RAJALAKSHMI ENGINEERING COLLEGE
RAJALAKSHMI NAGAR, THANDALAM – 602 105

RAJALAKSHMI
ENGINEERING COLLEGE

CS23432 SOFTWARE CONSTRUCTION LABORATORY

Laboratory Note Book

Name: Maiyesha N

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

University Register No. :2116231501093

College Roll No: 231501093

Semester: 4rd SEMESTER

Academic Year: 2024-2025

RAJALAKSHMI ENGINEERING COLLEGE

[AUTONOMOUS]

RAJALAKSHMI NAGAR, THANDALAM – 602 105

BONAFIDE CERTIFICATE

Name : . . . MAIYESHA N. . . . . .

Academic Year : . . . . 2024-2025. . . . . Semester : 04 Branch : AIML

Register No. **2116231501093**

Certified that this is the bonafide record of work done by the above student in the

CS23432 – SOFTWARE CONSTRUCTION during the year 2024 - 2025.

Signature of Faculty in-charge

Submitted for the Practical Examination held on . . . . . .

Internal Examiner **External Examiner**

EX NO: 1 STUDY OF AZURE DEVOPS

AIM:

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

STUDY:

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

1. Understanding Azure DevOps

Azure DevOps consists of five key services:

1.1 Azure Repos (Version Control)

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

1.2 Azure Pipelines (CI/CD)

- Automates build, test, and deployment processes.
- Supports multi-platform builds (Windows, Linux, macOS).
- Works with Docker, Kubernetes, Terraform, and cloud providers (Azure, AWS, GCP). 1.3 Azure Boards (Agile Project Management)
- Manages work using Kanban boards, Scrum boards, and dashboards.
- Tracks user stories, tasks, bugs, sprints, and releases.

1.4 Azure Test Plans (Testing)

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

1.5 Azure Artifacts (Package Management)

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

Getting Started with Azure DevOps:

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

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

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

- Choose Git or TFVC for version control.
- Clone the repository and push your code.

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

- Select a source code repository (Azure Repos, GitHub, etc.).
- Define the pipeline using YAML or the Classic Editor.

• Run the pipeline to build and deploy the application.

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

• Create work items, user stories, and tasks.

• Organize sprints and track progress.

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

• View test results and track bugs.

RESULT:

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

EX NO: 2 WRITING PROBLEM STATEMENT

AIM:

To prepare PROBLEM STATEMENT for your given project.

PROBLEM STATEMENT:

Online Banking System

- **Account Management**:
  - Users should be able to view their account balances, transaction history, and account details.
  - The system must support fund transfers between accounts, both internally and externally.
- **Bill Payment and Transfers**:
  - Users should be able to pay bills, set up recurring payments, and schedule future transactions.
  - The system must allow users to transfer funds to other accounts within the same bank or to external accounts securely.
- **Alerts and Notifications**:
  - Users should receive real-time alerts for transactions, low balances, and account activities.
  - The system must send notifications for bill due dates, account updates, and security alerts.

RESULT:

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

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

AIM:

To plan a agile model for the given problem statement.

THEORY:

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

With Agile planning, a project is broken down into smaller, more manageable tasks with the 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 designing project using agile-scrum methodology by using azure was completed successfully.

EX NO: 4 AGILE PLANNING

AIM:

SCOPE:

AGILE EPICS & USER STORIES:

Epics:

Epics represent large bodies of work that can be divided into smaller user stories. They generally span across multiple sprints.

Epic 1: Account Management

Objective: Allow users to view and manage their account details, balance, and transaction history.

User Stories:

As a user, I want to view my account balance.

As a user, I want to view my transaction history.

As a user, I want to view my account details (personal information, account type, etc.).

Epic 2: Fund Transfers

Objective: Enable users to transfer funds between their own accounts and externally.

User Stories:

As a user, I want to transfer funds between my accounts within the same bank.

As a user, I want to transfer funds to an external account.

As a user, I want to securely authenticate fund transfers.

Epic 3: Bill Payment and Recurring Transactions

Objective: Allow users to pay bills, set up recurring payments, and schedule future transactions.

User Stories:

As a user, I want to pay my bills through the online system.

As a user, I want to set up recurring bill payments.

As a user, I want to schedule future payments.

Epic 4: Alerts and Notifications

Objective: Enable real-time alerts for transactions, low balances, and other important activities

User Stories:

As a user, I want to receive alerts for transaction activities.

As a user, I want to receive notifications when my balance is low.

As a user, I want to get notified about bill due dates.

As a user, I want to receive security alerts (e.g., suspicious activity).

Epic 5: Security and Compliance

Objective: Ensure that the system is secure and compliant with financial regulations.

User Stories:

As a user, I want to authenticate securely (e.g., multi-factor authentication).

As a user, I want my sensitive data to be encrypted and protected.

As a developer, I need to ensure that the system complies with financial regulations (e.g., PCI DSS, GDPR).

Sprints:

Now, let’s break down the work into Sprints with focus areas for each sprint.

Sprint 1: Basic Account Management

Duration: 2 weeks

Focus: Basic account-related functionalities.

Epics Covered:

Account Management

User Stories:

As a user, I want to view my account balance.

As a user, I want to view my transaction history.

As a user, I want to view my account details.

Sprint 2: Internal Fund Transfers

Duration: 2 weeks

Focus: Enable internal fund transfers between accounts.

Epics Covered:

Fund Transfers

User Stories:

As a user, I want to transfer funds between my own accounts

As a user, I want to see a confirmation of my transfer.

As a user, I want to see a history of my internal transfers.

Sprint 3: Bill Payments and Recurring Transactions

Duration: 2 weeks

Focus: Bill payments, recurring transactions, and scheduling.

Epics Covered:

Bill Payment and Recurring Transactions

User Stories:

As a user, I want to pay my bills online.

As a user, I want to set up recurring bill payments.

As a user, I want to schedule a payment for a future date.

Sprint 4: External Fund Transfers

Duration: 2 weeks

Focus: Enable fund transfers to external accounts.

Epics Covered:

Fund Transfers

User Stories:

As a user, I want to transfer funds to an external account.

As a user, I want to securely authenticate external transfers.

As a user, I want to see transfer status and confirmation.

Sprint 5: Alerts and Notifications

Duration: 2 weeks

Focus: Implement real-time alerts and notifications for account activities.

Epics Covered:

Alerts and Notifications

User Stories:

As a user, I want to receive alerts for transactions.

As a user, I want to receive notifications when my balance is low.

As a user, I want to receive reminders about bill due dates.

Sprint 6: Security and Compliance

Duration: 2 weeks

Focus: Ensure that the system is secure and meets compliance requirements.

Epics Covered:

Security and Compliance

User Stories:

As a user, I want to authenticate securely (multi-factor authentication).

As a user, I want my sensitive data to be encrypted.

As a developer, I want to ensure the system meets PCI-DSS and GDPR regulations**.**

Sprint 7: Final Testing and Deployment

Duration: 1 week

Focus: Perform end-to-end testing and deploy the application.

Epics Covered:

All Epics (Account Management, Fund Transfers, Bill Payments, Alerts, and Security)

User Stories:

End-to-end testing for all functionalities.

Load testing for performance.

User acceptance testing (UAT) and deployment to production.

RESULT:

Thus, the agile plan for the problem statement is completed successfully.

EX NO: 5 USER STORIES – CREATION

AIM:

To create User Stories for the given problem statement.

