

**DECLARATION**

You must sign below. By signing this form, you agree to the following:

* We declare that all the work submitted for this assignment is our own original work except for material that is explicitly referenced and for which we have permission, or which is freely available (and also referenced)

The assignment shall be conducted in a team of 4 students, each team member must sign as it is a formal agreement that represents that everyone is contributing to the whole assignment.

|  |  |  |
| --- | --- | --- |
| **Team Member Details** | | |
| **Student Number** | **Student Name** | **Signature** |
| N9665072 | Gyeongmin Jee | **https://docs.google.com/drawings/d/sXOFJMUoM0VmUFJOfetqPlg/image?w=75&h=32&rev=8&ac=1** |
| N6903495 | David Nguyen | **https://docs.google.com/drawings/d/sO9T2v_yV0jfGRNQcOvYDrg/image?w=123&h=35&rev=9&ac=1** |
| N9345311 | Quang-Cheng Tsui |  |
| N9686801 | Yusef Acabani | **C:\Users\Lenovo\AppData\Local\Microsoft\Windows\INetCache\Content.Word\44aa9f83-a630-4bc6-a2ea-553591823b74.png** |
| **URL LINK TO DEMONSTRATION VIDEO:** <https://drive.google.com/open?id=0B4q7rcvvB2czZ25MQ1hPbUd3RDQ> | | |

Contents

[1. Introduction to “Where’s My Stuff?” 3](#_Toc497129357)

[2. User Interface 3](#_Toc497129358)

[2.1. UI Patterns and Design Considerations 3](#_Toc497129359)

[2.1.1. Updated UI 3](#_Toc497129360)

[2.1.2. Page Navigation 3](#_Toc497129361)

[2.1.3. Colour Scheme/Theme 4](#_Toc497129362)

[2.1.4. Form Patterns 4](#_Toc497129363)

[2.2. Home Page 4](#_Toc497129364)

[2.3. Add Room 5](#_Toc497129365)

[2.4. Box Management 6](#_Toc497129366)

[2.5. Searching and Adding Items 7](#_Toc497129367)

[2.6. Help Section 7](#_Toc497129368)

[2.7. List of Required Features 8](#_Toc497129369)

[2.7.1. Adding Items into Boxes 8](#_Toc497129370)

[2.7.2. Searchable Database 8](#_Toc497129371)

[2.7.3. Listing Boxes and its Contents 10](#_Toc497129372)

[2.7.4. Editing Items or Move Items Between Boxes 10](#_Toc497129373)

[2.7.5. Creating Box Labels 10](#_Toc497129374)

[2.8. Nice-To-Have Features 12](#_Toc497129375)

[2.8.1. Data Export 12](#_Toc497129376)

[3. Software Architecture 14](#_Toc497129377)

[3.1. Discussion of Software Architecture Adopted 14](#_Toc497129378)

[3.1. Discussion of Applied Software Architectural Patterns 14](#_Toc497129379)

[3.2. Implementation of Functionality of Each Component and Class 15](#_Toc497129380)

[3.2.1. Application Requirements 15](#_Toc497129381)

[4. Testing and Quality Assurance Strategy 16](#_Toc497129382)

[4.1. Discussion of Quality Assurance 16](#_Toc497129383)

[4.2. Discussion of Testing Methodology 16](#_Toc497129384)

[5. Reflection on Learning 16](#_Toc497129385)

# Introduction to “Where’s My Stuff?”

Ever have problems finding items in boxes which you’ve stored for various reasons such as moving houses or putting away for long-term storage? That is the problem in which “Where’s My Stuff?” aims to solve. “Where’s My Stuff?” is an application for smart devices which allows users to record and track which items are stored in which boxes thus, makes searching for items simpler. A lot of consideration was taken into the design of this application and that is what this report will discuss. In particular, the report will present discussions on how to use the application, the user interface design considerations, application functionality, the software architecture (in particular the MVVM architecture), implemented quality assurance strategies, testing methodologies and learning outcomes.

# User Interface

## UI Patterns and Design Considerations

### Updated UI

Similar to conceptual designs (As submitted previously), the overall look of the application was kept relatively familiar. The Home Page acts as the primary navigation menu and utilises the list menu view. Some of the navigation buttons were however changed since conceptual design. The addition of the Add/Remove Room button now allows users to create their own rooms for which boxes will be stored. This feature will be explained further shortly. Additionally, there is a margin applied left and right and is uniform across all screen. This will assist in making the pages look neater and better presented. To see the demonstration video of the application, refer to: <https://drive.google.com/open?id=0B4q7rcvvB2czZ25MQ1hPbUd3RDQ>

### Page Navigation

Page navigation has also been slightly altered. The new page navigation hierarchy is presented in Figure 1.

