```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
launcher aka entry point
#weekly sale
"""

import os
import sys
import subprocess
from app_config import BASE_DIR, IMAGES_DIR
print("Python executable:", sys.executable)
print("Python version:", sys.version)

def main():
    # Get the project root directory (parent of this launcher script)
    project_root = BASE_DIR

    # Run the gooey.main_window module using python -m
    #subprocess.run([sys.executable, "-m", "gooey.main_window"], cwd=project_root)

    subprocess.run([sys.executable, "-m", "core.no_gui_main"], cwd=project_root)

if __name__ == "__main__":
    main()
```

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
gooey main window:D
"""

# gui/main_window.py

import tkinter as tk
# import os
from tkinter import ttk
from tkinter import messagebox
# from gooey.header import Header
# from pathlib import Path
from app_config import BASE_DIR, ASSETS_DIR, GRAPHICS_EXCELS_DIR, IMAGES_DIR
# from core.excel_parser import parse_excel_to_collage
# from gooey.image_selector_frame import ItemSelectorFrame
# from typing import Optional
#from tkinter import filedialog
# import threading
from gooey.appclass import App
# from core.image_utils import build_image_index

def main():
    app = App()
    app.mainloop()

if __name__ == "__main__":
    main()
```

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
weekly sale
"""

import tkinter as tk
from tkinter import ttk

def open_instructions_window():
    instructions_win = tk.Toplevel()
    instructions_win.title("Instructions")
    instructions_win.geometry("400x300")

    label = ttk.Label(instructions_win, text="Instructions go here.", font=("Helvetica", 12
))
    label.pack(pady=20)

    close_btn = ttk.Button(instructions_win, text="Close", command=instructions_win.destroy
)
    close_btn.pack(pady=10)
```

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
album class stuff for weekly sale

"""
```

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
SUBFRAMES OF CREATEGRAPHIC
"""

import tkinter as tk
from tkinter import filedialog
from core.no_gui_main import make_collage
import tkinter as tk
from tkinter import filedialog

class SelectExcelFrame(tk.Frame):
    def __init__(self, parent, controller):
        super().__init__(parent, bg="#f7b4c6")  # Match background
        self.controller = controller  # This should be CreateFlyerPage instance managing st
eps
        self.selected_file = None

        # Center container
        container = tk.Frame(self, bg="#f7b4c6")
        container.place(relx=0.5, rely=0.5, anchor="center")

        # Instruction label
        tk.Label(container, text="Select an Excel file to load", font=("Helvetica", 16), bg
="#f7b4c6").pack(pady=20)

        # Browse button
        select_btn = tk.Button(container, text="Browse Excel File...", command=self.browse_
file)
        select_btn.pack(pady=10)

        # Selected filename label
        self.file_label = tk.Label(container, text="No file selected", fg="gray", bg="#f7b4
c6")
        self.file_label.pack(pady=5)

        # Next button (disabled until file selected)
        self.next_btn = tk.Button(container, text="Next", command=self.go_next, state="disa
bled")
        self.next_btn.pack(pady=20)

    def browse_file(self):
        file_path = filedialog.askopenfilename(
            title="Select Excel file",
            filetypes=[("Excel Files", "*.xlsx *.xls")]
        )
        if file_path:
            self.selected_file = file_path
            filename = file_path.split("/")[-1]  # just filename
            self.file_label.config(text=f"Selected: {filename}", fg="black")
            self.next_btn.config(state="normal")

    def go_next(self):
        if not self.selected_file:
            return

        # Save the selected file path in CreateFlyerPage's flyer_data dictionary
        self.controller.flyer_data['excel_path'] = self.selected_file

        # Call make_collage function (should be imported or defined elsewhere)
        collage_result = make_collage(self.selected_file)
        self.controller.collage = collage_result

        print("Collage created:", collage_result is not None)

        # Move to next step in CreateFlyerPage (step2)
        self.controller.show_step('step2')
```

```
class ChooseImagesFrame(tk.Frame):
    def __init__(self, parent, controller):
        super().__init__(parent)
        self.controller = controller

class PreviewEntriesFrame(tk.Frame):
    def __init__(self, parent, controller):
        super().__init__(parent)
        self.controller = controller

class ConfirmEditEntryFrame(tk.Frame):
    def __init__(self, parent, controller):
        super().__init__(parent)
        self.controller = controller

class DonePageFrame(tk.Frame):
    def __init__(self, parent, controller):
        super().__init__(parent)
        self.controller = controller
```

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sun Jun 29 17:01:11 2025

APP GOOEY I MOVED IT OUT OF STATE.py
"""
from app_config import IMG_EXTS, IMAGES_DIR
import tkinter as tk
from tkinter import filedialog, messagebox
from tkinter import ttk, Scrollbar, Listbox, StringVar, END
from core.state import AppState
from core.image_loader import load_image
from gooey.header import Header
import os
from core.no_gui_main import make_collage
from core.image_utils import find_matching_images

import warnings
warnings.simplefilter('always')

import threading
import queue
from PIL import Image, ImageTk
#from gooey.CreateFlyerPage_Subpages import SelectExcelFrame, ChooseImagesFrame, PreviewEnt
riesFrame, ConfirmEditEntryFrame, DonePageFrame

class App(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("ANNAXANNAXANNA")
        self.geometry("1000x800")
        self.configure(bg="#f7b4c6")

        # app state
        self.state = AppState(IMAGES_DIR)

        container = tk.Frame(self)
        container.pack(fill="both", expand=True)
        container.grid_rowconfigure(0, weight=1)
        container.grid_columnconfigure(0, weight=1)

        self.frames = {}

        for F in (StartPage, ProcessImagesPage, CreateFlyerPage, UpdateImagePage):
            page_name = F.__name__
            frame = F(parent=container, controller=self)
            self.frames[page_name] = frame
            frame.grid(row=0, column=0, sticky="nsew")

        self.show_frame("StartPage")

    def show_frame(self, page_name, **kwargs):
        frame = self.frames[page_name]
        if hasattr(frame, 'set_data'):
            frame.set_data(**kwargs)
        frame.tkraise()

    def show_home(self):
    # Clear old CreateFlyerPage
        old_page = self.frames.get("CreateFlyerPage")
        if old_page:
            old_page.destroy()

        # Reinitialize it
        container = old_page.master  # all pages share the same parent container
        new_page = CreateFlyerPage(parent=container, controller=self)
        new_page.grid(row=0, column=0, sticky="nsew")
        self.frames["CreateFlyerPage"] = new_page
```

```
        # Show StartPage (or your actual home page)
        self.show_frame("StartPage")

class StartPage(tk.Frame):
    def __init__(self, parent, controller):
        super().__init__(parent,bg="#f7b4c6")
        self.controller = controller

        header = Header(self, controller)
        header.pack(side="top", fill="x")


        tk.Label(self, text="Welcome!", font=("Helvetica", 40, "bold italic"),bg="#084b39",
fg="#f7b4c6").pack(pady=20)

        tk.Button(self, text="Process Images",
                  command=lambda: controller.show_frame("ProcessImagesPage")).pack(pady=10)

        tk.Button(self, text="Create Graphic Flyer",
                  command=lambda: controller.show_frame("CreateFlyerPage")).pack(pady=10)


# Assuming you have these already:
# - load_image(path, size, as_tk)
# - clear_cache()
# - Header class
# - controller.state.query_images
# - controller.show_frame

class ProcessImagesPage(tk.Frame):
    def __init__(self, parent, controller):
        super().__init__(parent, bg="#f7b4c6")
        self.controller = controller

        self.load_generation = 0  # to track image load cycles

        # Add header at the top
        header = Header(self, controller)
        header.grid(row=0, column=0, sticky="ew")

        # Configure grid for the rest of the page content
        self.grid_rowconfigure(3, weight=1)  # scrollable results area row
        self.grid_columnconfigure(0, weight=1)

        # --- Search & filter controls ---
        search_frame = tk.Frame(self, bg="#f7b4c6")
        search_frame.grid(row=1, column=0, sticky="ew", padx=10, pady=5)
        search_frame.grid_columnconfigure(4, weight=1)

        tk.Label(search_frame, text="Categories:", bg="#f7b4c6").grid(row=0, column=0, stic
ky="w")
        self.category_listbox = tk.Listbox(search_frame, selectmode=tk.MULTIPLE, height=5,
exportselection=False)
        self.category_listbox.grid(row=1, column=0, sticky="w")
        self.update_category_list()

        tk.Label(search_frame, text="Keyword:", bg="#f7b4c6").grid(row=0, column=1, sticky=
"w", padx=(10,0))
        self.keyword_entry = tk.Entry(search_frame)
        self.keyword_entry.grid(row=1, column=1, sticky="w", padx=(10,0))

        tk.Label(search_frame, text="Sort by:", bg="#f7b4c6").grid(row=0, column=2, sticky=
"w", padx=(10,0))
        self.sort_by_var = tk.StringVar(value='name')
        sort_options = ['name', 'category']
        ttk.OptionMenu(search_frame, self.sort_by_var, 'name', *sort_options).grid(row=1, c
olumn=2, sticky="w", padx=(10,0))
```