THEORY:

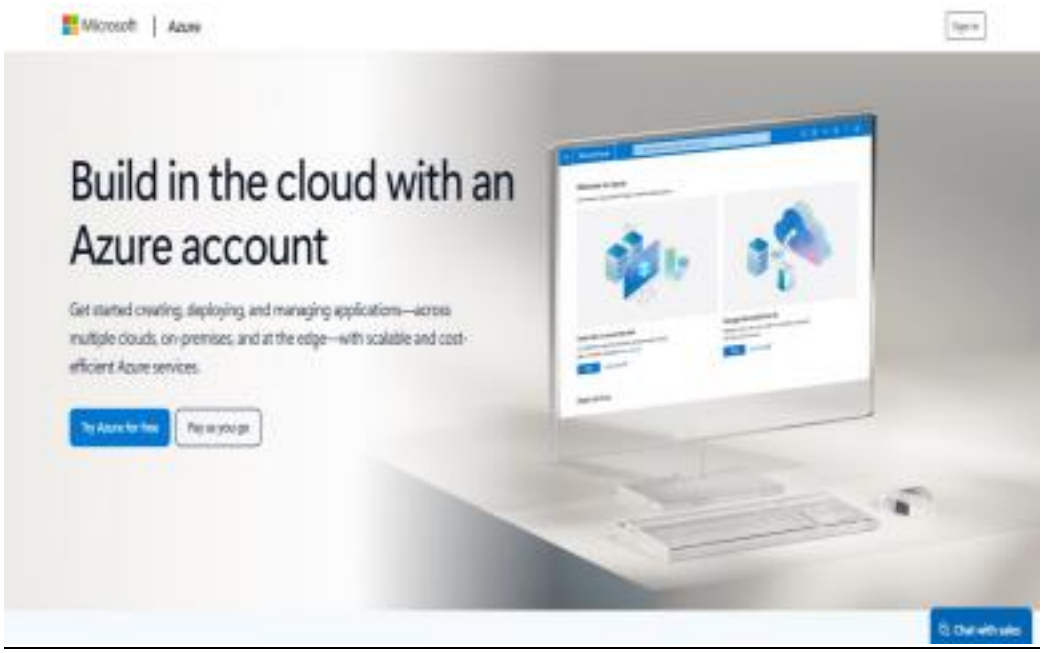
A user story is an informal, general explanation of a software feature written from the perspective of the end user. Its purpose is to articulate how a software feature will provide value to the customer.

User story template

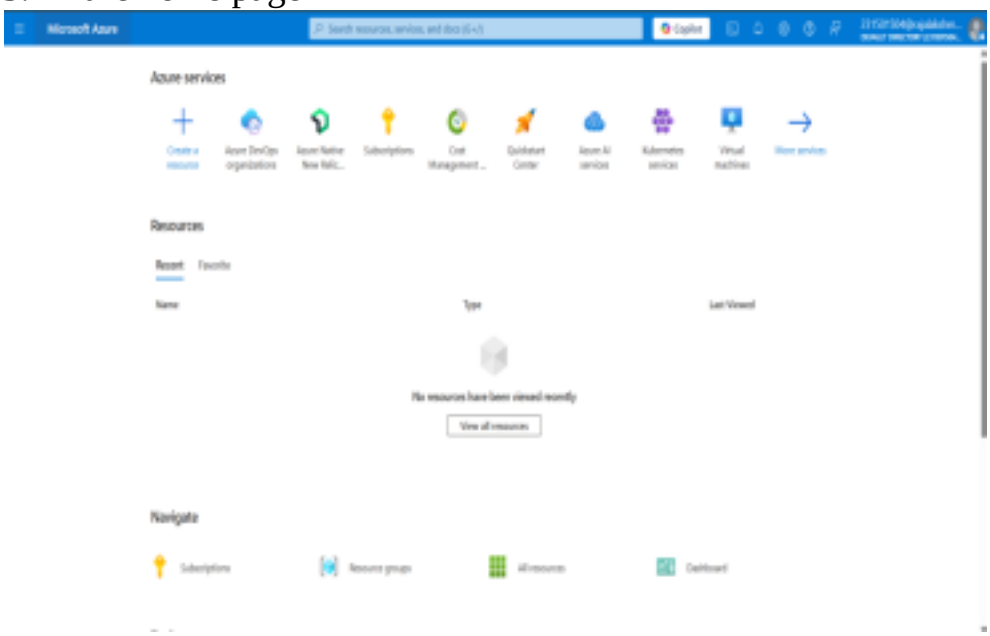
"As a [role], I [want to], [so that]."

PROCEDURE:

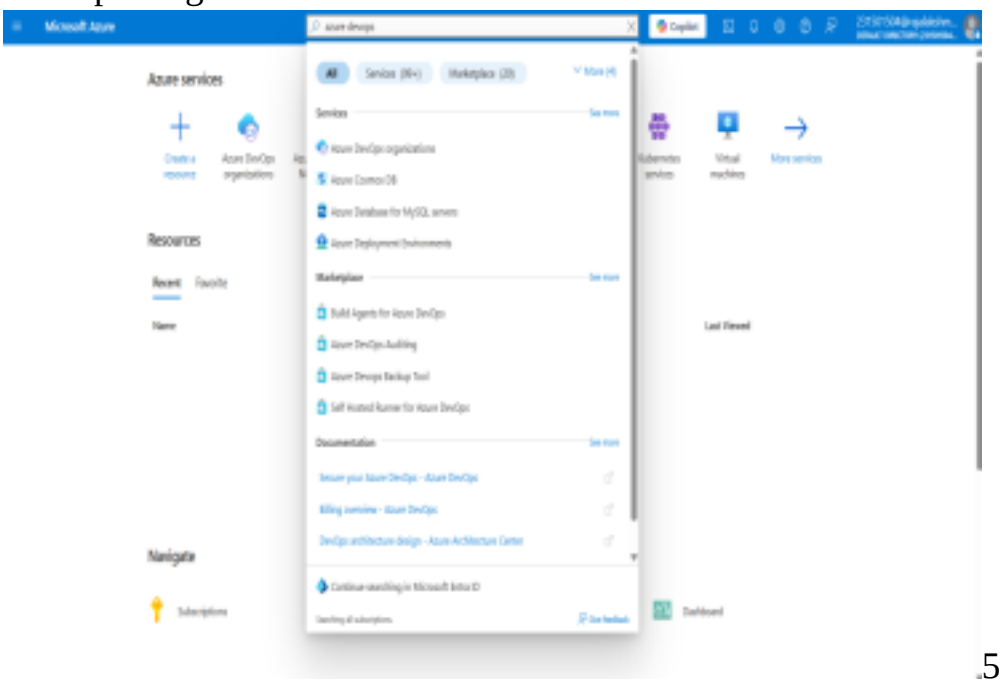
1. Open your web browser and go to the Azure website: *https://azure.microsoft.com/en-in* Sign in using your Microsoft account credentials. If you don't have an account, you'll need to create one.
2. If you don’t have a Microsoft account, you can sign up for *https://signup.live.com/?lic=1*



3. Azure home page

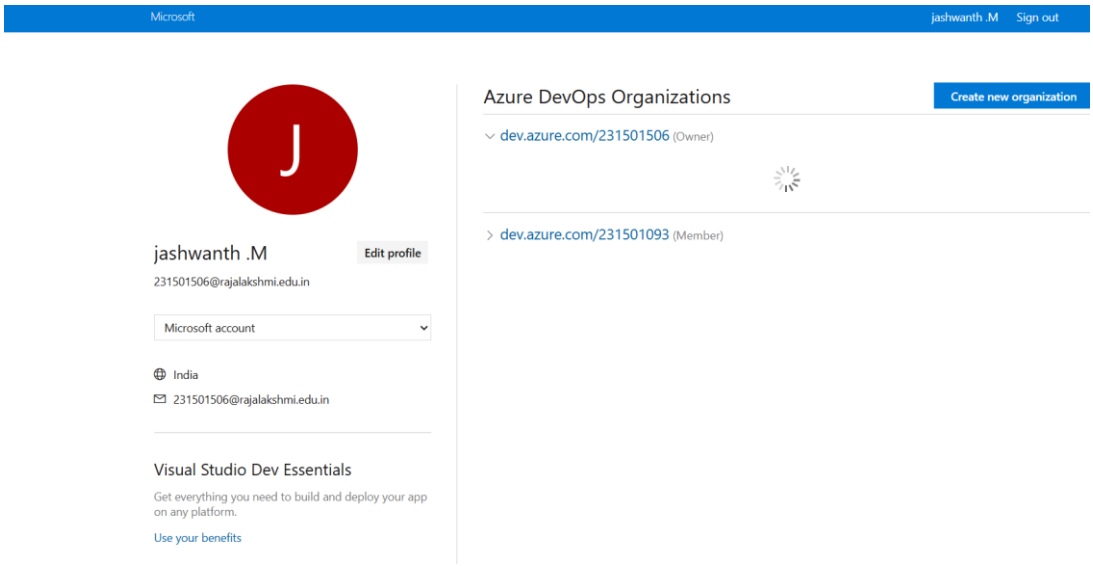


4. Open DevOps environment in the Azure platform by typing Azure DevOps Organizations in the search bar.



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.

