

**RAJALAKSHMI ENGINEERING
COLLEGE**
**RAJALAKSHMI NAGAR, THANDALAM – 602
105**



<p>CS23432 SOFTWARE CONSTRUCTION LABORATORY</p>
<p>Laboratory Note Book</p>

Name: JANANI V

Year / Branch / Section: 2nd YEAR / AIML/AC

University Register No. :2116231501066

College Roll No: 231501066

Semester: IV SEMESTER

Academic Year: 2024-2025

EX NO: 1 STUDY OF AZURE DEVOPS

AIM:

To study how to create an agile project in Azure DevOps environment.

STUDY:

Azure DevOps is a cloud-based platform by Microsoft that provides tools for DevOps practices, including CI/CD pipelines, version control, agile planning, testing, and monitoring. It supports teams in automating software development and deployment.

1. Understanding Azure DevOps

Azure DevOps consists of five key services:

1.1 Azure Repos (Version Control)

- Supports Git repositories and Team Foundation Version Control (TFVC).
- Provides features like branching, pull requests, and code reviews.

1.2 Azure Pipelines (CI/CD)

- Automates build, test, and deployment processes.
- Supports multi-platform builds (Windows, Linux, macOS).
- Works with Docker, Kubernetes, Terraform, and cloud providers (Azure, AWS, GCP).

1.3 Azure Boards (Agile Project Management)

- Manages work using Kanban boards, Scrum boards, and dashboards.
- Tracks user stories, tasks, bugs, sprints, and releases.

1.4 Azure Test Plans (Testing)

- Provides manual, exploratory, and automated testing.
- Supports test case management and tracking.

1.5 Azure Artifacts (Package Management)

- Stores and manages NuGet, npm, Maven, and Python packages
- Enables versioning and secure access to dependencies.

Getting Started with Azure DevOps:

Step 1: Create an Azure DevOps Account Visit Azure DevOps.

- Sign in with a Microsoft Account.
- Create an Organization and a Project.

Step 2: Set Up a Repository (Azure Repos) Navigate to Repos.

- Choose Git or TFVC for version control.

- Clone the repository and push your code.

Step 3: Configure a CI/CD Pipeline (Azure Pipelines) Go to Pipelines→ New Pipeline.

- Select a source code repository (Azure Repos, GitHub, etc.).
- Define the pipeline using YAML or the Classic Editor.
- Run the pipeline to build and deploy the application.

Step 4: Manage Work with Azure Boards Navigate to Boards.

- Create work items, user stories, and tasks.
- Organize sprints and track progress.

Step 5: Implement Testing (Azure Test Plans) Go to Test Plans. • Create and run test cases

- View test results and track bugs.

RESULT:

Thus, the study for the given problem statement was successfully completed.

EX NO: 2 WRITING PROBLEM STATEMENT

AIM:

To prepare PROBLEM STATEMENT for your given project.

PROBLEM STATEMENT:

Fitness App:

- Create an app that allows users to log their workouts, track progress, and receive workout recommendations based on their fitness goals.

User Profile and Goals:

- User registration and profile creation with basic information (age, weight, height).
- Setting fitness goals (weight loss, muscle gain, improved endurance).

Activity Tracking:

- Tracking workouts (duration, distance, calories burned).
- Integration with wearable devices (e.g., smartwatches, fitness trackers) for automatic data collection.
- Ability to manually log activities (running, swimming, yoga etc.).

Workout Management:

- Exercise library with descriptions, instructions, and video demonstrations.
- Pre-built workout plans or ability to create custom routines.
- Tracking progress for individual exercises (sets, reps, weight).

Reporting and Analytics:

- Generate reports on fitness trends over time (e.g., weekly, monthly, yearly).
- Visualize workout patterns through charts and graphs (e.g., pie charts, bar graphs).
- Allow filtering reports by category, date range, or custom criteria

RESULT:

Thus, the problem statement for the given problem is successfully written

EX NO: 3 DESIGNING PROJECT USING AGILE-SCRUM METHODOLOGY BY USING AZURE.

AIM:

To plan an agile model for the given problem statement.

THEORY:

Agile planning is a part of the Agile methodology, which is a project management style with an incremental, iterative approach. Instead of using an in-depth plan from the start of the project—which is typically product-related—Agile leaves room for requirement changes throughout and relies on constant feedback from end users.

With Agile planning, a project is broken down into smaller, more manageable tasks with the goal of having a defined image of a project's vision. Agile planning involves looking at different aspects of a project's tasks and how they'll be achieved, for example:

- Roadmaps to guide a product's release ad schedule

- Sprints to work on one specific group of tasks at a time
- A feedback plan to allow teams to stay flexible and easily adapt to change

User stories, or the tasks in a project, capture user requirements from the end user's perspective Essentially, with Agile planning, a team would decide on a set of user stories to action at any given time, using them as a guide to implement new features or functionalities in a tool. Looking at tasks as user stories is a helpful way to imagine how a customer may use a feature and helps teams prioritize work and focus on delivering value first.

STEPS IN AGILE PLANNING PROCESS:

1. Define vision
2. Set clear expectations on goals
3. Define and break down the product roadmap
4. Create tasks based on user stories
5. Populate product backlog
6. Plan iterations and estimate effort

7. Conduct daily stand-ups

8. Monitor and adapt

RESULT:

Thus, the designing project using agile-scrum methodology by using azure was completed successfully.

EX NO: 4 AGILE PLANNING

AIM:

To develop an agile plan for building a fitness app that allows users to log workouts, track progress, and receive personalized workout recommendations based on their fitness goals.

SCOPE:

This project involves creating a user-centric fitness application supporting profile management, workout logging, wearable integration, workout planning, and analytics. The development will follow Agile methodology using Epics, User Stories, and Sprints for iterative delivery.

AGILE EPICS & USER STORIES:

Epic 1: User Profile and Goals

Objective: Enable users to register, create profiles, and set fitness goals.

User Stories:

- As a user, I want to register and create a profile with my age, weight, and height.
- As a user, I want to set my fitness goals (e.g., weight loss, muscle gain, endurance).
- As a user, I want to edit my profile and update my fitness goals.

Epic 2: Activity Tracking

Objective: Allow users to log workouts manually and automatically via devices.

User Stories:

- As a user, I want to log my workouts manually (e.g., running, swimming).
- As a user, I want the app to track calories, duration, and distance for my activities.
- As a user, I want to sync my data from wearable devices (e.g., Fitbit, Apple Watch).
- As a user, I want to view my recent activities on a timeline or dashboard.

Epic 3: Workout Management

Objective: Provide access to an exercise library and manage custom or pre-built workouts.

User Stories:

- As a user, I want to browse an exercise library with instructions and videos.
- As a user, I want to select or create a custom workout plan.
- As a user, I want to track progress for individual exercises (sets, reps, weights).

Epic 4: Recommendations and Personalization

Objective: Suggest workouts based on user goals and progress.

User Stories:

- As a user, I want to receive workout suggestions based on my goals.
- As a user, I want my recommendations to change as I make progress.
- As a user, I want to receive motivational tips and reminders.

Epic 5: Reporting and Analytics

Objective: Generate insights and visualize workout trends.

User Stories:

- As a user, I want to generate weekly and monthly workout reports.
- As a user, I want to visualize my workout trends through graphs and charts.
- As a user, I want to filter data by activity type, date range, or intensity level.

SPRINT PLAN

Sprint 1: User Profile Setup

- **Duration:** 2 weeks
- **Focus Area:** User Profile and Goals
- **Covered Epics:**
 - User registration and profile creation
 - Goal setting and updates
- **User Stories:**
 - As a user, I want to register and create a profile with personal details.
 - As a user, I want to set and update my fitness goals (e.g., weight loss, muscle gain).

