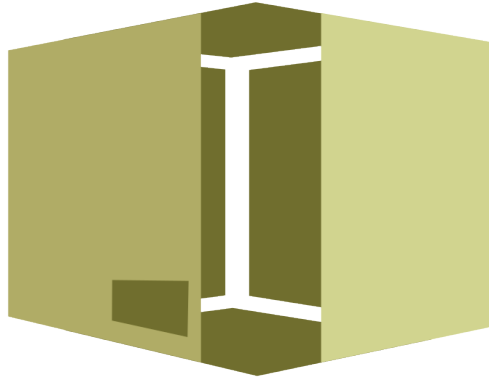


POLYTECHNIC UNIVERSITY OF THE PHILIPPINES  
BACHELOR OF SCIENCE IN COMPUTER SCIENCE



DISCRETE STRUCTURES II  
**Invento Management System  
Documentation**

**BSCS 2-1N**  
**Group 3**

**Project Manager:** Annalyn Belen  
**Designer:** Monika Jea Ng  
**Developer:** Steve Pabular  
**Systems Analyst:** John Nicolas Oandasan  
**Business Analyst:** Hazel Conception  
**Technical Writer:** Percian Cayaban

**Instructor:** Prof. Angie Payne

# Table of Contents

Title page	i
Contents	ii
<b>I Invento Management System</b>	<b>1</b>
Overview	1
Features	1
Setup	1
<b>II User Guide</b>	<b>2</b>
<b>III Process and Tools</b>	<b>5</b>
Flowchart	5
Database	5
User Interface	6
Development Tools	6
<b>IV Code Documentation</b>	<b>7</b>
Project Structure	7
Main file	7
Main.py . . . . .	8
<b>utils package</b>	<b>9</b>
accounts.py . . . . .	9
assets.py . . . . .	10
dependencies.py . . . . .	11
icons.py . . . . .	11
itemdata.py . . . . .	11
randompic.py . . . . .	13
settings.py . . . . .	14
<b>customwidget package</b>	<b>16</b>
CtmTreeView.py . . . . .	16
IntSpinBox.py . . . . .	17
LoginBg.py . . . . .	17
SalesGraph.py . . . . .	17

<b>Pages</b>	<b>18</b>
LoginPage.py . . . . .	18
RegisterPage.py . . . . .	18
InventoryPage.py . . . . .	19
<b>Tabs</b>	<b>20</b>
AccountTab.py . . . . .	20
DashboardTab.py . . . . .	20
ProductTab.py . . . . .	21
SettingsTab.py . . . . .	21
AboutTab.py . . . . .	21

## Part I

# Invento Management System

## Overview

---

**Invento** is an inventory management system that allows users to manage their inventory of products, track sales, and view sales data in real-time. The system supports multiple user roles, including **Admin** and **User** accounts, and provides other variety of features.

---

## Features

1. **Login and Registration** - To access the system, users must first register an account or log in with an existing account. This feature allows tracking whose changes were implemented in the inventory. This also saves the current session for future access.
2. **Admin and User Accounts** - Users can control the inventory and sales data. Administrators had access to additional features, including the ability to reset inventory, delete accounts, and manage user accounts.
3. **Sales Graph** - Displays a line graph of sales data for the past 7 days. The graph updates in real-time as new sales data is entered into the system.
4. **Product Management** - Users and admins can add, edit, and remove products from the inventory. Changes can be seen in the table displayed.
5. **Account Settings** - Users and admin can change their passwords and display pictures. They could also personalize the themes of the program in settings.

## Setup

This program requires the 3.10+ version of Python installed and the following packages:

- `customtkinter`
- `Pillow`
- `matplotlib`

Which can be installed with the following command:

```
pip install --upgrade customtkinter Pillow matplotlib
```

There also is a detailed setup guide available at <https://github.com/steguiosaur/invento>.

## Part II

# User Guide

The program can be executed by using the command `python Main.py` in a terminal. If there aren't any dependency conflicts and logged-in session, it will show the Login page (Figure 1) wherein it takes an input for the current registered accounts.

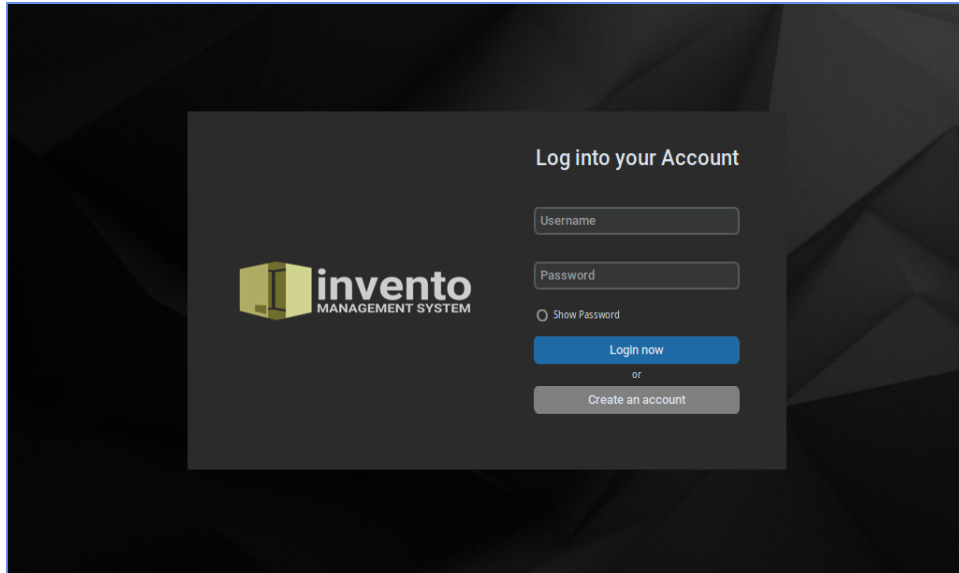


Figure 1: Login page

On this page (Figure 2), you could register a new account by entering the required information.

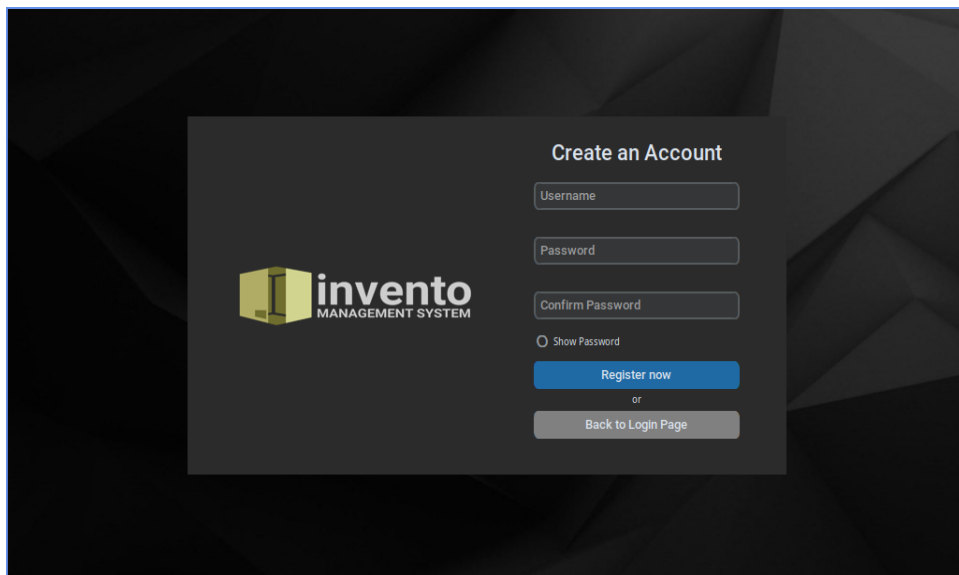


Figure 2: Register page

To access the inventory management system, log in with a valid username and password. If you do not have an account, Click the "Create an account" button on the

login page. Once you have created an account, you can log in and begin using the system.

