ECE Inventory System: Evolution 4

Tyler Bletsch, Duke University

January 19, 2017

1 Introduction and Use Cases

The ECE department would like an inventory system to track items ranging from $0.05 transistors to $5k oscilloscopes. This system will serve the following use cases:

- ECE lab managers will record the quantity, location, model numbers, vendor info, and documentation for varied lab assets.
- Duke students and faculty will be able to browse this available inventory information.
- Duke students and faculty will be able to request items from the inventory; managers will be able to approve or deny these requests, and the system will log approved disbursements as they reduce the inventory.
- All users will be able to log in using their Duke NetID (including priviledged users, whose accounts will be marked as either "managers" or "administrators").
- Users will be able to request multiple items at a time using the popular "shopping cart" metaphor.
- Managers will have permission to perform the bulk of the day-to-day operation of the system, while administrators will be able to override most of the rules of the system to correct erroneous situations. A comprehensive transaction log will track all operations.
- Developers will be able to interact with the system programmatically using APIs with the same access as their user account provides.
- Administrators will be able to create/delete new data fields to be tracked for all items.
- In addition to simply disbursing items, managers will also be able to officially *loan* resources out; the system will keep precise track of such loans.
- Key operations involving users (such as loan approvals) will be communicated via emails.
- System administrators will be able to restore the state of the system from a robust backup system using a clearly documented procedure.
- Managers will be able to bulk-import new items from a simple text format.

- Managers will have the option of setting the *minimum stock* for individual items, seeing a list of resrouces whose stock on hand falls below this threshold, and opting to receive emails whenever an item's stock falls below a threshold.
- Users will have the option of returning loaned items by *backfilling* them (i.e., providing evidence of an order of equivalent items from a vendor).
- Managers will be able to track outstanding backfill operations to ensure that received items are tracked and credited properly.
- Managers will be able to track certain items (such as costly testbench equipment) as distinct, named entities called *assets.*

