Junk Junction

By: Anthony Langer, Ian Flack, Sarkis Nazaryan, Leonel Villanueva

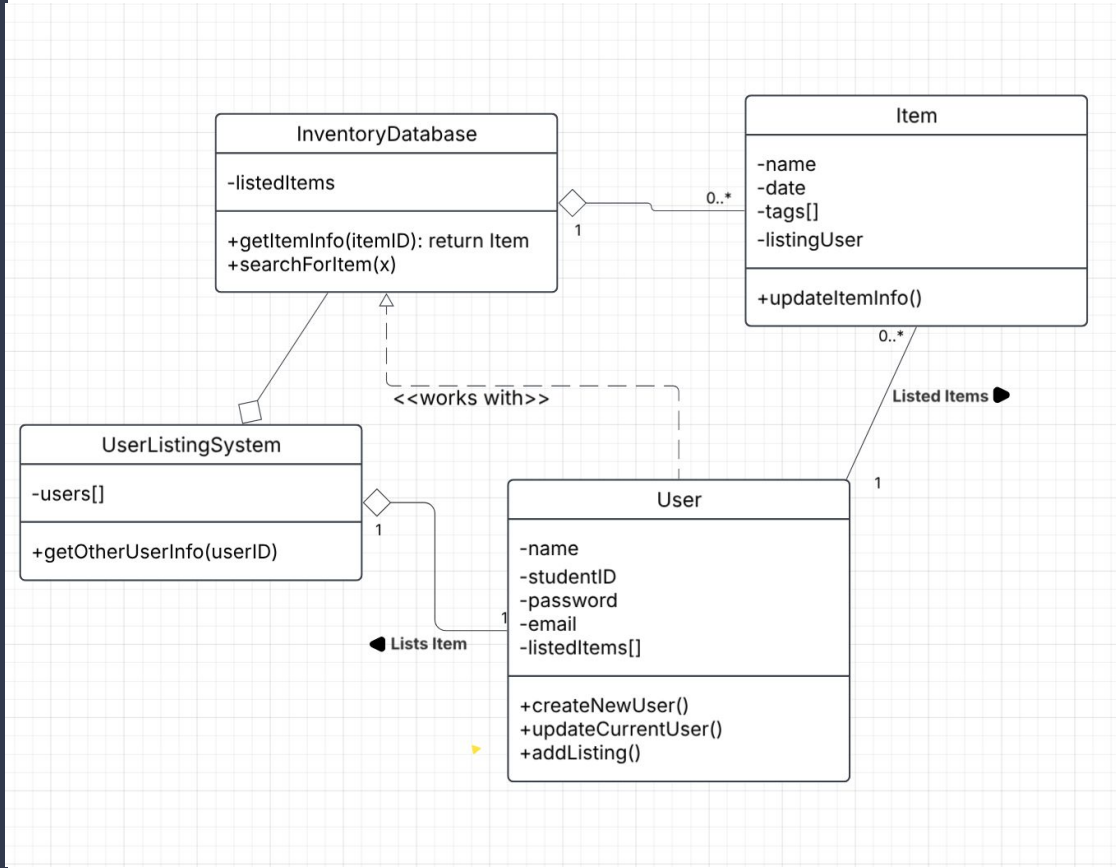
Motivation

- Create an application to connect students within the university to trade or sell items to each other.
- Encourages students to engage with each other in public discourse.
- Allows students to declutter their living spaces by listing items they don’t need
- Fosters community.

Original UML

How it changed

- UserListingSystem just became simply a UserDatabase manager.
- Both databases are now used only by UserCurrent controller. Every action is done through the logged in user after all.
- This UML is missing the boundary App class. It creates and transfers control to the different pages.



