RAJALAKSHMI ENGINEERING COLLEGE

An AUTONOMOUS Institution

Affiliated to ANNA UNIVERSITY, Chennai

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

ACADEMIC YEAR 2024-2025

EVEN SEMESTER

CS23432 - Software Construction

Lab Manual

Name:	Mohamed Ikram K.H.D.
Year / Branch / Section:	II year – CSE – B
Register No:	2116230701188
Semester:	IV
Academic Year:	2024 – 2025

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

CS234342 - SOFTWARE CONSTRUCTION LAB

Ex No	Date	Topic	Page No	Sign
1		Study of Azure DevOps		
2		Writing Problem Statement		
3		Agile Planning		
4		Creating User stories		
5		Designing Sequence Diagram		
6		Designing Class Diagram		
7		Designing Use Case Diagram		
8		Designing Activity Diagram		
9		Designing Architecture Diagram		
10		Design User Interface		
11		Implementation – Design a Web Page based on Scrum Methodology		
12		Testing and Deployment		

Course Outcomes (COs)

Course Name: Software Construction
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 product
CO 5	Manage Software Projects and to understand advanced engineering concepts

CO - PO – PSO matrices of course

PO/PSO / CO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
CS23432.1	2	2	3	2	2	2	2	2	2	2	3	2	1	3	-
CS23432.2	2	3	1	2	2	1	-	1	1	1	2	-	1	2	-
CS23432.3	2	2	1	1	1	1	1	1	1	1	1	1	2	2	1
CS23432.4	2	2	3	2	2	2	1	0	2	2	2	1	1	2	1
CS23432.5	2	2	2	1	1	1	1	0	2	1	1	1	2	1	-
Average	2.0	2.2	2.0	1.6	1.6	1.4	1.3	1.3	1.6	1.4	1.8	1.3	1.4	2.0	1.0

Correlation levels 1, 2 or 3 are as defined below:
1: Slight (Low) 2: Moderate (Medium) 3: Substantial (High) No correlation: "-"

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:

The study was successfully completed.

Ex. No. 2	WRITING PROBLEM STATEMENT

AIM:

To prepare a **problem statement** for your given project.

Problem Statement:

Smart Bus – Intelligent Campus Transportation Management

The current college transport system lacks real-time visibility, efficient route management, and effective communication, leading to delays, uncertainty, and reduced operational reliability. A digital, integrated solution is essential to enhance mobility, safety, and user experience.

Result:

The problem statement was 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

1. Sprints to work on one specific group of tasks at a time
2. 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	CREATING USER STORIES

Aim:

To create User Stories

THEORY

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

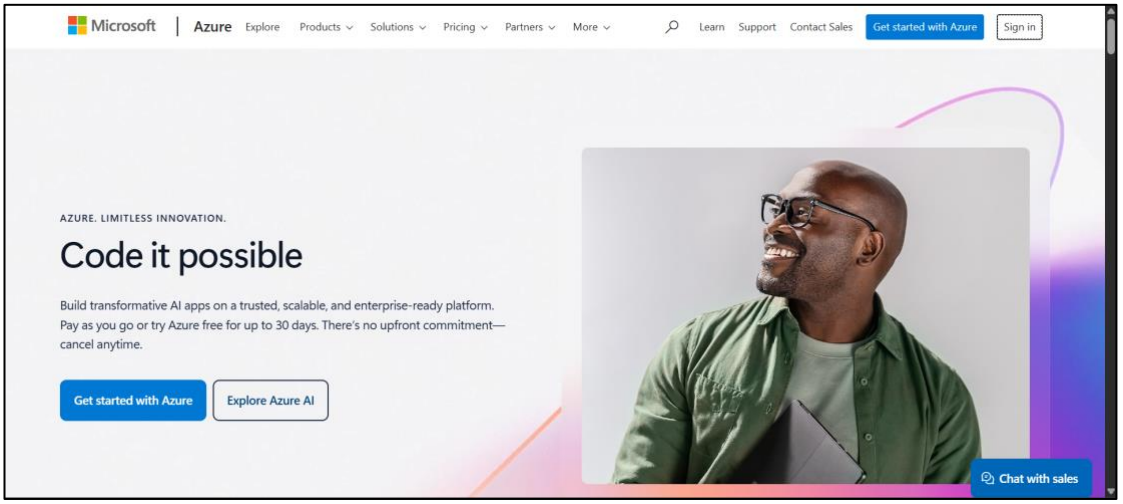
User story template:

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

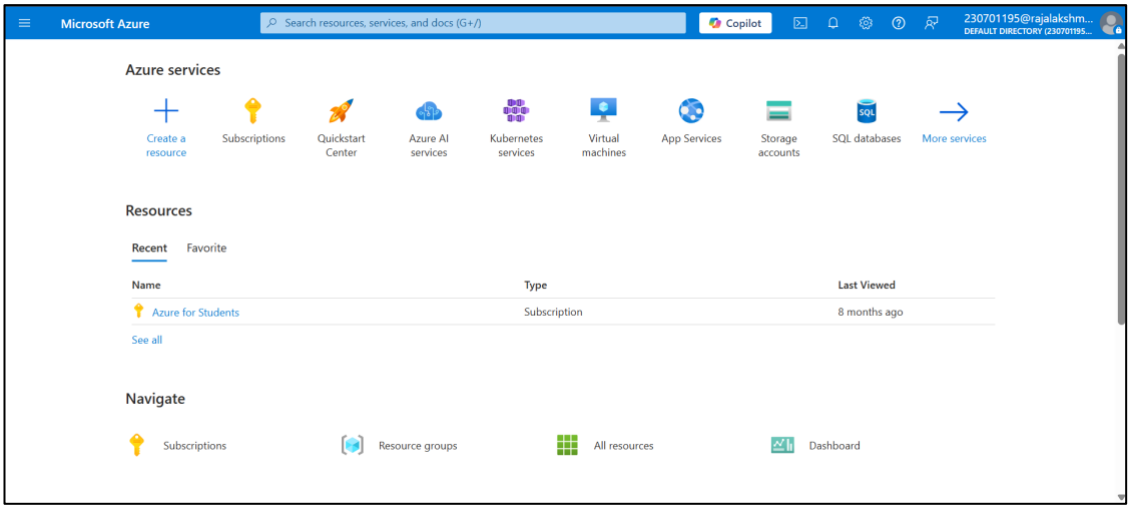
Procedure:

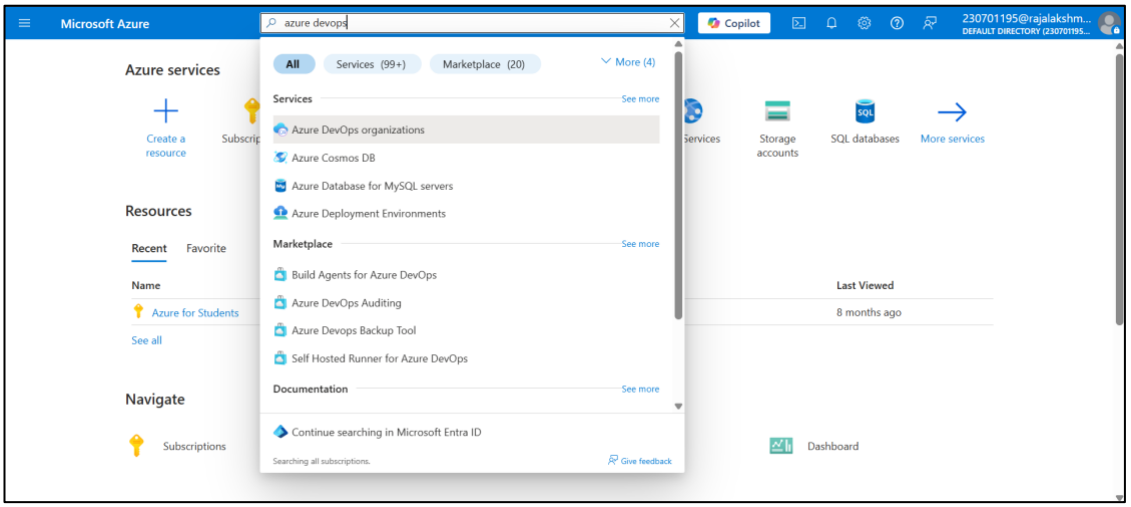
1. Open your web browser and go to the Azure website: *https://azure.microsoft.com/en-in.*
   Sign in using your Microsoft account credentials. If you don't have an account, you'll need to create one.

