



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

DEPARTMENT OF INFORMATION TECHNOLOGY

ACADEMIC YEAR 2024-2025

CS23432 SOFTWARE ENGINEERING LAB

LAB RECORD

2024- 2025

NAME :.....Meenakshi.R.....
REGISTER NO : 231001110
BRANCH :.BTech .Information Technology..
SEMESTER : 4 th
ACADEMIC YEAR : 2024-2025

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

Ex No	Date	Topic	Page No	Sign
1		Study of Azure DevOps Writing		
2		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		

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:

Candidate Performance Analysis & Recommendation Tool

Most of the IT Companies are recruiting people with good programming and analytical skills. Building a Performance Analysis tool will help the educational institutions/organizations to guide the candidate to excel in their career. This tool helps students to get their overall Strength, Weakness & To-Do-Course report for improving his/her performance.

RESULT:

The problem statement was written successfully.

AGILE PLANNING

AIM:

To prepare an Agile Plan.

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 ultimate 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 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]."

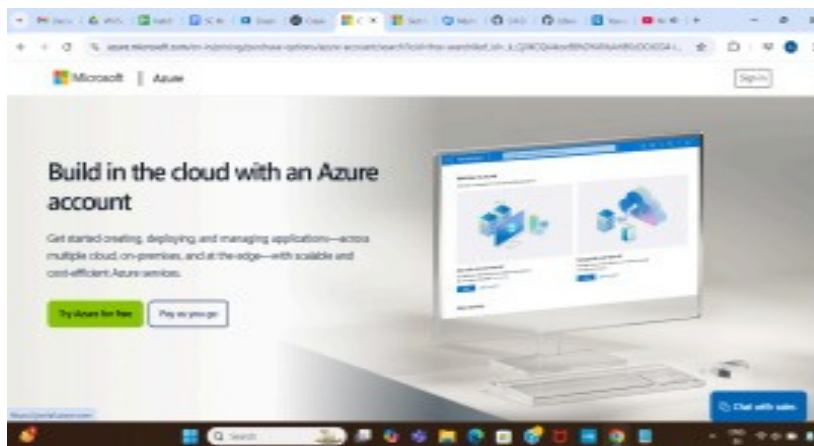
PROCEDURE:

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

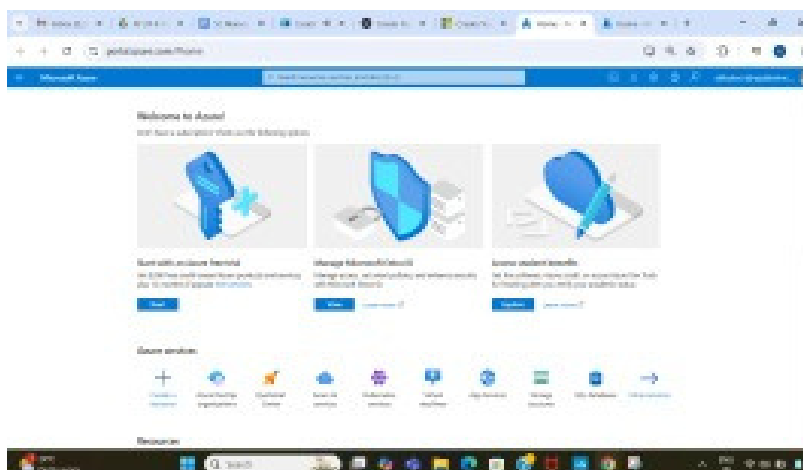
<https://azure.microsoft.com/en-in> Sign in using your Microsoft account credentials. If you don't have an account, you'll need to create one.