Back End

InventoryDatabase:

- InventoryDatabase is one of the biggest back end files for our project.
  - Retrieves items from users
  - Creates items to store them inside of our database
  - Can change an item to either “sold” or “unlisted”

```python
    @staticmethod
    def create_new_item(itemName, itemDescription, itemCondition, itemCategory, itemPrice, itemImage):
        ''' Creates a new item and adds it to the item list.
        Takes all item name, description, condition, category, and price as input. Generates a default itemID
        and userID from the current user. Returns the created ItemInfo object.
        '''
        new_id = str(InventoryDatabase.curItemID + 1)
        newItem = ItemInfo(
            itemID=new_id,
            userID=UserCurrent.current_user.userID,
            itemName=itemName,
            itemDescription=itemDescription,
            itemCondition=itemCondition,
            itemCategory=itemCategory,
            itemPrice=itemPrice,
            itemStatus="Available",
            itemComments="ItemComments/comments" + new_id.zfill(4) + ".csv",
            itemImage=itemImage
        )
        InventoryDatabase.itemList.append(newItem)
        InventoryDatabase.update_csv()
        return newItem
```

```python
    @staticmethod
    def make_sold(item):
        ''' Marks an item as sold by updating its status and moving it to the log list.
        Takes an ItemInfo object as input, and returns nothing.
        '''
        for logItem in InventoryDatabase.logItemList:
            if logItem.itemID == str(item.itemID):
                return 0
        item.itemStatus = "Sold"
        InventoryDatabase.logItemList.append(item)
        try:
            InventoryDatabase.itemList.remove(item)
        except ValueError:
            pass
        return 1

    @staticmethod
    def make_unlisted(item):
        ''' Marks an item as unlisted by updating its status and moving it to the log list.
        Takes an ItemInfo object as input, and returns nothing.
        '''
        for logItem in InventoryDatabase.logItemList:
            if logItem.itemID == str(item.itemID):
                return 0
        item.itemStatus = "Unlisted"
        InventoryDatabase.logItemList.append(item)
        try:
            InventoryDatabase.itemList.remove(item)
        except ValueError:
            pass
        return 1
```

(variable) logItemList: list

Back End

UserCurrent:

- UserCurrent is another one of the biggest files for the back end and helps the front end greatly.
  - Checks if users exist with their email or password
    - Provides checks for the front end when users log into the sign in page
- Loads UserInfo object from the UserDatabase into it's own special user instance object for OOP.
  - It uses that instance of the current user to do operations using the attributes of the logged in user.

```
class UserCurrent:
    '''
    UserCurrent.py
    11/06/2025
    Anthony Langer, Ian Flack
    This class manages the current user session.
    It includes methods to check if a user exists by email and password.
    '''
    current_user = None  # Initialize as None instead of dummy user

    def __init__(self, userID, name, email, password):
        '''Initializes a UserCurrent object with user details.'''
        self.userID = userID
        self.name = name
        self.email = email
        self.password = password
```

```
@staticmethod
def check_if_user_exists(email, password):
    ''' Checks if a user exists with the given email and password.
    Returns a UserCurrent object if found, else returns None.
    '''
    for user in UserDatabase.userList:
        if user.email == email and user.password == password:
            return user
    return None

@staticmethod
def check_if_user_exists_by_email(email):
    '''Checks if there is a user with the given email.
    Returns a UserCurrent object if found, else returns None.
    '''
    for user in UserDatabase.userList:
        if user.email == email:
            return user
    return None
```

Front End

App

- App class is main controller for the pages in the application.
    - Creates geometry of other pages.
    - Container frame holds all the pages.
    - The show_frame is switches between the different pages.

```
def __init__(self):
    super().__init__()
    self.title("Junk Junction")
    self.geometry("230x130")

    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 (SignInPage, CreateUserPage, HomePage, UserPage):
        page_name = F.__name__
        frame = F(parent=container, controller=self)
        frame.grid(row=0, column=0, sticky="nsew")
        self.frames[page_name] = frame

    self.show_frame("SignInPage")

def show_frame(self, page_name):
    sizes = {
        "SignInPage": "280x150",
        "HomePage": "800x600",
        "UserPage": "800x600",
        "CreateUserPage": "300x230"
        #TODO: adjust for ImagePopup
    }
    geom = sizes.get(page_name)
    if geom:
        self.geometry(geom)
    frame = self.frames[page_name]

    if page_name == "UserPage":
        frame.refresh_items()

    frame.tkraise()
```

Front End

HomeImagePopup

- HomeImagePopup class assists in the functionality of the HomePage class.
  - Creates a page on top of application that displays an enlarged item image as well as a description.
  - Displays a comment section that keeps track of user comments on all individual items.
- It's within the composition of HomePage.

