RAJALAKSHMI ENGINEERING COLLEGE

RAJALAKSHMI NAGAR, THANDALAM – 602 105

RAJALAKSHMI
ENGINEERING COLLEGE

CS23A34
USER INTERFACE AND DESIGN LAB

Laboratory Observation NoteBook

NAME: THARUN KUMAR S
YEAR/BRANCH/SECTION: II/CSE/FD
REGISTER NUMBER: 230701393
SEMESTER: IV
ACADEMIC YEAR: 2024-25

INDEX

Reg. No : 230701393 **Name : THARUN KUMAR S**

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
	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**

Experiment -1 Good design and bad design

Good design:

The above user interface is said to be a good design as the design of the page is very familiar and easy to understand.

Bad design:

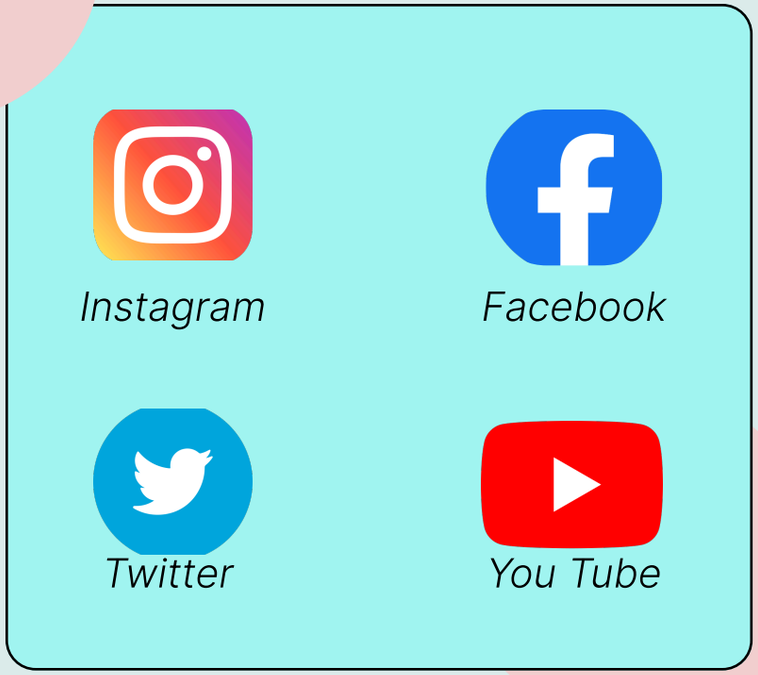
The above user interface is a bad design as there is no consistency, it is not familiar. Login button is not present in familiar position. Different fonts and colours have been used – no consistency.

Memory Recall Task

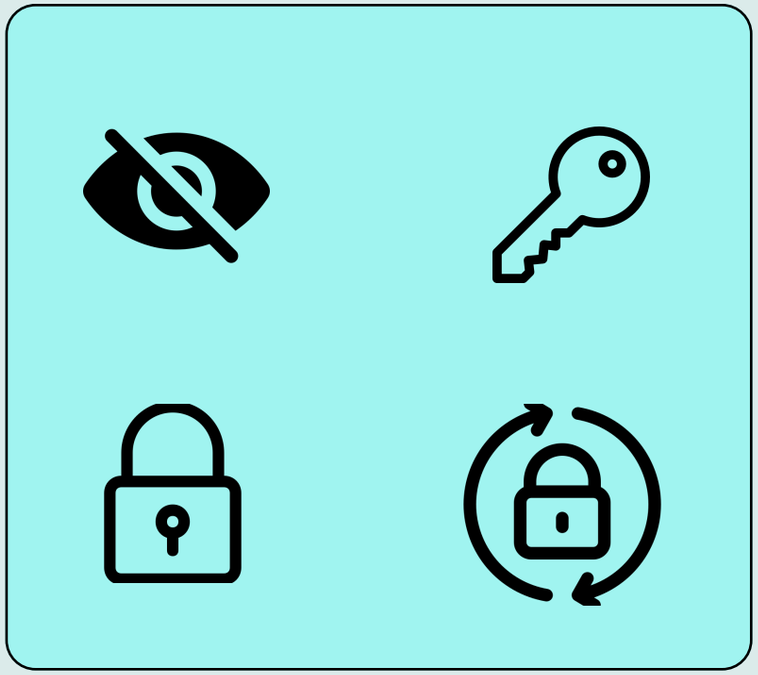
You will be shown several groups of icons or text.
After viewing, recall the items you remember.

You will have 5 seconds to view the items.
Then, recall them in the next screen.

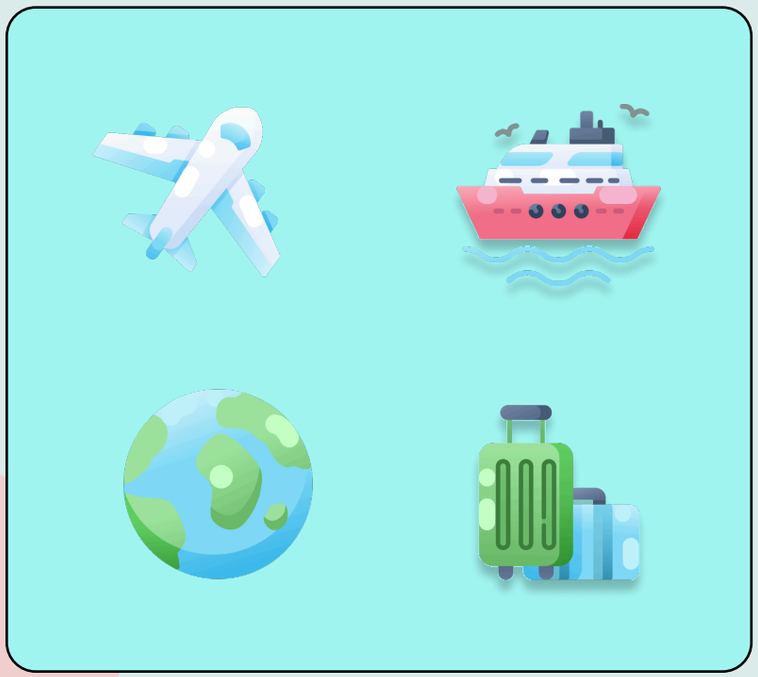
START



Social Media's



Security and Password Icon's



Travel icon's

Food Icon's

Select the items you remember seeing

Submit Recall

Congratulations

You recalled 4/5 items correctly!

USER INTERFACE AND DESIGN

EXPERIMENT 3

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 :

i) CLI (Command Line Interface)

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

```
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 your 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()
```

Output :

