RAJALAKSHMI ENGINEERING COLLEGE

RAJALAKSHMI NAGAR, THANDALAM – 602 105

RAJALAKSHMI
ENGINEERING COLLEGE

CS23432

SOFTWARE CONSTRUCTION

Laboratory Record Note Book

Name : . . . . . .

Year / Branch / Section : . . . . . .

Register No. : . . . . . .

Semester : . . . . . .

Academic Year : . . . . . .

RAJALAKSHMI ENGINEERING COLLEGE (AUTONOMOUS)

RAJALAKSHMI NAGAR, THANDALAM – 602 105

BONAFIDE CERTIFICATE

NAME _____ REGISTER NO. _____

ACADEMIC YEAR 2024-25 **SEMESTER**- IV **BRANCH**: B. Tech Information Technology [AD/AE]. This Certification is the Bonafide record of work done by the above student in the **CS23432- Software Construction** Laboratory during the year 2024-2025.

Signature of Faculty -in – Charge

Submitted for the Practical Examination held on _____

Internal Examiner External Examiner

LAB PLAN

CS23432-SOFTWARE CONSTRUCTION LAB

Ex No	Date	Topic	Page No	Sign
1	21/01/2025	Study of Azure DevOps		
2	28/01/2025	Problem Statement		
3	04/02/2025	Agile Planning		
4	18/02/2025	Create User stories with Acceptance Criteria		
5	25/02/2025	Designing Sequence Diagrams using Azure DevOps-WIKI		
6	04/03/2025	Designing Class Diagram using Azure DevOps-WIKI		
7	11/03/2025	Designing Use case Diagram using Azure DevOps-WIKI		
8	18/03/2025	Designing Activity Diagrams using Azure DevOps-WIKI		
9	25/03/2025	Designing Architecture Diagram Using Star UML		
10	01/04/2025	Design User Interface		
11	08/04/2025	Implementation – Design a Web Page based on Scrum Methodology		
12	15/04/2025	Testing-Test Plan, Test Case and Load Testing		

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

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

PROBLEM STATEMENT

AIM:

To prepare PROBLEM STATEMENT for your given project.

Problem Statement:

E-commerce Uploader:

In the rapidly evolving world of digital commerce, e-commerce platforms are witnessing a massive influx of products across various categories to meet the dynamic demands of consumers. Sellers, ranging from small businesses to large enterprises, are required to upload and manage thousands of product listings regularly. However, the traditional manual method of uploading product data — including titles, descriptions, prices, categories, images, inventory details, and specifications — is often tedious, error-prone, and time-consuming.

Many sellers struggle with inconsistencies, formatting errors, missing information, and redundant work while uploading or updating their product catalogs. These issues not only hinder operational efficiency but also affect the customer experience due to inaccurate or incomplete product data. A slow and inefficient product listing process can delay the time-to-market, impacting the overall business performance.

To overcome these challenges, there is a need for a smart and scalable E-commerce Product Uploader Tool that simplifies and automates the product listing workflow. This tool should support features such as bulk uploads via CSV/Excel files, real-time data validation, image preview and compression, auto-category detection, and error highlighting — thereby ensuring faster, more accurate, and hassle-free uploads.

By implementing such a system, sellers can manage their product inventory more efficiently, minimize manual errors, and focus more on their core business strategies. The tool also ensures that end-users receive accurate and complete product information, leading to improved customer satisfaction, reduced returns, and increased trust in the platform.

Overall, this project aims to build a reliable and user-friendly product uploader tool that not only optimizes the seller's workflow but also contributes to the success and scalability of e-commerce platforms.

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

· 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 STORIES

Aim:

To create User Stories

THEORY

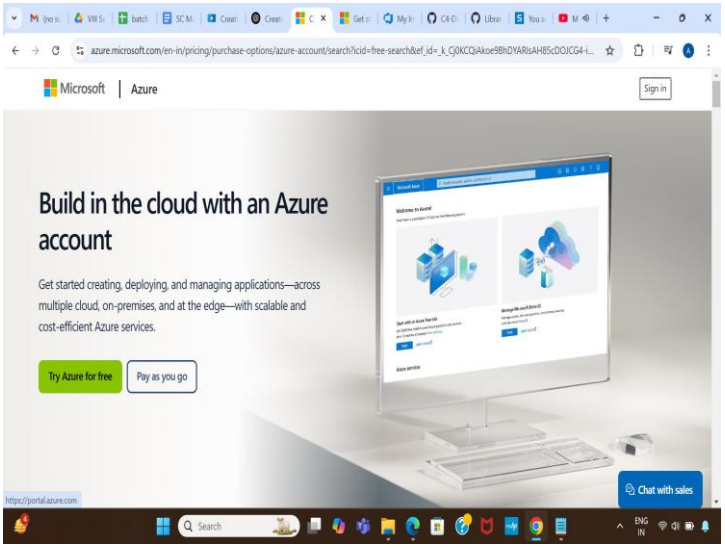
A user story is an informal, general explanation of a software feature written from the perspective of the end user. Its purpose is to articulate how a software feature will provide value to the customer.

User story template

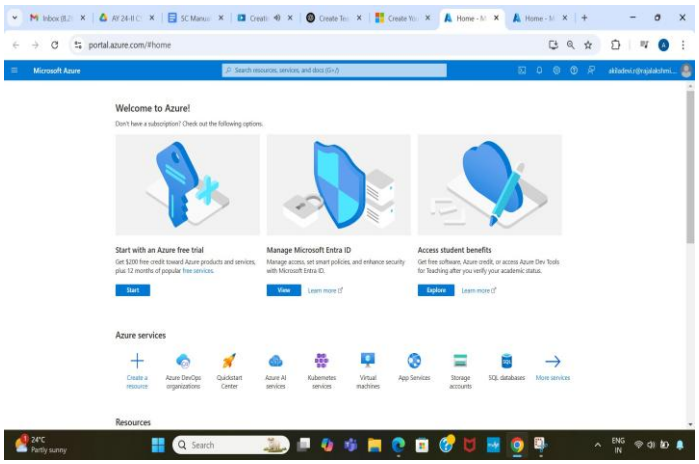
"As a [role], I [want to], [so that]."

Procedure:

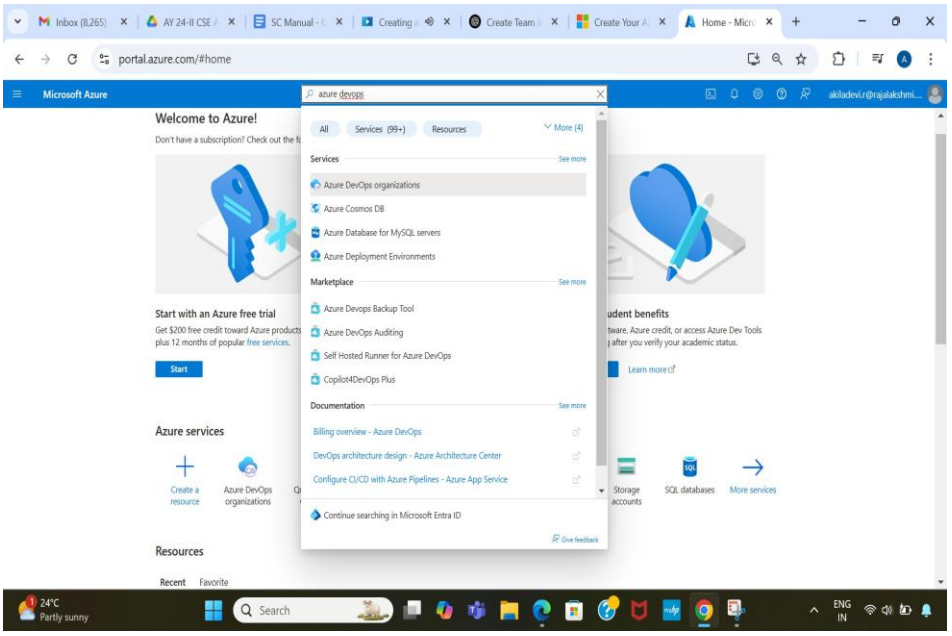
1. Open your web browser and go to the Azure website: *https://azure.microsoft.com/en-in* Sign in using your Microsoft account credentials. If you don't have an account, you'll need to create one.
2. If you don't have a Microsoft account, you can sign up for *https://signup.live.com/?lic=1*



