Inventory Tracker
Design and Testing

Contents

Application Design

Class Design

The basic structure of the application (pictured below) is as such:

- App: The App class (type Application) is the starting point and core of the application.
- MainPage: The MainPage class (type MasterDetailPage) is the primary View for the Application. It is, therefore, dependant on the App instance that it is related to.
- Views: The Views (type ContentPage) provide the majority of the user interface for input and output between the user and the application. Views are often dependant on MainPage; however, some views are dependant directly on App, rather than a MainPage instance.
  In addition to displaying graphical controls, Views also handle user events and interface with the backend components.
- Database: The Database stores all of the user data as records in tables. It is dependant on the Application to be accessible, but can exist even when the current instance of the application is terminated.
- Database tables:
  - TierItem is the base class for storing user data objects. TierItem (and subsequently, and derivative thereof) has a one-to-many relationship with any ChildItem type objects.
  - User is a derived class of the base class TierItem.
  - ChildItem is a derived class of the base class TierItem, and is a base class for the Item type.
  - Item is a derived class of the base class ChildItem, which is in turn a derived class of the base class TierItem.

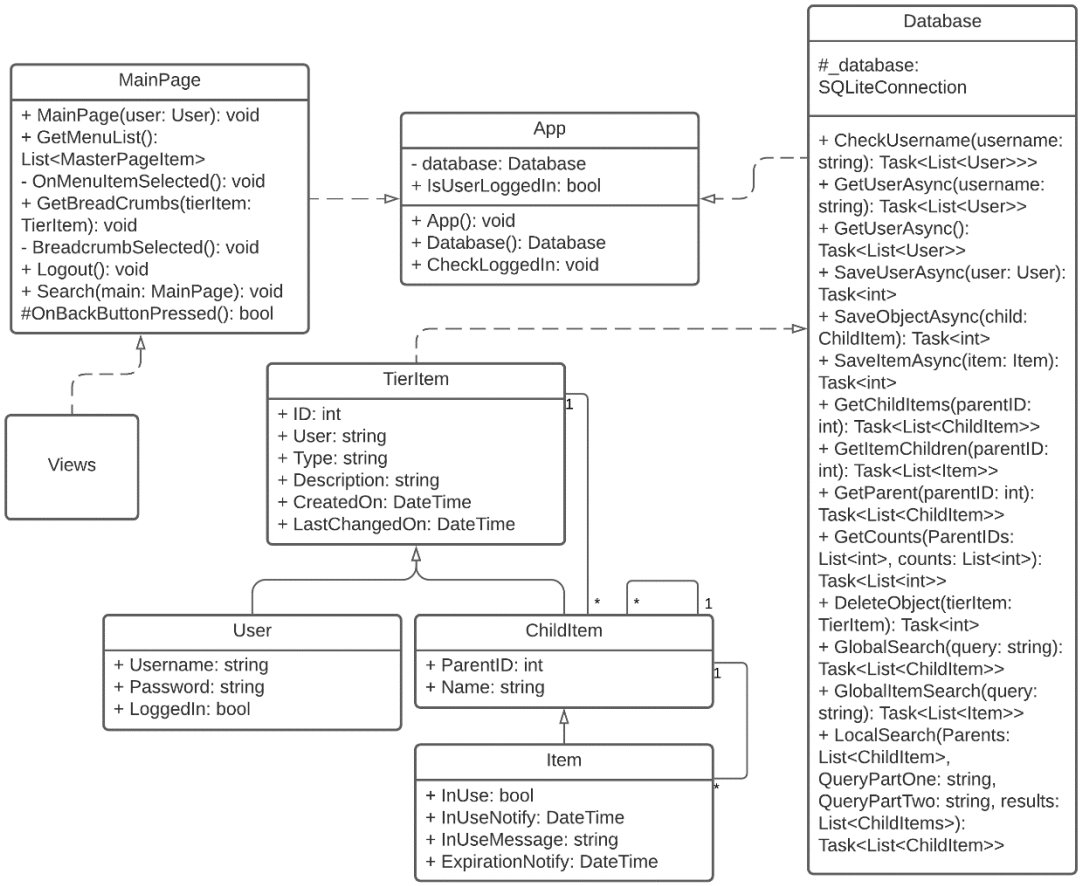

Figure 1 (UML Class Diagram)

UI Design

Following are some images representing the application GUI. The wireframe was designed before the start of the project, but modified slightly to better represent the final product design. Not all of the application views are included; however, these images were strategically chosen to represent every major element used within the application.