```
        self.reverse_var = tk.BooleanVar(value=False)
        tk.Checkbutton(search_frame, text="Descending", variable=self.reverse_var, bg="#f7b
4c6").grid(row=1, column=3, sticky="w", padx=(10,0))

        tk.Button(search_frame, text="Search", command=self.run_query).grid(row=1, column=4
, sticky="w", padx=(10,0))

        # --- Scrollable results area ---
        results_frame = tk.Frame(self, relief="sunken", borderwidth=1)
        results_frame.grid(row=3, column=0, sticky="nsew", padx=10, pady=5)
        results_frame.grid_rowconfigure(0, weight=1)
        results_frame.grid_columnconfigure(0, weight=1)

        self.canvas = tk.Canvas(results_frame, bg="#f7b4c6")
        self.scrollbar = tk.Scrollbar(results_frame, orient="vertical", command=self.canvas
.yview)
        self.scrollable_frame = tk.Frame(self.canvas, bg="#f7b4c6")

        self.scrollable_frame.bind("<Configure>", lambda e: self.canvas.configure(scrollreg
ion=self.canvas.bbox("all")))
        self.canvas.create_window((0, 0), window=self.scrollable_frame, anchor="nw")
        self.canvas.configure(yscrollcommand=self.scrollbar.set)

        self.canvas.grid(row=0, column=0, sticky="nsew")
        self.scrollbar.grid(row=0, column=1, sticky="ns")

        # --- Pagination ---
        pagination_frame = tk.Frame(self, bg="#f7b4c6")
        pagination_frame.grid(row=4, column=0, sticky="ew", padx=10, pady=5)
        pagination_frame.grid_columnconfigure(1, weight=1)

        self.prev_button = tk.Button(pagination_frame, text="Previous", command=self.prev_p
age)
        self.prev_button.grid(row=0, column=0, sticky="w")

        self.page_label = tk.Label(pagination_frame, text="Page 1", bg="#f7b4c6")
        self.page_label.grid(row=0, column=1)

        self.next_button = tk.Button(pagination_frame, text="Next", command=self.next_page)
        self.next_button.grid(row=0, column=2, sticky="e")

        # Data and pagination state
        self.current_results = []
        self.page = 0
        self.page_size = 15

        # Queue and threading setup
        self.load_queue = queue.Queue()
        self.after(100, self.process_load_queue)

        # Initial load
        self.run_query()

    def update_category_list(self):
        self.category_listbox.delete(0, tk.END)
        for cat in sorted(self.controller.state.categories):
            self.category_listbox.insert(tk.END, cat)

    def run_query(self):
        selected_indices = self.category_listbox.curselection()
        selected_categories = [self.category_listbox.get(i) for i in selected_indices]
        keyword = self.keyword_entry.get().strip()
        sort_by = self.sort_by_var.get()
        reverse = self.reverse_var.get()

        self.current_results = self.controller.state.query_images(
            categories=selected_categories if selected_categories else None,
            keyword=keyword if keyword else None,
            sort_by=sort_by,
```

```
            reverse=reverse
        )
        self.page = 0
        self.load_generation += 1  # New load cycle
        self.load_current_page()

    def load_current_page(self):
        # Clear current UI entries
        for widget in self.scrollable_frame.winfo_children():
            widget.destroy()

        start = self.page * self.page_size
        end = start + self.page_size
        page_items = self.current_results[start:end]

        # Put load requests in the queue
        thumbnail_size = (100, 100)
        generation = self.load_generation  # capture current generation for closure

        for img in page_items:
            self.load_queue.put((img, thumbnail_size, generation))

        total_pages = max(1, (len(self.current_results) - 1) // self.page_size + 1)
        self.page_label.config(text=f"Page {self.page + 1} of {total_pages}")

        self.prev_button.config(state="normal" if self.page > 0 else "disabled")
        self.next_button.config(state="normal" if self.page < total_pages - 1 else "disable
d")

    def process_load_queue(self):
        try:
            while True:
                img, size, generation = self.load_queue.get_nowait()
                # Start a thread to load the image so UI stays responsive
                threading.Thread(target=self.load_image_thread, args=(img, size, generation
), daemon=True).start()
        except queue.Empty:
            pass
        self.after(100, self.process_load_queue)  # Keep checking queue periodically

    def load_image_thread(self, img, size, generation):
        tk_img = load_image(img.file_path, size=size)
        # Return to main thread to update UI
        self.after(0, self.create_image_button, img, tk_img, generation)

    def create_image_button(self, img, tk_img, generation):
        # Only update UI if this load is still current
        if generation != self.load_generation:
            return  # Stale load, ignore

        # Find scrollable_frame is still valid (hasn't been destroyed)
        if not self.scrollable_frame.winfo_exists():
            return

        # Create the frame and button for the image
        row_frame = tk.Frame(self.scrollable_frame, bg="#f7b4c6")
        row_frame.pack(fill="x", padx=5, pady=5)

        if tk_img:
            btn = tk.Button(row_frame, image=tk_img,
                            command=lambda i=img: self.controller.show_frame('UpdateImagePa
ge', image_metadata=i))
            btn.image = tk_img  # keep reference
            btn.pack(side="left", padx=5)
        else:
            tk.Label(row_frame, text="[Missing image]", bg="#f7b4c6").pack(side="left", pad
x=5)

        text = f"{img.category} / {img.name}"
```

```
        tk.Label(row_frame, text=text, font=("Helvetica", 25), anchor="w", justify="left",
bg="#f7b4c6")\
            .pack(side="left", padx=10, fill="x", expand=True)

    def next_page(self):
        if (self.page + 1) * self.page_size < len(self.current_results):
            self.page += 1
            self.load_generation += 1  # Increment generation for new load
            self.load_current_page()

    def prev_page(self):
        if self.page > 0:
            self.page -= 1
            self.load_generation += 1
            self.load_current_page()


""" UPDATEIMAGEPAGE"""

class UpdateImagePage(tk.Frame):
    def __init__(self, parent, controller):
        super().__init__(parent, bg="#f7b4c6")
        self.controller = controller

        # Header
        tk.Label(self, text="Update Image", font=("Helvetica", 40, "bold italic"),
                 bg="#084b39", fg="#f7b4c6").pack(pady=10)

        # Image display
        self.image_label = tk.Label(self, bg="#f7b4c6")
        self.image_label.pack(pady=10)

        # Frame for inputs
        form_frame = tk.Frame(self, bg="#f7b4c6")
        form_frame.pack(pady=10)

        # Filename input
        tk.Label(form_frame, text="Filename:", bg="#f7b4c6").grid(row=0, column=0, sticky="
w", padx=5, pady=5)
        self.filename_entry = tk.Entry(form_frame, width=40)
        self.filename_entry.grid(row=0, column=1, padx=5, pady=5)

        # Folder/category dropdown
        tk.Label(form_frame, text="Folder (category):", bg="#f7b4c6").grid(row=1, column=0,
 sticky="w", padx=5, pady=5)
        self.folder_var = tk.StringVar()
        self.folder_menu = ttk.OptionMenu(form_frame, self.folder_var, '')
        self.folder_menu.grid(row=1, column=1, padx=5, pady=5)

        # Save button
        save_btn = tk.Button(self, text="Save", command=self.save_changes)
        save_btn.pack(pady=10)

        # Back button (optional)
        back_btn = tk.Button(self, text="Back", command=lambda: controller.show_frame('Proc
essImagesPage'))
        back_btn.pack(pady=5)

        # To store image metadata
        self.image_metadata = None
        self.tk_img = None  # keep reference to prevent GC

    def set_data(self, image_metadata):
        """Called by controller.show_frame to pass image metadata"""
        self.image_metadata = image_metadata

        # Prefill filename
        self.filename_entry.delete(0, tk.END)
        self.filename_entry.insert(0, image_metadata.name)
```

```
        # Update folder dropdown options
        categories = sorted(self.controller.state.categories)
        menu = self.folder_menu["menu"]
        menu.delete(0, "end")
        for cat in categories:
            menu.add_command(label=cat, command=lambda c=cat: self.folder_var.set(c))
        # Set current category
        self.folder_var.set(image_metadata.category)

        # Load and show image
        thumbnail_size = (300, 300)
        self.tk_img = load_image(image_metadata.file_path, size=thumbnail_size, as_tk=True)
        if self.tk_img:
            self.image_label.config(image=self.tk_img, text="")
        else:
            self.image_label.config(text="[Failed to load image]", image="")

    def save_changes(self):
        new_name = self.filename_entry.get().strip()
        new_category = self.folder_var.get().strip()

        if not new_name:
            messagebox.showerror("Error", "Filename cannot be empty.")
            return

        # Check for invalid characters (basic check)
        if any(c in new_name for c in r'<>:"/\|?*'):
            messagebox.showerror("Error", "Filename contains invalid characters.")
            return

