



**RAJALAKSHMI**  
**ENGINEERING COLLEGE**  
An AUTONOMOUS Institution  
Affiliated to ANNA UNIVERSITY, Chennai

**DEPARTMENT OF COMPUTER SCIENCE &  
ENGINEERING ACADEMIC YEAR 2024-2025**

**EVEN SEMESTER**



**CS23432 SOFTWARE ENGINEERING LAB**

**LAB MANUAL**

**SECOND YEAR**

**FOURTH SEMESTER**

**2024- 2025**

**EVEN SEMESTER**

Ex No	List of Experiments
1	Study of Azure DevOps
2	Designing Project using AGILE-SCRUM Methodology.
3	Agile Planning
4	User stories – Creation
5	Architecture Diagram Using AZURE
6	Designing Usecase and Class Diagram
7	Designing Interaction Diagrams
8	Design Interface
9	Implementation – Design a Web Page based on Scrum Methodology
10	Testing using Azure.
11	Deployment

Requirements	
Hardware	Intel i3, CPU @ 1.20GHz 1.19 GHz, 4 GB RAM, 32 Bit Operating System
Software	StarUML , Azure

## LAB PLAN

### CS19442-SOFTWARE ENGINEERING LAB

Ex No	Date	Topic	Page No	Sign
1		Study of Azure DevOps		
2		Writing Problem Statement		
3		Designing Project using AGILE-SCRUM Methodology by using Azure.		
4		Agile Planning		
5		User stories – Creation		
6		Architecture Diagram Using AZURE		
7		Designing Usecase Diagram using StarUML		
8		Designing Activity Diagrams using StarUML		
9		Designing Sequence Diagrams using StarUML		
10		Design Class Diagram		
10		Design User Interface		
11		Implementation – Design a Web Page based on Scrum Methodology		
12		Testing		
13		Deployment		

### Course Outcomes (COs)

**Course Name: Software Engineering**  
**Course Code: CS23432**

<b>CO 1</b>	Understand the software development process models.
<b>CO 2</b>	Determine the requirements to develop software
<b>CO 3</b>	Apply modeling and modeling languages to design software products
<b>CO 4</b>	Apply various testing techniques and to build a robust software products
<b>CO 5</b>	Manage Software Projects and to understand advanced engineering concepts

**CO - PO – PSO matrices of course**

<b>PO/PSO CO</b>	<b>PO1</b>	<b>PO2</b>	<b>PO3</b>	<b>PO4</b>	<b>PO5</b>	<b>PO6</b>	<b>PO7</b>	<b>PO8</b>	<b>PO9</b>	<b>PO10</b>	<b>PO11</b>	<b>PO12</b>	<b>PSO1</b>	<b>PSO2</b>	<b>PSO3</b>
<b>CS23432.1</b>	2	2	3	2	2	2	2	2	2	2	3	2	1	3	-
<b>CS23432.2</b>	2	3	1	2	2	1	-	1	1	1	2	-	1	2	-
<b>CS23432.3</b>	2	2	1	1	1	1	1	1	1	1	1	1	2	2	1
<b>CS23432.4</b>	2	2	3	2	2	2	1	0	2	2	2	1	1	2	1
<b>CS23432.5</b>	2	2	2	1	1	1	1	0	2	1	1	1	2	1	-
<b>Average</b>	<b>2.0</b>	<b>2.2</b>	<b>2.0</b>	<b>1.6</b>	<b>1.6</b>	<b>1.4</b>	<b>1.3</b>	<b>1.3</b>	<b>1.6</b>	<b>1.4</b>	<b>1.8</b>	<b>1.3</b>	<b>1.4</b>	<b>2.0</b>	<b>1.0</b>

Correlation levels 1, 2 or 3 are as defined below:

1: Slight (Low)    2: Moderate (Medium)    3: Substantial (High)    No correlation: “-”

## **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:**

The study was successfully completed.

**EX NO: 2**

## **PROBLEM STATEMENT**

### **AIM:**

To prepare PROBLEM STATEMENT for your given project.

### **Problem Statement:**

#### **Expense Tracker App**

In today's fast-paced world, individuals and small households often struggle to manage their finances effectively due to the lack of simple, affordable, and accessible tools for tracking expenses. While financial institutions and tech-savvy users utilize advanced budgeting apps, many people face challenges such as complex interfaces, subscription costs, and reliance on cloud-based services.

There is a need for a lightweight, user-friendly, and offline-capable application that enables users to:

- Monitor daily, weekly, and monthly expenses,
- Track sales performance,
- Set and track budget goals,
- And make informed financial decisions without requiring technical knowledge or internet access.

### **Result:**

The problem statement was written successfully.

## EX NO: 3

### AGILE PLANNING

#### Aim:

To prepare an Agile Plan.

#### THEORY

##### **Agile Planning for Expense Tracker App Development**

Agile planning is a key component of the Agile methodology—a flexible, iterative project management approach ideal for developing user-focused applications like an expense tracker. Unlike traditional methods that rely on a detailed upfront plan, Agile allows for evolving requirements and continuous feedback from users throughout the development process.

In Agile planning, the development of the expense tracker app is divided into smaller, manageable tasks, helping the team maintain a clear vision while remaining adaptable. This approach supports the delivery of a high-quality, user-friendly, and efficient app by focusing on what delivers the most value to end users.

Key elements of Agile planning for this project include:

- **Product roadmap** to outline key features and release timelines
- **Sprints** to focus on specific sets of features like expense input, categorization, or reports
- **Feedback loops** to ensure ongoing user input and iterative improvement

**User stories**, representing features from the user's point of view (e.g., "As a user, I want to track my daily expenses so I can manage my budget"), help guide the development process. They make it easier to prioritize tasks and ensure that the most valuable functionalities are delivered first.

