Helping Children Eat Well Via Mobile Software Technologies

Violetta Vylegzhanina, Douglas C. Schmidt, Pamela Hull, Janice S. Emerson,
Meghan E. Quirk, and Shelagh Mulvaney

Vanderbilt University, Nashville Tennessee, USA
{violetta.vylegzhanina, douglas.c.schmidt, pam.hull, shelagh.mulvaney}@vanderbilt.edu
Tennessee State University, Nashville, Tennessee, USA
{jemerson, mquirk}@tnstate.edu

Abstract

This paper describes an Android mobile app we developed to simplify the shopping experience of participants in the Special Supplemental Nutrition Program for Women, Infants, and Children (WIC), which provides low-income families vouchers to purchase life-stage appropriate, nutritious foods. Our app helps alleviate the tedious and error-prone use of paper WIC vouchers by allowing participants to scan food items in the store and automatically identify if an item (including its size and quantity) is authorized for the enrolled WIC participant. In addition to serving as a shopping tool, the app also provides a platform for nutrition education through healthy tip push notifications and a gallery of easy-to-fix snack recipes that are tailored for WIC participant shopping choices and other racial/ethnic patterns in dietary preferences. This paper explains the key domain and technical challenges we faced when creating our Android app and describes how we overcame these challenges by applying data normalization, software patterns, and Agile development methods.

Categories and Subject Descriptors D.2.11 [**Software Engineering**]: Software Architectures; H.2.1 [**Database Management**]: Logical Design; J.3 [**Computer Applications**]: Life and Medical Sciences

Keywords Mobile app; Android; software design patterns, Open mHealth; Agile; data normalization; data analysis.

1. Introduction

The *Special Supplemental Nutrition Program for Women, Infants, and Children* (WIC) [1,2] is administered by the *Food and Nutrition Service* (FNS) of the *United States Department of Agriculture* (USDA). The WIC program provides federal grants to states for supplemental foods, health care referrals, and nutrition education for low-income pregnant, breastfeeding, and non-breastfeeding postpartum women, as well as to infants and children up to age five who are at nutritional risk.

As part of the USDA-funded Nashville *CHildren Eating Well* (CHEW) project, the goal of our work presented in this paper was to develop an Android app to help parents and guardians of WIC-enrolled children shop more efficiently and effectively, in an effort to promote healthy snacks and beverages in the home environment. This paper explores challenges in both the domain of the WIC program and the technical approaches we applied to the CHEW app development. We also present our solutions to these challenges based on applying database schema design and normalization, software patterns, and Agile development methods. Finally, we outline our future work on developing data analytics that analyze WIC participant shopping data to personalize healthy tips for users and to help evaluate the impact of the app on enhancing nutrition education.

The remainder of the paper is organized as follows: Section 2 describes the WIC domain and its key domain challenges; Section 3 summarizes the CHEW app architecture and implementation; Section 4 explains how we addressed the key WIC domain challenges; Section 5 explains how we addressed the key technology challenges; Section 6 compares our work on the CHEW app with related work; and Section 7 presents concluding remarks.

2. An Overview of the WIC Domain and Key Domain Challenges

The WIC program focuses on improving the health and nutrition of low-income pregnant women, breastfeeding mothers, and infants, and children under the age of five. Currently, WIC serves over 50 percent of all infants born in the United States. The USDA sets the overall requirements for the WIC program, but leaves some areas flexible for the state-level agency to make select decisions about how to implement the program [1].

WIC participants in the state of Tennessee receive two types of paper vouchers on a monthly basis: a regular voucher and a cash value voucher [2]. Regular vouchers list the participant's food package based on their eligibility category and age (e.g., pregnant woman, breastfeeding mother, infant, 3- or 4-year old). Regular vouchers limit the quantities, sizes, and brands of specific WIC-approved products for each WIC-enrolee. Participants normally receive two regular vouchers per month, with roughly half of the food package on each voucher. Cash value vouchers, in contrast to the regular voucher, provide a dollar amount limit ($6 or $10) for the purchase of fresh or frozen fruits and vegetables.

The shopping experience using paper vouchers is complicated since each WIC-enrolled family member receives regular vouchers for different food packages. Some products (e.g., peanut butter, baby food, infant formula) are included in certain food packages but not others, depending on the life stage of the participant. Moreover, each food package has different limits on quantities and sizes of items in the regular vouchers. Some food items can be chosen either uniquely or in any given combination of each. For example, some regular vouchers allow WIC participants to choose either a maximum of three non-fat dry milks or any

SIGPLAN'05 June 12–15, 2005, Location, State, Country.

combination of three buttermilks, evaporated milks, and tofu. For other items, like cereal, multiple items can be selected based on the total number of ounces in all of the packages.

Cash value vouchers are provided in different dollar amounts depending on the eligibility category of the family members. For the cash value voucher, rather than quantity, size, or items, users have a maximum dollar amount to use, so the selection of items depends on the price. Using cash value vouchers requires math proficiency to calculate and keep track of the prices of various possible combinations of loose produce priced by weight, priced by the piece, or priced by the package. This process can be cumbersome, especially for participants with low numeracy skills.

WIC participants often face delays when checking out at the store since each voucher must be its own transaction. Families with multiple WIC-enrollees must separate items for each participant before checking out. Once at the register, cashiers must determine if an item is authorized for the particular voucher(s) used. Moreover, each paper voucher can only be used during one shopping trip. These factors contribute to suboptimal benefit utilization where WIC participants often do not purchase all the nutritious foods that they are allowed to get on each voucher.

The mobile software technologies presented in this paper were designed to simplify the shopping experience of WIC participants. Addressing these challenges motivated the following requirements and constraints on our software solution for the CHEW app:

- *Minimize user involvement* − WIC participants are often frustrated while shopping because they must manually keep track of everything they buy, which is tedious and error-prone. Likewise, participants with low numeracy skills struggle to calculate prices of fruit and vegetable produce items. One goal of our Android app was therefore to automate and simplify users' shopping experience as much as possible.
- *Automate the tailoring and delivery of nutrition education via the app* – WIC participants currently receive nutrition education during their quarterly appointments at the WIC clinic when receive their paper vouchers. It is also beneficial, however, to provide supplemental nutrition education content to participant in between appointments via the smartphone app. Our Android app thus automatically encourages guardians of WIC-enrollees to serve healthy snacks and beverages to their young children (and possibly the whole family) by providing informative tips and easy-to-prepare snack recipes.

3. Overview of the CHEW App Architecture and Implementation

Before describing our solutions to the domain challenges summarized in Section 2, this section presents an overview of the CHEW app architecture, which is shown in Figure 1. A cloud-based server runs Open mHealth data storage services [5] (discussed further in Section 5.3), which provides an open architecture that improves integration among mobile health solutions, storing the list of WIC-approved items in a MongoDB database [6] (Step 1).