After logging in, the Dashboard Tab (Figure 3) is shown. It displays the overall changes done in the inventory and current number of users, products, categories, and total sales.

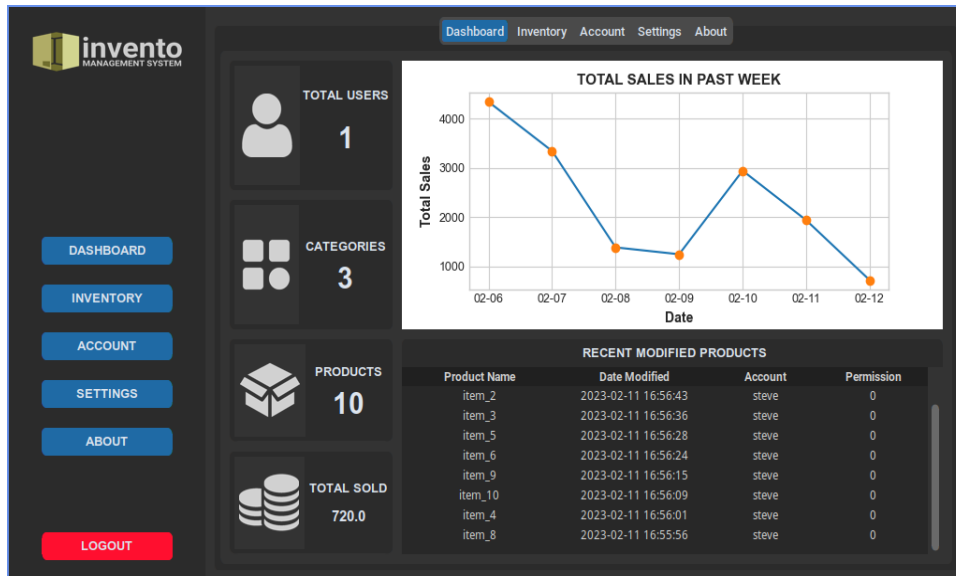


Figure 3: Dashboard Tab

The Inventory Tab (Figure 4), is where you manage the products, categories, and sales, There also is the search functionality that enables quickly look up for an item you desire to look into. If you wanted to sort the item based on stock, name, data modified, etc., you can click the header of the table to trigger it into ascending and descending order.

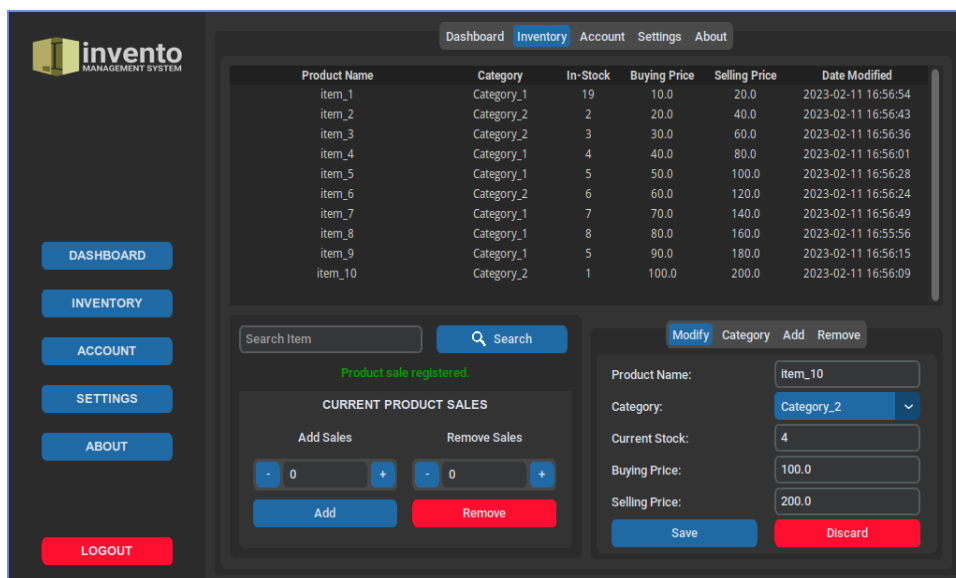


Figure 4: Inventory Tab

In this page (Figure 5), is where you manage your account and view other accounts. There are two levels of permission given for an account, the **User Account** and the **Administrator Account**. User accounts can access the normal features given in the program, like the inventory management feature. The latter, administrator account, can access the whole features including account deletion, resetting the inventory, and changing permissions for user accounts.

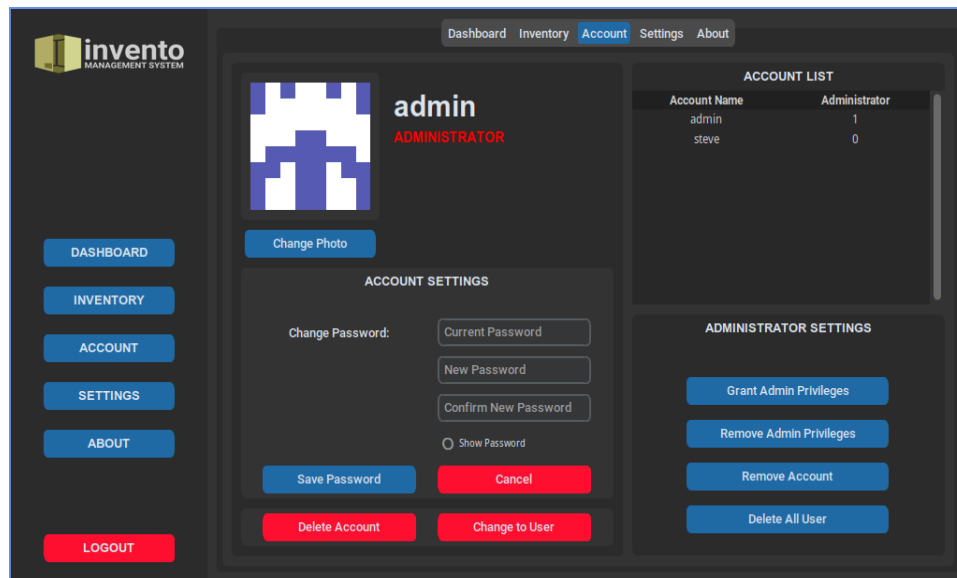


Figure 5: Accounts Tab

The Settings Tab (Figure 6) handles all theme changes and widget scaling. There currently are two appearances, the **Light** and **Dark** appearance. The theme can be changed into blue, dark-blue, and green. For the rest, you can find out by trying the program.

