

**RAJALAKSHMI ENGINEERING COLLEGE AN AUTONOMOUS
INSTITUTION**

Affiliated to ANNA UNIVERSITY

Rajalakshmi Nagar, Thandalam, Chennai-602105



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CS23A34 - USER INTERFACE DESIGN LABORATORY ACADEMIC

YEAR:2024-2025 (EVEN)

INDEX

Reg. No : 2116230701312

Name :S SHRUTHI

Branch :CSE

Year/Section :II/D

LIST OF EXPERIMENTS

Experiment No:	Title	Tools
1	Design a UI where users recall visual elements (e.g., icons or text chunks). Evaluate the effect of chunking on user memory.	Figma.
2.	Develop and compare CLI, GUI, and Voice User Interfaces (VUI) for the same task and assess user satisfaction.	Python (Tkinter for GUI, Speech Recognition for VUI) / Terminal
3	A) Create a prototype with familiar and unfamiliar navigation elements. Evaluate ease of use with different user groups.	Proto.io
	B) Create a prototype with familiar and unfamiliar navigation elements. Evaluate ease of use with different user groups.	Wireflow
4	A) Conduct task analysis for an app (e.g., online shopping) and document user flows. Create corresponding wireframes.	Lucid chart (free tier)
	B) Conduct task analysis for an app (e.g., online shopping) and document user flows. Create corresponding wireframes.	Dia (open source).
5.	A) Simulate the lifecycle stages for UI design using the RAD model and develop a small interactive interface.	Axure RP
	B) Simulate the lifecycle stages for UI design using the RAD model and develop a small interactive interface.	OpenProj.

6.	Experiment with different layouts and color schemes for an app. Collect user feedback on aesthetics and usability.	GIMP (open source for graphics).
7.	A)Develop low-fidelity paper prototypes for a banking app and convert them into digital wireframes.	Pencil Project
	B)Develop low-fidelity paper prototypes for a banking app and convert them into digital wireframes.	Inkscape.
8.	A) Create storyboards to represent the user flow for a mobile app (e.g., food delivery app).	Balsamiq
	B) Create storyboards to represent the user flow for a mobile app (e.g., food delivery app).	OpenBoard
9.	Design input forms that validate data (e.g., email, phone number) and display error messages.	HTML/CSS, JavaScript (with Validator.js).
10.	Create a data visualization (e.g., pie charts, bar graphs) for an inventory management system.	Java Script

EXPNO:1

Date:25/01/2025

TITLE: Design a UI where users recall visual elements (e.g., icons or text chunks). Evaluate the effect of chunking on user memory

AIM: To design a UI where users recall visual elements (e.g., icons or text chunks). Evaluate the effect of chunking on user memory.

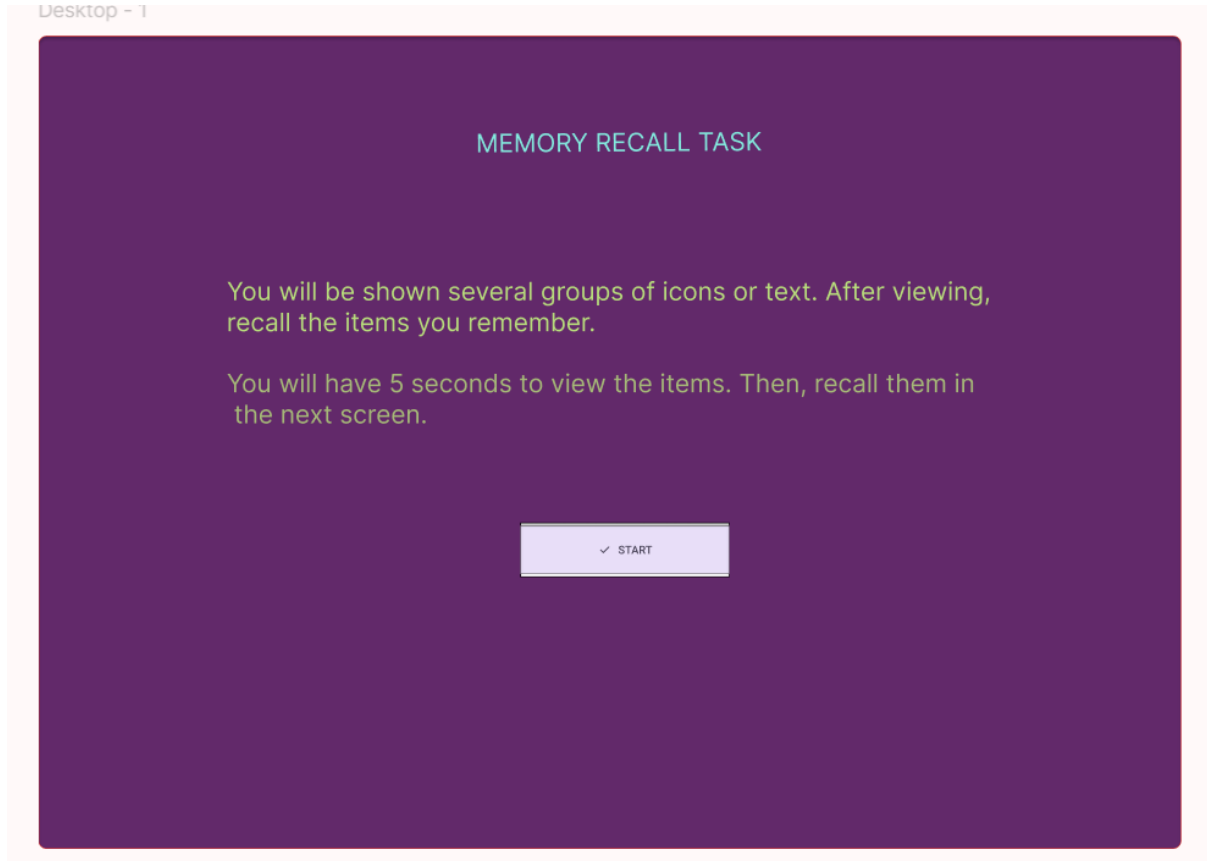
PROCEDURE:

1. Create Home Screen:
 - o Add a 1024x768px frame (File → New Frame).
 - o Insert a title ("Memory Recall Task") and instructions using the Text Tool (T).
 - o Design a "Start" button (Rectangle + Text) and link it to the Chunking Phase via Prototype mode.
2. Set Up Chunking Phase:
 - o Create a new frame for the chunking display.
 - o Add icons or text that users need to remember.
3. Apply Chunking Techniques:
 - o Chunking with Borders: Group 3-5 items using Rectangles (R).
 - o Chunking without Borders: Place items close together without clear separation.
4. Simulate Viewing Time:
 - o Select the Chunking Phase frame, go to Prototype mode, and set an "After Delay" transition (5000ms) to the Recall Phase.
5. Create Recall Phase UI:
 - o Add a new frame for user input.
 - o Add a question: "Select the items you remember seeing."
6. Design Recall Options:
 - o Multiple-choice method: Add checkboxes/radio buttons.
 - o Text input method: Create labeled text input fields (e.g., "Item 1").
7. Create Submit Button:
 - o Design a "Submit Recall" button (Rectangle + Text).
 - o Link it to the Result Screen in Prototype mode.
8. Create Result Screen:
 - o Add a title (e.g., "Your Recall Score") and feedback text (e.g., "You recalled 4/5 items!").
9. Provide Analysis:
 - o Test different chunk sizes (3 vs. 5 items) and content types (icons vs. text).

10. Final Testing & Sharing:

- Click Play to preview the prototype.
- Use the Share button to invite testers.

OUTPUT:



Desktop - 2

CHUNKING PHASE

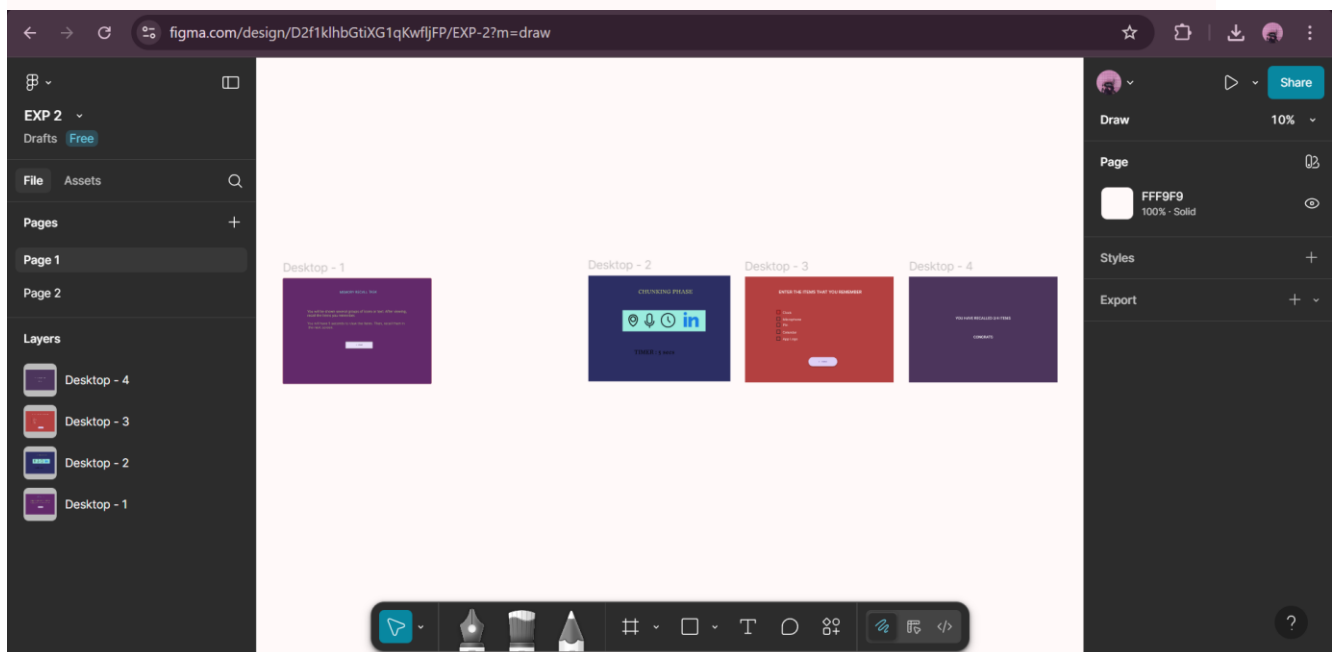
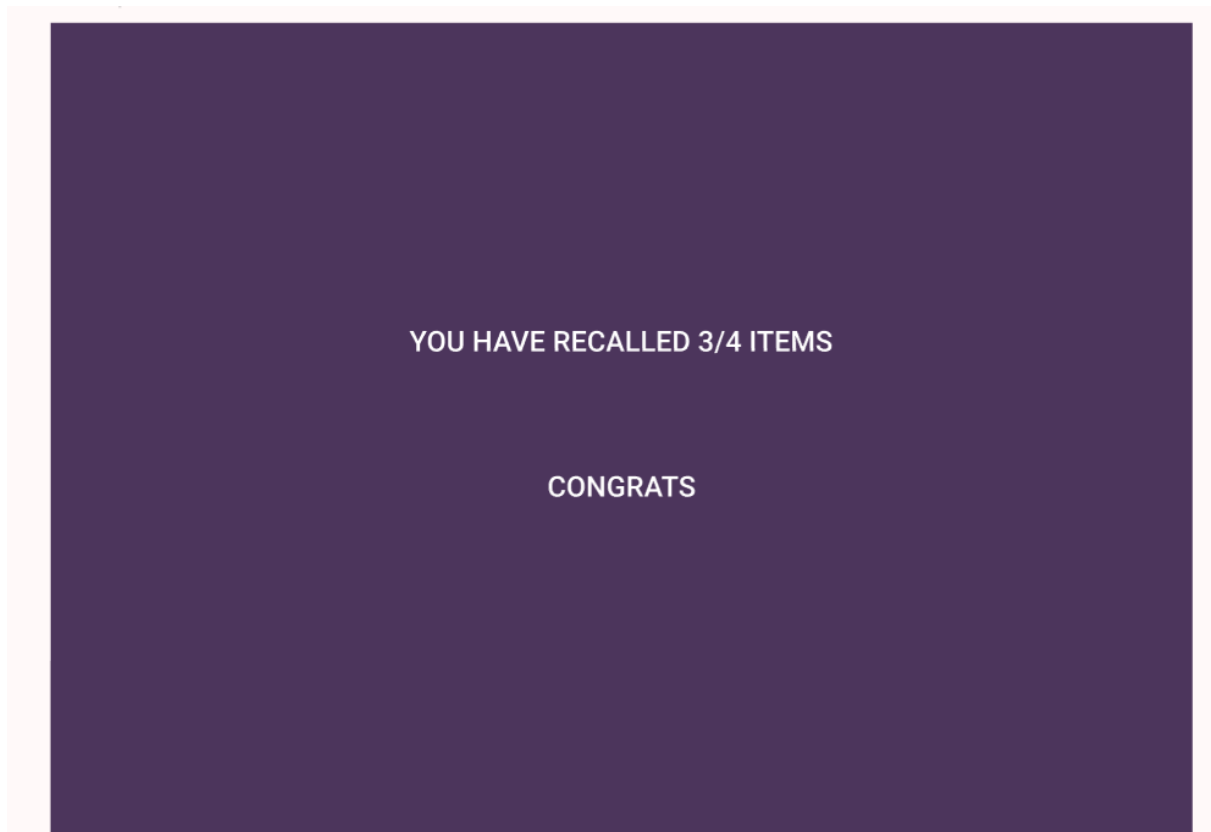