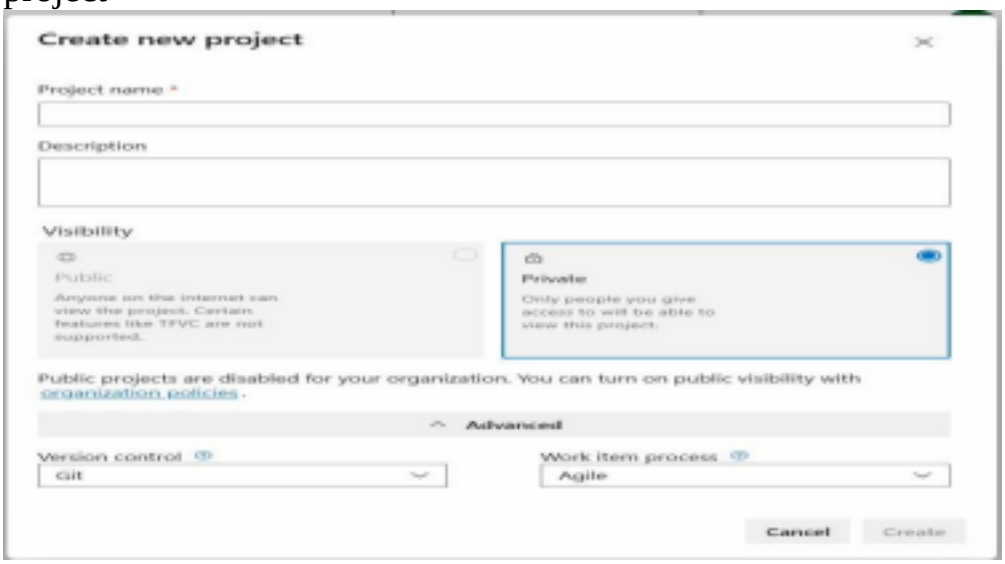
ii. Enter the project name, description, and visibility options:

○ **Name**: Choose a name for the project (e.g., LMS).

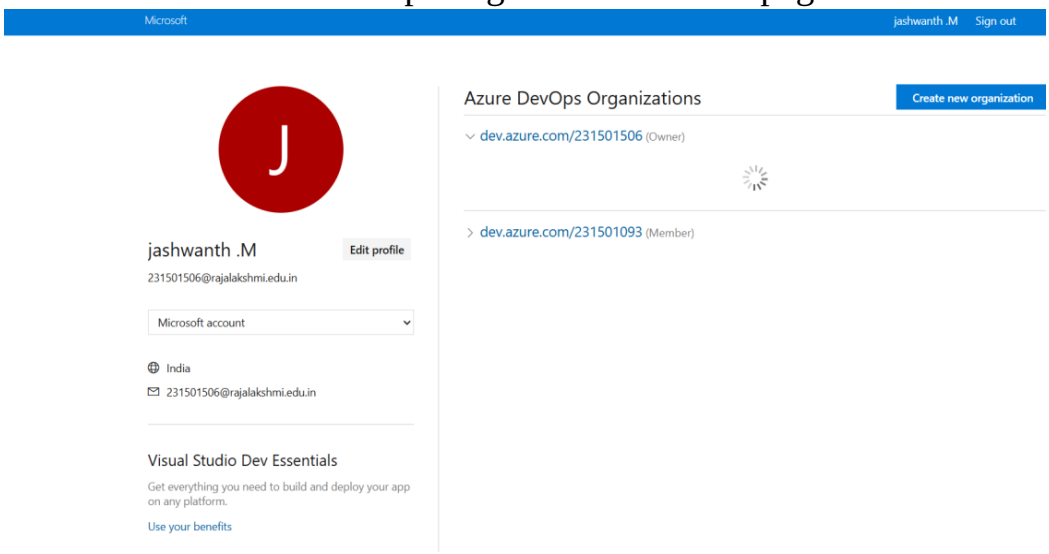
○ **Description**: Optionally, add a description to provide more context about the project. ○ **Visibility**: Choose whether you want the project to be **Private**

(accessible only to those invited) or **Public** (accessible to anyone).

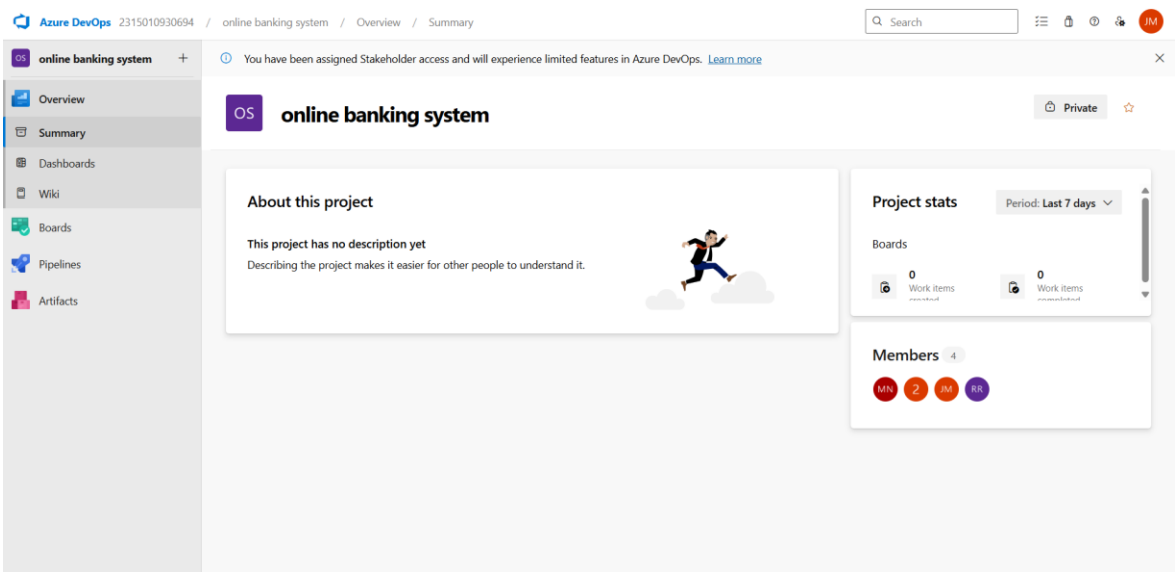
Once you've filled out the details, click **Create** to set up your first project