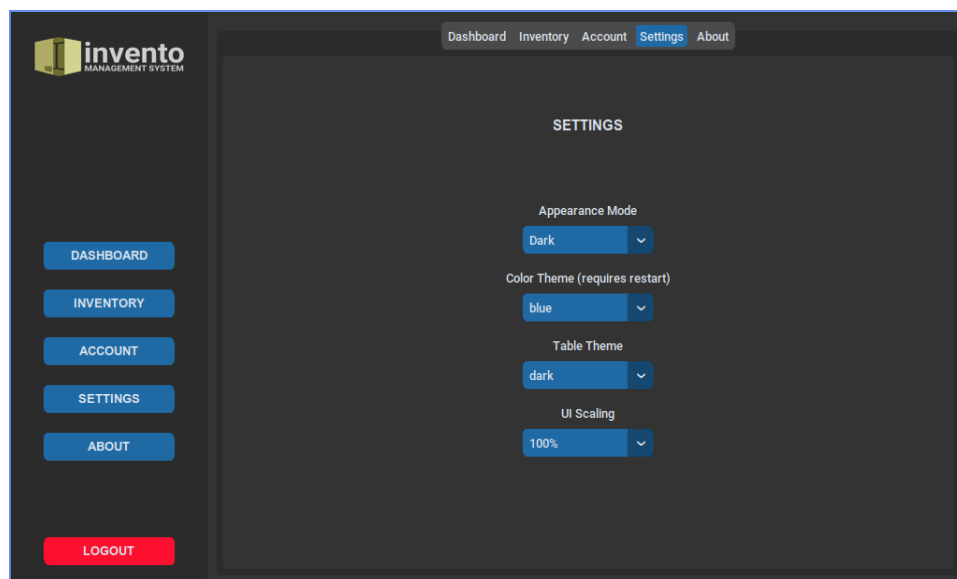


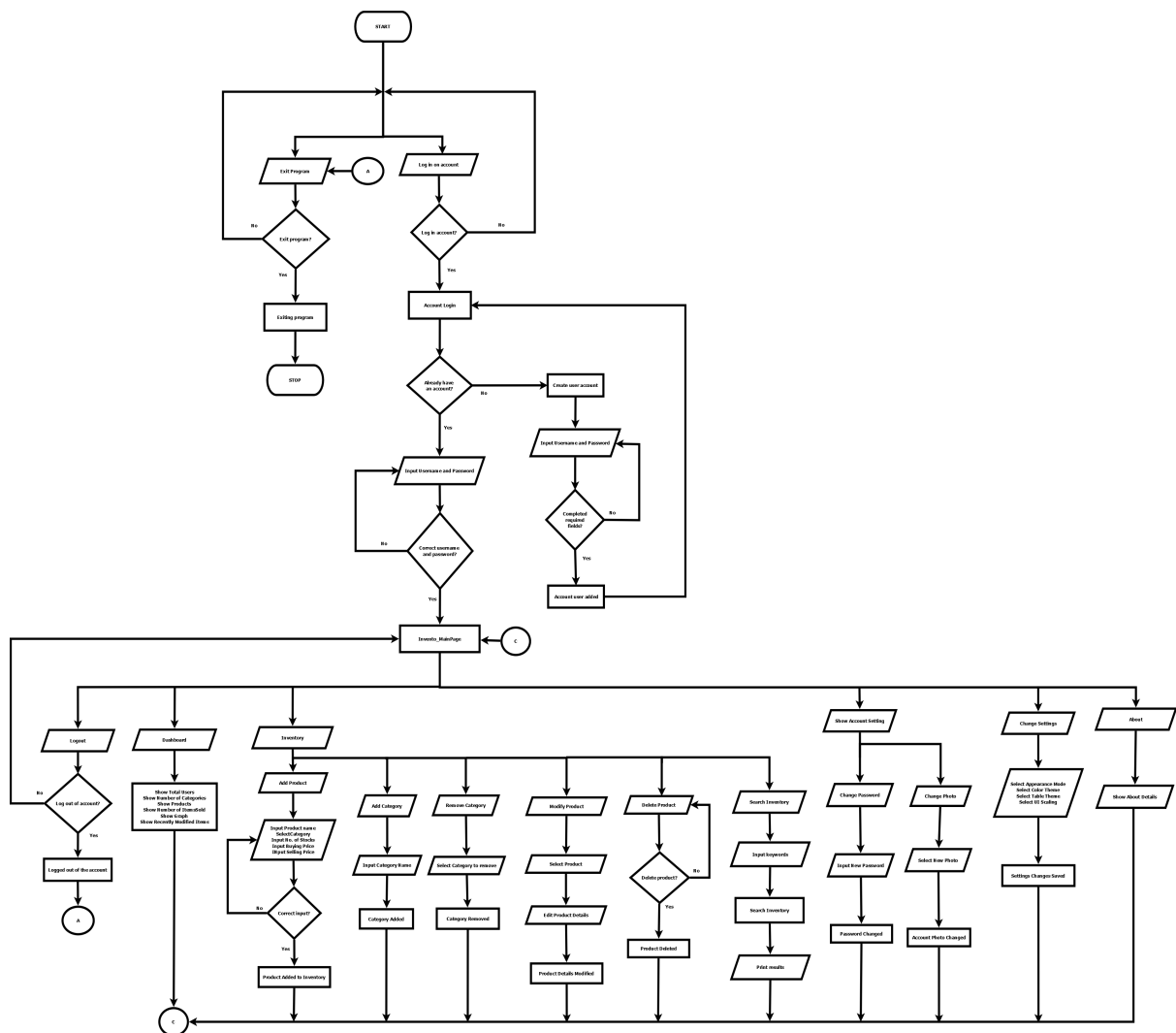
Figure 6: Settings Tab

# Part III

## Process and Tools

This software is built using modular, object-oriented structure, with focus on readability, maintainability, and extensibility. It follows a traditional Model-View-Controller (MVC) architecture, with separate components managing the user interface, logic, and database interactions.

### Flowchart



### Database

Invento uses a relational database to store product data, sales data, and account information. The database is managed using the Python `sqlite` module, which provides a simple and efficient interface for executing SQL queries and managing database connection.



## User Interface

The user interface of this software is implemented using graphical user interface (GUI) framework, such as `TKinter` and `Customtkinter`. We aimed for the interface design that is minimal, intuitive and user-friendly, with a clean modern layout.

## Development Tools

The following tools are used to develop the program.

### </> Programming Language

Python 3.10

### ⚙️ Frameworks and Libraries

`Customtkinter` - GUI framework/package

`TKinter` - GUI framework

`Pillow` - image processing

`Matplotlib` - data visualization

### 🗄️ Database and Configurations

`SQLite3` - creates `*.db` file for database

`Configparser` - creates `*.ini` file for configurations

### ✍️ Text editor or Integrated Development Environment (IDE)

`Neovim` - terminal based text editor

`Pycharm` - IDE

### 🔄 Version Control System (VCS)

`Git` - local VCS

`Github` - <https://github.com/steguiosaur/invento>.

### 🎨 Creative Tools

`GIMP` - photo editor

`Inkscape` - vector graphics editor; used in logo creation

`Canva` - used in presentations

`Figma` - used for structuring GUI in early versions

`Dia` - flowchart

### 📄 Mark-up Language

`Markdown` - README files

`LATEX` - used for creating this documentation

## Part IV

# Code Documentation

There were several naming conventions used in the code.