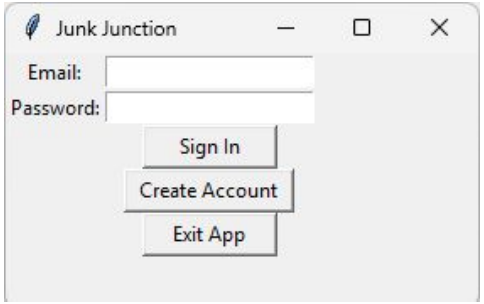
```
    def __init__(self, parent, image_path, caption, description, username="Guest"):
        super().__init__(parent)
        self.title("Listing Description")

        self.csv_path = "0001.csv"
        self.username= username

        # Display Larger Image
        try:
            photo = tk.PhotoImage(file=image_path)
            # Resize image by subsampling (scale down by factor of 2 if too large)
            if photo.width() > 300 or photo.height() > 200:
                scale_x = max(1, photo.width() // 300)
                scale_y = max(1, photo.height() // 200)
                scale = max(scale_x, scale_y)
                photo = photo.subsample(scale, scale)
        except tk.TclError as e:
            print(f"Image load error for '{image_path}': {e}")
            # create a gray placeholder with text
            photo = tk.PhotoImage(width=200, height=150)
            photo.put("gray", to=(0, 0, 200, 150))

        img_label = ttk.Label(self, image=photo)
        img_label.image = photo  # keep reference
        img_label.grid(row=0, column=0, sticky="n", pady=(0,5))

        #comment section
        comment_frame = ttk.Frame(self)
        comment_frame.grid(row=0, column=1, sticky="n", padx=20, pady=10)
        tk.Label(comment_frame, text="Comments", font=("Times New Roman",12)).grid(row= 0, column= 0, sticky= "w")

        self.comment_display = tk.Text(comment_frame, width = 40, height = 10, state= "disabled", wrap= "word")
        self.comment_display.grid(row= 1, column= 0, pady= (5,10))

        #comment text box
        scrollbar = ttk.Scrollbar(comment_frame, command=self.comment_display.yview)
        scrollbar.grid(row=1, column=1, sticky="ns")
        self.comment_display.config(yscrollcommand=scrollbar.set)

        self.comment_entry= tk.Text(comment_frame, width= 40, height= 3)
        self.comment_entry.grid(row= 2, column= 0, pady= 5, sticky='w')
```

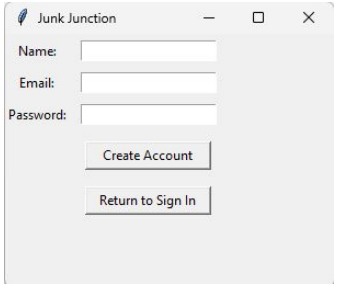
```
    img_label.bind("<Button>", lambda e, p = image_path, c = caption, d = description : HomeImagePopup(self, p, c, d, ))

    text_label = ttk.Label(frame, text=caption, font=("Times New Roman",10), wraplength=200)
    text_label.grid(row=1, column=0, sticky="s")
```

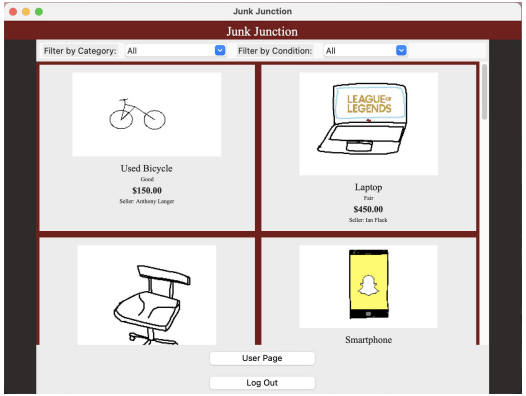
GUI

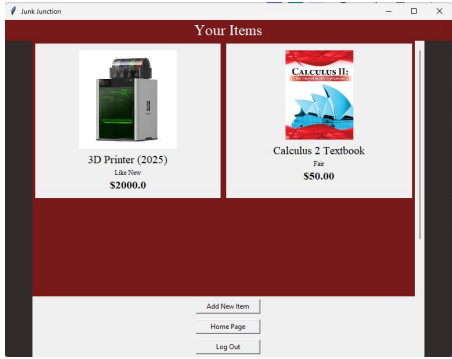
Using tKinter

- Simple Login only requiring email address and password.
- Create account screen asking for Username, Email, and password.
    - Checks Emails for duplicates
- Homepage displays items in database along with simple filter feature.
- Userpage displays personal items of logged in user.



