RAPTOR EXPERIMENTS:

NAME:V.RAKSHMITHA

REG:192465052

DEP:CSE-CYBER SECURITY

Exp 1:.Sum of series

Procedure:

1.Start the program.

2.Read the value of n.

3.Call the procedure SumSeries(n).

4.Inside the procedure, initialize sum to 0 and i to 1.

5.Repeat the loop while i ≤ n.

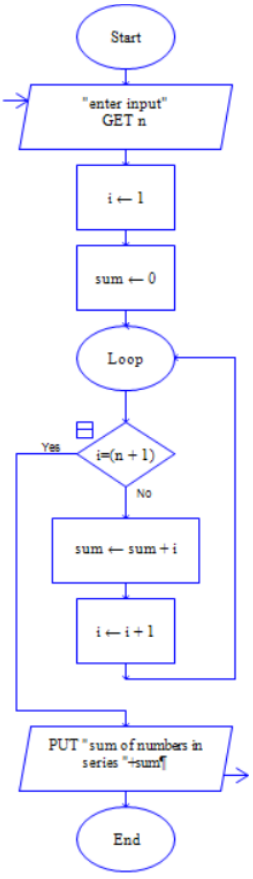
6.Add i to sum.

7.Increment i by 1.

8.End the loop.

9.Return the value of sum to the main program.

10.Display the sum of the series.



.

Exp 2:Binary Numbers

Procedure:

1.Start

2.Read binary number

3.Call procedure

4.Set dec = 0, base = 1

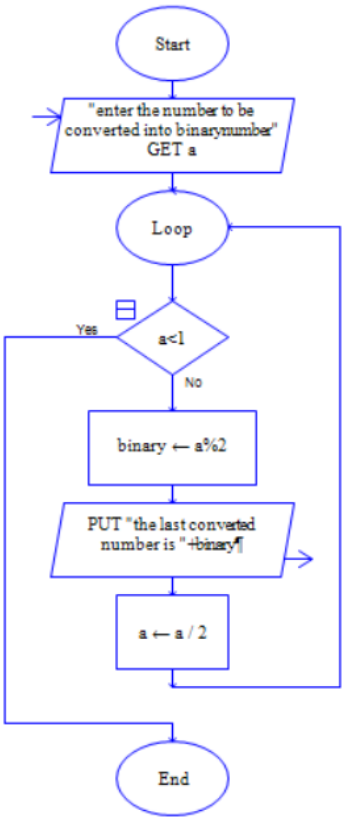
5.While number > 0

6.Add (last digit × base) to dec

7.Multiply base by 2

8.Remove last digit

9. Return dec



Exp3:Armstrong

Procedure:

1.Start

2.Read number n

3.Call procedure Armstrong(n)

4.Set sum = 0, temp = n

5.While temp > 0

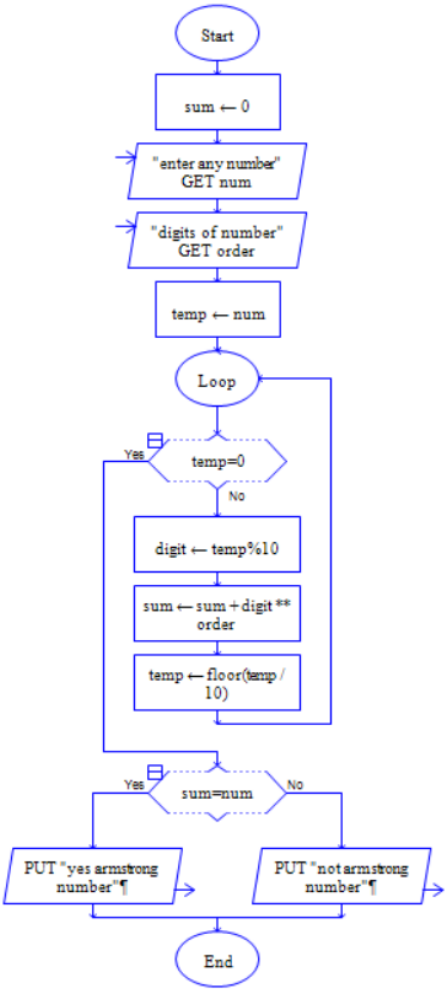
6.Get digit and add digit³ to sum

7.Remove last digit

8.Return sum

9.if sum = n, print **Armstrong**

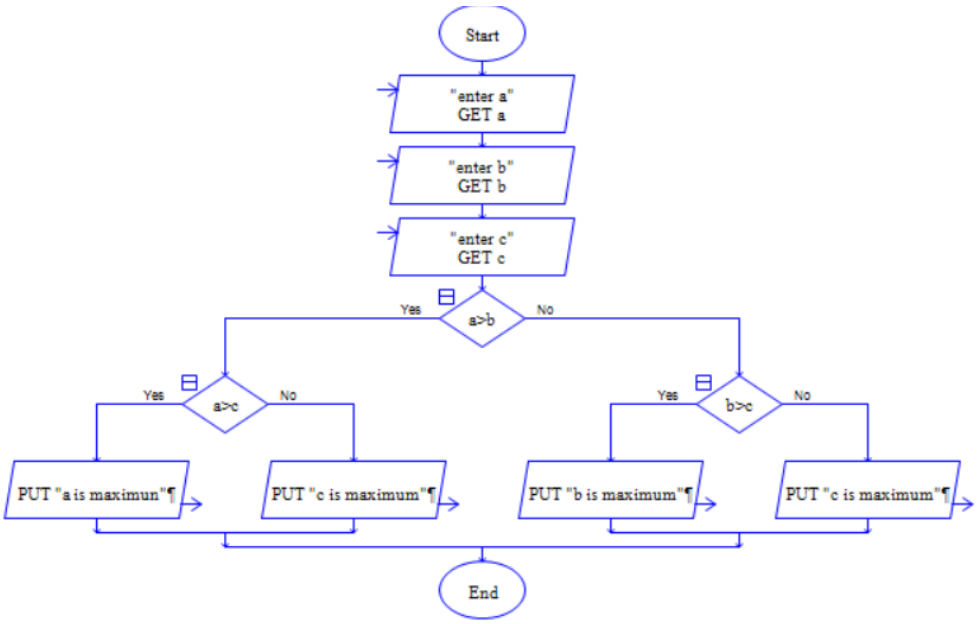
10.Else print **Not Armstrong**



Exp4:Greatest of 3

Procedure:

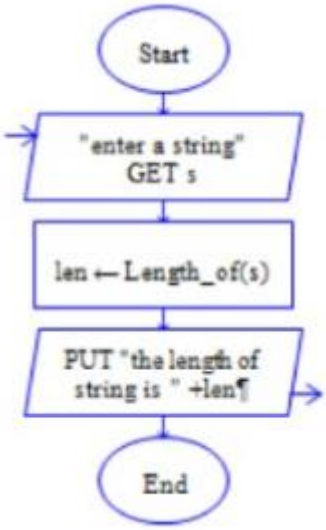
1. Start
2. Read a, b, c
3. Call procedure Greatest(a, b, c)
4. If a ≥ b and a ≥ c, greatest = a
5. Else if b ≥ a and b ≥ c, greatest = b
6. Else greatest = c
7. Return greatest value
8. Display greatest
9. Stop



Exp 5:String Length

Procedure:

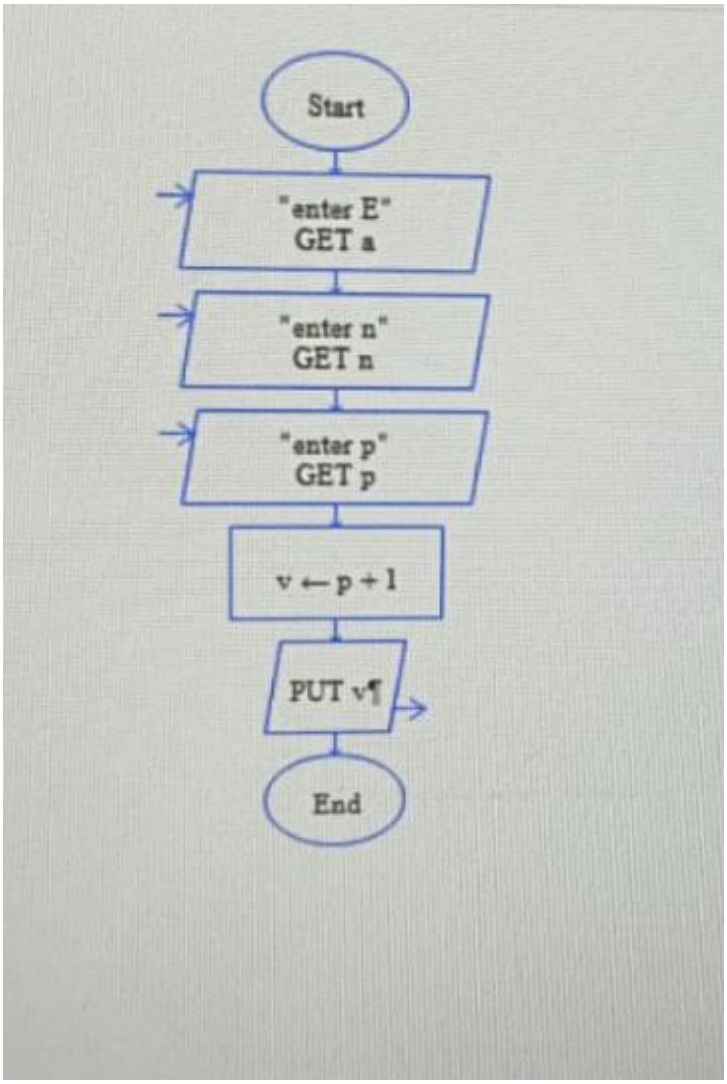
1. Start
2. Read string s
3. Call procedure StrLength(s)
4. Set count = 0
5. While count < Length(s)
6. Increment count
7. Return count
8. Display string length
9. Stop



Exp 6:LowerCase

Procedure:

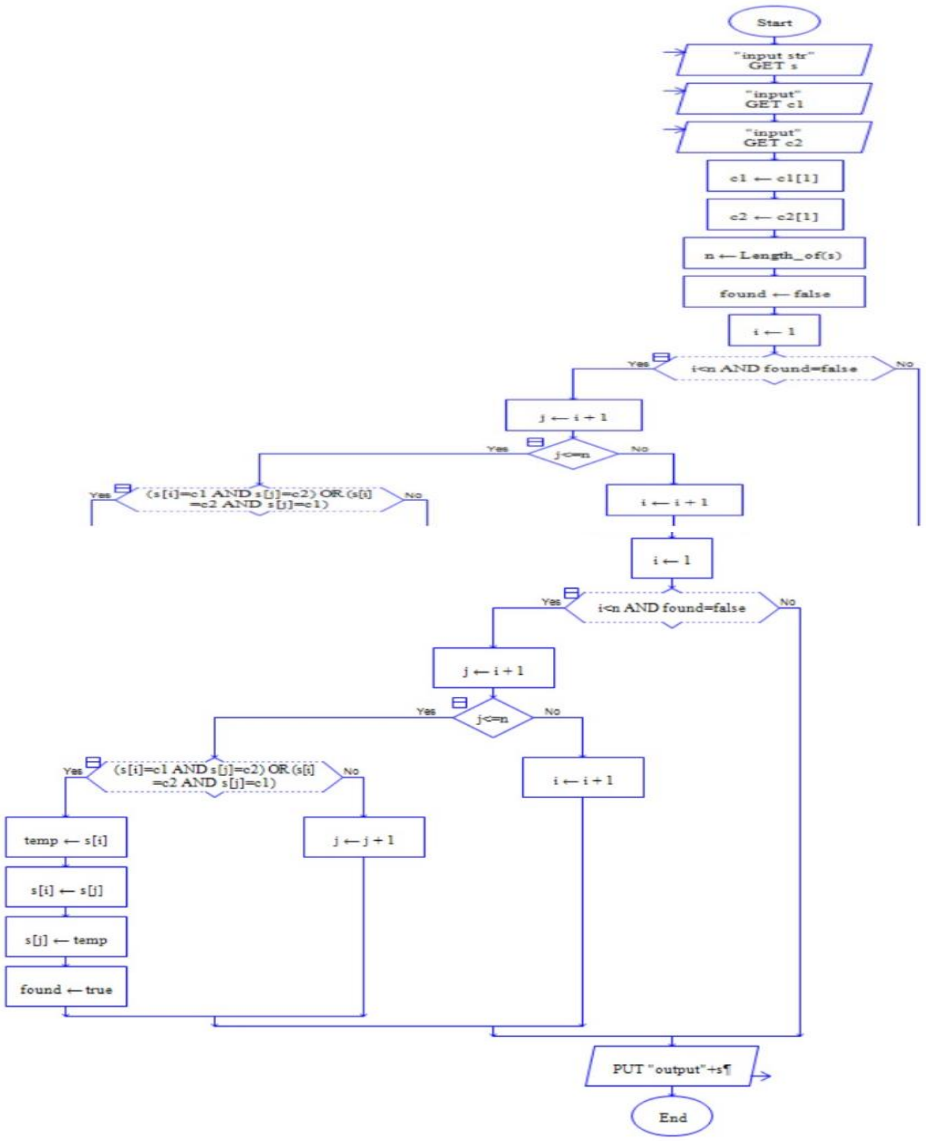
1. Start
2. Read string s
3. Call procedure ToLower(s)
4. Convert string to lowercase
5. Return lowercase string
6. Display result
7. Stop



Exp 7:Cyclometry

Procedure:

1. Start
2. Read radius r
3. Call procedure Circle(r)
4. Calculate area = π × r × r
5. Calculate circumference = 2 × π × r
6. Return area and circumference
7. Display results
8. Stop



GIT HUB EXPERIMENTS:

EXP 25:

1. Create a GitHub account using the credentials.
2. Create a new Repository named “personal-project” (any preferred name)
3. Make sure that to add README file while creating.
4. Click create Repository.
5. Open the Repository and click the code>HTTP.
6. Copy the link shown.
7. Open Git Bash and run the code as follows:

```
git clone https://github.com/your-username/personal-project.git
```

8. Move to the project folder.

```
cd personal-project
```

9. Open the README.md file in Notepad/ VS in local system.
10. Add the project description.

```
# Personal Project
This is my personal project repository created to practice Git
and GitHub operations. It demonstrates repository creation,
cloning, committing, and pushing changes.
```

11. Save the file.
12. Check the status.

```
git status
```

13. Stage the modified file.

```
git add README.md
```

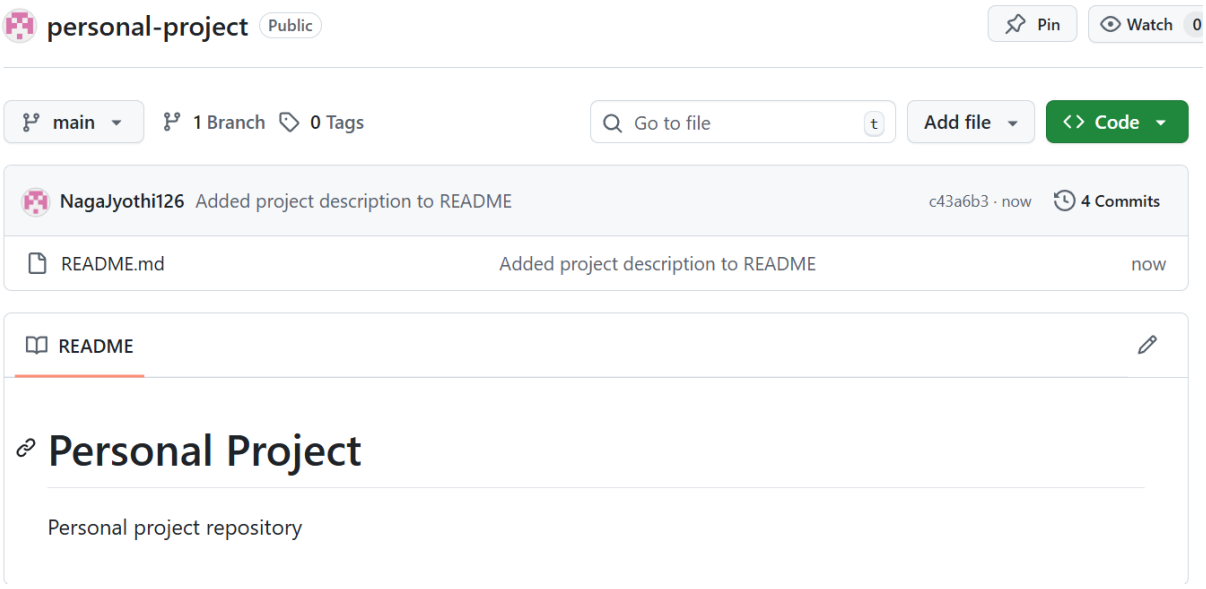
14. Commit the changes made.

```
git commit -m "Added project description to README"
```

