

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING ACADEMIC YEAR 2024-2025

EVEN SEMESTER

****

**CS23432 SOFTWARE ENGINEERING LAB LAB MANUAL**

**SECOND YEAR FOURTH SEMESTER**

**2024- 2025**

**EVEN SEMESTER**

|  |  |
| --- | --- |
| **Ex No** | **List of Experiments** |
| 1 | Study of Azure DevOps |
| 2 | Designing Project using AGILE-SCRUM Methodology. |
| 3 | Agile Planning |
| 4 | User stories – Creation |
| 5 | Architecture Diagram Using AZURE |
| 6 | Designing 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 |

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

# LAB PLAN

**CS19442-SOFTWARE ENGINEERING LAB**

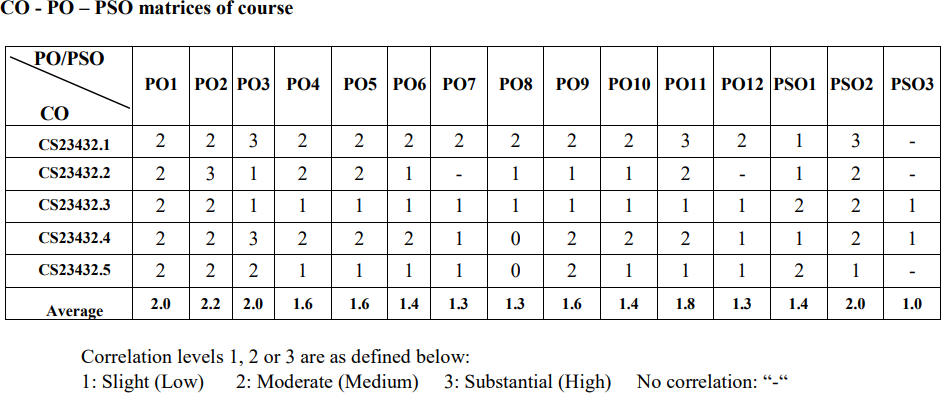
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ex No** | **Date** | **Topic** | **Page No** | **Sign** |
| 1 |  | Study of Azure DevOps |  |  |
| 2 |  | Writing Problem Statement |  |  |
| 3 |  | Designing Project using AGILE-SCRUM Methodology by using Azure. |  |  |
| 4 |  | Agile Planning | 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

**Course Outcomes (COs)**

**Course Name: Software Engineering**

**Course Code: CS23432**

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

****

**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. Azure Repos (Version Control)

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

* 1. 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. Azure Boards (Agile Project Management)

Manages work using Kanban boards, Scrum boards, and dashboards. Tracks user stories, tasks, bugs, sprints, and releases.

* 1. Azure Test Plans (Testing)

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

* 1. Azure Artifacts (Package Management)

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

## Getting Started with Azure DevOps

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

Sign in with a Microsoft Account.

Create an Organization and a Project. Step 2: Set Up a Repository (Azure Repos)

Navigate to Repos.

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

Step 3: Configure a CI/CD Pipeline (Azure Pipelines)

Go to Pipelines → New Pipeline.

Select a source code repository (Azure Repos, GitHub, etc.).

Define the pipeline using YAML or the Classic Editor. Run the pipeline to build and deploy the application.

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

Create work items, user stories, and tasks. Organize sprints and track progress.

Step 5: Implement Testing (Azure Test Plans) Go to Test Plans.

Create and run test cases

View test results and track bugs.

## Result:

The study was successfully completed.

**EX NO: 2**

# PROBLEM STATEMENT

**AIM:**

To prepare PROBLEM STATEMENT for your given project.

## Problem Statement:

**Candidate Performance Analysis & Recommendation Tool**

With the growing demand for self-paced learning and continuous skill evaluation, there

is a need for an interactive Online Quiz Application that allows users to assess their knowledge

in various domains. This application will help educational institutions and learners by providing

topic-wise quizzes, instant feedback, and cumulative scoring. The tool aids users in identifying their

strengths and weaknesses, and recommends further topics for improvement, thus enhancing the

overall learning experience.

## Result:

The problem statement was written successfully.

# EX NO: 3

**Aim**:

To prepare an Agile Plan.