#### **Steps in the Agile Planning Process for the Expense Tracker App:**

1. **Define the vision** – Create a clear purpose for the app (e.g., helping users manage finances offline with ease).
2. **Set clear goals** – Outline measurable targets such as budget alerts, monthly reports, and secure offline storage.
3. **Break down the product roadmap** – Divide the app's development into feature-based phases (e.g., expense logging, analytics, export options).
4. **Create tasks from user stories** – Convert user needs into actionable development tasks.
5. **Populate the product backlog** – List and prioritize all features and improvements to be implemented.



6. **Plan iterations and estimate effort** – Assign tasks to sprints and estimate time/resources needed.
7. **Conduct daily stand-ups** – Keep the team aligned and identify blockers quickly.
8. **Monitor progress and adapt** – Use feedback and performance metrics to adjust direction as needed.

**Result:**

Thus the Agile plan was completed successfully.

**EX NO: 4**

## **CREATE USER STORIES**

### **Aim:**

To create User Stories

### **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]."**

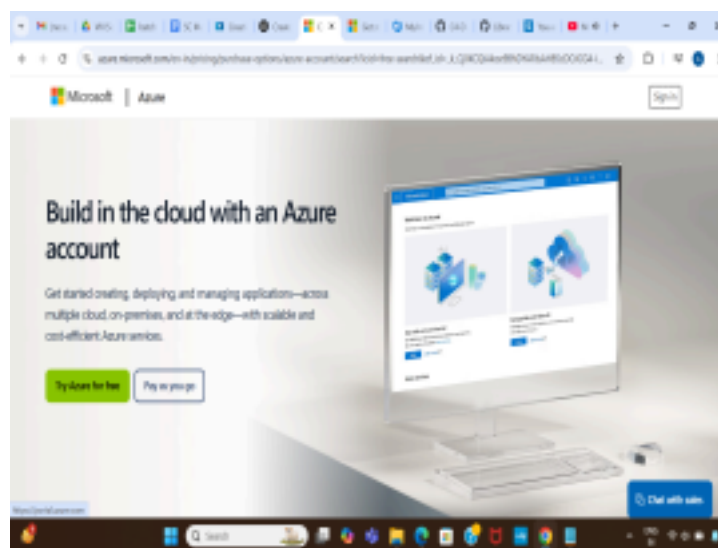
### **Procedure:**

1. Open your web browser and go to the Azure website:

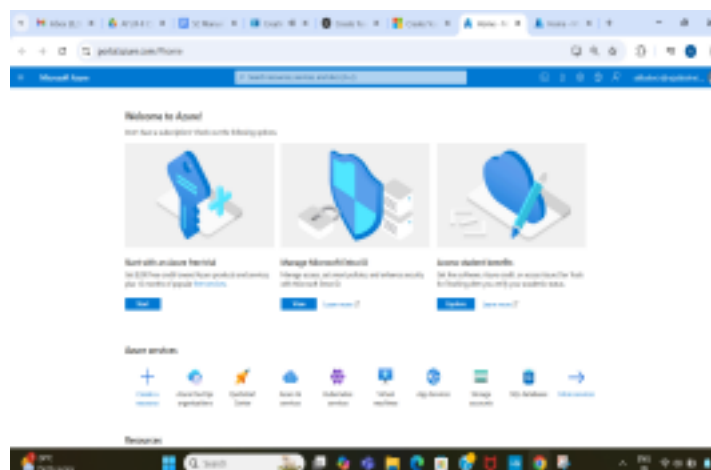
<https://azure.microsoft.com/en-in> Sign in using your Microsoft account credentials. If you don't have an account, you'll need to create one.