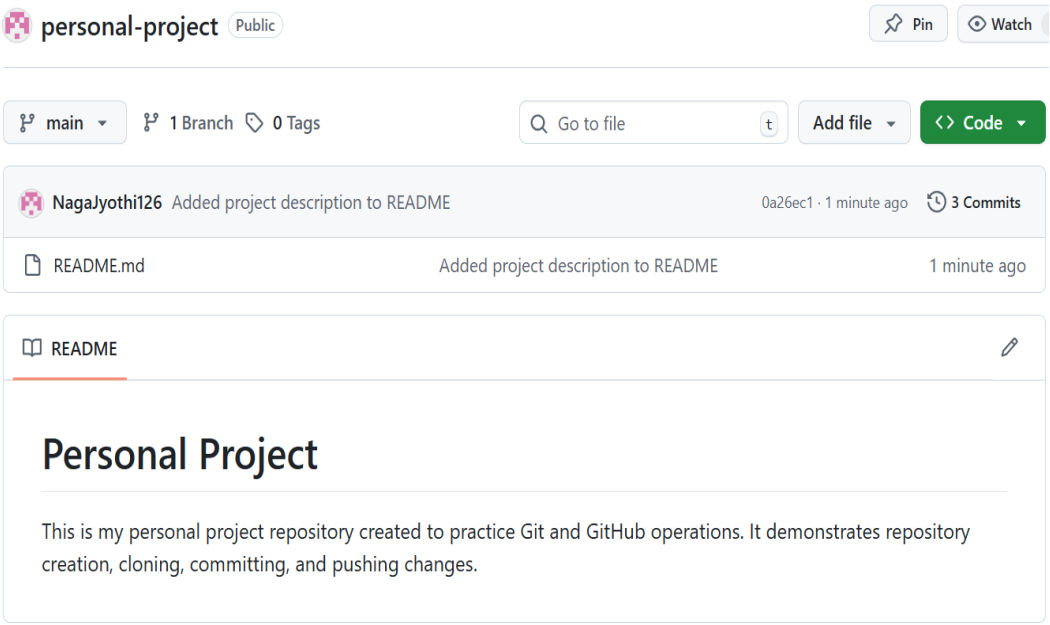
15. Push to the Repository.

```
git push origin main
```

Before Committing Changes



After Committing Changes



Exp 26:

1.Open the web browser and select an existing public GitHub repository.

2.Copy the repository URL from GitHub.

3.Open the command prompt or terminal on the local machine.

4.Use the command git clone <repository_url> to clone the repository.

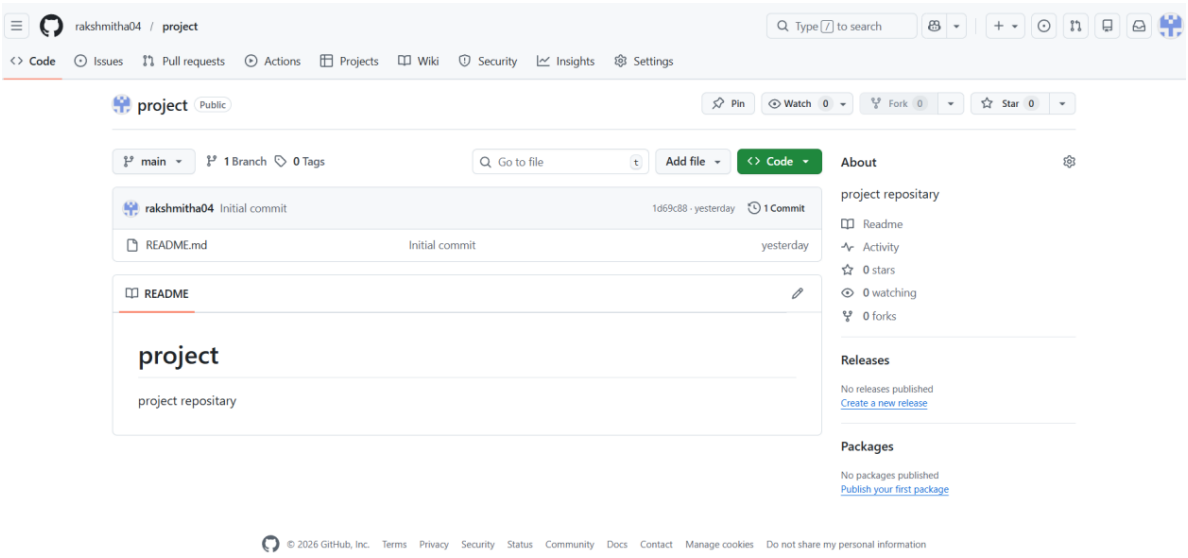
5.Navigate into the cloned repository using the cd <repository_name> command.

6.Open any file from the repository using a text editor.

7.Make a small change such as correcting a typo or modifying a line of text.

8.Save the file after making the changes.

9.Verify that the file has been successfully updated in the local repository.



Exp 27:

Procedure:

1.Open **Git Bash / Terminal**.