2. If you don't have a Microsoft account, you can sign up for *https://signup.live.com/?lic=1*

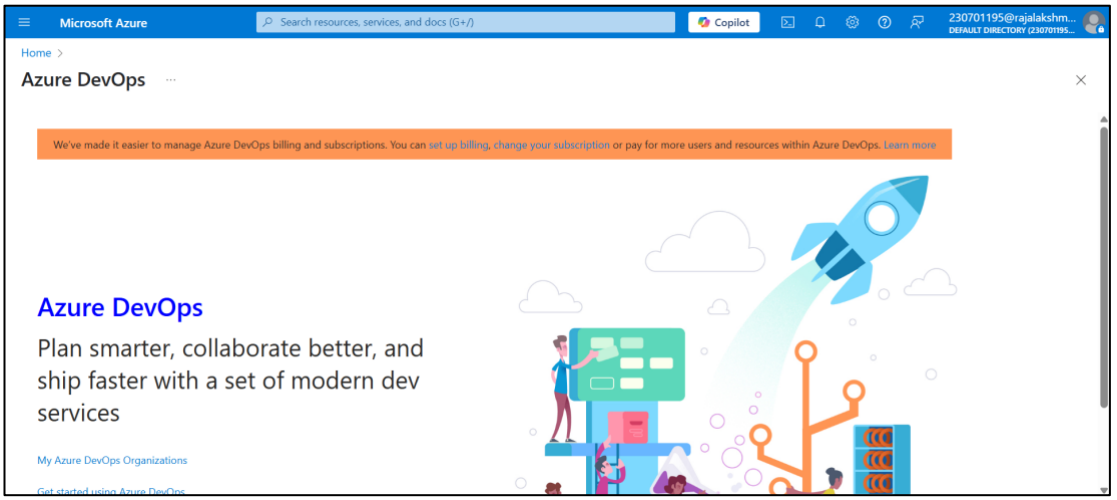


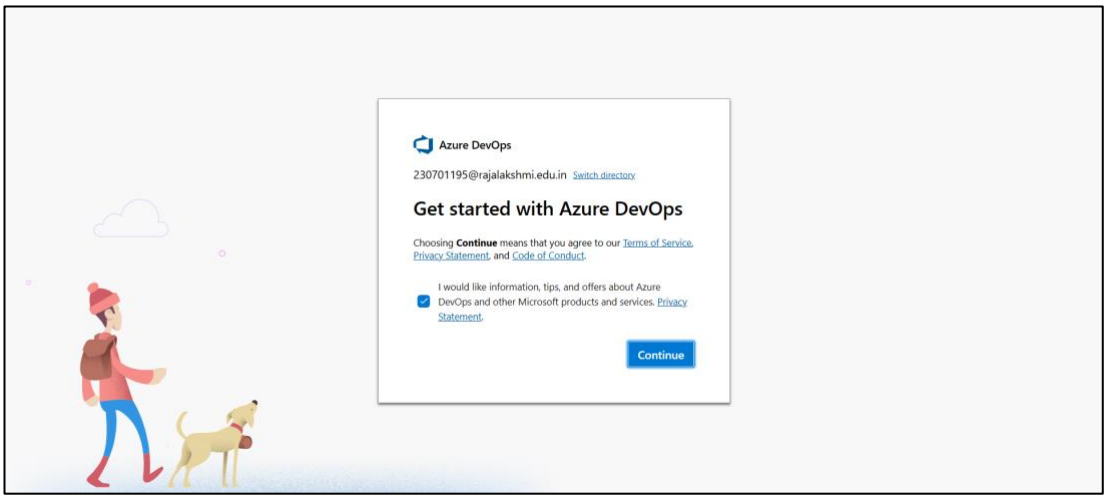
3. <u>Azure Home Page</u>

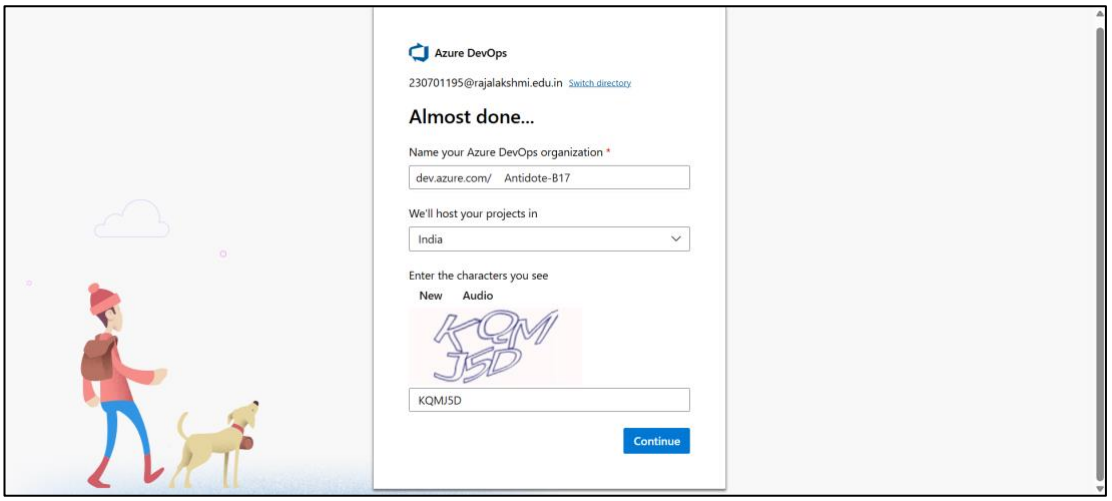


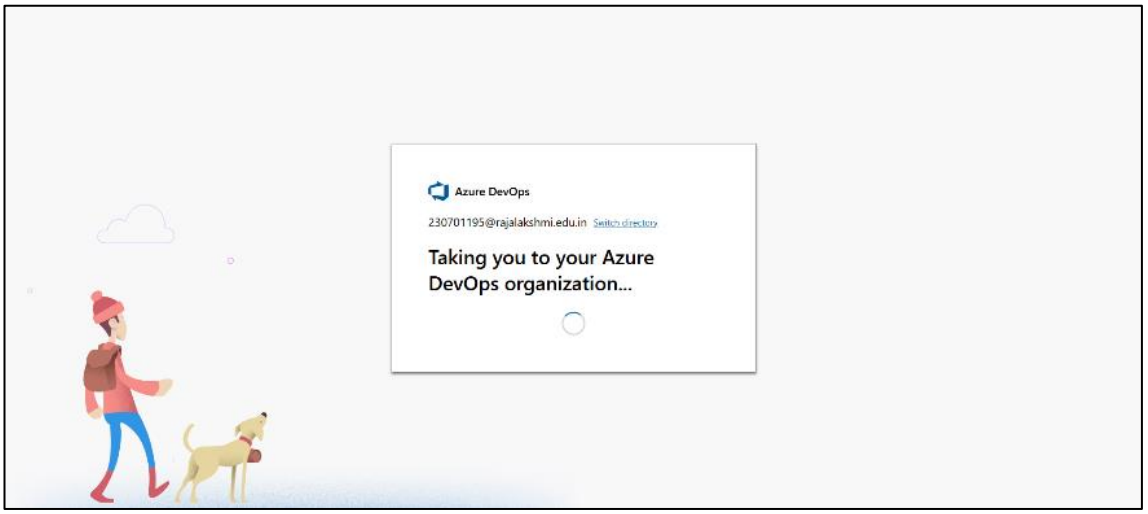


4. Open DevOps environment in the Azure platform by typing Azure DevOps Organizations in the search bar.
5. Click on the My Azure DevOps Organization link and create an organization and you should be taken to the Azure DevOps Organization Home page


Microsoft Azure
Search resources, services, and docs (G+/)
Copilot
230701195@rajalakshm...
DEFAULT DIRECTORY (230701195...
Home
Azure DevOps
We've made it easier to manage Azure DevOps billing and subscriptions. You can set up billing, change your subscription or pay for more users and resources within Azure DevOps. Learn more
Azure DevOps
Plan smarter, collaborate better, and ship faster with a set of modern dev services
My Azure DevOps Organizations


Azure DevOps
230701195@rajalakshmi.edu.in Switch directory
Get started with Azure DevOps
Choosing Continue means that you agree to our Terms of Service, Privacy Statement, and Code of Conduct.
I would like information, tips, and offers about Azure DevOps and other Microsoft products and services. Privacy Statement.
Continue


Azure DevOps
230701195@rajalakshmi.edu.in Switch directory
Almost done...
Name your Azure DevOps organization *
dev.azure.com/ Antidote-B17
We'll host your projects in
India
Enter the characters you see
New Audio
KQMJ5D
Continue



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.