# THEORY

# AGILE PLANNING

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

## Aim:

# CREATE USER STORIES

To create User Stories

# THEORY

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

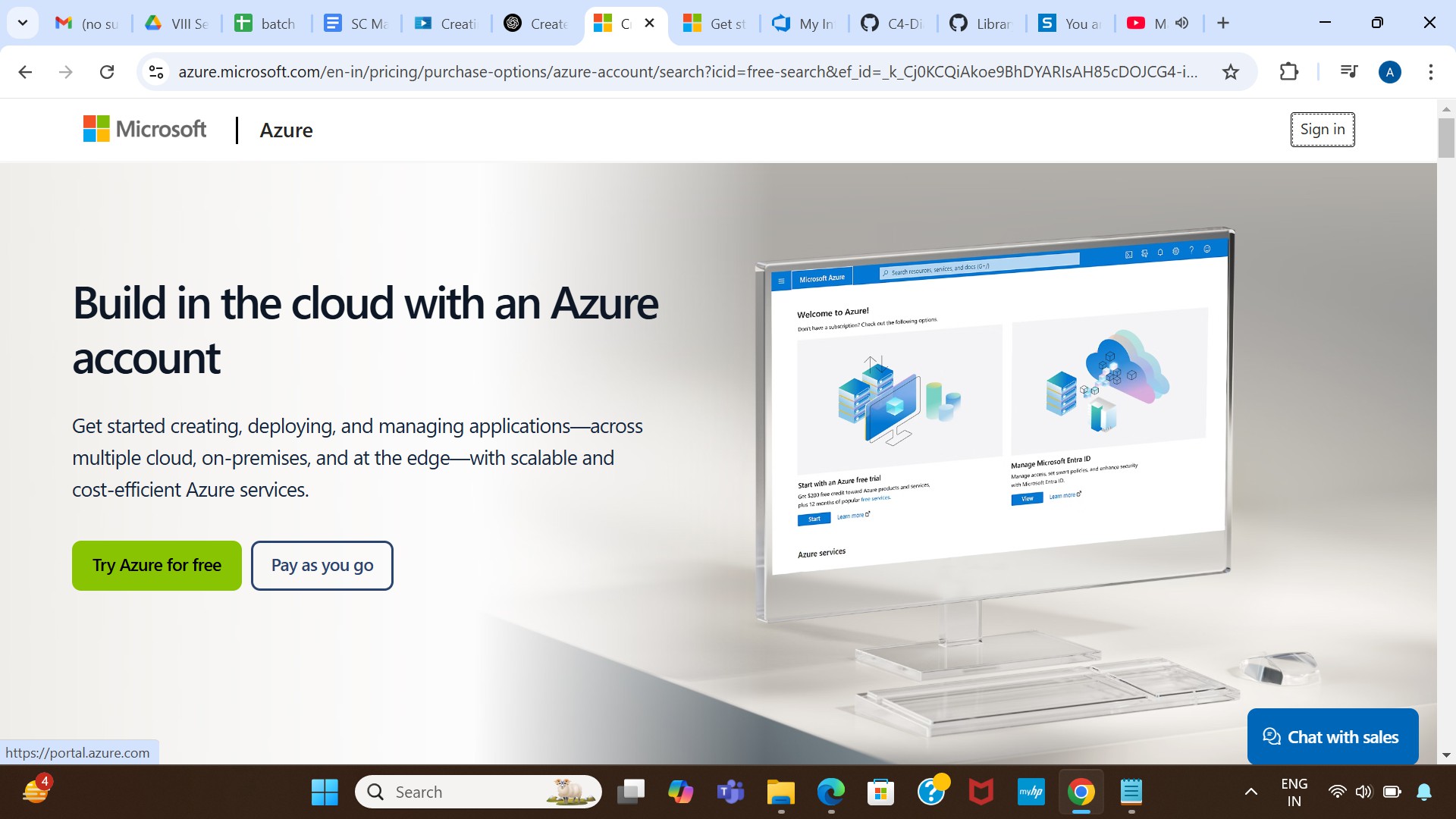
User story template

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

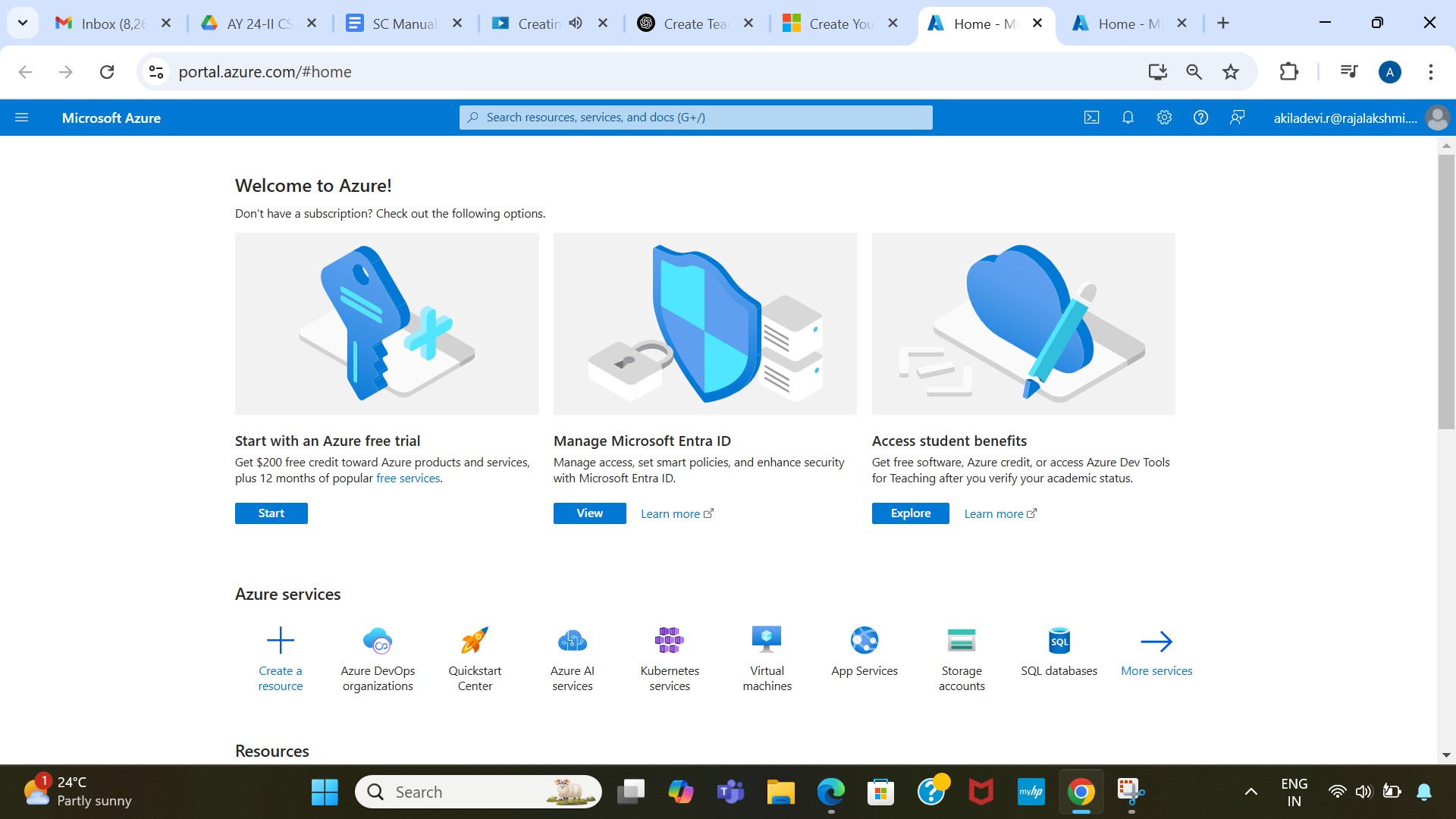
**Procedure:**

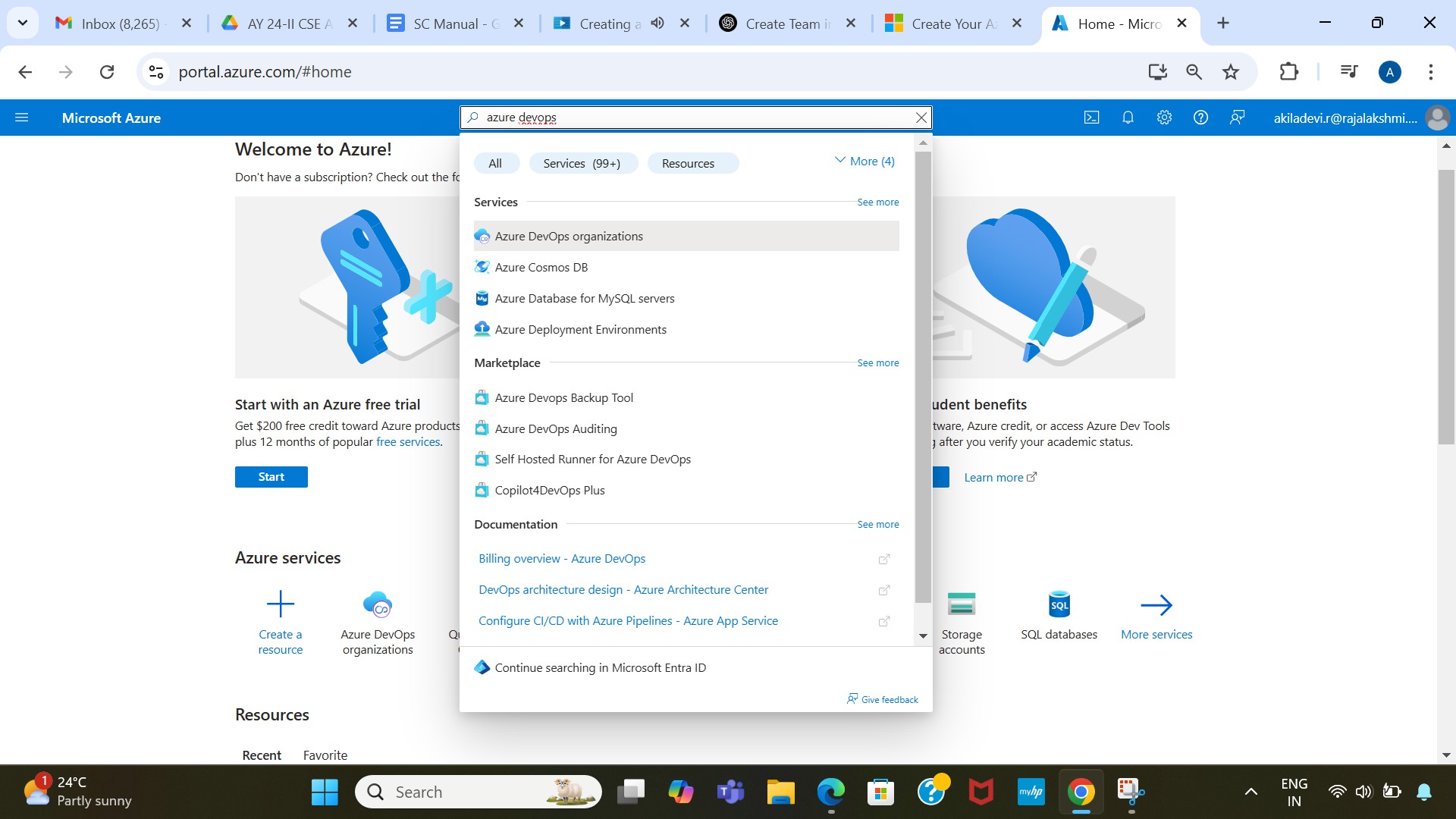
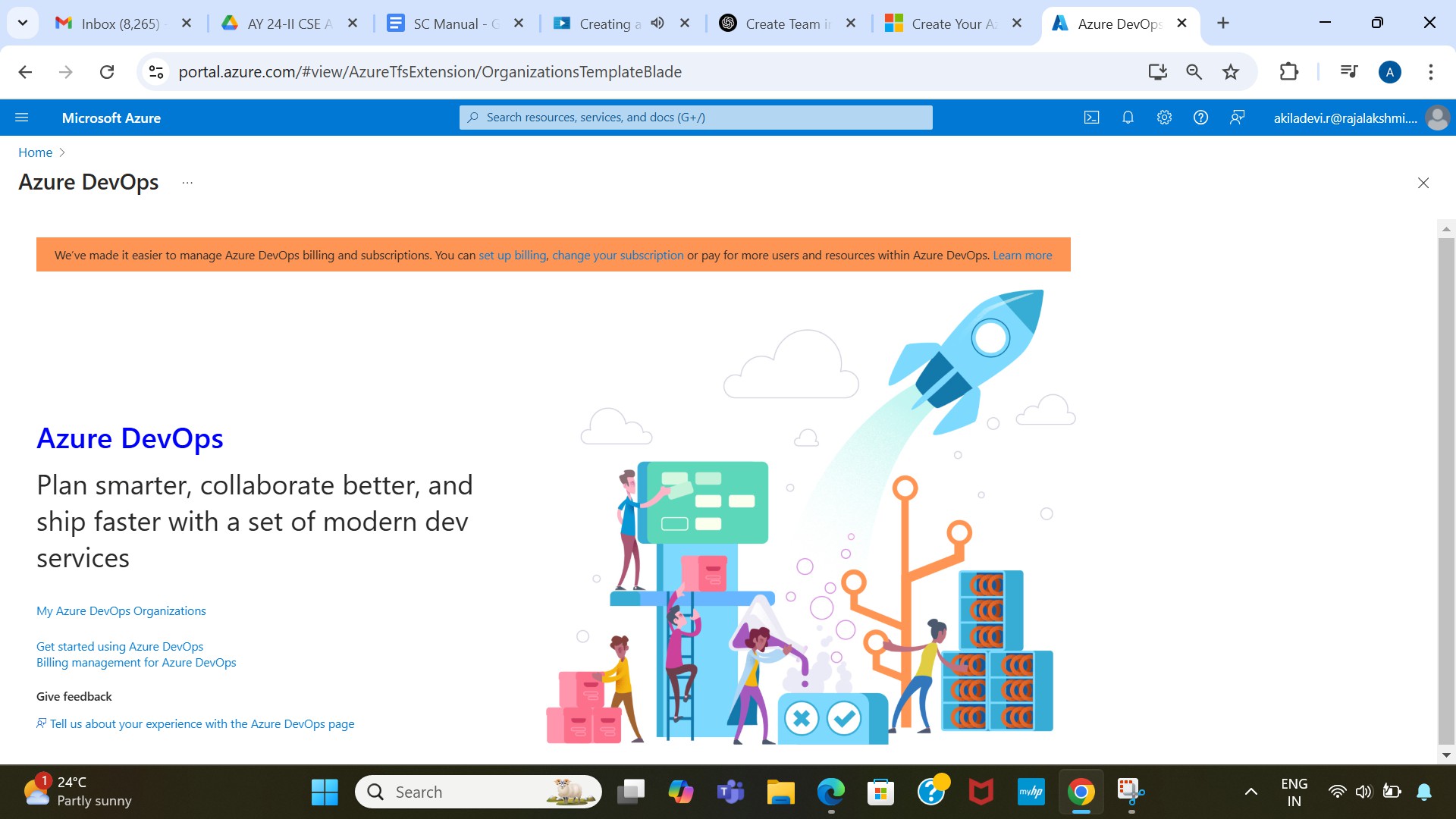
1. Open your web browser and go to the Azure website: [*https://azure.microsoft.com/en-in*](https://azure.microsoft.com/en-in)Sign in using your Microsoft account credentials. If you don't have an account, you'll need to create one.
2. *​* If you don’t have a Microsoft account, you can sign up for

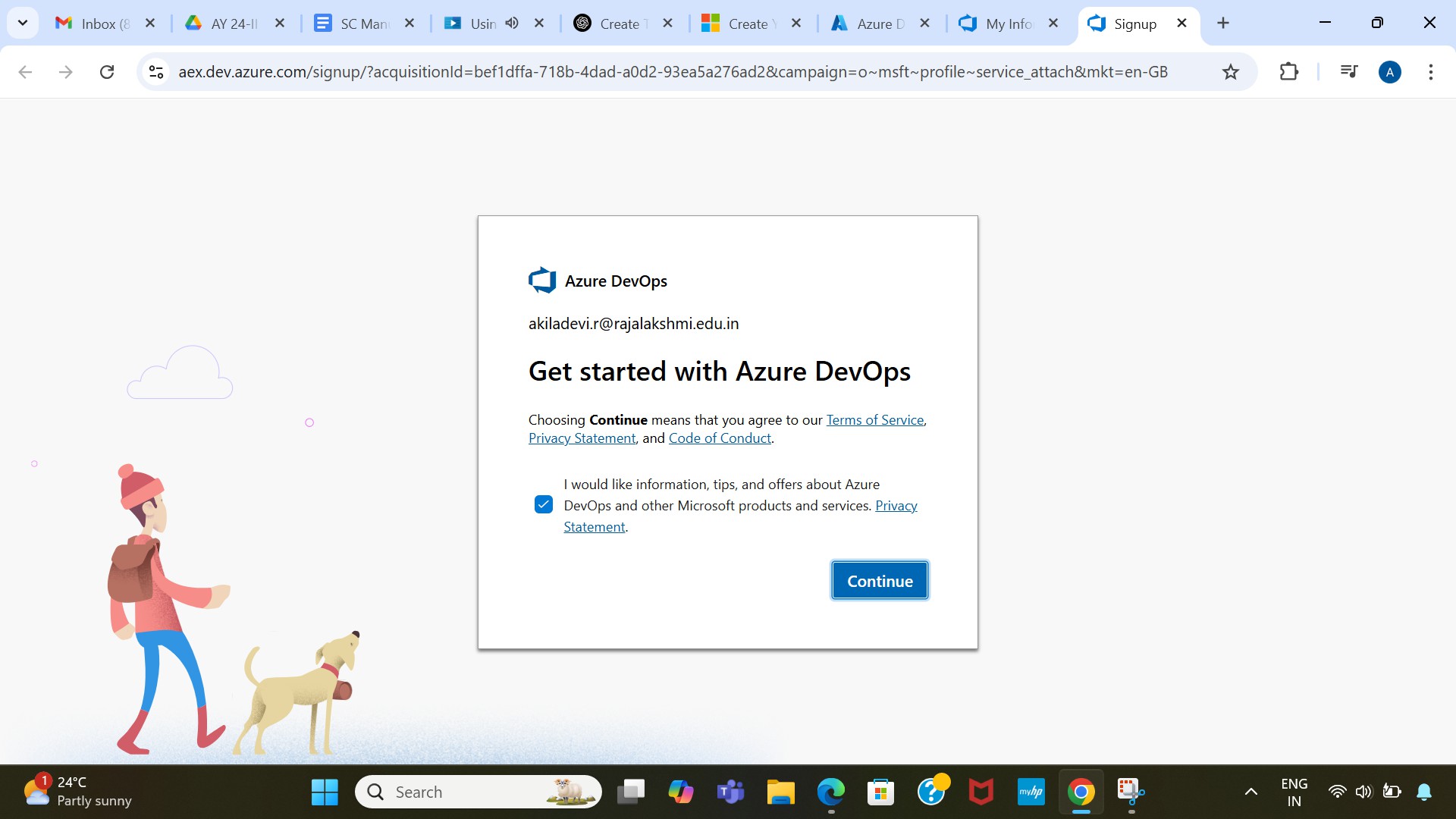
[*https://signup.live.com/?lic=1*](https://signup.live.com/?lic=1)