2. Clone the repository using
git clone <repository-URL>

3. Go into the cloned folder using
cd repository-name

4. Modify the file (for example, edit README.md) and save it.

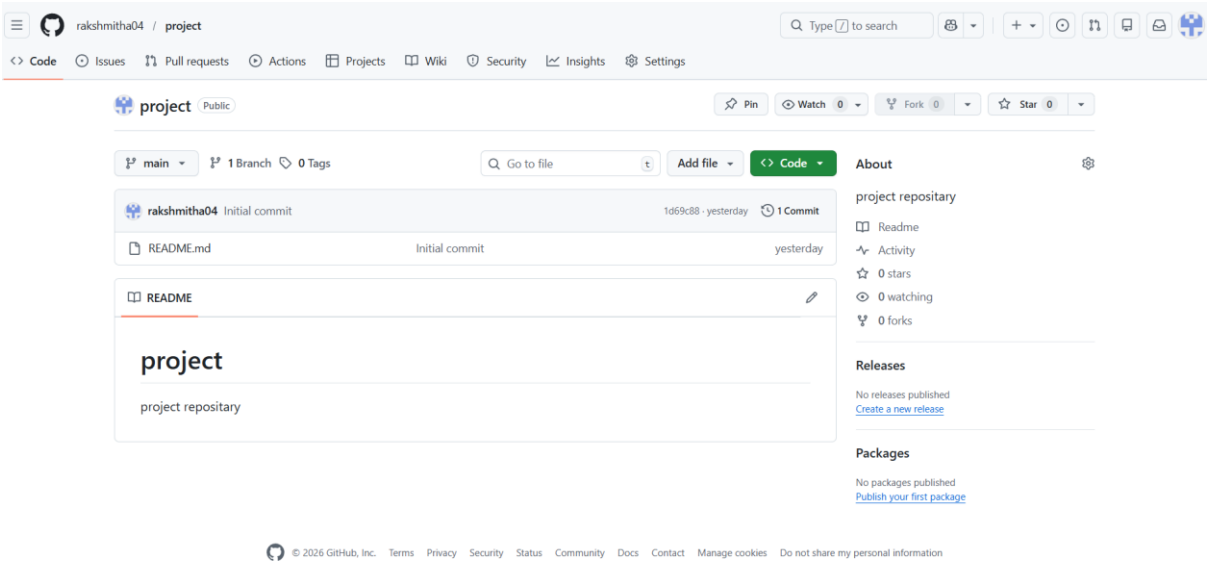
5. Check file status using
git status

6. Stage the changes using
git add README.md

7. Commit the changes using
git commit -m "Updated README"

8. Push the changes to GitHub using
git push origin main

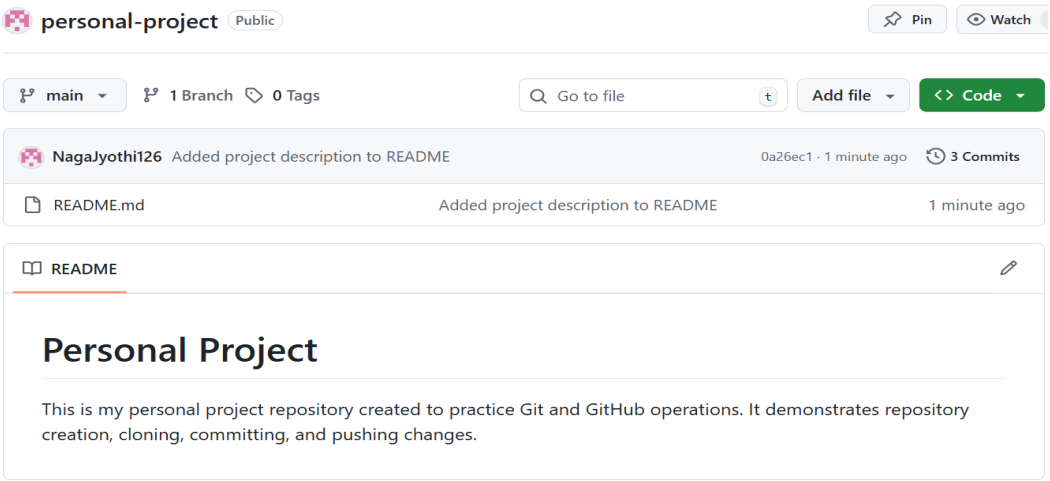
9. Verify the changes on GitHub.



Exp 29:

Procedure:

1. Open **Git Bash / Terminal** and navigate to the repository:
2. cd repository-name
3. Create a new branch named **feature-login**:
4. git checkout -b feature-login
5. Create a file named **login.py** and write a simple login function in it.
6. Check the status of the repository:
7. git status
8. Stage the new file:
9. git add login.py
10. Commit the changes:
11. git commit -m "Added login functionality"
12. Push the **feature-login** branch to GitHub:
13. git push origin feature-login
14. Open the repository on **GitHub**.
15. Click **Pull requests → New pull request**.
16. Select **base branch** as main and **compare branch** as feature-login.
17. Click **Create pull request**.
18. Review the changes and click **Merge pull request**.
19. Click **Confirm merge** to complete the process.



Exp 30:

Procedure:

1. Open the original repository on GitHub and click **Fork** to create a copy in your GitHub account.

2.Open Git Bash / Terminal and clone the forked repository using
git clone <forked-repository-URL>.

3.Navigate into the cloned repository folder using
cd repository-name.

3.create a new branch for your feature using
git checkout -b feature-branch.

4. Implement your feature by adding or modifying the required files and save the changes.

5. Check the status of the repository using
git status.

6. Stage the changes using
git add ..

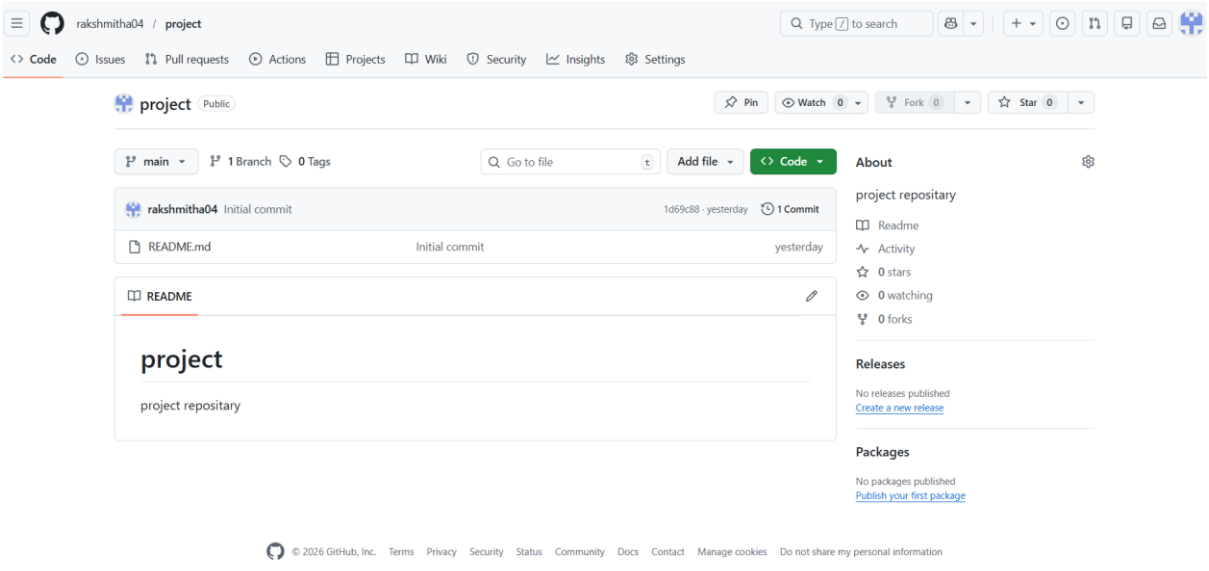
7. Commit the changes using
git commit -m "Added new feature".

8. Push the feature branch to your forked repository using
git push origin feature-branch.

9. Open your forked repository on GitHub and click **Compare & pull request**.

10. Select the base repository as the original repository and base branch as main.

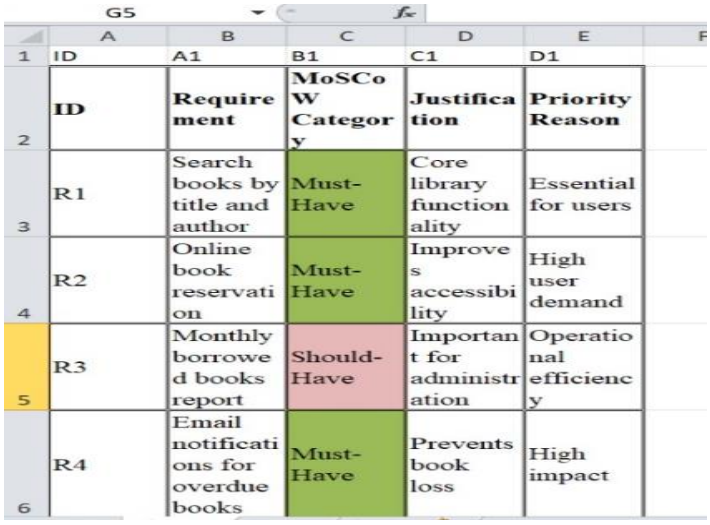
11. Click **Create pull request** to submit your changes.



Exp 10:

Procedure:

1. List all the Library Management System requirements in the first column of a Google Sheet or Excel file.
2. Understand the **MoSCoW method** categories:
    - **Must-Have**: Essential for system operation
    - **Should-Have**: Important but not critical
    - **Could-Have**: Optional enhancements
    - **Won't-Have**: Not included in current phase
3. Analyze each requirement based on:
    - Impact on users and stakeholders
    - Feasibility considering time, budget, and resources
4. Assign a **MoSCoW category** (Must, Should, Could, Won't) to each requirement.
5. Add a justification column explaining why each requirement was placed in that category.
6. Review the categorization to ensure core functionalities are marked as **Must-Have**.
7. Open **Google Sheets or Excel** and create columns:
8. Enter all categorized requirements into the sheet.
9. Save the file as **.xlsx** or ensure it is stored in Google Drive.
10. Submit the completed **Google Sheet or Excel file** as required.