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

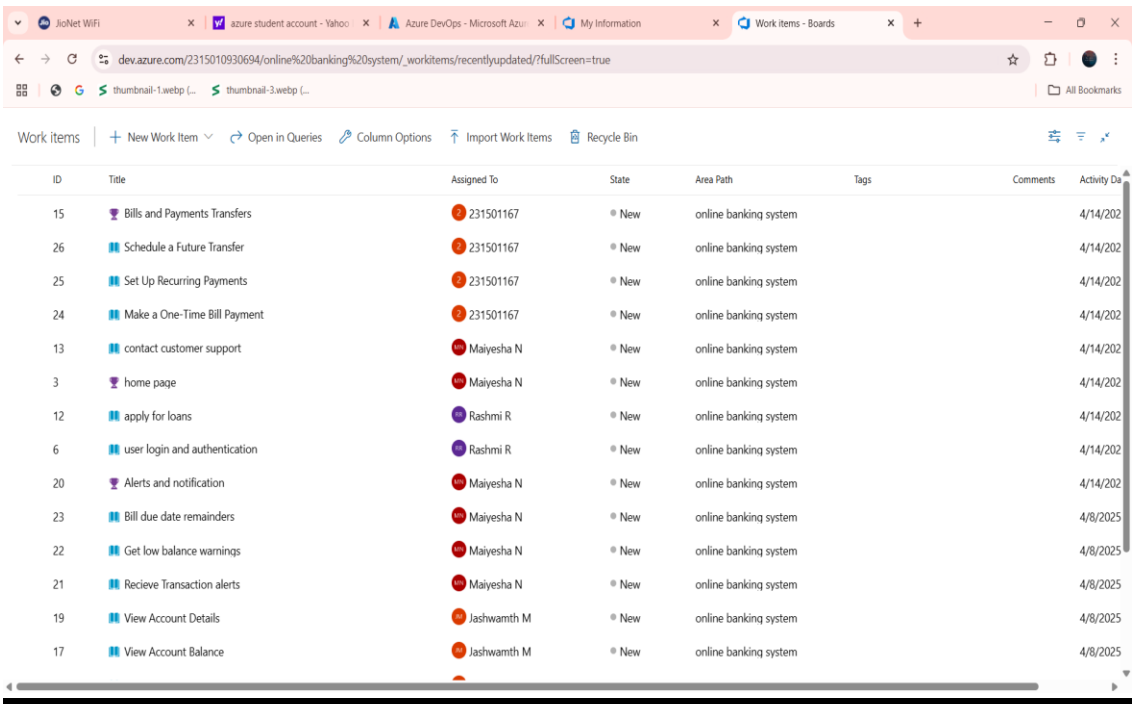


8. Project dashboard

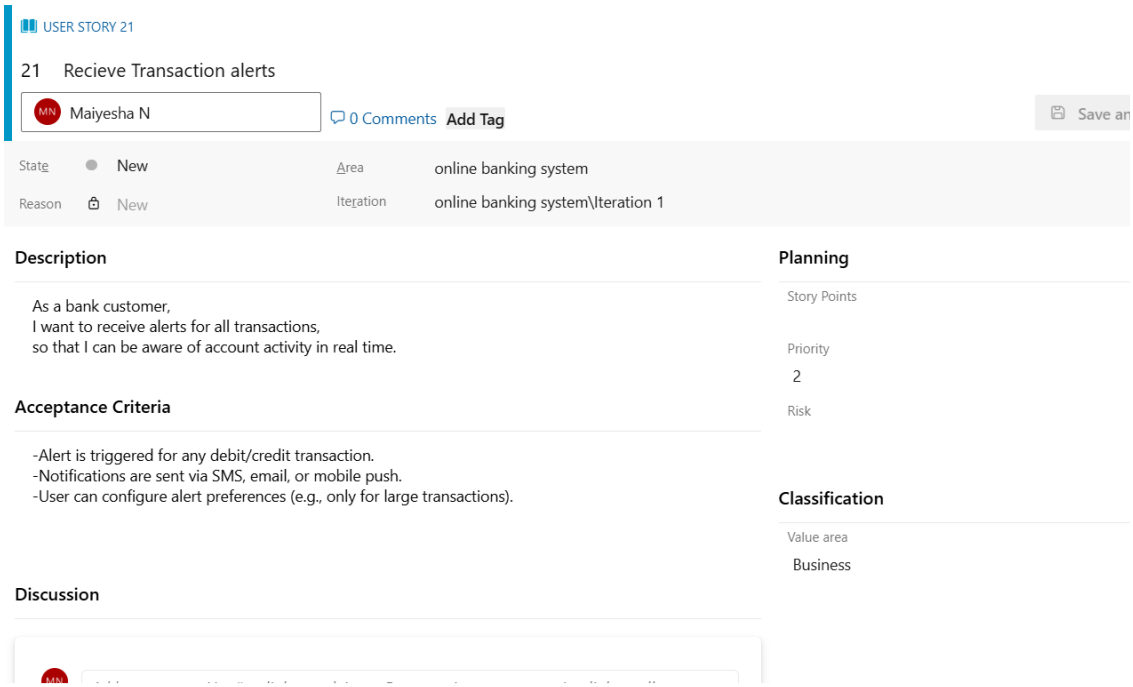


9 . To manage user stories

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



10. Fill in User Story Details



Result:

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

EX NO: 6 SEQUENCE DIAGRAM

AIM:

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

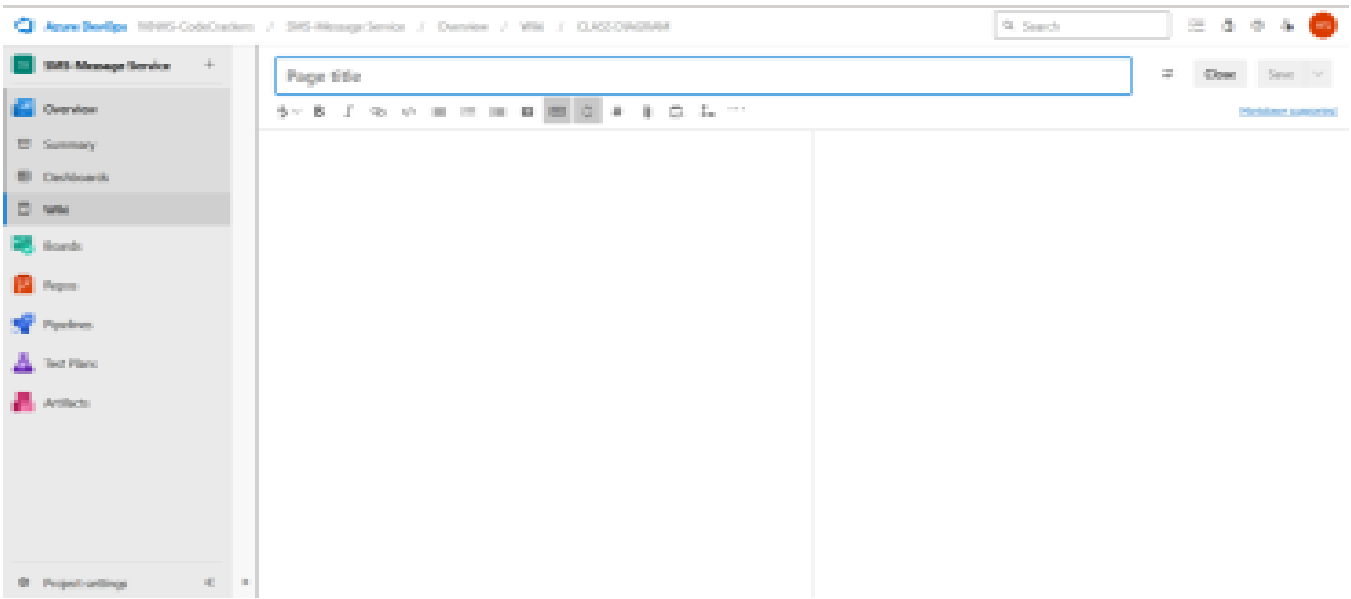
THEORY:

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

:::mermaid

sequenceDiagram

participant U as User

participant UI as User Interface

participant A as Account Service

participant T as Transaction Service

```
participant N as Notification Service
U->>UI: Login
UI->>A: Authenticate User
A->>UI: Authentication Response
U->>UI: View Account Overview
UI->>A: Fetch Account Details
A->>UI: Return Account Overview
U->>UI: Select Transfer Option
UI->>U: Enter Transfer Details (Amount, Recipient)
U->>UI: Submit Transfer Request
UI->>A: Validate Transfer Details
A->>UI: Return Validation Status (Valid/Invalid)
alt Valid Transfer
    UI->>A: Check Balance
    A->>UI: Return Balance Status (Sufficient/Insufficient)
    alt Sufficient Funds
        UI->>T: Initiate Transfer (Amount, Recipient)
        T->>A: Deduct Funds from Sender Account
        A->>T: Confirm Deduction
        T->>A: Add Funds to Receiver Account
        A->>T: Confirm Deposit
        T->>N: Send Transfer Confirmation Notification
        N->>U: Notify Transfer Success
        UI->>U: Show Transfer Confirmation
    else Insufficient Funds
```

UI->>U: Show Insufficient Funds Error

end

else Invalid Transfer Details

UI->>U: Show Invalid Transfer Error

end

EXPLANATION:

Bank Transfer Flow

1. **User logs in.**
   The system checks if the username and password are correct.