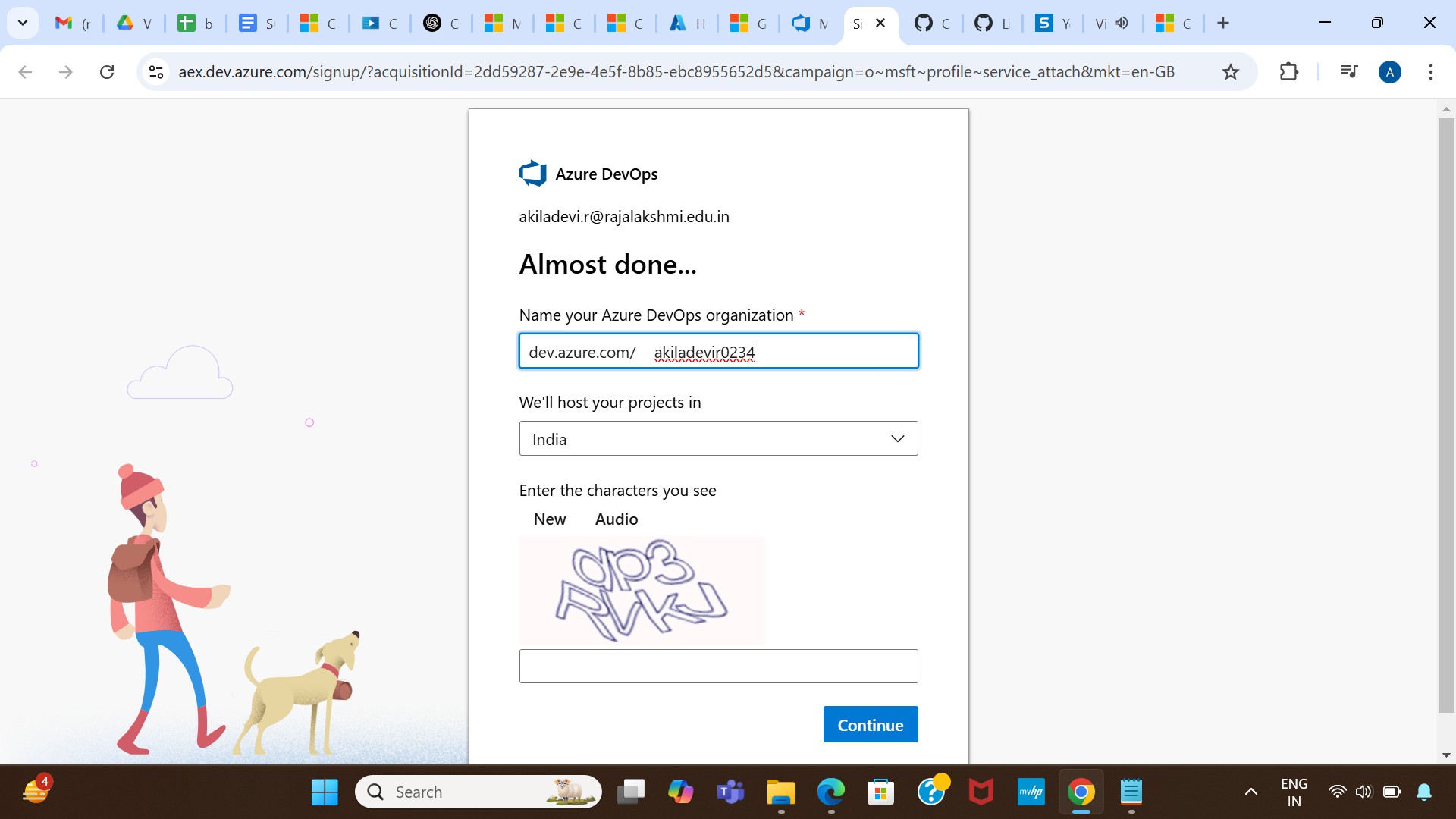
**

1. Azure home page



1. Open DevOps environment in the Azure platform by typing Azure DevOps Organizations in the search bar.
2. Click on the My Azure DevOps Organization link and create an organization and you should be taken to the Azure DevOps Organization Home page.





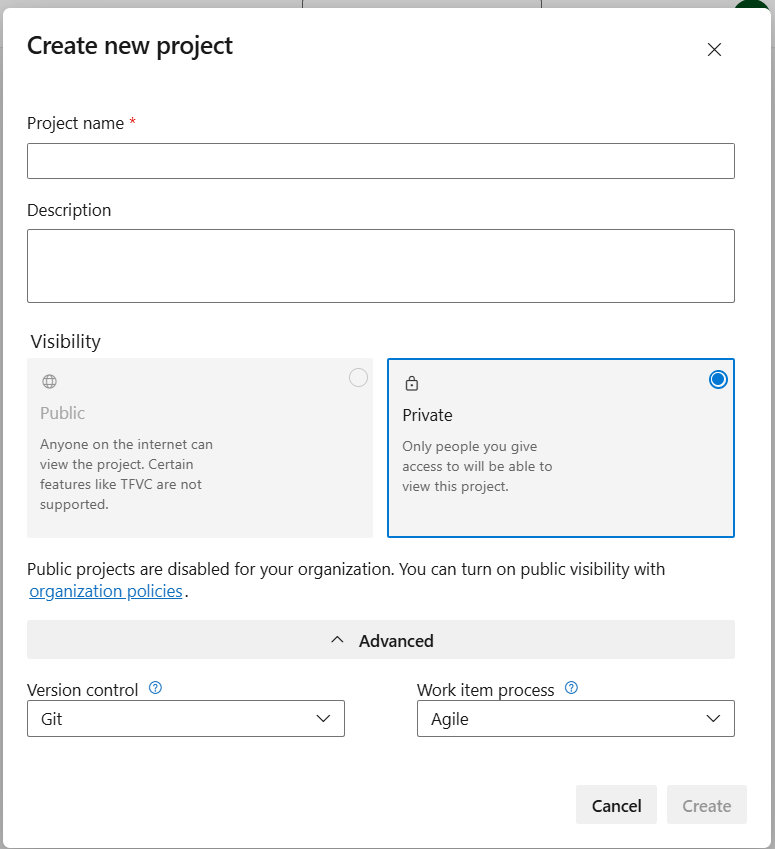
1. Create the First Project in Your Organization

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.

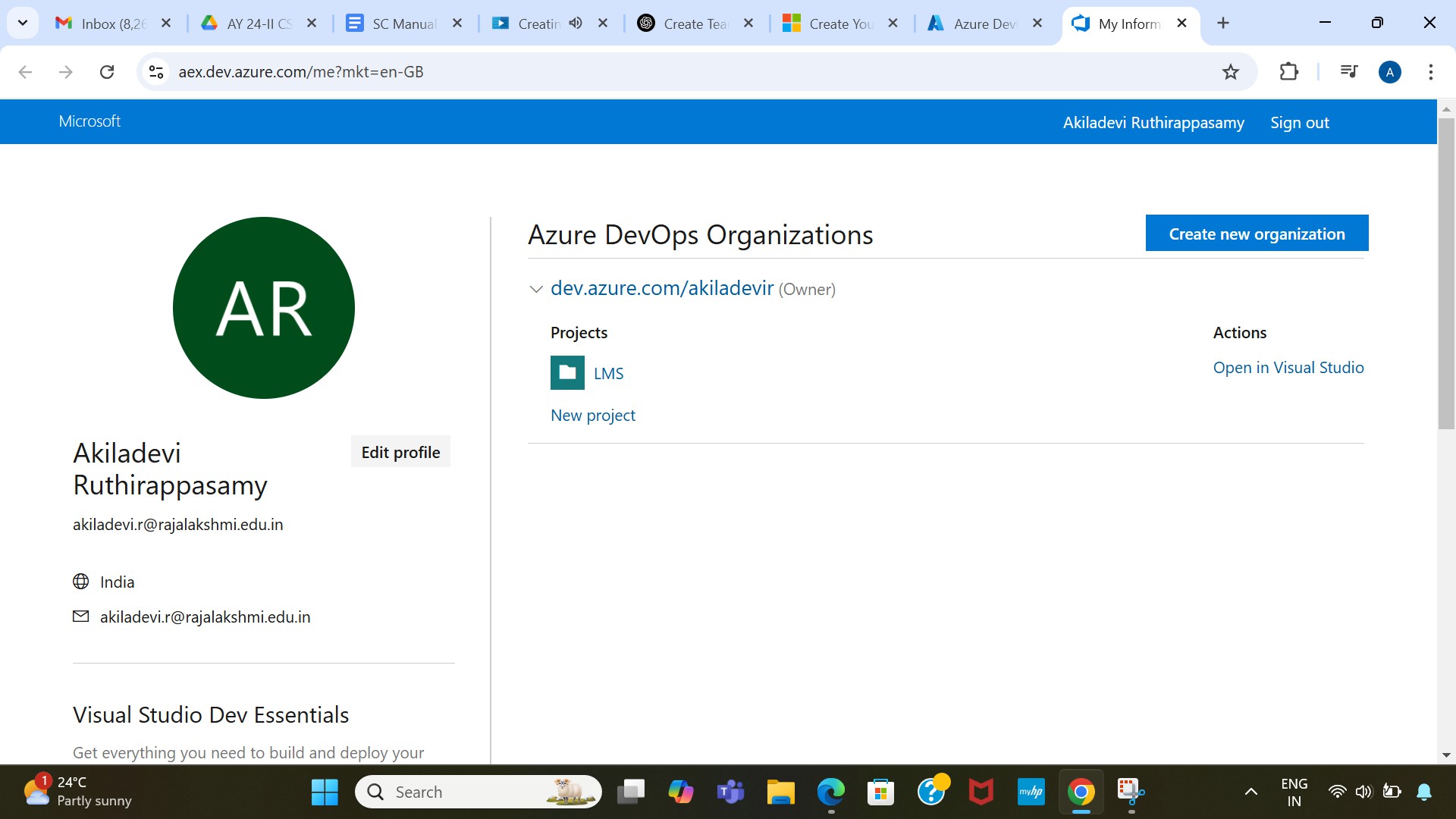
* 1. On the organization’s **Home page**, click on the **New Project** button.
  2. Enter the project name, description, and visibility options:
     + **Name**: Choose a name for the project (e.g., LMS).
     + **Description**: Optionally, add a description to provide more context about the project.
     + **Visibility**: Choose whether you want the project to be **Private**