	A	B	C	D	E
1	ID	A1	B1	C1	D1
2	**ID**	**Requirement**	**MoSCoW Category**	**Justification**	**Priority Reason**
3	R1	Search books by title and author	Must-Have	Core library functionality	Essential for users
4	R2	Online book reservation	Must-Have	Improves accessibility	High user demand
5	R3	Monthly borrowed books report	Should-Have	Important for administration	Operational efficiency
6	R4	Email notifications for overdue books	Must-Have	Prevents book loss	High impact

JIRA EXPERIMENTS:

Exp 1:

Procedure:

1.Open a tool such as **Trello, Jira, Notion, GitHub Projects, or a whiteboard/Excel sheet**.

2. Create a new **Kanban Board**.

3. Add three columns:

   - **To Do**
   - **In-Progress**
   - **Done**

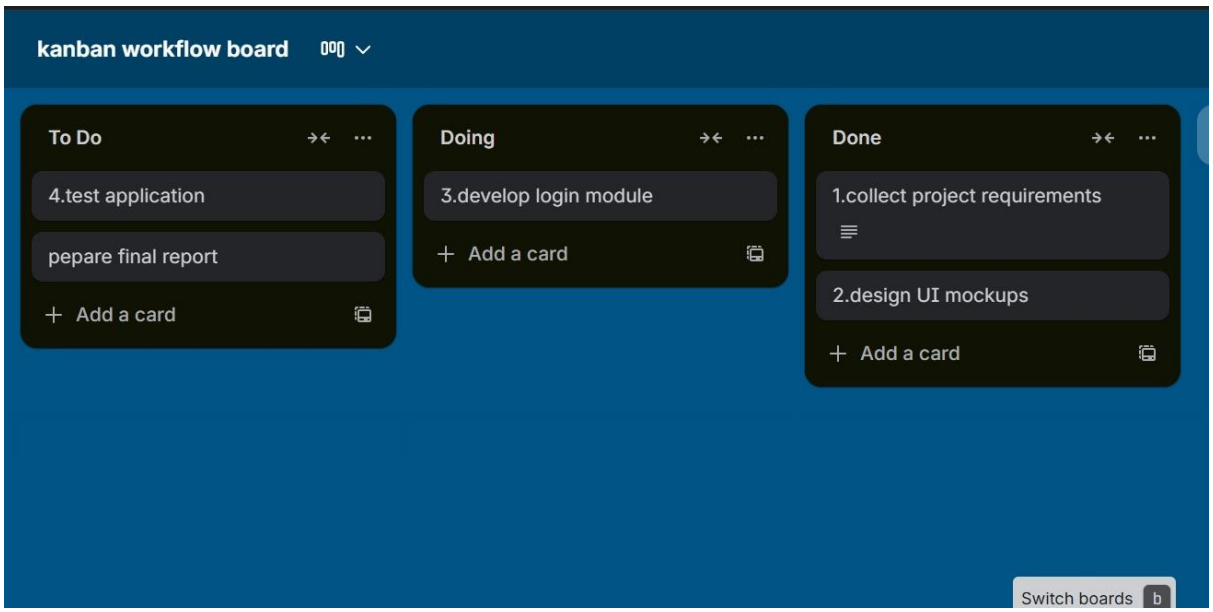
4. Add at least **five sample tasks** under the **To Do** column, for example:

   - Gather project requirements
   - Design system architecture
   - Develop login module
   - Test application features
   - Deploy the application

5. Move tasks from **To Do → In-Progress** when work starts on them.

6. After completing tasks, move them from **In-Progress → Done**.

7. Continue moving tasks across columns to simulate the project workflow.



Exp 11:

Procedure:

1.Open **Confluence** and click **Create** to add a new page.

2. Title the page **"Library Management System Project Overview."**

3. Add a short description of the project.

4. Click **Insert (+)** and select **Jira Issue/Filter** (Jira macro).

5. Search and embed **at least 5 Jira issues** related to the project.

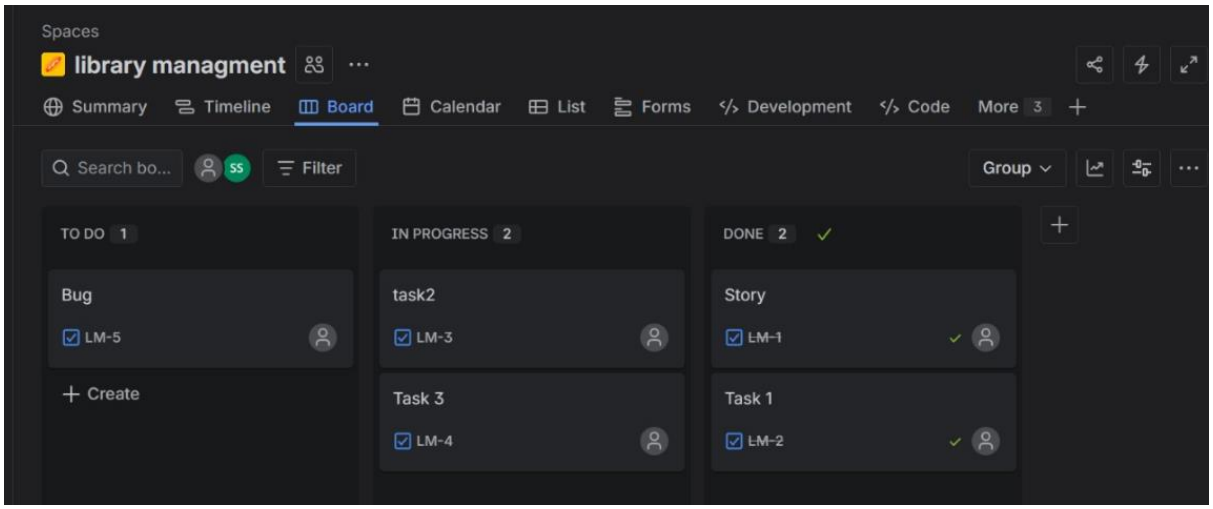
6. Configure the macro to display **issue status** (To Do, In Progress, Done).

7. Arrange the Jira issues in list or table view.

8. Insert a **Progress Bar** macro on the page.

9. Set the progress bar to show **percentage of completed tasks**.

10. Publish the page and take a **screenshot** for submission



Exp 9:

Procedure:

1.Open **Jira** and click **Create project**.

2.Select **Scrum** as the project template and create the project.

3.Open the **Backlog** section of the Scrum project.

4.Add at least **5 backlog items**, for example:

- Create user registration page
- Develop API for login
- Design database schema
- Implement book search feature
- Test user authentication

5. **Prioritize the backlog** by dragging items from highest to lowest priority.

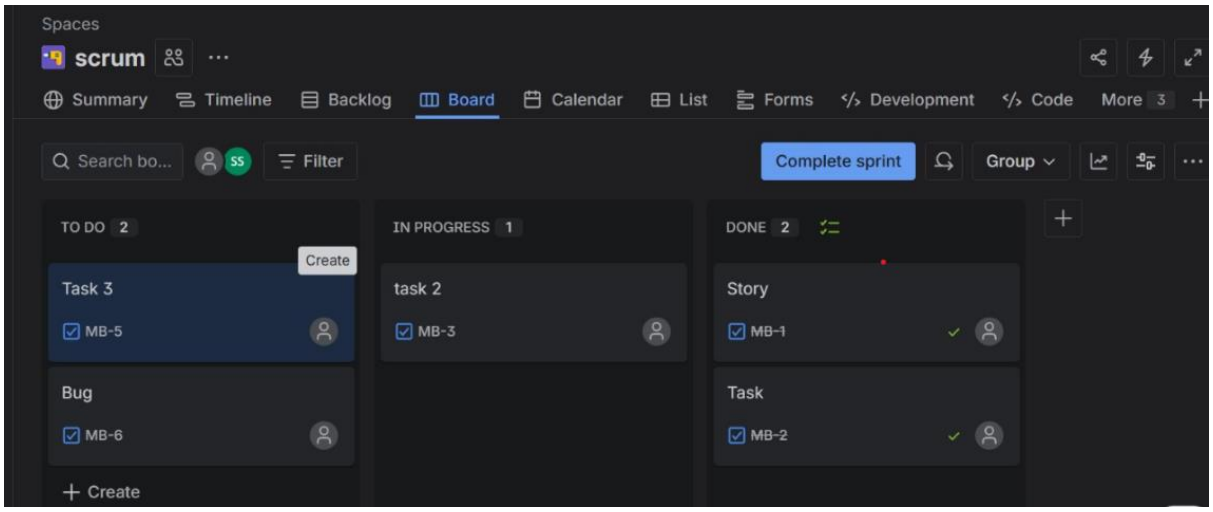
6. Click **Create Sprint** to create a new sprint.