2. **User views account.**
   The system shows account details like balance and recent transactions.

3. **User chooses to transfer money.**
   The user enters the amount and the receiver's details.

4. **System checks the details.**
   It verifies if the information is valid (like correct account number, amount not zero, etc.).

5. **If the details are valid:**

   The system checks if the user has enough balance.

   If the balance is enough:

   It deducts the money from the sender's account.

   It adds the money to the receiver's account.

   It sends a success notification to the user.

   The user sees a "Transfer Successful" message.

   If the balance is not enough:

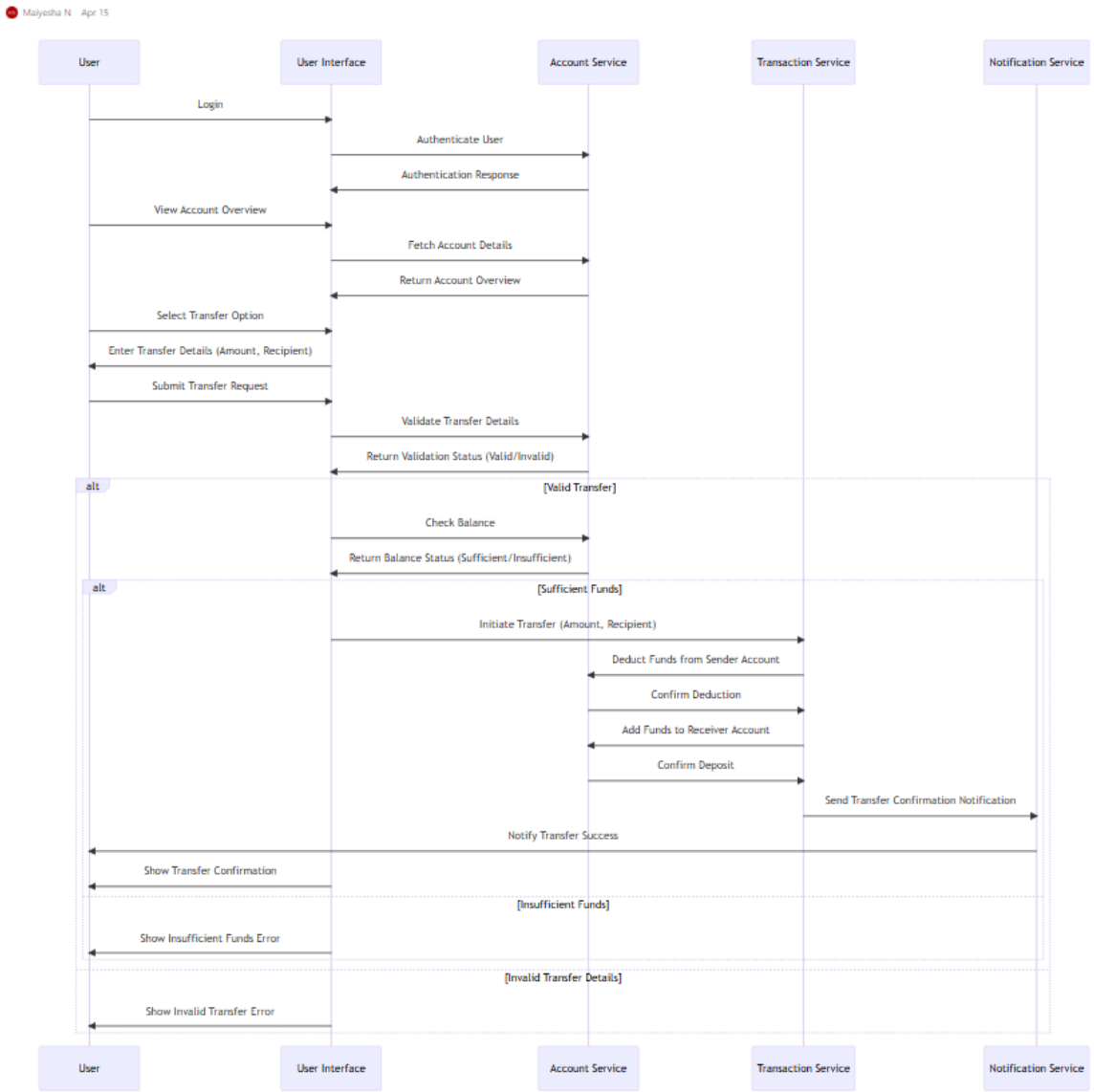
   The user sees an error saying "Insufficient Balance".

6. **If the details are invalid:**

The user sees an error saying "Invalid transfer details".

4. click wiki menu and select the page

sequence diagram



RESULT:

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

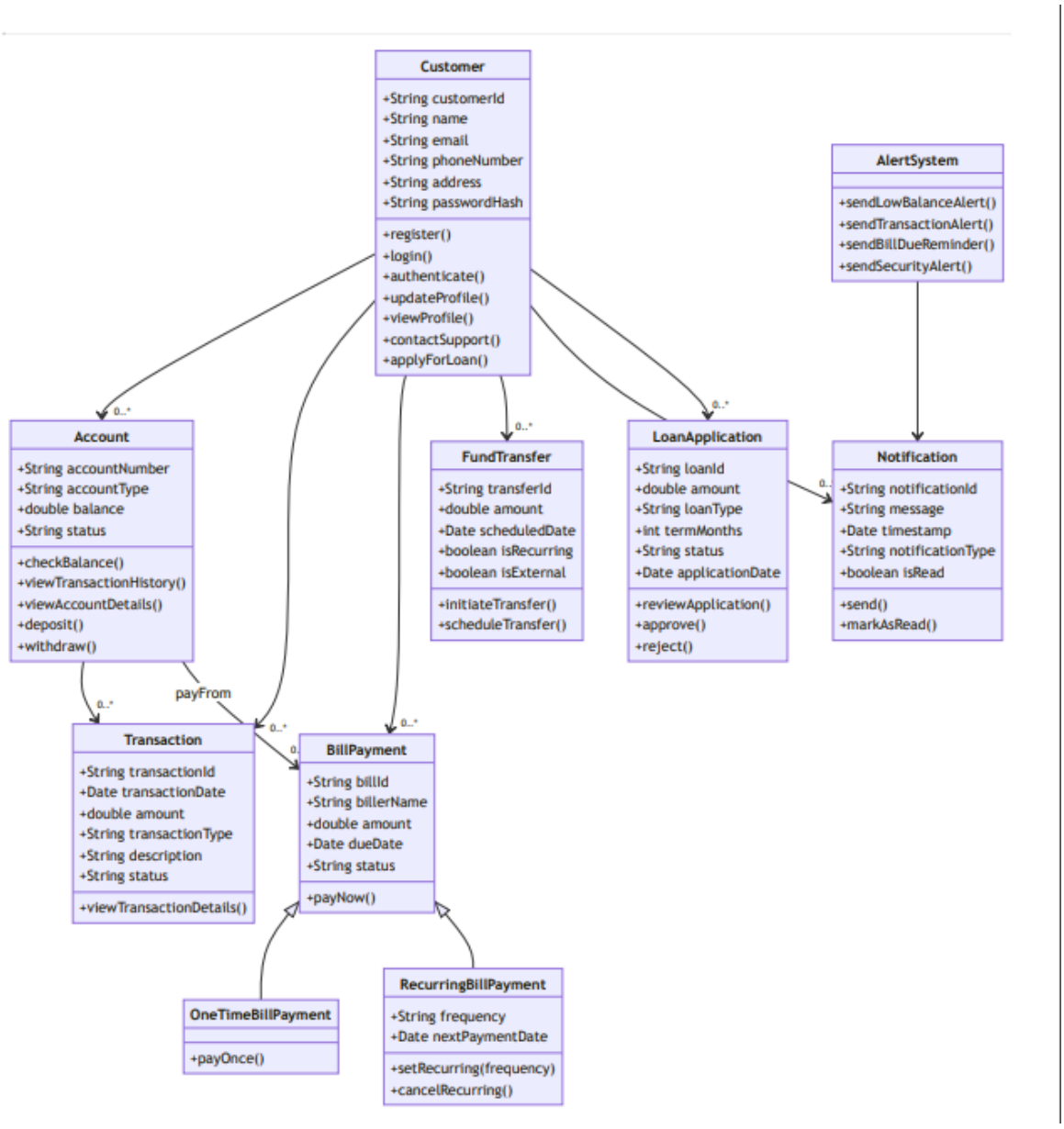
EX NO: 7 CLASS DIAGRAM

AIM:

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

THEORY:

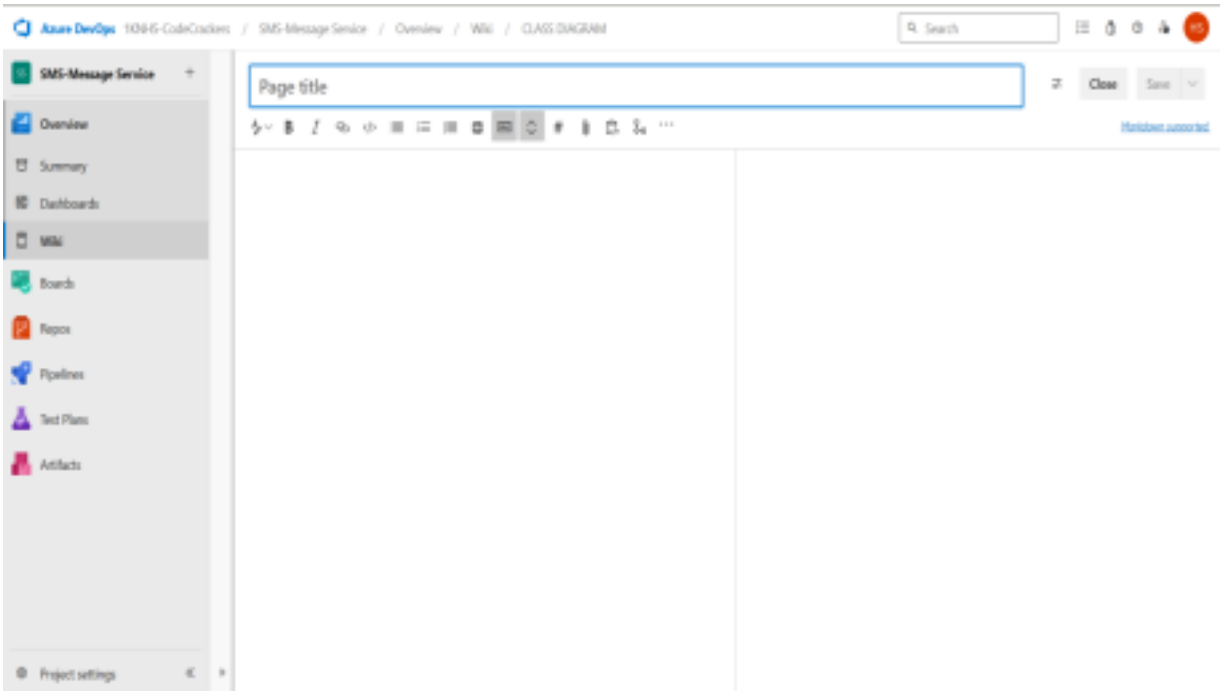
A UML class diagram is a visual tool that represents the structure of a system by showing its classes, attributes, methods, and the relationships between them.



PROCEDURE:

1. Open a project in Azure DevOps Organisations.

2. To design select wiki from menu



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

::: mermaid

classDiagram

%% ==============================

%% CORE CLASSES

%% ==============================

class Customer {

  +String customerId

  +String name

  +String email

  +String phoneNumber

  +String address

  +String passwordHash

  +register()

  +login()

  +authenticate()

  +updateProfile()

```
  +viewProfile()
  +contactSupport()
  +applyForLoan()
}
class Account {
  +String accountNumber
  +String accountType
  +double balance
  +String status
  +checkBalance()
  +viewTransactionHistory()
  +viewAccountDetails()
  +deposit()
  +withdraw()
}
class Transaction {
  +String transactionId
  +Date transactionDate
  +double amount
  +String transactionType
  +String description
  +String status
  +viewTransactionDetails()
}
%% ============================
```

```
%%       BILL PAYMENTS
%% ============================
class BillPayment {
  +String billId
  +String billerName
  +double amount
  +Date dueDate
  +String status
  +payNow()
}
class OneTimeBillPayment {
  +payOnce()
}
class RecurringBillPayment {
  +String frequency
  +Date nextPaymentDate
  +setRecurring(frequency)
  +cancelRecurring()
}
BillPayment <|-- OneTimeBillPayment
BillPayment <|-- RecurringBillPayment
%% ============================
%%     FUND TRANSFER & LOANS
%% ============================
class FundTransfer {
```

```
  +String transferId
  +double amount
  +Date scheduledDate
  +boolean isRecurring
  +boolean isExternal
  +initiateTransfer()
  +scheduleTransfer()
}
class LoanApplication {
  +String loanId
  +double amount
  +String loanType
  +int termMonths
  +String status
  +Date applicationDate
  +reviewApplication()
  +approve()
  +reject()
}
%% ============================
%%   SUPPORT, ALERTS, NOTIFY
%% ============================
class Notification {
  +String notificationId
  +String message
```

```
  +Date timestamp
  +String notificationType
  +boolean isRead
  +send()
  +markAsRead()
}
class AlertSystem {
  +sendLowBalanceAlert()
  +sendTransactionAlert()
  +sendBillDueReminder()
  +sendSecurityAlert()
}
%% ============================
%%        RELATIONSHIPS
%% ============================
Customer --> "0..*" Account
Customer --> "0..*" Transaction
Customer --> "0..*" BillPayment
Customer --> "0..*" FundTransfer
Customer --> "0..*" Notification
Customer --> "0..*" LoanApplication
Account --> "0..*" Transaction
Account --> "0..*" BillPayment : payFrom
AlertSystem --> Notification
```

RESULT:

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

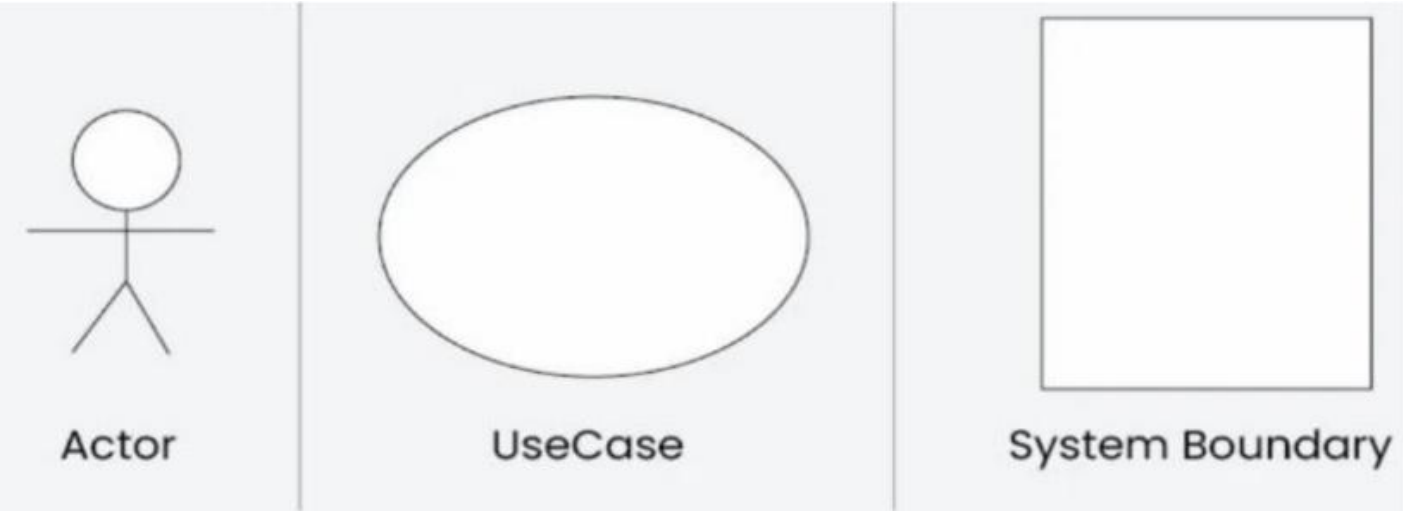
EX NO: 8 USECASE DIAGRAM

AIM:

Steps to draw the Use Case Diagram using draw.io

THEORY:

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



PROCEDURE:

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

- Open Draw.io (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 Diagram.

RESULT:

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

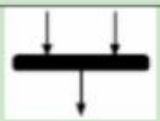
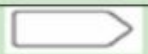
EX NO: 9 ACTIVITY DIAGRAM

AIM:

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

THEORY:

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

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

PROCEDURE:

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

RESULT:

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

EX NO: 10 ARCHITECTURE DIAGRAM

AIM:

Steps to draw the Architecture Diagram using draw.io.

THEORY:

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

PROCEDURE:

1. Draw diagram in draw.io

2. Upload the diagram in Azure DevOps wiki

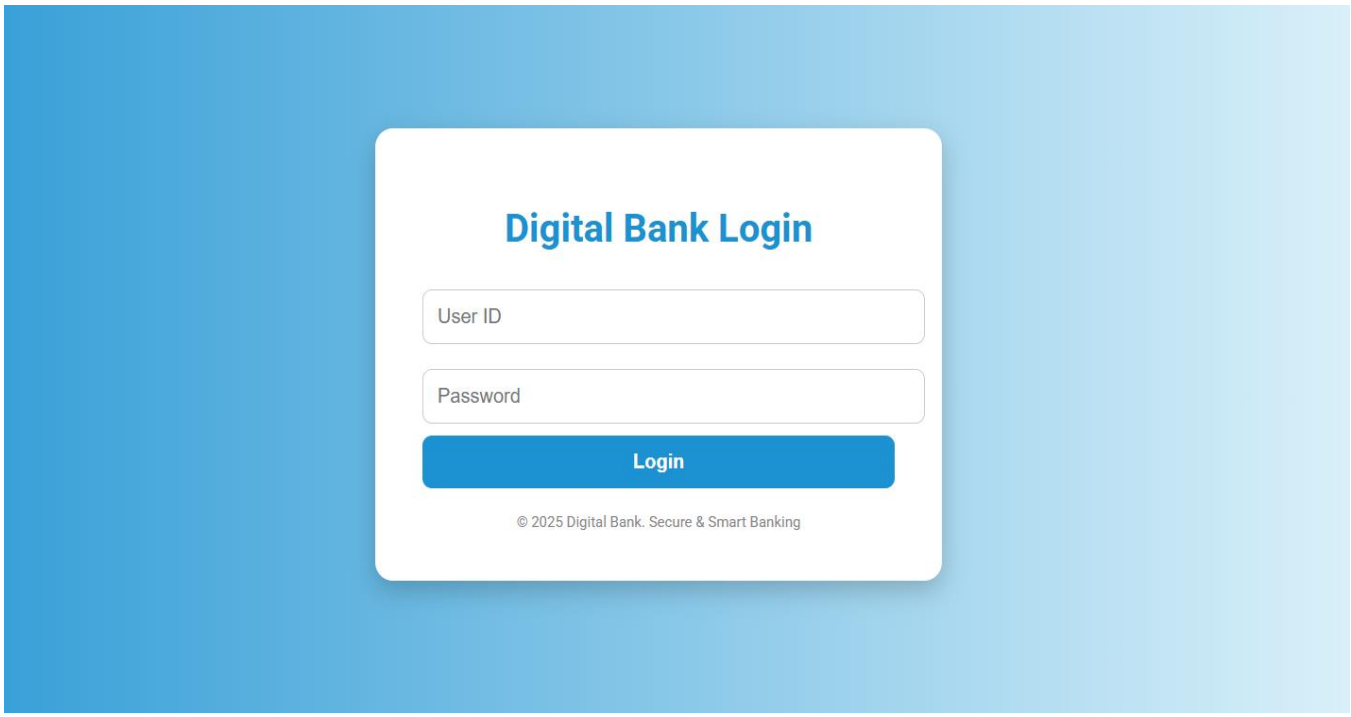
RESULT:

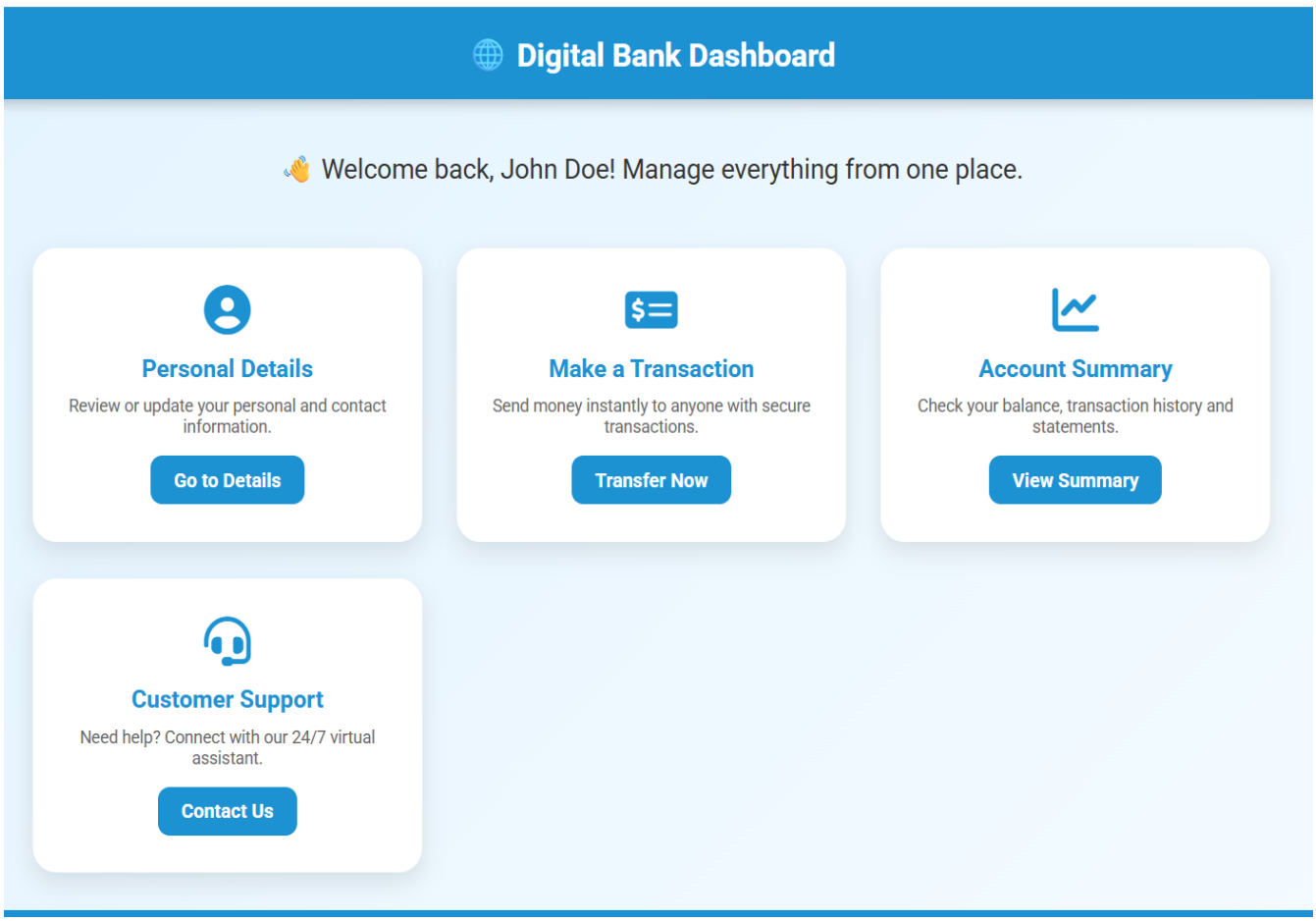
Thus, the architecture diagram for the given problem statement was designed successfully.