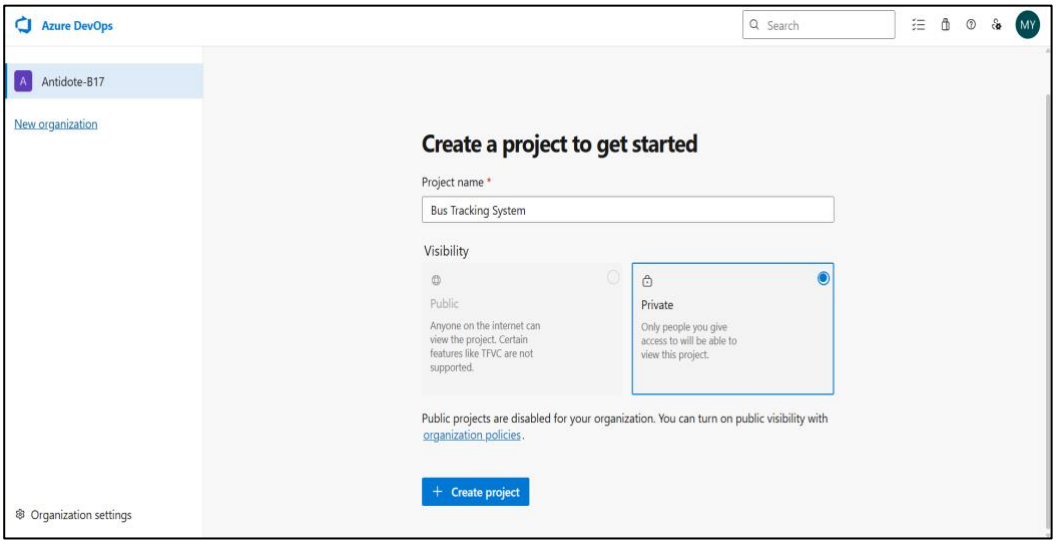
ii. Enter the project name, description, and visibility options:

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

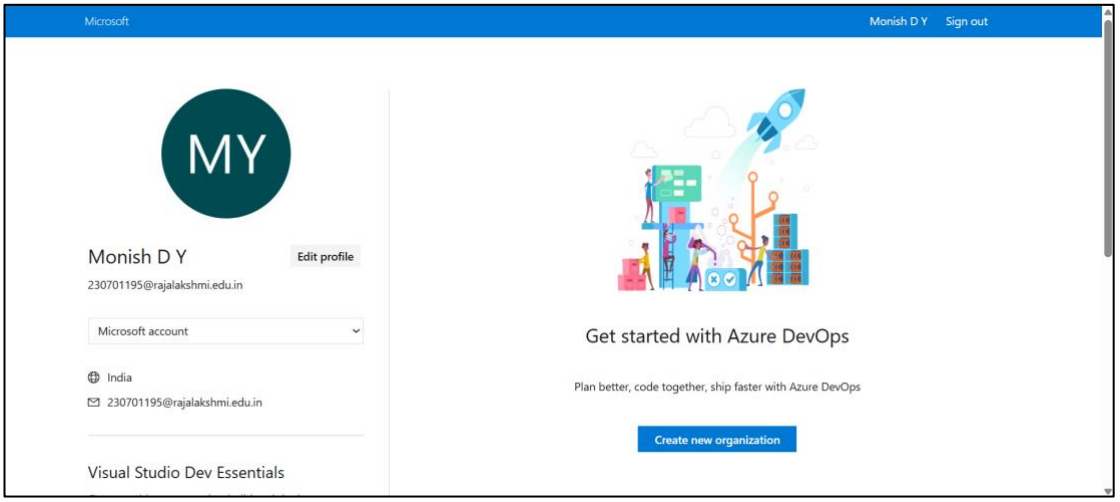
○ Description: Optionally, add a description to provide more context about the project.

○ Visibility: Choose whether you want the project to be Private (Accessible only to those invited) or Public (accessible to anyone).

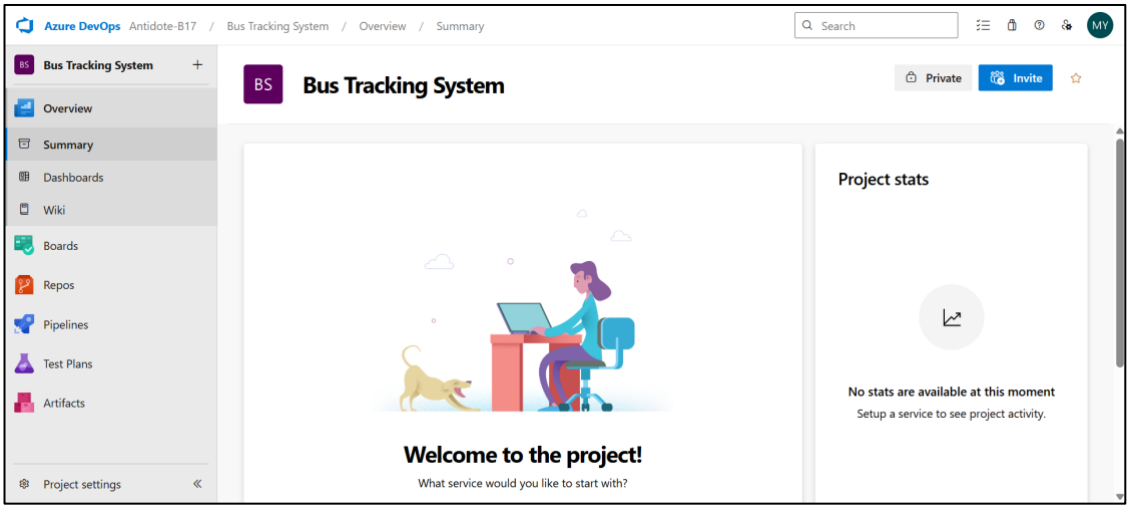
iii. Once you've filled out the details, click Create to set up your first project.



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.



