

# **RAJALAKSHMI ENGINEERING COLLEGE**

**THANDALAM – 602 105**

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

**ACADEMIC YEAR 2024-2025**



**RAJALAKSHMI  
ENGINEERING COLLEGE**

**CS23432**

**SOFTWARE CONSTRUCTION**

**Lab Manual**

**2024-2025**

**Name : Srihari S**

**Year/Branch/Section : II /CSE/ D**

**Register No. : 230701340**

**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 Use case 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:**

The SMS/Text Messaging Service is a real-time messaging application that enables registered users to send and receive text messages through a secure, scalable web-based and mobile-friendly platform. Built on Microsoft Azure and developed using Agile methodology, in the application, registered users can send text messages to other users through the app and group messaging with multimedia support and real-time delivery status tracking.

The goal of this project is to develop a web-based SMS/TEXT messaging app that provides a seamless text messaging service that includes **ONE to ONE MESSAGING, GROUP MESSAGING**. The application should feature a clean, interactive user interface built using **REACT, STREAMIO**.

This SMS/Text messaging app can be used as a reliable platform for both day to day users and also for commercial applications.

Key artifacts such as use case diagrams, sequence diagrams, data flow diagrams, business architecture diagram help visualize and validate the functional and non functional aspects.

### **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:**

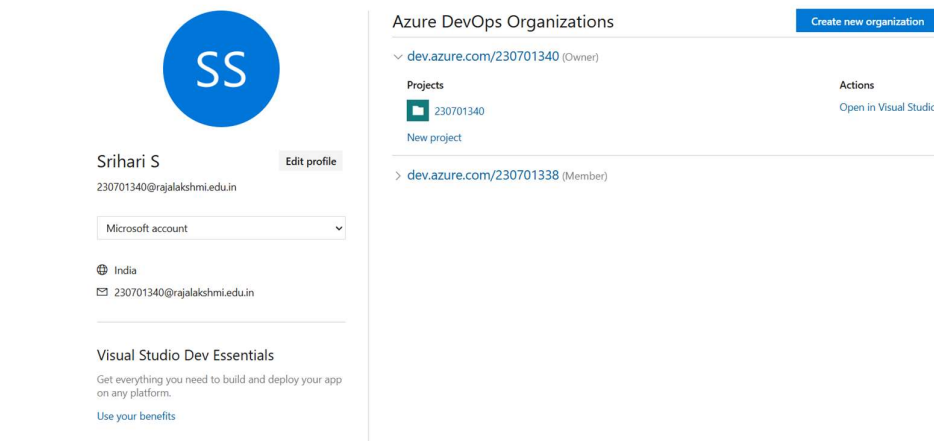
To create User Stories.

### **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.

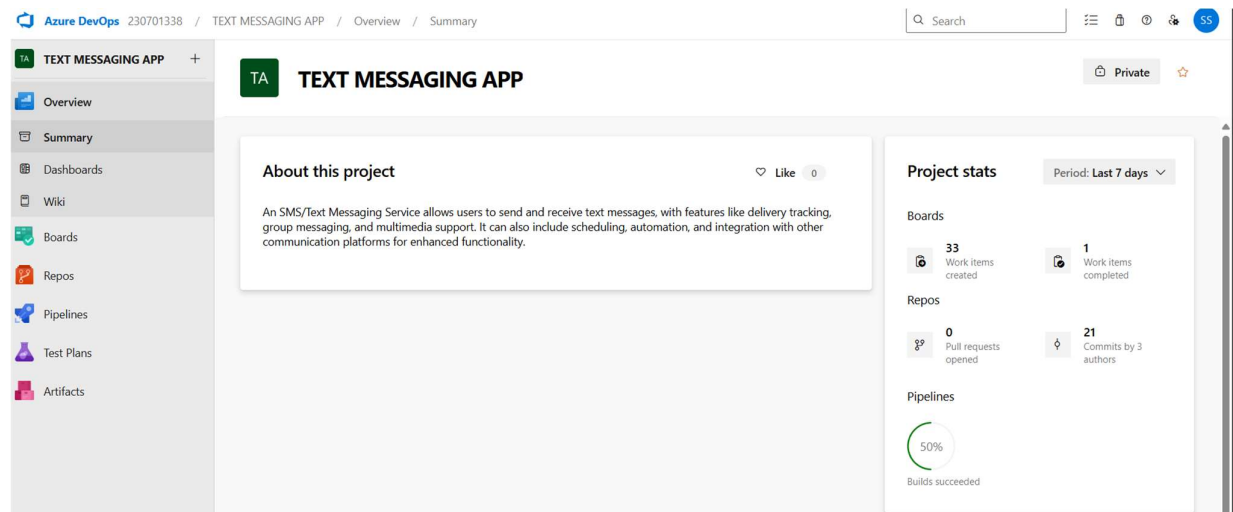
### **PROCEDURE:**

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.



USER STORY 10

10 Send a Text Message (SMS)

Srihari S

0 Comments Add Tag

Save Follow

Updated by Srihari S: 2h ago

State: New

Area: TEXT MESSAGING APP

Reason: Moved to the backlog

Iteration: TEXT MESSAGING APP

Details 1 0

Description

As a registered user, I want to send a text message to another user, so that I can communicate with them via SMS

Acceptance Criteria

1. The user must be logged in to send an SMS.

2. The system must allow the user to input a valid phone number or select a contact.

3. The message should have a 160-character limit per SMS segment.

4. The user must click the "Send" button to transmit the message.

5. The system should validate the recipient's number before sending.

6. Message status should update from "Sent" to "Delivered" and "Read"

7. If delivery fails due to network issues, the system must notify the sender and allow a retry.

8. All sent messages should be timestamped and stored in the user's message history.

Planning

Story Points

3

Priority

1

Risk

3 - Low

Classification

Value area

Business

Deployment

To track releases associated with this work item, go to [Releases](#) and turn on deployment status reporting for Boards in your pipeline's Options menu. [Learn more about deployment status reporting](#)

Development

Add link

Build

Build Chat App\_20250520.2

Updated 5h ago Succeeded

USER STORY 14

14 Create a Group Chat

Srihari S

0 Comments Add Tag

Save Follow

Updated by Srihari S: 2h ago

State: New

Area: TEXT MESSAGING APP

Reason: New

Iteration: TEXT MESSAGING APP

Details 1 0

Description

As a user, I want to create a group chat so that I can communicate with multiple participants at once.

Acceptance Criteria

Users must be logged in to create a group chat.

A group name must be required when creating a chat.

Users should be able to add at least two and up to 100 participants.

The system should validate whether participants are registered users.

Group members should be notified when they are added.

Users should be able to leave or delete a group chat.

The group should appear under a "Groups" section in the chat interface.

Group messages should be visible to all members in real time.

Planning

Story Points

5

Priority

2

Risk

2 - Medium

Classification

Value area

Business

Deployment

To track releases associated with this work item, go to [Releases](#) and turn on deployment status reporting for Boards in your pipeline's Options menu. [Learn more about deployment status reporting](#)

Development

Add link

Build

Build Chat App\_20250520.2

Updated 5h ago Succeeded

Related Work

Add link

[Add an existing work item as a parent](#)

USER STORY 13

13 Send an MMS

Srihari S

0 Comments Add Tag

Save Follow

Updated by Srihari S: 2h ago

State: New

Area: TEXT MESSAGING APP

Reason: New

Iteration: TEXT MESSAGING APP

Details 1 0

Description

As a user, I want to send an MMS so that I can share images, videos, or audio along with my messages.