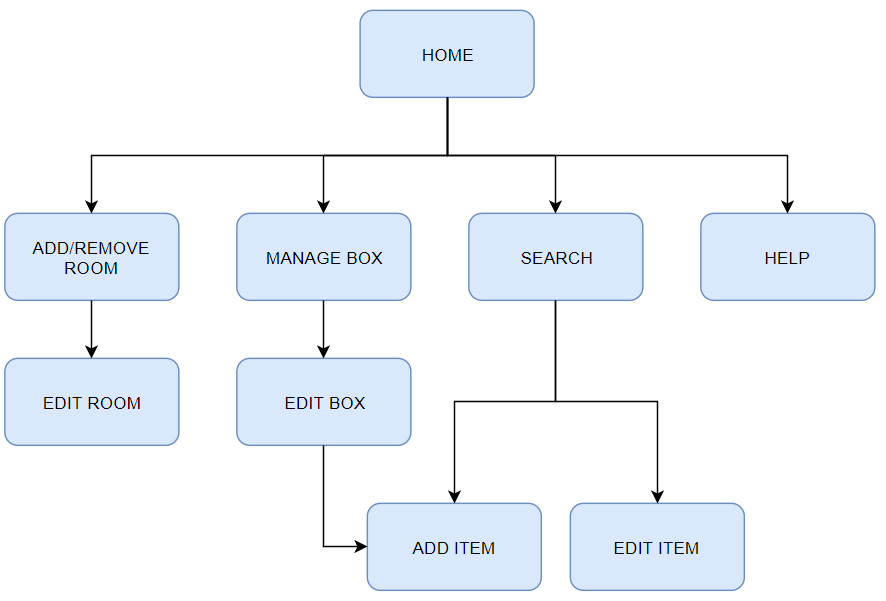


Figure 1. Updated Page Navigation Hierarchy

### Colour Scheme/Theme

The colour theme is very simple yet colourful. The home page will consist of different colours for the buttons and act as an indicator for which page it navigates to (e.g. the “Add Room” button is lime; therefore, the Add Room Page will also have lime accents). The background will be consistent among all pages with a white background and a dark grey navigation bar at the top.

### Form Patterns

The key factors to consider with forms is to make sure the forms are illustrated effectively, not cluttered and simple to comprehend. The primary form patterns used for this application include the search and registration form patterns. Where applicable, the text entry fields and picker items will be applied to pages and its forms. These can be seen in the following several sections which discusses each page and its functionality.

## Home Page

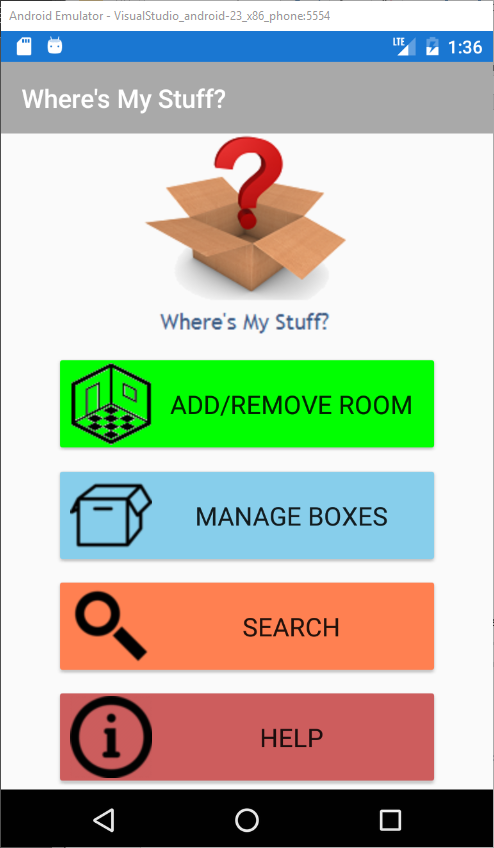


Figure 2. Screenshot of Home Page

The Home Page is the primary navigation for the “Where’s My Stuff?” application. As mentioned previously, it displays a simple list layout with minimal options on the page, to not overwhelm users. Note that the colour of each button also indicates the colour of that particular page (which was done to follow the colour theme mentioned previously) which will be demonstrated in the following sections.

## Add Room

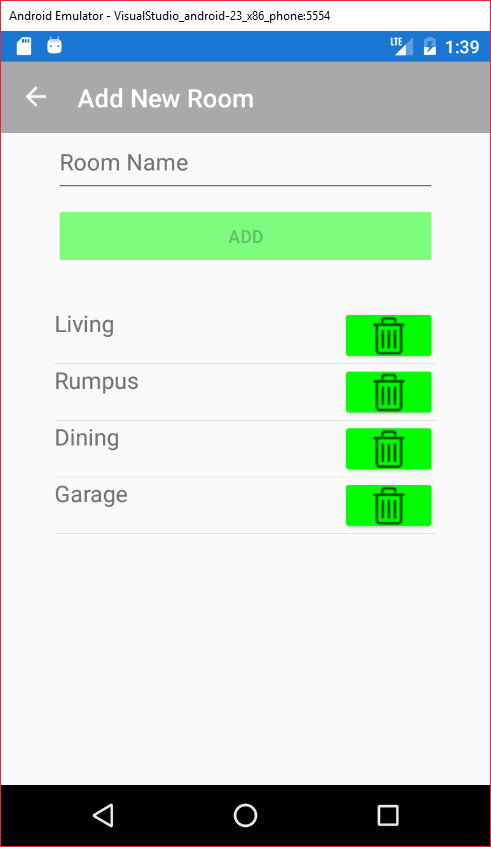


Figure 3. Screenshot of Add Room Page

Adding a room is a new feature added since the conceptual design. This page allows users to generate their own room types and thus allocate boxes to those rooms. Users will also not be able to generate duplicates. Therefore, users can only have one name for each room type. For example, if a room type “Garage” is already existing as shown in Figure 3, users cannot add another room called “Garage”. If users attempt to create duplicates, the application will display an alert notifying the user of this issue. Whilst on this page, users can also choose to delete any room type by simply tapping the bin icon for the respective room. Clicking the bin icon will prompt the user with another alert which will ask users to confirm their actions before deleting the room type.