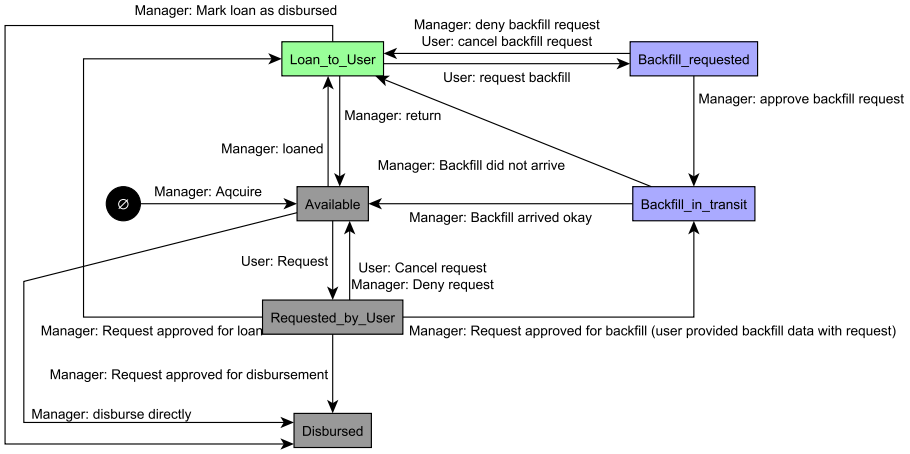
2 Defintions

- **Item**: A type of entity tracked in the inventory system, such as "1k resistor". Sometimes, to make clear that this term refers to a model of thing rather than an instance, the phrase *distinct item* may be used.
- **Instance**: An specific instance of an item, such as one specific 1k resistor. To clarify, if the system has five 1k resistors, that's one item with five instances.
- **Tag**: A small piece of text used to flexibly describe an item, e.g. "resistor", "microcontroller", "oscilloscope", etc. An item can have zero or more tags.
- **Transaction**: Any operation which alters the state of the inventory, including acquisitions, disbursements, and administrative corrections.
- **Disbursement**: The act of outright giving instances of an item to a user, thus removing them from the system.
- **Available pool**: The quantity of an item on hand. Doesn't include items on loan. Also doesn't include items in the backfill-request or backfill-transit stages.
- **Outstanding request**: A user request that has not yet been acted upon by an administrator.
- **Approved request**: A user request that has been approved. This approval may be for disbursement or loan. ...or backfill.
- **Denied request**: A user request that has been denied.
- **Manager**: A user with with the "manager permission" who is therefore able to respond to requests and perform inventory management tasks.
- **Administrator**: A user with the "administrator permission" who therefore has all manager rights plus the ability to override system rules to directly correct data and to grant manager and administrator rights to other users.
- **Unpriviledged user**: A user with neither of the above two permissions.

- **Intrinsic field**: A data field associated with an item which is built into the system, i.e., name, model, and description.
- **Custom field**: A data field associated with an item which is added to the system by the administrator.
- **Private field**: A custom field which the administrator has indicated should be hidden from unpriviledged users.
- **Per-asset field**: A field which is tracked for each instance of an asset. For example, a serial number field would be per-asset, while a model number field would not.
- **Loan**: The act of assigning instances of an item to a user while still tracking them. Note that loans do *not* have a specified "due date".
- **Item stack**: A quantity of instances of a single item which all share a common state, such as "available", "requested by user X", "requested by user Y", "loaned to user X", etc.
- **Subscribed managers**: The set of manager email addresses to which administrative emails are sent.
- **Backfill**: When a user in posession of a loaned item orders a replacement of the item in lieu of returning the original.
- **Backfill-request state**: The state an item instance may be in when a user has requested to backfill it, but the manager has not yet approved the transaction.
- **Backfill-transit state**: The state an item instance may be in when a user has requested to backfill it and the manager has approved, but the replacement part has not yet arrived.
- **Asset**: An item whose instances are tracked uniquely rather than as simple interchangable parts. This is likely to be a higher-value device such as a piece of test equipment.

To better understand the lifecycle of items, the figure below shows a flowchart for item instances.

Figure 1: Item instance flowchart. States introduced in evolution 3 are shown in green , evolution 4 in blue .



3 Requirements

A note on requirements: No set of requirements is perfect, and that is certainly true here. I'm sure that contradictions, under-specified behavior, and unintended consequences will be revealed. Your overriding goal should be to produce a quality system; if you believe that goal would be better served if a requirement were altered or interpreted a certain way, ask about it, and get the conclusion in writing. The result may be a variance in a requirement for a specific team, or even modification of this requirements document for all teams. In short, if unsure, ask.

1. Server
   - 1.1. Your software must have a server that supports an arbitrary number of users.
   - 1.2. During the install/setup process, a special user named "admin" is configured.
   - 1.3. The system shall allow the use of the Duke NetID system to allow all users to login using their Duke credentials in addition to supporting locally created users. The special local "admin" account remains, and has administrator permission.
   - 1.4. Any stored passwords must be kept in a secure manner (i.e., salted + hashed)
   - 1.5. All communication between the clients and server must be encrypted.
   - 1.6. The server must maintain state in a persistent fashion.
   - 1.7. For all views which show a potentially unbounded number of records, the response time of the interface shall not depend on the quantity of records.
   - 1.8. The system shall track permission level for each user: normal, manager, and administrator. The special local "admin" user comes has implicit administrator permission. Permissions are strictly nested: managers can do anything normal users can, and administrators can do anything that managers can.
   - 1.9. Users with administrator permission can create "local" (non-NetID) user accounts.
   - 1.10. Users with administrator permission can grant manager or administrator permission to any existing user (either NetID-based users or local users).
2. Basic inventory tracking functionality
   - 2.1. Managers shall be able to create new distinct items. New item records include a unique name (string, required), starting quantity (integer, required), a model number (string), description (multi-line string), currently defined custom fields, and zero or more *tags*. Administrators may opt to mark the new item as an asset (see req 13).
   - 2.2. Managers shall be able to modify the name, model number, description, currently defined custom fields, and tags of items.
   - 2.3. Managers shall be able to create and delete tags.
   - 2.4. Administrators shall be able to delete items. All tracking for instances of the item ceases when the item is deleted. As this operation is highly destructive, the user should be asked for confirmation before proceeding.
   - 2.5. Managers shall be able to log the acquisition of additional instances of existing items.
   - 2.6. Managers shall be able to log the loss/destruction of instances of existing items.