When the CHEW app is run for the first time, it retrieves the data containing WIC-approved items from the Open mHealth cloud server (Step 2) and caches a copy of this data on the smartphone in an Android Content Provider [3,4] (Step 3). This Content Provider stores the data locally in an SQLite database [7] (Step 4), which encapsulates the data and manages its access by the user-facing portion of the CHEW app.

When the app is used during a shopping trip, it stores the users' shopping choices in the Content Provider. These data are periodically synchronized with the cloud-based server (Step 5),

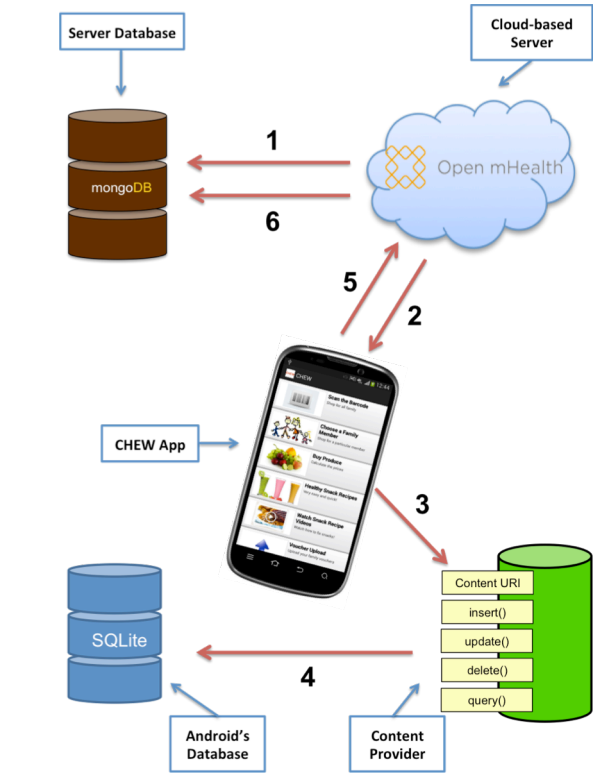

Figure 1: Architecture of the CHEW App.

which stores it in a MongoDB database for subsequent analysis of participant shopping choices (Step 6).

The CHEW app is written largely in Java, with some use of embedded SQL statements in the Content Provider to access the WIC and user data stored in the SQLite database. The number of source lines of code in the CHEW app is ~8k and the number of classes is ~70, excluding 3rd party libraries.

4. Resolving WIC Domain Challenges

As outlined in Section 2, the primary domain challenges faced by WIC benefit users were the difficulty in selecting and tracking selected WIC-approved items and the difficulty of calculating prices of fruit and vegetable produce due to the low numeracy skills of most participants. Another challenge was to provide the supplemental nutrition education via the CHEW app. This section describes how we applied principles of Agile software development to help us solve these challenges efficiently and effectively.

4.1 Applying Agile Development Methods

We applied Agile software development principles while working on the CHEW app to achieve an iterative and incremental development, which enabled a high degree collaboration with stakeholders, flexible responses to change, and (ultimately) higher quality and efficacy of the resulting software. We employed weekly sprints that consisted of design, development, and testing. At the end of each sprint we met with key project stakeholders to demonstrate our progress in the CHEW app development and receive/triage their feedback.

We also met often with WIC participants to demonstrate and test our prototype, e.g., we conducted several shopping trips with users to test the app in an actual deployment environment at local grocery stores. These short sprints and periodic meetings with

stakeholders increased the flexibility and quality of the CHEW app prototype, e.g., we detected and corrected bugs and enhanced the code more easily as development proceeded incrementally.

Figure 2 depicts the structure and example milestones in the CHEW app development process. We applied a spiral model to incrementally introduce ourselves to the WIC program and gain a deeper understanding of the domain problem we needed to solve. We simultaneously planned app features that helped simplify the shopping experience for WIC families and automate and customized key portions of their nutrition education. We also had to learn and master Android software features needed to create the app and resolve the challenges in the WIC domain.

Figure 2 also shows several examples of the spiral process as we applied Agile principles of software development. In particular, as we gained a deeper understanding of Android and Open mHealth features, we were better equipped to design appropriate software implementations, which in turn helped us provide more effective solutions for WIC participants.

4.2 Minimizing User Involvement

As discussed in Section 2, WIC benefit users currently have a complicated experience since they must manually check to ensure they select approved WIC items during their shopping trips, as well as manually keep track of the items' prices and quantities. Below we describe how we designed the CHEW app to automate and simplify users' shopping experience as much as possible.

4.2.1 Automating the WIC Shopping Experience

The CHEW app needed a means to enable participants to transfer all the information about a particular food item to the device, notify the user that the item is approved to buy, and then keep track of its quantity. We concluded that scanning a product's barcode would be the most effective way to automate these tasks. We therefore programmed the CHEW app to store a database of barcodes for WIC-approved items, together with all the necessary metadata about the items, such as their names (e.g., whole milk) and quantities (e.g., 16 ounces). By matching a scanned barcode with the barcode stored in a database on the smartphone the app would retrieve the information it needs.

Developing a barcode scanning tool ourselves would reinvented solutions that already exist, so we explored available packages and chose ZXing ("Zebra Crossing") [8], which is an open-source multi-format barcode image-processing library. As discussed in Section 5.2, one limitation we faced during the CHEW app development was supporting app features in the absence of network connectivity since we could not rely on WIC families having a data plan or Wi-Fi access within the grocery stores. Using ZXing library was advantageous here because it used the built-in camera on mobile devices to scan and decode barcodes on the device, without the need to communicate with a remote server.

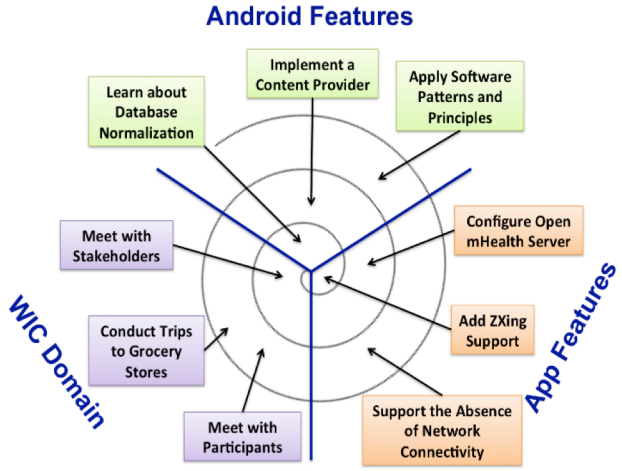

Figure 2: Development Process for the CHEW App.

Figure 3. App's Main Screen.

To support different user preferences and simply CHEW app use, we allowed WIC participants to shop in two different ways: (1) either shop for the whole family at once or (2) shop for each family member separately. They could interchange these shopping methods any number of times during their shopping.

