Source Code Documentation

For

Integrated Information Systems Inventory Tracking System

Version 1.0

Prepared by Hemroy Grant

CARICOM Secretariat

August 30, 2018

Table of Contents

1. Introduction

This document is intended for maintainers and developers of the Inventory Tracking System web application. The application will be used by staff of the Integrated Information Systems Programme to keep track of all assets assigned to them within the CARICOM Secretariat. The document will not cover HTML, JS and CSS languages, however, it will explain the framework used, including its subparts. The application UI was designed using Bootstrap 4 and Font Awesome Icons. The ITS server side was designed in Python using Django Framework.

Django includes its own default SQL engine (db.sqlite3) which was used during development. The Django framework includes an urls.py file that is used to create URL's using its integrated URL mapper which links to a template file via the views.py file. The views.py file also includes the database queries and functions used to return results to the front end (templates) of the application. The database tables are created in the models.py file.

This document will focus mainly of the three files mentioned above and explain each module individually.

2. Code Description

2.1 Models.py

The models included in the database are Staff, Inventory and Transactions as the main parts of the database. However, also included is another model named Mname used to store Manufacture name of the items in the Inventory table.
Below is an explanation of the individual models.

2.1.1 Staff

The staff model stores information on the users of the application. The fields required are: *first name, last name* and *position* within the Programme. Also, *username*, this field is a foreign key field linked to Django's default User model. The User model is the frameworks model for creating users with authentication for the application.
Therefore, users of the application are created in the User model with a username, password and account type, either individually or via groups (these includes the user permissions to add, change or/and delete). This account is then linked to the Staff model for use in the database.

2.1.2 Inventory

The inventory model stores information for each asset. The fields required are: *manufacture name, model, device type, serial number* and an optional field: *asset code*. The manufacture name is selected from the Mname model, if the name does not exist then the name can be typed in the input field then it will be automatically added to the Mname model. The asset code is optional and can be inputted later by editing the item when it is available.

2.1.3 Transactions

The transactions model stores information on each transaction made by each user on every item in the inventory until the item is disposed. After a disposal the item status change to disposed and cannot be transferred, or updated, unless an Administrator user changes its disposed status. Moreover, the fields required for this model are: *staff id, item id, date, operation, remarks, location, proprietor and status*. The date is automatically entered for each transaction made. The transactions made are indicated in the operation field as Add, Remove, Changed Status, Disposed or Edited. Remarks include by default the item name and operations performed with the users entered reason at the end of the remark.

2.2 Urls.py

There are two urls.py file. The main file is in the project files located in /iis_its_project/urls.py. The other is located in the application files in /its/urls.py.

2.2.1 Urls.py (/iis_its_project/urls.py)

This is the main urls.py file which includes the URL for Django's authentication urls (login, logout, password management). Moreover, it also includes the Django's admin urls which is the urls to the backend of the application.
The following URL Patterns will be explained:

- `path('admin/', admin.site.urls),`
  This path links the admin templates to the views of the application.

- `path('', include('its.urls')),`
  This path links the applications urls to the main **urls.py** file.

- `path('', include('django.contrib.auth.urls')),`
  This path links the Django's authentication templates to the views of the application. Those templates are found in under registration in the templates folder (/templates/registration/*.html).

2.2.2 Urls.py (/its/urls.py)

This file contains the urls for all the views of the application. The following URL patterns will be explained:

```
    path('', views.dashboard, name='dashboard'),
    path('inventory/', views.inventory, name='inventory'),
    path('staff/', views.staff, name='staff'),
    path('add/', views.add, name='add'),
    path('remove/', views.remove, name='remove'),
    path('transfer/', views.transfer, name='transfer'),
    path('dispose/', views.dispose, name='dispose'),
    path('update/', views.update, name='update'),
    path('/item/<key>/', views.item, name='item')
```

Each url pattern matches to a specific view in the views.py file.

2.3 Views.py

The **views.py** file is located in the same path as the last urls.py file mentioned above and the models.py file. The views is the file that links the templates to urls mentioned above. The views used in this application are:

- def dashboard()
- def inventory()
- def add()
- def remove()
- def transfer()
- def dispose()
- def update()
- def item()
- def nodupes()

2.3.1 def **dashboard**(request):

This view displays the dashboard template when the browser request the dashboard url. It matches url with the index.html template and includes the results from the query sets within the dashboard view.

2.3.2 def **add**(request):

This view displays the add item template when the browser request the add url. It matches the url with the add.html template and includes the results from the query sets within the view. The add view checks if the user fills out the form on the template and when submitted the details entered are added to the inventory and transactions model.

2.3.3 def **remove**(request):

This view displays the remove item template when the browser request the remove url. It matches the url with the remove.html template and includes the results from the query sets within the remove view. This view checks if the user filled out the form fields upon selecting the item to remove and when it is submitted the details entered are stored in the transactions model and the item is removed from the inventory model.

2.3.4 def **transfer**(request):

This view displays the transfer template when the browser request the transfer url. It matches the url with the transfer.html template and includes the results from the query sets within the view. The view checks if the user submitted a transfer form and adds the details entered into the transactions table which keeps track of location and proprietor of the item.

2.3.5 def **dispose**(request):

This view displays the dispose template when the browser request the dispose url. It matches the url with the dispose.html template and includes the results from the query sets within the view. The view checks if the user submitted a dispose form and adds the details entered to the transactions model which updates the status of the item to dispose.

2.3.6 def **update**(request):

This view displays the update status template when the browser request the update url. It matches the url with the update.html template and includes the results from the query sets within the view. The view checks if the user submitted an update form and adds the details to the transactions model. The status field for the item is updated in the transactions table and the operation field is assigned Change Status.

2.3.7 def **inventory**(request):

This view displays the inventory template when the browser request the inventory url. It matches the url with the inventory.html template and includes the results from the query sets within the view. This view lists all the items in the inventory and it comprises of all the functionalities of the views mentioned above. Within this view the user can add, remove, dispose, transfer and update items. The view checks to see which form was submitted by the user and performs the necessary actions for each function.

2.3.8 def **item**(request, key):

This view is accessed by clicking on an item in the inventory table. The item template is displayed when the browser sends the request for the item url. It matches that url with the item.html template and includes the results from the query sets within the item view. In this view a user will be able to perform remove, update, dispose, transfer and an additional operation of Edit. The user will be able to edit the item record in the database for the inventory and transactions model. The value entered in the edit form will update the fields in inventory and add a transaction record with the operation of Edit.

2.3.9 def **nodupes**():

This method is used to query the transactions model to return all the items in the inventory table without returning duplicates of the items. This method is called upon in all the above views.