**RAJALAKSHMI ENGINEERING COLLEGE**

**THANDALAM – 602 105**

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

**ACADEMIC YEAR 2024-2025**



|  |
| --- |
| **CS23432**  **SOFTWARE CONSTRUCTION** |
| **Lab Manual**  **2024-2025** |

|  |
| --- |
| **Name :** R YASHVINTHINI  **Year/Branch/Section :** II /CSE/ D  **Register No. :** 230701386  **Semester :** IV  **Academic Year:** 2024-25 |

**INDEX**

|  |  |
| --- | --- |
| **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 Usecse 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 : 1 STUDY OF AZURE DEVOPS**

**AIM:**

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

**STUDY:**

**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:**

Manual salary processing in organizations is time-consuming, error-prone, and inefficient, especially when dealing with varied pay structures, tax rules, and attendance patterns. HR departments often struggle to accurately track attendance, process payroll, and ensure compliance with tax regulations. Additionally, employees lack transparency in salary breakdowns and leave tracking, leading to confusion and dissatisfaction.  
There is a need for a centralized, automated salary management system that integrates attendance, payroll, and tax deductions to improve accuracy, reduce manual effort, and enhance transparency for both HR professionals and employees.

**RESULT:**

The Problem statement is written successfully.

**EX NO : 3 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 STORY**

**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.

**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. Go to 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.

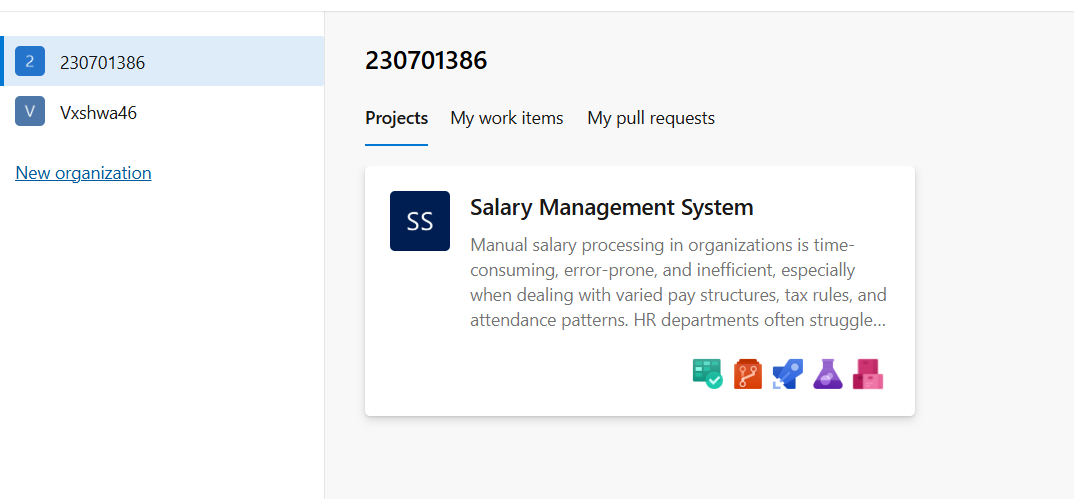
6. Create the First Project in Your Organization. After the organization is set up, you’ll need to create your first project. This is where you'll begin to manage code, pipelines, work items, and more.

(i)On the organization’s Home page, click on the New Project button.

(ii)Enter the project name, description, and visibility options:

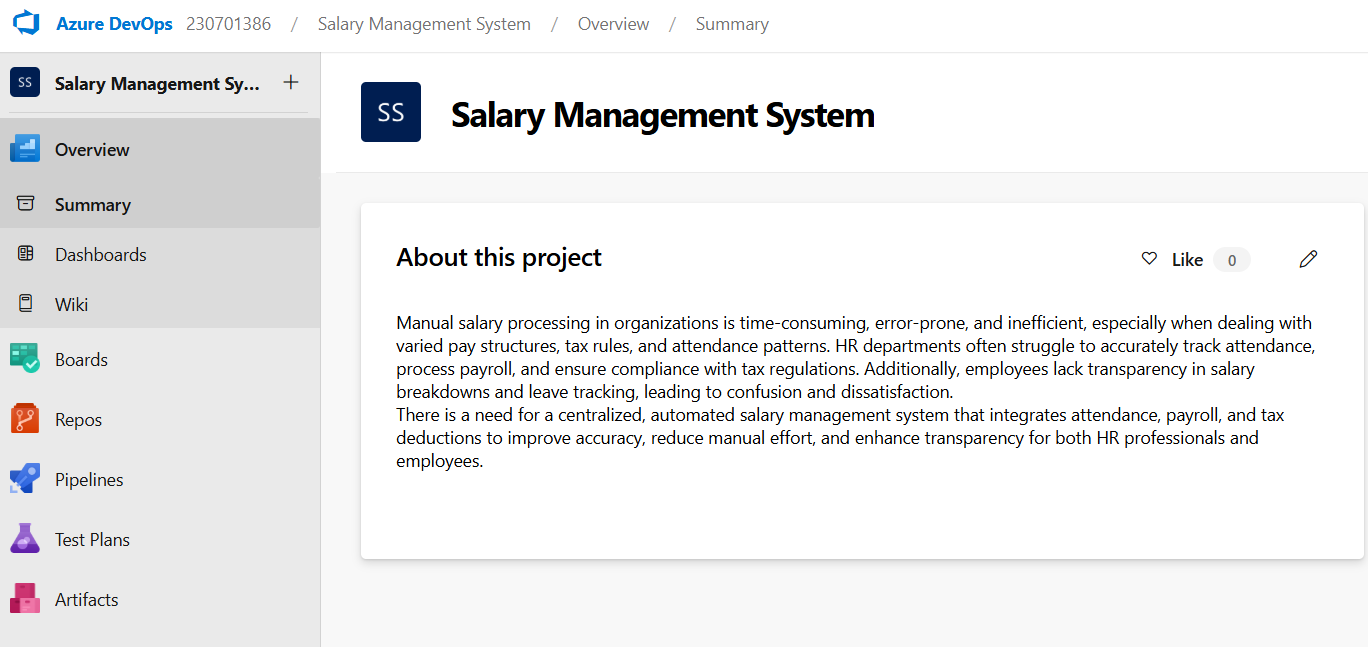
* Name: Choose a name for the project (e.g., LMS).
* Description: Optionally, add a description to provide more context about the project.
* Visibility: Choose whether you want the project to be Private
* (accessible only to those invited) or Public (accessible to anyone).

(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. Open project’s dashboard.