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

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

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

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

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

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

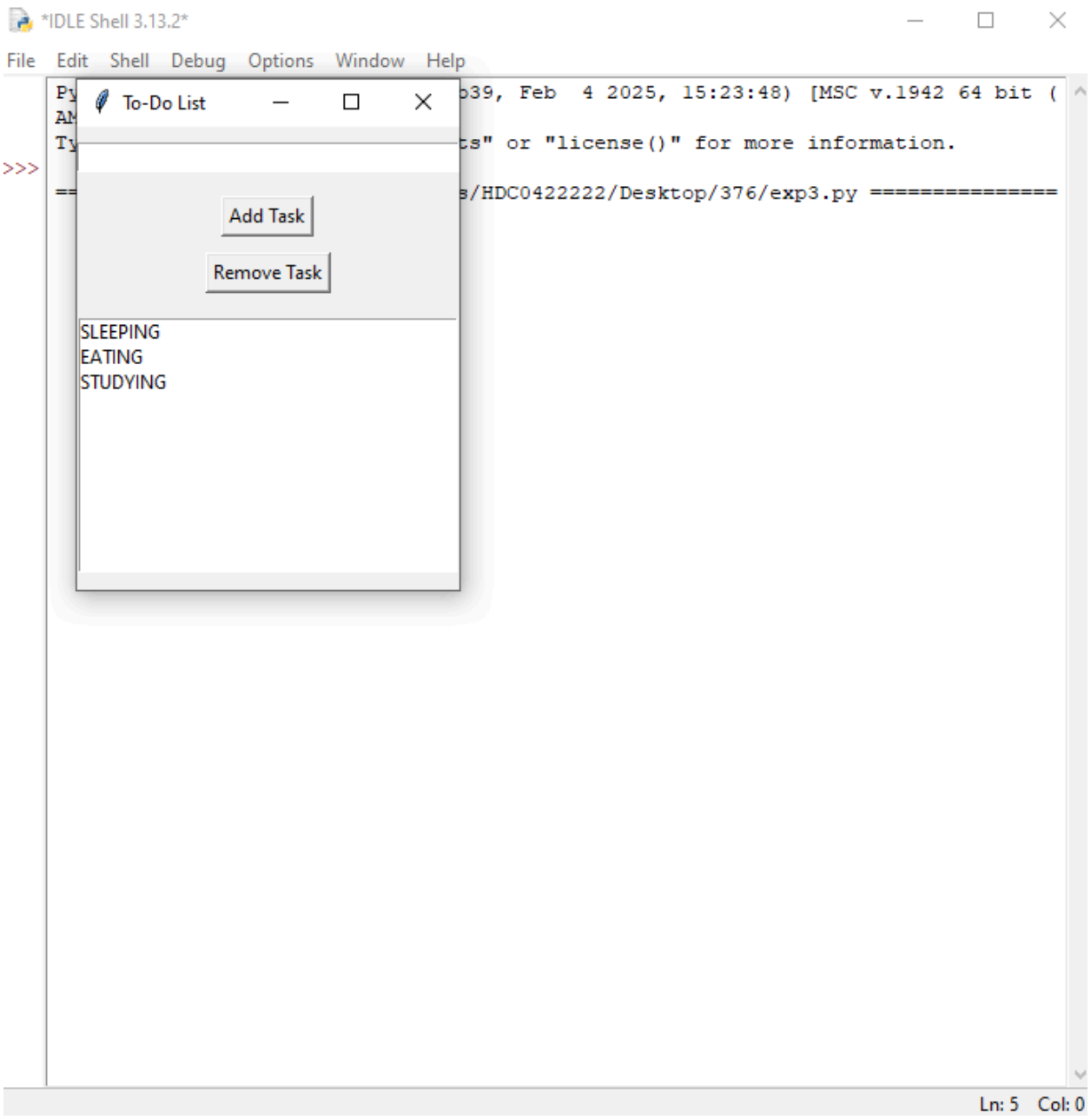
ii) GUI (Graphical User Interface)

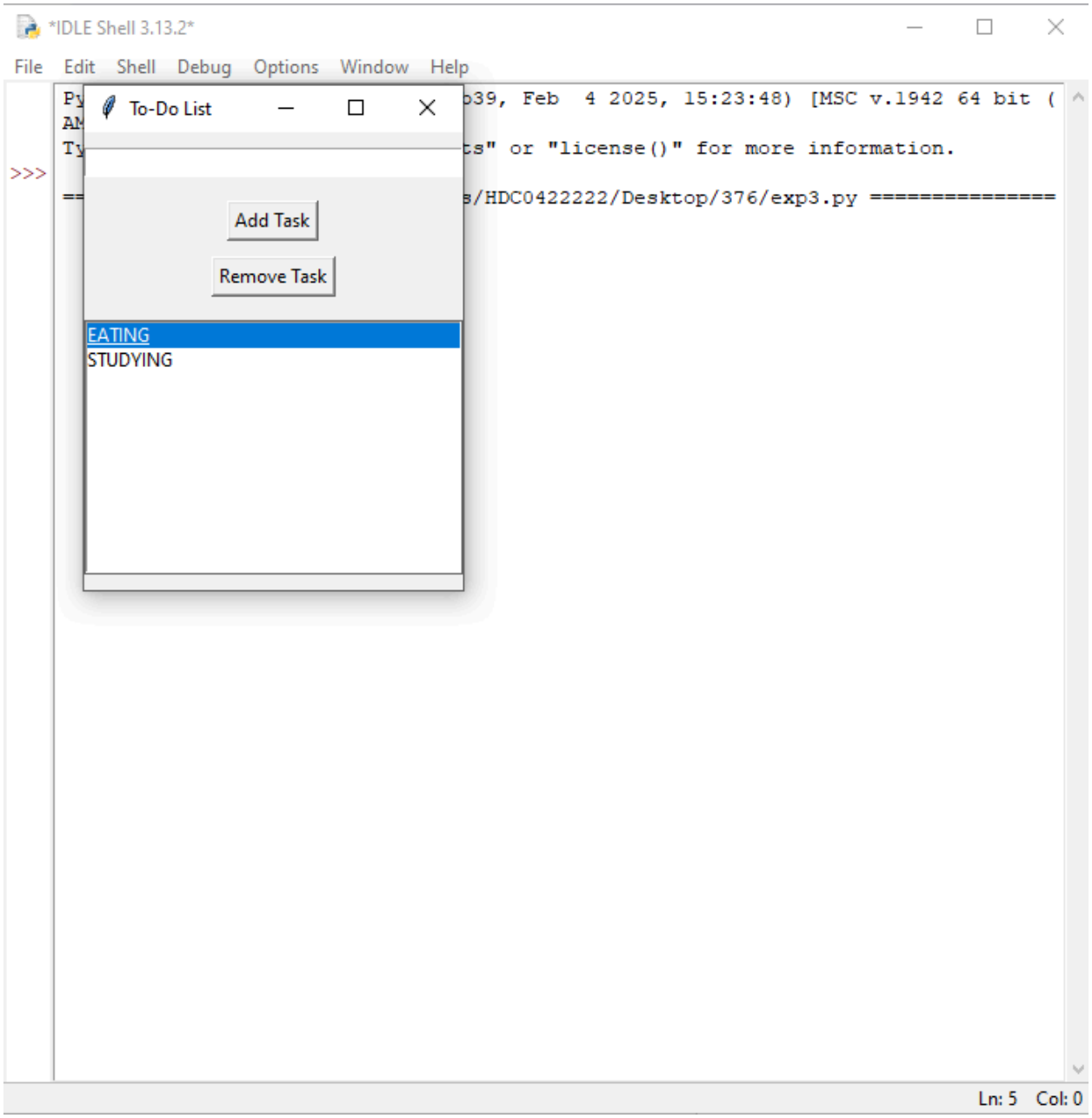
Tkinter to create a simple GUI for our To-Do List application.

```
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()
```

OUTPUT:




IDLE Shell 3.13.2
File Edit Shell Debug Options Window Help
b39, Feb 4 2025, 15:23:48) [MSC v.1942 64 bit (
ts" or "license()" for more information.
>>>
s/HDC0422222/Desktop/376/exp3.py ==========
To-Do List
Add Task
Remove Task
EATING
STUDYING
Ln: 5 Col: 0

iii) VUI (Voice User Interface)

speech_recognition library for voice input and the pyttsx3 library for text-to-speech output. Make sure you have these libraries installed (pip install SpeechRecognition pyttsx3).