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

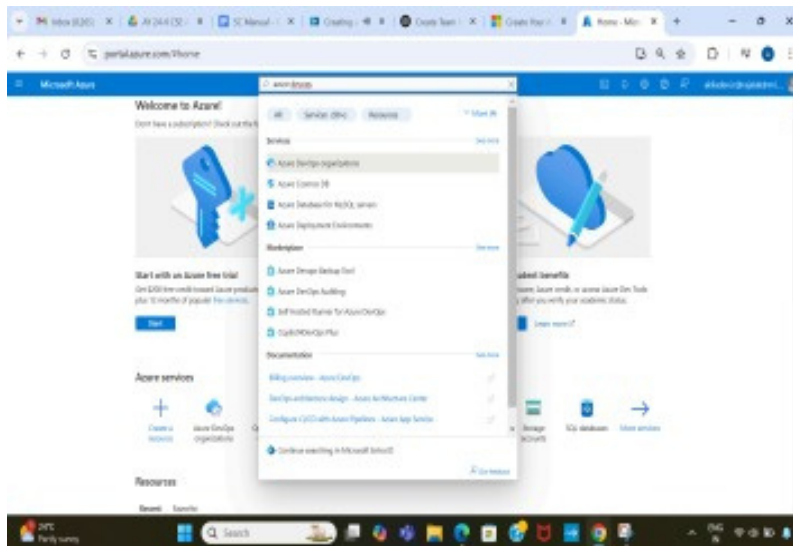
for <https://signup.live.com/?lic=1>



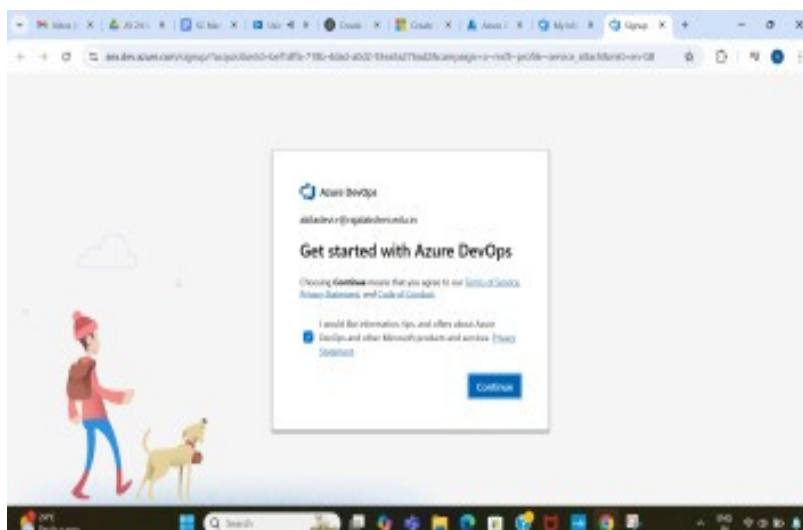
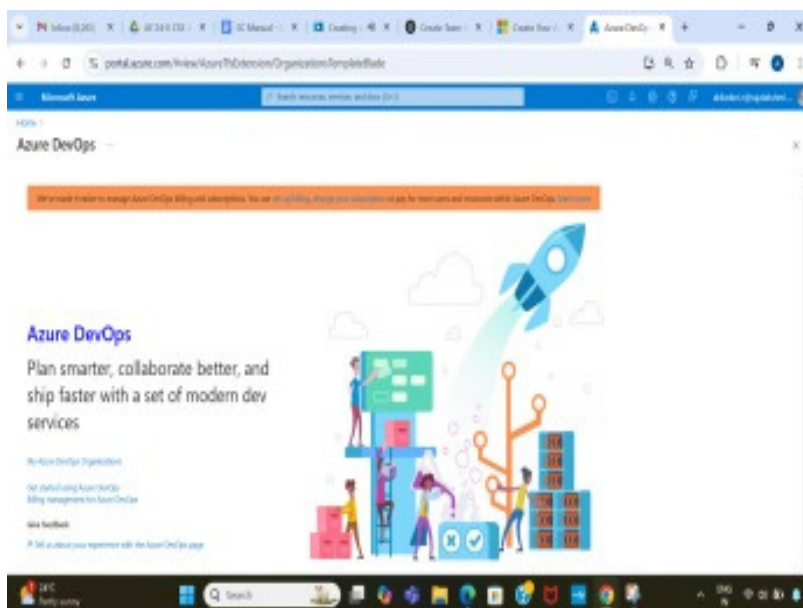
3. Azure home page

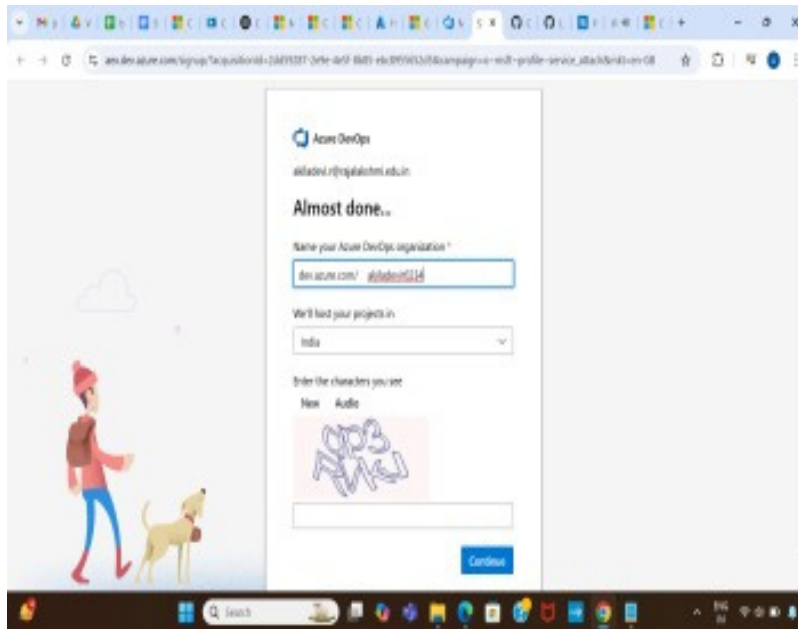


4. Open DevOps environment in the Azure platform by typing Azure DevOps Organizations in the search bar.



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

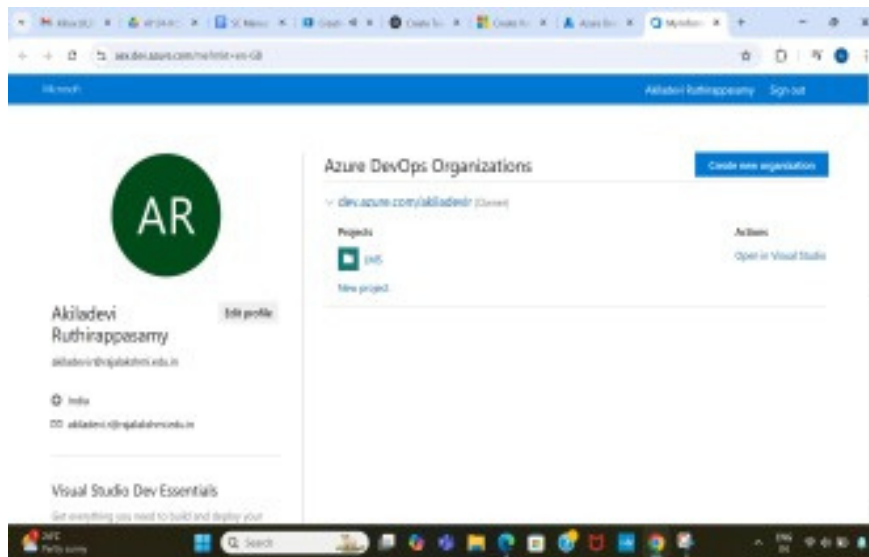
After the organization is set up, you'll need to create your first **project**. This is where you'll begin to manage code, pipelines, work items, and more.

i. On the organization's **Home page**, click on the **New Project** button. ii. Enter the project name, description, and visibility options:

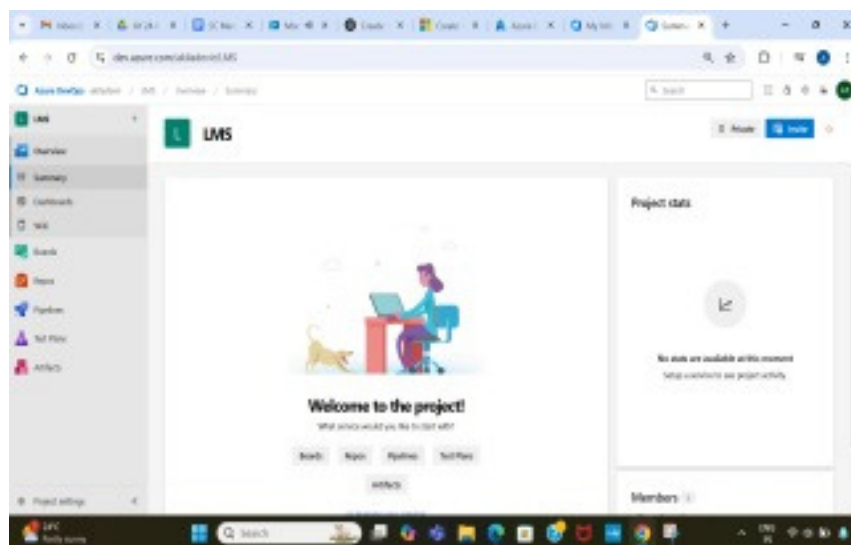
- **Name:** Choose a name for the project (e.g., **LMS**).
- **Description:** Optionally, add a description to provide more context about the project.
- **Visibility:** Choose whether you want the project to be **Private** (accessible only to those invited) or **Public** (accessible to anyone).