Acceptance Criteria

The user must be logged in to send an MMS.

Users should be able to attach images (.jpg, .png) or videos (.mp4).

The system should enforce a 5MB file size limit per attachment.

Users should see a preview of the attached media before sending.

The message status should be tracked similarly to SMS ("Sent," "Delivered," "Read").

If the file format is unsupported, the system should notify the user.

Recipients should be able to download and view the media.

If the recipient's device does not support MMS, the system should provide an alternative (e.g., a download link).

Planning

Story Points

8

Priority

3

Risk

1 - High

Classification

Value area

Business

Deployment

To track releases associated with this work item, go to [Releases](#) and turn on deployment status reporting for Boards in your pipeline's Options menu. [Learn more about deployment status reporting](#)

Development

Add link

Build

Build Chat App\_20250520.2

Updated 5h ago Succeeded

Related Work

Add link

[Add an existing work item as a parent](#)

## **EPIC: SMS System**

The SMS feature is designed to provide users with the ability to send text messages, ensuring timely communication for important information. This system is essential for delivering critical updates efficiently, especially in situations where internet access may be limited. By leveraging real-time data from reliable sources, the service ensures users receive text

This epic consists of two major components:

1. **Real-Time SMS Notification System** – Automatically fetches and delivers important updates via SMS, ensuring messages reach users promptly.
2. **Customizable SMS Preferences** – Allows users to personalize which messages they receive based on type and priority, ensuring relevant and actionable information is delivered.

### **USER STORY 1:**

***As a user, I want to send an MMS so that I can share images, videos, or audio along with my messages.***

Acceptance Criteria:

1. The user must be logged in to send an MMS.
2. Users should be able to attach images (.jpg, .png) or videos (.mp4).
3. The system should enforce a 5MB file size limit per attachment.
4. Users should see a preview of the attached media before sending.
5. The message status should be tracked similarly to SMS ("Sent," "Delivered," "Read").
6. If the file format is unsupported, the system should notify the user.
7. Recipients should be able to download and view the media.
8. If the recipient's device does not support MMS, the system should provide an alternative (e.g., a download link).
9. The system must encrypt multimedia files for security.
10. Users should receive a confirmation once the MMS is successfully sent.

## USER STORY 2:

***As a user, I want to create a group chat so that I can communicate with multiple participants at once.***

Acceptance criteria:

1. Users must be logged in to create a group chat.
2. A group name must be required when creating a chat.
3. Users should be able to add at least two and up to 100 participants.
4. The system should validate whether participants are registered users.
5. Group members should be notified when they are added.
6. Users should be able to leave or delete a group chat.
7. The group should appear under a "Groups" section in the chat interface.
8. Group messages should be visible to all members in real time.
9. The system should allow renaming the group and changing its settings.

## USER STORY 3:

***As a registered user, I want to send a text message to another user, so that I can communicate with them via SMS.***

1. Acceptance Criteria:
2. The user must be logged in to send an SMS.
3. The system must allow the user to input a valid phone number or select a contact.
4. The message should have a 160-character limit per SMS segment.
5. The user must click the "Send" button to transmit the message.
6. The system should validate the recipient's number before sending.
7. Message status should update from "Sent" to "Delivered" and "Read"
8. If delivery fails due to network issues, the system must notify the sender and allow a retry.
9. All sent messages should be timestamped and stored in the user's message history.

## 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 UserA
```

```
    participant App
```

```
    participant StreamIO
```

```
    participant UserB
```

```
    participant Group
```

```
    %% User Login
```

```
    UserA->>App: Open App
```

```
    App->>StreamIO: Authenticate UserA
```

```
    StreamIO-->>App: Auth Token
```

```
    App-->>UserA: Load UI
```

```
    %% One-on-One Chat
```

```
    UserA->>App: Select UserB
```

```
    App->>StreamIO: Start/Join 1-on-1 channel (UserA, UserB)
```

```
    StreamIO-->>App: Channel Info
```

App-->>UserA: Display Chat with UserB

UserA->>App: Type and send message

App->>StreamIO: Send message to UserB

StreamIO-->>UserB: Deliver message

%% Media Message

UserA->>App: Upload image/video

App->>StreamIO: Send media message

StreamIO-->>UserB: Deliver media message

%% Reactions

UserB->>App: React to message

App->>StreamIO: Send reaction

StreamIO-->>UserA: Update message with reaction

%% Group Messaging

UserA->>App: Create group (Group)

App->>StreamIO: Create group channel

StreamIO-->>App: Group created

UserA->>App: Add UserB to Group

App->>StreamIO: Add members to Group

StreamIO-->>Group: Notify members

UserB->>Group: Send message

StreamIO-->>UserA: Deliver group message

...

## Sequence Diagram

```
sequenceDiagram
    participant UserA
    participant App
    participant StreamIO
    participant UserB
    participant Group

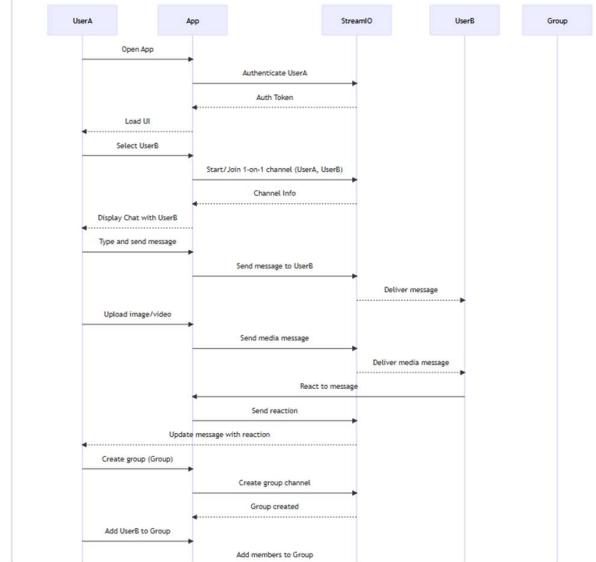
    %% User Login
    UserA->>App: Open App
    App->>StreamIO: Authenticate UserA
    StreamIO-->>App: Auth Token
    App->>UserA: Load UI

    %% One-on-One Chat
    UserA->>App: Select UserB
    App->>StreamIO: Start/Join 1-on-1 channel (UserA, UserB)
    StreamIO-->>App: Channel Info
    App->>UserA: Display Chat with UserB
    UserA->>App: Type and send message
    App->>StreamIO: Send message to UserB
    StreamIO-->>UserB: Deliver message