2.7. Administrators shall be able to modify items in any way (including directly editing quantities), as well as delete items. These operations are to correct errors and override system rules; confirmation should be requested before such operations are carried out.

2.8. A user shall be able to view the list of items.

2.8.1. Users shall be able to filter the view with a set of required and/or excluded tags. The system will make apparent the selection of tags available.

2.8.2. Users shall be able to search the view by item name or model.

2.9. Users shall be able to access a detailed view of a given item.

2.9.1. All attributes of the item shall be shown(except private fields, if the user is unpriviledged (see req 5.2)).

2.9.2. Users shall be able to see the item broken down into relevant item stacks: the quantity of available instances, any outstanding requests made by the user, and any loans held by the user. Also include any stacks in the backfill-requested or backfill-transit stage (see req 12).

2.9.3. Managers shall additionally be able to see *all* of the above, not just those relevant to the active user.

2.9.4. Managers shall be able to browse a log of all transactions involving the item.

2.10. Users shall be able to add various quantities of multiple items to a "cart", then request all of them at once while providing a required text response with the reason for the request. Requests do NOT remove resources from the available pool; requests can therefore be "oversubscribed" (i.e., the total quantity requested exceeds available inventory).

2.10.1. Users shall be able to indicate if a request is for disbursement or for loan (see req 7).

2.10.2. For loan requests, users may also propose a backfill (see req 12). To do so, the user must upload a PDF proof-of-purchase, which the system shall keep associated with the request.

2.11. A user shall be able to view a listing of requests.

2.11.1. Unpriviledged users should only see their own requests, while managers should see requests of all users. The requestor's reason and the type of requrest (disbursement vs. loan) should be displayed.

2.11.2. Requests should be distinguished visually depending on if they're outstanding, approved, or denied; outstanding requests should be especially easy to find. Loan vs. disbursement requests should also be differentiated , as should backfill status .

2.11.3. Users shall be able to cancel outstanding requests, thus removing them outright.

2.11.4. Users shall be able to see if requests were approved or denied, and the associated comment (if any).

2.11.5. Managers should be able to respond to the request in any of the following ways:

2.11.5.1. approve it for disbursement (thus removing the item instances from inventory)

2.11.5.2. deny it

2.11.5.3. approve it for loan (thus removing it from the available pool but still tracking it)

2.11.5.4. approve it for loan-with-backfill (see req 12.1)

2.11.6. The manager response to a request may include a comment.

2.11.7. The system shall prevent approval of any request where the available pool is insufficient to cover the items requested. In this case, the reason for the failure shall be made clear.

2.11.8. Managers shall be able to disburseor loan items from the available pool to a user directly without a user-initiated request. Managers shall be able to provide a comment when doing so similar to the response comment specified in 2.11.6.

2.11.9. Requests may be approved for disbursement ~~or~~ , loan , or backfill regardless of which the requestor asked for; the requestor's selection is merely advisory.

3. Documentation

3.1. **Developer guide**: A document shall be provided which orients a new developer to how your system is constructed at a high level, what technologies are in use, how to configure a development/build environment, and how the database schema (or equivalent) is laid out.

3.2. **Deployment guide**: A document shall be provided which describes how to install your software entirely from scratch. It should start by describing the platform prerequisites (e.g. Linux distro, required packages, etc.), then mechanically describe every step to deploying your system to production readiness.