Figure 3 shows the main screen of the CHEW app. A user can click the "Scan the Barcode" button, where after scanning an item the app would display all family members who could get this item based on the information in their vouchers. A WIC shopper could then select the number of items to get for each family member. Conversely, the shopper could select items for a specific family member by clicking the "Choose a Family Member" button. In this case, they would be presented with a *ListView* [4] of family members, where selecting a specific member opens up the member's profile and allows the shopper to view the items this particular member can get, what this member already bought, and an option to scan items specifically for this member.

While a user scans items, the CHEW app queries the Content Provider that stores data for the items and vouchers in the SQLite database to check whether an item is approved and if its limit has been reached. Programming this logic in the app was one of the hardest technical challenges we faced, as discussed in Section 5.1.

4.2.2 Providing the Fruit and Vegetable Produce Calculator

As discussed in Section 2, many WIC benefit users have limited numeracy skills, which make it hard for them to calculate the prices of fresh fruits and vegetables and keep track of the prices to know when they reach the limit specified on the cash value vouchers for each family member. As a result, many participants often do not use the cash value voucher to the full extent. We therefore had to devise a solution to help WIC participants get the most out of their cash value vouchers.

A WIC cash value voucher specifies the dollar amount up to which participants can buy fruits and vegetables, which include packaged fresh produce, packaged frozen produce, loose produce

with a price per item, and loose produce with a price per pound. We created a separate Android Activity (which is a single, focused thing that a user can do, such as interacting with the user via the touchscreen interface on a mobile device) to deal with the cash value voucher. As shown in the middle of Figure 3, a user can click on the "Buying Produce" button to open up an Activity that presents the user with information about the total price allowed for the family, total amount already spent, and total amount left, as well as amounts for each family member separately. The user is also presented with options for buying four different kinds of produce, e.g., packaged fresh, packaged frozen, loose produce with a price per item, and loose produce with a price per pound.

One goal of the CHEW project is to analyze the shopping data to study the impact of the CHEW app on WIC participant purchasing choices. It was therefore important for the CHEW app to collect the names of all items that participants buy, while still minimizing user involvement. Meeting this goal was difficult when dealing with produce since prices change often and this information is not stored in the smartphone database. As a result, we were unable to fully automate this process in our prototype—a more comprehensive solution would likely involve close interactions with electronic pricing systems used by grocery stores participating in the WIC program.

Since packaged fresh and frozen produce have barcodes, we allowed users to simply scan the packages to get the product names. They then entered the prices and selected the quantities of products and the family members these products apply to. Since participants can scan the packages, they were also able to shop for the packaged produce portion of the cash value voucher when they were scanning items from their regular vouchers. If they scanned a produce item, they would simply get transferred to the Activity for handling produce.

Since it was infeasible to scan loose produce (which lack standard barcodes), participants had to manually enter the names of the items. Typing the whole name of an item would be cumbersome for a user, so we used the autocomplete option supported by Android's *AutoCompleteTextView* [4] class. We implemented this feature by creating a custom *CursorAdapter* [4] that implements a *Filterable* [4] interface and connecting it to an *AutoCompleteTextView*, which then displayed to users the suggested names of fruits and vegetables stored in the Content Provider.

Shopping for "loose produce per item" requires a user to enter the price of an item. Likewise, shopping for "loose produce per pound" requires a user to weigh the produce and calculate the price. Once again, our goal was to make this process as simple for users as possible. Since users had to weigh the items on a scale, we reasoned that finding a widget to mimic an actual produce scale would be the most intuitive interface. We selected the Android *wheel* widget [9].

Figure 4. Produce per Pound Calculator.

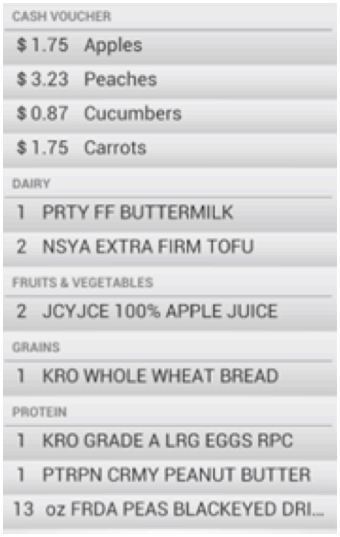

Figure 5. Items Selected Activity.

Figure 4 shows a closeup of how we use this widget in our CHEW app. Users can scroll the left wheel to choose the pounds as they see them on the actual scale, and they can scroll the right wheel to select the increments. They could then select the family member who gets the item and enter the produce name and its price per pound. Due to minor discrepancies, the app would calculate only the approximate cost of produce, but our periodical meetings with WIC participants indicated that this was still quite helpful to them.

As WIC participants shop for produce, the CHEW app keeps track of the prices and updates the activity's screen with the dollar amounts spent and left on the voucher. Participants can therefore concentrate on their shopping, rather than calculating and managing prices manually, which is tedious and error-prone.

4.2.3 Automatically Keeping Track of Selections

The CHEW app keeps track of all food items that WIC participants choose, including the quantities and the sizes of each. A WIC shopper can go to each family member's profile and view items selected for each member. Figure 5 shows an Activity that displays chosen products in a *ListView*. We wrote a custom *SimpleCursorAdapter* [4] to display the items in a similar format as presented on the vouchers (shown in Figure 7).

For example, vouchers specify limits either on the number of items, on the number of ounces, or on the dollar amount. Our custom adapter therefore leverages different *ListView* row layouts to display either the number of items, the ounces, or the dollars. *ListView* separators also help partition items into categories.

4.3 Automating the Tailoring and Delivery of Nutrition Education

In addition to assisting WIC families with their shopping experience, as described above, the CHEW app is also intended to provide nutrition education, as described below.

4.3.1 Providing Healthy and Engaging Snack Recipes

Our CHEW app allows parents and guardians to view and select recipes of easy-to-prepare healthy snack recipes for their children. Each recipe contains a limited number of steps, shown via both text and pictures to make it more entertaining and easier to follow.

To further simplify this process, the CHEW app reminds participants to choose some of the recipes before each shopping trip.

When users select a recipe, its ingredients are automatically uploaded into a shopping list that will be used during a shopping trip. These ingredients do not necessarily correspond to the items listed on the WIC vouchers, and participants can buy them with the vouchers or separately.

To make the CHEW app more engaging, it provides recipe videos that show how to prepare healthy snacks. Although not all participants can watch the videos as they are streamed over the Internet, we included this option so users could use it whenever they can. Figure 6 shows Android Activities that enable users to view/choose healthy snack recipes and watch recipe videos.