        # Rename if folder/category changed
        old_path = self.image_metadata.file_path
        current_category = self.image_metadata.category

        if new_category != current_category:
            # Move to new folder
            new_dir = os.path.join(self.controller.state.root_folder, new_category)
            if not os.path.exists(new_dir):
                os.makedirs(new_dir)

            new_path = os.path.join(new_dir, new_name)
        else:
            # Same folder, just rename
            new_path = os.path.join(os.path.dirname(old_path), new_name)

        try:
            self.controller.state.rename_image(self.image_metadata, new_name)
            # If category changed, physically move file
            if new_category != current_category:
                os.replace(old_path, new_path)
                # Update metadata
                self.image_metadata.file_path = new_path
                self.image_metadata.category = new_category
                self.controller.state.index_images()  # refresh all data

            messagebox.showinfo("Success", "Image updated successfully!")
            # Optionally go back
            self.controller.show_frame('ProcessImagesPage')
        except Exception as e:
            messagebox.showerror("Error", f"Failed to rename/move image: {e}")

class CreateFlyerPage(tk.Frame):
    def __init__(self, parent, controller):
        super().__init__(parent)
        self.controller = controller

        self.subpages = {}
        self.collage = None
```

```
        self.current_item_index = 0

        self.init_subpages()
        self.show_subpage("get_excel")

    def init_subpages(self):
        self.subpages["get_excel"] = GetExcelPage(self, self.controller)
        self.subpages["select_image"] = SelectImagePage(self, self.controller)
        self.subpages["review"] = ReviewPage(self, self.controller)
        self.subpages["done"] = DonePage(self, self.controller)

        for page in self.subpages.values():
            page.pack(fill='both', expand=True)
            page.pack_forget()

    def show_subpage(self, name):
        for page in self.subpages.values():
            page.pack_forget()
        self.subpages[name].pack(fill='both', expand=True)

    def load_excel(self):
        file_path = filedialog.askopenfilename(filetypes=[("Excel Files", "*.xlsx")])
        if file_path:
            try:
                collage = make_collage(file_path)
                if not collage or not hasattr(collage, 'items_list') or not collage.items_l
ist:
                    messagebox.showerror("Error", "Collage has no items.")
                    self.show_subpage("get_excel")
                    return

                self.collage = collage
                self.current_item_index = 0
                self.start_select_images()
            except ValueError:
                messagebox.showerror("Error", "Invalid data, please try again.")
                self.show_subpage("get_excel")

    def start_select_images(self):
        if self.current_item_index < len(self.collage.items_list):
            item = self.collage.items_list[self.current_item_index]

            # get full list of dicts
            state = self.controller.state
            search_results = find_matching_images(item.name, item.chinese_name, state)

            item.possible_images = search_results  # keep full data

            self.subpages["select_image"].load_item(item)
            self.show_subpage("select_image")
        else:
            self.go_to_review()


    def next_item(self):
        self.current_item_index += 1
        self.start_select_images()

    def go_to_review(self):
        self.subpages["review"].load_items(self.collage.items_list)
        self.show_subpage("review")

    def confirm_flyer(self):
        try:
            # Add your save logic here, e.g. self.collage.save()
            self.subpages["done"].set_message("File saved successfully!")
        except Exception:
            self.subpages["done"].set_message("Problem saving file.")
        self.show_subpage("done")
```

```
    def return_home(self):
        self.controller.show_home()


class GetExcelPage(tk.Frame):
    def __init__(self, parent, controller):
        super().__init__(parent)
        self.controller = controller
        self.master_page = parent  # parent is CreateFlyerPage

        tk.Label(self, text="Step 1: Select Excel File").pack(pady=20)
        tk.Button(self, text="Choose File", command=self.master_page.load_excel).pack(pady=
10)
        tk.Button(self, text="Cancel", command=self.controller.show_home).pack(pady=10)

class SelectImagePage(tk.Frame):
    """
    Wizard subpage to let user select an image for one item.
    Shows possible images as thumbnails in a grid (no scrollbar).
    """
    def __init__(self, parent, controller):
        super().__init__(parent)
        self.controller = controller
        self.item = None  # current item instance

        # Title label (item name + chinese name)
        self.title_label = tk.Label(self, text="", font=('Arial', 14))
        self.title_label.pack(pady=10)

        # Frame for image buttons grid
        self.grid_frame = tk.Frame(self)
        self.grid_frame.pack(padx=10, pady=10)

        # Keep references to image button widgets and their images
        self.image_buttons = []
        self.image_refs = []

        # Cancel button
        tk.Button(self, text="Cancel", command=self.controller.show_home).pack(pady=10)

    def load_item(self, item):
        self.item = item
        self.title_label.config(text=f"{item.name} / {item.chinese_name}")

        # Clear old buttons & images
        for btn in self.image_buttons:
            btn.destroy()
        self.image_buttons.clear()
        self.image_refs.clear()

        search_results = item.possible_images

        columns = 4
        row = 0
        col = 0

        for res in search_results:
            img_path = res['path']
            score = res.get('score', 0)

            thumb = load_image(img_path, size=(100, 100))

            btn = tk.Button(self.grid_frame, image=thumb,
                            command=lambda p=img_path: self.select_image(p))
            btn.image = thumb  # keep reference
            btn.grid(row=row, column=col, padx=5, pady=5)

            lbl = tk.Label(self.grid_frame, text=f"Score: {score}")
```

```
            lbl.grid(row=row+1, column=col, padx=5, pady=(0,10))

            self.image_buttons.append(btn)
            self.image_buttons.append(lbl)
            self.image_refs.append(thumb)

            col += 1
            if col >= columns:
                col = 0
                row += 2


    def select_image(self, img_path):
        """
        Called when user clicks on an image:
        - set selected image
        - tell CreateFlyerPage to go to next item
        """
        if self.item:
            self.item.set_selected_image(img_path)
            print(f"Selected image for {self.item.name}: {img_path}")

        flyer_page = self.controller.frames.get("CreateFlyerPage")
        if flyer_page:
            flyer_page.next_item()
        else:
            print("Error: CreateFlyerPage not found in controller.frames")



class ReviewPage(tk.Frame):
    def __init__(self, parent, controller):
        super().__init__(parent)
        self.controller = controller
        self.master_page = parent

        # Vertical scrollable frame
        self.scroll_canvas = tk.Canvas(self)
        self.scrollbar = ttk.Scrollbar(self, orient='vertical', command=self.scroll_canvas.
yview)
        self.scroll_canvas.configure(yscrollcommand=self.scrollbar.set)

        self.scroll_frame = tk.Frame(self.scroll_canvas)
        self.scroll_canvas.create_window((0,0), window=self.scroll_frame, anchor='nw')
        self.scroll_frame.bind("<Configure>", lambda e: self.scroll_canvas.configure(scroll
region=self.scroll_canvas.bbox("all")))

        self.scroll_canvas.pack(side='left', fill='both', expand=True)
        self.scrollbar.pack(side='right', fill='y')

        tk.Button(self, text="Confirm and Save", command=self.master_page.confirm_flyer).pa
ck(pady=10)
        tk.Button(self, text="Cancel", command=self.controller.show_home).pack(pady=5)

    def load_items(self, items):
        for widget in self.scroll_frame.winfo_children():
            widget.destroy()
        for item in items:
            frame = tk.Frame(self.scroll_frame, pady=2)
            tk.Label(frame, text=item.name).pack(side='left', padx=5)
            tk.Label(frame, text=item.chinese_name).pack(side='left', padx=5)
            tk.Label(frame, text=item.price).pack(side='left', padx=5)
            if item.selected_image:
                thumb = load_thumbnail(item.selected_image)
                lbl = tk.Label(frame, image=thumb)
                lbl.image = thumb  # keep reference
                lbl.pack(side='left', padx=5)
            frame.pack(fill='x', pady=2)
```

```
class DonePage(tk.Frame):
    def __init__(self, parent, controller):
        super().__init__(parent)
        self.controller = controller
        self.master_page = parent

        self.message = tk.Label(self, text="", font=('Arial', 14))
        self.message.pack(pady=20)

        tk.Button(self, text="Return Home", command=self.controller.show_home).pack(pady=10
)

    def set_message(self, text):
        self.message.config(text=text)

if __name__ == "__main__":
    app = App()
    app.mainloop()
```

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sun Jun 29 15:05:28 2025

@author: annax
WEEKLY SALE initializer (GOOEY)

in the main gooey we create an instance of this app class to start the app
"""


class App(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("GRAPHIC ASSISTANT")
        self.geometry("1000x600")

        # app state
        self.state = AppState(IMAGES_DIR)

        # container holds current frame
        container = tk.Frame(self)
        container.pack(fill="both", expand=True)

        self.frames = {}  # keeps all pages

        for F in (StartPage, ProcessImagesPage, CreateFlyerPage):
            page_name = F.__name__
            frame = F(parent=container, controller=self)
            self.frames[page_name] = frame

            # all frames are stacked but only one is shown
            frame.grid(row=0, column=0, sticky="nsew")

        self.show_frame("StartPage")

    def show_frame(self, page_name):
        '''Raise the frame by name'''
        frame = self.frames[page_name]
        frame.tkraise()

class StartPage(tk.Frame):
    def __init__(self, parent, controller):
        super().__init__(parent)
        tk.Label(self, text="Welcome!", font=("Helvetica", 16)).pack(pady=20)

        tk.Button(self, text="Process Images",
                  command=lambda: controller.show_frame("ProcessImagesPage")).pack(pady=10)

        tk.Button(self, text="Create Graphic Flyer",
                  command=lambda: controller.show_frame("CreateFlyerPage")).pack(pady=10)

class ProcessImagesPage(tk.Frame):
    def __init__(self, parent, controller):
        super().__init__(parent)


    def select_folder(self):
        folder = filedialog.askdirectory(title="Select Images Root Folder")
        if folder:
            self.controller.state.images_dir = folder
            messagebox.showinfo("Done", f"Set images directory to:\n{folder}")

class CreateFlyerPage(tk.Frame):
    def __init__(self, parent, controller):
        super().__init__(parent)
        tk.Label(self, text="Create Graphic Flyer", font=("Helvetica", 14)).pack(pady=10)
```

```
        #tk.Button(self, text="Select Excel File", command=self.select_excel).pack(pady=10)

        tk.Button(self, text="Back", command=lambda: controller.show_frame("StartPage")).pa
ck(pady=10)

        self.controller = controller

"""    def select_excel(self):
        excel_file = filedialog.askopenfilename(
            title="Select Excel File",
            filetypes=[("Excel files", "*.xlsx")]
        )
        if excel_file:
            basename = os.path.basename(excel_file)
            collage = make_collage(basename)
            if collage:
                u_choose_images(collage, self.controller.state)
                make_n_save_graphic(collage)
                messagebox.showinfo("Success", "Flyer created and saved!")
            else:
                messagebox.showerror("Error", "Failed to create flyer.")"""
```