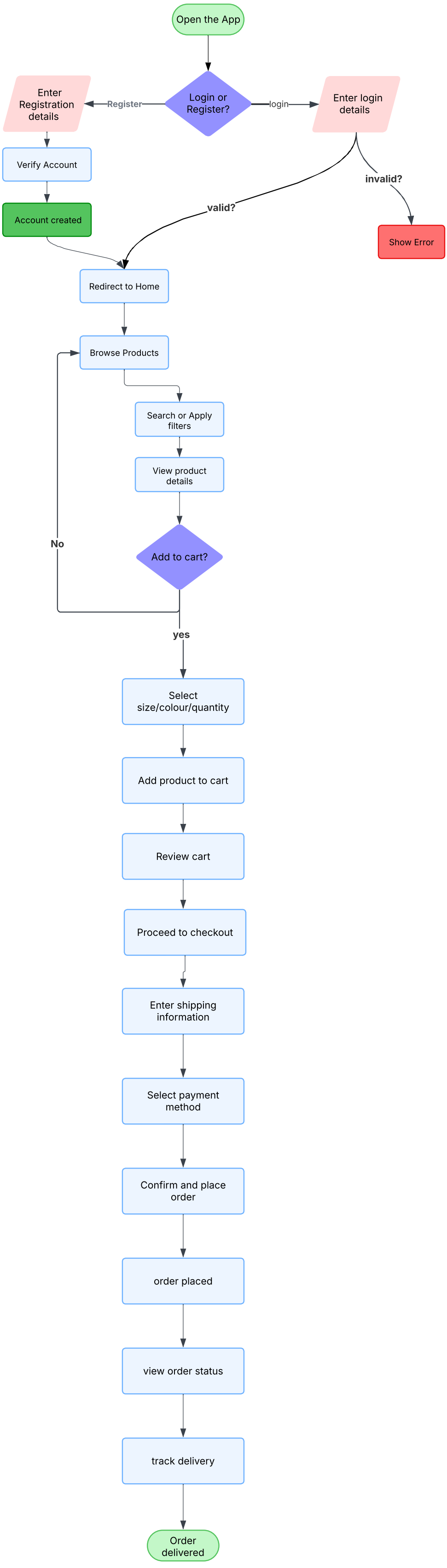
```
import speech_recognition as sr
import pyttsx3

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()
```

OUTPUT:

```
Listening...
Task Buy stationaries added.
Listening...
Task Finish UID observation added.
Listening...
Task Take printout of OS manual added.
Listening...
Task Complete UID project added.
Listening...
Task Take Bath added.
Listening...
Your tasks are: Buy stationaries, Finish UID observation, Take printout of OS manual, Complete UID project, Take Bath.
Listening...
Task Take Bath removed.
Listening...
Task Buy stationaries removed.
Listening...
Your tasks are: Finish UID observation, Take printout of OS manual, Complete UID project.
Listening...
Exiting
```

RESULT: Thus the VUI , CLI, GUI were all observed, studied and executed successfully.


Open the App
Login or Register?
Register
login
Enter Registration details
Enter login details
Verify Account
Account created
invalid?
valid?
Show Error
Redirect to Home
Browse Products
Search or Apply filters
View product details
Add to cart?
No
yes
Select size/colour/quantity
Add product to cart
Review cart
Proceed to checkout
Enter shipping information
Select payment method
Confirm and place order
order placed
view order status
track delivery
Order delivered

RAJALAKSHMI ENGINEERING COLLEGE

RAJALAKSHMI NAGAR, THANDALAM – 602 105

RAJALAKSHMI
ENGINEERING COLLEGE

CS23A34
USER INTERFACE AND DESIGN LAB

Laboratory Observation NoteBook

Name : THARUN KUMAR S
Year/Branch/Section : II/CSE/D
Register No. : 230701393
Semester : IV
Academic Year: 2024-25

Ex. No. : 4b **Date : 22.03.2025**

Register No. : 230701393 **Name : THARUN KUMAR S**

Conduct task analysis for an app (e.g., online shopping) 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:

Tool link: http://dia-installer.de/

1. Install Dia:

○ Download Dia from the official website (http://dia-installer.de/)

○ Install Dia on your computer

○ Open Dia:

○ Launch the Dia application.

2. Create New Diagram:

○ Go to File -> New Diagram.

○ Select Flowchart as the diagram type.

3. Add Shapes:

○ Use the shape tools (rectangles, ellipses, etc.) to create wireframes for each screen.

■ For example:

■ Home Page: Rectangle

■ Product Categories: Rectangle

■ Product Listings: Rectangle

■ Product Details: Rectangle

■ Cart: Rectangle

■ Checkout: Rectangle

■ Order Confirmation: Rectangle

■ Order History: Rectangle

4. Connect Shapes:

○ Use the line tool to connect shapes, representing the user flows.

■ For example:

■ Home Page -> Product Categories

■ Product Categories -> Product Listings

■ Product Listings -> Product Details

■ Product Details -> Cart

■ Cart -> Checkout

■ Checkout -> Order Confirmation

■ Order Confirmation -> Order History

5. Label Shapes:

○ Double-click on each shape to add labels.

■ For example:

■ Label the rectangle as "Home Page", "Categories",

"Product Listings", "Product Details", "Cart", "Checkout",

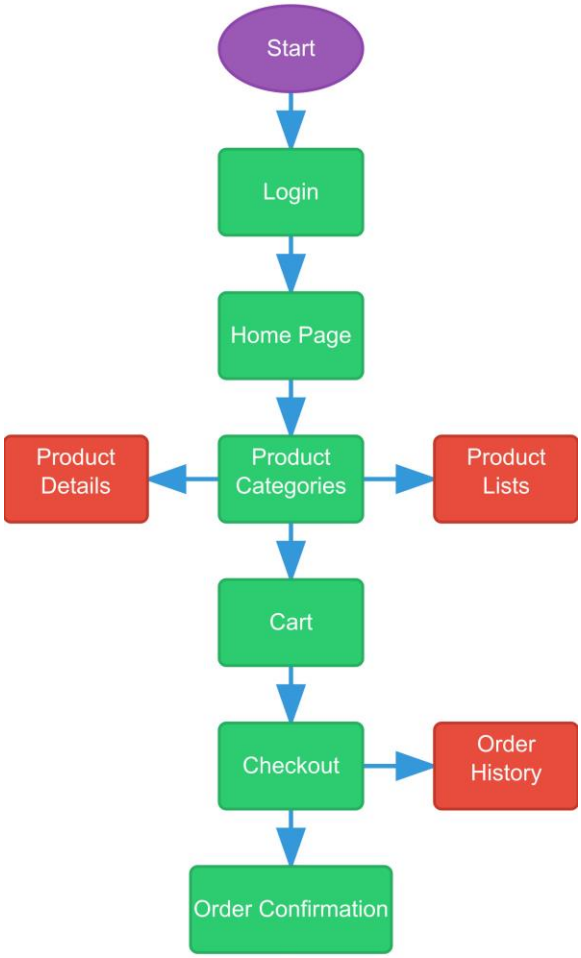
"Order Confirmation", "Order History".

6. Save the Diagram:

○ Go to File -> Save As.

○ Save the diagram with a meaningful name, such as "Online Shopping App User Flows".

OUTPUT:



RESULT:

Hence, to perform task analysis for an app, such as online shopping, document user flows, and create corresponding wireframes using Dia has been successfully executed.

RAJALAKSHMI ENGINEERING COLLEGE

RAJALAKSHMI NAGAR, THANDALAM – 602 105

RAJALAKSHMI
ENGINEERING COLLEGE

CS23A34
USER INTERFACE AND DESIGN LAB

Laboratory Observation NoteBook

Name : THARUN KUMAR S
Year/Branch/Section : II/CSE/D
Register No. : 230701393
Semester : IV
Academic Year: 2024-25

Ex. No. : 5a **Date : 29.03.2025**

Register No. : 230701393 **Name : THARUN KUMAR S**

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:

Tool Link: https://www.axure.com/

Simulating the Lifecycle Stages for UI Design Using the RAD Model

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

1. Requirements Planning:

○ Gather initial requirements and identify key features of the UI.

○ Engage stakeholders to understand their needs and expectations.

2. User Design:

○ Create initial prototypes and wireframes.

○ Conduct user feedback sessions to refine the designs.

○ Use tools like Axure RP to develop interactive prototypes.

3. Construction:

○ Develop the actual UI based on the refined designs.

○ Perform iterative testing and feedback cycles.

4. Cutover:

○ Deploy the final UI.

○ Conduct user training and support.