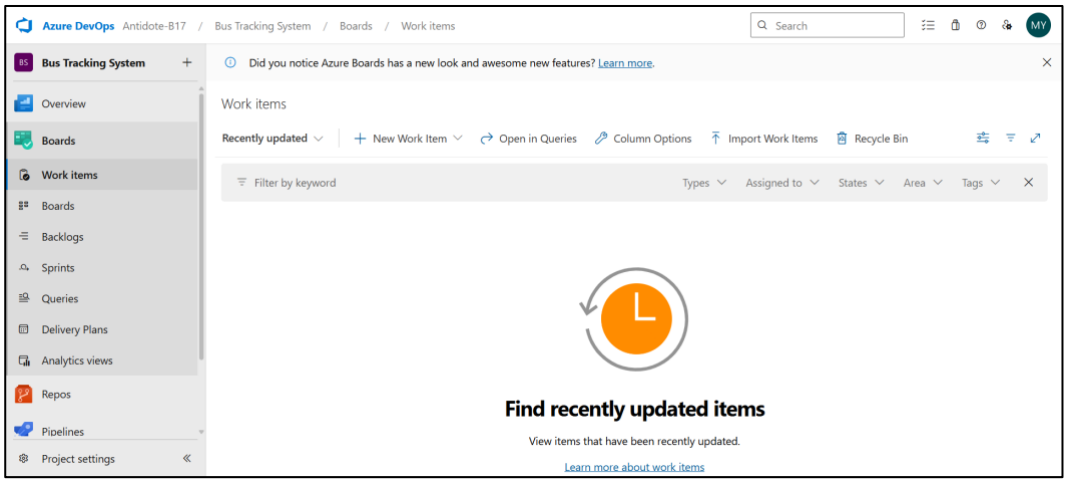
8. <u>Project Dashboard</u>



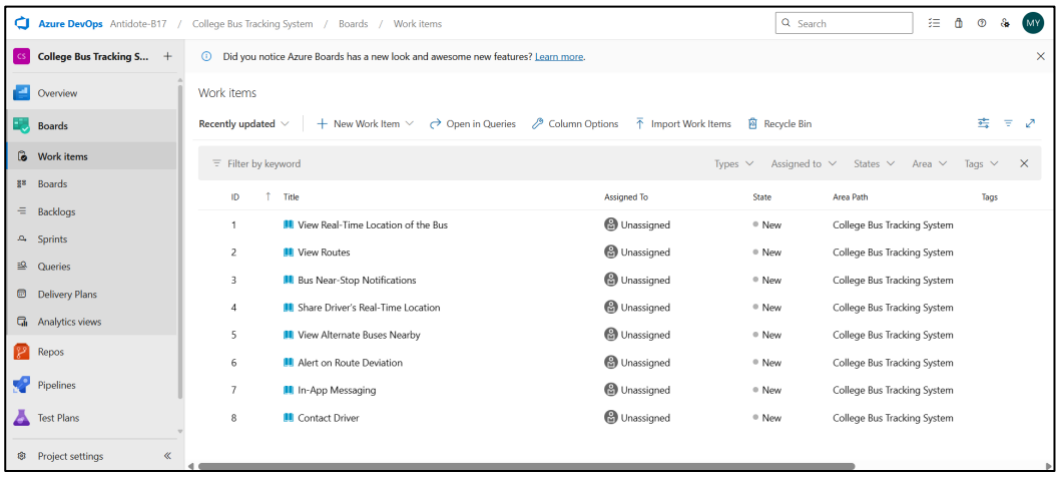
9. To manage user stories

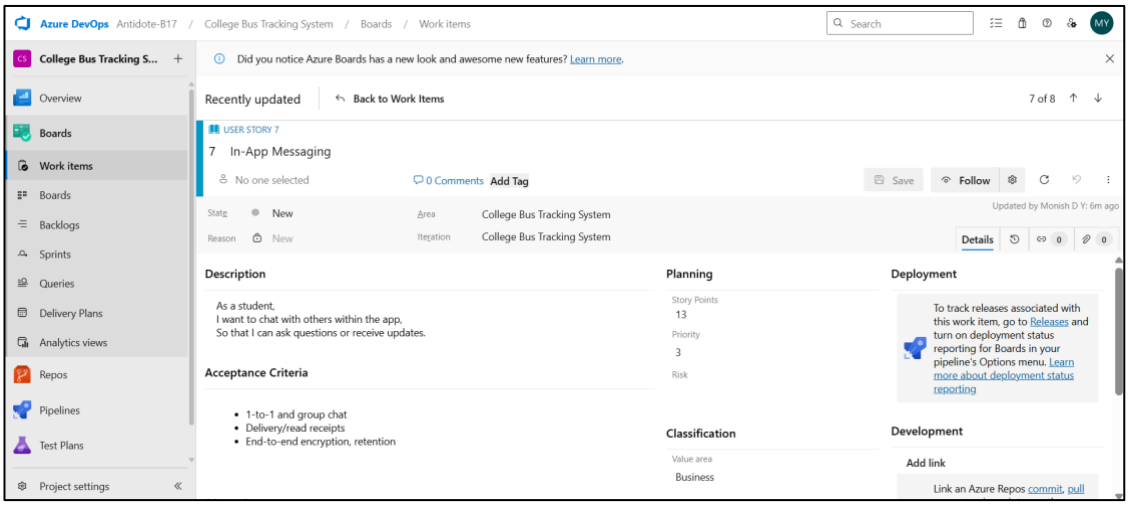
a. From the left-hand navigation menu, click on Boards. This will take you to the main Boards page, where you can manage work items, backlogs, and sprints.

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 details





Result:

The user story was written successfully.

Ex. No. 5	DESIGNING SEQUENCE DIAGRAM

Aim:

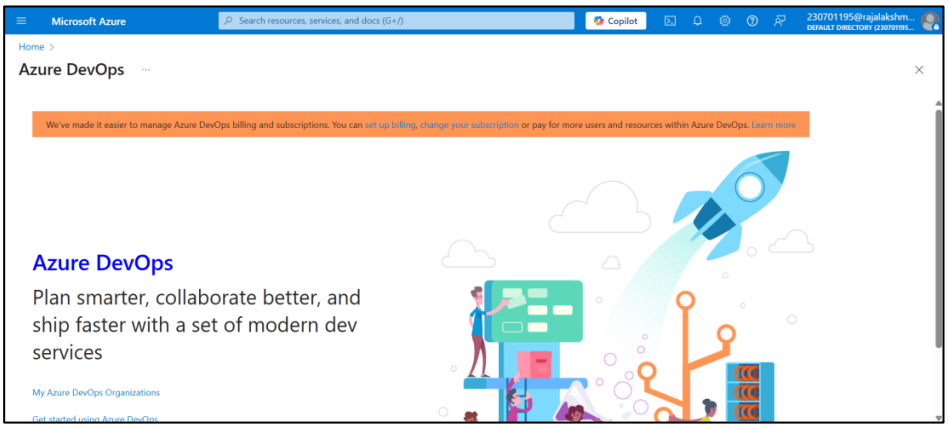
To design a Sequence Diagram using Mermaid.js

THEORY:

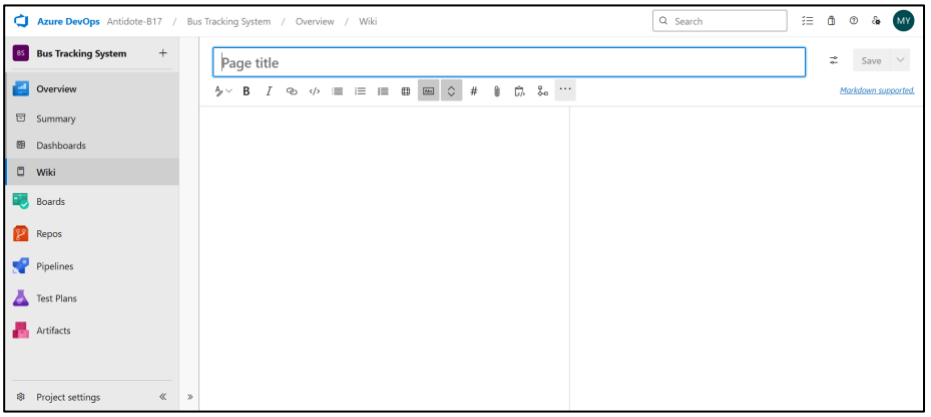
A Sequence Diagram is a key component of Unified Modeling Language (UML) used to visualize the interaction between objects in a sequential order. It focuses on how objects communicate with each other over time, making it an essential tool for modelling dynamic behavior in a system.

Procedure:

1. Open a project in Azure DevOps Organizations.



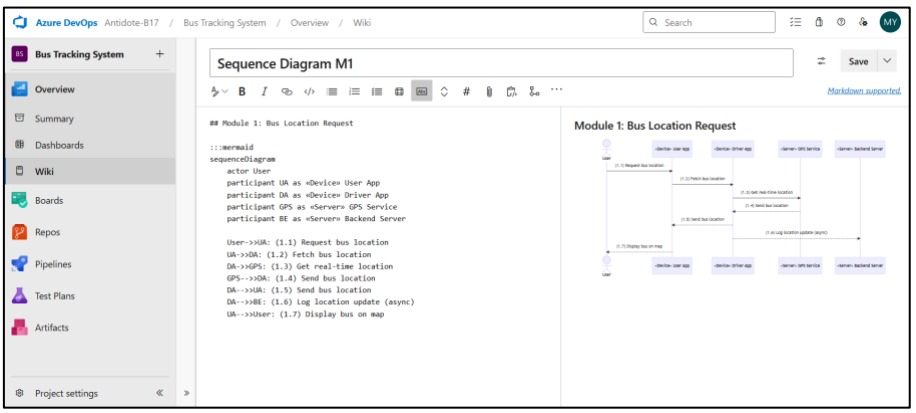
2. To design select wiki from menu



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

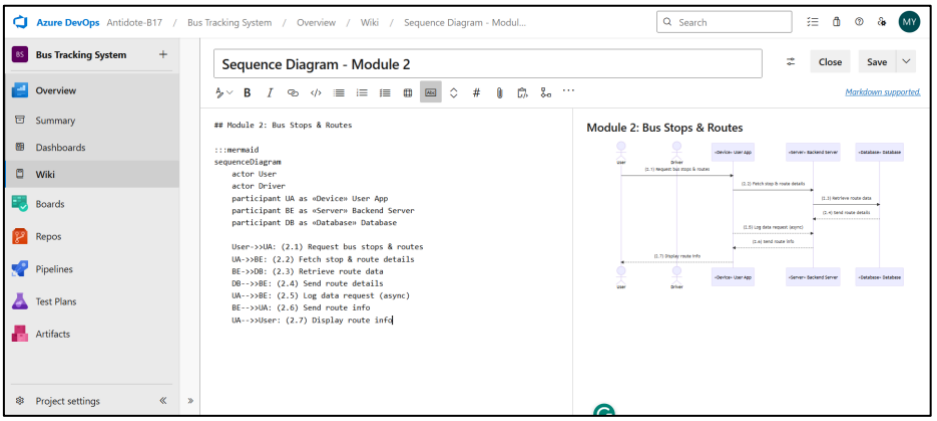
Module 1: Bus Location Request

```
## Module 1: Bus Location Request

:::mermaid
sequenceDiagram
    actor User
    participant UA as «Device» User App
    participant DA as «Device» Driver App
    participant GPS as «Server» GPS Service
    participant BE as «Server» Backend Server

    User->>UA: (1.1) Request bus location
    UA->>DA: (1.2) Fetch bus location
    DA->>GPS: (1.3) Get real-time location
    GPS-->>DA: (1.4) Send bus location
    DA-->>UA: (1.5) Send bus location
    DA-->>BE: (1.6) Log location update (async)
    UA-->>User: (1.7) Display bus on map
```



Module 2: Bus Location Request

```
## Module 2: Bus Stops & Routes

:::mermaid
sequenceDiagram
    actor User
    actor Driver
    participant UA as «Device» User App
    participant BE as «Server» Backend Server
    participant DB as «Database» Database

    User->>UA: (2.1) Request bus stops & routes
    UA->>BE: (2.2) Fetch stop & route details
    BE->>DB: (2.3) Retrieve route data
    DB-->>BE: (2.4) Send route details
    UA-->>BE: (2.5) Log data request (async)
    BE-->>UA: (2.6) Send route info
    UA-->>User: (2.7) Display route info
```