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.

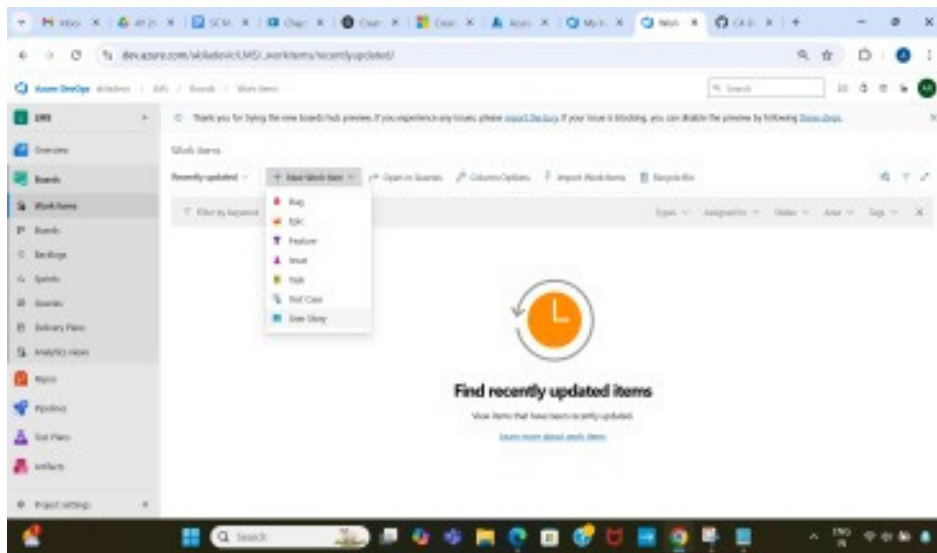


8. Project dashboard

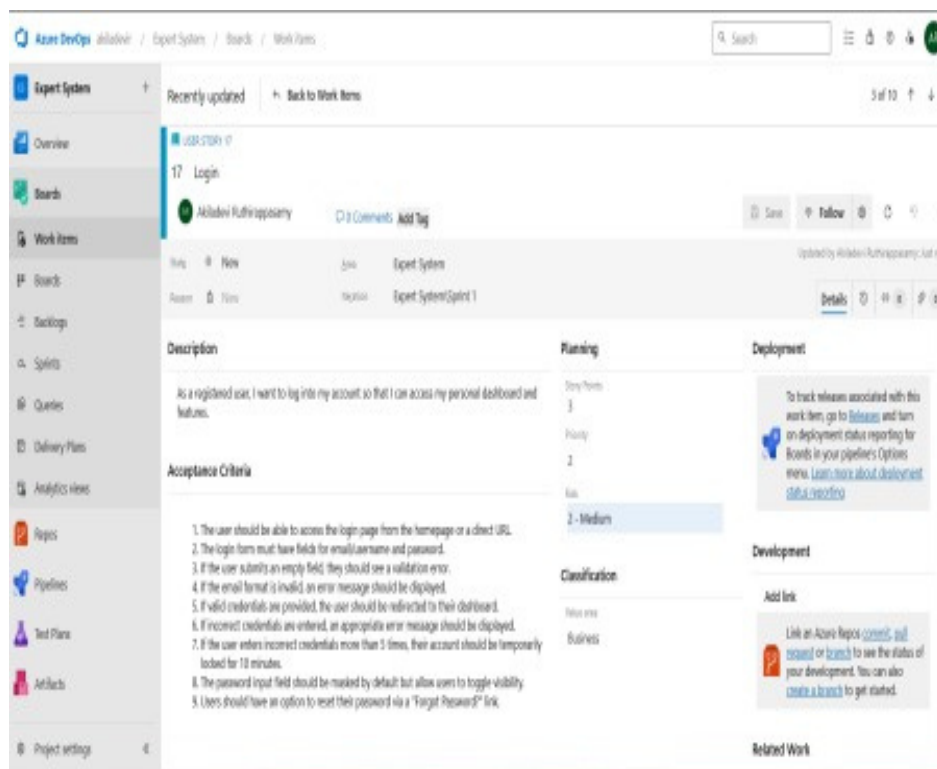


9. To manage user stories

- 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 was written successfully.

EX NO: 5

SEQUENCE DIAGRAM

AIM:

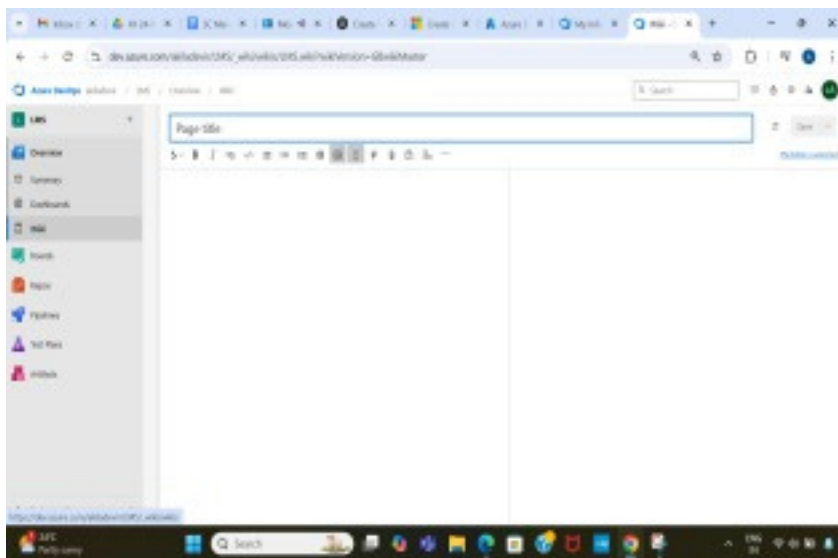
To design a Sequence Diagram by using Mermaid.js