3.2.1. In addition to covering how to install the system with "stock" default data, the procedure to install the system from scratch using backed up data should also be included (i.e., disaster recovery).

3.3. **API guide**: A document shall be provided which orients a developer to how to access your system's APIs from scratch. The actual details of each available API may be documented elsewhere, though. This document should be available from within the interface, but should be available as a standalone file. See requirement 4.

3.4. **Backup admin guide**: A document shall be provided which explains the backup solution so that a system administrator unfamiliar with your software could configure it from scratch, restore the database to any given backup, and test a backup for validity. See requirement 8.

4. API

4.1. Every interaction described in these requirements must also be achievable via a well-documented network-based API.

4.2. Interaction with the API shall not require the user's login/password, but rather an alphanumeric token which the user can request to be randomly generated and associated with their account.

4.3. A simple API debugger tool must be available which allows manual testing of every API call supported in a transparent way.

5. Administrator-defined fields

5.1. Administrators shall be able to add or remove new custom fields to be tracked for all items. Each field is defined by its name, its type (one of the set {short-form text (one

line), long-form text (multiple lines), integer, or floating-point number}), whether or not the field is per-asset, and whether or not it is private. Note: the administrator may not remove intrinsic fields such as name or description.

5.2. Data stored in custom fields marked 'private' should only be shown to managers.

5.3. The *location* field, formerly considered intrinsic, shall now just be a default custom field (short-form text, non-private), which an administrator could remove as per usual. This field should be per-asset by default.

5.4. A default custom field called *restock_info* (long-form text, private) shall be included. This field should not be per-asset by default.

5.5. Administrators shall be able to mark a field as being *per-asset*. This has no effect on normal items, but for assets, it means the field is tracked per instance rather than for the overall item.

6. Global transaction logging

6.1. The system shall record a log of all transactions undertaken in the system (i.e., any change to the inventory state). Entries shall include the initiating user, the item(s) involved, the nature of the event, the time and date, and the affected user (such as the recipient of an approved disbursement). This includes corrective actions undertaken by administrators (per 2.7).

6.2. Managers shall be able to view this log globally as well as per-item (see req 2.9.4).

6.3. The global log view should allow searching by user, item, and/or timespan.

6.4. Users consulting the log shall be able to navigate directly from a reference to an item to the item's detailed view (see req 2.9).

6.5. Users consulting the log shall be able to navigate directly from a reference to an outstanding user request to an interface allowing them to approve/deny the request as specified in 2.11.5.

6.6. Users should not be able to tamper with the log in any way, regardless of permission.

7. Loans: The system shall provide support for loans of items to users. The aspects of this affecting the request/approve facility are covered as sub-bullets under requirement 2, while additional considerations are detailed below.

7.1. Managers shall be able to view a list of current loans along with all relevant details.

7.1.1. Managers shall be able to filter the view with a set of required and/or excluded tags. The system will make apparent the selection of tags available.

7.1.2. Managers shall be able to search the view by item name as well as user name.

7.2. Managers shall be able to mark part or all of a loan is having been returned, thus moving the affected items back to the available pool.

7.3. Managers shall be able to convert part or all of a loan to a disbursement, thus giving the affected items to the user and removing them from the system.

7.4. Users shall be able to see a list of loans issued to that user. This view shall include manager-provided comments. In the case of loans awarded as a result of a user request, all details of the original request (including requestor comment) shall also be available.