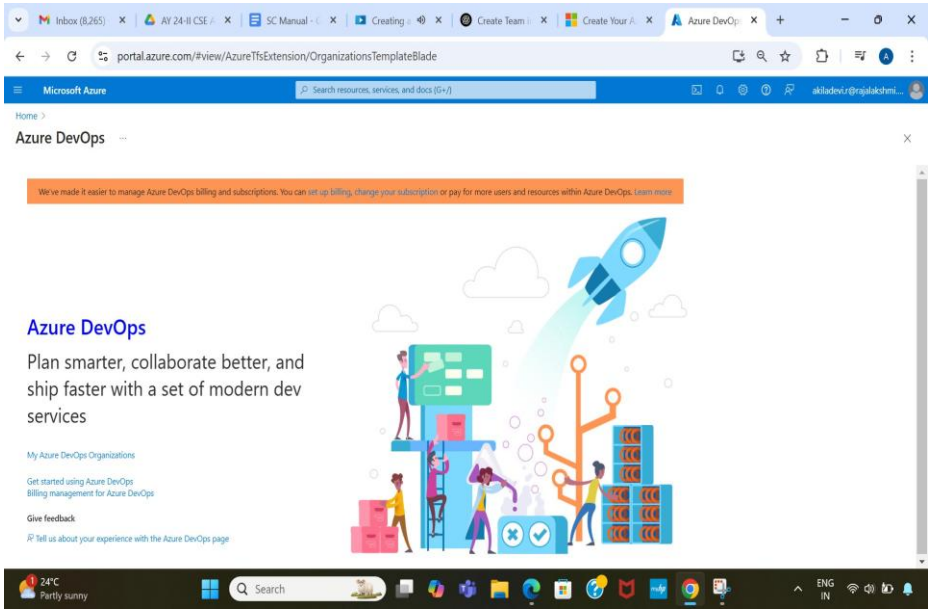
3. Azure home page

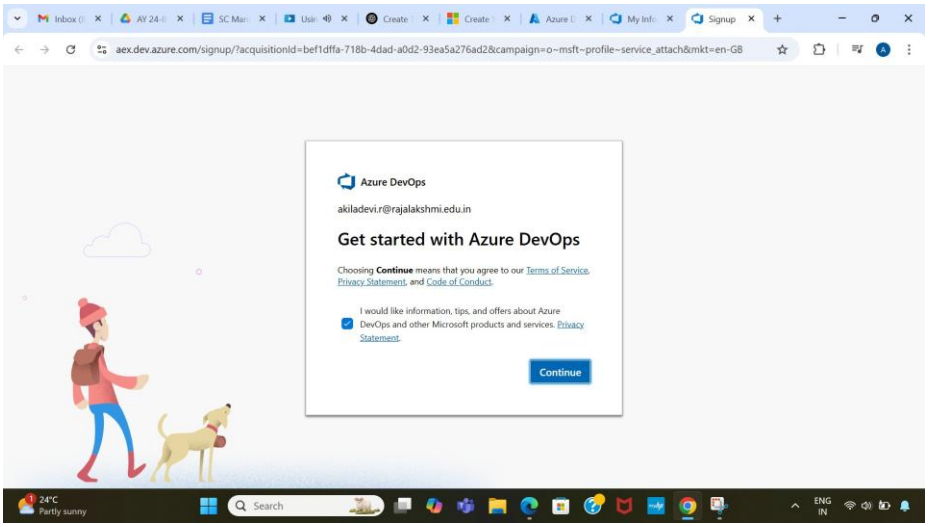


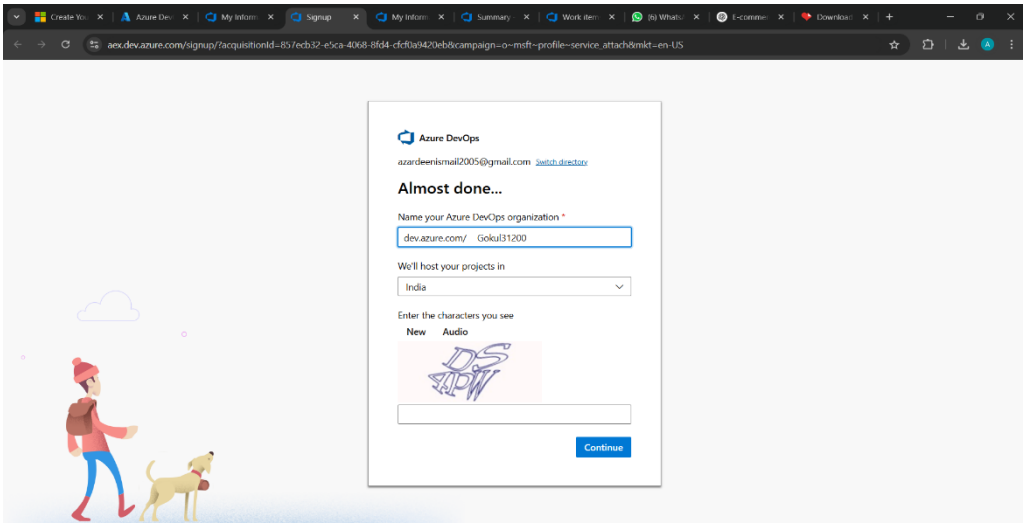
4. Open DevOps environment in the Azure platform by typing Azure DevOps Organizations in the search bar.



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.





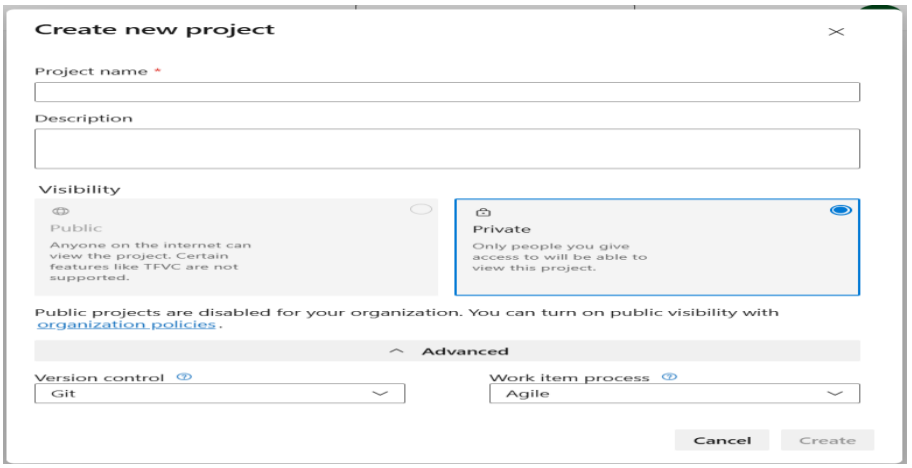


6. Create the First Project in Your Organization

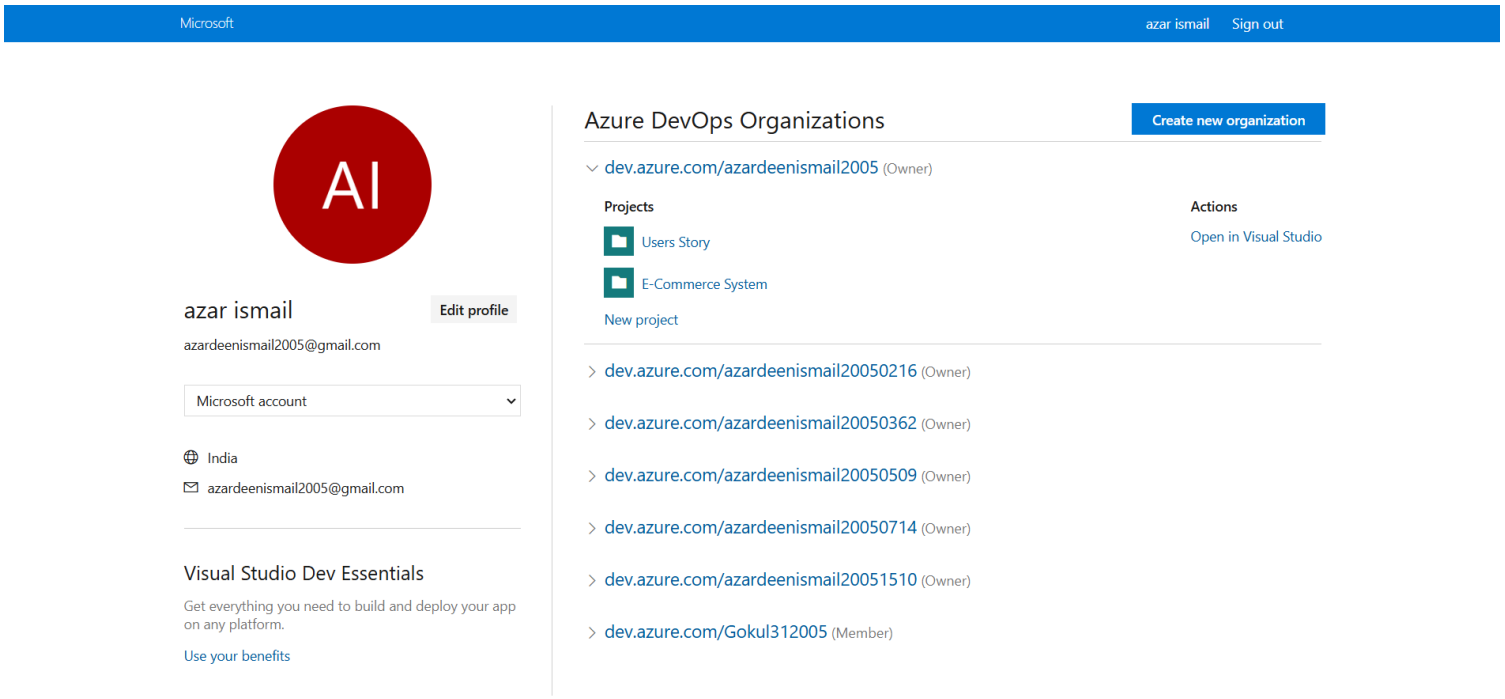
   After the organization is set up, you'll need to create your first **project**. This is where you'll begin to manage code, pipelines, work items, and more.