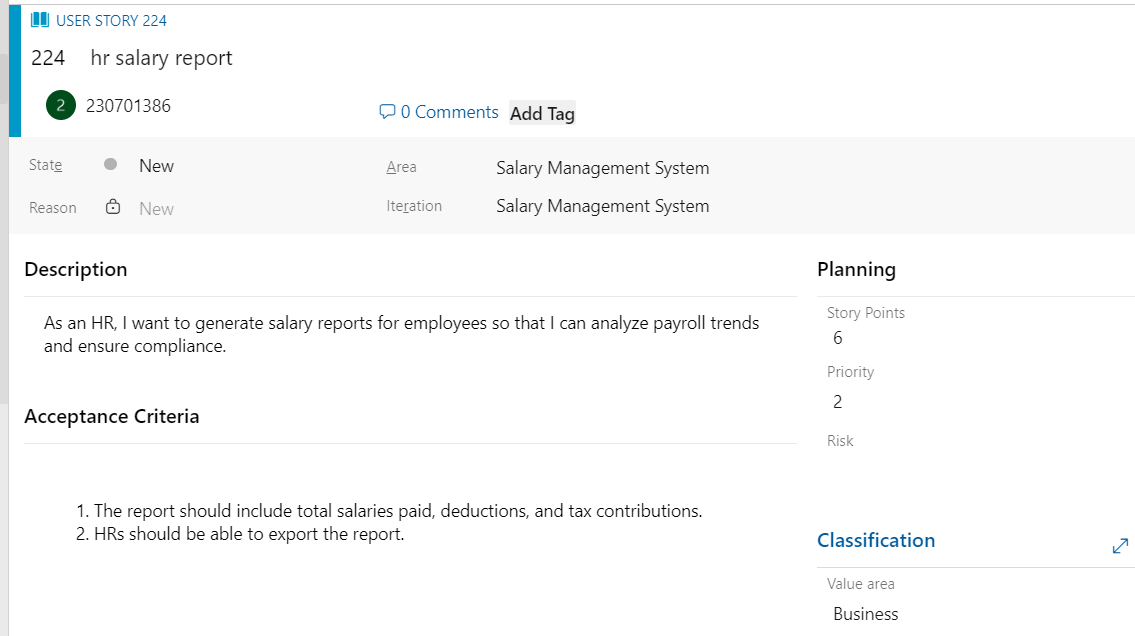


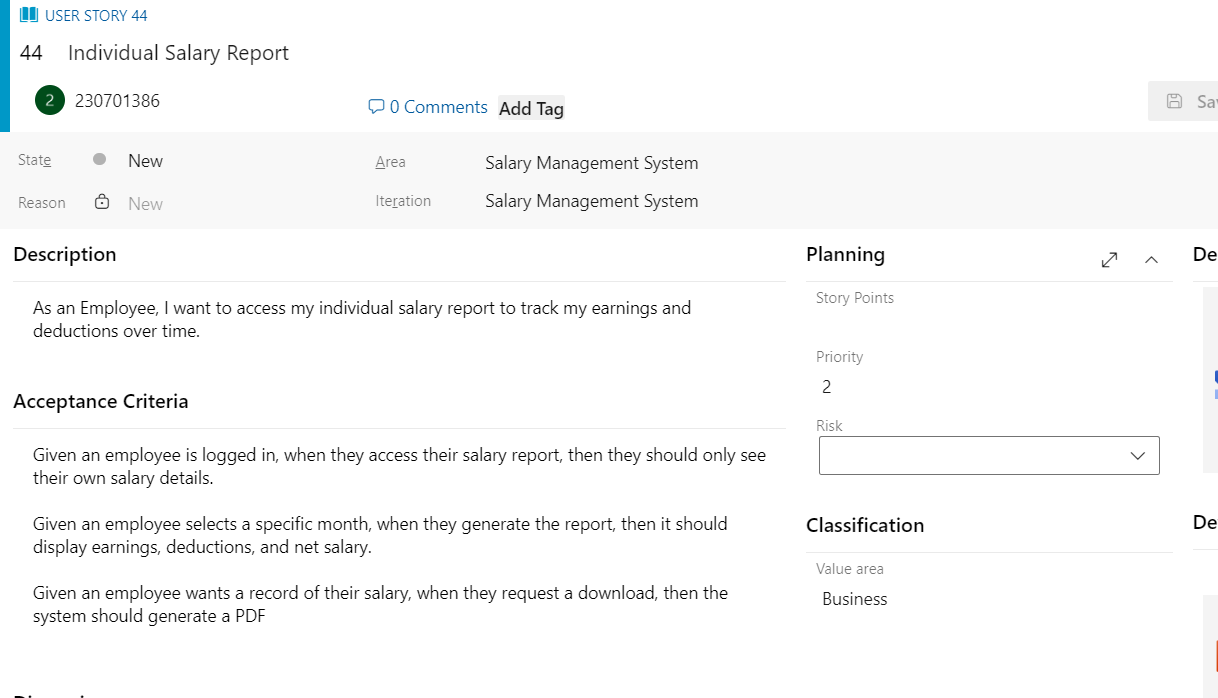
9. To manage user stories:

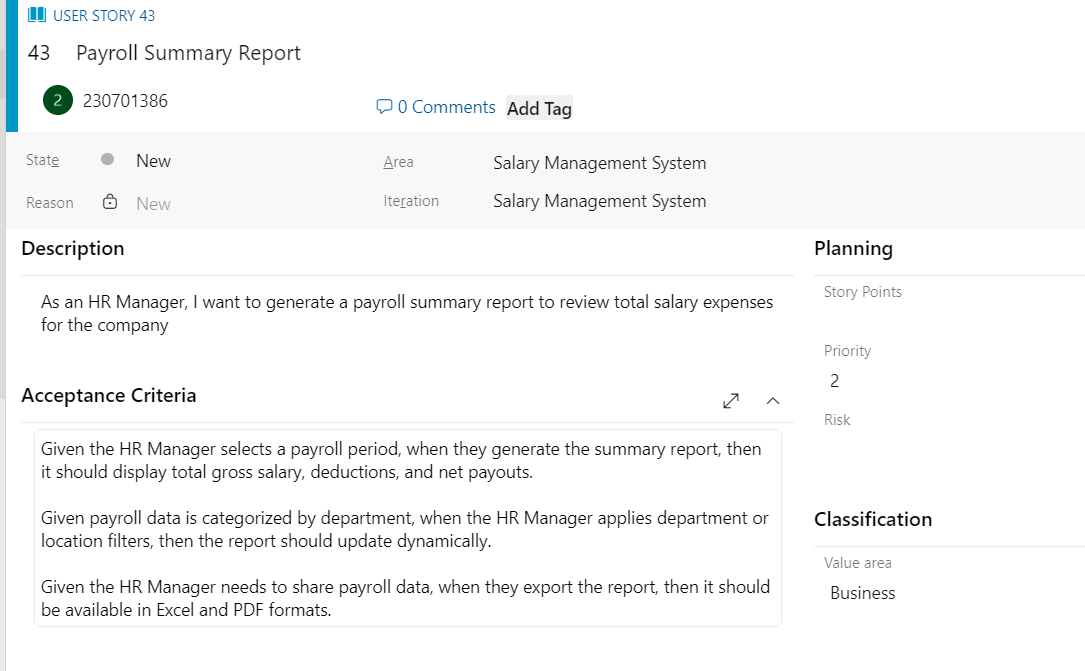
a. From the left-hand navigation menu, click on Boards. This will take you to the main Boards page, where you can manage work items, backlogs, and sprints.

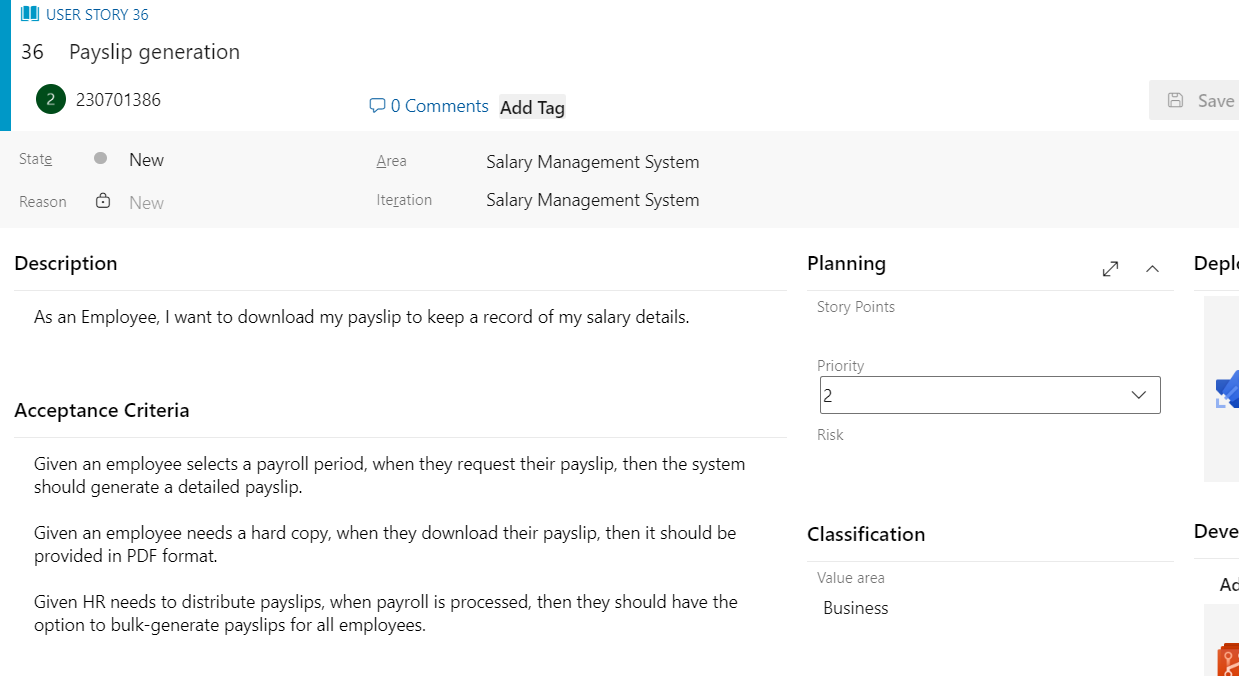
b. On the work items page, you'll see the option to Add a work item at the top. Alternatively, you can find a + button or Add New Work Item depending on the view you're in. From the Add a work item dropdown, select User Story. This will open a form to enter details for the new User Story.

10. Fill in the User Story.









**Epic: Payroll Management**

The **Payroll Management** epic focuses on automating and streamlining employee salary processing, ensuring compliance with tax regulations, and maintaining accurate financial records. It helps organizations manage payroll efficiently while reducing errors and manual workload.

This epic consists of two major components:

1. **Salary Calculation** – Computes pay based on hours worked, deductions, and bonuses.
2. **Tax Deductions** – Applies necessary tax and benefits deductions to salaries.
3. **Payment Processing** – Transfers salaries via direct deposit or other payment methods.
4. **Employee Records** – Stores employee details such as designation, salary structure, and tax details.

**Key Features & Their Functionalities :**

1. Salary Calculation & Processing

- Automatically calculates employee salaries based on working hours, deductions, bonuses, and tax regulations.

- Supports different pay structures, including hourly wages, fixed salaries, and commission-based payments.

- Ensures compliance with labor laws and tax policies.

2. Tax Deduction & Compliance Management

- Calculates and deducts applicable taxes (income tax, provident fund, social security, etc.).

- Generates tax reports for employees and government authorities.

- Ensures compliance with local and international tax regulations.

3. Payslip Generation

- Generates detailed payslips with earnings, deductions, and net salary breakdown.

- Allows employees to download payslips securely via a web portal or mobile app.

4. Employee Self-Service Portal

- Enables employees to view salary details, tax deductions, and payment history.

- Allows employees to update details and tax preferences.

- Provides access to downloadable payslips and tax documents.

5. Payroll Reporting & Analytics

- Generates payroll reports for financial planning and audits.

- Provides insights into salary trends, tax deductions, and employee compensation.

- Helps HR and finance teams make data-driven decisions.