4.3.2 Sending Healthy Tip Notifications

In addition to providing a gallery of healthy snack recipes, another component of nutrition education emphasized by the CHEW project involves healthy tips provided through notifications. The CHEW app stores a list of general healthy tips in its Content Provider. Examples of these tips include helping parents to deal with children who are picky eaters (e.g., parents may need to offer a new food as many as 15 times before their child will eat it), appropriate portion sizes for pre-schoolers, and distributing healthy snacks throughout the day.

The CHEW app uses Android *Service*, *BroadcastReceiver*, and *AlarmManager* [3,4] components to send notifications to users at specific time intervals (e.g., every other day, etc.). In future work, we will perform data analytics to customize notifications for each participant, depending on participant's ethnic background and purchase data, as discussed in Section 7.

5. Resolving Technical Challenges

We faced several technical challenges during the design and implementation of the CHEW app. Some challenges arose due to our inability to use efficient Android features, as we had to support the absence of network connectivity. This section describes these key challenges and presents our solutions.

5.1 Dealing with Complicated and Unstructured Data That May Change Frequently

Problems. During the early stages of the CHEW app development we realized that the app would need to store and process a great deal of data locally on the smartphone device due to the lack of network connectivity discussed in Section 4.2.1. In addition to shopping data, we had to store data containing product descriptions from participating grocery stores and data containing vouchers' descriptions, which was particularly complicated. Below we describe techniques for overcoming problems involved with storing and processing these data efficiently.

Figure 6. Viewing Recipes.

A key challenge we faced was how to store and process descriptions of the vouchers that specified limits on the food brands, quantities, and sizes. This information was not presented in a common format and it contained complicated logic. Figure 7 shows three examples of such vouchers. The voucher at the top of the figure allows a user to either get up to a specific quantity of a particular food, such as 36 oz cereal, or up to a specific number of a particular product or up to a specific number of a different product, but not both, e.g., 3–11.5 to 12 oz frozen or 3–46 to 48 oz containers of WIC approved juice.

A more complicated example is when a user can get either some amount of a particular item or any combination of items, but not both, e.g., 1–3 quart box nonfat dry milk or choose 3 (any combination) of these: 1 quart buttermilk, 1 can evaporated milk, 14–16 oz tofu. Each user can also get a certain dollar amount worth of produce items, which is given to them in the form of a separate type of cash value voucher.

The voucher in the middle of Figure 7 specifies different logic, e.g., this voucher does not allow a user to get a combination of items. Finally, the voucher at the bottom of the figure is completely different from the previous two; it specifies foods for infants, depending on whether the infant is fully or partially breastfed or formula fed. The types and the amounts of foods an infant can get also depend on infant's age.

The descriptions of vouchers shown in Figure 7 are highly unstructured, which poses a challenge to storing these data in a structured manner in a relational database, such as SQLite, while preserving the logic that goes into descriptions. Moreover, vouchers can be divided and users could either (1) use both vouchers in one shopping trip or (2) they can split each during two different shopping trips. Moreover, if users receive their vouchers later than when they were supposed to receive them, they would get different vouchers, containing reduced amounts of allowed foods.

Another challenge was linking the products' data from stores with the descriptions on the vouchers for the app to correctly identify if users' choices are correct, which was the critical requirement. For example, the data sets provided by grocery stores contained only specific food names and broader food categories, such as dairy, while vouchers specified limits on different food types, such as milk or cheese, within a food category, such as dairy. We had to find a solution with which the CHEW app understood the connection between the store data and the voucher data to allow an app user to make appropriate choices and keep track of the chosen quantities and the sizes of the food items.

Finally, the WIC approved items in each store often change, as well as the information on the vouchers, e.g., some food types get added and some get removed, and the allowed quantities also change. We therefore needed a solution that would require minimal changes whenever store data sets and vouchers were updated.

Solution. We used data normalization to organize data in the Android's SQLite database. We applied third normal form (3NF) [10,11] to split large amounts of data into small tables. Although this solution yielded a proliferation of tables, we were able to provide much more structure to data that initially appeared quite unstructured, such as the different descriptions of vouchers. 3NF made our database more flexible by eliminating redundancy and inconsistent dependency. In particular, it ensured that each table

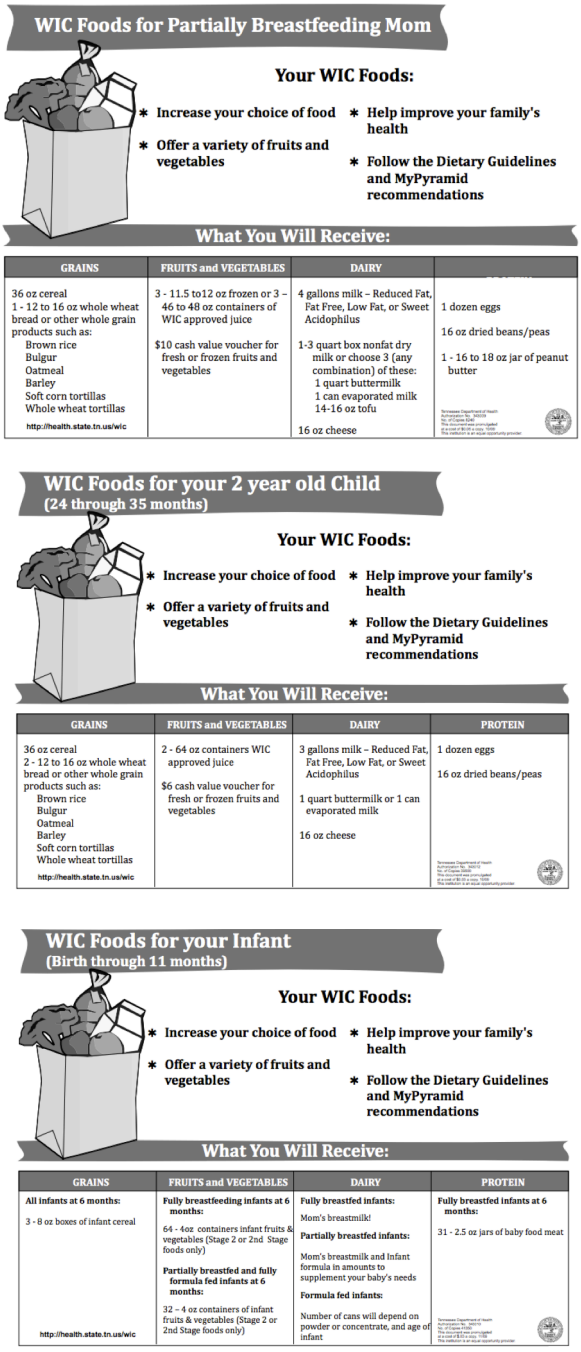

Figure 7. Examples of WIC Vouchers.

had a key, where the non-key attributes were dependent only on the key and no other attributes.

Design considerations. Applying 3NF excessively is not often practical. For example, having many small tables may degrade performance. We therefore applied 3NF only to data that changed frequently, such as descriptions of vouchers. Below we describe the process we used to organize the unstructured vouchers' data in an Android's SQLite database.