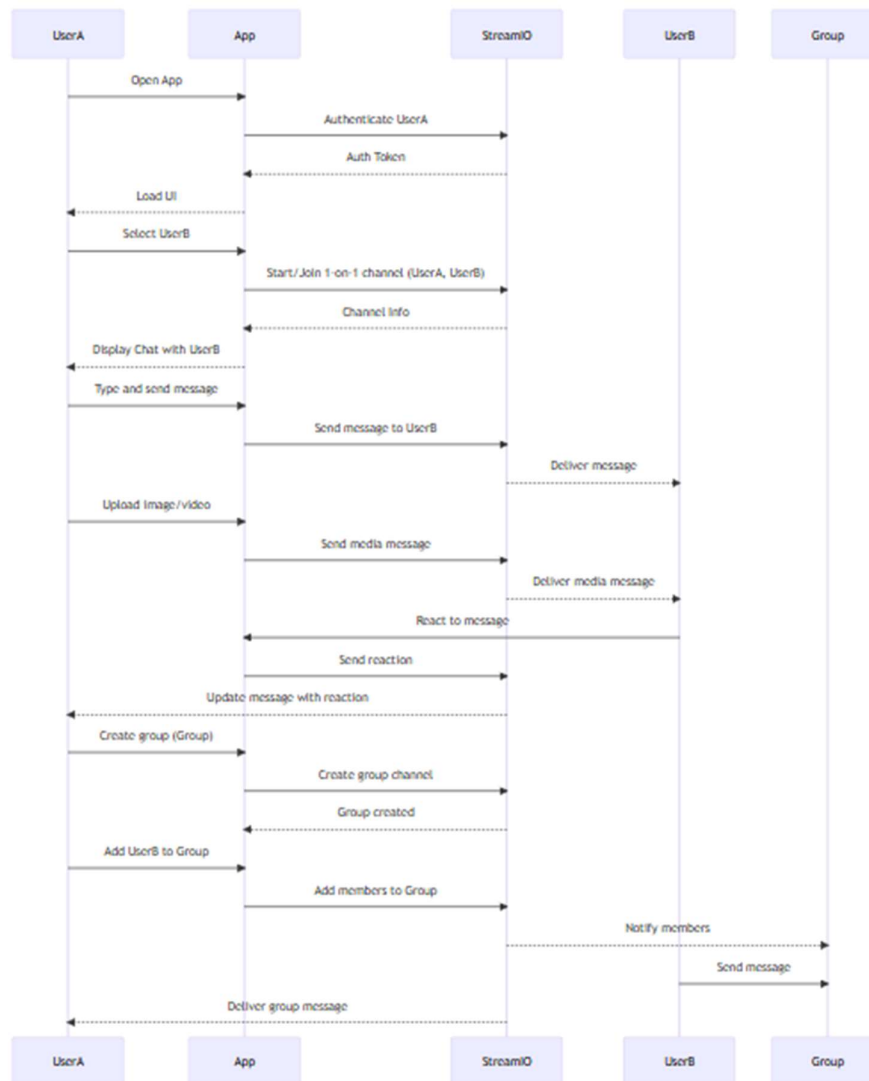
    %% Media Message
    UserA->>App: Upload image/video
    App->>StreamIO: Send media message
    StreamIO-->>UserB: Deliver media message

    %% Reactions
    UserB->>App: React to message
    App->>StreamIO: Send reaction
    StreamIO-->>UserA: Update message with reaction

    %% Group Messaging
    UserA->>App: Create group (Group)
    App->>StreamIO: Create group channel
    StreamIO-->>App: Group created
    App->>UserA: Add UserB to Group
    App->>StreamIO: Add members to Group
    StreamIO-->>UserB: Notify members
    UserB->>Group: Send message
    StreamIO-->>UserA: Deliver group message
```



4. Click wiki menu and select the page.



**RESULT:**

The sequence diagram is drawn successfully.

## EX NO: 6

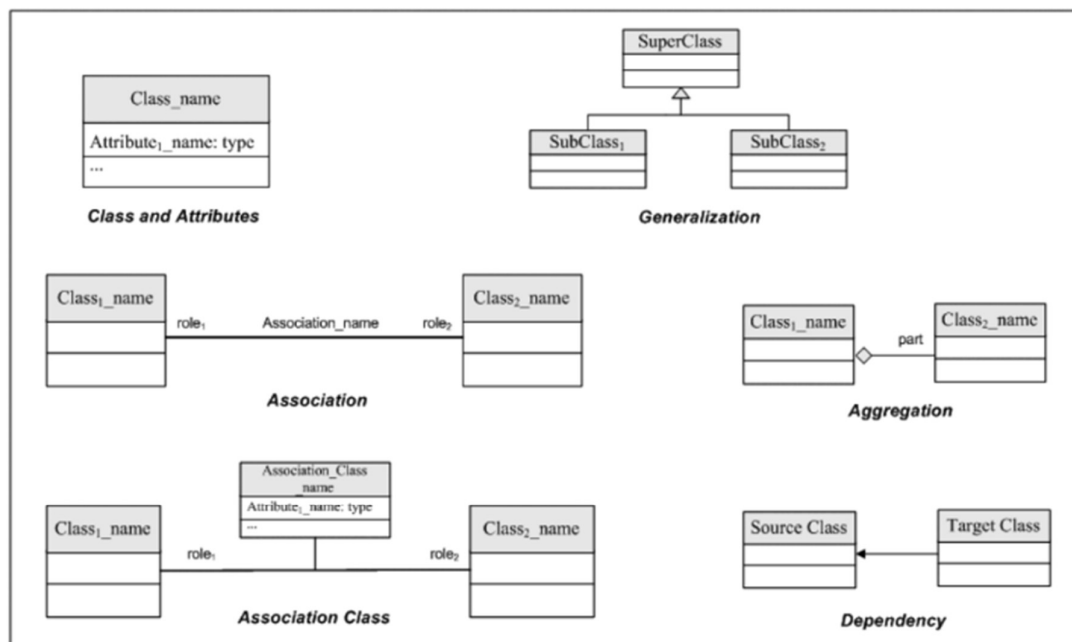
## CLASS DIAGRAM

### 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.



Notations in class diagram

### 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 User {  
    +String id
```



```
+String username
+String email
+String avatarUrl
+login()
+logout()
+searchUser()
}
class Chat {
    +String id
    +DateTime createdAt
    +sendMessage()
    +getMessages()
}
class GroupChat {
    +String name
    +String createdBy
    +addParticipant(user: User)
    +removeParticipant(user: User)
    +leaveGroup(user: User)
    +exitGroup()
}
class Message {
    +String id
    +String senderId
```

```
+String chatId
+String content
+String? mediaUrl
+String? replyToMessageId
+DateTime timestamp
+send()
```

```
}
```

```
class Reaction {
```

```
+String messageId
```

```
+String userId
```

```
+String emoji
```

```
}
```

User --> Chat : participates in

Chat <|-- GroupChat

Chat --> Message : contains

Message --> Reaction : has

Message --> Message : replies to

GroupChat --> User : has participants

...