2. If you don't have a Microsoft account, you can sign up for

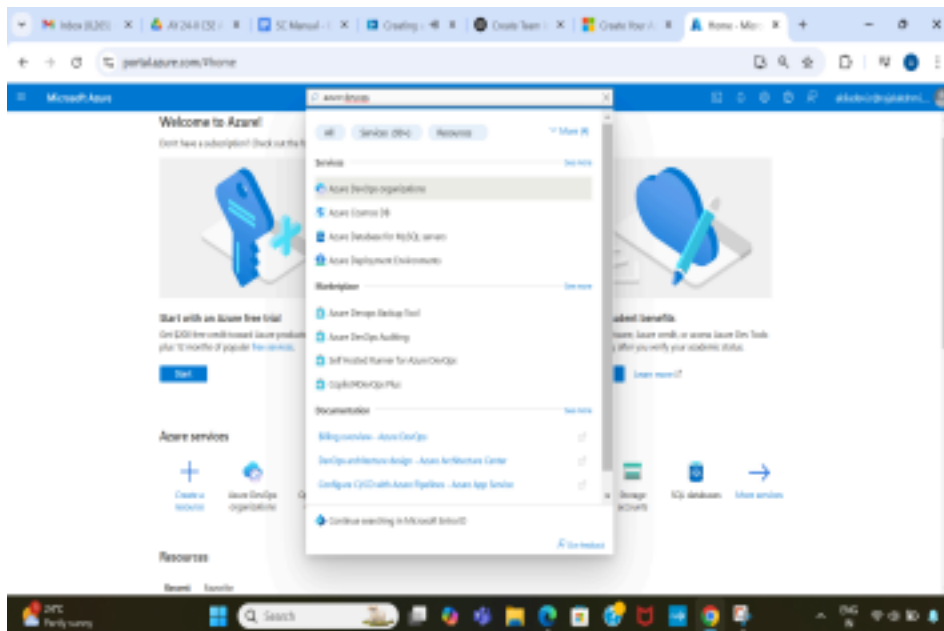
<https://signup.live.com/?lic=1>



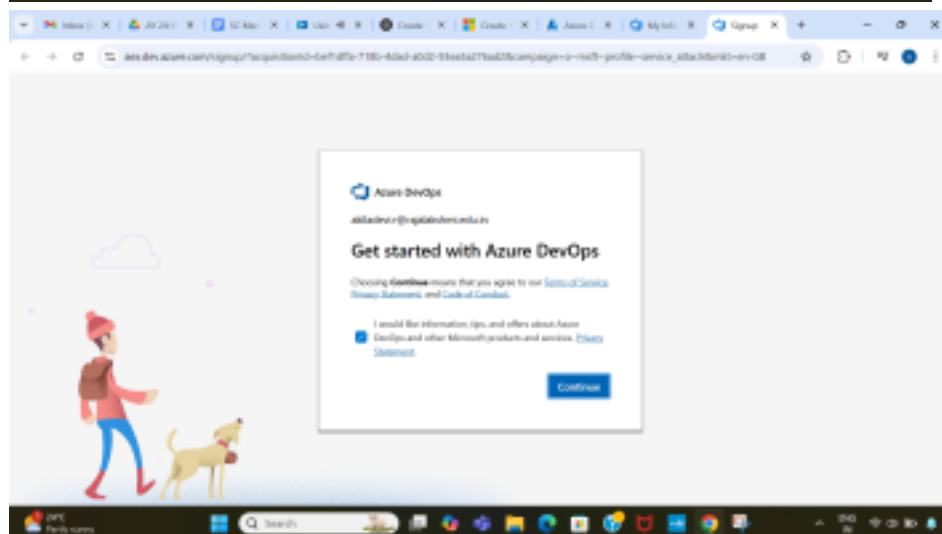
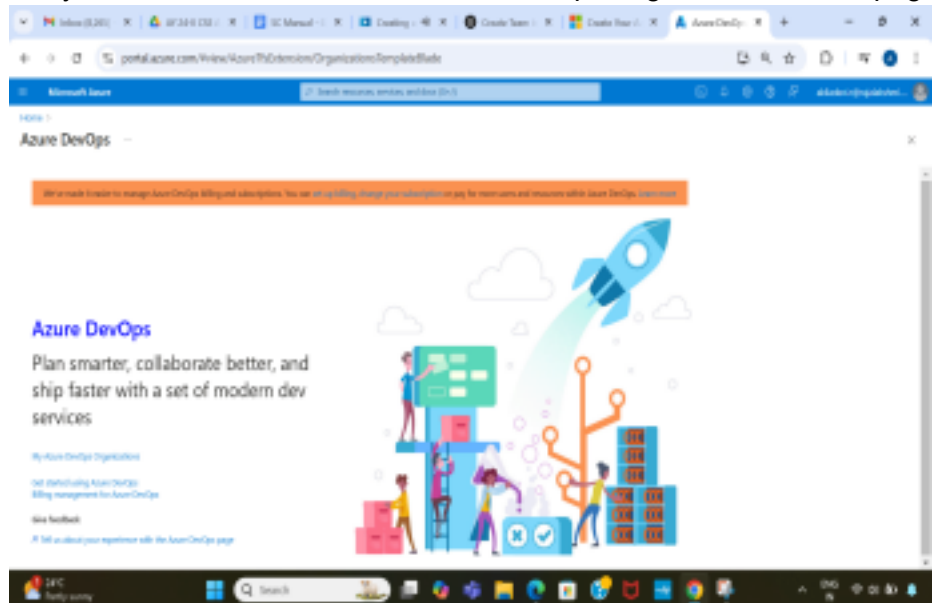
3. Azure home page