Design process. Figure 8 shows a portion of the CHEW app's SQLite database schema that handles data for a store and the vouchers. We created identifiers for all possible vouchers (voucher_type). Since vouchers could be split, we treated those vouchers as different vouchers and created separate identifiers. If users specified they want to use the whole voucher at once, the app would simply perform table joins when querying the database. We used similar logic for the vouchers with reduced amounts of products that participants receive when they come late to pick up the vouchers.

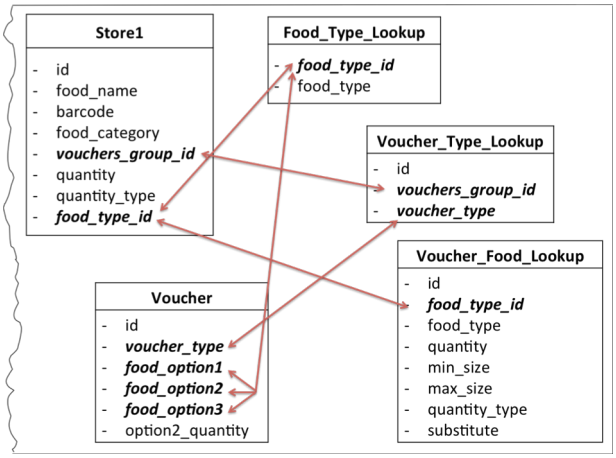

Figure 8. Portion of Database Schema.

Since a particular food item can apply to several vouchers, we also created a unique identifier for a group of voucher identifiers that this item applies to (vouchers_group_id). For example, a '1' could represent vouchers V1, V2, and V4, that all allow buying bread. We also created identifies for different food types, such as cereal or milk (food_type_id).

Unfortunately, we had to manually put these identifiers and the unique identifier for a group of vouchers into the databases received from participating grocery stores (automating this process is planned in our future work, as discussed in Section 7). We then created a lookup table to match food identifiers to the actual names of the food types (Food_Type_Lookup) and a lookup table to match vouchers group identifier to the identifiers of each single voucher (Voucher_Type_Lookup). These lookup tables allowed us to create linkage between data from the stores and the data in the vouchers, which was a key feature for making the CHEW app address the domain requirements presented in Section 2.

To organize the data and allow for uniform database queries, we created a table (Voucher) that contains each voucher identifier and three additional columns to specify possible food options (maximum of three at present), even if voucher descriptions did have all three of the options. If a voucher specified an option for a food type, the row would contain the identifier for that food type (otherwise, it would contain a negative identifier). The hardest part of our solution was handling data specifying that some foods can be bought in a combination. We thus identified a food type option that might allow for combinations (food_option2) and created a separate column in the table, specifying a number of combination items (option2_quantity). If combinations were not allowed for a particular food type, the column contained a zero.

We also created a separate table with food type identifier as a key (Voucher_Food_Lookup); this table specified the allowed quantities and sizes of food types. We again had to devise a structured way to incorporate data regarding which items in the table could be bought in combination. To solve this problem, we created a separate column to contain links to other food types in the same table (substitute).

For example, some vouchers allowed buying buttermilk, evaporated milk, and tofu in any combination of three. If we identified a buttermilk as '7', evaporated milk as '8', and tofu as '9', then '7' would point to '8', '8' would point to '9', and '9' would point to '7'. If combinations were not allowed, the column contained negative identifier. This design allowed us to query the Voucher table to check if a number exists for the combination of items, as well as use this information to query this table to see which actual food types could be bought in a combination.

While majority of the vouchers were relatively similar, the vouchers for infants were different, as shown in Figure 6. We therefore separated information on these vouchers into separate database tables (not shown in the figure).

Outcome. Having organized the unstructured vouchers' data in the SQLite database, our CHEW app had to use complicated queries involving multiple table joins to get the necessary information. Applying 3NF, however, reduced the possible number of changes to the database and the code that we would make in the future by minimizing the dependencies between data. This approach also decoupled the data and the logic for using the data by specifying the data in a database (even the data for the logic), but actually performing the logic programmatically.

5.2 Supporting the Absence of Network Connectivity

Problems. Many families participating in the WIC program lack data plans or Internet connectivity at home, which posed challenges for providing certain CHEW app features, such as downloading images of recipes from a remote HTTP server and streaming recipe videos. Our goal was to ensure that guardians and their children would view these recipes and that the recipes would be easy to follow.

To be more engaging, the recipes needed to contain pictures of each step, ideally presented as step-by-step high-resolution videos. To avoid requiring WIC participants to devote all their smartphone memory to storing this content, the videos need to be streamed and the pictures need to be downloaded from a remote server and cached on a device, which motivates the need for network connectivity.

Since we could not rely on WIC families having Internet access, we therefore had to ensure that the CHEW app would operate properly by providing engaging recipes, without being network-dependent. One option was to exclude pictures of every recipe step and to only include the images of a limited number of recipes that the app could store locally on the device. Reading recipe steps in plain text, however, would be less compelling for participants and would require extra user involvement. We therefore had to display pictures of each recipe and each recipe step to make it more appealing to users, but had to find a solution to do so locally on the device.

Solution. Since it is problematic to store a large number and sizes of images, we had to find a compromise with our stakeholders. We ended up using only a small number of recipes, where each recipe contains at most two steps, thus at most two pictures, and where all the images are small. The CHEW app stores the images in the Android resources (res) folder and the images' paths in the SQLite database.

Outcome. While this solution is not the most efficient way to deal with images on Android, it was necessary given the constraints we faced. As ubiquitous access to Internet connectivity improves, we hope to revisit the full potential of more efficient ways to store, retrieve, and display images.

We also left the recipe videos option for the users even though they might not be able to watch the videos often. We used the YouTube API [12] to stream the videos; a user can view videos' thumbnails displayed in a *ListView* where clicking a thumbnail would show the video to the user (Figure 6). In future work we may plan to have a single Activity that displays a recipe with its picture and video. If Internet connectivity is present, we will display a video thumbnail for the user to watch a recipe video if necessary and if connectivity is not present, we will simply display a recipe image only.

5.3 Finding an Efficient Way to Analyse Data

Problems. A key goal of the CHEW project is to collect and evaluate participants' shopping data to access the impact of nutrition education provided through the app and make necessary adjustments to the program. When WIC shoppers use paper vouchers, however, it is infeasible to collect their shopping choices to analyse this information for purposes of nutrition studies. We therefore needed a solution that would allow efficient data analytics to aid nutrition studies.

Solution. To implement this solution, we employed Open mHealth [5], which is an open architecture designed to improve integration among mobile health solutions. The Open mHealth platform fosters collaboration between software developers, clinical experts, and health researchers to addresses the problem of drawing meaning and scientifically valid inferences from collected mHealth data, often involving lots of bias and variability, by presenting more sophisticated and effective tools for data visualization and analysis [13].