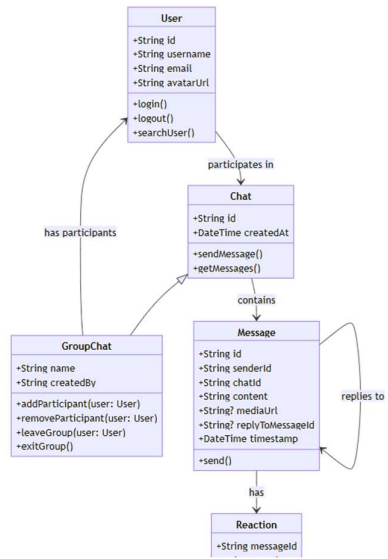
# Class Diagram

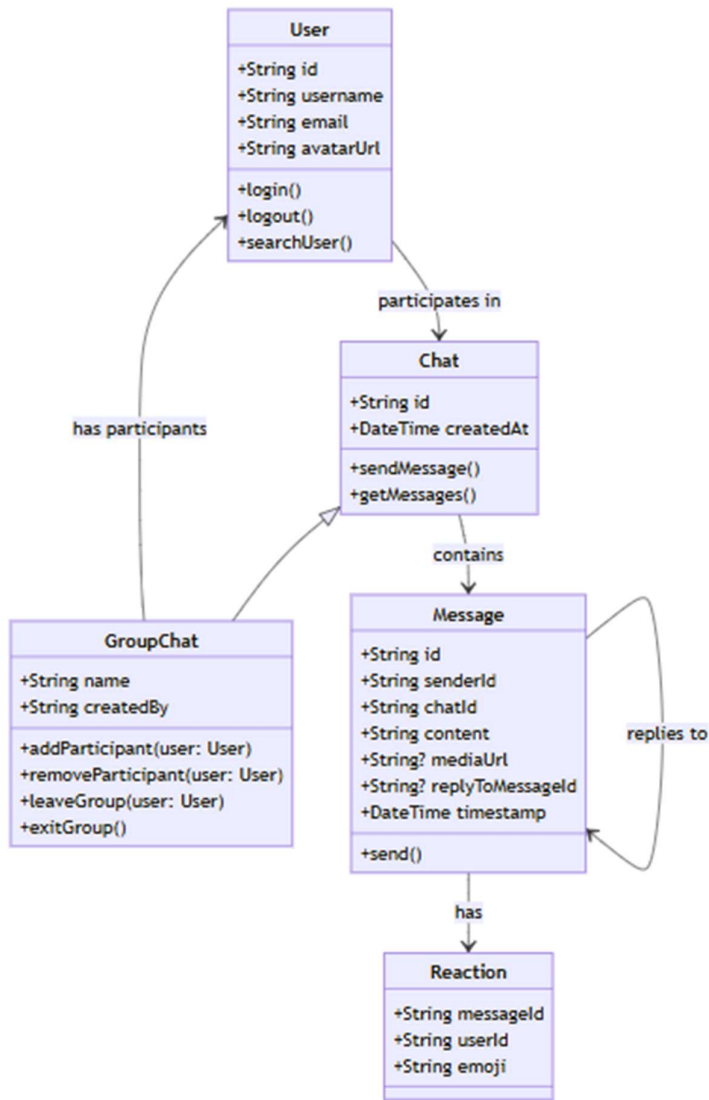
Close Save

🔍 B I 🔗 📄 📁 # 🏠 ⚙️ ...

Mermaid.js

```
%% mermaid
classDiagram
    class User {
        +String id
        +String username
        +String email
        +String avatarUrl
        +login()
        +logout()
        +searchUser()
    }
    class Chat {
        +String id
        +DateTime createdAt
        +sendMessage()
        +getMessages()
    }
    class GroupChat {
        +String name
        +String createdBy
        +addParticipant(user: User)
        +removeParticipant(user: User)
        +leaveGroup(user: User)
        +exitGroup()
    }
    class Message {
        +String id
        +String senderId
        +String chatId
        +String content
        +String? mediaUrl
        +String? replyToMessageId
        +DateTime timestamp
        +send()
    }
    class Reaction {
        +String messageId
        +String userId
        +String emoji
    }
    User --> Chat : participates in
    User --> GroupChat : has participants
    Chat --> GroupChat
    Chat --> Message : contains
    Message --> Message : replies to
    Message --> Reaction : has
```





## RESULT:

Thus the class diagram has been designed successfully.

## EX NO: 7

## USE CASE DIAGRAM

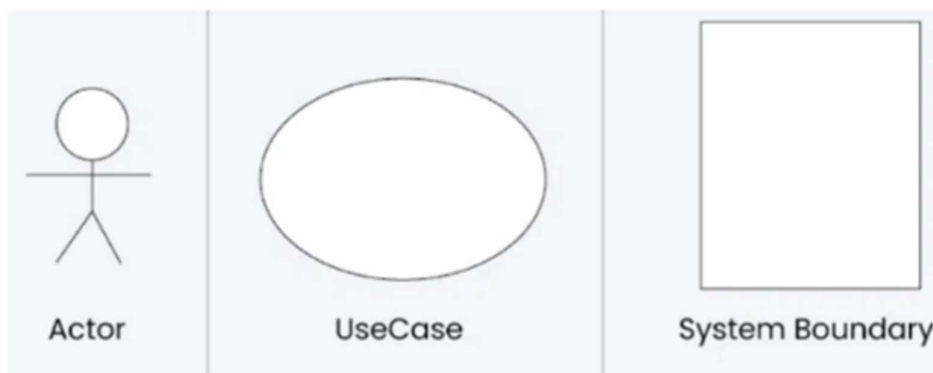
### AIM:

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 ([draw.io](https://draw.io)).
- Click "Create New Diagram" and select "Blank" or "UML Use Case" template.
- Add Actors (Users, Admins, External Systems) from the UML section.
- Add Use Cases (Functionalities) using ellipses.
- Connect Actors to Use Cases with lines (solid for direct interaction, dashed for `<<include>>` and `<<extend>>`).
- Save the diagram as .drawio or export as PNG/JPG/SVG.

Step 2: Upload the Diagram to Azure DevOps

Option 1: Add to Azure DevOps Wiki

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

## Option 2: Attach to Work Items in Azure Boards

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

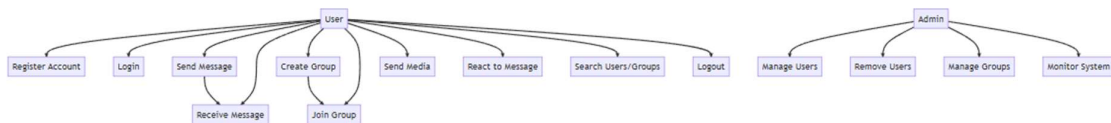
```

graph TD
    User[User]
    Admin[Admin]

    User --> RegisterAccount[Register Account]
    User --> Login[Login]
    User --> SendMessage[Send Message]
    User --> ReceiveMessage[Receive Message]
    User --> CreateGroup[Create Group]
    User --> JoinGroup[Join Group]
    User --> SendMedia[Send Media]
    User --> ReactMessage[React to Message]
    User --> Search[Search Users/Groups]
    User --> Logout[Logout]

    Admin --> ManageUsers[Manage Users]
    Admin --> RemoveUsers[Remove Users]
    Admin --> ManageGroups[Manage Groups]
    Admin --> MonitorSystem[Monitor System]

    SendMessage --> ReceiveMessage
    CreateGroup --> JoinGroup
  
```



## RESULT:

The use case diagram was designed successfully.