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

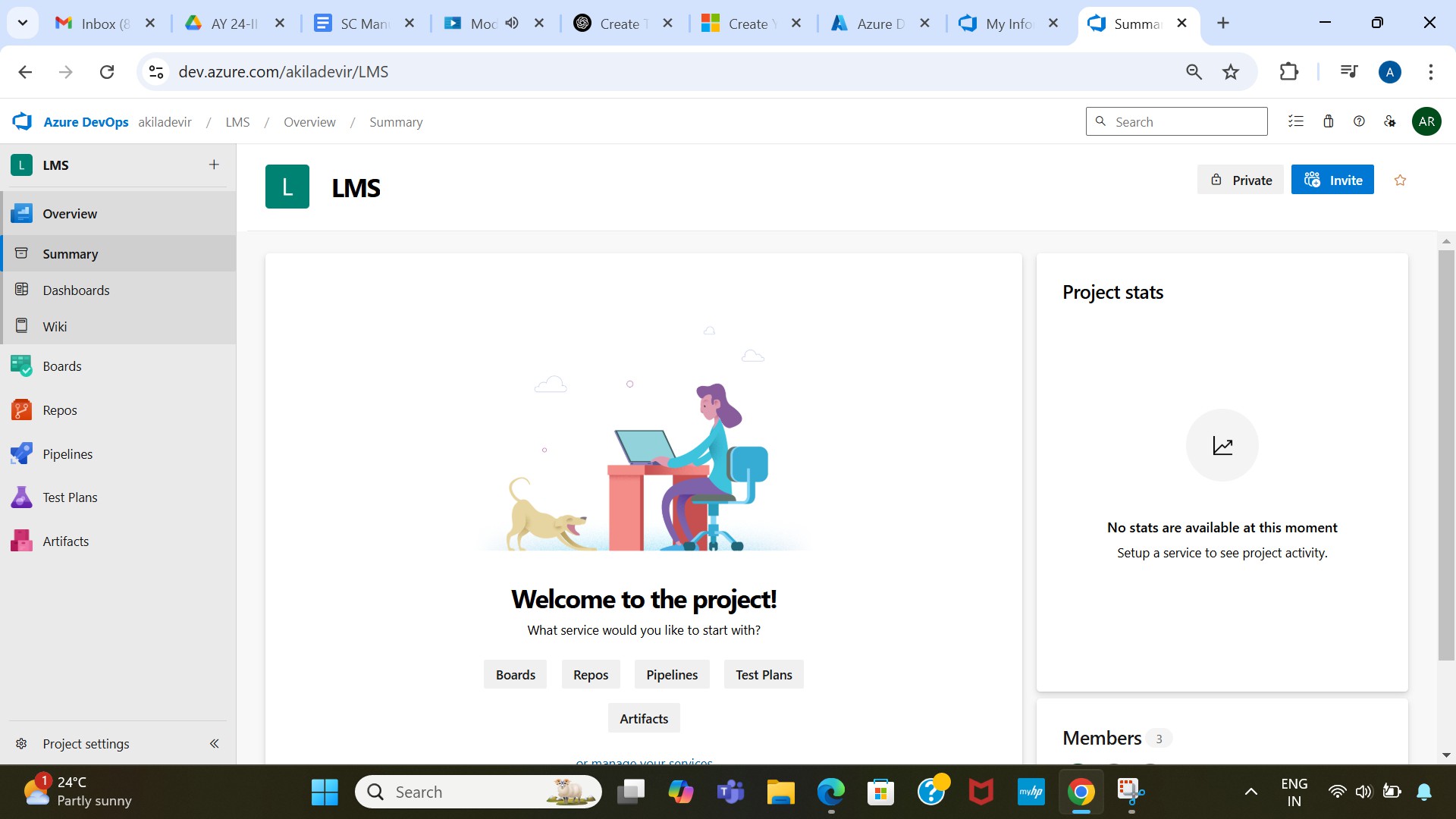
* 1. Once you’ve filled out the details, click **Create** to set up your first project.



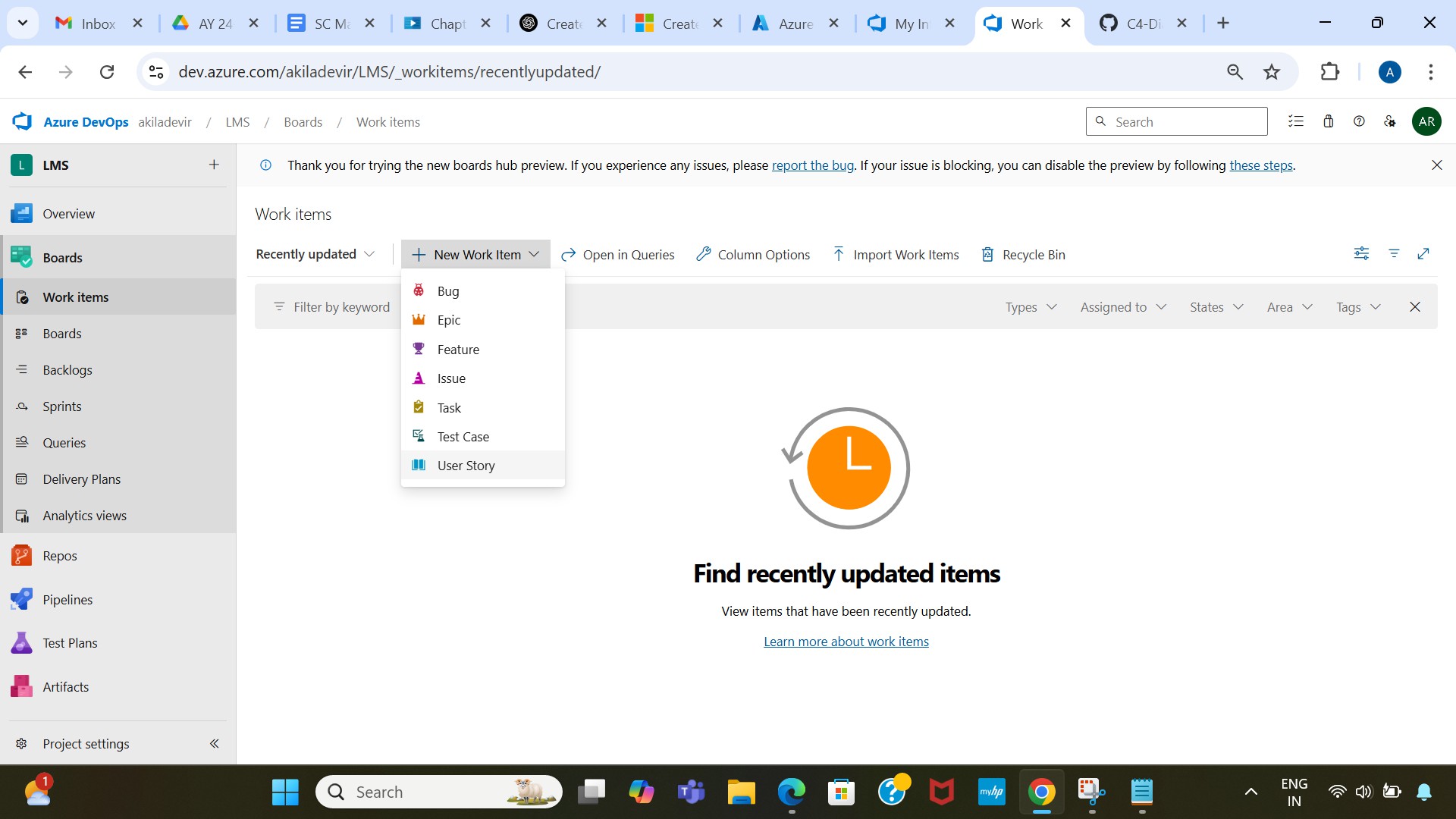
1. Once logged in, ensure you are in the correct organization. If you're part of multiple organizations, you can switch between them from the top left corner (next to your user profile). Click on the Organization name, and you should be taken to the Azure DevOps Organization Home page.



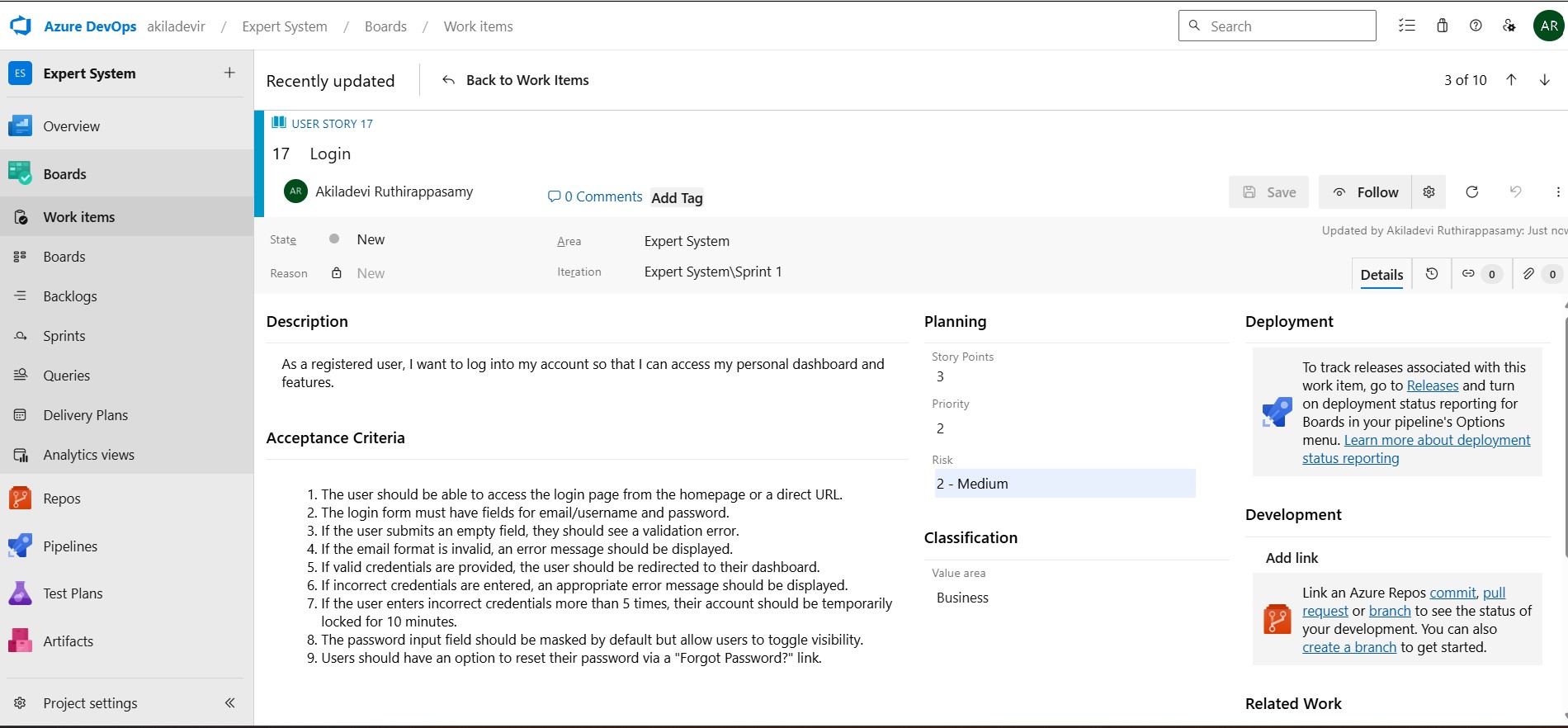
1. Project dashboard



1. To manage user stories
2. 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.
3. 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.



1. Fill in User Story Details



## Result:

The user story was written successfully.