   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).

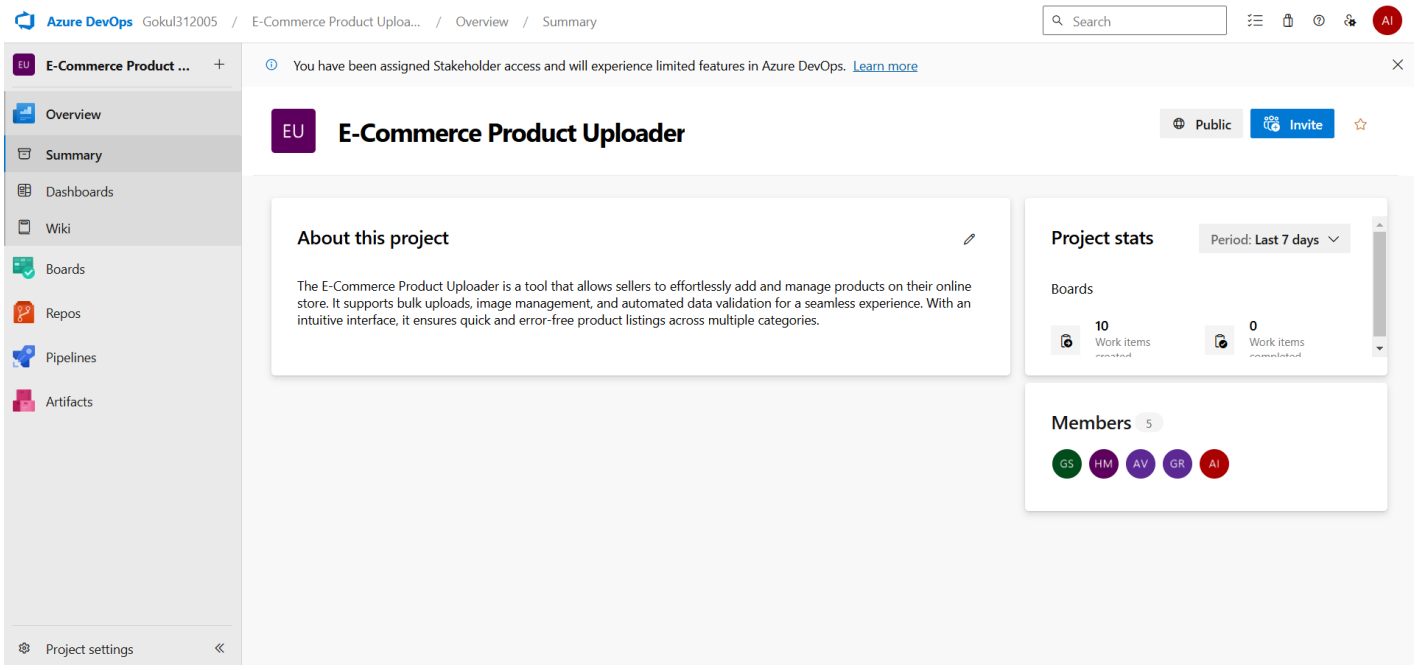
   iii. Once you've filled out the details, click **Create** to set up your first project.



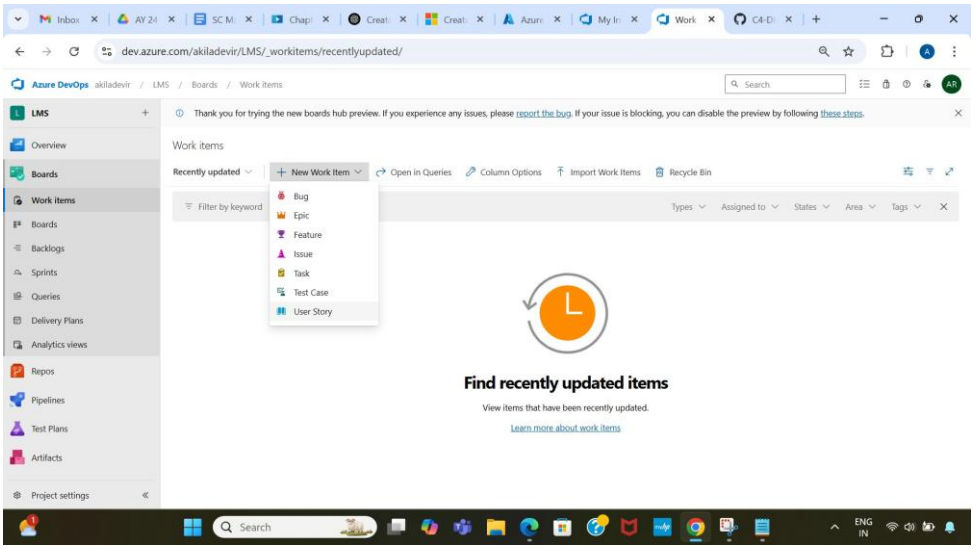
7. Once logged in, ensure you are in the correct organization. If you're part of multiple organizations, you can switch between them from the top left corner (next to your user profile). Click on the Organization name, and you should be taken to the Azure DevOps Organization Home page.

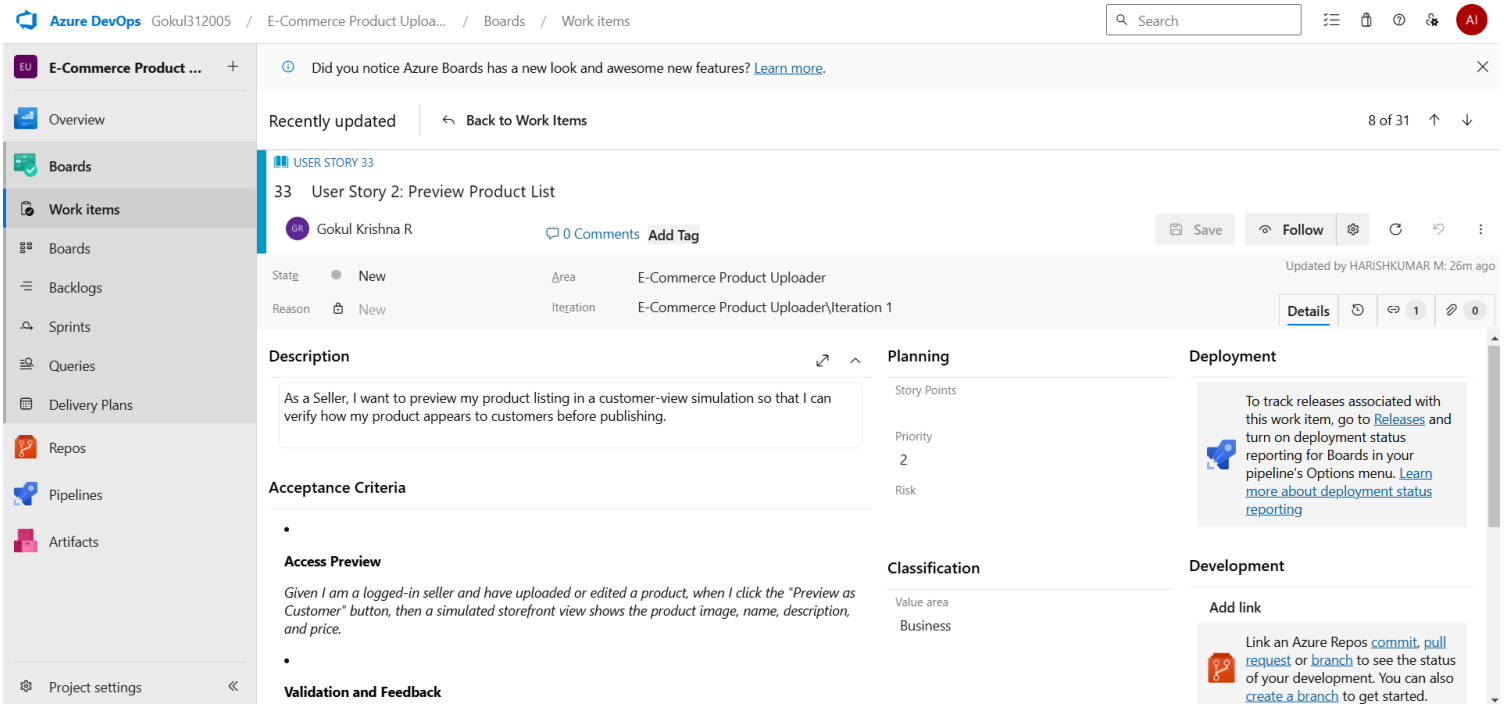


8. Project dashboard



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





10. Fill in User Story Details

Result:

The user story was written successfully.