TIMER : 5 secs

ENTER THE ITEMS THAT YOU REMEMBER

- ☐ Clock
- ☐ Microphone
- ☐ Pin
- ☐ Calendar
- ☐ App Logo

> SUBMIT



LINK: <https://www.figma.com/proto/D2f1klhbGtiXG1qKwfljFP/EXP-2?node-id=12-77&p=f&t=SOgG9NJw9R33IjZP-1&scaling=min-zoom&content-scaling=fixed&page-id=0%3A1&starting-point-node-id=1%3A2>

RESULT:

The Memory Recall UI successfully tests chunking effects by displaying grouped icons/text, prompting recall, and providing feedback on user memory accuracy.

EXPNO:02

DATE:01/02/2025

TITLE: Develop and compare CLI, GUI, and Voice User Interfaces (VUI) for the same task and assess user satisfaction using Python (Tkinter for GUI, Speech Recognition for VUI), 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 (with Tkinter for GUI and Speech Recognition for VUI) and Terminal.

PROCEDURE:

1.CLI:

Step 1: Set Up the Python Environment

Ensure Python is installed on your system (python --version).

Create a new Python script file, e.g., todo.py.

Step 2: Define Core Functions

add_task(task) → Appends a task to a list or file.

view_tasks() → Displays all tasks with numbering.

remove_task(task_number) → Deletes a selected task.

save_tasks() → Stores tasks in a file (e.g., tasks.txt).

load_tasks() → Retrieves tasks from the file at startup.

Step 3: Implement User Interaction via CLI

Use input() to get user commands (e.g., "add", "view", "remove", "exit").

Implement a loop to continuously accept commands.

Use if-elif conditions to call respective functions.

Step 4: Handle File Storage (Optional but Recommended)

Store tasks in a text file (tasks.txt) for persistence.

Read and write tasks using open() in append/read mode.

Ensure error handling for file operations.

Step 5: Test and Run the Program

Run the script: `python todo.py`.

Add, view, and remove tasks to test functionality.

Improve with enhancements like colored output (colorama) or JSON storage.

2.GUI:

Step 1: Set Up the Python Environment

Ensure Python is installed (`python --version`).

Install Tkinter (built-in) or other GUI frameworks like PyQt (`pip install PyQt6`).

Step 2: Create the GUI Layout

Use Tkinter to create a main window.

Add widgets:

Entry Box for adding tasks.

Listbox for displaying tasks.

Buttons for Add, Remove, and Clear actions.

Step 3: Implement Task Management Functions `add_task()`

→ Adds a task from the entry box to the listbox.

`remove_task()` → Deletes the selected task.

`clear_tasks()` → Clears all tasks.

`save_tasks()` → Saves tasks in a file (`tasks.txt`).

`load_tasks()` → Loads tasks on startup.

Step 4: Handle Events & User Input

Bind buttons to respective functions (`command=add_task`).

Use Double-click events to remove tasks.

Step 5: Test & Run the GUI Application

Run `python gui_todo.py`.

Ensure all buttons and actions function correctly.

Enhance UI with ttk styling or migrate to PyQt for advanced design.

3.VUI:

Step 1: Install Dependencies

Install speechrecognition, pyttsx3, and pyaudio:

bash

CopyEdit

```
pip install speechrecognition pyttsx3 pyaudio
```

Step 2: Set Up Speech Recognition & Synthesis

Use speech_recognition to convert voice to text.

Use pyttsx3 for text-to-speech responses.

Step 3: Implement Voice-Controlled Functions listen_command()

→ Captures user voice and converts it to text. add_task(task) →

Adds a task from spoken input. remove_task(task_number) →

Removes a spoken task index. speak(text) → Provides audio feedback.

Step 4: Implement Command Processing Logic

Recognize voice commands like "Add task: Buy groceries" or "Remove task 2".

Use if-elif to process recognized commands and call respective functions.

Step 5: Test & Improve Recognition

Run python vui_todo.py.

Test various commands and fine-tune recognition accuracy.

Add NLP improvements using Google Speech API or Whisper AI for better accuracy.

PROGRAM:

CLI- Command line interface

```
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,1):
            print(f"{idx}.{task}")

    else:
        print("No tasks to show.")
def remove_task(task_number):
    if 0< task_number <= len(tasks):
        removed_task=tasks.pop(task_number-1)
        print(f"Task'{removed_task}'removed.")
    else:
        print("Invalid task number.")
def main():
    while True:
        print("\nOptions: 1. Add Task 2.View Tasks 3.Remove Task 4.Exit")
        choice=input("enter yooor choice:")

        if choice=='1.':
            task=input("Enter task: ")
            add_task(task)
        elif choice=='2.':
            view_tasks()
        elif choice == '3.':
            task_number=int(input("Enter task number to remove: "))
            remove_task(task_number)
        elif choice == '4.' :
            print("Exiting..")
            break
        else:
            print("Invalid choice. Please try again.")
if __name__=="__main__":
    main()

```

```

AMD64) on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/HDC0422041/Documents/CLI 312.py =====

Options: 1. Add Task 2.View Tasks 3.Remove Task 4.Exit
enter your choice:2.
No tasks to show.

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

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

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

Options: 1. Add Task 2.View Tasks 3.Remove Task 4.Exit
enter your choice:2.
Your tasks:
1.we
2.the
3.mee

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

Options: 1. Add Task 2.View Tasks 3.Remove Task 4.Exit
enter your choice:4.
Exiting..
>>>

```

GUI – Graphical User Interface

```

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 task in tasks:
        task_list.insert(tk.END, task)
def remove_task():
    selected_task_index = task_list.curselection()

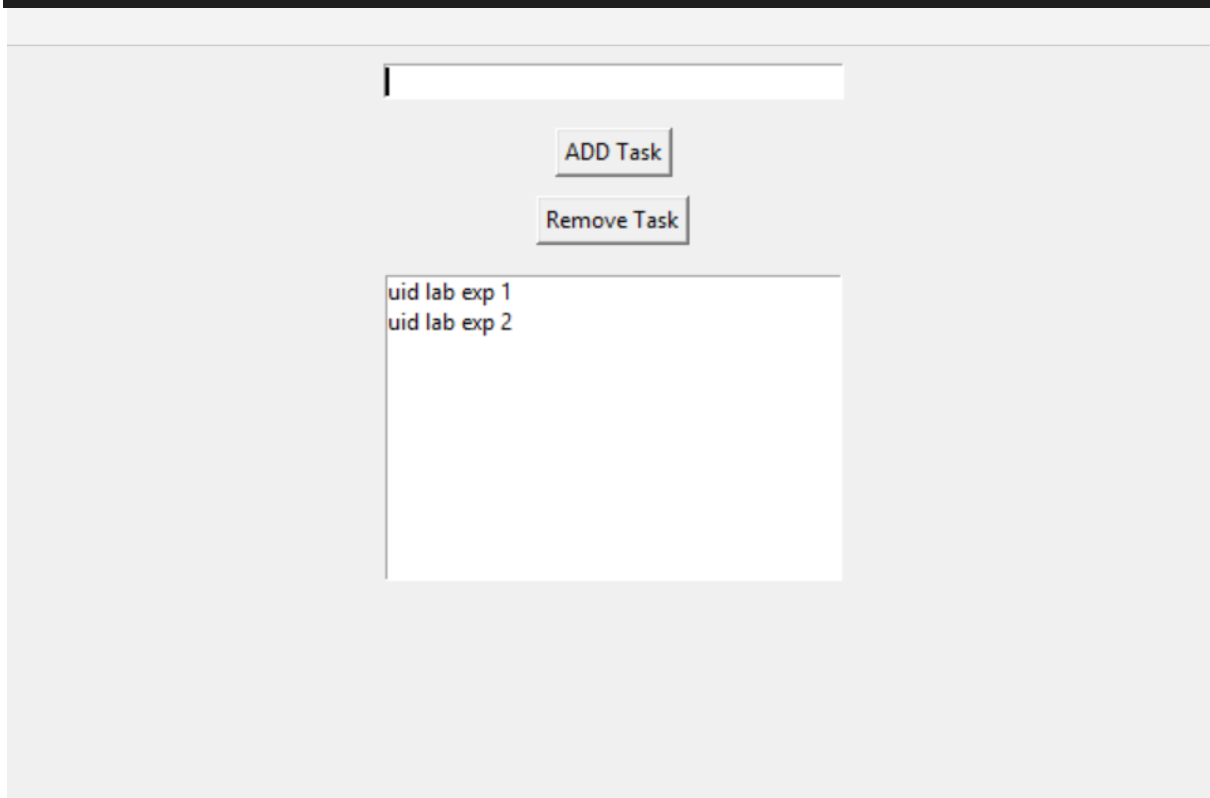
```

```

        if selected_task_index:
            task_list.delete(selected_task_index)
            tasks.pop(selected_task_index[0])
app = tk.Tk()
app.title("To-Do List")
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)
remove_button = tk.Button(app, text="Remove Task", command=remove_task)
remove_button.pack(pady=5)
task_list = tk.Listbox(app, width=40, height=10)

task_list.pack(pady=10)
app.mainloop()

```



VUI – Voice User Interface

```

import speech_recognition as sr
import pyttsx3

tasks = []
recognizer = sr.Recognizer()
engine = pyttsx3.init()

def add_task(task):
    tasks.append(task)
    engine.say(f"Task {task} added")

```

```
engine.runAndWait()

def view_tasks():
    if tasks:
        engine.say("Your tasks are")
        for task in tasks:
            engine.say(task)
    else:
        engine.say("No tasks to show")
    engine.runAndWait()

def remove_task(task_number):
    if 0 < task_number <= len(tasks):
        removed_task = tasks.pop(task_number - 1)
        engine.say(f"Task {removed_task} removed")
    else:
        engine.say("Invalid task number")
    engine.runAndWait()

def recognize_speech():
    with sr.Microphone() as source:
        print("Listening...")
        audio = recognizer.listen(source)
        try:
            command = recognizer.recognize_google(audio)
            return command
        except sr.UnknownValueError:
            engine.say(";Sorry, I did not understand that")
            engine.runAndWait()
            return None

def main():
    while True:
        engine.say("Options: add task, view tasks, remove task, or exit")
        engine.runAndWait()
        command = recognize_speech()
        if not command:
            continue
        if "add task" in command:
            engine.say("What is the task?")
            engine.runAndWait()
            task = recognize_speech()
            if task:
```

```
        add_task(task)
    elif "view tasks" in command:

        view_tasks()
    elif "remove task" in command:
        engine.say(";Which task number to remove?")
        engine.runAndWait()

task_number = recognize_speech()
    if task_number:
        remove_task(int(task_number))
    elif "exit" in command:
        engine.say(";Exiting...")
        engine.runAndWait()
        break
    else:
        engine.say(";Invalid option. Please try again.")
        engine.runAndWait()
if __name__ == "__main__":
    main()
```

Listening...