Axure RP Interactive Interface Development

Phase 1: Requirements Planning

1. Identify Key Features:

○ Navigation (Home, Product Categories, Product Details, Cart, Checkout,

Order Confirmation, Order History)

○ User actions (Browsing, Searching, Adding to Cart, Checkout, Tracking Orders)

2. Create a Requirements Document:

○ List all features and functionalities.

○ Document user stories and use cases.

Phase 2: User Design

1. Install and Launch Axure RP:

○ Download and install Axure RP from Axure&'s official website.

○ Launch the application.

2. Create a New Project:

○ Go to File -> New to create a new project.

○ Name the project (e.g., "Shopping App Interface").

3. Create Wireframes:

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

○ Design wireframes for each screen:

■ Home Page

■ Product Categories

■ Product Listings

■ Product Details

■ Cart

■ Checkout

■ Order Confirmation

■ Order History

4. Add Interactions:

○ Select an element (e.g., button) and go to the Properties panel.

○ Click on Interactions and choose an interaction (e.g., OnClick).

○ Define the action (e.g., navigate to another screen).

5. Create Masters:

○ Create reusable components (e.g., headers, footers) using Masters.

○ Drag and drop masters onto the wireframes.

6. Add Annotations:

○ Add notes to describe each element's purpose and functionality.

○ Use the Notes panel to add detailed annotations.

Phase 3: Construction

1. Develop Interactive Prototypes:

○ Convert wireframes into interactive prototypes by adding interactions and transitions.

○ Use dynamic panels to create interactive elements (e.g., carousels, pop - ups).

2. Test and Iterate:

○ Preview the prototype using the Preview button.

○ Gather feedback from users and stakeholders.

○ Make necessary adjustments based on feedback.

Phase 4: Cutover

1. Finalize and Export:

○ Finalize the design and interactions.

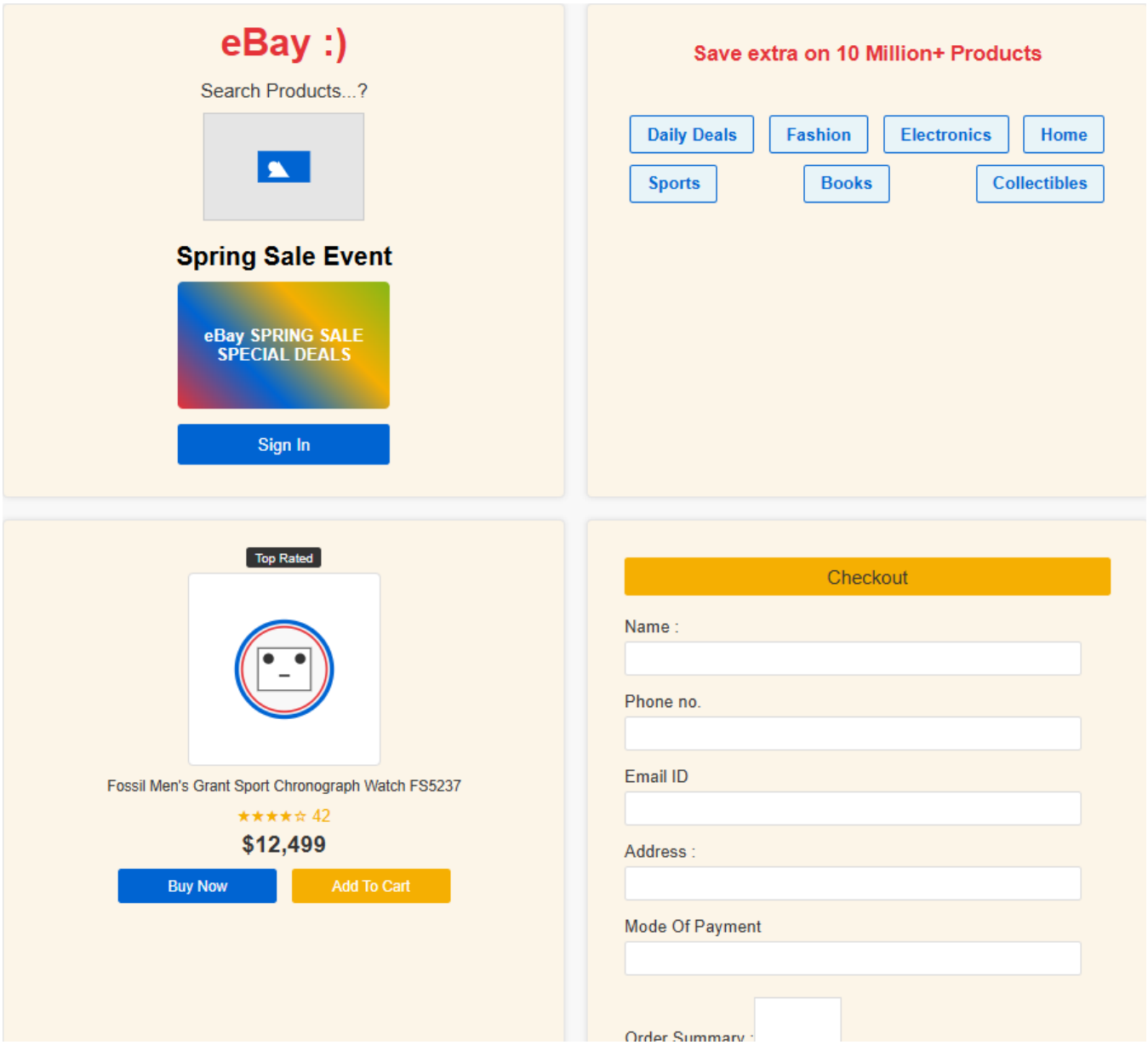
○ Export the prototype as an HTML file or share it via Axure Cloud.

2. User Training and Support:

○ Conduct training sessions to familiarize users with the new interface.

○ Provide documentation and support for any issues.

OUTPUT:


eBay :)
Search Products...?
Spring Sale Event
eBay SPRING SALE SPECIAL DEALS
Sign In
Save extra on 10 Million+ Products
Daily Deals
Fashion
Electronics
Home
Sports
Books
Collectibles
Top Rated
Fossil Men's Grant Sport Chronograph Watch FS5237
★★★★☆ 42
$12,499
Buy Now
Add To Cart
Checkout
Name :
Phone no.
Email ID
Address :
Mode Of Payment
Order Summary :

RESULT:

Hence, demonstration of the lifecycle stages of UI design via the RAD model and develop a small interactive interface employing Axure RP.

RAJALAKSHMI ENGINEERING COLLEGE

RAJALAKSHMI NAGAR, THANDALAM – 602 105

RAJALAKSHMI
ENGINEERING COLLEGE

CS23A34
USER INTERFACE AND DESIGN LAB

Laboratory Observation NoteBook

Name : THARUN KUMAR S
Year/Branch/Section : II/CSE/D
Register No. : 230701393
Semester : IV
Academic Year: 2024-25

Ex. No. : 5b **Date : 29.03.2025**

Register No. : 230701393 **Name : THARUN KUMAR S**

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:

Tool Link: https://sourceforge.net/projects/openproj/

Step 1: Requirements Planning

1. Gather Requirements:

    ○ Identify key features and functionalities needed for your interface.

    ○ Example: A simple "Login" and "Register" interface with debug logs.

2. Define Use Cases:

    ○ Specify use cases for user login and registration.

○ Example: User logs in with valid credentials, user registers with a new account.

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

1. Sketch Initial Designs:

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

2. Create Digital Wireframes:

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

Example Wireframes:

1. Login Screen: Username field, Password field, Login button, Register link.

2. Register Screen: Username field, Email field, Password field, Confirm Password field, Register button.

Output in OpenProj:

● Add tasks: "Sketch Initial Designs" and "Create Digital Wireframes."

● Allocate time and resources to complete these tasks.

Step 3: Rapid Prototyping

1. Develop Prototypes:

    ○ Use a tool like Axure RP to convert wireframes into interactive prototypes.

2. Test Prototypes:

    ○ Share prototypes with stakeholders for feedback.