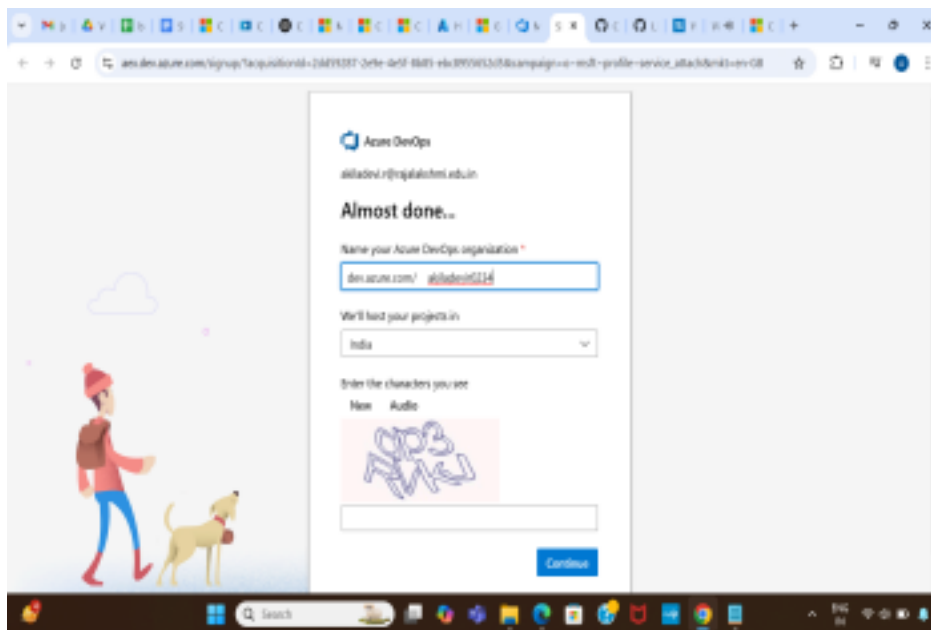


4. Open DevOps environment in the Azure platform by typing Azure DevOps Organizations in the search bar.



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

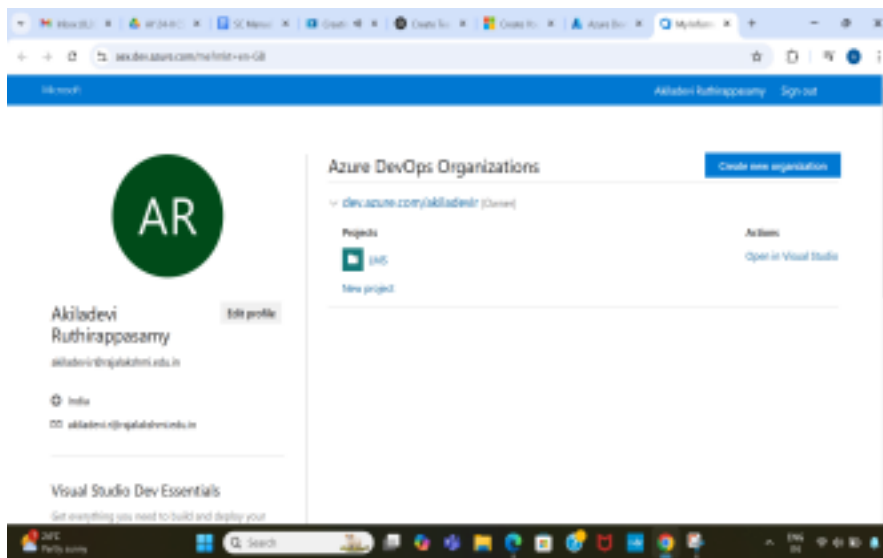




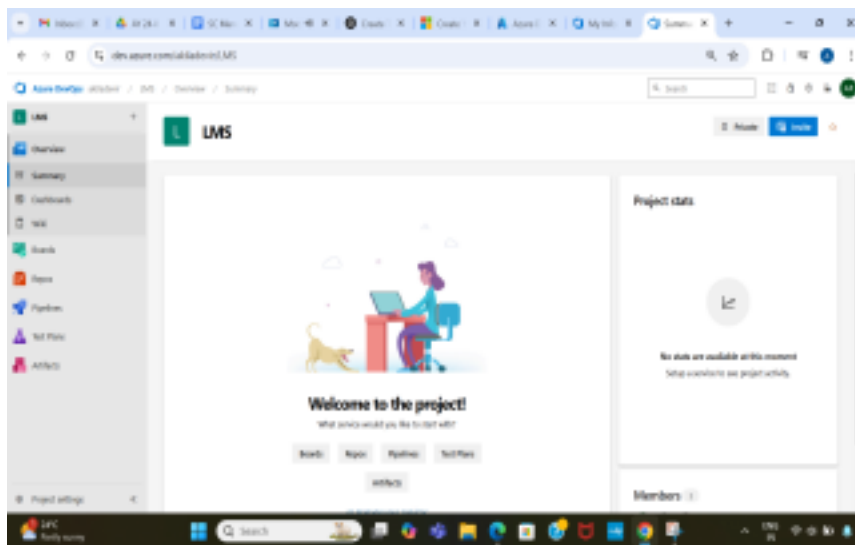
6. Create the First Project in Your Organization  
After the organization is set up, you'll need to create your first **project**. This is where you'll begin to manage code, pipelines, work items, and more.

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

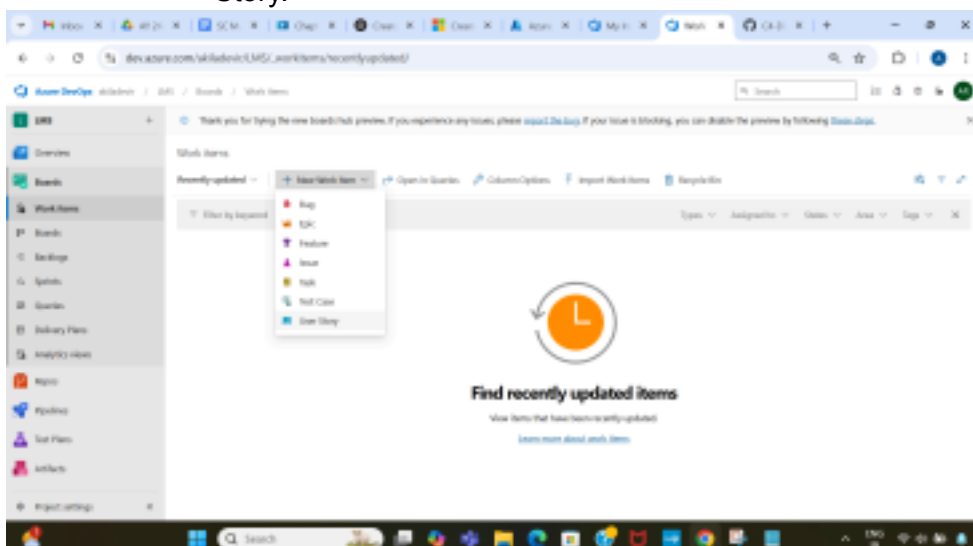


8. Project dashboard



9. To manage user stories

- 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.
- 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

The screenshot shows the Azure DevOps interface for a project named 'Expert System'. The left sidebar contains navigation links for Overview, Boards, Work Items, Backlogs, Sprints, Queries, Delivery Plans, Analytics views, Repos, Pipelines, Test Plans, Artifacts, and Project settings. The main area displays a 'Board' view with a 'Recently updated' filter. A user story titled '17 Login' is selected, showing details for user 'Abiladev Rathiappasamy'. The story is in the 'New' state and is associated with the 'Expert System' project and 'Expert System/Sprint 1' iteration. The story text is 'As a registered user, I want to log into my account so that I can access my personal dashboard and features.' The acceptance criteria are listed as follows:

1. The user should be able to access the login page from the homepage or a direct URL.
2. The login form must have fields for email/username and password.
3. If the user submits an empty field, they should see a validation error.
4. If the email format is invalid, an error message should be displayed.
5. If valid credentials are provided, the user should be redirected to their dashboard.
6. If incorrect credentials are entered, an appropriate error message should be displayed.
7. If the user enters incorrect credentials more than 5 times, their account should be temporarily locked for 10 minutes.
8. The password input field should be masked by default but allow users to toggle visibility.
9. Users should have an option to reset their password via a "Forgot Password?" link.