(Recognized: "add task")

Listening...

(Recognized: "Buy groceries")

Task Buy groceries added

Listening...

(Recognized: "add task")

Listening...

(Recognized: "Finish project report")

Task Finish project report added

Listening...

(Recognized: "view tasks")

Your tasks are

Buy groceries

Finish project report


```
Listening...
(Recognized: "remove task")
Listening...
(Recognized: "1")
Task Buy groceries removed

Listening...
(Recognized: "view tasks")
Your tasks are
Finish project report

Listening...
(Recognized: "exit")
Exiting...
```

RESULT:

User satisfaction varies based on familiarity—CLI is fast for experienced users, GUI is intuitive for general users, and VUI offers hands-free convenience but may have recognition limitations.

EXPNO:3A

DATE:08/02/25

**TITLE:Create a prototype with familiar and unfamiliar navigation elements.
Evaluate ease of use with 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 groups using Proto.io.

PROCEDURE:

Step 1: Sign Up & Log In

Go to proto.io

Sign up or log in

Step 2: Create a New Project

Click "Create New Project"

Enter project name (e.g., "Simple App Example")

Select device type (e.g., iPhone X)

Click "Create"

Step 3: Design Home Screen

Add Screen: Click "+" → Select "Blank" → Name it "Home"

Add Elements:

Drag "Header" → Edit text to "Home Screen"

Drag "Button" → Edit text to "Go to Profile"

Add Interaction:

Select button → "Interactions" tab → "+ Add Interaction"

Trigger: Tap/Click, Action: Navigate to Screen → Create "Profile" screen

Step 4: Design Profile Screen

Add Elements:

Drag "Header" → Edit text to "Profile Screen"

Drag "Image" → Upload profile picture

Drag "Text" → Add profile info (e.g., "John Doe, Software Engineer")

Add Back Button:

Drag "Button" → Edit text to "Back to Home"

Add Interaction:

Select button → "Interactions" tab → "+ Add Interaction"

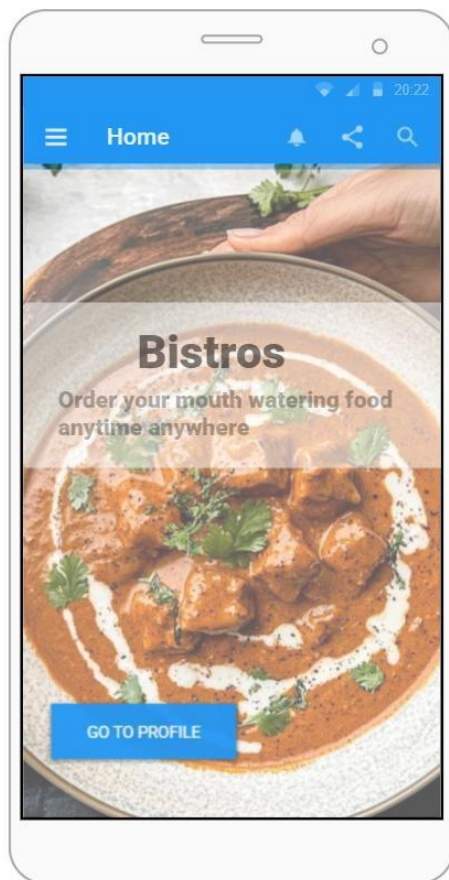
Trigger: Tap/Click, Action: Navigate to Home

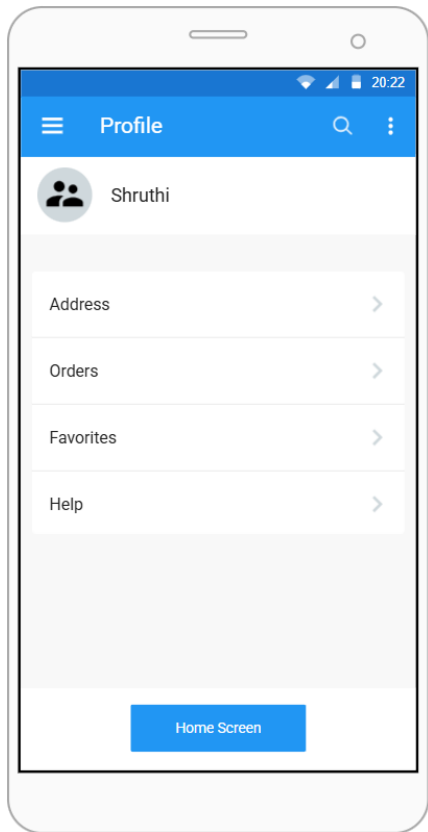
Step 5: Preview & Share

Preview: Click "Preview" to test navigation

Share: Click "Share" → Copy link → Share for feedback

OUTPUTS:





RESULT:

Successfully created an interactive prototype with a Home and Profile screen, implemented navigation using buttons, tested interactions using proto.io.

Link: <https://pr.to/2GSLJI/>

EXP NO: 3B

DATE:15/03/2025

TITLE: Create a prototype with familiar and unfamiliar navigation elements. Evaluate ease of use with different user groups using wireflow.

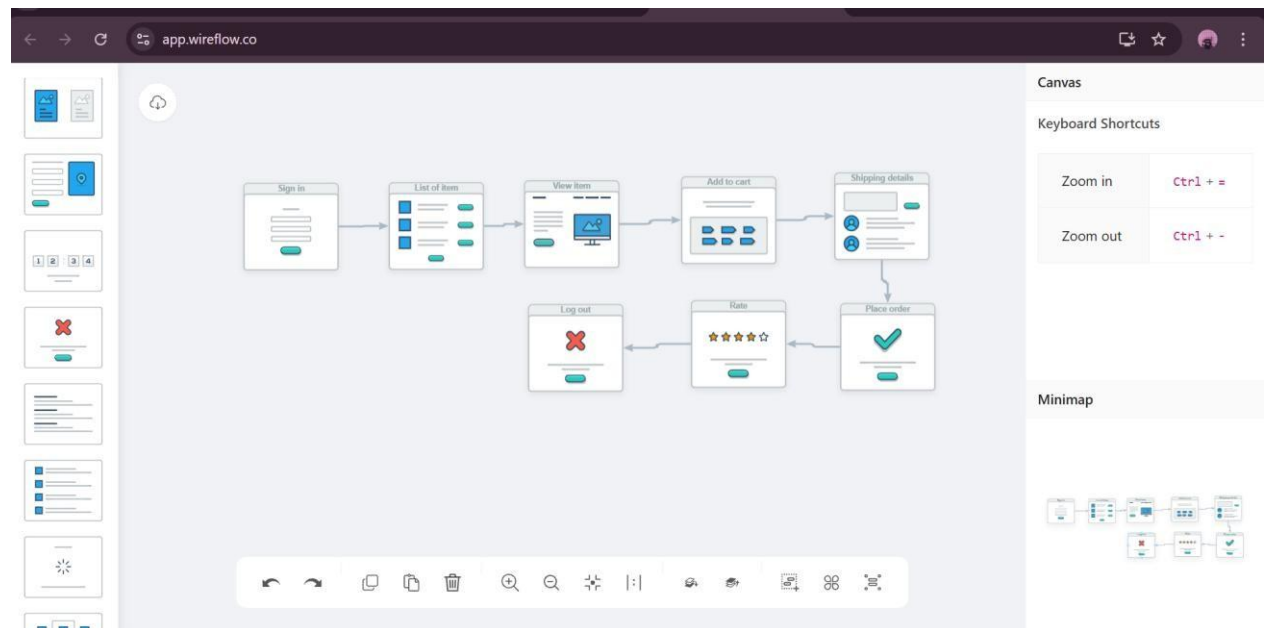
AIM:

The aim is to design a prototype with both well-known and new navigation elements and measure user-friendliness across different user groups using Wireflow.

PROCEDURE:

1. Define the navigation elements by deciding which ones will be familiar (like top bars and menus) and which will be unfamiliar (such as hidden menus or gesture controls).
2. Sketch the layout of your app using paper or digital tools like Figma to visualize the design and user flow.
3. Sign up or log in to Wireflow, then start a new project by naming it and choosing a blank canvas or template.
4. Design your prototype by adding familiar UI components and creatively incorporating unfamiliar navigation elements.
5. Link the screens using Wireflow's tools to simulate how users will navigate between different parts of the app.
6. Identify your target user groups, recruit participants online, and share your prototype link with them for usability testing.
7. Collect and analyze feedback from the test sessions, compare user.

OUTPUT:



RESULT: The prototype with both familiar and innovative navigation elements was successfully designed using Wireflow.

EXPNO: 4a

DATE:15/03/2025

TITLE:Conduct task analysis for an app (e.g., online shopping) and document user flows. Create corresponding wireframes using Lucid chart