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
init gooey
#weekly sale
"""
```

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
# gui/header.py
"""



import tkinter as tk
from gooey.instructions import open_instructions_window

class Header(tk.Frame):
    def __init__(self, parent, controller):
        super().__init__(parent, bg='#084b39')  # Header background color
        self.controller = controller
        self.configure(padx=10, pady=10)

        home_btn = tk.Button(
            self,
            text="Home",
            bg='#f7b4c6',
            fg='#000000',
            activebackground='#d9a8b2',
            activeforeground='#000000',
            command=lambda: controller.show_frame("StartPage")
        )
        home_btn.pack(side=tk.LEFT, padx=(0, 10))

        instructions_btn = tk.Button(
            self,
            text="Instructions",
            bg='#f7b4c6',
            fg='#000000',
            activebackground='#d9a8b2',
            activeforeground='#000000',
            command=open_instructions_window
        )
        instructions_btn.pack(side=tk.RIGHT)
```

```
import os
import tkinter as tk
from tkinter import ttk
from core.image_loader import load_image  # Ensure this provides PIL image loading and TK c
onversion

class ItemSelectorFrame(ttk.Frame):
    def __init__(self, master, items, image_index):
        super().__init__(master)
        self.items = items
        self.image_index = image_index
        self.item_frames = []

        # Reset All button
        top_controls = ttk.Frame(self)
        top_controls.pack(fill='x', pady=(5, 0))
        ttk.Button(top_controls, text="Reset All Images", command=self.reset_all_images).pa
ck(side="right", padx=5)

        # Scrollable area
        self.canvas = tk.Canvas(self)
        self.scrollbar = ttk.Scrollbar(self, orient="vertical", command=self.canvas.yview)
        self.canvas.configure(yscrollcommand=self.scrollbar.set)

        self.scrollable_frame = ttk.Frame(self.canvas)
        self.canvas.create_window((0, 0), window=self.scrollable_frame, anchor="nw")

        self.scrollable_frame.bind(
            "<Configure>",
            lambda e: self.canvas.configure(scrollregion=self.canvas.bbox("all"))
        )

        self.canvas.pack(side="left", fill="both", expand=True)
        self.scrollbar.pack(side="right", fill="y")

        # Add expandable frames
        for item in self.items:
            item_frame = ExpandableItemFrame(self.scrollable_frame, item, self.image_index)
            item_frame.pack(fill="x", padx=5, pady=3)
            self.item_frames.append(item_frame)

    def reset_all_images(self):
        for frame in self.item_frames:
            frame.reset()


class ExpandableItemFrame(ttk.Frame):
    def __init__(self, master, item, image_index):
        super().__init__(master)
        self.item = item
        self.image_index = image_index
        self.expanded = False
        self.selected_button = None
        self.image_buttons = {}
        self.images_frame = None

        # Header UI
        header = ttk.Frame(self)
        header.pack(fill='x')
        ttk.Label(header, text=item.name).pack(side="left", padx=5)

        controls = ttk.Frame(header)
        controls.pack(side="right")
        ttk.Button(controls, text="Reset", command=self.reset).pack(side="left", padx=2)
        ttk.Button(controls, text="+", width=2, command=self.toggle).pack(side="left")

    def toggle(self):
        if self.expanded:
            self.collapse()
```

```
        else:
            self.expand()

    def expand(self):
        if not self.images_frame:
            self.images_frame = ttk.Frame(self)
            self.images_frame.pack(fill="x")

            canvas = tk.Canvas(self.images_frame, height=100)
            canvas.pack(side="top", fill="x", expand=True)
            scroll_x = ttk.Scrollbar(self.images_frame, orient="horizontal", command=canvas
.xview)
            scroll_x.pack(side="bottom", fill="x")
            canvas.configure(xscrollcommand=scroll_x.set)

            inner = ttk.Frame(canvas)
            canvas.create_window((0, 0), window=inner, anchor="nw")

            # Get candidate image paths from index
            hint = self.item.image_hint
            image_paths = self.image_index.get(hint, [])

            for img_path in image_paths:
                thumb = load_image(img_path, size=(80, 80), as_tk=True)
                if thumb is None:
                    continue

                frame = ttk.Frame(inner, borderwidth=2, relief="flat")
                frame.pack(side="left", padx=4)

                btn = ttk.Label(frame, image=thumb)
                btn.image = thumb  # prevent garbage collection
                btn.pack()
                btn.bind("<Double-Button-1>", lambda e, p=img_path, f=frame: self.select_im
age(p, f))

                self.image_buttons[img_path] = frame

            inner.update_idletasks()
            canvas.config(scrollregion=canvas.bbox("all"))
        else:
            self.images_frame.pack(fill="x")

        self.expanded = True

    def collapse(self):
        if self.images_frame:
            self.images_frame.pack_forget()
        self.expanded = False

    def select_image(self, path, frame):
        if self.selected_button:
            self.selected_button.config(relief="flat", borderwidth=2)

        frame.config(relief="solid", borderwidth=3)
        self.selected_button = frame
        self.item.selected_image_path = path
        print(f"[{self.item.name}] selected image: {os.path.basename(path)}")

    def reset(self):
        self.item.selected_image_path = None
        if self.selected_button:
            self.selected_button.config(relief="flat", borderwidth=2)
            self.selected_button = None
        print(f"[{self.item.name}] image reset.")
```

```
# -*- coding: utf-8 -*-
"""app_config.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1SEOX2HEy7eLmSycZTQewx6L6gRw-N-CJ
"""

## should be called app_config.py
from pathlib import Path
import os
import sys

if getattr(sys, 'frozen', False):
    # Running as a bundled executable
    BASE_DIR = os.path.dirname(sys.executable)
else:
    # Running in development (as a script)
    BASE_DIR = os.path.abspath(os.path.dirname(__file__))

ASSETS_DIR = os.path.join(BASE_DIR, "assets")
# gives the folder that the app is in

# Simple: just use absolute paths to the parent directory
PROJECT_PARENT = os.path.abspath(os.path.join(BASE_DIR, ".."))
GRAPHICS_EXCELS_DIR = os.path.join(PROJECT_PARENT, "graphicsExcels")
IMAGES_DIR = os.path.join(PROJECT_PARENT, "goodimages")

IMG_EXTS = ('.png', '.jpg', '.jpeg', '.avif')  # used for quick validation
FALLBACK_IMAGE = os.path.join(ASSETS_DIR,'blank_square.png')
HEADER_PATH = os.path.join(ASSETS_DIR,'newgolden_header.png')
BHEADER_PATH = os.path.join(ASSETS_DIR,'newgolden_bottomheader.png')

FONT_PATH_CN = os.path.join(ASSETS_DIR,'NotoSansSC-SemiBold.ttf')
FONT_PATH_EN = os.path.join(ASSETS_DIR,'NotoSansSC-SemiBold.ttf')
FONT_PATH_BOLD = os.path.join(ASSETS_DIR,'Shrikhand-Regular.ttf')
PRICEBOX_PATH = os.path.join(ASSETS_DIR,'pop_speech_bubb.png')
FILLER_PATH = os.path.join(ASSETS_DIR,'luckycat3.png')

# You can add other paths or settings here as well
```

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
fake main aka no gui testing
"""
import tkinter as tk
import os
from tkinter import ttk
from tkinter import messagebox
from gooey.header import Header
from pathlib import Path
from app_config import BASE_DIR, ASSETS_DIR, GRAPHICS_EXCELS_DIR, IMAGES_DIR
from core.excel_parser import parse_excel_to_collage
from gooey.image_selector_frame import ItemSelectorFrame
from typing import Optional
from tkinter import filedialog
import threading
from core.state import AppState
import fpdf

def main():

    print('hello world')

    #initialize app state
    state = AppState(IMAGES_DIR)

    excel = 'weeklysale10-01-25to11-05-25.xlsx'
    collage = make_collage(excel)
    #make sure object instance was created
    if collage != None:
        u_choose_images(collage, state)


    make_n_save_graphic(collage)