## EX NO: 8






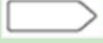





## ACTIVITY DIAGRAM

### AIM :

To draw a sample activity diagram for the Weather Application.

### THEORY:

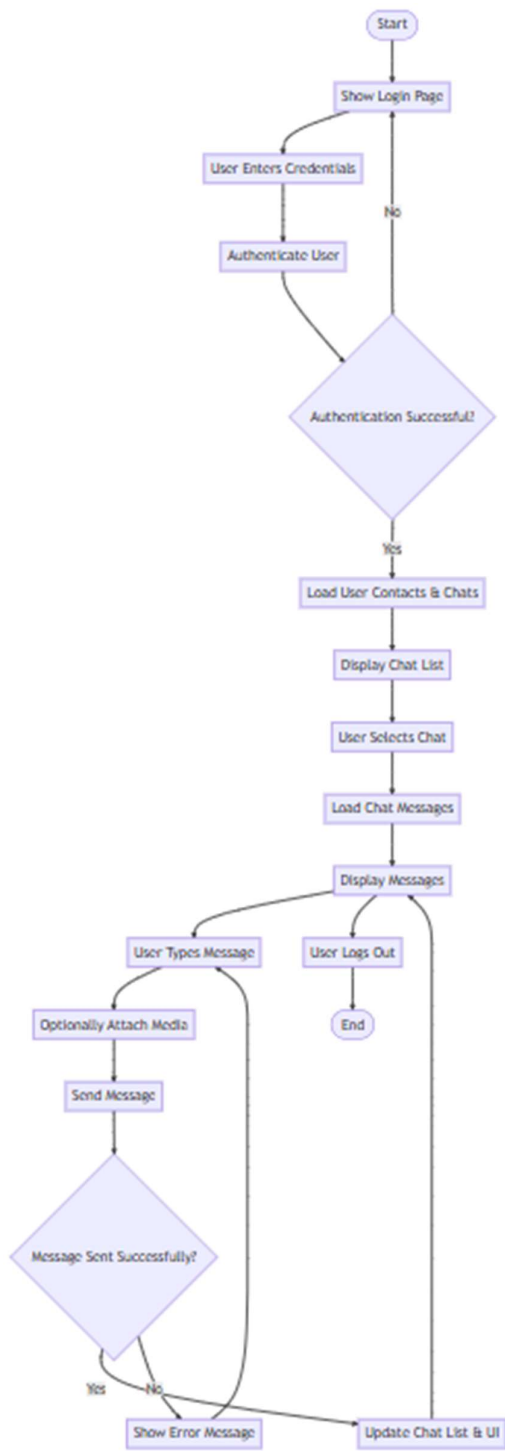
Activity diagrams are an essential part of the Unified Modelling Language (UML) that help visualize workflows, processes, or activities within a system. They depict how different actions are connected and how a system moves from one state to another.

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:

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.

```

::: mermaid
flowchart TD
    Client[Client Devices] --> APIM[API Management]
    APIM --> AppService[App Service]

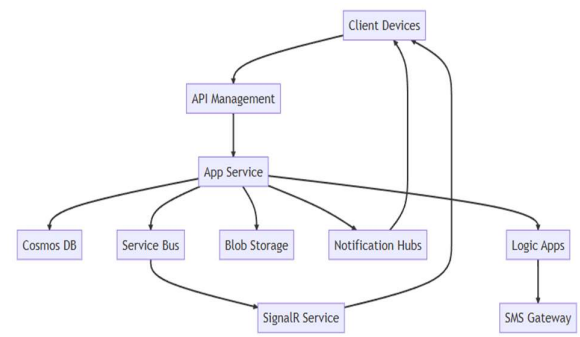
    AppService --> CosmosDB[Cosmos DB]
    AppService --> ServiceBus[Service Bus]
    AppService --> Storage[Blob Storage]

    ServiceBus --> SignalR[SignalR Service]
    SignalR --> Client

    AppService --> Notifications[Notification Hubs]
    Notifications --> Client

    AppService --> Logic[Logic Apps]
    Logic --> SMS[SMS Gateway]
    :::

```



## RESULT:

The architecture diagram was designed successfully

**EX NO: 10**

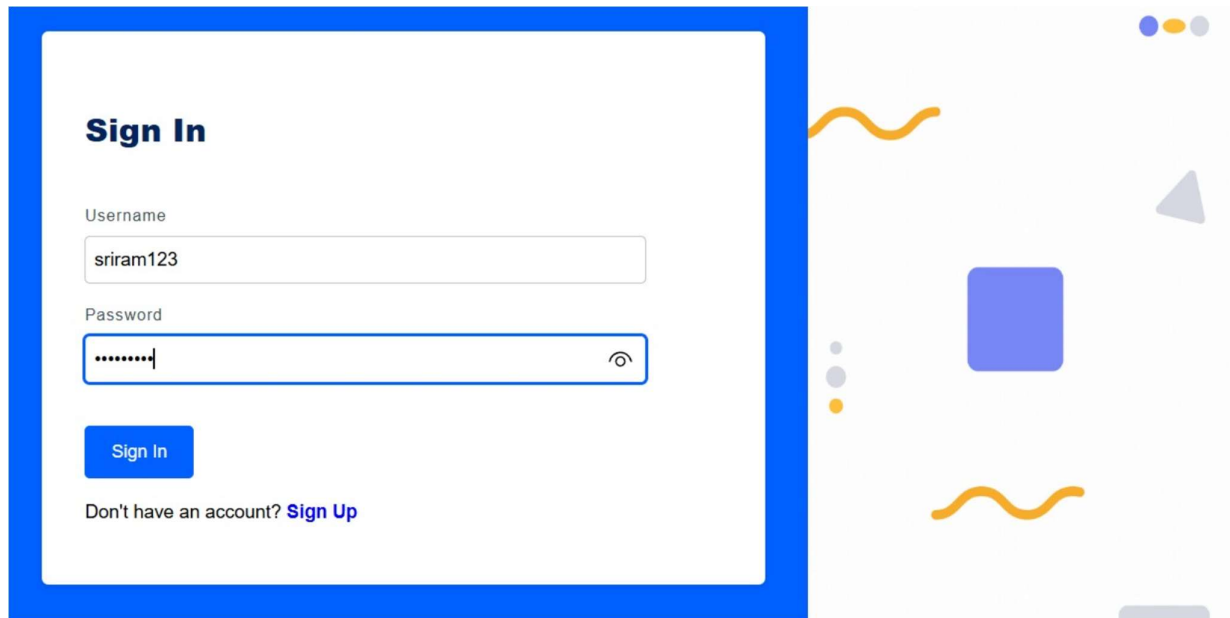
## **USER INTERFACE**

**AIM:**

Design User Interface for the Weather App.