## Box Management

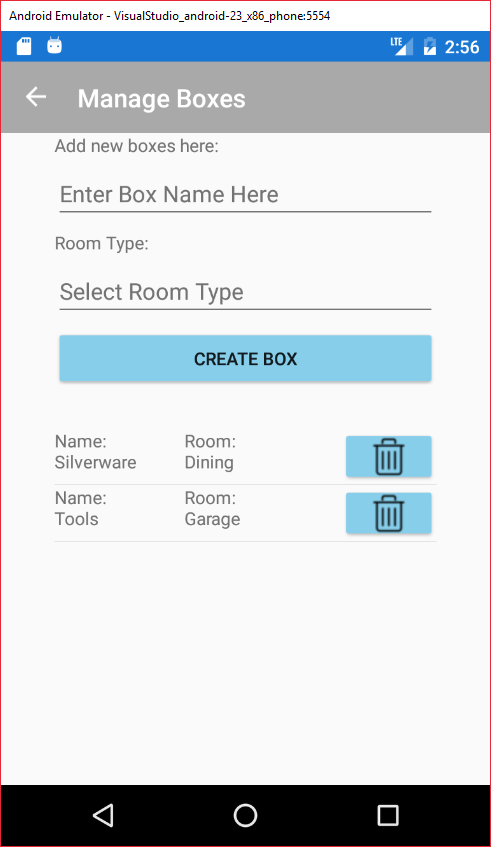
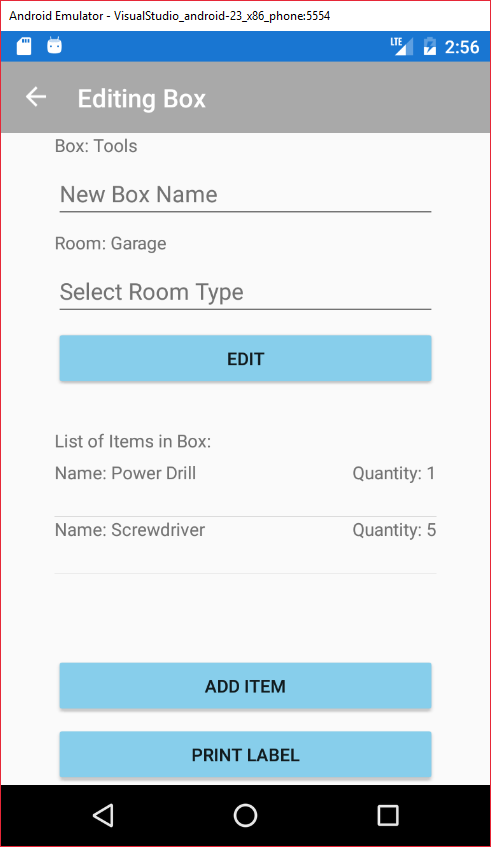
 

Figure 4. Screenshot of (Left to Right) Manage Box Page and Edit Box Page

Managing boxes can be achieved by selecting the “Manage Boxes” button whilst on Home Page. From here, users can see the list of available boxes as well as generate a box by assigning it a name and room (as illustrated in Figure 4). Users can also delete any box by tapping the bin icon which also displays an alert window for users to confirm their actions. Editing a box can be accomplished by tapping on the box users wish to edit. Whilst on the Edit Box page, users can change the box’s name as well as room type then tapping “Edit” to confirm the changes. Whilst on this page, users can also see all items present in the current box as well as the option to add new items or print a label displaying the box’s name, room type and list of contents.

## Searching and Adding Items

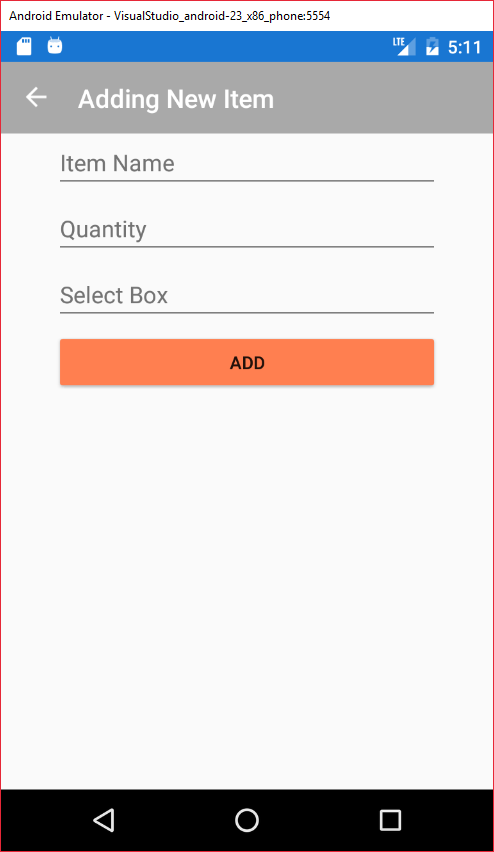
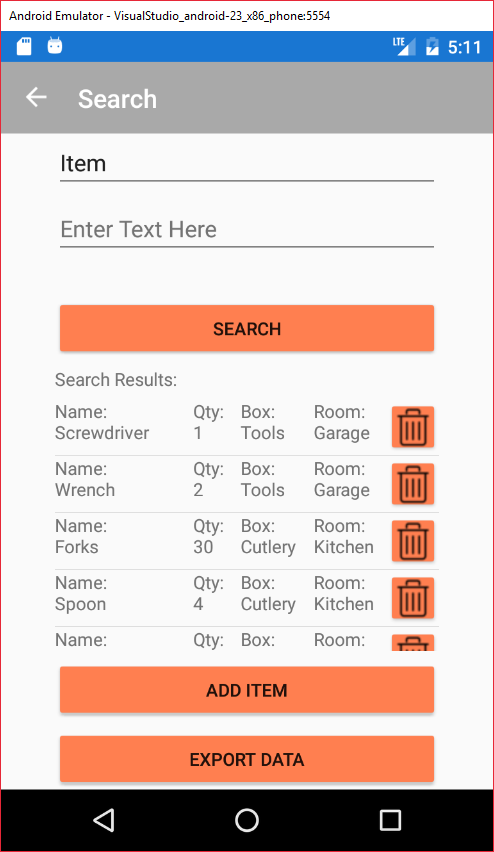