Sprint 2: Basic Activity Logging

- **Duration:** 2 weeks

- **Focus Area:** Activity Tracking
- **Covered Epics:**
 - Manual activity entry
 - Dashboard for viewing logged activities
- **User Stories:**
 - As a user, I want to manually log workouts like yoga or walking.
 - As a user, I want to view my logged activity data on a dashboard.

Sprint 3: Device Integration & Auto Tracking

- **Duration:** 2 weeks
- **Focus Area:** Activity Tracking
- **Covered Epics:**
 - Wearable device data sync
 - Automatic tracking of activity metrics
- **User Stories:**
 - As a user, I want the app to sync data from my wearable device.
 - As a user, I want to see duration, distance, and calories burned automatically tracked.

Sprint 4: Workout Library & Custom Plans

- **Duration:** 2 weeks
- **Focus Area:** Workout Management
- **Covered Epics:**
 - Access to exercise content
 - Personalized workout routines
- **User Stories:**
 - As a user, I want to view exercise instructions and videos.
 - As a user, I want to create custom workout plans from available exercises.

Sprint 5: Recommendations & Personalization

- **Duration:** 2 weeks
- **Focus Area:** Recommendations
- **Covered Epics:**
 - Smart workout suggestions
 - Dynamic plan adjustments
- **User Stories:**
 - As a user, I want to receive personalized workout plans based on my goals.
 - As a user, I want my workout plan to adjust automatically based on my progress.

Sprint 6: Progress Tracking & Analytics

- **Duration:** 2 weeks
- **Focus Area:** Reporting and Analytics
- **Covered Epics:**
 - Performance monitoring
 - Data visualization
- **User Stories:**
 - As a user, I want to generate weekly/monthly fitness reports.
 - As a user, I want to view my progress as charts and graphs.

Sprint 7: Final Testing & Deployment

- **Duration:** 1 week
- **Focus Area:** All Epics
- **Covered Epics:**
 - Quality assurance
 - Launch preparation
- **User Stories:**
 - As a team, we want to conduct full end-to-end testing.
 - As a team, we want to perform performance testing and deploy the app after user acceptance testing (UAT).

RESULT:

Thus, the agile plan for the fitness app has been successfully created using Epics, User Stories, and a Sprint-based approach for structured development.

EX NO: 5 USER STORIES – CREATION

AIM:

To create User Stories for the given problem statement.

THEORY:

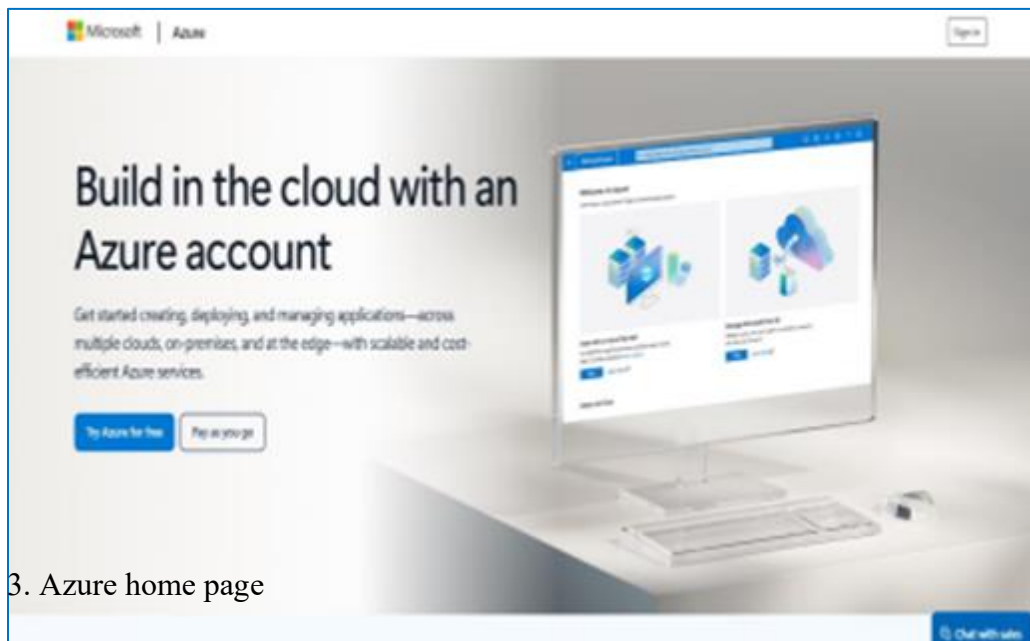
A user story is an informal, general explanation of a software feature written from the perspective of the end user. Its purpose is to articulate how a software feature will provide value to the customer.

User story template

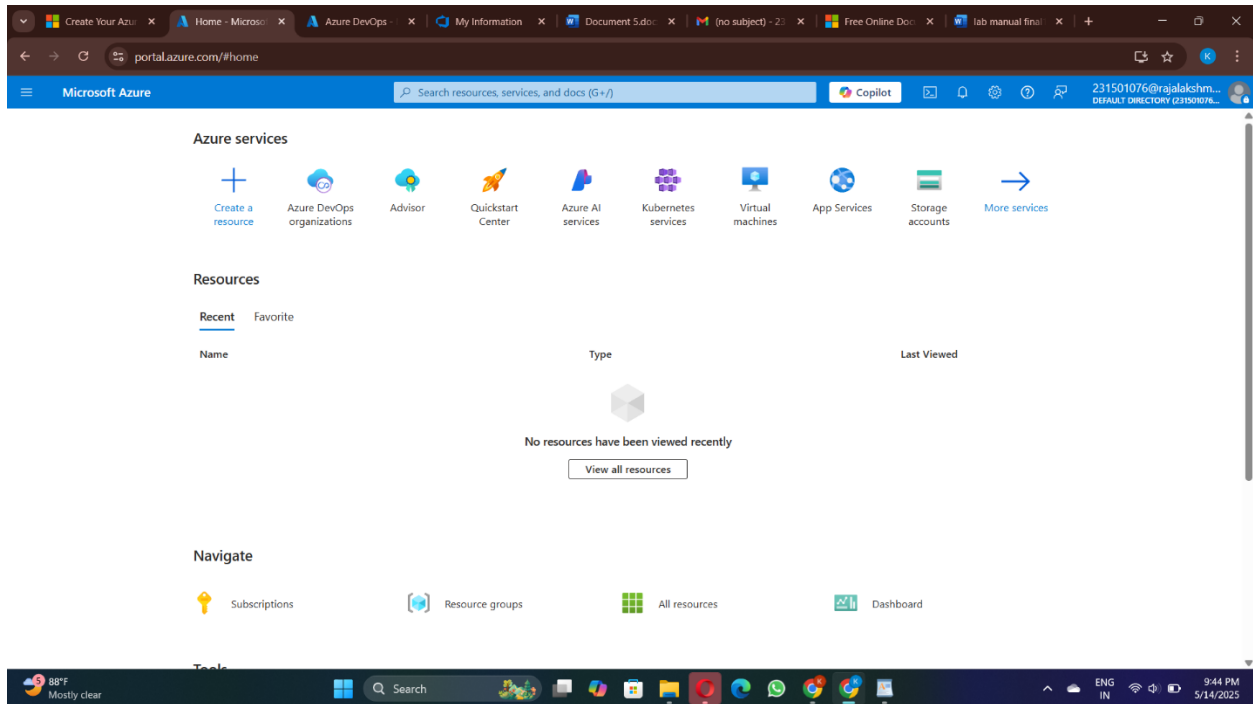
"As a [role], I [want to], [so that]."