    ○ Collect feedback and iterate on the design.

Output:

● Interactive prototypes for "Login" and "Register" screens.

    Output in OpenProj:

● Add tasks: "Develop Prototypes" and "Test Prototypes."

● Set dependencies and milestones.

Step 4: User Acceptance/Testing

1. Review Prototype:

    ○ Conduct user and stakeholder reviews.

2. Conduct Usability Testing:

    ○ Perform usability testing and document feedback.

Output:

● Documented feedback and test results.

Output in OpenProj:

● Add tasks: "Review Prototype" and "Usability Testing."

● Track progress and resources.

Step 5: Implementation

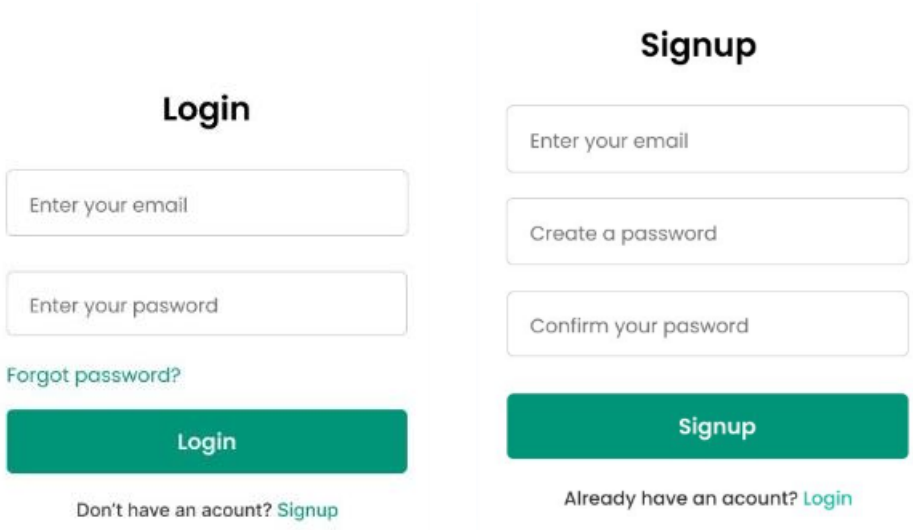
1. Develop Functional Interface:

    ○ Implement final designs and functionalities based on feedback.

2. Integrate Backend (if required):

    ○ Connect the UI with backend services for tasks like user authentication.

OUTPUT:



RESULT:

Hence the lifecycle stages of UI design using the RAD model and design of a small interactive interface with OpenProj has been successfully executed.

Excercise 6 **Date:**

ROLL NO:230701393 **NAME: THARUN KUMAR S**

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

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

Step 2: Create a New Project

1. **Open GIMP:**
    - Launch the GIMP application.
2. **Create a New Canvas:**
    - Go to File -> New to create a new project.
    - 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:**
    - Use the Rectangle Select Tool to create sections for different parts of your app (e.g., header, content area, footer).
    - 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:**
    - **Text Elements:** Use the Text Tool to add text elements like headers, buttons, and labels.
    - **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:**
    - Use layers to separate different UI elements. This allows you to easily modify or experiment with individual components.
    - Name each layer according to its content (e.g., Header, Button1, InputField).

Step 4: Experiment with Color Schemes

1. **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.

Example Output: Multiple color variants of the same layout.

2. **Save Each Variant:**

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

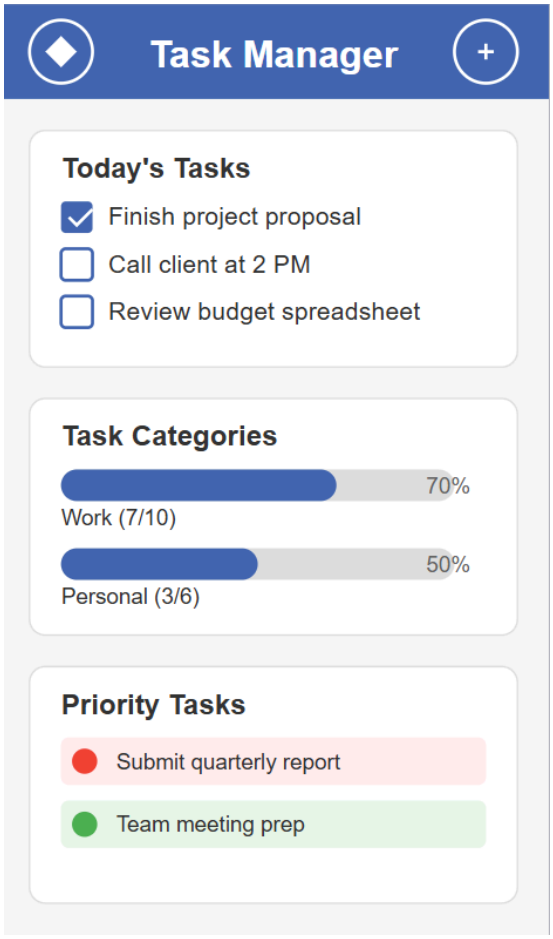
Step 5: Collect User Feedback

1. **Prepare a Feedback Form:**
   - **Create Form:** Create a feedback form using tools like Google Forms or Microsoft Forms.
   - **Include Questions:** Include questions about the aesthetics and usability of each layout and color scheme.
2. **Share the Variants:**
   - **Distribute Files:** Share the image files of the different layouts 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.
3. **Gather Feedback:**
   - Collect responses from users regarding their preferences and suggestions.
   - Analyze the feedback to determine which layout and color scheme are most preferred.

Step 6: Iterate and Refine

1. **Refine the Design:**
   - Based on the feedback, make necessary adjustments to the layout and color scheme.
   - Experiment with additional variations if needed.
2. **Final Testing:**
   - Conduct a final round of testing with the refined design to ensure usability and aesthetic satisfaction.

OUTPUT:



RESULT:

Excercise 7a **Date:**

ROLLNO:230701393 **NAME:THARUN KUMAR S**

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

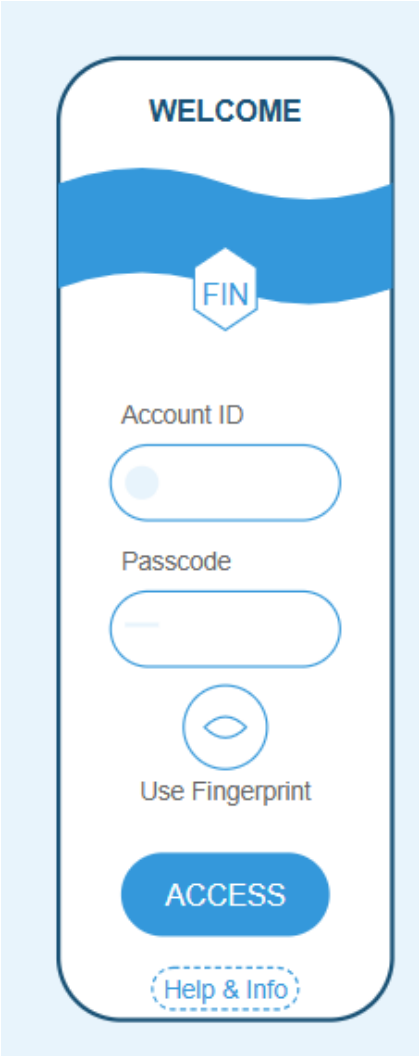
1. **Define the Purpose and Features**:
    - Identify the core features of the banking app (e.g., login, account balance, transfers, bill payments).
2. **Sketch Basic Layouts**:
    - Use plain paper and pencils to sketch basic screens.
    - Focus on primary elements like buttons, menus, and forms.
3. **Iterate and Refine**:
    - Get feedback from users or stakeholders.
    - Iterate on your sketches to improve clarity and functionality.