The right sidebar contains sections for 'Planning' (Story Points: 3, Priority: 2, Risk: 2 - Medium), 'Classification' (Value area: Business), 'Deployment' (To track releases associated with this work item, go to Releases and turn on deployment status reporting for boards in your pipeline's Options menu. Learn more about deployment status reporting), 'Development' (Add link: Link an Azure Repos commit, pull request or branch to see the status of your development. You can also create a branch to get started.), and 'Related Work'.

### Result:

The user story was written successfully.

**EX NO: 5**

## **SEQUENCE DIAGRAM**

**Aim:**

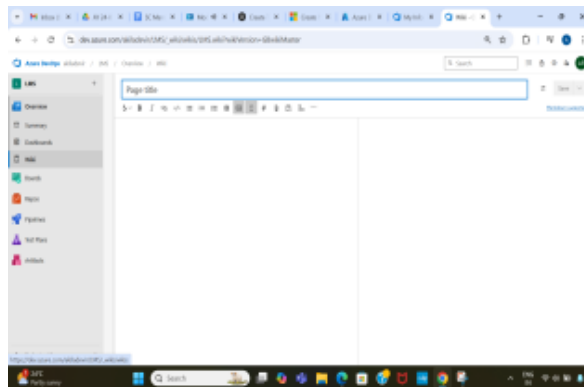
To design a Sequence Diagram by using Mermaid.js

**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 behaviour in a system.

**Procedure:**

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



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

```
sequenceDiagram
    participant User
    participant UI
    participant Project
    participant Task
```

```
User->>UI: Select "Create Task"
UI->>UI: Display "Create Task Form"
User->>UI: Enter task details (title, description, etc.)
User->>UI: Submit form
UI->>Project: createTask(taskData)
Project->>Task: Task(taskData)
Project->>Project: addTask(task)
Project->>UI: Return success/task details
UI->>UI: Display updated task list/message
User->>UI: Navigates to Task List
UI->>Project: getTasks()
Project->>UI: Returns List<Task>
```

## Explanation:

### Task Creation Flow

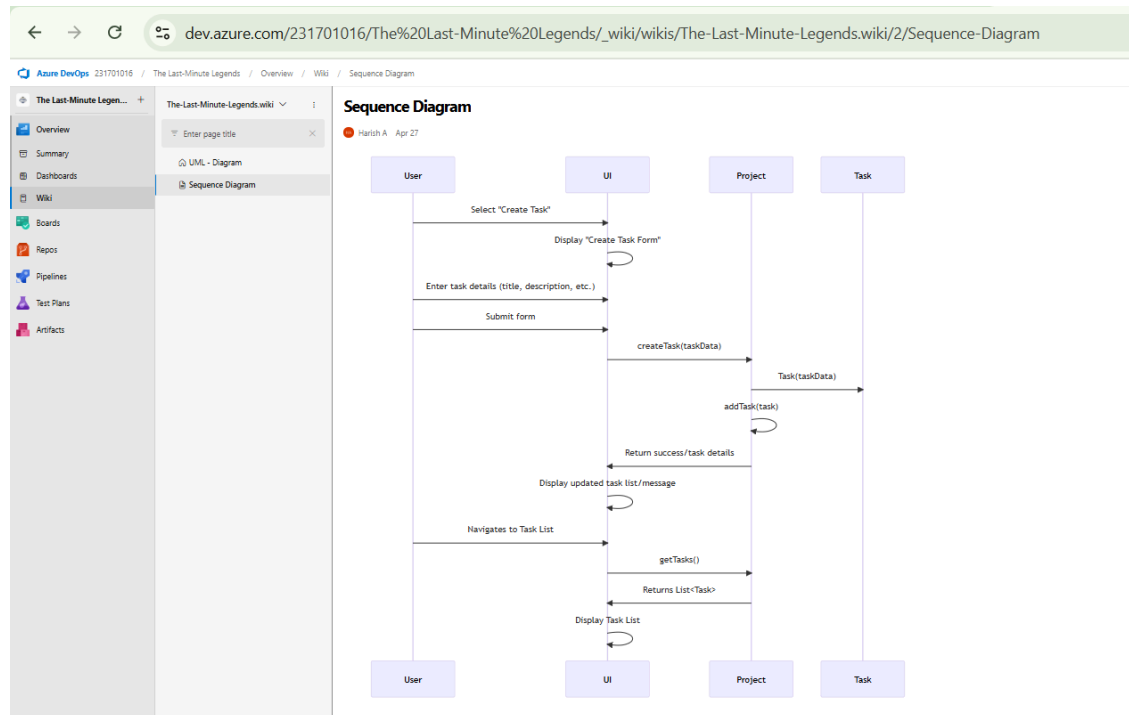
- **User ->> UI: Select "Create Task"**  
The user clicks a button or option in the UI to initiate the task creation process.
- **UI ->> UI: Display "Create Task Form"**  
The UI updates itself to show a form where the user can enter the new task details.
- **User ->> UI: Enter task details (title, description, etc.)**  
The user fills out the form with information like title, description, due date, etc.
- **User ->> UI: Submit form**  
The user submits the form to create the task.
- **UI ->> Project: createTask(taskData)**  
The UI sends the task data to the backend **Project** object to handle task creation.
- **Project ->> Task: Task(taskData)**  
The **Project** creates a new instance of a **Task** using the submitted data.
- **Project ->> Project: addTask(task)**  
The new task is added to the project's internal task list or database.
- **Project ->> UI: Return success/task details**  
The **Project** sends a confirmation and possibly the created task details back to the UI.
- **UI ->> UI: Display updated task list/message**  
The UI shows a success message or updates the task list view to include the new task.