EX NO: 11 USER INTERFACE

AIM:

Design User Interface for the given project.





Account Overview

User ID: USER123456

Account Number: 1234567890

Account Holder: John Doe

Account Type: Savings

Balance: $5,000.00

Transaction Details

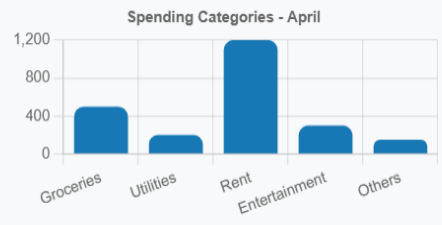
Transaction ID: TXN789654123

Recipient Name: Jane Smith

Amount: $200.00

Back to Home

Spending Overview



Make a Transaction

Recipient Name

e.g., Sarah Williams

Recipient Account Number

e.g., 123456789012

SWIFT/BIC Code

e.g., ABCDUS33XXX

Amount (USD)

$100.00

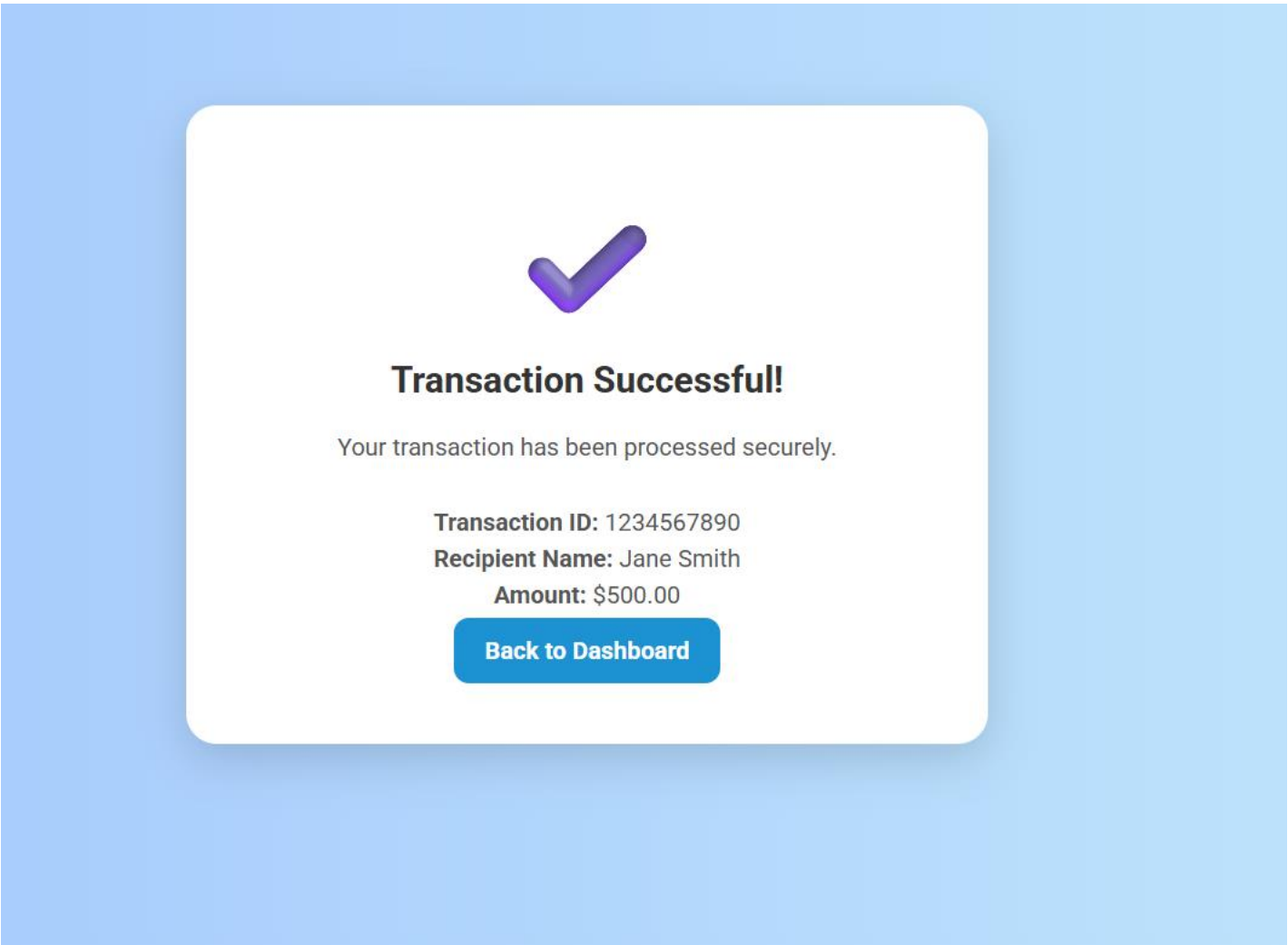
Purpose

Fund Transfer

Remarks (Optional)

Confirm Transaction

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



RESULT:

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

EX NO: 12 IMPLEMENTATIONS

AIM:

To implement the given project based on Agile Methodology.

PROCEDURE:

Step 1: Set Up an Azure DevOps Project

• Log in to Azure DevOps.

• Click "New Project" → Enter project name → Click "Create".

• Inside the project, navigate to "Repos" to store the code.

Step 2: Add Your Web Application Code

• Navigate to Repos → Click "Clone" to get the Git URL.

• Open Visual Studio Code / Terminal and run: git clone cd

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

• Commit & push: git add . git commit -m "Initial commit" git push

origin main Step 3: Set Up Build Pipeline (CI/CD - Continuous

Integration)

• Navigate to Pipelines → Click "New Pipeline".

• Select Git Repository (Azure Repos, GitHub, or Bitbucket).

• Choose Starter Pipeline or a pre-configured template for your framework.

• Modify the azure-pipelines.yml file (Example for a Node.js app):

trigger:

- main

```
pool:
 vmImage: 'ubuntu-latest'

steps:

 task: UseNode@1

inputs:

 version: '16.x'

-script: npm install

 displayName: 'Install dependencies'

-script: npm run build

 displayName: 'Build application'

-task: PublishBuildArtifacts@1

 inputs:

 pathToPublish: 'dist'

 artifactName: 'drop'
```

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

Step 4: Set Up Release Pipeline (CD - Continuous Deployment) • Go to Releases → Click "New Release Pipeline".

• Select Azure App Service or Virtual Machines (VMs) for deployment. • Add an artifact (from the build pipeline).

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

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

RESULT:

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