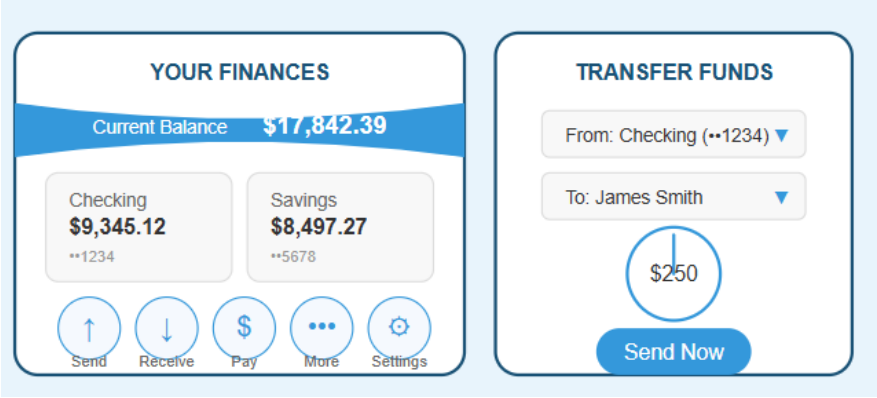
Step 2: Convert Paper Prototypes to Digital Wireframes Using Pencil Project

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).
4. **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.
5. **Organize and Align**:
    - Arrange and align the elements to match your paper prototype.
    - Ensure that the design is user-friendly and intuitive.
6. **Link Screens**:
    - Use connectors to link different screens together.
    - Create navigation flows to show how users will interact with the app.
7. **Add Annotations**:
    - Include annotations to explain the functionality of different elements.
8. **Export Your Wireframes**:
    - Once satisfied with your digital wireframes, export them in your preferred format (e.g., PNG, PDF).

OUTPUT:


WELCOME
FIN
Account ID
Passcode
Use Fingerprint
ACCESS
Help & Info


YOUR FINANCES
Current Balance
$17,842.39
Checking
$9,345.12
••1234
Savings
$8,497.27
••5678
Send
Receive
Pay
More
Settings
TRANSFER FUNDS
From: Checking (••1234)
To: James Smith
$250
Send Now

RESULT:

Excercise 7b

ROLL NO:230701393

NAME: THARUN KUMAR S

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:

Tool Link: https://inkscape.org/

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.

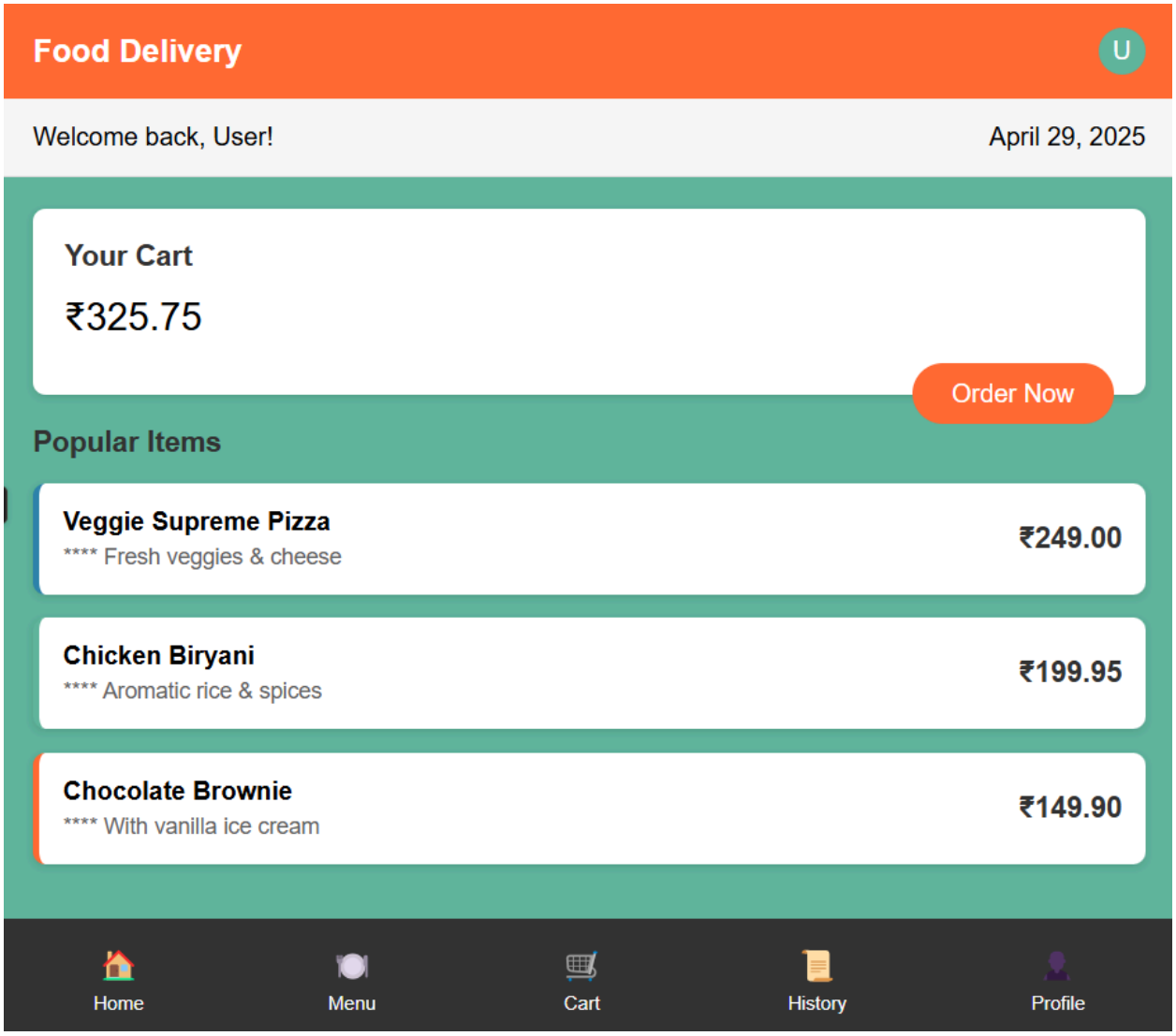
Step 2: Convert Paper Prototypes to Digital Wireframes Using Inkscape

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:


Food Delivery
U
Welcome back, User!
April 29, 2025
Your Cart
₹325.75
Order Now
Popular Items
Veggie Supreme Pizza
**** Fresh veggies & cheese
₹249.00
Chicken Biryani
**** Aromatic rice & spices
₹199.95
Chocolate Brownie
**** With vanilla ice cream
₹149.90
Home
Menu
Cart
History
Profile

Excercise 8a **ROLL NO:230701393**

NAME: THARUN KUMAR S

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:

Tool Link: https://balsamiq.com/

Step 1: Define the User Flow

1. **Identify Key Screens**:
    - List the main screens your app will have (e.g., Home, Menu, Cart, Checkout, Order Confirmation).
2. **Map the User Journey**:
    - Understand the typical user journey through these screens (e.g., browsing menu, adding items to cart, checking out).

Step 2: Create Storyboards Using Balsamiq

1. **Install Balsamiq**:
    - Download and install Balsamiq from the https://balsamiq.com/ website.
2. **Create a New Project**:
    - Open Balsamiq and create a new project.
3. **Add Wireframe Screens**:

    - Use the “+” button to add new wireframe screens for each key screen in your app.
4. **Design Each Screen**:
    - Use Balsamiq's components to design the UI for each screen.
    - Include basic elements like buttons, text fields, and images.
5. **Organize the Flow**:
    - Arrange the screens in the order users will navigate through them.
    - Connect the screens with arrows to represent user actions.

