Expiration Tracker Report

Yue Zhang and Mengfan Zhu
Department of Computer Science and Engineering
The Ohio State University
Columbus, OH 43210
Email: {zhang.8016, zhu.2420}@osu.edu

I. Introduction

Food and medicine wastage is a worldwide issue and needs to be addressed. Busy lifestyles and poor planning are reasons that most of us don't remember when the product is about to expire or has already expired. The research shows that about 150,000 tons of food are tossed out in US households every day due to expiration [1], which equals to almost one pound per person waste. Hence a mobile application has been proposed by solving this issue, which keeps track of the expiration dates of products and notifies the user when a product is about to expire so that the wastage will be reduced.

Expiration Tracker Android Application provides customers several useful features. Firstly, the user can add a product with the expiration date, description, quantity by either scanning the barcode through camera or entering name manually after signing up and logging into their account. Additionally, the user can create their own categories and categorize their items into different categories and set up notification time for each category. They can always edit their products' and categories' information. The application will notify users of those products that are about to expire at a certain time and certain frequency based on users' requests. Besides, the Expiration Tracker Application provides an incredibly quick way for users to keep track of all the items' expiration date they added. With these features, people can use this application to remind themselves of the expiration time of those products and that can help them manage the products they have.

II. Design Process

The first step in this process was to identify the nouns that could potentially translate into classes or other structures in a programmatic space. Some nouns such as ”User”, ”Item”, ”Category” were found as three main classes.

Several nouns and verbs related to User class are user account(username, name, password), add, edit, delete, login, logout, and register. That means the user can register an account with their email and login to their own account. They can edit their name and password but not their username(i.e. email used to login). Users are allowed to create, update, and delete their own categories, such as Food, Medicine, and inside each category, they can create, update, and delete their items as well.

For category class, its functions can be known by some nouns and verbs including category name, reminding frequency, date and time to start to notify, and some basic operations including add, edit, delete. Therefore, the user can set the reminding frequency for each category, which is the number of times the user is notified about their products. They are free to choose to get notices every day, every two or three days, every week, or every two weeks. Users can also choose a date to start notification which is the number of days before expiration. Various options are provided for different kinds of categories, which include one or three days in advance, one or two weeks in advance or one month in advance. Considering different schedules for different users, several notification times are provided so that users can choose to get notified at any time during the day.

Besides, here are some nouns and verbs for item class: item name, description, quantity, expiration date, and basic operation(add, edit, delete). Thus, inside a specific category, users can add, edit, or delete an item's information. On the home page, the expired and non-expired items will be listed separately and sorted based on their expiration date so that users can find the expired items directly and also easy to find the items nearly to be expired.

Table 1: Original Domain Classes

Domain Class	Responsibility	Collaborator
User	-Register an account -login to account -logout to account -update password -update name -List all categories -Add/Edit/Delete category information -List all items -Add/Edit/Delete item information	-Category -Item
Category	-Create a category -Edit a category name -Edit when to begin the reminder for the category -Edit reminder frequency for the category -Edit reminder time for the category -List all categories -Delete a category(include all items in it)	-User -Item
Item	-Create an item under a category -Scan the barcode to obtain item name -Edit item name -Edit item quantity -Edit item expiration date -Edit item description -List all expired items and not expired items -List items under a category -Delete an item	-User -Category

For this application, JSON tree in Firebase has been used to store the data, and each node in the JSON tree is a data representation with an associated key. [2] JSON tree uses nesting structure, and Firebase allows at most 32 levels deep of the nesting data. [2] However, since the nesting structure can influence search performance, it's necessary to flatten the data structure, in other words, split data into separate paths to reduce the nesting level. Therefore, instead of making categories as a child node of the user and items as a child node of each category, we make categories and items separated from users.

Overall, our JSON tree has three children including users, categories, and items. For users, it just has several children and each child represents a user. Each user has an ID as the associated key which is automatically generated by Firebase. For categories, each child node has the user ID and includes all categories belongs to that user. Similar to the category, in the item tree, each child node associates with one user with the user ID. Under each user, several categories are listed as the children nodes and inside each category node, all items are listed belongs to one specific category for that user. All the user IDs in the item and category trees are corresponding to the user IDs (i.e. the associated key for each user generated by Firebase [2]) in the user tree.