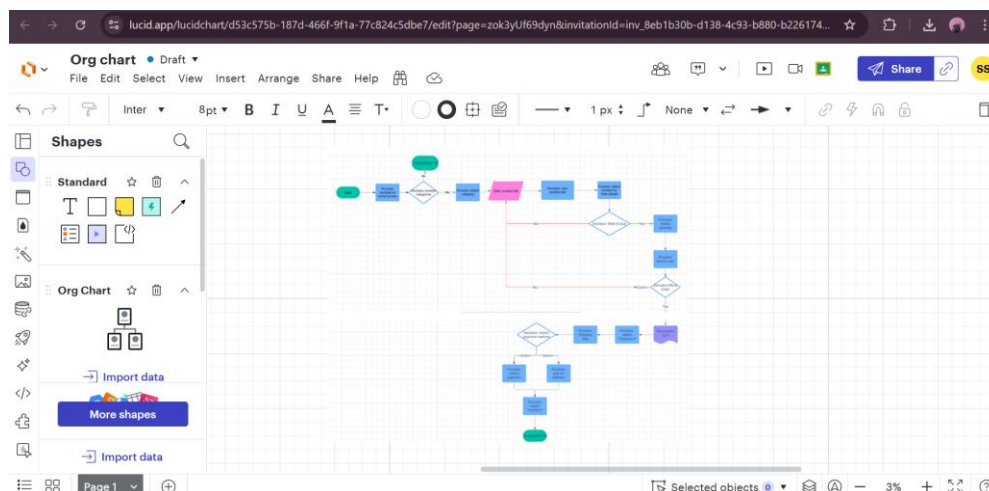
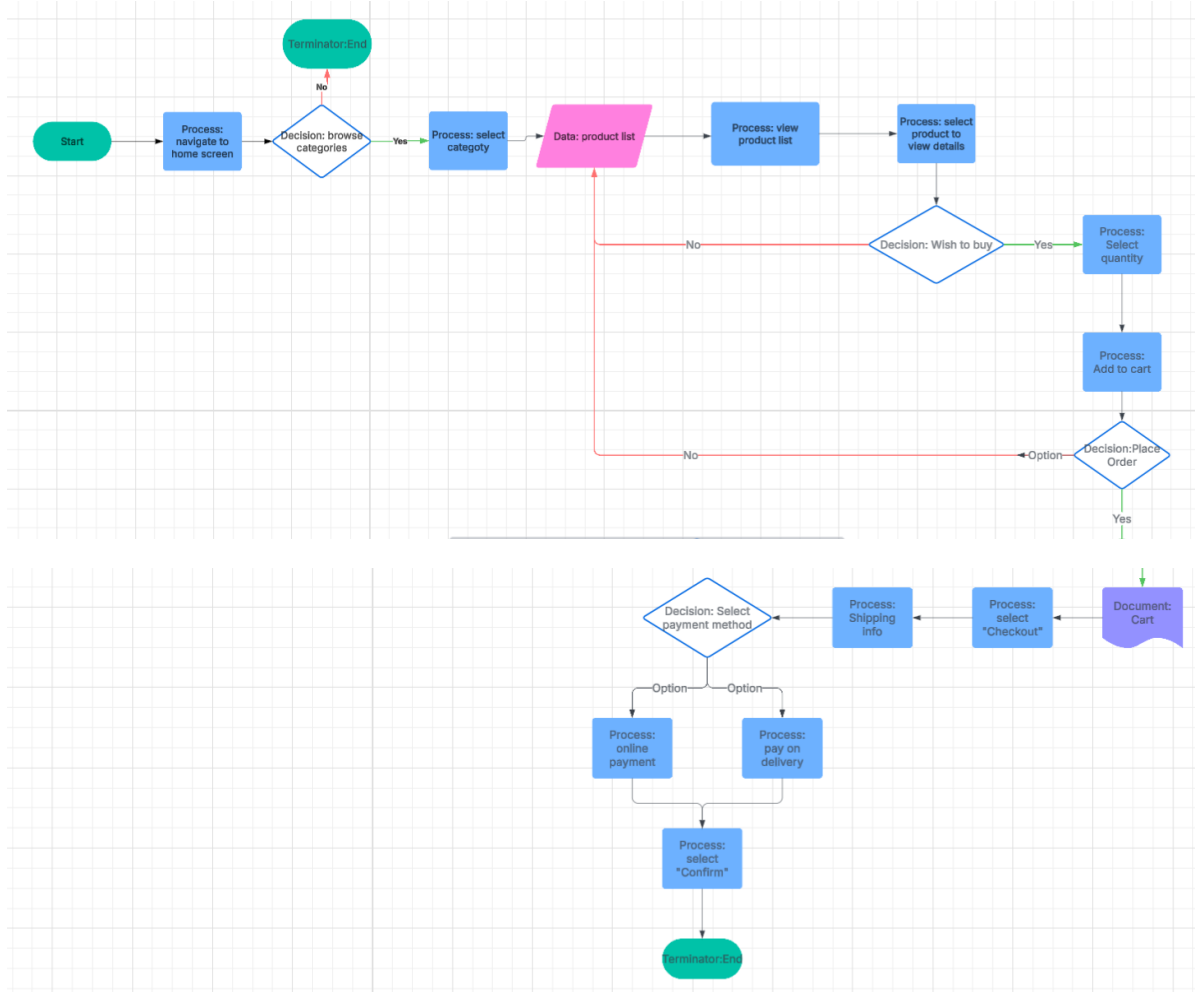
AIM:

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

Procedure:

1. Create a New Document – Sign in to Lucidchart, click *Create New Diagram*.
2. Select a Template – Choose a blank document or a flowchart template.
3. Add Shapes – Drag rectangles for actions, diamonds for decisions, and ovals for start/end points.
4. Connect Shapes – Use arrows to show the sequence of steps.
5. Label Steps – Add text inside shapes describing each user action.
6. Customize Flowchart – Use colors, align elements, and group steps logically.
7. Review & Save – Ensure correctness, click *File → Save*.
8. Share & Export – Click *Share* for collaboration or export as an image/PDF.

OUTPUT:



Result: A structured flowchart visualizing user flows for browsing, searching, adding to cart, checkout, and order tracking.

LINK: https://lucid.app/lucidchart/f048c719-a6e3-42d4-bfac-ca40f11ded49/edit?invitationId=inv_d520bc66-0f6b-4ce4-9626-4e892ac49207

EXP No : 4b

Date: 20 / 03 / 2025

TITLE: Conduct task analysis for an app and document user flows. Create corresponding wireframes 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:

1. Install Dia:

Download Dia from <http://dia-installer.de/>

Install and launch the application.

2. Create a New Diagram:

Go to File → New Diagram

Select Flowchart as the diagram type.

3. Add Shapes:

Use rectangles, ellipses, and diamonds to represent different stages:

Start → Oval

Login/Register → Rectangle

Search Tickets → Rectangle

Select Date, Time, Preferences → Rectangle

Check Availability → Rectangle

Decision (Tickets Available?) → Diamond

Payment Process → Rectangle

Generate Confirmation → Rectangle

Send Ticket via Email/SMS → Rectangle

End → Oval

4. Connect Shapes:

Use lines to define user flow:

Start → Login/Register

Login/Register → Search Tickets

Search Tickets → Select Date, Time, Preferences

Select Date, Time, Preferences → Check Availability

Check Availability → (Decision: Tickets Available?)

Yes → Proceed to Payment

No → Show Error & End

Proceed to Payment → Generate Confirmation → Send Ticket via Email/SMS
→ End

5. Label Shapes:

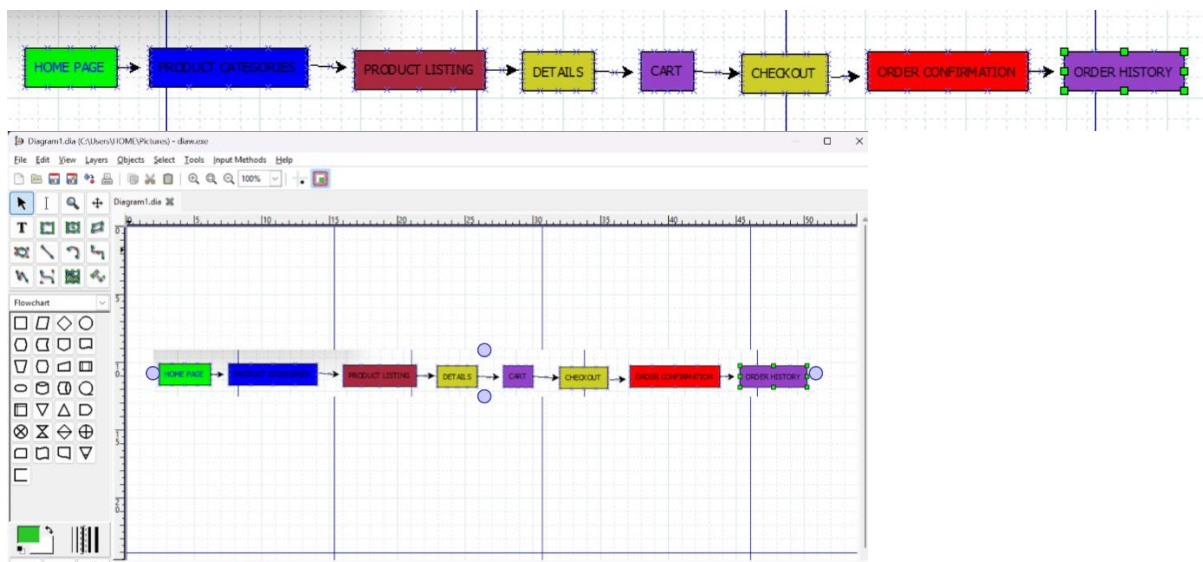
Double-click each shape to add labels like Login, Search, Payment, Confirm Ticket, etc.

6. Save the Diagram:

Go to File → Save As

Name it "Ticket Booking System Flowchart".

OUTPUT:



RESULT:

A wireframe is created for ticket booking system in Dia by adding and connecting shapes for login, search, seat selection, payment, and ticket generation.

EXPNo:5A

DATE:07/04/2025

TITLE:Simulate the lifecycle stages for UI design using the 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:

Phase 1: Requirements Planning

1. Identify Key Features:
 1. Navigation: Home, Product Categories, Product Details, Cart, Checkout, Order Confirmation, Order History
 2. User Actions: Browsing, Searching, Adding to Cart, Checkout, Tracking Orders
2. Create Requirements Document:
 1. List out functionalities and user flow.
 2. Define User Stories (e.g., *As a user, I want to view product details after clicking a product*).
 3. Document Use Cases for each user action.

Phase 2: User Design

1. Install and Launch Axure RP:
 1. Download and install from the official website.
 2. Launch the application.
2. Create New Project:
 1. Go to File → New.
 2. Name the project as *Shopping App Interface*.
3. Create Wireframes:
 1. Drag UI components from the widget library.
 2. Design wireframes for:
 1. Home Page
 2. Product Categories
 3. Product Listings
 4. Product Details
 5. Cart
 6. Checkout
 7. Order Confirmation
 8. Order History
4. Add Interactions:
 1. Select a button → Go to Properties → Add OnClick interaction.

2. Link navigation from one page to another.
5. Create Masters:
 1. Create reusable components like Header, Footer.
 2. Use them across multiple pages.
6. Add Annotations:
 1. Add descriptive notes to each element using the Notes panel.

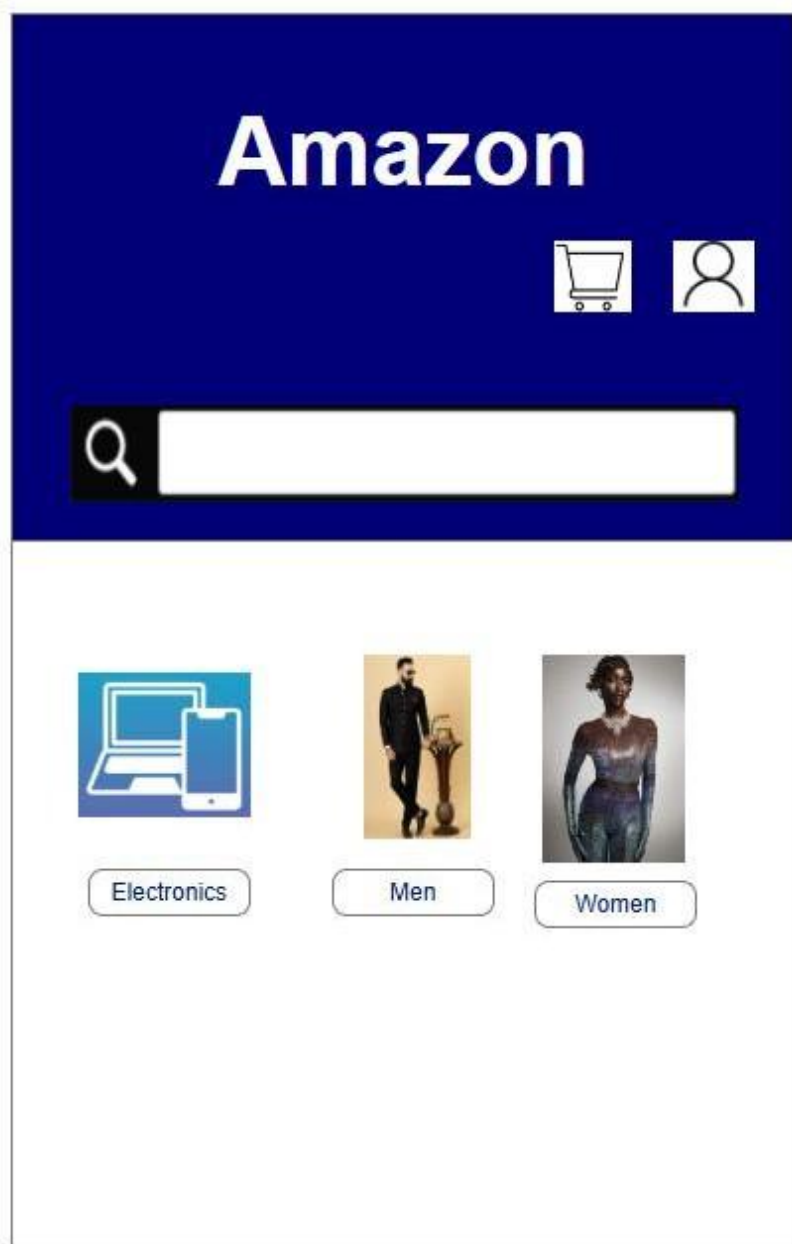
Phase 3: Construction

1. Develop Interactive Prototypes:
 1. Convert wireframes into clickable prototypes.
 2. Use dynamic panels for carousels/popups.
2. Test and Iterate:
 1. Use the Preview button to test.
 2. Share with stakeholders and collect feedback.
 3. Refine the design based on feedback.