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
```

```
sequenceDiagram
```

```
participant Employee participant HR participant Admin
```

```
participant System as "Employee management system"
```

```
Employee->>System: Enter the attendance
```

```
System->>HR: Update the attendance
```

```
Employee->>HR: Check Deduction()
```

```
Employee->>HR: Request Leave()
```

```
HR->>Employee: Accept/Reject Leave()
```

```
Employee->>HR: Check Salary()
```

```
HR->>System: Request salary()
```

```

HR ->> System: Request salary()
System ->> Admin: Request Employee Details()
Admin ->> System: Provide Employee
Details() System ->> System: Reduce Tax()
System ->> System: Add Allowance()
System ->> System: Calculate Salary()
System ->> Employee: Issue Salary()
...

```

EXPLANATION:

sequenceDiagram

%% participant defines the entities involved

participant Employee

participant HR

participant Admin

participant System as "Employee management system"

%% - >> Solid line with arrowhead: Direct message

%% + after ->> activates a participant

Employee ->>+ System: Enter the attendance

%% ->> Solid line with arrowhead: Direct message

System ->> HR: Update the attendance

%% ->> Solid line with arrowhead: Direct message

Employee ->> HR: Check Deduction()

%% ->> Solid line with arrowhead: Direct message

Employee ->> HR: Request Leave()

%% -->> Dotted line with arrowhead: Response message

%% - after -->> deactivates a participant

HR -->>- Employee: Accept/Reject Leave()

%% ->> Solid line with arrowhead: Direct message

Employee ->> HR: Check Salary()

%% ->> Solid line with arrowhead: Direct message

%% + after ->> activates a participant

HR ->>+ System: Request salary()

%% ->> Solid line with arrowhead: Direct message

%% + after ->> activates a participant

System ->>+ Admin: Request Employee Details()

%% -->> Dotted line with arrowhead: Response message

%% - after -->> deactivates a participant

Admin -->>- System: Provide Employee Details()

%% ->> Self-message: Internal operation (solid line with arrowhead)

System ->> System: Reduce Tax()

%% ->> Self-message: Internal operation (solid line with arrowhead)

System ->> System: Add Allowance()

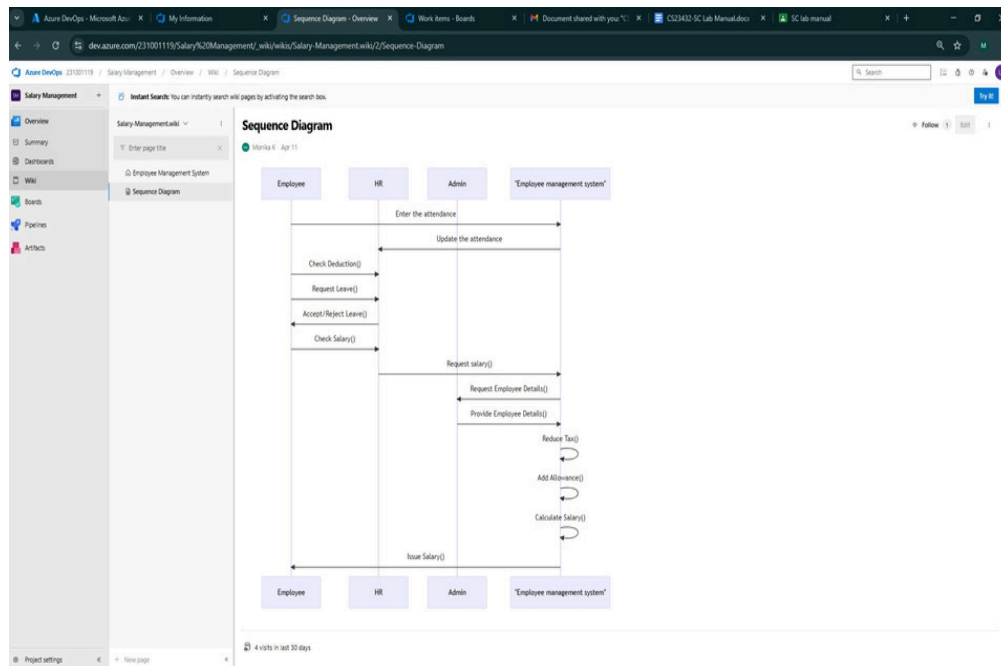
%% ->> Self-message: Internal operation (solid line with arrowhead)

System ->> System: Calculate Salary()

%% -->> Dotted line with arrowhead: Response message to Employee

System -->> Employee: Issue Salary()

4. click wiki menu and select the page



RESULT:

The sequence diagram was drawn successfully.

EX NO. 6

CLASS DIAGRAM

AIM :

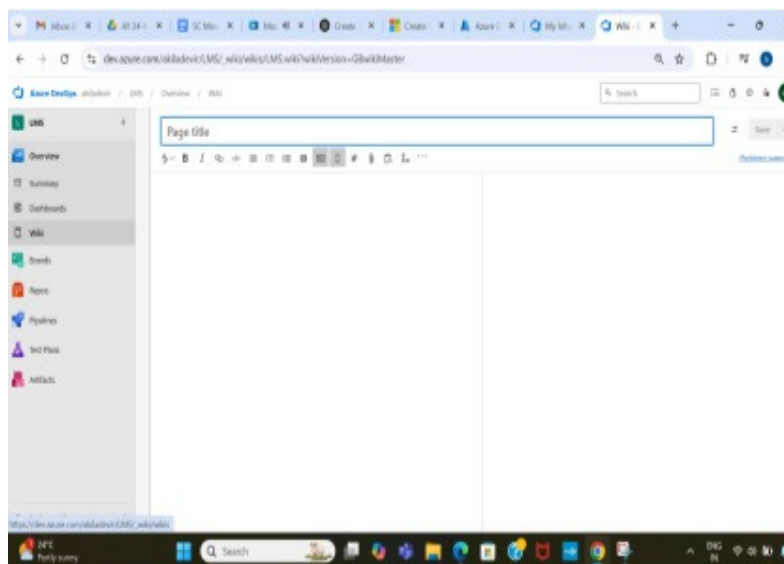
To draw a sample class diagram for your project or system.

THEORY:

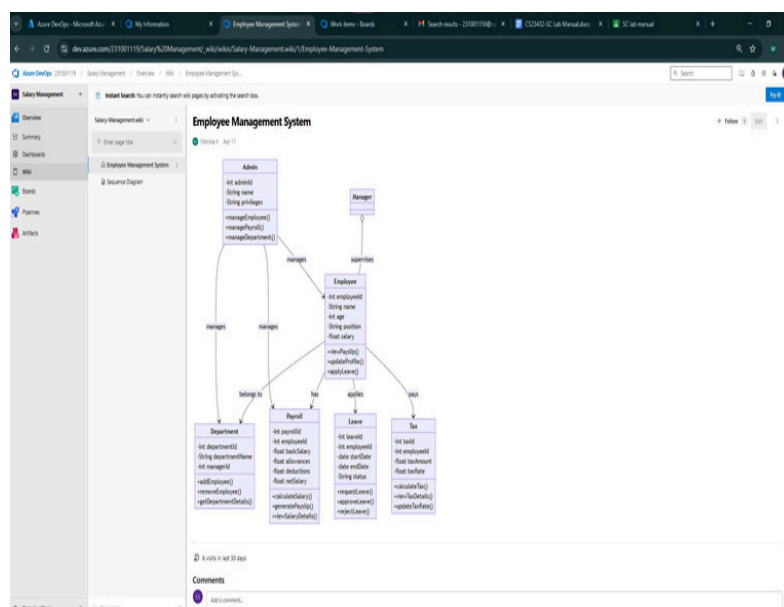
A UML class diagram is a visual tool that represents the structure of a system by showing its classes, attributes, methods, and the relationships between them.

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.



EXPLANATION:

@startuml

title Employee Management System Class Diagram

class Admin {

-int adminId

-String name

-String privileges

+manageEmployee()

+managePayroll()

+manageDepartment()

}

class Manager {

}

class Employee {

-int employeeId

-String name

-int age

-String position

-float salary

+viewPayslip()

+updateProfile()

+applyLeave()

}

class Department {

-int departmentId

-String departmentName

-int managerId

+addEmployee()

+removeEmployee()

+getDepartmentDetails()

}

class Payroll {

-int payrollId

-int employeeId