# EX NO: 5

## Aim:

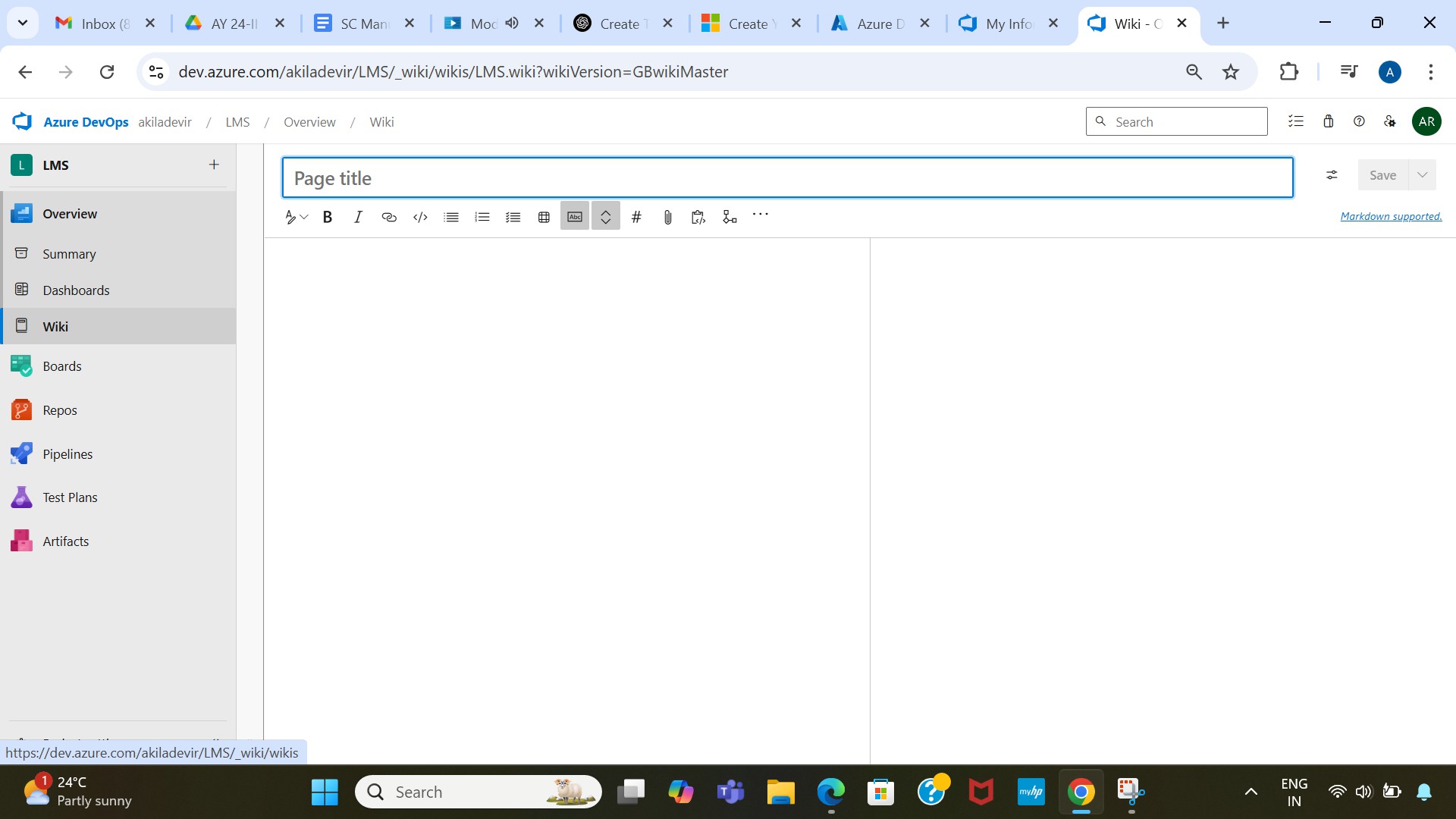
# SEQUENCE DIAGRAM

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**

%% Participants

participant USER

participant BLOG-POST

participant COMMENT

participant CATEGORY

participant ADMIN

participant NOTIFICATION

participant TAG

participant DRAFT

participant MEDIA

participant ANALYTICS

%% Main interactions

USER ->> BLOG-POST : creates

USER ->> DRAFT : saves as draft

USER ->> BLOG-POST : publishes

USER ->> BLOG-POST : edits

USER ->> BLOG-POST : deletes

USER ->> COMMENT : writes

USER ->> COMMENT : edits

USER ->> COMMENT : deletes

USER ->> CATEGORY : browses

USER ->> TAG : browses

USER ->> MEDIA : uploads images

USER ->> MEDIA : embeds videos

USER ->> ANALYTICS : views stats

USER ->> NOTIFICATION : receives alerts

%% Admin interactions

ADMIN ->> BLOG-POST : manages

ADMIN ->> COMMENT : moderates

ADMIN ->> USER : bans

ADMIN ->> USER : warns

ADMIN ->> CATEGORY : manages

ADMIN ->> TAG : manages

ADMIN ->> MEDIA : reviews

ADMIN ->> ANALYTICS : views site stats

ADMIN ->> NOTIFICATION : sends alerts

%% Post interactions

BLOG-POST ->> CATEGORY : assigned-to

BLOG-POST ->> TAG : tagged-with

BLOG-POST ->> MEDIA : contains

BLOG-POST ->> COMMENT : receives

BLOG-POST ->> USER : owned-by

BLOG-POST ->> DRAFT : converted-from

%% Comment interactions

COMMENT ->> BLOG-POST : linked-to

COMMENT ->> USER : posted-by

COMMENT ->> USER : replies-to

%% Notification interactions

NOTIFICATION ->> USER : notifies

NOTIFICATION ->> ADMIN : alerts

%% Analytics interactions

ANALYTICS ->> BLOG-POST : tracks views

ANALYTICS ->> COMMENT : tracks interactions

ANALYTICS ->> USER : aggregates activity

%% Loops and conditions

loop Frequent publishing

USER ->> BLOG-POST : publishes new post

end

alt New user registers

USER ->> ADMIN : requests approval

ADMIN ->> USER : approves/rejects

else Guest visits

USER ->> BLOG-POST : reads

end

%% Errors

USER --x BLOG-POST : fails to publish

USER --x COMMENT : fails to post

Explanation

**Creating a Blog Post**

1. **User ->> UI: Click "Create Post"**  
   The user clicks a button or selects an option in the UI to start creating a new blog post.
2. **UI ->> UI: Display "Create Blog Post Form"**  
   The UI updates to show a form where the user can enter the blog post’s details.
3. **User ->> UI: Enter post details (title, content, tags, category, media, etc.)**  
   The user fills out the form with the post’s title, main content, assigns categories and tags, and optionally uploads media like images or videos.
4. **User ->> UI: Click "Publish" or "Save as Draft"**  
   The user decides to either publish the post right away or save it as a draft.
5. **UI ->> BlogPostManager: createPost(postData)**  
   The UI sends the blog post data (title, content, tags, etc.) to the backend BlogPostManager for processing.
6. **BlogPostManager ->> BlogPost: BlogPost(postData)**  
   The BlogPostManager creates a new instance of a BlogPost with the submitted data.
7. **BlogPostManager ->> Database/Repository: savePost(post)**  
   The newly created post is stored in the database or another storage system.
8. **BlogPostManager ->> UI: Return success/created post details**  
   The BlogPostManager sends a confirmation and the details of the created post back to the UI.
9. **UI ->> UI: Display updated blog list/message**  
   The UI updates to show the new post in the list or displays a success message.

**Viewing the Blog Post List**

1. **User ->> UI: Navigate to "Blog List"**  
   The user goes to the section that shows all blog posts (like selecting “All Posts” or a menu item).
2. **UI ->> BlogPostManager: getPosts()**  
   The UI asks the BlogPostManager for the current list of blog posts.
3. **BlogPostManager ->> Database/Repository: fetchAllPosts()**  
   The BlogPostManager retrieves all existing posts from the database.
4. **BlogPostManager ->> UI: Return List<BlogPost>**  
   The list of blog posts is sent back to the UI.
5. **UI ->> UI: Display blog list**  
   The UI shows all the blog posts in a list, card view, or any suitable layout.

**Viewing a Single Blog Post**

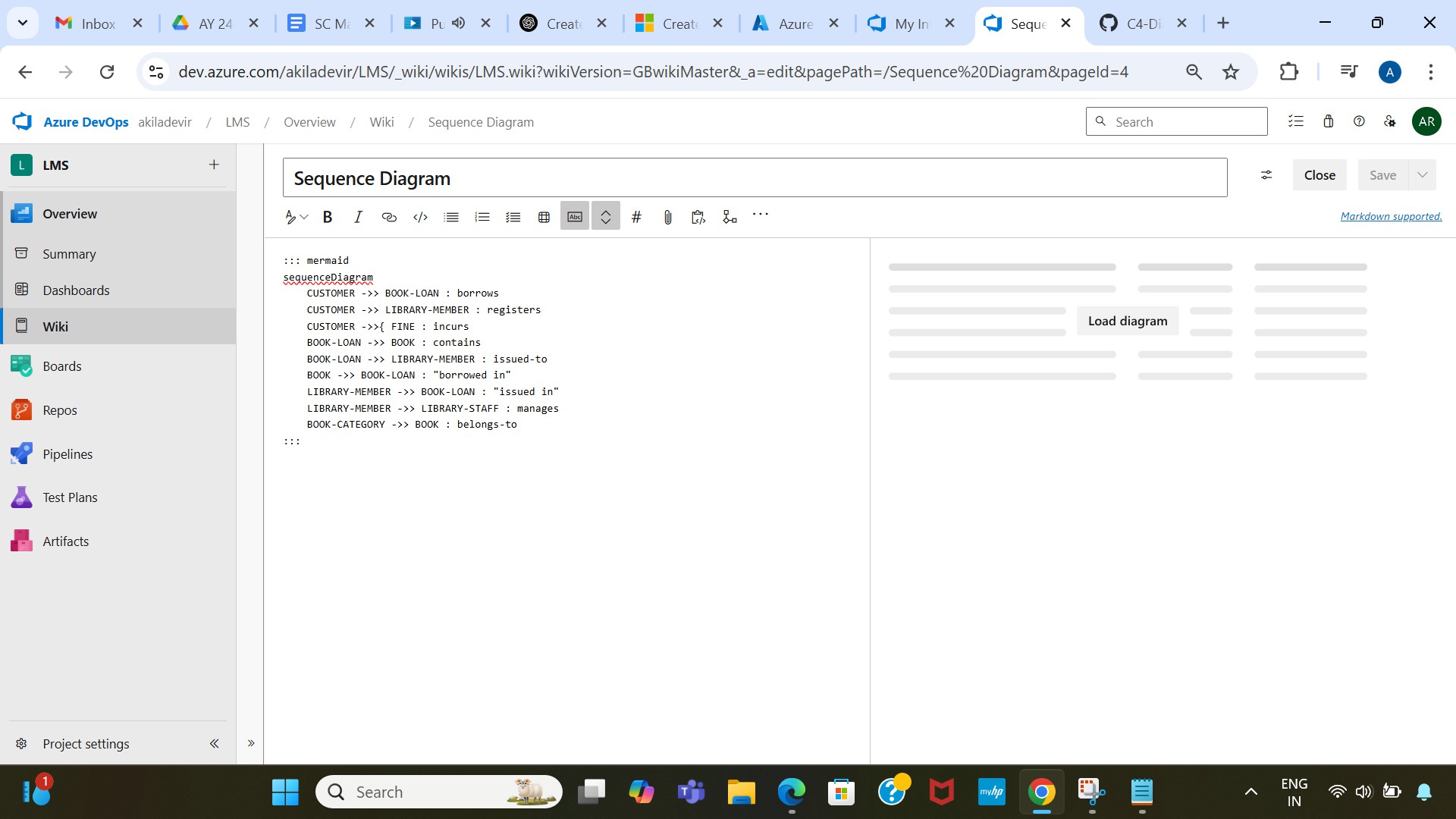
1. **User ->> UI: Click on a blog post**  
   The user selects a specific blog post to view its details.
2. **UI ->> BlogPostManager: getPost(postId)**  
   The UI requests the full content of that blog post by its ID.
3. **BlogPostManager ->> UI: Return BlogPost details**  
   The blog post details (title, content, media, comments, etc.) are sent to the UI.
4. **UI ->> UI: Display blog post details**  
   The UI shows the complete post to the user.