**USER STORY 1: Payslip Generation**

*As an employee, I want to generate and download my payslip securely so that I can keep track of my salary details and deductions.*

**Acceptance Criteria:**

1. The system must allow employees to download their payslip in PDF format.
2. Payslips must include earnings, deductions, tax details, and net salary.
3. Employees should be able to access past payslips for at least the last 12 months.
4. Payslips must be password-protected to ensure data security.
5. The system must send automated notifications when a new payslip is generated.
6. Payslips should be accessible on both web and mobile platforms.
7. The payslip format should be clear and well-structured, ensuring easy readability.
8. Employees should be able to print their payslips without formatting issues.
9. HR managers must be able to bulk-generate payslips for multiple employees.
10. The system should allow employees to request a payslip correction if discrepancies are found.
11. Payslips must comply with local tax and labor regulations.
12. Employees should have an option to export their payslips in different formats (CSV, PDF).
13. Payslips should automatically update if salary adjustments occur.
14. Employees should be able to view the breakdown of their benefits in the payslip.
15. The system should ensure fast and error-free processing for payslip generation.

**USER STORY 2: Employee Profile Management**

*As an HR manager, I want to create and manage employee profiles so that I can store and access employee details efficiently*

**Acceptance Criteria:**

1. The system must allow HR personnel to add, edit, and delete employee profiles.
2. Employee profiles should include personal details, job designation, and department.
3. Each employee must be assigned a unique identification number.
4. The system should support document uploads, such as contracts and identification proof.
5. Employees should be able to update limited personal details (e.g., phone number, emergency contact).
6. HR personnel must have role-based access control to modify employee data.
7. The system should send notifications to HR and employees when profile changes occur.
8. Historical changes to employee profiles should be trackable for auditing.
9. The employee database must ensure data confidentiality and security.
10. Employees should be able to view their profiles via the self-service portal.

**User Story 3: Leave & Attendance Tracking**

**As an employee, I want to submit leave requests and track my attendance so that I can manage my work schedule effectively.**

**Acceptance Criteria:**

1. Employees must be able to apply for leave via the system.
2. Leave requests should be sent for manager approval before finalization.
3. Approved leave requests should reflect in the attendance system automatically.
4. Employees must be able to view their leave balance and history.
5. The system should track working hours and absences in real-time.
6. The system must integrate with biometric or time-tracking devices.
7. Attendance reports should be available to HR and employees.
8. Employees should receive notifications on leave approval or rejection.
9. The system should automatically deduct absences from salary calculations.
10. HR personnel should have access to department-wide attendance reports.

**User Story 4: Payroll Processing & Salary Calculation**

As a payroll administrator, I want to process employee salaries efficiently so that payments are accurate and timely.

**Acceptance Criteria:**

1. Salaries must be calculated automatically based on attendance, deductions, and benefits.
2. Tax deductions should comply with local labor laws and government policies.
3. Employees should receive their salaries via direct deposit.
4. Payroll calculations must integrate with leave tracking for accurate salary adjustments.
5. Employees should receive notifications upon salary deposit.
6. The system should generate payroll reports for auditing and financial tracking.
7. Payslips should be accessible via the employee self-service portal.
8. HR and payroll admins should be able to manually override salary calculations.
9. Employees should be able to view salary breakdowns, including bonuses and deductions.
10. The payroll system must support multi-currency processing for international employees.
11. Payroll processing must be error-free and scalable for organizations with large teams.
12. The system should generate automatic payroll schedules for recurring payments.
13. Employees should have access to tax documents for filings and compliance.
14. Payroll administrators should receive alerts for inconsistencies or errors before finalizing payments.
15. Salary adjustments due to promotions or deductions should reflect immediately in payroll records.

**User Story 5: Payslip Generation & Download**

As an employee, I want to securely download my payslip so that I can keep a record of my salary details.

**Acceptance Criteria:**

1. Payslips must be available for download in PDF format.
2. Payslips should include earnings, deductions, net salary, and tax breakdown.
3. The system should allow employees to access past payslips for at least 12 months.
4. Payslips must be password-protected to ensure security.
5. Employees should receive automated notifications when a new payslip is available.
6. Payslips must comply with legal payroll and tax regulations.
7. Employees should be able to print their payslips without formatting issues.
8. Payroll admins should have the ability to bulk-generate payslips.
9. The system should allow employees to request corrections for discrepancies.
10. Payslip formatting should be optimized for both web and mobile access.

**USER STORY 6: Salary Calculation**

*As a payroll administrator, I want to automate salary calculations so that I can ensure accurate and timely payments for employees.*

**Acceptance Criteria:**

1. The system must calculate salaries based on employee work hours, leaves, and overtime.
2. It should automatically deduct applicable taxes, provident funds, and benefits.
3. The system must support different salary structures (fixed, hourly, commission-based).
4. Overtime pay should be computed accurately based on company policies.
5. The salary calculation should integrate with leave and attendance management systems.
6. Employees should be able to view a detailed breakdown of their salary components.
7. Salary adjustments due to promotions, deductions, or bonuses should be reflected in real-time.
8. The system must generate error-free salary slips based on payroll data.
9. HR should be able to override calculations and manually adjust salaries if required.
10. Tax computations should comply with local, national, and international tax laws.
11. Payroll administrators should receive notifications for any discrepancies in salary processing.
12. Employees should be able to raise queries regarding payroll calculations.
13. The system should enable multi-currency salary processing for global teams.
14. Salary payments should be automated through direct deposits into employee accounts.
15. The system must generate real-time payroll reports for auditing and compliance tracking.

**RESULT:**

The assigned user story for my project has been 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.

::: mermaid

sequenceDiagram

participant Employee

participant Admin

participant HR

participant Payroll System

participant database

participant Tax department

Employee ->> Admin : login request

Tax department ->> Payroll System : sends new tax rules

Admin -->> Employee : login access

Admin ->> Payroll System : initiates salary calculation

Payroll System ->> Tax department : handle taxes

HR ->> Payroll System : updates salary, employee details

Tax department -->> Payroll System : return tax info

Payroll System -->> HR : confirms updates

Payroll System -->> Admin : notifies payroll completion

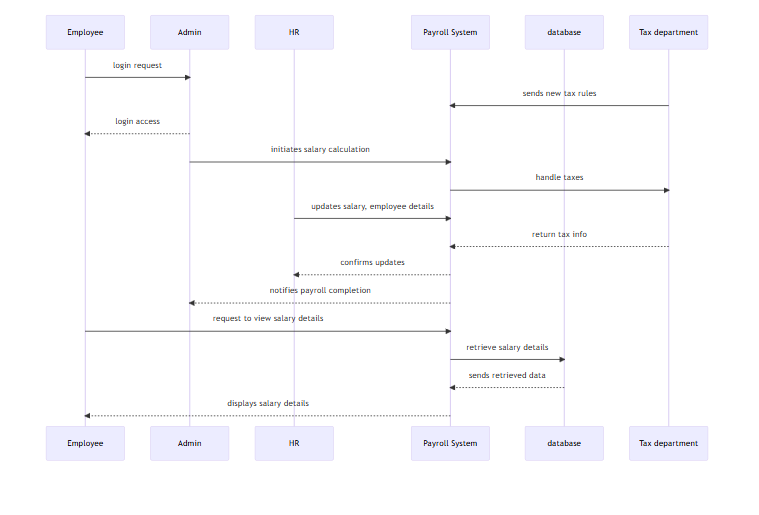
Employee ->> Payroll System : request to view salary details

Payroll System ->> database : retrieve salary details

database -->> Payroll System : sends retrieved data

Payroll System -->> Employee : displays salary details

4. Click wiki menu and select the page.

****

**RESULT:**

The sequence diagram is drawn successfully.

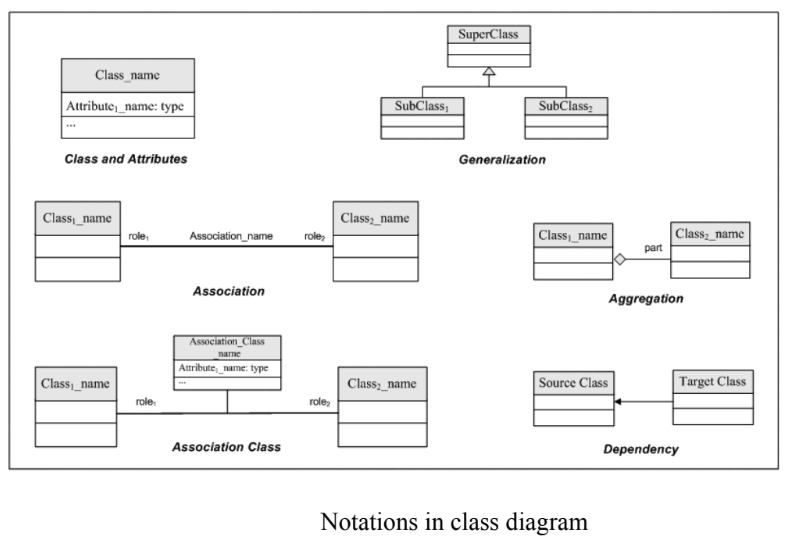
**EX NO: 6 CLASS DIAGRAM**

**AIM:**

To draw a simple class diagram.

