1. Create a Kanban Board to visualize the task.

Aim: To create and simulate a Kanban board for task visualization and workflow management

Apparatus required: A computer with internet acces, Jira or Trello software.

Procedure: Open jira or Trello and create a new project or board.

Create three columns: To Do, in progress and do

Add at least 5 sample tasks under the To D column.

Move tasks to the In progress and done column as they start.

Shift completed tasks to the 'Done' colum

Save and take a screenshot of the fin Kanban board.

Result:

A functional Kanban board was created, simulating task workflows effectively.

2. Sketch a simple prototype of a bus ticket booking system.

Aim: To design a prototype of a bus ticket booking system to visualize its functionality.

Apparatus required: Figma tool.

Procedure: Open figma and create a new project
Design a simple interface with features like bu
- selection, ticket booking, and payment.
Add buttons and text fields for data input
Include a summary page displaying booking
details.
Save the design and export the prototyp
as a PNG or PDF file.
Submit the file as proof of completion.

Result:

A functional prototype of a bus ticket booking system was created using figma.

3. Prototype for an e-commerce mobile app.

Aim: To design a prototype of an e-commerce mobile app interface to gather stakeholder feedback.

Apparatus required: Figma.

Procedure: Open figma and create a new project

Design screens for login, product display, shopping cart and checkout.

Incorporate sample buttons for functionality like add to cart and checkout.

Use interactive elements for navigation b/w screens.

Share the design with stakeholders for review and feedback.

Make modifications based on feedback and finalize the prototype.

result:

An interactive e-commerce app prototype was created and refined based on stakeholder input.

4. Create a Scrum project in jira.

Aim: To understand the Scrum methodology by creating and managing a project in jira.

procedure:

Open jira and create a new Scrum project

Add at least 5 times to the backlog.

Prioritize the backlog items and, create a 1-week sprint.

Move backlog items into the sprint and start it.

Track progress during the sprint Using the Scrum board.

capture screenshots at the beginning and end of the sprint.

Result: A Scrum project was Successfully created, managed and completed in jira.

5. Categorize Library management system requirements using MoSCoW method.

Aim:

To categorize the requirements of a Library Management System using the MoSCoW method based on user impact and feasibility.

Apparatus required:

Computer with spreadsheet software.

Documented System requirements.

Procedure:

List all the provided requirements for the Library management System in a Spread sheet.

Define criteria for Must-have, Should-have, Could-have, and Won't-have features.

Categorize each requirements based on impact on stakeholders and feasibility within time, budget and resources constraints.

Verify the categorization by reviewing Stakeholder feedback.

Result: The Library Management System requirements were effectively categorized and justified using MoSCoW method.

6. Link Jira Tasks with confluence.

Aim: To streamline task tracking by linking jira issues with confluence accounts for project management.

Apparatus required:

Computer with internet acess.

Jira and confluence accounts

Procedure:

Create a new page in confluence titled Library management System project overview.

In jira, create at least 5 issues related to the system.

Use the jira macro to embed these issues into the confluence page, displaying their status

Add a progress bar to track the completion of task visually.

Save the confluence page and Screenshot.

result:

The project tasks were sucessfully linked between jira and confluence, enabling efficient tracking and monitoring.

7. Prioritize Task Management System features using MoSCoW and Kano models.

Aim: To prioritize features of a task management system using MoSCoW and Kano models.

Apparatus required:

Computer with internet access.

Jira Software.

procedure:

List the system requirements: User Login, Task assignment, prioritization and progress Tracking.

Categorize the features using the features using the MoSCoW model.

Apply the Kano Model to assess user satisfaction levels for each feature.

Input the prioritized features into Jira as tasks.

create a backlog and rank Tasks based on priority.

Result: The task Management System features were prioritized effectively using the MoSCoW and Kano models.

8. Categorize Online learning platform features Using MoSCoW and kano Models.

Aim: To prioritize features of an online learning platform using the MoSCoW and kano method

Apparatus required:

Jira software

Computer with internet access.

Procedure:

List the required features: Course enrollment, video streaming, Quizzes, progress Tracking, forums and certificates.

Categorize each feature using the MoSCoW method.

Use the KanoModel to assess user Satisfaction and desirability for each feature.

Create a sprint plan and assign tasks for implementation.

Result:

The features of the online learning platform were prioritized using MoSCoW and Kanomodel and tasks were organized for development.

9. Demonstrate collaborative work using Git/Github.

Aim:

To demonstrate collaboration using the fork and pull request workflow on Github.

Apparatus required:

Git and Github accounts
computer with Git installed.

Procedure:

Fork a public Github repository
clone the forked repository to your local machine.
create a new branch for you changes.
make modifications or add a new feature
commit your changes and push the branch to Github.
create a pull request to merge the branch into the main repository.
respond to feedback, if any, and finalize the pull request.

Result: Collaborative work was demonstrated using the Git fork-and-pull request workfl

10. Resolve merge conflicts using Git.

Aim: To learn how to handle and resolve merge conflicts during collaboration in Git

Apparatus required:

Git and Github accounts

Computer with Git installed.

Procedure:

clone a shared repository to your local machine.

Create a new branch and switch to it

make changes to a file.

commit and push the changes to the repository.

Pull the latest changes form the main branch before merging;)

Merge the main branch into your main branch before merging.

