## SDEV220\_teamC\_2023F\_Final\_Project\_Results

Updated 12/13/2023

Team members:

Mark Atkins [matkins@Ivytech.edu](mailto:matkins@Ivytech.edu) Indianapolis, IN

Joshua Hoover [jhoover82@ivytech.edu](mailto:jhoover82@ivytech.edu) Fishers, IN

Derek Kolb: [dkolb1@ivytech.edu](mailto:dkolb1@ivytech.edu) Medellin, Colombia

### 1. Project description:

(Fictitious) Client: “ a local company”

Comlux: Derek’s former employer <https://comlux.com/completion/about-us>

Scope of the system:

* + Prototype, simple demo of concept
  + Graphical user-friendly interface
  + Uses three or more classes
  + Utilize collections: lists, tuples, and dictionaries
  + Runs with no syntax or runtime errors and produces the correct results
    - Results based on written test plan SDEV220\_teamC\_2023F\_Project\_TestPlan <https://docs.google.com/spreadsheets/d/1YpsZssFirAUMzI7dSiGogC-HA5jHsOecShsfRHtTVkM/edit#gid=0>
  + Documentation: [the proposal with class diagram](https://docs.google.com/document/d/1M1M1AyNSaRyPPNAt5ip6-93kuCTanl0ZG5mIWvXCxyU/edit?pli=1#heading=h.ikbvxuy49h12), and this report Gdoc of results with sample output, including images of GUI

Purpose of the system:

* + Inventory control, tracking (like Trackit @ Comlux @ indy airport)
    - Inputs: parts (items) and user badge #s (via barcode reader emulated, not really implemented), keeping track of aircraft parts received, removed from, returned to storeroom for aircraft repairs. Parts may include new and used (removed from the aircraft)
    - Classes:
      1. Item
         * Attributes: Item unique ID (S/N?), Manufacturer, Model, Name, Description, Supplier, image\_file\_path, History (list of dates, conditions, and locations)
         * Methods: Add\_history, Get\_history
      2. History (not implemented, would be possible in future)
         * List of (date, location, status) tuples (new, used good, used bad. Location = in inventory or out in field. Perhaps the location value would be the technician’s ID or the airplane ID.)
      3. Tracker (Stockroom)
         * Attributes: list of Items (in SQLite3 database file, can’t save the actual Item object, just the Item attributes)
         * Methods: Search\_item, Get\_item\_data, Get\_inventory, Add\_item, Move\_item, other TBD
      4. GUI (using Tkinter package)
         * Dialog box for inventory
         * “Add item” button
         * Entry fields for item’s attributes,
         * “Checkout item” button
         * “Search” button, with entry fields for item’s attributes
         * A list displaying entire inventory, scrollable, sortable
         * A list displaying a catalog of possible items (not exhaustive), scrollable, sortable
         * Image of the item, if an item is selected in the Catalog or Inventory list
      5. ToolTip
         * Used in GUI to add mouse-hover tooltips to clarify the entry blanks and button behaviors
    - UML class diagram:



### **2. GitHub repo:**

[*https://github.com/joshuagh00/SDEV220\_Final\_Project\_Group\_C*](https://github.com/joshuagh00/SDEV220_Final_Project_Group_C)

We used the collaborator features of GitHub

### **3. Trello** (KanBan project management) board: SDEV220\_project\_teamC\_2023F

<https://trello.com/b/jh9HHOrf/sdev220projectteamc2023f>

Team member roles/responsibilities:

Mark Atkins: project manager + 50% code + 80% of testing + final report

Joshua Hoover:, did no work though was assigned code to do.

Derek Kolb: GUI, 50% of code + 20% of testing

### **4. Code modules (files)**

Main.py

Starts GUI and tracker, includes catalog and inventory lists

Gui.py

Tracker.py: database via SQLite3, functions to update data

ToolTips.py: short class to add tooltips to entry blanks and buttons

Settings.py (includes catalog of part descriptions and name/path for SQL database file)

Images folder: a bunch of .png images, one for each item in catalog

**5. Program usage (quick-start guide)**

Running main.py in VScode or from the command line via “python main.py” produces the application GUI shown in the shots below. The GUI is intended to be intuitive and obvious, no user manual needed. Tooltips on the entry blanks and buttons explain the functionality when the mouse hovers over the object.

The Catalog and Inventory lists are tkinter TreeView objects, scrollable. They can be sorted by column by mouse clicking on the column header.