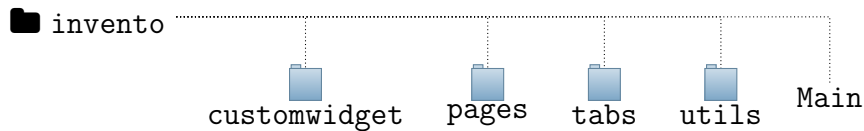
**Pascal case** is used for `ClassNames`

**Camel case** is used for `objectNames`

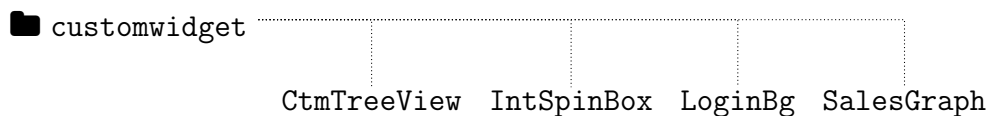
**Snake case** is used for `function_names` and `method_names`

## Project Structure

Invento packages and main file.



customwidget modules



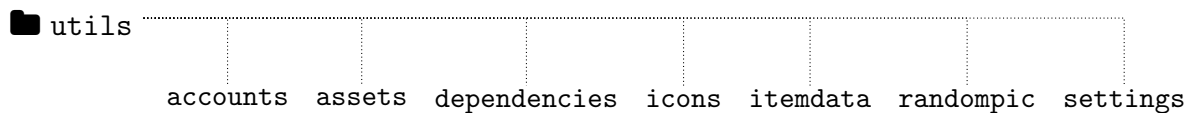
pages modules



tabs modules

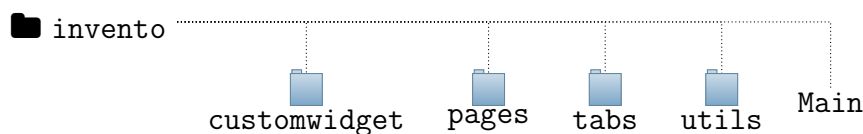


utils modules



## Main file

`Main.py` or the main file, is responsible for executing the app. This can be triggered by using the command `python Main.py`. It is located at the root of the project with other packages.



## </> Main.py

This first part of the code imports several modules from `utils` package. It calls `dependency_installer()` function from `dependencies` module to automate the installation of packages that are not installed.

```
from utils import accounts, itemdata, settings, dependencies, Assets
dependencies.dependency_installer() # install dependencies
```

After the installation of required packages, it proceeds to import several other packages. `Customtkinter` and `TKinter` are responsible for creating the window where the frames will be placed. The `pages` package imports all of its module [`LoginPage`, `RegisterPage`, `InventoryPage`] to be added onto the frame dictionary.

```
from customtkinter import CtkFrame, set_appearance_mode, set_default_color_theme,
    ↪ set_widget_scaling
from tkinter import PhotoImage, Tk
from pages import *

class Main(Tk):
    def __init__(self):
        super().__init__()

        # creates container for frames
        container = CtkFrame(self)
        container.pack(side="top", fill="both", expand=True)
        container.grid_rowconfigure(0, weight=1)
        container.grid_columnconfigure(0, weight=1)

        self.frames = {} # create page dictionary
        for f in [InventoryPage, LoginPage, RegisterPage]:
            page = f.__name__
            frame = f(container, self)
            frame.grid(row=0, column=0, sticky="NSEW")
            self.frames[page] = frame
```

In this part, the `self.get_session()` will display `InventoryPage` if there is an account that is currently logged in. If not, it will display `LoginPage` instead.

```
# initialize starting frame
self.get_session()

# display selected page on top
def show_frame(self, page, id=None):
    self.id = id
    self.frames[page].tkraise()

# current logged in account
def get_session(self):
    if accounts.get_session() is not None:
        self.show_frame("InventoryPage")
    else:
        self.show_frame("LoginPage")
```

The comment already explains what it does in this part.

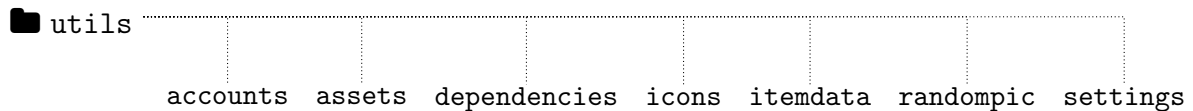
```
# create database and admin account if not exists
accounts.create_table()
itemdata.create_inventory_table()

# initialize settings and themes
settings.initialize_config()
set_appearance_mode(settings.appearance_read())
set_default_color_theme(settings.theme_read())
set_widget_scaling(settings.int_scale_read())

# start application
app = Main()
app.title("Invento")
app.resizable(True, True)
width = 1024
height = 576
x = (app.wininfo_screenwidth()/2) - width/2
y = (app.wininfo_screenheight()/2) - height/2
app.geometry('%dx%d+%d+%d' % (width, height, x, y))
app.minimize(1024, 576)
app.iconphoto(True, PhotoImage(file=Assets.asset_path('logo.png')))
app.mainloop()
```

## utils package

This package contains the overall functionality of the program. It includes all modules that handle the database, file paths, generation of image, configurations, and other miscellaneous functions.



### </> accounts.py

The `accounts` module handles all account related functionality. It connects itself to the database file named `invento.db` and the database table named `accounts` and `sessions`.

accounts			sessions
Username	Password	Admin	Username
admin	hashedpass0e4afc3	1	admin

The following are its functions:

- ❖ `change_pass(username, passwd, new_passwd, confirm_passwd)`
  - Verifies password changes. Used by `AccountTab` on account settings.
- ❖ `count_non_admin_accounts()`
  - Used by `DashboardTab` to display current users.

- ❖ `create_table()`
  - Creates table for accounts, login session, and an admin account.
- ❖ `delete_all_users()`
  - Needs admin privileges to delete all users. Accessed by `AccountTab`.
- ❖ `delete_user(username)`
  - Accessed by `AccountTab` to delete an account.
- ❖ `get_all_accounts()`
  - Displays current accounts in the table.
- ❖ `get_permission_level(username)`
  - Returns 1 if session is an admin account, else 0. Used to verify account permissions.
- ❖ `get_session()`
  - Returns the current logged in account.
- ❖ `grant_admin_privilege(username)`
  - Gives a user account admin privileges. Requires an admin account.
- ❖ `login(username, passwd)`
  - Used by `LoginPage` to verify username and password.
- ❖ `logout()`
  - Removes account in session. Changes frame to `LoginPage`.
- ❖ `register(username, passwd, confirm_passwd, admin=False)`
  - Creates a new account in the database.
- ❖ `remove_admin_privilege(username)`
  - Removes admin permission. Accessed by `AccountTab`.

## </> assets.py

The `assets` module locates the location of the `./assets/` folder in the project. Due to different file pathing between platforms, `Linux` and `Windows`, using this module makes it compatible on both operating systems.

```
from pathlib import Path

class Assets:
    OUTPUT_PATH = Path(__file__).parent
    ASSETS_PATH = OUTPUT_PATH / Path("../assets")

    @staticmethod
    def asset_path(path: str) -> Path:
        return Assets.ASSETS_PATH / Path(path)
```

## </> dependencies.py

This module is responsible for automatically installing the required packages listed on `requirements.txt`. It creates a loop, verifying if the package is installed or not. This script only runs on initial execution of the program. It will be triggered again if the config file `config.ini` is deleted.

```

from os.path import isfile
import subprocess
import sys

def install(package):
    subprocess.call([sys.executable, "-m", "pip", "install", package])

def dependency_installer():
    # executes installer on first startup
    if not isfile('./config.ini'):
        with open("requirements.txt") as f:
            dependencies = f.read().splitlines()

    for package in dependencies:
        try:
            __import__(package)
        except ImportError:
            install(package)

```

## </> icons.py



The `icons` module manage the icons being used in `Dashboard` and `ProductTab`. It changes according to appearance that was set in the configuration file.