Example Screens for Food Delivery App

1. **Home Screen**:
    - Search bar for finding restaurants
    - Categories for different cuisines
2. **Menu Screen**:
    - List of food items with images, names, and prices
    - Add to Cart buttons
3. **Cart Screen**:
    - Items added to the cart with quantity and total price
    - Checkout button
4. **Checkout Screen**:
    - Delivery address form
    - Payment options
    - Place Order button
5. **Order Confirmation Screen**:
    - Order summary
    - Estimated delivery time

Example Output

Here's how the wireframes might look:

Home Screen

- **Search Bar**: Allows users to search for restaurants.
- **Categories**: Buttons for different cuisines (e.g., Italian, Chinese).

Menu Screen

- **Food Items List**: Displays food items with images, names, and prices.
- **Add to Cart**: Button to add items to the cart.

Cart Screen

- **Items Added**: Lists items added to the cart with quantity and prices.
- **Checkout Button**: Proceed to checkout.

Checkout Screen

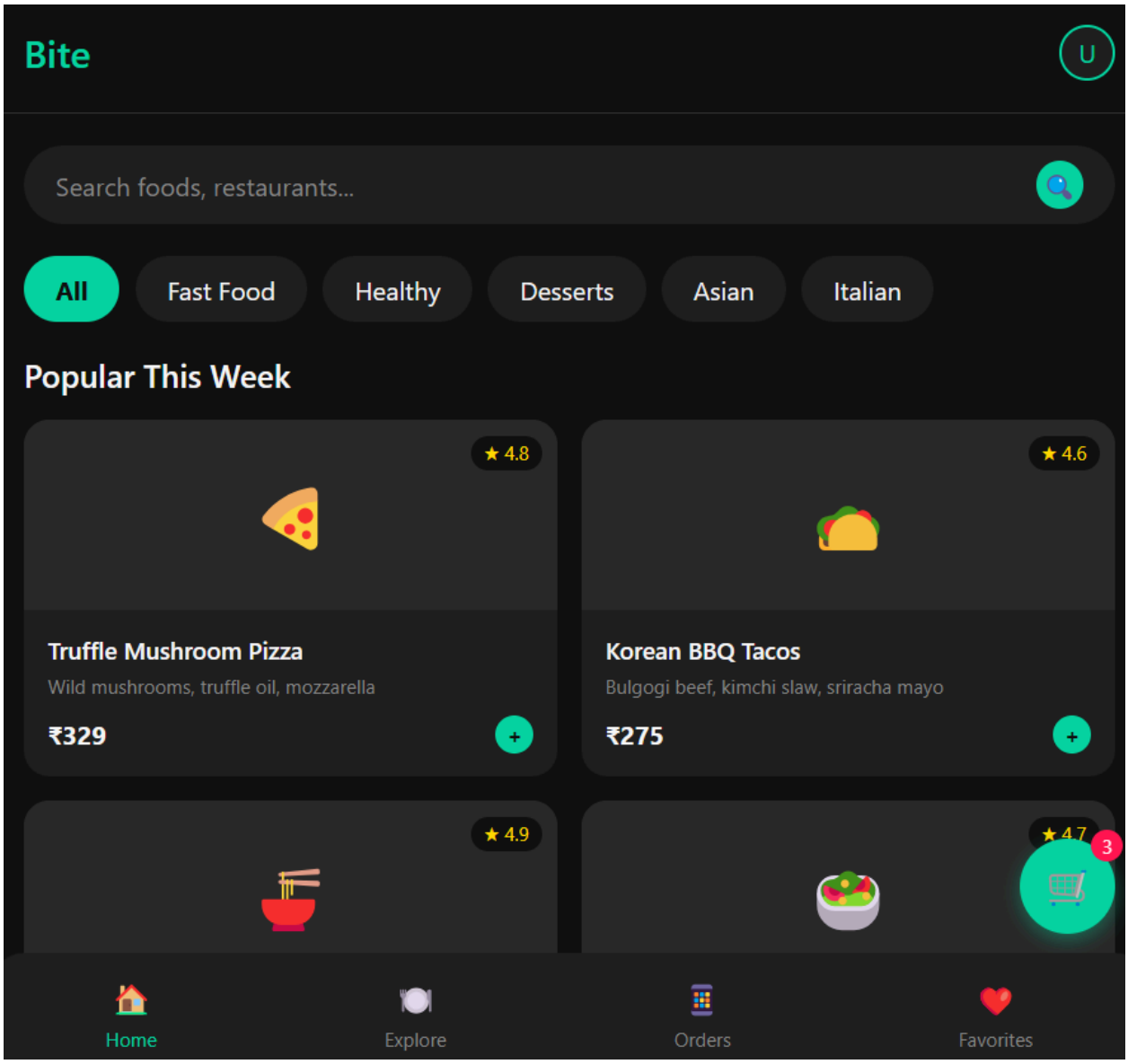
- **Delivery Address Form**: Users enter their delivery address.
- **Payment Options**: Choose between different payment methods.
- **Place Order Button**: Finalize the order.

Order Confirmation Screen

- **Order Summary**: Shows the order details.
- **Estimated Delivery Time**: Provides an estimated delivery time.

OUTPUT:

RESULT:


Bite
U
Search foods, restaurants...
All
Fast Food
Healthy
Desserts
Asian
Italian
Popular This Week
★ 4.8
Truffle Mushroom Pizza
Wild mushrooms, truffle oil, mozzarella
₹329
★ 4.6
Korean BBQ Tacos
Bulgogi beef, kimchi slaw, sriracha mayo
₹275
★ 4.9
★ 4.7
3
Home
Explore
Orders
Favorites

Excercise 8b **ROLLNO:230701393**

NAME: THARUN KUMAR S

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:

Tool Link: https://openboard.ch/download.en.html

Step 1: Define the User Flow

1. **Identify Key Screens**:
    - List the main screens your app will have (e.g., Home, Menu, Cart, Checkout, Order Confirmation).
2. **Map the User Journey**:
    - Understand the typical user journey through these screens (e.g., browsing menu, adding items to cart, checking out).

Step 2: Create Storyboards Using OpenBoard

1. **Install OpenBoard**:
    - Download and install OpenBoard from the official website.
2. **Create a New Document**:
    - Open OpenBoard and create a new document.
3. **Add Frames for Each Screen**:
    - Use the drawing tools to create frames representing each key screen of your app.

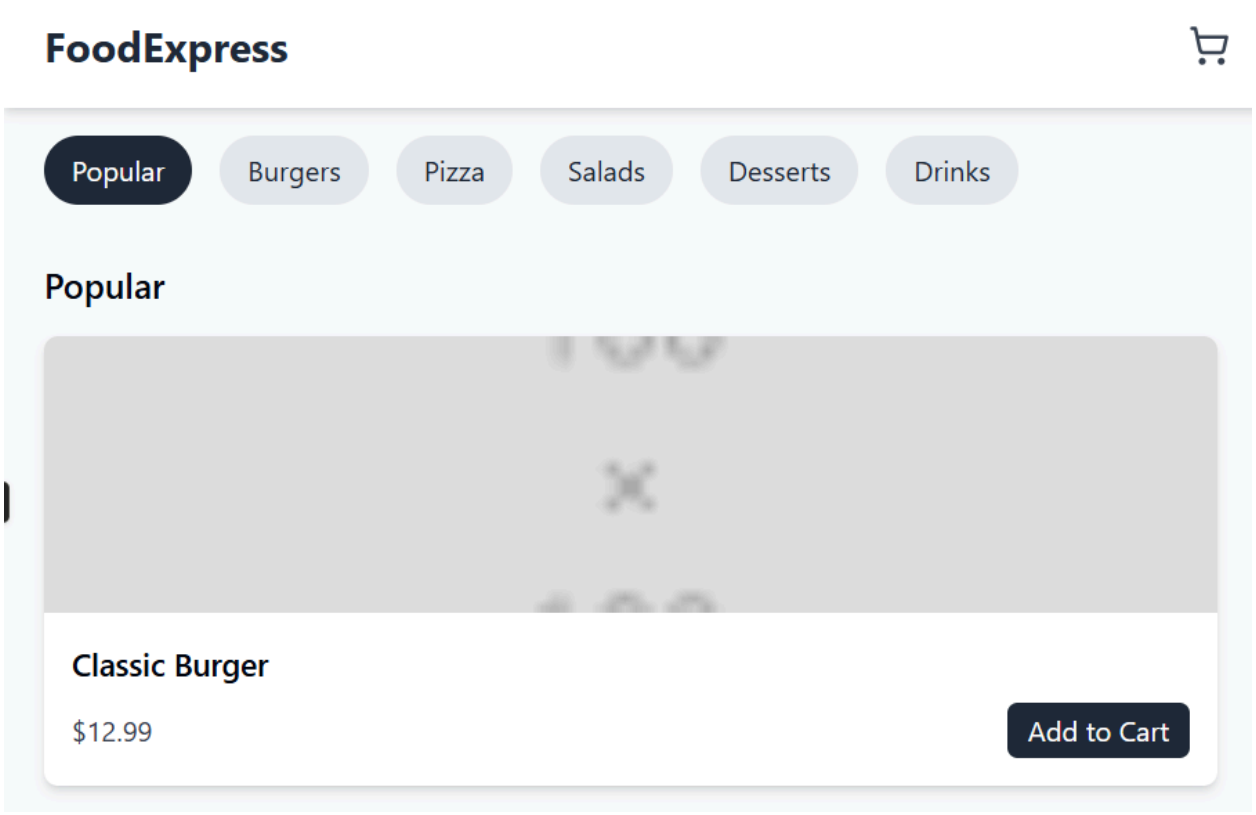
4. **Sketch Each Screen**:
    - Use the pen or shape tools to draw basic elements for each screen.
    - Focus on major UI components like buttons, text fields, and icons.
5. **Organize the Flow**:
    - Arrange the frames in a sequence that represents the user journey.
    - Use arrows or lines to show navigation paths between screens.