**UI DESIGNS OF WEATHER APP:**


**LOGIN PAGE:**



**Sign In**

Username

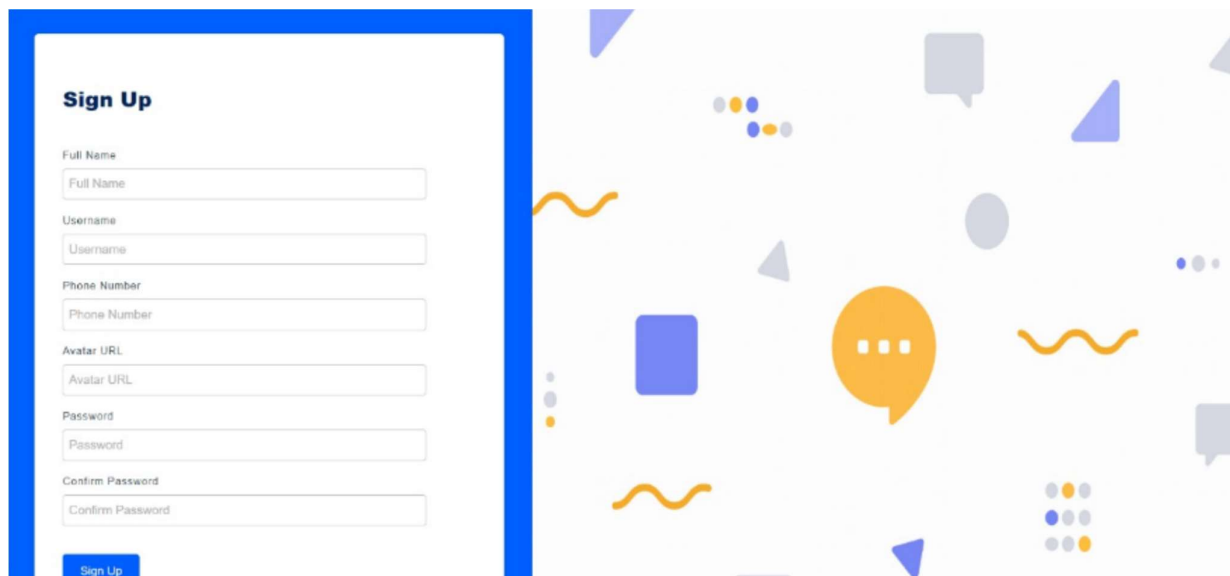
Password

[Sign In](#)

Don't have an account? [Sign Up](#)

**SIGN UP:**



**Sign Up**

Full Name

Username

Phone Number

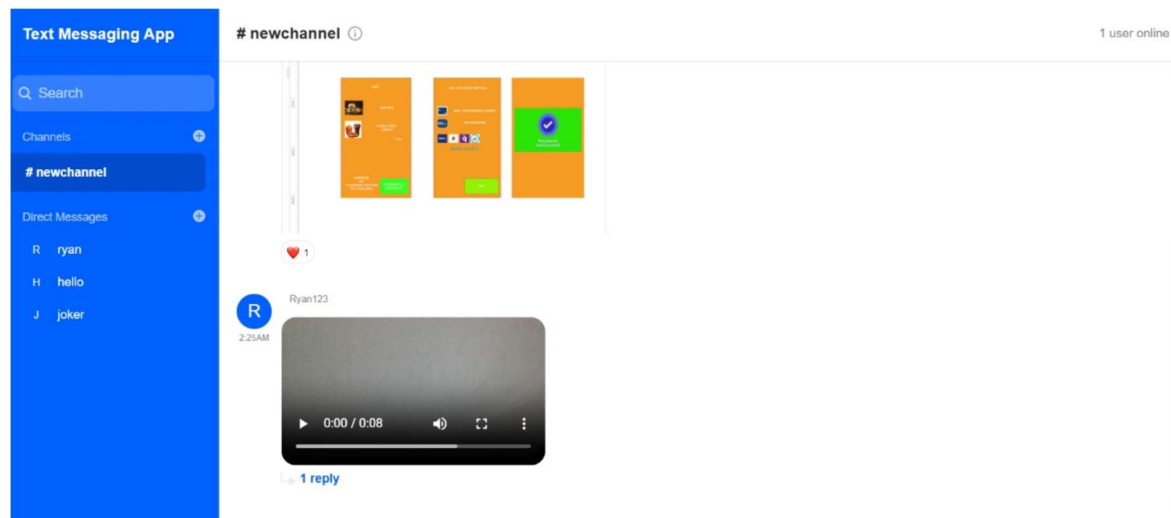
Avatar URL

Password

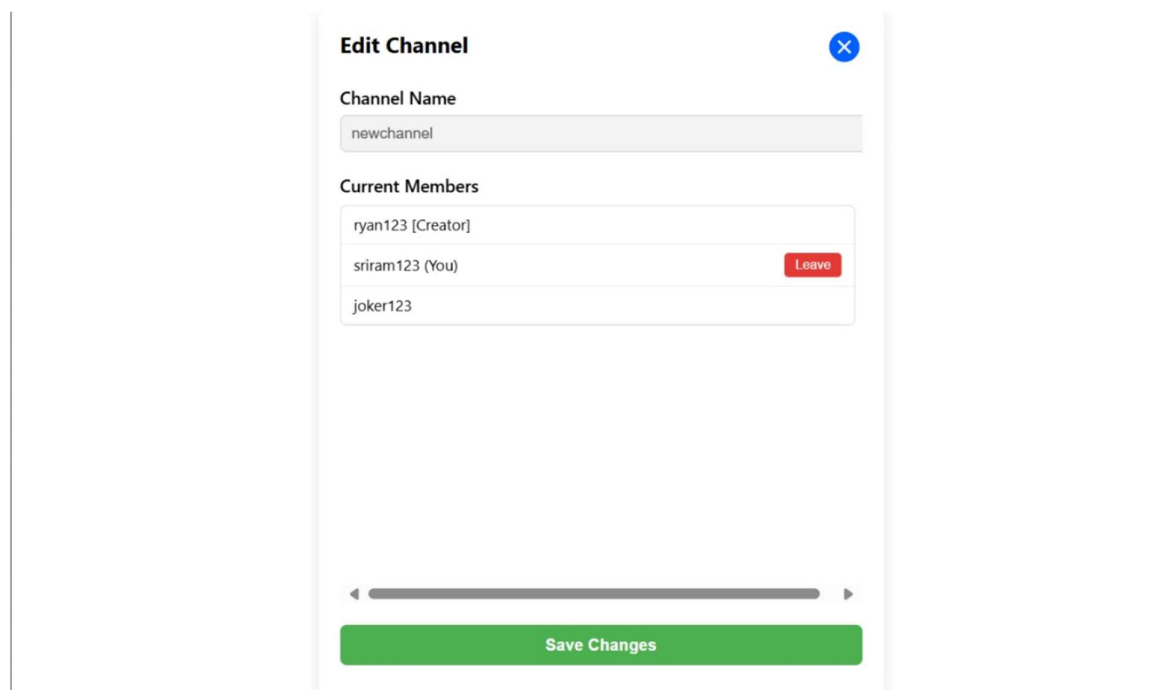
Confirm Password

[Sign Up](#)