Phase 4: Cutover

1. Finalize and Export:
 1. Finalize interactions and visual elements.
 2. Export to HTML or share via Axure Cloud link.
2. User Training and Support:
 1. Conduct demo/training for users.
 2. Provide user documentation for reference.

OUTPUT:



Amazon

Women's Fashion



Dress XY

Sizes :

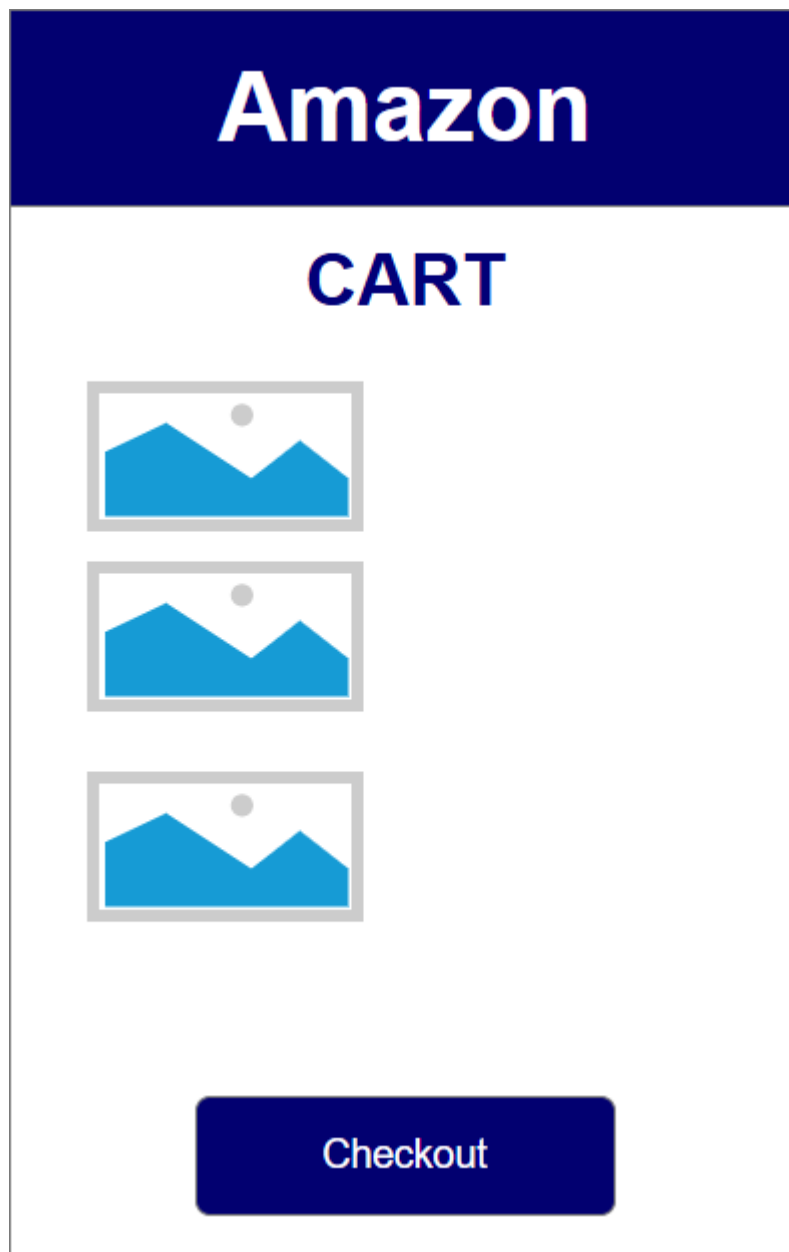
S



Buy now

Add to cart

View Cart



RESULT:

The UI for the Shopping App was successfully designed using the RAD model. Interactive wireframes were developed in Axure RP and shared with stakeholders. Feedback was incorporated through iterations. The final prototype was deployed via Axure Cloud, and users were trained to interact with the system.

EXP No: 5b

Date:07/04/2025

TITLE:Simulate the life cycle stages for UI design using the RAD model and develop a small interactive interface using OpenProj

AIM:

The aim is to recreate the lifecycle stages of UI design using the RAD model and design a small interactive interface with OpenProj.

PROCEDURE:

Step 1: Requirements Planning

Gather Requirements

Identify features: Login and Register screens with debug logs.

Define Use Cases

Example Use Cases:

User logs in with valid credentials

User registers with a new account

Step 2: User Design

Sketch Initial Designs

Create rough paper sketches of the Login and Register screens.

Create Digital Wireframes

Use Figma, Sketch, or any wireframing tool to design the UI.

Example Screens:

Login Screen: Username, Password fields, Login button, Register link

Register Screen: Username, Email, Password, Confirm Password fields, Register button

Step 3: Rapid Prototyping

Develop Prototypes

Convert wireframes into clickable, interactive prototypes using tools like Axure RP.

Test Prototypes

Share with stakeholders, gather feedback, and refine the design.

Step 4: User Acceptance/Testing

Review Prototype

Conduct reviews with stakeholders to assess functionality and design.

Conduct Usability Testing

Perform tests with users to evaluate ease of use and collect improvement suggestions.

Step 5: Implementation

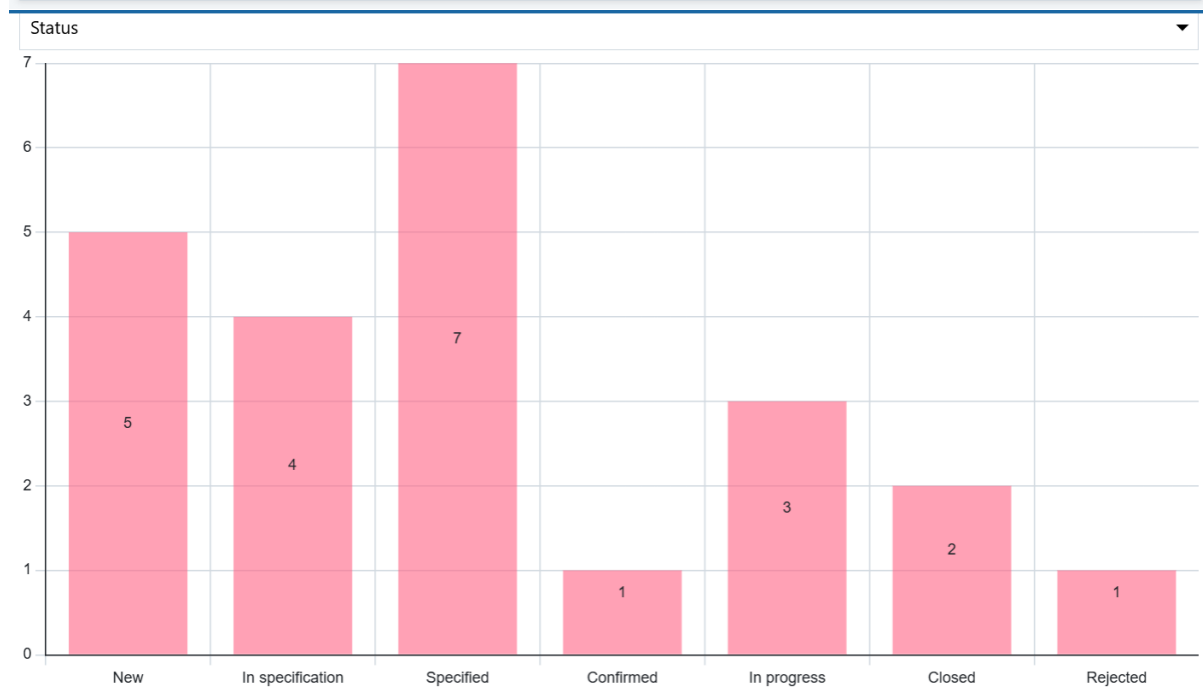
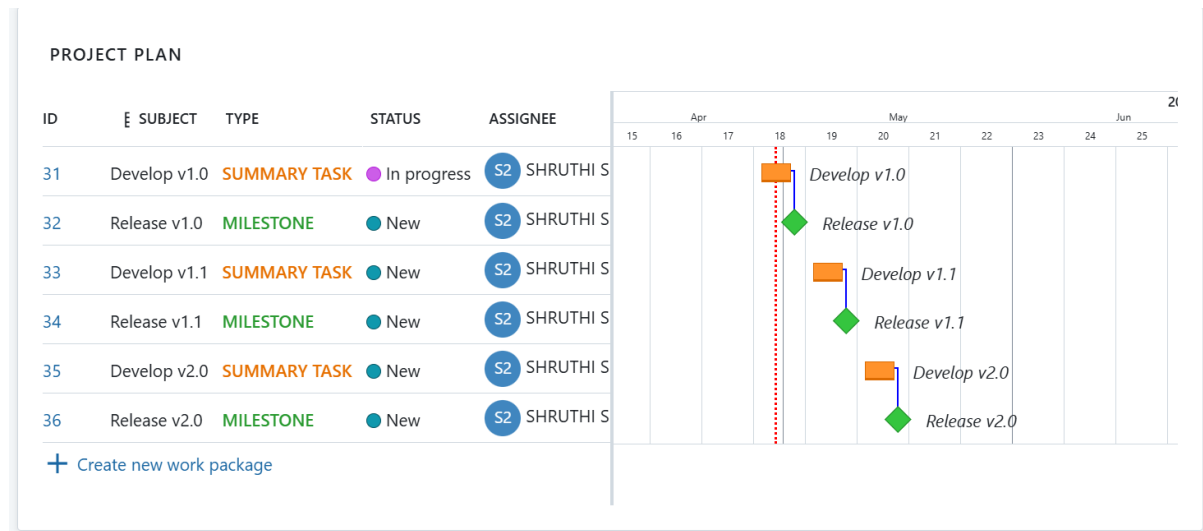
Develop Functional Interface

Build the final Login/Register interface using the approved design.

Integrate Backend (if required)

Connect to backend services for login/authentication.

OUTPUT:



RESULT:

Successfully designed and implemented a simple interactive Login/Register interface using the RAD model using OpenProj.

EXP NO: 6

DATE:07/04/2025

**TITLE:Experiment with different layouts and color schemes for an app.
Collect user feedback on aesthetics and usability using GIMP(GNU Image
Manipulation Program (GIMP))**

AIM:

The aim is to trial different app layouts and color schemes and evaluate user feedback on aesthetics and usability using GIMP.

PROCEDURE:

Tool Link: <https://www.gimp.org/>

Step 1: Install GIMP

1. Download and Install:

- Download GIMP from GIMP Downloads and install it on your computer.

Step 2: Create a New Project

1. Open GIMP:

- 1. Launch the GIMP application.