**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 the code for drawing Class Diagram and save the code.

:::mermaid

classDiagram

class Employee {

-name : String

-employeeID : String

-designation : String

-department : String

-salaryDetails : String

-email : String

-phoneNumber : String

+login()

+requestSalaryDetails()

+updateContactInfo(email: String, phone: String)

}

class Admin {

-adminID : String

-name : String

-email : String

-role : String

+manageAccess()

+processPayroll()

+generateReports()

}

class HR {

-hrID : String

-name : String

-email : String

-department : String

+updateEmployeeDetails()

+manageSalaryStructure()

+manageLeaveRequests()

}

class TaxDepartment {

-taxID : String

-name : String

-rules : String

-taxPercentage : float

+submitTaxRules()

+validateTaxCalculations()

+calculateNetSalary(gross: float)

}

class PayrollSystem {

- systemID : String

- loginCredentials : Map<String, String>

- employeeData : List<Employee>

- payrollHistory : Map<String, List<Salary>>

+ calculateSalaries()

+ updateRecords()

+ authenticateUsers(username: String, password: String)

}

class Database {

-databaseID : String

-employeeRecords : List

-salaryRecords : List

-taxRules : List

+storeData()

+retrieveData()

+updateData()

}

%% Each employee is associated with one HR (e.g., for leave/tax/salary queries)

Employee "1" --> "0..1" HR

%% Payroll system handles all employees

Employee "0..\*" --> "1" PayrollSystem

%% Admin manages employees through the payroll system

Admin "1" --> "1" PayrollSystem

%% One HR handles many employees

HR "1" --> "0..\*" Employee

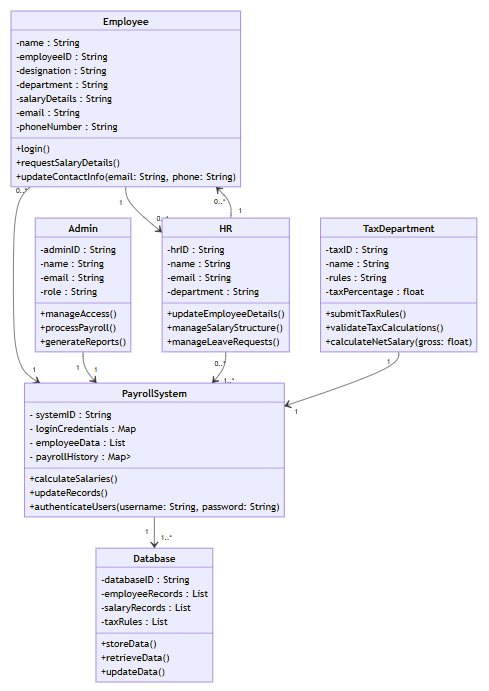
%% HR interacts with PayrollSystem for multiple operations

HR "0..\*" --> "1..\*" PayrollSystem

PayrollSystem "1" --> "1..\*" Database

%% TaxDepartment supplies tax rules to the PayrollSystem

TaxDepartment "1" --> "1" PayrollSystem



**RESULT:**

Thus the class diagram has been designed successfully.

**EX NO: 7 USE CASE DIAGRAM**

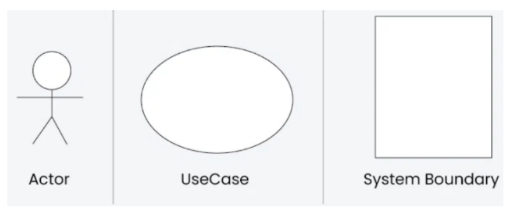
**AIM:**

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



**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.

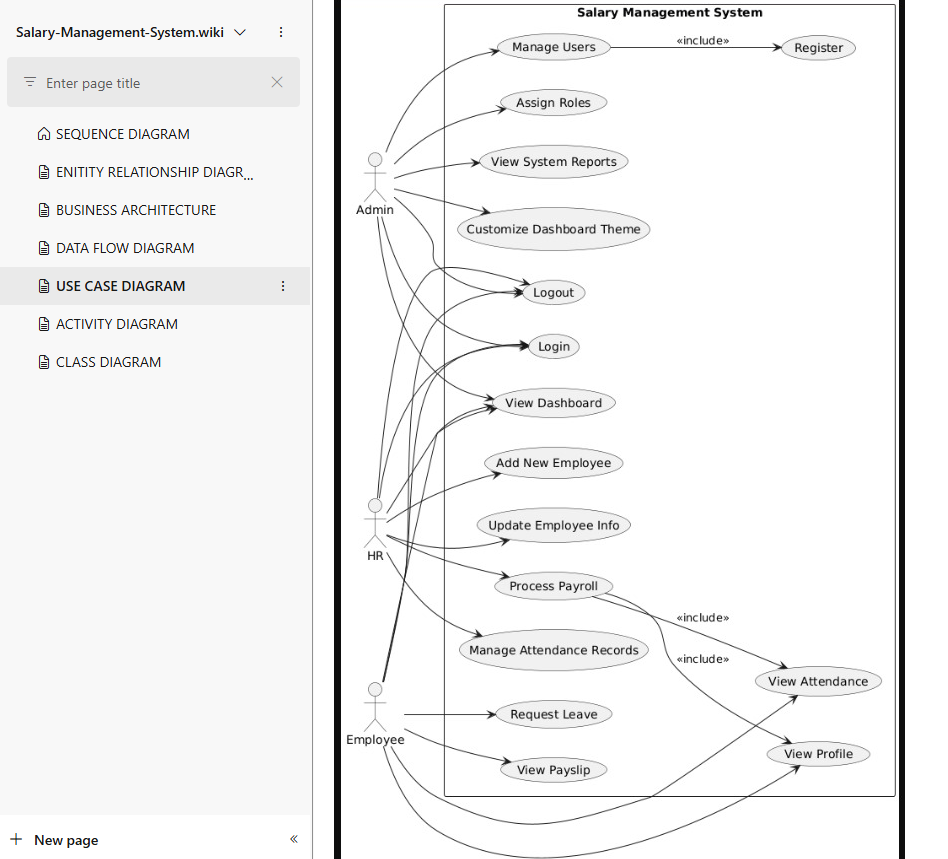
Step 2: Upload the Diagram to Azure DevOps

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.



**RESULT**:

The use case diagram was designed successfully.

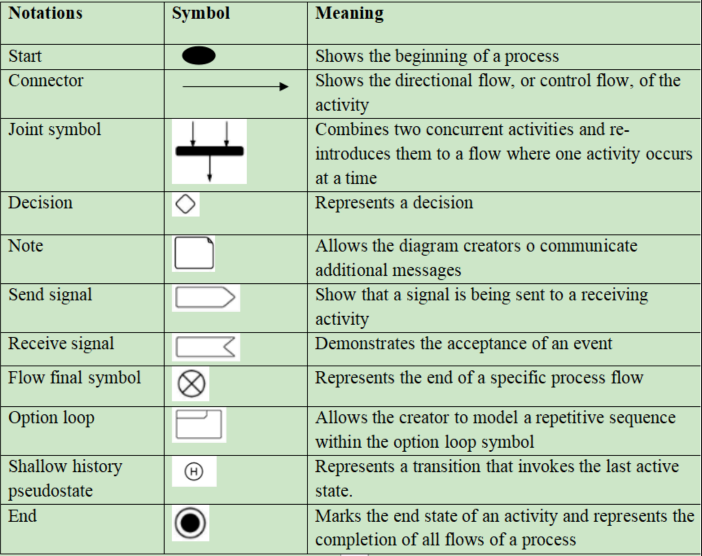
**EX NO: 8 ACTIVITY DIAGRAM**

**AIM :**

To draw a sample activity diagram for the salary management Application.

**THEORY:**

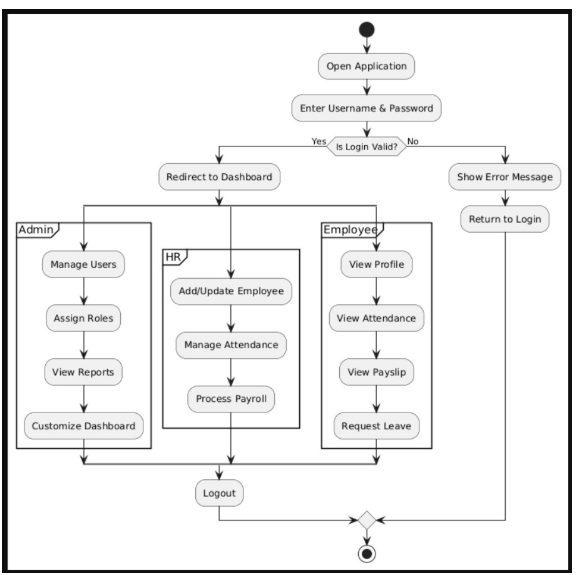
Activity diagrams are an essential part of the Unified Modelling Language (UML) that help visualize workflows, processes, or activities within a system. They depict how different actions are connected and how a system moves from one state to another.



**PROCEDURE:**

Step 1. Draw diagram in draw.io.

Step 2. Upload the diagram in Azure DevOps wiki.

****

**RESULT:**

The activity diagram was designed successfully.

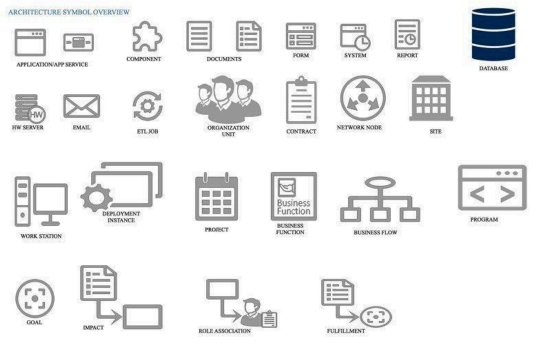
**EX NO: 9 ARCHITECTURE DIAGRAM**

**AIM:**

To draw the Architecture Diagram using draw.io.