## GROUP CHAT:



## GROUP CHAT CREATION:



## 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>
```

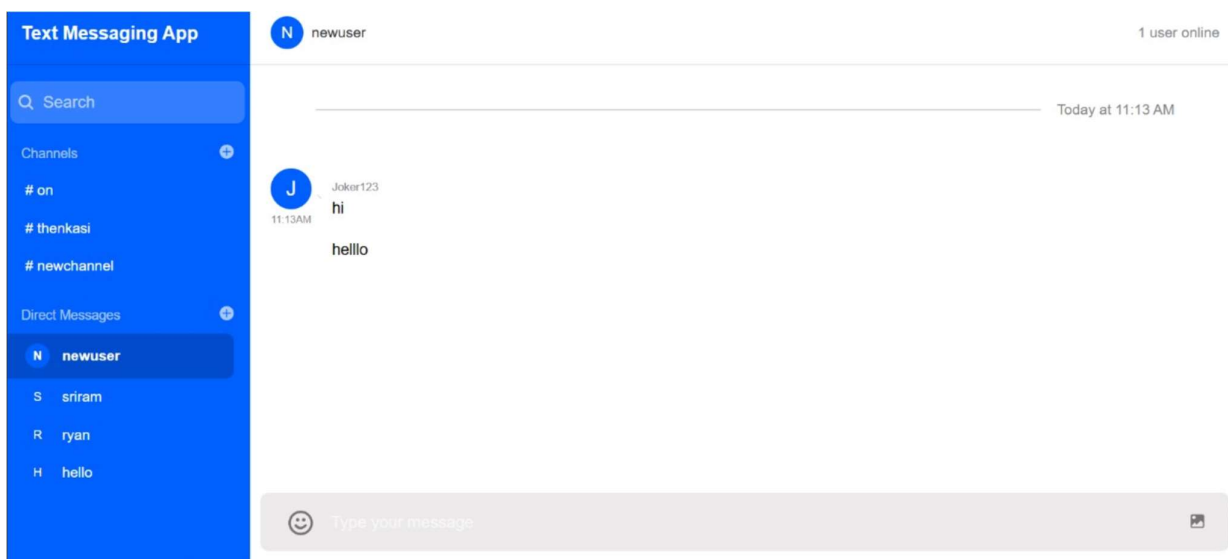
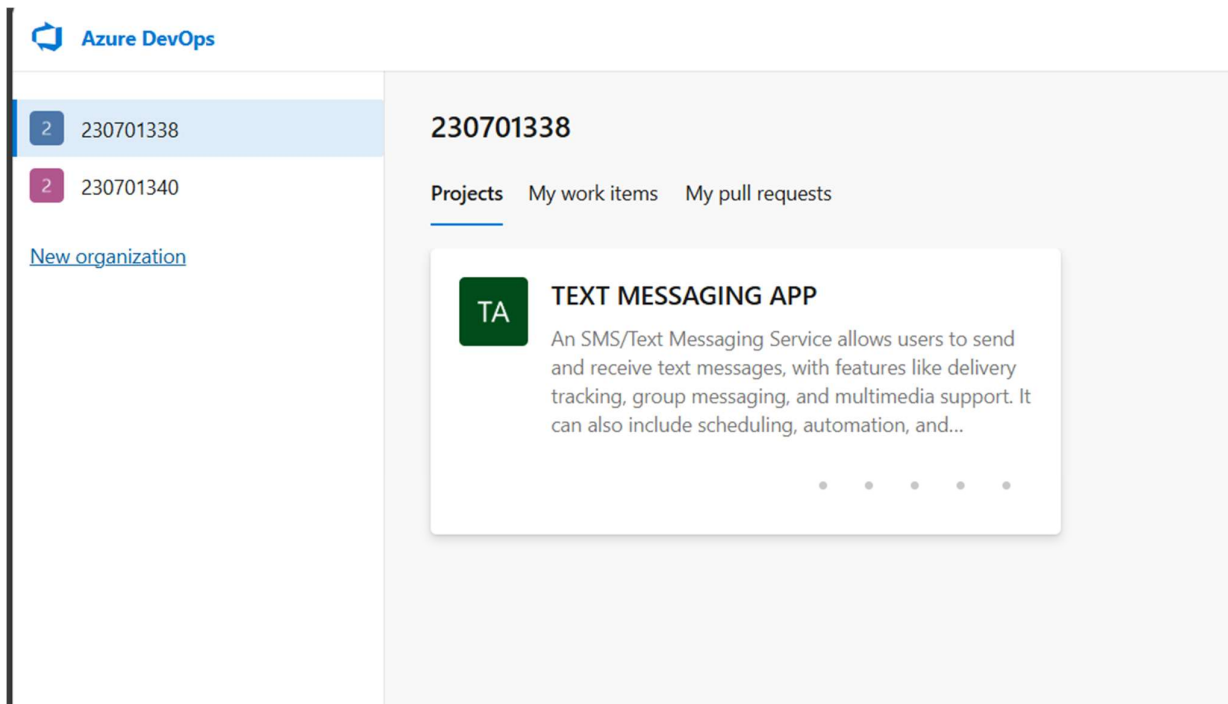
- Add web application code (HTML, CSS, JavaScript, React, Angular, or backend like Node.js, .NET, Python, etc.).
- Commit & push:

```
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 Weather App project using Azure DevOps Test Plans, ensuring that all user stories meet their acceptance criteria as defined in Agile methodology.

### PROCEDURE:

#### Step 1: Open Azure DevOps Project

- Log in to your Azure DevOps account.
- Open your project.

#### Step 2: Navigate to Test Plans

- In the left menu, click on Test Plans.
- Click on “New Test Plan” and give it a name (e.g., *Emergency Alerts*).

#### Step 3: Create Test Suites

- Under the test plan, click “+ New Suite” to add test suites for each Epic or Feature.
  - Suite 1: *Emergency Alerts*
  - Suite 2: *Custom Alert Preferences*
  - Suite 3: *Safety Tips Display*

#### Step 4: Add Test Cases

- Click on a suite → + New Test Case.
- Enter the following details for each test case:
  - Test Case ID: (e.g., TC001)
  - Title: (e.g., *Receive Notification in Time*)
  - Scenario: Describe the situation being tested.
  - Steps:
    - Launch the app
    - Trigger extreme weather API event
    - Observe time taken for notification
  - Expected Result: Notification should appear within 10 seconds.

#### Step 5: Execute Test Cases

- Click on each test case and select Run for web application.
- Follow the steps, mark results as Pass/Fail, and provide Actual Result and Remarks.

## Step 6: Track and Report

- Go to Test Plans → Charts to view test progress.
- Use filters to track:
  - Passed/Failed test cases
  - Test coverage per user story
  - Bugs linked to failed test cases

<

Group Management...