Open mHealth overview. As shown in Figure 9, Open mHealth helps to evolve the mHealth ecosystem from a silo'd architecture to a "narrow-waisted" architecture, where a common communications protocol acts as a simple point of commonality at the narrow waist, and where innovation flourishes through open interfaces or APIs, both above and below the waist.

Open mHealth provides the following features [14]:

- *Software reusability* to aid in the development of new applications through sharing of software components amongst software developers and health innovators.
- *Standardization*, by providing a shared set of open APIs for back-end data stores to enable client software with a uniform way to access data.

Figure 10 shows the main Open mHealth components, which are defined as the modular software units outlined below:

- *Data Storage Unit (DSU)*. A DSU provides a uniform way to access data through a series of API calls.
- *Data Processing Unit (DPU).* A DPU represents a unit of work to be performed on JSON data. It is the building block for extracting relevant features from data streams.

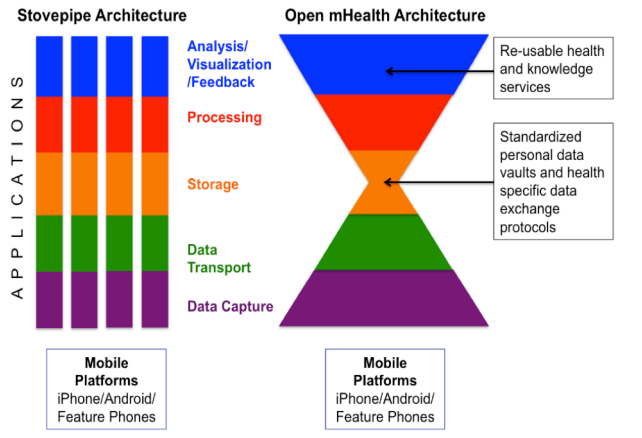

Figure 9. Stovepipe vs. Hourglass Architecture.

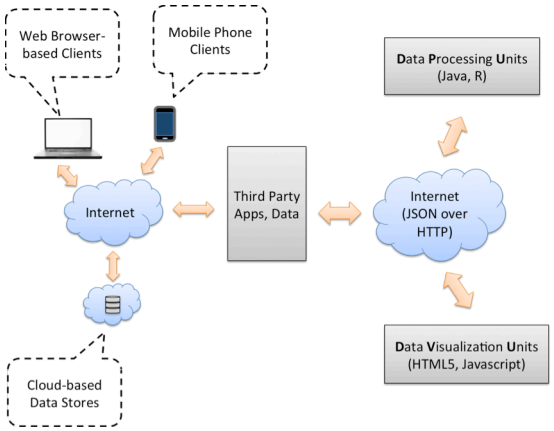

Figure 10. Open mHealth Components.

- *Data Visualization Unit (DVU).* A DVU enables the visual presentation of features processed by a DPU and patterns.

Open mHealth visualization modules can combine data from many mobile apps and devices. Likewise, evaluations modules can be embedded directly into an app to improve clinical impact.

Open mHealth configuration. We hosted a cloud-based server using Amazon Web Services [15]. As shown in Figure 1, the server runs a MongoDB database that interfaces with Open mHealth. The server database stores the products data from the grocery stores, which are uploaded to the SQLite database on a mobile device via Android's *IntentService* [3,4] when the app first runs. The app can periodically uploads the users' shopping data back to the server for future analysis, assuming users can obtain network connectivity, e.g., during their visit to the WIC clinic.

Outcome. By hosting a cloud-based server that runs Open mHealth, we created a platform that allows efficient processing of collected data to derive useful insights for action. Open mHealth also promotes easier sharing of data across platforms, which will enhance future project iterations.

5.4 Applying Software Patterns

Problems. As mentioned in Section 5.1, we expect frequent changes to our CHEW app, e.g., as vouchers and products get added, updated, or removed from the program. We expended considerable effort in making the database schema flexible to change, as shown in Figure 8. We also designed our app to require minimal changes to the code as requirements change by applying software principles and patterns by both using the patterns ourselves and by applying Android frameworks that leverage patterns to enhance code reuse and make the app more flexible to change.

Solutions. We used the commonality and variability [16] analysis to separate the varying parts of the application from the non-varying parts. For example, we had to represent users of the app and their vouchers. We concluded that the vouchers users receive represent the varying parts since they may get different ones every time and since vouchers' descriptions often change. Thus, applying the Commonality and Variability analysis, we identified that the *Strategy* pattern [17] would help decouple interfaces from implementations so the implementations could vary without affecting client code that used the common interfaces.

Applying *Strategy*, we created an abstract class *Member*, representing an app user, and several concrete classes that inherit from *Member*, which represent that actual user types, such as 3 to 4 year old child (Figure 11). We then created interfaces for regular

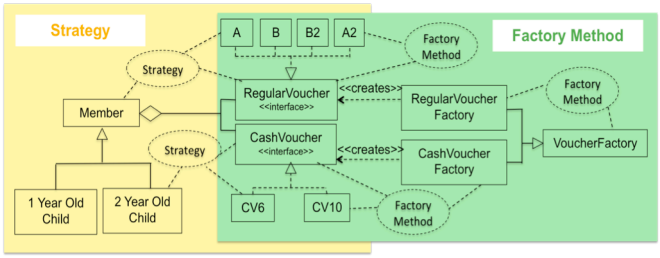

Figure 11. Strategy and Factory Method.

voucher and cash value vouchers, and several classes representing a specific voucher that implement one of the interfaces.

Using the *Strategy* pattern, where each concrete *Member* class is composed with appropriate vouchers, we were also able to apply the *Composite Reuse* principle (CRP) [18], which favors composition over inheritance. This design increased the flexibility of our CHEW app, both by encapsulating vouchers (the family of algorithms) and by allowing the users to change vouchers they want to use at runtime.

As discussed in Section 5.1, users could spend all the vouchers during a single shopping trip or divide them between separate trips. To maximize flexibility, we applied the *Factory Method* pattern [17] to allow CHEW app users to select vouchers they want to use at runtime. This pattern encapsulates the creation of the types of vouchers by letting subclasses decide which vouchers to create. We applied this pattern to create an abstract class *VoucherFactory* and the two concrete classes that inherit from it, the *CashVoucherFactory* and the *RegularVoucherFactory* that create the specific types of vouchers.

We also used many Android frameworks in our CHEW app that themselves leverage software patterns to aid in code reuse and ease the application development by encapsulating tedious and error-prone details from app developers. For example, our application uses *AsyncQueryHandler* [4] to execute Content Provider operations asynchronously without blocking the UI since *AsyncQueryHandler* provides callback methods that are executed after the completion of Content Provider operations.