**THEORY:**

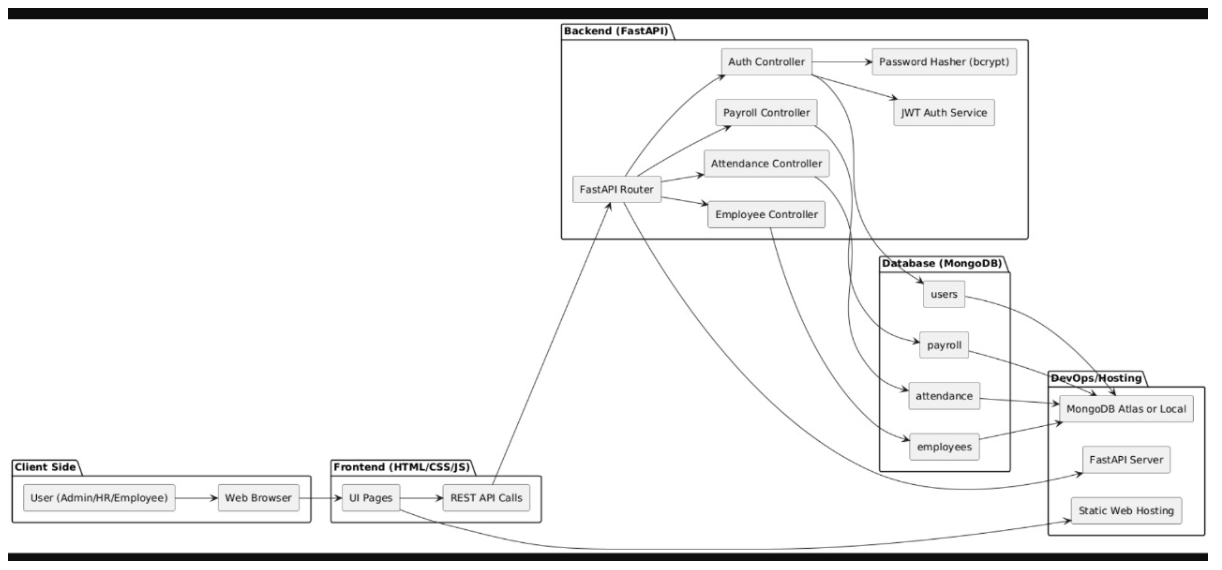
An architectural diagram is a visual representation that maps out the physical implementation for components of a software system. It shows the general structure of the software system and the associations, limitations, and boundaries between each element.



**PROCEDURE:**

Step 1. Draw diagram in draw.io

Step 2. Upload the diagram in Azure DevOps wiki.



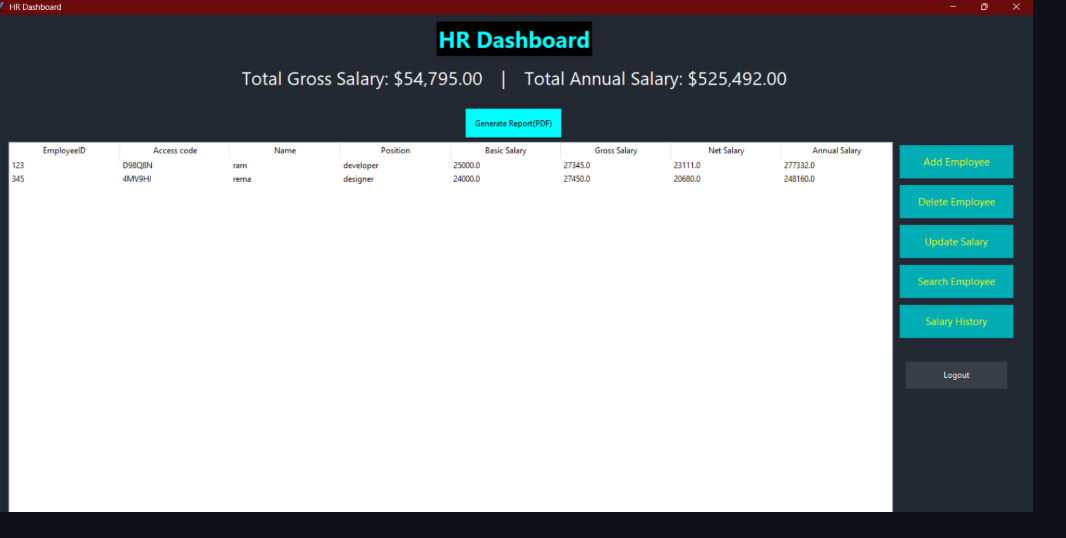
**RESULT:**  
The architecture diagram was designed successfully