8. Backups: You must deploy a backup solution for your system's database.
   - 8.1. Backups shall be automatic and taken daily.
   - 8.2. Backups shall be kept with a staggered retention (7 daily backups, 4 weekly backups, 12 monthly backups).
   - 8.3. Backups must be stored on a separate system.
   - 8.4. The backup system must require separate credentials to access.
   - 8.5. The backup system should report on progress and alert on failure; this could be via email or another directed communication mechanism.
9. Bulk import facility
   - 9.1. Administrators shall be able to import new distinct items into the database by means of a text-based import based on a simple format such as CSV, JSON, XML, or another plaintext-based format.
   - 9.2. The import interface shall include documentation as to the import format.
   - 9.3. The import action shall only occur if the *entire* input is free of name conflicts or otherwise problematic issues; if such issues arise, the precise nature of the error should be presented to the administrator in enough detail that it can be corrected.
10. Email support
    - 10.1. Managers shall have the option of "subscribing" to the system, meaning that their email address is included in the "subscribed managers" list.
    - 10.2. The system shall send an email for every user request to all subscribed managers as well as the requesting user.
    - 10.3. The system shall send an email to the affected user for any transaction dealing with his or her request, disbursement, or loan. ...or backfill.
    - 10.4. Managers shall be able to configure "loan reminder emails".
      - 10.4.1. Managers shall be able to configure on which dates emails will be sent.
      - 10.4.2. Managers shall be able to configure a body of text to be prepended in the body of this email.
      - 10.4.3. Loan reminder emails shall, when a configured date occurs, be sent to all users with recorded loans. The email shall contain the configured preamble as well as a list of loaned items.
    - 10.5. Emails shall be sent in such a way that subscribed manager email addresses are not revealed to unpriviledged users.
    - 10.6. To allow users to filter emails from the system easily, managers shall be able to configure a "subject tag": a small piece of text to be prepended to the subject line of all emails sent by the system.
    - 10.7. When an item's available pool goes below the configured threshold, an email shall be sent to subscribed managers (see req 11).
11. Inventory stock management

11.1. Managers shall be able to set a low-inventory threshold per item. Managers may also disable the threshold per item.

11.2. Managers shall be able to set thresholds in a "bulk" manner efficiently, for example with a checkbox-style selection with select-all and select-none shortcuts being applied to a search/filter view.

11.3. Managers shall be able to filter the list of items to show only those whose available stock is less than the configured threshold.

12. Backfill support

12.1. Users shall, for any loan they hold, be able to request to end all or part of the loan by *backfilling* the item(s) rather than returning the original. To do so, the user must upload a PDF proof-of-purchase, which the system shall keep associated with the loan in question. This places affected items into the backfill-request state.

12.2. Managers shall be able to view a list of backfill requests (i.e., items in the backfill-request state). The interface should differentiate backfills requested to forgive a loan (req 12.1) from backfills requested as part of a new item request (req 2.10.2).

12.3. Managers shall be able to respond to a backfill request either by approving it (moving such items to the backfill-transit state) or denying it (for backfills requested to forgive a loan (req 12.1), this returns affected items to the loan state, whereas for backfills requested as part of a new item request (req 2.10.2), this simply denies the request).

12.4. Managers shall be able to view a list of approved backfill operations in progress (i.e., items in the backfill-transit state).

12.5. Managers shall be able to mark an approved backfill as having been satisfied (meaning the items in question have been received properly), thus adding the affected items to the available pool, and forgiving the associated loan, if any.

12.6. Managers shall be able to mark an approved backfill as having failed (meaning the items in question never arrived or were somehow objectionable). The result of this is that for backfills requested to forgive a loan (req 12.1), this returns affected items to the prior loan state, whereas for backfills requested as part of a new item request (req 2.10.2), this converts the request into a new loan. In this latter case, the system should make clear the cause and nature of the created loan.

13. Asset tracking

13.1. Items may be configured as *assets*, meaning that instances will be identified with an *asset tag*, which is a unique automatically generated number.

13.2. In addition to creating new items as assets (per req 2.1), managers may convert existing items to become assets. In this case, all instances will be assigned an asset tag.

13.3. In any view in which an asset is being displayed, it should be identified with its unique asset tag.

13.4. Administrators shall be able to change asset tags, but even in this case, asset tags shall always remain globally unique.

13.5. Managers shall be able to edit per-asset custom fields for individual assets.

13.6. While requests may be made for any available instance of an asset, managers approving such requests must identify the specific asset instance that is being used to fulfill the request.