*EasyKeeper HotelInventory* Guide

This guide is intended not so much for app users as it is for those managing the app code and its distribution in general, briefly listing usage, then tips, bugs, and advice.

# App and Web Interface

The app itself is actually restricted to only moving around amounts of a given item for a given storage between 3 categories. **On the app, these categories are *“Dirty”, “Rooms”, “Storage”.* On the interface, these 3 categories are *“In process”, “In room”, “In stock”,* respectively*.*** The web interface first and foremost catalogues who can use the app. The "Users" webpage gives you the ability to assign custom roles like Administrator, Manager, or House Keeper, and controls who is allowed to login to the app and database. It is arranged such that a user belongs to one out of the hotels provided on the Hotels webpage described in the list below.

In our app and database, we develop a hierarchy of the following groupings:

* ***Hotels*** -- because we wanted to provide for the instance where a user of this system might own more than one hotel. As a user is tied to a hotel, when a user logs into the app, it will pull up a list of the Storages for that hotel ("Floor One", "Floor Two", etc., including an item for "process", "dirty", or "laundry").
* ***Storages*** -- we consider these to be analogous to anything like a general storage location on a given floor, though it is possible to add multiple storage locations on the same floor. When a user touches a row with the name of a location, the app will pull from that storage's associated "stock" of various types of items.
* ***Item Types, Items, and Stocks*** -- we have provided here an inner hierarchy to attempt to help organize the selection of item types that a hotel may provide in general, as well as the items specific to a given floor. Not all types would be used for all floors, for example there wouldn't be pool equipment in every floor's storage closet(s). As can be noted from the Items section, the items are themselves sorted into the item types.

Item amounts can be modified via web. As noted in Tips and Troubleshooting, this provides for 2 situations we felt not all staff should be able to perform in-app:

1. Initially receiving or getting additional new items in that are introduced into the inventories.
2. Stolen or damaged items that must be removed from the system entirely.

# Tips and Troubleshooting

* We suggest that on the web interface storage locations be added to the database such that the “id” increases, so does the number of the floor. For example, first we have Floor One with an id of 2, then Floor Two with an id of 3. To justify this need, consider that if we had attempted to sort the storages by name instead of by id, “Floor Four” would precede “Floor Three”, etc.
* A potential trouble spot you may run into while attempting to manage the web interface for the inventory system arises when you require the addition of newly acquired stock or the subtraction of items that have been either lost or damaged, and so intend to alter the “Total amount” of the item. When you attempt to save a change to this value, the database will appear to simply ignore the alteration. This happens because within the server the total count is held constant as a kind of validation check for the other 3 counts. In order to work around this, we ask that you please first Delete and then re-add that particular item with the proper amounts as well as the revised Total amount.
* One special, rare case we considered but (like the 2 cases above for new or stolen items) did not allow the app to handle was when items need to be moved from the storages/”in stock” category to the dirty/”in process” section, in the scenario that an item falls off a housekeeper’s cart and becomes soiled. An alternative idea for the UI that could provide for this would be something similar to what you see below.



# Bug and Advice

* Currently HotelInvItemsTableViewController.m makes 2 requests to the URLRequestManager, one for fetching item types and one for fetching items. However, these requests are sent asynchronously, so at times they will come back in different orders and as a result will not always cause the items to be displayed to the user on screen. This is the biggest bug that needs fixing, though going back and reselecting the storage will resend the requests, so it is not an app-crashing error, though still a problem that needs resolution.
* We were unable to integrate the item types section headers more cleanly into our Matt\_sFilesStorageLocation class’s NSMutableArray\* itemsInLocation collection, hence within HotelInvItemsTableViewController.m there is the effective but not very clean existence of the combinedItemAndTypeArr NSMutableArray\* object set up in requestFinished: and used solely in cellForRowAtIndexPath: to get the correct rows sorted beneath their proper sections—and still retain the correct text titles. One suggestion for future versions of the app would be to integrate the itemTypes more fully into the Matt\_sFilesStorageLocation class to allow for better sorting abilities.
* Another potential code revision would be to modify the URLRequestManager class to accept and store the user token received from the authentication server request, as opposed to the current public method of simply passing the token between the view controllers. It would be more secure to set a property on the requestManager object in HotelInvLoginViewController.m and let it be retained and referenced in all future requests by slightly modifying the request string received as an argument in sendRequest:. Because all server requests are prefixed with this user token, future developers may simply have the view controllers send in the request string *except* for the user token substring “token=%@”, and instead handle prefixing the request string argument with the user token inside of sendRequest: manually.
* If the client does not quite like the aforementioned need to order their database entries by hand, we suggest adding another id-like column to the storages table that could be set by the administrator. The storages would then need to be modified in Matt\_sFilesStorageLocation to first receive (in a new property) and second sort its array of locations by these new values.
* Finally, we believe that there is another potential, if small, area of confusion in the scenario where the database entries are modified while a user has the app open to one of the tables, say Storages, on their iOS device. We suggest the addition of an asynchronous background update thread to continue to fetch the newest tables for the current screen in the future to address this.
* That being all of the major downsides of the coding, we present a way to improve the app by the addition of something new: a search feature at the top of every table. We initially sought to add an indexed list to the right side of the HotelInvItemsTableViewController, however, it did not quite turn out as nice as we had hoped. <http://www.raywenderlich.com/16873> discusses adding this feature. If in future versions the indexed list is desired, the code for it has been commented inside of HotelInvItemsTableViewController to get the list to show up along the view’s right side. Look for the commented out code for tableView:sectionForSectionIndexTitle:atIndex:, sectionIndextitlesForTableView:, the @property NSMutableArray\* itemTypesFirstLetters, and the 2 lines to initialize this property inside of populateItemTypes:. This is the sum of all code used to implement the indexed list and get it to show with the first letters of all item types as listed in the sections headers of the table. As you may notice, however, this gets problematic because some item types share the first letters, as with Terry and Towel. They show up represented by the same first letter in the right-side listing.
* Perhaps you might also provide a UILabel on the item addition/removal screen that describes it, or if feasible a way for the web interface to supply an image that is used as the background of the screen. Though this is likely not as needed, because the hotel staff would be acquainted with the item names. This is why we have not implemented it in our first version.