PROCEDURE:

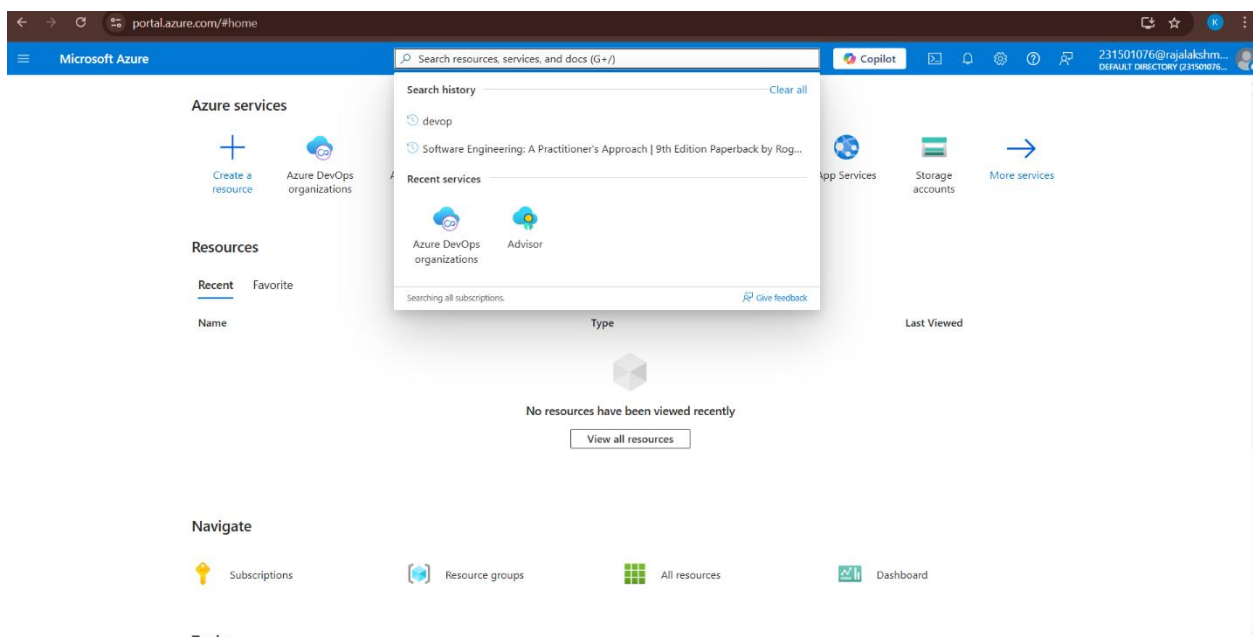
1. Open your web browser and go to the Azure website: <https://azure.microsoft.com/en-in> Sign in using your Microsoft account credentials. If you don't have an account, you'll need to create one.
2. If you don't have a Microsoft account, you can sign up for <https://signup.live.com/?lic=1>



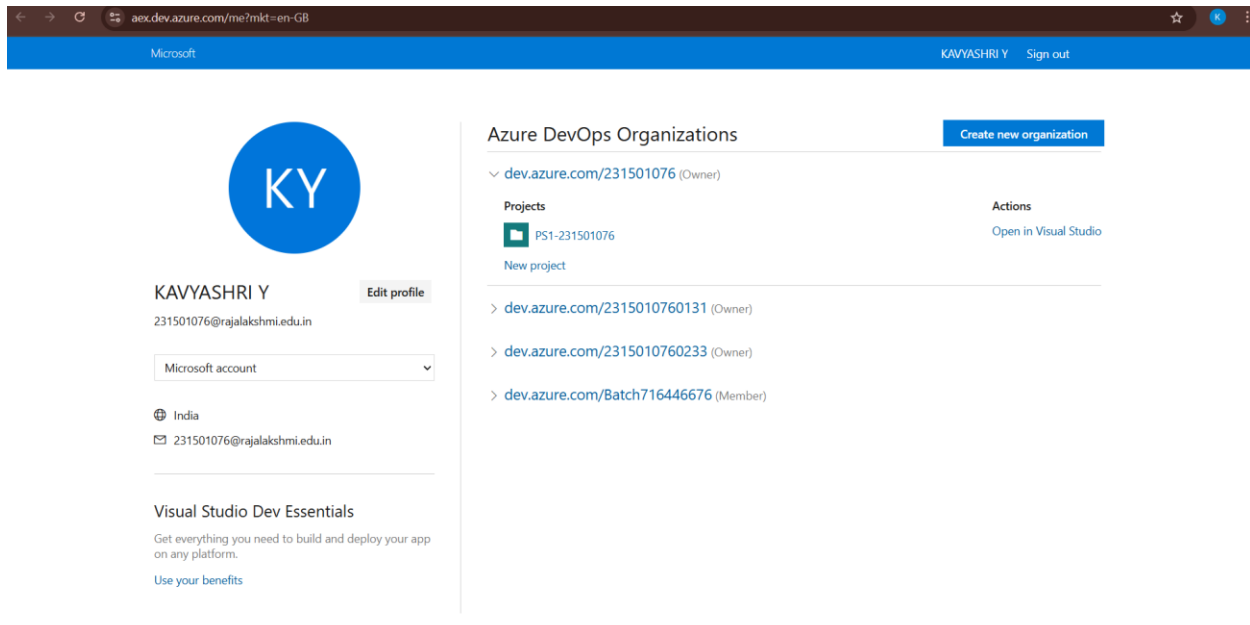
3. Azure home page



4. Open DevOps environment in the Azure platform by typing Azure DevOps Organizations in the search bar.



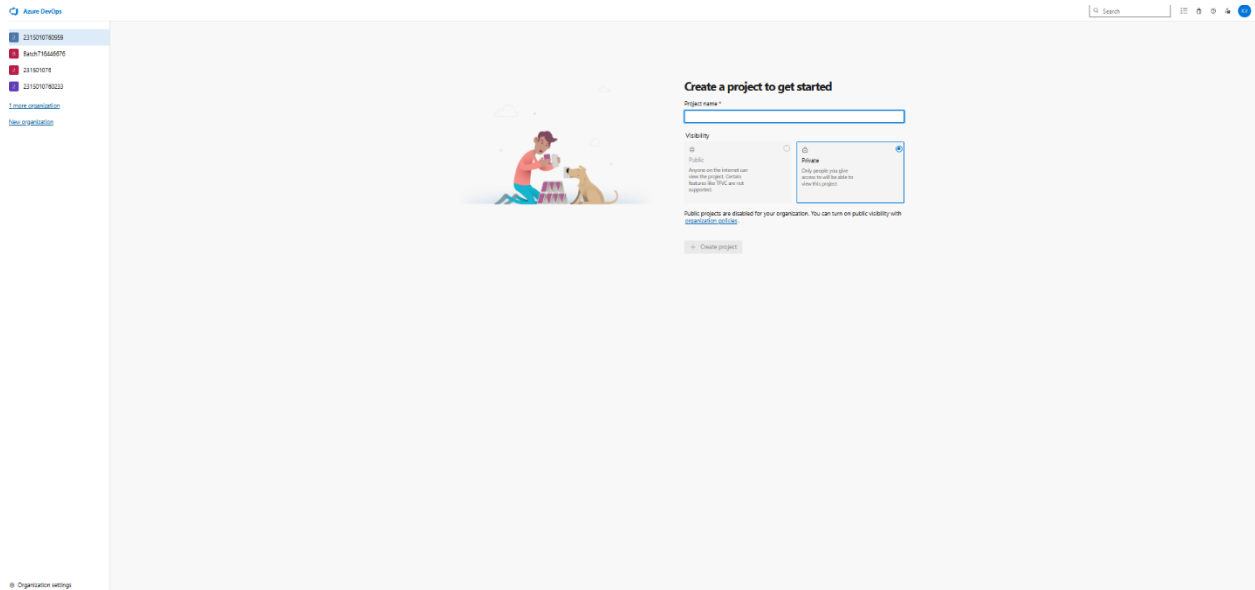
My Azure DevOps Organization link and create an organization and you should be taken to the Azure DevOps Organization Home page.