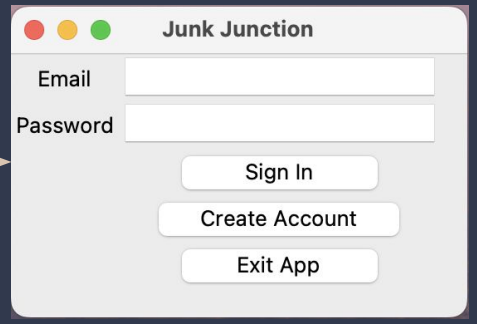


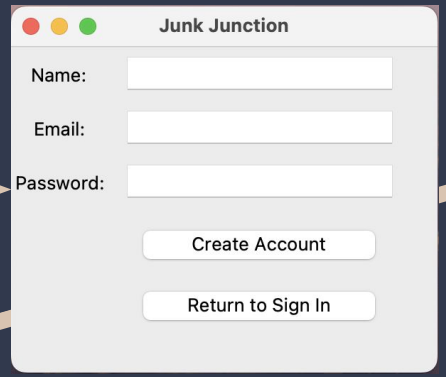




Functionality

- The program requires a user to either sign in or create an account with their email and password.
- The create account page requires the user to input their name, email, and password to log in.

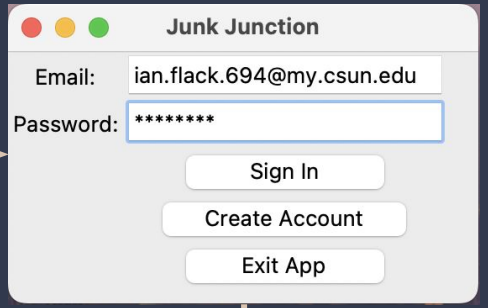


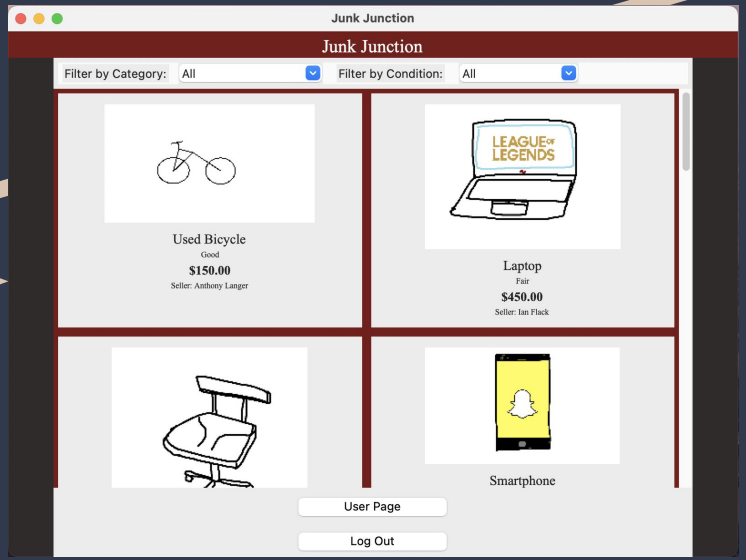


- *Once the user enters their information, they do not have to exit the application for their credentials to refresh and automatically return to sign-in page.