#turn excel into collage

def make_collage(excelfile):
#excelfile = str(input('excel filename'))
    excelpath = Path(os.path.join(GRAPHICS_EXCELS_DIR,excelfile))
    if excelpath.is_file():
        try:
            collage = parse_excel_to_collage(excelpath)
            return collage
        except IOError:
            print("couldn't open excel")
            return None

def u_choose_images(collage_inst,state):
    for item in collage_inst.items_list.values():
        #invoke the item method to generate possible images attribute
        item.possible_images = item.select_image(state)
        item.set_selected_image(item.possible_images[0]['path'])
        print(f"CHOOSING AN IMAGE{item.selected_image_path}")

def make_n_save_graphic(collage):
    collage.generate_pdf()
    collage.makepages()
    print('plzzz so closeS')
    grafic = collage.graphic
    save_as = collage.name+'.pdf'
    grafic.output(os.path.join(GRAPHICS_EXCELS_DIR,save_as))

if __name__ == "__main__":
```

```
main()
```

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
#image index
"""

import os

def build_image_index(images_dir):
    image_index = {}
    for dirpath, _, filenames in os.walk(images_dir):
        for f in filenames:
            if f.lower().endswith(('.png', '.jpg', '.jpeg', '.gif')):
                full_path = os.path.join(dirpath, f)
                image_index[f] = full_path  # key by filename (or tweak)
    return image_index
```

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
useful funcs core weeklysale
"""
DPI = 300
def pixtomm(pix):
    pix = float(pix)
    #print('hello',pix)
    mm = pix/DPI*25.4
    #print('hellomm',mm)
    return mm

def mmtopix(mm):
    mm = float(mm)
    pix = int( mm/25.4*DPI)
    return pix
```

```
# -*- coding: utf-8 -*-
"""excel_parser.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1ecQtjz-7PxQWmQNAWQauZyCuHAKwLl6z
"""

# going to be excel_parser.py
from pathlib import Path
import pandas as pd
from models.item import Item
from models.collage import Collage
from datetime import datetime
import re

def parse_excel_to_collage(excel_path: Path) -> Collage:
    df = pd.read_excel(excel_path)

    # Normalize column names
    df.columns = [col.lower().strip() for col in df.columns]

    required_columns = {'name', 'chinese name', 'price', 'image'}
    if not required_columns.issubset(df.columns):
        raise ValueError(f"Excel is missing required columns: {required_columns}")

    collage_name, start_date, end_date = extract_collage_metadata_from_filename(excel_path.
name)

    items = {
        i: Item(row['name'], row['chinese name'], row['price'], row['image'])
        for i, row in df.iterrows()
    }

    return Collage(name=collage_name, items_list= items, start_date=start_date, end_date=en
d_date)


def extract_collage_metadata_from_filename(filename: str):
    """
    Parses filenames like 'weeklysale01102025-01112025.xlsx' and returns:
    - Human-readable collage name
    - start_date (datetime)
    - end_date (datetime)
    """
    match = re.search(r'(\d{8})-(\d{8})', filename)
    if not match:
        return "Weekly Sale", None, None  # fallback

    start_str, end_str = match.groups()
    fmt = "%m%d%Y"

    try:
        start_date = datetime.strptime(start_str, fmt)
        end_date = datetime.strptime(end_str, fmt)

        # Format the name
        if start_date.year == end_date.year:
            if start_date.month == end_date.month:
                # Jan 10â\200\22311, 2025
                date_str = f"{start_date.strftime('%b')} {start_date.day}â\200\223{end_date
.day}, {start_date.year}"
            else:
                # Jan 30 â\200\223 Feb 2, 2025
                date_str = f"{start_date.strftime('%b %d')} â\200\223 {end_date.strftime('%
b %d')}, {start_date.year}"
        else:
            # Dec 31, 2024 â\200\223 Jan 1, 2025
```

```
            date_str = f"{start_date.strftime('%b %d, %Y')} â\200\223 {end_date.strftime('%
b %d, %Y')}"

        name = f"Weekly Sale {date_str}"
        return name, start_date, end_date

    except ValueError:
        return "Weekly Sale", None, None
```

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
#for indexing imgs
"""
import os
import re

from pypinyin import lazy_pinyin
from core.image_utils import score_filename
import tkinter as tk
from tkinter import ttk, Scrollbar, Listbox, StringVar, END
from app_config import IMG_EXTS, IMAGES_DIR
# core/state.py

class ImageMetadata:
    def __init__(self, path):
        self.file_path = path
        self.name = os.path.basename(path)
        self.category = os.path.basename(os.path.dirname(path))

        # Add created and modified timestamps (seconds since epoch)
        stat = os.stat(path)
        self.created_time = stat.st_ctime
        self.modified_time = stat.st_mtime


class AppState:
    def __init__(self, root_folder):
        self.root_folder = root_folder
        self.image_list = []
        self.image_dict = {}
        self.categories = set()
        self.index_images()



    def index_images(self):
        self.image_list.clear()
        self.image_dict.clear()
        self.categories.clear()

        for dirpath, _, files in os.walk(self.root_folder):
            for f in files:
                if f.lower().endswith(IMG_EXTS):
                    full_path = os.path.join(dirpath, f)
                    metadata = ImageMetadata(full_path)
                    self.image_list.append(metadata)
                    self.image_dict[full_path] = metadata
                    self.categories.add(metadata.category)

    def rename_image(self, image_metadata, new_full_path):
        old_path = image_metadata.file_path
        new_full_path = os.path.abspath(new_full_path)  # Ensure absolute path

        new_dir = os.path.dirname(new_full_path)
        if not os.path.exists(new_dir):
            raise FileNotFoundError(f"Target folder '{new_dir}' does not exist.")

        if os.path.exists(new_full_path):
            raise FileExistsError("File with this name already exists.")

        os.rename(old_path, new_full_path)

        # Update metadata
        image_metadata.file_path = new_full_path
        image_metadata.name = os.path.basename(new_full_path)
        image_metadata.category = os.path.basename(new_dir)
```

```
        # Update dict keys
        del self.image_dict[old_path]
        self.image_dict[new_full_path] = image_metadata


    def query_images(self, categories=None, keyword=None, sort_by='name', reverse=False):
        # If categories is None or empty list, include all images (no category filter)
        if not categories:
            filtered_images = self.image_list
        else:
            category_set = set(categories)
            filtered_images = [img for img in self.image_list if img.category in category_s
et]

        keywords = []
        if keyword:
            keywords = [kw.strip() for kw in re.split(r'\s+', keyword.lower()) if kw.strip(
)]
            # Filter further by keyword presence (optional but speeds scoring)
            filtered_images = [img for img in filtered_images if any(kw in img.name.lower()
 for kw in keywords)]

        if keywords:
            scored = []
            for img in filtered_images:
                score, _ = score_filename(img.name, keywords)
                scored.append((score, img))
            # Sort primarily by score descending
            scored.sort(key=lambda x: x[0], reverse=True)
            result = [img for score, img in scored]
            # Reverse if requested
            if reverse:
                result.reverse()
        else:
            # Sort by given sort key if no keywords
            if sort_by == 'name':
                result = sorted(filtered_images, key=lambda img: ''.join(lazy_pinyin(img.na
me)).lower(), reverse=reverse)
            elif sort_by == 'created':
                result = sorted(filtered_images, key=lambda img: img.created_time, reverse=
reverse)
            elif sort_by == 'modified':
                result = sorted(filtered_images, key=lambda img: img.modified_time, reverse
=reverse)
            else:
                raise ValueError(f"Unknown sort key: {sort_by}")

        return result



"""    def select_excel(self):
        excel_file = filedialog.askopenfilename(
            title="Select Excel File",
            filetypes=[("Excel files", "*.xlsx")]
        )
        if excel_file:
            basename = os.path.basename(excel_file)
            collage = make_collage(basename)
            if collage:
                u_choose_images(collage, self.controller.state)
                make_n_save_graphic(collage)
                messagebox.showinfo("Success", "Flyer created and saved!")
            else:
                messagebox.showerror("Error", "Failed to create flyer.")"""
```

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
#image_utils.py in core

"""

import os
import re
from app_config import BASE_DIR, ASSETS_DIR, IMAGES_DIR, IMG_EXTS, FALLBACK_IMAGE, FONT_PAT
H_EN, FONT_PATH_CN
from core.useful_funcs import DPI
from PIL import Image, ImageDraw, ImageFont
import warnings
warnings.simplefilter('always')

# Make sure these are defined in your config or passed in
IMAGE_SEARCH_ROOT = IMAGES_DIR
image_extensions = IMG_EXTS
scale = int(DPI/70)
def extract_keywords(text):
    return set(re.findall(r'[\u4e00-\u9fff]+|\w+', text.lower()))


def find_matching_images(name, chinese_name, state, additional_search=None):
    keywords = extract_keywords(name + " " + chinese_name)

    if additional_search:
        keywords.update(extract_keywords(additional_search))

    results = []

    for img_meta in state.image_list:
        fname, _ = os.path.splitext(img_meta.name.lower())
        score, matched = score_filename(fname, keywords)
        if score > 0:
            results.append({
                'path': img_meta.file_path,
                'score': score,
                'matched_keywords': matched,
                'image': img_meta  # Include original metadata if needed
            })

    results.sort(key=lambda x: x['score'], reverse=True)
    if results == []:
        results = [{'path': FALLBACK_IMAGE, 'score': 0, 'matched_keywords': ['no results fo
und']}]
    return results

def score_filename(filename, keywords):
    """
    Score how well a filename matches a set of keywords (Chinese or English).
    Prioritizes:
    - Exact phrase matches
    - Whole-word matches
    - Substring matches
    """

    filename = filename.lower()
    score = 0
    matched_keywords = set()