The attributes of users, categories, items class are the same as described above with only two differences. First, for security reasons, we don't store the password in the database. Instead, we use the Firebase Authentication for the register, login, and logout functions. The Firebase Authentication [2] uses the

Scrypt algorithm to encrypt the password which enhances the security of the user account. Besides, we added an attribute event id for items as a helper for reminders.

III. Translating Design to Implementation

After completing the design of the application, Android Studio [3] has been decided to use for the whole project. The Java programming language has been used for app development.

Firstly, we created the register and login page so that users can create their account log in to their account to add items. Users need to enter the password twice to register for security. Firebase Authentication [2] has been used to implement register and login. As Expiration Tracker doesn't include critical data, it's not necessary to make a too complex way to do authentication. Therefore, user can use the email address and password to register and login which is a basic way in Firebase Authentication. The password should have at least six digits. Once the user login to their account, they can create their own categories and items. They can also go to the setting page to change their name and reset the password. Changing the password would need the second confirmation as well. The app will do auto login after the registered success. It will also check the user's authentication state every time the user starts the app and does auto login when launching the app, if the user has login before and hasn't logout yet.

From there we began to implement the categories and items parts. For the category editing page, we created a text box for category name so that the users can enter any category name they want. Empty is not allowed for the category name. The notification beginning date is created as a dropdown menu. Users can choose one of them from the list based on his/her requirement and the default date is one day in advance. The frequency of notification is a set of radio button and getting notified every day would be the default choice. The notification starting time is implemented by the clock widgets in Android Studio [3] so that the user can choose anytime to get notified. This layout made it a lot more user friendly and easier to format.

The item editing page is actually created in a similar way. The difference is that the expiration date is created by the time widget and the default setting would be the current date, which would be efficient for users to pick up an expiration date. The quantity uses a text box with only a positive integer allowed. Any other non-integer input would not be allowed to enter. The text box is also used for description but it is optional so the user can still add an item without any description. If the item is already expired or not need to be recorded anymore, users are allowed to delete or edit the information anytime. Once the user adds an item or saves the changes, the information would be stored in the database automatically.

Once we have done the category and item edition part, we began making views to show the user data. We have created the category list view and edit list view. Our category list view simply displays a list of the category with all the category information to the user with an edit button and a delete button to change the data. Double clicking a category could lead the user to the item list page, where listed all items belong to that category. Similar to the category list page, the item view showed all the items with their information and each with an edit and delete button as well. On the item page, there is another scan button to allow the user to scan the barcode to get item information.

The implementation of scanning the barcode can be divided into two steps. First, using the camera to scan the barcode and get the barcode number. Second, using the barcode number to search on the Internet to get the item name. To get the barcode number from scanning, we used a third-party library [4] which integrates the zxing library and provides the scan barcode function. To get the item name, we need to use the barcode number obtained to search in a third-party database. After some search and comparison, we decided to use the Barcode Lookup website [5] to get the item name. The process is that we combined the home page URL of this website and the barcode number to get the URL of a web page for a specific item, and then use the HttpURLConnection library in Android [3] to get the content of this website. After that, we use the regular expression to get the item name from the content. After the item name obtained, the app will jump to the item edit page with the item name filled in. If the web doesn't have information about that item which means failing to find the corresponding item name or the user doesn't give permission of using the camera, the app will jump to the empty edit page to let the user input the item name and other information.

The reminder is an important part of the Expiration Tracker Application. A problem we need to handle is that the users may not want this app to always run in the backend of their mobile phone, and that may lead to failing to receive a reminder from the app. Therefore, we decided to add the reminders in the system calendar and let the system calendar to give the reminders so that users can always get reminders. We used the CalendarContract library in Android [3] When there is a new item added. There will be a corresponding calendar event be created and the event id will be store in the Firebase database. Then, a calendar reminder will be inserted in that event. And when the user edits or deletes an item or category, the corresponding events can be found by event id and the reminder will be changed according to the changes made by the user.

Besides getting notices from the calendar, the users can also track the expiration dates easily from the home page. On the home page, all the items created by the user will be listed and sorted by their expiration dates. The expiration date for each item has been assigned as a variable in Item class so that it is easy to compare the value with the current date to check if the item is expired. All expired items will be marked in red to catch their attention.