Figure 5. Screenshot of (Left to Right) Search Page and Add Items Page

When first entering the search page, users will be able to see all items currently saved in the database. To search for items, users can use the drop-down menu at the top of the screen to select to search by item name, room type and box name. Users can also edit an item from this page by tapping an item. If users want to add a new item, they can click “Add Item” button at the bottom to navigate to the “Add Items” page.

## Help Section

The Help Page is reachable via tapping the “Help” button whilst on the Home Page. The Help Page serves as a how-to guide for users to get into using the application. It provides step by step instructions on how to achieve whatever tasks the user requires of the application.

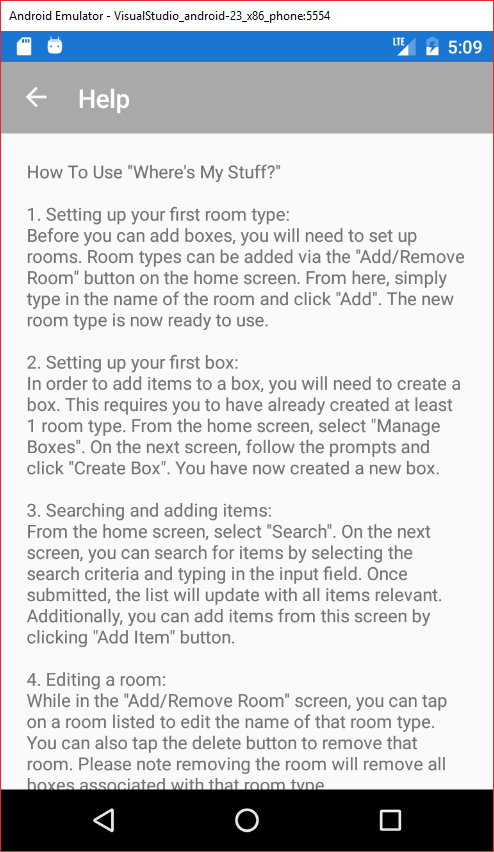


Figure 6. Screenshot of Help Page

## List of Required Features

### Adding Items into Boxes

Adding items into boxes can be done by two methods. First method is to go to the “Search Page” and clicking “Add Item” at the bottom. The second method is to navigate to the “Edit Box” page and select “Add Item” at the bottom. Once selected, the “Add Item” page will appear, and users can follow the prompts to add an item. Note that this feature requires a box to be already created.

### Searchable Database

The user can navigate to the “Search” page to search for an item. There are three categories which users can search by selecting it in the top drop-down menu, illustrated in Figure 7. The categories are Item name, Box name and Room type. Once category is selected, users can enter their search text parameter and then submitting. The list of items will update with the appropriate items relevant to the search. An example of searching via room category is demonstrated in .