```
-float basicSalary  
  
-float allowances  
  
-float deductions  
  
-float netSalary  
  
+calculateSalary()  
  
+generatePayslip()  
  
+viewSalaryDetails()  
  
}
```

```
class Leave {
```

```
-int leaveId  
  
-int employeeId  
  
-date startDate  
  
-date endDate  
  
-String status  
  
+requestLeave()  
  
+approveLeave()  
  
+rejectLeave()
```

}

class Tax {

-int taxId

-int employeeId

-float taxAmount

-float taxRate

+calculateTax()

+viewTaxDetails()

+updateTaxRate()

}

Admin --> Employee : manages

Admin --> Department : manages

Admin --> Payroll : manages

Department --> Employee : belongs to

Employee --> Payroll : has

Employee --> Leave : applies

Employee --> Tax : pays

Manager *-- Employee : supervises

@enduml

RESULT:

The use case diagram was designed successfully.

EX NO: 7

USE CASE DIAGRAM

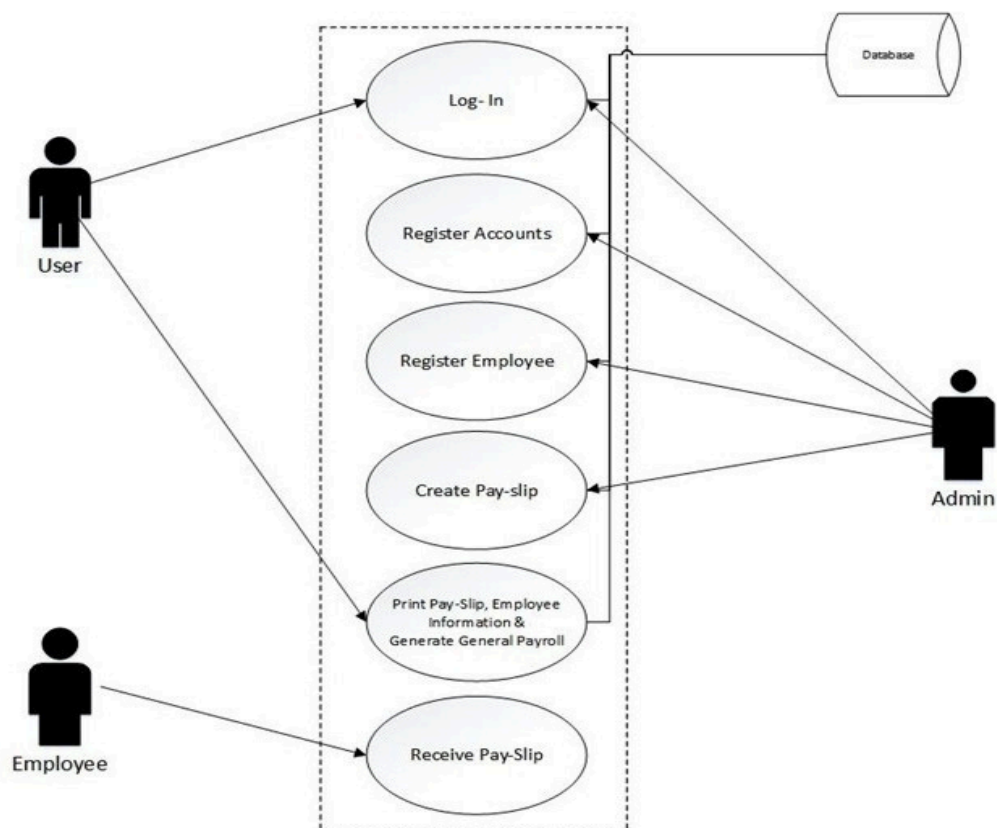
AIM:

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.
 - Use Markdown to embed the diagram:
 - Drag & Drop the exported PNG/JPG image.
 - Navigate to Wiki (Project > Wiki).
 - Click "Edit Page" or create a new page.
- 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.

RESULT:

The use case diagram was designed successfully.



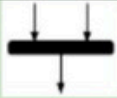








ACTIVITY DIAGRAM

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

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.



ARCHITECTURE OF SALARY MANAGEMENT SYSTEM

PROCEDURE:

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

RESULT:

The architecture diagram was designed successfully

USER INTERFACE

AIM:

Design User Interface for the given project.

THEORY:

A Salary Management System (SMS) user interface (UI) theory focuses on designing an interface that enables HR personnel, accountants, and employees to efficiently manage and interact with salary-related information. Below are the key theoretical aspects of a well-designed UI for such a system.

1. User Roles & Access Control

Admin: Full access to manage salaries, users, roles, and settings.

HR Manager: Access to employee records, salary configurations, and reports.

Employee: View-only access to their own salary details, payslips, and tax information.

2. Key UI Components

a. Dashboard

Overview of payroll status, recent activity, upcoming pay cycles.

Alerts for due dates or errors in salary processing.

b. Employee Management Module

Add/edit/delete employee records.

View individual salary structure, allowances, deductions, bonuses.

c. Salary Processing Module

Generate monthly payroll.

Auto-calculation of taxes, benefits, deductions.

Bulk upload options (e.g., Excel/CSV).

d. Payslip Generation

Option to download/print payslips.

Filters for date ranges, departments, etc.

Customizable reports: salary summaries, tax reports, department-wise payroll.

Export options (PDF, Excel).

f. Settings

Define pay grades, tax slabs, deduction rules.

Currency, financial year, and organizational preferences.

3. UI Design Principles

Clarity: Clean layout with easy navigation.