After finishing the homepage, we added the navigation bar to make the application more user-friendly. The navigation bar is created in the activity in Android Studio [3]. By calling OpenFragment() method for different fragments, such as category list or setting, etc, the navigation page would automatically load different fragments on it. Thus, on each page, it would show the navigation bar at the bottom so users can go to each page easily.

We also realized a back button for this app is necessary so that the user could easily go back to the previous page. The main problem we need to solve is that if the user did some edit before, we always want the user to go back to the updated previous page. As the relation of our pages is clear, we just set a parent page for each page and let the app show the parent page when the user clicks the back bottom.

After completing all functional requirements, it's also important to handle some non-functional requirements to improve this application.

One important non-functional requirement is performance. For the performance part, we mainly improved memory usage. We used the Android [3] profiler to analyze the memory usage when the app is running. The biggest problem was that we had some memory leaks which could make the usage of memory increase dramatically. This problem is mainly caused by inappropriately rewriting OnStop function. After realizing the problem, we use a firebase event listener and remove the event listener in the OnStop method to prevent a memory leak when reading data from the Firebase [2]. Besides, when we use Internet to jump between activities, we need to finish the previous activity in the OnStop method. After solving the memory leak problem, the memory usage of our app is reduced by about 30%.

We also solve the thread-safe problem. At first, we change the UI directly from threads, which is not safe. Therefore, we use runOnUiThread method and Runnable object to change the UI.

Reliability has also been improved to make sure the app won't crash when the user refuses to give the permissions we need. The permissions include two parts. We need permissions for reading and writing the system calendar for our reminder and permission for the camera for the scan part. As the usage of the system calendar is an important part of the app, we apply for permission when the user first opens the app, and if the user refuses to give the permission, we just quit the app. For the camera permission, as the scanning is not a necessary function, we apply for the permission when the user uses the scan function. And if the user refuses to give permission, the app just jumps to an empty edit page.

Beyond those, firebase security has also been checked. Since each registered user has been authorized in firebase, by the default setting, all users would have permission to access all data, which would cause potential security problems if they access other users' data. Therefore, a firebase rule has been added so that for each user, he/she can only read and write the part of the data that belongs to him/her. Therefore, users can easily access and modify their data without knowing others.

Last but not least, another non-functional requirement that has been solved is the network connection. The network is checked by ConnectivityManager in Android Studio [3]. If the network is not connected, users cannot read or write any data through the database, so an empty page with the navigation bar would be shown. Whenever the networking connection gets back, clicking any button on the page would reload the page.

After finishing all non-functional requirements, testing has been added to make sure the app will run correctly in different scenarios. The UI test cases have been created for most of fragments and activities to ensure the functionalities. JUnit tests are also created to test main classes and attributes.

IV. Suggested Changes and Improvements

In our current design, we used the Firebase Realtime Database [2] for the data storage. We chose Firebase because, in that way, the users won't lose data if there are problems with their own devices and they also can get their data when they change devices. However, when we began thinking about non-functional requirements, we found using the Firebase Realtime Database could have a problem, whenever the app needs to read or write data, it needs to connect to the Firebase Database, which needs to use the network connection. Therefore, our app uses the network very often and that can influence the performance of the app.

Therefore, we think a better approach is to have both the local database and the Firebase Realtime Database. Normally, when the users use this app, they actually just change the data in the local database. And we can provide the user with a ”synchronize” function, and that function allows the users to synchronize the local data with the Firebase data. That means the user can back up their local data to Firebase and when there are some problems with their devices or they want to change devices, they can just get the data from Firebase and won't lose their previous data. In that way, we avoid the frequent network service request, and that can improve the performance of our app.

References

[1] O. Milman, “Americans waste 150,000 tons of food each day – equal to a pound per person.” [Online]. Available: https://www.theguardian.com/environment/2018/apr/18/americans-waste-food-fruit-vegetables-study

[2] Firebase. [Online]. Available: https://firebase.google.com/

[3] Android, “Android open source project.” [Online]. Available: http://developers.android.com

[4] yuzhiqiang1993, “zxing.” [Online]. Available: https://github.com/yuzhiqiang1993/zxing

[5] B. LOOKUP. [Online]. Available: https://www.barcodelookup.com/