Push the resolved branch to Github and create a pull requested.

result: Merge conflicts were resolved sucessfully, and changes were merged into the main branch.

11. Containerize and serve a static website using Docker.

Aim: To containerize a static website and serve it using Docker.

Apparatus required:

computer with Docker installed.

Basic knowledge of HTML and Dockerfile.

Procedure:

Create a simple static website with an file

write a Dockerfile to serve the website using Nginx.

Build the Docker image using the comman docker build -t static-website.

Access the website in a browser by visiti http://localhost:8080.

Test the container functionality and stop it after completion.

Result:

The static website was successfully containerized and served using Docker.

12. Deploy a flask API using Docker and Kubernetes.

Aim:

To develop a simple flask API, containerize it using Docker, and deploy it with kubernetes

Apparatus required:

Python and flask, Docker and kubernetes CLI, kubernetes cluster.

Procedure:

Write a basic flask API with at least one endpoint. Create a Dockerfile to containerize the flask application. Build and push the Docker image to Docker hub.

Create kubernetes YAML manifests for deployment and service. Apply the manifests using kubectl apply deployment.yaml.

Access the API using the assigned nodeport

Result:

The flask API was successfully deployed and accessed via kubernetes.

13. Set up a CI/CD pipeline using Jenkins.

Aim:

To automate the building, testing and deployment of a containerized application using jenkins.

Apparatus required:

Jenkins installed on a server or local machine

Docker and sample application.

Procedure:

Install and configure jenkins on a server or local machine.

Write a Dockerfile for a sample application

Create a Jenkinsfile to define the build, test, and deployment stages.

Configure jenkins to trigger builds on code commits or pull requests.

Test the pipeline by committing code to the repository.

Verify the deployment of the application

result:

A CI/CD pipeline was successfully set up and verified using Jenkins.

15. Implent Continuous Deployment Using Github Actions.

Aim: To automate the deployment of a Dockerized application using Github action

Apparatus required:

Github repository with a Dockerized application. Github action.

Procedure:

Create a new Github repository and push the Dockerized application code.

Write a Github actions workflow file to automate builds and deployments.

Configure the workflow to build the docker image and push it to docker hub.

Automate deployment to a cloud platfor like AWS or Google kubernetes engine.

result:

Continous deployment was Successfully implemented and tested using Github actions.

16. Create a Github repository and implement version Control.

Aim: To demonstrate version control in Github by setting up a repository and managing branches.

Apparatus required:

Github account, computer with Git installed.

procedure:

Create a new github repository and initialize it with a readme.md file.

clone the repository to your local machine

Create separate branches for individual project modules.

make changes to the files in each branch and commit them.

Open pull requests and merge branches after code reviews.

Document the workflow in the readme.md file

result:

Version control was successfully implemented using Github with multiple branches and pull requests.

19. Containerize a python flask application using Github with multiple branches and pull requests.

Aim: To containerize a simple python flask application using Docker.

Apparatus required:

Python and flask installed.

Docker software.

Procedure:

Create a simple flask application for a "To-Do" list.

Write a Dockerfile to containerize the flask application.

Build the Docker image using docker build . flask-todo-app.

Run the Docker container locally with docker run -p 5000:5000 flask-todo-app.

Push the Docker image to Docker hub.

result:

The python flask application was successfully containerized and deployed using Docker.

4. Push and pull Docker images using Docker hub.

Aim: To demonstrate the process of pushing and pulling Docker images using Docker hub.

Apparatus required:

Docker installed, Docker hub account.

Procedure:

Create a Docker image for a static HTML website.

Tag the image using docker tag static-website: lastest /static-website.

Push the image to Docker hub using docker push.

Pull the image on another machine using docker pull.

Run the pulled image to verify functionality.

Document the steps and submit screenshots.

Result:

Docker image were successfully pushed to and pulled from Docker hub.

23. Deploy a multi-container application using Kubernetes.

Aim: To deploy a multi-container application with frontend, backend components using Kubernetes.

Apparatus required:

Dockerized multi-container application.

Kubernetes CLI and cluster.

procedure:

Create Docker images for the frontend, backend, and database.

Write Kubernetes YAML files for deployments and services configurations.

Deploys the application using Kubectl apply -f deployment.yaml.

Monitor pods and services with kubectl get pods and Kubectl get services.

result:

The multi-container application was deployed and scaled successfully using Kubernetes.

Eg. Create a CI/CD pipeline Using Github actions.

Aim: To automate the testing and deployment of a containerized application using Github actions.

Apparatus required:

Github repository, Github actions enabled.

Procedured:

Set up a Github repository for a containerized flask application.

Create a Github actions workflow file.

Define steps to build, test, and push the Docker image to Docker hub.

Trigger the workflow by pushing changes to the repository.

verify the deployment and Submit workflow logs.

result:

The CI/CD pipeline was Successfully implemented and tested using Github actions.

25. Initialize and Update a Github repository

Aim: To set up and update a Github repository for a personal project

Apparatus required:

Github account, computer with Git installed.

Procedure:

Create a Github repository and initialize it with a readme.md file.

clone the repository to your local machine using git clone.

edit the readme.md file to add project details.

Stage and commit changes with git add and git commit.

Push the changes to github using git push.

result:

The Github repository was successfully initialized, updated, and documented.