7. Set the sprint duration to **1 week**.

8. Move selected backlog items into the sprint.

9. Click **Start Sprint** and take a **screenshot of the sprint board at the start**.

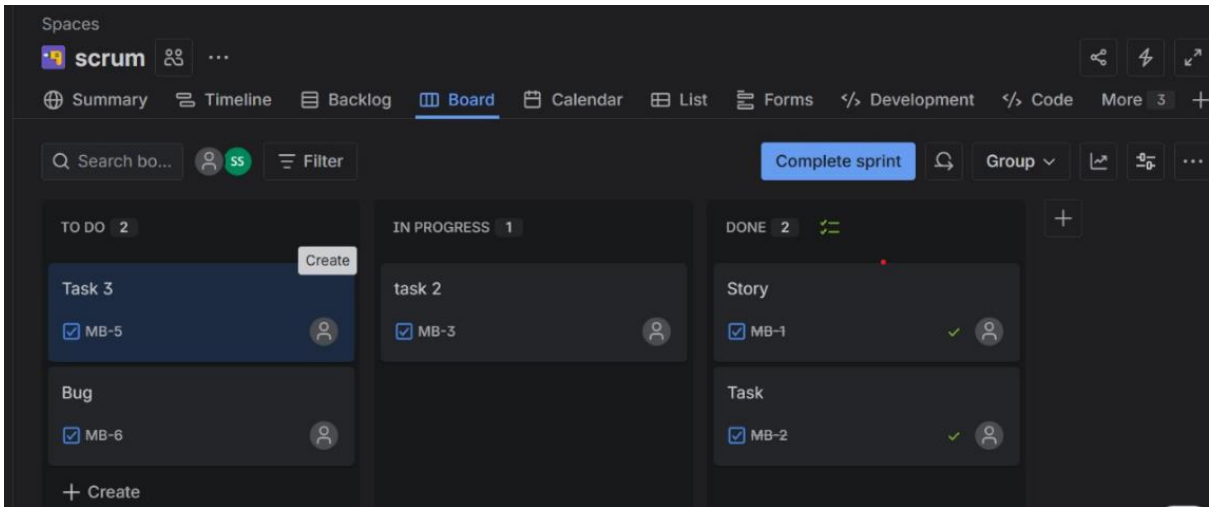
10. After moving tasks to **Done**, take a **screenshot of the sprint board at the end** and submit it.



Exp 12:

Procedure:

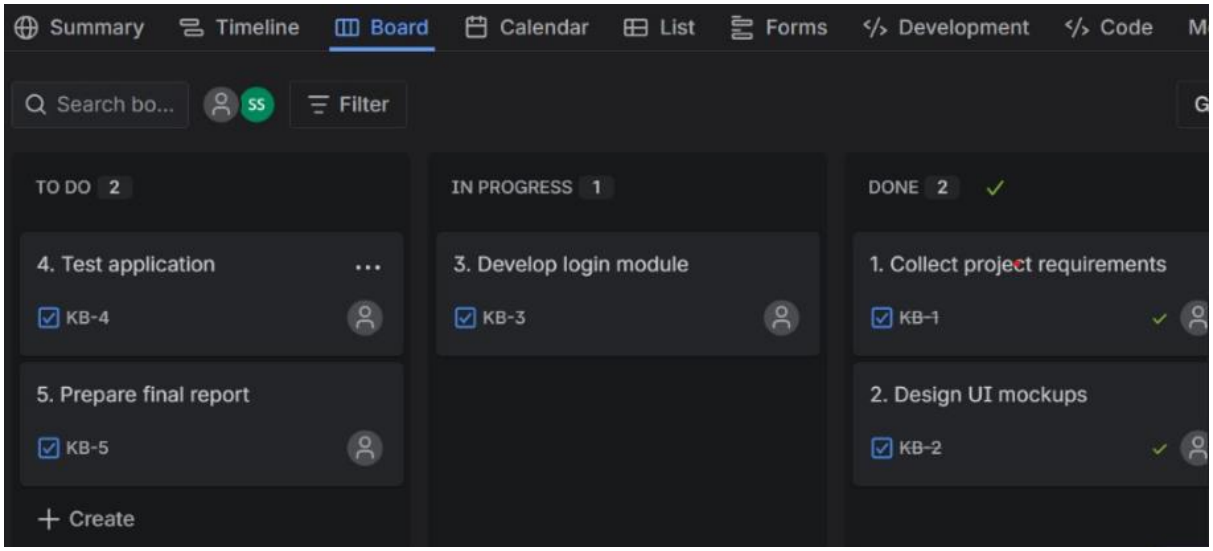
1. Open **Jira** and create a new project (Scrum or Kanban).
2. Add the requirements as **issues/stories**:
    - User Login and Role Assignment
    - Task Creation and Assignment
    - Task Prioritization and Deadlines
    - Progress Tracking and Reporting
3. Open the **Backlog** and view all added issues.
4. Assign **MoSCoW priorities** using Priority or Labels:
    - Must-Have: User Login and Role Assignment
    - Must-Have: Task Creation and Assignment
    - Should-Have: Task Prioritization and Deadlines
    - Should-Have: Progress Tracking and Reporting
5. Add **Kano categories** in the issue description or labels:
    - Basic (Must-Be): Login and Task Creation
    - Performance: Prioritization, Deadlines, and Reporting
6. Reorder the backlog in Jira based on MoSCoW priority.
7. Review priorities to ensure essential features are at the top.
8. Save and update the Jira project



Exp 13:

Procedure:

1. Create a **Jira project** (Scrum or Kanban).
2. Add the requirements as **issues/stories**:
    - Course Enrollment and Registration
    - Video Lecture Streaming
    - Interactive Quizzes and Assignments
    - Progress Tracking Dashboard
    - Peer-to-Peer Discussion Forums
    - Certificate Generation
3. Assign **MoSCoW priorities** (Must, Should, Could) to each requirement.
4. Assign **Kano categories** (Basic, Performance, Excitement) using labels or description.
5. Reorder the backlog based on **MoSCoW priority**.
6. Review and update priorities; use this for sprint planning and development.



FIGMA EXPERIMENTS:

Exp 2:

Procedure:

1. Open **Figma** and create a **New File**.

2. Use the **Frame Tool** to create multiple screens: Home, Select Bus, Passenger Details, Payment, Confirmation.

3.Design **Home Screen** with input fields: From, To, Date, and Search Button.