Consistency: Standardized color schemes, button placement, and iconography.

Responsiveness: Optimized for desktops, tablets, and mobile devices.

Security: Clearly visible logout, password update, and 2FA features.

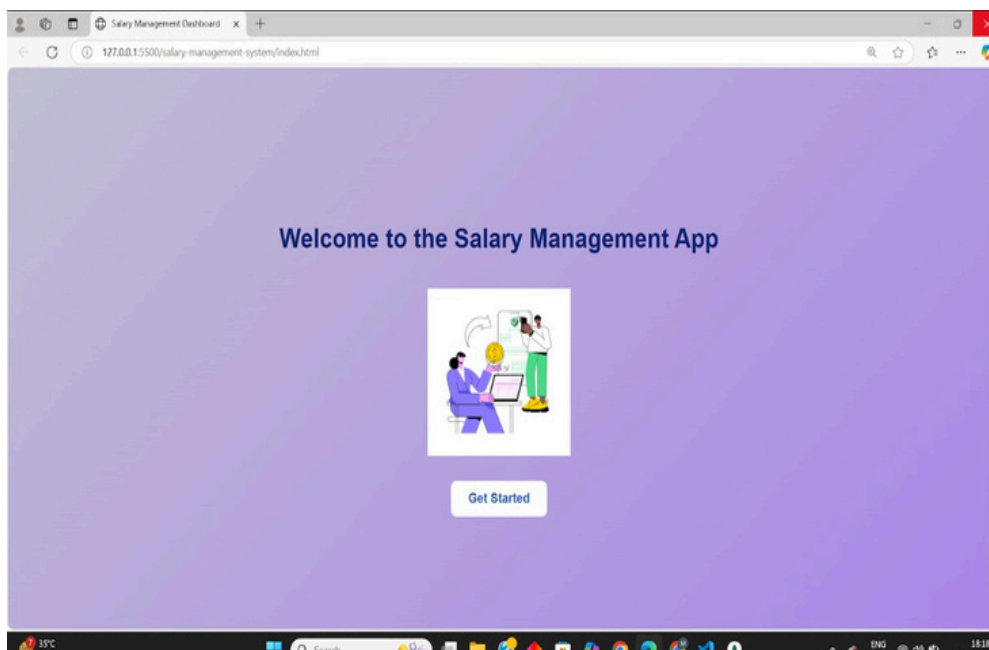
Accessibility: Support for keyboard navigation, screen readers, and high-contrast modes.

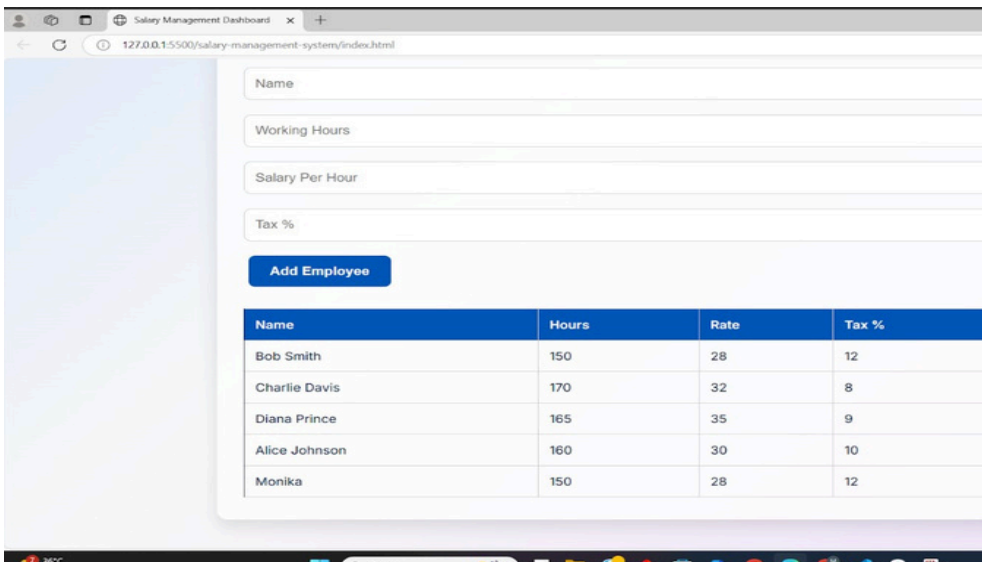
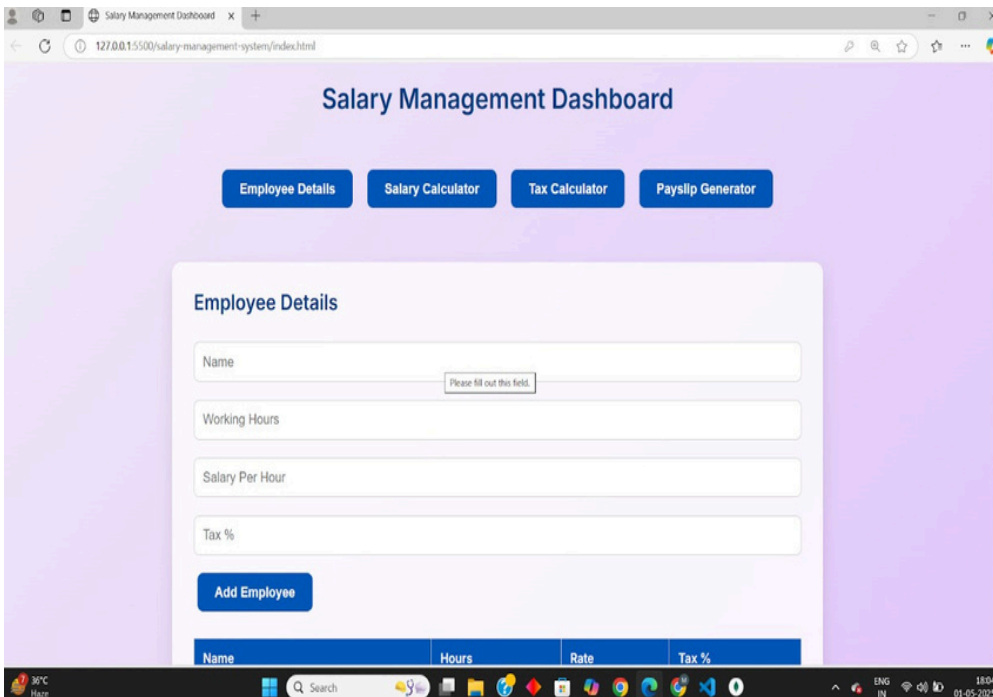
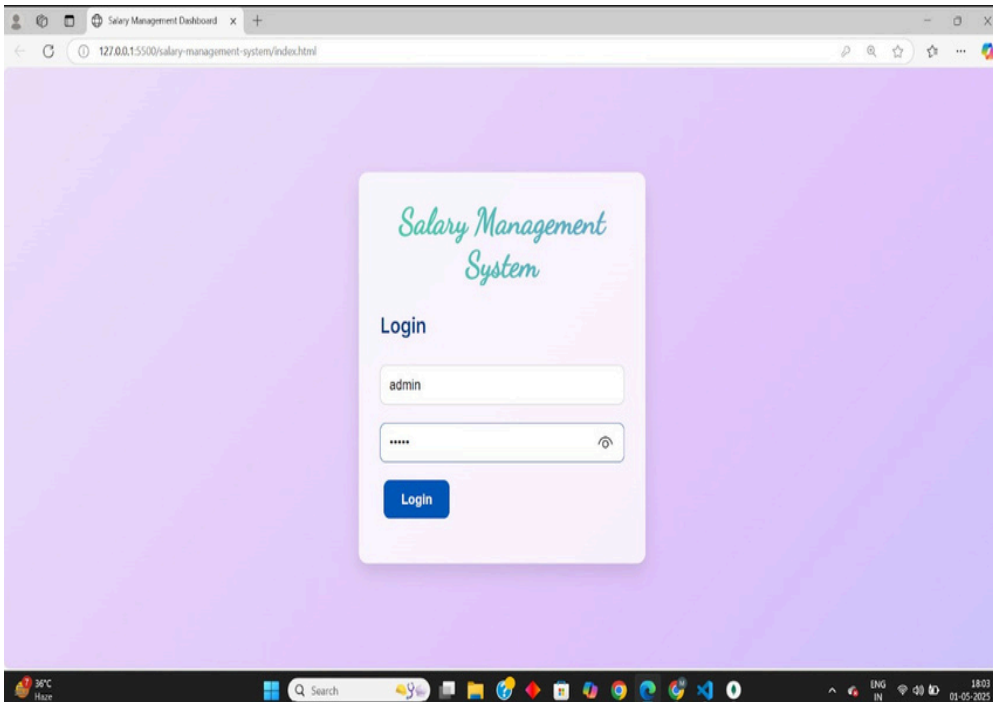
4. Technologies (Optional for Implementation)

Frontend: HTML, CSS

Backend: Node.js

Database: MONGODB





Salary Management Dashboard

Employee Details

Salary Calculator

Tax Calculator

Payslip Generator

Salary Calculator

Name	Monthly Salary	Annual Salary
Bob Smith	\$4200.00	\$50400.00
Charlie Davis	\$5440.00	\$65280.00
Diana Prince	\$5775.00	\$69300.00
Alice Johnson	\$4800.00	\$57600.00
Monika	\$4200.00	\$50400.00
Total	\$24415.00	\$292980.00

Salary Management Dashboard

Employee Details

Salary Calculator

Tax Calculator

Payslip Generator

Tax Calculator

Name	Monthly Tax
Bob Smith	\$504.00
Charlie Davis	\$435.20
Diana Prince	\$519.75
Alice Johnson	\$480.00
Monika	\$504.00
Total Annual Tax	\$29315.40

Salary Management Dashboard

Employee Details

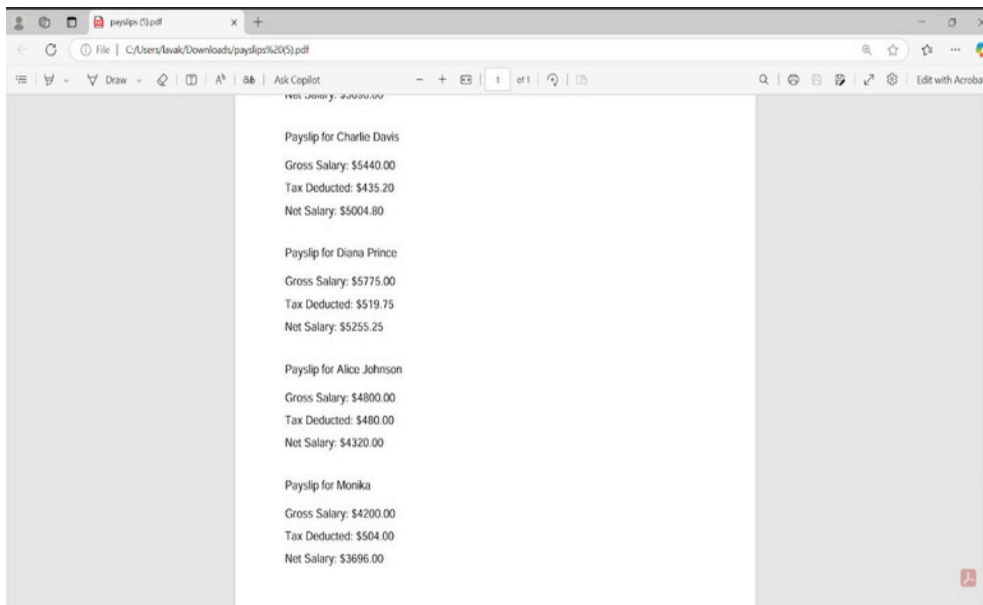
Salary Calculator

Tax Calculator

Payslip Generator

Generate Payslip

Download Payslips as PDF



RESULT:

The UI was designed successfully.

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 <repo_url>
cd <repo_folder>
```
- 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.