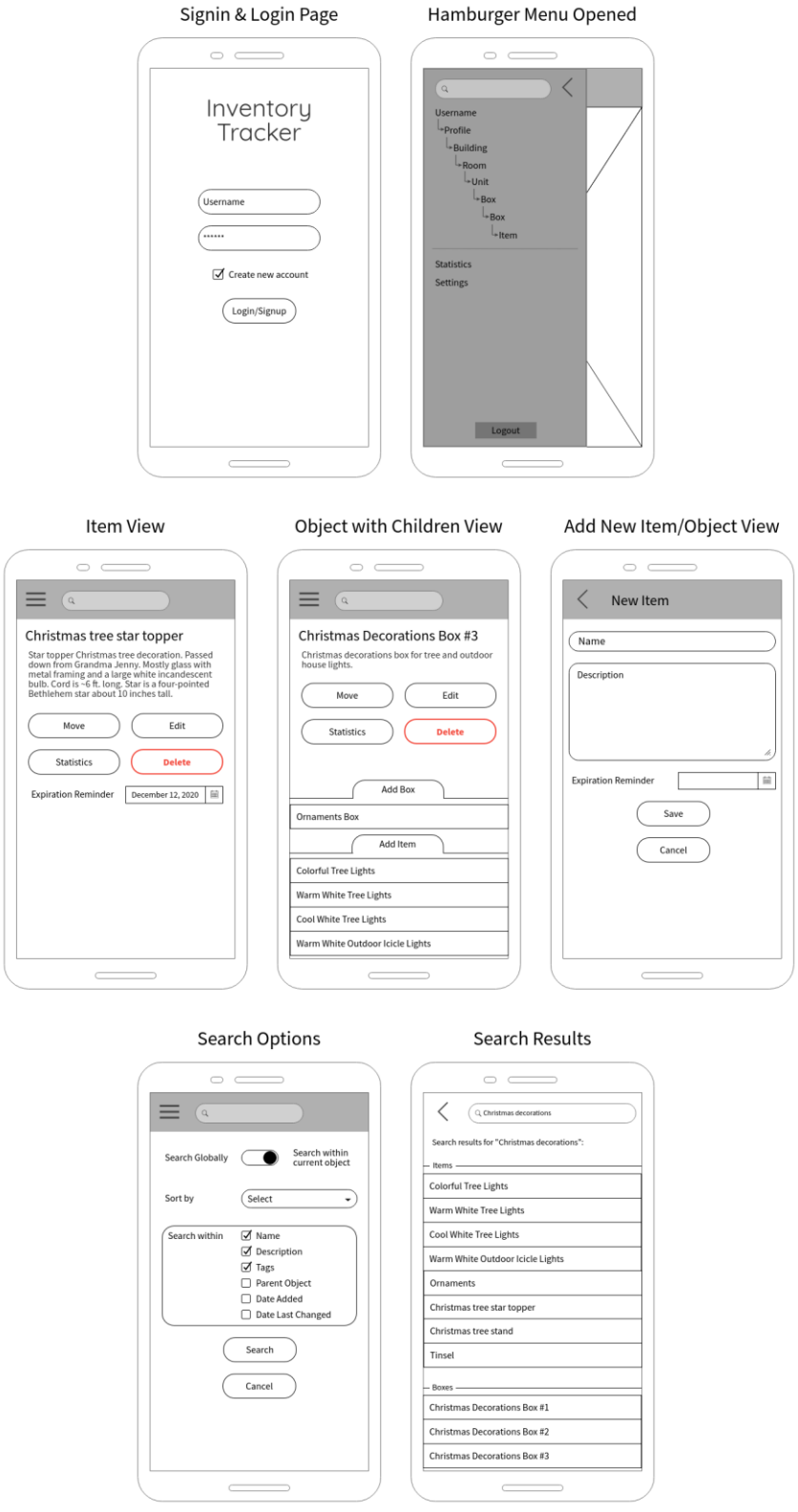

Figure 2 (Wireframes)