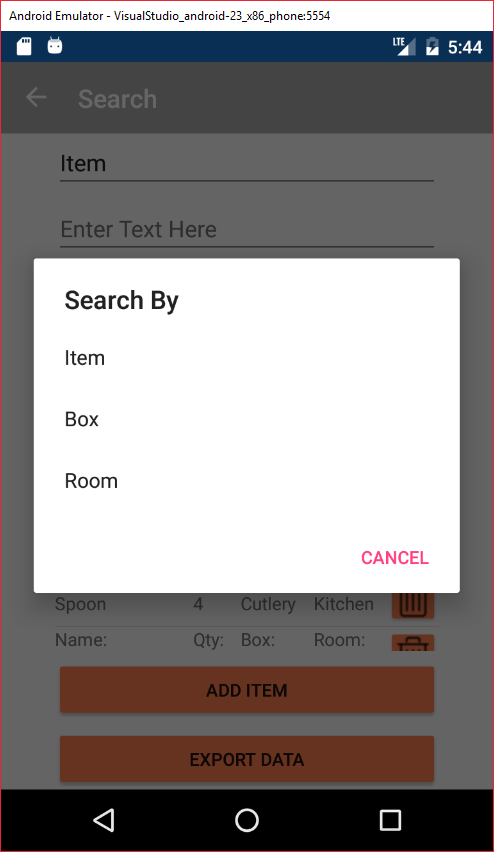


Figure 7. Search Options

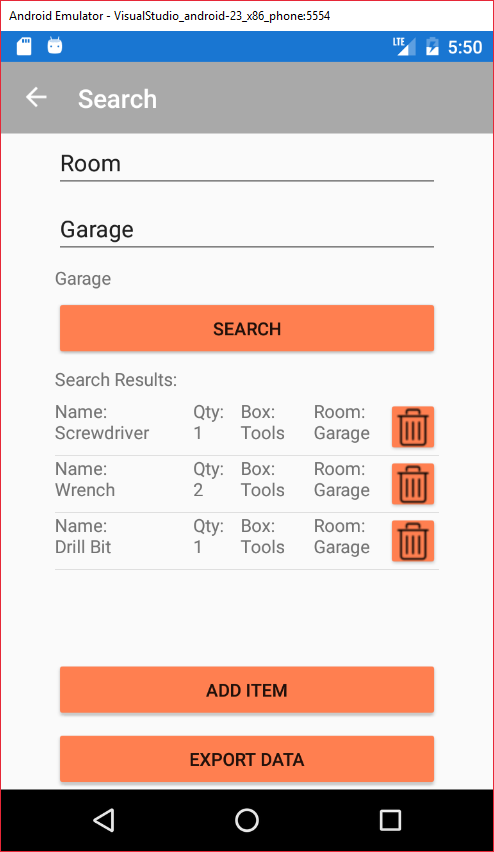


Figure 8. Example of Searching by Room Function

### Listing Boxes and its Contents

The ability to view boxes and its contents can be achieved in the “Edit Box” page as illustrated back in Figure 4. From this page its also possible to edit the name of the box and even move the box to another room.

### Editing Items or Move Items Between Boxes

Modifying an item can be accomplished while on the “Edit Item” page. This page is reachable by tapping a listed item in the “search” page. From here, users can change the name of the item, change the quantity of the item and move the item into another box via the drop-down menu. To commit the changes, users will need to click “Edit”. A small message will appear to notify the user that the item has been edited successfully.

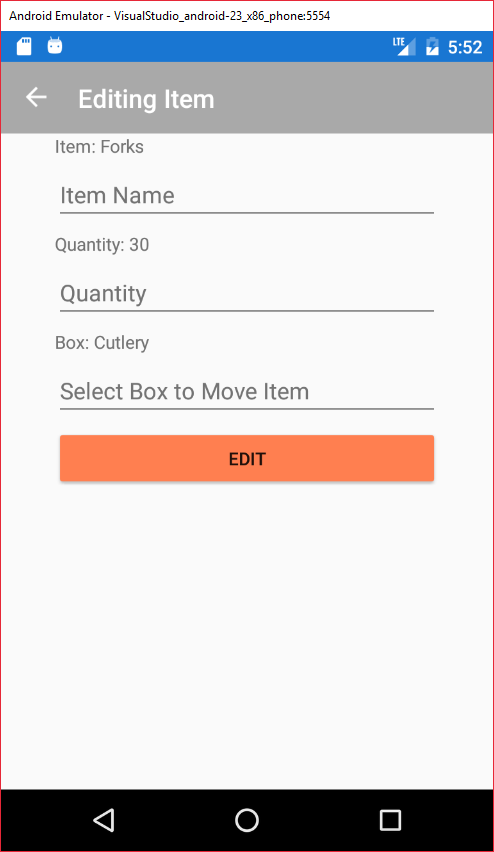


Figure 9. Editing an Item

### Creating Box Labels

The application was made to allow users to generate a simple text file which they can print to apply to a box. This is accomplished by clicking “Print Label” button on the “Edit Box” page. When clicked, a message will appear asking the user if they would like to print the label that is shown. Printing however cannot be completed unless the user’s device can send the file to a printer. In any case, the label text file is generated and saved to the device’s file directory if they which to retrieve it to print elsewhere. This process is illustrated in the screenshots below.

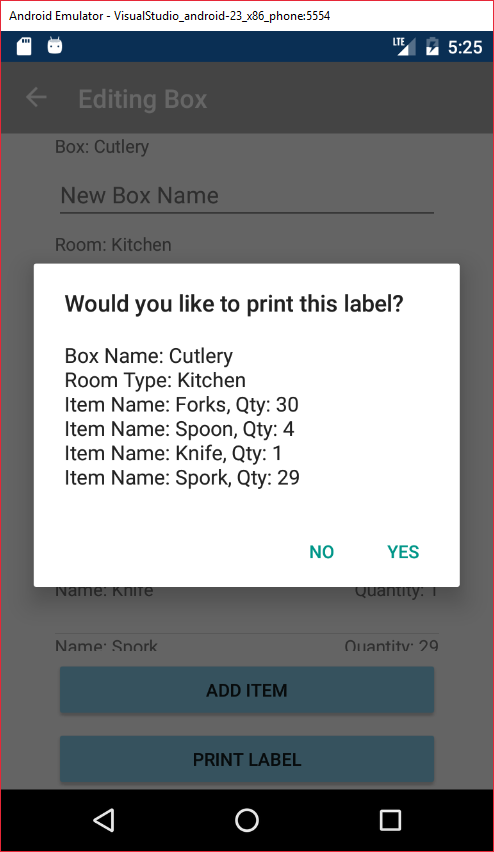


Figure 10. Screenshot of When User Prints Label

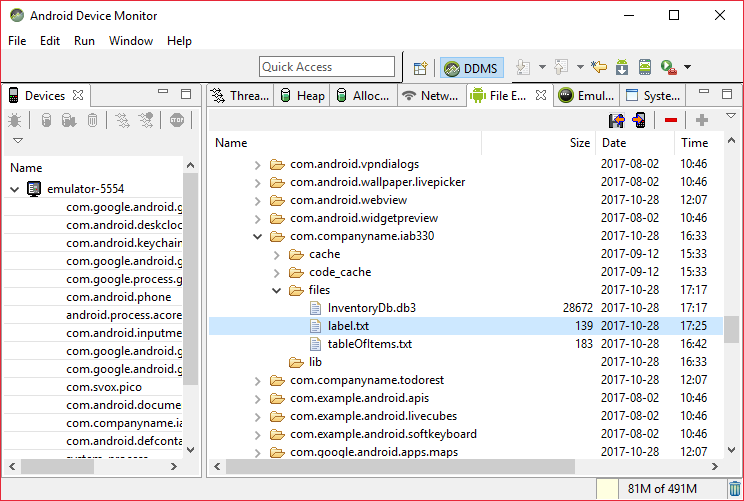


Figure 11. Screenshot of Device's File Explorer showing Label Generated

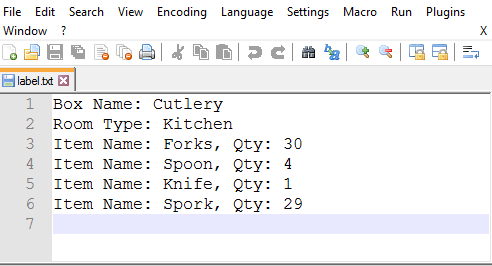


Figure 12. Screenshot of Label Text File

## Nice-To-Have Features

### Data Export

The application can export the list of items into a text file. From this text file, users can import it into an excel spreadsheet to generate a table that can be sorted and organized. The following screenshots illustrates this process.