Step 5: Implementation code for Salary Management System.

index.html(UI code):

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
  <title>Salary Management Dashboard</title>
  <!-- Fonts -->
  <link href="https://fonts.googleapis.com/css2?family=Inter:wght@400;600&display=swap"
rel="stylesheet">
  <link href="https://fonts.googleapis.com/css2?family=Dancing+Script&display=swap"
rel="stylesheet">
  <script src="https://cdnjs.cloudflare.com/ajax/libs/jspdf/2.5.1/jspdf.umd.min.js"></script>
  <style>

  * {
    box-sizing: border-box;
    margin: 0;
    padding: 0;
  }
  body {

    font-family: 'Inter', sans-serif;
    background: linear-gradient(-45deg, #8ec5fc, #e0c3fc, #f9f9f9, #a1c4fd);
    background-size: 400% 400%;
    animation: gradientBG 15s ease infinite;
    min-height: 100vh;
    color: #002244;
  }
  @keyframes gradientBG {

    0% { background-position: 0% 50%; }
    50% { background-position: 100% 50%; }
    100% { background-position: 0% 50%; }
  }
  h1, h2 {

    color: #003d80;
    margin-bottom: 20px;
  }
}
```

```

.gradient-title {
  font-family: 'Dancing Script', cursive;
  font-size: 2.5rem;
  font-weight: bold;
  background: linear-gradient(270deg, #ff6ec4, #7873f5, #42e695);
  background-size: 600% 600%;
  -webkit-background-clip: text;
  -webkit-text-fill-color: transparent;
  animation: gradientFlow 5s ease infinite;
  text-align: center;
  margin-bottom: 30px;
}
@keyframes gradientFlow {
  0% { background-position: 0% 50%; }
  50% { background-position: 100% 50%; }
  100% { background-position: 0% 50%; }
}
.screen {
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  min-height: 100vh;
  padding: 20px;
}
.hidden {
  display: none;
}
.page {
  background: #ffffffcc;
  padding: 30px;
  margin: 20px auto;
  border-radius: 12px;
  box-shadow: 0 10px 20px rgba(0,0,0,0.1);
  width: 100%;
  max-width: 900px;
  transition: all 0.3s ease;
}
input[type="text"],
input[type="number"],
input[type="password"] {
  width: 100%;
  padding: 12px;
  margin: 10px 0;
  border: 1px solid #ccc;
  border-radius: 8px;
  font-size: 1rem;
}
input:focus {

```

```

outline: none;
border-color: #0057b7;
}
button {

background-color: #0057b7;
color: #ffffff;
border: none;
padding: 12px 24px;
font-size: 1rem;
font-weight: 600;
border-radius: 8px;
margin: 10px 5px;
cursor: pointer;
transition: background-color 0.3s;
}
button:hover {

background-color: #003d80;
}
.nav {

margin: 30px 0;
display: flex;
flex-wrap: wrap;
gap: 10px;
justify-content: center;
}
table {

width: 100%;
border-collapse: collapse;
margin-top: 20px;
font-size: 0.95rem;
}
th, td {

border: 1px solid #ccc;
padding: 12px;
text-align: left;
}
th {

background-color: #0057b7;
color: white;
}
#login-error {

color: red;
margin-top: 10px;
}
@media (max-width: 600px) {

```

```

.page {
padding: 20px;
input, button {
width: 100%;
}
.nav {

flex-direction: column;
align-items: stretch;
}
}
</style>
</head>
<body>
<div id="login-screen" class="screen">
<div class="page" style="max-width: 400px;">
<h1 class="gradient-title">Salary Management System</h1>
<h2>Login</h2>
<input type="text" id="username" placeholder="Username" />
<input type="password" id="password" placeholder="Password" />
<button onclick="login()">Login</button>
<p id="login-error"></p>
</div>
</div>
<div id="dashboard" class="screen hidden">
<h1>Salary Management Dashboard</h1>
<div class="nav">

<button onclick="showPage('employee')">Employee Details</button>
<button onclick="showPage('salary')">Salary Calculator</button>
<button onclick="showPage('tax')">Tax Calculator</button>
<button onclick="showPage('payslip')">Payslip Generator</button>
</div>
<div id="employee" class="page">

<h2>Employee Details</h2>
<form id="employeeForm">
<input type="text" id="empName" placeholder="Name" required />
<input type="number" id="workingHours" placeholder="Working Hours" required />
<input type="number" id="salaryPerHour" placeholder="Salary Per Hour" required />
<input type="number" id="taxPercent" placeholder="Tax %" required />
<button type="submit">Add Employee</button>
</form>
<table id="employeeTable"></table>
</div>
<div id="salary" class="page hidden">

<h2>Salary Calculator</h2>
<div id="salaryOutput"></div>

```

```
</div>
<div id="tax" class="page hidden">
  <h2>Tax Calculator</h2>
  <div id="taxOutput"></div>
</div>
<div id="payslip" class="page hidden">
  <h2>Generate Payslip</h2>
  <button onclick="generatePayslip()">Download Payslips as PDF</button>
  <div id="payslipOutput"></div>
</div>
</div>
<script src="app.js"></script>

</body>
</html>
```

RESULT:

Thus the application was successfully implemented.