    # Split filename into chunks for word-level matching
    filename_words = set(re.findall(r'[\u4e00-\u9fff]+|\w+', filename))

    for kw in keywords:
        kw = kw.lower().strip()
        if not kw:
            continue
```

```
        # If whole phrase (e.g. "corn beef") appears in filename
        if kw in filename:
            score += len(kw) * 15
            matched_keywords.add(kw)
            continue

        # If keyword is a full word in filename
        if kw in filename_words:
            score += len(kw) * 12
            matched_keywords.add(kw)
            continue

        # If keyword is a substring of any word
        partial_match_found = False
        for word in filename_words:
            if kw in word:
                score += len(kw) * 3
                matched_keywords.add(kw)
                partial_match_found = True
                break

        # Optional: slight boost if prefix of a word
        if not partial_match_found:
            for word in filename_words:
                if word.startswith(kw[:3]):
                    score += 2
                    matched_keywords.add(kw)
                    break

    return score, matched_keywords



def render_price_to_image(price_text: str, box_size: tuple[int, int], fonts: dict) -> Image
.Image:
    box_w, box_h = box_size
    padding_sides = 6  # keep side padding as is

    image = Image.new("RGBA", box_size, (255, 255, 255, 0))
    draw = ImageDraw.Draw(image)

    font_big = fonts['big']
    font_super = fonts['super']
    font_unit = fonts['unit']
    font_prefix = fonts.get('prefix', font_unit)

    # Enhanced regex: prioritize multi_match first
    multi_match = re.match(r"^\s*(\d+)\s+for\s+\$?\s*(\d+(?:\.\d{1,2})?)\s*$", price_text,
re.IGNORECASE)
    unit_match = None
    if not multi_match:
        unit_match = re.match(r"^\$?\s*(\d+)?(?:\.(\d{2}))?\s*(?:/|\s)?\s*(\w+)?\s*$", pric
e_text)

    if multi_match:
        qty, price_str = multi_match.groups()
        if '.' in price_str:
            dollars, cents = price_str.split(".")
        else:
            dollars, cents = price_str, None

        prefix_text = f"{qty} for"
        dollar_text = dollars
        cents_text = f".{cents}" if cents else ""
        super_text = "$"

        prefix_bbox = draw.textbbox((0, 0), prefix_text, font=font_prefix)
```

```
    dollar_bbox = draw.textbbox((0, 0), dollar_text, font=font_big)
    cents_bbox = draw.textbbox((0, 0), ".00", font=font_super)
    super_bbox = draw.textbbox((0, 0), super_text, font=font_super)

    prefix_w = prefix_bbox[2] - prefix_bbox[0]
    prefix_h = prefix_bbox[3] - prefix_bbox[1]
    dollar_w = dollar_bbox[2] - dollar_bbox[0]
    dollar_h = dollar_bbox[3] - dollar_bbox[1]
    cents_w = cents_bbox[2] - cents_bbox[0]
    cents_h = cents_bbox[3] - cents_bbox[1]
    super_w = super_bbox[2] - super_bbox[0]
    super_h = super_bbox[3] - super_bbox[1]

    vertical_spacing = 4
    content_height = prefix_h + vertical_spacing + max(dollar_h, super_h)

    y_offset = (box_h - content_height) // 2
    y_prefix = y_offset
    y_price = y_prefix + prefix_h + vertical_spacing

    x_prefix = (box_w - prefix_w) // 2
    draw.text((x_prefix, y_prefix), prefix_text, font=font_prefix, fill=(0, 0, 0, 255))

    total_price_width = super_w + dollar_w + (cents_w if cents else 0)
    x_price = (box_w - total_price_width) // 2
    x_super = x_price
    x_dollar = x_super + super_w
    x_cents = x_dollar + dollar_w

    y_super = y_price + (dollar_h - super_h)
    draw.text((x_super, y_super), super_text, font=font_super, fill=(0, 0, 0, 255))
    draw.text((x_dollar, y_price), dollar_text, font=font_big, fill=(0, 0, 0, 255))
    if cents:
        draw.text((x_cents, y_price), cents_text, font=font_super, fill=(0, 0, 0, 255))

elif unit_match:
    dollars, cents, unit = unit_match.groups()
    dollars = dollars or "0"
    unit = unit or ""
    has_cents = cents is not None
    cents_text = f".{cents}" if has_cents else ""
    dummy_cents = ".00"

    big_text = dollars
    dollar_sign = "$"
    sub_text = f"/{unit}" if unit else ""

    big_bbox = draw.textbbox((0, 0), big_text, font=font_big)
    dollar_bbox = draw.textbbox((0, 0), dollar_sign, font=font_super)
    real_cents_bbox = draw.textbbox((0, 0), cents_text, font=font_super)
    dummy_cents_bbox = draw.textbbox((0, 0), dummy_cents, font=font_super)
    sub_bbox = draw.textbbox((0, 0), sub_text, font=font_unit)

    big_w = big_bbox[2] - big_bbox[0]
    big_h = big_bbox[3] - big_bbox[1]
    dollar_w = dollar_bbox[2] - dollar_bbox[0]
    dollar_h = dollar_bbox[3] - dollar_bbox[1]
    cents_w = dummy_cents_bbox[2] - dummy_cents_bbox[0]
    cents_h = dummy_cents_bbox[3] - dummy_cents_bbox[1]
    sub_w = sub_bbox[2] - sub_bbox[0]
    sub_h = sub_bbox[3] - sub_bbox[1]

    total_price_width = dollar_w + big_w + cents_w
    x_start = (box_w - total_price_width) // 2

    x_dollar = x_start
    x_big = x_dollar + dollar_w
    x_cents = x_big + big_w
```

```
        top_row_height = max(big_h, dollar_h, cents_h)
        bottom_row_height = sub_h if sub_text else 0
        vertical_spacing = 4

        content_height = top_row_height + (vertical_spacing if bottom_row_height else 0) +
bottom_row_height
        y_offset = (box_h - content_height) // 2

        y_top = y_offset
        y_sub = y_top + top_row_height + vertical_spacing

        draw.text((x_dollar, y_top), dollar_sign, font=font_super, fill=(0, 0, 0, 255))
        draw.text((x_big, y_top), big_text, font=font_big, fill=(0, 0, 0, 255))
        if has_cents:
            draw.text((x_cents, y_top), cents_text, font=font_super, fill=(0, 0, 0, 255))

        if sub_text:
            x_sub = x_cents + cents_w - sub_w
            x_sub = min(x_sub, box_w - sub_w - padding_sides)
            x_sub = max(x_sub, padding_sides)
            draw.text((x_sub, y_sub), sub_text, font=font_unit, fill=(0, 0, 0, 255))

    else:
        fallback_bbox = draw.textbbox((0, 0), price_text, font=font_unit)
        fallback_w = fallback_bbox[2] - fallback_bbox[0]
        fallback_h = fallback_bbox[3] - fallback_bbox[1]
        x = (box_w - fallback_w) // 2
        y = (box_h - fallback_h) // 2
        draw.text((x, y), price_text, font=font_unit, fill=(0, 0, 0, 255))

    return image

def render_stacked_text(chinese_text, english_text, font_size):
    # Scale fonts up
    font_cn = ImageFont.truetype(FONT_PATH_CN, font_size * scale)
    font_en = ImageFont.truetype(FONT_PATH_EN, font_size * scale-scale)

    dummy_img = Image.new("RGBA", (1, 1))
    draw = ImageDraw.Draw(dummy_img)

    bbox_cn = draw.textbbox((0, 0), chinese_text, font=font_cn)
    bbox_en = draw.textbbox((0, 0), english_text, font=font_en)

    w_cn = bbox_cn[2] - bbox_cn[0]
    h_cn = bbox_cn[3] - bbox_cn[1]
    w_en = bbox_en[2] - bbox_en[0]
    h_en = bbox_en[3] - bbox_en[1]

    total_width = max(w_cn, w_en)
    total_height = h_cn + h_en

    # Create hi-res canvas
    img = Image.new("RGBA", (total_width, total_height), (0, 0, 0, 0))
    draw = ImageDraw.Draw(img)

    # Calculate centered positions
    x_cn = (total_width - w_cn) // 2 - bbox_cn[0]
    y_cn = -bbox_cn[1]

    x_en = (total_width - w_en) // 2 - bbox_en[0]
    y_en = h_cn - bbox_en[1]

    # Draw text
    draw.text((x_cn, y_cn), chinese_text, font=font_cn, fill=(0, 0, 0, 255))
    draw.text((x_en, y_en), english_text, font=font_en, fill=(0, 0, 0, 255))

    # Downscale for smoothness
    final_img = img.resize(
        (total_width // scale, total_height // scale),
```

```
        resample=Image.LANCZOS
    )
    return final_img


def center_text_on_canvas(text_img, width, height):
    # Create transparent canvas
    canvas = Image.new("RGBA", (width, height), (255, 255, 255, 0))

    # Get position to paste (centered)
    x = (width - text_img.width) // 2
    y = (height - text_img.height) // 2

    # Paste text image with alpha
    canvas.paste(text_img, (x, y), text_img)

    return canvas
```

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
init logic code
#weekly graphic
"""
```

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
#weekly sale
#core/image_loader
"""

from PIL import Image, ImageTk
import os

# Two caches:
_thumbnail_cache = {}  # Key: (abs_path, size)
_fullsize_cache = {}   # Key: abs_path

def load_image(path: str, size: tuple[int, int] = None, as_tk: bool = True):
    """
    Loads an image from disk. Resizes and converts to ImageTk if needed.