4. Design **Select Bus & Seat Screen** with available buses and seat selection grid.

5. Design **Passenger Details Screen** with Name, Age, Gender, Contact Info.

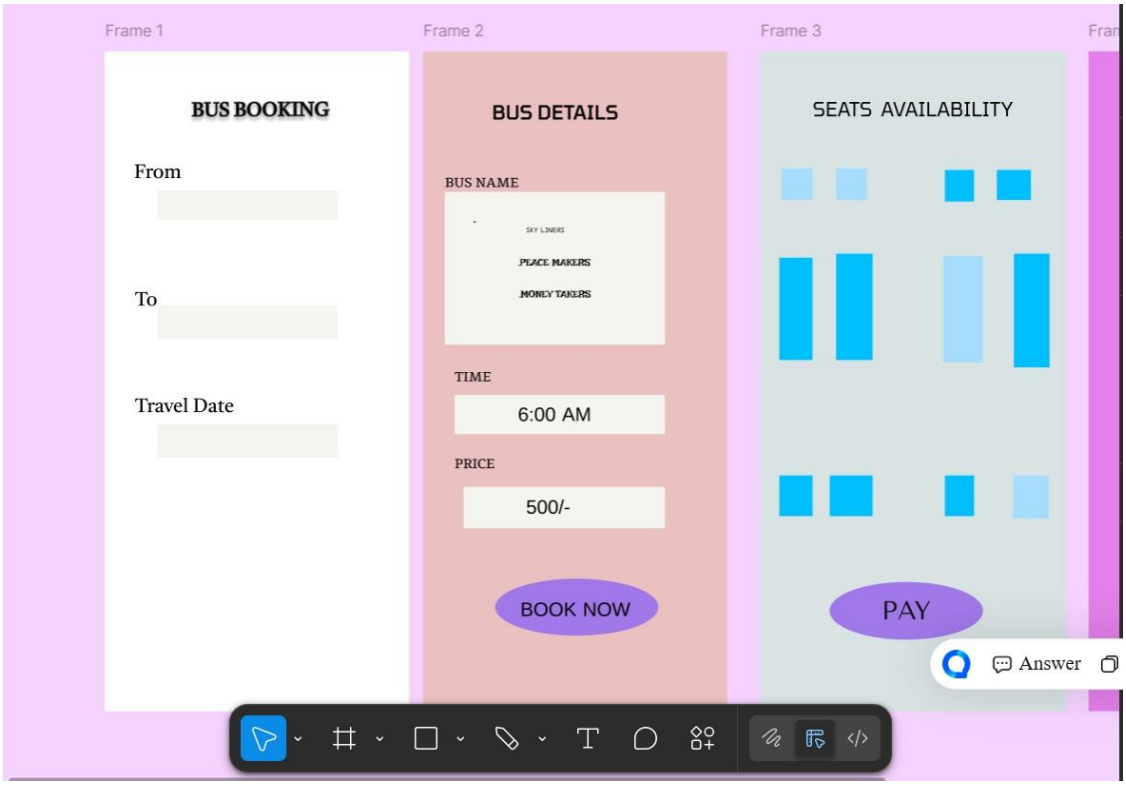
6. Design **Payment Screen** with payment options and Pay Now button.

7. Design **Booking Confirmation Screen** showing ticket details and Download/Share buttons.

8. Connect screens using **Prototype mode** to simulate the booking flow.

9. Add **colors, labels, and icons** to improve UI clarity.

10. Preview in **Present Mode** and take a **screenshot** or share the Figma link

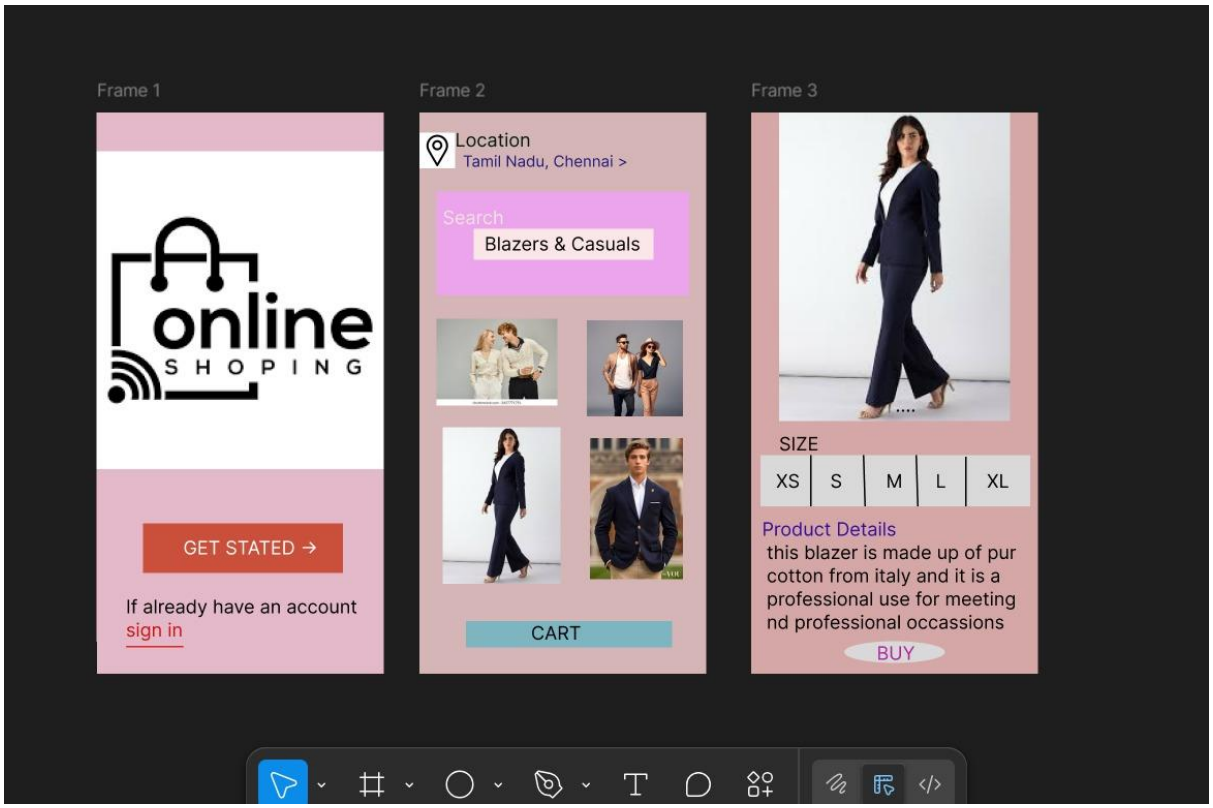


Exp 3:

Procedure:

1.Open **Figma** and create a **New File**.

2. Create frames for screens: Home, Product Details, Cart/Checkout, User Profile.
3. Design **Home Screen** with search bar, categories, and product cards.
4. Design **Product Details Screen** with image, description, price, and Add to Cart button.
5. Design **Cart/Checkout Screen** with products, total price, and Checkout button.
6. Design **User Profile/Settings Screen** with user info and order history.
7. Use **Prototype mode** to connect buttons and simulate app flow.
8. Add **colors, icons, and fonts** for UI clarity and realism.
9. Preview in **Present Mode** and share the prototype link with stakeholders.
10. Collect feedback and **iterate the design** based on stakeholder suggestions



UMBRELLO EXPERIMENTS:

1. Open **Umbrello** and create a **New Project**.

2. **Use Case Diagram:**

- Click **Diagram → New Diagram → Use Case Diagram**.
- Add **Actors** (e.g., Passenger, Admin, Airline Staff).
- Add **Use Cases** (e.g., Register, Login, Book Flight, Cancel Booking).
- Connect **Actors to Use Cases** using **Association tool**.
- Add **Include / Extend relationships** if needed.
- Arrange diagram neatly.

3. **Activity Diagram:**

- Click **Diagram → New Diagram → Activity Diagram**.
- Add a **Start node** to indicate the beginning.
- Add **Activity nodes** for each process step (e.g., Enter Details, Select Flight, Make Payment).
- Connect activities using **Control Flows / Arrows** to show sequence.
- Add **Decision nodes** for branching (e.g., Payment Successful?).
- Add **End node** for process completion.
- Arrange for clarity.

4.**Class Diagram:**

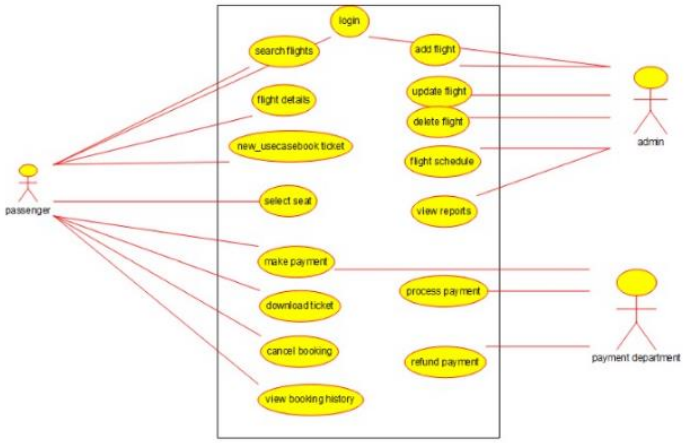
- Click **Diagram → New Diagram → Class Diagram**.
- Add **Classes** (e.g., User, Flight, Booking, Payment).
- Add **Attributes** and **Methods** for each class.
- Connect classes using **Relationships**:
    - Association (User → Booking)
    - Inheritance (Admin → User)
    - Aggregation/Composition (Flight → Seat)
- Arrange classes for readability.