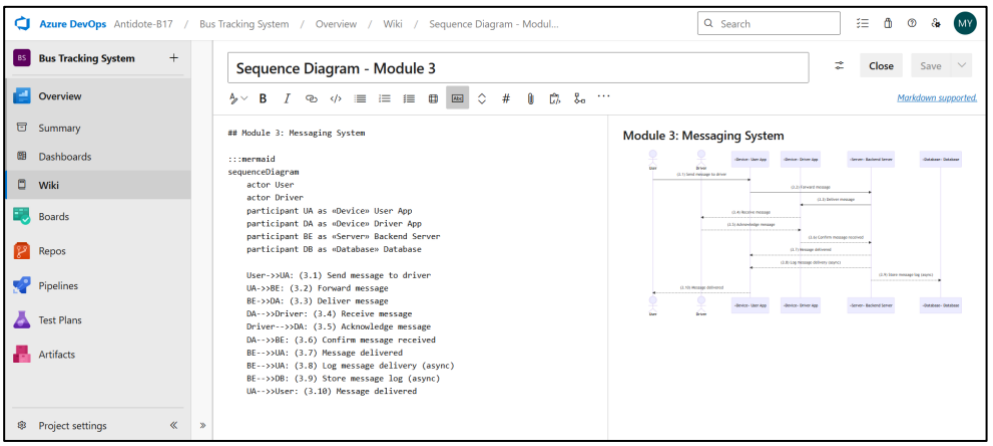


Module 3: Messaging System

Module 3: Messaging System

:::mermaid
sequenceDiagram
actor User
actor Driver
participant UA as «Device» User App
participant DA as «Device» Driver App
participant BE as «Server» Backend Server
participant DB as «Database» Database

User->>UA: (3.1) Send message to driver
UA->>BE: (3.2) Forward message
BE->>DA: (3.3) Deliver message
DA-->>Driver: (3.4) Receive message
Driver-->>DA: (3.5) Acknowledge message
DA-->>BE: (3.6) Confirm message received
BE-->>UA: (3.7) Message delivered
BE-->>UA: (3.8) Log message delivery (async)
BE-->>DB: (3.9) Store message log (async)
UA-->>User: (3.10) Message delivered

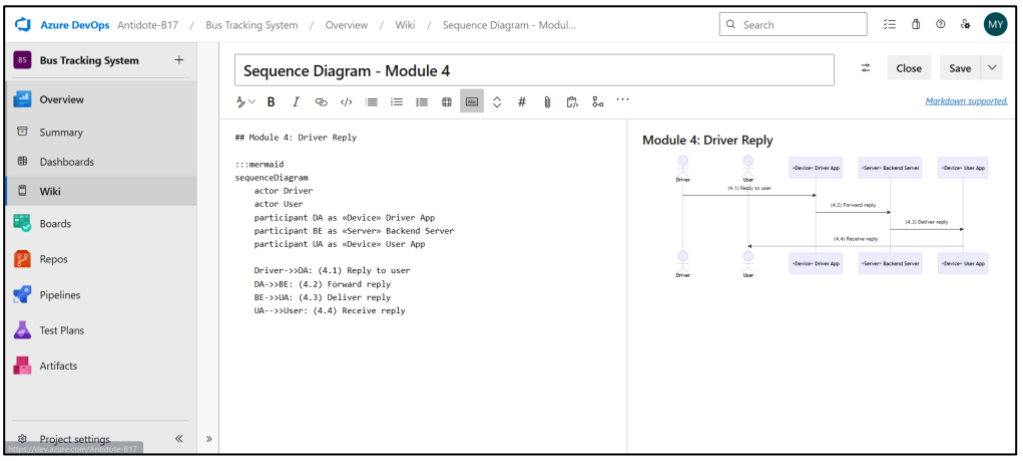


Module 4: Driver Reply

Module 4: Driver Reply

:::mermaid
sequenceDiagram
    actor Driver
    actor User
    participant DA as «Device» Driver App
    participant BE as «Server» Backend Server
    participant UA as «Device» User App

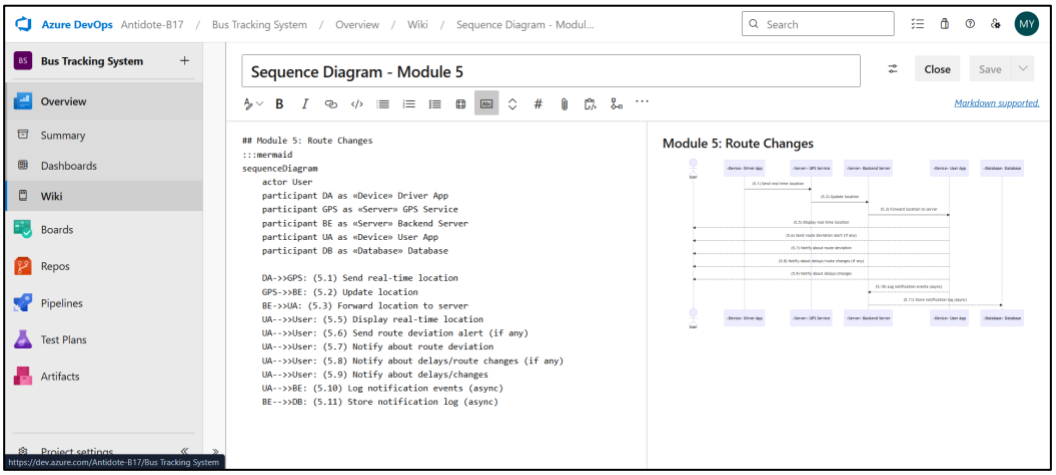
    Driver->>DA: (4.1) Reply to user
    DA->>BE: (4.2) Forward reply
    BE->>UA: (4.3) Deliver reply
    UA-->>User: (4.4) Receive reply



Module 5:

Module 5: Route Changes
:::mermaid
sequenceDiagram
    actor User
    participant DA as «Device» Driver App
    participant GPS as «Server» GPS Service
    participant BE as «Server» Backend Server
    participant UA as «Device» User App
    participant DB as «Database» Database

    DA->>GPS: (5.1) Send real-time location
    GPS->>BE: (5.2) Update location
    BE->>UA: (5.3) Forward location to server
    UA-->>User: (5.5) Display real-time location
    UA-->>User: (5.6) Send route deviation alert (if any)
    UA-->>User: (5.7) Notify about route deviation
    UA-->>User: (5.8) Notify about delays/route changes (if any)
    UA-->>User: (5.9) Notify about delays/changes
    UA-->>BE: (5.10) Log notification events (async)
    BE-->>DB: (5.11) Store notification log (async)



Explanation:

participant defines the entities involved.
->> represents a direct message.
-->> represents a response message.
+ after ->> activates a participant.
- after -->> deactivates a participant. alt /
else for conditional flows.
loop can be used for repeated actions.

->	Solid line without arrow
-->	Dotted line without arrow
->>	Solid line with arrowhead
-->>	Dotted line with arrowhead
<<->>	Solid line with bidirectional arrowheads (v11.0.0+)
<<-->>	Dotted line with bidirectional arrowheads (v11.0.0+)
-x	Solid line with a cross at the end
--x	Dotted line with a cross at the end
-)	Solid line with an open arrow at the end (async)
--)	Dotted line with an open arrow at the end (async)

4. click wiki menu and select the page

Result:

The sequence diagram was drawn successfully.

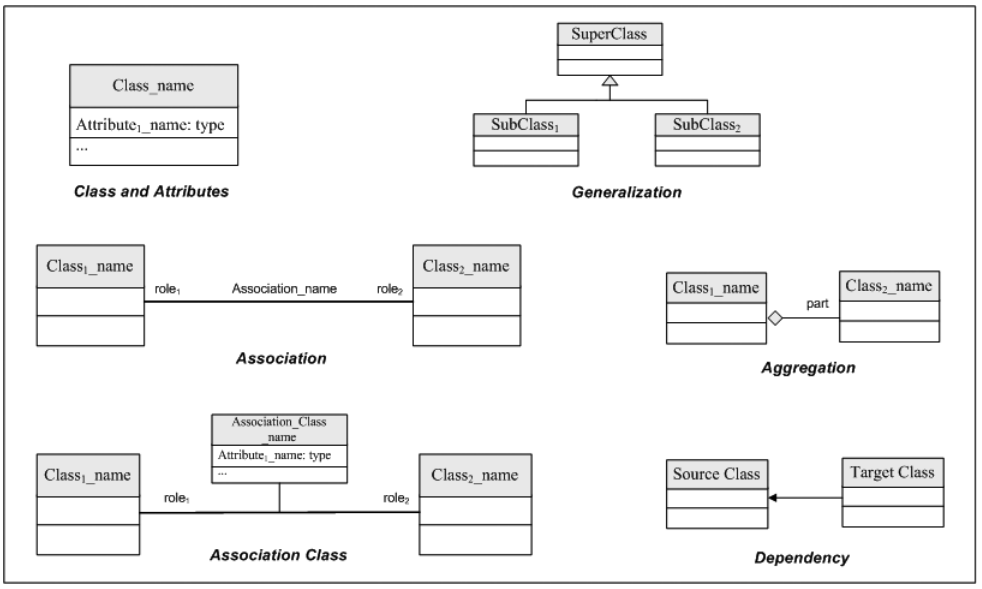
Ex. No. 6	DESIGNING CLASS DIAGRAM

AIM: -

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

THEORY

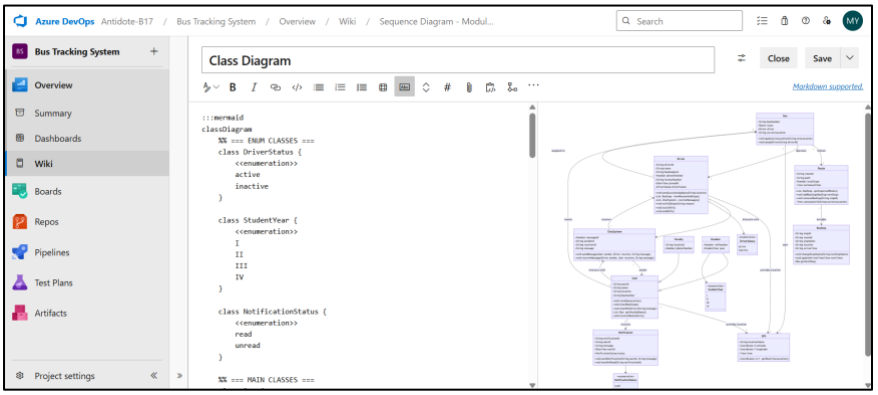
A UML class diagram is a visual tool that represents the structure of a system by showing its classes, attributes, methods, and the relationships between them.