## </> itemdata.py

This module handles all inventory related functionality that accesses the database. It connects on the database file named `invento.db` and controls three (3) tables named as `products`, `categories`, and `sales`.

**products**

items	category	in.stock	buying_price	selling_price
Golden Onion	Spices	30	200.00	250.00
		date_modified	modified_by	permission_level
		23-02-19 23:43	admin	1

categories
category_name
Drinks

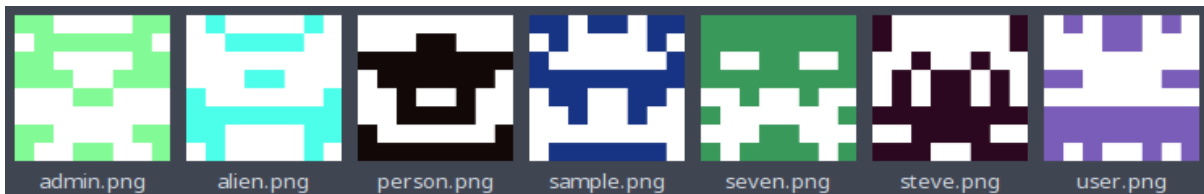
sales	
total_sales	date_sale
11980.00	23-02-19

The following are its functions:

- ❖ `add_category(category_name)`  
- Used in category panel located in `ProductTab`.
- ❖ `add_product(item, category, in_stock, buying_price, selling_price)`  
- Adds the product to inventory table. Used in `ProductTab`'s add panel.
- ❖ `add_sales(earned)`  
- Used by the frame "Current Product Sales" in `ProductTab`.
- ❖ `count_category()`  
- Returns the number of category from the database to be displayed in `DashboardTab`.
- ❖ `count_products()`  
- Returns the number of products from the database to be displayed in `DashboardTab`.
- ❖ `create_inventory_table()`  
- Responsible for creating the tables named `accounts`, `categories`, and `sales` in the database.
- ❖ `delete_all_products()`  
- Removes all listed products. Requires admin permission.
- ❖ `delete_product(product)`  
- Deletes a single selected product. Located at the remove panel in `ProductTab`.
- ❖ `edit_product(product, category, in_stock, buying_price, selling_price, product_focus)`  
- Updates the product information based on input from modify panel.
- ❖ `get_all_category()`  
- Returns all listed categories from the database to be accessed by the dropdown option menu from `ProductTab`.
- ❖ `get_current_date_sales()`  
- Returns the sales of the current date. Unused functionality.
- ❖ `get_current_in_stock(item_name)`  
- Reads current number of stock an item have. Used to limit the maximum value of the current stock in adding a sale. Used on "Current Product Sales" panel in `ProductTab`.

- ❖ `get_sales_data()`
  - Data is used by `SalesGraph` to be plotted in the line graph at `DashboardTab`.
- ❖ `get_selling_price(item_name)`
  - Used by `add_sales` and `remove_sales` to determine the price of the item.  
 $\text{add/remove\_sales} = \text{number\_of\_product} * \text{product\_price}$
- ❖ `get_today_sales()`
  - Returns the total sales from the database to be displayed in `DashboardTab`.
- ❖ `reduce_sales(remove_earned)`
  - Used by the frame "Current Product Sales" in `ProductTab`.
- ❖ `remove_category(category_name)`
  - Removes the selected category in the database's `categories` table.
- ❖ `search_product(item_name)`
  - Used by search entry in `ProductTab` that filters the entered product to be displayed in the inventory table.
- ❖ `sort_table(column, ascending)`
  - Sorts all columns in ascending and descending order. Triggered in the inventory table header.
- ❖ `update_stock(item_name, new_stock)`
  - Updates the stock after adding or removing a sale.
- ❖ `view_inventory()`
  - Displays the inventory table in the `ProductTab`.
- ❖ `view_modified()`
  - Displays the recent modified products table in the `DashboardTab`.

### </> `randompic.py`



This module creates a somewhat high resolution 8x8 pixelated image that is symmetrical in the center x-axis. It acts as a display photo that is different for every account.

In this part of the code, it creates the size, size of pixel boxes, and colors.

```
size = (128, 128)
box_size = size[0] // 8
white = (255, 255, 255)
random_color = (random.randint(0, 255), random.randint(0, 255), random.randint(0,
    ↪ 255))
```



The variable `random_color` can generate a total of 16,777,216 different RGB color values. This is randomized by the built-in `random` module of Python.

$$256 * 256 * 256 = 16777216$$

In this for loop, it paints the selected box per index with the result from the if-else statement.

```
for i in range(4):    # 4 boxes on x-axis
    for j in range(8): # 8 on y-axis
        if random.choice([True, False]):
            color = random_color
        else:
            color = white
        x1 = i * box_size
        y1 = j * box_size
        x2 = (i + 1) * box_size
        y2 = (j + 1) * box_size
        draw.rectangle([x1, y1, x2, y2], fill=color)
        draw.rectangle([(size[0] - x2), y1, (size[0] - x1), y2], fill=color)
```

We could calculate the total number of patterns this module could generate using this simple permutation formula:

$$n^r = 2^{8*4} = 2^{32} = 4294967296$$

Where  $n$  = number of colors, which is `white` and the `random_color`. And  $r$  = number of pixels or boxes to be generated with a color. It can generate 4,294,967,296 different patterns without considering the randomization of color value.

If we try to get the overall randomization with patterns and color value, it reaches an almost incomprehensible total of permutations.

$$(16777216 + 1(\textit{white}))^{32}$$

After all the generation of colors and image, it will be stored under the `./assets/image/` folder.

```
# store account photo
path = Path("assets/image") / (username + ".png")
path.parent.mkdir(parents=True, exist_ok=True)
image.save(path)
```

## </> settings.py

This module is responsible for reading and writing the preset configuration in the file `config.ini`.