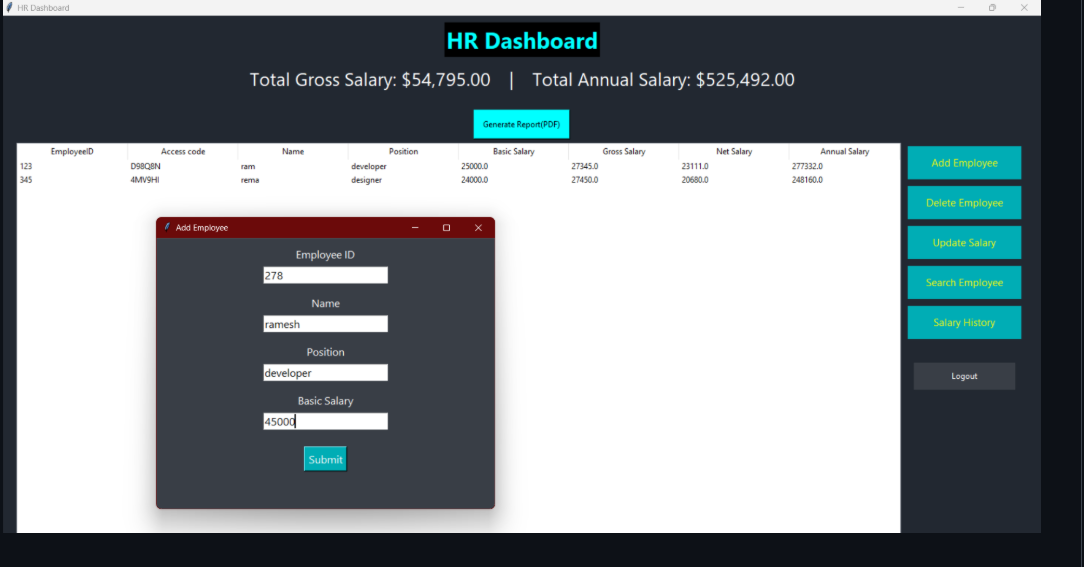
**EX NO: 10 USER INTERFACE**

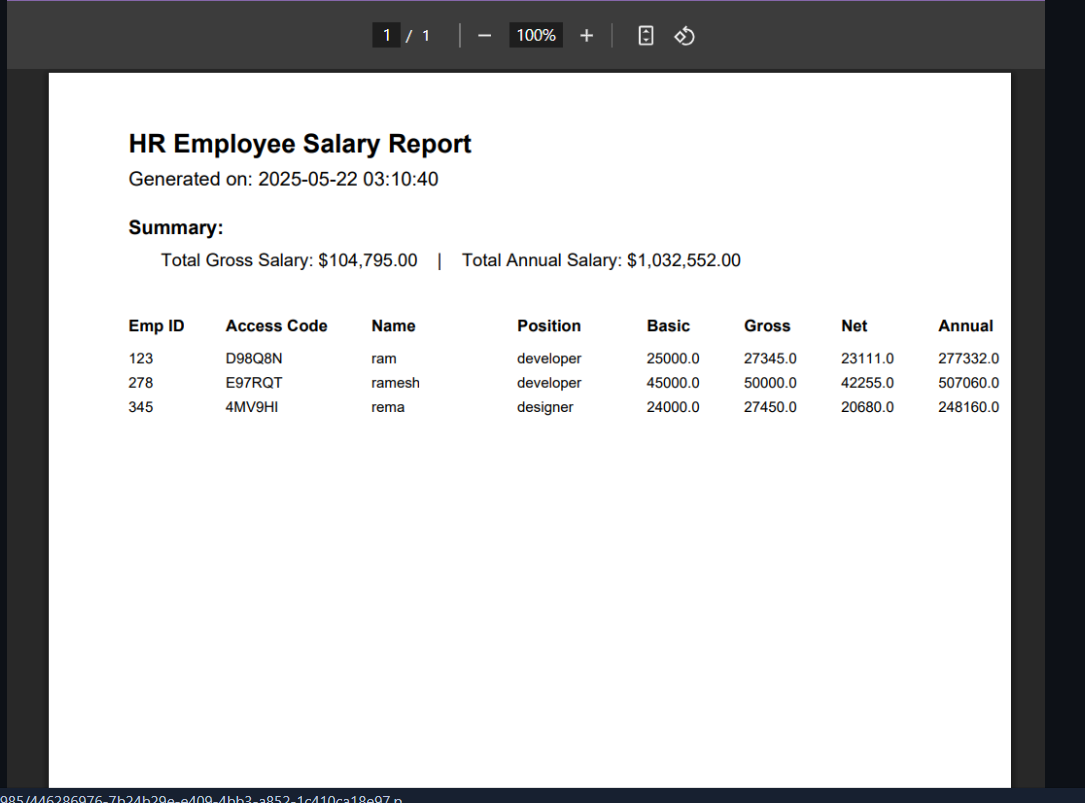
**AIM:**

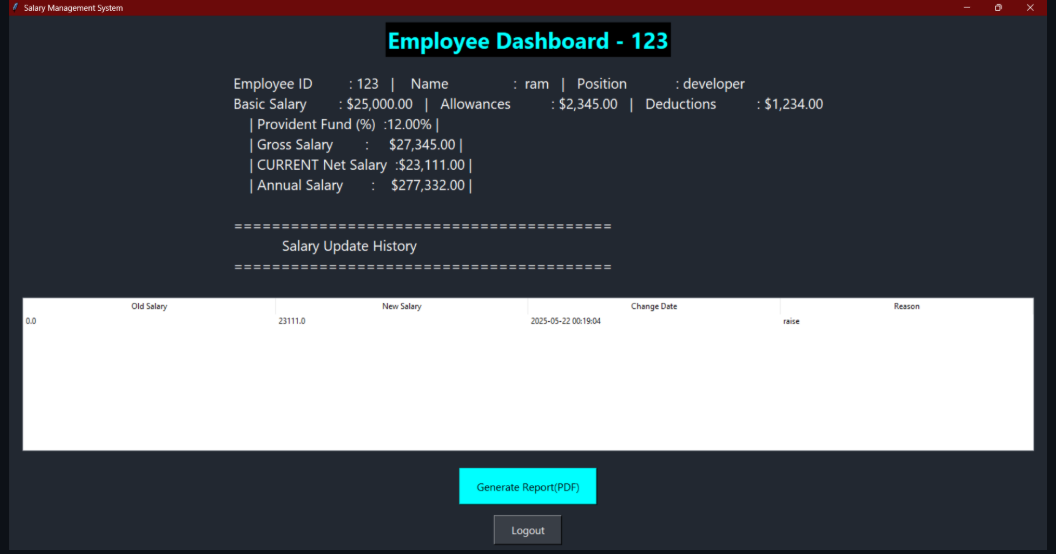
Design User Interface for the salary management system.

**UI DESIGNS OF WEATHER APP:**









**RESULT :**

The UI was Designed successfully.

**EX NO: 11 IMPLEMENTATION**

**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>

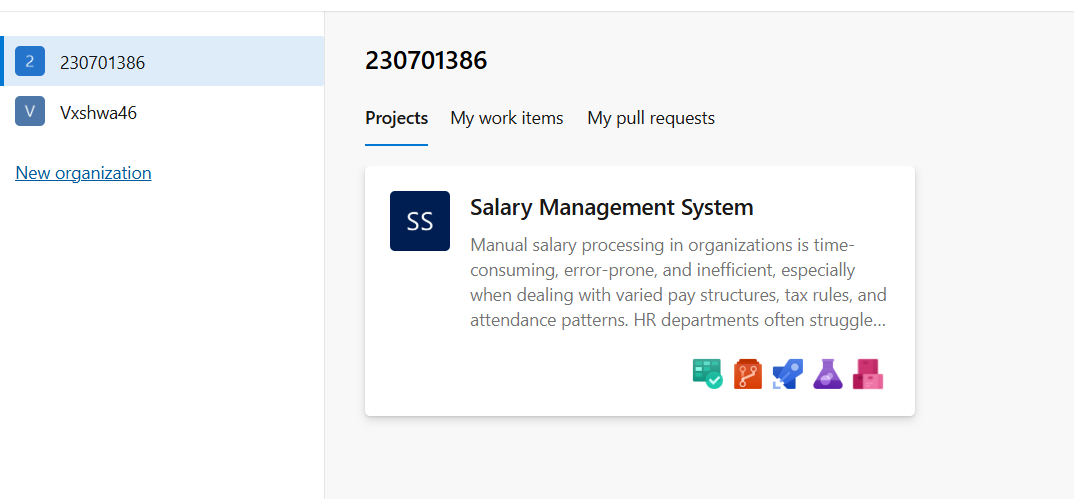
* Add web application code (HTML, CSS, JavaScript, React, Angular, or backend like Node.js, .NET, Python, etc.).
* Commit & push:

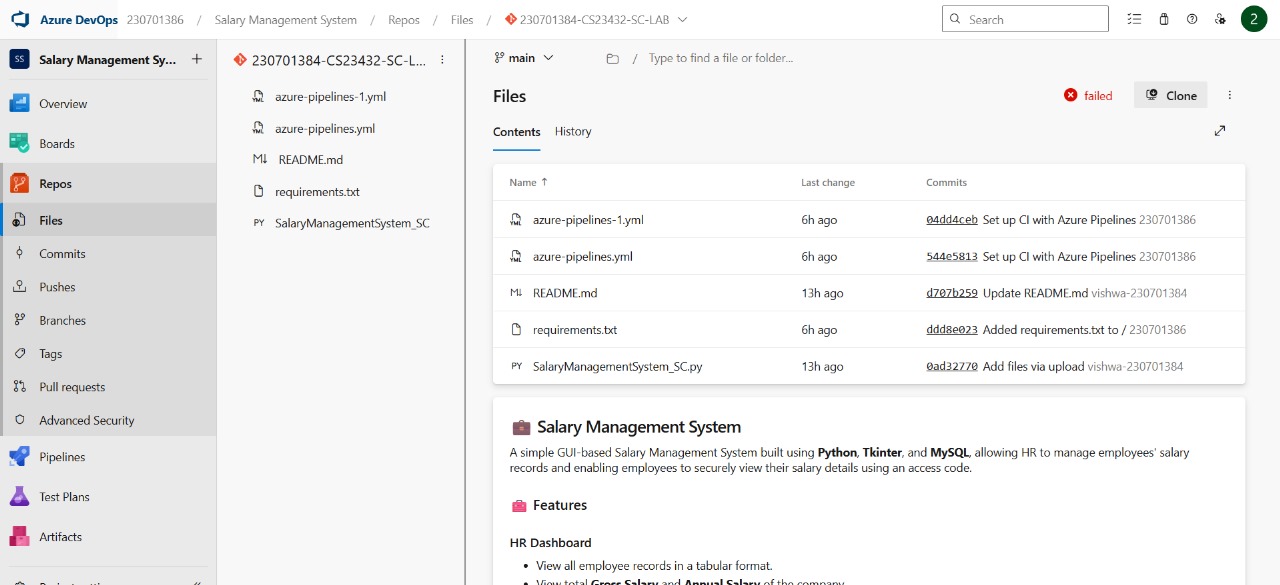
git add .

git commit -m "Initial commit"

git push origin main

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





**RESULT :**

Thus the application was successfully implemented.

**EX NO: 12 TESTING USING AZURE**

**AIM:** To perform testing of the Salary Management App using Azure DevOps Test Plans, specifically validating the functionality of downloading employee details as a PDF and ensuring that the feature meets the acceptance criteria under Agile methodology.

**PROCEDURE:**

**Step 1: Open Azure DevOps Project**

* Log in to your Azure DevOps account.
* Navigate to the Salary Management App project.

**Step 2: Navigate to Test Plans**

* In the left menu, click on Test Plans.
* Click on “New Test Plan” and name it (e.g., Employee Management PDF Export).

**Step 3: Create Test Suites**

Create a new suite for each feature/module being tested.

Suite 1: Add Employee

Suite 2: Update Salary

Suite 3: Delete Employee

Suite 4: Download Employee Details as PDF

**Step 4: Add Test Cases**

Inside Suite 4, click “+ New Test Case” and fill in the details as follows:

**Test Case ID:** TC004

**Title:** Download Employee Details PDF with Header Info

**Scenario:** Test if the application successfully generates a downloadable PDF containing all employee details along with total gross salary and total annual salary.

**Steps:**

1. Launch the Salary Management App.

2. Log in as HR.

3. Navigate to the HR Dashboard.

4. Click the "Download PDF" button.

5. In the popup, select a valid folder location and filename.

6. Confirm the file is downloaded.

7. Open the downloaded PDF.

8. Verify the presence of:

* Employee table with correct data
* Total gross salary
* Total annual salary

**Expected Result:**

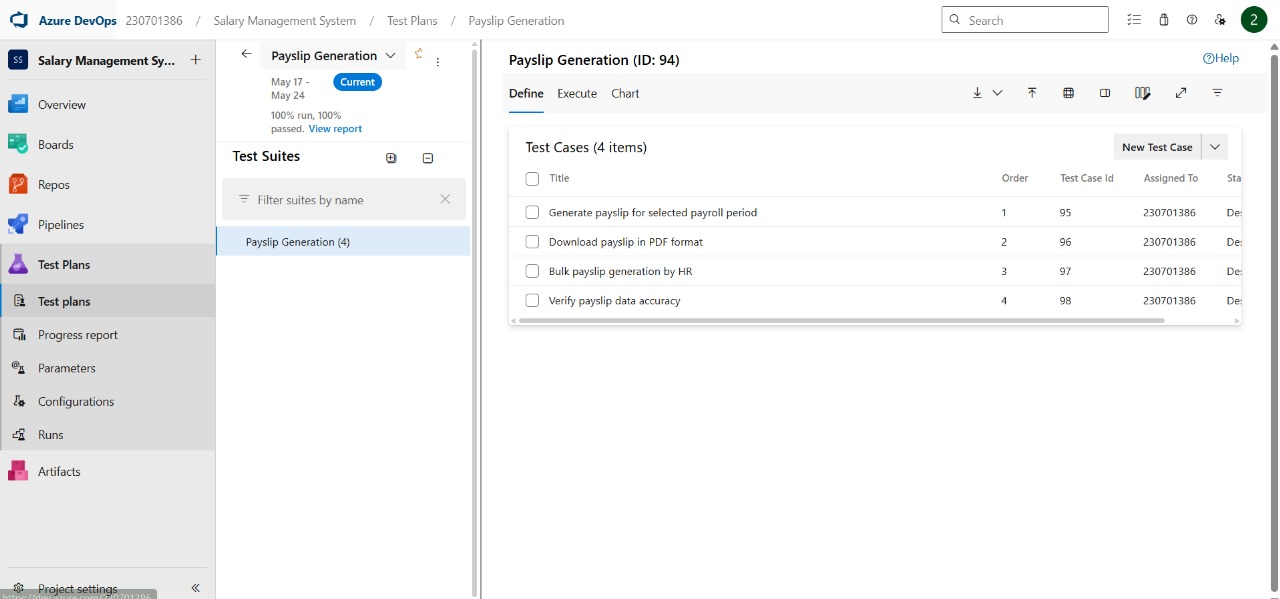
A PDF file is generated successfully, containing complete employee details and the correct totals, formatted properly and readable.