Notations in class diagram

Procedure:

1. Open a project in Azure DevOps Organizations.
2. To design select wiki from menu



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

Code:

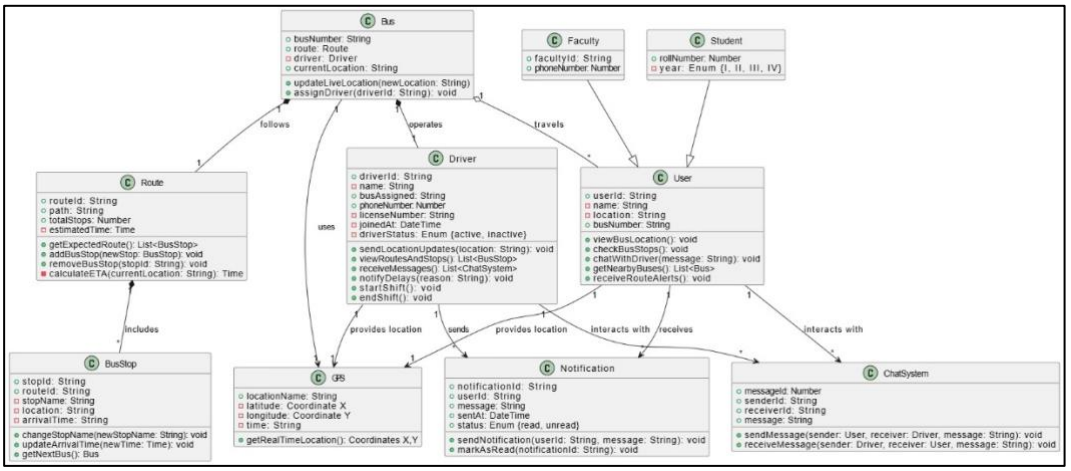
```
:::mermaid
classDiagram
    %% === ENUM CLASSES ===
    class DriverStatus {
        <<enumeration>>
        active
        inactive
    }

    class StudentYear {
        <<enumeration>>
        I
        II
        III
        IV
    }

    class NotificationStatus {
        <<enumeration>>
        read
        unread
    }

    %% === MAIN CLASSES ===
    class Bus {
        +String busNumber
        +Route route
        +Driver driver
        +String currentLocation
        +void updateLiveLocation(String newLocation)
        +void assignDriver(String driverId)
    }

    class Driver {
        +String driverId
        +String name
        +String busAssigned
        +Number phoneNumber
        +String licenseNumber
        +DateTime joinedAt
        +DriverStatus driverStatus
        +void sendLocationUpdates (String location)
        +List~BusStop~ viewRoutesAndStops ()
        +List~ChatSystem~ receiveMessages ()
        +void notifyDelays (String reason)
        +void startShift ()
        +void endShift ()
    }
```

```
class Route {
    +String routeId
    +String path
    +Number totalStops
    -Time estimatedTime
    +List~BusStop~ getExpectedRoute()
    +void addBusStop(BusStop newStop)
    +void removeBusStop(String stopId)
    -Time calculateETA(String currentLocation)
}

class BusStop {
    +String stopId
    +String routeId
    +String stopName
    +String location
    +String arrivalTime
    +void changeStopName(String newStopName)
    +void updateArrivalTime(Time newTime)
    +Bus getNextBus()
}

class GPS {
    +String locationName
    +Coordinate X latitude
    +Coordinate Y longitude
    +Time
    +Coordinates~X,Y~ getRealTimeLocation()
}

class User {
    +String userId
    +String name
    +String location
    +String busNumber
    +void viewBusLocation()
    +void checkBusStops()
    +void chatWithDriver(String message)
    +List~Bus~ getNearbyBuses()
    +void receiveRouteAlerts()
}

class Faculty {
    +String facultyId
    +Number phoneNumber
}

class Student {
    +Number rollNumber
    +StudentYear year
}
```

```
class Notification {
    +String notificationId
    +String userId
    +String message
    +DateTime sentAt
    +NotificationStatus status
    +void sendNotification(String userId, String message)
    +void markAsRead(String notificationId)
}

class ChatSystem {
    +Number messageId
    +String senderId
    +String receiverId
    +String message
    +void sendMessage(User sender, Driver receiver, String message)
    +void receiveMessage(Driver sender, User receiver, String message)
}

%% Inheritance
Faculty --|> User
Student --|> User

%% Relationships with names and cardinality
Bus "1" --> "1" Route: follows
Bus "1" --> "1" Driver: operates
Bus "1" --> "1" GPS: uses

Route "1" --> "*" BusStop : includes

Driver "1" --> "1" GPS: provides location
Driver "1" --> "*" ChatSystem : interacts with
Driver "1" --> "1" Bus: assigned to

User "1" --> "1" Bus: travels
User "1" --> "1" GPS: provides location
User "1" --> "*" ChatSystem : interacts with
User "1" --> "*" Notification : receives

ChatSystem "*" --> "1" User : sender
ChatSystem "*" --> "1" Driver : receiver

%% Enum relations
Driver --> DriverStatus
Student --> StudentYear
Notification --> NotificationStatus
```

Relationship Types

Type	Description
<|	Inheritance
*	Composition
o	Aggregation
>	Association
<	Association
|>	Realization



Result:

The use case diagram was designed successfully.

Ex. No. 7	DESIGNING USE CASE DIAGRAM

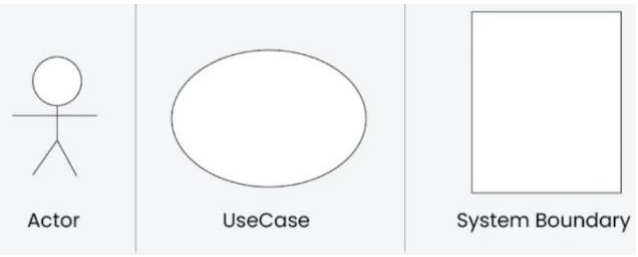
Aim:

Steps to draw the Use Case Diagram using draw.io

Theory:

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

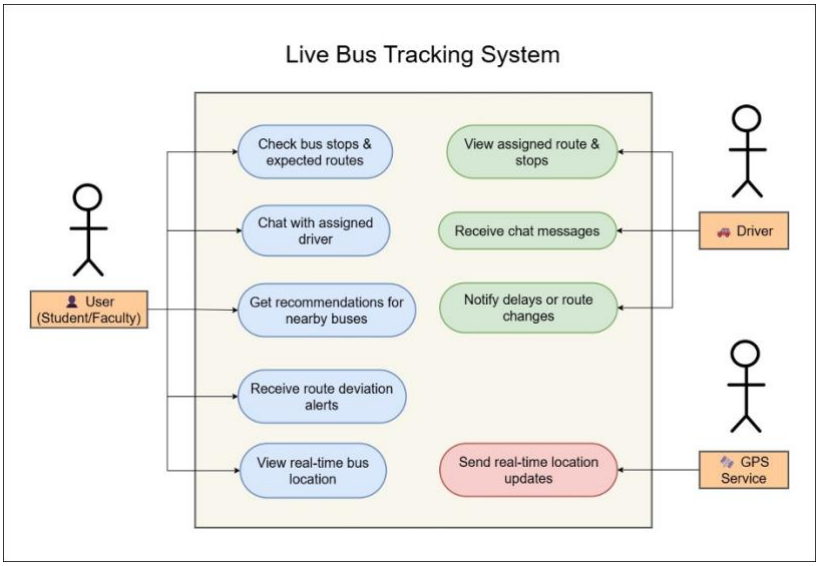
- **Use Cases**
- **Actors**
- **Relationships**
- **System Boundary Boxes**