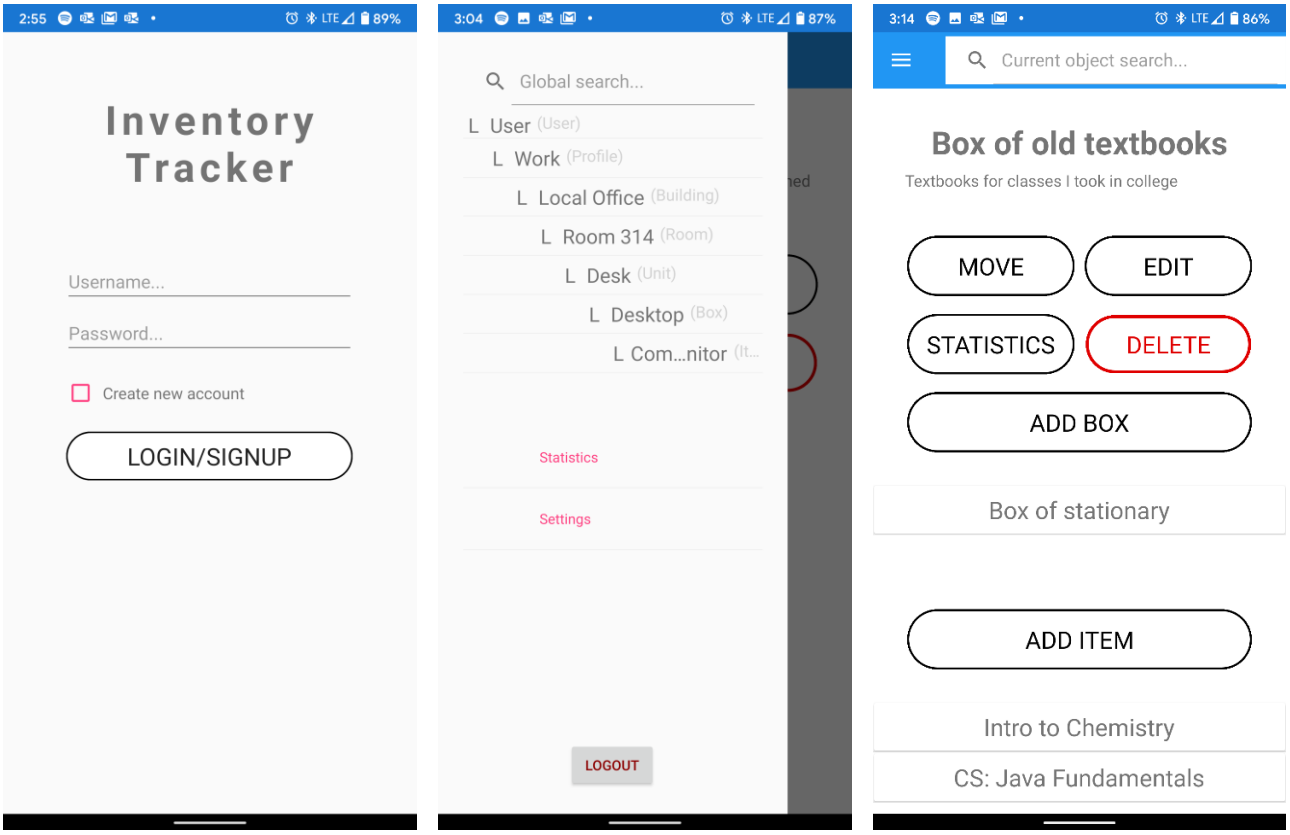

Figure 3 (from left: Login, Hamburger Menu, Item with Children)