**Adding a Comment**

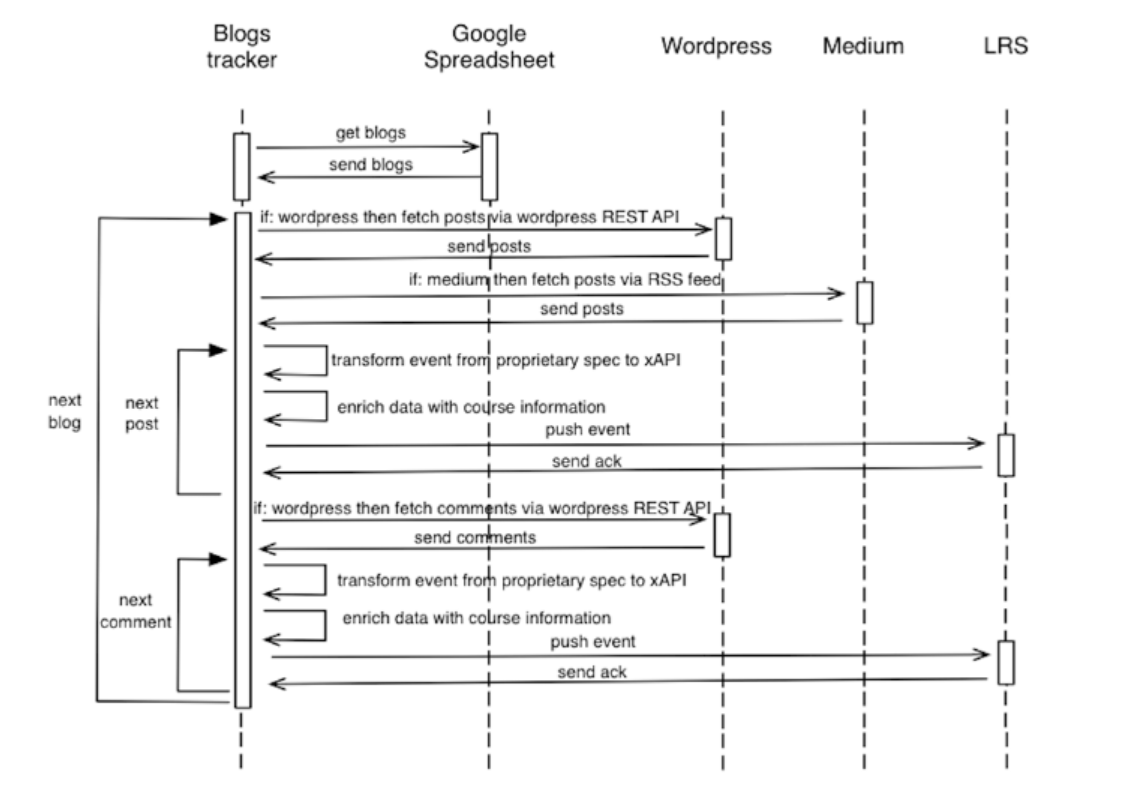
1. **User ->> UI: Enter comment and submit**  
   The user writes a comment and clicks to submit it.
2. **UI ->> CommentManager: addComment(commentData)**  
   The UI sends the comment data to the CommentManager for creation.
3. **CommentManager ->> Comment: Comment(commentData)**  
   A new Comment instance is created with the provided data.
4. **CommentManager ->> Database/Repository: saveComment(comment)**  
   The comment is stored in the database and linked to the post.
5. **CommentManager ->> UI: Return success/new comment**  
   The new comment is sent back to the UI.
6. **UI ->> UI: Update comment list**  
   The UI displays the new comment under the blog post.

**Navigating the Wiki or Additional Pages**

1. **User ->> UI: Click wiki menu**  
   The user clicks on a “Wiki” or “Help” menu option.
2. **UI ->> UI: Display wiki pages**  
   The UI shows the available wiki/help pages.
3. **User ->> UI: Select page**  
   The user selects a specific wiki/help page to view.
4. **UI ->> WikiManager: getWikiPage(pageId)**  
   The UI requests the selected wiki/help page’s content.
5. **WikiManager ->> UI: Return wiki page content**  
   The content of the wiki/help page is returned to the UI.
6. **UI ->> UI: Display wiki page**  
   The UI displays the selected wiki/help page to the user.



1. click wiki menu and select the page



## Result:

The sequence diagram was drawn successfully.

**EX NO. 6**

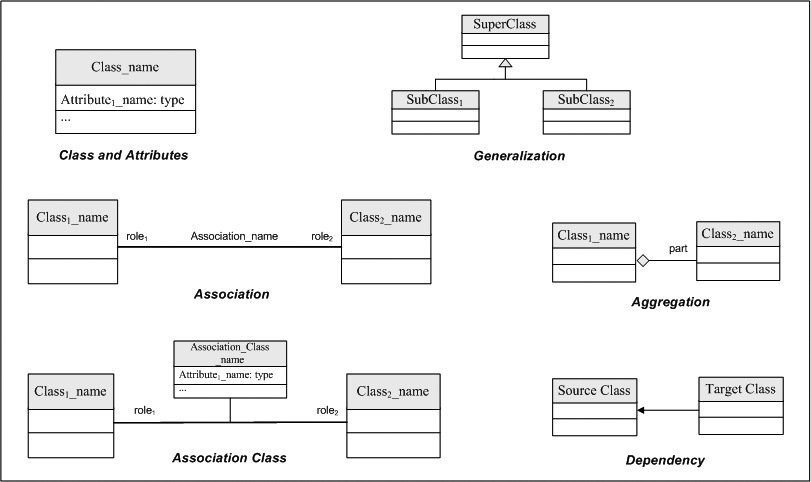
**CLASS DIAGRAM**

**AIM :-**

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

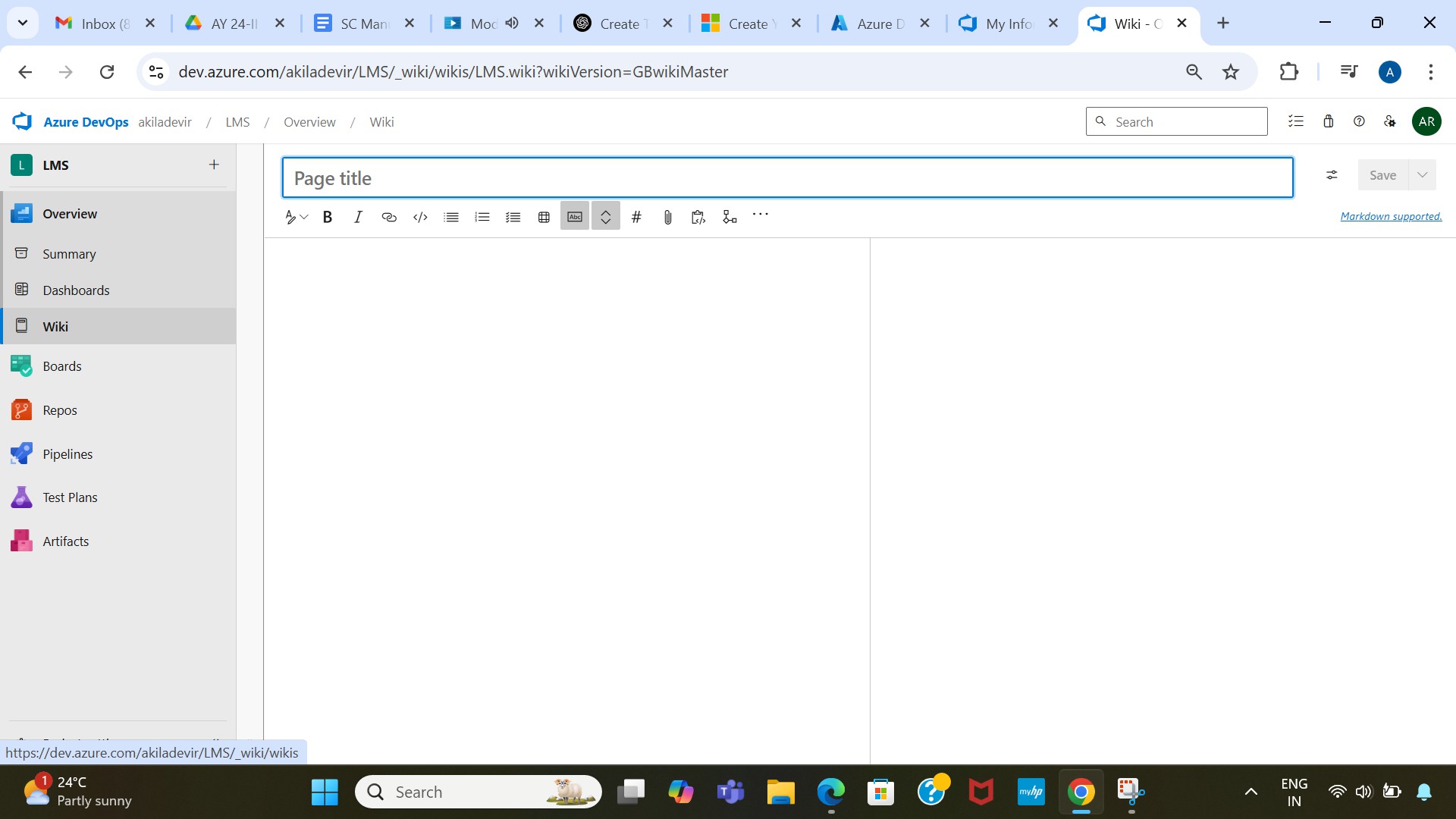
# THEORY

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



Notations in class diagram

Procedure:

1. Open a project in Azure DevOps Organisations.
2. To design select wiki from menu
3. Write code for drawing class diagram and save the code

classDiagram

%% Core classes

class User {

+String username

+String email

+String password

+createPost()

+editPost()

+deletePost()

+writeComment()

+editComment()

+deleteComment()

+viewAnalytics()

}

class Admin {

+String username

+String email

+String password

+managePosts()

+moderateComments()

+banUser()

+warnUser()

+manageCategories()

+manageTags()

+viewSiteAnalytics()

}

class BlogPost {

+String title

+String content

+DateTime createdAt

+DateTime updatedAt

+publish()

+edit()

+delete()

}

class Comment {

+String content

+DateTime createdAt

+DateTime updatedAt

+edit()

+delete()

}

class Category {

+String name

+String description

}

class Tag {

+String name

}

class Draft {

+String title

+String content

+DateTime savedAt

}

class Media {

+String fileName

+String fileType

+String url

}

class Notification {

+String message

+DateTime sentAt

}

class Analytics {

+int views

+int comments

+generateReport()