Example Screens for Food Delivery App

1. **Home Screen**:
    - Search bar for finding restaurants
    - Categories for different cuisines
2. **Menu Screen**:
    - List of food items with images, names, and prices
    - Add to Cart buttons
3. **Cart Screen**:
    - Items added to the cart with quantity and total price
    - Checkout button
4. **Checkout Screen**:
    - Delivery address form
    - Payment options
    - Place Order button
5. **Order Confirmation Screen**:
    - Order summary
    - Estimated delivery time

OUTPUT:

RESULT:


FoodExpress
Popular
Burgers
Pizza
Salads
Desserts
Drinks
Popular
Classic Burger
$12.99
Add to Cart

Excercise 9 **ROLLNO:230701393**

NAME: THARUN KUMAR S

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:

Step 1: Setting Up the HTML Form

Start by creating an HTML form with input fields 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-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>
```

Step 2: Styling the Form with CSS

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

css

```
/* 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;
}
```

Step 3: Adding JavaScript for Validation

Finally, add JavaScript to validate the input fields using Validator.js and display error messages.

javascript

```
/* 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 no errors, submit the form (for demonstration purposes, we'll just log the values)
    if (validator.isEmail(email) && validator.isMobilePhone(phone, 'any')) {
        console.log('Email:', email);
        console.log('Phone:', phone);
    }
});
```

OUTPUT:

Login

Email Address

your@email.com

Password

••••••••

Remember me

Forgot your password?

Login

Don't have an account? Sign up

RESULT:

Excercise 10 **ROLLNO:230701393**

NAME: THARUN KUMAR S

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

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: 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 for Charts

Next, create a JavaScript file (script.js) to handle the data visualization logic.

javascript

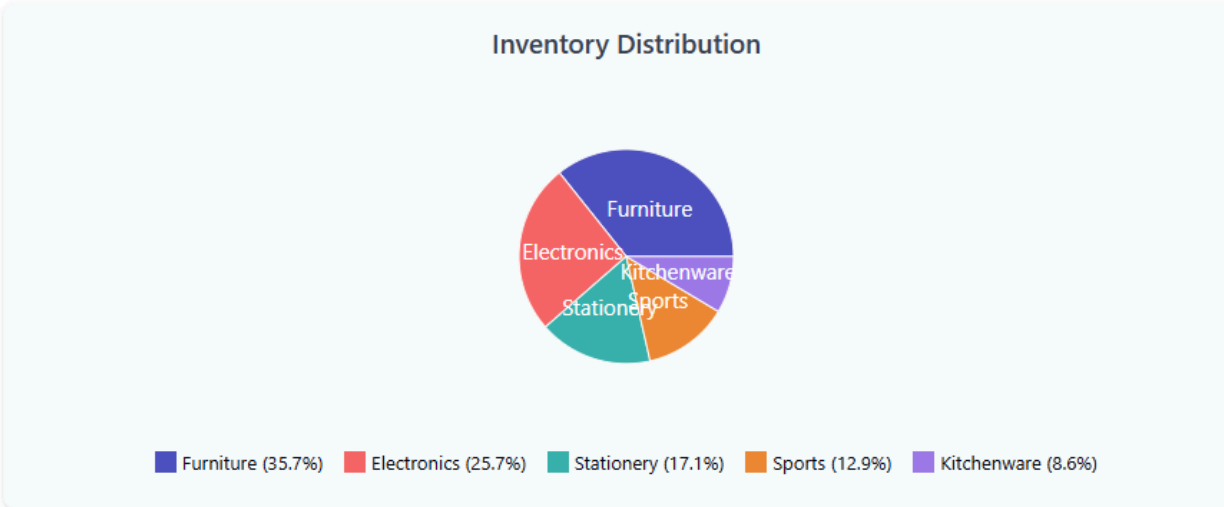
```
// script.js

// 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',
```

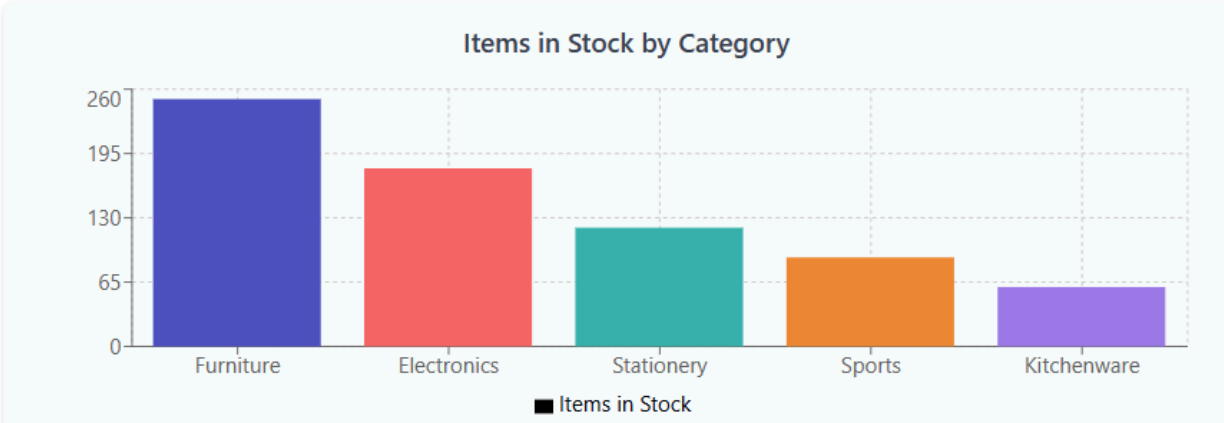
```
                '#FFCE56',
                '#4BC0C0',
                '#9966FF'
            ],
        }
    ]
};

// Creating the Pie Chart
const ctxPie = document.getElementById('pieChart').getContext('2d');
const pieChart = new Chart(ctxPie, {
    type: 'pie',
    data: inventoryData,
    options: {
        responsive: true,
        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,
```

```
        title: {
            display: true,
            text: 'Items in Stock by Category'
        },
        scales: {
            yAxes: [{
                ticks: {
                    beginAtZero: true
                }
            }]
        }
    }
});
```

OUTPUT:

Inventory Management System





RESULT: