction for us. Similar to the map.js file, we have two basic functions in this file: getPeriodChoice() and getTransactionData(). getPeriodChoice() will simply set a variable which represents how much data the user wants to see. For example, they can choose between "all", "past week", or "past month" which will specify how many transactions the table displays. After the user makes their choice, we generate the table by calling getTransactionData(). getTransactionData() will first fetch transaction data from the back-end server.

Once we have that array of transaction objects, we slightly edit the data, converting the string data of the date column, to a JavaScript Date object. This is because if the user wants to sort the date column, we do not want to sort the dates alphanumerically (which is what would happen if we left it as a string.) If we convert it to a Date object, AG Grid will know to sort it chronologically. After that, we either update the grid if one has already been created, or create the new grid. Notice that we have a boolean variable which determines if a grid has already been made. This is so that we do not continue making new grids every time the user refreshes the data, and rather we simply update the current rows. Once that is complete, we finally sort the table chronologically, since that is the view the user is most likely interested in.

We then move on to the "report" buttons. As mentioned before, we have 4 buttons: facility weekly, facility monthly, error, and total. Implementing these features was quite straight forward. We simply took the back-end server endpoint which generated the specific report, and put them into an "a" tag in the index.html file. This then takes the user to that URL, the moment they click the respective button. The only thing to note here was that each button would have a separate URL, since the parameters are different for each type of report.

To wrap it all up, we have the index.html file itself. This is the backbone of the webpage. We created a dropdown menu at the top of the page, which will list the available vaccines in the system. To create this, we have a file named dropdown.js which makes a call to the back-end to get all the vaccine names. After this, we have a small HTML div which generates the map, (which is abstracted away from us using Leaflet.) Next, we have a wrapper which contains the line chart. Inside the wrapper we have some inputs where the user can type in the start and end date, as well as refreshing the line chart. Finally, we included a wrapper for the transaction table as well (with its corresponding time period dropdown menu) as well as the four report buttons.

DISCUSSION

The vaccine stock tracking system presented an interesting technical challenge. As a group, we have relied on and built on top of the ODK-X framework which evidently abstracted some technical details and allowed for quicker development. Then, we explored other technologies and developed a web service server, and a web-based visual dashboard with data visualizations. The development process allowed us to learn existing technologies and tools such as the ODK-X framework, Spring framework, HTML, CSS, Javascript, D3, Leaflet, and AG Grid. We also become familiar with many aspects of software development such as user-centric design, iteration, error-checking, and integration between multiple components.

Beyond this, we demonstrated the design process for a system attempting to be independent of its deployment context. While our prototype is modelled off of Uganda, we explored the way in which a domain-independent technology–such as ODK-X–can serve as the foundation for a domain-dependent but location-independent system, like ours. It is evident both from previous attempts and our own experience that any such system is founded with a lack of assumptions, compared to normal immunization information systems. However, the overall design of an ODK-X mobile application, a backend webserver, and a web dashboard can be further generalized to other domains, providing what hopefully emerges as a blueprint of such development.

Future Work

Given more time and resources, the vaccine stock tracking system can certainly be expanded and improved on. Firstly, tracking expiration and efficacy of vaccine batches can be implemented with related data aggregation and transformation functions on the web service server as well as data visualization on the visual dashboard. Secondly, for each transaction, changes to the vaccine doses can be averaged over the transaction period rather than the having the changes committed on the end date of the transaction period. Thirdly, an onboarding and authentication process can be set up to allow multiple API keys for multiple parties to access relevant data from the vaccine stock tracking system through the web service server.

The web service server currently uses a customized data structure to store aggregated and transformed data. Another possible approach to be explored is to represent data in a relational format and store them in a relational database such as MySQL. The web service server can then serve data request by making relevant queries to the relational database and retrieving requested data. This would require a drastic redesign of the web server.

There are many areas in which the mobile application could be fleshed out if provided additional time. The first category of them is ease of use improvements. One such improvement involves the "facility stock" table, which was mentioned earlier. Being able to use the app to see the estimated count in each facility would be more useful to individual facility administrators when they are handling their error correction. The current path requires them to visit the dashboard to consider such adjustments. With our current system, this change would require us to introduce another flow of data from the backend to the ODK-X server as the backend would need to push this updated stock data. This would be a significant modification in the design of the system. Another useful improvement would be to consolidate the entry of multiple transactions into a single form. That way, they can select a single facility and date range and add an arbitrary number of transactions, each with that period. This could be a common use case if entering the vaccines used, spoiled, and received for some period. Presently, the user would need to choose the facility and date range for each type of change in the vaccine.

The other category of changes would move into expansions to the current system. One useful change would be taking advantage of the various authentication levels of ODK-X. Right now, our app has binary permission levels. If you are not signed in, you have no access to the app nor its data. And signing in gives full access to both. Setting up a hierarchy of permission levels would make sense. For example, a facility worker should not be able to add new facilities or new vaccine types. This would help lock down pieces of the system to ensure changes do not introduce errors. The most important change would be to flesh out the localization. ODK-X has built-in translation handling, but this requires the app developers to input the translation options for the various pieces of text. As of now, the app would only work in countries in which all healthcare workers are expected to know English. This is not even the case for our model country of Uganda, so it would be necessary for this change to take place before any deployment.

Overall, the next stage of this system would be to continue to develop some of these features and put it in deployment. Getting proper user experience and feedback would make a better case for the existence of such a system that could be deployed to any country.

CONCLUSION

As low-and-middle-income countries continue to face challenges with managing and maintaining health information systems, it is important to consider solutions which can target these challenges effectively. Through careful consideration of these countries' needs, we have developed a system which can effectively update vaccine stock information, at the facility, district, and country-wide level. This information is updated by on-site healthcare workers through a mobile app built with ODK-X, and information flow eventually reaches healthcare administrators through the visual dashboard. In the future, improvements and expansions can be added to the system to make it field-ready such as implementing expiration date tracking into the system.

REFERENCES

[1] Waylon Brunette, Clarice Larson, Shourya Jain, Aeron Langford, Yin Yin Low, Andrew Siew, and Richard Anderson. 2020. Global Goods Software for the Immunization Cold Chain. In *Proceedings of the 3rd ACM SIGCAS Conference on Computing and Sustainable Societies (COMPASS '20)*. Association for Computing Machinery, New York, NY, USA, 208–218. DOI:http://dx.doi.org/10.1145/3378393.3402278

[2] Tracy Colburn. 2020. A Quality Improvement Initiative to Manage Shingrix Vaccine Inventory in Ambulatory Clinics. *DNP Scholarly Projects* (2020).

[3] Marie-Pierre Gagnon, Amelie Lampron, and Ronald Buyl. 2016. Implementation and Adoption of an Electronic Information System for Vaccine Inventory Management. In *2016 49th Hawaii International Conference on System Sciences (HICSS)*. IEEE, 3172–3178.

[4] Holly Groom, David P. Hopkins, Laura J. Pabst, Jennifer Murphy Morgan, Mona Patel, Ned Calonge, Rebecca Coyle, Kevin Dombkowski, Amy V. Groom, Mary Beth Kurilo, Bobby Rasulnia, Abigail Shefer, Cecile Town, Pascale m. Wortley, Jane Zucker, and the Community Preventive Services Task Force. 2015. Immunization Information Systems to Increase Vaccination Rates: A Community Guide Systematic Review. *JPHMP* 21, 3 (2015), 227–48.

[5] Richard Heeks. 2008. ICT4D 2.0: the Next Phase of Applying ICT for International Development. *Computer* 41 (07 2008), 26 – 33. DOI: http://dx.doi.org/10.1109/MC.2008.192

[6] UOA MS. 2004. Hepatitis B vaccine. (2004).

[7] Mercy Mvundura, Laura Di Giorgio, Dafrossa Lymo, Francis Dien Mwansa, Bulula Ngwegwe, and Laurie Werner. 2019. The costs of developing, deploying and maintaining electronic immunisation registries in Tanzania and Zambia. *BMJ global health* 4, 6 (2019), e001904–e001904.

[8] PATH National Expanded Program on Immunization. 2020. Overcoming the Challenges and Adoption for New Users in Electronic Immunization Registry Implementation- A Case Study From Vietnam. (2020).

[9] Samia Razaq, Amna Batool, Umair Ali, Muhammad Salman Khalid, Umar Saif, and Mustafa Naseem. 2016. Iterative Design of an Immunization Information System in Pakistan. In *Proceedings of the 7th Annual Symposium on Computing for Development (ACM DEV '16)*. Association for Computing Machinery, New York, NY, USA, Article 9, 10 pages. DOI: http://dx.doi.org/10.1145/3001913.3001925

[10] Thiago Dias Sarti, Welington Serra Lazarini, Leonardo Ferreira Fontenelle, and Ana Paula Santana Coelho Almeida. 2020. What is the role of primary health care in the COVID-19 pandemic? (2020).

[11] Alexandra J. Stewart and Phillip M. Devlin. 2006. The history of the smallpox vaccine. *Journal of Infection* 52, 5 (2006), 329–334. DOI:http://dx.doi.org/https://doi.org/10.1016/j.jinf.2005.07.021

[12] Geoffrey Tabo, Walter Yagos, and Emilio Ovuga. 2015. Knowledge and attitudes of doctors towards e-health use in healthcare delivery in government and private hospitals in Northern Uganda: A cross-sectional study. *BMC Medical Informatics and Decision Making* 15 (11 2015). DOI: http://dx.doi.org/10.1186/s12911-015-0209-8

[13] Laurie Werner, Dawn Seymour, Chilunga Puta, and Skye Gilbert. 2019. Three Waves of Data Use Among Health Workers: The Experience of the Better Immunization Data Initiative in Tanzania and Zambia. *Global health science and practice* 7, 3 (2019), 447–456.