Ex. No.	Date	Title	Teacher Sign
1.	15/02/2025	Design a UI where users recall visual elements. Evaluate the effects of chunking on user memory	
2.	22/02/2025	Develop and compare CLI, GUI and VUI for same task and assess user satisfaction.	
3a.	01/03/2025	create a prototype with familiar and unfamiliar navigation elements. Evaluate ease of use with different user groups.	
4a	21/03/2025	conduct task analysis for an app and document user flows - Lucidchart	
4b	05/04/2025	conduct task analysis for an app and document user flows - Dia.	
3b	07/04/2025	create a prototype with familiar & unfamiliar navigation elements. Evaluate ease of use with different user groups.	
5a	12/04/2025	Simulate the lifecycle stages for UI design using the RAD model and develop a small interactive interface	
5b	12/04/2025	simulate the lifecycle stages for UI design using the RAD model and develop a small interactive interface.	
6	12/04/2025	Experiment with different layouts and colour schemes for an app. collect user feedback on aesthetic and usability.	
7a	19/04/2025	Develop low-fidelity paper prototypes for a banking app and convert them into digital wireflow.	
7b.	19/04/2025	develop low-fidelity paper prototype for a banking app and convert them into digital wireflow	
8a.	19/04/2025	create storyboards to represent the user flow for mobile app.	
8b.	26/04/2025	create storyboards to represent the user flow for mobile app.	
9.	26/04/2025	Design input form that validate data and display error message.	
10.	26/04/2025	create a data visualization for an inventory management system.	

Ex. 1 25/2/25

Design a UI where user recall visual element (Eg: icon / text / chunks). Evaluate the effect of chunking on user memory.

Aim:

The aim of this UI design is to investigate how chunking influence user ability to recall visual element such as icons by comparing recall performance between chunked and non-chunked presentation.

Procedure:

Step-1: set up your workspace.

⇒ open figma: Either go to figma's website or open figma desktop app.

⇒ Create a New file: click on 'New File' button to create new project.

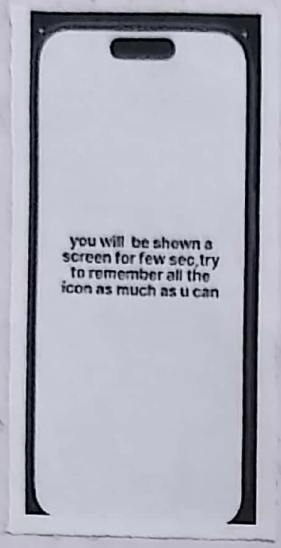
Step-2: Create Frames.

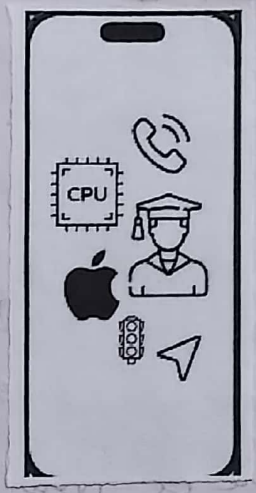
⇒ Add Frame: on left toolbar, select the frame tool (F). Add several frames as you will need multiple screens to test chunking.

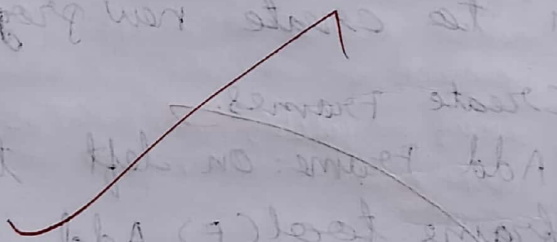
⇒ Name Frames: Name these frames for ease of reference. Eg: 'chunked Icon', 'Recall Screen1', 'Random icon', 'Recall screen2'.

Step-3: Design Icon/Images and text.

⇒ Insert Icons: User figma's assets







frames. You can use plugins like icons. group relation icon for chunked icons place icon randomly for random icon.

=> Add text chunks: Use the 'text tool' (T) to type chunks of text if you're testing text recall.

Step-4: Instruction Screen Design

=> Create Instruction Screen: Design the first frame to provide users with instruction on what they need to do.

Eg: "You will see some icon for few sec".

Step-5: Transition Design.

=> Add timed Interactions:

* Select prototype tab.

* Link chunked icons frame to recall screen set the interaction to transition after few seconds.

* Repeat for random icon frame transitioning to second recall screen.

=> set Delay Time: Adjust the delay time ensure users have enough time to view the chunks but not too much time.

=> Use Smart Animate: Use figma's 'Smart Animate' to create smooth transition.

Creating prototype in figma:

* select the frame you want to transition from.

* Click on prototype link icon.

* Drop the arrow to the frame you want to

transition to.

* Set the instruction to 'After Delay' and specify the duration.

* Choose the animation type like 'Smart Animation' for smooth transition.

⇒ Step-6: User Testing.

* Conduct user Testing: Recruit users and have them go through the test sequence.

* Record data: Note down the response time for each user during the recall phase. You can use spreed sheet/ simple notepad to track this data.

⇒ Step-7: Analyze Results.

* Compare Results: Evaluate which chunking method resulted in better recall accuracy.

* Document Findings: Use figma to add notes / comments on your findings directly on the frames if needed.

Result:

The above task has been executed successfully.

Exc. No. 2 22/2/25

Develop and Compare CLI, GUI and VUI for the same test and assess user satisfaction using python, Terminal

Aim:

The aim is to develop and compare command Line Interface (CLI), Graphical user Interface (GUI) and Voice user Interface (VUI) for the same task and assess user satisfaction using python or terminal.

Procedure:

i) Command Link Interface (CLI)

CLI implementation where user can add view and remove tasks using the terminal.

program:

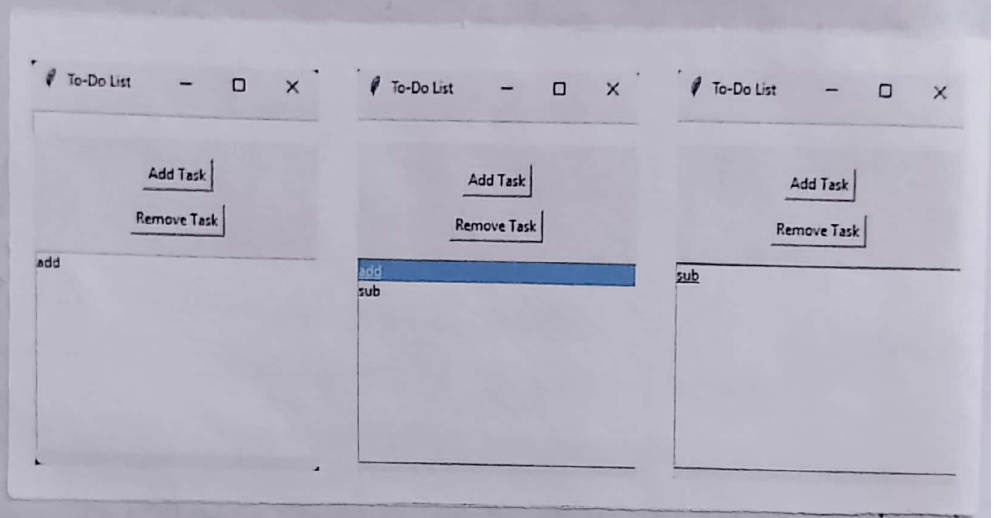
```
tasks = [ ]
def add_task (task):
    tasks.append (task)
    print (f "Task 'task' added.")
def view_tasks():
    if tasks:
        print ("your tasks:")
        for idx, task in enumerate (tasks):
            print (f "{idx}. {task}")
    else:
        print ("No tasks to show.")
def remove_task (task_number):
    if 0 < task_number <= len (tasks):
        remove_task = task.pop (task_number - 1)
        Print ("f "Task '{removed_task}' removed")
```

```
kamal\OneDrive\Desktop\program\UI types\cli.py'

Options:
1.Add Task
2.View Tasks
3.Remove Task
4.Exit
Enter your choice: 1
Enter task: add
Task 'add' added.

Options:
1.Add Task
2.View Tasks
3.Remove Task
4.Exit
Enter your choice: 1
Enter task: sub
Task 'sub' added.

Options:
1.Add Task
2.View Tasks
3.Remove Task
4.Exit
Enter your choice: 2
Your tasks:
1. add
2. sub

Options:
1.Add Task
2.View Tasks
3.Remove Task
4.Exit
Enter your choice: 3
Enter task number to remove: 2
Task 'sub' removed.

Options:
1.Add Task
2.View Tasks
3.Remove Task
4.Exit
Enter your choice: 3
Enter task number to remove: 1
Task 'add' removed.

Options:
1.Add Task
2.View Tasks
3.Remove Task
4.Exit
Enter your choice: 3
Enter task number to remove: 0
Invalid task number.

Options:
1.Add Task
2.View Tasks
3.Remove Task
4.Exit
Enter your choice: 4
```

```
        else:
            print("Invalid task number!")
def main():
    while True:
        print("\n Options:\n 1. Add task
              \n 2. View task\n 3. Remove
              task\n 4. Exit")
        c = input("Enter Your choices:")
        if c == '1':
            task = input("Enter task:")
            add_task(task)
        elif c == '2':
            View_tasks()
        elif c == '3':
            task_number = int(input(
                "Enter task number"))
            remove_task(task_number)
        elif c == '4':
            print("Exiting....")
            break
        else
            print("Invalid choice")
if __name__ == "__main__":
    main().
```

ii) Graphical User Interface (GUI)

Thinter is used to create a simple GUI for our TO-Do List application.