AsyncQueryHandler is implemented using the *Proactor* and *Asynchronous Completion Token* patterns [20] (Figure 12). It uses the *Proactor* pattern to simplify asynchronous application development by splitting an app's functionality into async operations (such as database queries) and completion handlers that use the results of asynchronous operations to implement the app's business logic. Likewise, it uses the *Aysnchronous Completion Token* pattern to allow an app to efficiently demultiplex and process the responses of asynchronous operations. By applying these two patterns together, *AsyncQueryHandler* allows decoupling app's independent asynchrony mechanisms from app specific functionality, and simplifying event-handling algorithms.

Our CHEW app also uses Android *AsyncTask* framework [4] to simplify the creation of long running tasks, such as querying

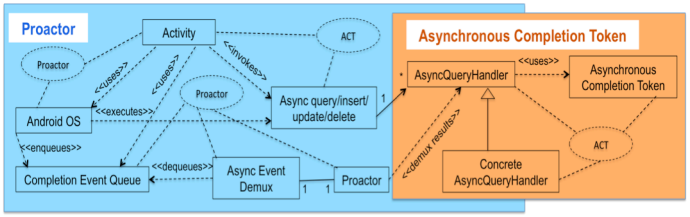

Figure 12. Proactor and Asynchronous Completion Token.

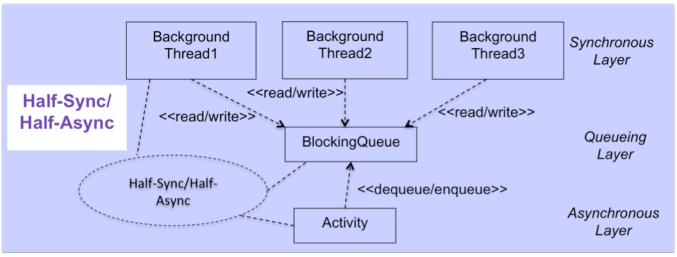

Figure 13. Half-Sync/Half-Async.

the database, without the need to communicate with Threads and Handlers. The *AsyncTask* framework leverages the *Half-Sync/Half-Async* pattern [20,21] to integrate synchronous and asynchronous operations in an efficient and well-structured manner by decoupling these operations to simplify concurrent programming while supporting execution efficiency (Figure 13). Decomposing overall system intro three layers (i.e., synchronous, asynchronous, and queueing), AsyncTask centralizes the inter-layer communication because the queueing layer mediates all interactions; and synchronization policies in each layer are decoupled such that each layer may use different concurrency strategies.

We use Android's *IntentService* framework to pull data from a remote Open mHealth server to the app's SQLite database on the app's first run. The *IntentService* framework implements the *Command Processor* pattern [19] to encapsulate the request for a service, the data download request in our case, as an object that is passed to a Service to execute (Figure 14). The use of *Command Processor* ensures that the client's thread is not blocked for duration of command processing, and that different users can communicate with a service in different ways via commands. Thus, the same code can be reused for multiple purposes.

Outcome. Applying software patterns and Android frameworks that leverage software patterns enhance the maintainability and flexibility of our CHEW app. Android frameworks also allowed us to focus on the app requirements and minimize development time by abstracting away many low-level implementation details. In addition, they increased code reuse since we could simply extend their behaviours to support different app needs.

6. Related Work

This section compares our work on the CHEW app with related research and development efforts. The San Diego WIC App for Android [22] and iOS [23] notifies users with updates about WIC activities, promotions, and events. It also provides information on healthy eating and recipes made from WIC approved foods.

The WIC Calculators Android [24] and iOS [25] app is designed for WIC agencies, but not for actual participants, to aid in determining infant formula issuance amounts, identifying products that meet minimum specification requirements for whole grain/whole wheat products.

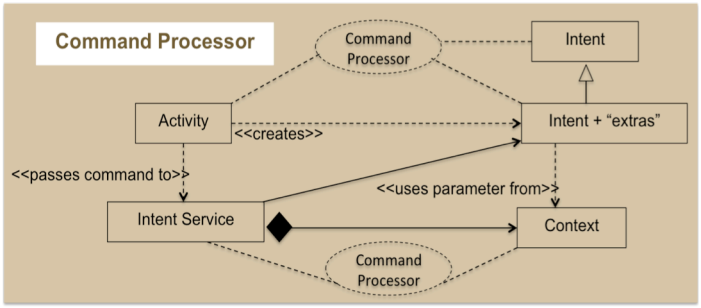

Figure 14. Command Processor.

The EBT Shopper Android and iOS app [26] simplifies WIC participants' shopping experience by allowing participants to determine WIC eligible items as they scan products in a store, and to list the items that can be purchased based on the WIC food package. It is used in states with electronic WIC benefits (not paper vouchers). It is not clear from the available documentation how this app keeps track of each individual person's selections and WIC voucher limits.

The aforementioned apps provide only a limited support to WIC participants compared to the app we are creating. They either serve as a shopping tool (The EBT Shopper app) or as a nutrition education tool (The San Diego WIC app), or they are not even used by WIC participants directly (The WIC Calculators app). Our app attempts to give the most benefits to WIC participants by serving both as a shopping tool and as a nutrition education tool, simplifying and automating the previously cumbersome shopping experience of participants and helping them to have healthier lifestyles.

7. Concluding Remarks

This paper described the CHEW app, which is implemented on the Android and Open mHealth platforms to simplify the shopping experience of participants in the Tennessee WIC program and provide nutrition education. WIC participants in Tennessee currently use paper vouchers to get supplemental nutritious foods. These types of vouchers are challenging for participants to use since they must manually keep track of the products they are get and the quantities of each, which is tedious and error prone.

Many WIC participants also have issues with numeracy, so they often do not use their cash value vouchers to buy produce since they cannot easily calculate and keep track of the prices in the store. Nutrition education provided in between WIC appointments is meant to encourage and support healthy lifestyle choices.

Our CHEW app helps alleviate problems with paper vouchers by allowing participants to scan products in the store and automatically identify if they can get a product based on their voucher prescription. The app also keeps track of its allowed quantities and sizes and provides an easy-to-use calculator of the prices of produce items to help participants use their cash value vouchers to the full extent. The CHEW app also serves as a nutrition education tool by providing healthy tips and engaging snack recipes.

Our experience in developing and applying the CHEW app thus far has yielded the following lessons learned:

- **Maintaining consistency between databases manually is tedious and error-prone.** We had to manually edit the database tables received from participating grocery stores to link them with the tables that store vouchers' descriptions (we manually put the identifiers for different food types and voucher groups). We want to automate this process in future work, e.g., by employing machine-learning techniques to assign values in a column of a table based on the observed data in other columns. We plan to study the feasibility of this solution and evaluate its ability to produce accurate results.
- **Data analytics solutions are needed to provide tips that are customized for each user.** Our current CHEW app provides general healthy tips for each user. This feature can be improved to enhance nutrition education by personalizing tips based on user's shopping choices and ethnic background. For example, participants could be encouraged to buy more apples and carrots to receive a sufficient amount of dietary fiber that improves digestive health and extends the feeling of fullness. Working with our stakeholders, we also learned that ethnic

background plays an important role when deciding which foods should be avoided to stay healthy. Our future work will therefore develop data analytics solutions to analyse user's shopping history, considering user's ethnic background, to provide tips tailored for each user, which will further enhance nutrition education.