Functionality pt.2

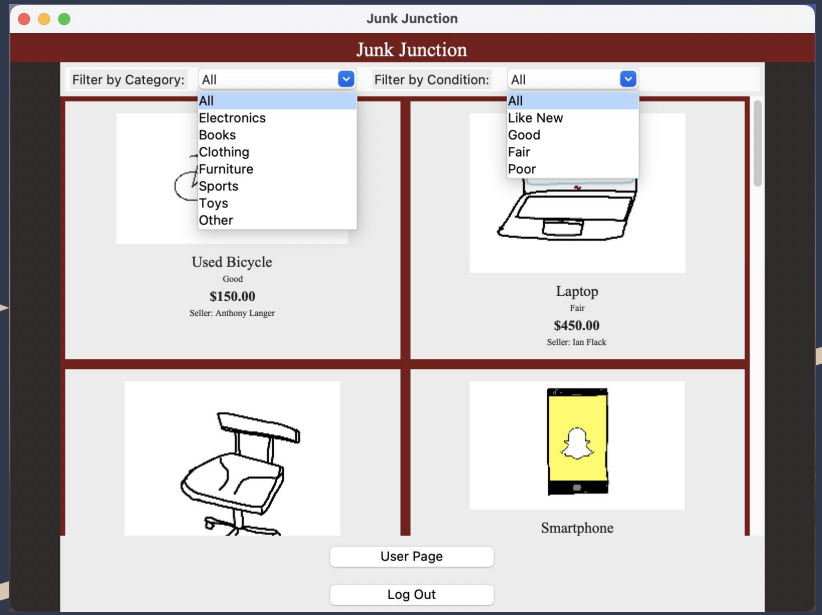
- Once the proper credentials are entered, the user is welcomed into the program
- After successfully logging in, the user is met with the homepage of the application

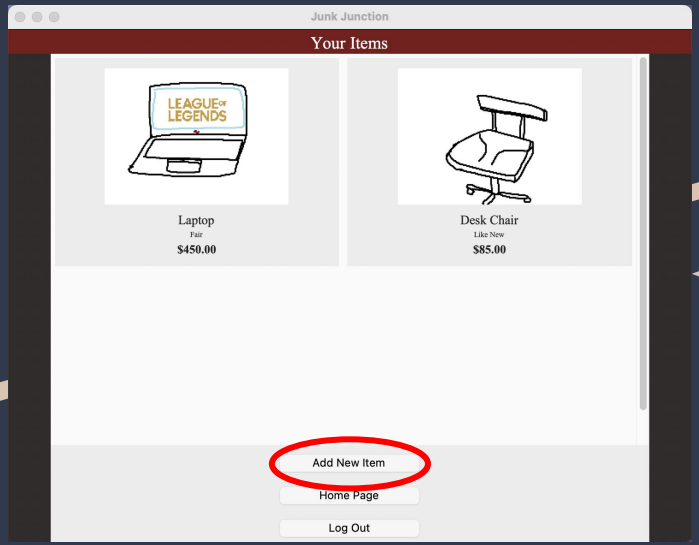




Functionality pt.3

- The user has the ability to filter the marketplace by Category of the item and Condition of the item

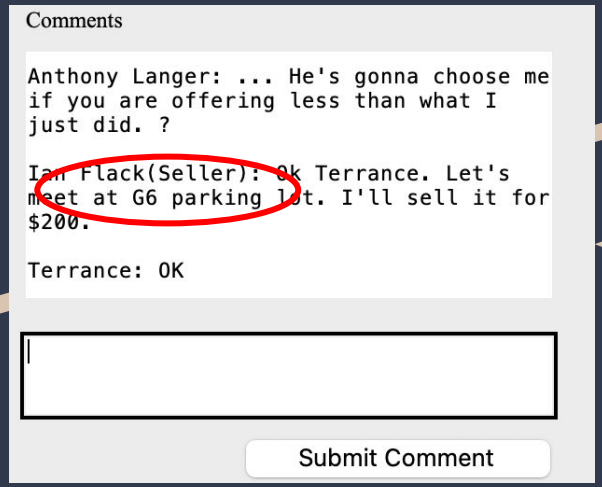




- As well as browsing items, the user can also create items that will be listed on the userpage via the “Add New Item” button

Functionality pt.4

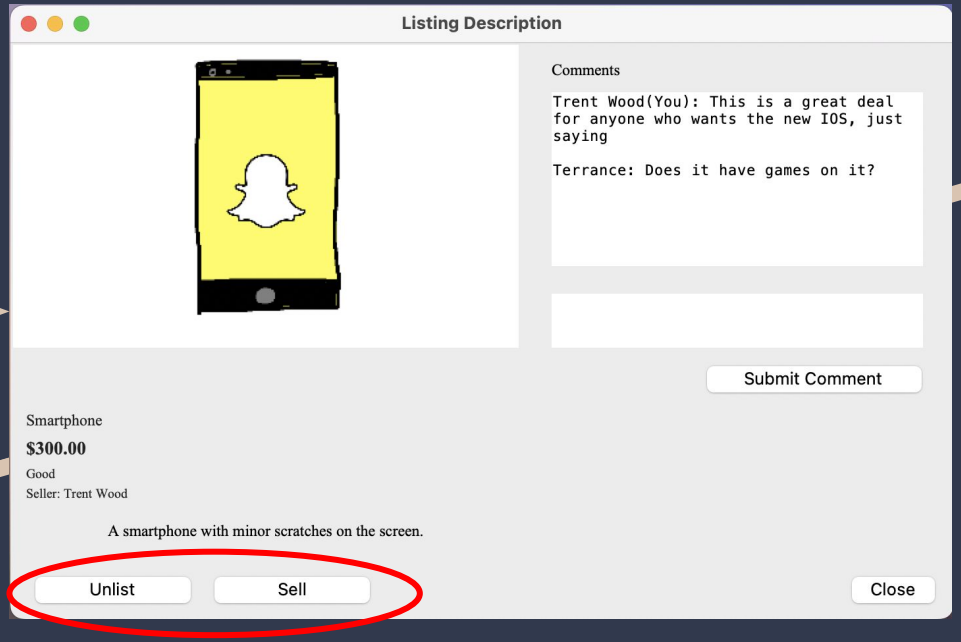
- When a user clicks on an item from the homepage, they are met with a pop-up of the item displaying its contents and the comment section



- The comment section features the recognition of the seller in the chat by “(Seller)” which informs the other users who they are chatting with

Functionality pt.5

- When accessing the user item page, the user can click their items and find the buttons named “Unlist” and “Sell” to finalize a sale or remove their item from the homepage



Design Patterns

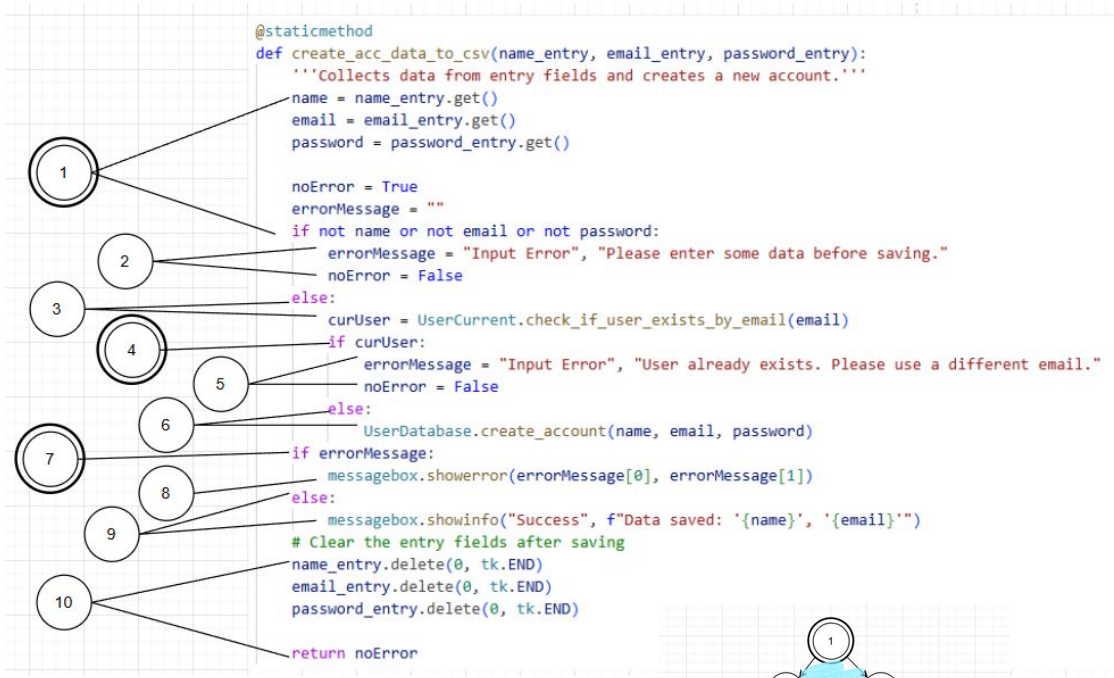
- Facade - Frontend pages use the application class as its foundation, initializing the main window, setting up the frame container, and introducing methods that switch between different pages.
- Factory - InventoryDatabase and UserDatabase files create Objects corresponding to their correct class that is instantiating them in order to file them into the CSV's
- Delegation - Sign in Page delegates creating a new user to the Create User Page instead of it all being in one giant class.
- MVC- The view displays data pulled from the model (via a CSV). When the user clicks "Log In," the controller receives the input, updates the model using CSV methods, and the updated model data is then reflected back in the view.

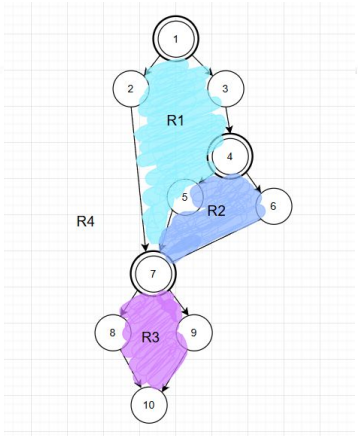
Testing

Flow Graph

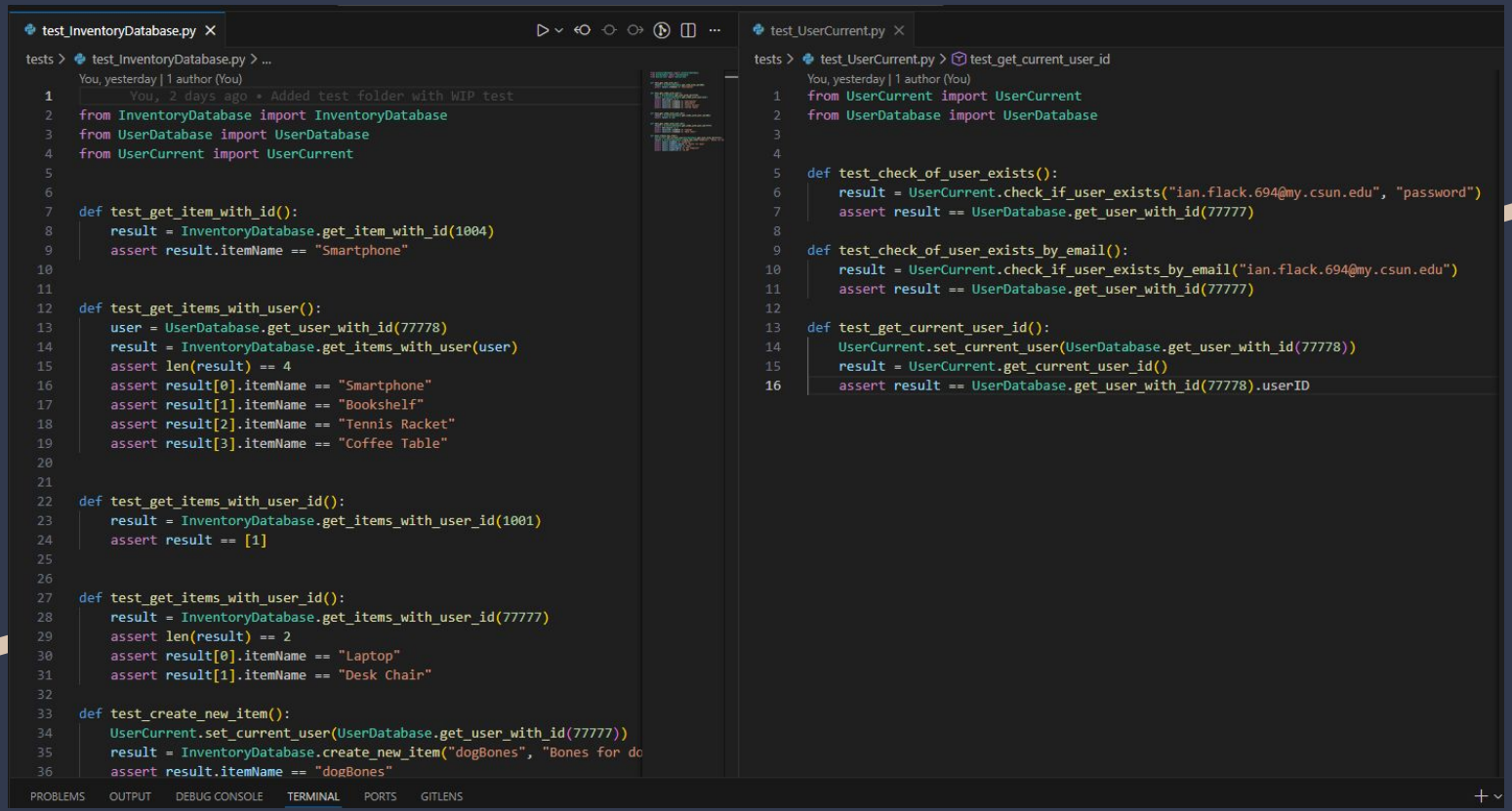
- create_acc_data_to_csv() is a method in the CreateUserPage scene. It validates the three passed in entry fields that contain the user input.
- Once it reaches Node 6, it calls a UserDatabase method that deals with creating a new UserInfo object and saving it.
- There are 4 independent paths of inputs that display an error to the user to fix.

```python
@staticmethod
def create_acc_data_to_csv(name_entry, email_entry, password_entry):
    '''Collects data from entry fields and creates a new account.'''
    name = name_entry.get()
    email = email_entry.get()
    password = password_entry.get()

    noError = True
    errorMessage = ""
    if not name or not email or not password:
        errorMessage = "Input Error", "Please enter some data before saving."
        noError = False
    else:
        curUser = UserCurrent.check_if_user_exists_by_email(email)
        if curUser:
            errorMessage = "Input Error", "User already exists. Please use a different email."
            noError = False
        else:
            UserDatabase.create_account(name, email, password)
    if errorMessage:
        messagebox.showerror(errorMessage[0], errorMessage[1])
    else:
        messagebox.showinfo("Success", f"Data saved: '{name}', '{email}'")
    # Clear the entry fields after saving
    name_entry.delete(0, tk.END)
    email_entry.delete(0, tk.END)
    password_entry.delete(0, tk.END)

    return noError
```





Test Cases



test_InventoryDatabase.py

tests > test_InventoryDatabase.py > ...

You, yesterday | 1 author (You)

```
1   You, 2 days ago • Added test folder with WIP test
2   from InventoryDatabase import InventoryDatabase
3   from UserDatabase import UserDatabase
4   from UserCurrent import UserCurrent
5
6
7   def test_get_item_with_id():
8       result = InventoryDatabase.get_item_with_id(1004)
9       assert result.itemName == "Smartphone"
10
11
12  def test_get_items_with_user():
13      user = UserDatabase.get_user_with_id(77778)
14      result = InventoryDatabase.get_items_with_user(user)
15      assert len(result) == 4
16      assert result[0].itemName == "Smartphone"
17      assert result[1].itemName == "Bookshelf"
18      assert result[2].itemName == "Tennis Racket"
19      assert result[3].itemName == "Coffee Table"
20
21
22  def test_get_items_with_user_id():
23      result = InventoryDatabase.get_items_with_user_id(1001)
24      assert result == [1]
25
26
27  def test_get_items_with_user_id():
28      result = InventoryDatabase.get_items_with_user_id(77777)
29      assert len(result) == 2
30      assert result[0].itemName == "Laptop"
31      assert result[1].itemName == "Desk Chair"
32
33  def test_create_new_item():
34      UserCurrent.set_current_user(UserDatabase.get_user_with_id(77777))
35      result = InventoryDatabase.create_new_item("dogBones", "Bones for do
36      assert result.itemName == "dogBones"
```

test_UserCurrent.py

tests > test_UserCurrent.py > test_get_current_user_id

You, yesterday | 1 author (You)

```
1   from UserCurrent import UserCurrent
2   from UserDatabase import UserDatabase
3
4
5   def test_check_of_user_exists():
6       result = UserCurrent.check_if_user_exists("ian.flack.694@my.csun.edu", "password")
7       assert result == UserDatabase.get_user_with_id(77777)
8
9   def test_check_of_user_exists_by_email():
10      result = UserCurrent.check_if_user_exists_by_email("ian.flack.694@my.csun.edu")
11      assert result == UserDatabase.get_user_with_id(77777)
12
13  def test_get_current_user_id():
14      UserCurrent.set_current_user(UserDatabase.get_user_with_id(77778))
15      result = UserCurrent.get_current_user_id()
16      assert result == UserDatabase.get_user_with_id(77778).userID
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS

The End