May 19 - May 26

Current

100% run, 100% passed.[View report](#)

Test Suites

Filter suites by name

Group Management (4)

Group Management (ID: 89)

DefineExecuteChart

☐🔗↶≡

Run for web application

Test Points (4 items)

⌵

Run for web application

<input type="checkbox"/> Title	Outcome	Order	Test Case Id
<input type="checkbox"/> Creator adds a new participant	✔ Passed	1	90
<input type="checkbox"/> Creator removes a participant	✔ Passed	2	91
<input type="checkbox"/> Participant leaves group	✔ Passed	3	92
<input type="checkbox"/> Non-creator tries to remove someone	✔ Passed	4	93

Group Management (ID: 89)

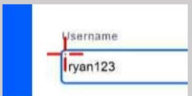
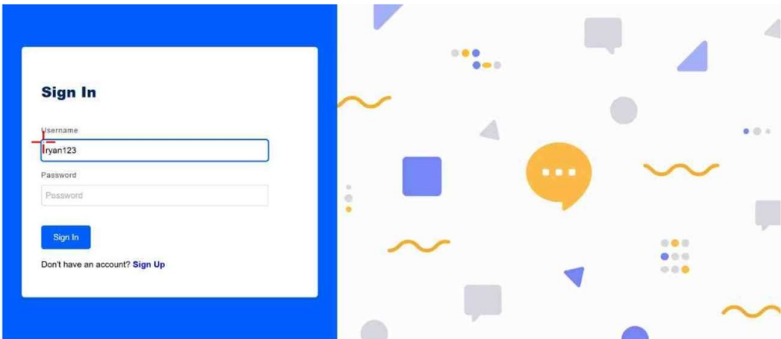




Define

Execute

Chart

Test Points (4 items)			
<input type="checkbox"/> Title	Outcome	Order	Test Case Id
<input type="checkbox"/> Creator adds a new participant	✓ Passed	1	90
<input type="checkbox"/> Creator removes a participant	✓ Passed	2	91
<input type="checkbox"/> Participant leaves group	✓ Passed	3	92
<input type="checkbox"/> Non-creator tries to remove someone	✓ Passed	4	93



Actions	Image	Full screen image
Update input field with value ryan123		
Click on text box		
Enter password		
Press <ENTER>		
Click on		

## Summary



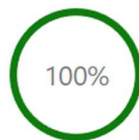
1  
Test plans



4  
Test points



4 (4 / 4)  
Test points run



Run



100% (4 / 4)  
Pass rate



4 ● Passed

## 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 Weather 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)

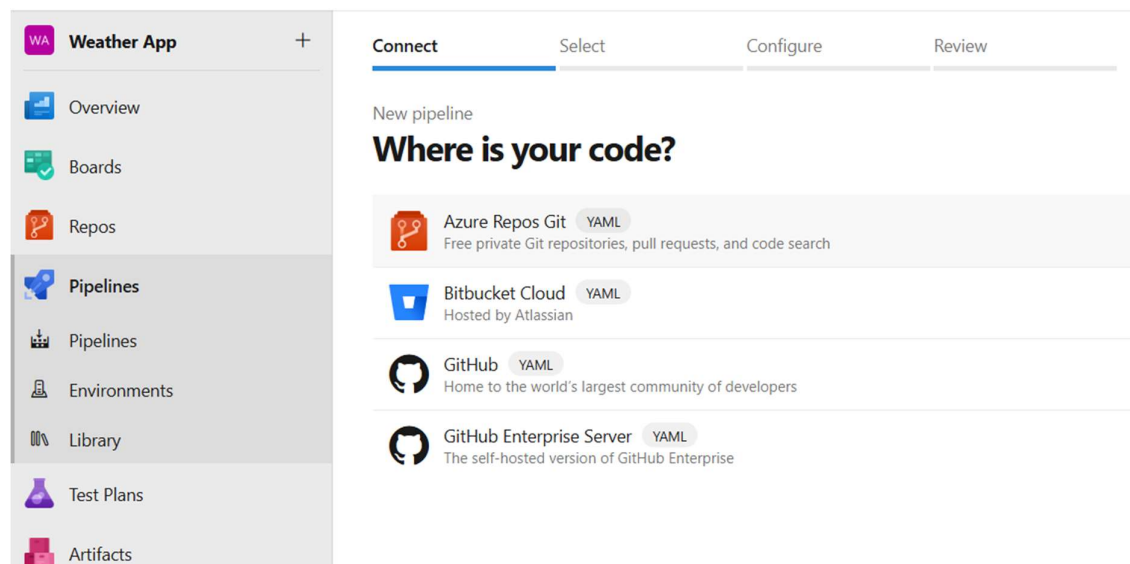
- Add pre-deployment approvals to each stage for review.
- Add gates like API health checks or test validations.

## 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.



WA Weather App +

Overview

Boards

Repos

Pipelines

Pipelines

Environments

Library

Test Plans

Artifacts

✓ Connect

✓ Select

Configure

Review

New pipeline

Configure your pipeline

HTML

Archive your static HTML project and save it with the build record.

Starter pipeline

Start with a minimal pipeline that you can customize to build and deploy your code.

Existing Azure Pipelines YAML file

Select an Azure Pipelines YAML file in any branch of the repository.

Show more

WA Weather App +

Overview

Boards

Repos

Pipelines

Pipelines

Environments

Library

Test Plans

Artifacts

✓ Connect

✓ Select

✓ Configure

Review

New pipeline

Review your pipeline YAML

230701332-CS23432-SC / azure-pipelines-1.yml \* 🔗

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

# HTML

# Archive your static HTML project and save it with the build record.

# Add steps that build, run tests, deploy, and more:

# <https://aka.ms/yaml>

trigger:

- main

pool:

- name: Default

steps:

Settings

- task: ArchiveFiles@2

- inputs:

- rootFolderOrFile: '\$(build.sourcesDirectory)'

- includeRootFolder: false

Settings

- task: PublishBuildArtifacts@1

#20250520.2 • Update azure-pipelines.yml for Azure Pipelines
 Run new

This run is being retained as one of 3 recent runs by master (Branch).
 View retention leases

Summary
Code Coverage

Individual CI by Sriram Umakanthan
 View 15 changes

Repository and version  
 ♦ Chat App  
 📦 master    🔗 387686b8

Time started and elapsed  
 🕒 Yesterday at 4:45 pm  
 ⌚ 1m 4s

Related  
 📁 0 work items  
 📦 0 artifacts

Tests and coverage  
 📊 [Get started](#)

Warnings
3

Free memory is lower than 5%. Currently used: 96.57%  
Install and build frontend

Free memory is lower than 5%. Currently used: 96.57%  
Install and build frontend

Free memory is lower than 5%. Currently used: 96.57%  
Install and build frontend

Jobs

Name	Status	Duration
Job	Success	⌚ 49s

USER STORY 10

10 Send a Text Message (SMS)
 

Srihari S
 0 Comments
Add Tag

State New
 Area TEXT MESSAGING APP

Reason Moved to the backlog
 Iteration TEXT MESSAGING APP

### Description

As a registered user, I want to send a text message to another user, so that I can communicate with them via SMS

### Acceptance Criteria

1. The user must be logged in to send an SMS.
2. The system must allow the user to input a valid phone number or select a contact.
3. The message should have a 160-character limit per SMS segment.
4. The user must click the "Send" button to transmit the message.
5. The system should validate the recipient's number before sending.
6. Message status should update from "Sent" to "Delivered" and "Read"
7. If delivery fails due to network issues, the system must notify the sender and allow a retry.
8. All sent messages should be timestamped and stored in the user's message history.

### Planning

Story Points  
3

Priority  
1

Risk  
3 - Low

### Classification

Value area  
Business

## RESULT:

Thus the CI/CD pipeline has been successfully implemented.