EX NO: 5

SEQUENCE DIAGRAM

Aim:

To design a Sequence Diagram by using Mermaid.js

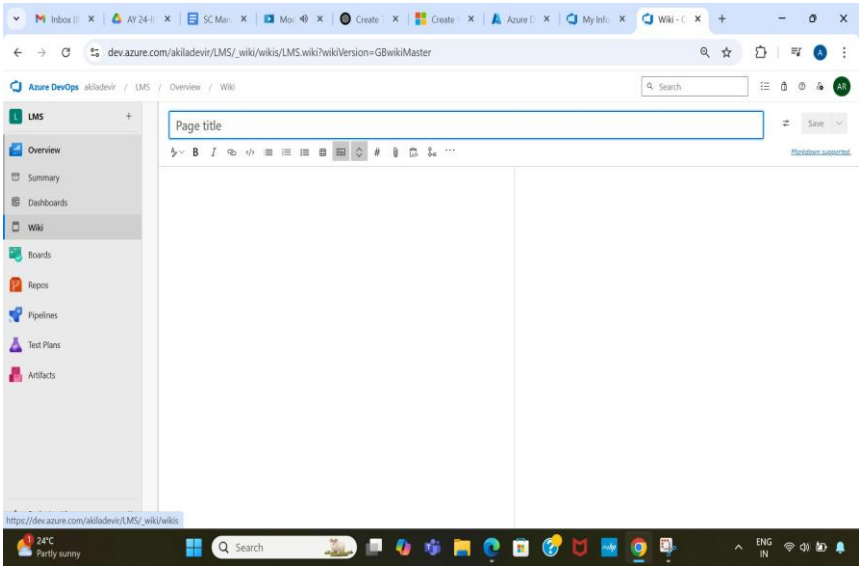
THEORY:

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
    actor User
    participant Uploader as ProductUploader
    participant Validator as ProductValidator
    participant Storage as ImageStorageService
    participant DB as ProductDatabase
    participant Docs as ProductDocumentationService

    User->>Uploader: uploadProduct(product, image)
    Uploader->>Validator: validateProduct(product)
    Validator-->>Uploader: validationResult

    Uploader->>Validator: validateImage(image)
    Validator-->>Uploader: imageValidationResult
```

Uploader->>Storage: uploadImage(image)
Storage-->>Uploader: imageUrl

Uploader->>DB: saveProduct(product with imageUrl)
DB-->>Uploader: saveResult

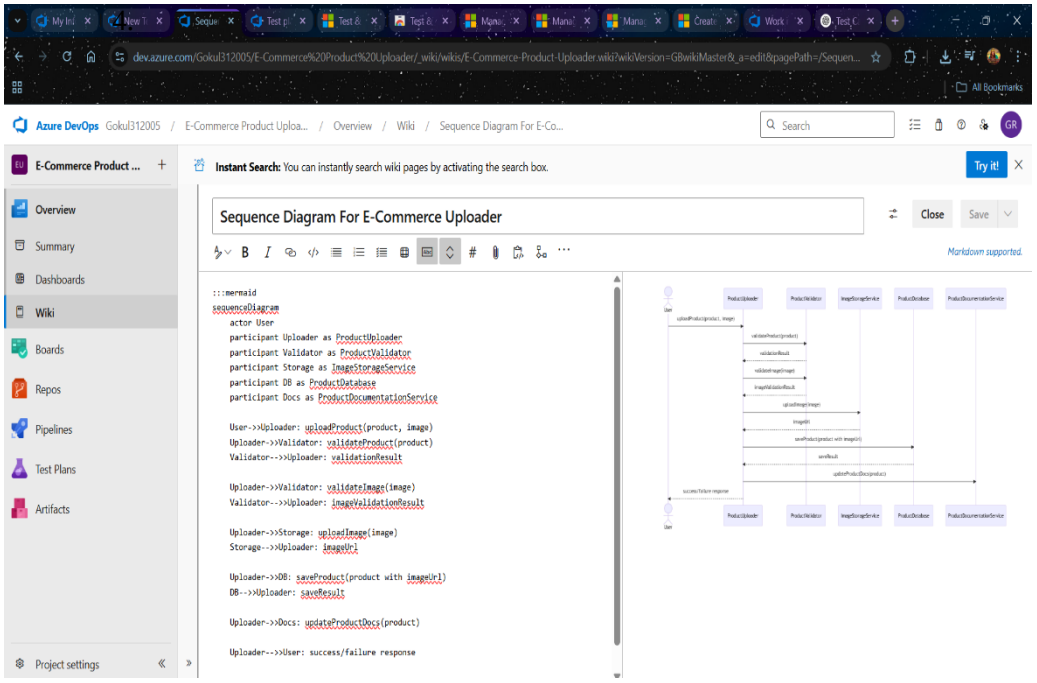
Uploader->>Docs: updateProductDocs(product)

Uploader-->>User: success/failure response

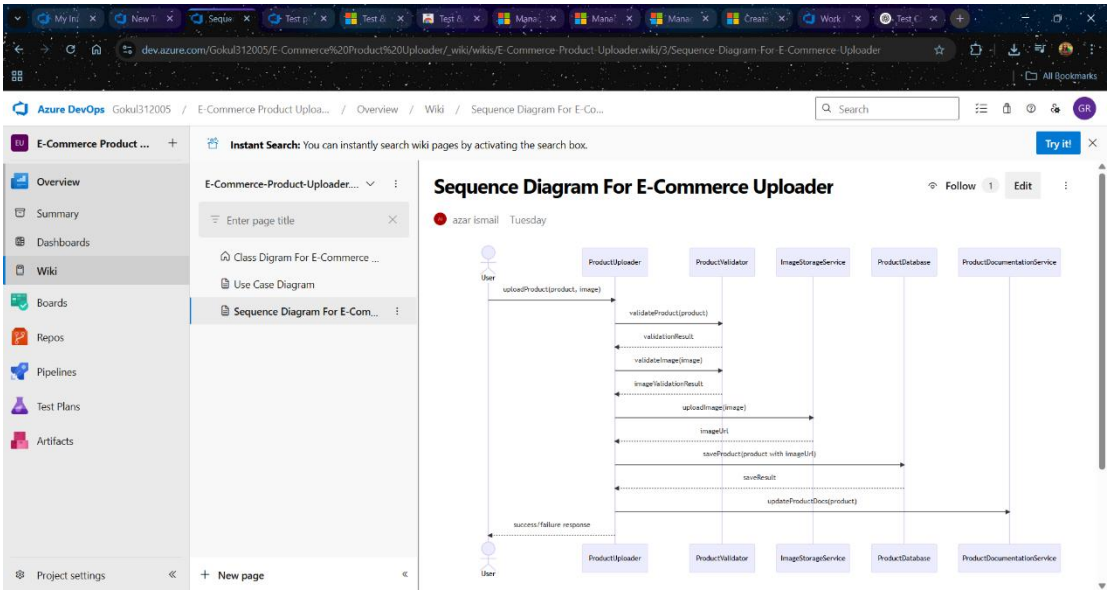
Explanation:

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 a open arrow at the end (async)



4.click wiki menu and select the page



Result:

The sequence diagram was drawn successfully.

EX NO. 6

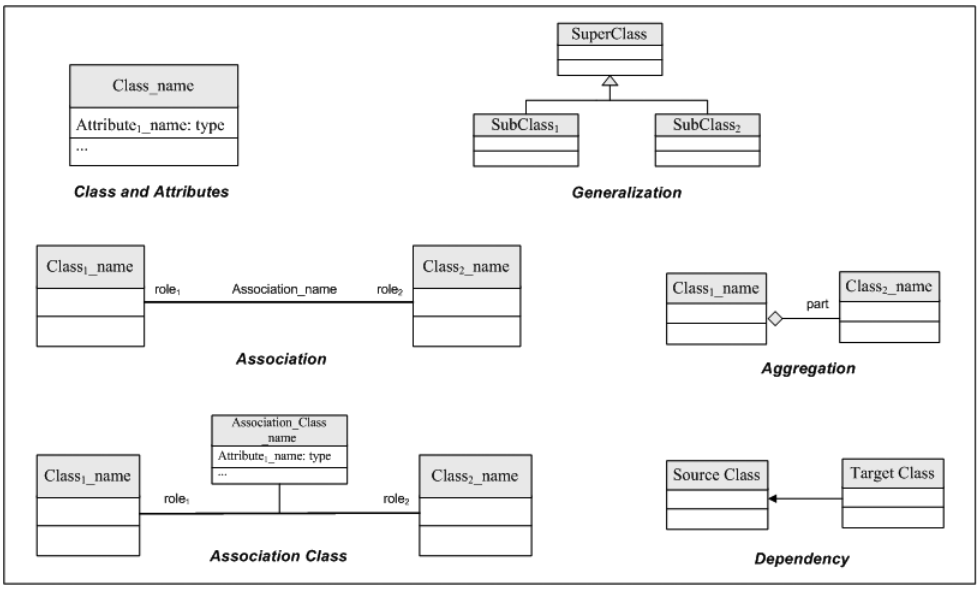
CLASS DIAGRAM

AIM :-

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

THEORY

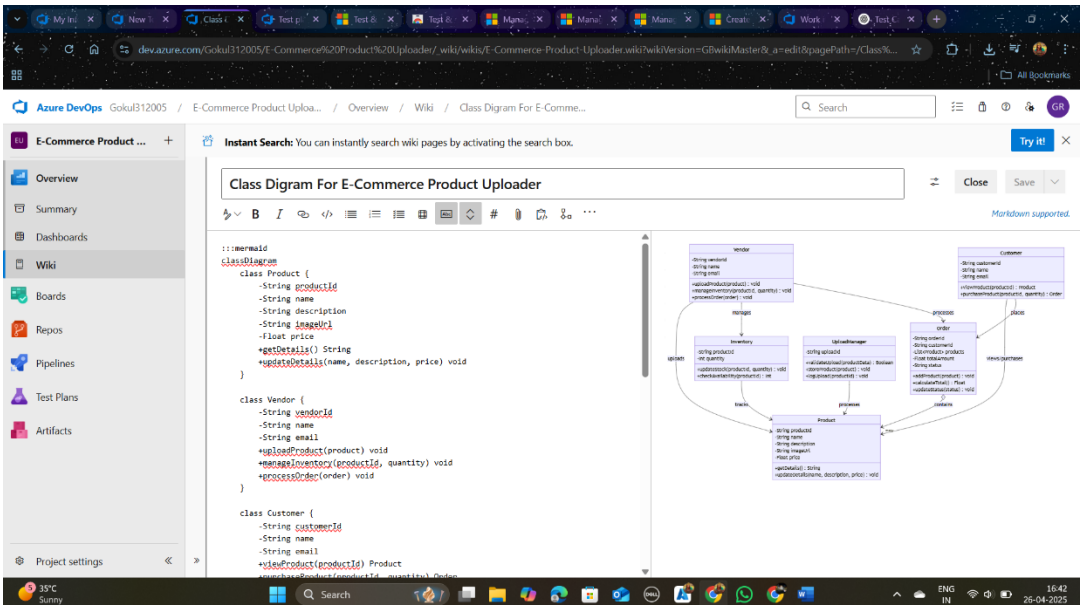
A UML class diagram is a visual tool that represents the structure of a system by showing its classes, attributes, methods, and the relationships between them.



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