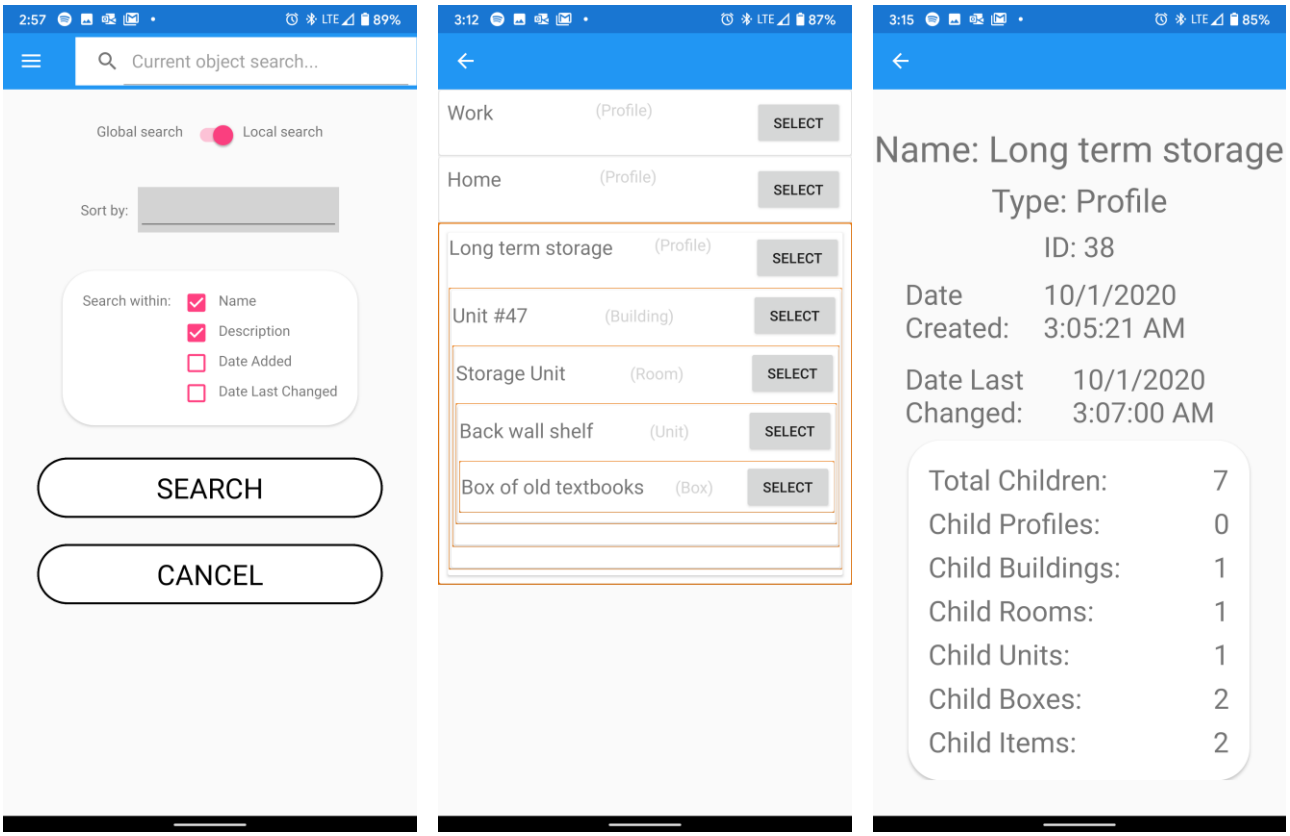

Figure 4 (from left: Local Search, Move Object, Object Statistics)

Unit Testing

Introduction

The primary unit of this application is a View, or “window” of the GUI. A smaller sub-unit that also lends itself to unit testing is individual Controls, or UI elements, such as buttons, entry forms, labels, etc.

Methods

A few different methods were used to test the units of this application during development. One primary method was using the built-in debugging features in Visual Studio, such as Breakpoints and Console Output commands. Another primary method was designing and programming application alerts, reports, and prompts that would provide feedback on whether a function behaved as expected, and what the expectation is when not.

- Breakpoints: Visual Studio provides the ability to pause the application execution at any time. Doing so often halts the ability to progress through the application, but is immensely useful as a troubleshooting tool. When the application reaches a Breakpoint, the IDE allows the tester to inspect the current memory values of the nearby variables, and troubleshoot any bugs or problems that may be occurring.
- Console Output: Writing to Console is one of the most basic, yet useful features of an IDE. It provides the ability to record the values of variables during runtime without interrupting it. The values can be viewed in real-time or after the test has been completed.
- Alerts and prompts: Often a user may become confused about what the application requires for a specific function. Other times an error may have occurred without the user’s knowledge of the event. These are cases when receiving feedback in a visual form are most useful. It is important to know whether a form has submitted properly or not, or when a back-end function has completed a task, but without alerts and prompts to inform the user of these events, they would go by unnoticed. Most often these manifest as a pop-up window with a message and confirmation or acknowledge buttons, or as message labels within the page.
- Reports: Reports are an important feature of the application. They range from results to a text search, to statistics about a specific object in the database, to the breadcrumb trail used for navigation. They are often the primary subject of a page and are often intractable.

Testing in Practice

Following are a few examples of the kind of tests that were completed, as well as their expected results and remedies if failed.