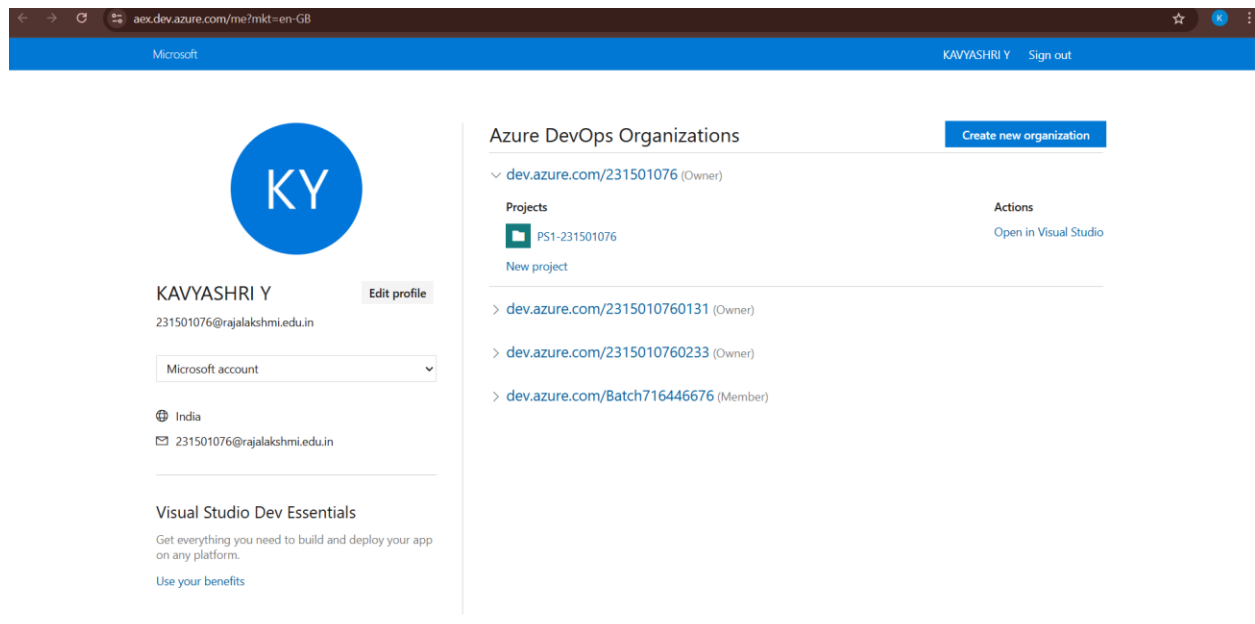
6. Create the First Project in Your Organization

- i. **Home page**, click on the **New Project** button.
- ii. Enter the project name, description, and visibility options:
 - **Name**: Choose a name for the project (e.g., **LMS**).
 - **Description**: Optionally, add a description to provide more context about the project.
 - **Visibility**: Choose whether you want the project to be **Private** (accessible only to those invited) or **Public** (accessible to anyone).

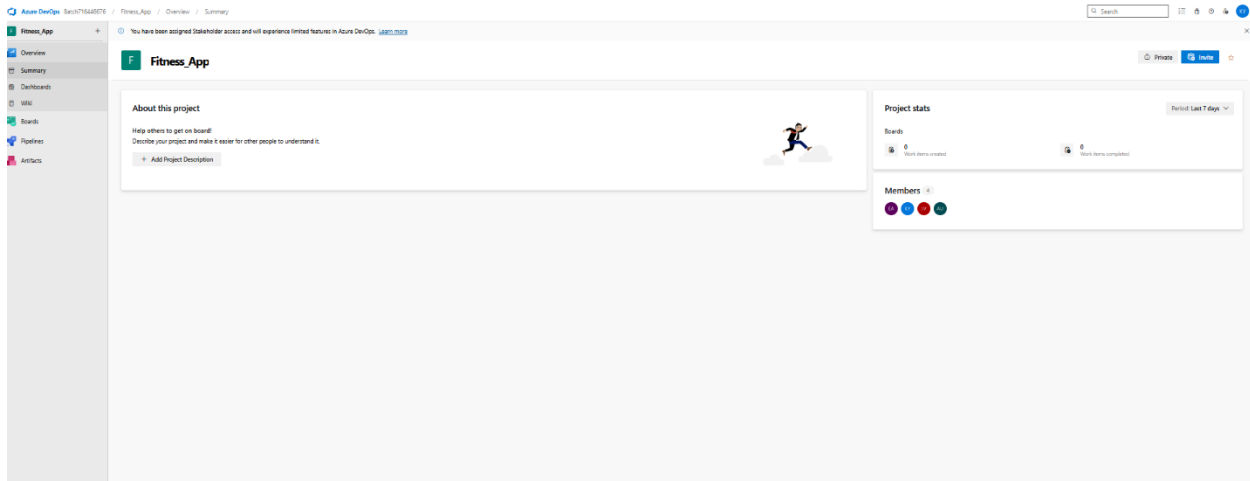
Once you've filled out the details, click **Create** to set up your first project.



7. Once logged in, ensure you are in the correct organization. If you're part of multiple organizations, you can switch between them from the top left corner (next to your user profile). Click on the Organization name, and you should be taken to the Azure DevOps Organization Home page.