Procedure

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

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



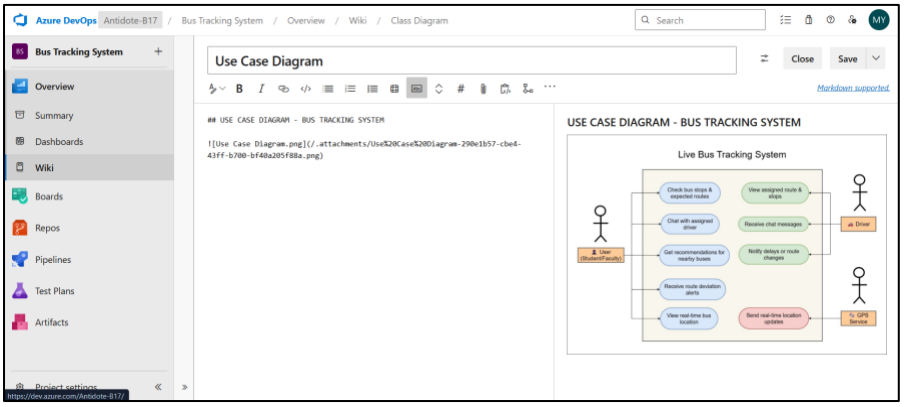
Step 2: Upload the Diagram to Azure DevOps

Option 1: Add to Azure DevOps Wiki

- Open Azure DevOps and go to your project.
- 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	DESIGNING ACTIVITY DIAGRAM

AIM :-

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

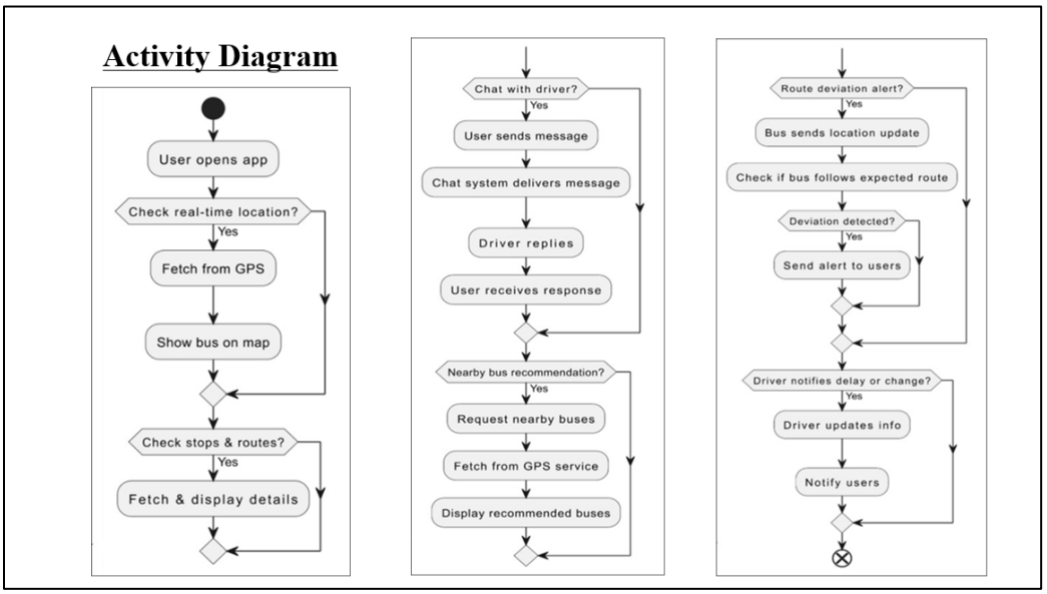
THEORY

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.

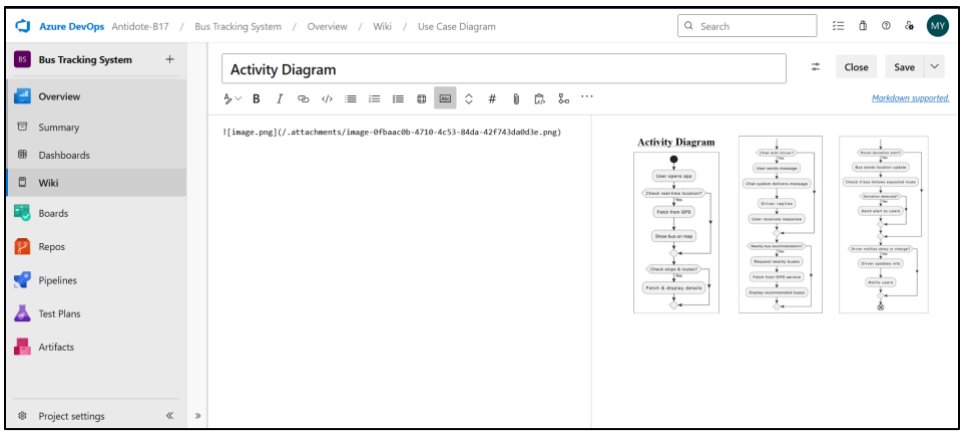
Notations	Symbol	Meaning
Start		Shows the beginning of a process
Connector		Shows the directional flow, or control flow, of the activity
Joint symbol		Combines two concurrent activities and re-introduces them to a flow where one activity occurs at a time
Decision		Represents a decision
Note		Allows the 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	H	Represents a transition that invokes the last active state.
End		Marks the end state of an activity and represents the completion of all flows of a process

Procedure:

1. Draw diagram in draw.io



2. Upload the diagram in Azure DevOps wiki



Result:

The activity diagram was designed successfully

Ex. No. 9	DESIGNING ARCHITECTURE DIAGRAM

Aim:

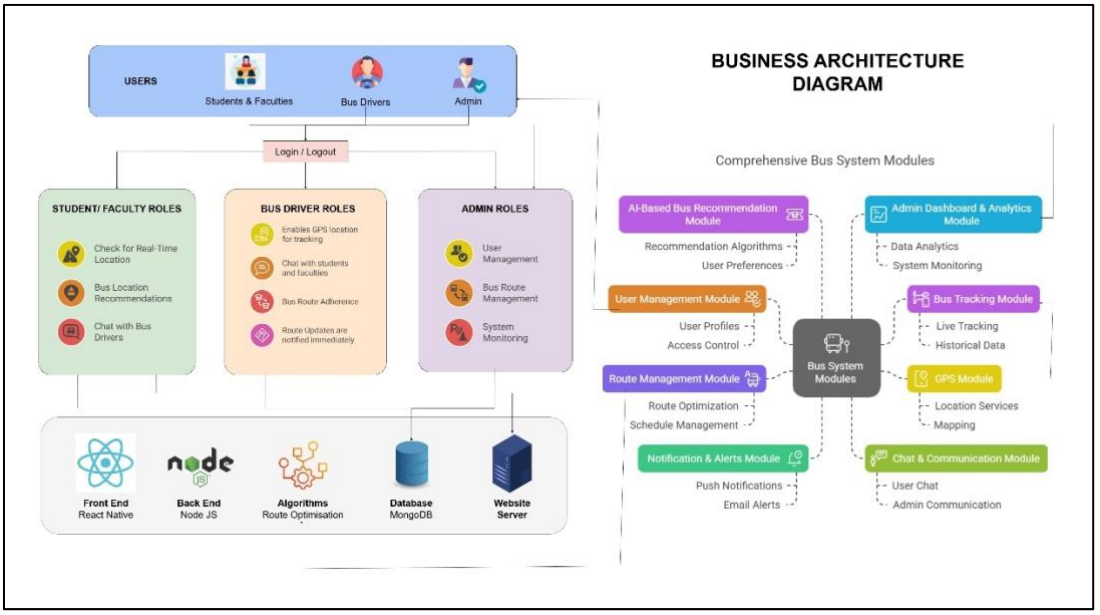
Steps to draw the Architecture Diagram using draw.io.

Theory:

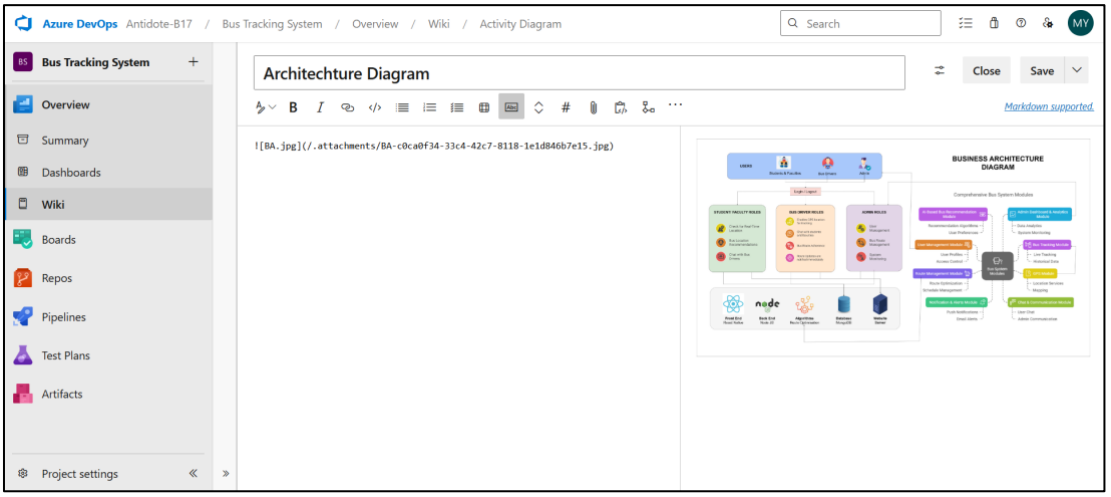
An architectural diagram is a visual representation that maps out the physical implementation of components of a software system. It shows the general structure of the software system and the associations, limitations, and boundaries between each element.