### config.ini

```
[settings]
appearance = Dark
theme = blue
tablecolor = dark
scale = 100
```

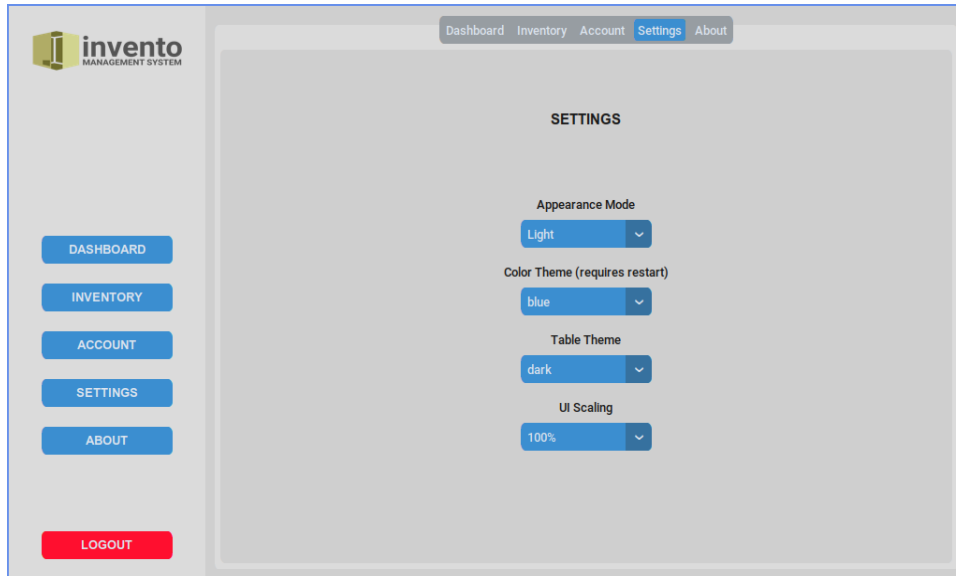


Figure 7: Settings Tab

All the backend functionality that Settings Tab does is shown in this code.

```

from configparser import ConfigParser
from os.path import isfile

config = ConfigParser()
config.read('config.ini')

# create config at first execute
def initialize_config():
    if not isfile('config.ini'):
        config_set()

# default configuration
def config_set():
    config.add_section('settings')
    config.set('settings', 'appearance', 'Dark')
    config.set('settings', 'theme', 'blue')
    config.set('settings', 'tablecolor', 'dark')
    config.set('settings', 'scale', '100')
    config.write(open('config.ini', 'w'))

# appearance [light, dark]
def appearance_save(appearance):
    config.set('settings', 'appearance', appearance)
    config.write(open('config.ini', 'w'))

# color theme [blue, dark-blue, green]
def theme_save(theme):
    config.set('settings', 'theme', theme)
    config.write(open('config.ini', 'w'))

# table theme [light, dark]
def table_theme_save(table):
    config.set('settings', 'tablecolor', table)
    config.write(open('config.ini', 'w'))

```

```
# zoom value [80%, 90%, 100%, 110%, 120%]
def scale_save(scale):
    str_scale = str(int(scale * 100))
    config.set('settings', 'scale', str_scale)
    config.write(open('config.ini', 'w'))
```

This second half of the code is used to initialize the preset configuration when the program starts. You can see it being called on the Main module. It is also used to view the current configuration that was set.

```
# get current configuration
def appearance_read():
    return (str(config.get('settings', 'appearance'))))

def theme_read():
    return (str(config.get('settings', 'theme'))))

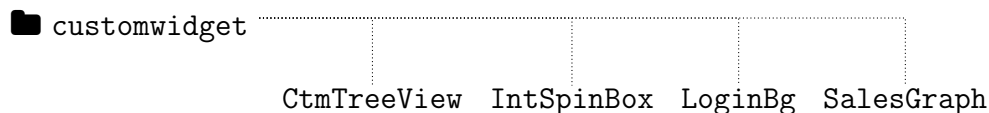
def table_theme_read():
    return (str(config.get('settings', 'tablecolor'))))

def scale_read():
    return (str(config.get('settings', 'scale'))+"%")

def int_scale_read():
    return int(config.get('settings', 'scale')) /100
```

## customwidget package

All modules that can be seen here are mostly customized widgets that displays their own functionality and can be called as objects. It is to be able to place them inside other classes.



### </> CtmTreeView.py

Shows the table widget and manages the table style. Used in DashboardTab, ProductTab, and AccountTab.

Product Name	Category	In-Stock	Buying Price	Selling Price	Date Modified
item_1	category_1	0	10.0	20.0	2023-02-12 18:52:22
item_2	category_2	2	20.0	40.0	2023-02-12 18:52:22
item_3	category_3	3	30.0	60.0	2023-02-12 18:52:22
item_4	category_4	3	40.0	80.0	2023-02-12 18:52:22
item_5	category_5	5	50.0	100.0	2023-02-12 18:52:22
item_6	category_6	6	60.0	120.0	2023-02-12 18:52:22
item_7	category_7	5	70.0	140.0	2023-02-12 18:52:23
item_8	category_8	3	80.0	160.0	2023-02-12 18:52:23
item_9	category_9	80	90.0	180.0	2023-02-15 10:13:38
item_10	category_10	2	100.0	200.0	2023-02-12 18:52:23

Figure 8: Inventory Table

## </> IntSpinBox.py

Used in AccountTab to input product sales.



Figure 9: SpinBox

## </> LoginBg.py

Inherited by LoginPage and RegisterPage to easily manage theme changes.

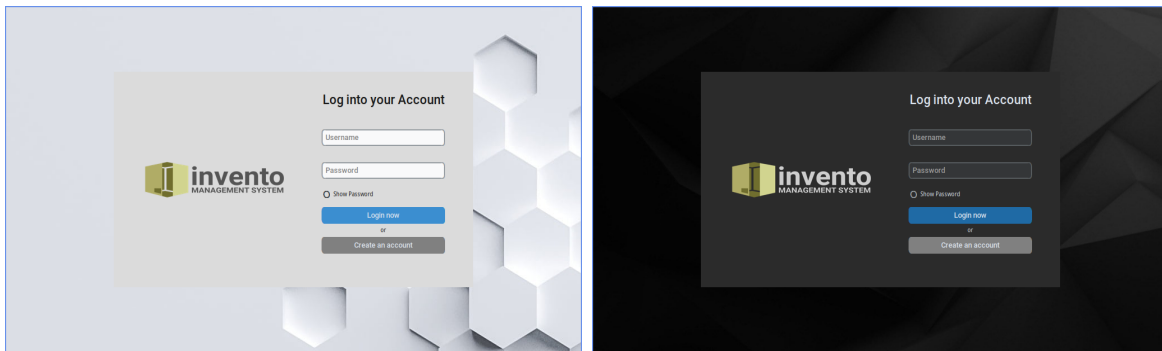


Figure 10: LoginBg Light and Dark Mode

## </> SalesGraph.py

Displays a line graph and plots the sales per day in DashboardTab. It uses Matplotlib to display the data.