```
:::mermaid
classDiagram
    class Product {
        -String productId
        -String name
        -String description
        -String imageUrl
        -Float price
        +getDetails() String
        +updateDetails(name, description, price)
void
    }
    class Vendor {
        -String vendorId
        -String name
        -String email
        +uploadProduct(product) void
        +manageInventory(productId, quantity) void
        +processOrder(order) void
    }
    class Customer {
        -String customerId
        -String name
        -String email
        +viewProduct(productId) Product
        +purchaseProduct(productId, quantity) Order
    }
    class Order {
        -String orderId
```

```
    -String customerId

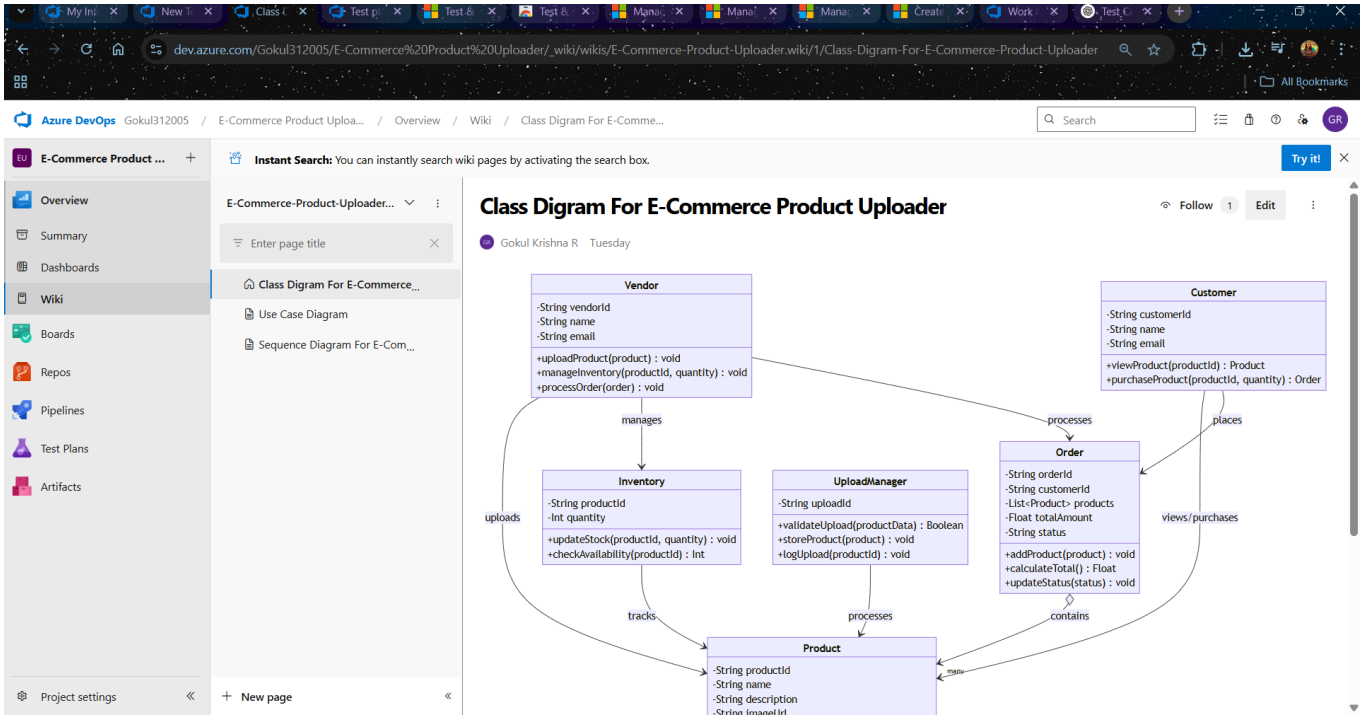
    -List~Product~ products
    -Float totalAmount
    -String status
    +addProduct(product) void
    +calculateTotal() Float
    +updateStatus(status) void
}
class Inventory {
    -String productId
    -Int quantity
    +updateStock(productId, quantity) void
    +checkAvailability(productId) Int
}
class UploadManager {
    -String uploadId
    +validateUpload(productData) Boolean
    +storeProduct(product) void
    +logUpload(productId) void
}
%% Relationships
Vendor --> Product : uploads
Vendor --> Inventory : manages
Vendor --> Order : processes
Customer --> Product : views/purchases
Customer --> Order : places
Order o--> "many" Product : contains
Inventory --> Product : tracks
UploadManager --> Product : processes