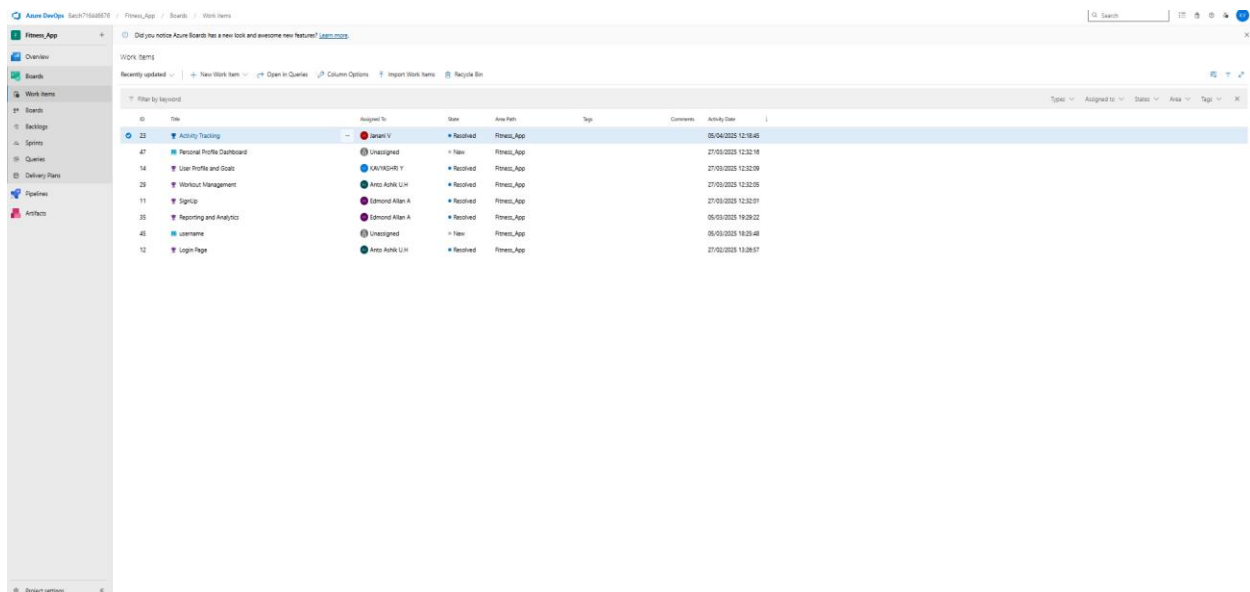


8. Project dashboard

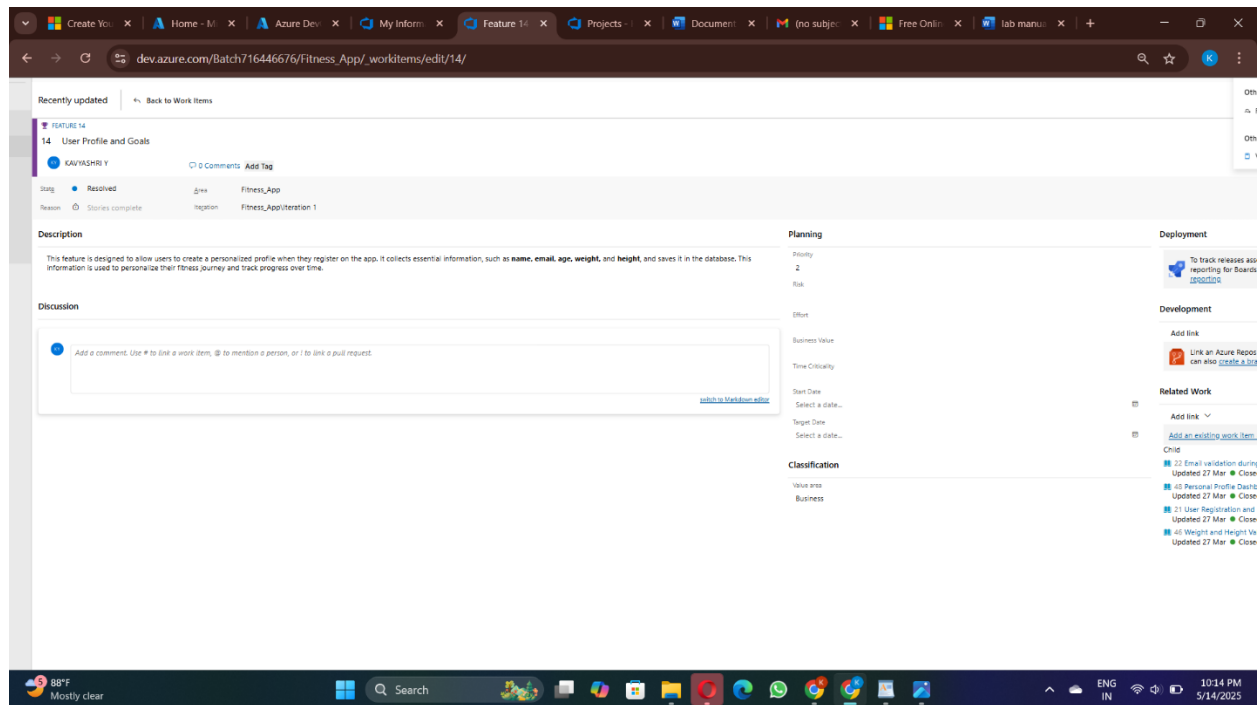


9. To manage user stories

a. From the **left-hand navigation menu**, click on **Boards**. This will take you to the main Boards page, where you can manage work items, backlogs, and sprints. b. On the **work items** page, you'll see the option to **Add a work item** at the top. Alternatively, you can find a + button or **Add New Work Item** depending on the view you're in. From the **Add a work item** dropdown, select **User Story**. This will open a form to enter details for the new User Story.



10. Fill in User Story Details



Result:

The user story for the given problem statement was written successfully.

EX NO: 6 SEQUENCE DIAGRAM

AIM:

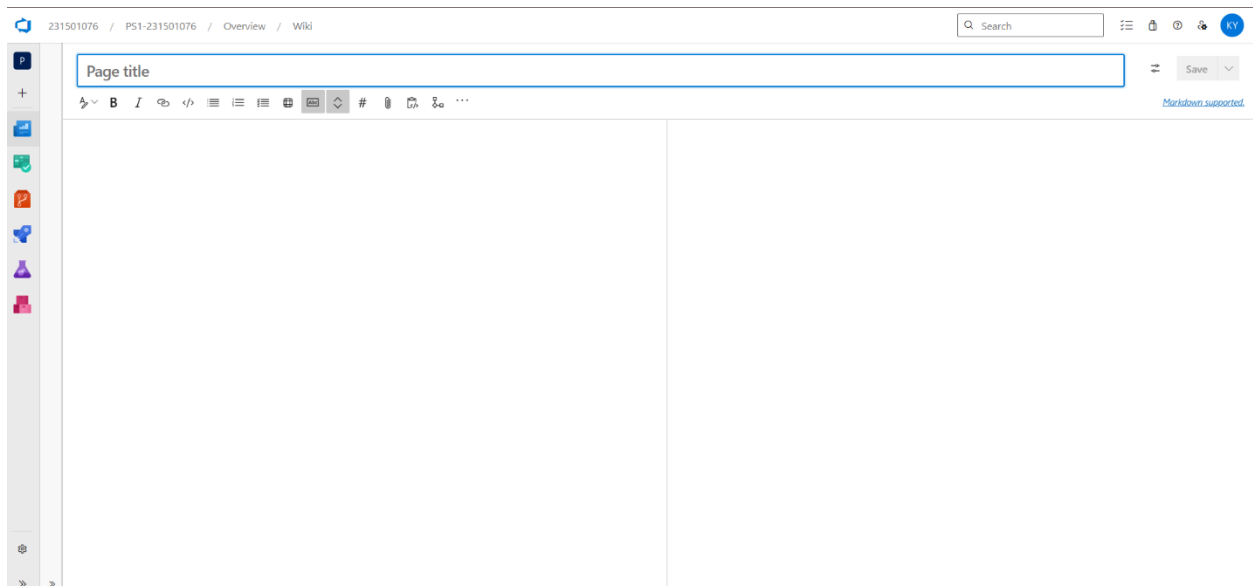
To design a Sequence Diagram by using Mermaid.js for the given problem statement.

THEORY:

A Sequence Diagram is a key component of Unified Modelling Language (UML) used to visualize the interaction between objects in a sequential order. It focuses on how objects communicate with each other over time, making it an essential tool for modelling dynamic behavior in a system.