    Args:
        path (str): Path to image.
        size (tuple[int, int], optional): Resize image to this size (width, height).
        as_tk (bool): If True, return ImageTk.PhotoImage. Otherwise, return PIL.Image.

    Returns:
        ImageTk.PhotoImage or PIL.Image or None if failed.
    """
    abs_path = os.path.abspath(path)

    if size:
        key = (abs_path, size)
        if key in _thumbnail_cache:
            return _thumbnail_cache[key]
    else:
        if abs_path in _fullsize_cache:
            return _fullsize_cache[abs_path]

    try:
        with Image.open(abs_path) as img:
            img = img.convert("RGBA")
            if size:
                img = img.resize(size, Image.LANCZOS)
            if as_tk:
                img_tk = ImageTk.PhotoImage(img)
                if size:
                    _thumbnail_cache[key] = img_tk
                return img_tk
            else:
                # Return copy to keep it usable after `with`
                img_copy = img.copy()
                if not size:
                    _fullsize_cache[abs_path] = img_copy
                return img_copy

    except Exception as e:
        print(f"[ImageLoader] â\235\214 Failed to load image '{path}': {e}")
        return None

def clear_cache():
    _thumbnail_cache.clear()
    _fullsize_cache.clear()
```

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
class Collage
"""

import os
from app_config import {
    BASE_DIR, ASSETS_DIR, FONT_PATH_BOLD, FONT_PATH_CN, FONT_PATH_EN, FILLER_PATH, PRICEBOX
_PATH
}
from PIL import Image, ImageFont, ImageDraw
from models.item import Item
from core.image_loader import load_image

COLUMNS = 3

SQ_SIZE = (2160,2160)

class Collage2:
    def __init__(self,name,items_list,start_date=None,end_date=None):
        self.name = name
        self.items_list = items_list
        self.start = start_date
        self.end = end_date
        self.graphic = None

    def find_longest_names(self):
        longest_name = ''
        longest_chinese_name = ''

        for item in self.items_list.values():
            print(item)
            if len(item.name) > len(longest_name):
                longest_name = item.name
            if len(item.chinese_name) > len(longest_chinese_name):
                longest_chinese_name = item.chinese_name
        self.longest,self.longest_c = longest_name, longest_chinese_name

    #call text_size
    def text_size(self):
        # Get longest names
        self.find_longest_names()
        longest_en = self.longest
        longest_cn = self.longest_c
        #print(f'here i am and the longest_en word is: {longest_en}, {longest_cn}')
        #print(f'aqui is the rect wxh: {self.rectw} {self.recth}')
        ### HERE WE'RE SETTING HOW LARGE THE TEXT FOR THE NAMES' OF ITEMS WILL BE
        box_width = int(mmtopix(self.rectw))
        box_height = int(mmtopix(self.recth))

        max_font_size = 100  # Try from large to small

        for size in range(max_font_size, 5, -1):
            font_cn = ImageFont.truetype(FONT_PATH_CN, size)
            font_en = ImageFont.truetype(FONT_PATH_EN, size)

            img = Image.new("RGB", (box_width, box_height), "white")
            draw = ImageDraw.Draw(img)

            # Use textbbox to get dimensions
            bbox_cn = draw.textbbox((0, 0), longest_cn, font=font_cn)
            w_cn = bbox_cn[2] - bbox_cn[0]
            h_cn = bbox_cn[3] - bbox_cn[1]

            bbox_en = draw.textbbox((0, 0), longest_en, font=font_en)
            w_en = bbox_en[2] - bbox_en[0]
            h_en = bbox_en[3] - bbox_en[1]
```

```
        total_height = h_cn + h_en
        max_width = max(w_cn, w_en)
        #print(f'total_height= {total_height}, total_width = {max_width}')
        #print(f'lets see if max font size for loop runs {size}')
        if total_height <= box_height and max_width <= box_width:
            set2= size  # Save the first fitting font size
            break  # Done! We found the largest one that fits
    else:
        set2= 10  # Fallback if none fit
    self.max_text_size = set2
    print(f"â\234\205 Max font size that fits: {self.max_text_size}")

def generate_graphics()
```

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sat Jun 21 00:44:10 2025

#weekly sale
"""
from app_config import IMAGES_DIR, ASSETS_DIR, IMG_EXTS, FALLBACK_IMAGE
import os
from core.image_utils import find_matching_images  # your utility function
from PIL import Image

import warnings
warnings.simplefilter('always')
FOLDER_PATH = IMAGES_DIR  # used for quick validation

import os

class Item:
    def __init__(self, name, chinese_name, price, image_hint):
        self.name = name
        self.chinese_name = str(chinese_name)
        self.price = str(price)
        self.image_hint = str(image_hint)  # Formerly imagepath
        self.selected_image_path = None
        self.possible_images = []

    def __repr__(self):
        return f'item: {self.name} at {self.price}'

    def select_image(self, state):
        if self.image_hint.lower() == "blank":
            return [{'path': FALLBACK_IMAGE, 'score': 100, 'matched_keywords': ['blank']}]

        elif not self.image_hint:
            return find_matching_images(self.name, self.chinese_name, state)

        elif self.image_hint.lower().endswith(IMG_EXTS):
            full_path = os.path.join(FOLDER_PATH, self.image_hint)
            if os.path.exists(full_path):
                return [{'path': full_path, 'score': 100, 'matched_keywords': ['direct matc
h']}]
            else:
                return find_matching_images(self.name, self.chinese_name, state)

        else:
            return find_matching_images(self.name, self.chinese_name, state, additional_sea
rch=self.image_hint)




    def set_selected_image(self, path):
        """
        Assigns selected_image_path safely.
        Falls back to blank if path is invalid or None.
        """
        if path and os.path.isfile(path):
            self.selected_image_path = path
            print(f"â\234\205 Image set: {path}")
        else:
            fallback = os.path.join(ASSETS_DIR, FALLBACK_IMAGE)
            self.selected_image_path = fallback
            print(f"â\232 ï¸\217 Invalid path. Using fallback: {fallback}")

    def foodpic(self):
        """
        Return a PIL image (500x500 transparent square),
        centered and resized.
```

```
        """
        path = self.selected_image_path or os.path.join(FOLDER_PATH, FALLBACK_IMAGE)

        try:
            base_img = Image.open(path)
        except IOError:
            print("â\232 ï¸\217 Failed to open image. Using fallback.")
            base_img = Image.open(os.path.join(FOLDER_PATH, FALLBACK_IMAGE))

        square_w = 1000
        canvas = Image.new('RGBA', (square_w, square_w), (255, 255, 255, 255))

        w, h = base_img.size
        ratio = min(square_w / w, square_w / h)
        new_size = (int(w * ratio), int(h * ratio))
        resized = base_img.resize(new_size, resample=Image.BILINEAR)

        paste_x = (square_w - new_size[0]) // 2
        paste_y = (square_w - new_size[1]) // 2
        canvas.paste(resized, (paste_x, paste_y), resized if resized.mode == 'RGBA' else No
ne)

