Food Pantry Inventory System

Release 0.1.1a1

(See Contributors)

Mar 01, 2021

OVERALL CONTENTS:

This system is designed to manage a local food pantry inventory. Initially it will manage the inventory in the warehouse.

CHAPTER

ONE

USER DOCUMENTATION

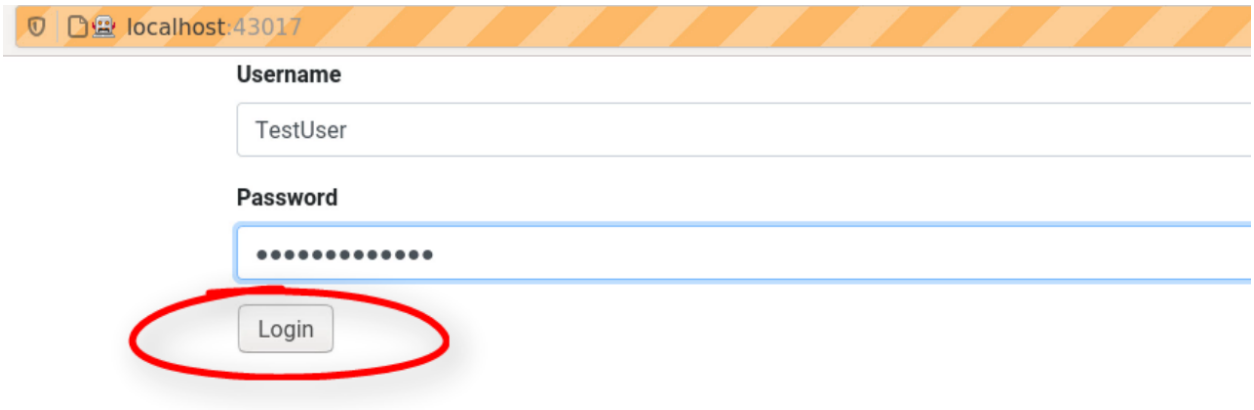
1.1 Overview

Describe an overview of the sytem and what equipment is expected to be able to access it.

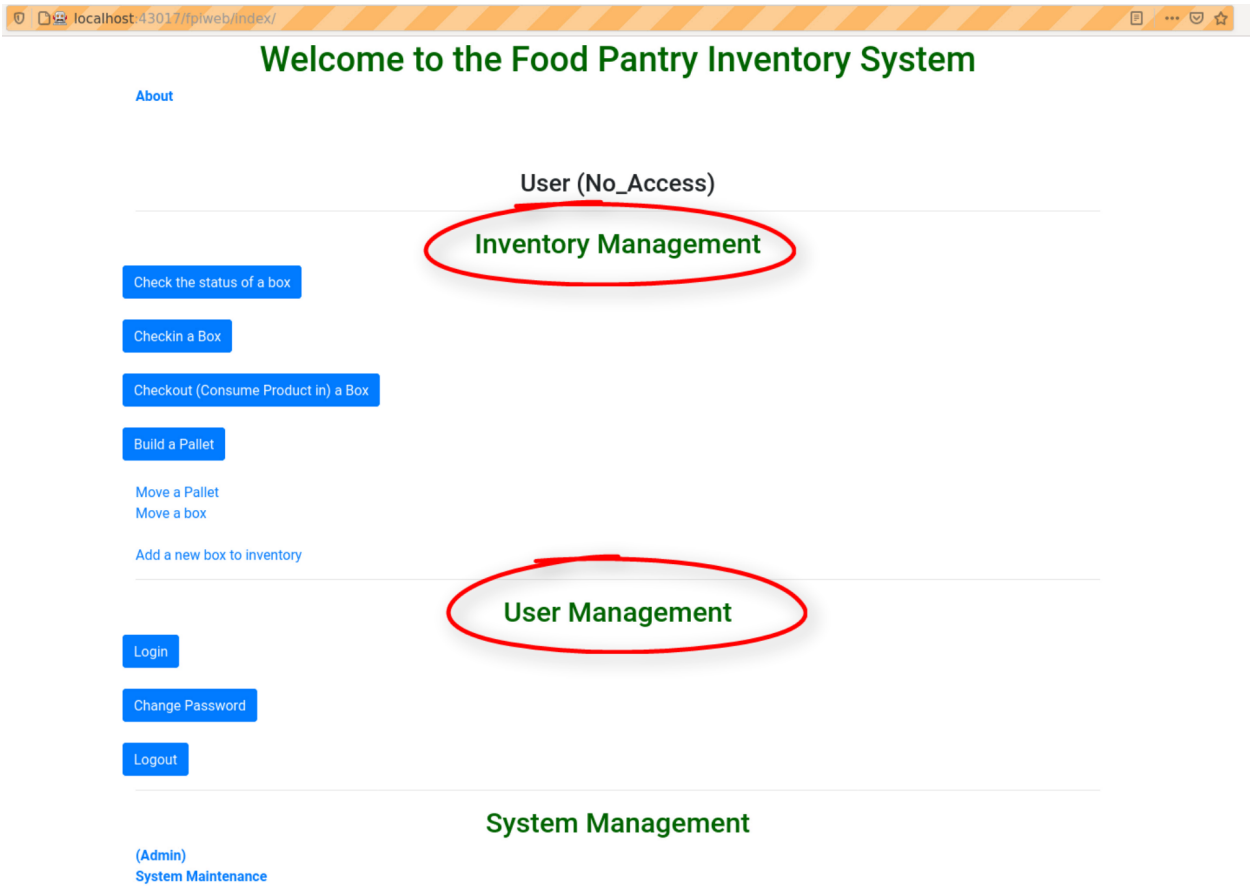
1.2 User Management

1.2.1 Login

Start at the **Login Screen** by entering your user name and password. In the graphic below the user name is 'TestUser' and of course the password remains hidden on the screen. Then click the **Login** button with your mouse.



You should now come to the **Main Menu** screen as seen below. Here you will be working with the **User Management** menus and the **Inventory Management** menus. **System Management** is for program administrators and you should not be able to click on and enter those menu items.

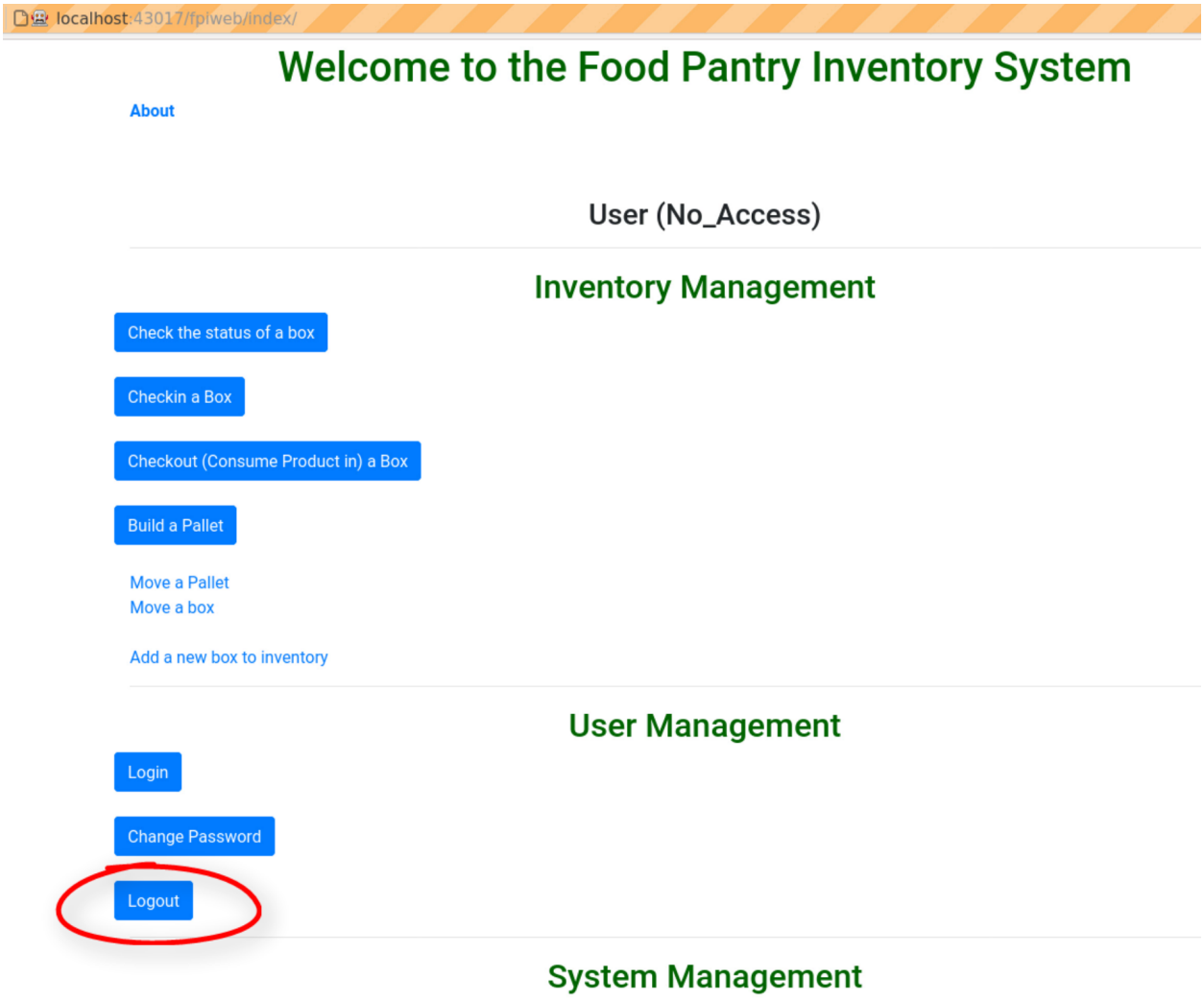


1.2.2 Change Password

To change you password click on the **Change Password** button. Unfortunately this page is not implemented yet. If you click on the **Change Password** button you will be directed to the **About Screen** as shown below. Click on **Return to main page** to return to the **Main Menu** screen.

1.2.3 Logout

Whenever you have finished you should always log out! This will prevent the next person using the program from making inadvertent changes in your name. Always click on the **Logout** button when you are finished using the program.

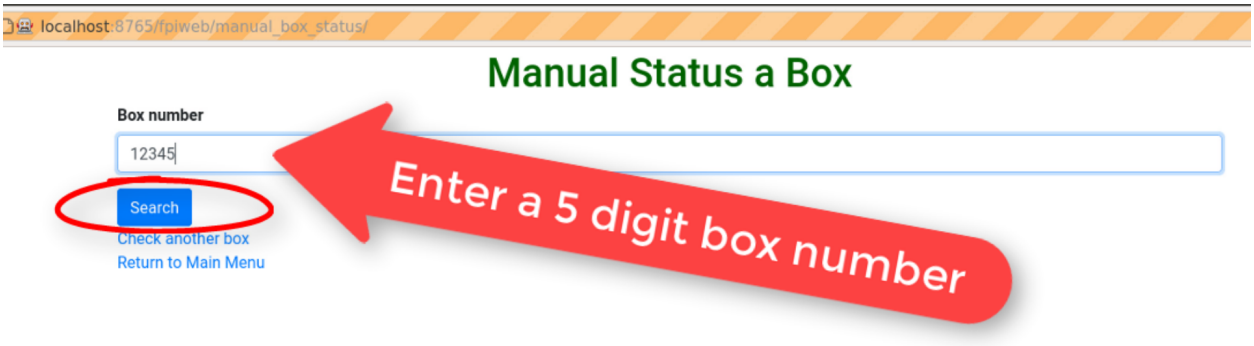


After logging out you should see the **Now logged out.** screen. To go back to the login screen click on **Go to login page.**

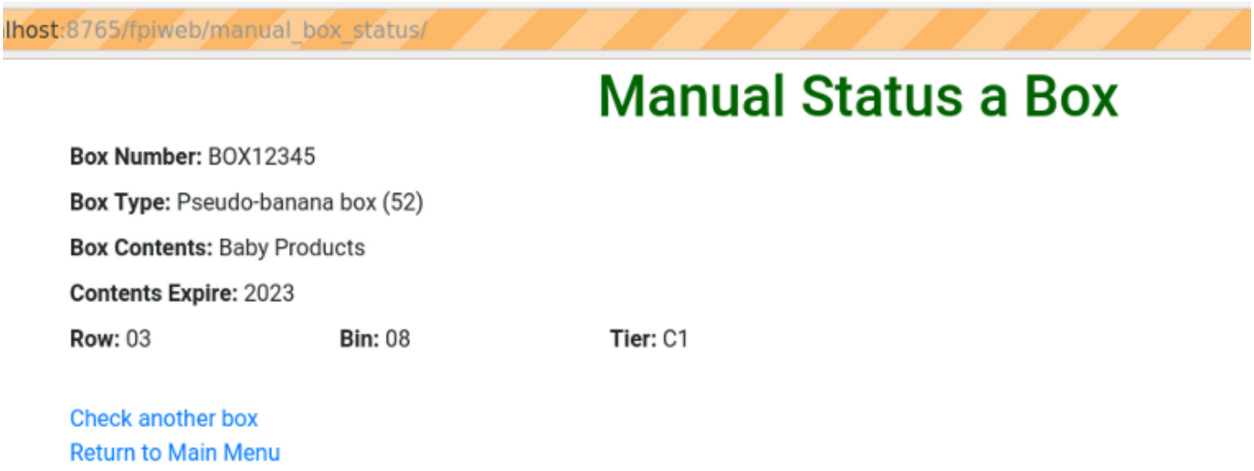
1.3 Inventory Management

1.3.1 Check the Status of a Box

From the Inventory Management section on the Manin screen clicking on **Check the status of a box** will bring you to the **Manual Status a Box** screen. Here you can check the box status by entering a 5 digit box number and then clicking on the **Search** button.

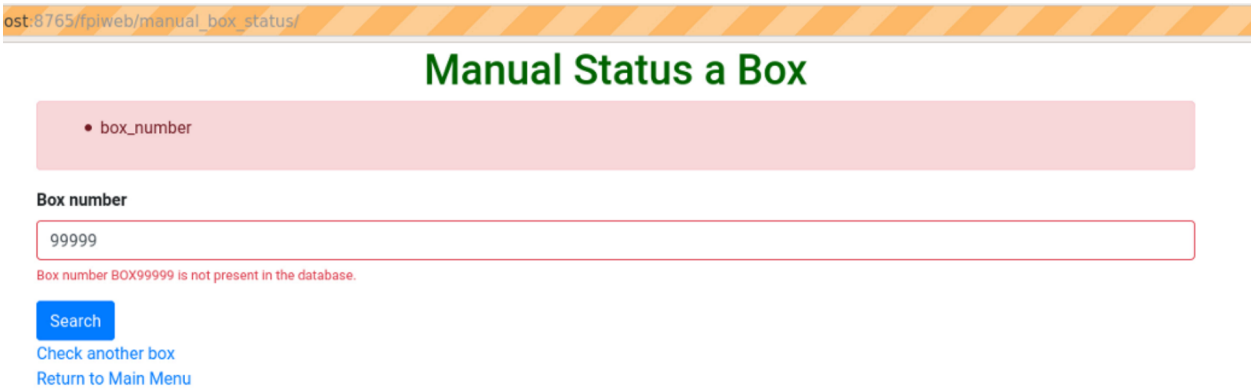


After clicking on the **Search** button you should see a screen listing the Box Number, Box Type, Box Contents, Contents Expire (expiration date) and location- Row, Bin and Tier number.



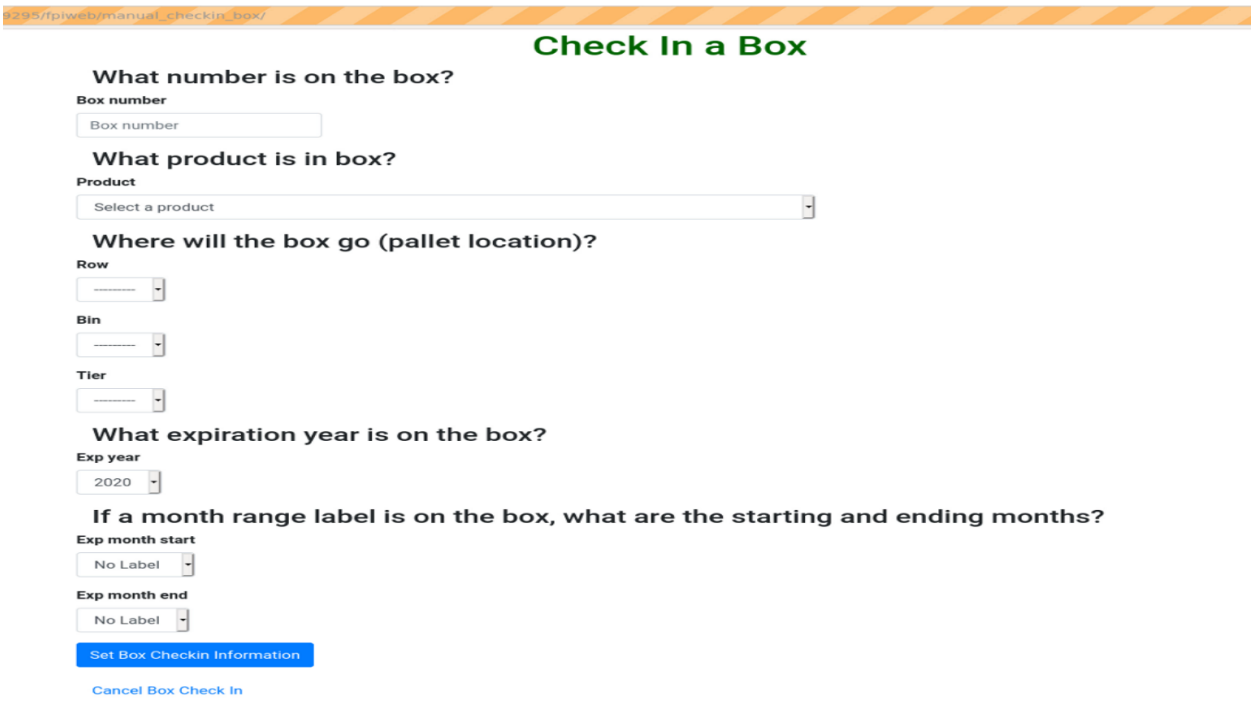
From this screen you can click on **Check another box** or **Return to Main Menu**.

If you see the screen below that means the **Box Number** is invalid or has not been entered into inventory.

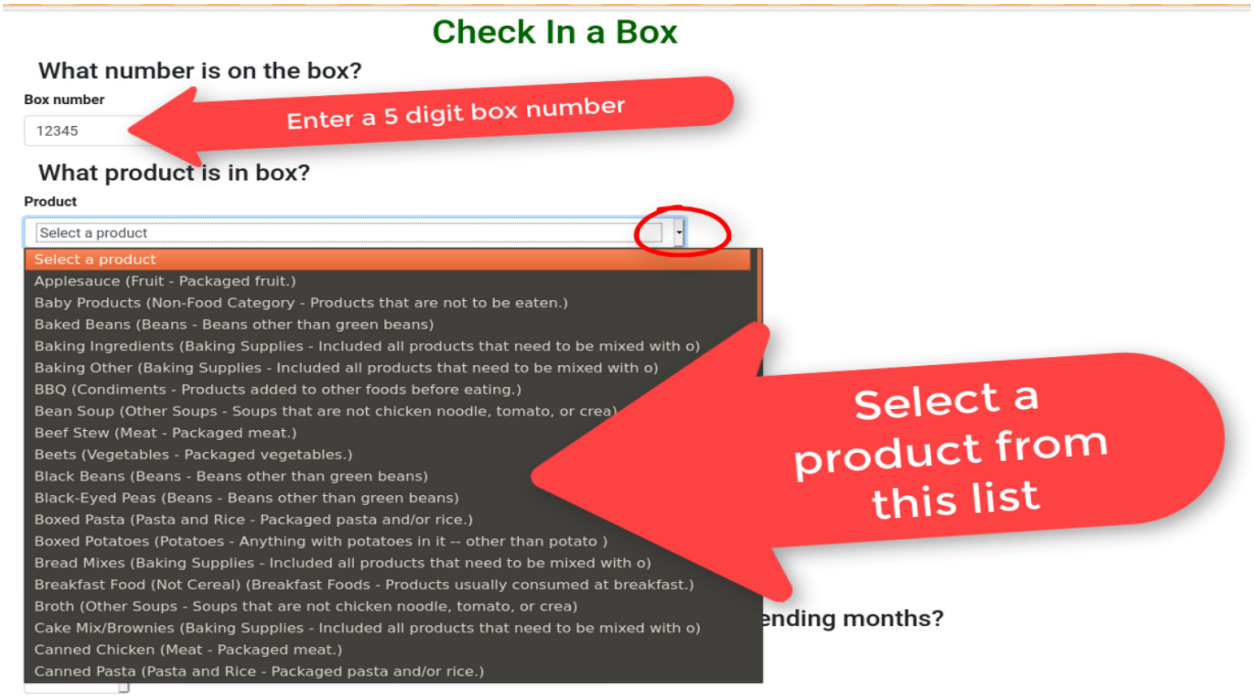


1.3.2 Checkin a Box

With this screen you can add all the information needed to check in a box of food items to warehouse inventory. The blank **Checkin a Box** screen is shown below.



The first two items to enter are the **Box number** and **Product**. With the keyboard enter a 5 digit 'Box number' in the **Box number** field. Then click on the small triangle in the circle at the right of the **Product** drop down list to select a product with your mouse.



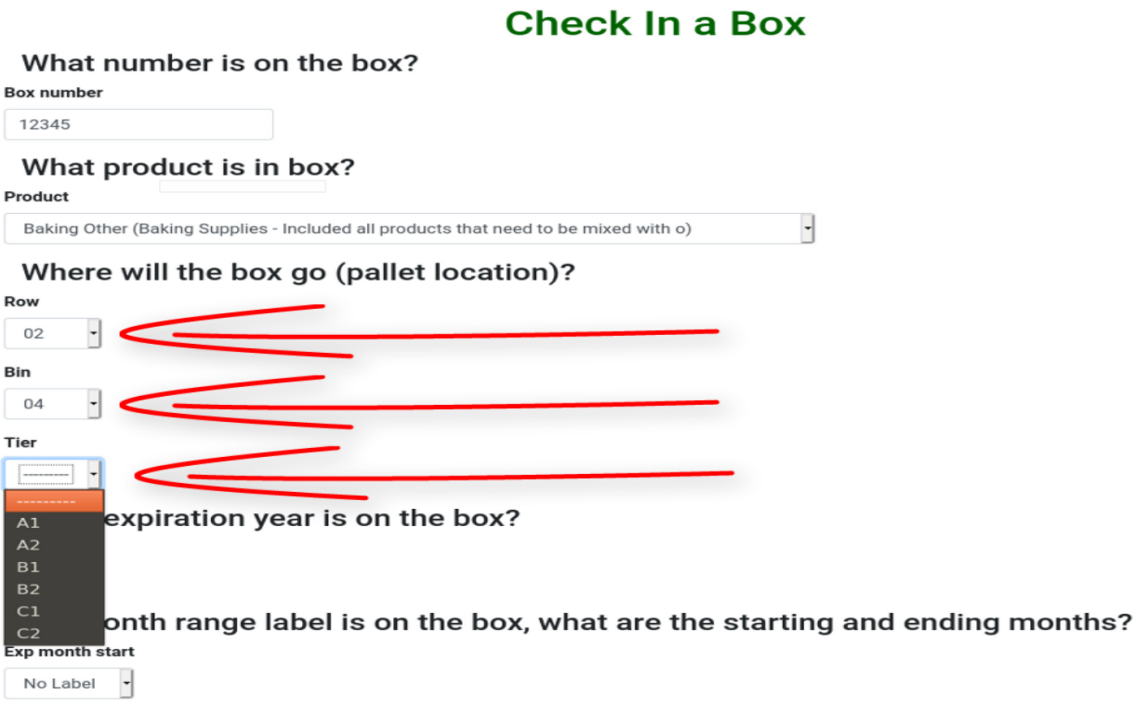
After entering the **Product** the next step is to enter the pallet location. You will have to select 3 different entries with your mouse.

(1) Row number

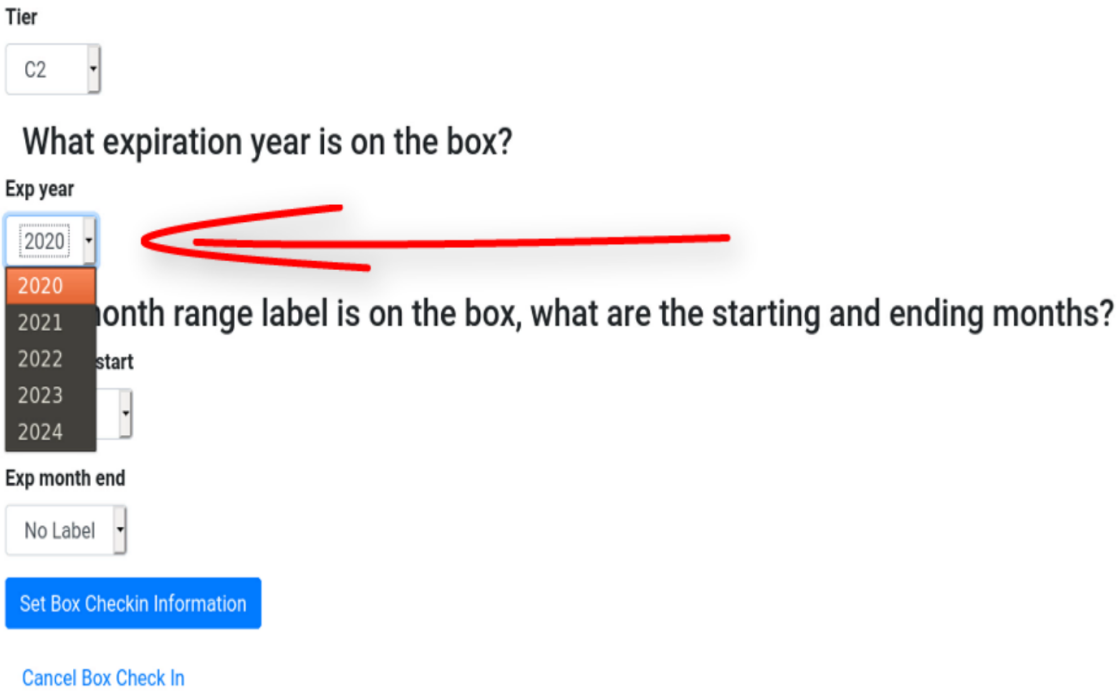
(2) Bin number (Bin number in the length of a row)

(3) Tier number (level up or down

Each entry uses a drop down list and you will make a choice from each drop down list. In the graphic below the **Row** and **Bin** numbers have been chosen and the drop down list is shown with the **Tier** number.



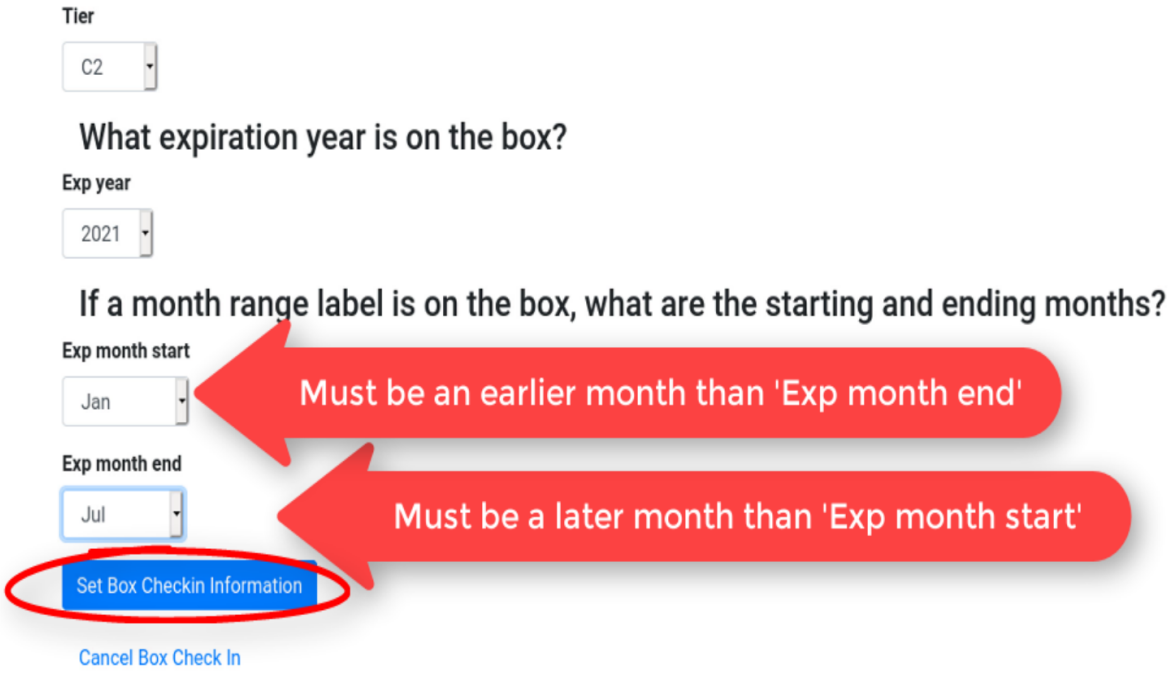
After entering the location enter the expiration year by clicking on the small triangle to the right of the **Exp year** field and making a selection from the drop down list.



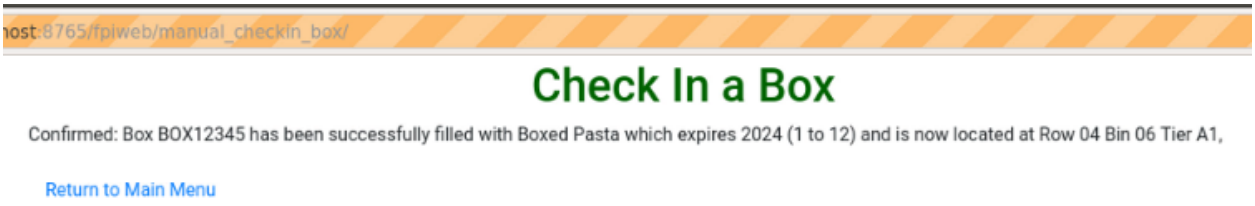
The last selection you will have to make is the expiration month. There are 2 drop down list fields here **Exp month start** and **Exp month end**. These fields are optional and it is not necessary to fill these fields out. These drop down

lists show a list of all the months in the year. You can also choose to **only** fill out the **Exp month start** field also. However if you choose to fill out the **Exp month end** field you must make sure it is a later month in the year than the **Exp month start** field. So if you enter **April** in the **Exp month start** field, then you must enter **May** or a later month in the **Exp month end** field.

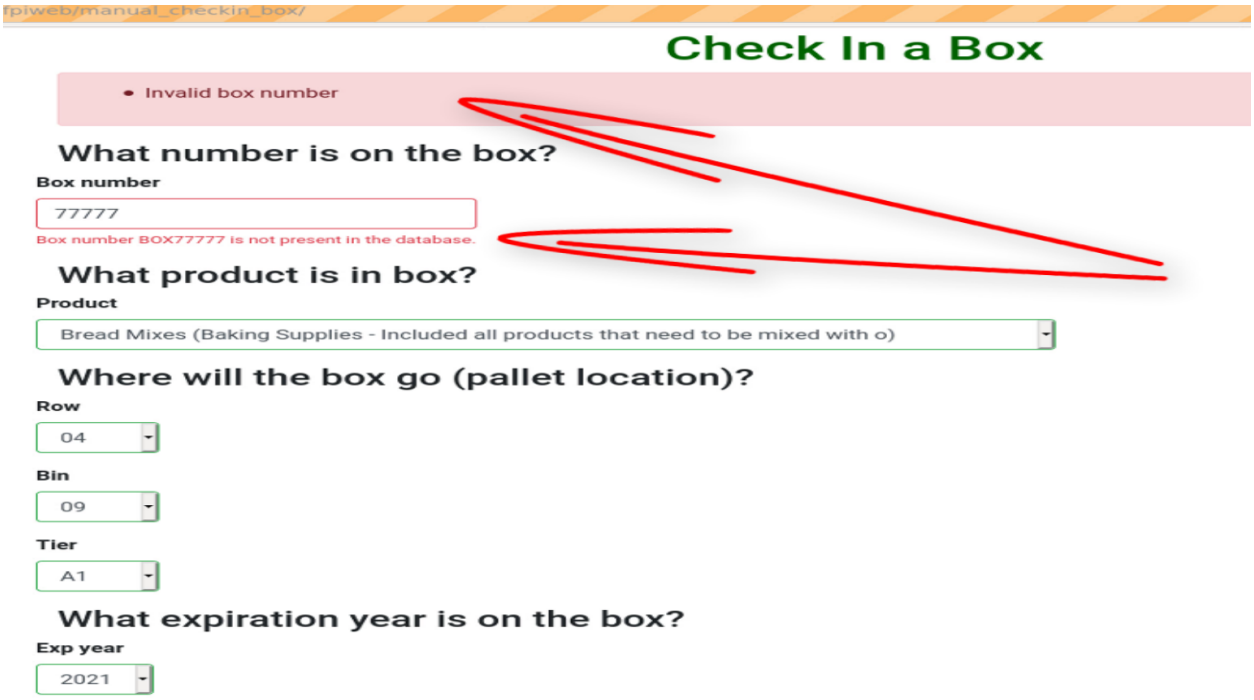
When you are through with the **Exp month** fields click on the blue **Set Box Checkin Information** button to enter all the Checkin data.



If everything has worked correctly you should see the following screen below. Simply click on the **Return to Main Menu** link to continue.

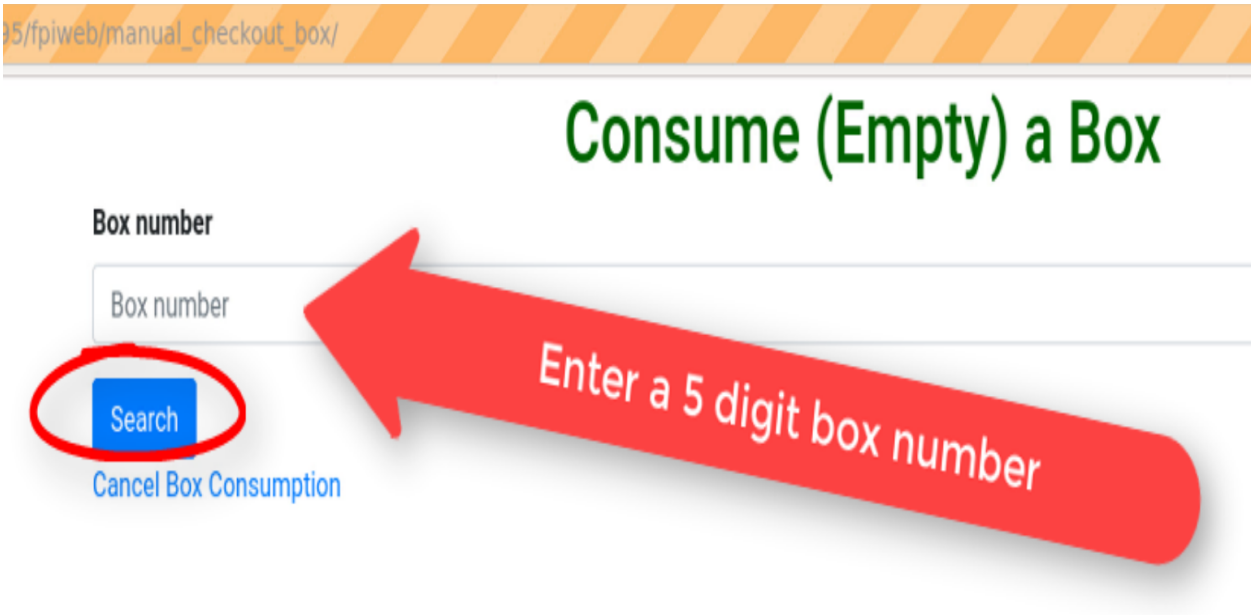


If there is an error you should see a screen like the one below. The red arrows point to what has to be fixed before a box can be Checked in. Click on the **Cancel Box Checkin** link at the bottom of the page to go back to the **Main Menu** screen to start over.

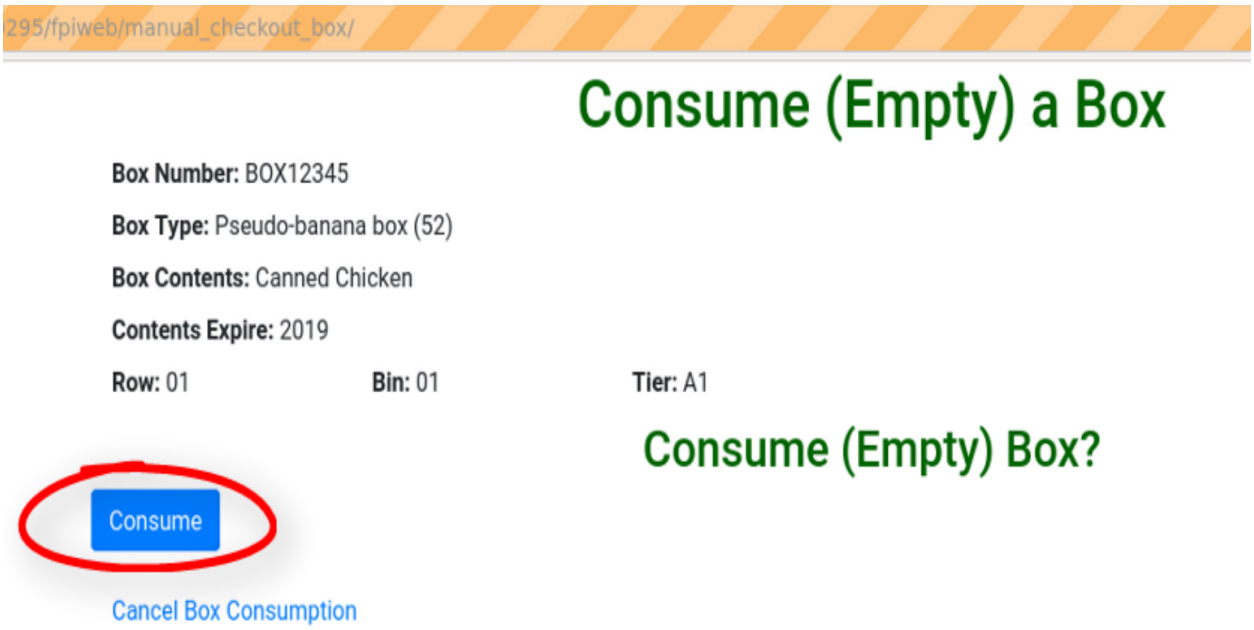


1.3.3 Checkout (Consume Product in) a Box

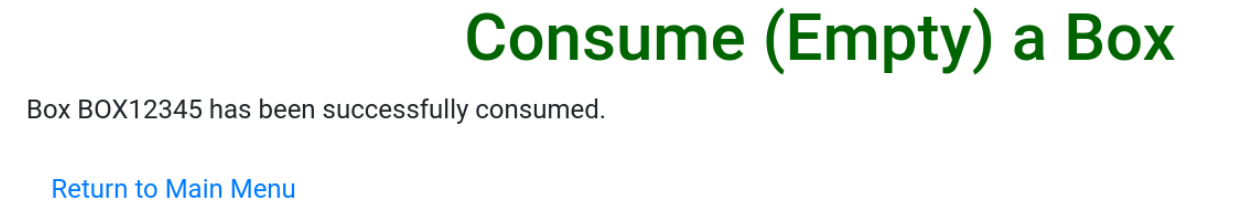
To consume or empty a box enter a 5 digit box number in the **Box Number** field. Then click on the blue **Search** button.



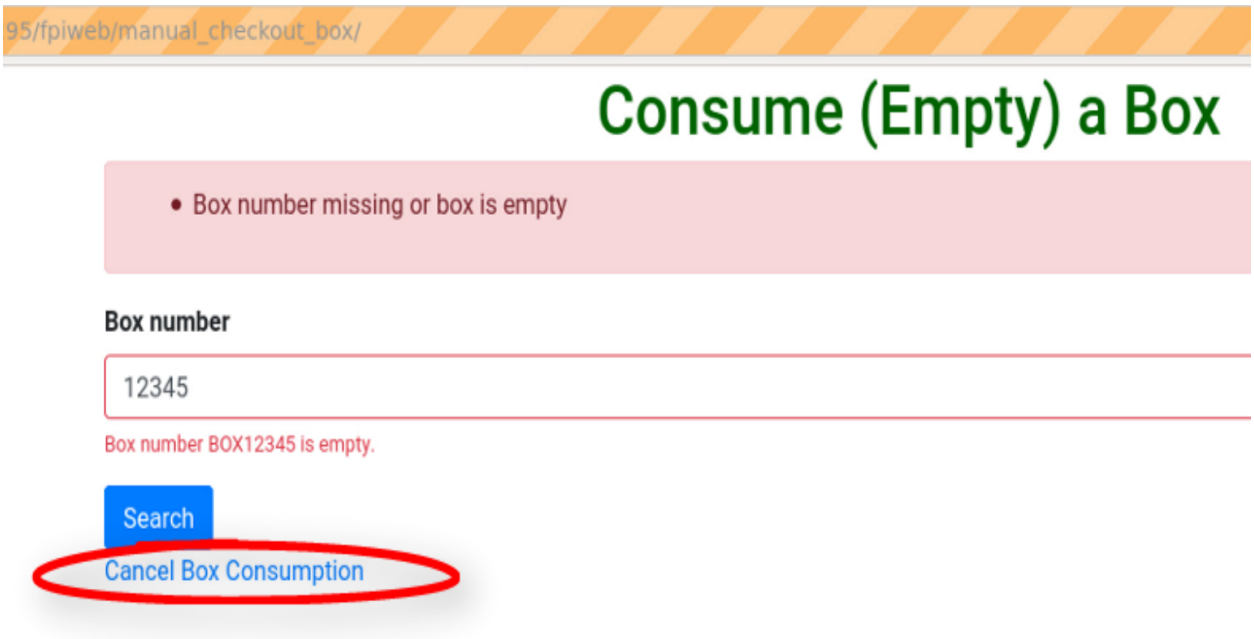
If everything went ok you should see the box information on the next screen. The box information includes the Box Number, the Box Type, the Box Contents, the year the Contents Expire and the location- Row, Bin, Tier. Click on the blue **Consume** button to continue if all the information seems correct.



The next screen should contain a message stating that a box 'has been succesfully consumed'. Click on the **Return to Main Menu** link to return to the **Main Menu** page.



In case of error you should see a screen similar to the one below listing the error. Click on the **Cancel Box Consumption** link to return to the **Main Menu Page** page.



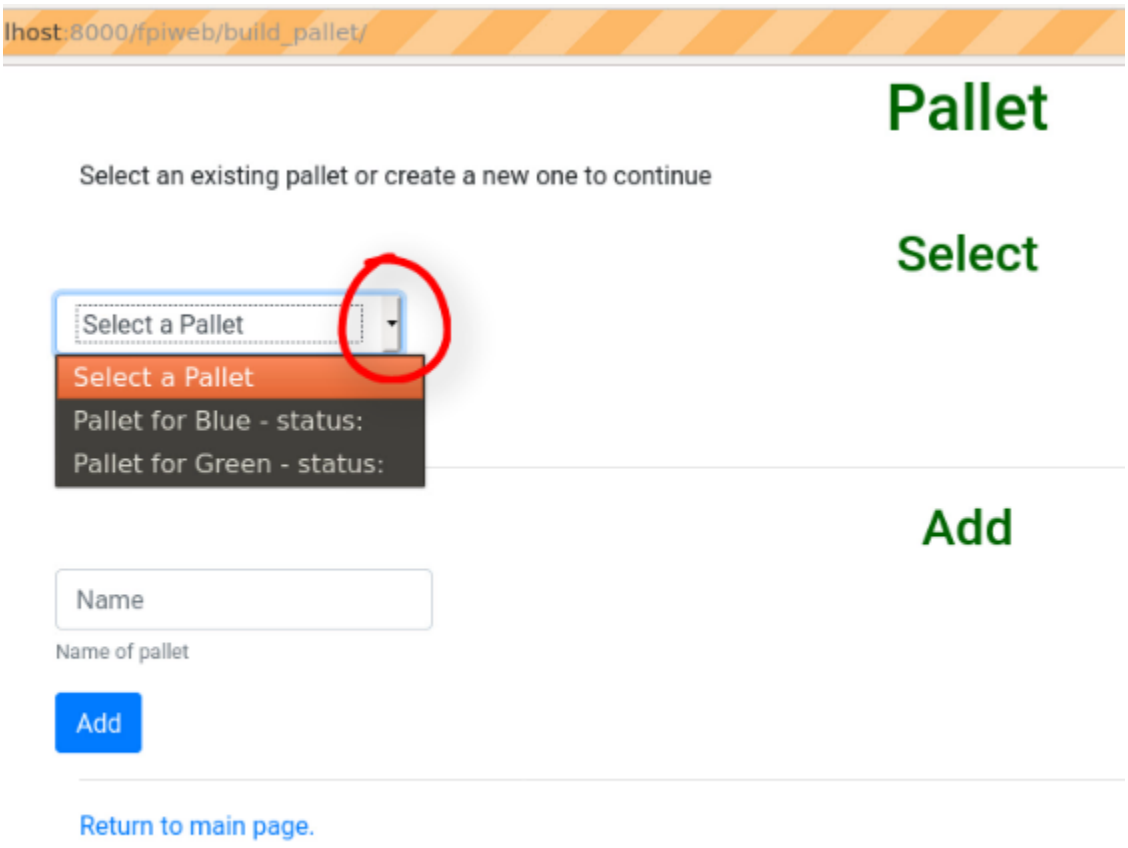
1.3.4 Build a Pallet

The next screen you will see is the **Build Pallet** screen. Here you will have two choices, **Select** or **Add**. **Select** gives you a choice of pallets to work with. If there are no pallets in **Select**, you must create a new pallet listing using **Add**.