PROCEDURE:

1. Open a project in Azure DevOps Organizations.
2. To design select wiki from menu.



3. Write code for drawing sequence diagram and save the code :::: mermaid sequence

sequence Diagram:

participant User

participant App

participant Wearable Device

participant Database

participant Analytics

User->>App: Create Profile (name, age, weight, height)

App->>Database: Save User Profile

Database-->>App: Confirm Profile Saved

App-->>User: Profile Created Successfully

User->>App: Set Fitness Goal (weight loss, muscle gain)

App->>Database: Save Fitness Goal

Database-->>App: Confirm Goal Saved

App-->>User: Goal Set Successfully

User->>App: Start Workout (e.g., running)

App->>Wearable Device: Sync Data (heart rate, calories, distance)

Wearable Device->>Database: Save Activity Log (duration, calories burned, distance)

Database-->>App: Confirm Activity Saved

App-->>User: Activity Tracked Successfully

User->>App: Log Manual Activity (yoga)

App->>Database: Save Manual Activity Log

Database-->>App: Confirm Manual Activity Saved

App-->>User: Manual Activity Logged Successfully

User->>App: Request Workout Report (weekly)

App->>Database: Retrieve Workout Logs (weekly data)

Database-->>App: Return Workout Logs

App->>Analytics: Generate Analytics Report

Analytics-->>App: Provide Visual Data (charts, trends)

App-->>User: Display Workout Report (visuals, trends)

User-->>App: View Exercise Library

App-->>Database: Retrieve Exercises

Database-->>App: Return Exercise List

App-->>User: Display Exercise Library

User-->>App: Select Exercise (e.g., squats)

App-->>Database: Retrieve Exercise Details (description, video)

Database-->>App: Return Exercise Details

App-->>User: Display Exercise Details (video, instructions)

User-->>App: Create Custom Workout Plan

App-->>Database: Save Custom Workout Plan

Database-->>App: Confirm Custom Plan Saved

App-->>User: Custom Workout Plan Created Successfully

EXPLANATION:

1. User creates a profile.

The user enters details like name, age, weight, and height.

- The app sends this information to the database.
- The database saves the profile.
- The app confirms the profile is created successfully.

2. User sets a fitness goal.

The user selects a goal like weight loss or muscle gain.

- The app sends the goal data to the database.
- The database saves the goal.
- The app confirms the goal is set successfully.

3. User starts a workout.

For example, the user starts running.

- The wearable device collects data (heart rate, calories, distance).
- This data is sent to the app and saved in the database.
- The app confirms that the activity was successfully tracked.

4. User logs a manual activity.

For example, yoga, which isn't tracked by a device.

- The user manually enters the details into the app.
- The app sends the log to the database.
- The database saves it and confirms.
- The app shows a message that the manual activity was logged successfully.

5. User requests a workout report (weekly).

- The app fetches workout logs from the database.
- It sends the logs to the analytics system.
- The analytics system generates visual reports like charts and trends.
- The app displays the workout report to the user.

6. User views the exercise library.

- The app requests a list of exercises from the database.
- The database returns a list of available exercises.
- The app displays the exercise library to the user.

7. User selects an exercise (e.g., squats).

- The app fetches detailed information (description, video) from the database.
- The app shows the exercise details to the user.

8. User creates a custom workout plan.

- The user selects exercises and creates a custom plan in the app.
- The app saves the plan in the database.
- The database confirms the plan is saved.
- The app shows a message that the plan was created successfully.



RESULT:

Thus, the sequence diagram for the given problem statement was drawn successfully.

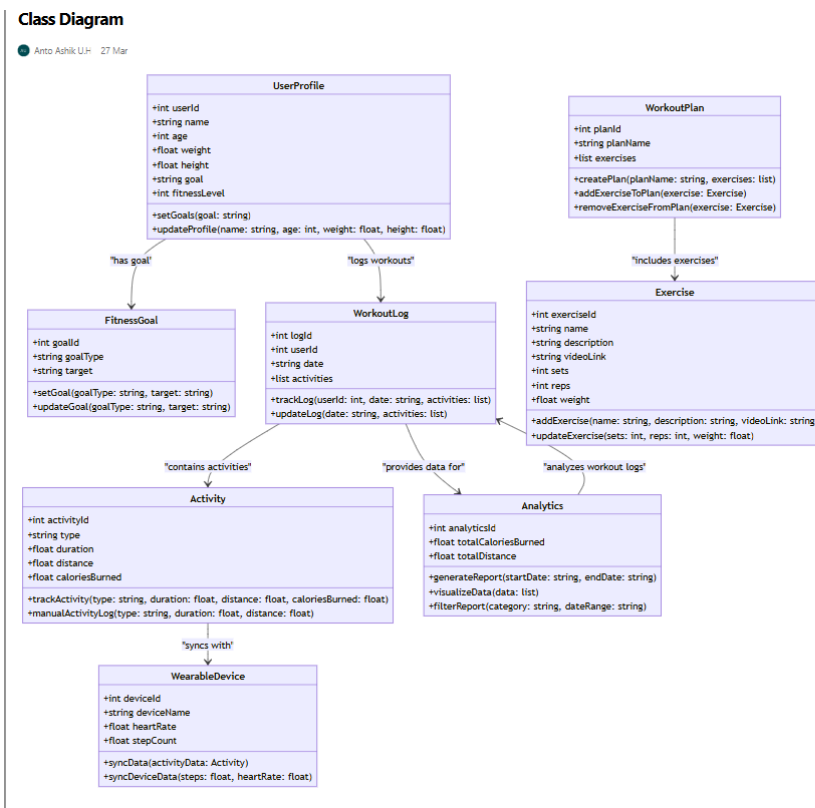
EX NO: 7 CLASS DIAGRAM

AIM:

To draw a sample class diagram for your project or system.

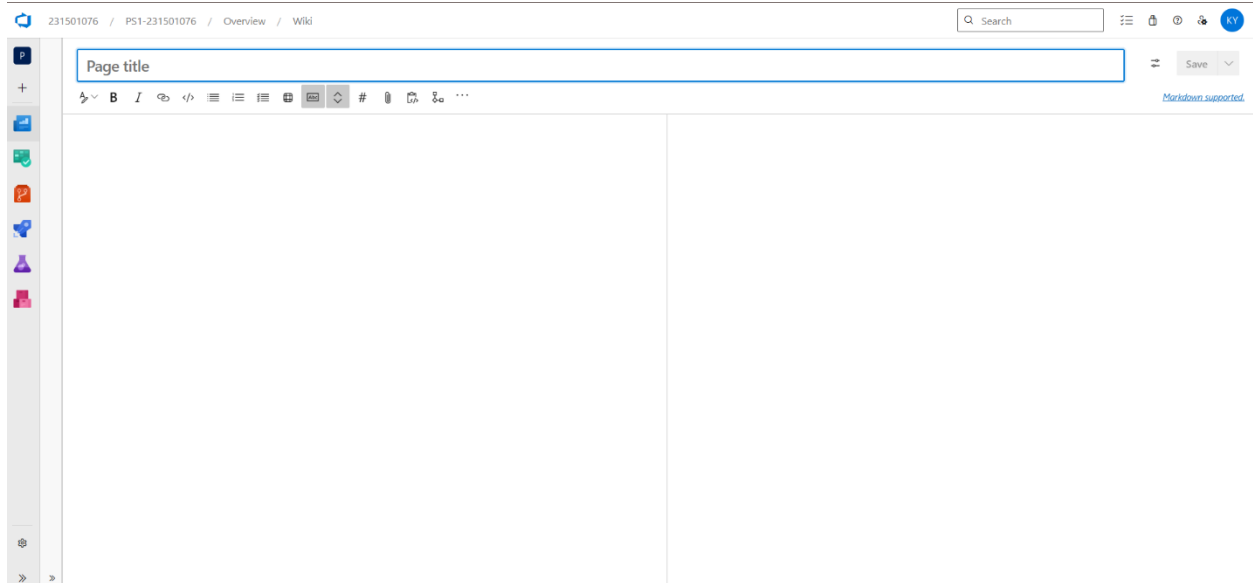
THEORY:

A UML class diagram is a visual tool that represents the structure of a system by showing its classes, attributes, methods, and the relationships between them.