Navigation

Navigating from one page to the next has lots of potential for errors to occur –navigation may fail and leave the user on a blank or incorrect page, or variables may be incorrectly passed between pages and cause the page to incorrectly display the GUI or give the user false information. One remedy can be to double check the values displayed with those same values displayed on another page, or with the initial input values. To prevent the user from wondering if the navigation failed, outputting an error or results message is often useful, as shown in Figure 5.

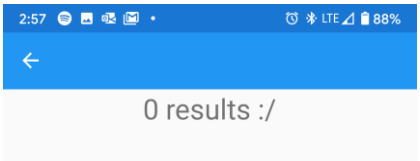

Figure 5 (No results for search output)

Forms

Submitting forms properly is crucial to a productive user experience and to prevent many problems and errors in the future. The expected result of submitting a form is that the input data has been collected, recorded, and used for its purpose. However, input data may be rejected for one reason or another, and the user must know it has happened and how to remediate the problem. The primary way this is done is by outputting alerts and prompts upon form submission and button presses.

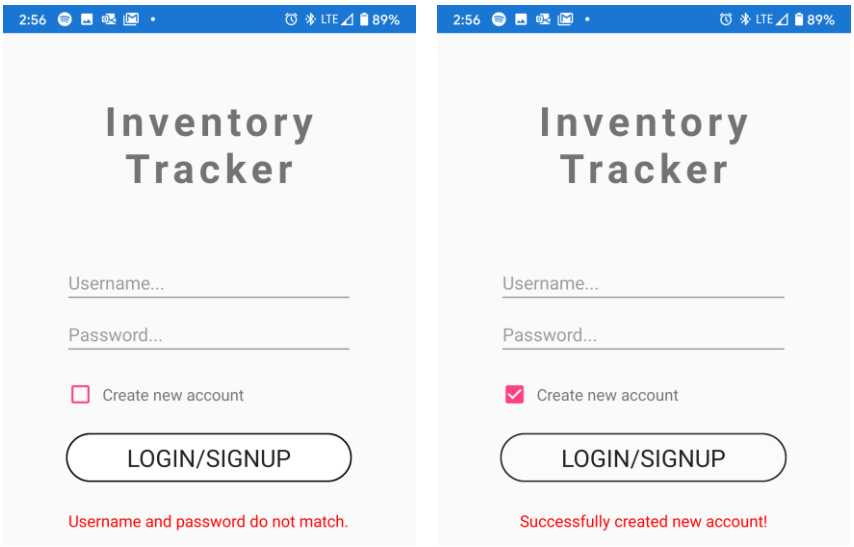

Figure 6 (Failed vs. Successful results)

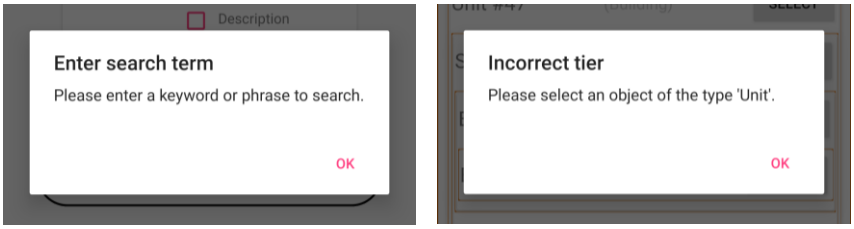

Figure 7 (Guiding feedback)

Creating and Deleting Objects

Probably one of the most critical functions of this application is creating and deleting objects. Depending on which object is erroneously deleted or incorrectly created, the user may lose hours of work. As such, informing the user of successful and failed creation, as well as what data will be deleted is highly critical to a successful application. In this application, this is done by alerting the user when an object is successfully created and how many child objects will be deleted in addition to the current object.

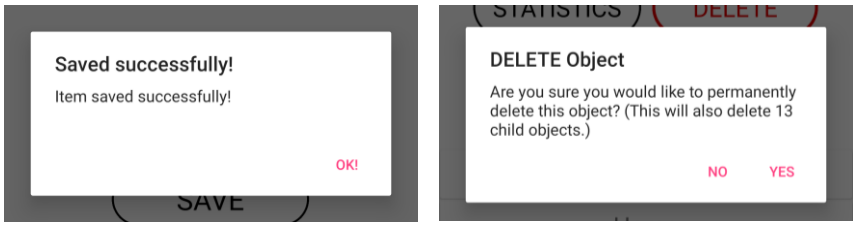

Figure 8 (Successfully saved item and DELETE confirmation)