}

%% Relationships

User "1" --> "\*" BlogPost : creates

User "1" --> "\*" Comment : writes

User "1" --> "\*" Draft : saves

User "1" --> "\*" Media : uploads

User "1" --> "\*" Notification : receives

User "1" --> "\*" Analytics : views

Admin "1" --> "\*" BlogPost : manages

Admin "1" --> "\*" Comment : moderates

Admin "1" --> "\*" User : manages

Admin "1" --> "\*" Category : manages

Admin "1" --> "\*" Tag : manages

Admin "1" --> "\*" Media : reviews

Admin "1" --> "\*" Notification : sends

BlogPost "1" --> "\*" Comment : receives

BlogPost "1" --> "1" Category : belongs-to

BlogPost "\*" --> "\*" Tag : tagged-with

BlogPost "1" --> "\*" Media : contains

BlogPost "1" --> "1" User : owned-by

BlogPost "1" --> "0..1" Draft : converted-from

Comment "1" --> "1" User : posted-by

Comment "1" --> "1" BlogPost : linked-to

Notification "1" --> "1" User : notifies

Notification "1" --> "1" Admin : alerts

Analytics "1" --> "\*" BlogPost : tracks

Analytics "1" --> "\*" Comment : tracks

Analytics "1" --> "\*" User : aggregates

**Relationship Types**

Type Description

<| Inheritance

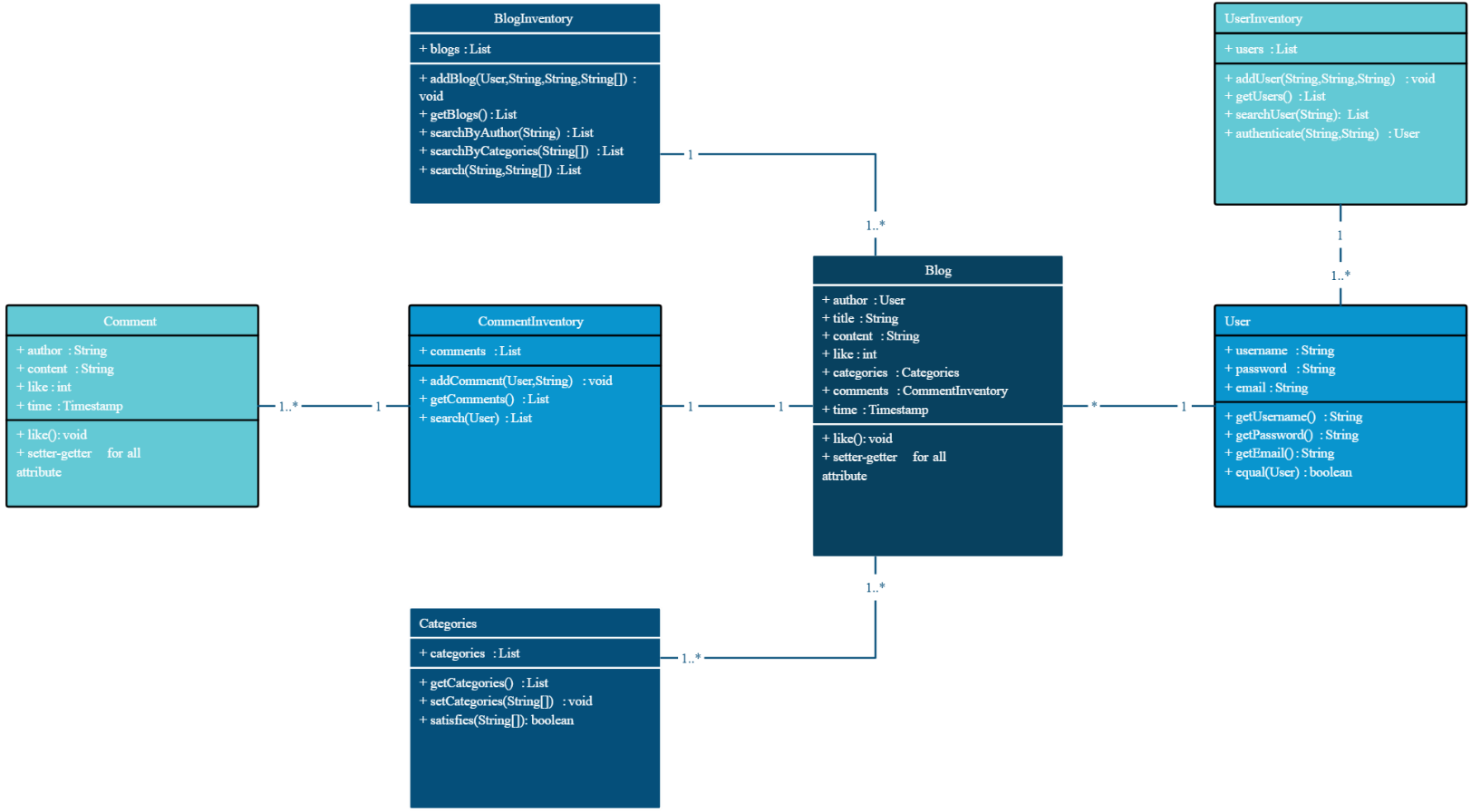
\\* Composition

o Aggregation

> Association

< Association

|> Realization



Visit : https://mermaid.js.org/syntax/classDiagram.html

## Result:

The use case diagram was designed successfully.

# EX NO: 7

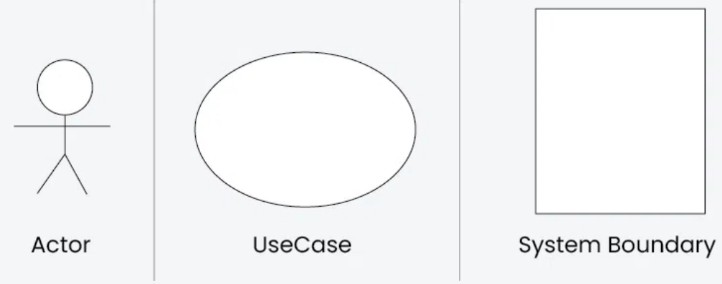
**Aim**:

# USECASE DIAGRAM

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.

## Result:

The use case diagram was designed successfully

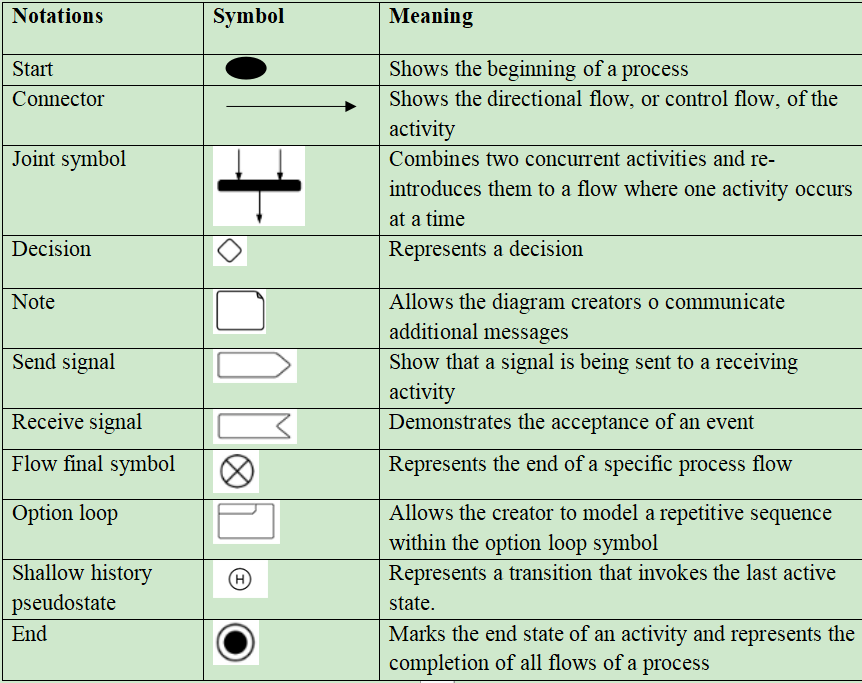
**EX NO. 8**

**ACTIVITY DIAGRAM**

**AIM :-**

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

# THEORY

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

Procedure

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

## Result:

The activity diagram was designed successfully

# EX NO. 9

**ARCHITECTURE DIAGRAM**

## Aim:

Steps to draw the Architecture Diagram using draw.io.