```Person <|-- Employee  // Inheritance Employee
```
```

<|-- Manager // Inheritance Person --> Employee
: "has a" // Association

Manager --* Employee : "manages" // Composition

Relationship Types

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



Result:

The use case diagram was designed successfully.

EX NO: 7

USECASE DIAGRAM

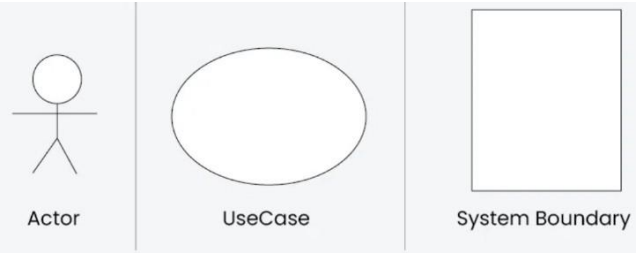
Aim:

Steps to draw the Use Case Diagram using draw.io

Theory:

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

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



Procedure

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

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

Step 2: Upload the Diagram to Azure DevOps

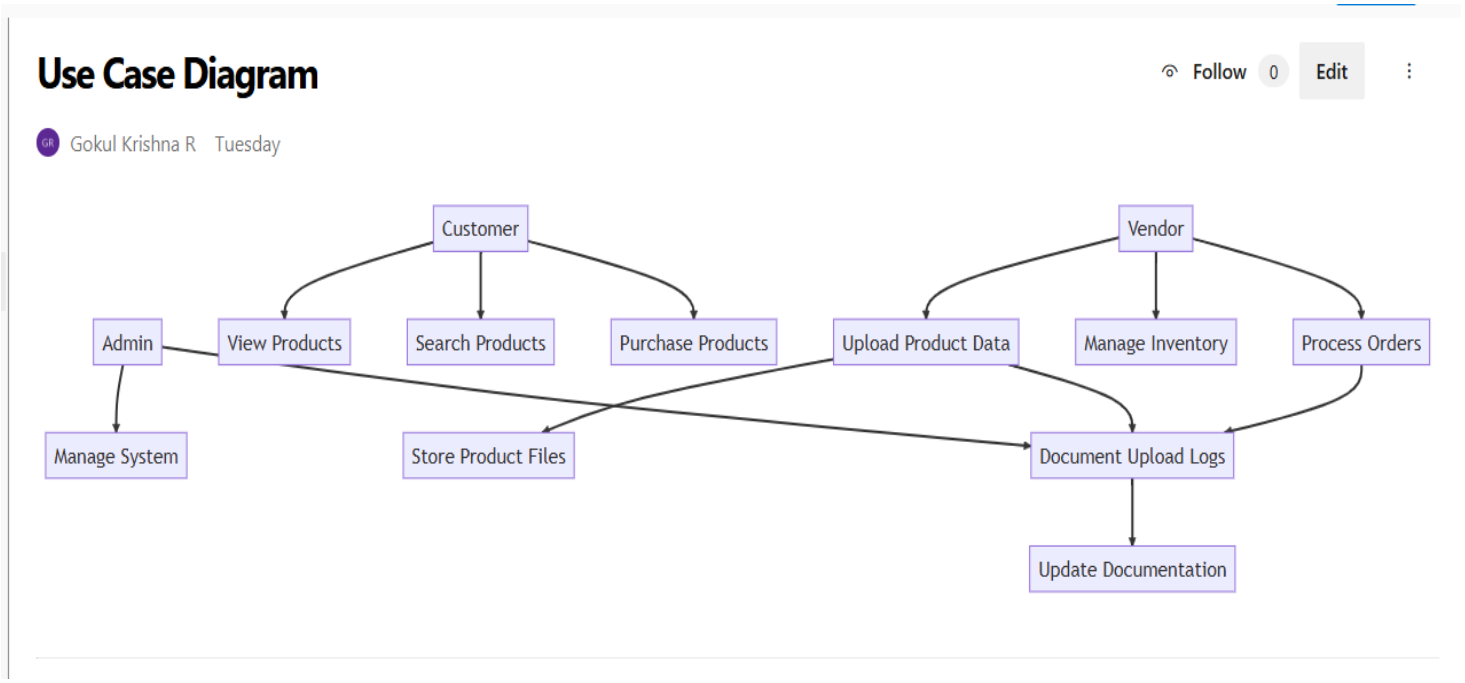
Option 1: Add to Azure DevOps Wiki

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

Option 2: Attach to Work Items in Azure Boards

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

USE CASE DIAGRAM:



Result:

The use case diagram was designed successfully

EX NO. 8

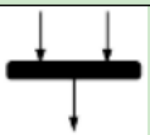
ACTIVITY DIAGRAM

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.

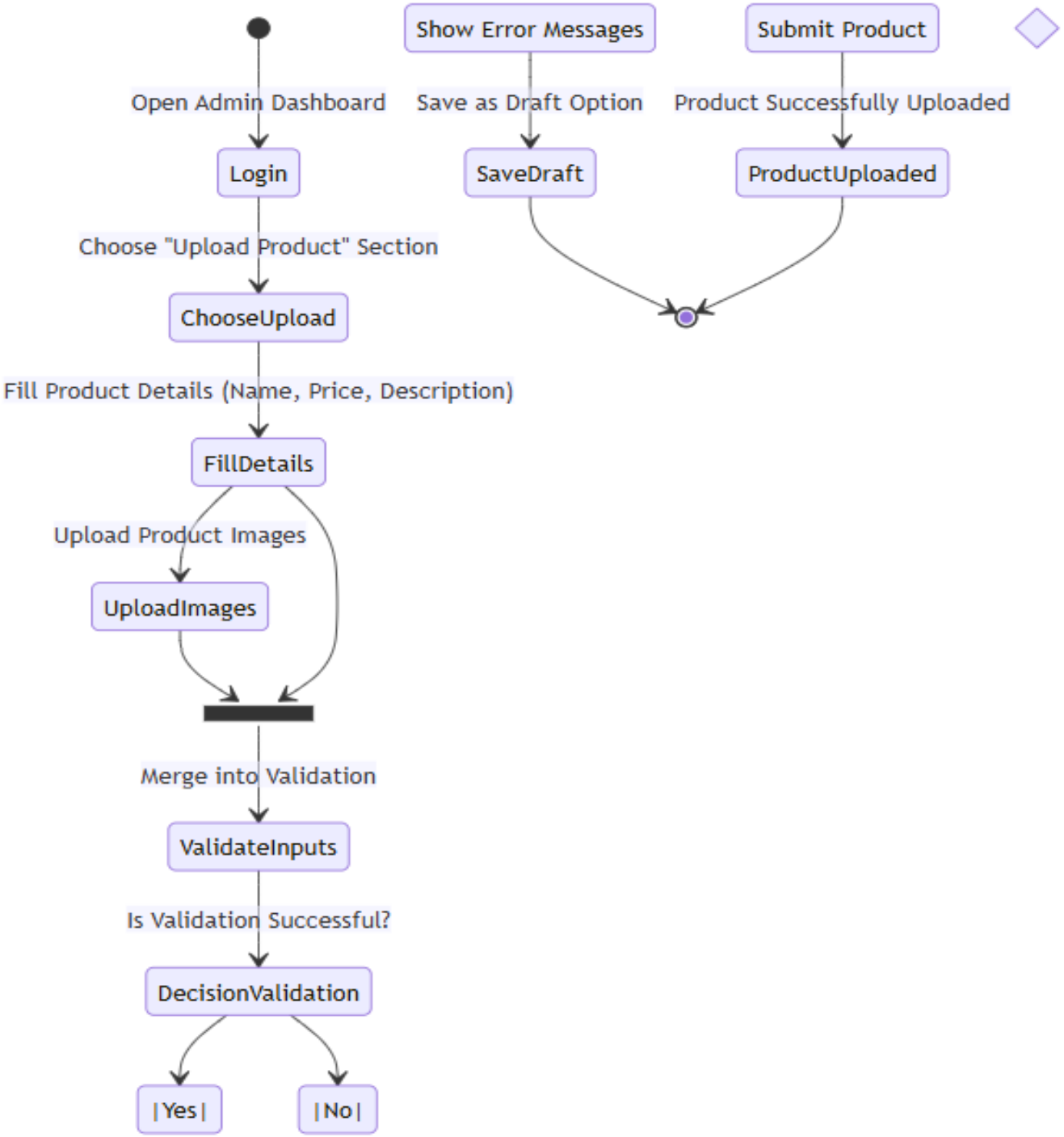
Notations	Symbol	Meaning
Start		Shows the beginning of a process
Connector		Shows the directional flow, or control flow, of the activity
Joint symbol		Combines two concurrent activities and re-introduces them to a flow where one activity occurs at a time