### Viewing the Task List

- **User ->> UI: Navigates to Task List**  
The user goes to the task list view (e.g., by selecting a menu or tab).
- **UI ->> Project: getTasks()**  
The UI requests the current list of tasks from the **Project**.
- **Project ->> UI: Returns List<Task>**  
The **Project** returns the list of existing tasks (including the one just created).
- **UI ->> UI: Display Task List**  
The UI displays all the tasks to the user in a list or card format.



#### 4. click wiki menu and select the page



#### Result:

The sequence diagram was drawn successfully.

## EX NO. 6

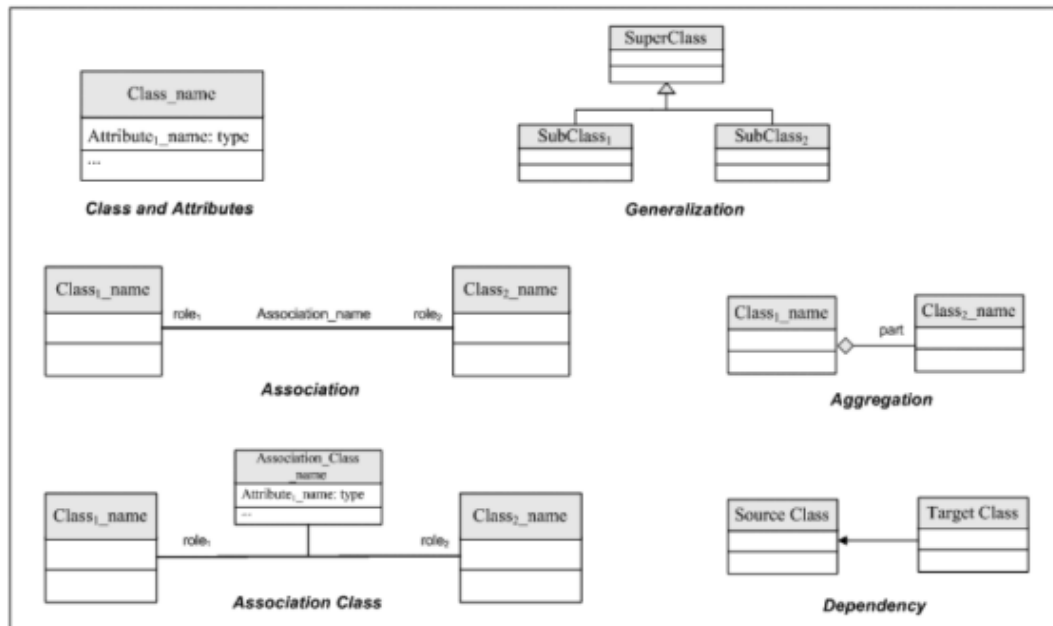
### CLASS DIAGRAM

#### AIM :-

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

#### THEORY

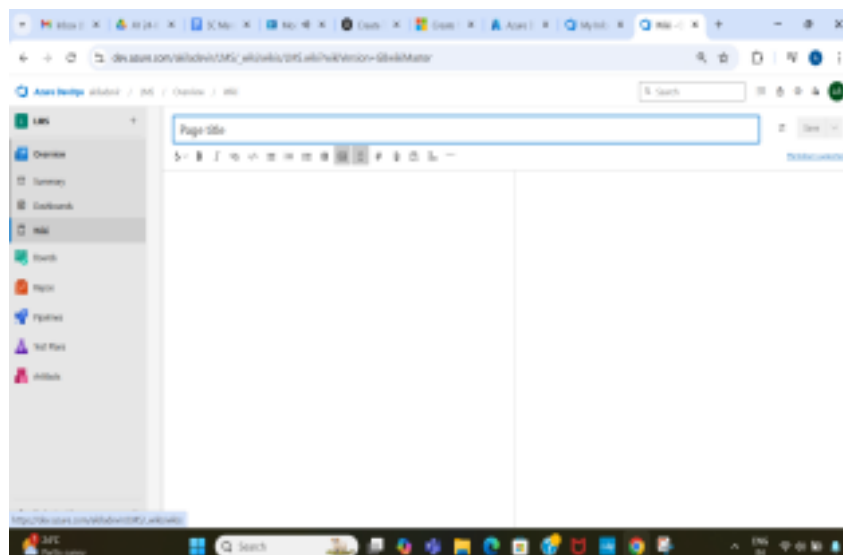
A UML class diagram is a visual tool that represents the structure of a system by showing its classes, attributes, methods, and the relationships between them.