## Theory:

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



Procedure

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

## Result:

The architecture diagram was designed successfully

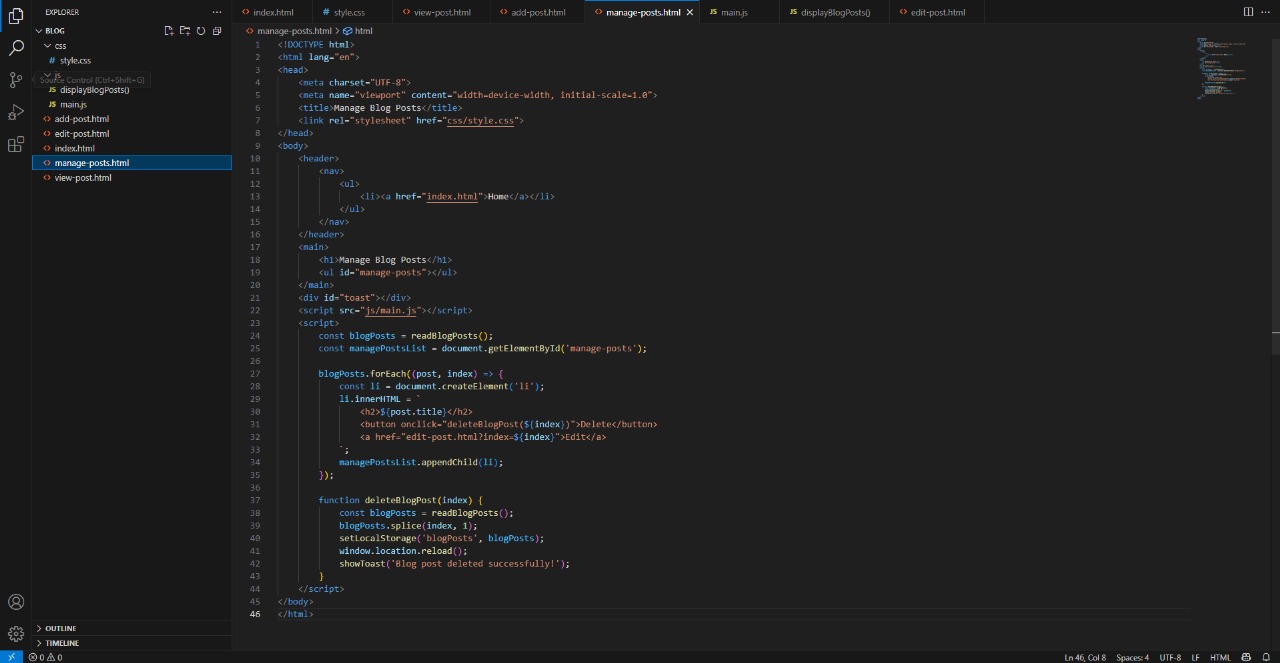
# EX NO. 10

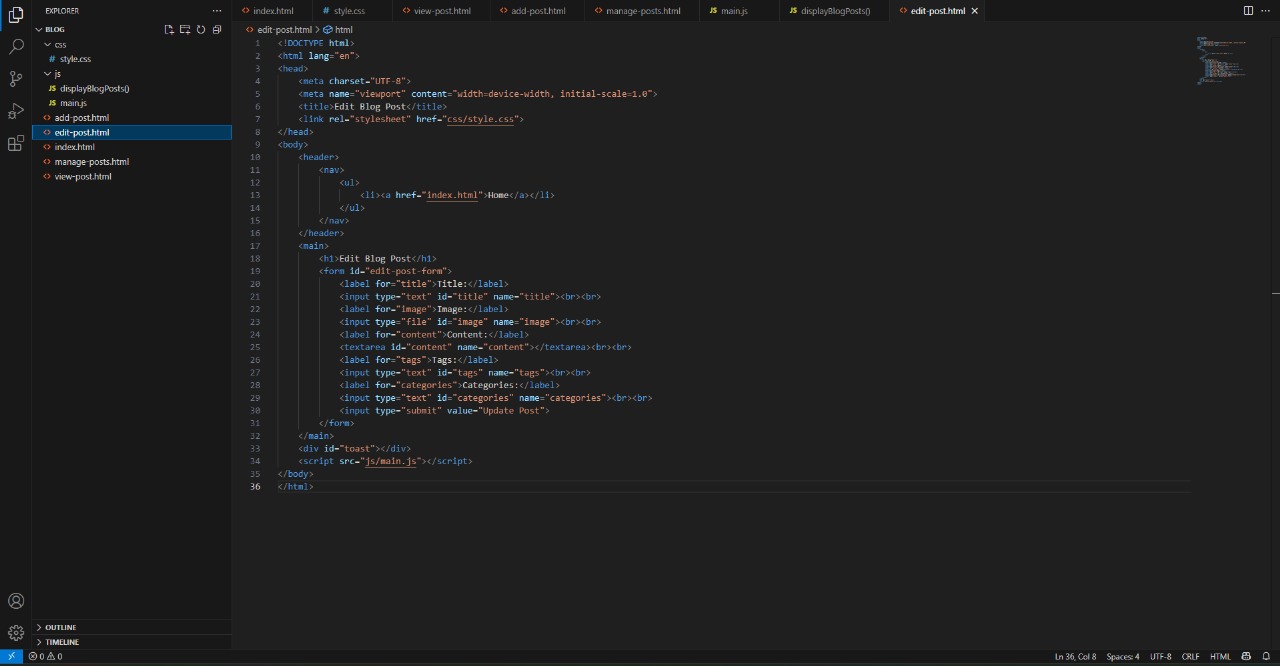
## Aim:

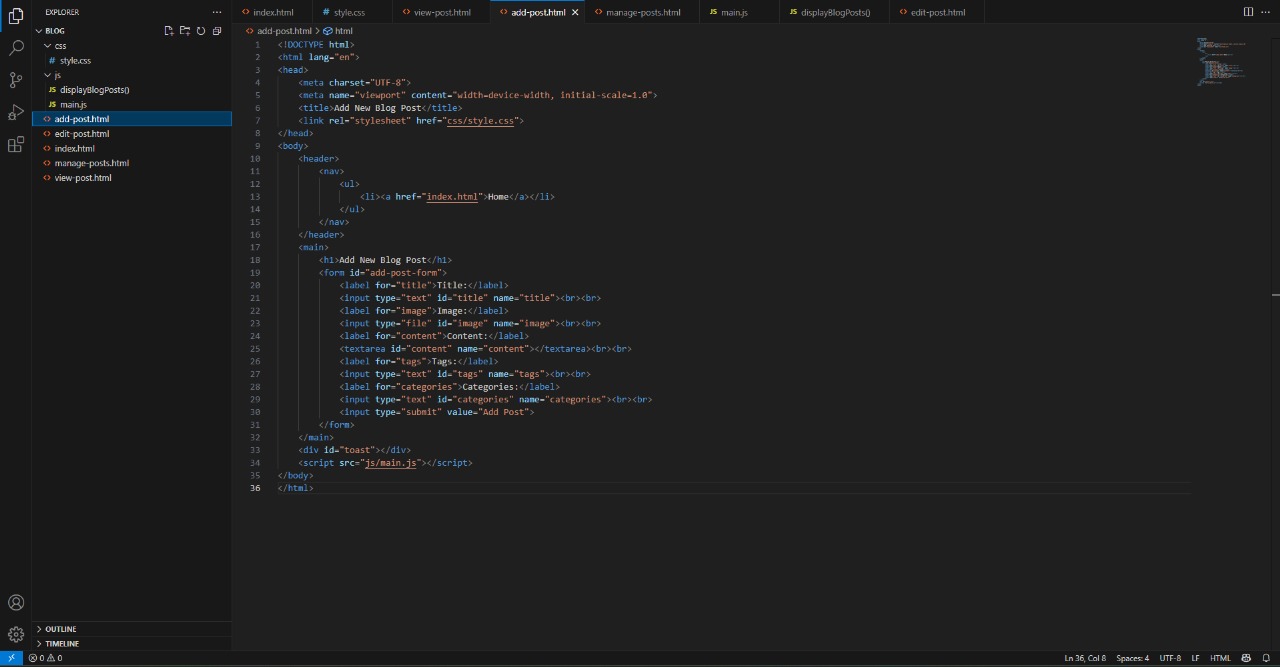
# USER INTERFACE

Design User Interface for the given project

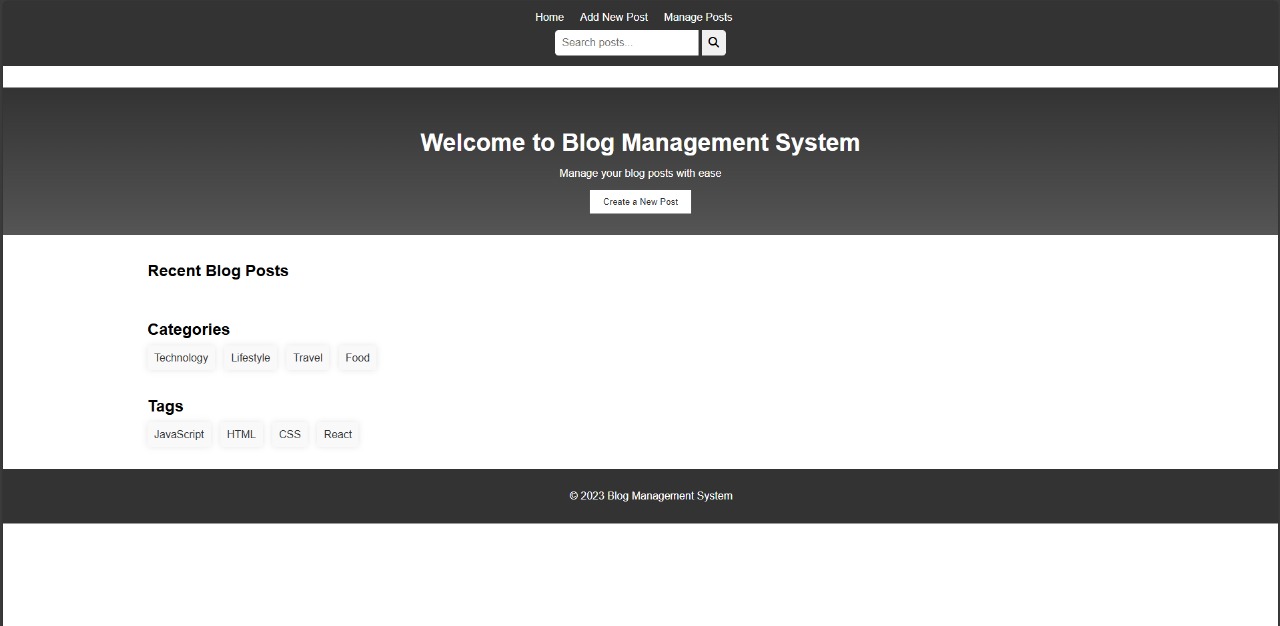
**Code:**

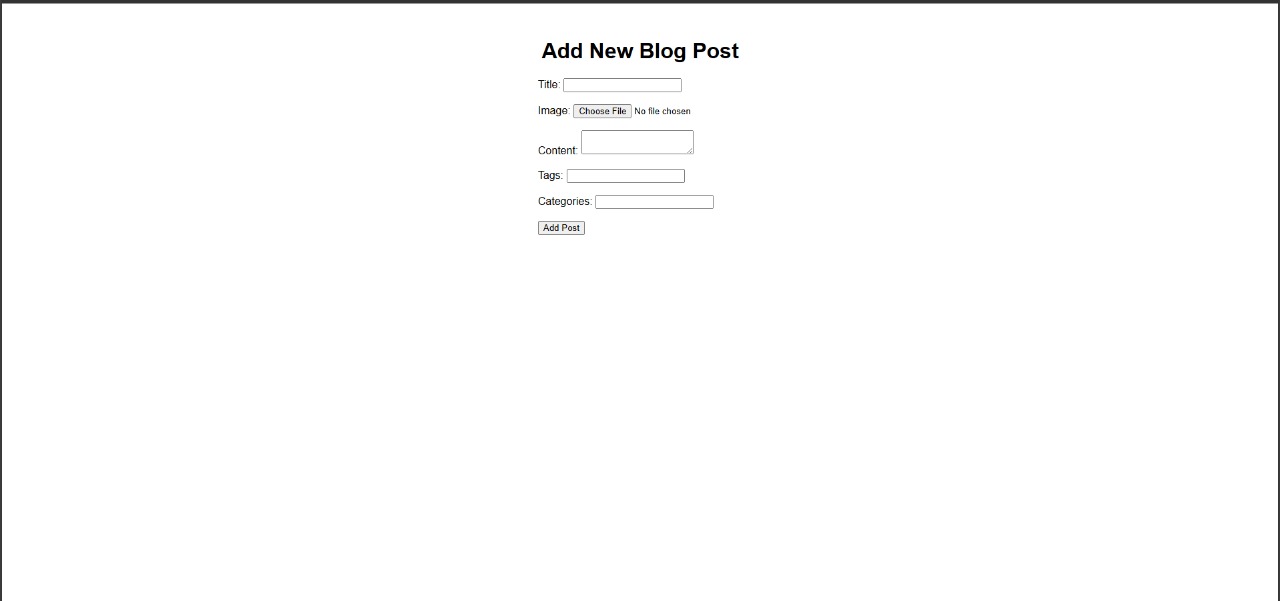






**Interface**





Result:

The UI was designed successfully.

# EX NO. 11

## Aim:

# IMPLEMENTATION

To implement the given project based on Agile Methodology.

Procedure:

Step 1: Set Up an Azure DevOps Project

* Log in to Azure DevOps.
* Click "New Project" → Enter project name → Click "Create".
* Inside the project, navigate to "Repos" to store the code.

Step 2: Add Your Web Application Code

* Navigate to Repos → Click "Clone" to get the Git URL.
* Open Visual Studio Code / Terminal and run: git clone <repo\_url>

cd <repo\_folder>

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

git add .

git commit -m "Initial commit" git push origin main

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

* Navigate to Pipelines → Click "New Pipeline".
* Select Git Repository (Azure Repos, GitHub, or Bitbucket).
* Choose Starter Pipeline or a pre-configured template for your framework.
* Modify the azure-pipelines.yml file (Example for a Node.js app):

trigger:

- main

pool:

vmImage: 'ubuntu-latest'

steps:

* task: UseNode@1 inputs:

version: '16.x'

* script: npm install

displayName: 'Install dependencies'

* script: npm run build displayName: 'Build application'
* task: PublishBuildArtifacts@1 inputs:

pathToPublish: 'dist' artifactName: 'drop'

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

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

* Go to Releases → Click "New Release Pipeline".
* Select Azure App Service or Virtual Machines (VMs) for deployment.
* Add an artifact (from the build pipeline).
* Configure deployment stages (Dev, QA, Production).
* Click "Deploy" to push your web app to Azure.

## Result

Thus the application was successfully implemented.