Decision		Represents a decision
Note		Allows the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

Procedure

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

Activity diagram

Gokul Krishna R Just now



Result:

The activity diagram was designed successfully

EX NO. 9

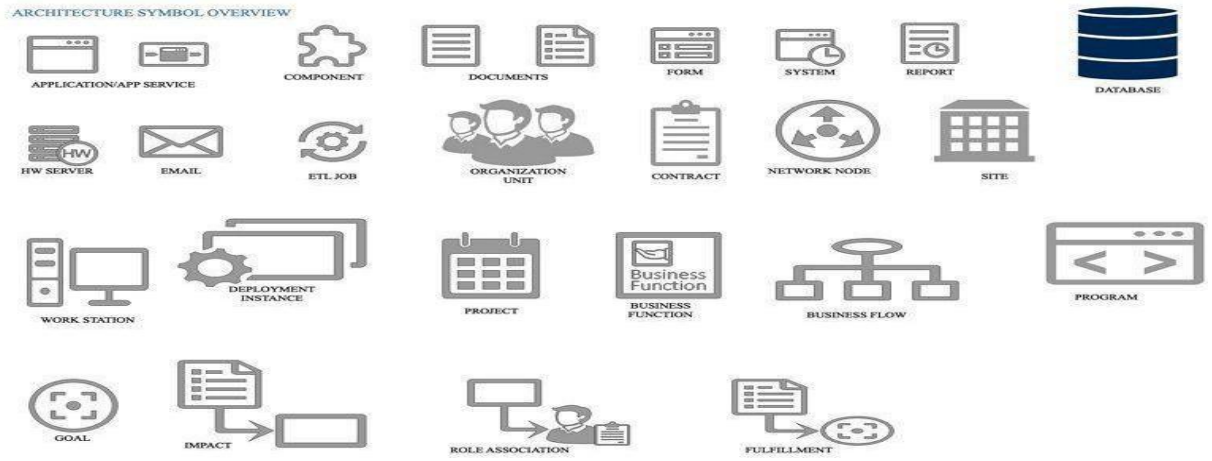
ARCHITECTURE DIAGRAM

Aim:

Steps to draw the Architecture Diagram using draw.io.

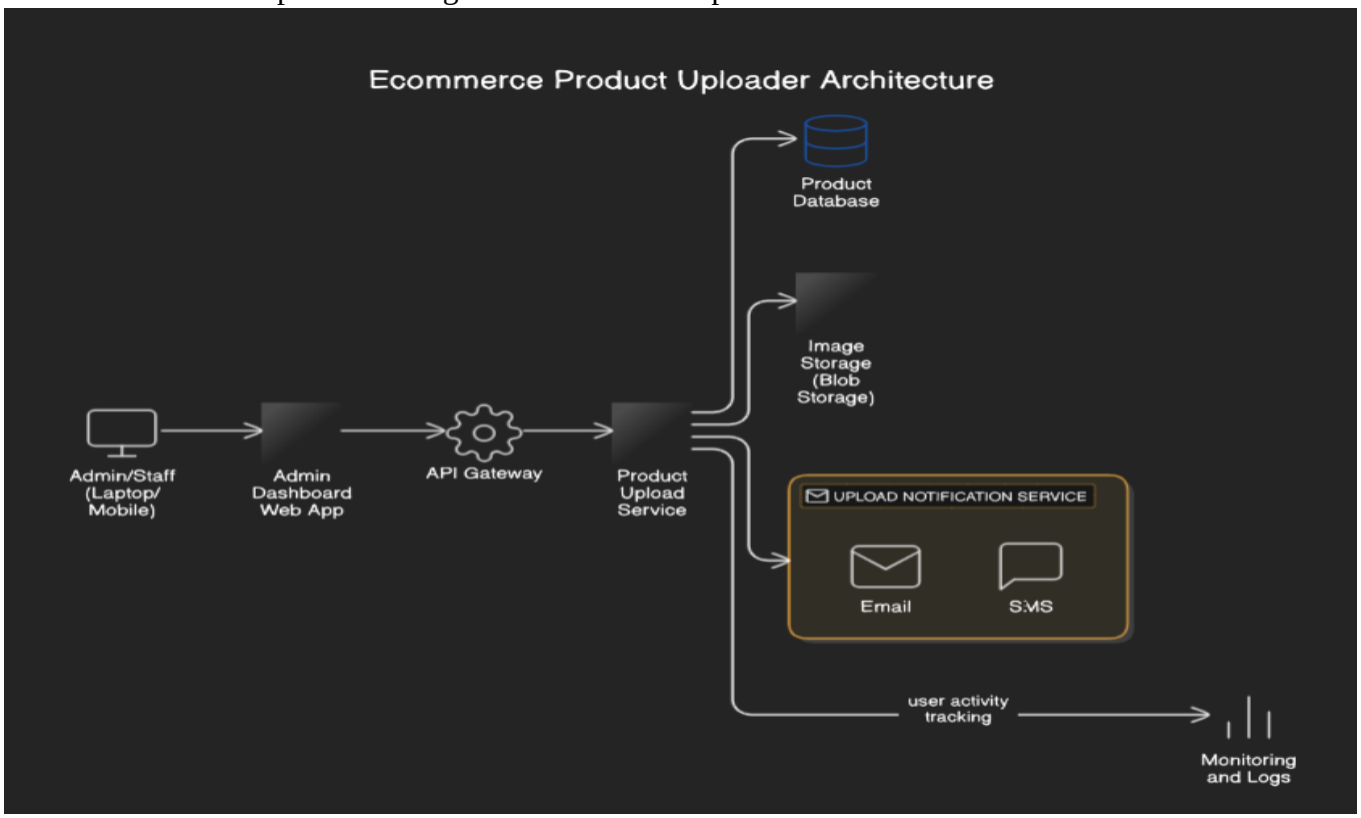
Theory:

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

USER INTERFACE

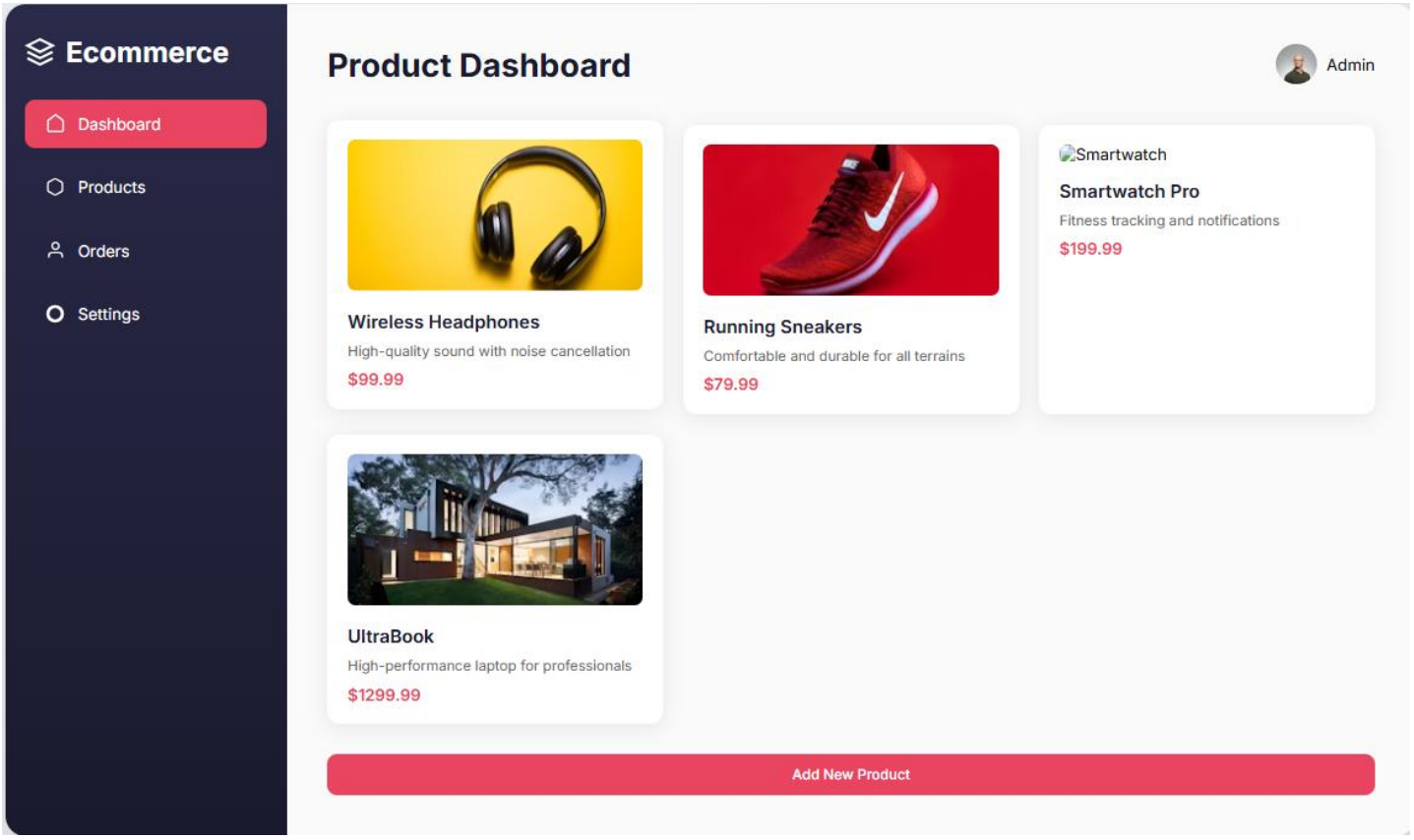
Aim:

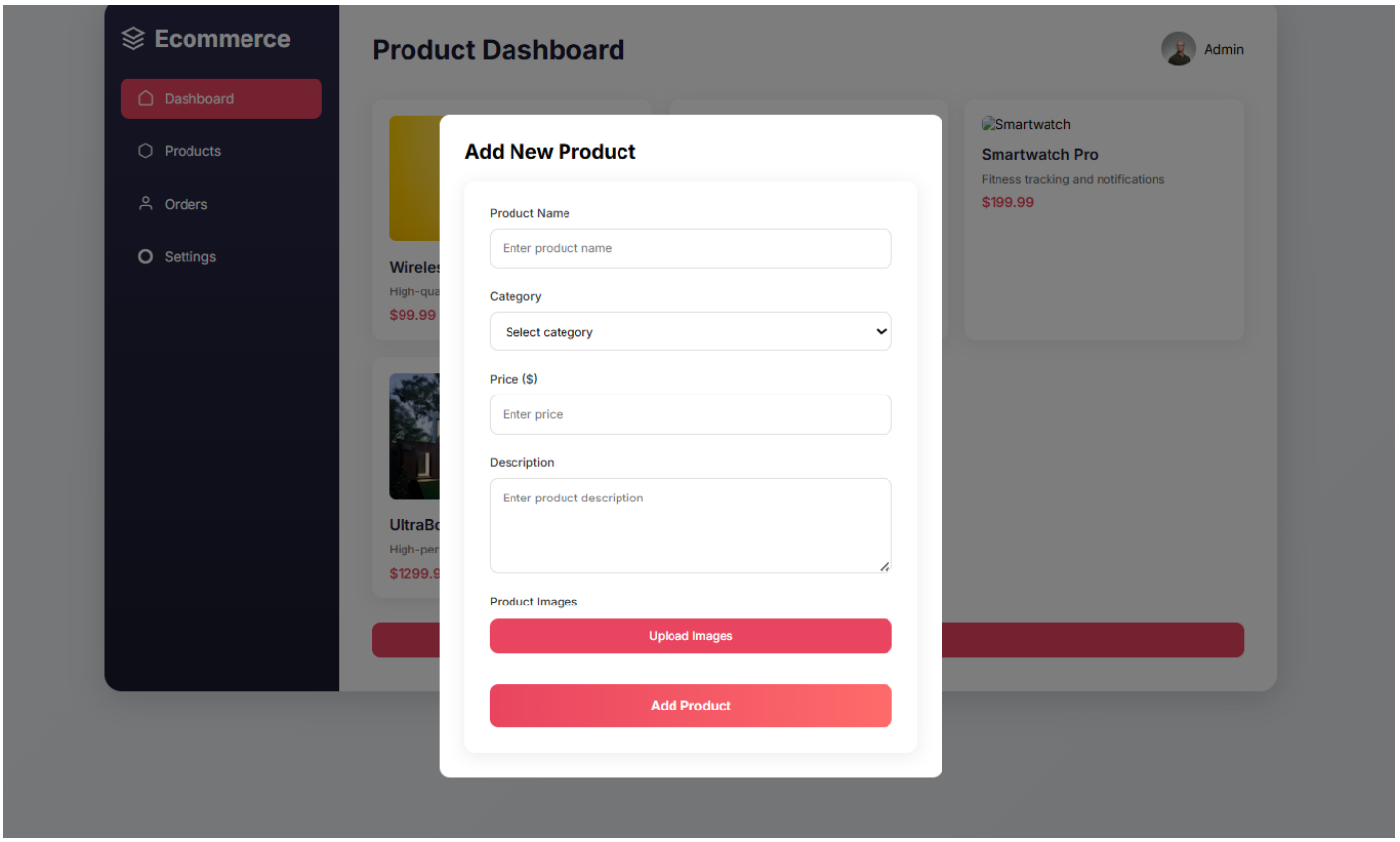
Design User Interface for the given project

User Interface:

User Interface (UI) refers to the visual layout and interactive elements of a software application or website that allow users to interact with the system. It includes components like buttons, menus, input fields, icons, colors, typography, and the overall screen layout.

A well-designed UI ensures that users can easily and efficiently navigate, understand, and use the application to achieve their goals.





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)

- 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.

EX NO. 12

TESTING

a) TESTING-TEST PLANS & TEST CASES

Aim:

Test Plans and Test Case and write two test cases for at least five user stories showcasing the happy path and error scenarios in azure DevOps platform.

Test Planning and Test Case
Test Case Design Procedure

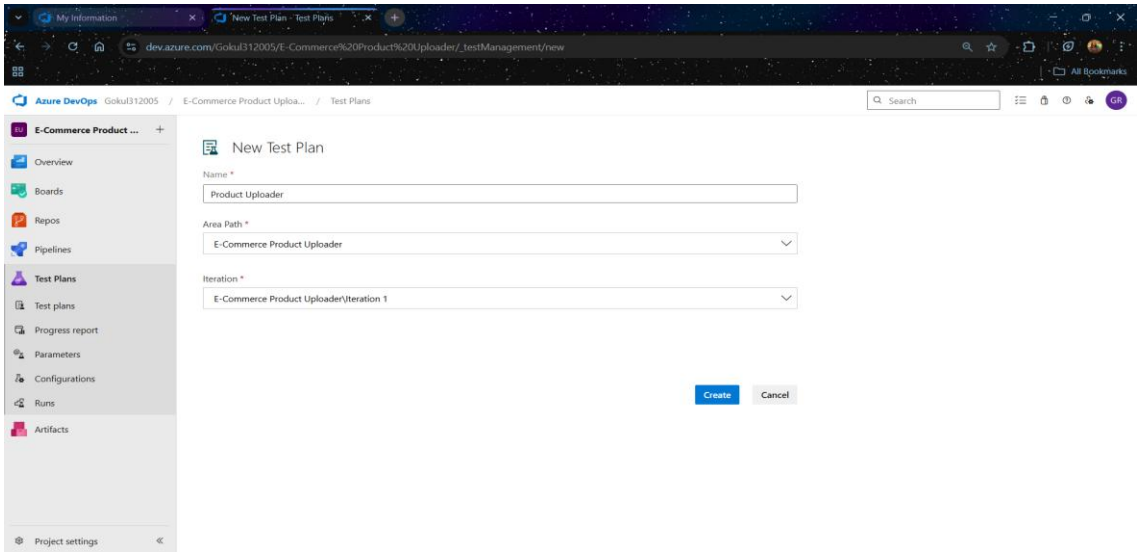