2. Create a New Canvas:

- 1. Go to File -> New to create a new project.
- 2. Set the dimensions for your app layout (e.g., 1080x1920 pixels for a standard mobile screen).

Step 3: Design the Base Layout

1. Create the Base Layout:

- 1. Use the Rectangle Select Tool to create sections for different parts of your app (e.g., header, content area, footer).
- 2. Fill these sections with basic colors using the Bucket Fill Tool.

Example Output: A base layout with defined sections for header, content, and footer.

2. Add UI Elements:

- 1. Text Elements: Use the Text Tool to add text elements like headers, buttons, and labels.
- 2. Interactive Elements: Use the Brush Tool or Shape Tools to draw buttons, input fields, and other interactive elements.

Example Output: A layout with labeled sections and basic UI elements.

3. Organize Layers:

1. Use layers to separate different UI elements. This allows you to easily modify or experiment with individual components.
2. Name each layer according to its content (e.g., Header, Button1, InputField).

Step 4: Experiment with Color Schemes

1. Create Color Variants:

1. Duplicate Layout: Duplicate the base layout by right-clicking on the image tab and selecting Duplicate.
2. Change Colors: Use the Bucket Fill Tool or Colorize Tool to change the colors of the UI elements in each duplicate.

Example Output: Multiple color variants of the same layout.

2. Save Each Variant:

1. Save each color variant as a separate file (e.g., Layout1.png, Layout2.png, etc.).
2. Go to File -> Export As and choose the file format (e.g., PNG).

Step 5: Collect User Feedback

1. Prepare a Feedback Form:

1. Create Form: Create a feedback form using tools like Google Forms or Microsoft Forms.
2. Include Questions: Include questions about the aesthetics and usability of each layout and color scheme.

2. Share the Variants:

1. Distribute Files: Share the image files of the different layouts and color schemes with your users.
2. Provide Instructions: Provide clear instructions on how to view each variant and how to fill out the feedback form.

3. Gather Feedback:

1. Collect responses from users regarding their preferences and suggestions.
2. Analyze the feedback to determine which layout and color scheme are most preferred.

Step 6: Iterate and Refine

1. Refine the Design:

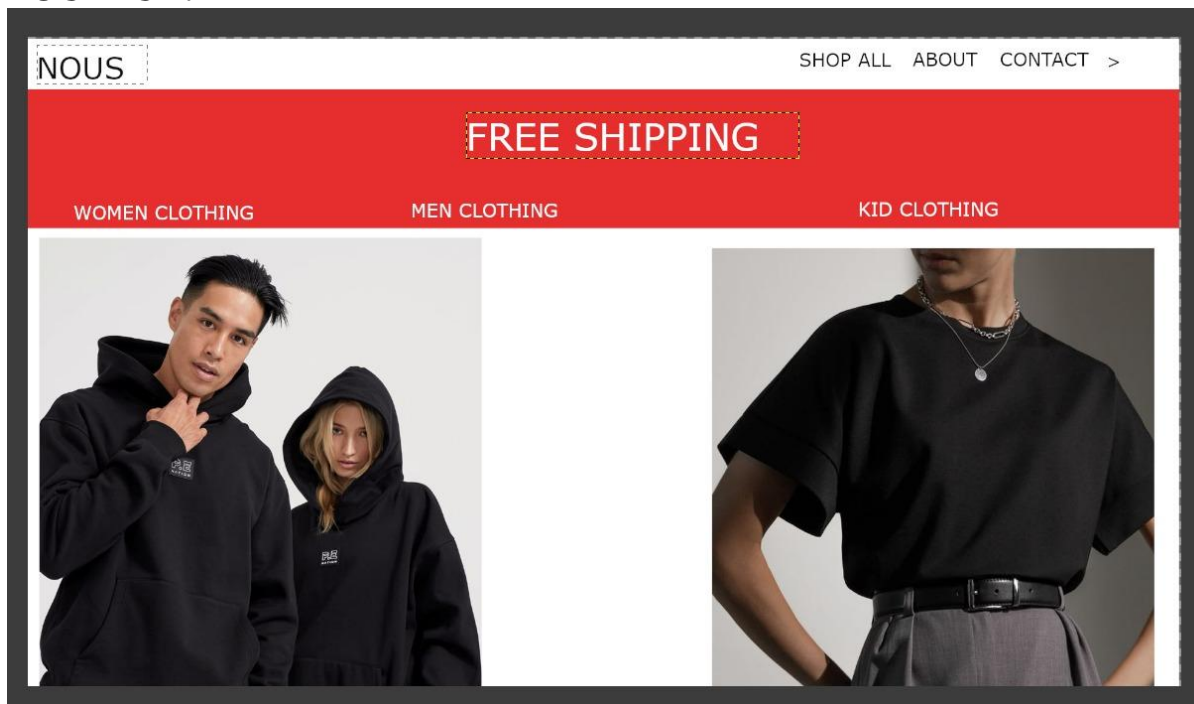
1. Based on the feedback, make necessary adjustments to the layout and color scheme.

2. Experiment with additional variations if needed.

2. Final Testing:

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

OUTPUT:



RESULT:

Trial on different app layouts and color schemes and evaluate user feedback on aesthetics and usability using GIMP is implemented.

Exp No: 7A

DATE:07/04/2025

TITLE:Develop low-fidelity paper prototypes for a banking app and convert them into digital wireframes using Pencil Project

AIM:

The aim is to develop low-fidelity paper prototypes for a banking app and convert them into digital wireframes with Pencil Project.

PROCEDURE:

Tool Link: <https://pencil.evolus.vn/>

Step 1: Create Low-Fidelity Paper Prototypes

1. Define the Purpose and Features:

- Identify the core features of the banking app (e.g., login, account balance, transfers, bill payments).

2. Sketch Basic Layouts:

- Use plain paper and pencils to sketch basic screens.
- Focus on primary elements like buttons, menus, and forms.

3. Iterate and Refine:

- Get feedback from users or stakeholders.
- Iterate on your sketches to improve clarity and functionality.

Step 2: Convert Paper Prototypes to Digital Wireframes Using Pencil Project

1. Install Pencil Project:

- Download and install Pencil Project from the official website.

2.Create a New Document:

- Open Pencil Project and create a new document.

3.Add Screens:

- Click on the "Add Page" button to create different screens (e.g., Login, Dashboard, Transfer).

2. Use Stencils and Shapes:

- Use the built-in stencils and shapes to create UI elements.
- Drag and drop elements like buttons, text fields, and icons onto your canvas.

3. Organize and Align:

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

4. Link Screens:

- Use connectors to link different screens together.
- Create navigation flows to show how users will interact with the app.

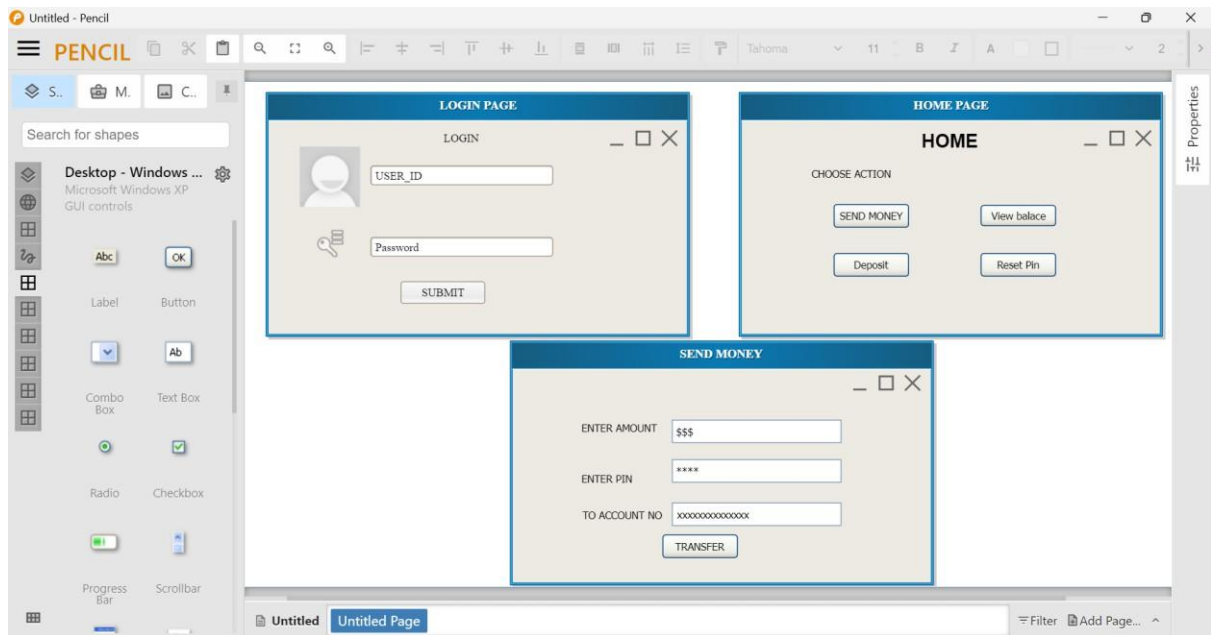
5. Add Annotations:

- Include annotations to explain the functionality of different elements.

6. Export Your Wireframes:

- Once satisfied with your digital wireframes, export them in your preferred format (e.g., PNG, PDF).

OUTPUT:



RESULT: A low-fidelity paper prototypes for a banking app are developed and converted them into digital wireframes with Pencil Project.

EXP No 7B

DATE:19/04/2025

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

AIM:

The aim is 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

1. Identify Core Features:

- Determine the essential features of the banking app (e.g., login, dashboard, account management, transfers).

2. Sketch Basic Layouts:

- Use plain paper and pencils to sketch the main screens.
- Focus on the primary elements like buttons, navigation menus, and input fields.

3. Iterate and Refine:

- Get feedback from users or stakeholders.
- Make necessary adjustments to improve clarity and functionality.

Step 2: Convert Paper Prototypes to Digital Wireframes Using Inkscape

1. Install Inkscape:

- Download and install Inkscape from the official website.

2. Create a New Document:

- Open Inkscape and create a new document by clicking on File > New.

3. Set Up the Document:

- Set the dimensions and grid for your design. Go to File > Document Properties to adjust the size.
- Enable the grid by going to View > Page Grid.

4. Draw Basic Shapes:

- Use the rectangle and ellipse tools to draw the basic shapes for your UI elements (e.g., buttons, input fields, icons).

5. Add Text:

- Use the text tool to add labels and placeholder text to your elements.

6. Organize and Align:

- Arrange and align the elements to match your paper prototype.

- Use the alignment and distribution tools to keep everything organized.

7. Group Elements:

- Select related elements and group them together using Object > Group.
- This helps keep your design organized and easy to edit.

8. Create Multiple Screens:

- Duplicate your base layout to create different screens (e.g., login, dashboard, transfer).
- Use Edit > Duplicate to create copies of your elements and arrange them for each screen.