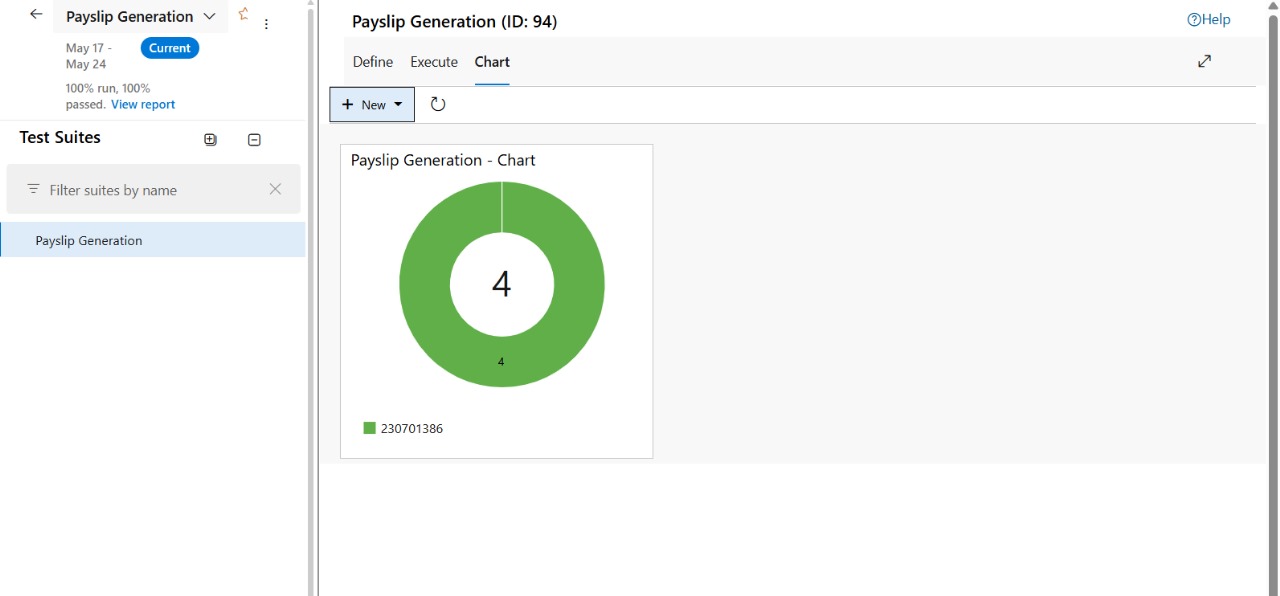
**Step 5: Execute Test Cases**

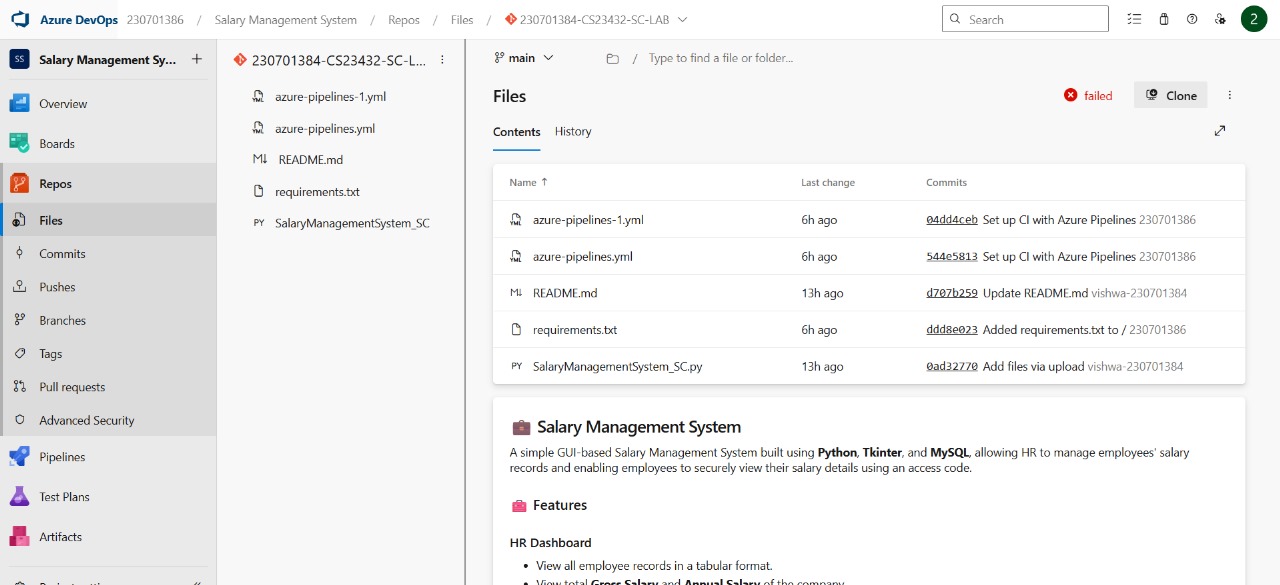
* Open each test case and click “Run for web application.”
* Perform the steps manually or with automation support.
* Mark the result as Pass or Fail.
* Provide Actual Result and Remarks if needed.

**Step 6: Track and Report**

* Navigate to Test Plans → Charts to view real-time progress.
* Use filters to track:
* Number of test cases passed/failed
* Percentage of coverage for related user stories
* Linked bugs or issues discovered during testing







**RESULT:**

Thus the application was successfully tested in Azure.

**EX NO: 13 CI/CD PIPELINE**

**AIM:**

To implement a Continuous Integration and Continuous Deployment (CI/CD) pipeline for the salary management app using Azure DevOps, ensuring automated build, test, and deployment of the application.

**PROCEDURE:**

Step 1: Create a Build Pipeline (CI)

* Go to Pipelines → Create Pipeline.
* Select Azure Repos Git → Choose your repository.
* Choose Starter pipeline or YAML file.
* Add pipeline tasks like:

trigger:

- main

pool:

name: Default

steps:

- task: UseNode@2

inputs:

version: '18.x'

- script: npm install

displayName: 'Install Dependencies'

- script: npm run build

displayName: 'Build Application'

- script: npm run test

displayName: 'Run Tests'

* Save and Run the pipeline to verify.

Step 4: Set Up Release Pipeline (CD)

* Navigate to Pipelines → Releases → New pipeline.
* Add an Artifact (your build pipeline output).
* Add Stages like:
  + Development
  + Production
* Configure Deploy tasks in each stage:
  + For web apps: Use Azure Web App Deploy task.
  + For mobile: Use relevant deployment tools.

Step 5: Add Approvals and Gates (Optional)

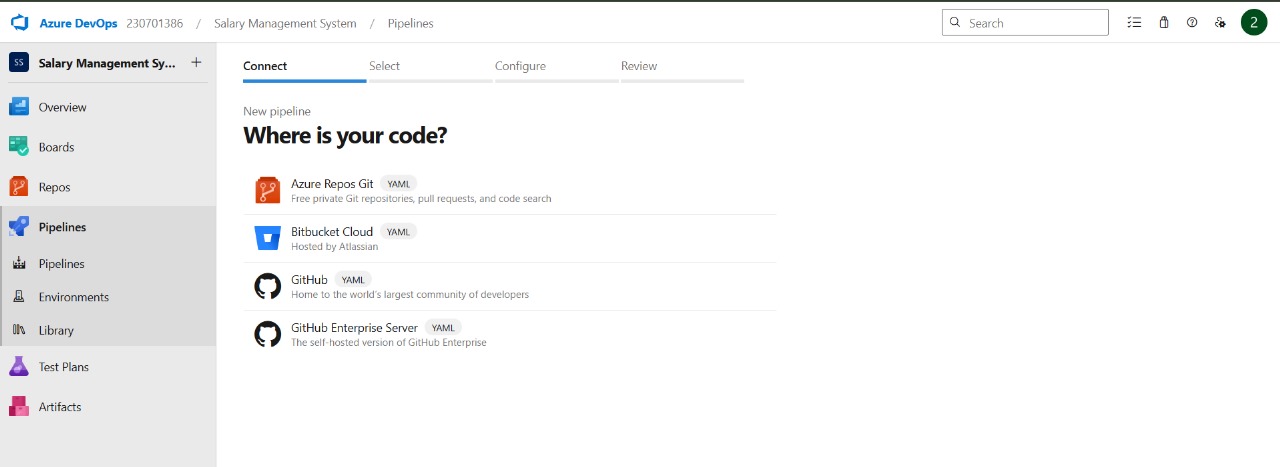
* Add pre-deployment approvals to each stage for review.
* Add gates like API health checks or test validations.