        return canvas
```

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
class Collage
"""

#images in assets folder
import os
from app_config import BASE_DIR, ASSETS_DIR, HEADER_PATH, BHEADER_PATH, FONT_PATH_CN, FONT_
PATH_EN, FONT_PATH_BOLD,FILLER_PATH, PRICEBOX_PATH
from PIL import Image, ImageFont, ImageDraw
from models.item import Item
from core.image_loader import load_image
from core.useful_funcs import mmtopix, pixtomm
from core.image_utils import render_price_to_image, render_stacked_text, center_text_on_can
vas, scale
from fpdf import FPDF, Align
import math
from io import BytesIO
import warnings
warnings.simplefilter('always')

COLUMNS_NO=2

A4_SIZE_MM = (210, 297)          # A4: 210mm x 297mm
LEGAL_SIZE_MM = (215.9, 355.6)   # Legal: 8.5" x 14" converted to mm
TABLOID_SIZE_MM = (279.4, 431.8) # Tabloid: 11" x 17" converted to mm
LONGASS_SIZE = (150,700)

pdfsize = LONGASS_SIZE
class Collage:
    def __init__(self,name,items_list,start_date=None,end_date=None):
        self.name= name
        self.items_list=items_list
        self.start = start_date
        self.end = end_date
        self.graphic = None


    def find_longest_names(self):
        longest_name = ''
        longest_chinese_name = ''

        for item in self.items_list.values():
            print(item)
            if len(item.name) > len(longest_name):
                longest_name = item.name
            if len(item.chinese_name) > len(longest_chinese_name):
                longest_chinese_name = item.chinese_name
        self.longest,self.longest_c = longest_name, longest_chinese_name


    def headers(self):
        if self.graphic is not None:
            pdfw, pdfh = self.graphic.epw, self.graphic.eph

            # --- Top Header ---
            if os.path.isfile(HEADER_PATH):
                with Image.open(HEADER_PATH) as header_img:
                    top = header_img.convert('RGBA').copy()

                w, h = top.size
                self.header_ratio = h / w
                header_h = pdfw * self.header_ratio

                if header_h <= pdfh / 2:
                    self.top = top
                    self.header_h = header_h
```

```
                print(f'header height = {self.header_h}')
            else:
                print(f'header too tall: {header_h}px > half of page height ({pdfh / 2}
px)')

        # --- Bottom Header ---
        if os.path.isfile(BHEADER_PATH):
            with Image.open(BHEADER_PATH) as bheader_img:
                bottom = bheader_img.convert('RGBA').copy()

            w, h = bottom.size
            self.bheader_ratio = h / w
            bheader_h = pdfw * self.bheader_ratio

            if bheader_h <= pdfh / 2:
                self.bottom = bottom
                self.bheader_h = bheader_h
                print(f'bottom header height = {self.bheader_h}')
            else:
                print(f'bottom header too tall: {bheader_h}px > half of page height ({p
dfh / 2}px)')




        # header = os.path.join(ASSETS_DIR,'headerpic.png')
        # image1 = Image.open(header)
        # bottomheader = os.path.join(ASSETS_DIR,'bottomlogo.png')
        # image2 = Image.open(bottomheader)
        # top_header = image1.resize((2168,760),resample=Image.LANCZOS)
        # bottom_header = image2.resize((800,200),resample=Image.LANCZOS)

        # self.header = top_header # file path goes here
        # self.bottom = bottom_header #file path goes here
        # self.collage_pgs = None # how many pgs will it need

    def generate_pdf(self):
        # check there is a selected image for each item
        #if {
               # }
        pdf = FPDF(unit='mm',format=pdfsize)
        pdf.set_margin(0)
        self.graphic=pdf

        self.headers()

        #calculate collage area
        collage_spaceh = pdf.eph-(self.header_h+self.bheader_h)
        self.collage_space = (pdf.epw,collage_spaceh)
        self.collage_cols = COLUMNS_NO #how many items per column
        self.rectw = pdf.epw/self.collage_cols
        self.max_rows = math.floor(collage_spaceh/self.rectw)
        self.recth = collage_spaceh/self.max_rows
        print(f'RECTW {self.rectw}, RECTH {self.recth}')

        #items pp
        print(f'look here \n max rows = {self.max_rows} and collage_cols={self.collage_cols
}')
        self.items_pp = self.collage_cols*self.max_rows
        self.pages = math.ceil(len(self.items_list)/self.items_pp)
        print(f'\n\n look here self pages = {self.pages} and self.items_pp={self.items_pp}'
)
        self.text_size()

    #call text_size
    def text_size(self):
        # Get longest names
        self.find_longest_names()
```

```
        longest_en = self.longest
        longest_cn = self.longest_c
        #print(f'here i am and the longest_en word is: {longest_en}, {longest_cn}')
        #print(f'aqui is the rect wxh: {self.rectw} {self.recth}')
        ### HERE WE'RE SETTING HOW LARGE THE TEXT FOR THE NAMES' OF ITEMS WILL BE
        box_width = int(mmtopix(self.rectw))
        box_height = int(mmtopix(self.recth))

        max_font_size = 100  # Try from large to small

        for size in range(max_font_size, 5, -1):
            font_cn = ImageFont.truetype(FONT_PATH_CN, size)
            font_en = ImageFont.truetype(FONT_PATH_EN, size)

            img = Image.new("RGB", (box_width, box_height), "white")
            draw = ImageDraw.Draw(img)

            # Use textbbox to get dimensions
            bbox_cn = draw.textbbox((0, 0), longest_cn, font=font_cn)
            w_cn = bbox_cn[2] - bbox_cn[0]
            h_cn = bbox_cn[3] - bbox_cn[1]

            bbox_en = draw.textbbox((0, 0), longest_en, font=font_en)
            w_en = bbox_en[2] - bbox_en[0]
            h_en = bbox_en[3] - bbox_en[1]

            total_height = h_cn + h_en
            max_width = max(w_cn, w_en)
            #print(f'total_height= {total_height}, total_width = {max_width}')
            #print(f'lets see if max font size for loop runs {size}')
            if total_height <= box_height and max_width <= box_width:
                set2= size  # Save the first fitting font size
                break  # Done! We found the largest one that fits
        else:
            set2= 10  # Fallback if none fit
        self.max_text_size = set2
        print(f"â\234\205 Max font size that fits: {self.max_text_size}")

    def makepages(self):
        pdf = self.graphic
        print('hi im at makepages')
        items = self.items_list
        header = self.top
        bottomheader = self.bottom
        bheadh_mm = self.bheader_h
        print(self.pages)

        for i in range(self.pages):
            print(f'i={i},itemspp={self.items_pp}')
            pdf.add_page()
            pdf.image(header, x=0, y=0, w=pdf.epw)
            pdf.image(bottomheader, x=0, y=pdf.eph - bheadh_mm, w=pdf.epw)

            for j in range(i * self.items_pp, (i + 1) * self.items_pp):
                local_index = j % self.items_pp
                xcol = (local_index % self.collage_cols) * self.rectw
                yrow = (local_index // self.collage_cols) * self.recth + self.header_h

                frame = Image.new('RGBA', (mmtopix(self.rectw), mmtopix(self.recth)), (250,
20, 140, 100))

                if j < len(items):
                    item = items[j]
                    img = item.foodpic().convert("RGBA")
                    img = img.resize((mmtopix(self.rectw), mmtopix(self.rectw)))
                    frame.paste(img, (0, 0))

                    # --- PRICE BOX ---
                    with Image.open(PRICEBOX_PATH) as pb_img:
```

```
                        pricebox = pb_img.convert('RGBA').copy()

                    resized_pricebox = pricebox.resize(
                        (int(mmtopix(self.rectw) / 2), int(mmtopix(self.recth) / 3)),
                        resample=Image.LANCZOS
                    ).copy()

                    base_font_scale = resized_pricebox.height / 3
                    min_size = 10
                    big_size = max(int(base_font_scale * 1.2), min_size)
                    super_size = max(int(base_font_scale * 0.72), min_size)
                    unit_size = max(int(base_font_scale * 0.5), min_size)
                    prefix_size = max(int(base_font_scale * 0.72), min_size)

                    fonts = {
                        'big': ImageFont.truetype(FONT_PATH_BOLD, size=big_size),
                        'super': ImageFont.truetype(FONT_PATH_EN, size=super_size),
                        'unit': ImageFont.truetype(FONT_PATH_EN, size=unit_size),
                        'prefix': ImageFont.truetype(FONT_PATH_EN, size=prefix_size),
                    }

                    scale_factor = 4
                    large_box_size = (
                        resized_pricebox.width * scale_factor,
                        resized_pricebox.height * scale_factor
                    )
                    fonts_large = {
                        k: ImageFont.truetype(f.path, size=f.size * scale_factor)
                        for k, f in fonts.items()
                    }

                    hires_price_image = render_price_to_image(
                        price_text=item.price.strip(),
                        box_size=large_box_size,
                        fonts=fonts_large
                    )

                    raw_price_image = hires_price_image.resize(
                        (int(resized_pricebox.width * 0.8), int(resized_pricebox.height * 0
.8)),
                        resample=Image.LANCZOS
                    )
                    offset_x = (resized_pricebox.width - raw_price_image.width) // 2
                    offset_y = (resized_pricebox.height - raw_price_image.height) // 2
                    resized_pricebox.paste(raw_price_image, (offset_x, offset_y), raw_price
_image)

                    # Paste pricebox first
                    pricebox_x = frame.width - resized_pricebox.width - 1
                    pricebox_y = frame.height - resized_pricebox.height
                    frame.paste(resized_pricebox, (pricebox_x, pricebox_y), resized_pricebo
x)

                    # --- STACKED TEXT ---
                    stacked_text_img = render_stacked_text(item.chinese_name, item.name, fo
nt_size=self.max_text_size)
                    centered_text = center_text_on_canvas(
                        stacked_text_img,
                        int(mmtopix(self.rectw) / 2),
                        int(mmtopix(self.rectw) / 3)
                    )
                    text_y = frame.height - centered_text.height
                    text_y = max(0, text_y)  # avoid negative placement
                    frame.paste(centered_text, (0, text_y), centered_text)

                else:
                    with Image.open(FILLER_PATH) as filler_img:
                        img = filler_img.convert('RGBA').resize((frame.width, frame.width))
                        frame.paste(img, (0, 0))
```

```
                buffer = BytesIO()
                frame.convert("RGB").save(buffer, format="PNG")
                buffer.seek(0)
                pdf.image(buffer, w=self.rectw, h=self.recth, x=xcol, y=yrow)
```

```
    def __str__(self):
        return f'new collage instance made: {self.name}, start { self.start}, end {self.end
}'
```