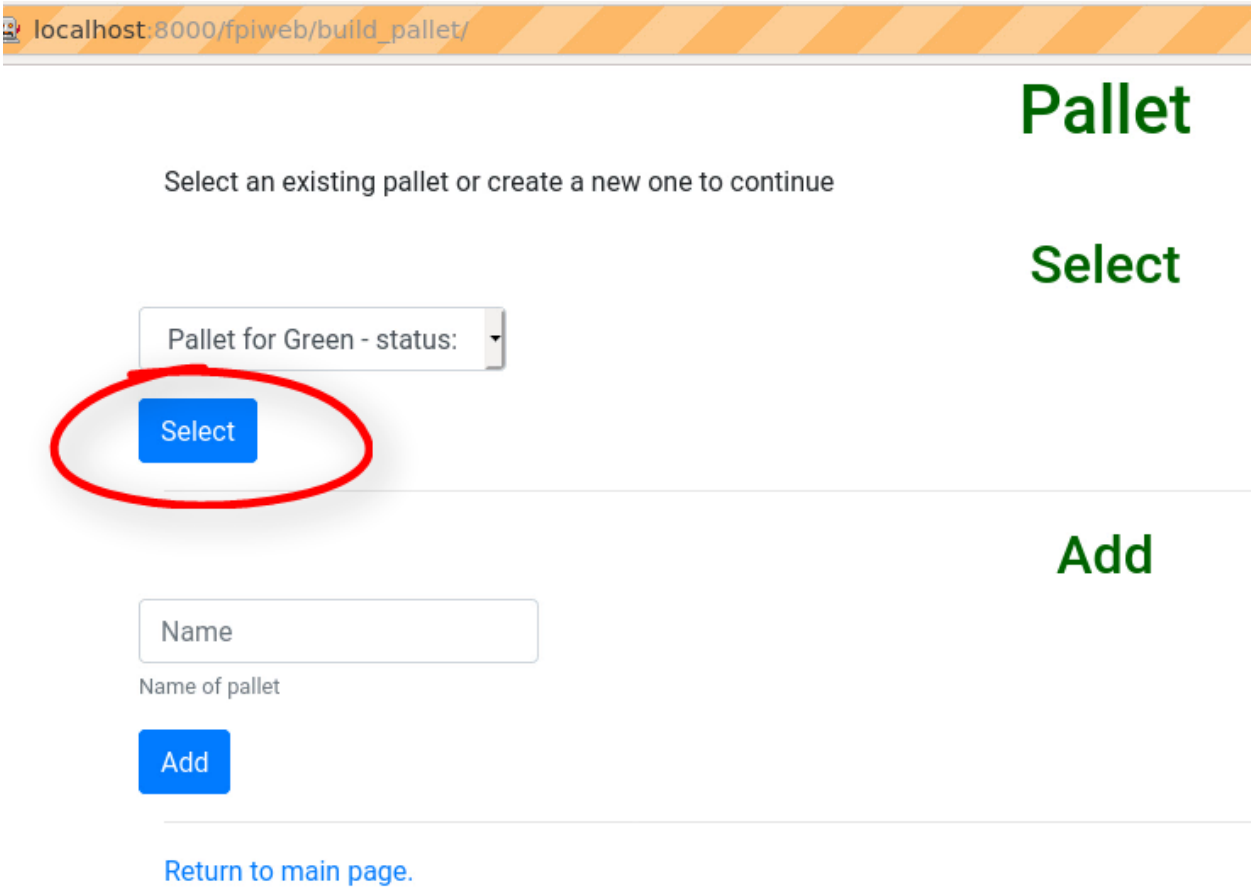
Select

To check if there are pallets that you can select click on the 'Dropdown icon' shown below. If there are pallets available you will see a drop down list as shown below. Select one of the pallets from the drop down list.

If there are no pallets available nothing will show on the drop down list. You will have to go the **Add** section.

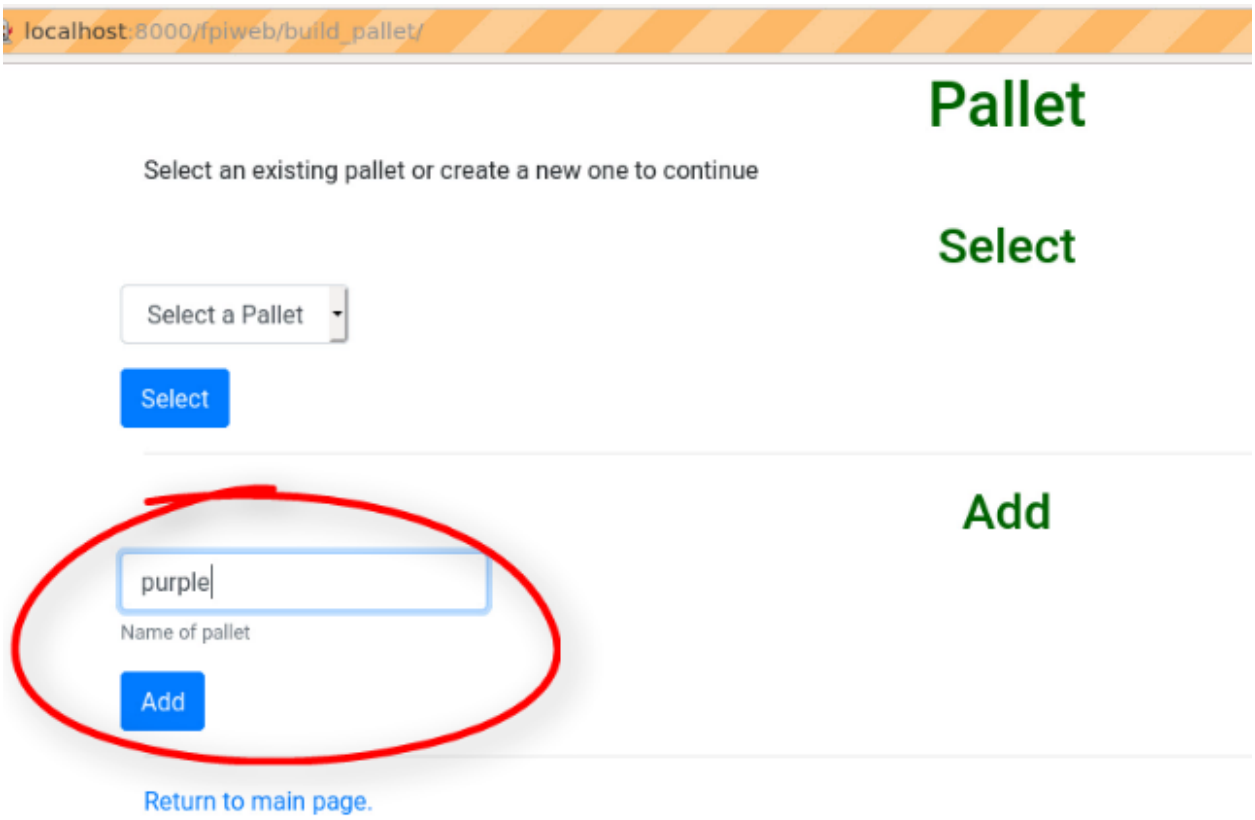


After selecting a pallet click on the **Select** button to go to the **Build Pallet** screen as shown below.



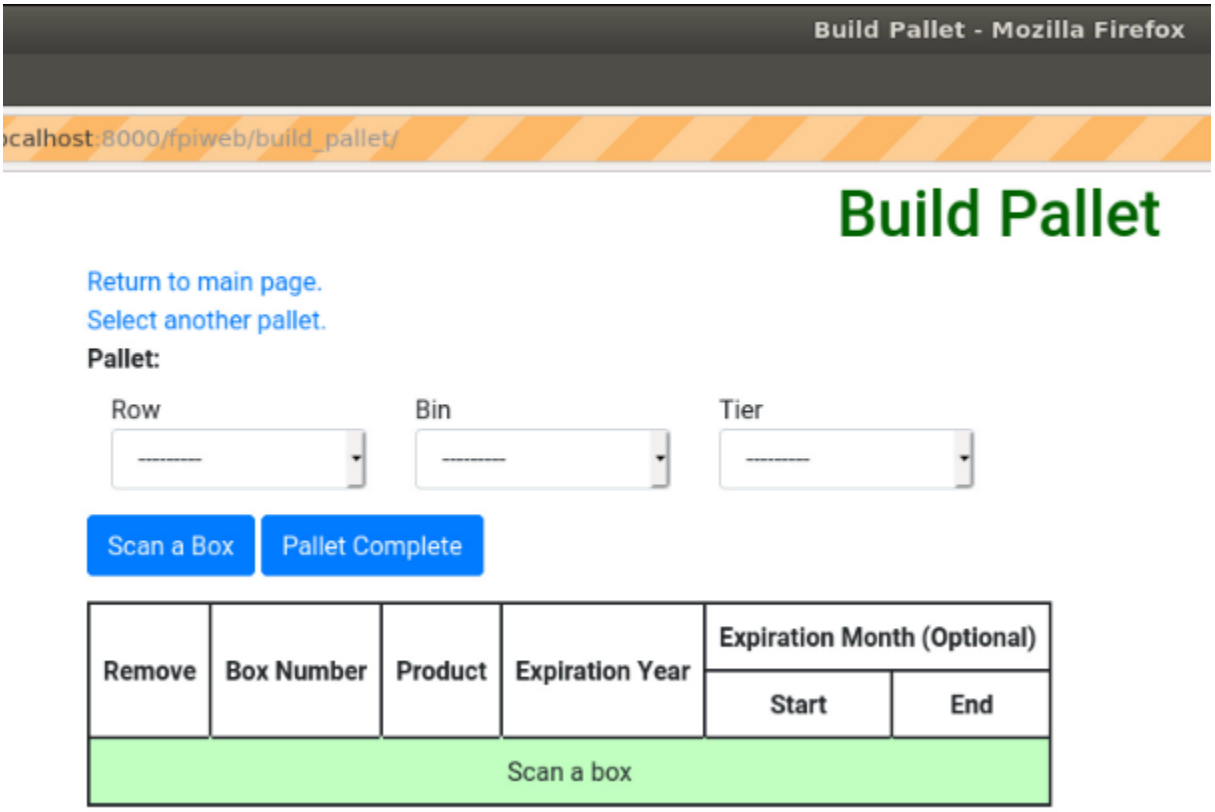
Add

Enter or key in a pallet name. In the screen below the pallet name 'purple' has been entered. You can pick any name you choose. After keying in the new pallet name click on **Add** for the **Build Pallet** screen.

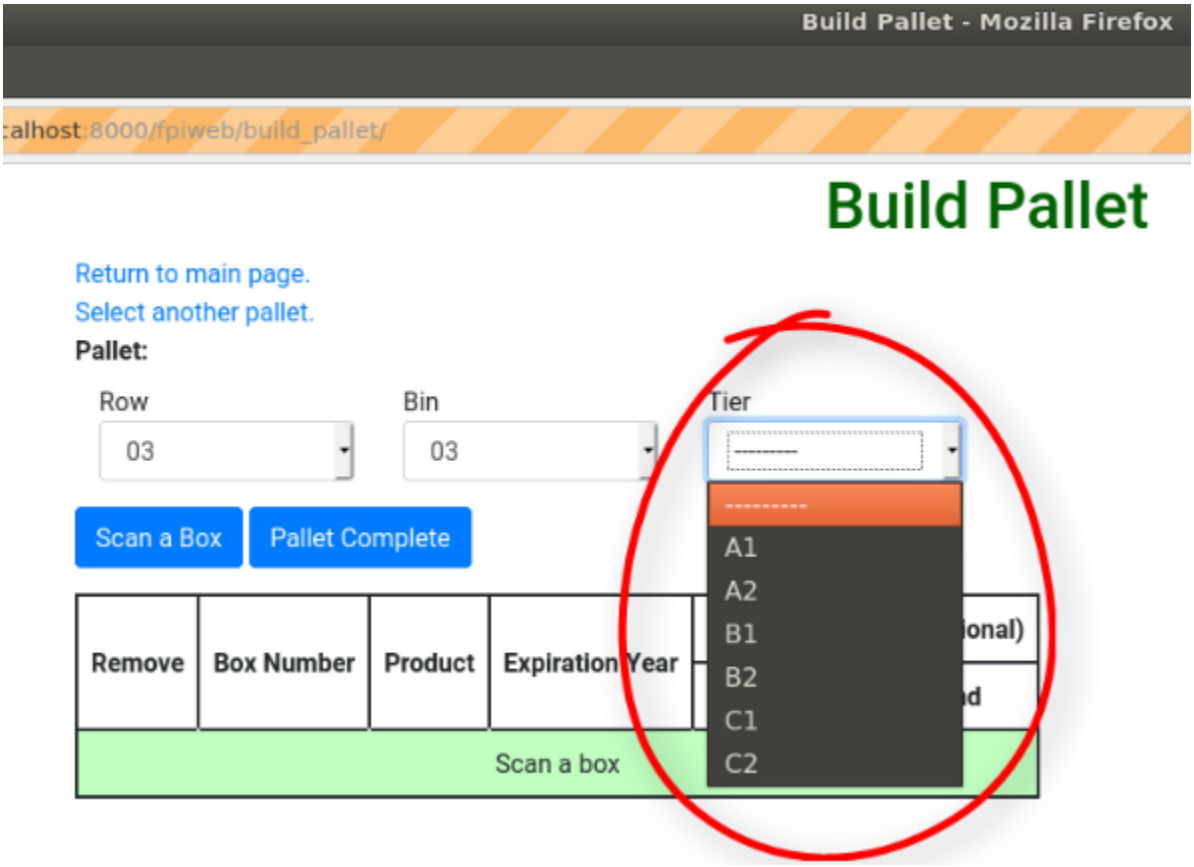


Build Pallet Screen

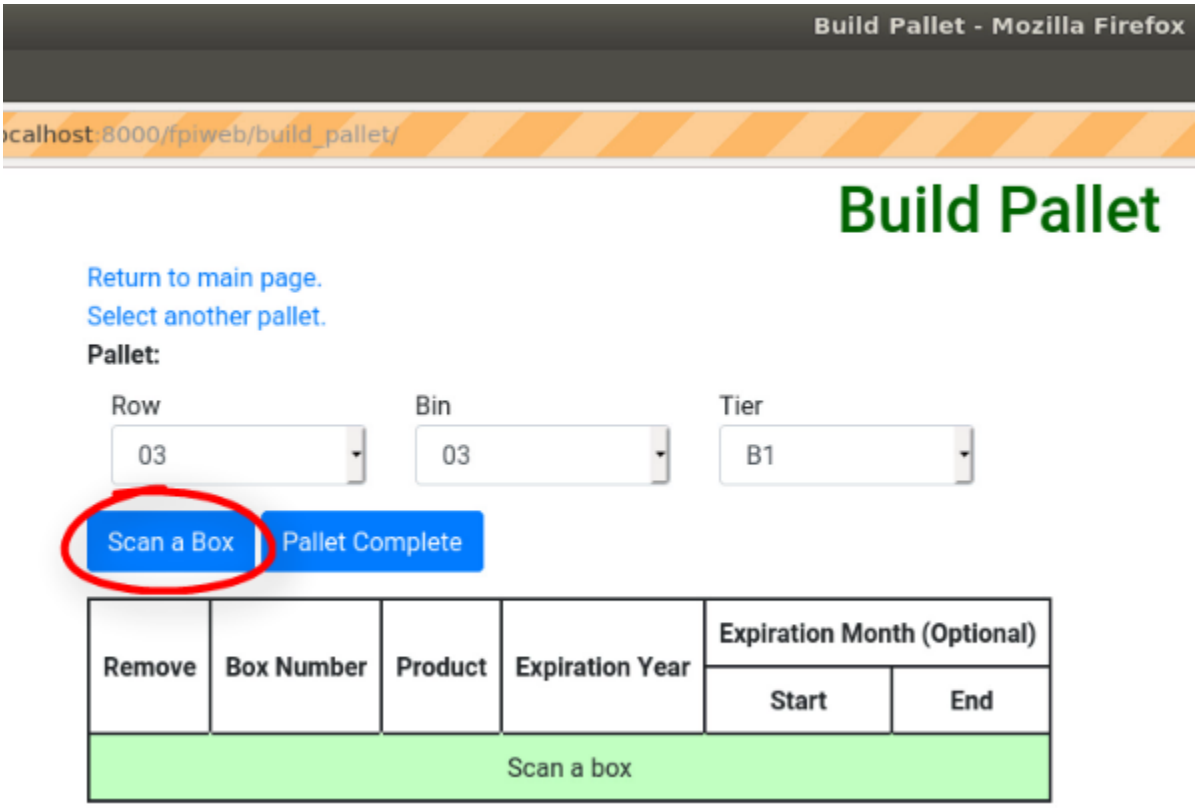
The next screen you will come to is the **Build Pallet** screen. On this screen the first thing you should do is to create a location for the pallet.



You will create the pallets location for 'Row', 'Bin' and 'Tier'. Row refers to which long row the pallet is on. Currently rows are 2 bins wide. Bin refers pallet bins located down the length of the row. Tier refers to the height level of the location. Use the drop down list boxes to enter 'Row', 'Bin' and 'Tier'.



Once you have entered the pallet location, click on the **Scan a Box** button in the middle of the **Build Pallet** page.



Scan a Box Popup

You will be directed to a **Scan a Box** popup window. If your computer has a camera you will be asked to enter a QR code. QR is short for Quick Response and is simply a funny looking label that can be read by computers.

Scan a Box QR code popup (for computers with a camera)

Use the camera on your computer to scan the QR code as shown below. Some computers may request your permission to use the computer camera. You will have to agree to the camera permission request to scan the QR code. In the image below an individual is holding a scan code up to the computer camera. With the QR code in the center of the 'Picture Window' click on the blue **Scan** button at the bottom right.

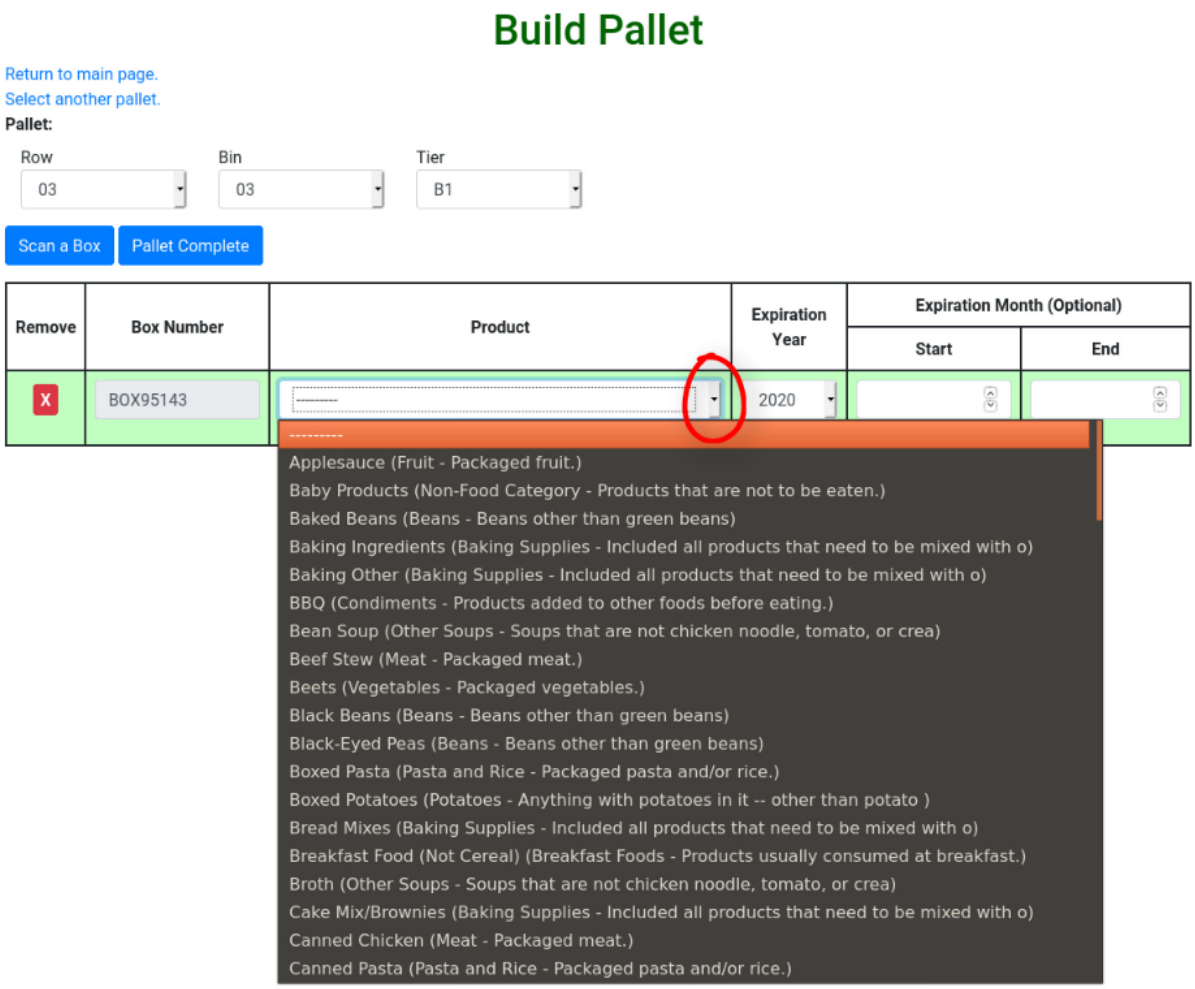
Occasionally there may be a situation where the computer is unable to scan the QR code, due to poor lighting or some other technical issue. In that case you can always add the QR code manually be entering or keying in the 5 digit box number manually in the pop up window at the bottom left. Then click the blue **Scan** button at the bottom right.

Scan a Box popup (for computers without a camera)

On this page you first have to enter or key in a 5 digit box number in the bottom left of the screen. You can enter any number but it must contain exactly 5 digits. Once you have entered the number click on the **Scan** button in the bottom right of the popup window.

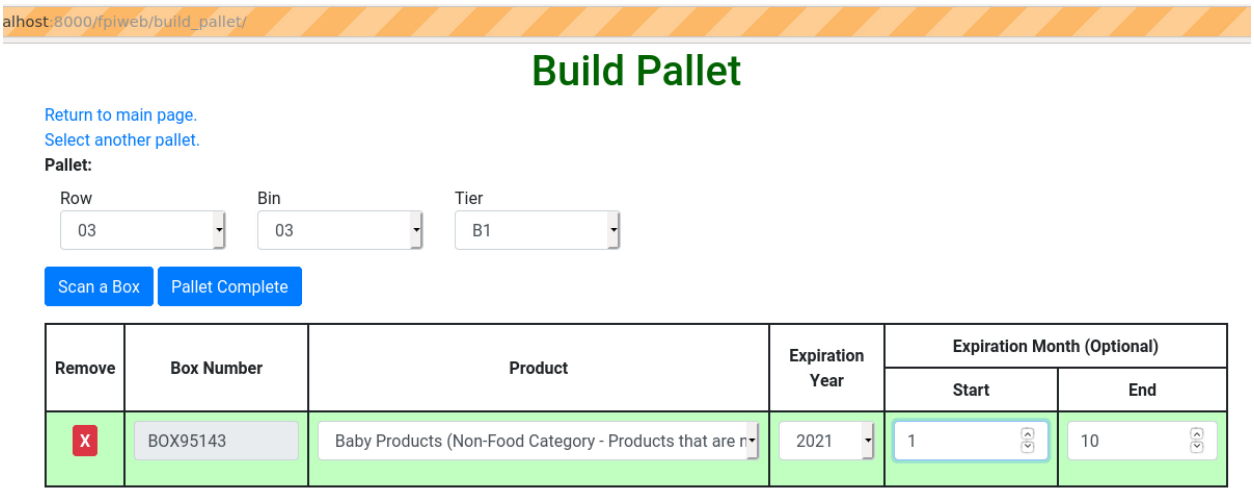
Return to Build Pallet

After clicking on the **Scan Button** you will be returned to the **Build Pallet** screen. You should see the 'Box Number' you have entered as well as a small white 'X' in a red box at the left of your screen. On this screen you will select a product to go in the box from a drop down product list. Click on the 'Dropdown icon' at the bottom right of the 'Product' rectangle'. Then choose an item from the list.

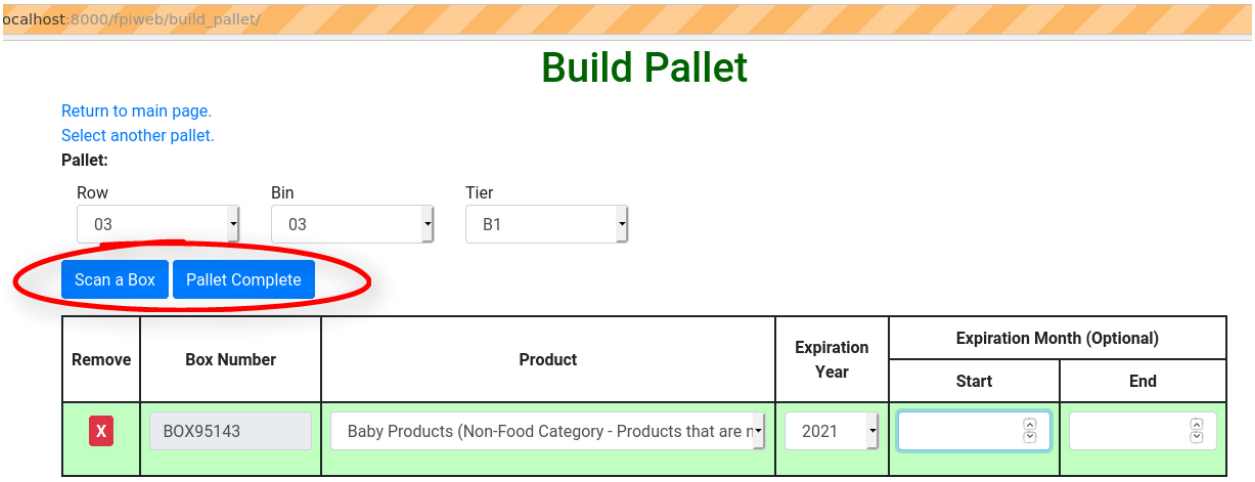


Once you have entered the product, use the 'Expiration Year' drop down list to enter the 'Expiration Year'.

Entering the 'Expiration Month' is optional. Months are entered from a drop down list that holds numbers from 1 to 12 that correspond to the months of the year- January to December. If you do decide to enter the 'Expiration Month' please make sure that the 'Start' month **is always less than** the 'End' month. This means the 'End' month **must not equal** the 'Start' month and **must be greater than** the 'Start' month. *Not all food items will have an 'Expiration Month' with a 'Start' and an 'End' month.* Once you have entered everything on this page the page should look similar to what is below.



From here if you click on **Scan a Box** you will be directed back to the **Scan a Box** popup window. There you can add another box in the same manner as you did before. If you click **Pallet Complete** you will be directed to the **Pallet Complete** screen.



Pallet Complete

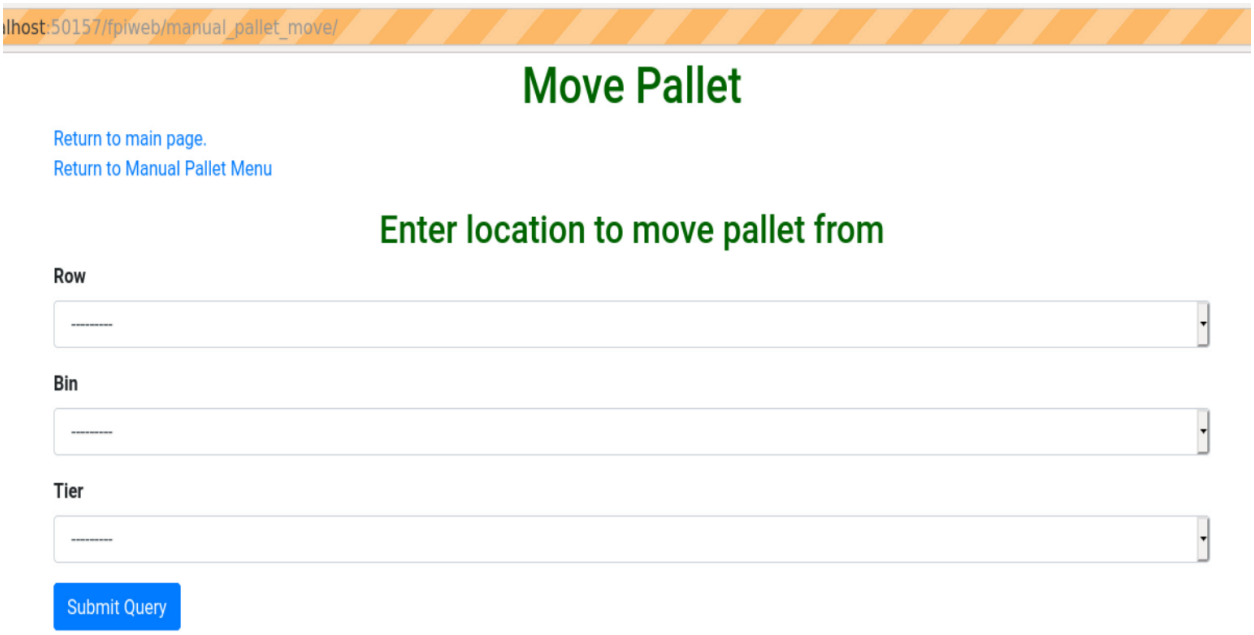
You should now see the **Pallet Complete** screen.

From here you can return to the main screen by clicking 'Return to main page'.

1.3.5 Move a Pallet

Currently the Move Pallet screen has an error or bug in it. There is a link on the Move Pallet screen which states 'Return to Manual Pallet Menu'. This link should NOT be clicked and should be ignored. If you do not see this link then the bug has been fixed but the User Documentation has not been updated.

After clicking on **Move a Pallet** you should see a screen like the one below. This screen allows you to move the location of each pallet along with its boxes in the database records. Basically if you move a pallet you are also moving the boxes the pallet contains.



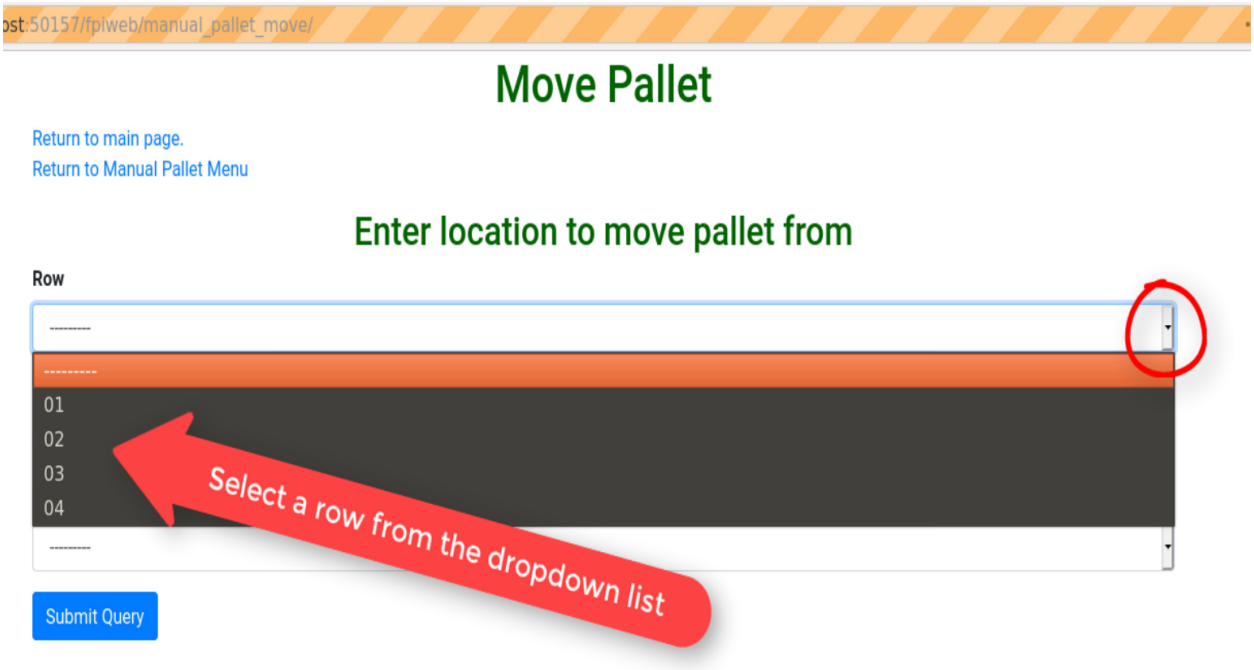
When moving a pallet (with its boxes) three different conditions can occur.

1. Move a pallet and its boxes to an empty pallet location.
2. Move a pallet and its boxes to non-empty pallet location.
3. Attempt to move an empty pallet with no boxes.

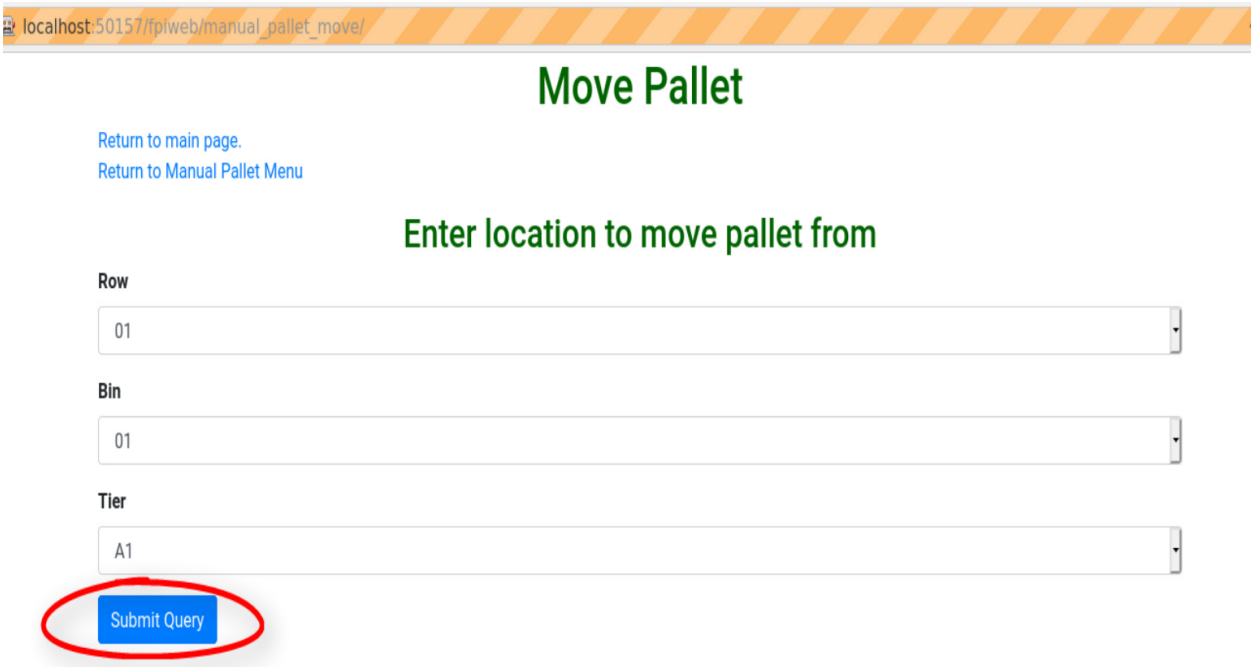
All three conditions will be shown below.

Move a Pallet to an Empty Pallet Location

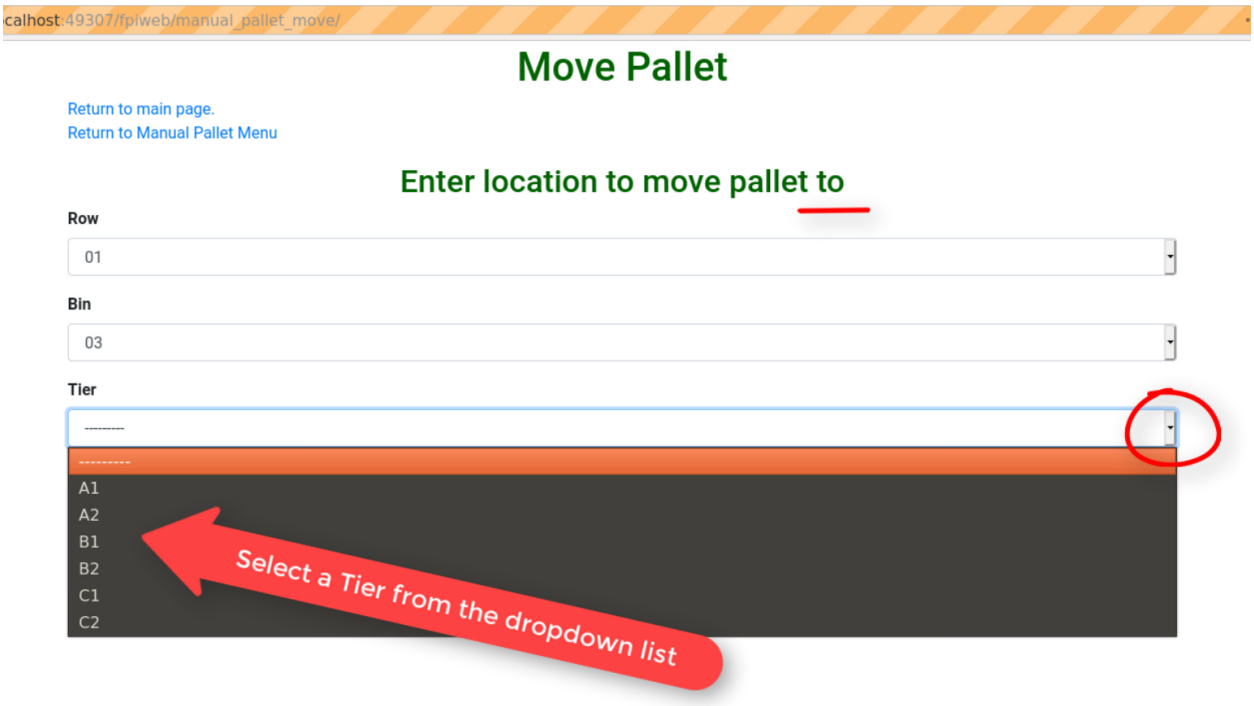
When moving a pallet and its boxes to a new location the first thing you need to do is to "Enter location to move pallet from". You enter the position of the pallet by selecting the Row, Bin, and Tier dropdown lists by using the mouse to click on the down arrow at the right of each dropdown list. Selecting the down arrow as shown below in the Row dropdown list brings a set of choices. Simply click on the current Row location choice to "Enter the location to move pallet from".



Do the same with the Row and Tier locations. Once that is done you should see the **Move Pallet** screen as filled out below with Row, Bin, and Tier locations filled out. Then click on the blue **Submit Query** button.



This will bring you to a similar screen but this new screen will say "Enter location to move pallet **to**". Enter the Row, Bin, and Tier location for where the pallet and it's boxes will be moved to. The screen below shows the new Tier location being chosen.

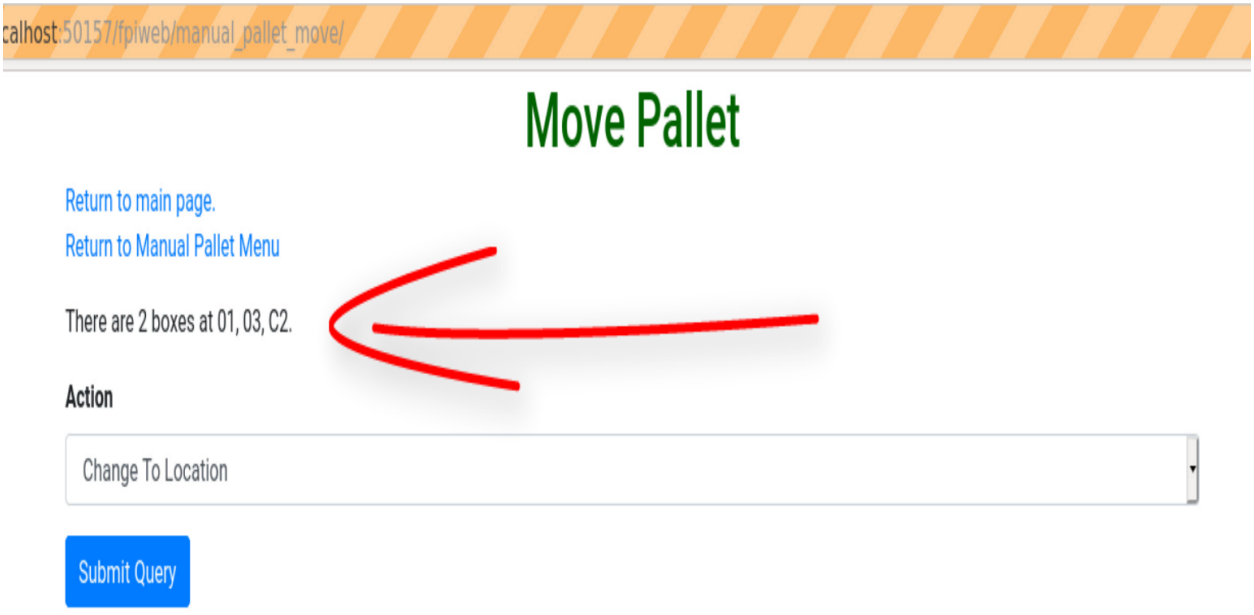


Once the **Enter location to move pallet to** screen has been filled out click on the blue **Submit Query** button. If the new location you want to move the pallet is empty and has no boxes you should see a screen similar to the one shown immediately below.

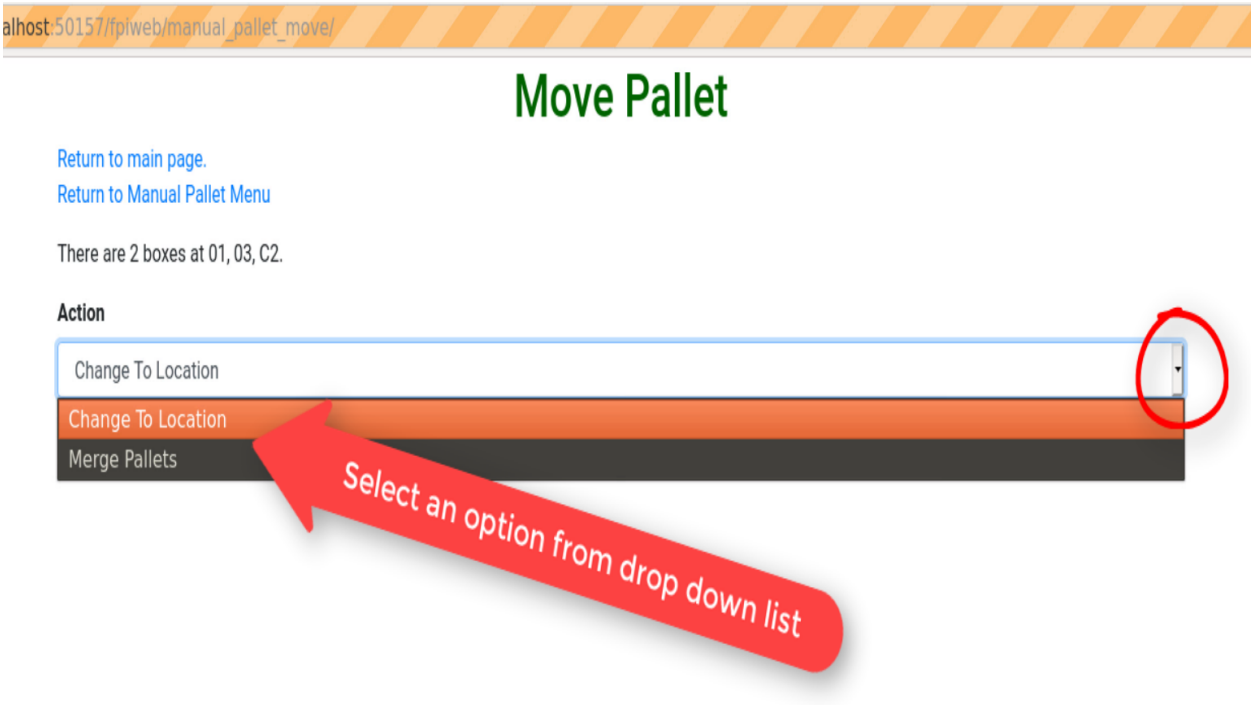
If the "Enter location to move pallet to" is NOT EMPTY then you will see a screen like the one shown in the next section below.

Move a Pallet and Its Boxes to a Non-Empty Pallet Location

The screen below shows up when you try to move a pallet to a location where a pallet is already located. Notice that the message states "There are 2 boxes at 01,03,C2".



This means you will have to make a decision, either (1) choose a new location by clicking the **Change To Location** choice or (2) merge the pallets by clicking the **Merge Pallets** choice.

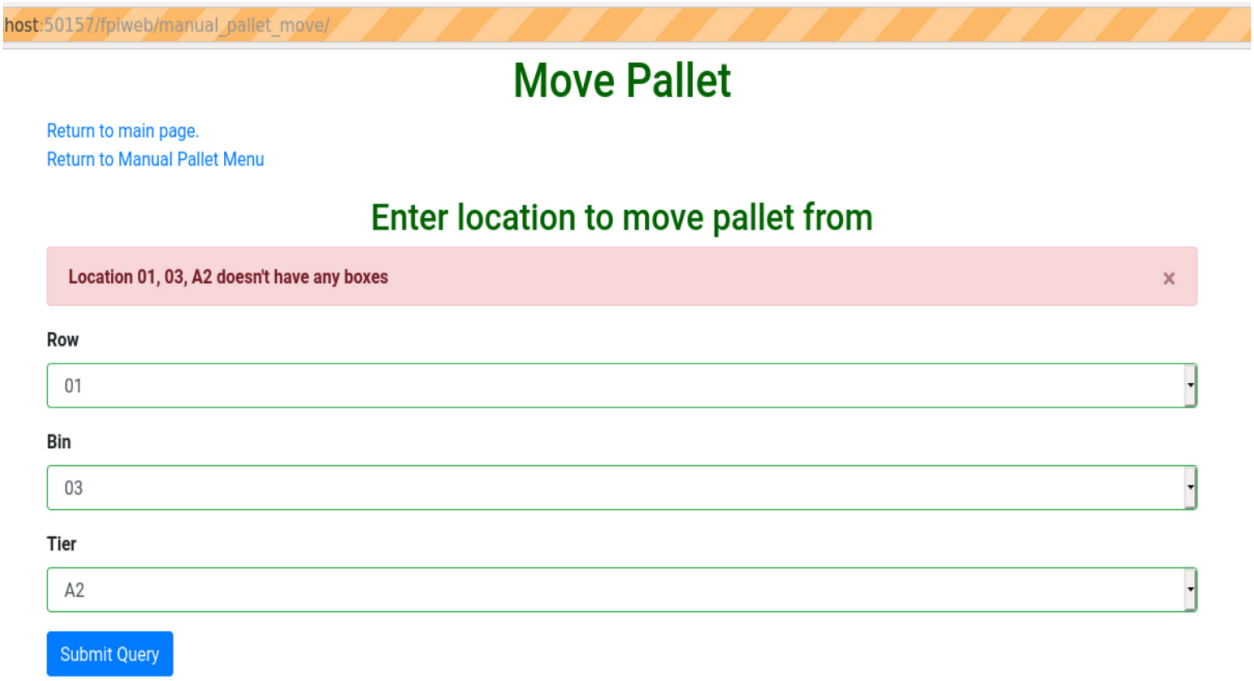


If you click on **Change to Location** you will be directed back to the **Enter location to move pallet to** screen as shown above. From there you can select another location to move the pallet to.

If you click on **Merge Pallets** both pallets along with their boxes will be merged into the new location you picked from the **Enter location to move pallet to** screen. You will see a screen stating "boxes move to row, bin, tier".

Attempt to Move an Empty Pallet

If you attempt "Enter location to move pallet from" and there are no boxes recorded in the database for that pallet location you will see a screen similar to the one shown below.

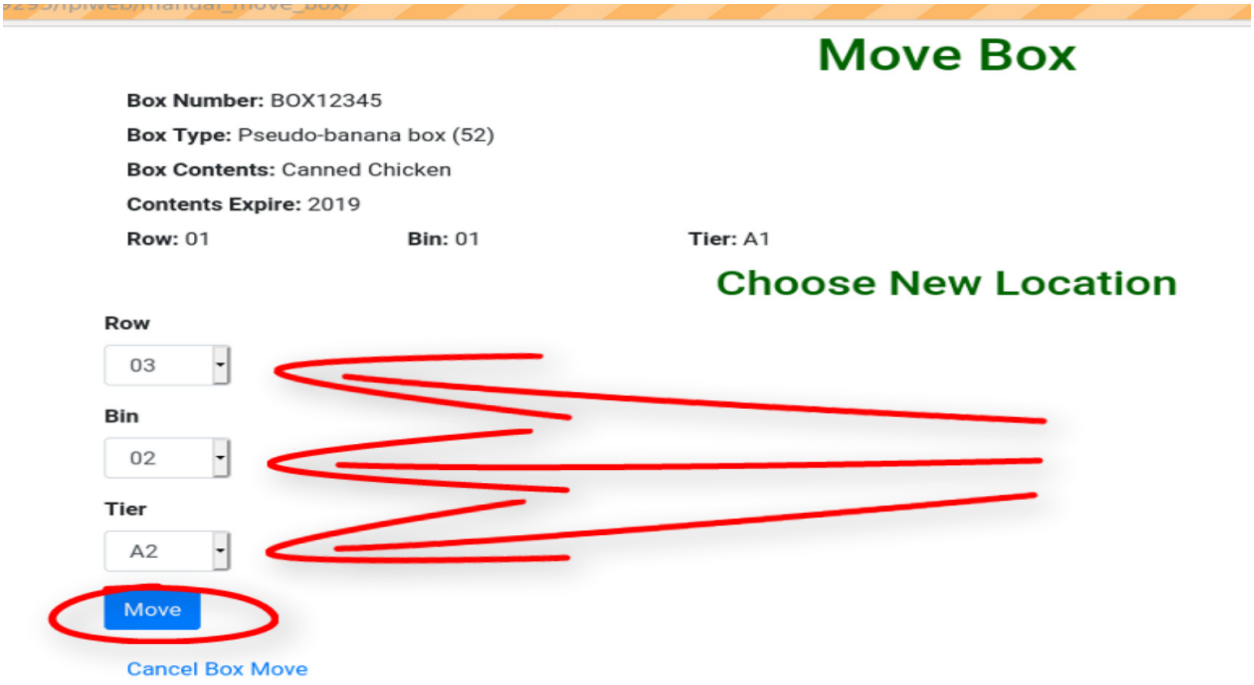


1.3.6 Move a Box

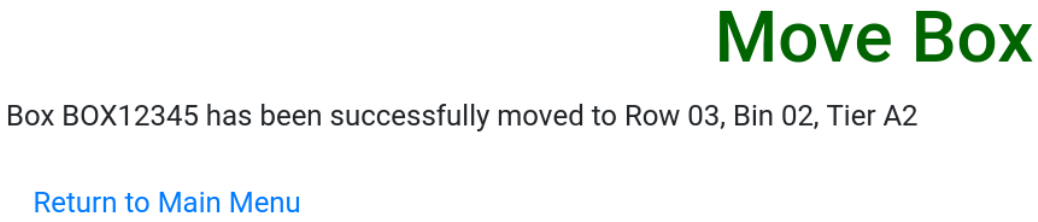
To move a box to a different pallet enter the 5 digit box number in the **Box Number** field and then click on the blue **Search** button.

The next **Move Box** screen contains all the box information. In this screen you can choose a new location by entering new **Row**, **Bin**, **Tier** locations. Click on the small triangle to the right of each **Row**, **Bin**, **Tier** to get your drop down

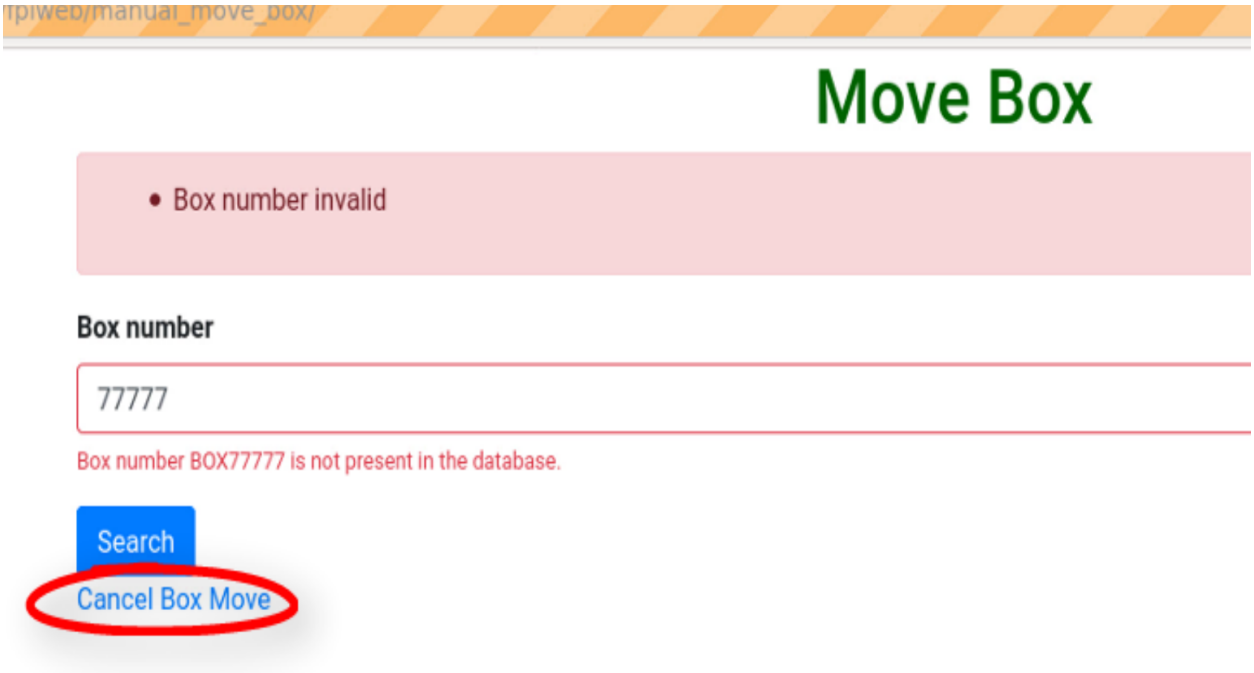
list choices. After choosing your new location entries click on the blue **Move** button to change the information in the database.



If there are no errors you should see a screen similar to the one below. Click on the **Return to Main Menu** link to return to the **Main Menu Page** screen.



In case of error you will see a screen similar to the one below with an error message. Click on the **Cancel Box Move** link to return to the **Main Menu** screen.



1.3.7 Add a New Box to Inventory

With this screen you can add a new box following the 4 steps listed below.

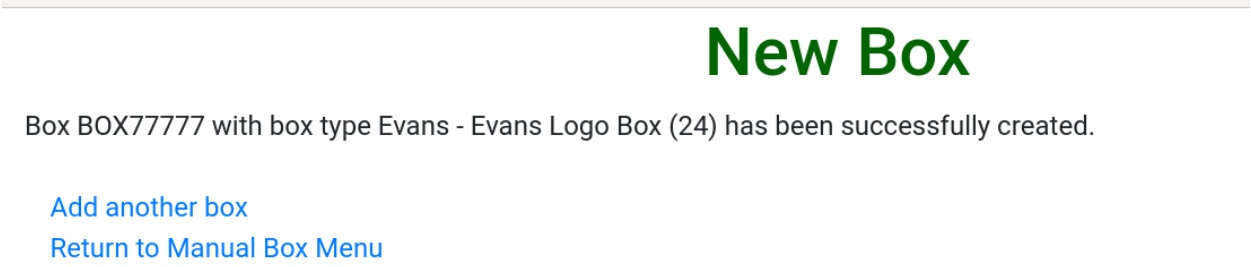
(1) Enter a 5 digit box number.

(2) Click on the drop down list triangle at the far right.

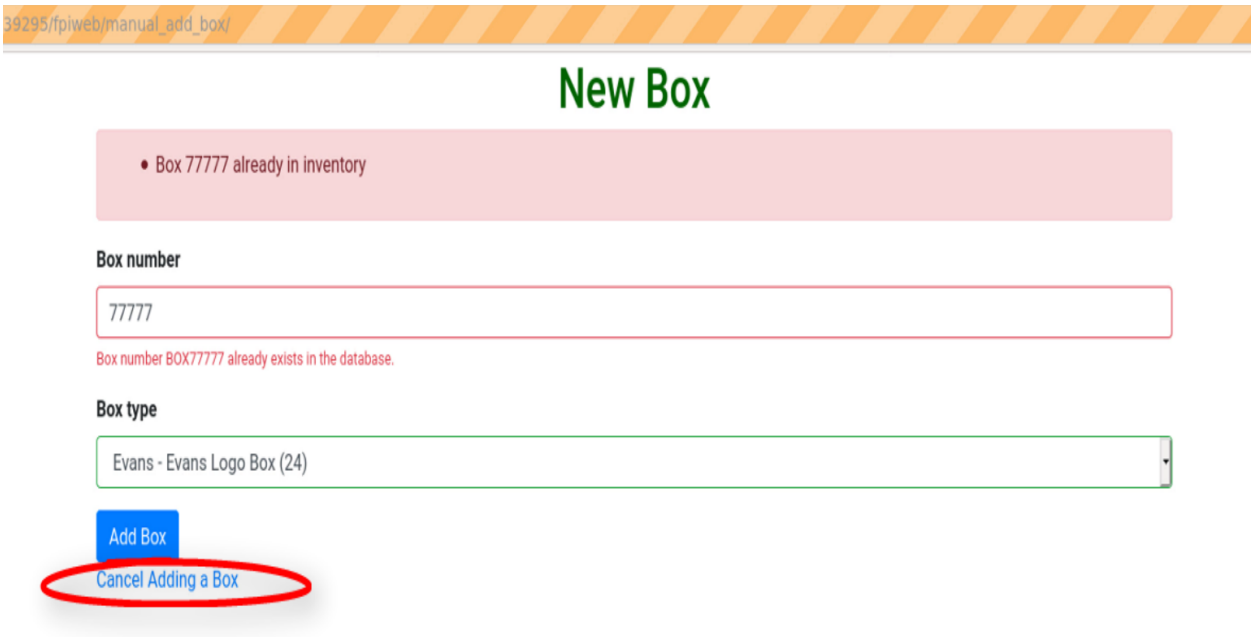
(3) Choose an item from the drop down list.

(4) Finally click on the **Add Box** button (hidden from view in the below graphic).

After clicking on **Add Box** you should see the following screen. From there you can **Add another box** or **Return to Main Menu** by clicking on the links in the **New Box** screen.



If there was an error you will see the following screen. Simply click on **Cancel Adding a Box** to return to the **Main Menu** screen.



1.4 Other Tools

Other tools will be documented here.

CHAPTER

TWO

SPECIFICATIONS

This application will track the product managed by a food pantry.

2.1 Goals

The key objectives of this system are:

1. Track all product (food, personal care, etc.) destined to be given to clients.
    - Record product coming into the warehouse supporting the pantry - what it is, where it is located, quantity, and expiration date.
    - Record any movement of product in the warehouse so staff know where the product is at all times.
    - Record when product is consumed or used up.
2. Provide staff with information about the product in and flowing through the system. Key points:
    - How much of each product is in the warehouse?
    - Where is product that is about to expire located?
    - How much of each product has been consumed over various periods of time?
    - Is there insufficient product to meet expected demand? If so, how much is needed?
    - Is there too much of a given product? If so, how much will expire before it can be used?
3. Provide a means of tracking the product that is easy to use – expecially by inexperienced people.
    - Most of the people handling the product are volunteers who are helping for only an hour or two at a time.
    - Ideally parts of this system can accommodate teenagers or people with disabilities.

2.2 Initial Approach

This application will initially take this approach to track the product.

2.2.1 Product Management

- Product will be stored in boxes that are uniformly sized (with some exceptions).
- The boxes will have three labels on the end.
  - The first label will be a printed QR code that has a URL that includes a unique box number. The URL will be such that scanning it will access the system to bring up a web page. That web page will allow the user to check the box in, move the box to a new location, or check the box out of the system.
  - The second label will identify the category of procduct contained in the box. The granularity of the category will be determined by the distribution manager.
  - The third label will identify the expiration date of the product contained in the box. At least the year will always be identified. At the discretion of the distribution manager, additional labels may be attached that identify the half-year, quarter, or month of expiration.
- The location of a box will be in a fixed place in the warehouse. The distribution manager will designate the available location categories. Currently, locations are identified by:
  - Row (01 - 04)
  - Bin (01 - 09)
  - Tier
    - Ground level - A1, A2
    - First level (above ground level) - B1, B2
    - Second level (above first level) - C1, C2
  - Boxes locations will have all three identifiers. For example: row 1, bin 4, tier A2. The short form is 01-04-A2.
- Boxes will be stored (presumably) full of product at one of the locations designated above. If a box is moved, an entry in the system is required so it will know where the box is.
- If a box is consumed (taken out of inventory in preparation for giving the contents to clients), the system will create a record with the following information:
  - Box number/id
  - Product category name
  - Date the box was filled
  - Expiration information
  - Date box was emptied
  - Location of the box when it was emptied
  - Number of days the box was in inventory
- The URL in the QR label will be such that scanning it will access the system to bring up a web page. That web page will allow the user to check the box in, move the box to a new location, or check the box out of the system.
- Although the boxes are usually collected and stored on a pallet, the system will track by box.

2.3 Web Site

Django will be used to present a web site that accepts the URL encoded in a QR code.

The web site brought up by the QR code will have the following functions:

- Identify and validate the user.
- Checkin Product
  - Checkin will allow the user to designate that the box of product is being checked in, the contents (product), location to be stored, and expiration date given.
- Checkout Product
  - Checkout will allow the box of product to be consumed (checked out) of inventory.
- Move Box of Product to a new location.
- Logout the user.

2.4 Backend

A separate directory in the project will hold the backend code.

2.4.1 API's

The backend will present the following APIs for the use of the web site:

Initial Scan

This API will accept the box number from the initial QR oode scan. It will query the database to determine:

- The box exists and is empty. If so, send back information so the contents, etc. of the box can be recorded. If the box number does not exist, create a record for it and send back the same information.
- The box exists and it is full. Send back the contents of the box and other information so the box can be moved or consumed.
- If the format of the box number is misconfigured, send back an error.

2.5 Development Configuration

2.5.1 Software

Developers will have the following software installed and configured on their system:

- PyCharm (Professional or Community)
- Python 3.6 or later
- PostgreSQL - Server 11.0 or later - pgAdmin 4.0 or later

2.5.2 Project

Each developer will have a fork of the main repository on GitHub. A local clone of their fork will be used on the developer's system to manage changes to the system.

Changes will be implemented by:

1. Branching the local repository
2. Making changes in the branch
3. Testing the changes in the branch
4. Merging the changes in the branch into the local master
5. Retest/regression test
6. Push the changes into the developer's fork on GitHub
7. Submit a pull request into the main repository

See the *Developer Documentation* for more information about how to be a contributor to this project and how to get started.

2.6 Tasks

Please refer to the document *Outstanding Tasks* for specific items that need to be accomplished.

CHAPTER

THREE

DEVELOPER DOCUMENTATION

3.1 System Design

3.1.1 User Management

Summary

Users allowed access to the system are divided into three groups: Volunteer, Staff, and Administrator. The three

Group Identifier(1)	Access
Volunteer	*Volunteer User Access*
Staff	*Staff User Access*
Admin	**`Administrator User Access`_**

1. This is the value in the auth_group table.

Volunteer User Access

Volunteers have the most limited access. They are allowed the following functions:

- Login to the system
- Add boxes
- Checkin/checkout boxes and pallets
- Move boxes and pallets
- Check box status
- change their own password
- change their name, email address, title (but not their userid)
- view product examples

Staff User Access

Staff access can do the following:

- Can manage all user data:
  - boxes
  - durable data (e.g. product, box type, location, etc.)
  - constraints
- Can print QR labels
- Can dump activity data
- Can add user information:
  - Add a new userid, name, title, permisison level (staff or volunteer) and initial password
- Can change user information:
  - permission level (volunteer -> staff, staff -> volunteer).
  - Can force user to have a different (temporary) password.
    - The user must use the temporary password the first time and change it to something else immediately.
  - Can block a staff or volunteer user's access (inactivate a user).
- Can use the Django admin web site – except for accessing the user and group tables.
- Everything that a volunteer can do

Admin User Access

Admin access has full access to all menu picks - including all Django admin pages.

An administrator can:

- Add or block another administrator
- Add or block any other user
- Everything a staff member can do

User Management Screens

This section documents each user screen and provides details not given elsewhere.

Login Screen

This is a simple login screen that requests a user id and password. It will be accessible for all users. This screen will also be the first screen presented to the user when the main (default) URL is given in the user's browser.

- The userid and password are kept in the default Django table (django_user).
- This userid must have several associated records.
  - A fpiweb_profile record to hold application-specific data.
  - At least one auth_user_groups record to identify what level of permissions are permitted for this userid.

- If the fpiweb_profile record indicates that the user must change their password, this screen will provide a place to do so.
- A user must be an "active" user.

Users who fail to provide a valid login and pasword, who fail to provide a valid new password when requested to do so, or whose credentials fails to meet the above criteria will be notified with a message "Invalid credentials, please try again". No more detail will be provided on the login screen. After three tries, for a userid that is valid, the userid will be set to inactive and must be changed by another qualified user before this userid can be used again.

Additional options:

- Include a check box to allow the user to change their password as desired (after being validated).
- Forbid changing passwords more often than once a day (future).
- Do not require a user to change thier password (except if forced by a staff or administrator.

Passwords

Passwords must meet the following criteria:

- Must be at least eight (8) characters long
- If less than twelve (12) characters long, must contain a mix of at least three of these classes:
  - upper case letters
  - lower case letters
  - numbers
  - special characters
- Can be up to at least 128 characters long.
  - The entire password will be hashed, not just the first eight characters

There may be other constraints added later as needed for security protection.

Logout Screen

This screen is accessible by all users. When chosen, it will invalidate any cookies, csrf token, etc. so that the user must login again before resuming operations.

This screen will have a link to the login screen.

System User Maintenance Screen

Only staff or administrators are permitted to use this screen. Screen will:

- Show (scrolling) list of userids and associated info:
  - fields shown in scrolling list - userid - first and last name - permission level - active flag
  - Administrator level permissions will see all users
  - Staff level permissons will see only staff and volunteers
- Selecting a user will show all the current details about that user.
- Button available to add a new user.

Fields Displayed (see *Legend* and *Table Notes* for additional explanation)

Field	New User	Change User
userid	CM(1)	U
first name	CM	C
last name	CM	C
title	C(2)	C(2)
email address	CN	CN
permission level	SM(4)	S(4)
active	BM	CB
password	CME(3)	CE(3)

Legend

Indicator	Meaning
M	Mandatory
C	Changeable
U	Unchangeable
N	Not required (blank allowed)
E	Encrypted
S	Selection of: Administrator(4), Staff, or Voluneer
B	Boolean: Yes or No selection

Table Notes

1. Userid must be unique and are encouraged to be at least 6 characters long.
2. Title will default to “Volunteer”, “Staff”, or “Administrator” depending on the permission level given.
3. If set or changed, the password will never be visible at any time, and must be entered twice.
4. Only administrators can set the permission level of a user to administrator.

3.1.2 System Design Summary

The system is designed to be used by folks with various levels of experience using computers, tablets, and smartphones.

3.2 UML Diagrams

3.2.1 Overview

An overview of this inventory system is illustrated by the following UML diagrams.

For more information about UML diagrams, please see any of these links.

- Wikipedia article
- Unofficial Explanation
- Official UML Site

3.2.2 Scan QR Code

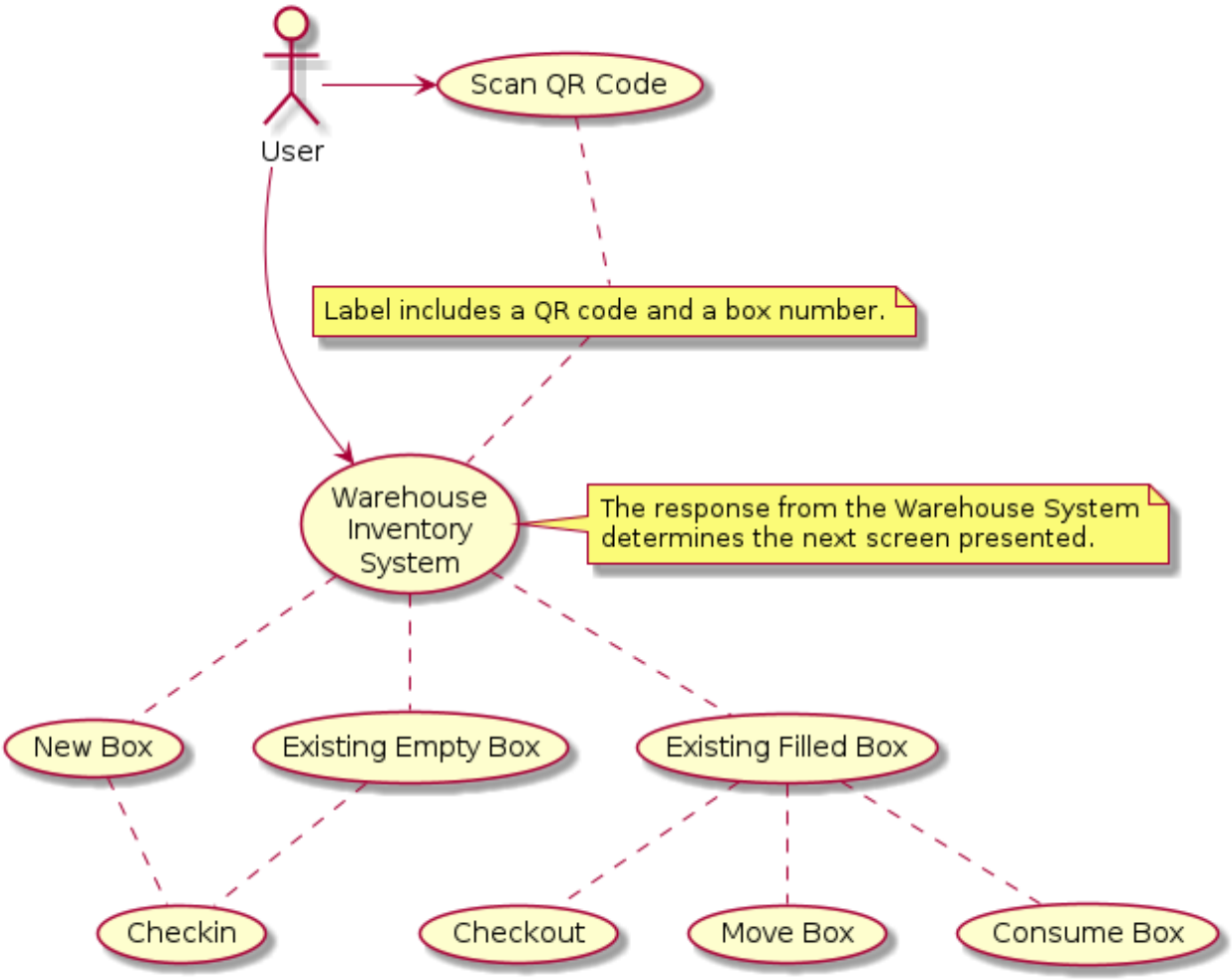


Fig. 1: **Scan QR Code Use Case**

3.2.3 Checkin

3.2.4 Checkout

3.2.5 Move Box

3.2.6 Fill a Pallet

3.2.7 Move a Pallet

3.2.8 TBD

More diagrams will be needed in the future.

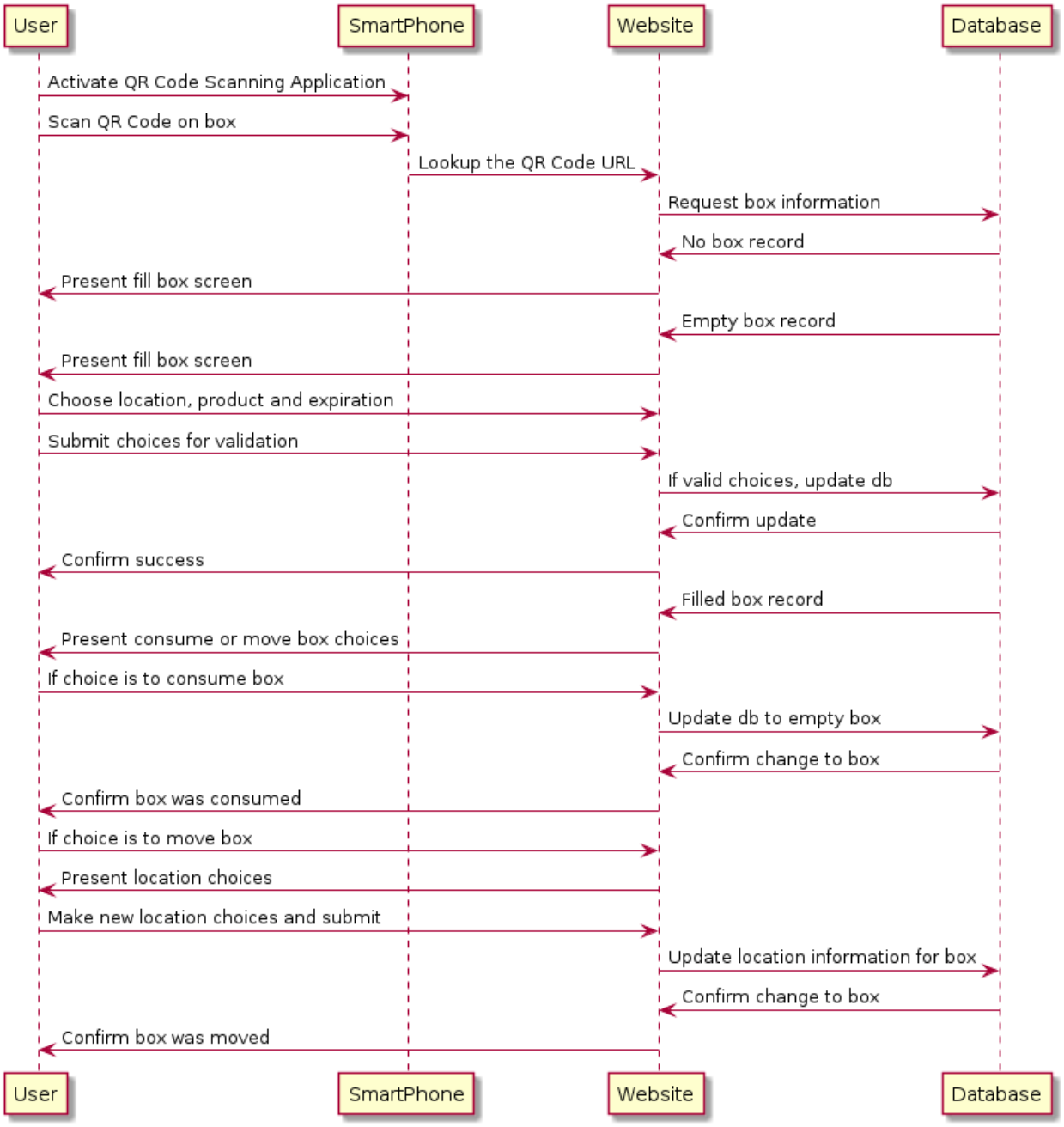


Fig. 2: **Scan QR Code Sequence Diagram**

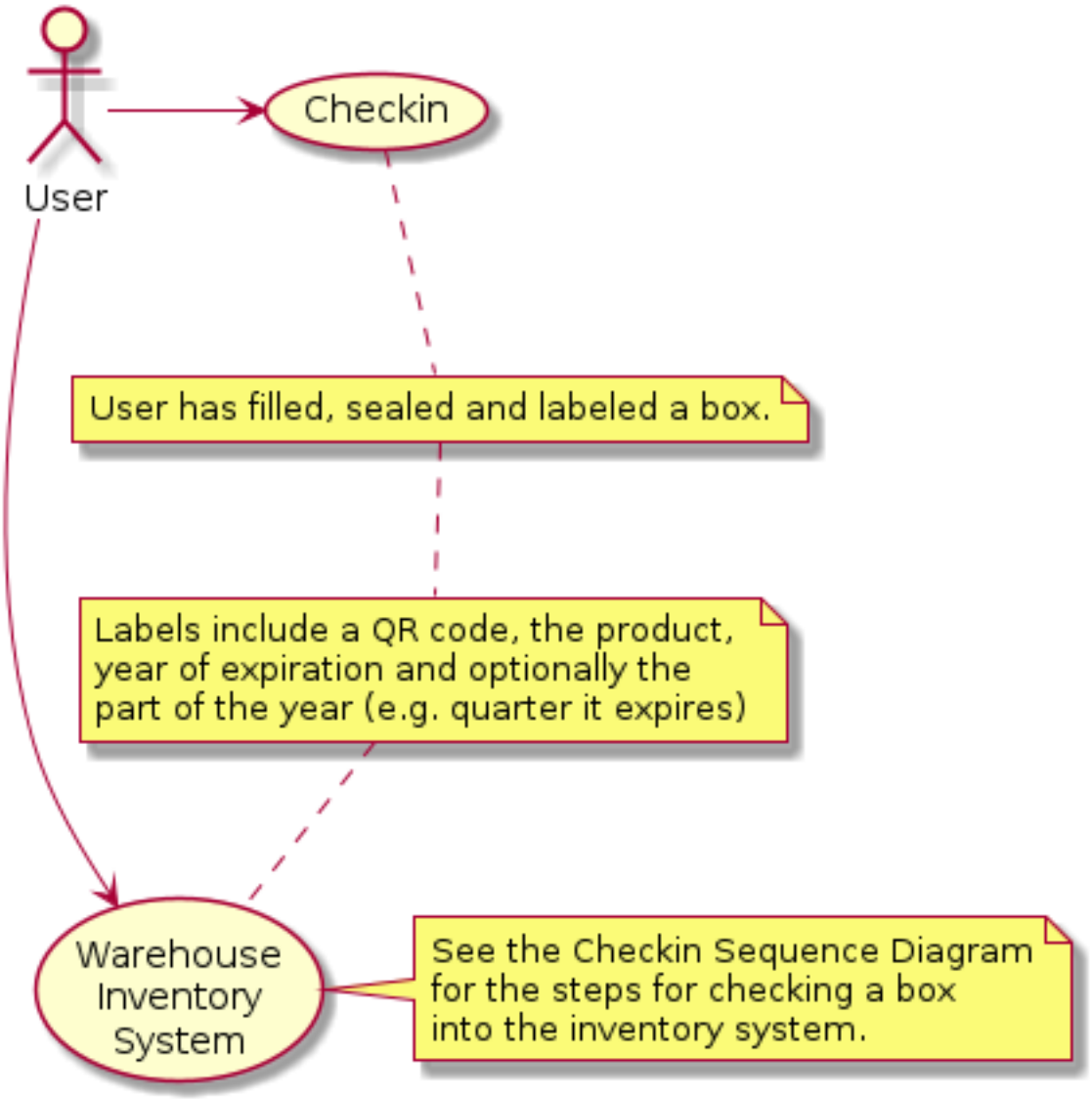


Fig. 3: **Checkin Inventory Use Case**

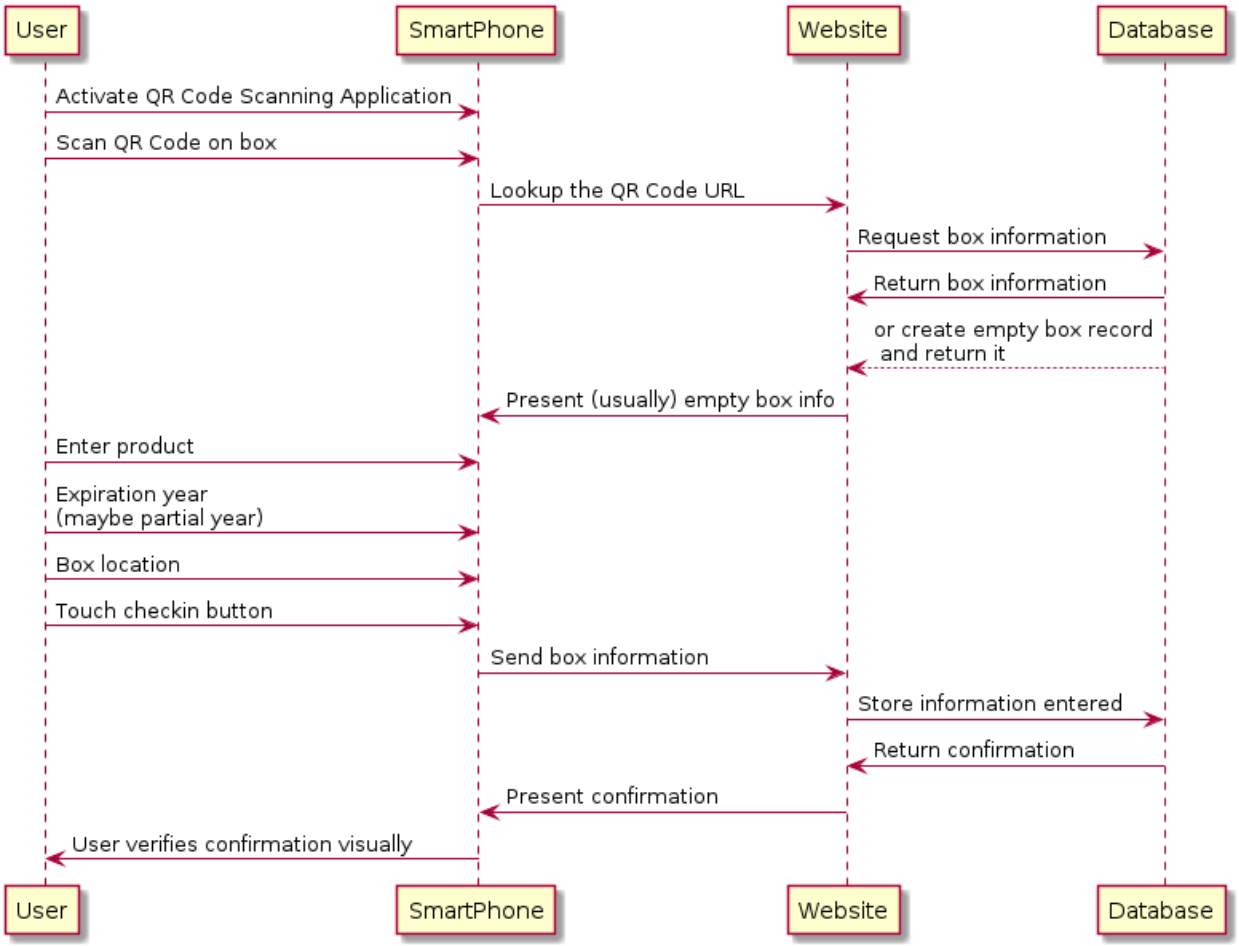


Fig. 4: **Checkin Inventory Sequence Diagram**

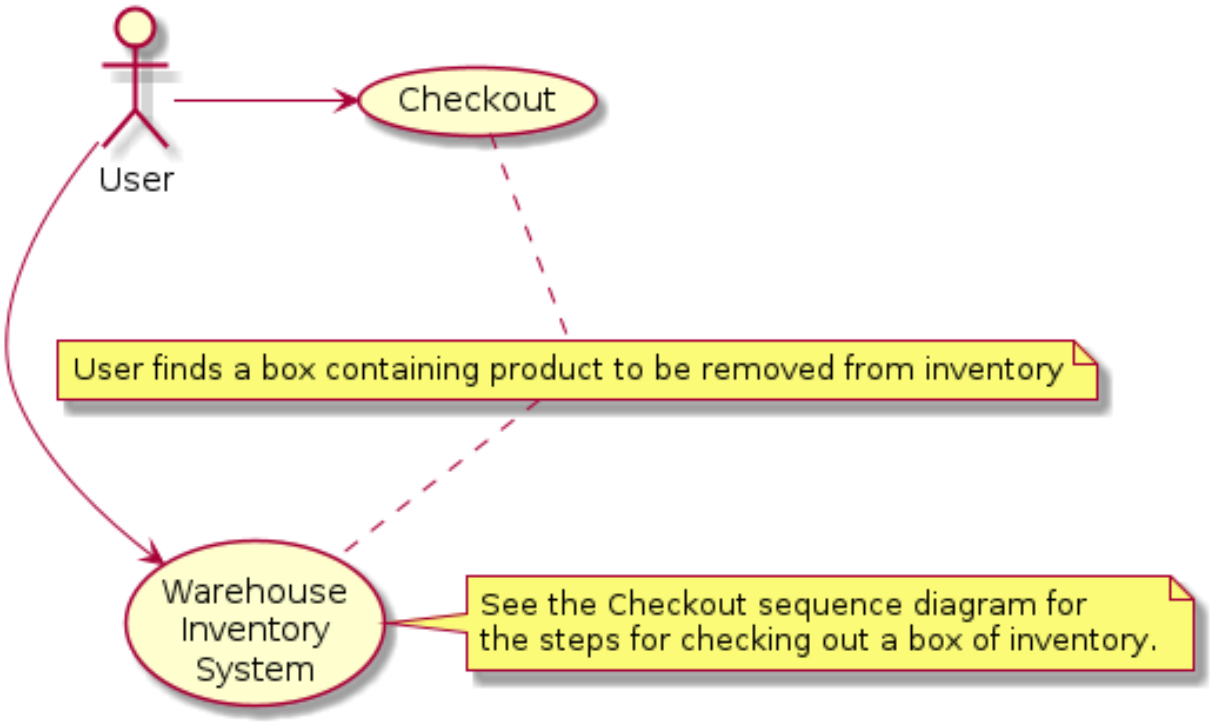


Fig. 5: **Checkout Inventory Use Case**

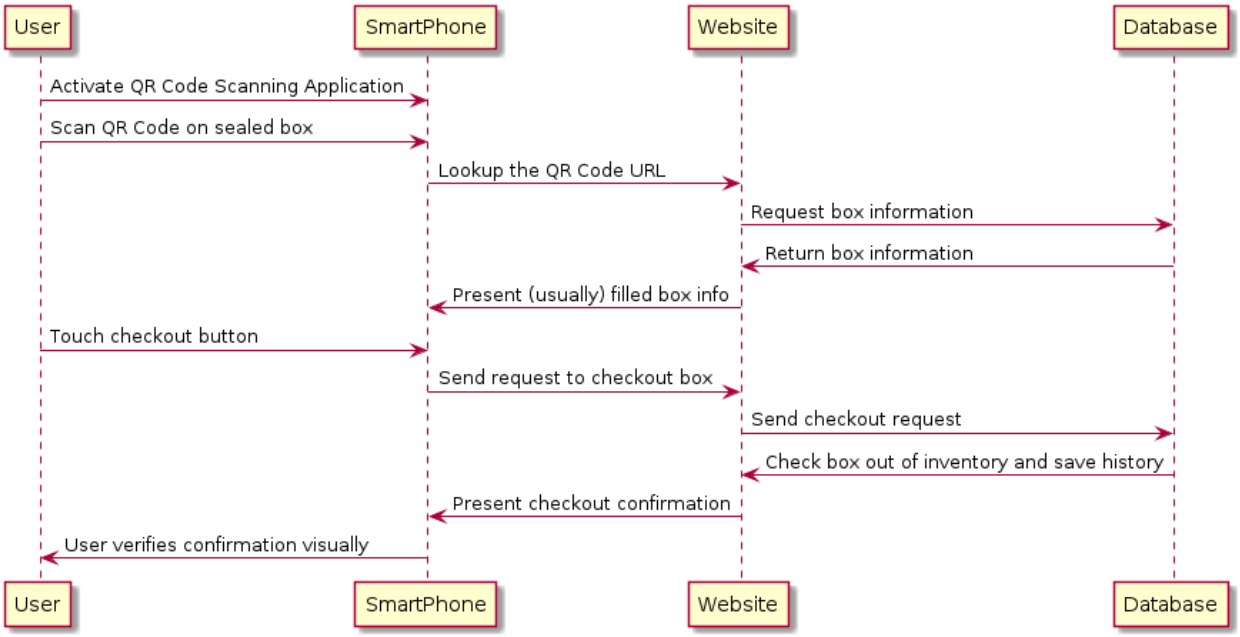


Fig. 6: **Checkout Inventory Sequence Diagram**

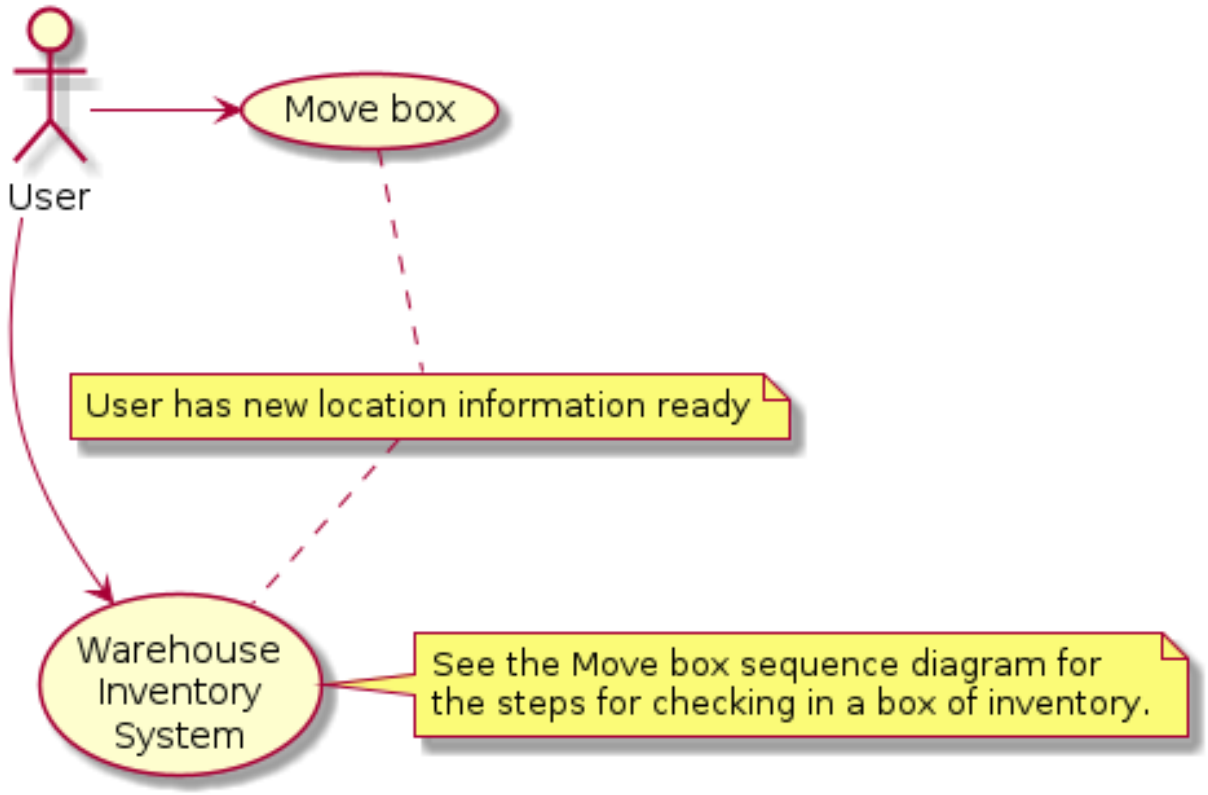


Fig. 7: **Move Box Use Case**

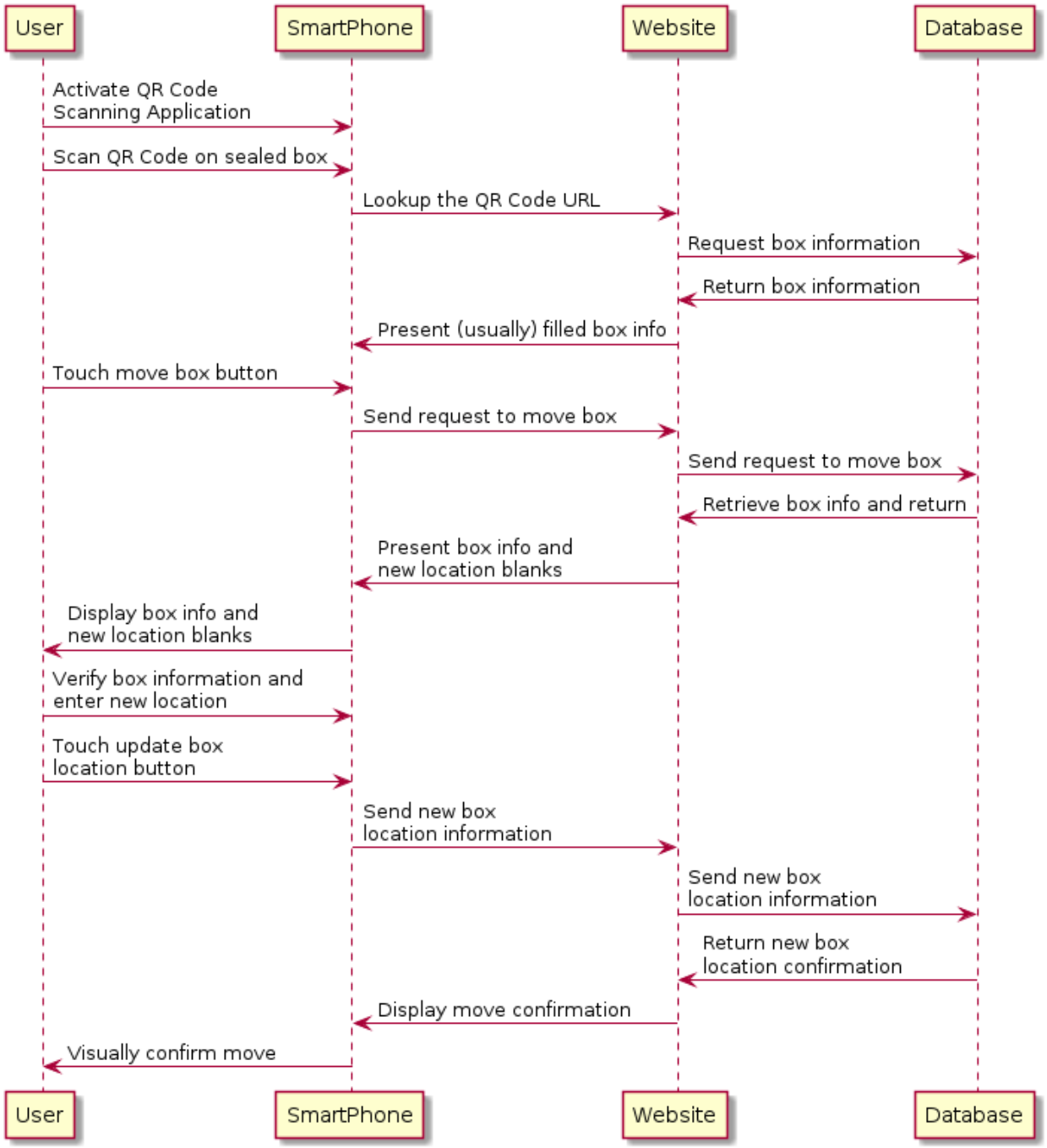


Fig. 8: **Move Box Sequence Diagram**

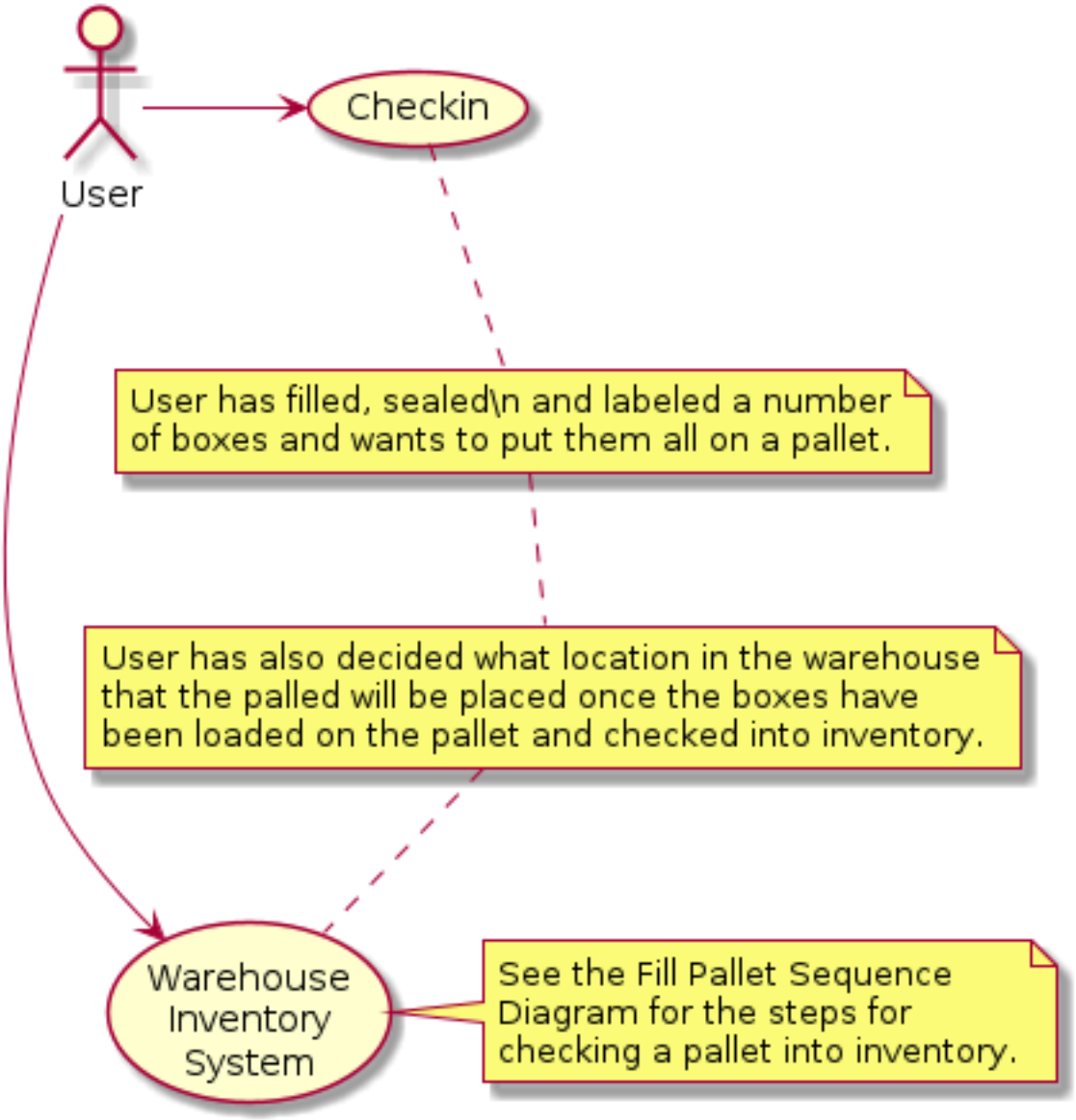


Fig. 9: **Fill Pallet Use Case**

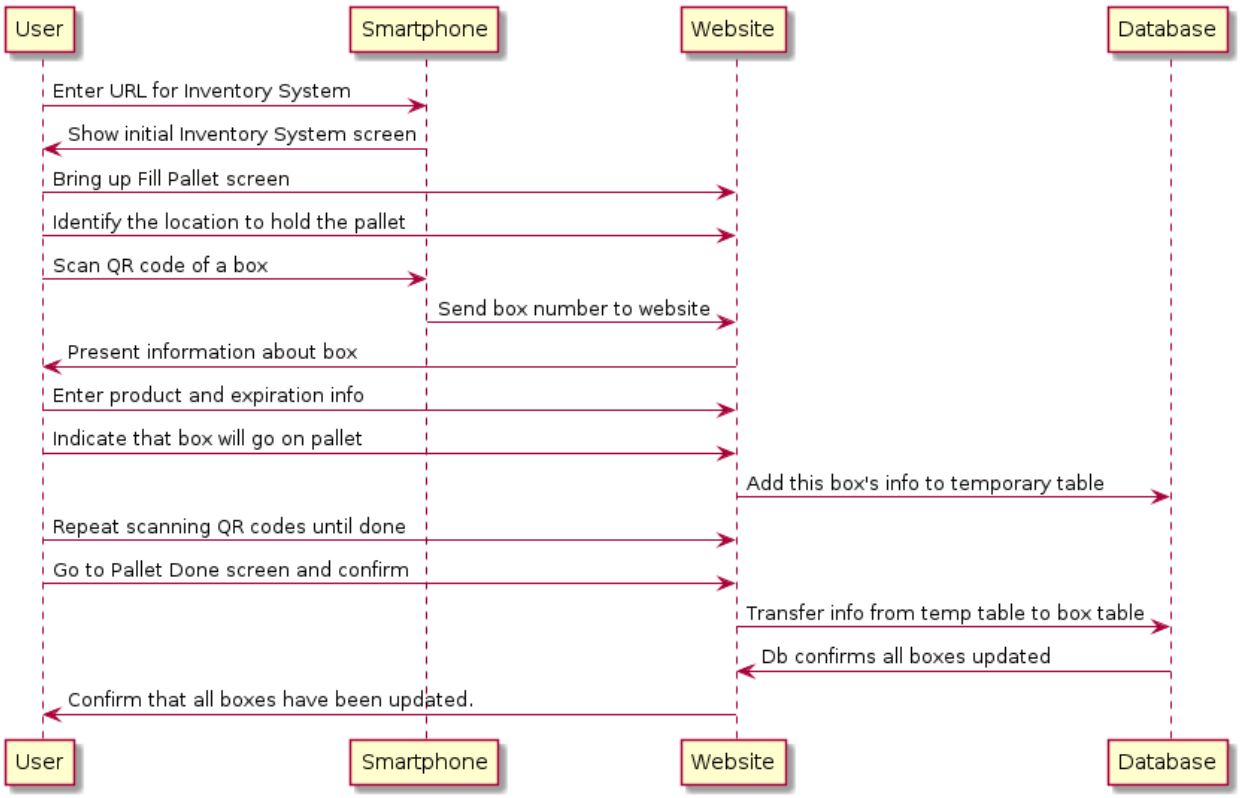


Fig. 10: **Fill Pallet Sequence Diagram**

3.3 Box and Activity Summary

This food pantry inventory system tracks the food and other product in the warehouse by box. When a box has product in it, the system tracks what is in the box, where the box is located, and when the product in it will expire.

The system also tracks box activity. The activity information is designed to help the folks running the food pantry understand the flow of product through the pantry over time. This information can help them see the trends of their clients. They can anticipate when they are about to run low or when they have an excess of some product that another food pantry can use before it expires and has to be thrown away.

3.3.1 Box Summary

All boxes have a box type – which indicates an approximate quantity of product it might contain. All boxes have a QR code label with a unique identifier included for humans to read. The QR code can be easily read by devices such as smart phones, tablets and laptops with cameras.

A box may be either empty or full in the inventory system. Scanning the QR code or typing in the identifier into the appropriate web page will reveal the state of the box in the system.

A full box in the inventory system has information about what product is in it, where it is located, and when the product in it will expire. The expiration information may indicate the only the year it expires or may indicate a part of a year, such as a quarter or month. If a box is moved, the system is updated so finding the current location of any box is known.

An empty box will have nothing but the box type. The inventory system will have no product, location, or expiration information. However, the boxes are expected to be sufficiently durable that they can be used a number of times.

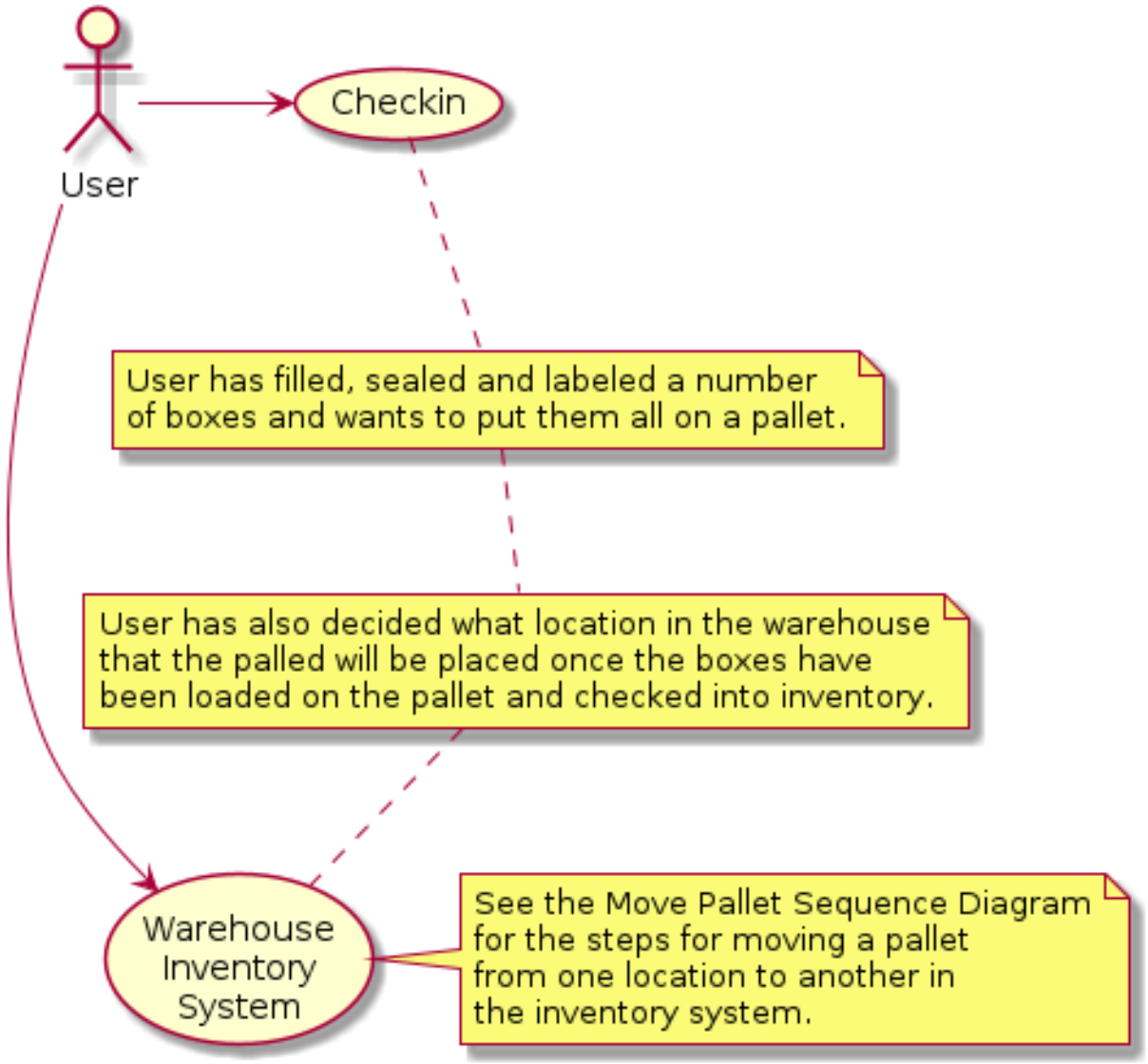


Fig. 11: **Move Pallet Use Case**

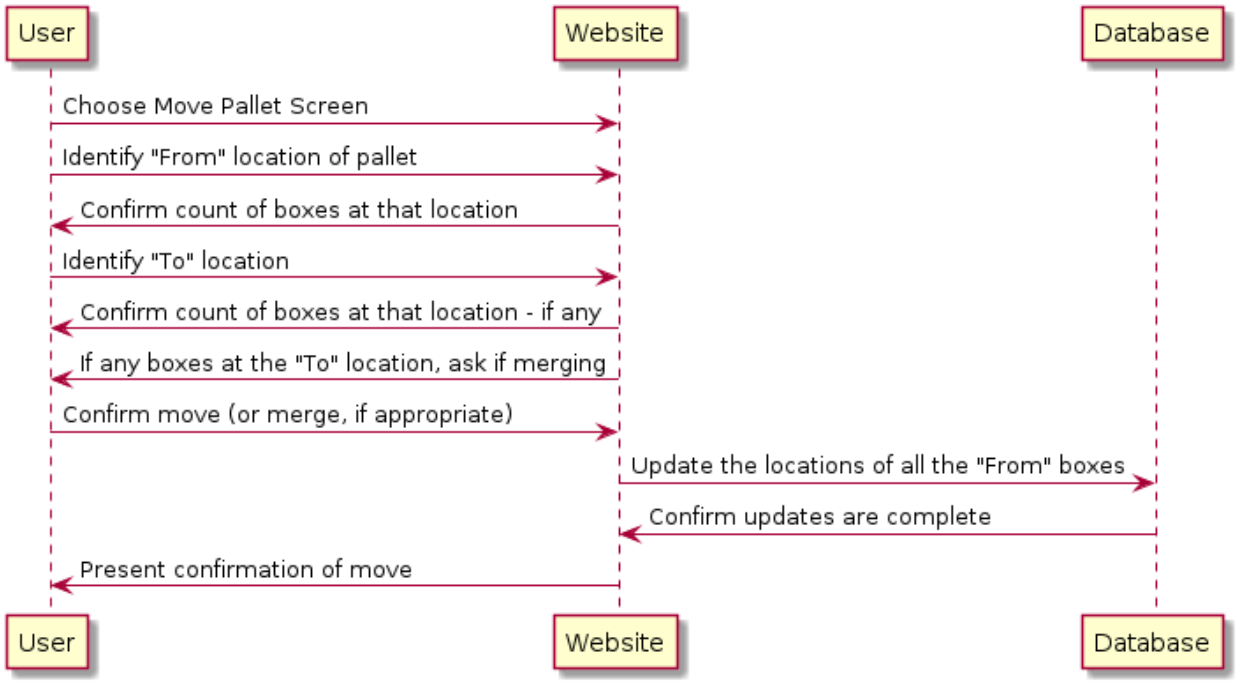


Fig. 12: **Move Pallet Sequence Diagram**

3.3.2 Activity Summary

When a box has product in it, the activity information will also have the product, approximate quantity, location and expiration information about that box. When a box is consumed (i. e. emptied) the activity information will be updated to indicate when it was consumed, how long the product was in the box, etc. When a box is used again, a new activity record will be created so the information about the previous contents is preserved.

3.3.3 Box Management Details

Ideally all boxes are handled correctly so this inventory system can accurately report the quantity and location of all products in it.

Alas, we all know that mistakes can be made. A box may get overlooked and not get checked in. A box may get moved without letting the inventory system know. A box may get emptied by someone who is unaware that there is an inventory system. The inventory system is tolerant of these foibles and tries to compensate rather than be rigid and require extra work.

The management of updating the database after user interaction has been isolated to two classes: BoxManagementClass and BoxActivityClass. These two classes should be the only code actually updating the database. Additional guidance is provided below in the section *Pallet Management* about how pallets of boxes are handled by this inventory system.

Box Management API Summary

The *BoxManagementClass* has five API's or public methods that can be called to update the database with details about a box (or a pallet of boxes).

Individual Box API's

For individual boxes the calls are:

BoxManagementClass.box_new Add a new box with a unique number to the inventory system. It will also have a predefined box type to indicate approximately how many items (cans, boxes, packages, etc.) are in the box.

BoxManagementClass.box_fill Fill an individual box with a named product at a specified location with a specified expiration year (and possibly range of months).

BoxManagementClass.box_move Move a filled box from one location to another in the warehouse.

BoxManagementClass.box_consume Empty a box (consume it's contents) by putting the contents into QC or directly in the pantry.

Box Management API Details

The expected input, actions, and output of the box management API calls are shown below.

BoxManagementClass.box_new

Add a new box with a unique box number to the inventory system.

Box New Expected Input

A unique box number and a valid box type.

Box New Action

A box record will be added to the inventory with the supplied box type. All other fields in the box will be empty.

Box New Output

The new box record will be returned.

BoxManagementClass.box_fill

Add an individual box containing a named product at a specified location with a specified expiration year (and possibly range of months).

Box Fill Expected Input

The unmodified box record and the information needed to update the box.

Box Fill Actions

This API call will mark the indicated box with the contents, location and expiration indicated. It will also make the call to create the appropriate Activity record.

Box Fill Output

The modified box record will be returned.

BoxManagementClass.box_move

Move a filled box from one location to another in the warehouse.

Box Move Expected Input

The unmodified box record and the target location.

Box Move Actions

This API call will change the location in the box record to the new location specified. It will also make the call to create the appropriate Activity record.

Box Move Output

The modified box record will be returned.

BoxManagementClass.box_consume

Empty a box (consume it's contents) by putting the contents into QC or directly in the pantry.

Box Consume Expected Input

The unmodified box record.

Box Consume Actions

This API call will make the call to create the appropriate Activity record. It will then clear all the product, location, and expiration fields in the box record.

Box Consume Output

The modified box record will be returned.

3.3.4 Pallet Management

Pallet management is designed to make dealing with pallets of boxes swift and easy. Rather than require strict conformance to some arbitrary rules in the inventory system, the system will accommodate variations to the typical scenarios. The scenarios below are not meant to be inclusive but are designed to show developers what might happen.

Pallet Management Scenarios

New Pallet Scenario The user will start with an empty pallet and add newly filled boxes of product to it. When the pallet is full or there are no more newly filled boxes, a location is selected and (after scanning the QR codes and filling in the pallet box info) the pallet is placed in the new location.

Variation: The user decides to take similarly packed boxes from another pallet and add them to this pallet. The system will recognize that these boxes were originally somewhere else and will process them as a move.

Add to a Pallet Scenario The user will pull a pallet out of the racks and add newly filled boxes to it. After scanning the QR codes and filling in the pallet box info for the boxes just added, the user puts the pallet back in the racks at the same location.

Variation 1: The user decides to take similarly packed boxes from another pallet and add them to this pallet. The system will recognize that these boxes were originally somewhere else and will process them as a move.

Variation 2: The user accidentally scans the QR code for a box that was already on the pallet. As long as the product and expiration information are the same, the system will ignore the entry.

Variation 3: The user decides to put the pallet in a different location. As long as all the boxes are scanned, when the finished pallet is processed (with the new location in the pallet record), all the pallet boxes that were originally on the pallet will be treated as a move to the new location.

Move a Pallet Scenaro The user will choose the old location. The system will prepopulate the pallet boxes with all the boxes at the old location. The user will then designate the new location and the system will move the boxes indicated by the box pallet records accordingly.

Developer Suggestions

Perhaps a way of minimizing the amount of scanning by the user would be to either prepopulate the pallet boxes when a location with boxes is selected, or to have a button for the user to select to have the system prepopulate the pallet boxes on demand.

Another suggestion is that when a QR code is scanned for a box that is filled, to populate the pallet box with all the information from the box record.

Pallet Management API's

The call for processing a pallet of boxes is:

BoxManagementClass.pallet_finish If pallet status is "Fill", this API will add the pallet of filled boxes to the specified location in the warehouse.

If pallet status is "Merge" it will merge two or more pallets and put the resulting pallet boxes at the specified location.

If the pallet status is "Move" it will move a pallet of filled boxes to a different location.

Note - at this time, there is no option to consume a whole pallet of filled boxes.

Pallet Management API Details

The expected input, actions, and output of the box management API calls are shown below.

BoxManagementClass.pallet_finish

Process a pallet of boxes and do the appropriate action to each box.

Pallet Finish Expected Input

The pallet record with an appropriate pallet status in it. The pallet status will indicate if this is an addition of product to inventory ("FILL"), a move of a pallet of boxes from one location to another ("MOVE"), or a consolidation of boxes from various old locations to a new location ("MERGE"). The associated PalletBox records will have the product and expiration information, as well as an individual status that will guide this processing.

Pallet Finish Actions

This API will walk through the PalletBox records associated with this Pallet. Each corresponding box will be modified as needed.

If the pallet box status is:

NEW: The existing box record is expected to be empty and is being filled with the information from the pallet box and pallet. The pallet box will be added to the system.

If the information from the ballet box and pallet record matches the box record already in inventory, there will be no change to the database.

If the product and expiration information is the same but the location is different, this action will be treated as a move.

ORIGINAL: The box information should have been copied into the pallet box record because other boxes are being added to the pallet at this location or that this entire pallet is being moved to a new location. If the only difference is the location, it will be treated as a move.

If the information from the ballet box and pallet record matches the box record already in inventory, there will be no change to the database.

MERGE: This record will be treated the same way as if its status was "ORIGINAL".

Pallet Finish Output

Nothing will be returned.

3.3.5 Activity Management

Box Activity API

The Box Activity API (*BoxActivityClass*) records information in the Activity table so that what is available can be readily discerned and that the flow of product through the facility can be determined.

Although it has three public methods, none of them should be called directly because the box management API's will take care of calling the appropriate box activity API.

Box Activity API Details

The details for each Box Management API call are documented in the source code of the call.

BoxManagementClass see BoxManagementClass in fpiweb/support/BoxManagement.py.

BoxActivityClass see BoxActivityClass in fpiweb/support/BoxActivity.py.

3.4 Deployment Documentation

3.4.1 Deployment Overview

3.4.2 Django and Nginx Interaction

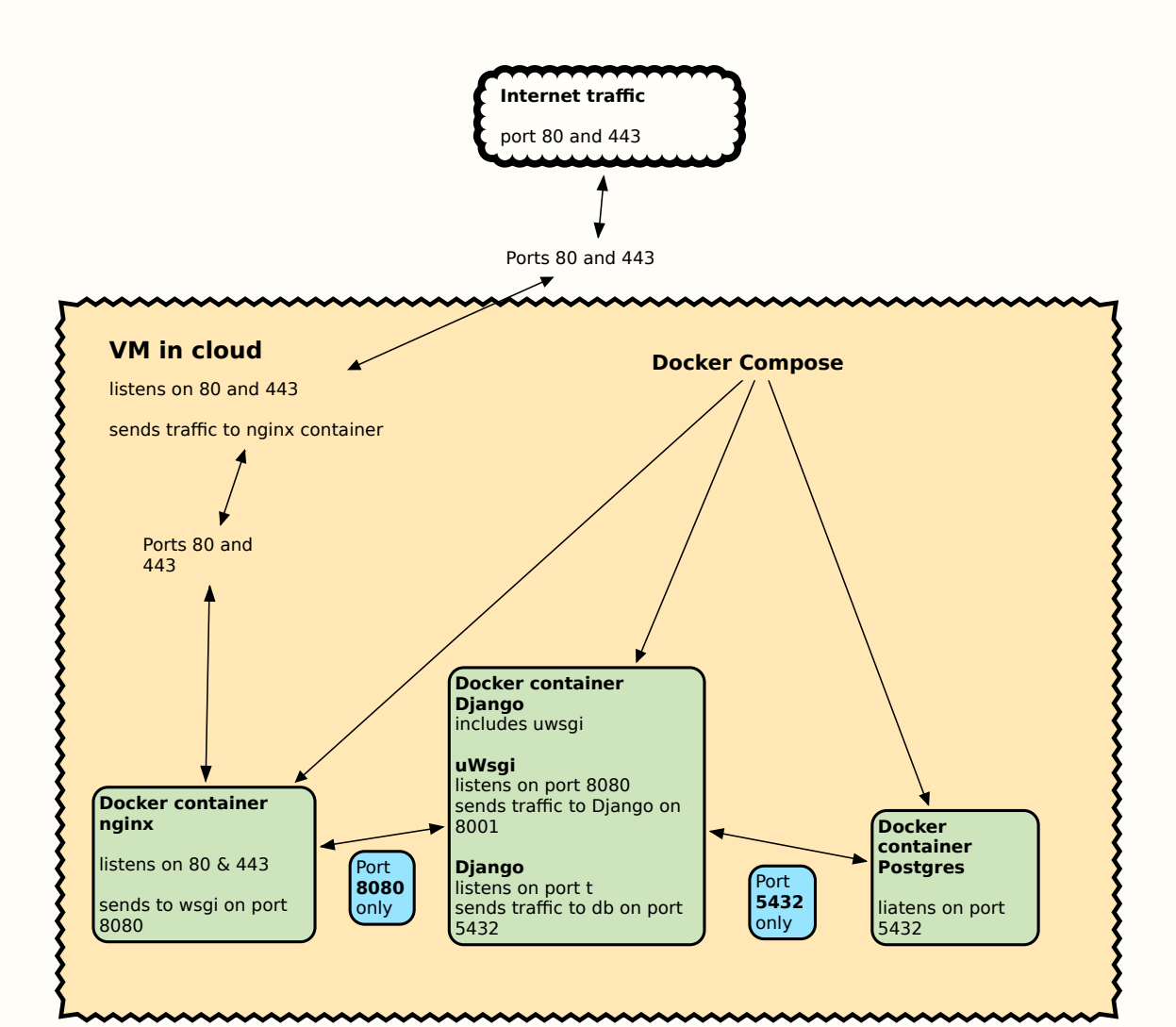
Interaction between Django and Nginx is given in *Django and Nginx*

3.5 Django and Nginx

Using nginx to be a web server for a Django project.

3.5.1 Requirements

- Install uwsgi library with pip into the Django virtual environment.
- Install nginx (I used homebrew)
- Configure the nginx configuration file
  - Nginx only looks for configuration files in certain places
  - Nginx only writes log files in certain places
    - I needed to manually create the log directory.


Internet traffic
port 80 and 443
Ports 80 and 443
VM in cloud
listens on 80 and 443
sends traffic to nginx container
Ports 80 and 443
Docker Compose
Docker container nginx
listens on 80 & 443
sends to wsgi on port 8080
Port 8080 only
Docker container Django
includes uwsgi
uWsgi
listens on port 8080
sends traffic to Django on 8001
Django
listens on port t
sends traffic to db on port 5432
Port 5432 only
Docker container Postgres
liatens on port 5432

Nginx configuration file (sample)

```
    worker_processes 2;

    error_log  logs/error.log;
    error_log  logs/error.log  notice;
    error_log  logs/error.log  info;

    pid        logs/nginx.pid;

    events {
        worker_connections  1024;
    }

    http {
        include       mime.types;
        default_type  application/octet-stream;

        log_format  main  '$remote_addr - $remote_user [$time_local] "$request" | '
                          '$status | $body_bytes_sent | "$http_referer" | '
                          '"$http_user_agent" | "$http_x_forwarded_for"';

        access_log  logs/access.log  main;

        sendfile        on;
        #tcp_nopush     on;

        #keepalive_timeout  0;
        keepalive_timeout  65;

        #gzip  on;

        server {
            listen 127.0.0.1:8080;
            server_name  localhost;

            access_log  logs/host.access.log  main;

            location / {
                include uwsgi_params;
                uwsgi_pass 127.0.0.1:8765;
            }

        }
    }
```

- Uwsgi Info

Uwsgi provides the link between Django and the web server(e.g. nginx)

uwsgi

- uwsgi command line for following configuration

```
uwsgi --yaml uwsgi_conf.yaml
```

- uwsgi Configuration File (e.g. sample)

```
# uwsgi_conf.yaml - Configure wsgi
%YAML 1.2
---
version: 1

uwsgi:
    # Django-related settings
    # chdir : '/Volumes/MBPC/Dvl/Python/PythonProjects/Food-Pantry-Inventory
    # -Others/Food-Pantry-Inventory-Jan/'

    # Django's wsgi file'
    wsgi-module : 'FPIDjango/wsgi.py'
    # home : '/Volumes/MBPC/Dvl/Python/PythonProjects/Food-Pantry-Inventory
    # -Others/Food-Pantry-Inventory-Jan/'
    # Enable python threading
    enable-threads : True

    # process-related settings
    # master
    master : True

    # maximum number of worker processes
    processes : 4

    # use http style interface on a particular port
    http : 8765
```

3.6 JavaScript and JQuery Tips

3.6.1 JavaScript Library Sources

3.6.2 JavaScript Library Sources

jQuery

Bootstrap4 apparently wants a specific version of jQuery:

jquery-3.3.1.min.js

It can be downloaded from the jQuery site (as of Feb 2020). Copy the file into the appropriate project directory. Currently that is:

```
./Food-Project-Inventory/fpiweb/static/fpiweb/
```

Popper

Bootstrap also wants to use a library called popper. The developers of this library strongly urge that it be downloaded and maintained using a version control system such as npm. The following are suggested steps for obtaining a current version of popper:

1. Install npm (or pnpm). Currently is is available via Homebrew. (See Pnpm site for other ways to obtain pnpm.)

    brew install pnpm

#. If you already have a js repository go there. Otherwise move to an empty directory. Run this command to download and build popper.

```
pnpm install @popperjs/core
```

#. Go to the directory where the one file we need is located.

```
cd ./node_modules/@popperjs/core/dist/umd/
```

#. Copy the file "popper.min.js" into the appropriate project directory. Currently that is:

```
./Food-Project-Inventory/fpiweb/static/fpiweb/
```

The Three Levels of DOM Manipulation

There are 3 levels on which the page's Document Object Model (DOM) may be manipulated.

1. Low-Level DOM manipulation

```
// Add a paragraph with text to a div
let div = document.getElementById('myDiv');
let p = document.createElement('p');
div.appendChild(p);
let text = document.createTextNode("Hello World");
p.appendChild(text);

// Search through children of div element (Only 1 level down in tree)
for(var i
```

2. Element-specific properties and methods

```
let field = document.getElementById('name');
field.value = "John";
```

3. JQuery manipulations

```
let allDivs = $('div');
allDivs.append('<p>hello</p>');
```

3.7 Django Extensions

One of the libraries included in this project is django-extensions (https://django-extensions.readthedocs.io/). It has a number of useful extension to the manage.py module provided by Django. The extensions used by this project are:

Model Relationships This extention produces a graph of the tables in the application and their interrelationships. More details are provided in *Model (Table) Relationships* below.

Model vs. Database Differences This extension produces output that shows if your models and database schema are in sync. If not, it outputs the SQL statements that will put it in sync. See Model vs. Database Differences below for an example.

Show all URLs This extension is useful to show all defined URL's, the view associates with each URL, and the name attached to that URL. See Show URLs below.

Dump Script A extension is available to dump the entire schema and all data to a Python script. See Dump Script below.

Dump Data The dump data command can dump the contents of a table in a format suitable as a fixture file. See Dump Data below.

3.7.1 Model (Table) Relationships

The extension "graph-models" generates a graph showing all the models (tables) defined and how they are related. Each foreign key is shown with an arrow pointing to the source table and field. The arrows indicate if the relationship is one-to-one, one-to-many or many-to-many.

A graph for this project can be downloaded from

This extension generatss a ".dot" file that is interpreted by the GraphViz library (see GraphViz documentation).

By also loading the library PyGraphViz (see PyGraphViz documentation) the " .dot" file generated above is converted into a format usable for humans.

The command used to make all this happen is:

```
./manage.py graph_models fpiweb auth -g -o InventoryProjectModelsGrouped.pdf
```

3.7.2 Model vs. Database Differences

This extension shows any difference between the models and the database.

The command:

```
./manage.py sqldiff -a -e
```

produced this output:

```
BEGIN;
-- Application: fpiweb
-- Model: Activity
DROP INDEX "fpiweb_activity_BoxNumber_10d7ca04_like";
-- Model: Constraints
ALTER TABLE "fpiweb_constraints"
        ADD CONSTRAINT "fpiweb_constraints_constraint_name_0fa0e463_uniq" UNIQUE (
↪"constraint_name");
CREATE INDEX "fpiweb_constraints_constraint_name_0fa0e463_like"
        ON "fpiweb_constraints" ("constraint_name" varchar_pattern_ops);
COMMIT;
```

This command reported the result as text rather than SQL.

```
./manage.py sqldiff -a -e -t
```

Results:

```
+ Application: fpiweb
|-+ Differences for model: Activity
|--+ field 'fpiweb_activity_BoxNumber_10d7ca04_like' INDEX defined in database schema␣
↪but missing in model
|-+ Differences for model: Constraints
|--+ field 'constraint_name' UNIQUE named 'fpiweb_constraints_constraint_name_
↪0fa0e463_uniq' defined in model but missing in database
|--+ field 'constraint_name' INDEX named 'fpiweb_constraints_constraint_name_0fa0e463_
↪like' defined in model but missing in database
```

3.7.3 Show URLs

The "show_urls" command is not as polished as the other commands although it still produces useful output.

The command:

```
./manage.py show_urls
```

produces output like this (partial):

```
/       fpiweb.views.LoginView
/fpiweb/        fpiweb.views.IndexView  fpiweb:index
/fpiweb/about/  fpiweb.views.AboutView  fpiweb:about
/fpiweb/activity/download/      fpiweb.views.ActivityDownloadView       ␣
↪fpiweb:download_activities
/fpiweb/box/<int:pk>/   fpiweb.views.BoxDetailsView     fpiweb:box_details
/fpiweb/box/<int:pk>/edit/      fpiweb.views.BoxEditView        fpiweb:box_edit
/fpiweb/box/<int:pk>/empty/     fpiweb.views.BoxEmptyMoveView   fpiweb:box_empty
/fpiweb/box/<int:pk>/empty_move/        fpiweb.views.BoxEmptyMoveView   fpiweb:box_
↪empty_move
/fpiweb/box/<int:pk>/fill/      fpiweb.views.BoxEmptyMoveView   fpiweb:box_fill
/fpiweb/box/<int:pk>/move/      fpiweb.views.BoxMoveView        fpiweb:box_move
/fpiweb/box/box<int:number>/    fpiweb.views.BoxScannedView     fpiweb:box_scanned
/fpiweb/box/box_form/   fpiweb.views.BoxItemFormView    fpiweb:box_form
/fpiweb/box/new/<str:box_number>/       fpiweb.views.BoxNewView fpiweb:box_new
/fpiweb/build_pallet/   fpiweb.views.BuildPalletView    fpiweb:build_pallet
/fpiweb/build_pallet/   fpiweb.views.BuildPalletView    fpiweb:build_pallet
/fpiweb/build_pallet/<int:box_pk>/      fpiweb.views.BuildPalletView    fpiweb:build_
↪pallet_add_box
```

(continues on next page)

(continued from previous page)

```
/fpiweb/build_pallet/<str:box_number>/  fpiweb.views.BuildPalletView    fpiweb:build_
↪pallet_add_box
/fpiweb/constraint/add/ fpiweb.views.ConstraintCreateView       fpiweb:constraint_new
/fpiweb/constraint/delete/<int:pk>/     fpiweb.views.ConstraintDeleteView       ␣
↪fpiweb:constraint_delete
/fpiweb/constraint/edit/<int:pk>/       fpiweb.views.ConstraintUpdateView       ␣
↪fpiweb:constraint_update
/fpiweb/constraints/    fpiweb.views.ConstraintsListView        fpiweb:constraints_
↪view
/fpiweb/index/  fpiweb.views.IndexView  fpiweb:index
/fpiweb/loc_bin/        fpiweb.views.LocBinListView     fpiweb:loc_bin_view
/fpiweb/loc_bin/add/    fpiweb.views.LocBinCreateView   fpiweb:loc_bin_new
/fpiweb/loc_bin/delete/<int:pk>/        fpiweb.views.LocBinDeleteView   fpiweb:loc_
↪bin_delete
/fpiweb/loc_bin/edit/<int:pk>/  fpiweb.views.LocBinUpdateView   fpiweb:loc_bin_update
/fpiweb/loc_row/        fpiweb.views.LocRowListView     fpiweb:loc_row_view
/fpiweb/loc_row/add/    fpiweb.views.LocRowCreateView   fpiweb:loc_row_new
/fpiweb/loc_row/delete/<int:pk>/        fpiweb.views.LocRowDeleteView   fpiweb:loc_
↪row_delete
/fpiweb/loc_row/edit/<int:pk>/  fpiweb.views.LocRowUpdateView   fpiweb:loc_row_update
/fpiweb/loc_tier/       fpiweb.views.LocTierListView    fpiweb:loc_tier_view
/fpiweb/loc_tier/add/   fpiweb.views.LocTierCreateView  fpiweb:loc_tier_new
/fpiweb/loc_tier/delete/<int:pk>/       fpiweb.views.LocTierDeleteView  fpiweb:loc_
↪tier_delete
/fpiweb/loc_tier/edit/<int:pk>/ fpiweb.views.LocTierUpdateView  fpiweb:loc_tier_update
/fpiweb/login/  fpiweb.views.LoginView  fpiweb:login
/fpiweb/logout/ fpiweb.views.LogoutView fpiweb:logout
/fpiweb/maintenance/    fpiweb.views.MaintenanceView    fpiweb:maintenance
/fpiweb/manual_add_box/ fpiweb.views.ManualNewBoxView   fpiweb:manual_add_box
/fpiweb/manual_box_status/      fpiweb.views.ManualBoxStatusView        fpiweb:manual_
↪box_status
...
```

The output has the URL, the view it invokes, and the name attached to it. The output has a tab character between each field so it could be imported into a spreadsheet for easy reference and manipulation. At this point in our development, run the command to get the current list of URL's.

3.7.4 Dump Script

The "dumpscript" command will dump all the schema and data from the current database into a script. The script can subsequently be used to reload into a database that was just created. The script will create the schema for the application (and other Django tables) and load all the previously known data into it. The advantage of the script vs. a database backup is that the script is completely in python so there are no SQL syntax details get in the way of moving to a different database vendor.

Unfortunately, this extension chokes on our database because we have "time zone aware" data in it. Perhaps we can file an issue with the maintainers and get a fix or workaround.

3.7.5 Dump Data

The "dumpdata" command can be used to create a file suitable for inclusion in the fixtures directory. Once the table has all the desired records and values, use a command similar to the one below.

```
./manage.py dumpdata  -o pallot.json fpiweb.pallet
```

This command is an example of dumping the fpiweb pallet table to a file called pallet.json in the current directory. It can then be moved to the fixtures directory or other desired location.

(more Django extensions used by this project)

3.8 Outstanding Tasks

There are numerous tasks and activities that remain to be accomplished by someone before this project will be complete. They have been divided into the following milestones.

3.8.1 Milestone Descriptions

M-V0.95 Version 0.95 - Complete functionality

M-V1.0 Version 1.0 - Minimum Viable Product (MVP)

M-V1.1 Version 1.1 - First polish

M-ReWk Needs to be reworked.

3.8.2 Task Tables by Milestone

Table 1: Milestone 0.95 Tasks - Complete Functionality

Issue #	Milestone	Title
I-216	*M-V0.95*	Get formal permission to use WARM logo
I-215	*M-V0.95*	Determine source for X509 certificates
I-213	*M-V0.95*	Ensure system can be used from a smart phone/tablet
I-212	*M-V0.95*	Cloud VM
I-210	*M-V0.95*	Admin Permissions
I-209	*M-V0.95*	Staff Permissions - can:
I-208	*M-V0.95*	Volunteer Permissions - can:
I-207	*M-V0.95*	Implement permission levels though out the app
I-205	*M-V0.95*	Setup a docker container for Nginx
I-204	*M-V0.95*	Setup a docker container for Django and uWSGI
I-203	*M-V0.95*	Setup docker container with PostgreSQL configuration
I-202	*M-V0.95*	Implement Manual Pallet Management
I-178	*M-V0.95*	Manual box checkin needs to be streamlined
I-173	*M-V0.95*	Build Pallet needs to set pallet status
I-167	*M-V0.95*	Results of Mike R.'s original testing
I-161	*M-V0.95*	Move Pallet
I-115	*M-V0.95*	Rework the API of theQR label print program
I-88	*M-V0.95*	Make a docker image of project to run locally

Table 2: Milestone 1.0 MVP Tasks

Issue #	Milestone	Title
I-222	*M-V1.0*	Let's Encrypt X509 certificate
I-221	*M-V1.0*	Setup production with "dummy" data initially
I-220	*M-V1.0*	Test production for security
I-219	*M-V1.0*	Test backup
I-218	*M-V1.0*	Arrange backup of data and schema in database.
I-217	*M-V1.0*	Script production fallback
I-214	*M-V1.0*	Script production (docker, VM, cloud provider)...
I-206	*M-V1.0*	View/search the Product Examples table
I-190	*M-V1.0*	Build Pallet: set pallet ID in profile
I-176	*M-V1.0*	Change user password
I-175	*M-V1.0*	Add/Change/Delete a User
I-145	*M-V1.0*	Document clearing start/end exp. mo in Build Pallet

Table 3: Milestone 1.1 First Polish Tasks

Issue #	Milestone	Title
I-227	*M-V1.1*	Checkout a pallet of boxes (future)
I-226	*M-V1.1*	Generate Location Table from Row, Bin, and Tier
I-225	*M-V1.1*	Reorganize menu picks by most frequently used
I-224	*M-V1.1*	Location Table - build ACID screens
I-223	*M-V1.1*	Change a box's quantity and box type
I-211	*M-V1.1*	Eliminate the standalone QR print program
I-198	*M-V1.1*	Build Pallet Screen UI Change w/ no previous pallets
I-192	*M-V1.1*	Build Pallet Screen. Save more in Pallet Box record
I-189	*M-V1.1*	Ensure that the "Evans" box type is default
I-174	*M-V1.1*	Delete a pallet
I-138	*M-V1.1*	Enhance README.md with a more detailed purpose.
I-89	*M-V1.1*	Improve README.md etc. for new developers
I-54	*M-V1.1*	Product Examples Table - Build ACID screens
I-42	*M-V1.1*	Box Type Table - Build ACID screens
I-41	*M-V1.1*	Product Table - build ACID screens
I-40	*M-V1.1*	Product Category Table - build ACID screens

3.8.3 Task Descriptions by Issue

These task provide details for all the known activities that remain to be done. Each is identified with an issue number as labeled in Github. As tasks are completed the details will be moved to an appropriate location elsewhere in this documentation.

I-227 Checkout a pallet of boxes (future)

On rare occasions, the food pantry will give a whole pallet of product to another food pantry. It would be convenient to add this functionality to the system – eventually.

I-226 Generate Location Table from Row, Bin, and Tier

Factor in exclusions given in the Constraints table

I-225 Reorganize menu picks by most frequently used

- Perhaps this can be done when first going to production.
  - If so, it may need to be reviewed again after the clients gain experience with the system.

I-224 Location Table - build ACID screens

Add editing for the Location Table - somewhat similar to the Constraints Table.

- Perhaps a new entry could be "built up" by choosing a combination of valid row, bin, and tier.
- Changing or deleting an entry should be forbidden if any currently filled box or pallet refers to it.
- Currently this functionality is available with the Django admin subsite.

I-223 Change a box's quantity and box type

Provide a way for staff to change a box's quantity and box type.

- A particular box can have a quantity not equal to the quantity in the box type
- Verify that no other code changes the box quantity once set
- Initial quantity set from box type when box is added to inventory
  - All ways of adding a box to inventory should use BoxManagement rather than setting the box type and quantity directly.

I-222 Let's Encrypt X509 certificate

Identify and document how to obtain and automatically renew a Let's Encrypt X509 certificate

I-221 Setup production with "dummy" data initially

This is so staff and volunteers can gain experience with the system

I-220 Test production for security

Brian Caslow offered to do this.

The system should be tested for vulnerabilities at least quarterly.

I-219 Test backup

- Restore data from backup to ensure the the backup is valid
- Schedule the tests at least quarterly

I-218 Arrange backup of data and schema in database.

- Backup data and schema only to minimize the size of backup
- Transfer backup out of cloud to other storage securely
- Determine frequency of backup to minimize data loss
- Test!

I-217 Script production fallback

So system can be quickly reverted to the previous version if needed

- test!

I-216 Get formal permission to use WARM logo

Get formal permission to use the WARM logo for the web site and the documentation

I-215 Determine source for X509 certificates

Determine if we need to set up X509 certificates from Let's Encrypt or if WARM already has certificate we will be allowed to use.

This issue may depend on issue #222.

I-214 Script production (docker, VM, cloud provider)...

This issue depends on issues #203, #204, and #205 to be completed.

Also depends on issue #215.

I-213 Ensure system can be used from a smart phone/tablet

Test using both Android and iOS devices, both smart phones and well as tablets of each kind.

- Verify that a camera can be used to scan a QR code at the appropriate place.
- Optionally test with other devices, such as a Fire, etc.

I-212 Cloud VM

- Identify cloud provider we will use for production
- Identify costs, who will pay for it and how it will be billed

I-211 Eliminate the standalone QR print program

Reason - it needs access to the database to skip box numbers already in use. However, when the live database is in production, providing secure access to the database will be more difficult than just using the menu pick.

I-210 Admin Permissions

Has full access to all menu picks - including the Django admin pages

I-209 Staff Permissions - can:

- Can manage all user data:
  - boxes
  - durable data (e.g. product, box type, location, etc.)
  - constraints
- Can print QR labels
- Can dump activity data
- Can add user information - including permission level and initial password
- Can change user permission level (volunteer -> staff, staff -> volunteer)
- Can block a user's access
- Everything that a volunteer can do

I-208 Volunteer Permissions - can:

- Add boxes
- Checkin/checkout boxes and pallets
- Move boxes and pallets
- Check box status
- change their own password
- change their name, email address, title

- view product examples

I-207 Implement permission levels though out the app

- Only login page does not require the LoginMixIn.
- Each page will verify if the user is allowed to access that page.
- Throws a polite error message and allow return to previous page if not.
- So far, only the Add user page and the change user permission page will have conditional logic based on the permission of the user.
- Perhaps this can be isolated to one class or function that contains the necessary logic.

I-206 View/search the Product Examples table

- This view/search should be a "full text" search of the Product Examples Table. E.g. a search for "peas" should show all records that have peas somewhere in the description including black eyed peas, green peas, young peas and peasant (if exists).
- Each entry found should list the product that is appropriate for it so that the user can select the proper product.
- This is particularly important for product examples that could be in in more than one product.
  - Beef stew could be in either meat or soup.
  - It is up to the food pantry to decide.
- This view search should be available to any logged in user.

This view or search should be available whenever a product is being selected. How to accomplish this is up to the implementor.

I-205 Setup a docker container for Nginx

- Build container with a fresh copy of a specified version of Ngnix
- Uses a "secret" configuration including:
  - Specified IP and port to allow web traffic (possibly non-standard)
  - Specified IP and port to send web traffic to the Django container (possibly non-standard)
- Includes X509 certificates to enable HTTPS - possibly from Let's Encrypt
- Includes any hardening needed for production

I-204 Setup a docker container for Django and uWSGI

- Builds container with a fresh copy of a specified version of Django
- Loads other production dependencies and libraries
- Uses a "secret" configuration including:
  - Database user id and password
  - Specified IP and port to access database (possibly non-standard)
  - Specified IP and port to receive web traffic (possibly non-standard)

- Includes uWSGI installation and configuration
- Includes any hardening needed for production

I-203 Setup docker container with PostgreSQL configuration

- Builds container with fresh copy of a specified version of PostgreSQL
- Uses a "secret" superuser id and password and other configuration parameters
- Loads the schema from a specified source
- Loads data from a specified backup source
- Allows access from a specified IP and port (possibly non-standard)
- Includes automatic backup of the database at a specified interval
- Includes means of exporting the backup securely
- Includes any hardening needed for production

I-202 Implement Manual Pallet Management

Implement the remaining Manual Pallet Management screens.

- Start a new pallet
  - Request a pallet name
  - Verify that the new pallet name is different from any other pallet name
    - If match, show error message and allow different entry
  - Proceed to common screen described below
  - Allow user to return to menu if desired
- Select a pallet in progress
  - Show a list of existing pallet names and allow selection of one of them
  - Proceed to common screen described below
  - Allow user to return to menu if desired

Common screen - Present a screen somewhat similar to the second Build Pallet screen. - Request a box number (text only, no QR code)

- Display error if box not in system or is full — allow reentry
  - (Different from Build Pallet - may be keying error)
- Show box number
- Request product, expiration year (and optional start/end months)
- Allow box to be removed from the pallet
- Allow box information to be changed
- Provide menu pick to complete the pallet.

Internal requirements:

- Set or replace Pallet ID in Profile Record as soon as pallet is created or selected.

- Add or update Pallet Box Records as soon as user moves on to the next box.

Validate product and expiration date information

- Delete Pallet Box Record immediately if user deletes it from list
- Verify that a golem box number is not specified twice
  - Check can be delayed until ready to complete the pallet

I-198 Build Pallet Screen UI Change w/ no previous pallets

On the Build Pallet screen, an improvement would be that when there are no Pallet records, only the Add Pallet input area would be shown. In that case the Select Pallet input area would be hidden.

I-192 Build Pallet Screen. Save more in Pallet Box record

Comments from existing issue:

Here's the scenario as Travis describes it:

I added a few boxes to a pallet using the main build pallet screens, then went back to the manual pallet menu to see how it displayed. I discovered that the main build pallet screen was adding the box to the pallet box records, but was not filling in the product or expiration date until the pallet complete is selected. This can be a problem because if one person starts a pallet, then leaves, another person cannot pick up where the first person left off

This behavior isn't surprising. When the box is scanned the product and expiration dates are unknown. If the user fills in product and expiration date and clicks the "Pallet complete" the product and expiration date information is saved in the database. To allow a user to partially complete a pallet and then leave it and pick it up later we need an additional mechanism. One thought would be to have a save button, but the user has to remember to click it. I'm thinking of adding an on-change event handler that would update the record in the database whenever a field changes.

Scenario and comments from Travis' original email:

I added a few boxes to a pallet using the main build pallet screens, then went back to the manual pallet menu to see how it displayed. I discovered that the main build pallet screen was adding the box to the pallet box records, but was not filling in the product or expiration date until the pallet complete is selected. This can be a problem because if one person starts a pallet, then leaves, another person cannot pick up where the first person left off. I second concern is that two people cannot work together to check in all the boxes on a new pallet. A third concern is that a person cannot switch back and forth between two pallets while they are filling up.

Is this a limitation that we need to document? Please share your thoughts about this.

If an on-change event handler or a save button is appropriate to save the data to the pallet_box record, then so be it.

If it would be simpler, perhaps the "Scan a Box" button and the two links "Select another pallet" and "Return to main page" could be overloaded to save any previous box data? Would that be easier?

Ideally the solution would include functionality to:

- Allows someone to stop working on a pallet and resume work later.
- Allows someone else to resume work on a pallet started by someone else.
- Allows multiple people to work on separate pallets.
- Allows multiple people to work on the same pallet.

However, current functionality is sufficient for initial production.

I-190 Build Pallet: set pallet ID in profile

Currently, Build Pallet creates a pallet record with a new pallet is chosen. It needs to also set the profile for this user to point to the pallet record just created.

If a pallet is selected in the first Build Pallet screen, the profile record needs to be updated immediately.

The reason this is needed is that various pages display the information about the active pallet – and sometimes the boxes associated with the pallet.

I-189 Ensure that the "Evans" box type is default

Ensure that any time a box is added to inventory (new box number, not just filled) that the "Evans" box type is at the top of the list. This will be the default box type for any new boxes.

Perhaps the default box type should be an entry in the constraints table so it is not hard-coded to the WARM food pantry.

The documentation should how to set this default.

- Perhaps as a default indicator so that it can be first regardless of sort order
- An alternate possibility would be to add a "sort order" field so all the box types can be shown from most frequently used to least frequent

I-178 Manual box checkin needs to be streamlined

Currently manual box checkin asks for box number, contents, etc. in separate screens. it needs to be changed so all the needed information if filled in on one screen.

- Add ability to specify start/end expiration months.

I-176 Change user password

Provide a way for any active user to change their password. Passwords must meet certain minimum requirements:

- Must be at least 8 characters in length
- Must have at least one alphanumeric character.
- If length is less than 12 characters, must have:
- A upper case character
- A lower case character
- A number
- Cannot contain any form of these words:
- WARM, password, 1234, the current month in letters or numbers

The screen must require both the valid old password and require the new password be typed in twice.

The screen should provide help about what a minimum password requires, except that it should only say that certain words are forbidden.

Access to this screen should require that the user login first.

- Perhaps use a library (like password-checker) to prevent short and easy to guess passwords and password permutations (e.g. changing the suffix to the next month name or number)

- If easy to do, provide a password expiration after some number of days (set in Constraints) but do not implement it for now.

I-175 Add/Change/Delete a User

Need code to add a user to the system.

- Must supply values for these fields:
- Assigned userid
- First and Last names
- Email address (optional)
- Initial password
- Title
- Role (group permission)

Need code to change a user

- Update all but the password

Need code to delete a user

- Change Active flag to effectively stop a user from accessing the system.

Add requires an initial password which user is required to change on first login

- Permitted roles (Permission levels)
- Staff can set staff or volunteer
- Admin can set any level

I-174 Delete a pallet

- Current workaround, complete the pallet with no boxes
- Possibly limited to staff

Provide a way to remove a pallet.

- Perhaps it is adequate to do this from the Admin menu.
- Ensure that all pallet_box records are also deleted – but without deleting the associated box records.
- Require that this can be done only by Staff level permissions or above.
- Document how to do this.
- If separate code, it can wait until after the first release.

If done via Admin, test it before going live.

I-173 Build Pallet needs to set pallet status

The Build Pallet screen and supporting code needs to set the pallet status to Pallet.FILL by the time the finish pallet action is taken.

- There is temporary code in BoxManagement.pallet_finish which needs to be removed after the above fix is in place.
- Probably a test needs to be added to verify that the pallet status is always set.

I-167 Results of Mike R.'s original testing

Attached is the results of Mike's initial testing. When you address any of these issues, please create a separate issue for it so it can be assigned to you. QWebPageCheck.pdf

Attached is a spreadsheet equivalent of the PDF attached above. QWebPageCheck.xlsx

I-161 Move Pallet

1. Enter "Move from" location using row, bin, and tier <select> elements. Upon submit display form with errors if the location doesn't exist or there are no boxes at that location.
2. If the "Move from" location exists and has boxes display row, bin, and tier <select> elements to select the "Move to" Location. This page will also list the boxes at the "Move from" location. Upon submit display form with errors if the "Move to" location doesn't exist
3. If there are boxes at the "Move to" location let the user know that and give them an option go back and select another location or continue. (continuing is merging pallets and that's ok)
4. Generic "N boxes moved from A to B" message.
5. Use BoxManagementClass.pallet_finish to update the database.

   Verify that the code sets appropriate pallet status before passing pallet to BoxManagement

I-145 Document clearing start/end exp. mo in Build Pallet

When using a QR code to scan a box, the product and expiration date will default to whatever the previous box contained – including the beginning and ending months. That is valid behavior. The product and expiration date can be adjusted as needed. However, if the previous box had a non-zero start and end month, the start and end month for this box cannot be set back to zero.

Although the month start and end cannot be set to zero, they can be blanked out (via backspace). Thus this is now just an issue that needs to be documented and closed.

- Alternately allow the start and end months to be reset to zero to clear

I-138 Enhance README.md with a more detailed purpose.

Add more comments about what the project does, how it plans to accomplish it, sample screen shots, etc. The goal is to provide a first-time visitor with a what they can expect to accomplish with this project.

Perhaps comments about how it could be extended to be used for other food pantries would be helpful.

I-115 Rework the API of theQR label print program

The current version of the QR label print program does not provide an easy API to be used by a web-based call or a GUI standalone program. It needs to be reworked so that a simple call can be made to get back the SVG XML. Perhaps it should also be able to provide a png on request.

Rather than implement the QR printing program as an API to the standalone program, this functionality will be moved to the online code. The reasons are given in issue #71.

Requirements: - Produce full pages of PDF formatted document to download and subsequently print. - if direct access to the printer is easy to implement, allow that as an option. - Ask for starting box number and count of labels desired

- Skip any box numbers that are already in the system.
- Ask if boxes are to be created in the system
  - If so, ask for the default box type.
- Optionally ask for a prefix to the box number
  - May need to prefix the box number with a URL for access to production
- Format box number with a mandatory “BOX” (in caps)
- Format the box number as a five digit number with leading zeros as needed.
- Assume:
  - letter size paper
    - portrait orientation
  - 1/2 inch outer margin on all sides
  - 3 labels across
  - 4 labels down
  - each label has 1/4 in margin on all sides
  - print in ascending order, left to right, top to bottom
  - leave unused label location completely blank
  - format pdf to print whole pages with no blank pages at the end

I-89 Improve README.md etc. for new developers

Revise the developer documentation from README.md on back to reflect current practices and procedures.

Improve the flow to the startup documentation so it is either all in the README.md or linked directly so that someone does not need to visit the wiki and then dig in the docs to find out how to set up their copy of the project.

I-88 Make a docker image of project to run locally

A Dockerfile needs to be made to build an image of the Inventory system so that it can be ran locally in a container. This ultimately might be expanded later to having the app and its components in containers.

One of the requirements of the docker image is a web server such as a Apache web server. This may mean one Docker container or perhaps up to three are needed.

- Database
- Application (Django)
- Web Server (e. g. Apache or equivalent)

This is to be decided after someone researches how best to configure Docker.

- Includes defining the docker containers and loading configurations, code and data
- Separate docker containers and configuration for PostgreSQL, Nginx, and Django with uWSGI.

Notes:

- Nginx was selected as the web server
- uWSGI was selected as the production Django interface to the web server

See also these issues:

- Issue #203 For configuring Docker for PostgreSQL
- Issue #204 For configuring Docker for Django and uWSGI
- Issue #205 For configuring Docker for Nginx

I-54 Product Examples Table - Build ACID screens

Recently we added a new table - product_examples. We need screens to list/add/change/delete the entries in this table. Each entry is associated with exactly one product.

More documentation about this table will be added in the next few days.

Build web pages that will allow an authorized user (i.e. staff or admin) to add, change and delete product example entries.

- Each product example entry must be associated with one (and only one) product.
- Changing a product example entry may include changing which product is associated with it.
- Currently only the roles staff or admin may access this set of screens.

See also issue #206 for how this table will be used.

I-42 Box Type Table - Build ACID screens

Build screens to list, add, change, and delete box types.

- Include requirement that the user must be logged in before these screens can be used.
- Add link to index page to get to the list screen.

Build web pages that will allow an authorized user to add, change and delete box type entries.

- Deleting a box type will not be permitted if there are any box entries referencing this box type.

This could be done using the admin screens. Create a BoxTypeAdmin class and register it in fpiweb/admin.py. There are a couple examples in the admin.py or you can check the documentation here. In addition to list_display you might want to consider other ModelAdmin options. The Django permission system provides a lot of granularity to control who can do what.

I-41 Product Table - build ACID screens

Build screens to list, add, change, and delete products.

- Include requirement that the user must be logged in to use these screens.
- Add link to index page to get to the list screen.

Build web pages that will allow an authorized user to add, change and delete product entries.

- Each product entry must be associated with one (and only one) product category.
- Changing a product entry may include changing which which product category is associated with it.
- Deleting a product entry will be permitted only if it has no associated product example entries.

Currently handled by the Admin menus.

I-40 Product Category Table - build ACID screens

Need screens to list, add, change, and delete product categories.

- Include requirement that the user must be logged in to use these screens.
- Include link on index page to get to the list screen.

This functionality is currently provided by the Admin menus.

3.9 Getting Started

This project is based on using PyCharm as the IDE, Django as the web framework, and PostgreSQL as the database backend for development.

3.9.1 Contributing to this Project

In order for you to contribute to this project, you will need to spend some time setting up your environment.

PyCharm

PyCharm is available from JetBrains (http://www.jetbrains.com) in two editions - Professional($) and Community(Free). This project only requires the Community edition, althogh it will work with the Professional edition if you are lucky enough to have it.

GitHub

This project requires you to have a GitHub account.

Django

The Django web framework code will be downloaded as part of the project code.

3.9.2 Starting Point

Please start with the Git Guidelines.

3.10 General Developer Documentation

Helpful *Developer Resources* are available.

3.11 Django Documentation

Documentation about Django, including documentation specific to this project.

Django

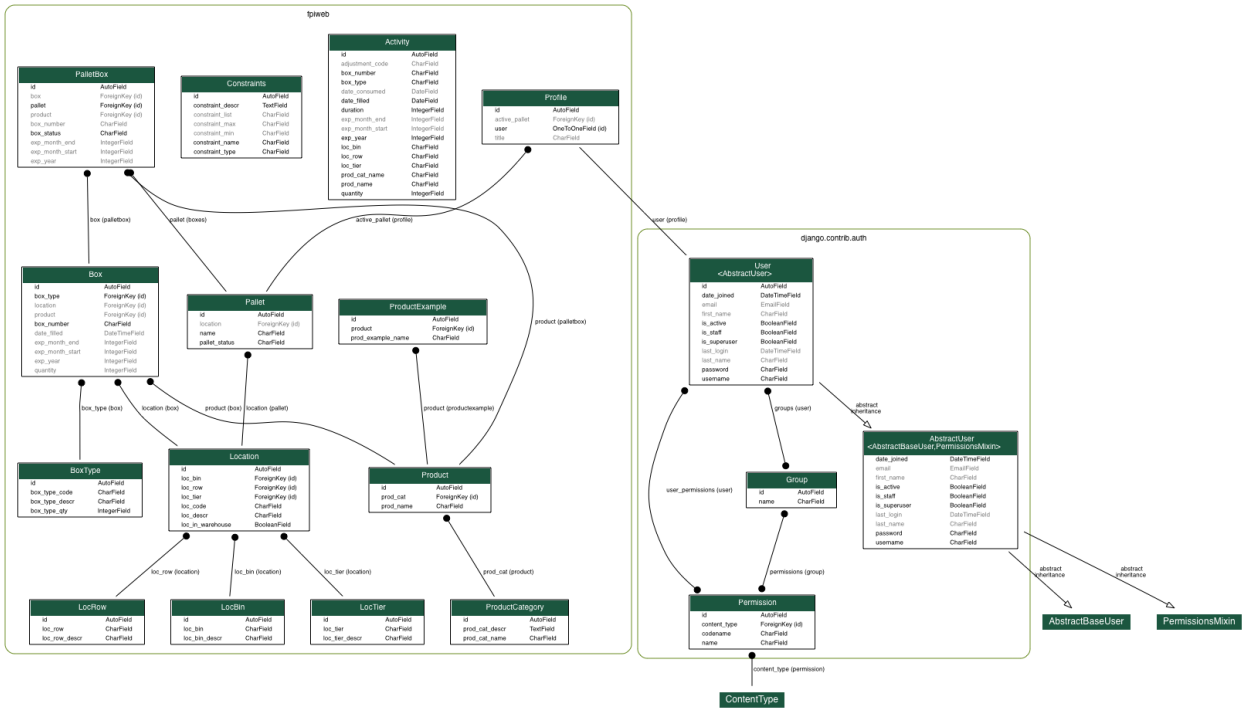
Django Guidelines

3.12 PostgreSQL Documentation

Documentation about PostgreSQL that is specific to this project.

PostgreSQL Guidelines

A graph of the database tables and their relationships is shown here.



A bigger, sharper image of this graph can be downloaded from here.

`Inventory Tables and Relationships`

3.13 Git Documentation

Documentation about git (the source code control software) that is specific to this project.

Git Guidelines

3.14 Internal Documentation

Javascript and JQuery information specific to this project.

Javascript and JQuery

How information about boxes and their workflows.

Box Management

3.15 Deployment Documentation

This system is planned to be deployed in a "cloud" platform. See *Deployment Documentation* for both an overview of the deployment plans and details as they are implemented.

3.16 Other Developer Resources

Additional developer resources:

Additional Developer Resources

CHAPTER

FOUR

TECHNICAL DOCUMENTATION

4.1 FPIDjango

4.1.1 FPIDjango package

Subpackages

FPIDjango.private package

Submodules

FPIDjango.private.settings_private module

settings_private.py - Shadow or pseudo-private file.

This file has dummy settings in it. The purpose is to show the format of your real settings_private file in the private subdirectory.

The files at this level are dummy files that are safe to upload to GitHub. The equivalent files in the private subdirectory are ignored by git so it is safe to put your sensitive (and really private) parameters in those files.

When you run Django on your system for real, change the environment variable for DJANGO_SETTINGS_MODULE from FPIDjango.settings to FPIDjango.private.settings.

Submodules

FPIDjango.settings module

Django settings for FPIDjango project.

Unfortunately, the modified import statement of

    from FPIDjango.private.settings_private import *

is required for the PyCharm Python Console and the PyCharm manage.py to work properly. It obviously knows better than I do. It cannot pay attention to some silly environment variable that I set in Preferences like we were able to do in the run settings.

This will work – and not reveal our credentials – as long as the **private** directory is included in our **.gitignore** file.

(Comments autogenerated by Django) Generated by 'django-admin startproject' using Django 2.1.7.

For more information on this file, see https://docs.djangoproject.com/en/2.1/topics/settings/

For the full list of settings and their values, see https://docs.djangoproject.com/en/2.1/ref/settings/

FPIDjango.settings_private module

settings_private.py - Shadow or pseudo-private file.

This file has dummy settings in it. The purpose is to show the format of your real settings_private file in the private subdirectory.

The files at this level are dummy files that are safe to upload to GitHub. The equivalent files in the private subdirectory are ignored by git so it is safe to put your sensitive (and really private) parameters in those files.

When you run Django on your system for real, change the environment variable for DJANGO_SETTINGS_MODULE from FPIDjango.settings to FPIDjango.private.settings.

FPIDjango.urls module

FPIDjango URL Configuration

The *urlpatterns* list routes URLs to views. For more information please see: https://docs.djangoproject.com/en/2.1/topics/http/urls/

Examples: Function views

1. Add an import: from my_app import views
2. Add a URL to urlpatterns: path('', views.home, name='home')

Class-based views

1. Add an import: from other_app.views import Home
2. Add a URL to urlpatterns: path('', Home.as_view(), name='home')

Including another URLconf

1. Import the include() function: from django.urls import include, path
2. Add a URL to urlpatterns: path('blog/', include('blog.urls'))

FPIDjango.wsgi module

WSGI config for FPIDjango project.

It exposes the WSGI callable as a module-level variable named `application`.

For more information on this file, see https://docs.djangoproject.com/en/2.1/howto/deployment/wsgi/

4.2 fpiweb

4.2.1 fpiweb package

Subpackages

fpiweb.fpiweb_views package

Submodules

fpiweb.fpiweb_views.PrintLabelView module

PrintLabelView.py - manage the view to print QR codes on labels or paper.

class fpiweb.fpiweb_views.PrintLabelView.**LabelPosition**(*page_offset: dataclasses.InitVar[Point]*, *lower_left_offset: fpiweb.fpiweb_views.PrintLabelView.Point = Point(x=0, y=0)*, *lower_right_offset: fpiweb.fpiweb_views.PrintLabelView.Point = Point(x=0, y=0)*, *upper_left_offset: fpiweb.fpiweb_views.PrintLabelView.Point = Point(x=0, y=0)*, *upper_right_offset: fpiweb.fpiweb_views.PrintLabelView.Point = Point(x=0, y=0)*, *offset_on_page: fpiweb.fpiweb_views.PrintLabelView.Point = Point(x=0, y=0)*, *image_start: fpiweb.fpiweb_views.PrintLabelView.Point = Point(x=0, y=0)*, *title_start: fpiweb.fpiweb_views.PrintLabelView.Point = Point(x=0, y=0)*)

Bases: object

Container for measurements for one label.

All measurements are in points. x denotes horizontal measurement y denotes vertical origin is in lower left corner label is assumed to be 2 in x 2 in (144 pt x 144 pt)

image_start: *fpiweb.fpiweb_views.PrintLabelView.Point* = Point(x=0, y=0)

lower_left_offset: *fpiweb.fpiweb_views.PrintLabelView.Point* = Point(x=0, y=0)

lower_right_offset: *fpiweb.fpiweb_views.PrintLabelView.Point* = Point(x=0, y=0)

offset_on_page: *fpiweb.fpiweb_views.PrintLabelView.Point* = Point(x=0, y=0)

page_offset: dataclasses.InitVar[*Point*]

title_start: *fpiweb.fpiweb_views.PrintLabelView.Point* = Point(x=0, y=0)

upper_left_offset: *fpiweb.fpiweb_views.PrintLabelView.Point* = Point(x=0, y=0)

upper_right_offset: *fpiweb.fpiweb_views.PrintLabelView.Point* **= Point(x=0, y=0)**

class fpiweb.fpiweb_views.PrintLabelView.**Point**(*x: int*, *y: int*)

Bases: object

Horizontal (x) and vertical (y) coordinate.

x: int

y: int

class fpiweb.fpiweb_views.PrintLabelView.**PrintLabelForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *field_order=None*, *use_required_attribute=None*, *renderer=None*)

Bases: django.forms.forms.Form

Form to request number of labels and starting number.

base_fields = {'number_to_print': <django.forms.fields.IntegerField object>, 'startin

declared_fields = {'number_to_print': <django.forms.fields.IntegerField object>, 'sta

property media

Return all media required to render the widgets on this form.

number_to_print: int

starting_number: int

class fpiweb.fpiweb_views.PrintLabelView.**PrintLabelView**(***kwargs*)

Bases: django.contrib.auth.mixins.PermissionRequiredMixin, django.views.generic.base.View

Manage the request for starting number and count of QR code to print.

form_class

alias of *fpiweb.fpiweb_views.PrintLabelView.PrintLabelForm*

get(*request*, **args*, ***kwargs*)

Prepare to display request for starting box number and count.

Parameters

- **request** –
- **args** –
- **kwargs** –

Returns

static get_base_url(*meta*) → str

Determine the URL prefix to add to each QR code for a box.

Modify this code as needed.

Parameters **meta** –

Returns

permission_required = ('fpiweb.print_labels_box',)

post(*request*, **args*, ***kwargs*)
Validate the info returned from the user and generate the QR codes.

Parameters

- **request** –
- **args** –
- **kwargs** –

Returns

success_url = '/fpiweb/index/'

template_name = 'fpiweb/print_labels.html'

class fpiweb.fpiweb_views.PrintLabelView.**QRCodePrinter**(*url_prefix: str*)
Bases: object

Write the pdf of QR codes into the file or byte buffer provided.

compute_box_dimensions()
Compute the dimensions and bounding boxes for each label on the page.

Called from __init__

Returns

draw_bounding_box(*label_pos: int*)
Draw a bounding box around the specified label.

Parameters **label_pos** – position in the labels locations list.

Returns

fill_pdf_pages(*starting_number: int*, *count: int*)
Fill one or more pages with labels.

draw lines around the boxes that will be filled with labels # self.draw_boxes_on_page() # self.pdf.setFillColorRGB(1, 0, 1) # self.pdf.rect(2*inch, 2*inch, 2*inch, 2*inch, fill=1)

Returns

finalize_pdf_file()
All pages have been generated so flush all buffers and close.

Returns

finish_page()
Finish off the previous page before starting a new one.

generate_label_pdf()
Generate the pdf file with the requested labels in it.

Returns

static get_next_box_number(*start*, *count) -> (<class 'str'>*, *<class 'int'>*)
Search for the next box number to go on a label.

Returns

get_next_box_url(*start_number: int*, *count: int) -> (<class 'str'>*, *<class 'str'>*)
Build the URL for the next box.

Returns

get_next_qr_img(*start_number: int*, *count: int) -> (<class 'str'>*, *<class 'str'>*)
Build the QR image for the next box label.

Returns a QR code image file name and the prefixed box number

initialize_pdf_file(*buffer: _io.BytesIO*)
Prepare to scribble on a new pdf file.

Parameters **buffer** – May be a string with a filename or a BytesIO or other File-like object

place_label(*file_name: str*, *label_name: str*, *pos: int*)
Place the label in the appropriate location on the page.

Parameters

- **file_name** –
- **label_name** –
- **pos** –

Returns

print(*starting_number: int*, *count: int*, *buffer*)
Starting point once view retrieved the requested info.

Parameters

- **starting_number** – starting box number without the BOX prefix
- **count** – number of labels desired
- **buffer** – byte buffer to write pdf bytes into

Returns

fpiweb.fpiweb_views.PrintLabelView.**logger = <Logger fpiweb (INFO)>**
Assuming: - letter size paper

- portrait orientation
- 1/2 inch outer margin on all sides
- all measurements in points (1 pt = 1/72 in)
- 3 labels across
- 4 labels down
- each label has 1/4 in margin on all sides
- 0, 0 of axis is in lower left corner

fpiweb.migrations package

Submodules

fpiweb.migrations.0001_initial module

class fpiweb.migrations.0001_initial.**Migration**(*name*, *app_label*)
Bases: django.db.migrations.migration.Migration

dependencies = []

initial = True

operations = [<CreateModel name='Activity', fields=[('Activity_ID', <django.db.models.

fpiweb.migrations.0002_auto_20190324_0332 module

class fpiweb.migrations.0002_auto_20190324_0332.**Migration**(*name*, *app_label*)
Bases: django.db.migrations.migration.Migration

dependencies = [('fpiweb', '0001_initial')]

operations = [<AddField model_name='activity', name='Duration', field=<django.db.model

fpiweb.migrations.0003_constraints_constraintdescr module

class fpiweb.migrations.0003_constraints_constraintdescr.**Migration**(*name*, *app_label*)
Bases: django.db.migrations.migration.Migration

dependencies = [('fpiweb', '0002_auto_20190324_0332')]

operations = [<AddField model_name='constraints', name='ConstraintDescr', field=<django

fpiweb.migrations.0004_auto_20190330_0215 module

class fpiweb.migrations.0004_auto_20190330_0215.**Migration**(*name*, *app_label*)
Bases: django.db.migrations.migration.Migration

dependencies = [('fpiweb', '0003_constraints_constraintdescr')]

operations = [<AlterModelOptions name='activity', options={'ordering': ['DateConsumed

fpiweb.migrations.0005_auto_20190403_0205 module

class fpiweb.migrations.0005_auto_20190403_0205.**Migration**(*name*, *app_label*)
Bases: django.db.migrations.migration.Migration

dependencies = [('fpiweb', '0004_auto_20190330_0215')]

operations = [<AlterModelOptions name='activity', options={'ordering': ['date_consume

fpiweb.migrations.0006_auto_20190406_1711 module

class fpiweb.migrations.0006_auto_20190406_1711.**Migration**(*name*, *app_label*)
Bases: django.db.migrations.migration.Migration

dependencies = [('fpiweb', '0005_auto_20190403_0205')]

operations = [<RenameField model_name='Constraints', old_name='constraint_id', new_nam

fpiweb.migrations.0007_auto_20190406_1717 module

class fpiweb.migrations.0007_auto_20190406_1717.**Migration**(*name*, *app_label*)
Bases: django.db.migrations.migration.Migration

dependencies = [('fpiweb', '0006_auto_20190406_1711')]

operations = [<AlterField model_name='box', name='product', field=<django.db.models.fi

fpiweb.migrations.0008_auto_20190406_1737 module

class fpiweb.migrations.0008_auto_20190406_1737.**Migration**(*name*, *app_label*)
Bases: django.db.migrations.migration.Migration

dependencies = [('fpiweb', '0007_auto_20190406_1717')]

operations = [<RenameField model_name='Activity', old_name='activity_id', new_name='id

fpiweb.migrations.0009_auto_20190410_0118 module

class fpiweb.migrations.0009_auto_20190410_0118.**Migration**(*name*, *app_label*)
Bases: django.db.migrations.migration.Migration

dependencies = [('fpiweb', '0008_auto_20190406_1737')]

operations = [<AlterModelOptions name='activity', options={'ordering': ['-date_consum

fpiweb.migrations.0010_auto_20190415_0404 module

class fpiweb.migrations.0010_auto_20190415_0404.**Migration**(*name*, *app_label*)
Bases: django.db.migrations.migration.Migration

dependencies = [('fpiweb', '0009_auto_20190410_0118')]

operations = [<CreateModel name='ProductExample', fields=[('id', <django.db.models.fie

fpiweb.migrations.0011_auto_20190415_0430 module

class fpiweb.migrations.0011_auto_20190415_0430.**Migration**(*name*, *app_label*)
Bases: django.db.migrations.migration.Migration

dependencies = [('fpiweb', '0010_auto_20190415_0404')]

operations = [<AlterModelOptions name='productexample', options={'ordering': ['prod_e

fpiweb.migrations.0012_auto_20190419_2341 module

class fpiweb.migrations.0012_auto_20190419_2341.**Migration**(*name*, *app_label*)
Bases: django.db.migrations.migration.Migration

dependencies = [('fpiweb', '0011_auto_20190415_0430')]

operations = [<AlterField model_name='box', name='exp_month_end', field=<django.db.mod

fpiweb.migrations.0013_auto_20190420_1612 module

class fpiweb.migrations.0013_auto_20190420_1612.**Migration**(*name*, *app_label*)
Bases: django.db.migrations.migration.Migration

dependencies = [('fpiweb', '0012_auto_20190419_2341')]

operations = [<RenameField model_name='Activity', old_name='box_type_code', new_name='

fpiweb.migrations.0014_auto_20190523_2043 module

class fpiweb.migrations.0014_auto_20190523_2043.**Migration**(*name*, *app_label*)
Bases: django.db.migrations.migration.Migration

dependencies = [('fpiweb', '0013_auto_20190420_1612')]

operations = [<AlterModelOptions name='activity', options={'ordering': ['-date_consum

fpiweb.migrations.0015_box_print_box_number_label module

class fpiweb.migrations.0015_box_print_box_number_label.**Migration**(*name*, *app_label*)
Bases: django.db.migrations.migration.Migration

dependencies = [('fpiweb', '0014_auto_20190523_2043')]

operations = [<AddField model_name='box', name='print_box_number_label', field=<django

fpiweb.migrations.0016_remove_box_print_box_number_label module

class fpiweb.migrations.0016_remove_box_print_box_number_label.**Migration**(*name*, *app_label*)
Bases: django.db.migrations.migration.Migration

dependencies = [('fpiweb', '0015_box_print_box_number_label')]

operations = [<RemoveField model_name='box', name='print_box_number_label'>]

fpiweb.migrations.0017_location_locbin_locrow_loctier_profile module

class fpiweb.migrations.0017_location_locbin_locrow_loctier_profile.**Migration**(*name*, *app_label*)

Bases: django.db.migrations.migration.Migration

dependencies = [('auth', '__first__'), ('fpiweb', '0016_remove_box_print_box_number_la

operations = [<CreateModel name='LocBin', fields=[('id', <django.db.models.fields.Auto

fpiweb.migrations.0018_profile_active_location module

class fpiweb.migrations.0018_profile_active_location.**Migration**(*name*, *app_label*)

Bases: django.db.migrations.migration.Migration

dependencies = [('fpiweb', '0017_location_locbin_locrow_loctier_profile')]

operations = [<AddField model_name='profile', name='active_location', field=<django.db

fpiweb.migrations.0019_pallet_tables_and_tweaks module

class fpiweb.migrations.0019_pallet_tables_and_tweaks.**Migration**(*name*, *app_label*)

Bases: django.db.migrations.migration.Migration

dependencies = [('auth', '__first__'), ('fpiweb', '0018_profile_active_location')]

operations = [<CreateModel name='Pallet', fields=[('id', <django.db.models.fields.Auto

fpiweb.migrations.0020_auto_20191021_2016 module

class fpiweb.migrations.0020_auto_20191021_2016.**Migration**(*name*, *app_label*)

Bases: django.db.migrations.migration.Migration

dependencies = [('fpiweb', '0019_pallet_tables_and_tweaks')]

operations = [<AlterField model_name='activity', name='date_consumed', field=<django.d

fpiweb.migrations.0021_auto_20191022_2113 module

class fpiweb.migrations.0021_auto_20191022_2113.**Migration**(*name*, *app_label*)

Bases: django.db.migrations.migration.Migration

dependencies = [('fpiweb', '0020_auto_20191021_2016')]

operations = [<AlterField model_name='profile', name='user', field=<django.db.models.f

fpiweb.migrations.0022_consolidate_location_fields module

class fpiweb.migrations.0022_consolidate_location_fields.**Migration**(*name*, *app_label*)

Bases: django.db.migrations.migration.Migration

dependencies = [('fpiweb', '0021_auto_20191022_2113')]

operations = [<RemoveField model_name='box', name='loc_bin'>, <RemoveField model_name=

fpiweb.migrations.0023_auto_20200102_2045 module

class fpiweb.migrations.0023_auto_20200102_2045.**Migration**(*name*, *app_label*)

Bases: django.db.migrations.migration.Migration

dependencies = [('fpiweb', '0022_consolidate_location_fields')]

operations = [<AlterField model_name='box', name='exp_month_end', field=<django.db.mod

fpiweb.migrations.0024_change_pallet_handling module

class fpiweb.migrations.0024_change_pallet_handling.**Migration**(*name*, *app_label*)

Bases: django.db.migrations.migration.Migration

dependencies = [('fpiweb', '0023_auto_20200102_2045')]

operations = [<AlterModelOptions name='pallet', options={'ordering': ('name',), 'verb

fpiweb.migrations.0025_restore_pallet_box_number_tweak_activity module

class fpiweb.migrations.0025_restore_pallet_box_number_tweak_activity.**Migration**(*name*, *app_label*)

Bases: django.db.migrations.migration.Migration

dependencies = [('fpiweb', '0024_change_pallet_handling')]

operations = [<AddField model_name='palletbox', name='box_number', field=<django.db.mo

fpiweb.migrations.0026_auto_20200210_2015 module

class fpiweb.migrations.0026_auto_20200210_2015.**Migration**(*name*, *app_label*)

Bases: django.db.migrations.migration.Migration

dependencies = [('fpiweb', '0025_restore_pallet_box_number_tweak_activity')]

operations = [<AlterField model_name='box', name='location', field=<django.db.models.f

fpiweb.migrations.0027_set_constraint_name_choices module

class fpiweb.migrations.0027_set_constraint_name_choices.**Migration**(*name*, *app_label*)

Bases: django.db.migrations.migration.Migration

dependencies = [('fpiweb', '0026_auto_20200210_2015')]

operations = [<AlterField model_name='constraints', name='constraint_name', field=<dja

fpiweb.migrations.0028_add_activity_adj_code module

class fpiweb.migrations.0028_add_activity_adj_code.**Migration**(*name*, *app_label*)

Bases: django.db.migrations.migration.Migration

dependencies = [('fpiweb', '0027_set_constraint_name_choices')]

operations = [<AlterModelOptions name='location', options={'verbose_name_plural': 'Lo

fpiweb.migrations.0029_add_model_permissions module

class fpiweb.migrations.0029_add_model_permissions.**Migration**(*name*, *app_label*)

Bases: django.db.migrations.migration.Migration

dependencies = [('fpiweb', '0028_add_activity_adj_code')]

operations = [<AlterModelOptions name='box', options={'ordering': ['box_number'], 'pe

fpiweb.migrations.0030_AddPermissionData module

class fpiweb.migrations.0030_AddPermissionData.**Migration**(*name*, *app_label*)

Bases: django.db.migrations.migration.Migration

dependencies = [('fpiweb', '0029_add_model_permissions'), ('auth', '0012_alter_user_fi

operations = [<RunPython <function setup_groups_and_permissions>>]

fpiweb.migrations.0030_AddPermissionData.**iterate_permissions**(*permissions*)

Generator to pick up the needed foreign keys for a given permission.

Parameters **permissions** – attributes associated with this permission

Returns

fpiweb.migrations.0030_AddPermissionData.**setup_group_permissions**(*group*, *permissions*)

For a given group, create or replace the permissions for it.

Parameters

- **group** – group to be created or replaced
- **permissions** – list of permissions to be applied to this group

Returns

fpiweb.migrations.0030_AddPermissionData.**setup_groups_and_permissions**(**args*, ***kwargs*)

Create or replace groups and add permissions.

Parameters **groups_and_permissions** – table of groups and associated permissions

Returns

fpiweb.support package

Submodules

fpiweb.support.BoxActivity module

BoxActivity.py - Record activity for changes to a box.

Error messages from this module are prefixed by 2nn, e.g. "201 - blah blah..."

class fpiweb.support.BoxActivity.**BOX_ACTION**(*value*)
Bases: enum.Enum

Actions to be applied to a box.

ADD: str = 'add'

EMPTY: str = 'empty'

FILL: str = 'fill'

MOVE: str = 'move'

class fpiweb.support.BoxActivity.**BoxActivityClass**
Bases: object

BoxManagementClass - Manage db for changes to a box.

I decided that (for now) empty boxes should not be added to the activity records. Activity records will show only when a box was filled or emptied. The activity records for filled boxes will show only their current location. Activity records for consumed boxes will show their last location. (tr)

For now, the activity records will not show empty boxes, damaged or discarded boxes removed from the inventory system.

box_empty(*box_id: <django.db.models.query_utils.DeferredAttribute object at 0x7fcbf43a4430>*)
Record activity for a box being emptied (consumed).

This method expects the box record to still have the location, product, etc. information still in it. After recording the appropriate information in the activity record, this method will clear out the box so it will be empty again.

Parameters **box_id** –

Returns

box_fill(*box_id: <django.db.models.query_utils.DeferredAttribute object at 0x7fcbf43a4430>*)
Record activity for a box being filled and added to inventory.

This method expects that the box record already has the box type. location, product, and expiration date filled in.

This method will write a new activity record that "starts the clock" for this box. If the box was already marked as checked in to inventory, the previous contents will be checked out and the new contents checked in.

Parameters **box_id** – internal box ID of box being added to inventory

Returns

box_move(*box_id: <django.db.models.query_utils.DeferredAttribute object at 0x7fcbf43a4430>*)
Record activity for a box being moved in tne inventory.

This method expects that the box record already has the box type. location, product, and expiration date filled in.

This method will change the current location of the box in the activity record. The old location will not be retained nor will any "clocks" for the activity record be updated.

If the box does not have an open activity record, a new one will be created.

Parameters **box_id** – internal box ID of box being moved

Returns

box_new(*box_id: <django.db.models.query_utils.DeferredAttribute object at 0x7fcbf43a4430>*)
Record that a new (empty) box has been added to inventory.

Parameters **box_id** – internal box ID of box being added to inventory

Returns

compute_duration_days(*date_filled: datetime.date*) → tuple
compute the days between the date filled and today

Parameters **date_filled** –

Returns tuple of date consumed and number of days in box

fpiweb.support.BoxManagement module

BoxManagement.py - Manage box creating, filling, moving and emptying.

Error messages from this module are prefixed by 1nn, e.g. "101 - blah blah..."

class fpiweb.support.BoxManagement.**BoxManagementClass**
Bases: object

BoxManagementClass - Manage db changes for box changes.

The public API's validate parameter values. The private methods perform the actual actions.

box_consume(*box: Union[*fpiweb.models.Box*, int]*) → *fpiweb.models.Box*
Consume (e.g. empty) a box. The box will be marked empty, the activity record will be updated, and the box will be returned.

Requirements:

- The box must not be empty when passed in.

Exception:

131 - Box not in system

132 - The box was already empty

Parameters **box** –

Returns the box record freshly emptied

box_fill(**, box: Union[*fpiweb.models.Box*, int]*, *location: Union[*fpiweb.models.Location*, int]*, *product: Union[*fpiweb.models.Product*, int]*, *exp_year: int*, *exp_mo_start: int = 0*, *exp_mo_end: int = 0*)
Fill an individual box with product and add to the inventory. If the box is not empty, an activity record

will empty the box of its previous contents and a new activity record will note the new contents profiled. If successful, it will return the box just filled.

Requirements:

- Box record has not been modified
- All required fields are valid
- Optional month start and end, if specified, bracket one or more months

Exceptions:

111 - Attempting to fill a box that does not exist

112 - location supplied is not valid

113 - the product supplied is not valid

114 - the expiration year, start month, and/or end month are not valid or are out of range

Parameters

- **box** – Box record or id of a box already in the system
- **location** – Target location record or ID
- **product** – Target product record or ID
- **exp_year** – year (current year +/- 10)
- **exp_mo_start** – 1 - 12 if specified - usually beginning quarter
- **exp_mo_end** – 1-12 if specified - usually ending quarter

Returns box record after modifications

box_move(*box: Union[*fpiweb.models.Box*, int]*, *location: Union[*fpiweb.models.Location*, int]*)

Move an individual box in the inventory. The activity record for this box will be changed to show the new location. The old location will be dropped from the activity record. If successful, the box record will be returned.

Requirements:

- Box is filled
- Location is valid

Exceptions:

121 - Box not in system

122 - Cannot move an empty box

123 - Location is not valid

Parameters

- **box** –
- **location** –

Returns

box_new (*box_number: str*, *box_type: Union[str, int,* fpiweb.models.BoxType*]*) → *fpiweb.models.Box*
Add a new empty box to the inventory system. If successful it adds an activity record for an empty box and returns the box record just created.

Requirements:

- Box number is valid, unique, and not previously assigned
- Box type is valid

Exceptions:

101 - Box number supplied is not in the valid format ('BOXnnnn')

102 - Box number is not unique

102 - Box type is not valid

Parameters

- **box_number** – in the form of 'BOXnnnnn'
- **box_type** – a valid box type code, BoxType record or record id

Returns the newly created box record

pallet_finish (*pallet: Union[*fpiweb.models.Pallet*, int, str]*)
Finish the processing of a pallet of boxes into inventory. Each box of the pallet will be processed. Nothing will be returned.

Note - a pallet is still considered valid even if there are no boxes associated with it.

Requirements:

- A valid pallet record, pallet name, or ID
- A pallet status indicating if the boxes have just been filled or are being moved to a new location

Exception:

161 - An invalid pallet ID was passed in

162 - An invalid pallet name was passed in

166 - The pallet has an invalid location

Parameters **pallet** –

Returns

fpiweb.support.PermissionsManagement module

PermissionsManagement.py - Manage user permissions

class fpiweb.support.PermissionsManagement.**ManageUserPermissions**
Bases: object

Manage user permissions

add_a_user (*target_user:* fpiweb.constants.TargetUser) → django.contrib.auth.models.User
Add a new user account to the system.

Parameters **target_user** –

Returns

change_a_user(*userid: int*, *target_user:* fpiweb.constants.TargetUser) → django.contrib.auth.models.User
Change an attribute of a user account

Parameters

- **userid** – key of existing user record
- **target_user** – all potential changes to the user and profile

Returns modified user recors from db

get_highest_access_level(*user*) → *fpiweb.constants.AccessLevel*
Determine the highest level of access this user currently has.

Parameters **user** – an auth_user record

Returns access level for this user

get_user_info(*user_id: int*) → *fpiweb.constants.UserInfo*
Return full user info for a given user.

Parameters **user_id** – User record or user id

Returns Full user info

log_change(*username: str*, *field: str*, *old: str*, *new: str*)
write an entry to the log about the change made.

Parameters

- **username** – the userid of the person being changed
- **field** – the descriptive name of the field being change
- **old** –
- **new** –

Returns

fpiweb.templatetags package

Submodules

fpiweb.templatetags.form module

fpiweb.templatetags.form.**initial_or_cleaned**(*form*, *field_name*)

fpiweb.templatetags.qr_codes module

fpiweb.templatetags.qr_codes.**qr_code**(*data_string*)
Interface to allow a URL to be converted into a QR code.

Parameters **data_string** – URL to be conveted

Returns image of QR code

fpiweb.tests package

Submodules

fpiweb.tests.AddPermissionData module

fpiweb.tests.AddPermissionData.**iterate_permissions**(*permissions*)
Generator to pick up the needed foreign keys for a given permisison.

Parameters **permissions** – attributes associated with this permission

Returns

fpiweb.tests.AddPermissionData.**setup_group_permissions**(*group*, *permissions*)
For a given group, create or replace the permisisons for it.

Parameters

- **group** – group to be created or replaced
- **permissions** – list of permissions to be applied to this group

Returns

fpiweb.tests.AddPermissionData.**setup_groups_and_permissions**(**args*, ***kwargs*)
Create or replace groups and add permissions.

Parameters **groups_and_permissions** – table of groups and associated permissions

Returns

fpiweb.tests.rework_func_ManualBoxManagement module

class fpiweb.tests.rework_func_ManualBoxManagement.**ManualBoxManagement**(*methodName='runTest'*)
Bases: django.contrib.staticfiles.testing.StaticLiveServerTestCase

HEADLESS_MODE = True

RECORD = False

delay_for_recording()

fixtures = ['BoxType.json', 'LocBin.json', 'LocRow.json', 'LocTier.json', 'Location.js

classmethod get_browser_mode()

select_random_dropdown(*dropdown_int*)

setUp()
Prepare for StaticLiveServerTestCase :return:

classmethod setUpClass()
Hook method for setting up class fixture before running tests in the class.

set_location_test()

classmethod tearDownClass()
Hook method for deconstructing the class fixture after running all tests in the class.

test_1LogIn()

test_2ManualBoxPalletManagementPage()

test_3ManualStatusBox()

test_4AddNewBox()

test_5CheckinBox()

test_6Checkout_a_box()

test_7Move_a_box()

test_user = ''

fpiweb.tests.rework_func_ManualPalletManagement module

class fpiweb.tests.rework_func_ManualPalletManagement.**ManualPalletMaintenance**(*methodName='run*

Bases: django.contrib.staticfiles.testing.StaticLiveServerTestCase

HEADLESS_MODE = True

RECORD = False

START_LOCATION = True

delay_for_recording()

fixtures = ['BoxType.json', 'LocBin.json', 'LocRow.json', 'LocTier.json', 'Location.js

classmethod run_headless_mode()

select_random_dropdown(*dropdown_int*)

setUp()
Hook method for setting up the test fixture before exercising it.

classmethod setUpClass()
Hook method for setting up class fixture before running tests in the class.

set_pallet_location(*row*, *bin*, *tier*, *start_location*)

classmethod tearDownClass()
Hook method for deconstructing the class fixture after running all tests in the class.

test_1A_Move_a_pallet()

test_1B_MovePallet()

test_1C_MovePallet()

test_user = ''

fpiweb.tests.rework_func_UserManagement module

class fpiweb.tests.rework_func_UserManagement.**UserManagementTest**(*methodName='runTest'*)

Bases: django.contrib.staticfiles.testing.StaticLiveServerTestCase

HEADLESS_MODE = True

RECORD = True

delay_for_recording()

fixtures = ['Activity.json', 'Constraints.json', 'Group.json', 'PalletBox.json', 'BoxT

classmethod get_browser_mode()

setUp()
Hook method for setting up the test fixture before exercising it.

classmethod setUpClass()
Hook method for setting up class fixture before running tests in the class.

classmethod tearDownClass()
Hook method for deconstructing the class fixture after running all tests in the class.

test_UserManagement()

fpiweb.tests.test_forms module

class fpiweb.tests.test_forms.**BoxItemFormTest**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

Test adding boxes to the pallet form.

fixtures = ('BoxType', 'Product', 'ProductCategory', 'Constraints')

test_box_number_validation()

test_expire_months()
Ensure that start month <= end month

class fpiweb.tests.test_forms.**BuildPalletFormTest**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

Test the form for building a pallet of boxes.

test_is_valid__location_not_specified()

class fpiweb.tests.test_forms.**ConfirmMergeFormTest**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

fixtures = ('Location', 'LocBin', 'LocRow', 'LocTier')

test_boxes_at_location_int()

test_is_valid()

test_to_location_str()

class fpiweb.tests.test_forms.**ExistingLocationFormTest**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

fixtures = ('LocRow', 'LocBin', 'LocTier', 'Location')

test_clean__multiple_locations_found()

test_clean__nonexistent_location()

test_clean__successful_run()

class fpiweb.tests.test_forms.**ExistingLocationWithBoxesFormTest**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

fixtures = ('BoxType', 'LocRow', 'LocBin', 'LocTier', 'Location')

test_clean()

class fpiweb.tests.test_forms.**LocationFormTest**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

fixtures = ('LocRow', 'LocBin', 'LocTier')

test_is_valid__missing_value()

class fpiweb.tests.test_forms.**MoveToLocationFormTest**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

fixtures = ('Location', 'LocBin', 'LocRow', 'LocTier')

test_is_valid()
MoveToLocationForm adds the from_location field to the ExistingLocationForm so we only look at how from_location effects validation. :return: None

class fpiweb.tests.test_forms.**NewBoxFormTest**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

Test creating a new box number not previously in inventory.

fixtures = ('BoxType', 'Constraints')

test_save()
Test saving a new box number.

Returns

fpiweb.tests.test_models module

class fpiweb.tests.test_models.**BoxTest**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

fixtures = ('ProductCategory', 'BoxType', 'Product')

static test_add() → None

fpiweb.tests.test_password_validation module

class fpiweb.tests.test_password_validation.**PasswordValidationTest**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

test_password_validation()

static validate(*password*, *user=None*)

fpiweb.tests.test_setup module

test_setup.py - Activities required before testing can begin.

fpiweb.tests.test_setup.**add_permission_data**()
Define a fixtureto add the permission data.

Returns

fpiweb.tests.test_setup.**setup_module**()
Setup activities required before a test can begin.

Returns

fpiweb.tests.test_support_box_and_activity module

test_support_box_and_activity.py - Test handling activity records.

class fpiweb.tests.test_support_box_and_activity.**BoxSupportTestCase**(*methodName='runTest'*)

Bases: django.test.testcases.TransactionTestCase

test_support_activityClass - Test handling activity records.

fixtures = ['Box', 'BoxType', 'Product', 'ProductCategory', 'LocRow', 'LocBin', 'LocTi

classmethod setUpClass() → None

Prepare to test box creation and manipulation.

Returns

classmethod tearDownClass() → None

Testing done, discard testing data. :return:

test_box_consume() → None

Test consuming a box under various circumstances.

Returns

test_box_fill() → None

Test adding a box under various circumstances.

Returns

test_box_move() → None

Test moving a box under various circumstances.

Returns

test_box_new() → None

Test adding a new box number to the inventory.

Returns

test_pallet_finish() → None

Test loading and finishing off a pallet.

Build a new pallet, add some boxes to it, stash the pallet id, the pallet box ids, and the box ids associated with them. Then finish the pallet and make sure the pallet and all its pallet boxes have been deleted, while the boxes themselves have been preserved. Since we are going through the new and fill logic above, we are going to assume the activity records have been properly created.

Returns

class fpiweb.tests.test_support_box_and_activity.**PalletBoxInfo**(*pallet_box_id: int*, *box_id: int*, *box_number: str*, *product: str*, *exp_year: int*)

Bases: tuple

Holds an entry in the dictionary of pallet box info.

box_id: int

Alias for field number 1

box_number: str

Alias for field number 2

exp_year: int
Alias for field number 4

pallet_box_id: int
Alias for field number 0

product: str
Alias for field number 3

fpiweb.tests.test_views module

class fpiweb.tests.test_views.**AboutViewTest**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

test_get()

class fpiweb.tests.test_views.**BoxNewViewTest**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

fixtures = ('Constraints', 'BoxType')

test_get_success_url()

class fpiweb.tests.test_views.**BuildPalletViewTest**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

static build_boxes(*number_of_boxes*)

static build_pallet_boxes(*pallet*, *boxes*, *product*, *exp_year*)

fixtures = ('Location', 'LocRow', 'LocBin', 'LocTier', 'Product', 'ProductCategory', '

static get_box_forms_post_data(*pallet_boxes*)

static get_box_item_form_data(*box_number*, *product*, *exp_year=None*)

static get_build_pallet_form(*location*)

static get_build_pallet_form_post_data(*location*)

static get_hidden_pallet_form_post_data(*pallet*)

static get_pallet_form(*pallet*)

static setup_user_and_client(*first_name: str*, *last_name: str*) → django.test.client.Client

test_get()

test_post_bad_form_name()

test_post_pallet_name_form()

test_post_pallet_select_name_form()

test_post_process_box_forms()

test_prepare_pallet_and_pallet_boxes__duplicate_box_numbers()

test_prepare_pallet_and_pallet_boxes__successful_run()

url = '/fpiweb/build_pallet/'

class fpiweb.tests.test_views.**IndexViewTest**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

test_get_success_url()

class fpiweb.tests.test_views.**LoginViewTest**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

test_get()

test_login_required_redirects_to_login_view()

test_post()

class fpiweb.tests.test_views.**LogoutViewTest**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

test_get()

class fpiweb.tests.test_views.**ManualMoveBoxViewTest**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

fixtures = ('BoxType', 'ProductCategory', 'Product', 'LocRow', 'LocBin', 'LocTier', 'L

static setup_user_and_client(*first_name: str*, *last_name: str*) → django.test.client.Client

test_get()

test_post_box_number_form()

test_post_location_form()

url = '/fpiweb/manual_move_box/'

class fpiweb.tests.test_views.**ManualPalletMoveViewTest**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

static build_confirm_merge_post_data(*from_location*, *to_location*, *action*)

static build_to_location_form_post_data(*from_location*, *to_location*)

fixtures = ('BoxType', 'Constraints', 'Location', 'LocBin', 'LocRow', 'LocTier', 'Prod

static setup_user_and_client(*first_name*, *last_name*)

test_get()

test_get_pallet_and_box_count()

test_post__missing_mode()

test_post__unrecognized_mode()

test_post_confirm_merge_form__action_change_location()

test_post_confirm_merge_form__action_merge_pallets()

test_post_confirm_merge_form__form_invalid()

test_post_from_location_form__form_invalid()

test_post_from_location_form__form_valid()

test_post_to_location_form__boxes_in_to_location()

test_post_to_location_form__form_invalid()

test_post_to_location_form__move_complete()

url = '/fpiweb/manual_pallet_move/'

class fpiweb.tests.test_views.**TestBoxItemFormView**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

test_get_form()

fpiweb.tests.test_views.**add_prefix**(*post_data*, *prefix*)

fpiweb.tests.test_views.**formset_form_post_data**(*prefix*, *form_index*, *form_data*)

fpiweb.tests.test_views.**management_form_post_data**(*prefix*, *total_forms*, *initial_forms=0*, *min_num_forms=0*, *max_num_forms=100*)

fpiweb.tests.utility module

class fpiweb.tests.utility.**ManageUserPermissions**
Bases: object

fpiweb.tests.utility.**create_user**(*, *username: str*, *first_name: str = 'first'*, *last_name: str = 'last'*, *title: str = 'title'*, *password: str = 'abc123'*, *access: fpiweb.constants.AccessLevel = <AccessLevel.Volunteer: 10>*)

fpiweb.tests.utility.**grant_required_permissions**(*user: django.contrib.auth.models.User*, *view: django.views.generic.base.View*) → None

fpiweb.tests.utility.**logged_in_user**(*first_name: str*, *last_name: str*, *view=None*) → django.test.client.Client

Submodules

fpiweb.admin module

Admin.py - Identify what can be managed by administrators.

class fpiweb.admin.**ActivityAdmin**(*model*, *admin_site*)
Bases: django.contrib.admin.options.ModelAdmin

list_display = ('id', 'box_number', 'date_filled', 'date_consumed', 'adjustment_code')

list_filter = ('adjustment_code',)

property media

class fpiweb.admin.**BoxAdmin**(*model*, *admin_site*)
Bases: django.contrib.admin.options.ModelAdmin

list_display = ('id', 'box_number', 'box_type', 'location', 'quantity', 'product')

list_filter = ('box_type',)

property media

class fpiweb.admin.**BoxTypeAdmin**(*model*, *admin_site*)
Bases: django.contrib.admin.options.ModelAdmin

list_display = ('box_type_code', 'box_type_descr', 'box_type_qty')

property media

class fpiweb.admin.**ConstraintsAdmin**(*model*, *admin_site*)
Bases: django.contrib.admin.options.ModelAdmin

list_display = ('constraint_name', 'constraint_type', 'constraint_min', 'constraint_ma

property media

class fpiweb.admin.**LocBinAdmin**(*model*, *admin_site*)
Bases: django.contrib.admin.options.ModelAdmin

list_display = ('loc_bin', 'loc_bin_descr')

property media

class fpiweb.admin.**LocRowAdmin**(*model*, *admin_site*)
Bases: django.contrib.admin.options.ModelAdmin

list_display = ('loc_row', 'loc_row_descr')

property media

class fpiweb.admin.**LocTierAdmin**(*model*, *admin_site*)
Bases: django.contrib.admin.options.ModelAdmin

list_display = ('loc_tier', 'loc_tier_descr')

property media

class fpiweb.admin.**Location**(*model*, *admin_site*)
Bases: django.contrib.admin.options.ModelAdmin

list_display = ('pk', 'loc_row', 'loc_bin', 'loc_tier')

list_filter = ('loc_row', 'loc_bin', 'loc_tier')

property media

class fpiweb.admin.**PalletAdmin**(*model*, *admin_site*)
Bases: django.contrib.admin.options.ModelAdmin

list_display = ('pk', 'name')

property media

class fpiweb.admin.**PalletBoxAdmin**(*model*, *admin_site*)
Bases: django.contrib.admin.options.ModelAdmin

list_display = ('box_number', 'pallet', 'box', 'product', 'exp_year', 'exp_month_start

property media

class fpiweb.admin.**ProductAdmin**(*model*, *admin_site*)
Bases: django.contrib.admin.options.ModelAdmin

list_display = ('prod_name', 'prod_cat')

property media

class fpiweb.admin.**ProfileAdmin**(*model*, *admin_site*)
Bases: django.contrib.admin.options.ModelAdmin

list_display = ('user', 'title', 'active_pallet_id')

property media

fpiweb.apps module

class fpiweb.apps.**FpiwebConfig**(*app_name*, *app_module*)
Bases: django.apps.config.AppConfig

name = 'fpiweb'

fpiweb.code_reader module

exception fpiweb.code_reader.**CodeReaderError**
Bases: RuntimeError

fpiweb.code_reader.**delete_file**(*image_file_path*)

fpiweb.code_reader.**get_scan_file_path**()

fpiweb.code_reader.**read**(*scan_data*)

fpiweb.code_reader.**read_box_number**(*scan_data*)

fpiweb.constants module

Global constants, constructs and exceptions for this project.

Constants and constructs that don't have an obvious home elsewhere are defined here.

Non-Django-based exceptions are defined here.

class fpiweb.constants.**AccessGroupsAndFlags**(*access_level:* fpiweb.constants.AccessLevel)
Bases: object

Unpacked attributes of an access level.

access_level: *fpiweb.constants.AccessLevel* = None

is_admin_group: bool = None

is_staff_flag: bool = None

is_staff_group: bool = None

is_superuser_flag: bool = None

is_volunteer_group: bool = None

name: str = None

value: int = None

class fpiweb.constants.**AccessLevel**(*value*)
Bases: *fpiweb.constants.OrderedEnum*

Level of access for a user - lowest to highest.

Admin: int = 99

No_Access: int = 0

Staff: int = 50

Volunteer: int = 10

do_not_call_in_templates = True

fpiweb.constants.**CURRENT_YEAR = 2021**
The current year - used for validating expiration dates

fpiweb.constants.**EnumForDjango**(*cls*)

exception fpiweb.constants.**InternalError**(*message*, *code=None*, *params=None*)
Bases: *fpiweb.constants.ProjectError*

The error is raised when there is some interal logic problem.

exception fpiweb.constants.**InvalidActionAttemptedError**(*message*, *code=None*, *params=None*)
Bases: *fpiweb.constants.ProjectError*

A requested action cannot be done at this time.

exception fpiweb.constants.**InvalidValueError**(*message*, *code=None*, *params=None*)
Bases: *fpiweb.constants.ProjectError*

Used when an invalid value has been passed as a parameter.

class fpiweb.constants.**OrderedEnum**(*value*)
Bases: enum.Enum

Enhanced Enum class that considers the members as ordered

exception fpiweb.constants.**ProjectError**(*message*, *code=None*, *params=None*)
Bases: django.core.exceptions.ValidationError

All exceptions unique to this project wil be based on this class.

class fpiweb.constants.**TargetUser**(*username: str = ''*, *pswd_changed: bool = False*, *force_password: bool = False*, *first_name: str = ''*, *last_name: str = ''*, *email: str = ''*, *title: str = ''*, *access_level: fpiweb.constants.AccessLevel = <AccessLevel.No_Access: 0>*, *is_active: bool = True*)
Bases: tuple

User information to be added or updated.

access_level: *fpiweb.constants.AccessLevel*
Alias for field number 7

email: str
Alias for field number 5

first_name: str
Alias for field number 3

force_password: bool
Alias for field number 2

is_active: bool
Alias for field number 8

last_name: str
Alias for field number 4

pswd_changed: bool
Alias for field number 1

title: str
Alias for field number 6

username: str
Alias for field number 0

class fpiweb.constants.**UserInfo**(*user: django.contrib.auth.models.User*, *profile: fpiweb.models.Profile*, *highest_access_level: fpiweb.constants.AccessLevel*, *is_active: bool*, *is_superuser: bool*)
Bases: tuple

Information about a specific user - abstracted from various tables.

get_sort_key() → str
Return a usable sort key :return:

highest_access_level: *fpiweb.constants.AccessLevel*
Alias for field number 2

is_active: bool
Alias for field number 3

is_superuser: bool
Alias for field number 4

profile: *fpiweb.models.Profile*
Alias for field number 1

user: django.contrib.auth.models.User
Alias for field number 0

class fpiweb.constants.**ValidOrErrorResponse**(*is_valid: bool = True*, *error_msg_list: List[str] = <factory>*)
Bases: object

A constructed response denoting either valid or has error messages.

add_error(*msg: str*)
Set as invalid and add an error message.

Parameters **msg** – error message to add

Returns

error_msg_list: List[str]

is_valid: bool = True

fpiweb.constants.**load_access_dict**() → Dict
Load a multipurpose dictionary with all access level identifieres.

Returns a dictionary with lots of keys

fpiweb.forms module

forms.py - provide validation of a forms.

class fpiweb.forms.**BoxItemForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *field_order=None*, *use_required_attribute=None*, *renderer=None*)
Bases: django.forms.forms.Form

Form for the Box as it appears as part of a formset on the Build Pallet page

base_fields = {'box_number': <fpiweb.forms.BoxNumberField object>, 'exp_month_end':

clean()
Hook for doing any extra form-wide cleaning after Field.clean() has been called on every field. Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field named '__all__'.

declared_fields = {'box_number': <fpiweb.forms.BoxNumberField object>, 'exp_month_end

static get_initial_from_box(*box*)

Parameters **box** – Box or PalletBox record

Returns

property media
Return all media required to render the widgets on this form.

class fpiweb.forms.**BoxNumberField**(***kwargs*)
Bases: django.forms.fields.CharField

Accepts box number with or without BOX prefix. Returns BoxNumber with BOX prefix and leading zeros

clean(*value*)
Validate the given value and return its "cleaned" value as an appropriate Python object. Raise ValidationError for any errors.

class fpiweb.forms.**BoxTypeForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *field_order=None*, *use_required_attribute=None*, *renderer=None*)
Bases: django.forms.forms.Form

A form to use whenever a box type selection is needed.

base_fields = {'box_type': <django.forms.models.ModelChoiceField object>}

declared_fields = {'box_type': <django.forms.models.ModelChoiceField object>}

property media
Return all media required to render the widgets on this form.

class fpiweb.forms.**BoxTypeMaintenanceForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *instance=None*, *use_required_attribute=None*, *renderer=None*)
Bases: django.forms.models.ModelForm

class Meta
Bases: object

fields = ['id', 'box_type_code', 'box_type_descr', 'box_type_qty']

model
alias of *fpiweb.models.BoxType*

base_fields = {'box_type_code': <django.forms.fields.CharField object>, 'box_type_des

clean()
Hook for doing any extra form-wide cleaning after Field.clean() has been called on every field. Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field named '__all__'.

declared_fields = {'box_type_code': <django.forms.fields.CharField object>, 'box_type

property media
Return all media required to render the widgets on this form.

static validate_box_type_fields(*box_type_code: str*, *box_type_descr: str*, *box_type_qty: int*)

class fpiweb.forms.**BuildPalletForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *instance=None*, *use_required_attribute=None*, *renderer=None*)
Bases: django.forms.models.ModelForm

class Meta
Bases: object

fields = ('loc_row', 'loc_bin', 'loc_tier')

model
alias of *fpiweb.models.Location*

base_fields = {'loc_bin': <django.forms.models.ModelChoiceField object>, 'loc_row':

clean()
Hook for doing any extra form-wide cleaning after Field.clean() has been called on every field. Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field named '__all__'.

declared_fields = {}

property media
Return all media required to render the widgets on this form.

class fpiweb.forms.**ConfirmMergeForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *field_order=None*, *use_required_attribute=None*, *renderer=None*)
Bases: django.forms.forms.Form

ACTION_CHANGE_LOCATION = 'C'

ACTION_CHOICES = (('C', 'Change To Location'), ('M', 'Merge Pallets'))

ACTION_MERGE_PALLETS = 'M'

base_fields = {'action': <django.forms.fields.ChoiceField object>, 'boxes_at_to_locat

boxes_at_to_location_int()

declared_fields = {'action': <django.forms.fields.ChoiceField object>, 'boxes_at_to_l

property media
Return all media required to render the widgets on this form.

to_location_str()

class fpiweb.forms.**ConstraintsForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *instance=None*, *use_required_attribute=None*, *renderer=None*)
Bases: django.forms.models.ModelForm

Manage Constraint details with a generic form.

class Meta
Bases: object

Additional info to help Django provide intelligent defaults.

fields = ['id', 'constraint_name', 'constraint_descr', 'constraint_type', 'constrai

model
alias of *fpiweb.models.Constraints*

base_fields = {'constraint_descr': <django.forms.fields.CharField object>, 'constraint

clean()
Clean and validate the data given for the constraint record.

Returns

declared_fields = {'constraint_list': <django.forms.fields.CharField object>, 'constra

property media
Return all media required to render the widgets on this form.

static validate_constraint_fields(*con_name: str*, *con_descr: str*, *con_type: Int-MM, Integer Min/Max, Char-MM, Character Min/Max, Int-List, Integer Valid List, Char-List, Character Valid List*, *con_min: Union[str, int]*, *con_max: Union[str, int]*, *con_list: str*)
Validate the various constraint record fields.

Parameters

- **con_name** – name of constraint
- **con_descr** – description of constraint
- **con_type** – type of constraint
- **con_min** – minimum value, if given
- **con_max** – maximum value, if given
- **con_list** – list of values, if given

Returns

class fpiweb.forms.**EmptyBoxNumberField**(***kwargs*)
Bases: *fpiweb.forms.ExtantBoxNumberField*

Checks whether there's a Box with the specified box number in the database. If a matching Box is found, this Box is stored in the field's box attribute

clean(*value*)
Validate the given value and return its "cleaned" value as an appropriate Python object. Raise ValidationError for any errors.

class fpiweb.forms.**EmptyBoxNumberForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *field_order=None*, *use_required_attribute=None*, *renderer=None*)
Bases: django.forms.forms.Form

base_fields = {'box_number': <fpiweb.forms.EmptyBoxNumberField object>}

declared_fields = {'box_number': <fpiweb.forms.EmptyBoxNumberField object>}

property media
Return all media required to render the widgets on this form.

class fpiweb.forms.**ExistingBoxTypeForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *field_order=None*, *use_required_attribute=None*, *renderer=None*)

Bases: *fpiweb.forms.BoxTypeForm*

A form to validate that the box type exists.

base_fields = {'box_type': <django.forms.models.ModelChoiceField object>}

clean()
Validate the box type.

declared_fields = {'box_type': <django.forms.models.ModelChoiceField object>}

property media
Return all media required to render the widgets on this form.

class fpiweb.forms.**ExistingLocationForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *instance=None*, *use_required_attribute=None*, *renderer=None*)

Bases: *fpiweb.forms.LocationForm*

base_fields = {'loc_bin': <django.forms.models.ModelChoiceField object>, 'loc_row':

clean()
Hook for doing any extra form-wide cleaning after Field.clean() has been called on every field. Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field named '__all__'.

declared_fields = {}

property media
Return all media required to render the widgets on this form.

class fpiweb.forms.**ExistingLocationWithBoxesForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *instance=None*, *use_required_attribute=None*, *renderer=None*)

Bases: *fpiweb.forms.ExistingLocationForm*

base_fields = {'loc_bin': <django.forms.models.ModelChoiceField object>, 'loc_row':

clean()
Hook for doing any extra form-wide cleaning after Field.clean() has been called on every field. Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field named '__all__'.

declared_fields = {}

property media
Return all media required to render the widgets on this form.

class fpiweb.forms.**ExistingProductForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *field_order=None*, *use_required_attribute=None*, *renderer=None*)

Bases: *fpiweb.forms.ProductForm*

base_fields = {'product': <django.forms.models.ModelChoiceField object>}

clean()

Hook for doing any extra form-wide cleaning after Field.clean() has been called on every field. Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field named '__all__'.

declared_fields = {'product': <django.forms.models.ModelChoiceField object>}

property media

Return all media required to render the widgets on this form.

class fpiweb.forms.**ExpMoEndForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *field_order=None*, *use_required_attribute=None*, *renderer=None*)

Bases: django.forms.forms.Form

A form for use whenever you need to select an ending month.

base_fields = {'exp_month_end': <django.forms.fields.TypedChoiceField object>}

declared_fields = {'exp_month_end': <django.forms.fields.TypedChoiceField object>}

property media

Return all media required to render the widgets on this form.

valid_months = [(0, 'No Label'), ('1', 'Jan'), ('2', 'Feb'), ('3', 'Mar'), ('4', 'Apr'

class fpiweb.forms.**ExpMoStartForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *field_order=None*, *use_required_attribute=None*, *renderer=None*)

Bases: django.forms.forms.Form

A form for use whenever you need to select a starting month.

base_fields = {'exp_month_start': <django.forms.fields.TypedChoiceField object>}

declared_fields = {'exp_month_start': <django.forms.fields.TypedChoiceField object>}

property media

Return all media required to render the widgets on this form.

valid_months = [(0, 'No Label'), ('1', 'Jan'), ('2', 'Feb'), ('3', 'Mar'), ('4', 'Apr'

class fpiweb.forms.**ExpYearForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *field_order=None*, *use_required_attribute=None*, *renderer=None*)

Bases: django.forms.forms.Form

A form for use whenever you need to select a year.

base_fields = {'exp_year': <django.forms.fields.TypedChoiceField object>}

declared_fields = {'exp_year': <django.forms.fields.TypedChoiceField object>}

property media
Return all media required to render the widgets on this form.

class fpiweb.forms.**ExtantBoxNumberField**(***kwargs*)
Bases: *fpiweb.forms.RelaxedBoxNumberField*

Checks whether there's a Box with the specified box number in the database. If a matching Box is found, this Box is stored in the field's box attribute

clean(*value*)
Validate the given value and return its "cleaned" value as an appropriate Python object. Raise ValidationError for any errors.

class fpiweb.forms.**ExtantBoxNumberForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *field_order=None*, *use_required_attribute=None*, *renderer=None*)
Bases: django.forms.forms.Form

base_fields = {'box_number': <fpiweb.forms.ExtantBoxNumberField object>}

declared_fields = {'box_number': <fpiweb.forms.ExtantBoxNumberField object>}

property media
Return all media required to render the widgets on this form.

class fpiweb.forms.**FillBoxForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *instance=None*, *use_required_attribute=None*, *renderer=None*)
Bases: django.forms.models.ModelForm

class Meta
Bases: object

fields = ['product', 'exp_year', 'exp_month_start', 'exp_month_end']

model
alias of *fpiweb.models.Box*

base_fields = {'exp_month_end': <django.forms.fields.TypedChoiceField object>, 'exp_m

clean()
Clean and validate the data in this box record.

Returns

declared_fields = {'exp_month_end': <django.forms.fields.TypedChoiceField object>, 'e

property media
Return all media required to render the widgets on this form.

class fpiweb.forms.**FilledBoxNumberField**(***kwargs*)
Bases: *fpiweb.forms.ExtantBoxNumberField*

Checks whether there's a Box with the specified box number in the database. If a matching Box is found, this Box is stored in the field's box attribute

clean(*value*)
Validate the given value and return its "cleaned" value as an appropriate Python object. Raise ValidationError for any errors.

class fpiweb.forms.**FilledBoxNumberForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *field_order=None*, *use_required_attribute=None*, *renderer=None*)

Bases: django.forms.forms.Form

base_fields = {'box_number': <fpiweb.forms.FilledBoxNumberField object>}

declared_fields = {'box_number': <fpiweb.forms.FilledBoxNumberField object>}

property media

Return all media required to render the widgets on this form.

class fpiweb.forms.**HiddenPalletForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *field_order=None*, *use_required_attribute=None*, *renderer=None*)

Bases: django.forms.forms.Form

base_fields = {'pallet': <django.forms.models.ModelChoiceField object>}

declared_fields = {'pallet': <django.forms.models.ModelChoiceField object>}

property media

Return all media required to render the widgets on this form.

class fpiweb.forms.**Html5DateInput**(*attrs=None*, *format=None*)

Bases: django.forms.widgets.DateInput

input_type = 'date'

property media

class fpiweb.forms.**LocBinForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *instance=None*, *use_required_attribute=None*, *renderer=None*)

Bases: django.forms.models.ModelForm

Manage Location bin details with a generic form.

class Meta

Bases: object

Additional info to help Django provide intelligent defaults.

fields = ['id', 'loc_bin', 'loc_bin_descr']

model

alias of *fpiweb.models.LocBin*

base_fields = {'loc_bin': <django.forms.fields.CharField object>, 'loc_bin_descr': <

clean()

Clean and validate the data given for the bin record.

Returns

declared_fields = {'loc_bin': <django.forms.fields.CharField object>}

property media

Return all media required to render the widgets on this form.

static validate_loc_bin_fields(*loc_bin_name: str*, *loc_bin_descr: str*)
Validate the various location bin record fields.

Parameters

- **loc_bin_name** – name of bin
- **loc_bin_descr** – description of bin

Returns True if valid

class fpiweb.forms.**LocRowForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *instance=None*, *use_required_attribute=None*, *renderer=None*)
Bases: django.forms.models.ModelForm

Manage Location row details with a generic form.

class Meta
Bases: object

Additional info to help Django provide intelligent defaults.

fields = ['id', 'loc_row', 'loc_row_descr']

model
alias of *fpiweb.models.LocRow*

base_fields = {'loc_row': <django.forms.fields.CharField object>, 'loc_row_descr': <

clean()
Clean and validate the data given for the constraint record.

Returns

declared_fields = {'loc_row': <django.forms.fields.CharField object>}

property media
Return all media required to render the widgets on this form.

static validate_loc_row_fields(*loc_row_name: str*, *loc_row_descr: str*)
Validate the various location row record fields.

Parameters

- **loc_row_name** – name of row
- **loc_row_descr** – description of row

Returns True if valid

class fpiweb.forms.**LocTierForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *instance=None*, *use_required_attribute=None*, *renderer=None*)
Bases: django.forms.models.ModelForm

Manage Loction tier details with a generic form.

class Meta
Bases: object

Additional info to help Django provide intelligent defaults.

fields = ['id', 'loc_tier', 'loc_tier_descr']

model
alias of *fpiweb.models.LocTier*

base_fields = {'loc_tier': <django.forms.fields.CharField object>, 'loc_tier_descr':

clean()
Clean and validate the data given for the tier record.

Returns

declared_fields = {'loc_tier': <django.forms.fields.CharField object>}

property media
Return all media required to render the widgets on this form.

static validate_loc_tier_fields(*loc_tier_name: str*, *loc_tier_descr: str*)
Validate the various location tier record fields.

Parameters

- **loc_tier_name** – name of tier
- **loc_tier_descr** – description of tier

Returns True if valid

class fpiweb.forms.**LocationForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *instance=None*, *use_required_attribute=None*, *renderer=None*)
Bases: django.forms.models.ModelForm

A form for use whenever you need to select row, bin, and tier

class Meta
Bases: object

fields = ('loc_row', 'loc_bin', 'loc_tier')

model
alias of *fpiweb.models.Location*

base_fields = {'loc_bin': <django.forms.models.ModelChoiceField object>, 'loc_row':

declared_fields = {}

property media
Return all media required to render the widgets on this form.

class fpiweb.forms.**ManualLocTableForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *instance=None*, *use_required_attribute=None*, *renderer=None*)
Bases: django.forms.models.ModelForm

class Meta
Bases: object

fields = ['loc_code', 'loc_descr', 'loc_in_warehouse', 'loc_bin', 'loc_row', 'loc_t

model
alias of *fpiweb.models.Location*

base_fields = {'loc_bin': <django.forms.models.ModelChoiceField object>, 'loc_code':

clean()
Hook for doing any extra form-wide cleaning after Field.clean() has been called on every field. Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field named '__all__'.

declared_fields = {'loc_code': <django.forms.fields.CharField object>}

property media
Return all media required to render the widgets on this form.

static validate_manual_loc_table_fields(*loc_code: str*, *loc_descr: str*, *loc_in_warehouse: bool*, *loc_bin: int*, *loc_row: int*, *loc_tier: int*)
Validate the various Location record fields :param loc_code: Location Code :param loc_descr: Location Description :param loc_bin: Location Bin :param loc_row: Location Row :param loc_tier: Location Tier

class fpiweb.forms.**MoveToLocationForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *instance=None*, *use_required_attribute=None*, *renderer=None*)
Bases: *fpiweb.forms.ExistingLocationForm*

base_fields = {'from_location': <django.forms.models.ModelChoiceField object>, 'loc_b

declared_fields = {'from_location': <django.forms.models.ModelChoiceField object>}

property media
Return all media required to render the widgets on this form.

class fpiweb.forms.**NewBoxForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *instance=None*, *use_required_attribute=None*, *renderer=None*)
Bases: django.forms.models.ModelForm

class Meta
Bases: object

fields = ['box_number', 'box_type']

model
alias of *fpiweb.models.Box*

base_fields = {'box_number': <django.forms.fields.CharField object>, 'box_type': <dj

declared_fields = {'box_number': <django.forms.fields.CharField object>}

property media
Return all media required to render the widgets on this form.

save(*commit=True*)
Save this form's self.instance object if commit=True. Otherwise, add a save_m2m() method to the form which can be called after the instance is saved manually at a later time. Return the model instance.

class fpiweb.forms.**NewBoxNumberField**(***kwargs*)
Bases: *fpiweb.forms.RelaxedBoxNumberField*

Add a new box number to the database.

Checks whether there's a Box with the specified box number in the database. If a matching Box is not found, store the box number in the field's box attribute

clean(*value*)
Validate the given value and return its "cleaned" value as an appropriate Python object. Raise ValidationError for any errors.

class fpiweb.forms.**NewBoxNumberForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *field_order=None*, *use_required_attribute=None*, *renderer=None*)
Bases: django.forms.forms.Form

base_fields = {'box_number': <fpiweb.forms.NewBoxNumberField object>}

declared_fields = {'box_number': <fpiweb.forms.NewBoxNumberField object>}

property media
Return all media required to render the widgets on this form.

fpiweb.forms.**NormalizeEmail**(*email*)
Normalize an email address.

Django normalizes by lower-casing the domain portion of the address.

Parameters email –

Returns

class fpiweb.forms.**PalletNameForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *instance=None*, *use_required_attribute=None*, *renderer=None*)
Bases: django.forms.models.ModelForm

class Meta
Bases: object

fields = ('name',)

model
alias of *fpiweb.models.Pallet*

base_fields = {'name': <django.forms.fields.CharField object>}

declared_fields = {}

property media
Return all media required to render the widgets on this form.

class fpiweb.forms.**PalletSelectForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *field_order=None*, *use_required_attribute=None*, *renderer=None*)
Bases: django.forms.forms.Form

base_fields = {'pallet': <django.forms.models.ModelChoiceField object>}

declared_fields = {'pallet': <django.forms.models.ModelChoiceField object>}

property media
Return all media required to render the widgets on this form.

class fpiweb.forms.**ProductCategoryForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *instance=None*, *use_required_attribute=None*, *renderer=None*)

Bases: django.forms.models.ModelForm

Manage Product Category details with a generic form.

class Meta

Bases: object

Additional info to help Django provide intelligent defaults.

fields = ['id', 'prod_cat_name', 'prod_cat_descr']

model

alias of *fpiweb.models.ProductCategory*

base_fields = {'prod_cat_descr': <django.forms.fields.CharField object>, 'prod_cat_na

clean()

Clean and validate the data given for the constraint record.

Returns

declared_fields = {'prod_cat_name': <django.forms.fields.CharField object>}

property media

Return all media required to render the widgets on this form.

static validate_prod_cat_fields(*prod_cat_name: str*, *prod_cat_descr: str*)

Validate the various product category record fields.

Parameters

- **prod_cat_name** – name of product catagory
- **prod_cat_descr** – description of product description

Returns True if valid

class fpiweb.forms.**ProductExampleForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *instance=None*, *use_required_attribute=None*, *renderer=None*)

Bases: django.forms.models.ModelForm

Manage Loction row details with a generic form.

class Meta

Bases: object

Additional info to help Django provide intelligent defaults.

fields = ['id', 'prod_example_name', 'product']

model

alias of *fpiweb.models.ProductExample*

base_fields = {'prod_example_name': <django.forms.fields.CharField object>, 'product'

clean()

Clean and validate the data given for the constraint record.

Returns

declared_fields = {'prod_example_name': <django.forms.fields.CharField object>}

property media
Return all media required to render the widgets on this form.

static validate_product_example_fields(*prod_example_name: str*, *product: int*)
Validate the Product Example Name and Foreigh Key Product ID

Parameters

- **product_example_name** – name of product example
- **product** – foreign key from products table

Returns True if valid

class fpiweb.forms.**ProductForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *field_order=None*, *use_required_attribute=None*, *renderer=None*)
Bases: django.forms.forms.Form

A form for use whenever you need to select a product.

base_fields = {'product': <django.forms.models.ModelChoiceField object>}

declared_fields = {'product': <django.forms.models.ModelChoiceField object>}

property media
Return all media required to render the widgets on this form.

class fpiweb.forms.**ProductNameForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *instance=None*, *use_required_attribute=None*, *renderer=None*)
Bases: django.forms.models.ModelForm

Manage Product details with a generic form.

class Meta
Bases: object

Additional info to help Django provide intelligent defaults.

fields = ['id', 'prod_name', 'prod_cat']

model
alias of *fpiweb.models.Product*

base_fields = {'prod_cat': <django.forms.models.ModelChoiceField object>, 'prod_name'

clean()
Clean and validate the data given for the constraint record.

Returns

declared_fields = {'prod_name': <django.forms.fields.CharField object>}

property media
Return all media required to render the widgets on this form.

static validate_product_fields(*prod_name: str*, *prod_cat: int*)
Validate the various product category record fields.

Parameters

- **prod_cat_name** – name of product
- **prod_cat_id** – foreign key from product category

Returns True if valid

class fpiweb.forms.**RelaxedBoxNumberField**(***kwargs*)
Bases: django.forms.fields.CharField

Define box nummber field that allows all numberics.

Accepts box number with or without BOX prefix. Returns BoxNumber with BOX prefix and leading zeros

clean(*value: str*) → str
Validate the given value and return its "cleaned" value as an appropriate Python object. Raise ValidationError for any errors.

class fpiweb.forms.**UserInfoForm**(**args*, ***kwargs*)
Bases: django.forms.forms.Form

Define all the fields and modes needed to add or edit a user

LEVEL_CHOICES = [('No_Access', 'No_Access'), ('Volunteer', 'Volunteer'), ('Staff', 'St

class Meta
Bases: object

fields = ['userid', 'username', 'force_password', 'first_name', 'last_name', 'email

base_fields = {'access_level': <django.forms.fields.ChoiceField object>, 'email': <d

clean()
Validate the consistency across firlds.

Returns

clean_access_level()

clean_email()

clean_first_name()

clean_force_password()

clean_is_active()

clean_last_name()

clean_title()

clean_username()
ensure that the username does not contain screwball characters

declared_fields = {'access_level': <django.forms.fields.ChoiceField object>, 'email':

level = 99

level_entry = ('Admin', 'Admin')

property media
Return all media required to render the widgets on this form.

class fpiweb.forms.**UserInfoModes**(*value*)
Bases: enum.Enum

Mode identifiers used in both UserInfoForm and in multiple views.

These modes are used throughout the user management processing in an attempt to minimize code duplication. They are defined here (rather than in the views) to eliminate circular references.

MODE_ADD_USER: str = 'add user'

MODE_CONFIRM: str = 'confirm action'

MODE_SHOW_USERS: str = 'show users'

MODE_UPDATE_USER: str = 'update user'

do_not_call_in_templates = True

fpiweb.forms.**add_no_selection_choice**(*other_choices*, *dash_count=2*)

fpiweb.forms.**bin_choices**()

fpiweb.forms.**box_number_validator**(*value*) → None

fpiweb.forms.**char_list_choices**(*constraint_name*)

fpiweb.forms.**expire_year_choices**()

fpiweb.forms.**min_max_choices**(*constraint_name*)

fpiweb.forms.**month_choices**()

fpiweb.forms.**none_or_int**(*text: str*) → Optional[int]
Convert test to a valid integer or None.

Parameters **text** –

Returns

fpiweb.forms.**none_or_list**(*text: str*) → Optional[list]
Convert text to list or None.

Parameters **text** –

Returns

fpiweb.forms.**none_or_str**(*text: str*) → Optional[str]
Convert text to non-empty string or None.

Parameters **text** –

Returns

fpiweb.forms.**row_choices**()

fpiweb.forms.**tier_choices**()

fpiweb.forms.**validate_exp_month_start_end**(*exp_month_start: Optional[int]*, *exp_month_end: Optional[int]*) → bool
Validate the start and end month, if given.

Parameters

- **exp_month_start** – number 1-12 (integer or string)
- **exp_month_end** – number 1-12 (integer or string)

Returns

fpiweb.forms.**validate_int_list**(*char_list: list*) → bool
Verify that all values in the list are integers.

Parameters **text** –

Returns

fpiweb.forms.**validation_exp_months_bool**(*exp_month_start: Optional[int]*, *exp_month_end: Optional[int]*) → *fpiweb.constants.ValidOrErrorResponse*

Validate the expiration months returning only a boolean.

Parameters

- **exp_month_start** –
- **exp_month_end** –

Returns True if valid, False if not

fpiweb.models module

models.py - Define the database tables using ORM models.

class fpiweb.models.**Activity**(**args*, ***kwargs*)

Bases: django.db.models.base.Model

Activity (history) from the past.

ADJUSTMENT_CODE_CHOICES: list = (('Fill Emptied', 'Fill emptied previous contents'), (

CONSUME_ADDED: str = 'Consume Added'

CONSUME_EMPTIED: str = 'Consume Emptied'

exception DoesNotExist

Bases: django.core.exceptions.ObjectDoesNotExist

FILL_EMPTIED: str = 'Fill Emptied'

MOVE_ADDED: str = 'Move Added'

MOVE_CONSUMED: str = 'Move Consumed'

exception MultipleObjectsReturned

Bases: django.core.exceptions.MultipleObjectsReturned

adjustment_code

Coded reason if this entry was adjusted

adjustment_code_help_text = 'Coded reason if this entry was adjusted'

box_number

Box number on box at time of consumption.

box_number_help_text = 'Box number on box at time of consumption.'

box_type

Box type holding consumed product.

box_type_help_text = 'Box type holding consumed product.'

date_consumed

Date product was consumed.

date_consumed_help_text = 'Date product was consumed.'

date_filled

Approximate date product was put in the box.

date_filled_help_text = 'Approximate date product was put in the box.'

duration
Number of days between date box was filled and consumed.

duration_help_text = 'Number of days between date box was filled and consumed.'

exp_month_end
Optional ending month product would have expired.

exp_month_end_help_text = 'Optional ending month product would have expired.'

exp_month_start
Optional starting month product would have expired.

exp_month_start_help_text = 'Optional starting month product would have expired.'

exp_year
Year product would have expired.

exp_year_help_text = 'Year product would have expired.'

get_adjustment_code_display(*, *field=<django.db.models.fields.CharField: adjustment_code>*)

get_next_by_date_filled(*, *field=<django.db.models.fields.DateField: date_filled>*, *is_next=True*, ***kwargs*)

get_previous_by_date_filled(*, *field=<django.db.models.fields.DateField: date_filled>*, *is_next=False*, ***kwargs*)

id
Internal record identifier for an activity.

id_help_text = 'Internal record identifier for an activity.'

loc_bin
Bin box was in at the time product was consumed.

loc_bin_help_text = 'Bin box was in at the time product was consumed.'

loc_row
Row box was in at the time product was consumed.

loc_row_help_text = 'Row box was in at the time product was consumed.'

loc_tier
Tier box was in at the time product was consumed.

loc_tier_help_text = 'Tier box was in at the time product was consumed.'

objects = <django.db.models.manager.Manager object>

prod_cat_name
Category of product consumed.

prod_cat_name_help_text = 'Category of product consumed.'

prod_name
Product contained in box at time of consumption.

prod_name_help_text = 'Product contained in box at time of consumption.'

quantity
Approximate number of items in the box when it was filled.

quantity_help_text = 'Approximate number of items in the box when it was filled.'

class fpiweb.models.**Box**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Box or container for product.

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception MultipleObjectsReturned
Bases: django.core.exceptions.MultipleObjectsReturned

box_number
Number printed in the label on the box.

box_number_help_text = 'Number printed in the label on the box.'

box_number_max_length = 8

box_number_min_length = 8

box_type
Type of box with this number.

static box_type_default()

box_type_help_text = 'Type of box with this number.'

box_type_id

date_filled
Approximate date box was filled, if filled.

date_filled_help_text = 'Approximate date box was filled, if filled.'

exp_month_end
Optional emding month range of when the product expires, if filled.

exp_month_end_help_text = 'Optional ending month range of when the product expires, if

exp_month_start
Optional starting month range of when the product expires, if filled.

exp_month_start_help_text = 'Optional starting month range of when the product expires

exp_year
Year the product expires, if filled.

exp_year_help_text = 'Year the product expires, if filled.'

get_absolute_url()

id
Internal record identifier for box.

id_help_text = 'Internal record identifier for box.'

is_filled()

location
Location of box

location_help_text = 'Location of box'

location_id

objects = <django.db.models.manager.Manager object>

palletbox_set
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

product
Product contained in this box, if filled.

product_help_text = 'Product contained in this box, if filled.'

product_id

quantity
Approximate or default number of items in the box, if filled.

quantity_help_text = 'Approximate or default number of items in the box, if filled.'

static select_location(*queryset*)
Since we're probably going to have a lot of Box queries where we also want to pull in location data

exception fpiweb.models.**BoxError**
Bases: `RuntimeError`

class fpiweb.models.**BoxNumber**
Bases: `object`

box_number_regex = re.compile('^BOX\\d{5}$')

box_number_search_regex = re.compile('box\\d{5}', re.IGNORECASE)

static format_box_number(*int_box_number: int*) → str
format an integer into a box number

static get_next_box_number() → str
get the next unused box number

static validate(*box_number: str*) → bool
validate that a string is of the form 'BOXnnnnn'

class fpiweb.models.**BoxType**(**args*, ***kwargs*)
Bases: `django.db.models.base.Model`

Type of box (Evan's boxes, large boxes, etc.) and default quantity.

exception DoesNotExist
Bases: `django.core.exceptions.ObjectDoesNotExist`

exception MultipleObjectsReturned
Bases: `django.core.exceptions.MultipleObjectsReturned`

box_set
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

box_type_code
Type of box (code or shorthand).

box_type_code_help_text = 'Type of box (code or shorthand).'

box_type_code_max_len = 10

box_type_descr
Type of box (description).

box_type_descr_help_text = 'Type of box (description).'

box_type_descr_max_len = 30

box_type_qty
Number of items (usually cans) that can typically fix in this box.

box_type_qty_help_text = 'Number of items (usually cans) that can typically fit in thi

id
Internal record identifier for box type.

id_help_text = 'Internal record identifier for box type.'

objects = <django.db.models.manager.Manager object>

class fpiweb.models.**Constraints**(**args*, ***kwargs*)
Bases: `django.db.models.base.Model`

Constraints of valid values.

BIN: str = 'Bin'

CHAR_LIST = 'Char-List'

CHAR_RANGE = 'Char-MM'

CONSTRAINT_NAME_CHOICES = (('Row', 'Rows in the warehouse'), ('Bin', 'Bins in the Ware

CONSTRAINT_TYPE_CHOICES = (('Int-MM', 'Integer Min/Max'), ('Char-MM', 'Character Min/M

exception DoesNotExist
Bases: `django.core.exceptions.ObjectDoesNotExist`

FUTURE_EXP_YEAR_LIMIT = 'Future Expiration Year Limit'

INT_LIST = 'Int-List'

INT_RANGE = 'Int-MM'

LOCATION_EXCLUSIONS = 'Location Exclusions'

exception MultipleObjectsReturned
Bases: `django.core.exceptions.MultipleObjectsReturned`

QUANTITY_LIMIT: str = 'Quantity Limit'

ROW: str = 'Row'

TIER: str = 'Tier'

constraint_descr
Description of this constraint.

constraint_descr_help_text = 'Description of this constraint.'

constraint_list
If a list, what are the valid values?

constraint_list_help_text = 'If a list, what are the valid values?'

constraint_max
If a range, what is the maximum valid value?

constraint_max_help_text = 'If a range, what is the maximum valid value?'

constraint_min
If a range, what is the minimum valid value?

constraint_min_help_text = 'If a range, what is the minimum valid value?'

constraint_name
Coded name of a constraint.

constraint_name_help_text = 'Coded name of a constraint.'

constraint_type
Type of constraint (integer or character, list or range).

constraint_type_help_text = 'Type of constraint (integer or character, list or range).

get_constraint_name_display(*, *field=<django.db.models.fields.CharField: constraint_name>*)

get_constraint_type_display(*, *field=<django.db.models.fields.CharField: constraint_type>*)

static get_values(*constraint_name*)

id
Internal record identifier for a constraint.

id_help_text = 'Internal record identifier for a constraint.'

objects = <django.db.models.manager.Manager object>

class fpiweb.models.**LocBin**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Location Bin in warehouse.

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception MultipleObjectsReturned
Bases: django.core.exceptions.MultipleObjectsReturned

id
Internal record identifier for the location bin.

id_help_text = 'Internal record id for location bin.'

loc_bin
Location Bin Designation

loc_bin_descr
Location Bin Description

loc_bin_descr_help_text = 'Location bin description'

loc_bin_descr_max_length = 20

loc_bin_help_text = 'Location bin designation'

loc_bin_max_length = 2

loc_bin_min_length = 2

location_set
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

objects = <django.db.models.manager.Manager object>

class fpiweb.models.**LocRow**(**args*, ***kwargs*)
Bases: `django.db.models.base.Model`

Location Row in warehouse.

exception DoesNotExist
Bases: `django.core.exceptions.ObjectDoesNotExist`

exception MultipleObjectsReturned
Bases: `django.core.exceptions.MultipleObjectsReturned`

id
Internal record identifier for the location row.

id_help_text = 'Internal record id for location row.'

loc_row
Location Row Designation

loc_row_descr
Location Row Description

loc_row_descr_help_text = 'Location row description'

loc_row_descr_max_length = 20

loc_row_help_text = 'Location row designation'

loc_row_max_length = 2

loc_row_min_length = 2

location_set
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

objects = <django.db.models.manager.Manager object>

class fpiweb.models.**LocTier**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Location Tier in warehouse.

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception MultipleObjectsReturned
Bases: django.core.exceptions.MultipleObjectsReturned

id
Internal record identifier for the location tier.

id_help_text = 'Internal record id for location tier.'

loc_tier
Location Tier Designation

loc_tier_descr
Location Tier Description

loc_tier_descr_help_text = 'Location tier description'

loc_tier_descr_max_length = 20

loc_tier_help_text = 'Location tier designation'

loc_tier_max_length = 2

loc_tier_min_length = 2

location_set
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Parent.children is a ReverseManyToOneDescriptor instance.

Most of the implementation is delegated to a dynamically defined manager class built by create_forward_many_to_many_manager() defined below.

objects = <django.db.models.manager.Manager object>

class fpiweb.models.**Location**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Location for a filled box or other container.

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception MultipleObjectsReturned
Bases: django.core.exceptions.MultipleObjectsReturned

box_set
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

static get_location(*loc_row: Union[fpiweb.models.LocRow, int, str]*, *loc_bin: Union[fpiweb.models.LocBin, int, str]*, *loc_tier: Union[fpiweb.models.LocTier, int, str]*)

This method originated as convenient way to retrieve a Location from the database inside a test. :param loc_row: :param loc_bin: :param loc_tier: :return: Location or None

id

Internal record identifier for location.

id_help_text = 'Internal record identifier for location.'

loc_bin

Bin indicator of this location.

loc_bin_help_text = 'Loc bin'

loc_bin_id

loc_code

Coded Location.

loc_code_help_text = 'Location code'

loc_code_max_length = 12

loc_descr

Location description.

loc_descr_help_text = 'Location description'

loc_descr_max_length = 25

loc_in_warehouse

Is this location inside the warehouse?

loc_in_warehouse_help_text = 'In warehouse?'

loc_row

Row indicator of this location.

loc_row_help_text = 'Loc row'

loc_row_id

loc_tier

Tier indicator of this location.

loc_tier_help_text = 'Loc tier'

loc_tier_id

objects = <django.db.models.manager.Manager object>

pallet_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

class fpiweb.models.**Pallet**(**args*, ***kwargs*)

Bases: `django.db.models.base.Model`

Temporary file to build up a list of boxes on a pallet.

exception DoesNotExist

Bases: `django.core.exceptions.ObjectDoesNotExist`

FILL: str = 'Fill'

MERGE: str = 'Merge'

MOVE: str = 'Move'

exception MultipleObjectsReturned

Bases: `django.core.exceptions.MultipleObjectsReturned`

PALLET_STATUS_CHOICES = (('Fill', 'Fill pallet for new location'), ('Merge', 'Merging

boxes

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

get_pallet_status_display(**, field=<django.db.models.fields.CharField: pallet_status>*)

id

Internal record identifier for a pallet.

id_help_text = 'Internal record identifier for a pallet.'

location

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

location_id

name

Name of pallet.

name_help_text = 'Name of pallet'

objects = <django.db.models.manager.Manager object>

pallet_status
Current status of pallet

pallet_status_help_text = 'Current status of pallet.'

profile_set
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

class fpiweb.models.**PalletBox**(**args*, ***kwargs*)
Bases: `django.db.models.base.Model`

Temporary file to hold the individual boxes for a pallet. The goal of this is to ensure that either a Box record has product, expiration, and location or it has no product, no expiration, and no location.

exception DoesNotExist
Bases: `django.core.exceptions.ObjectDoesNotExist`

MOVE: str = 'Move'

exception MultipleObjectsReturned
Bases: `django.core.exceptions.MultipleObjectsReturned`

NEW: str = 'New'

ORIGINAL: str = 'Original'

PALLET_BOX_STATUS_CHOICES = (('New', 'New box added'), ('Original', 'Box already here'

box
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

box_help_text = 'Internal record identifier for a box.'

box_id

box_number
Number printed in the label on the box.

box_status
Box on pallet status

box_status_help_text = 'Box on pallet status.'

exp_month_end
Optional emding month range of when the product expires, if filled.

exp_month_end_help_text = 'Optional ending month range of when the product expires, if

exp_month_start
Optional starting month range of when the product expires, if filled.

exp_month_start_help_text = 'Optional starting month range of when the product expires

exp_year
Year the product expires, if filled.

exp_year_help_text = 'Year the product expires, if filled.'

get_box_status_display(*, *field=<django.db.models.fields.CharField: box_status>*)

id
Internal record identifier for a pallet box.

id_help_text = 'Internal record identifier for a pallet box.'

objects = <django.db.models.manager.Manager object>

pallet
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

pallet_help_text = 'Internal record identifier for a pallet.'

pallet_id

product
Product contained in this box, if filled.

product_help_text = 'Product contained in this box, if filled.'

product_id

class fpiweb.models.**Product**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Product name and attributes. Oranges, Pineapple, Mixed Fruit are products within the Fruits category

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception MultipleObjectsReturned
Bases: django.core.exceptions.MultipleObjectsReturned

box_set
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

id
Internal record identifier for product.

id_help_text = 'Internal record identifier for product.'

objects = <django.db.models.manager.Manager object>

palletbox_set
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

prod_cat
Product category associated with this product.

prod_cat_help_text = 'Product category associated with this product.'

prod_cat_id

prod_name
Name of this product.

prod_name_help_text = 'Name of this product.'

prod_name_max_length = 30

productexample_set
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

class fpiweb.models.**ProductCategory**(**args*, ***kwargs*)
Bases: `django.db.models.base.Model`

Category or group of product. i.e. Tomato Soup, Canned Pasta, Fruits

exception DoesNotExist
Bases: `django.core.exceptions.ObjectDoesNotExist`

exception MultipleObjectsReturned
Bases: `django.core.exceptions.MultipleObjectsReturned`

id
Internal record identifier for product category.

id_help_text = 'Internal record identifier for product category.'

objects = <django.db.models.manager.Manager object>

prod_cat_descr
Description of this product category.

prod_cat_descr_help_text = 'Description of this product category.'

prod_cat_name
Name of this product category.

prod_cat_name_help_text = 'Name of this product category.'

prod_cat_name_max_length = 30

product_set
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

class fpiweb.models.**ProductExample**(**args*, ***kwargs*)
Bases: `django.db.models.base.Model`

Examples of items that go into a labeled product.

exception DoesNotExist
Bases: `django.core.exceptions.ObjectDoesNotExist`

exception MultipleObjectsReturned
Bases: `django.core.exceptions.MultipleObjectsReturned`

id
Internal reccord identifier for product example

id_help_text = 'Internal record identifier for product example'

objects = <django.db.models.manager.Manager object>

prod_example_name
Name of example product.

prod_example_name_help_text = 'Name of example product.'

prod_example_name_max_length = 30

product
Product with which this product example is associated.

product_help_text = 'Product with which this product name is associated.'

product_id

class fpiweb.models.**Profile**(**args*, ***kwargs*)
Bases: `django.db.models.base.Model`

Track more information about the users of our system.

exception DoesNotExist
Bases: `django.core.exceptions.ObjectDoesNotExist`

exception MultipleObjectsReturned
Bases: `django.core.exceptions.MultipleObjectsReturned`

active_pallet
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

active_pallet_help_text = 'Active Pallet'

active_pallet_id

id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = <django.db.models.manager.Manager object>

title
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

title_help_text = 'Job title'

title_max_length = 30

user
Internal link to the default or custom Django User table.

user_id

fpiweb.password_validation module

class fpiweb.password_validation.**CurrentMonthInPasswordValidator**
Bases: `object`

get_help_text()

validate(*password*, *user=None*)

class fpiweb.password_validation.**ShortPasswordValidator**(*length_threshold=12*)
Bases: `object`

get_help_text()

validate(*password*, *user=None*)

class fpiweb.password_validation.**WarmInPasswordValidator**
Bases: `object`

get_help_text()

validate(*password*, *user=None*)

fpiweb.qr_code_utilities module

qr_code_utilities.py - support code for scanning QR codes.

fpiweb.qr_code_utilities.**get_qr_code_svg**(*data_string*, *include_xml_declaration=False*)
Manages svg images

Parameters **include_xml_declaration** –

Returns

fpiweb.urls module

Manage the urls for the F. P. I. application.

fpiweb.views module

views.py - establish the views (pages) for the F. P. I. web application.

class fpiweb.views.**AboutView**(***kwargs*)
Bases: django.views.generic.base.TemplateView

The About View for this application.

context_object_name = 'about_context'

get_context_data(***kwargs*)
Add Site Information to About page.

Parameters **kwargs** –

Returns

template_name = 'fpiweb/about.html'

class fpiweb.views.**ActivityDownloadView**(***kwargs*)
Bases: django.contrib.auth.mixins.PermissionRequiredMixin, django.views.generic.base.View

class Echo
Bases: object

An object that implements just the write method of the file-like interface.

static write(*value*)
Write the value by returning it, instead of storing in a buffer.
Parameters **value** –
Returns

date_format = '%m/%d/%Y'

get(*request*, **args*, ***kwargs*)

permission_required = ('fpiweb.view_activity',)

write_rows()

class fpiweb.views.**BoxDetailsView**(***kwargs*)
Bases: django.contrib.auth.mixins.PermissionRequiredMixin, django.views.generic.detail.DetailView

context_object_name = 'box'

get_context_data(***kwargs*)
Insert the single object into the context dict.

model
alias of *fpiweb.models.Box*

permission_required = ('fpiweb.dummy_profile',)

template_name = 'fpiweb/box_detail.html'

class fpiweb.views.**BoxEditView**(***kwargs*)
Bases: django.contrib.auth.mixins.PermissionRequiredMixin, django.views.generic.edit.UpdateView

context_object_name = 'box'

form_class
alias of *fpiweb.forms.NewBoxForm*

model
alias of *fpiweb.models.Box*

permission_required = ('fpiweb.dummy_profile',)

success_url = '/fpiweb/index/'

template_name = 'fpiweb/box_edit.html'

class fpiweb.views.**BoxEmptyMoveView**(***kwargs*)
Bases: django.contrib.auth.mixins.PermissionRequiredMixin, django.views.generic.base.TemplateView

get_context_data(***kwargs*)

permission_required = ('fpiweb.dummy_profile',)

template_name = 'fpiweb/box_empty_move.html'

class fpiweb.views.**BoxEmptyView**(***kwargs*)
Bases: django.contrib.auth.mixins.PermissionRequiredMixin, django.views.generic.base.View

permission_required = ('fpiweb.dummy_profile',)

class fpiweb.views.**BoxItemFormView**(***kwargs*)
Bases: django.contrib.auth.mixins.PermissionRequiredMixin, django.views.generic.base.View

static get_form(*pallet_box*, *prefix=None*)

permission_required = ('fpiweb.add_box',)

post(*request*)

template_name = 'fpiweb/box_form.html'

class fpiweb.views.**BoxMoveView**(***kwargs*)
Bases: django.contrib.auth.mixins.PermissionRequiredMixin, django.views.generic.base.TemplateView

get_context_data(***kwargs*)

permission_required = ('fpiweb.dummy_profile',)

template_name = 'fpiweb/box_empty_move.html'

class fpiweb.views.**BoxNewView**(***kwargs*)
Bases: django.contrib.auth.mixins.PermissionRequiredMixin, django.views.generic.base.View

get(*request*, **args*, ***kwargs*)
Prepare to present the new box view.

Parameters

- **request** –
- **args** –
- **kwargs** –

Returns

permission_required = ('fpiweb.dummy_profile',)

post(*request*, **args*, ***kwargs*)

template_name = 'fpiweb/box_new.html'

class fpiweb.views.**BoxScannedView**(***kwargs*)
Bases: django.contrib.auth.mixins.PermissionRequiredMixin, django.views.generic.base.View

get(*request*, ***kwargs*)

permission_required = ('fpiweb.dummy_profile',)

class fpiweb.views.**BoxTypeMaintenanceCreateView**(***kwargs*)
Bases: django.contrib.auth.mixins.PermissionRequiredMixin, django.contrib.messages.views.SuccessMessageMixin, django.views.generic.edit.CreateView

context_object_name = 'box_type_maintenance'

form_class
alias of *fpiweb.forms.BoxTypeMaintenanceForm*

get_context_data(***kwargs*)
Add info needed for the navigation bar

Parameters **kwargs** –

Returns

model
alias of *fpiweb.models.BoxType*

permission_required = 'fpiweb.add_box_type_maintenance'

success_message = ' A new BoxType has been successfully added.'

success_url = '/fpiweb/box_type_maintenance/'

template_name = 'fpiweb/box_type_maintenance_edit.html'

class fpiweb.views.**BoxTypeMaintenanceDeleteView**(***kwargs*)
Bases: django.contrib.auth.mixins.PermissionRequiredMixin, django.views.generic.edit.DeleteView

context_object_name = 'box_type_maintenance'

delete(*request*, **args*, ***kwargs*)
Call the delete() method on the fetched object and then redirect to the success URL.

error_message = 'Unable to Delete this Box Type. Before deleting this Box Type it is n

form_class
alias of *fpiweb.forms.BoxTypeMaintenanceForm*

get_context_data(***kwargs*)
Add info needed for the navigation bar

Parameters **kwargs** –

Returns

model
alias of *fpiweb.models.BoxType*

permission_required = 'fpiweb.delete_box_type_maintenance'

success_message = 'Box Type successfully deleted.'

success_url = '/fpiweb/box_type_maintenance/'

template_name = 'fpiweb/box_type_maintenance_delete.html'

class fpiweb.views.**BoxTypeMaintenanceListView**(***kwargs*)
Bases: django.contrib.auth.mixins.PermissionRequiredMixin, django.views.generic.list.ListView

context_object_name = 'box_type_maintenance_list_content'

get_context_data(***kwargs*)
Add info needed for the navigation bar

Parameters **kwargs** –

Returns

model
alias of *fpiweb.models.BoxType*

permission_required = 'fpiweb.view_box_type'

template_name = 'fpiweb/box_type_maintenance_list.html'

class fpiweb.views.**BoxTypeMaintenanceUpdateView**(***kwargs*)
Bases: django.contrib.auth.mixins.PermissionRequiredMixin, django.contrib.messages.views.SuccessMessageMixin, django.views.generic.edit.UpdateView

context_object_name = 'box_type_maintenance'

form_class
alias of *fpiweb.forms.BoxTypeMaintenanceForm*

get_context_data(***kwargs*)
Add info needed for the navigation bar

Parameters **kwargs** –

Returns

model
alias of *fpiweb.models.BoxType*

permission_required = 'fpiweb.change_box_type_maintenance'

success_message = 'The Box Type has been successfully updated.'

success_url = '/fpiweb/box_type_maintenance/'

template_name = 'fpiweb/box_type_maintenance_edit.html'

exception fpiweb.views.**BuildPalletError**
Bases: RuntimeError

class fpiweb.views.**BuildPalletView**(***kwargs*)
Bases: django.contrib.auth.mixins.PermissionRequiredMixin, django.views.generic.base.View

Manage preparing a pallet of bxes of product to store in the warehouse.

BoxFormFactory
alias of django.forms.formsets.BoxItemFormFormSet

PALLET_NAME_FORM_NAME = 'pallet_name_form'

PALLET_SELECT_FORM_NAME = 'pallet_select_form'

build_pallet_form_prefix = 'build_pallet'

confirmation_template = 'fpiweb/build_pallet_confirmation.html'

form_template = 'fpiweb/build_pallet.html'

formset_prefix = 'box_forms'

get(*request*)
Show page to select/add new pallet.

This page will POST back to this view.

Parameters **request** – info from Django about this page to display

Returns prepared web page

hidden_pallet_form_prefix = 'pallet'

page_title = 'Build Pallet'

permission_required = ('fpiweb.build_pallet',)

post(*request*)

static prepare_pallet_and_pallet_boxes(*pallet_form*, *build_pallet_form*, *box_forms*)

process_build_pallet_forms(*request*)

show_forms_response(*request*, *build_pallet_form*, *box_forms*, *pallet_form*, *status=<HTTPStatus.BAD_REQUEST: 400>*)
Display page with BuildPalletForm and BoxItemForms :param request: :param build_pallet_form: :param box_forms: :param pallet_form: :param status: :return:

static show_pallet_management_page(*request*, *pallet_select_form=None*, *pallet_name_form=None*, *status_code=<HTTPStatus.OK: 200>*)
Prepare info to name or select a pallet.

Note that this is invoking show_page in a different class.

Parameters

- **request** – Info from Django
- **pallet_select_form** – Form to use to select an existing pallet
- **pallet_name_form** – form to use to name a new pallet
- **status_code** – status code to send to browser

Returns prepated web page

class fpiweb.views.**ConfirmPasswordChangeView**(***kwargs*)
Bases: django.contrib.auth.mixins.LoginRequiredMixin, django.views.generic.base.View

Confirm the password was successfully changed.

get(*request*, **args*, ***kwargs*)
Add user info for the template.

Parameters

- **request** –
- **args** –
- **kwargs** –

Returns

success_url = '/fpiweb/index/'

template_name = 'fpiweb/confirm_password_change.html'

class fpiweb.views.**ConstraintCreateView**(***kwargs*)
Bases: django.contrib.auth.mixins.PermissionRequiredMixin, django.views.generic.edit.CreateView

Create a constraint using a generic CreateView.

context_object_name = 'constraints'

fields = ['constraint_name', 'constraint_descr', 'constraint_type', 'constraint_min',

formClass
alias of *fpiweb.forms.ConstraintsForm*

get_context_data(***kwargs*)
Modify the context before rendering the template.

Parameters **kwargs** –

Returns

model
alias of *fpiweb.models.Constraints*

permission_required = ('fpiweb.add_constraints',)

success_url = '/fpiweb/constraints/'

template_name = 'fpiweb/constraint_edit.html'

class fpiweb.views.**ConstraintDeleteView**(***kwargs*)
Bases: django.contrib.auth.mixins.PermissionRequiredMixin, django.views.generic.edit.DeleteView

Delete a constraint using a generic DeleteView.

context_object_name = 'constraints'

form_class
alias of *fpiweb.forms.ConstraintsForm*

get_context_data(***kwargs*)
Modify the context before rendering the template.

Parameters **kwargs** –

Returns

model
alias of *fpiweb.models.Constraints*

permission_required = ('fpiweb.delete_constraints',)

success_url = '/fpiweb/constraints/'

template_name = 'fpiweb/constraint_delete.html'

class fpiweb.views.**ConstraintUpdateView**(***kwargs*)
Bases: django.contrib.auth.mixins.PermissionRequiredMixin, django.views.generic.edit.UpdateView

Update a constraint using a generic UpdateView.

context_object_name = 'constraints'

form_class
alias of *fpiweb.forms.ConstraintsForm*

get_context_data(***kwargs*)
Modify the context before rendering the template.

Parameters **kwargs** –

Returns

model
alias of *fpiweb.models.Constraints*

permission_required = ('fpiweb.change_constraints',)

success_url = '/fpiweb/constraints/'

template_name = 'fpiweb/constraint_edit.html'

class fpiweb.views.**ConstraintsListView**(***kwargs*)
Bases: django.contrib.auth.mixins.PermissionRequiredMixin, django.views.generic.list.ListView

List of existing constraints.

context_object_name = 'constraints_list_content'

get_context_data(**, object_list=None, **kwargs*)
Add additional content to the context dictionary.

Parameters

- **object_list** –
- **kwargs** –

Returns

model
alias of *fpiweb.models.Constraints*

permission_required = ('fpiweb.view_constraints',)

template_name = 'fpiweb/constraints_list.html'

class fpiweb.views.**IndexView**(***kwargs*)
Bases: django.contrib.auth.mixins.LoginRequiredMixin, django.views.generic.base.TemplateView

Default web page (/index)

get_context_data(***kwargs*)
Add info needed for the navigation bar

Parameters **kwargs** –

Returns

template_name = 'fpiweb/index.html'

class fpiweb.views.**LocBinCreateView**(***kwargs*)
Bases: django.contrib.auth.mixins.PermissionRequiredMixin, django.views.generic.edit.CreateView

Create a bin using a generic CreateView.

context_object_name = 'loc_bin'

form_class
alias of *fpiweb.forms.LocBinForm*

get_context_data(***kwargs*)
Modify the context before rendering the template.

Parameters **kwargs** –

Returns

model
alias of *fpiweb.models.LocBin*

permission_required = ('fpiweb.add_locbin',)

success_url = '/fpiweb/loc_bin/'

template_name = 'fpiweb/loc_bin_edit.html'

class fpiweb.views.**LocBinDeleteView**(***kwargs*)
Bases: django.contrib.auth.mixins.PermissionRequiredMixin, django.views.generic.edit.DeleteView

Delete a bin using a generic DeleteView.

context_object_name = 'loc_bin'

form_class
alias of *fpiweb.forms.LocBinForm*

get_context_data(***kwargs*)
Modify the context before rendering the template.

Parameters **kwargs** –

Returns

model
alias of *fpiweb.models.LocBin*

permission_required = ('fpiweb.delete_locbin',)

success_url = '/fpiweb/loc_bin/'

template_name = 'fpiweb/loc_bin_delete.html'

class fpiweb.views.**LocBinListView**(***kwargs*)
Bases: django.contrib.auth.mixins.PermissionRequiredMixin, django.views.generic.list.ListView

List of existing bins using a generic ListView.

context_object_name = 'loc_bin_list_content'

get_context_data(***kwargs*)
Modify the context before rendering the template.

Parameters **kwargs** –

Returns

model
alias of *fpiweb.models.LocBin*

permission_required = ('fpiweb.view_locbin',)

template_name = 'fpiweb/loc_bin_list.html'

class fpiweb.views.**LocBinUpdateView**(***kwargs*)
Bases: django.contrib.auth.mixins.PermissionRequiredMixin, django.views.generic.edit.UpdateView

Update a bin using a generic UpdateView.

context_object_name = 'loc_bin'

form_class
alias of *fpiweb.forms.LocBinForm*

get_context_data(***kwargs*)
Modify the context before rendering the template.

Parameters **kwargs** –

Returns

model
alias of *fpiweb.models.LocBin*

permission_required = ('fpiweb.change_locbin',)

success_url = '/fpiweb/loc_bin/'

template_name = 'fpiweb/loc_bin_edit.html'

class fpiweb.views.**LocRowCreateView**(***kwargs*)
Bases: django.contrib.auth.mixins.PermissionRequiredMixin, django.views.generic.edit.CreateView

Create a row using a generic CreateView.

context_object_name = 'loc_row'

form_class
alias of *fpiweb.forms.LocRowForm*

get_context_data(***kwargs*)
Modify the context before rendering the template.

Parameters **kwargs** –

Returns

model
alias of *fpiweb.models.LocRow*

permission_required = ('fpiweb.add_locrow',)

success_url = '/fpiweb/loc_row/'

template_name = 'fpiweb/loc_row_edit.html'

class fpiweb.views.**LocRowDeleteView**(***kwargs*)
Bases: django.contrib.auth.mixins.PermissionRequiredMixin, django.views.generic.edit.DeleteView

Delete a row using a generic DeleteView.

context_object_name = 'loc_row'

form_class
alias of *fpiweb.forms.LocRowForm*

get_context_data(***kwargs*)
Modify the context before rendering the template.

Parameters **kwargs** –

Returns

model
alias of *fpiweb.models.LocRow*

permission_required = ('fpiweb.delete_locrow',)

success_url = '/fpiweb/loc_row/'

template_name = 'fpiweb/loc_row_delete.html'

class fpiweb.views.**LocRowListView**(***kwargs*)
Bases: django.contrib.auth.mixins.PermissionRequiredMixin, django.views.generic.list.ListView

List of existing rows using a generic ListView.

context_object_name = 'loc_row_list_content'

get_context_data(***kwargs*)
Modify the context before rendering the template.

Parameters **kwargs** –

Returns

model
alias of *fpiweb.models.LocRow*

permission_required = ('fpiweb.view_locrow',)

template_name = 'fpiweb/loc_row_list.html'

class fpiweb.views.**LocRowUpdateView**(***kwargs*)
Bases: django.contrib.auth.mixins.PermissionRequiredMixin, django.views.generic.edit.UpdateView

Update a row using a generic UpdateView.

context_object_name = 'loc_row'

form_class
alias of *fpiweb.forms.LocRowForm*

get_context_data(***kwargs*)
Modify the context before rendering the template.

Parameters **kwargs** –

Returns

model
alias of *fpiweb.models.LocRow*

permission_required = ('fpiweb.change_locrow',)

success_url = '/fpiweb/loc_row/'

template_name = 'fpiweb/loc_row_edit.html'

class fpiweb.views.**LocTierCreateView**(***kwargs*)
Bases: django.contrib.auth.mixins.PermissionRequiredMixin, django.views.generic.edit.CreateView

Create a tier using a generic CreateView.

context_object_name = 'loc_tier'

form_class
alias of *fpiweb.forms.LocTierForm*

get_context_data(***kwargs*)
Modify the context before rendering the template.

Parameters **kwargs** –

Returns

model
alias of *fpiweb.models.LocTier*

permission_required = ('fpiweb.add_loctier',)

success_url = '/fpiweb/loc_tier/'

template_name = 'fpiweb/loc_tier_edit.html'

class fpiweb.views.**LocTierDeleteView**(***kwargs*)
Bases: django.contrib.auth.mixins.PermissionRequiredMixin, django.views.generic.edit.DeleteView

Delete a tier using a generic DeleteView.

context_object_name = 'loc_tier'

form_class
alias of *fpiweb.forms.LocTierForm*

get_context_data(***kwargs*)
Modify the context before rendering the template.

Parameters **kwargs** –

Returns

model
alias of *fpiweb.models.LocTier*

permission_required = ('fpiweb.delete_loctier',)

success_url = '/fpiweb/loc_tier/'

template_name = 'fpiweb/loc_tier_delete.html'

class fpiweb.views.**LocTierListView**(***kwargs*)
Bases: django.contrib.auth.mixins.PermissionRequiredMixin, django.views.generic.list.ListView

List of existing tiers using a generic ListView.

context_object_name = 'loc_tier_list_content'

get_context_data(***kwargs*)
Modify the context before rendering the template.

Parameters **kwargs** –

Returns

model
alias of *fpiweb.models.LocTier*

permission_required = ('fpiweb.view_loctier',)

template_name = 'fpiweb/loc_tier_list.html'

class fpiweb.views.**LocTierUpdateView**(***kwargs*)
Bases: django.contrib.auth.mixins.PermissionRequiredMixin, django.views.generic.edit.UpdateView

Update a tier using a generic UpdateView.

context_object_name = 'loc_tier'

form_class
alias of *fpiweb.forms.LocTierForm*

get_context_data(***kwargs*)
Modify the context before rendering the template.

Parameters **kwargs** –

Returns

model
alias of *fpiweb.models.LocTier*

permission_required = ('fpiweb.change_loctier',)

success_url = '/fpiweb/loc_tier/'

template_name = 'fpiweb/loc_tier_edit.html'

class fpiweb.views.**MANUAL_NOTICE_TYPE**(*value*)
Bases: enum.Enum

Manual generic notice type.

NOTICE: str = 'NOTICE'

QUESTION: str = 'QUESTION'

class fpiweb.views.**MaintenanceView**(***kwargs*)
Bases: django.contrib.auth.mixins.PermissionRequiredMixin, django.views.generic.base.TemplateView

Default web page (/index)

get_context_data(***kwargs*)
Modify the context before rendering the template.

Parameters **kwargs** –

Returns

permission_required = ('fpiweb.view_system_maintenance',)

template_name = 'fpiweb/system_maintenance.html'

class fpiweb.views.**ManualBoxStatusView**(***kwargs*)
Bases: django.contrib.auth.mixins.PermissionRequiredMixin, django.views.generic.base.View

MODE_CONFIRMATION = 'confirmation'

MODE_ENTER_BOX_NUMBER = 'enter_box_number'

static build_context(**, mode, box_number_form=None, box=None, box_type=None, product_form=None, product=None, location_form=None, location=None, errors=None*)

get(*request*, **args*, ***kwargs*)
Prepare to display consume box number form for the first time.

Parameters

- **request** –
- **args** –
- **kwargs** –

Returns

permission_required = ('fpiweb.view_box',)

post(*request*, **args*, ***kwargs*)

post_box_number(*request*)

template_name = 'fpiweb/manual_box_status.html'

class fpiweb.views.**ManualCheckinBoxView**(***kwargs*)
Bases: django.contrib.auth.mixins.PermissionRequiredMixin, django.views.generic.base.View

Manually check a box into inventory.

MODE_CONFIRMATION = 'confirmation'

MODE_ENTER_BOX_INFO = 'enter_box_info'

static build_context(**, mode, box_number_form=None, box_number=None, box=None, product_form=None, product=None, location_form=None, location=None, exp_year_form=None, exp_year=None, exp_month_start_form=None, exp_month_start=None, exp_month_end_form=None, exp_month_end=None, errors: Optional[list] = None*)

get(*request*, **args*, ***kwargs*)

permission_required = ('fpiweb.check_in_box',)

post(*request*, **args*, ***kwargs*)

post_box_info(*request*)
Validate the posted information.

Parameters **request** – container of initial or latest post

Returns

template_name = 'fpiweb/manual_check_in_box.html'

class fpiweb.views.**ManualConsumeBoxView**(***kwargs*)
Bases: django.contrib.auth.mixins.PermissionRequiredMixin, django.views.generic.base.View

MODE_CONFIRMATION = 'confirmation'

MODE_CONSUME_BOX = 'consume_box'

MODE_ENTER_BOX_NUMBER = 'enter_box_number'

static build_context(**, mode, box_number_form=None, box=None, box_type=None, product_form=None, product=None, location_form=None, location=None, errors: Optional[list] = None*)

get(*request*, **args*, ***kwargs*)
Prepare to display consume box number form for the first time.

Parameters

- **request** –
- **args** –
- **kwargs** –

Returns

permission_required = ('fpiweb.check_out_box',)

post(*request*, **args*, ***kwargs*)

post_box_number(*request*)

post_consume_box(*request*)

template_name = 'fpiweb/manual_check_out_box.html'

class fpiweb.views.**ManualLocTableCreateView**(***kwargs*)
Bases: django.contrib.auth.mixins.PermissionRequiredMixin, django.views.generic.edit.CreateView

Create a new entry for Location Table using a generic CreateView.

Added by Mike Rehner

context_object_name = 'manual_loc_table'

form_class
alias of *fpiweb.forms.ManualLocTableForm*

get_context_data(***kwargs*)
Add info needed for the navigation bar

Parameters **kwargs** –

Returns

model
alias of *fpiweb.models.Location*

permission_required = ('fpiweb.add_manual_location_table',)

success_url = '/fpiweb/manual_loc_table/'

template_name = 'fpiweb/manual_loc_table_edit.html'

class fpiweb.views.**ManualLocTableListView**(***kwargs*)
Bases: django.contrib.auth.mixins.PermissionRequiredMixin, django.views.generic.list.ListView

List the location table entries so they can be edited as needed.

Added by Mike Rehner

context_object_name = 'manual_loc_table'

get_context_data(***kwargs*)
Add info needed for the navigation bar

Parameters **kwargs** –

Returns

model
alias of *fpiweb.models.Location*

permission_required = ('fpiweb.view_manual_loc_table',)

template_name = 'fpiweb/manual_loc_table_list.html'

class fpiweb.views.**ManualLocTableUpdateView**(***kwargs*)
Bases: django.contrib.auth.mixins.PermissionRequiredMixin, django.views.generic.edit.UpdateView

Update a Rebuild Location Table using a generic UpdateView.

context_object_name = 'manual_loc_table'

form_class
alias of *fpiweb.forms.ManualLocTableForm*

get_context_data(***kwargs*)
Add info needed for the navigation bar

Parameters **kwargs** –

Returns

model
alias of *fpiweb.models.Location*

permission_required = ('fpiweb.change_manual_loc_table',)

success_url = '/fpiweb/manual_loc_table/'

template_name = 'fpiweb/manual_loc_table_edit.html'

class fpiweb.views.**ManualMoveBoxView**(***kwargs*)
Bases: django.contrib.auth.mixins.PermissionRequiredMixin, django.views.generic.base.View

MODE_CONFIRMATION = 'confirmation'

MODE_ENTER_BOX_NUMBER = 'enter_box_number'

MODE_ENTER_LOCATION = 'enter_location'

static build_context(*, *mode*, *box_number_form=None*, *box=None*, *location_form=None*, *errors: Optional[list] = None*)

get(*request*, **args*, ***kwargs*)

permission_required = ('fpiweb.move_box',)

post(*request*, **args*, ***kwargs*)

post_box_number(*request*)

post_location(*request*)

template_name = 'fpiweb/manual_move_box.html'

class fpiweb.views.**ManualNewBoxView**(***kwargs*)

Bases: django.contrib.auth.mixins.PermissionRequiredMixin, django.views.generic.base.View

MODE_CONFIRMATION = 'confirmation'

MODE_ENTER_BOX_NUMBER = 'enter_box_number'

static build_context(*, *mode*, *box_number_form=None*, *box=None*, *box_type=None*, *box_type_form=None*, *errors: Optional[list] = None*)

get(*request*, **args*, ***kwargs*)

Prepare to display add new box number form for the first time.

Parameters

- **request** –
- **args** –
- **kwargs** –

Returns

permission_required = ('fpiweb.add_box',)

post(*request*, **args*, ***kwargs*)

post_box_number(*request*)

template_name = 'fpiweb/manual_new_box.html'

class fpiweb.views.**ManualPalletMoveView**(***kwargs*)

Bases: django.contrib.auth.mixins.PermissionRequiredMixin, django.views.generic.base.View

FORM_PREFIX_CONFIRM_MERGE = 'confirm_merge'

FORM_PREFIX_FROM_LOCATION = 'from'

FORM_PREFIX_TO_LOCATION = 'to'

MODE_COMPLETE = 'complete'

MODE_CONFIRM_MERGE = 'confirm merge'

MODE_ENTER_FROM_LOCATION = 'enter from location'

MODE_ENTER_TO_LOCATION = 'enter to location'

build_response(*request*, *mode*, *from_location_form=None*, *to_location_form=None*, *confirm_merge_form=None*, *boxes_moved=0*, *to_location=None*, *errors=None*, *status=<HTTPStatus.OK: 200>*)

get(*request*)

static get_next_temp_name()

static get_pallet_and_box_count(*from_location*, *to_location*)

move_boxes(*request*, *from_location*, *to_location*)

permission_required = ('fpiweb.move_pallet',)

post(*request*)

post_confirm_merge_form(*request*)

post_from_location_form(*request*)

post_to_location_form(*request*)

show_to_location_form(*request*, *from_location*)

template = 'fpiweb/manual_pallet_move.html'

class fpiweb.views.**ManualPalletStatus**(***kwargs*)

Bases: django.contrib.auth.mixins.PermissionRequiredMixin, django.views.generic.list.ListView

Show the status of a pallet.

context_object_name = 'manual_pallet_status'

get_(***kwargs*)

Add Site Information to About page.

Parameters **kwargs** –

Returns

model

alias of *fpiweb.models.Pallet*

permission_required = ('fpiweb.dummy_profile',)

success_url = '/fpiweb/index/'

template_name = 'fpiweb/manual_pallet_status.html'

class fpiweb.views.**PalletManagementView**(***kwargs*)

Bases: django.contrib.auth.mixins.PermissionRequiredMixin, django.views.generic.base.View

Select current pallet, add new pallet, delete pallet

get(*request*)

permission_required = ('fpiweb.dummy_profile',)

static show_page(*request*, *page_title='Pallet Management'*, *show_delete=True*, *prompt=None*, *pallet_select_form=None*, *pallet_name_form=None*, *status_code=<HTTPStatus.OK: 200>*)

Prepare web page from the info received.

Parameters

- **request** – infor from Django
- **page_title** – title to put in the browser tab
- **show_delete** – should a delete option be displayee?
- **prompt** – suggestion to the user about what to do

- **pallet_select_form** – form to use to select an existing pallet
- **pallet_name_form** – form to use to name a new pallet
- **status_code** – status code to send to the browser

Returns a fully rendered web page to send to the browser

template_name = 'fpiweb/pallet_management.html'

class fpiweb.views.**PalletSelectView**(***kwargs*)
Bases: django.contrib.auth.mixins.PermissionRequiredMixin, django.views.generic.edit.FormView

form_class
alias of *fpiweb.forms.PalletSelectForm*

form_valid(*form*)
If the form is valid, redirect to the supplied URL.

get_context_data(***kwargs*)
Add info needed for the navigation bar

Parameters **kwargs** –

Returns

permission_required = ('fpiweb.dummy_profile',)

success_url = '/fpiweb/index/'

template_name = 'fpiweb/pallet_select.html'

class fpiweb.views.**ProductCategoryCreateView**(***kwargs*)
Bases: django.contrib.auth.mixins.PermissionRequiredMixin, django.views.generic.edit.CreateView

Create a Product Category using a generic CreateView.

context_object_name = 'product_category'

form_class
alias of *fpiweb.forms.ProductCategoryForm*

get_context_data(***kwargs*)
Add info needed for the navigation bar

Parameters **kwargs** –

Returns

model
alias of *fpiweb.models.ProductCategory*

permission_required = ('fpiweb.add_product_category',)

success_url = '/fpiweb/product_category/'

template_name = 'fpiweb/product_category_edit.html'

class fpiweb.views.**ProductCategoryListView**(***kwargs*)
Bases: django.contrib.auth.mixins.PermissionRequiredMixin, django.views.generic.list.ListView

List of existing Product Categories using a generic ListView.

context_object_name = 'product_category_list_content'

get_context_data(***kwargs*)
Add info needed for the navigation bar

Parameters **kwargs** –

Returns

model
alias of *fpiweb.models.ProductCategory*

permission_required = ('fpiweb.view_product_category',)

template_name = 'fpiweb/product_category_list.html'

class fpiweb.views.**ProductCategoryUpdateView**(***kwargs*)
Bases: django.contrib.auth.mixins.PermissionRequiredMixin, django.views.generic.edit.UpdateView

Update a Product Category using a generic UpdateView.

context_object_name = 'product_category'

form_class
alias of *fpiweb.forms.ProductCategoryForm*

get_context_data(***kwargs*)
Add info needed for the navigation bar

Parameters **kwargs** –

Returns

model
alias of *fpiweb.models.ProductCategory*

permission_required = ('fpiweb.change_product_category',)

success_url = '/fpiweb/product_category/'

template_name = 'fpiweb/product_category_edit.html'

class fpiweb.views.**ProductExampleCreateView**(***kwargs*)
Bases: django.contrib.auth.mixins.PermissionRequiredMixin, django.contrib.messages.views.SuccessMessageMixin, django.views.generic.edit.CreateView

Create a Product Example using a generic CreateView.

context_object_name = 'product_example'

form_class
alias of *fpiweb.forms.ProductExampleForm*

get_context_data(***kwargs*)
Add info needed for the navigation bar

Parameters **kwargs** –

Returns

model
alias of *fpiweb.models.ProductExample*

permission_required = ('fpiweb.add_product_example',)

success_message = 'A new Product Example has been successfully added.'

success_url = '/fpiweb/product_example/'

template_name = 'fpiweb/product_example_edit.html'

class fpiweb.views.**ProductExampleDeleteView**(***kwargs*)
Bases: django.contrib.auth.mixins.PermissionRequiredMixin, django.views.generic.edit.DeleteView

Delete a Product Example using a generic DeleteView.

context_object_name = 'product_example'

delete(*request*, **args*, ***kwargs*)
Call the delete() method on the fetched object and then redirect to the success URL.

form_class
alias of *fpiweb.forms.ProductExampleForm*

get_context_data(***kwargs*)
Modify the context before rendering the template.

Parameters **kwargs** –

Returns

model
alias of *fpiweb.models.ProductExample*

permission_required = ('fpiweb.delete_product_example_name',)

success_message = 'The Product Example has been successfully deleted.'

success_url = '/fpiweb/product_example/'

template_name = 'fpiweb/product_example_delete.html'

class fpiweb.views.**ProductExampleListView**(***kwargs*)
Bases: django.contrib.auth.mixins.PermissionRequiredMixin, django.views.generic.list.ListView

List of existing Product Examples using a generic ListView.

context_object_name = 'product_example_list_content'

get_context_data(***kwargs*)
Add info needed for the navigation bar

Parameters **kwargs** –

Returns

model
alias of *fpiweb.models.ProductExample*

permission_required = ('fpiweb.view_product_example',)

template_name = 'fpiweb/product_example_list.html'

class fpiweb.views.**ProductExampleUpdateView**(***kwargs*)
Bases: django.contrib.auth.mixins.PermissionRequiredMixin, django.contrib.messages.views.SuccessMessageMixin, django.views.generic.edit.UpdateView

Update a Product Example using a generic UpdateView.

context_object_name = 'product_example'

form_class
alias of *fpiweb.forms.ProductExampleForm*

get_context_data(***kwargs*)
Add info needed for the navigation bar

Parameters **kwargs** –

Returns

model
alias of *fpiweb.models.ProductExample*

permission_required = ('fpiweb.change_product_example_name',)

success_message = 'The Product Example has been has been successfully updated.'

success_url = '/fpiweb/product_example/'

template_name = 'fpiweb/product_example_edit.html'

class fpiweb.views.**ProductNameCreateView**(***kwargs*)
Bases: django.contrib.auth.mixins.PermissionRequiredMixin, django.views.generic.edit.CreateView

Create a Product using a generic CreateView.

context_object_name = 'product_name'

form_class
alias of *fpiweb.forms.ProductNameForm*

get_context_data(***kwargs*)
Add info needed for the navigation bar

Parameters **kwargs** –

Returns

model
alias of *fpiweb.models.Product*

permission_required = ('fpiweb.add_product_name',)

success_url = '/fpiweb/product_name/'

template_name = 'fpiweb/product_name_edit.html'

class fpiweb.views.**ProductNameListView**(***kwargs*)
Bases: django.contrib.auth.mixins.PermissionRequiredMixin, django.views.generic.list.ListView

List of Products using a generic ListView.

context_object_name = 'product_name_list_content'

get_context_data(***kwargs*)
Add info needed for the navigation bar

Parameters **kwargs** –

Returns

model
alias of *fpiweb.models.Product*

permission_required = ('fpiweb.view_product_name',)

template_name = 'fpiweb/product_name_list.html'

class fpiweb.views.**ProductNameUpdateView**(***kwargs*)
Bases: django.contrib.auth.mixins.PermissionRequiredMixin, django.views.generic.edit.UpdateView

Update a Product using a generic UpdateView.

context_object_name = 'product_name'

form_class
alias of *fpiweb.forms.ProductNameForm*

get_context_data(***kwargs*)
Add info needed for the navigation bar

Parameters **kwargs** –

Returns

model
alias of *fpiweb.models.Product*

permission_required = ('fpiweb.change_product_name',)

success_url = '/fpiweb/product_name/'

template_name = 'fpiweb/product_name_edit.html'

class fpiweb.views.**RebuildLocTableFinishView**(***kwargs*)
Bases: django.contrib.auth.mixins.PermissionRequiredMixin, django.views.generic.base.View

build_exclusion_constraint_list()

context_object_name = 'rebuild_loc_table'

get(*request*)

model
alias of *fpiweb.models.Location*

permission_required = ('fpiweb.view_constraints', 'fpiweb.view_locationfpiweb.add_loca

rebuild_location_table() → int

template_name = 'fpiweb/rebuild_loc_table_finish.html'

update_row_bin_tier_tables() → Tuple[int, int, int]

class fpiweb.views.**RebuildLocTableProgressView**(***kwargs*)
Bases: django.contrib.auth.mixins.PermissionRequiredMixin, django.views.generic.base.View

get(*request*)

permission_required = ('fpiweb.view_constraints', 'fpiweb.view_locationfpiweb.add_loca

class fpiweb.views.**RebuildLocTableStartView**(***kwargs*)
Bases: django.contrib.auth.mixins.PermissionRequiredMixin, django.views.generic.base.View

context_object_name = 'rebuild_loc_table'

get(*request*)

model
alias of *fpiweb.models.Location*

permission_required = ('fpiweb.view_constraints', 'fpiweb.view_locationfpiweb.add_loca

template_name = 'fpiweb/rebuild_loc_table_start.html'

class fpiweb.views.**ScannerView**(***kwargs*)
Bases: django.contrib.auth.mixins.PermissionRequiredMixin, django.views.generic.base.View

static error_response(*errors*, *status=<HTTPStatus.BAD_REQUEST: 400>*)

static get_box(*scan_data=None*, *box_number=None*)

static get_box_data(*scan_data=None*, *box_number=None*)

static get_keyed_in_box_number(*box_number*)

Parameters **box_number** – the box number (a string), may be None

Returns

permission_required = ('fpiweb.dummy_profile',)

post(*request*, **args*, ***kwargs*)

static response(*success*, *data=None*, *errors=None*, *status=<HTTPStatus.OK: 200>*)

exception fpiweb.views.**ScannerViewError**
Bases: RuntimeError

class fpiweb.views.**TestScanView**(***kwargs*)
Bases: django.contrib.auth.mixins.PermissionRequiredMixin, django.views.generic.base.TemplateView

static get_box_scanned_url(*box_number*)

static get_box_url_by_filters(***filters*)

get_context_data(***kwargs*)

permission_required = ('fpiweb.dummy_profile',)

template_name = 'fpiweb/test_scan.html'

class fpiweb.views.**UserCreateview**
Bases: django.contrib.auth.mixins.PermissionRequiredMixin, django.views.generic.edit.CreateView

Create a new user to this system.

static build_add_context(***, *mode*, *key=None*, *this_user_info=None*, *target_user=None*, *target_user_form=None*, *errors=None*)

fields = ['username', 'userpswd', 'confirm_pwd', 'force_password', 'first_name', 'last_

form_class
alias of *fpiweb.forms.UserInfoForm*

get(*request*, **args*, ***kwargs*)
Prepare to display list of users to change.

Parameters

- **request** –
- **args** –
- **kwargs** –

Returns

model = 'auth.User'

permission_required = ('fpiweb.view_system_maintenance',)

post(*request*, **args*, ***kwargs*)
Validate the new user info.

Parameters

- **request** –
- **args** –
- **kwargs** –

Returns

success_url = '/fpiweb/user_mgmt/'

template_name = 'fpiweb/user_edit.html'

class fpiweb.views.**UserManagementView**
Bases: django.contrib.auth.mixins.PermissionRequiredMixin, django.views.generic.base.View

Allow staff and administrators to manage any users access

static build_context(***, *mode*, *this_user_info=None*, *target_user_info=None*, *perm_level=None*, *user_info_list=None*, *user_list_count=None*, *errors=None*)

get(*request*, **args*, ***kwargs*)
Prepare to display list of users to change.

Parameters

- **request** –
- **args** –
- **kwargs** –

Returns

permission_required = ('fpiweb.view_system_maintenance',)

template_name = 'fpiweb/user_management.html'

class fpiweb.views.**UserUpdateView**
Bases: django.contrib.auth.mixins.PermissionRequiredMixin, django.views.generic.base.View

Update a user by someone else.

static build_update_context(***, *mode*, *key=None*, *this_user_info=None*, *target_user=None*, *target_user_form=None*, *errors=None*)

form_class
alias of *fpiweb.forms.UserInfoForm*

get(*request*, **args*, ***kwargs*)
Modify the context before rendering the template.

Parameters

- **request** –
- **args** –

- **kwargs** –

Returns

permission_required = ('fpiweb.view_system_maintenance',)

post(*request*, **args*, ***kwargs*)
Validate the new user info.

Parameters

- **request** –
- **args** –
- **kwargs** –

Returns

success_url = '/fpiweb/user_mgmt/'

template_name = 'fpiweb/user_edit.html'

fpiweb.views.**add_navbar_vars**(*user: Optional*, *context: Dict*) → Dict
Add context variables needed by the navigation bar.

Parameters

- **user** – user record (usually extracted from the request object
- **context** – current context

Returns modified context

fpiweb.views.**error_page**(*request*, *message=None*, *message_list=()*, *status=<HTTPStatus.BAD_REQUEST: 400>*)

fpiweb.views.**get_user_and_profile**(*request*)

fpiweb.views.**manual_generic_notification**(*request*, *note_type: fpiweb.views.MANUAL_NOTICE_TYPE*, *title: Optional[str] = None*, *message_list: tuple = ()*, *action_message: str = ''*, *yes_url: str = 'fpiweb:about'*, *no_url: str = 'fpiweb:about'*, *return_url: str = 'fpiweb:about'*, *status: int = <HTTPStatus.OK: 200>*)
Provide a generic notification screen for the manual box subsystem.

Parameters

- **request** – request info from calling view
- **note_type** – type of notice (error or note)
- **title** – title to use for notification
- **message_list** – List of lines to display in notification
- **action_message** – final message or question
- **yes_url** – if question, url for yes action
- **no_url** – if question, url for no action
- **return_url** – if notice, url to go to after notification
- **status** – status code to flag notification if needed

Returns

4.3 StandaloneTools

4.3.1 StandaloneTools package

Submodules

StandaloneTools.GenerateSecretKey module

GenerateSecretKey.py - program to generate a secret key.

StandaloneTools.LoadLocationData module

LoadLocationData.py - Load location data into Row/Bin/Tier and Location tables.

Although the Constraints table has the min/max or list values this program needs, the values are hard coded here because the records in the Constraints table are about to go away.

The philosophy used here is that there are certain essential rows, bins, and tiers that make up the components of warehouse locations. This program will add those rows, bins, tiers and locations to the database. This program doesn't care if the descriptions for these records have changed, but requires that records with specific keys must be present in the database. Once the database has been pre-loaded, the users are free to make changes as desired.

`StandaloneTools.LoadLocationData.`**`BIN_MAX = 9`**
Maximum bin number

`StandaloneTools.LoadLocationData.`**`BIN_MIN = 1`**
Minimum bin number

`StandaloneTools.LoadLocationData.`**`ECHO_SQL_TO_LOG = True`**
Indicator of if the SQL statements should be copied to the log

class `StandaloneTools.LoadLocationData.`**`LoadLocationDataClass`**
Bases: `object`

LoadLocationDataClass - Load location data into location tables.

class LocBin(***kwargs*)
Bases: `sqlalchemy.ext.automap.Base`

Location Bin table definition

class LocRow(***kwargs*)
Bases: `sqlalchemy.ext.automap.Base`

Location Row table definition

class LocTier(***kwargs*)
Bases: `sqlalchemy.ext.automap.Base`

Location Tier table definition

class Location(***kwargs*)
Bases: `sqlalchemy.ext.automap.Base`

Location table definition

connect(*user: str*, *password: str*, *db: str*, *host: str = 'localhost'*, *port: int = 5432*) → <module 'sqlalchemy.engine' from '/home/docs/checkouts/readthedocs.org/user_builds/food-pantry-inventory/envs/dev/lib/python3.8/site-packages/sqlalchemy/engine/__init__.py'>
Establish a connection to the desired PostgreSQL database.

Parameters

- **user** –
- **password** –
- **db** –
- **host** –
- **port** –

Returns

load_bin_table(*session: sessionmaker(class_='Session', bind=None, autoflush=True, autocommit=False, expire_on_commit=True)*)
Load the bin table with values from min to max.

Returns

load_location_table(*session: sessionmaker(class_='Session', bind=None, autoflush=True, autocommit=False, expire_on_commit=True)*)
Construct location records from the row/bin/tier records.

The location code consists of the row code, bin code, and tier code jammed together into a six character id.

Returns

load_row_table(*session: sessionmaker(class_='Session', bind=None, autoflush=True, autocommit=False, expire_on_commit=True)*)
Load the row table with values from min to max.

Session

Returns

load_tier_table(*session: sessionmaker(class_='Session', bind=None, autoflush=True, autocommit=False, expire_on_commit=True)*)
Load the tier table with values from the list.

Returns

run_load_loc_data()
Top method for running Load location data into tables.

Returns

class StandaloneTools.LoadLocationData.**Main**
Bases: object

Main class to start things rolling.

run_load_loc_data()
Prepare to run Load location data into tables.

Returns

static start_logging(*work_dir: pathlib.Path*, *debug_name: str*)
Establish the logging for all the other scripts.

Parameters

- **work_dir** –
- **debug_name** –

Returns (nothing)

StandaloneTools.LoadLocationData.**ROW_MAX = 4**
Maximum row number

StandaloneTools.LoadLocationData.**ROW_MIN = 1**
Minimum row number

StandaloneTools.LoadLocationData.**TIER_LIST = ['A1', 'A2', 'B1', 'B2', 'C1', 'C2']**
List of valid tier names

StandaloneTools.LoadLocationData.**log = None**
Constants

StandaloneTools.LoadLocationData.**session_scope**()
Provide a transactional scope around a series of operations.

StandaloneTools.dump_group_permissions module

CHAPTER

FIVE

RESTRUCTURED TEXT EXAMPLES

This document contains examples of some of the techniques used by reStructuredText (rST) to enhance the plain text of the documentation. Although other references will describe additional markup syntax, the examples given here have been vetted to work as expected after being processed by Sphinx (at least with Sphinx 2.2.4). A single page cheatsheet is also available.

Note that some *rST* editors, such as PyCharm, ReText, etc. may not understand or display markup in the same way that Sphinx does. They might display error messages that end up being bogus. The final Sphinx output will be correct.

5.1 Headings

Headings in *rST* act the same way as HTML headings.

Here is an example of how the level 1 heading above was coded:

```
##########################
ReStructured Text Examples
##########################
```

Suggested list of Python heading levels:

Symbol	Name	Overline?	Level	Description
#	pound sign	yes	1	for parts
*	asterisk	yes	2	for chapters
=	equal	no	3	for sections
-	dash	no	4	for subsections
^	circumflex	no	5	for subsubsections
“	quote	no	6	for titled paragraphs

All of the headers are aligned with the left margin when using the Read-The-Docs theme.

5.2 Paragraphs

Each paragraph must be preceded by a blank line. Each line of a paragraph must immediately follow the preceding line. A blank line indicates the start of a new paragraph.

This is a paragraph before processing:

```
Each line of a paragraph
should be less than 80 characters.  This is more of
a Python convention.  PyCharm can be set to automatically wrap as you
type.  Sphinx will automatically rewrap the lines to fit the current
margin so don't be concerned if you it looks
"ugly" when editing causes
lines to split.  Other markdown editors might not let you set the line
length to be that short.
```

and after:

Each line of a paragraph should be less than 80 characters. This is more of a Python convention. PyCharm can be set to automatically wrap as you type. Sphinx will automatically rewrap the lines to fit the current margin so don't be concerned if you it looks "ugly" when editing causes lines to split. Other markdown editors might not let you set the line length to be that short.

If the short lines need to be preserved (e.g. a poem[1]) code it with a vertical bar followed by a spce like this:

```
| "You are old, Father William," the young man said,
|     "And your hair has become very white;
| And yet you incessantly stand on your head--
|     Do you think, at your age, it is right?"
|
| "In my youth," Father William replied to his son,
|     "I feared it might injure the brain;
| But now that I'm perfectly sure I have none,
|     Why, I do it again and again."
```

and is displayed like this:

"You are old, Father William," the young man said,
    "And your hair has become very white;
And yet you incessantly stand on your head—
    Do you think, at your age, it is right?"

"In my youth," Father William replied to his son,
    "I feared it might injure the brain;
But now that I'm perfectly sure I have none,
    Why, I do it again and again."

[1] Poem by Lewis Carroll in Alice in Wonderland.

5.3 Inline markup

Blah blah `*italics*` = *italics*, `**bold**` = **bold**. Text inside double backticks is displayed literally. See *Literal Text* below.

5.4 Images

Images can be added to the documentation with this syntax:

```
.. image:: DevDocs/DeploymentOverview.png
```

which will be displayed like this:

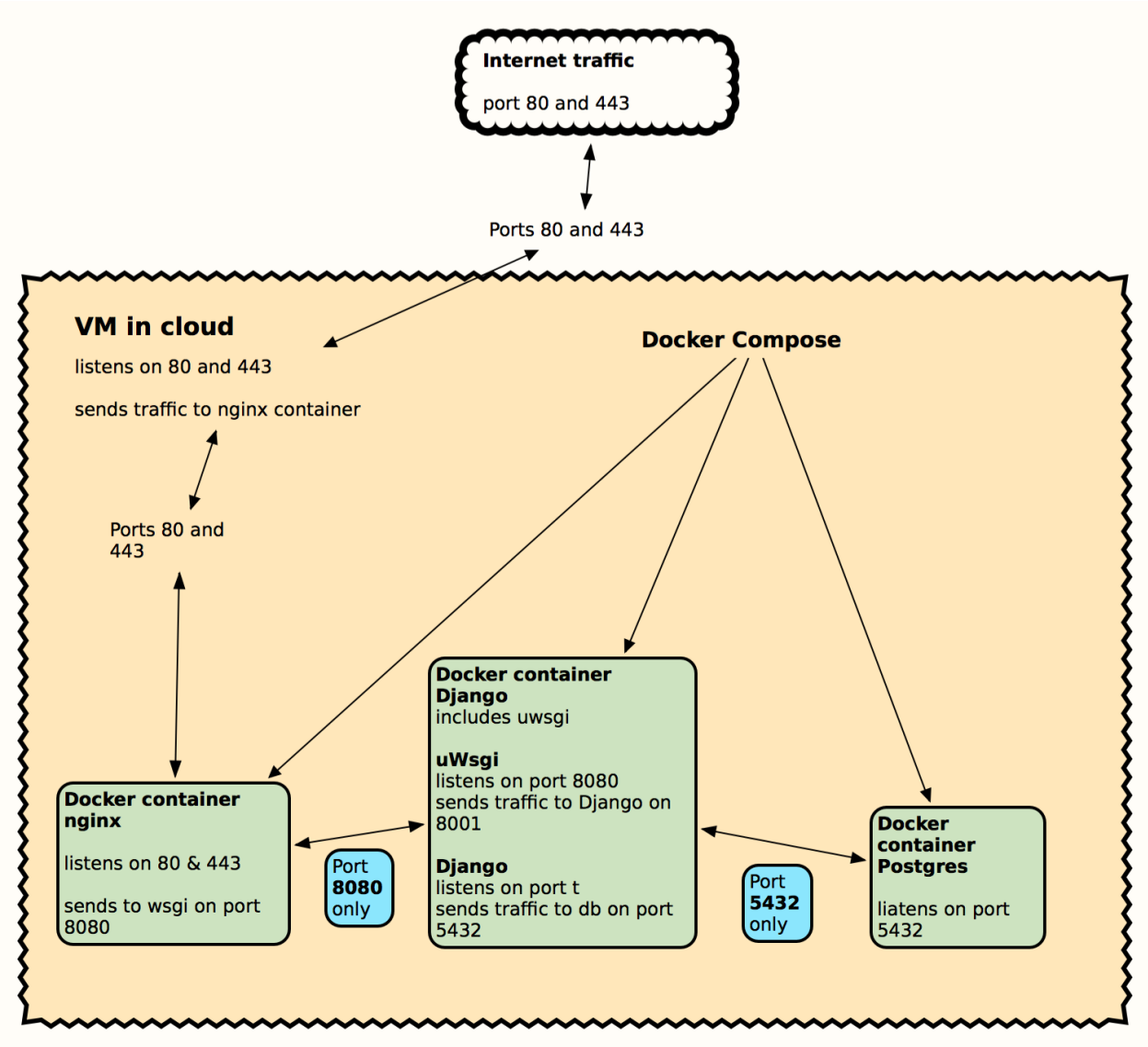


Image notes:

- Image file names that include spaces should be avoided.

- Images are always separate files and can be in the same directory or in a subdirectory of this *rST* file.
- The images will be resized to fit the page width.
- When creating an image, save it as a ".png" file.
  - If available, create a copy of the image as a ".pdf" file. Then use an astrisk as a suffix for the file name, e. g. <imageName>.* in the image directive above. This allows the HTML builder choose the .png file while the PDF builder will choose the .pdf file.
- For now put the image files in the same directory as this text file. Later, they will be collected into a subdirectory and these image directives will be adjusted accordingly.

5.5 Lists

Lists can be numbered, bulleted, or mixed.

5.5.1 Numbered List Example

This is a numbered list:

```
4. Numbered items (need not start at 1.)
#. Numbered items (indent 4 spaces)
#. Period can be replaced by a dash, right paren, etc., but is required.
#. Must have a blank line before and after the list

    *   A numbered list can have numbered or bulleted subitems.  This
        one is bulleted.
    *   Subitems must be preceded and followed by a blank line like this.

#.  Long lines can wrap.  The subsequent lines must be aligned with the
    start of text of the bulleted or numbered item.

    #.  This shows a numbered sublist.
```

which looks like:

4. Numbered items (need not start at 1.)
5. Numbered items (indent 4 spaces)
6. Period can be replaced by a dash, right paren, etc., but is required.
7. Must have a blank line before and after the list
   - A numbered list can have numbered or bulleted subitems. This one is bulleted.
   - Subitems must be preceded and followed by a blank line like this.
8. Long lines can wrap. The subsequent lines must be aligned with the start of text of the bulleted or numbered item.
   1. This shows a numbered sublist.

5.5.2 Bulleted List Example

This is a bulleted list:

```
- Bulleted items (can start with "-", "*" or "+".

- Bulleted items

    - sublist items (indent 4 spaces)

- Must have a blank line before and after the list
```

which looks like this:

- Bulleted items (can start with "-", "*" or "+".
- Bulleted items
  - sublist items (indent 4 spaces)
- Must have a blank line before and after the list

Note: For both numbered and bulleted lists, a blank line can be added between each item or subitem at the same level if desired. Any subitems must have a blank line both before and after the group of subitems. Continuation lines however, must immediately follow the item being continued to be properly rewrapped by Sphinx.

5.6 Literal text

5.6.1 Literal Text Block

A literal text block coded like this:

::

    literal text ...

    more text indented text

looks like this:

```
literal text
...

more text
    indented text
```

Note that literal text starts with two colons and a blank line both before and after the colons. All text that is part of theliteral block must be indented.

An alternative is to put the two colons at the end of the preceding paragraph. If the two colons are adjacent to the last character of the paragraph, they will be rendered as a single colon. If there is a space before the two colons, they will not appear in the final output.

Literal text can have lines that are indented further and also blank lines interspersed. However, text indented the same or less than the "::" marker (or the text of the paragraph) ends the literal text.

5.6.2 Inline Literal Text

Any text surrounded by double backticks is displayed as is, and in a different font.

Embedded literal text coded like this:

```
A sentence with ``:> GoOfY text :)`` in it.
```

looks like this:

> A sentence with `:> GoOfY text :)` in it.

5.7 Tables

5.7.1 Simple Table

A simple Table coded like this:

```
============   ========================
Cell Title     Another Cell  Title
============   ========================
contents       more contents
item 1         item 2
green          purple
============   ========================
```

is rendered list this:

Cell Title	Another Cell Title
contents	more contents
item 1	item 2
green	purple

If a cell needs to span rows or columns use the grid table format instead.

5.7.2 Grid Table

A grid table coded like this:

```
..  table:: Sample Grid Table

    +----------------+--------------+--------------+
    | Header Col 1   | Header 2     |   Centered   |
    | Extended       |              |   Header     |
    +================+==============+==============+
    | Body 1         |   Body 2     |       Body 3 |
    +----------------+--------------+--------------+
    | Left Just      |   Centered   |   Right Just |
    +----------------+--------------+--------------+
    | This entry spans these cols   | This entry   |
    +----------------+--------------+ spans rows   +
    | Blah           | Blah         |              |
    +----------------+--------------+--------------+
```

is rendered like this:

Table 1: Sample Grid Table

Header Col 1 Extended	Header 2	Centered Header
Body 1	Body 2	Body 3
This entry spans these cols		This entry spans rows
Blah	Blah	

Notes:

- Text in the cell can be wrapped. The table will be expanded if possible and the text in each cell will be rewrapped as needed.
- The "`.. table:: Sample Grid Table`" line is completely optional. If given, it will provide a title to the table (as shown). If given, the table must be indented under it.

5.8 Links

Links are created by a single backtick (accent grave) on each side of the link. A single underscore immediately before the trailing backtick identifies it as the definition, while a reference is created by putting the underscore immediately before the first backtick.

5.8.1 External URL Links

External link is coded like this:

```
`Apple main web site <http://www.apple.com>`_
```

and is rendered like this in a paragraph.

Create a ink to an external URL, such as the Apple main web site, anywhere you desire.

The actual URL is wrapped with the "<" and ">" characters. If a title for the URL is desired, it must be specified between the initial backtick and the less than symbol. There must be a space between the title and the less than symbol.

5.8.2 Internal Links

Reference to arbitrary text elswhere in the same document is accomplished by identifying a source location (which rST documentation calls a target) then making a reference to it elsewhere in the same or other document.

Source (target) of a reference:

```
_`Link to elsewhere`            (multiple words)

_`document`                     (single word)
```

Reference to a source:

```
`Link to elsewhere`_

document_                       (note lack of backticks)
```

In context a link to elsewhere in the same document can be referenced with *link to elsewhere* in the *document*. See *Cross-References* below.

A link to headers in the same document such as *ReStructured Text Examples* and *Links* can be easily referenced since Sphinx automatically creates links out of all headers. Headers can be in some other *rST* document, such as *Developer Documentation* can also be referenced if the header to be referenced is given additional markup.

Links to glossary terms can be imbeded in a paragraph uaing the syntax:

```
blah blah :term:`Glossary-type definition` blah blah
```

Links to glossary terms, such as *Glossary-type definition*, can also be easily made.

Note: Every reference to a target must be spelled exactly the same – including capitalization!

5.9 Transitions or Horizontal Rules

A transition (or horizontal rule) is coded as:

```
----
```

and shows up like this:

A transition should not begin or end a section or document, nor should two transitions be immediately adjacent.

5.10 Glossary Definitions

A glossary definition like this:

```
..  glossary::

    Glossary-type definition
        The definition for the term must be indented and immediately below
        the term.

        Blank lines may appear in the definition body, but must not
        come between the term and the first line of definition.

        The term defined can be referenced elsewhere as shown above.

        Each glossary term header must be preceded by a blank line.

    reStructuredText
        This is plain text with a minimal amount of annotation.  This allows
        program (such as Sphinx) to process the text and use the annotations
        as clues to how detect headings, tables, cross-references, etc. so
        that web pages, PDFs, epubs and other formal documentation can be
        created from the original text.
```

will be formated like this:

Glossary-type definition The definition for the term must be indented and immediately below the term.

Blank lines may appear in the definition body, but must not come between the term and the first line of definition.

The term defined can be referenced elsewhere as shown above.

Each glossary term header must be preceded by a blank line.

reStructuredText This is plain text with a minimal amount of annotation. This allows program (such as Sphinx) to process the text and use the annotations as clues to how detect headings, tables, cross-references, etc. so that web pages, PDFs, epubs and other formal documentation can be created from the original text.

5.11 Notes

Short paragraphs with special offsets or formatting for emphasis can be created by using this notation:

```
.. note::
    This is a noteworthy comment.
```

which displays like:

Note: This is a noteworthy comment.

Comments about this kind of markup.

- The text of the note musb be indented at least three spaces and can be more (as shown).
- Instead of the word "note", other words such as "caution", "hint", "important", "tip", "attention", "danger", "error" and "warning" can be used.
- Sphinx with the "Read the Docs" theme renders these with different colors in html.
  - Notes are shown with a blue background.
  - Hints, tips and importants are shown in green.
  - Attentions, cautions and warnings are shown with a brown background.
  - Error and danger are shown with a pink background.
- The pdf output from Sphinx with the "Read the Docs" theme sets these off differently.

5.12 Substitutions

Today's date will be added to the by using `|today|` like this Mar 01, 2021.

5.13 Cross-References

This section shows examples of how to refer to arbitrary text that is some distance away from this part of the document.

This is a reference to the *link to elsewhere* in this *document* written using *rST*.

5.14 Summary

This document is a sampling of the rich formatting that can enhance plain text. The reStructuredText and Sphinx documentation provide many more ways to subtly markup the text than what is shown here.

The original plain text – as much as possible – is still very readable, while a processing program such as Sphinx can create a web site or PDF with beautiful output, lots of cross-references, a table of contents and an index.

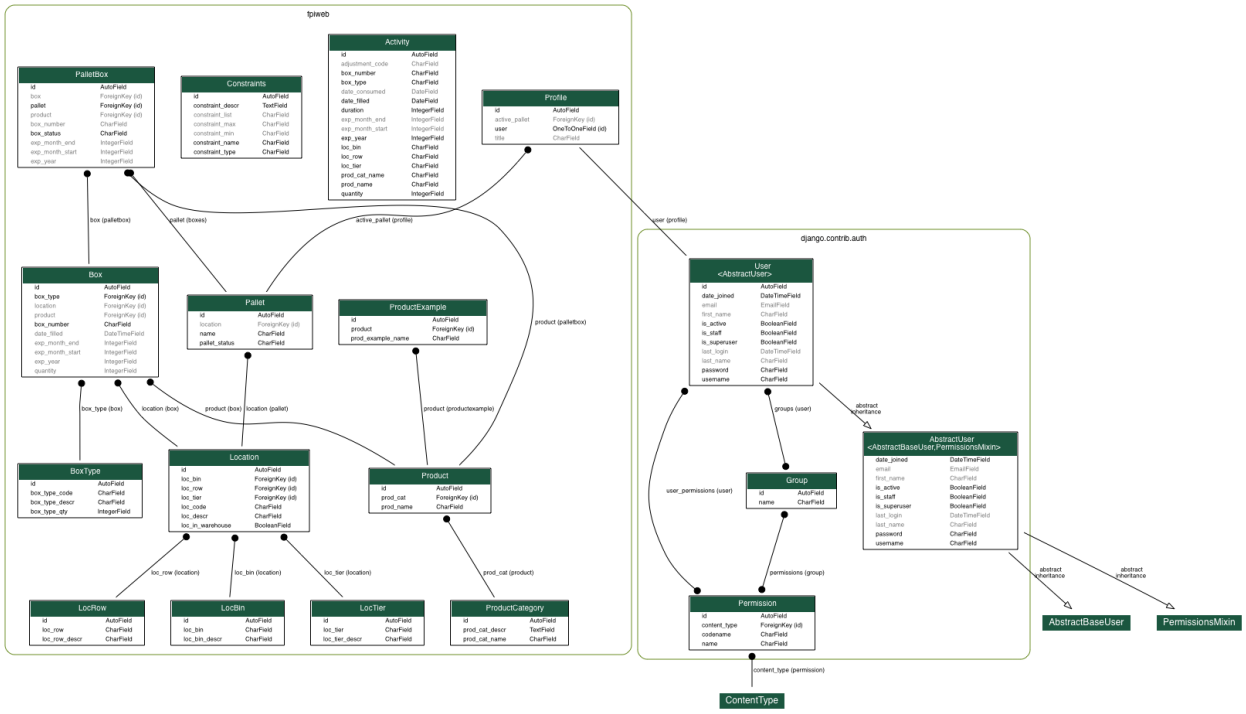
CHAPTER

SIX

DEVELOPER RESOURCES

6.1 Table Relationships

Image expressed as a png file.



6.2 Documentation Tools

Some good possible sources to learn things:

#. Writing Sphinx documentation

   Explains what Sphinx is and why we should use it.

1. Django main web site

   (Currently we are using version 2.2.1.)

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

This documentation was last generated on Mar 01, 2021 for version 0.9.1 (release 0.1.1a1).

PYTHON MODULE INDEX

f

S

INDEX

C

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y