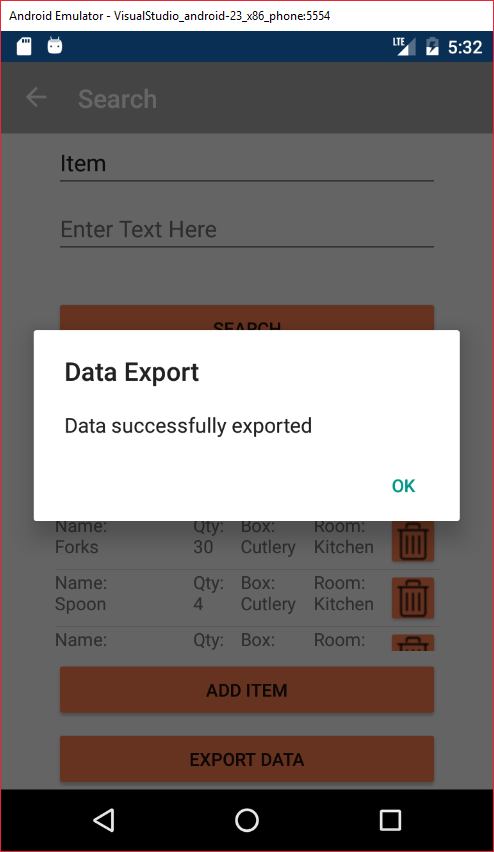


Figure 13. Screenshot Showing Data Export Completed Successfully

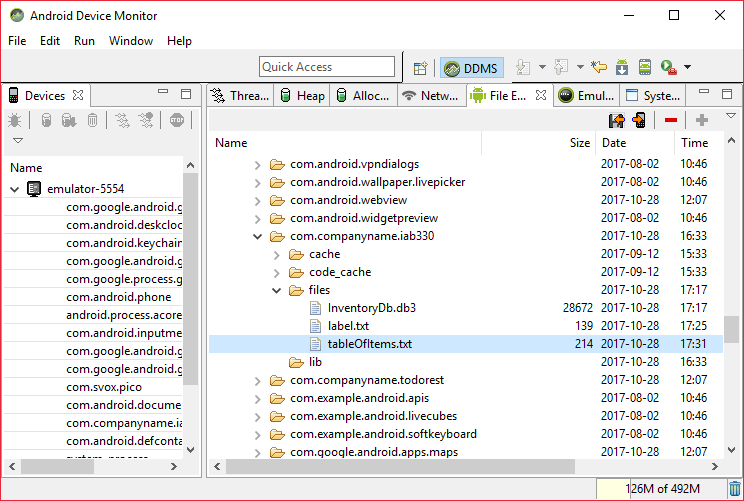


Figure 14. Screenshot Showing List is Generated

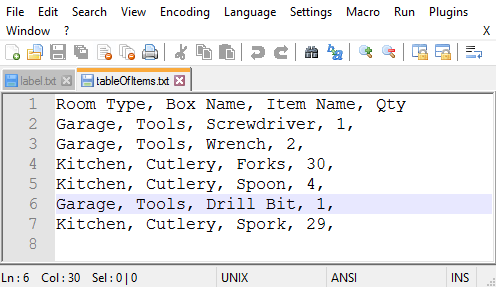


Figure 15. Screenshot of List Generated

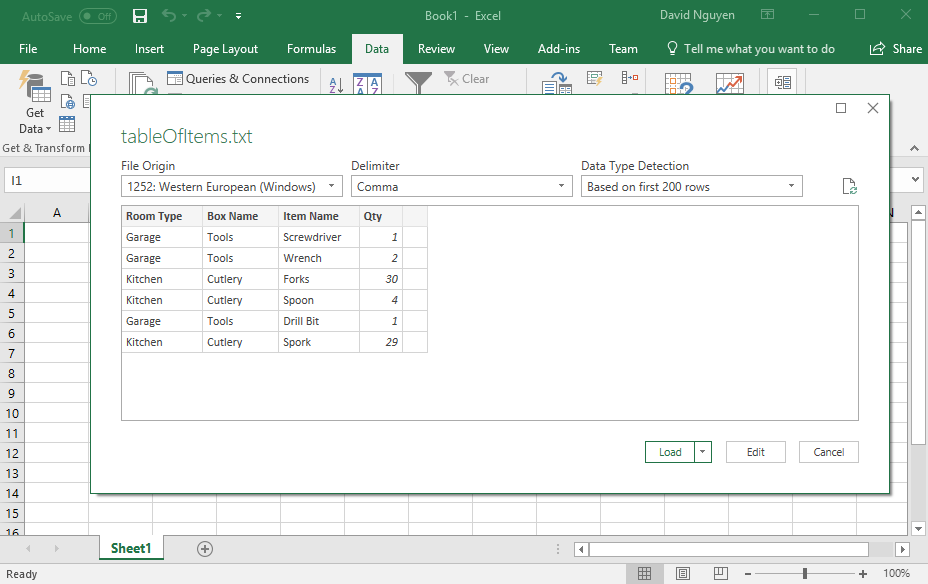


Figure 16. Importing File into Excel

# Software Architecture

## Discussion of Software Architecture Adopted

Our application consists of three logical layers: Presentation (UI) layer, Service (Application) layer, and Data layer. This means that the entire application separates its functionality into logical (not physical) levels. The presentation layer is constructed using Xaml and Code behind. It is what is displayed to the users. Its main purpose is to display data to users and receive their inputs. The service layer is responsible for handling business logic, which means that it deals with real-world business rules. For example, it determines how data can be created and changed, how data should be validated, and how input should be parsed (E.g. how input should be used to query the database). Data layer provides an abstraction to access the database with ease. It provides an API that will be used by the service layer to store and query data. In our application, Sqlite-net ORM library was used to provide access to the local database.

## Discussion of Applied Software Architectural Patterns

The software architectural pattern applied in our application is a Model-View-ViewModel (MVVM) pattern. MVVM provides a separation of concern in each layer, especially between the UI layer and service layer. This is mainly because the UI (view) has no knowledge of model, and it interacts with the ViewModel via databinding and commands instead of calling methods directly from the view. This is different from MVC because in MVC, the controllers are tightly coupled to the views. The controllers must know what to render to the view and how to render the view. For such reason, changing the view will often require changes in the controller. In MVVM, the controller is replaced by ViewModel. ViewModel provides observable variables to which the view must bind. Input from the user invokes an action (or command). The ViewModel will change the observable variables and the view will reflect on the changes.

## Implementation of Functionality of Each Component and Class

The model classes (Item.cs, Box.cs, and Room.cs) are the objects that represent instances of each model. It specifies the relationship and columns (properties) of each row in the Sqlite database table. The classes in the Services directory implements access to the database via SQLiteConnection object which implements platform specific connection to the database (which is made possible through a dependency injection service). ItemDataAccess.cs, BoxDataAccess.cs, and RoomDataAccess.cs classes implement methods for CRUD operations on Item, Box, and Room tables respectively. DataAccessLocator.cs uses singleton design pattern that provide access to only a single instance of each DataAccess class.

The ViewModel classes in ViewModels directory are the actual implementation of the ViewModels in MVVM. These classes extend BaseViewModel which extends InotifyPropertyChanged interface to override OnPropertyChanged method. These classes hold variables (setters and getters) which calls OnPropertyChanged() to notify the views that binds to these variables of changes. ViewModelLocator.cs class performs the same role as DataAccessLocator.cs.

The xaml files and cs files in the Views directory are implementations of the view layer. The cs files are the code-behind that initializes the components as well as assigning binding context for the components of the page. The code-behind is responsible for handling view related events such as clicks of buttons and navigation. The xaml file defines the looks structure of the UI.

