Inventory Management System

IFT 402: Information Technology Capstone

Professor Damien Doheny

Professor Usha Jagannathan

Nathan Glucksman

2017

# 

[Declaration of Originality 2](#_Toc480885002)

[Abstract 2](#_Toc480885003)

[Overview 3](#_Toc480885004)

[User Guide 4](#_Toc480885005)

[Original Proposal 5](#_Toc480885006)

# Declaration of Originality

I, Nathan Glucksman, state that I am the sole author of this work. This project is submitted as part of the IFT 402: Information Technology Capstone Project course for Arizona State University.

Nathan Glucksman, 4/25/2017

# Abstract

This paper describes the implementation of an Inventory Management System through the Django Framework written in Python. It uses a web-based front end with a back-end in various programming languages including database software for managing multiple stores’ inventories. It is expansible to include further features as needed by the client. A user access control module is included to restrict access to authenticated users.

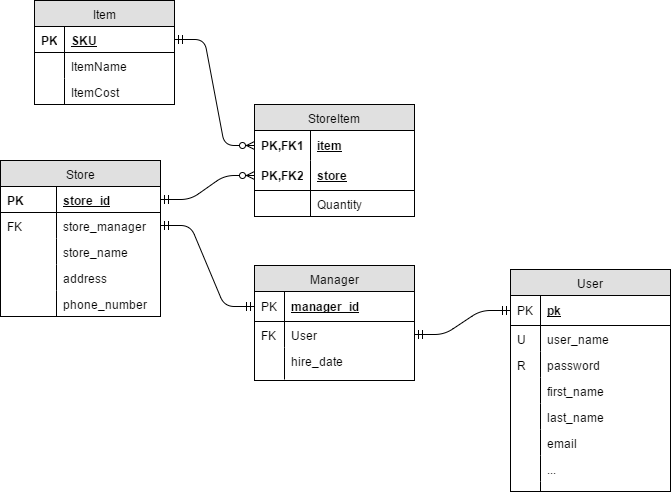
# Overview

For my capstone project I asked a peer in the IT field for some advice on what he thought would make for a good learning experience. He introduced me to the Django framework and explained some of its potential uses. One of the most common uses was as an inventory management database. I use a similar program at my job and thought it would be great to reverse engineer it into a more modern programming language and make it as extensible as possible.

I created my own team for this semester. Documentation was kept organized by the use of a journal. In it I included a record of what I accomplished during each day I worked on the software as well as a prioritized TODO list to keep myself organized. These items were created from personal insights gathered from the model software (an inventory database I use at my work) as well as input from a peer who would act as a user for me to gather stories from. Prioritize were determined on factors of required functionality (ie creating working models, an active webpage, etc.) and things I wished to learn and experience (ie a user authentication module).

To begin with I had to create my development environment. I chose a virtualized installation of Ubuntu 16.10 (UbuntuWebsite) in order to further my knowledge of Linux as well as for its flexibility when dealing with multiple programming languages. Most of the programming was done in a python environment using the Django framework (Django). I also created a virtual python environment in order to make my code more portable. Finally, all of my code was kept version controlled using github (GithubSource).

The first step was to create my database models. By default Django Framework interacts with SQLLite3. It can be changed in the future to support any range of database software, but for simplicity I used the pre-packaged software. Below is the ERD for the current models:



Some of these models are very simple: The item model is simply a primary key with some fields attached to it. There are also instances of foreign keys being used, like in the ‘Store’ -> ‘Manager’ relationship. I’m particularly fond of the intermediary class ‘StoreItem’. This uses the Item and Store models as a unique pair of foreign keys to store information about a particular store’s relationship with an item, such as quantity held in inventory.

Handling of the database tables and rows is taken care of by the Django framework, so each of these is created by the handler program ‘manage.py’. Making changes is quick and simple and can be done in real time without taking down the front-end. This also allows python methods to run while manipulating data. For example, upon creation of a new ‘Store’ row a set of all combinations of the new store and each ‘Item’ is created in the ‘StoreItem’ table. Similarly the creation of a ‘User’ causes a ‘Manager’ to be created.

The front-end web interface is also handled by the Django framework. It parses URLs passed to it into a set of view rules, which create HTML documents based on a set of templates and database table information. This allows webpages to update their contents based on changes to the database. It can also be used to change website layouts across the entire webpage by changing the way Django creates the HTML files.

Finally there is an underlying access control module. Unauthorized users are not allowed to view critical information like details about a store’s inventory set. It also allows users to change their own passwords and information, such as name, email address, phone etc. Since each manager is tied to a store it also provides a good way of limiting access to each store to each individual manager. In a production environment it can be used to fully restrict every user to an individual store.