Procedure

1. Draw diagram in draw.io



2. Upload the diagram in Azure DevOps wiki



Result:

The architecture diagram was designed successfully

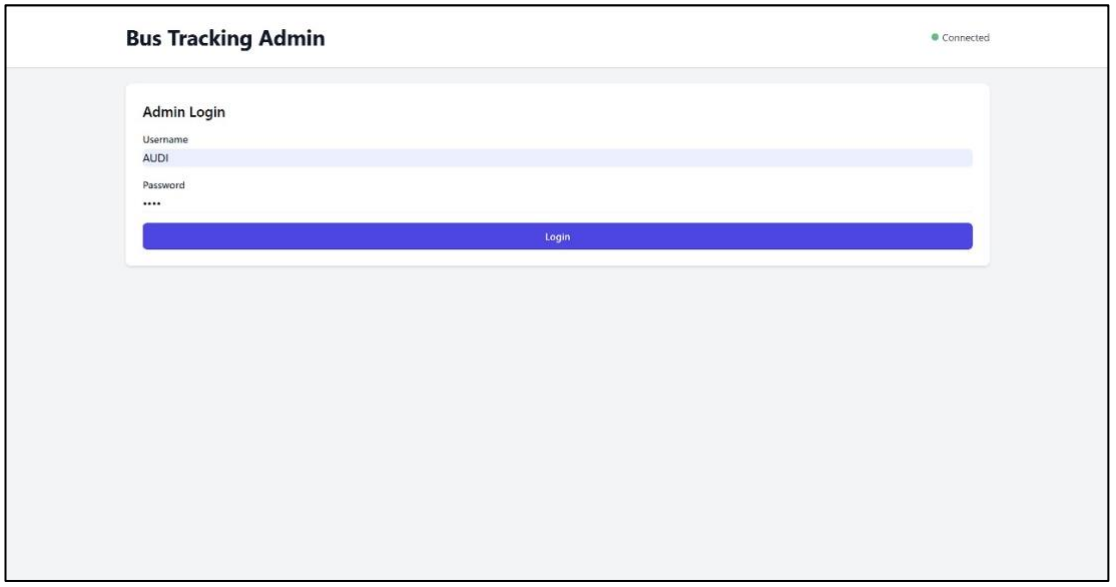
Ex. No. 10	DESIGNING A USER INTERFACE

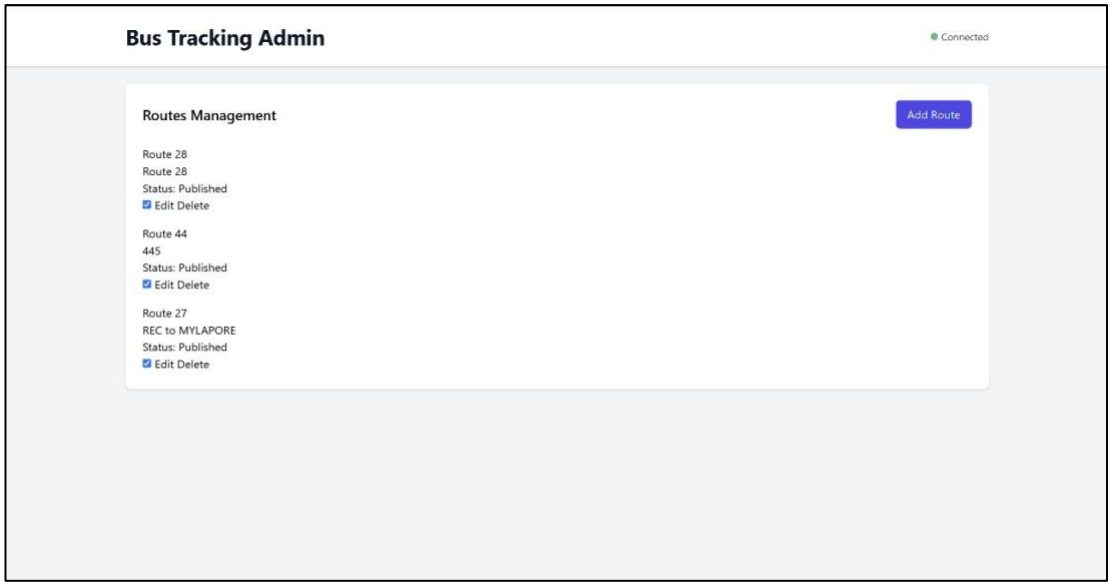
Aim:

Design a User Interface for the given project

Output:

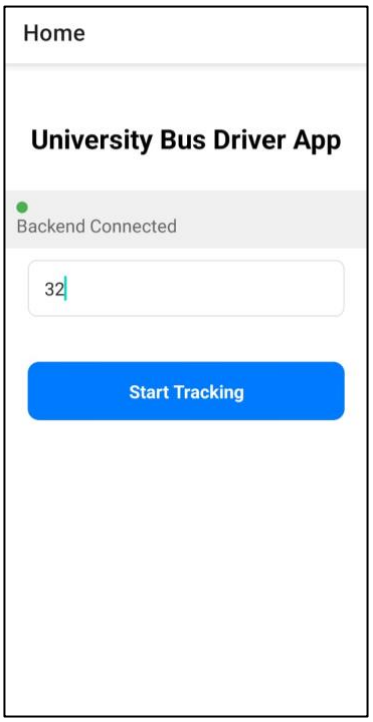
1. **Admin Page Login (Website)**

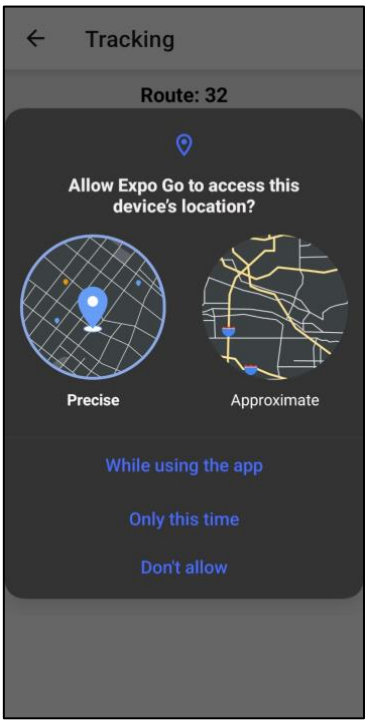




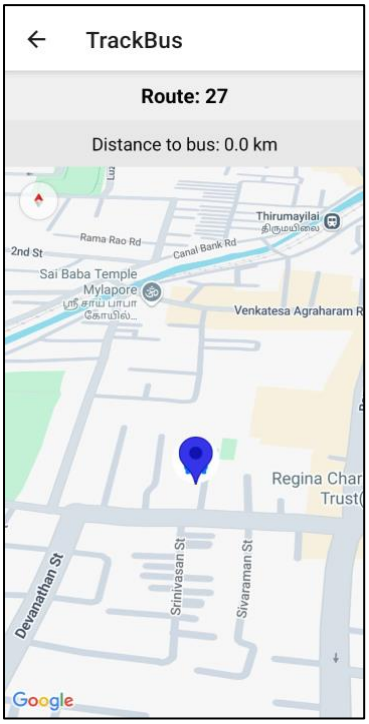
2. User Side Login (Mobile App)

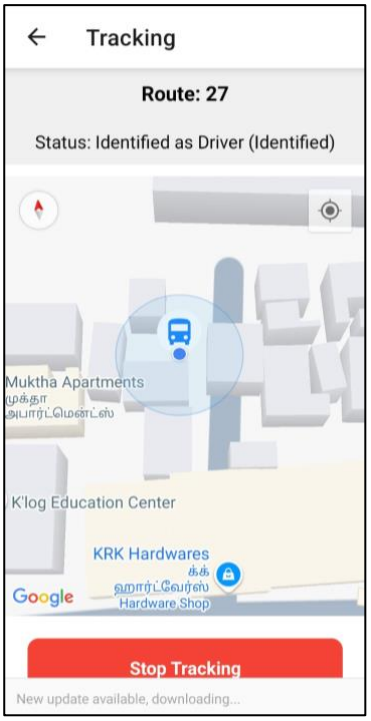
i. Driver Login:





ii. Student / Faculty Login:





Result: The UI was designed successfully.

Ex. No. 11	IMPLEMENTATION OF THE PROJECT

Aim:

To implement the given project based on Agile Methodology.

Procedure:

Step 1: Set Up an Azure DevOps Project

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

Step 2: Add Your Web Application Code

- Navigate to Repos → Click "Clone" to get the Git URL.
- Open Visual Studio Code / Terminal and run:

```
git clone <repo_url>
cd <repo_folder>
```

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

```
git add .
git commit -m "Initial commit"
git push origin main
```

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

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