The Catalog automatically populates based on the descriptions dictionary in the settings.py file. To add or remove catalog items, edit the dictionary source and restart the application.

The Inventory list displays the contents of the SQL database named in the settings.py file. It includes a column to show the exact time of the last change for a given item model. The Model (leftmost) field is the unique key for the database and for the Search button.

Selecting (via mouse and/or arrow keys) an item in either list populates the entry blanks with the corresponding data.

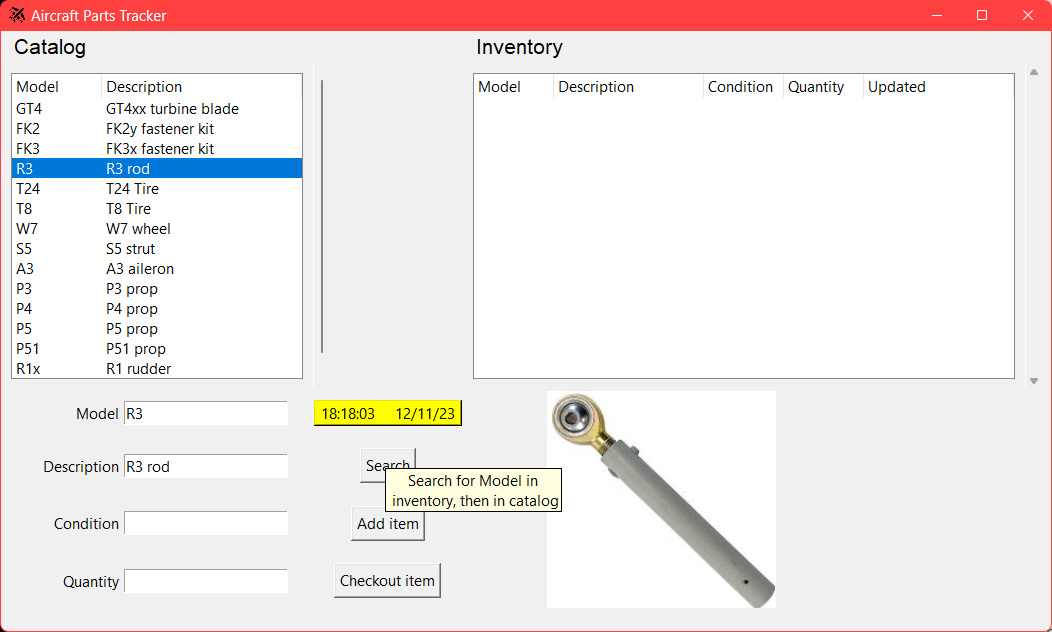
The Search button looks in the Inventory, then in the Catalog if no match in inventory, for the given model. If found, the row highlights in the corresponding list and populates the other entry blank fields.

Otherwise, MessageBox alerts the user that the Search found no match, as shown in the screenshots below.

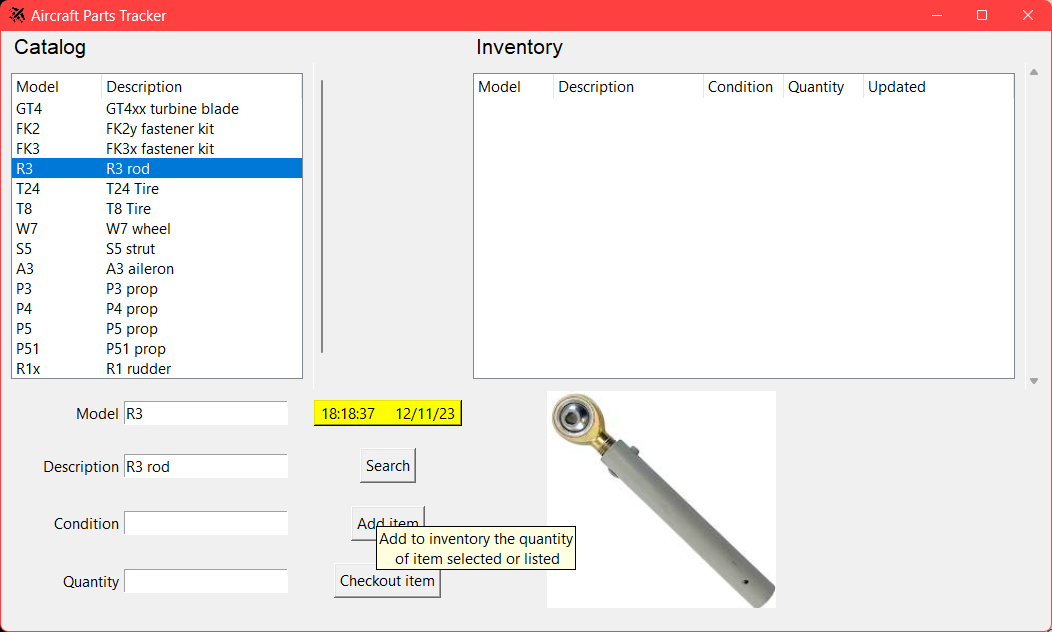
The Add item and Checkout Item buttons adjust the inventory for the Model listed in the Entry box, using the Description entry box if populated and condition and quantity Entry boxes. Both buttons give appropriate warnings to the user for a missing quantity value or invalid quantity value (nonnumeric, or exceeds inventory for a Checkout).