PROCEDURE:

1. Open a project in Azure DevOps Organizations.
2. To design select wiki from menu.



3. Write code for drawing class diagram and save the code

```
::: mermaid
```

```
classDiagram
```

```
class UserProfile {
    +int userId
    +string name
    +int age
    +float weight
    +float height
    +string goal
    +int fitnessLevel
    +setGoals(goal: string)
    +updateProfile(name: string, age: int, weight: float, height: float)
}
```

```
class FitnessGoal {
    +int goalId
    +string goalType
    +string target
}
```



```
+setGoal(goalType: string, target: string)
+updateGoal(goalType: string, target: string)
}
```

```
class WorkoutLog {
  +int logId
  +int userId
  +string date
  +list activities
  +trackLog(userId: int, date: string, activities: list)
  +updateLog(date: string, activities: list)
}
```

```
class Activity {
  +int activityId
  +string type
  +float duration
  +float distance
  +float caloriesBurned
  +trackActivity(type: string, duration: float, distance: float, caloriesBurned: float)
  +manualActivityLog(type: string, duration: float, distance: float)
}
```

```
class WearableDevice {
  +int deviceId
  +string deviceName
  +float heartRate
  +int stepCount
  +syncData(activityData: Activity)
  +syncDeviceData(steps: float, heartRate: float)
```

```
}
```

```
class Analytics {  
    +int analyticsId  
    +float totalCaloriesBurned  
    +float totalDistance  
    +generateReport(startDate: string, endDate: string)  
    +visualizeData(data: list)  
    +filterReport(category: string, dateRange: string)  
}
```

```
class WorkoutPlan {  
    +int planId  
    +string planName  
    +list Exercises  
    +createPlan(planName: string, exercises: list)  
    +addExerciseToPlan(exercise: Exercise)  
    +removeExerciseFromPlan(exercise: Exercise)  
}
```

```
class Exercise {  
    +int exerciseId  
    +string name  
    +string description  
    +string videoLink  
    +int sets  
    +int reps  
    +float weight  
    +addExercise(name: string, description: string, videoLink: string)  
    +updateExercise(sets: int, reps: int, weight: float)
```

}

UserProfile "1" --> "1" FitnessGoal : has goal

UserProfile "1" --> "*" WorkoutLog : logs workouts

WorkoutLog "1" --> "*" Activity : contains activities

Activity --> WearableDevice : syncs with

WorkoutLog --> Analytics : provides data for

Analytics --> WorkoutLog : analyzes workout logs

WorkoutPlan "1" --> "*" Exercise : includes exercises

RESULT:

Thus, the use case diagram for the given problem statement was designed successfully.

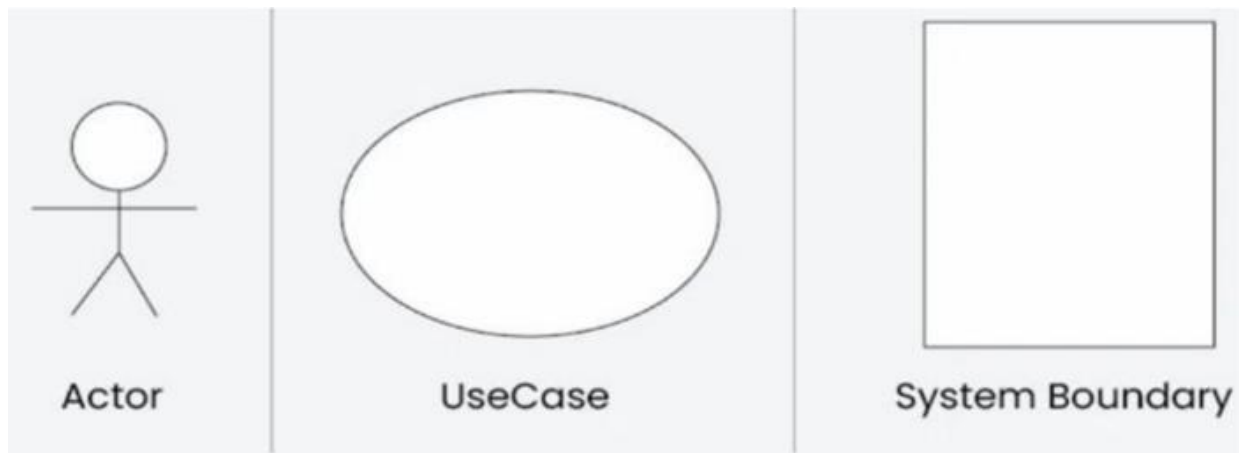
EX NO: 8 USECASE DIAGRAM

AIM:

Steps to draw the Use Case Diagram using draw.io

THEORY:

- UCD shows the relationships among actors and use cases within a system which Provide an overview of all or part of the usage requirements for a system or organization in the form of an essential model or a business model and communicate the scope of a development project
- Use Cases
- Actors
- Relationships
- System Boundary Boxes



PROCEDURE:

Step 1: Create the Use Case Diagram in Draw.io

- Open Draw.io (draw.io).
- Click "Create New Diagram" and select "Blank" or "UML Use Case" template.
- Add Actors (Users, Admins, External Systems) from the UML section.
- Add Use Cases (Functionalities) using ellipses.

- Connect Actors to Use Cases with lines (solid for direct interaction, dashed for <<include>> and <<extend>>).
- Save the diagram as .drawio or export as PNG/JPG/SVG.

Step 2: Upload the Diagram to Azure DevOps

Option 1: Add to Azure DevOps Wiki

- Open Azure DevOps and go to your project.
- Navigate to Wiki (Project > Wiki).
- Click "Edit Page" or create a new page.
- Drag & drop the exported PNG/JPG image.
- Use Markdown to embed the diagram:
- ! [Use Case Diagram] (attachments/use_case_diagram.png)

Option 2: Attach to Work Items in Azure Boards

- Open Azure DevOps → Navigate to Boards (Project > Boards).
- Select a User Story, Task, or Feature.
- Click "Attachments" → Upload your Use Case Diagram
- Add comments or descriptions to explain the use case Diagram.

RESULT:

The use case diagram for the given problem statement was designed successfully.







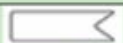




EX NO: 9 ACTIVITY DIAGRAM

AIM:

To draw a sample activity diagram for your project or system.

THEORY:

Activity diagrams are an essential part of the Unified Modelling Language (UML) that help visualize workflows, processes, or activities within a system. They depict how different actions are connected and how a system moves from one state to another.

Notations	Symbol	Meaning
Start		Shows the beginning of a process
Connector		Shows the directional flow, or control flow, of the activity
Joint symbol		Combines two concurrent activities and re-introduces them to a flow where one activity occurs at a time
Decision		Represents a decision
Note		Allows the diagram creators to communicate additional messages
Send signal		Show that a signal is being sent to a receiving activity
Receive signal		Demonstrates the acceptance of an event
Flow final symbol		Represents the end of a specific process flow
Option loop		Allows the creator to model a repetitive sequence within the option loop symbol
Shallow history pseudostate		Represents a transition that invokes the last active state.
End		Marks the end state of an activity and represents the completion of all flows of a process

PROCEDURE:

1. Draw diagram in draw.io
2. Upload the diagram in the Azure Wiki

RESULT:

Thus, the Activity diagram for the above problem statement done successfully.

EX NO: 10 ARCHITECTURE DIAGRAM

AIM:

Steps to draw the Architecture Diagram using draw.io.

THEORY:

An architectural diagram is a visual representation that maps out the physical implementation for components of a software system. It shows the general structure of the software system and the associations, limitations, and boundaries between each element.