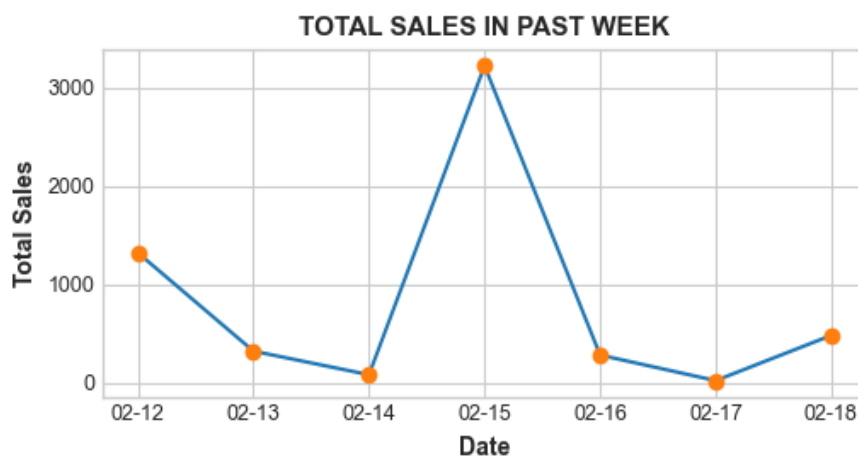


Figure 11: SalesGraph

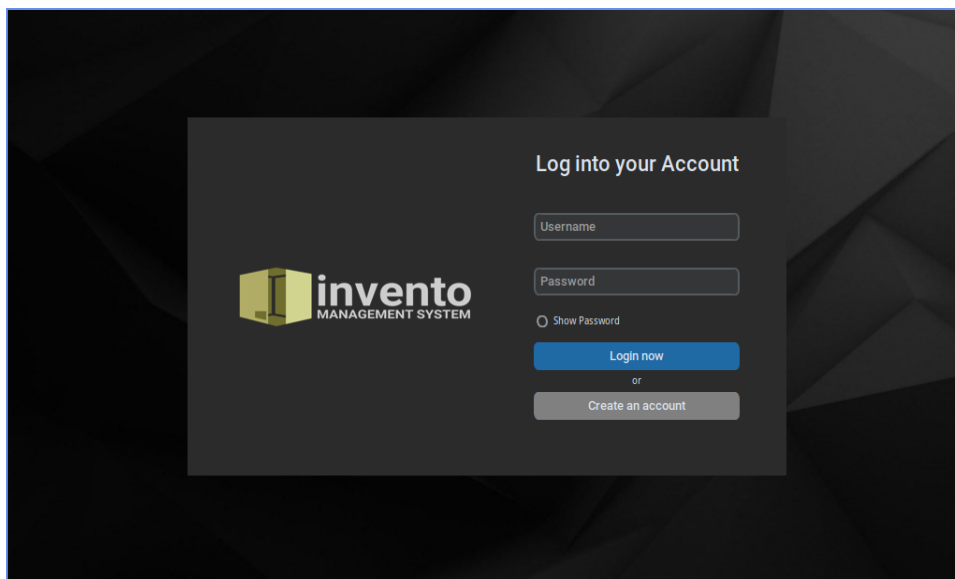
## Pages

The `pages` package handles all the frames for login, register, and the inventory. The `Main.py` displays them accordingly to the user's action.



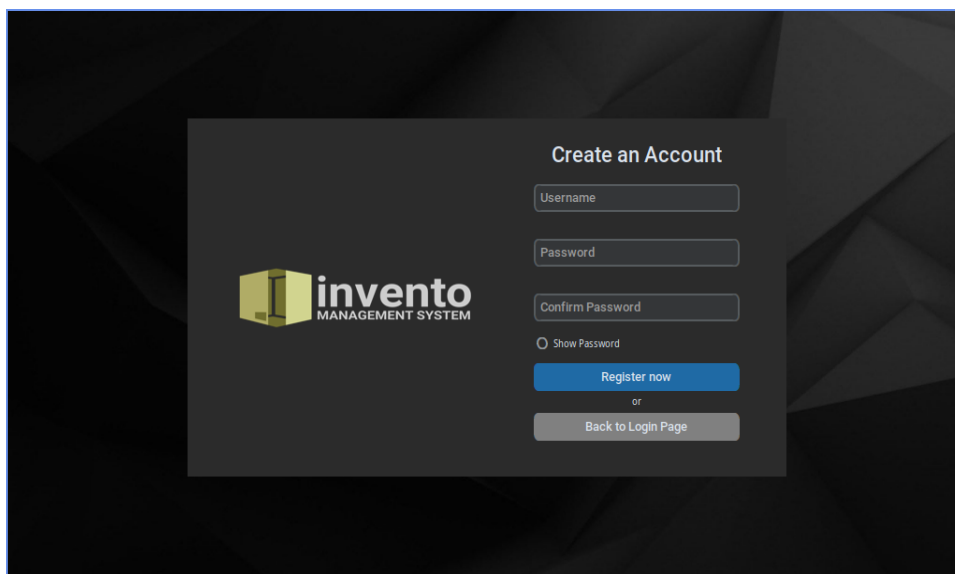
### </> LoginPage.py

The `LoginPage` module creates an environment inheriting the `LoginBg` where it allows the user to access the inventory. This also allows the modification of a user, be recorded.



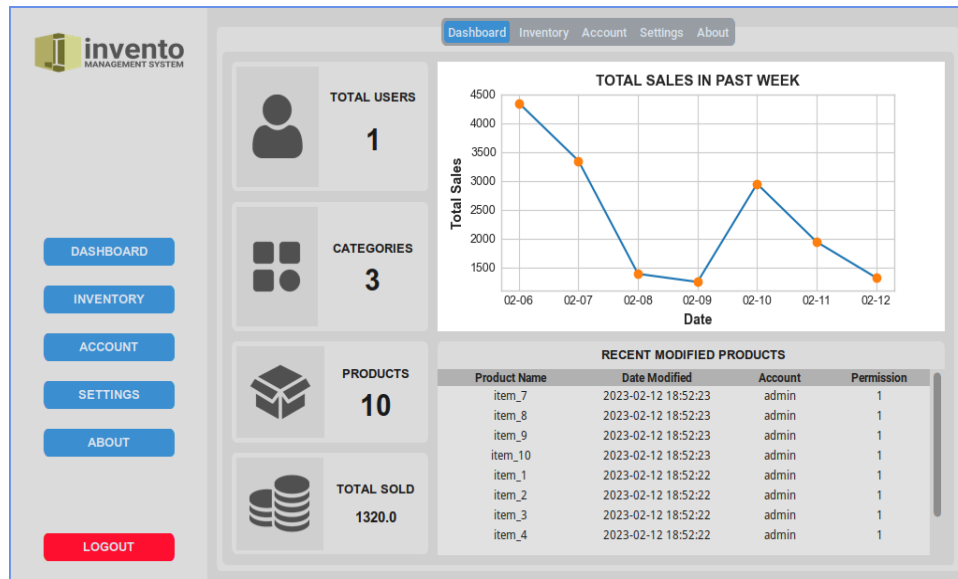
### </> RegisterPage.py

This module enables the creation of an account.



## </> InventoryPage.py

Holds all the tabs, buttons to traverse the tabs, and the logout button.



### # DASHBOARD

```
self.tabview.tab("Dashboard").grid_columnconfigure(0, weight=1)
self.tabview.tab("Dashboard").grid_rowconfigure(0, weight=1)
self.dashboardDisplay = DashboardTab(self.tabview.tab("Dashboard"))
self.dashboardDisplay.grid(row=0, column=0, sticky="nsew")
```

### # INVENTORY

```
self.tabview.tab("Inventory").grid_columnconfigure(0, weight=1)
self.tabview.tab("Inventory").grid_rowconfigure(0, weight=1)
self.inventoryDisplay = ProductTab(self.tabview.tab("Inventory"), controller)
self.inventoryDisplay.grid(row=0, column=0, sticky="nsew")
```

### # ACCOUNT

```
self.tabview.tab("Account").grid_columnconfigure(0, weight=1)
self.tabview.tab("Account").grid_rowconfigure(0, weight=1)
self.accountDisplay = AccountTab(self.tabview.tab("Account"))
self.accountDisplay.grid(row=0, column=0, sticky="nsew")
```

### # ABOUTMENU

```
self.tabview.tab("About").grid_columnconfigure(0, weight=1)
self.tabview.tab("About").grid_rowconfigure(0, weight=1)
self.aboutDisplay = AboutTab(self.tabview.tab("About"))
self.aboutDisplay.grid(row=0, column=0, sticky="nsew")
```

### # SETTINGS

```
self.tabview.tab("Settings").grid_columnconfigure(0, weight=1)
self.tabview.tab("Settings").grid_rowconfigure(0, weight=1)
self.settingsDisplay = SettingsTab(self.tabview.tab("Settings"), controller)
self.settingsDisplay.grid(row=0, column=0, sticky="nsew")
```