Notations in class diagram

#### Procedure:

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



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

```

::: mermaid
classDiagram
class Task {
    -String title
    -String description
    -String progress
    -DateTime creationDate
    -DateTime dueDate
    -String assignedTo
    +Task(String title, String description, String progress,
DateTime creationDate, DateTime dueDate, String
assignedTo)
    +String getTitle()
    +String getDescription()
    +String getProgress()
    +DateTime getCreationDate()
    +DateTime getDueDate()
    +String getAssignedTo()
    +void setProgress(String progress)
}
class User {
    -String name
    -String userId
    -List<Task> assignedTasks
    +User(String name, String userId)
    +String getName()
    +String getUserId()
    +List<Task> getAssignedTasks()
    +void assignTask(Task task)
}

class Project {
    -String projectName
    -List<User> members
    -List<Task> tasks
    +Project(String projectName)
    +String getProjectName()
    +List<User> getMembers()
    +List<Task> getTasks()
    +void addTask(Task task)
    +void addMember(User user)
}
class UI {
    +void displayTaskList()
    +void displayTaskDetails(Task task)
    +void displayProjectList()
    +void displayCreateTaskForm()
    +void displayCreateProjectForm()
}
Task --|> UI : Uses
User --|> UI : Uses
Project --|> UI: Uses
User --* Task : "assigns"
Project --* User : "manages"
Project --* Task : "contains"

```

### Result:

The use case diagram was designed successfully.

**EX NO: 7**

## **USECASE DIAGRAM**

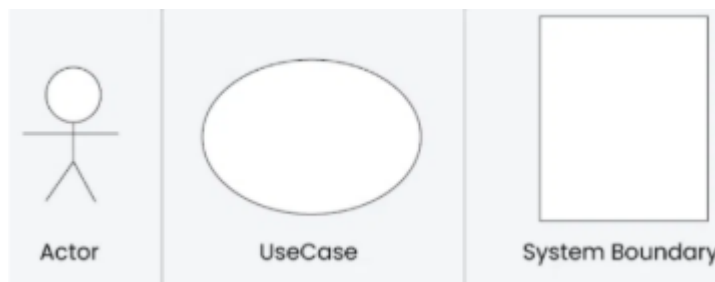
### **Aim:**

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 (diagrams.net).
- 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.

dev.azure.com/231701016/The%20Last-Minute%20Legends/\_wiki/wikis/The-Last-Minute-Legends.wiki?wikiVersion=GBwikiMaster&a=edit&pagePath=/Usecase%20Diagram&pageId=3

Azure DevOps 231701016 / The Last-Minute Legends / Overview / Wiki / Usecase Diagram

Usecase Diagram

Overview  
Summary  
Dashboards  
Wiki  
Boards  
Repos  
Pipelines  
Test Plans  
Artifacts

```

::: mermaid
graph LR;
  A[User] --> B(Upload Customer Data)
  A[User] --> C(Upload Sales Data)
  A[User] --> D(View Dashboard)
  A[User] --> E(Analyze Customer Behavior)
  A[User] --> F(Analyze Sales Performance)
  A[User] --> G(Filter & Segment Data)
  A[User] --> H(Export Insights Report)
  
```

Markdown supported

## Result:

The use case diagram was designed successfully

## EX NO. 8


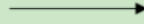
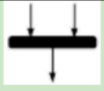








### ACTIVITY DIAGRAM

#### 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

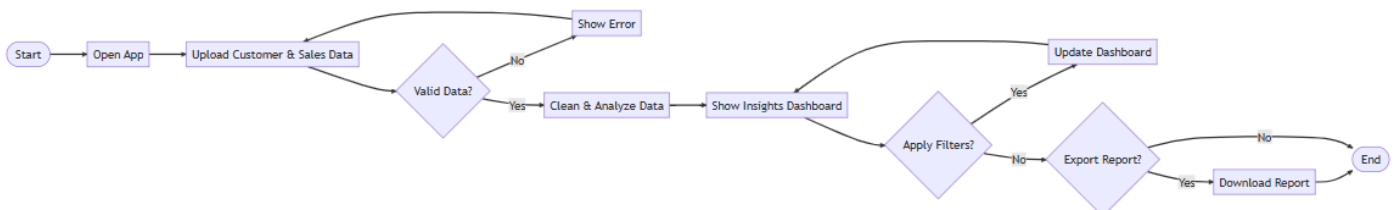
#### Procedure

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

### Activity Diagram

Unfollow 1 Edit

Harish A Just now



#### Result:

The activity diagram was designed successfully

## EX NO. 9

### ARCHITECTURE DIAGRAM

#### Aim:

Steps to draw the Architecture Diagram using draw.io.

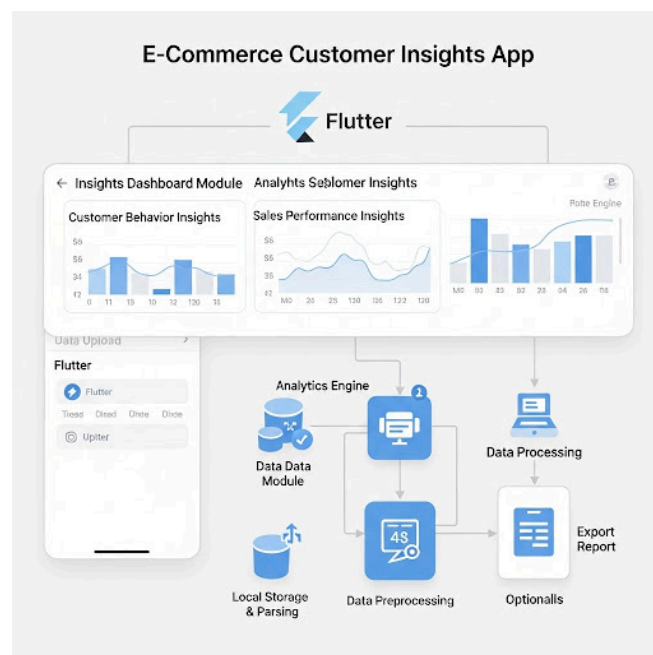
#### Theory:

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:

The architecture diagram was designed successfully

## EX NO. 10

## USER INTERFACE

### Aim:

Design User Interface for the given project

### Code:

```
ExpenseTracker
ExpenseTracker app.py
class ExpenseTracker:
    def __init__(self, page):
        self.page = page
        self.expenses = []
        self.current_tab = 0

        self.expense_name = ft.TextField(
            label="Expense Name", prefix_icon=ft.icons.TITLE, width=300, border_radius=10
        )
        self.expense_amount = ft.TextField(
            label="Expense Amount", prefix_icon=ft.icons.ATTACH_MONEY, width=300, border_radius=10, keyboard_type=ft.KeyboardType.NUMBER
        )
        self.expense_list = ft.Column(scroll=ft.ScrollMode.AUTO, height=300)
        self.total_expense_text = ft.Text("Total Expense: ₹0", style=ft.TextThemeStyle.HEADLINE_SMALL, weight=ft.FontWeight.BOLD, color="#2196F3")
        self.toast = None

        self.chartBars = ft.Row([], alignment=ft.MainAxisAlignment.CENTER, spacing=5)

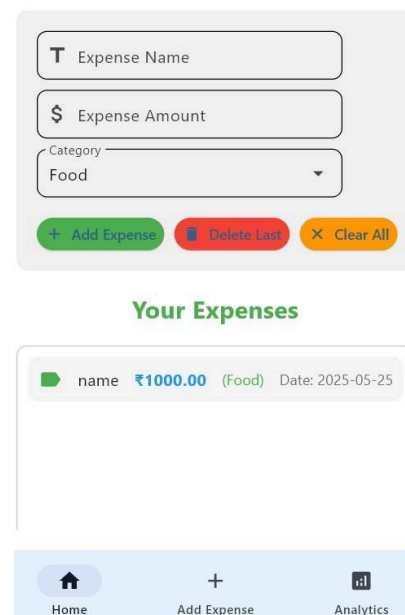
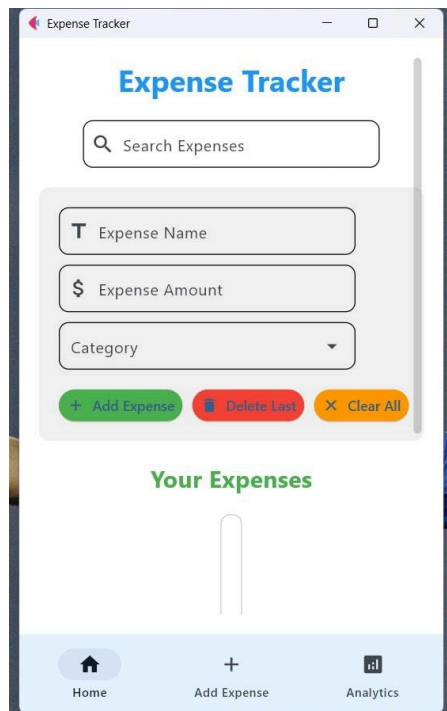
        self.navbar = ft.NavigationBar(
            destinations=[
                ft.NavigationBarDestination(icon=ft.icons.HOME, label="Home"),
                ft.NavigationBarDestination(icon=ft.icons.ADD, label="Add Expense"),
                ft.NavigationBarDestination(icon=ft.icons.ANALYTICS, label="Analytics")
            ],
            selected_index=self.current_tab,
            on_change=self.switch_tab,
            bgcolor="#E3F2FD"
        )

    def add_expense(self, e):
        try:
            name = self.expense_name.value.strip()
            amount = float(self.expense_amount.value.strip())

            if not name:
                self.page.snack_bar = ft.SnackBar(content=ft.Text("Please enter an expense name!"))
                self.page.snack_bar.open = True
                self.page.update()
                return

            # Add expense logic here
        except ValueError:
            # Invalid amount handling
            pass
```

### Output:



### Result:

The UI was designed successfully.



## EX NO. 11

### IMPLEMENTATION

#### **Aim:**

To implement the **Expense Tracker App** project using **Agile Methodology** and streamline development and deployment using **Azure DevOps**.

---

#### **Procedure:**

##### **Step 1: Create an Azure DevOps Project**

- Log in to [Azure DevOps](#).
- Click "**New Project**" → Enter project name (e.g., *ExpenseTrackerApp*) → Click "**Create**".
- Inside the project, go to "**Repos**" to manage and store your source code.

##### **Step 2: Add Your Expense Tracker App Code**

- Navigate to **Repos** → Click "**Clone**" to get the Git URL.  
Open **Visual Studio Code** or a terminal and run the following:

```
git clone <repo_url>
```

```
cd <repo_folder>
```

- Add your application code (HTML, CSS, JavaScript, React, Angular, or backend like Node.js, Python, .NET, etc.).

```
git add .
```

```
git commit -m "Initial commit of Expense Tracker App"
```

```
git push origin main
```

##### **Step 3: Set Up the Build Pipeline (CI - Continuous Integration)**

- Go to **Pipelines** → Click "**New Pipeline**".

- Select your repository (e.g., Azure Repos Git).
- Choose a **Starter Pipeline** or a template based on your tech stack.
- Edit the `azure-pipelines.yml` file (example below for a **Node.js** app):

```

yaml
CopyEdit
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 Expense Tracker App'
  - task: PublishBuildArtifacts@1
    inputs:
      pathToPublish: 'dist'
      artifactName: 'drop'

```

- Click "**Save and Run**" → This will start the build process.

#### Step 4: Set Up the Release Pipeline (CD - Continuous Deployment)

- Go to **Releases** → Click "**New Release Pipeline**".
- Choose a deployment target (e.g., **Azure App Service**, **Virtual Machine**, etc.).
- Add the **build artifact** from the previous step.
- Configure environments like **Development**, **Testing**, and **Production**.
- Define deployment tasks (e.g., install packages, configure environment variables).
- Click "**Deploy**" to publish the **Expense Tracker App** to your chosen environment.

#### Result

Thus the application was successfully implemented.