Program:

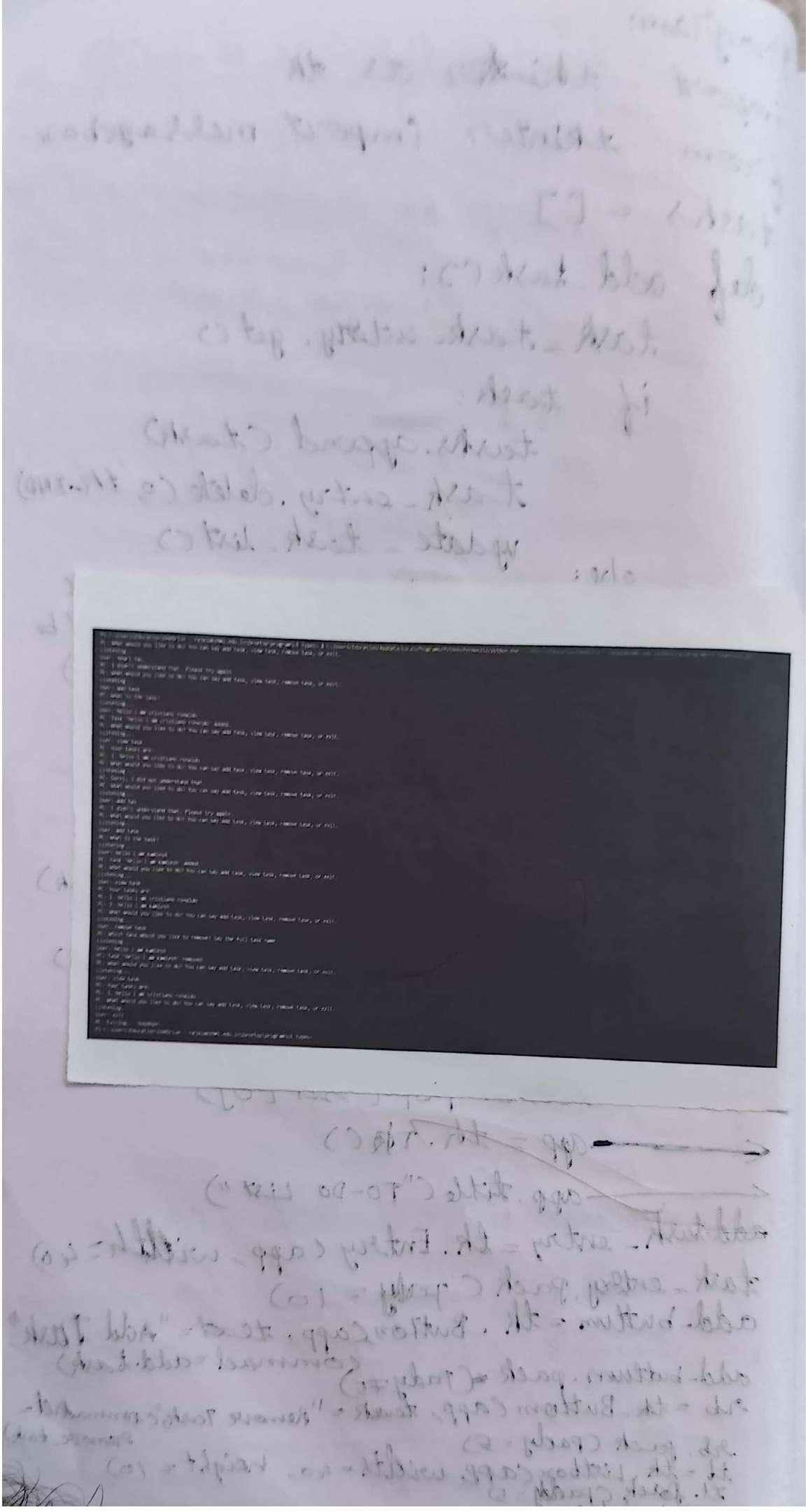
```
import tkinter as tk
from tkinter import messagebox
tasks = []
def add_task():
    task = task_entry.get()
    if task:
        tasks.append(task)
        task_entry.delete(0, tk.END)
        update_task_list()
    else:
        messagebox.showwarning(
            "Warning", "Task cannot be
                                empty")
def update_task_list():
    task_list.delete(0, tk.END)
    for t in tasks:
        task_list.insert(tk.END, task)
def remove_task():
    sti = task_list.curselection()
    if sti:
        task_list.delete(sti)
        tasks.pop(sti[0])
app = tk.Tk()
app.title("TO-DO List")
add_task_entry = tk.Entry(app, width=40)
task_entry.pack(pady=10)
add_button = tk.Button(app, text="Add Task",
                       command=add_task)
add_button.pack(pady=5)
rb = tk.Button(app, text="Remove Task", command=
               remove_task)
rb.pack(pady=5)
tl = tk.Listbox(app, width=40, height=10)
tl.pack(pady=5)
app.mainloop()
```

iii) Voice User Interface (VUI):

Speech recognition library for voice input and pyttsx3 library for text-to-speech output. make sure you have these libraries installed (pip install speech Recognition pyttsx3)