# User Guide

### Page list and functions

* Store list – A list of the stores available in the database. Each store link leads to details regarding the store’s inventory
  + Store Detail – Each store has a list of the store’s name, its manager, and finally its inventory as stored in the ‘StoreItem’ table. Clicking on an item brings you to its item detail page.
* Manager list – A list of the managers in the database. Includes their name, number, and email address.
* Items list – a list of the items stored in a database. Each link is a page that has the item’s details including a description.
* Admin Menu – A link to the admin page of the Django framework. Only members of the ‘staff’ user group can access this page.
* Store Recount – Authenticated users can edit the amount of inventory in a particular s tore

### User functions

* Logging in – Clicking the ‘login’ link will bring a user to a login page. Users can enter their username and password and then be redirected to the original page.
* Logging out – Clicking the ‘logout’ link will allow users to logout.
* Changing personal info – By clicking on a logged in user’s name, that user can change their name, phone number, and email address.

# Original Proposal

I got in contact with a friend who currently works for an IT department. He recommended I look into the Django Framework as a starting point for my project. After reading through the documents I noticed many similarities between its capabilities and a program I use at my job: An inventory management system.

I want to take the concept and execution of this inventory management system and apply it to the Django Framework. I want it to be able to do the ‘typical’ inventory tracking across multiple stores with abilities to manipulate the inventories based on different real-world conditions, such as ordering, transferring, physically recounting and daily sales. It should be able to handle multiple stores with separate logins for each store location. There should also be an administrator that can create new stores/logins.

I have a few stretch goals in mind if all goes according to plan. I want to enable communication between this inventory database and a separate ‘distributor’ database. The Inventory management system will create an order list to send to the distributor, and the distributor program will automatically fill the order and send an invoice back to the inventory management system. I also want a separate program to automatically fill out any new orders with a prediction based on the previous X days worth of sales.

Journal

## January 22nd, 2017

Set up environment

* Ubuntu
* Python
* Django - https://www.djangoproject.com/
* Virtual machine Shared Folder

Begin todo list

## January 25th, 2017

Read through tutorials for Django

## January 29th, 2017

Setup Github

* <https://www.howtoforge.com/tutorial/install-git-and-github-on-ubuntu-14.04/>
  + Git config –global user.name “user\_name”
  + Git config –global user.email “Email.id”
  + Git init ## IN SOURCE FOLDER
  + Git add . ##add everything in this folder
  + Git commit –m “MESSAGE’
  + ##Create repository on Github
  + Git remod add origin *URL*
  + Git push origin master

Flesh out Backlog / TODO list

Setup SDE

* Django
  + <https://docs.djangoproject.com/en/1.10/intro/install/>

This program was recommended by a friend in the IT field to manage the database and quickly create the programs I needed.

* PyDev
  + <http://www.pydev.org/>
  + <http://www.pydev.org/manual_adv_django.html>
* Liclipse
  + <http://www.liclipse.com/>

PyDev and LiClipse were chosen as my IDE of choice to use when writing my code. I have used eclipse in the past and LiClipse is a community updated version that includes PyDev.

## February 2nd, 2017

I started today by going over the work I had previously completed. However when I tried adding an ‘app’ as Django calls modules, or running the server, I got errors that indicated my install hadn’t gone according to plan. After reviewing the documentation and starting the project over a couple times I finally got a solid installation.

I had originally installed a Virtual environment for Django / Python to work on. This was done with the command *pynenv.* It was recommended during the pydev installation. I was unaware, however, that Liclipse needed to have the virtual interpreter set up in order to compile and use many of the Django functions. This caused a slew of errors when trying to edit programs in my environment.

I also had quite a bit more work to do when setting up a Django server. In starting through the tutorials I learned there was a lot of command-line commands using the *manage.py* program included with the framework. I found out that thankfully LiClipse could directly access the *manage.py* functions to create apps start projects etc.

I also discovered migrations, which are required when dealing with the databases:  
*You should think of migrations as a version control system for your database schema.****makemigrations****is responsible for packaging up your model changes into individual migration files - analogous to commits - and****migrate****is responsible for applying those to your database.*

I now have a working website with the tutorial programs running on it.

## February 12th, 2017

Created an ERD here: <https://drive.google.com/file/d/0B-J7GIH5FCJ5aExURFE5QlR6N0E/view?usp=sharing>

Do:

* Write ERD (done)
* Finish Tutorials (done)
* Email professor with updated ERD and a timeline for a working program (done)

## February 19th, 2017

Do:

* Setup models for IMS (Done)
* Setup views for debugging (store only)

I’ve gotten through enough of the tutorials to take a stab at creating the IMS skeleton. Working with multiple primary keys takes a different kind of field declaration. This was documented under: <https://docs.djangoproject.com/en/1.10/ref/models/fields/#django.db.models.ManyToManyField> There were some issues creating the models from there, but after a few bugs were worked out creating new stores via the web-based admin menu in Django.