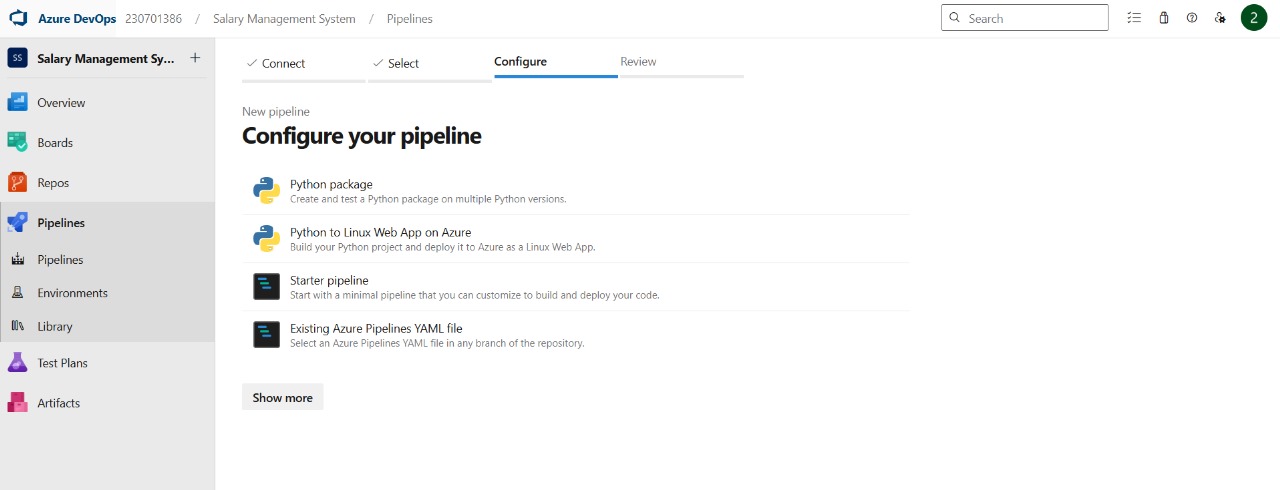
Step 6: Automate Triggering

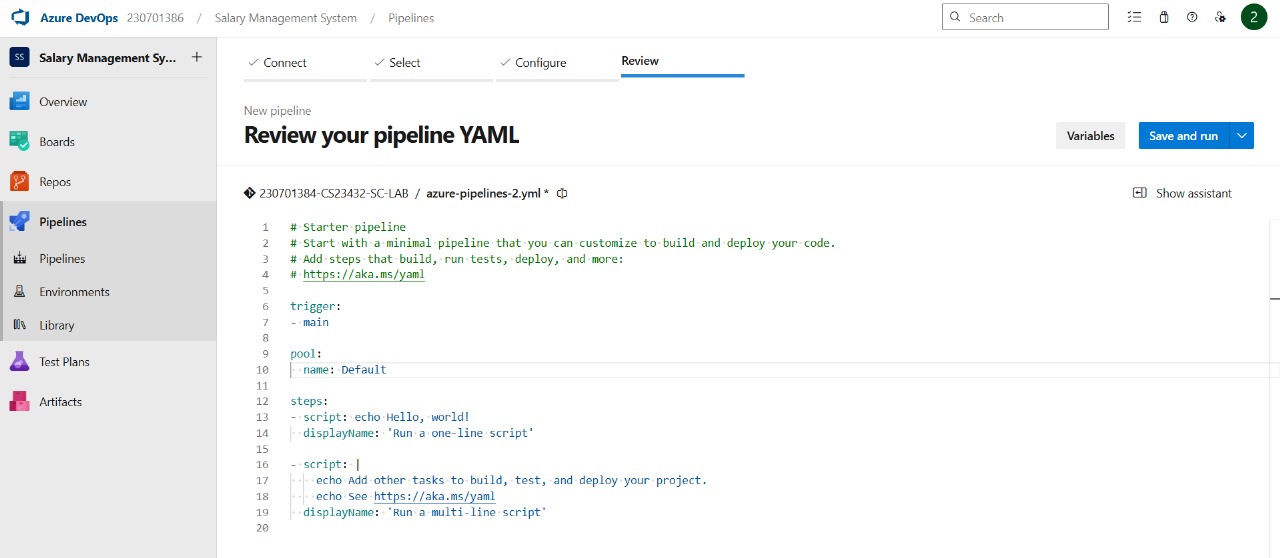
* Ensure the pipeline triggers:
  + On code push to main branch (CI)
  + After successful build for deployment (CD)

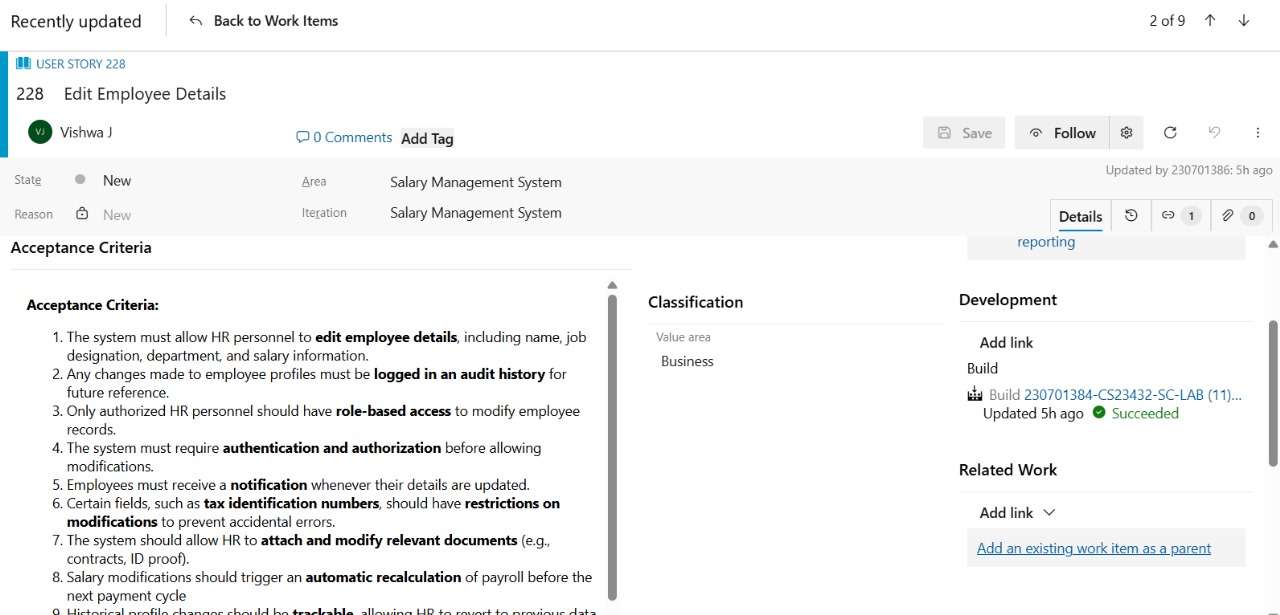
Step 7: Monitor Pipeline

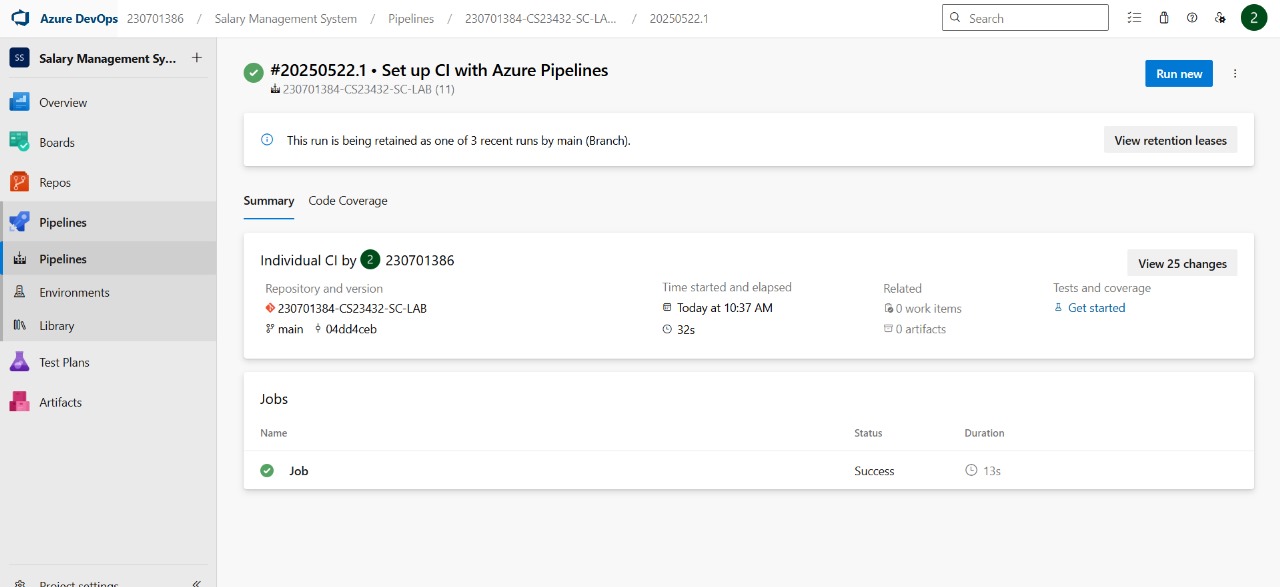
* Track pipeline status under Pipelines → Runs.
* Debug failures and download logs if necessary.
* Use Azure Boards to link builds with user stories and bugs.











**RESULT:**Thus the CI/CD pipeline has been successfully implemented.