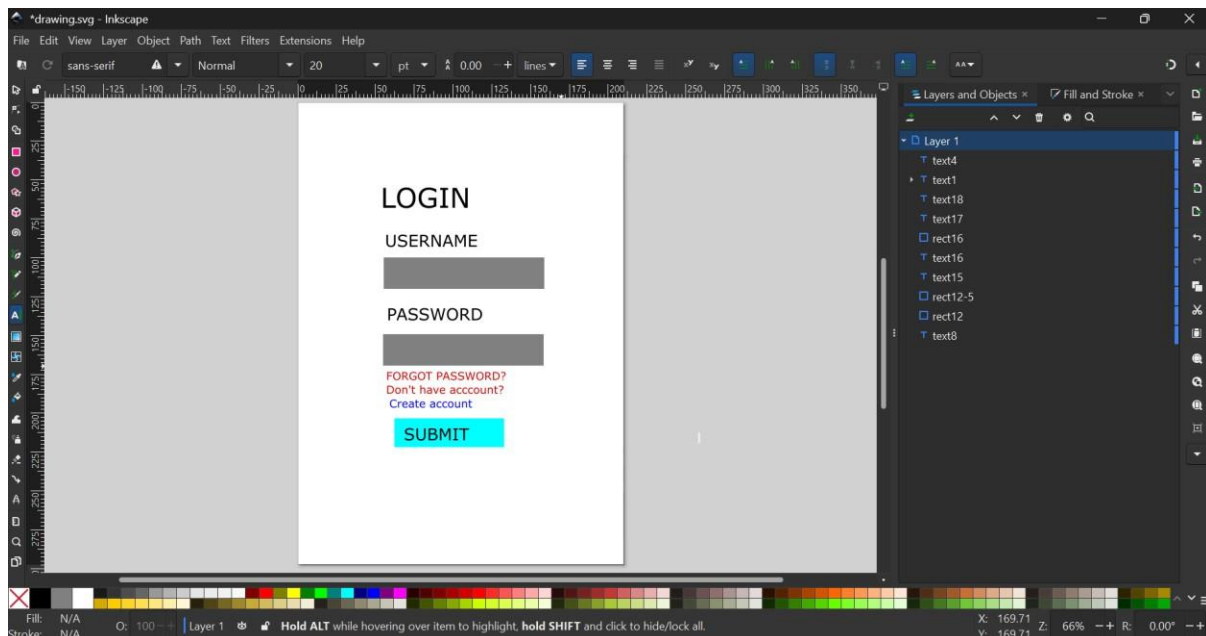
9. Link Screens (Optional):

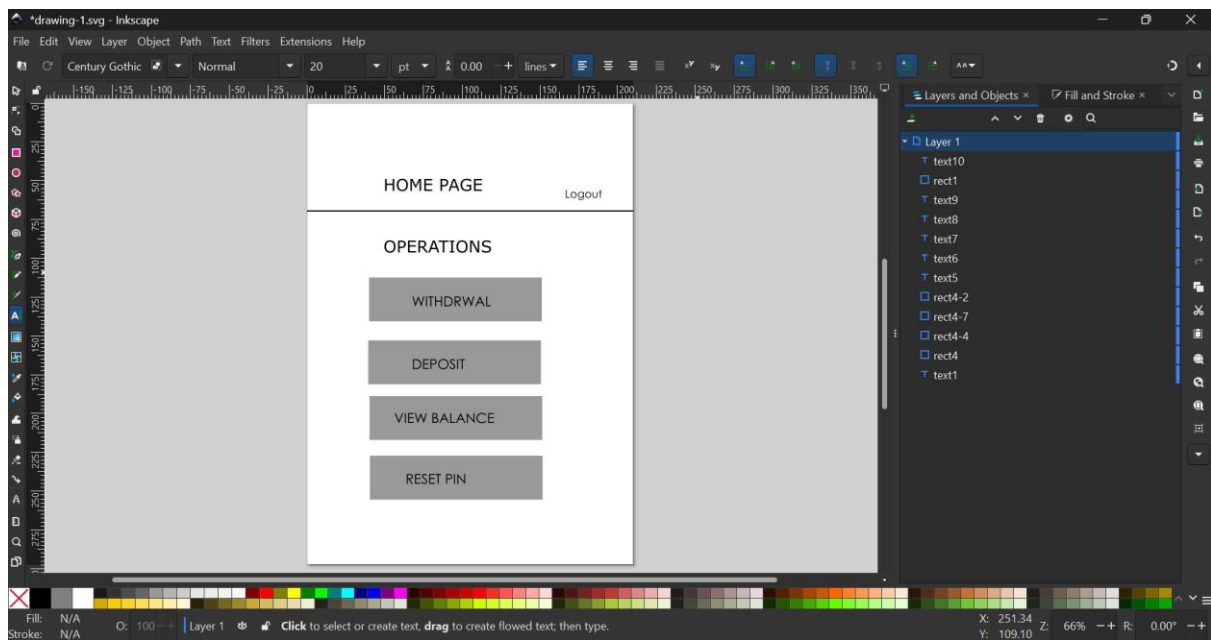
- If you want to show navigation flows, you can add arrows or other indicators to demonstrate how users will move between screens.

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.

OUTPUT:





RESULT: A low-fidelity paper prototypes for a banking app is developed and digitized into wireframes using Inkscape.

EXP No: 8A

DATE:19/04/2025

TITLE:Create storyboards to represent the user flow for a mobile app (e.g., food delivery app) using Balsamiq

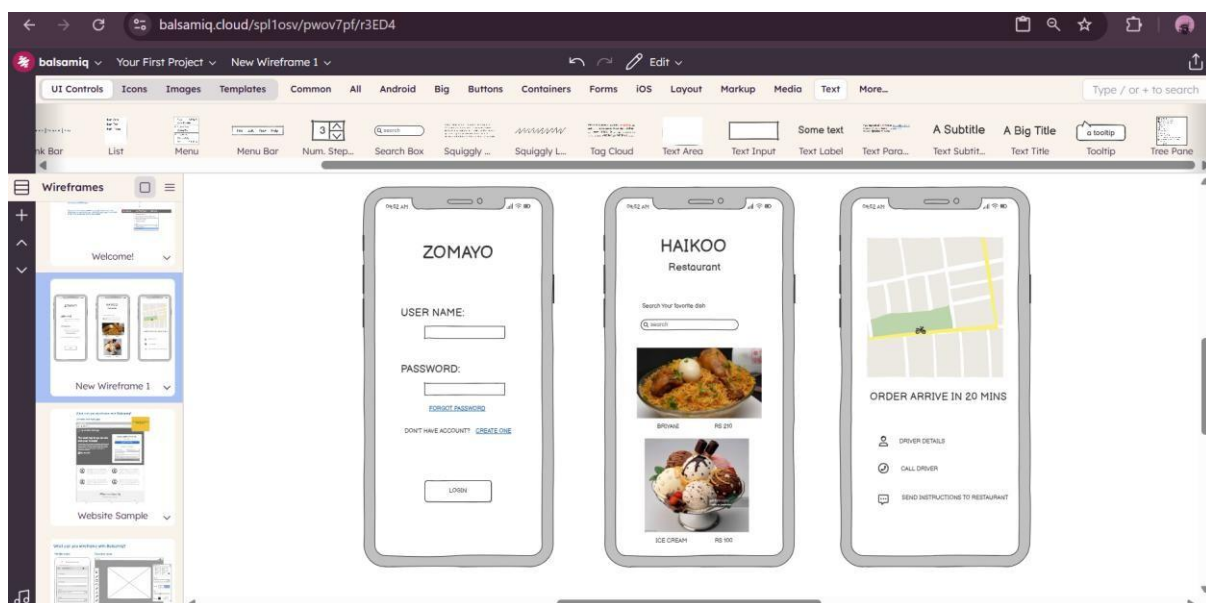
AIM:

The aim is to create storyboards representing the user flow for a mobile app, such as a food delivery app, using Balsamiq.

PROCEDURE :

1. Define User Flow
 - Key Screens: Home, Menu, Cart, Checkout, Order Confirmation
 - User Journey: Search → Browse Menu → Add to Cart → Checkout → Confirm Order
2. Create Storyboards in Balsamiq
 - Install Balsamiq from <https://balsamiq.com>
 - Create a new project
 - Add wireframe screens for each key screen
 - Design UI using Balsamiq components (buttons, images, text fields)
 - Arrange screens in logical order and connect them with arrows

OUTPUT:



RESULT:

Storyboards created using Balsamiq clearly represent the user flow of a food delivery app, showcasing all key screens and interactions from searching restaurants to order confirmation.

Exp No: 8B

DATE:19/04/2025

TITLE:Create storyboards to represent the user flow for a mobile app (e.g., food delivery app) using OpenBoard

AIM:

To map out the user flow for a mobile app (e.g., a food delivery app), storyboards will be designed using OpenBoard.

PROCEDURE:

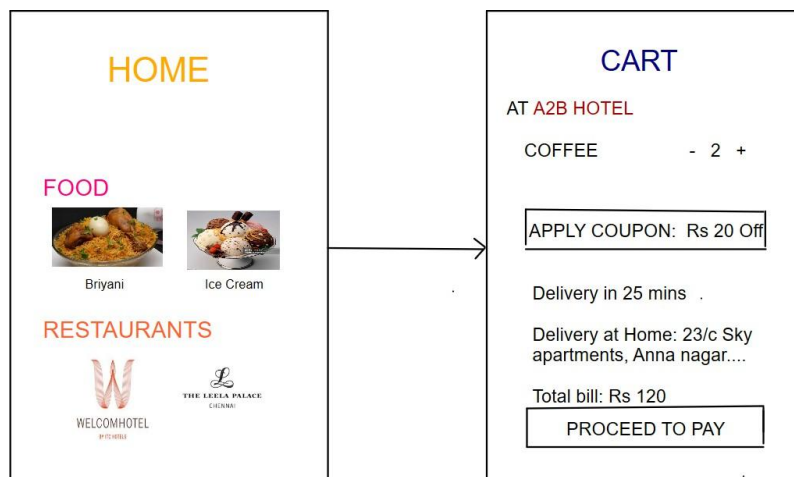
Step 1: Define the User Flow

1. Identify Key Screens:
 - Home, Menu, Cart, Checkout, Order Confirmation
2. Map the User Journey:
 - User opens Home → Browses Menu → Adds Items to Cart → Proceeds to Checkout → Sees Order Confirmation

Step 2: Create Storyboards in OpenBoard

1. Download and install OpenBoard from openboard.ch
2. Open OpenBoard and create a new document
3. Use frames to sketch the following screens:
 - Home: Search bar, cuisine categories
 - Menu: Food list with images, Add to Cart buttons
 - Cart: Items with quantity, total, Checkout button
 - Checkout: Address form, payment options, Place Order button
 - Order Confirmation: Order summary, estimated delivery time
4. Use arrows to indicate flow between screens

OUTPUT:



RESULT:

Storyboards for the mobile food delivery app were created in OpenBoard, clearly representing the user journey from browsing to order confirmation.

EXPNo: 9

DATE:26/04/2025

TITLE:Design input forms that validate data (e.g., email, phone number) and display error messages using HTML/CSS, JavaScript (with Validator.js)

AIM:

The aim is to design input forms that validate data, such as email and phone number, and display error messages using HTML/CSS and JavaScript with Validator.js.