## February 27th, 2017

Adding models to the admin screen becoming very easy. This week added manager, item, and stores. Added items into the admin menu and tested views with these new objects. After trying to add InventoryItem the whole site breaks down. It likely has something to do with the ManyToManyField unique\_together. I’ll have to redo some of the model code with through instead.

## March 15th, 2017

I’ve spent the last month reworking some test cases on my previous models along with adding the final models, StoreItems, and a handful of classes to be used in the future. The previous manytomanyfield attempt did not work, and I had to rewrite a large portion of the code to accommodate my new method of using a set of paired foreign keys as a primary key for a table. However the next large problem I’ve come across is actually rendering the intermediary class into a coherent HTML page. This is going to take a LOT of research…

## March 27th, 2017

GREAT SUCCESS, I’ve finally got the intermediary class passed through my urls, through my views, and into my template. This allows me to render the variables I want on the web page quickly, easily, and in a much more modular way than I imagined would be possible. I spent a good amount of time working with templates and views and have created many new debugging functionality as well as templates. The next step is to allow a user to change the quantity of an item in a store, again using the intermediary class.

## April 2nd, 2017

Finalized work on templates that added editable forms for anyone to change the inventory in a store. Creating these took learning how to use a new Django class: Forms. The made creating the HTML for form editing much easier than completing it manually. I also had to create and add handlers for HTML requests, such as GET and POST. When receiving a POST request the program had to update the back-end database.

## April 15th, 2017

HTML templates have been completed. A base HTML has been created to use as an extendable template for all other pages. This allows updates to be made across the entire site quickly and with as little repeated work done as possible. This adds links to all other pages and allows users to navigate around the page, when previously everything had to be typed in directly.

CSS Beautification is also complete. These are stored in a separate folder managed by Django. The files can be quickly extended to new webpages, and also used on new Django apps by referencing them in new projects.

## April 22nd, 2017

User authentication has been added. Users have been created but are currently not directly used in the site.

## April 23rd, 2017

## TODO April - Backlog

1. CSS Beautification and HTML linking (4/15)
2. Extendable Base template (4/15)
3. User Authentication (4/22)
4. User Access Control (4/22)
5. Intermediate views and functionality ( see February backlog )

### User Requests

* Allow managers to add / remove items to the store
  + Forced the system to create pairs between stores and items upon each’s creation. (4/23)

## TODO February – Backlog

### Models –

* Basic Models for
  + Item (2/19)
  + InventoryItem (2/19)
  + Store (2/19)
  + Manager (2/19)

### Views –

* Basic Views to verify Data (2/27)
  + Store (2/19)
  + Managers (2/27)
  + Item (2/27)
  + All Inventory (2.27)
* Intermediate Views
  + Admin store changer
  + Inventory in a store (3/20)
  + Recount Inventory (4/2)
  + Inter-store Transfer
  + Items sold in a day
    - Lists all items in store, editable qty field
    - Removes inventory from store based on qty field
  + Order (per store)
    - List all item with editable qty field
    - Adds qty of items to store
* CSS beautification
* Advanced
  + Auto-populate order based on most recent items sold in day (**EXTRA)**

## Tests –

* Test cases for each model

## TODO January (Out of Date)

Create object relationships – ERD DONE 2/12

Inventory Management –

* Logins: Manager, Admin
  + **TABLE**: Admin store changer
    - List allows admin to pick which store to edit
* On-Hand inventory tracking
  + **TABLE:** Inventory in a store (Manager, Admin)
  + **TABLE:** ReCount Inventory (Manager, Admin)
    - List all items in store with expected value, editable qty field
    - Manually add/remove inventory (Manager, Admin)
    - Select by Department (EXTRA)
  + **APP**: Inter-store transferring
    - Ability to move inventory between stores
* Add/remove stores (Admin)
  + Add accounts to access individual stores (Admin)
  + Auto-add all inventory to store, qty 0
* Daily ‘sales’ to reduce inventory
  + **TABLE**: Items sold in a day (Manager, Admin)
    - Lists all items in store, editable qty field
    - Removes inventory from store based on qty field
  + Items w/ atomic ingredients? (milk, syrup,
  + Random numbers based on a usage chart? (Admin)
* Ordering
  + **TABLE:** Basic ‘order’ (Manager, Admin)
    - List all item with editable qty field
    - Adds qty of items to store
    - Auto-populate order based on most recent items sold in day (**EXTRA)**
  + **TABLE:** List of previous orders (Manager, Admin)
    - List all orders by store
    - **TABLE:** Order (Manager, Admin)
      * List items ordered in a previous order (qty>0)
* Receiving (EXTRA)
  + **Distribution centers**
    - Second database with Distribution centers inventories
    - Allows receipts, invoices and ‘communication’ between two databases