## Application Requirements

### Packages Required

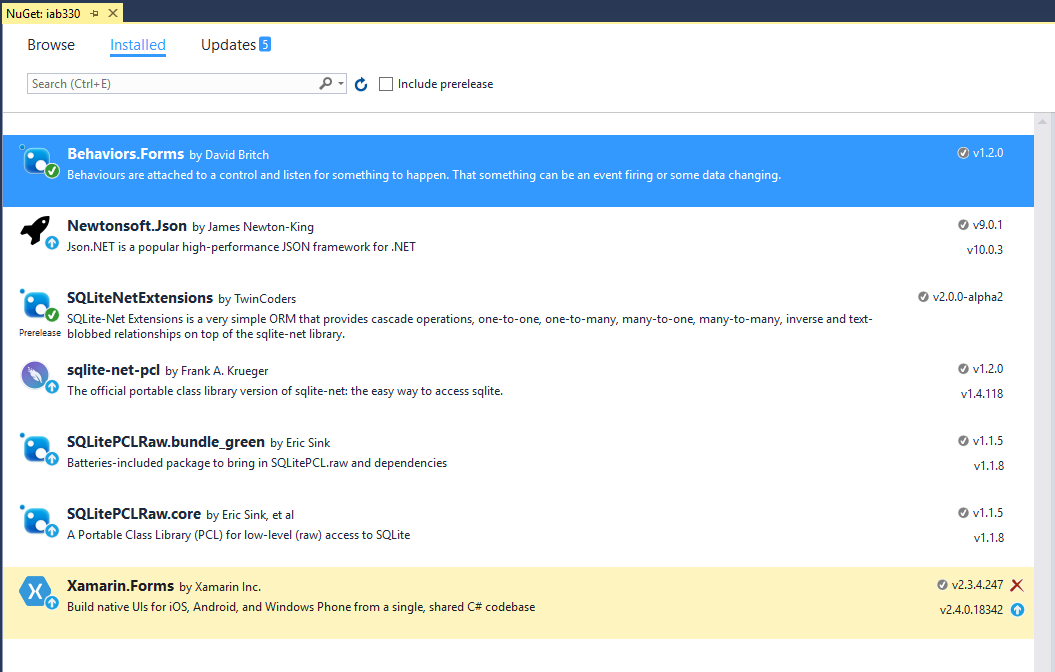


Figure 17. Screenshot of Installed Packages

The packages included in our project were:

* + Sqlite.net Extensions: An ORM library for visual studio projects to provide access to sqlite database. It is an extension on top of an existing library (sqlite-net-pcl) to provide relationships (E.g. one-to-one, many-to-many, and more) and cascading operations.
  + Sql-net-pcl: the PCL version of SQlite. It also depends on SQLitePCLRaw packages.
  + SQLitePCLRaw.bundle\_green: Package to bring in SQLitePCL.raw and dependencies.
  + SQLitePCLRaw.core: A PCL for low-level access to SQLite
  + Behaviors.Forms: Used for when an event is triggered by data changing.
  + Newtonsoft.Json: JSON framework for .NET.

### Android Emulator Information

The application was developed and test using the default Visual Studio Android Emulator. The emulator details are shown in Figure 18. This emulator will require Intel HAXM and the Google API x86 atom image.

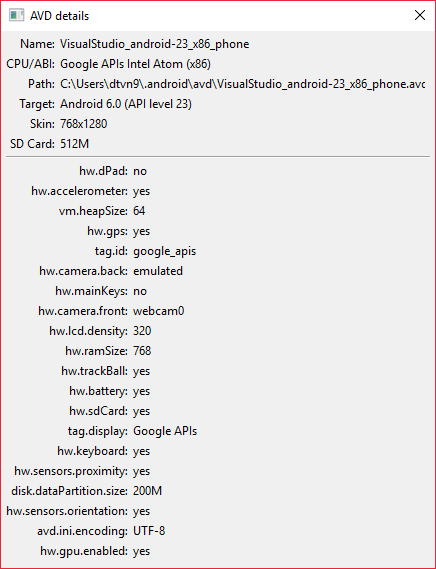


Figure 18. Screenshot of AVD Details

# Testing and Quality Assurance Strategy

## Discussion of Quality Assurance

While at first look a mobile application may look like it’s working properly, we never know what could be going on that could cause an error to appear or the application to crash. That’s why, to guarantee the best performance from our application, we had to run number of tests. They work by testing specific parts of the of the application to confirm the effectiveness of that certain part, or to find errors that might go unnoticed.

This is the most effective way of making sure that the application runs flawlessly, and that every exception has been taken care of, so that the user can have the best experience.

## Discussion of Testing Methodology

We tested our app using to different methods: unit testing and UI testing.

* Unit testing: We run the tests by isolating specific parts of the code, in this case methods. We now test those methods separately to confirm that each one of them is working properly.

But we don’t just test for things that work, we also test for things that might not work. We run test to see if the exceptions work properly. For example we run tests to make sure that when submitting forms, like adding items, if the text fields necessary to create those

Items are empty then an error should display.

* UI Testing:

Note: due to problems with Yusef’s computer (hard drive failure), most of the test were lost.

# Reflection on Learning

We have learned about the system architecture and design patterns that we were able to implement in our application. We have learned that even if each member is assign a task to work in different layers in the application, it is necessary to have at least some knowledge of each other’s tasks in order to review other’s code for quality assurance. We learned that learning the new platform and language as a team is more effective than learning alone.

We as a team faced many challenges, which mostly were technical issues. Our first major problem was related to source control and managing gitignore files to exclude files that would cause merge conflicts in every update. At first, we were unable to identify the exact files responsible for causing the conflicts. We as a team came up with ideas to solve the problem but all seemed very inefficient. It was believed finding an existing gitignore file from website such as stackoverflow will solve our problem most effectively. A suitable file was found and the problem was solved. We did not face any major conflicts in terms of teamwork and task distribution. Tasks that needed to be completed were discussed in the group chat space and members were willing to take part when necessary. Whenever an error or undesirable behaviour was found in the app or code, it was immediately notified to the group chat space such that anyone with the knowledge about the error can fix it immediately. Frequent communication between members ensured that our application stays up to date without piling up errors.