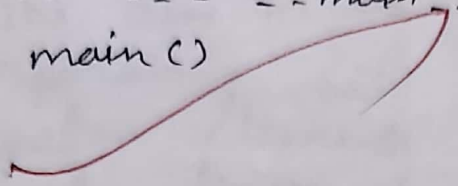
Program:

```
import speech_recognition as sr
import pyttsx3
tasks = []
recognizer = sr.Recognizer()
engine = pyttsx3.init()
def speak(text):
    print(f"PC: {text}")
    engine.say(text)
    engine.runAndWait()
def add_task(task):
    tasks.append(task)
    speak(f"Task '{task}' added.")
def view_task():
    if tasks:
        speak("Your tasks are:")
        for i, task in enumerate(task, 1):
            Speak(f"{i}. {task}")
    else:
        speak("No tasks to show.")
def remove_task(task_name):
    if task_name in tasks:
        tasks.remove(task_name)
        speak(f"Task '{task_name}' removed.")
    else:
        speak("Task not found. Please try
                again.")
```



```
def recognize_speech():
    with sr.Microphone() as source:
        print("Listening...")
        audio = recognizer.listen(source)
        try:
            text = recognizer.Recognize_google(audio).
                                                lower()
            print(f"User: {text}")
            return text
        except sr.UnknownValueError:
            speak("sorry, I did not
                                understand that.")
            return None
        except sr.RequestError:
            Speak("sorry, I am having
                                trouble connecting.")
            return None
def main():
    While True:
        speak("What would you like to do? you can
                say add task, View task,
                remove task, or exit.")
        command = recognize_speech()
        if not command:
            continue
        if "add" in command and "task" in
                                        command:
            speak("What is the task?")
            task = recognize_speech()
            if task:
                add_task(task)
        elif "view" in command and "task" in
                                        command:
            ViewTask()
        elif "remove" in command and "task" in
                                        command:
            speak("Which task whould you like
                to remove? say the full
                task.name.")
            th = recognize_speech()
            if th:
                return remove_task(th)
```

```
elif exit in command:
    speak("Exiting... Goodbye!")
    break
else:
    speak("I didn't understand that
           please try again.")

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



Result:

The above all programs has been executed successfully.

Ex. :4

11/3/25

create a prototype with familiar and unfamiliar navigation elements. Evaluate ease of use with different user different user groups using proto.io

Aim:

The aim is to develop a prototype incorporating both familiar and novel navigation elements and assess usability among diverse user group using proto.io.

Procedure:

Eg:1

Step 1: Sign Up and Log In

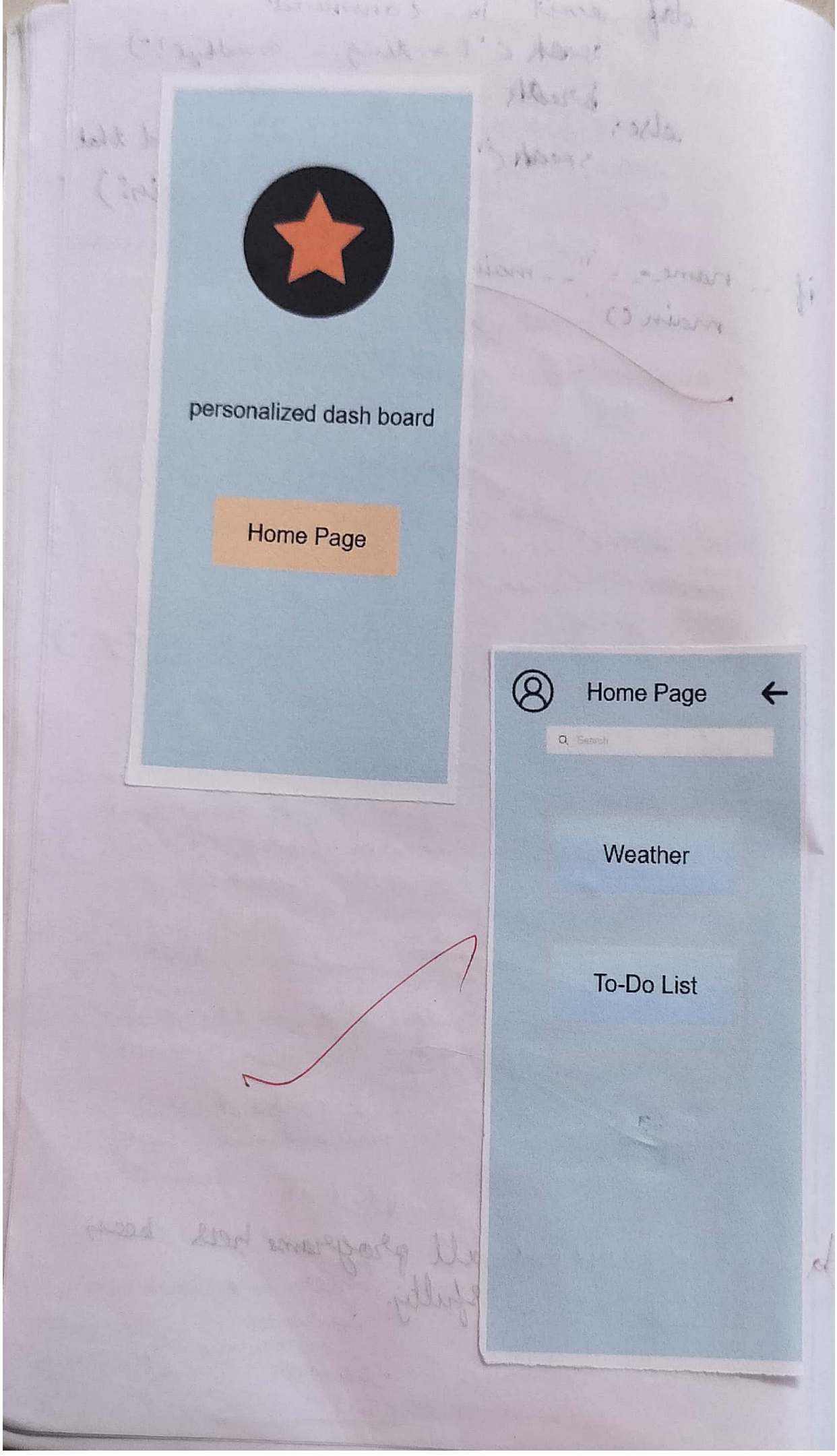
* Go to proto.io.
* Sign up for a new account or log in if you already have one.

Step-2: Create a New Project

* Click on "Create New Project."
* Give your project a name (eg: "sam.app")
* Select the device type (eg: "Mobile: oppo")
* Click "create" to start the project.

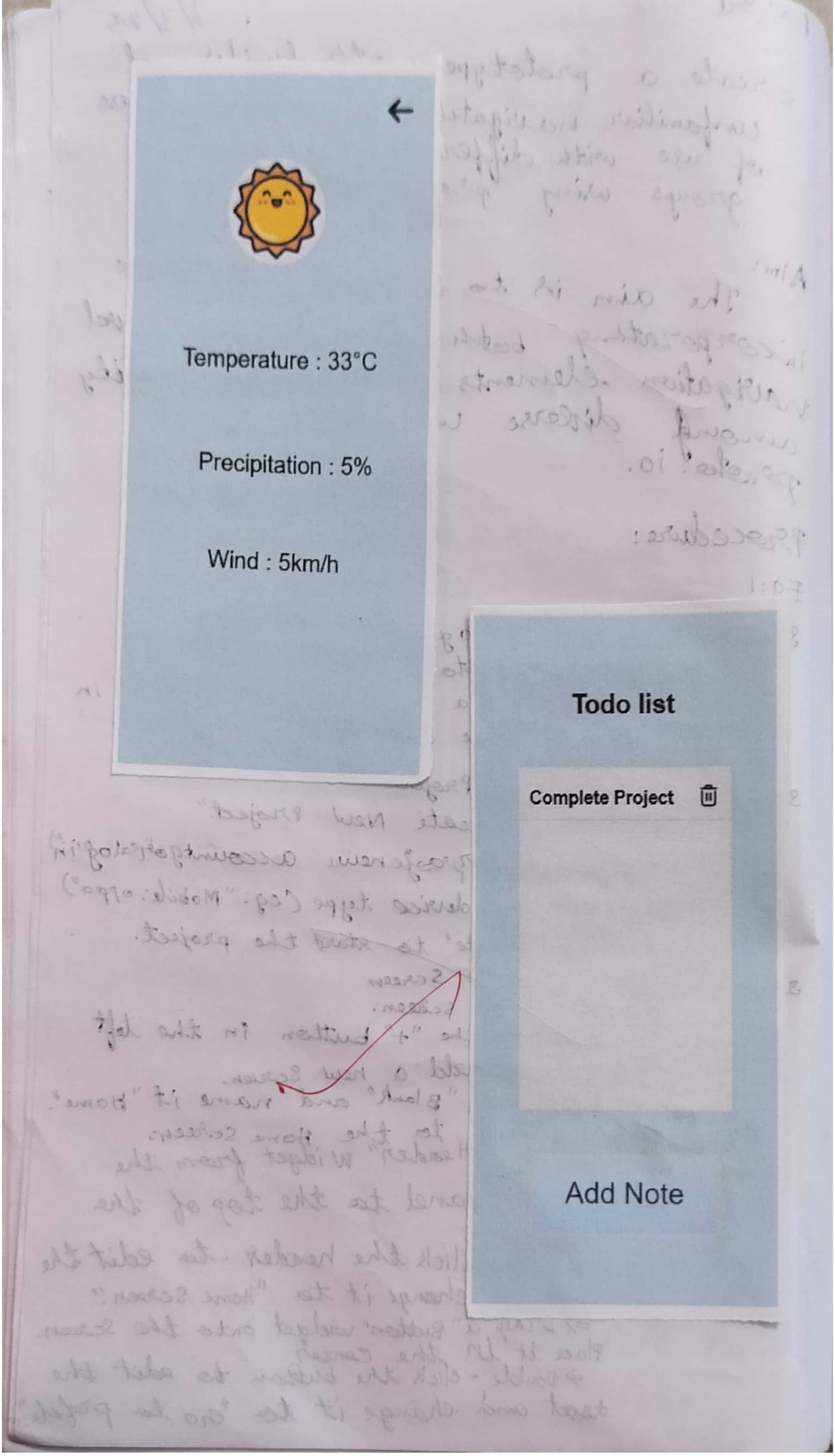
Step-3: Design the Home Screen

* Add a New screen:
  ⇒ Click on the "+" button in the left Panel to add a new screen.
  ⇒ Choose "Blank" and name it "Home".
* Add Elements to the Home Screen:
  ⇒ Drag a "Header" Widget from the "Widgets" panel to the top of the Screen.
  ⇒ Double-click the header to edit the text and change it to "Home Screen".
  ⇒ Drag a "Button" widget onto the Screen. Place it in the center.
  ⇒ Double-click the button to edit the text and change it to "Go to profile".



* Add Interactions:
=> select the button and click on the "Interactions" tab on the right panel.
=> Click "+ Add Interaction".
=> Set the trigger to "Tap/click".
=> Set the action to "Navigate to screen" and choose "New screen."
=> Create a new screen and name it "profile".

Step-4: Design the profile Screen.
* Add elements to the profile Screen:
=> On the newly created profile screen, drag a "Header" widget to the top of the screen.
=> Double-click the header to edit the text and change it to the "profile screen".
=> Drag an "Image" widget onto the screen. Place it below the header.
=> Double-click the image to upload a profile picture or any placeholder image.
=> Drag a "Text" widget onto the screen to add some profile information (eg:- "Kanalesh, Software Engineer").
* Add Back Button:
=> Drag a "Button" widget onto the screen.
=> Double-click the button to edit the text and change it to "Back to Home".
* Add Interaction:
=> select the button and click on the "Interactions" tab on the right panel.
=> Click "+ Add Interaction".
=> Set the trigger to "Tap/click".
=> Set the action to "Navigation"



to screen and choose "Home".

Step-5: preview the prototype
* Click on the "preview" button in the top-right corner.
* Interact with the prototype by clicking on the buttons to navigate between the Home and profile screens.

Step-6: share the prototype.
* click on the "share" button in the top-right corner.
* copy the shareable link and send it to others for feedback.

Eg. 2:

Step-1: plan your prototype
* Identify your Elements:
=> Familiar: common navigation elements such as a top menu bar, side panels, bread crumb trails, and Footer links.
=> Unfamiliar: Experiment with things like hidden menus, gesture-based navigation, or voice commands.
* sketch out your concept:
=> Draft wireframes on paper, using tools like Figma or sketch to visualize how both elements will coexist.

Step-2: Start your project on proto.io
* sign UP/Log In:
=> Go to proto.io and either create an account or log in if you already have one.
* Create New project:
=> Click on the "create a new project" button, select the type of project and give it a name.
* choose a Template:
=> select a template that suits your needs or start from scratch.

Step-3: Design your Screens:

* Familiar Navigation:
  => Drag and Drop elements like menu, tabs, button that users are accustomed to.
* Unfamiliar Navigation:
  => Add unique elements such as swipe gestures, hover interactions, or voice commands.
* Link Screens:
  => Use proto.io's interaction design tools to set up transitions between Screens.

Step-4: Gather User Groups:

* Defined User Groups:
  => Segment user into different categories such as age group, tech-saviness, or experience with similar products.
* Recruit participants:
  => use platforms like 'user-Testing, Surveys, or social media to find participants.

Step-5: Conduct usability Testing.

* Deploy the prototype:
  => Share the unique project link or invite users to test your prototype directly through proto.io.
* Test session:
  => Conduct usability tests with users from each group, giving them specific tasks to accomplish.
* Collect Feedback:
  => use proto.io's feedback tools or conduct interviews to gather their thoughts and experiences.

Step-4: Analyze and Evaluate.

* Data Analysis:
  => Look at how user interacted with each element use product's analytics tools to draw insights.
* Compare Groups:
  => Compare how different user groups responded to familiar Vs unfamiliar navigation.
* Report Findings:
  => Summarize the results in a detailed report highlighting key insights, pain points and recommendation.

Result:

The above task has been executed successfully.

Ex. 4 a

29/3/25

Conduct task analysis for an app (eg. Shopping apps) and document user flows. Create corresponding wireframes using Lucid chart.

Aim:

To understand and document the step a user takes to complete the main tasks within tasks within an online shopping app.

Procedure:

Step-1: Assignment Tasks.

* Browsing products
* searching for a specific product
* Adding a product to the cart
* checking out.

Step-2: Document user flows

* Browsing products:

⇒ Home Screen: User lands on the home page with product categories.

⇒ product categories: user taps on a category to view products.

⇒ product list: User scrolls through the product list.

⇒ Product Details: User taps on a specific product to see details.

• Home Screen → Product Categories → Product List → product Details.

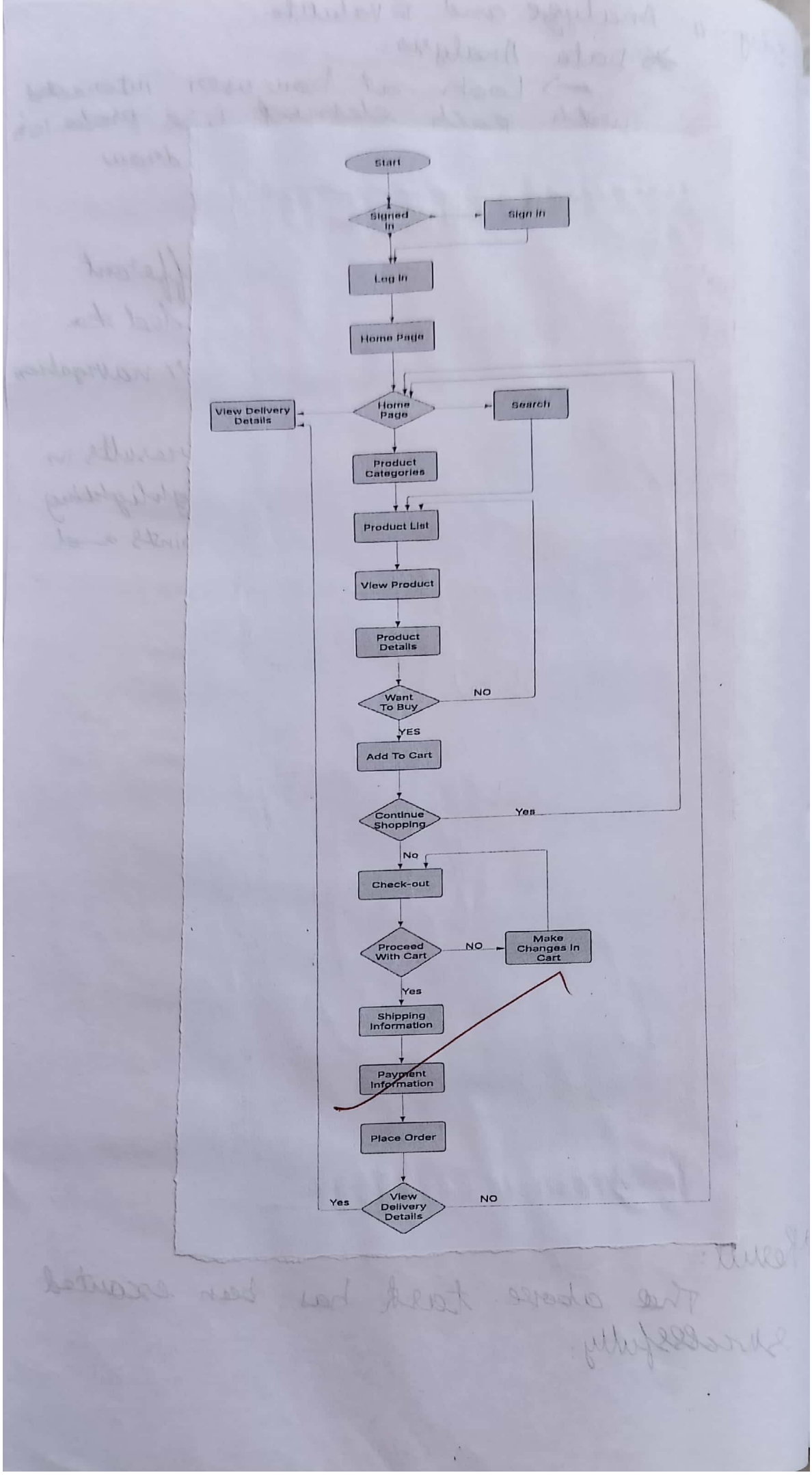
* Searching for a specific product

⇒ search: User taps the search bar or icon.

⇒ Enter Query: User types the product name or keyword.

⇒ Search Result: User reviews matching items.

⇒ Product details: user taps on a specific product to see details.


Start
Signed In
Sign In
Log In
Home Page
View Delivery Details
Home Page
Search
Product Categories
Product List
View Product
Product Details
Want To Buy
NO
YES
Add To Cart
Continue Shopping
Yes
No
Check-out
Proceed With Cart
NO
Make Changes In Cart
Yes
Shipping Information
Payment Information
Place Order
Yes
View Delivery Details
NO

- search → Enter Query → Search Results → product Details.

  * Adding a product to the cart.
    => View products: User browses or searches for a product.
    => Product Details: User taps on the product to see more info.
    => Add to cart: User clicks "Add to cart."

- View product → Product Details → Add to cart

  * checking out.
    => Open cart: User taps on the cart icon.
    => Review cart: User checks all products.
    => Proceed to checking out: User clicks "checkout"
    => Enter Shipping Info: User provides shipping details.
    => Enter ~~Shipping~~ Payment Info: User provides payment details.
    => Place Order: User click "Place order".

- Open cart → Review cart → proceed to checkout → Enter Shipping Info → Enter payment Info → Place order.

step by step procedure to create user flows in Lucid Chart:

* create a New Document:
  => Go to Lucidchart and sign in or sign up if you don't have an account.
  => click on + Document or create New Diagram.
* Select a Template
  => You can start with a blank document or select a flow chart template
  => For this example, let's start with a blank document.
* Add Shapes for Each Step:
  => Drag and drop shapes from the left sidebar to represent different steps in your flow.
  => Name each shape based on the steps

* Connect the shapes:
⇒ Use connectors to link the shapes, indicating the flow from one step to the next.
⇒ Add arrow to show the diff direction of the flow.
* Add details to each steps.
⇒ Double-click on each shape to add text describing the action or decision.
* Use different shapes for different Actions:
⇒ Use rectangles for general actions.
⇒ Use diamonds for decision points.
⇒ Use ovals for start and end points.
* Customize and organize your flowchart
⇒ Arrange the shape and connectors logically.
⇒ Use different colors to distinguish between types of steps or user roles.
⇒ Group related steps into sections for better clarity.
* Review and save your flowchart
⇒ Review the flowchart to ensure all steps are included and connected correctly.
⇒ Save your flowchart by clicking on file → Save.
* Share and collaborate:
⇒ Click on the share button to collaborate with others.
⇒ You can also export your flowchart as an image or PDF for presentation purposes.

Example Flowchart Breakdown:

Log in/Register Flow:
⇒ open the app.
⇒ Click on Login / Register
⇒ Enter details.
⇒ Verify
⇒ Redirect to the home screen

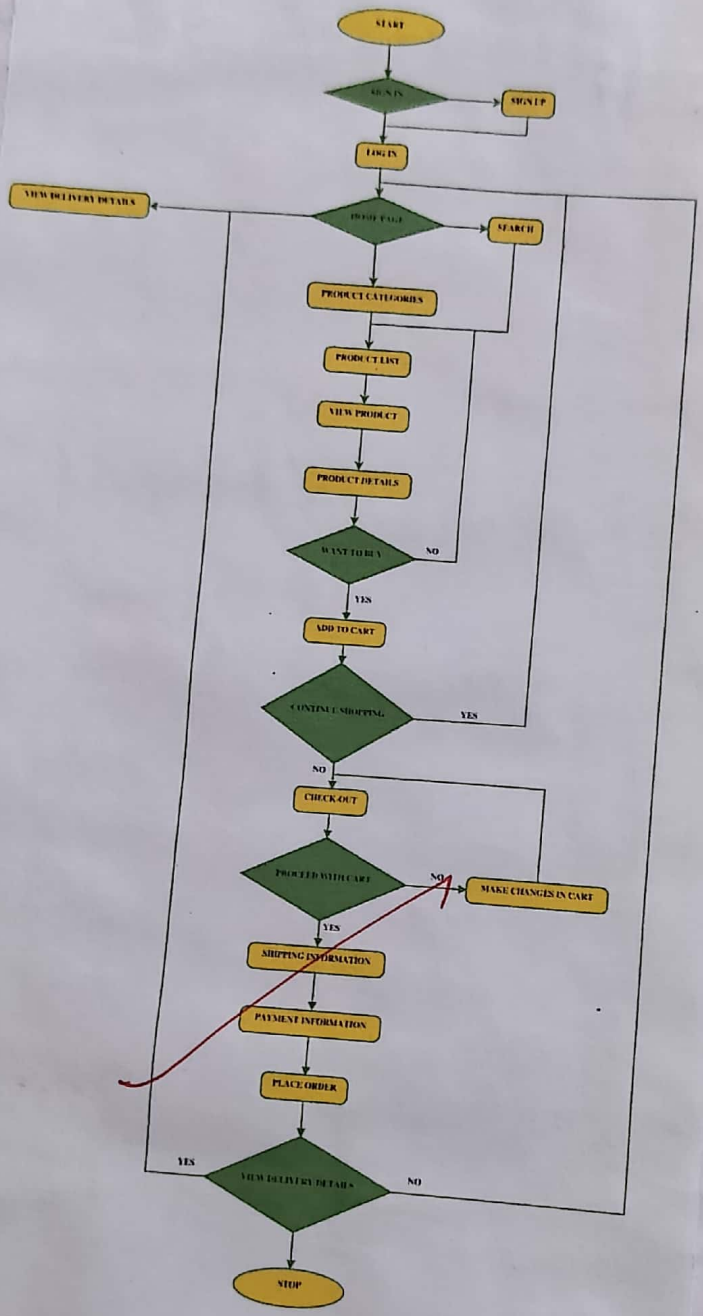
Browse and search flow:
=> Navigate to categories and use search bar,
=> Apply filters/sorting Option.
=> View product details.

Add to cart flow:
=> View product details
=> Select Options.
=> Add product to cart

Check out flow:
=> Review cart
=> proceed to checkout
=> Enter shipping information.
=> Select Payment method.
=> ~~Select~~ confirm and place order.

Order Track flow:
=> Navigate to my orders.
=> select order to track.
=> View tracking details.

Result:
Thus the above all procedure has been completed successfully.


START
SIGN IN
SIGN UP
LOG IN
VIEW DELIVERY DETAILS
SEARCH
PRODUCT CATEGORIES
PRODUCT LIST
VIEW PRODUCT
PRODUCT DETAILS
NO
YES
ADD TO CART
YES
NO
CHECK OUT
NO
MAKE CHANGES IN CART
YES
SHIPPING INFORMATION
PAYMENT INFORMATION
PLACE ORDER
YES
NO
STOP

Ex.4b 5/4/25

Conduct task analysis for an app and document user flows. Create corresponding Wireframe using Dia.

Aim:

The aim is to perform task analysis for an app, such as online shopping, document user flows and create corresponding wireframes using Dia

Procedure:

* Install Dia:
  => Download and install Dia from official website.
* oper Dia.
  => Lanch the Dia application.
* Create New Diagram:
  => Go to file → New Diagram.
  => select flowchart as the diagram type.
* Add Shapes:
  => Use the shape tools to create wireframes for each screen.
  Eg: Rectangle: Home page, product categories, product Listing, product details, cart, checkout, order confirmations, order history.
* connect Shapes:
  => Use the line tool to connect shapes, represent the user flow.

Eg: Home page → product categories → product Listing → product details → cart → check-out → order confirmation → orders History.

* Lable Shapes:

⇒ Double-click on each shape to add labels.

Eg: Rectangle: Home page, product categories, product listing, product details, cart, checkout, order confirmation, order History.

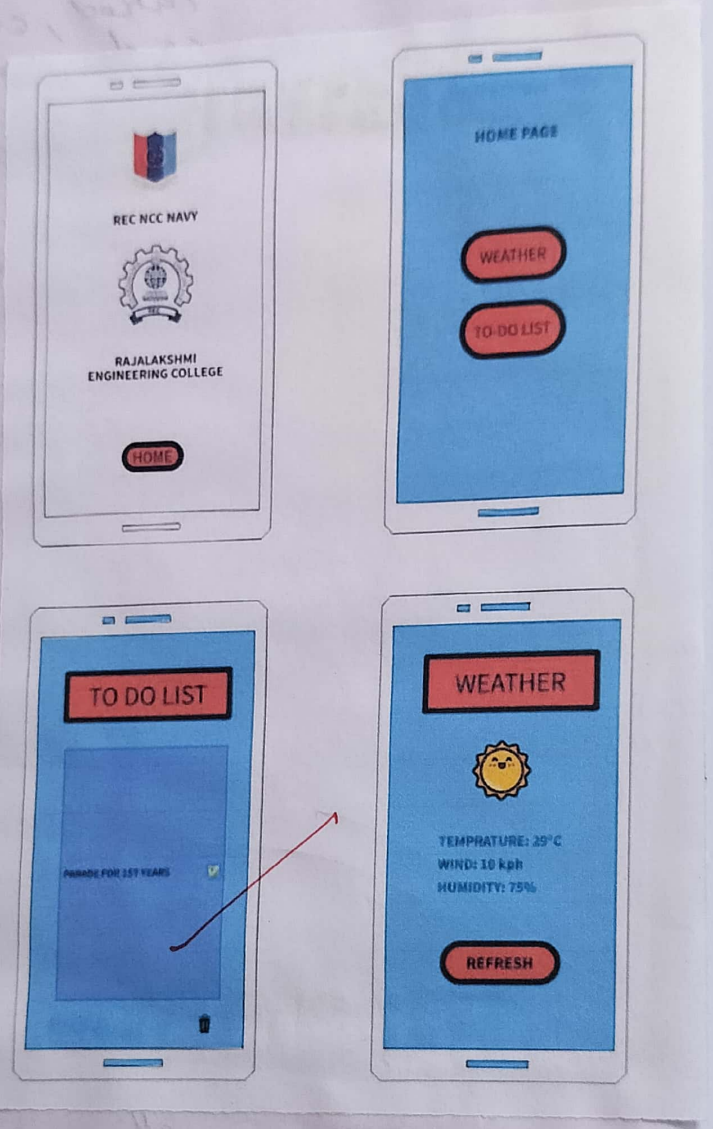
* Save the diagram.

⇒ Go to file → Save As.

⇒ Save the diagram with a meaningful name.

Result:

Thus the above all procedures are executed successfully.



Ex. 3 b

Create a prototype with familar & unfamilar navigation elements.
Evaluate ease of use with different user group.

Aim:

The aim is to design a prototype with both Well-Known and new navigation elements and measure user-friendliness across different user group using mockflow/wireflow

procedure:

step-1: plan your prototype

* Define Navigation Elements:
  - =>Familiar: Standard menus, top bar, footers and sidebar navigation
  - =>Unfamiliar: Novel features such as hidden menus, gesture-based navigation, or custom swipes.
* Sketch your Layout:
  - => Start with paper sketches or use tools like figma or sketch to visulize your design concepts.

Step-2: Set Up your wireflow project

* Sign Up/Sign In:
  - => Head to wireflow and create an account or log in if you already have one.
* Start a New project:
  - => Click on "New project" and Name it. choose a template or start from scratch.

Step-3: Design the prototype.

* Add Familiar Navigation Elements:
  - => Drag and Drop component like menus, header bar, button, etc., into your screens.
* Incorporate Unfamiliar Elements:
  - => Introduce hidden menus, unique gestures or unexpected interactions.

* Link Screens:
=> Use wireflow's linking tools to create connections and transitions between screens.

Step-4: prepare for Usability Testing
* Identify User Groups:
=> segment user based on age, tech savviness, or previous experience with similar products.
* Re-cruit Participants:
=> Use online tools like User Testing, forms or social media to find participants.

Step-5: conduct Testing.
* Share the prototype:
=> Share Invite user to interact with your prototype via a shareable link from wireflow.
* Test sessions:
=> Ask users to complete task using both types of navigation. observe their interactions and collect feedback.
* Collect Feedback:
=> Utilize wireflow's feedback features or conduct following-up interviews to gather detailed responses.

Step-6: Analysis and Report:
* Analysis Data:
=> Review the feedback and data collected. Look for patterns in ease of use and user preferences.
* Compare Results:
=> Compare how different user groups interacted with familiar Vs unfamiliar navigation.
* Create a Report:
=> Summarize your finding, highlighting insights, chenlleges and recommendations.

Result:
Hence all the above processes are executed successfully.

Ex. No: 5a 12/4/25

Simulate the lifecycle stage for UI design using RAD model and develop a small interactive interface using Axure RP.

Aim:

The aim is to demonstrate the lifecycle stages of UI design via the RAD model and develop a small interactive interface employing Axure RP.

Procedure:

Simulating for lifecycle stage for UI design using the RAD model.

RAD (Rapid Application Development) model emphasizes quick development and iteration. It consists of the following phases:

=> Requirement planning:
* Gather initial requirements and identify key features of the UI.
* Engage stakeholders to understand their needs and expectations.

=> User design:
* Create initial prototypes and wireframes.
* Conduct user feedback sessions to refine the designs.
* Use tools like Axure RP to develop interactive prototypes.

=> Construction:
* Develop the actual UI based on the refined designs.
* perform interactive testing and feedback cycle.

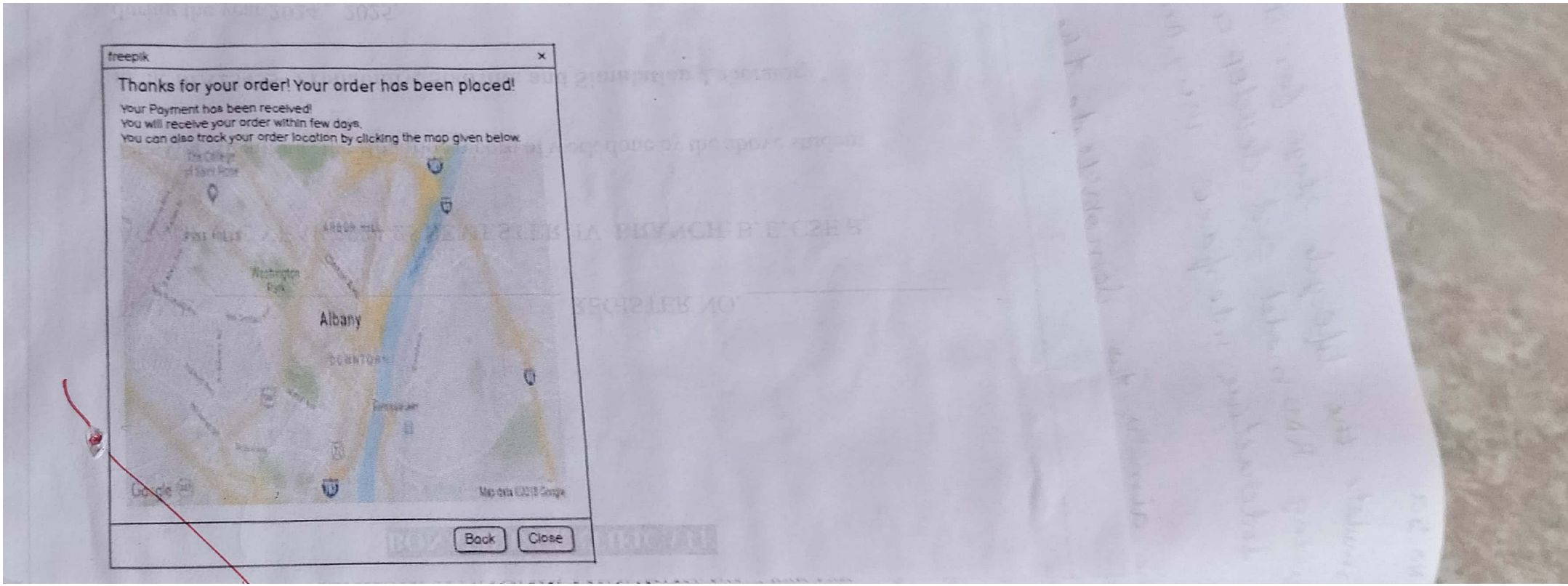
=> Cutover:
* Deploy the final UI.
* Conduct user training and support.

Axure RP Interactive Interface Development:

Phase 1: Requirement planning.

=> Identify Key Features:
* Navigation
* User action.

=> Create a Requirement:
* List all feature and functionalities
* Document user stories and use case.



Phase 2: User Design.

=> Install and Lauch Axure RP:

* Download and install Axure RP from Axure's official website.

* Lauch the application.

=> Create a New project:

* Go to file -> New to create a new project

* Name the project.

=> Create Wireframes:

* Use the widget library to drag and drop elements onto the canvas.

* Design wireframes for each system.

- Home page
- Product categories
- product listing
- product details.
- cart
- checkout
- order confirmation
- order History.

=> Add Interaction:

* Select element and go to properties panel

* Click on interaction and choose an interaction.

* Define the action.

=> create masters:

* create reusable component using masters.

* Drag and drop master on the wireframes

=> Add Annotation:

* Add notes to describe each element's purpose and functionality

* Use the notes panel to add detailed annotation.

Phase 3: Construction.

=> Develop Interactive prototypes:

* Convert wireframes into interactive prototype by adding interaction and transition.

* Use dynamic panel to create interactive element.

=> Test and Iterate:

* preview the prototype using the preview button.

* Gather feedback from user and stake-holder.

* make necessary adjustment based on feedback.

Phase 4. [illegible]:

=> Finalize and Export.

* Finalizing the design and Interaction.

* Export the prototype as an HTML file or Share it via Axure cloud.

=> User Training and support.

* conduct training session to familiarize user with the new interfaces.

* Provide document and support for any issues.

Result:

The above process completed Successfully.

Ex. No. 5b

Simulate the life cycle stages for UI design using the RAD model and develop a small interface using Openproj

Aim:

To recreate the lifecycle of UI design using the RAD model and design a small interactive interface with openproj.

Procedure:

Step-1: Requirements Planning

=> Gather Requirements:

* Identify key features and functionalities needed for your interface.

=> Define Use case:

* specify use cases for user login and registration.

- Output in Openproj

=> Create a new project

=> Add tasks: "Gather Requirements" and "Define Use cases."

=> Set durations and dependencies for each task.

Step-2: User Design

=> Sketch Initial Designs:

* Draw rough sketches of the "Login" and "Register" screens on paper.

=> Create Digital Wireframes:

* Use a tool like Figma or Sketch to create digital wireframes.

- Example

=> Login Screen

=> Register Screen.

- Output

=> Add tasks: "Sketch Initial Designs" and "Create Digital wireframes".

=> Allocate time and resources to complete these tasks.

Step-3: Rapid prototyping

=> Develop prototypes:

* Use a tool like Axure RP to convert wireframe into interactive prototypes.

=> Test Prototypes:
* Share prototypes with stakeholders for feedback.
* collect feedback and iterate on the design.

- Output:
=> Interactive prototypes for "Login" and "Register" screens.

- Output:
=> Add tasks: "Develope prototypes" and "Test prototypes".
=> set dependencies and milestones.

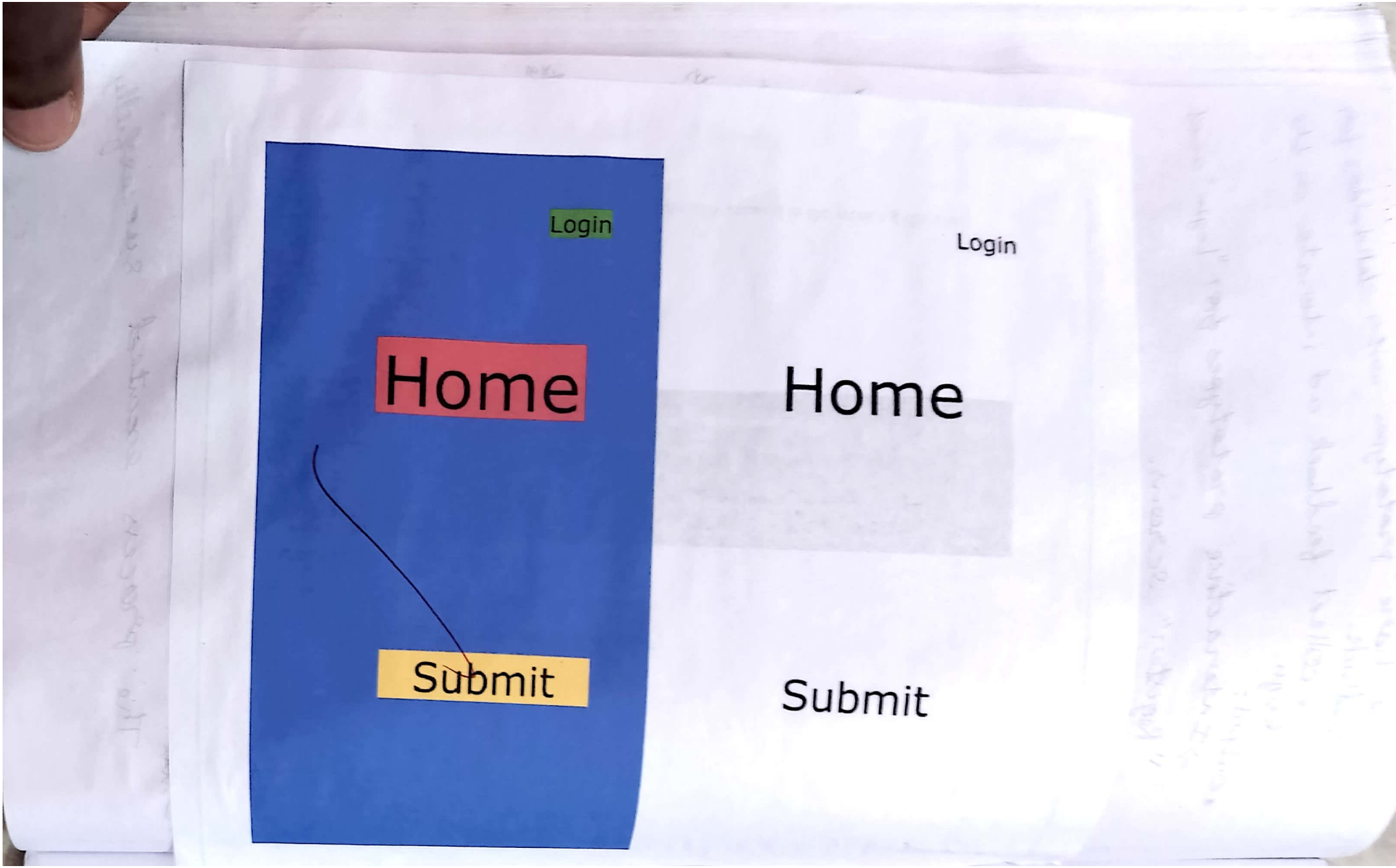
Step - 4: User Acceptance (Testing.
=> Review Prototype:
* conduct user and stakeholder reviews.
=> conduct Usability testing:
* perform usability testing and document feedback.

- Output:
=> Documented feedback and test results.

- output:
=> Add tasks: "Review Prototype" and "Usability Testing".
=> Track progress and resources.

Steps - 5: Implementation.
=> Develop functional Interface:
* Implement final designs and functionalities based on the feedback.
=> Integrate Backend (if required):
* Connect the UI with backend services for tasks like user authentication.

Result:
The process executed successfully.



Ex. NO.: 6 12/4/25

Experiment with different layout and colour schemes for an app. collect user feedback on aesthetics and usability using GIMP

Aim:

To trial different app layouts and colour schemes and evaluate user feedback on aesthetics and usability using GIMP.

Procedure:

Step-1: Install GIMP

=> Download and Install: Download GIMP from GIMP Downloads and install it on your computer.

Step-2: Create a New project

=> Open GIMP

* Launch the GIMP application.

=> Create a New Canvas:

* Go to file → New to create a new project.

* Set the dimensions for your app layout.

Step-3: Design the Base Layout

=> Create the Base layout:

* Use the Rectangle select tool to create sections for different parts of your app.

* Fill these sections with basic colors using the Bucket fill tool.

Eg O/P: A based layout with defined sections for header, content and footer.

=> Add UI elements:

* Text elements: Use the text tool to add text element like header, buttons and labels

* Interactive Element: Use the Brush tool or shape tools to draw buttons input field and other interactive elements.

UI 1: Blue Background Version

Insights and data showing user preferences and experience:

- **75%** of users found the interface **visually striking**, but **56%** reported it was **too intense or distracting** due to clashing colors.
- **Login button placement** in the top corner was noticed by only **28%** of users during the first 5 seconds of interaction.
- Users appreciated the **large "Home" button**, saying it was **"hard to miss"**, but **41% felt it overpowered** other elements on the screen.
- **Color usage** was mentioned by **62%** of testers as needing better balance or **more consistent design language.**

UI 2: White Background Version

Insights and data showing user preferences and experience:

- **81%** of users said the design felt **clean, minimal, and easy on the eyes.**
- However, **67% struggled** to identify the buttons at first glance, with feedback stating they **"look like plain text"** without visual indicators.
- The lack of button styling led to **49%** saying **it lacked a clear visual hierarchy.**
- Despite its plainness, **73% of users preferred** this UI for **professional or productivity apps** due to its neutral tone and readability.

Eg. O/P: A layout with labeled section and basic UI element.

⇒ Organize layers:

* User layers to seperate different UI elements. This allows you to easily modify or experiment with individual components.

* Name each layer according to its content.

Step-4: Experiment with Color schemes

⇒ Create color variants:

* Duplicate Layout: Duplicate the base layout by right-clicking on the image tab and selecting Duplicate.

* change colors: Use the Bucket fill tool or colorize tool to change the colors of the UI elements in each duplicate.

Eg. O/p: Multiple color Variant of the same layout.

⇒ Save Each Variant:

* Save each color variant as a separate file.

* Go to file → Export as and choose the file format.

Step-5: Collect user feedback.

⇒ prepare a Feedback form:

* Create Form: create a feedback from using tools like google forms or Mforms.

* Include Questions: Include questions about the aesthetics and usability of each layout and color scheme.

⇒ Share the Variants:

* Distribute files: Share the image files of the different layout and color schemes with your users.

* provide Instructions: provide clear instructions on how to view each variant and how to fill out the feedback form.

⇒ Gather feedback:

* Collect responses from user regarding their preferences and suggestions.

* Analysis the feedback to determine which layout and color scheme are most preferred.

Step-6: Iterate and Improve

⇒ Refine the Design:

* Based on the feedback, make necessary adjustments to the layout and color scheme.

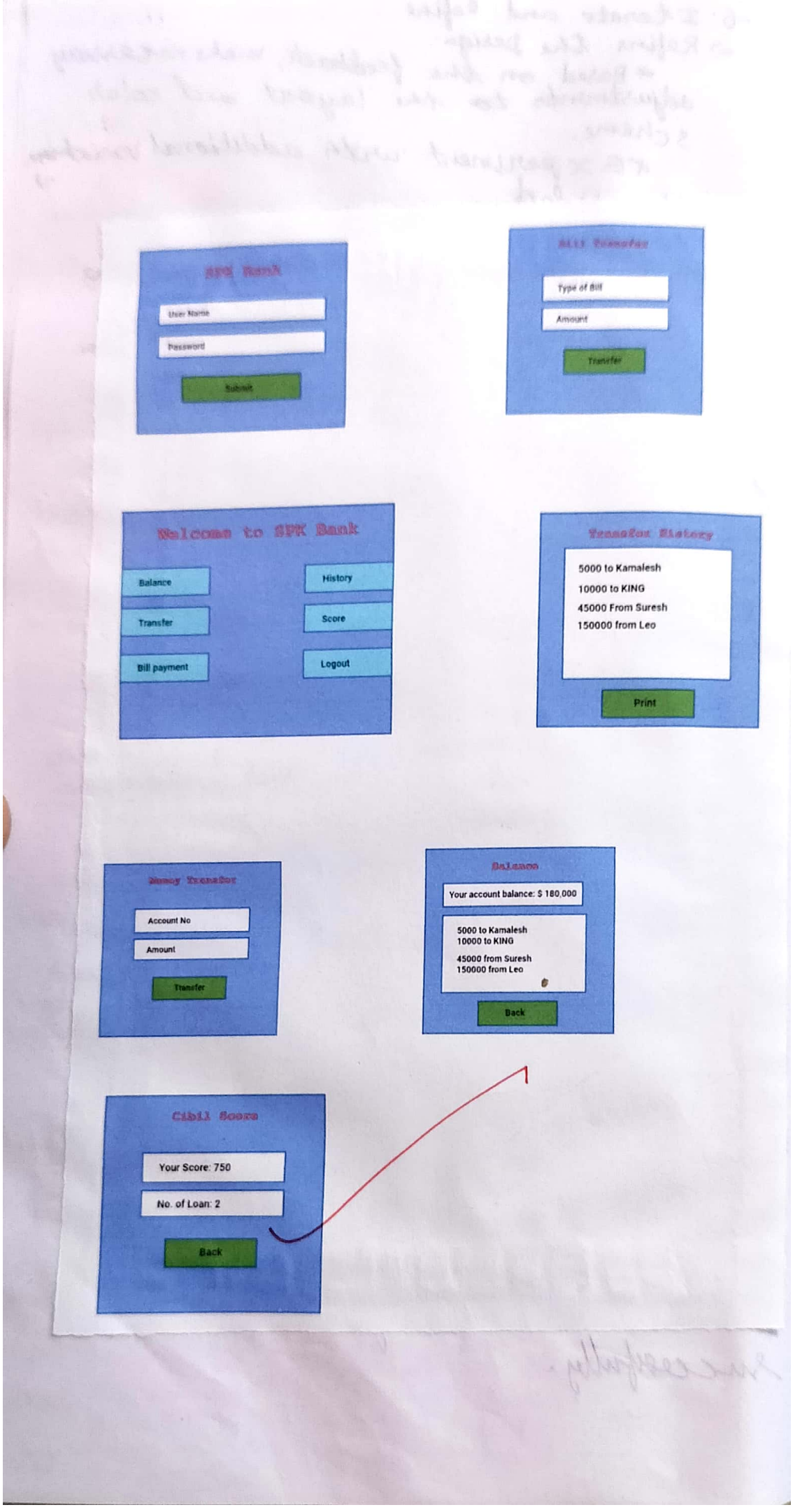
* Experiment with additional variations if needed.

⇒ Final Testing:

* Conduct a final round of testing with the refined design to ensure usability and aesthetic satisfaction.

Result:

The above program executed successfully.



Ex. No.: 7a

Develop low-fidelity paper prototypes for a banking app and convert them into digital wireframes using pencil project

Aim:

To develop low-fidelity paper prototype for a banking app and convert them into digital wireframes with pencil project

Procedure:

Step-1: Create Low-fidelity paper prototypes

=> Define the purpose and features:

* Identify the core features of the banking app.

=> Sketch Basic Layouts:

* Use plain paper and pencils to sketch basic screens.

* Focus on primary elements like buttons, menus and forms.

=> Iterate and Refine:

* Get feedback from user or stakeholders.

* Iterate on your sketches to improve clarity and functionality.

Step-2: Convert paper prototypes to Digital wireframes using pencil project.

=> Install pencil project:

* Download and Install pencil project from the official website.

=> Create a New Document:

* open pencil project and create a new document.

Step-3: Add screens.

=> Click on the "App Page" button to create different screens.

Step-4: Use Stencils and shapes:

=> Use the build-in stencil and shapes to create UI elements.

=> Drag and drop elements like buttons, text fields and icons onto your canvas.

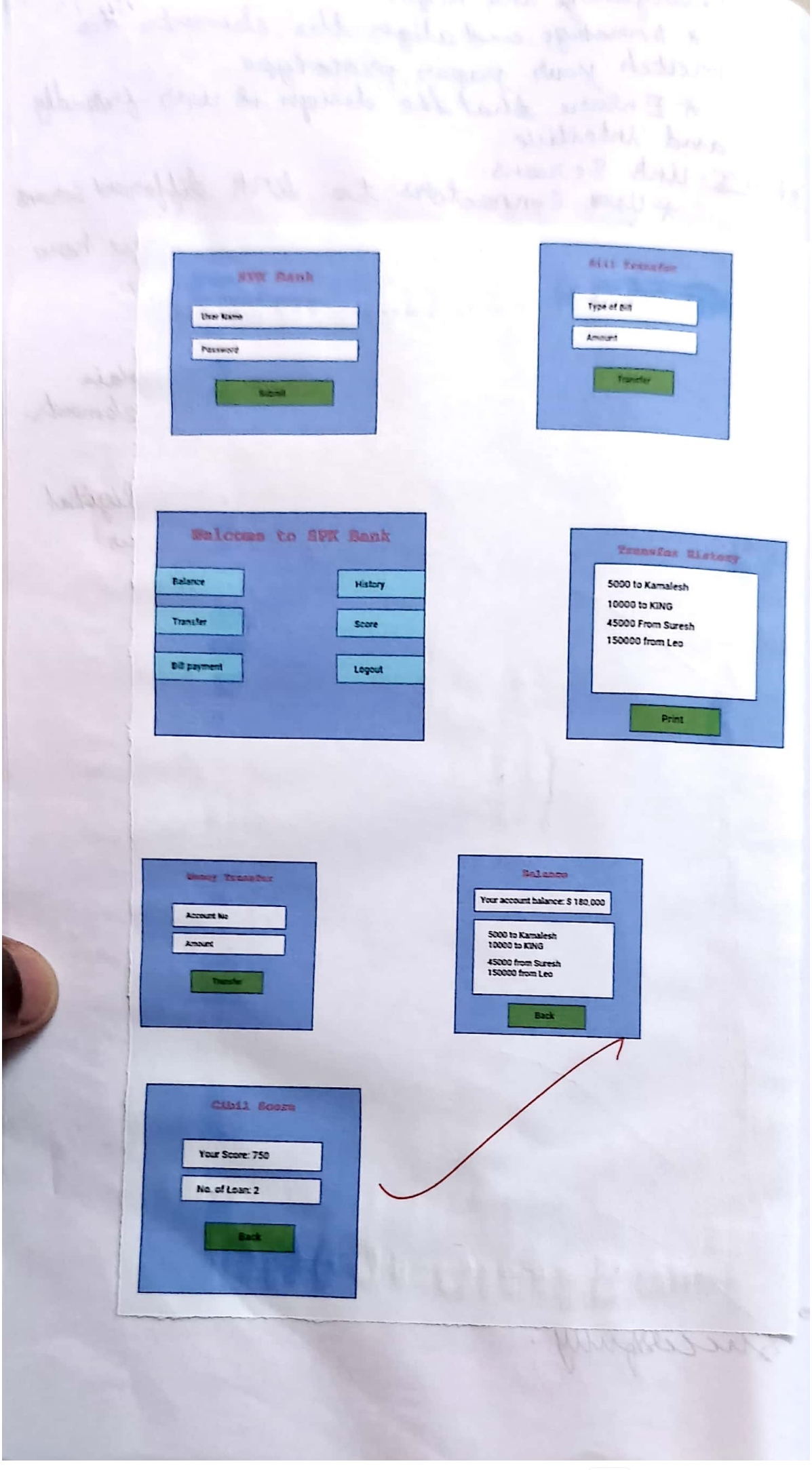
Step-5: Arrange and align the elements to match your paper prototype.
* Ensure that the design is user-friendly and intuitive.

Step-6: Link Screens.
* Use Connectors to link different screens together.
* Create navigation flows to show how user will interact with the app.

Step-7: Add Annotation.
* Include annotation to explain the functionality of different elements.

Step-8: Export your Wireframes.
* Once satisfied with your digital wireframes, export them in your preferred format.

Result:
The above process excuted Successfully.



Exp. No. 1 09/4/25

Develop low-fidelity paper prototypes for a banking app and convert them into digital wireframes using Inkscape.

Aim:

To construct low-fidelity paper prototypes for a banking app and digitize them into wireframes using Inkscape.

Procedure:

Step-1: create low-Fidelity Paper prototypes.

* Identify core Features:
  => Determine the essential features of the banking app.
* Sketch Basic layout:
  => Use plain and pencils to sketch the main screens.
  => Focus on the primary element like buttons, navigation menus ad input fields.
* Iterate and Refine:
  => Get feedback from users or stakeholders
  => Make necessary adjustment to improve clarity ad functionality

Step-2: Convert paper prototypes to Digital wireframes using Inkscape.

* Intall Inkscape:
  => Download and Install Inkscape from the official website.
* Create a New document:
  => Open Inkscape and create a new document by clicking on file -> New.

~~Step-3~~: Step UP the Document
=> set the dimensions and grid for your design. Go to file -> Document property to adjust the size.
=> Enable the grid by going to view ->

Step 5: Use the text tool to add labels and placeholder text to your elements.

Step 6: Add Text
⇒ Use the rectangle and ellipse tools to draw the basic shapes for your UI elements.

Step 6: Organize and Align:
⇒ Arrange and align the elements to match your paper prototype.
⇒ Use the alignment and distribution tools to keep everything organized.

Step 7: Group Elements:
⇒ Select related elements and group them together using object → Group.
⇒ This helps keep your design organized and easy to edit.

Step 8: Create Multiple Screens.
⇒ Duplicate your base layout to create different screens.
⇒ Use Edit → Duplicate to create copies of your elements and arrange them for each screen.

Step 9: Link Screens (Optional)
⇒ If you want to show navigation flows, you can add arrows or other indicators to demonstrate how users will move between screens.

Step 10: Export your wireframes:
⇒ Once you're satisfied with your digital wireframes, export them by going to file → Export PNG Image.
⇒ Choose the appropriate settings and export each screen as needed.

Result:
The above process executed successfully.

Ex. No. 8a

Create storyboard that represent the user flow for a mobile app using Balsamiq.

Aim:

To create storyboard to represent the workflow for a mobile app using Balsamiq.

Procedure:

Step-1: Define the user flow

* Identify key screen
  - ⇒ List the main screen your app will have.
* Map the user journey:
  - ⇒ Understand the typical user journey through those screens.

Step-2: Create storyboard using Balsamiq.

⇒ Install Balsamiq:
  - * Download and install Balsamiq web.

⇒ Create a new project:
  - * open and create new project

⇒ Add wireframe screen:
  - * Use the "+" button to add new wireframe screen for each key screen in your app.

⇒ Design Each Screen.
  - * Use the component to design the UI for each screen.
  - * Include basic elements like buttons, text field, and images.

⇒ Organize the flow
  - * Arrange the screen in the order user will navigate through them.
  - * connect the screen with arrows to represent user actions.

Eg: App:

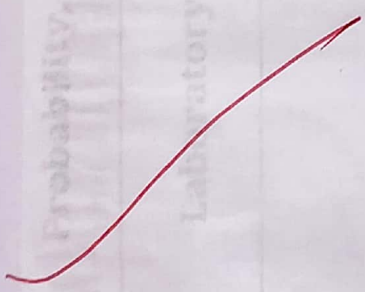
⇒ Home Screen:
* search bar
* Categories

⇒ Menu Screen:
* List of food items.
* Add to cart buttons.

⇒ Cart Screen.
* Items added with quantity and total price.
* check out button

⇒ Checkout screen
* Delevery address form
* payment option
* place order button.

⇒ order confirmation Screen:
* Order Summary
* Estimated delivery time.



Result:

The above process executed successfully.

Ex.NO. 8b

Create storyboards to represent the user flow for a mobile app using openBoard

Aim:

To map out the user flow for a mobile app, storyboads will be designed using openBoard.

Procedure:

step-1: Define the userflow.

=> Identify Key Screens:

* List the main screens your app will have.

=> Map the User Journey:

* Understand the typical user journey through these screens.

step-2: Create storyboards using openboard

=> Install OpenBoard:

* Download and install OpenBoard from the official website.

=> Create a New Document.

* Open OpenBoard and create a new document.

=> Add Frames for Each Screen:

* Use the drawing tools to create frames representing each key screen of your app.

=> Sketch Each Screen:

* Use the pen or shape tools to draw basic elements for each screen.

* Focus on major UI components like buttons, text field and icons.

=> Organize the flow:

* Arrange the frames in a sequence that represents the user journey.

* Use arrows or lines to show

navigation

Eg. O/p:

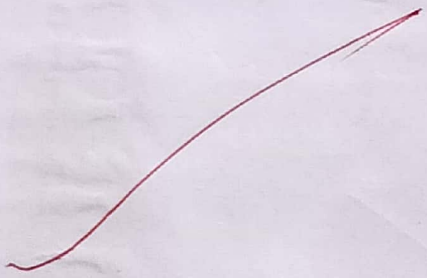
=> Home Screen:
* search bar
* categories.

=> Main Screen:
* List of food items.
* Add to card button.

=> Cart Screen:
* Items added to the cart with quantity and total price.
* Checkout button.

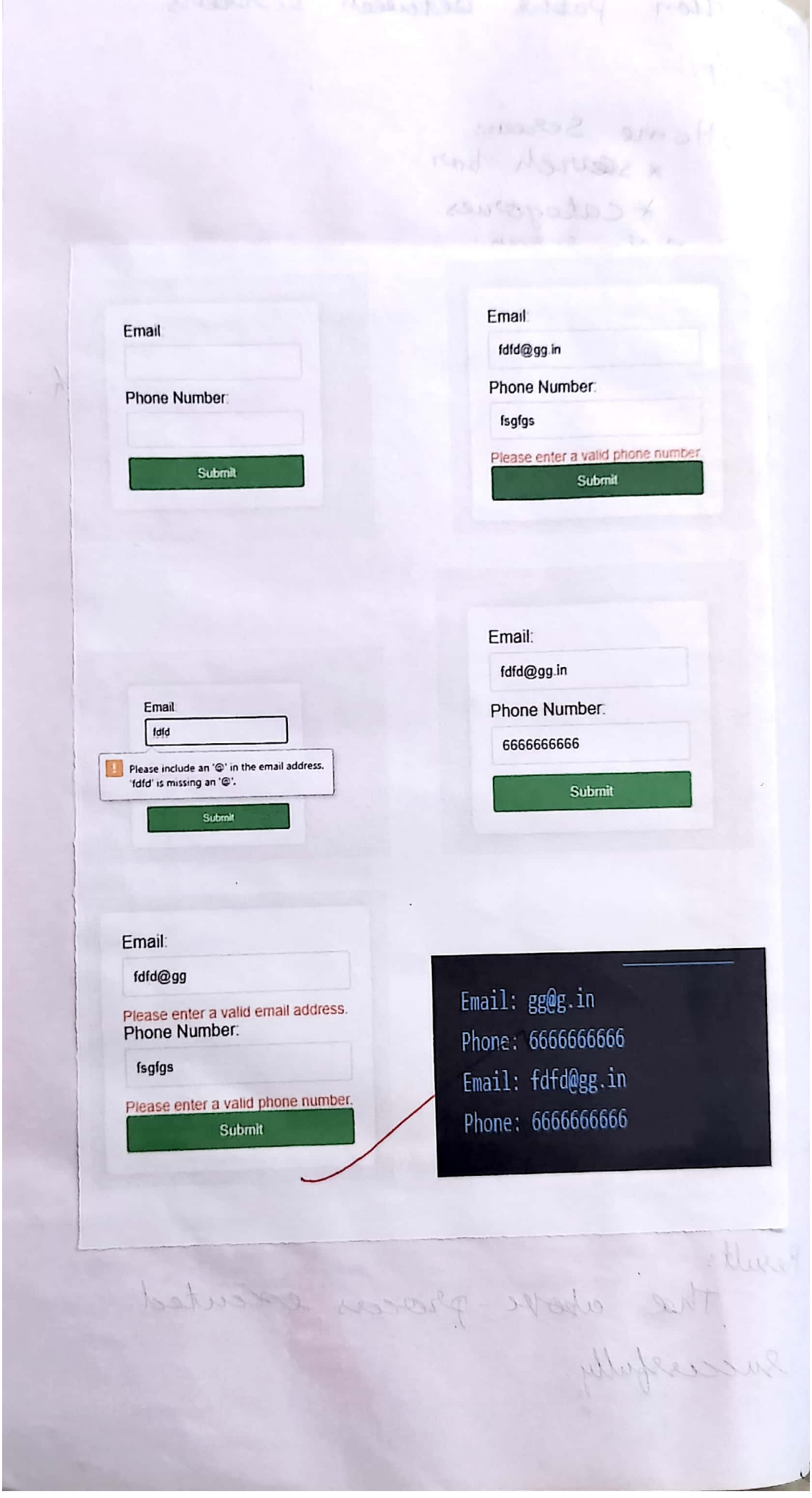
=> Checkout Screen:
* Delivery address form.
* Payment options.
* Place order button.

=> Order Confirmation Screen:
* Order summary
* Estimated delivery time.



Result:

The above process executed successfully.



Ex. No. 9 26/4/25

Design input forms that Validate data and error message using HTML/CSS/JS

Aim:

To design input forms that Validate data and error message using HTML/CSS and JavaScript with Validator.js.

Procedure:

Step-1: Setting Up the HTML Form

→ Start by creating an HTML form with input field for the email and phone number.

html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
                                   initial-scall=1.0">
    <title> Form Validation </title>
    <link rel="styleSheet" href="style.css">
</head>
<body>
    <div class="container">
        <form id="myForm">
            <label for="email"> Email: </label>
            <input type="email" id="email" name="email"
                                                required>
            <span id="emailError" class="error"></span>
            <label for="phone"> Phone Number: </label>
            <input type="text" id="phone" name="phone"
                                                required>
            <span id="phoneErros" class="error"></span>
            <button type="submit"> Submit </button>
        </form>
    </div>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/
                validator/13.6.0/validator.min.js"></script>
    <script src="script.js"></script>
```

</body>
</html>

Step-2: Styling the form with CSS

=> Next, add some basic styling to make the form look nice.

CSS:

```
body {
    font-family: Arial, sans-serif;
    background-color: #f4f4f4;
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100 vh;
    margin: 0;
}
.container {
    background-color: white;
    padding: 20px;
    border-radius: 5px;
    box-shadow: 0 0 10px rgba(0,0,0,0.1);
}
form {
    display: flex;
    flex-direction: column;
}
table {
    margin-bottom: 5px;
}
input {
    margin-bottom: 10px;
    padding: 10px;
    border: 1px solid #ccc;
    border-radius: 3px;
}
```

```
button {
    Padding : 10px;
    background-color: #28a745;
    color : White;
    border : none;
    border-radius : 3px;
    cursor : pointer;
}
button:hover {
    background-color: #218838;
}
.error {
    color : red;
    font-size: 0.875 em;
}
```

step-3: Adding Javascript for Validation.

=> Finally, add Javascript to Validate the input fields using Validator.js and display error messages.

JS:

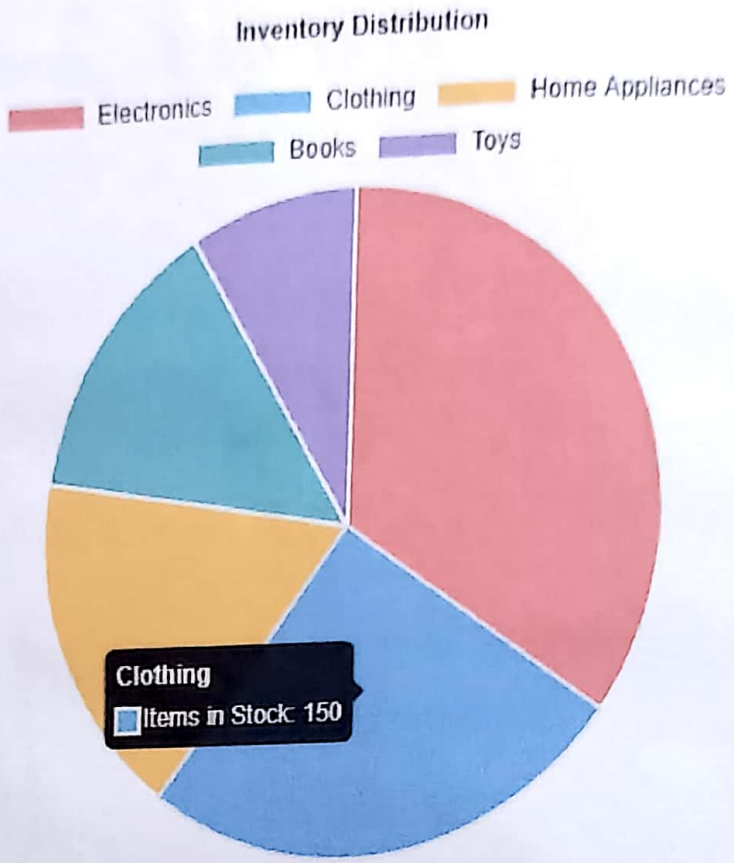
```
document.getElementById('myForm').addEventListener(
                    'submit', function(e) {
    e.preventDefault();
    let email = document.getElementById('email').value;
    let phone = document.getElementById('phone').value;
    let emailError = document.getElementById('email
                                        Error');
    let phoneError = document.getElementById('Phone
                                        Error');
    emailError.textContent = '';
    phoneError.textContent = '';
    if (!validator.isEmail(email)) {
        emailError.textContent = 'Please enter a valid
                                        email address.';
    }
```

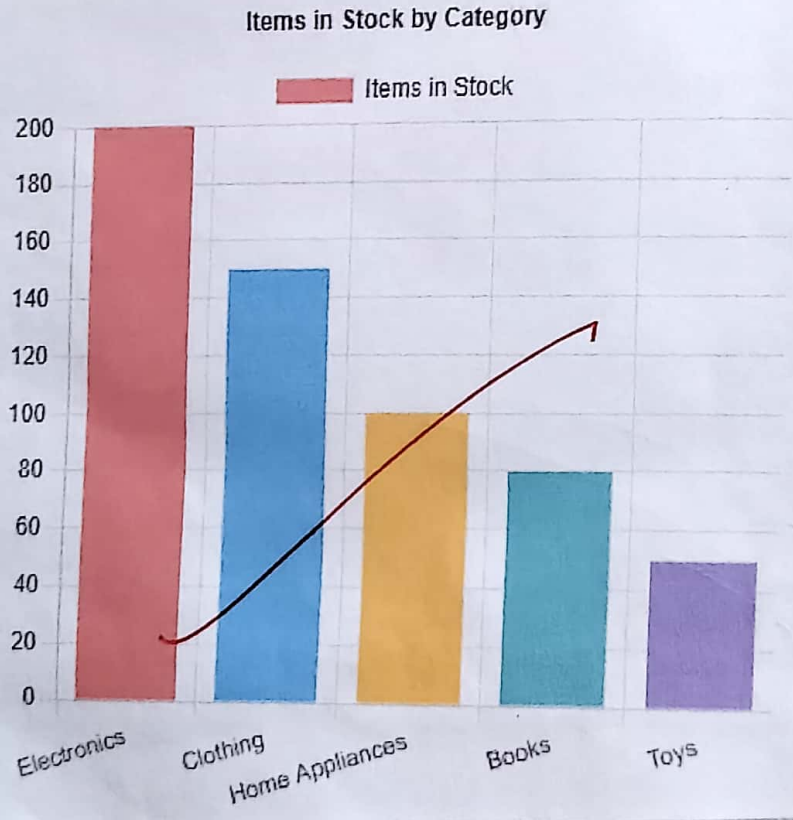
```
if (!Validator.isMobilePhone(phone, 'any')) {
    phoneError.textContent = 'Please enter a valid
                              phone number.';
}
if (Validator.isEmail(email) &&
    Validator.isMobilePhone(phone, 'any')) {
    console.log(`Email: ${email}`);
    console.log('Phone:', phone);
}
});
```

Result:

The above program executed successfully.

Inventory Management System





Ex. No. 10 26/4/23

Create a data visualization for an inventory management system using JS.

Aim:

To create a data visualization for an inventory management system using Java script.

Procedure:

step-1: set up your HTML file.

=> First, create an HTML file to hold your canvas for the chart and include chart.js.

```
html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width = device-width,
                                   initial-scale = 1.0">
    <title>Inventory Management Visualization</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            text-align: center;
            margin: 50 px;
        }
        canvas{
            margin: 20px auto;
        }
    </style>
</head>
<body>
    <h1> Inventory Management System </h1>
    <canvas id = "pieChart" width = "400" height = "400"></canvas>
    <canvas id = "barChart" width = "400" height = "400"></canvas>
    <script src = "https://cdn.jsdelivr.net/npm/chart.js">
                                                    </script>
```

</body>
</html>

step-2: Create the javascript file for charts.
=> Next, create a JavaScript file (script.js) to handle the data visulization logic.

JS.

```
const inventoryData = {
    labels: ['Electronics', 'clothing', 'Home appliances',
                                        'Books', 'Toys'],
    datasets: [
        {
            label: 'Item in stock',
            data: [200, 150, 100, 80, 50],
            backgroundColor: [
                '#FF6384',
                '#36A2EB',
                '#FFCE56',
                '#4BC0C0',
                '#9966FF'
            ],
        }
    ]
};
const ctxPie = document.getElementById('pieChart').get
                                        context('2d');
const piechart = new Chart(ctxPie, {
    type: 'Pie',
    data: inventoryData,
    options: {
        responsive: true,
        title: {
            display: true,
            text: 'Inventory Distribution'
        }
    }
});
```

```
const ctxBar = document.getElementById('barChart').getContext('2d');
const barChart = new Chart(ctxBar, {
    type: 'bar',
    data: inventoryData,
    options: {
        responsive: true,
        title: {
            display: true,
            text: 'Items in Stock by Category'
        },
        scales: {
            yAxes: [{
                ticks: {
                    beginAtZero: true
                }
            }]
        }
    }
});
```

Result:

The above program executed successfully.