**6. Screen shots:**

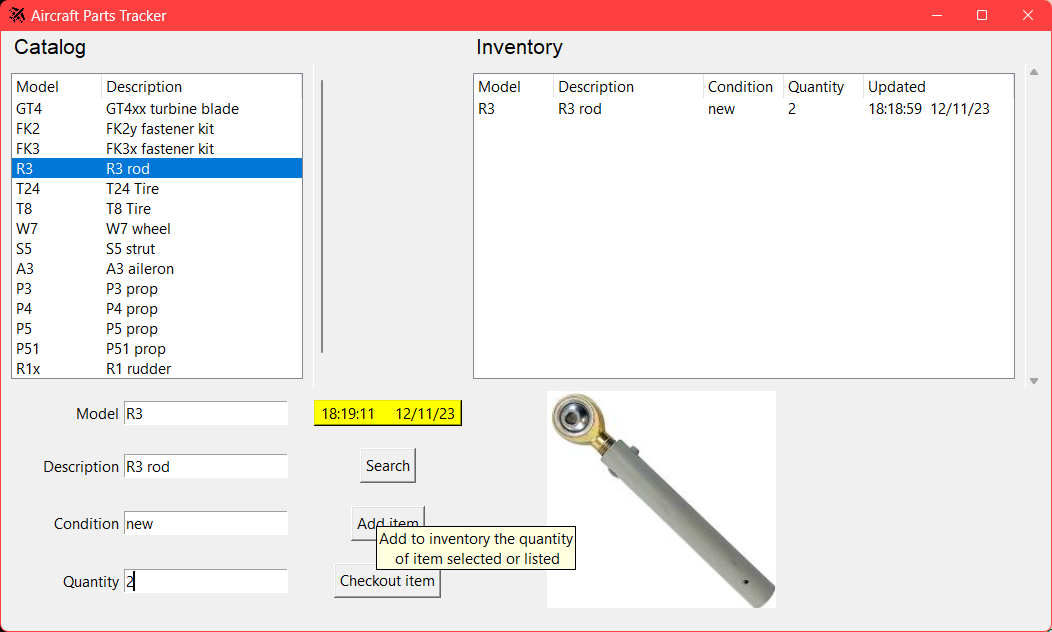
Starting with an empty inventory, and clicking Search button for model R3 (tooltip appears before the click). Image appears when item found:



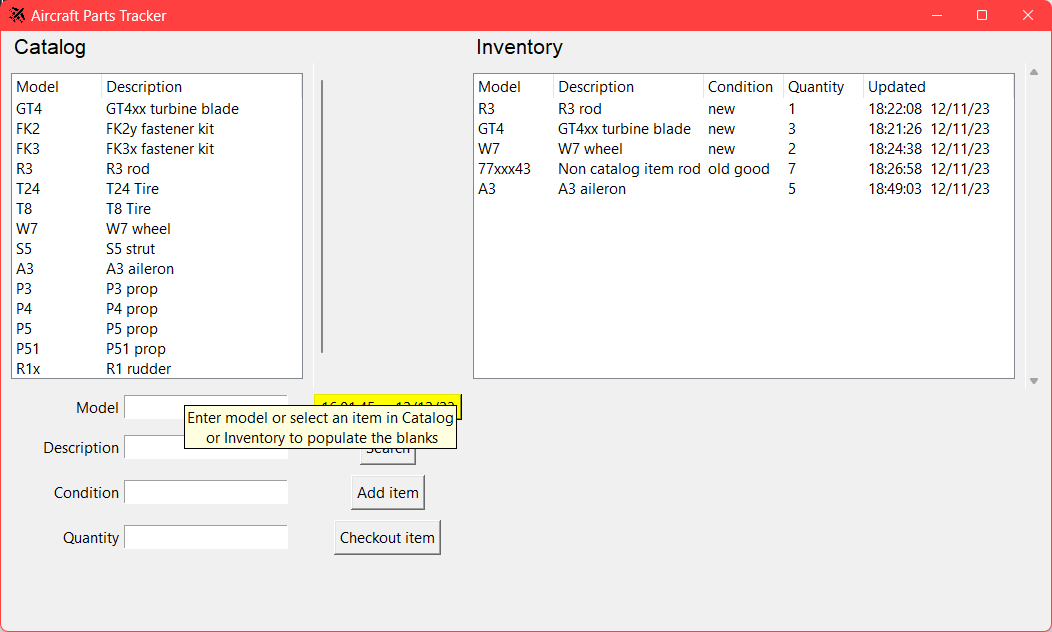
Starting with an empty inventory, and clicking Add Item button for model R3 (tooltip appears before the click). A warning dialog will appear for user if Quantity entry blank remains empty:



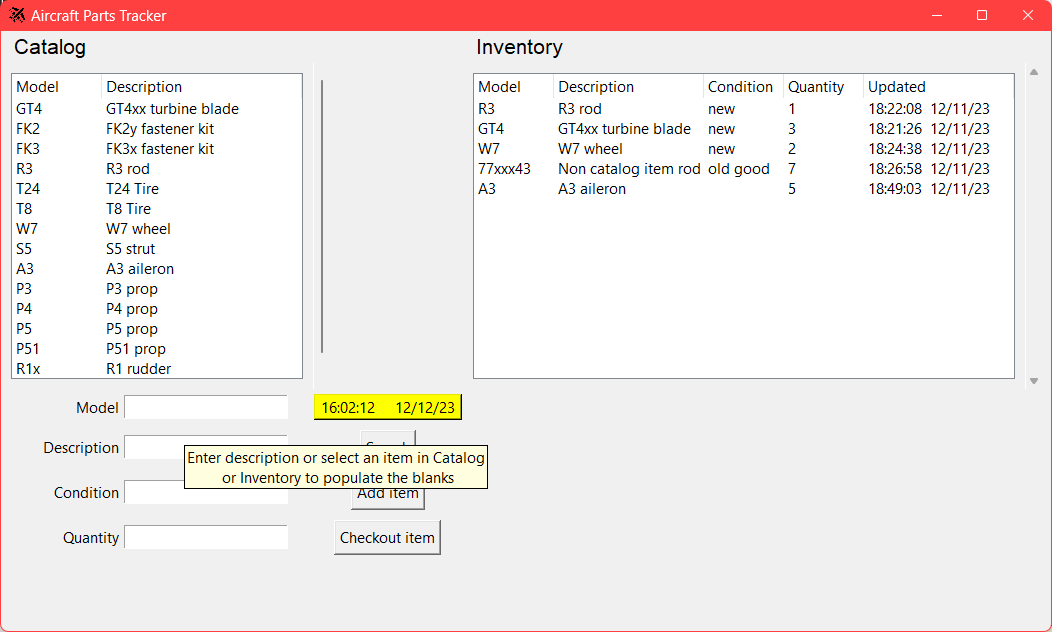
After adding Quantity 2 of R3 to inventory:



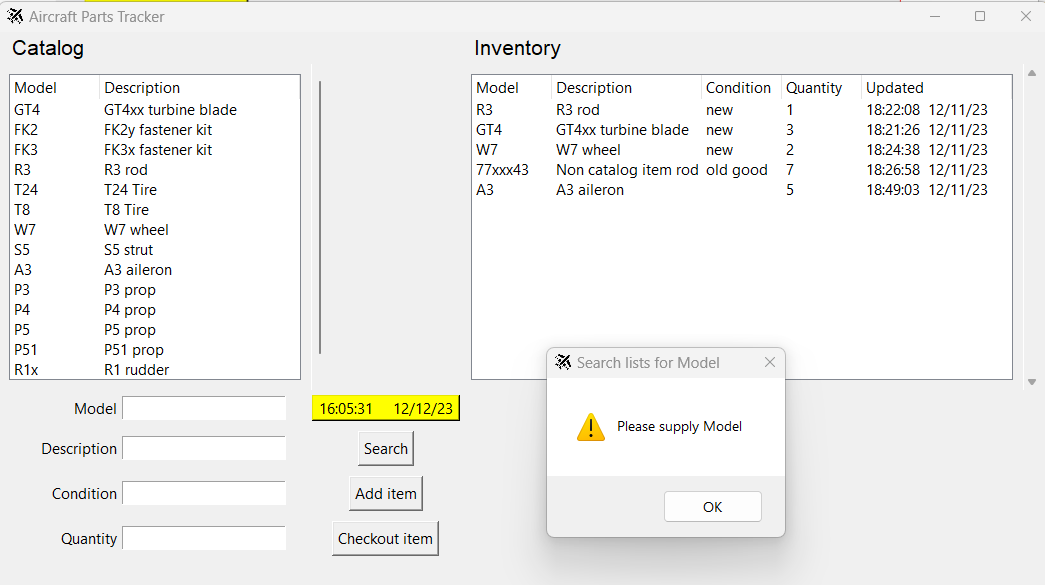
Tooltip for the Model entry blank:



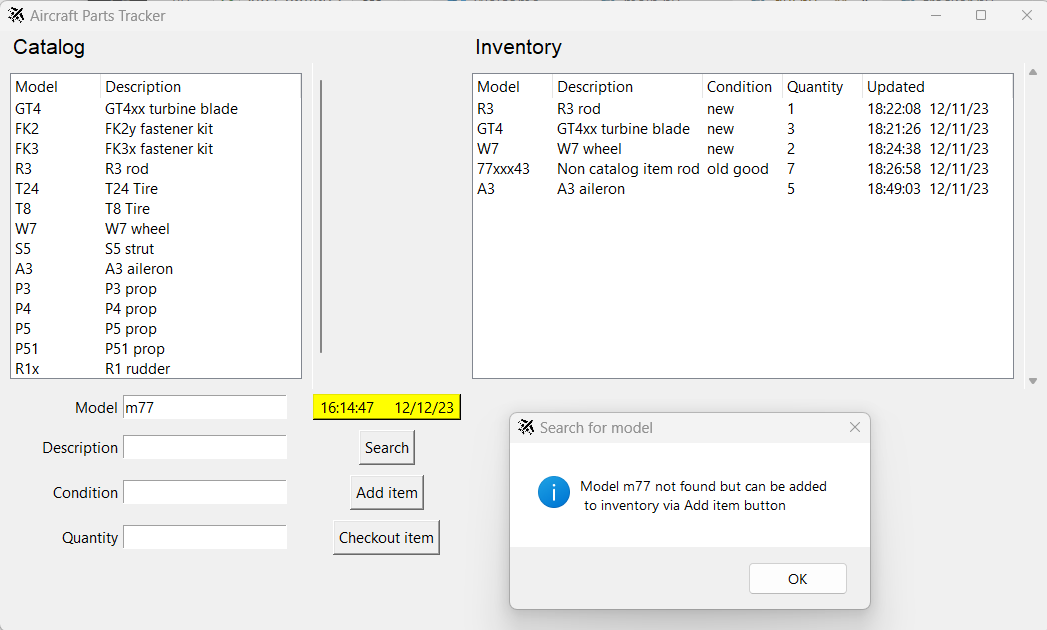
Tooltip for the Description entry blank:



User alerted via dialog when Search button pressed with empty Model entry:



User alerted via dialog when Search button pressed with a Model not in Catalog nor in Inventory:



**7. Conclusions and learnings:**

Tkinter has many features needed for a decently functional GUI. However, it is awkward to use, with so many functions and parameters. A good IDE-based WYSIWYG authoring tool like Visual Studio for C# is needed. We found a couple of websites that helped create a very basic GUI but were still woefully inadequate:

* www.python-gui-builder.com
* <https://visualtk.com/>

Perhaps making the app UI browser-based would have worked better.

SQLite3 for the SQL database was reasonably easy to use to create and maintain the database.

Handling all the possible exceptions for user data entry in a GUI requires a lot of code and testing.

A more thorough inventory database would use the unique part serial number as the key, and track the history of every part individually. However, that would require much more SQL (a table for each Serial number in addition to the inventory table, and a much longer inventory list table). The current implementation is a compromise that groups items by model number, which is typical when parts need not be individually tracked.