PROCEDURE:

1. Draw diagram in draw.io
2. Upload the diagram in Azure DevOps wiki

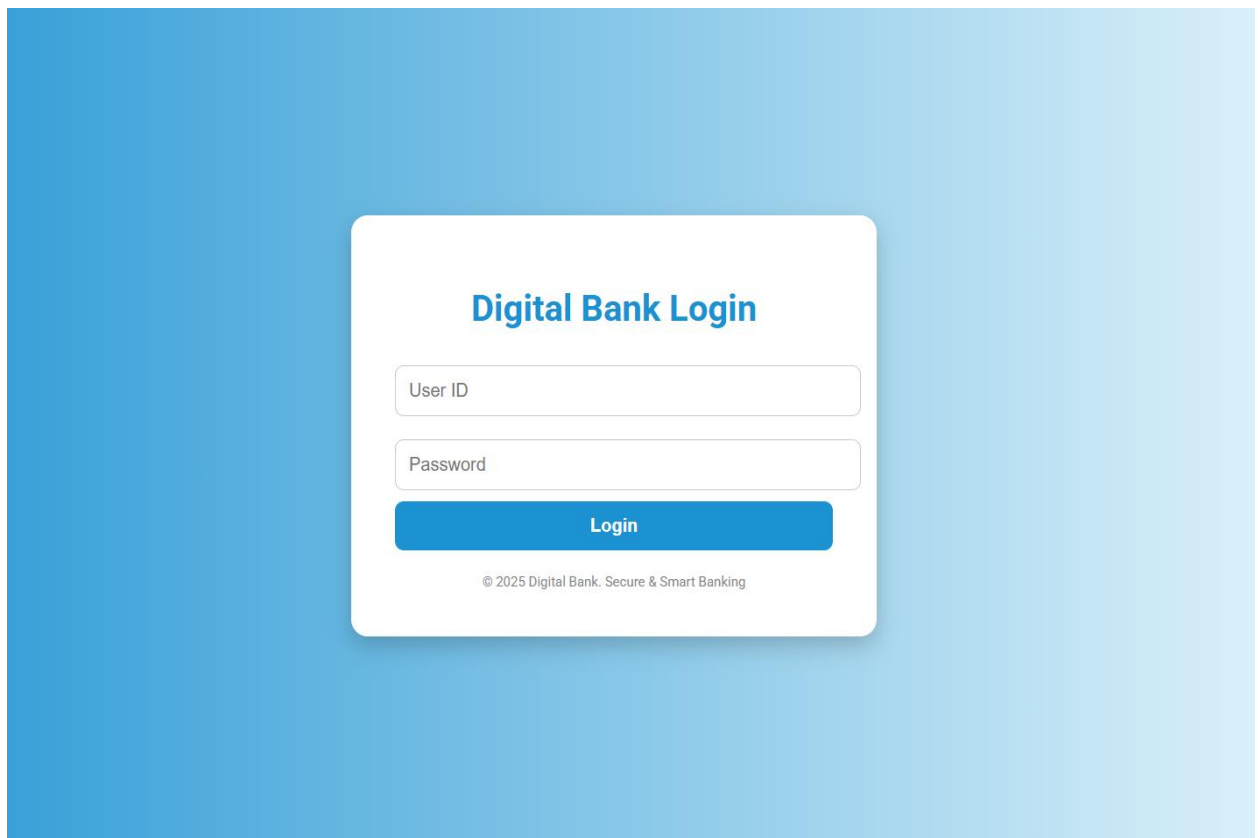
RESULT:

Thus, the architecture diagram for the given problem statement was designed successfully.

EX NO: 11 USER INTERFACE

AIM:

Design User Interface for the given project.



The image displays a user interface for a digital bank login. It features a white login card centered on a blue gradient background. The card has a title 'Digital Bank Login' in blue, followed by two input fields for 'User ID' and 'Password'. A blue 'Login' button is positioned below the fields. At the bottom of the card, there is a copyright notice: '© 2025 Digital Bank. Secure & Smart Banking'.

Digital Bank Login

User ID

Password

Login

© 2025 Digital Bank. Secure & Smart Banking



Digital Bank Dashboard

👋 Welcome back, John Doe! Manage everything from one place.



Personal Details

Review or update your personal and contact information.

[Go to Details](#)

Make a Transaction

Send money instantly to anyone with secure transactions.

[Transfer Now](#)

Account Summary

Check your balance, transaction history and statements.

[View Summary](#)

Customer Support

Need help? Connect with our 24/7 virtual assistant.

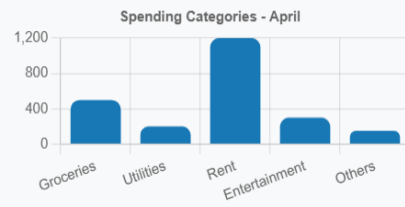
[Contact Us](#)



Account Overview

User ID: USER123456
Account Number: 1234567890
Account Holder: John Doe
Account Type: Savings
Balance: \$5,000.00

Spending Overview



Transaction Details

Transaction ID: TXN789654123
Recipient Name: Jane Smith
Amount: \$200.00

[Back to Home](#)

Make a Transaction

Recipient Name

e.g., Sarah Williams

Recipient Account Number

e.g., 123456789012

SWIFT/BIC Code

e.g., ABCDUS33XXX

Amount (USD)

\$100.00

Purpose

Fund Transfer



Remarks (Optional)

Confirm Transaction



Ensure the account number and SWIFT/BIC code are correct before confirming.



Transaction Successful!

Your transaction has been processed securely.

Transaction ID: 1234567890

Recipient Name: Jane Smith

Amount: \$500.00

[Back to Dashboard](#)

RESULT:

Thus, the UI for the given problem statement is completed successfully.

EX NO: 12 IMPLEMENTATIONS

AIM:

To implement the given project based on Agile Methodology.

PROCEDURE:

Step 1: Set Up an Azure DevOps Project

- Log in to Azure DevOps.
- Click "New Project" → Enter project name → Click "Create".
- Inside the project, navigate to "Repos" to store the code.

Step 2: Add Your Web Application Code

- Navigate to Repos → Click "Clone" to get the Git URL.
- Open Visual Studio Code / Terminal and run: `git clone cd`
- Add web application code (HTML, CSS, JavaScript, React, Angular, or backend like Node.js, .NET, Python, etc.).
- Commit & push: `git add. git commit -m "Initial commit" git push`

origin main

Step 3: Set Up Build Pipeline (CI/CD - Continuous Integration)

- Navigate to Pipelines → Click "New Pipeline".
- Select Git Repository (Azure Repos, GitHub, or Bitbucket).
- Choose Starter Pipeline or a pre-configured template for your framework.
- Modify the `azure-pipelines.yml` file (Example for a Node.js app):

trigger:

- main

pool:

vmImage: 'ubuntu-latest'

steps:

task: UseNode@1

inputs:

version: '16.x'

-script: npm install

displayName: 'Install dependencies'

-script: npm run build

displayName: 'Build application'

-task: PublishBuildArtifacts@1

inputs:

pathToPublish: 'dist'

artifactName: 'drop'

Click "Save and Run" → The pipeline will start building app.

Step 4: Set Up Release Pipeline (CD - Continuous

Deployment) • Go to Releases → Click "New

Release Pipeline".

- Select Azure App Service or Virtual Machines (VMs) for

deployment. • Add an artifact (from the build pipeline).

- Configure deployment stages (Dev, QA, Production).

- Click "Deploy" to push your web app to Azure.

RESULT:

Thus, the implementation of the given problem statement is done successfully.