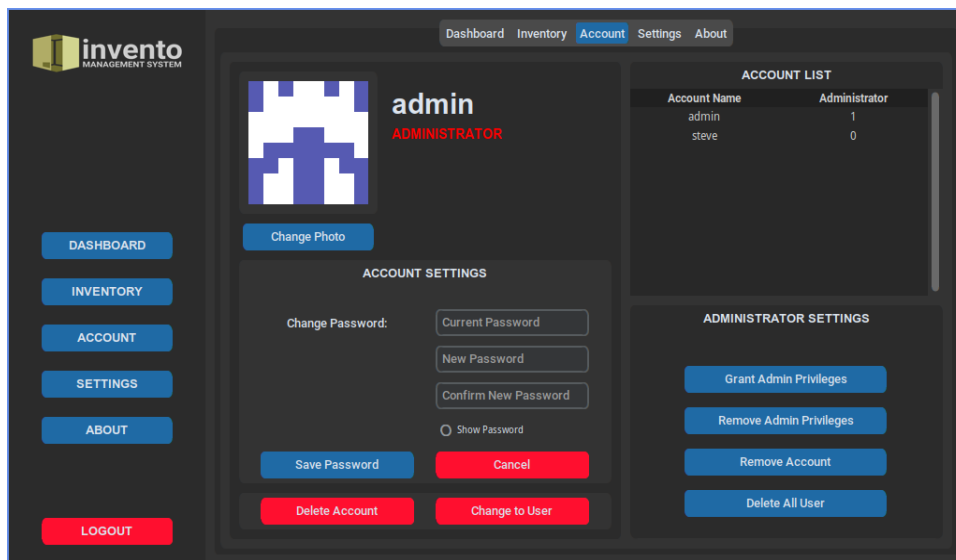
## Tabs

This package contains all the frames for `InventoryPage`. Every core functionality of this program is accessed in this part. Each tab provides a different set of features and functionality.



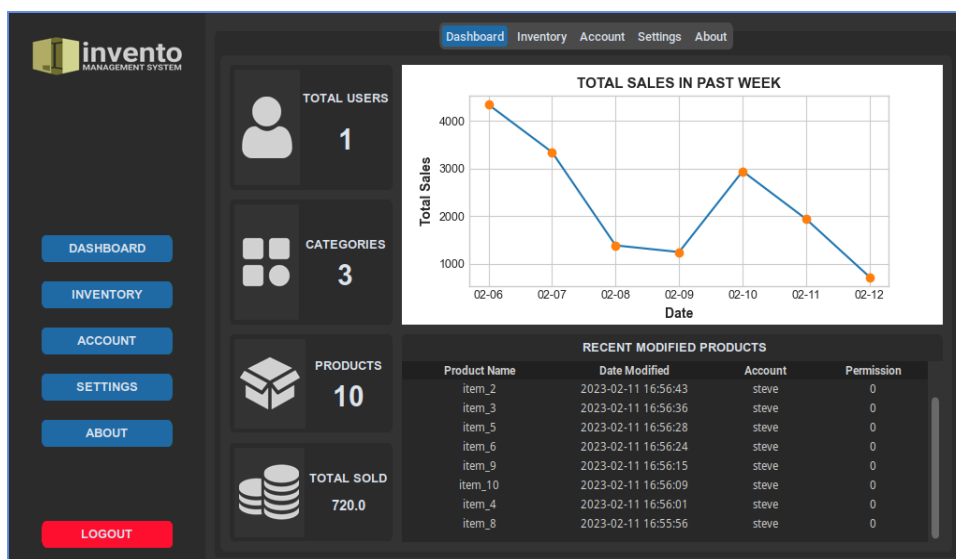
### </> AccountTab.py

The `AccountTab` module provides the functionality for managing accounts. Users can update their password and delete their account. Administrators on the other hand, has more access to modify its own and other user accounts.



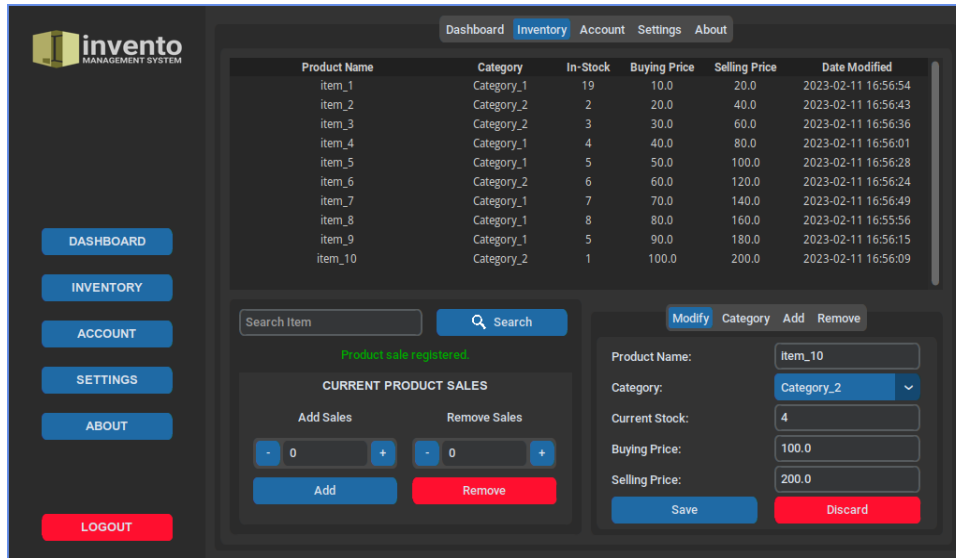
### </> DashboardTab.py

The `DashboardTab` provides an overview of the inventory’s modification and sales.



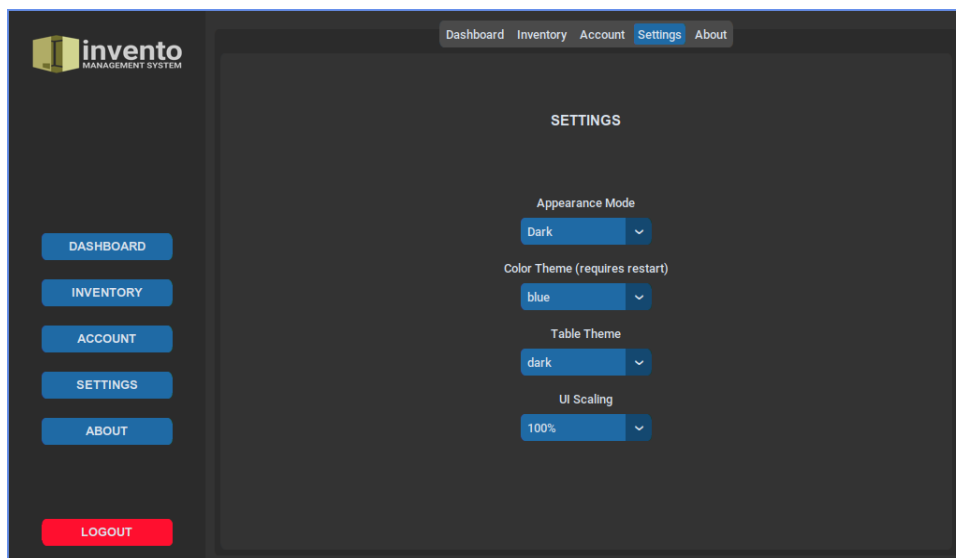
## </> ProductTab.py

The ProductTab provides the functionality to manage the inventory of the business. Users can basically, add, edit, and delete products in the inventory. It also does track the sales in this part.



## </> SettingsTab.py

This part provides options for customizing the application’s appearance, theme, and scaling.



## </> AboutTab.py

Shows a short description of the program, the team who created this project, and the core features.

..... END .....