- **Assessing the impact of the CHEW app on enhancing nutrition education.** As described in Section 5.3, we integrated the Open mHealth platform to provide effective means of visualizing the collected participants' shopping data to aid nutrition studies in evaluating the impact of the app on enhancing nutrition education. Although we spent a lot of time understanding Open mHealth components and integrating them into our solution, much work remains to study the various components of Open mHealth, such as Data Visualization Units (DVUs), to aid nutrition studies in evaluating the collected data more efficiently.
- **Leveraging mobile Augmented Reality to further simplify participants' shopping experience.** Mobile Augmented Reality is a new approach to visualizing cyber information on top of physical imagery [27]. In future work we plan to support WIC vouchers for special user groups consisting of participants with food allergies, lactose intolerance, and gluten intolerance. Since these participants have more limited options of the items than the non-special category participants, they might not easily find the approved products in the store. We therefore plan to use mobile Augmented Reality to help the participants with special voucher categories locate approved items faster in the grocery store [27,28]. For example, participants would point their phone at the store shelf, and the app would automatically recognize all the gluten-free products and mark them for the user.
- **Supporting automatic store detection.** Our CHEW app currently asks users to manually select the grocery store where they plan to shop. In future work we plan to enhance the app to automatically detect the appropriate store based on its location. The app could store the locations of participating stores and detect a location to identify a particular store when a user comes to shop. This capability would require network connectivity, however, which we cannot support consistently at present. As more WIC participants gain Internet access, however, this feature will become more useful.
- **Supporting multiple platforms.** We are currently developing the CHEW app for Android platform. This choice poses limitations because we cannot support participants who use other mobile operating platforms. In future work we are therefore planning to create a hybrid app that embeds HTML5 inside a thin native container to combine elements of both native and HTML5 apps (and thus allow both online and offline connectivity) to support multiple mobile operating platforms.

Acknowledgments

This Project was supported by Agriculture and Food Initiative Grant #2011-68001-30113, from the USDA National Institute of Food and Agriculture, Integrated Research, Education, and Extension to Prevent Childhood Obesity program – USDA-NIFA-AFRI-003327.

References

1. USDA: United States Department of Agriculture. Food and Nutrition Service: Women, Infants and Children (WIC). www.fns.usda.gov/wic/about-wic-wic-glance. Accessed Jan 2014.
2. Department of Health: About WIC. http://health.state.tn.us/wic/. Accessed Jan 2014.
3. Meier, R. *Professional Android 4 Application Development.* John Wiley & Sons, 2012.
4. Murphy, M. L. *The Busy Coder's Guide to Android Development.* CommonsWare, 2013. commonsware.com/Android/.
5. Open mHealth. openmhealth.org. Accessed Jan 2014.
6. mongoDB. www.mongodb.org. Accessed Jan 2014.
7. SQLite. www.sqlite.org. Accessed Jan 2014.
8. zxing: Multi-format 1D/2D barcode image processing library with clients for Android, Java. code.google.com/p/zxing. Accessed Jan 2014.
9. Android-wheel: Android Picker widget. code.google.com/p/android-wheel/. Accessed Jan 2014.
10. Kreibich, J. A. *Using SQLite.* O'Reilly Media, 2010.
11. Microsoft Support: Description of the database normalization basics. support.microsoft.com/kb/283878. Accessed Jan 2014.
12. Google Developers: YouTube Android Player API. developers.google.com/youtube/android/player. Accessed Jan 2014.
13. Chen, C., Haddad, D., Selsky J., Hoffman, J. E., Kravitz, R. l., Estrin, D. E., and Sim, I. Making Sense of Mobile Health Data: An Open Architecture to Improve Individual-and Population-Level Health. *Journal of Medical Internet Research 14,* 4 (2012), p10. www.jmir.org/2012/4/e112. Accessed Jan 2014.
14. Open mHealth: Developers. openmhealth.org/developers. Accessed Jav 2014.
15. Amazon Web Service. aws.amazon.com/. Accessed Jan 2014.
16. Coplien, J., Hoffman, D. and Weiss, D. Commonality and Variability in Software Engineering. *IEEE Software 15,* 6 (1998), 37-45.
17. Gamma, E., Helm, R., Johnson, R. and Vlissides, J. *Design Patterns: Elements of Reusable Object Oriented Software.* Addison-Wesley, 1994.
18. Knoernschild, K. *Java Design: Objects, UML, and Process.* Addison-Wesley, 2001.
19. Buschmann, F., Meunier, R., Rohert, H., Sommerlad, P. and Stal, M. *Pattern-Oriented Software Architecture Volume 1: A System of Patterns.* John Wiley & Sons, 1996.
20. Schmidt, D., Stal, M., Rohnet, H. and Buschmann F. *Pattern-Oriented Software Architecture Volume 2: Patterns for Concurrent and Networked Objects.* John Wiley & Sons, 2000.
21. Schmidt., D. C. and Cranor., C. D. Half-Sync/Half-Async: An Architectural Pattern for Efficient and Well-structured Concurrent I/O. *Proc. 2nd Pattern Languages of Programs '95*.
22. San Diego State University: California WIC. SDSU WIC Android App Screenshots. www.sdsuwic.org/wic-program/wic-san-diego-android-app-now-available.html. Accessed Jan 2014.
23. San Diego State University: California WIC. Install WIC San Diego iPhone App. sdsuwic.org/wic-program/wic-san-diego-iphone-app-coming-soon.html. Accessed Jan 2014.
24. Google play. WIC Calculators. play.google.com/store/apps/details?id=com.bluepanestudio.FormulaAndWholeGrainCalculator&hl=en. Accessed Jan 2014.
25. iTunes Preview. WIC Calculators. itunes.apple.com/us/app/wic-calculators/id501212303?mt=8. Accessed Jan 2014.
26. EBT Shopper: WIC shopping, simplified. http://ebtshopper.com. Accessed Jan 2014.
27. Jules White, Douglas C. Schmidt, and Mani Golparvar-Fard, "Applications of Augmented Reality," *IEEE Proceedings* Special issue on Applications of Augmented Reality, 2014 (to appear).
28. IBM Research. Augmented reality makes shopping more personal: New mobile application from IBM Research helps both consumers and retailers. www.research.ibm.com/articles/augmented-reality.shtml. Accessed Jan 2014.