5. **Save the Project**

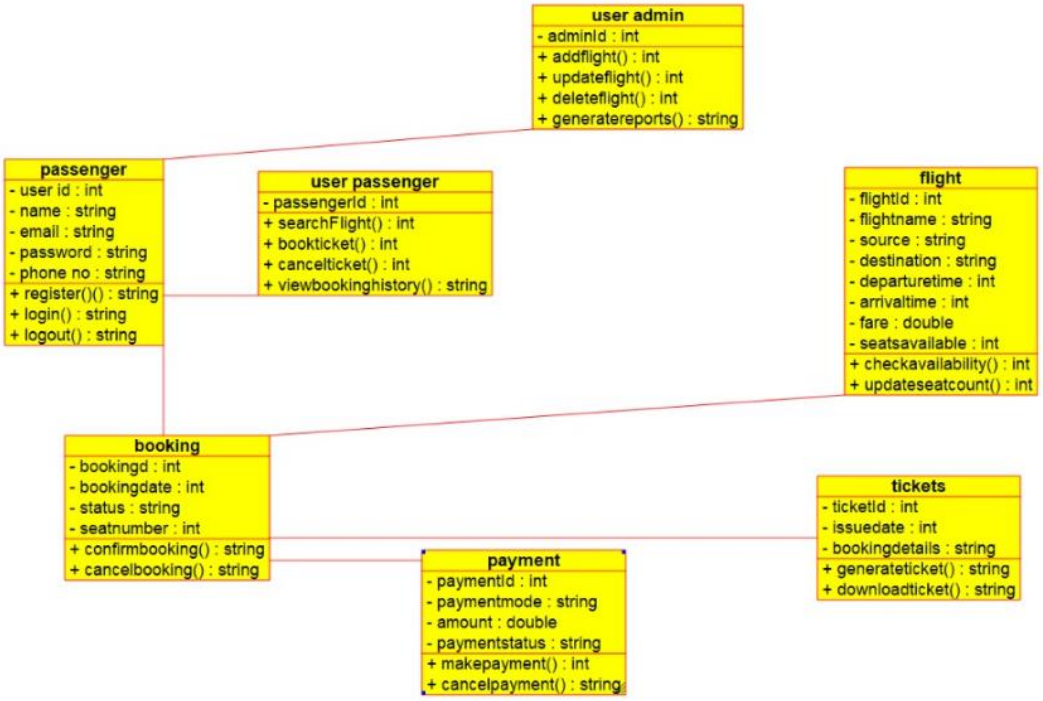
- Click **File → Save As** and give a descriptive name.

6. **Export Diagrams (Optional)**
   - Export each diagram as an image (PNG/JPG/SVG) for reports or presentations.
7. **Review**
   - Ensure all actors, processes, and classes are represented clearly.
   - Check associations, flows, and relationships for correctness.

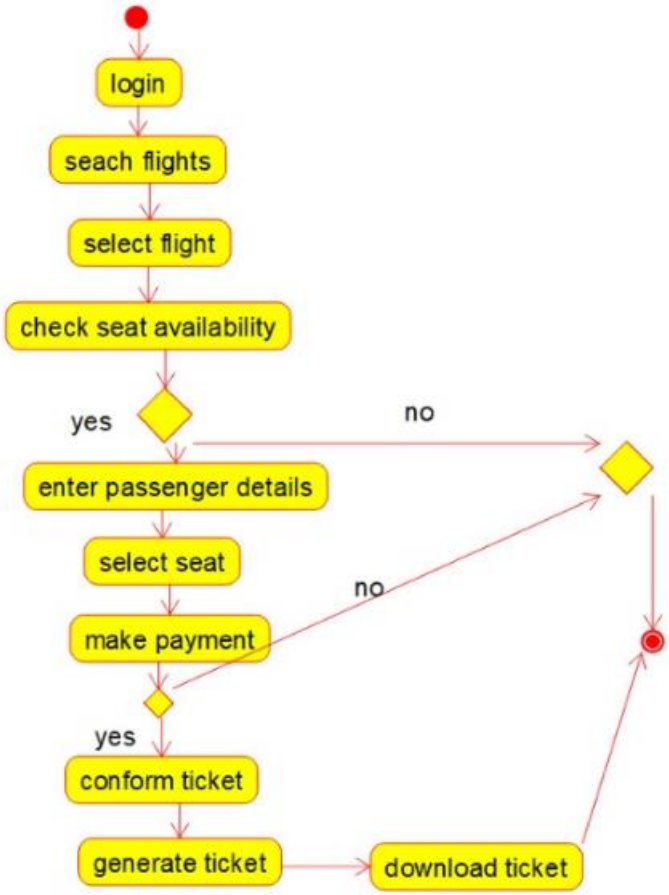
USE CASE:



CLASS DIAGRAM:



ACTIVITY DIAGRAM:



DOCKER EXPERIMENTS:

Exp 21:

1. Create a project folder and write a simple Flask app (app.py) for a To-Do list.

2. Create requirements.txt with required packages (e.g., Flask).

3. Create a Dockerfile with Python base image, copy files, install requirements, expose port, add run app.

4. Open terminal and navigate to project folder.

5. Build Docker image: docker build -t todo-app .

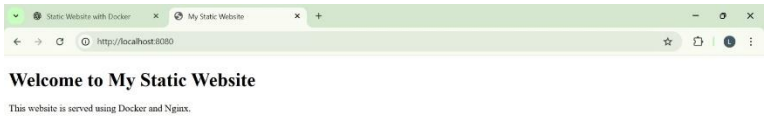
6. Run Docker container: docker run -p 5000:5000 todo-app

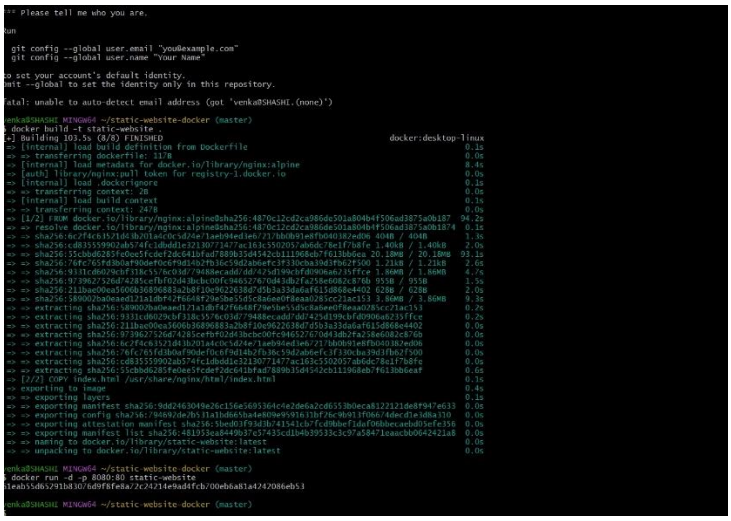
7. Test the application inside the container (using browser, Postman, or curl).

8. Log in to Docker Hub: docker login

9. Tag Docker image: docker tag todo-app username/todo-app:latest

10. Push Docker image to Docker Hub: docker push username/todo-app:latest





Exp 22:

Procedure:

1. Create a folder and add index.html for your static website.
2. Create a Dockerfile using Nginx and copy index.html into it.
3. Build Docker image: docker build -t static-site .
4. Log in to Docker Hub: docker login
5. Tag the image: docker tag static-site username/static-site:latest
6. Push the image: docker push username/static-site:latest
7. On another machine, log in: docker login
8. Pull the image: docker pull username/static-site:latest
9. Run the container: docker run -d -p 80:80 username/static-site:latest
10. Verify by opening http://localhost in a browser.