PROCEDURE:

Index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=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="phoneError" 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>
```

```
style.css
body {
  font-family: Arial, sans-serif;
  background-color: #f4f4f4;
  display: flex;
  justify-content: center;
  align-items: center;
  height: 100vh;
  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;
}

label {
  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: 0.875em;  
}
```

```
script.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('emailError');  
  let phoneError = document.getElementById('phoneError');
```

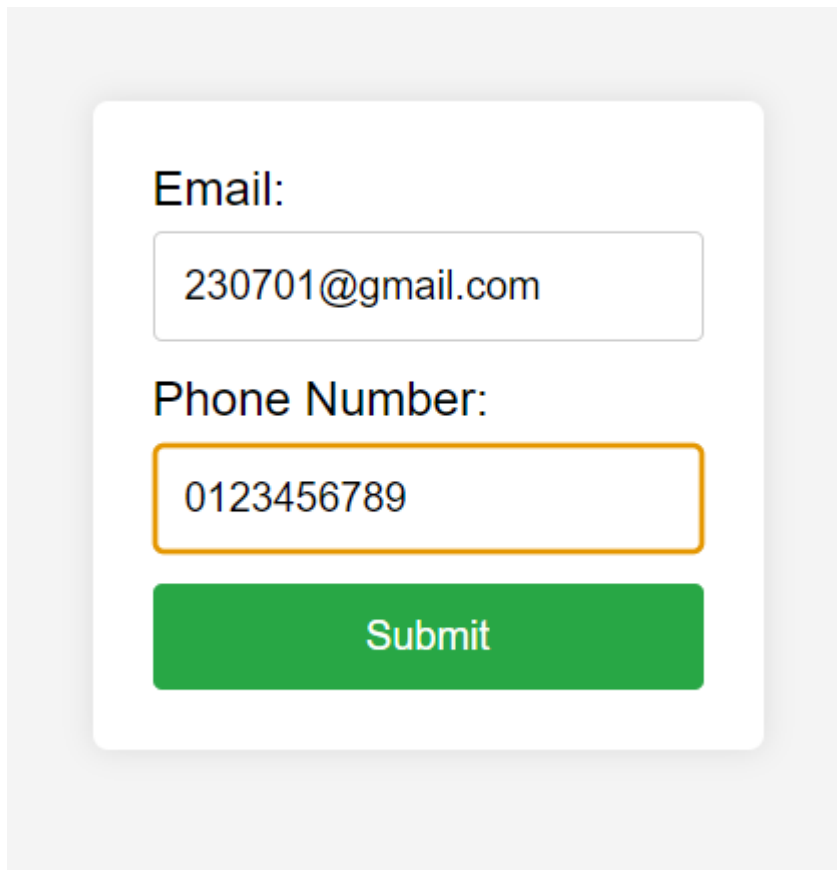
```
  // Clear previous error messages  
  emailError.textContent = "";  
  phoneError.textContent = "";
```

```
  // Validate email  
  if (!validator.isEmail(email)) {  
    emailError.textContent = 'Please enter a valid email address.';  
  }
```

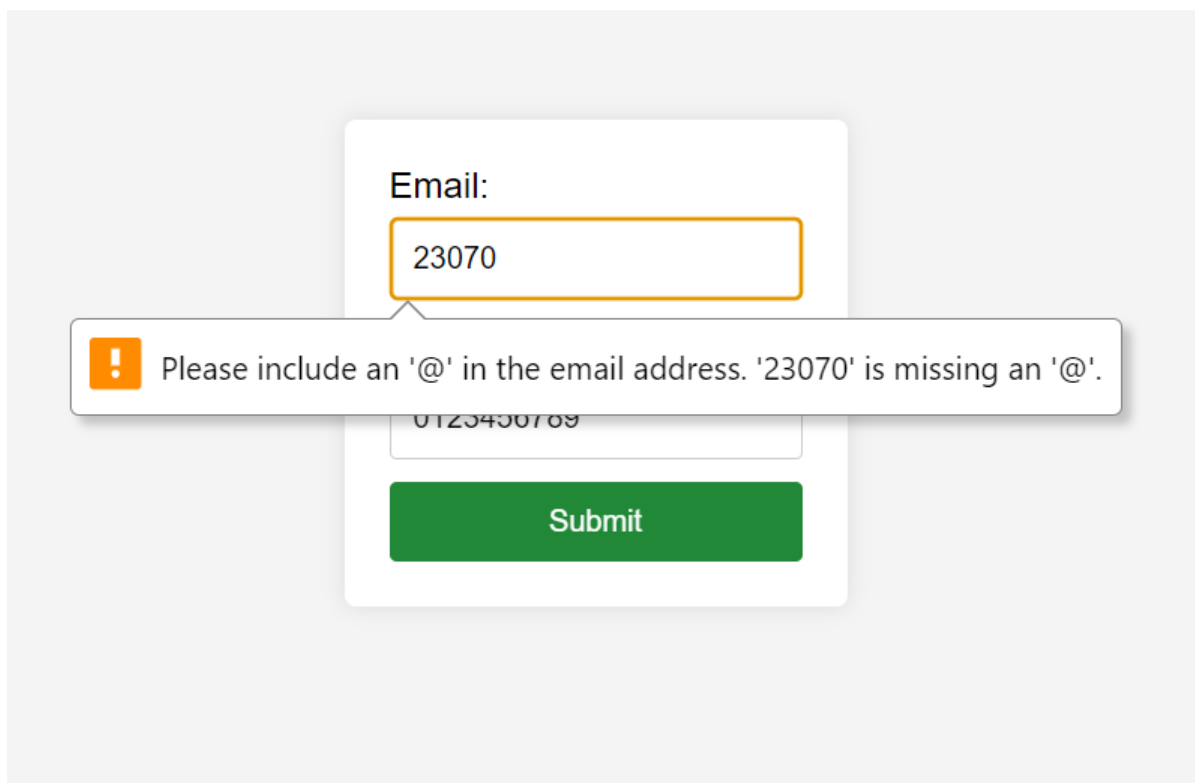
```
  // Validate phone number  
  if (!validator.isMobilePhone(phone, 'any')) {  
    phoneError.textContent = 'Please enter a valid phone number.';  
  }
```

```
  // If valid, log to console  
  if (validator.isEmail(email) && validator.isMobilePhone(phone, 'any')) {  
    console.log('Email:', email);  
    console.log('Phone:', phone);  
    alert('Form submitted successfully!');  
  }  
});
```

OUTPUT:



A form with a light gray background. It contains a white rounded rectangle with a shadow. Inside, the label "Email:" is followed by a text input field containing "230701@gmail.com". Below this, the label "Phone Number:" is followed by a text input field containing "0123456789". At the bottom is a green "Submit" button.



A form with a light gray background, similar to the one above but with an error. The "Email:" input field contains "23070" and has an orange border. A white error message box with a shadow is positioned over the bottom of the form. It contains an orange exclamation mark icon and the text: "Please include an '@' in the email address. '23070' is missing an '@'." The "Phone Number:" field and "Submit" button are visible below the error message.

Email:

Please enter a valid email address.

Phone Number:

Please enter a valid phone number.

Submit

RESULT:

The input form was successfully designed using HTML and styled with CSS. Validation of user inputs such as email and phone number was effectively implemented using JavaScript with the Validator.js library. The form displayed appropriate error messages for invalid inputs and allowed submission only when all data was valid, thereby ensuring accurate and user-friendly data entry.

EXPNo: 10

DATE:26/04/2025

TITLE:Create a data visualization (e.g., pie charts, bar graphs) for an inventory management system using javascript

AIM:

The aim is to create data visualizations, such as pie charts and bar graphs, for an inventory management system using JavaScript.

PROCEDURE:

Step 1: Set Up Your HTML File

Create an HTML file to hold the canvas for the charts and include Chart.js.

```
<!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: 50px;
    }
    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>
  <script src="script.js"></script>
</body>
</html>
```

Step 2: Create the JavaScript File (script.js)

Write the JavaScript to generate the pie and bar charts using Chart.js.

```
// Data for the inventory
```

```
const inventoryData = {
  labels: ['Electronics', 'Clothing', 'Home Appliances', 'Books', 'Toys'],
  datasets: [
    {
      label: 'Items in Stock',
      data: [200, 150, 100, 80, 50],
      backgroundColor: [
        '#FF6384',
        '#36A2EB',
        '#FFCE56',
        '#4BC0C0',
        '#9966FF'
      ]
    }
  ]
};
```

// Creating the Pie Chart

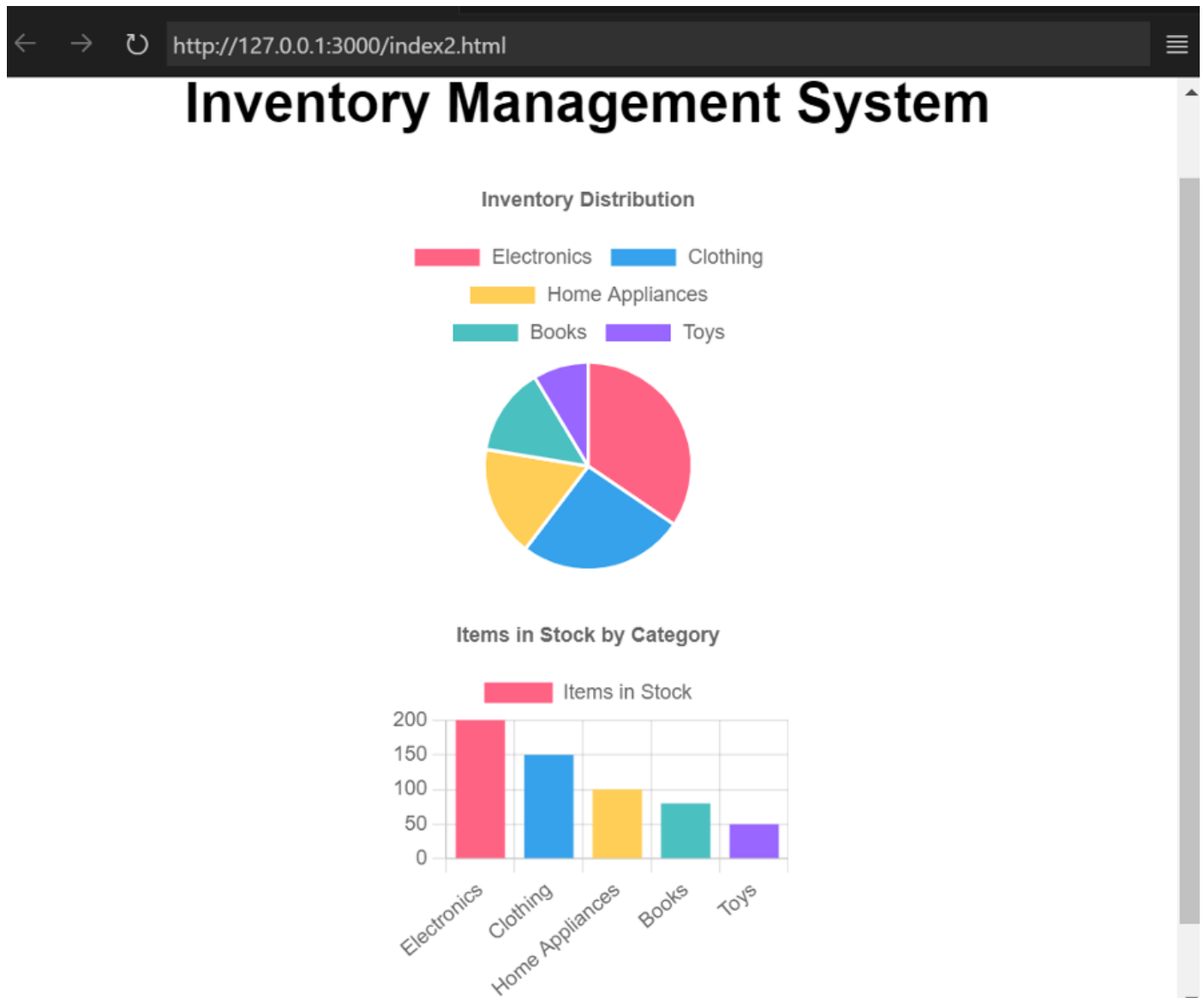
```
const ctxPie = document.getElementById('pieChart').getContext('2d');
const pieChart = new Chart(ctxPie, {
  type: 'pie',
  data: inventoryData,
  options: {
    responsive: true,
    plugins: {
      title: {
        display: true,
        text: 'Inventory Distribution'
      }
    }
  }
});
```

// Creating the Bar Chart

```
const ctxBar = document.getElementById('barChart').getContext('2d');
const barChart = new Chart(ctxBar, {
  type: 'bar',
  data: inventoryData,
  options: {
    responsive: true,
    plugins: {
      title: {
        display: true,
```

```
        text: 'Items in Stock by Category'
      }
    },
    scales: {
      y: {
        beginAtZero: true
      }
    }
  });
```

OUTPUT:



RESULT:

Successfully created data visualizations (Pie and Bar charts) using Chart.js to represent the stock levels in different inventory categories for an inventory management system.