1. **Understand Core Features of the Application**
    - User Signup & Login
    - Viewing and Managing Playlists
    - Fetching Real-time Metadata
    - Editing playlists (rename, reorder, record)
    - Creating smart audio playlists based on categories (mood, genre, artist, etc.)
2. **Define User Interactions**
    - Each test case simulates a real user behaviour (e.g., logging in, renaming a playlist, adding a song).
3. **Design Happy Path Test Cases**
    - Focused on validating that all features function as expected under normal conditions.
    - Example: User logs in successfully, adds item to playlist, or creates a category-based playlist.
4. **Design Error Path Test Cases**
    - Simulate negative or unexpected scenarios to test robustness and error handling.
    - Example: Login fails with invalid credentials, save fails when offline, no recommendations found.
5. **Break Down Steps and Expected Results**
    - Each test case contains step-by-step actions and a corresponding expected outcome.
    - Ensures clarity for both testers and automation scripts.
6. **Use Clear Naming and IDs**
    - Test cases are named clearly (e.g., TC01 – Successful Login, TC10 – Save Playlist Fails).
    - Helps in quick identification and linking to user stories or features.
7. **Separate Test Suites**
    - Grouped test cases based on functionality (e.g., Login, Playlist Editing, Recommendation System).

- Improves organization and test execution flow in Azure DevOps.

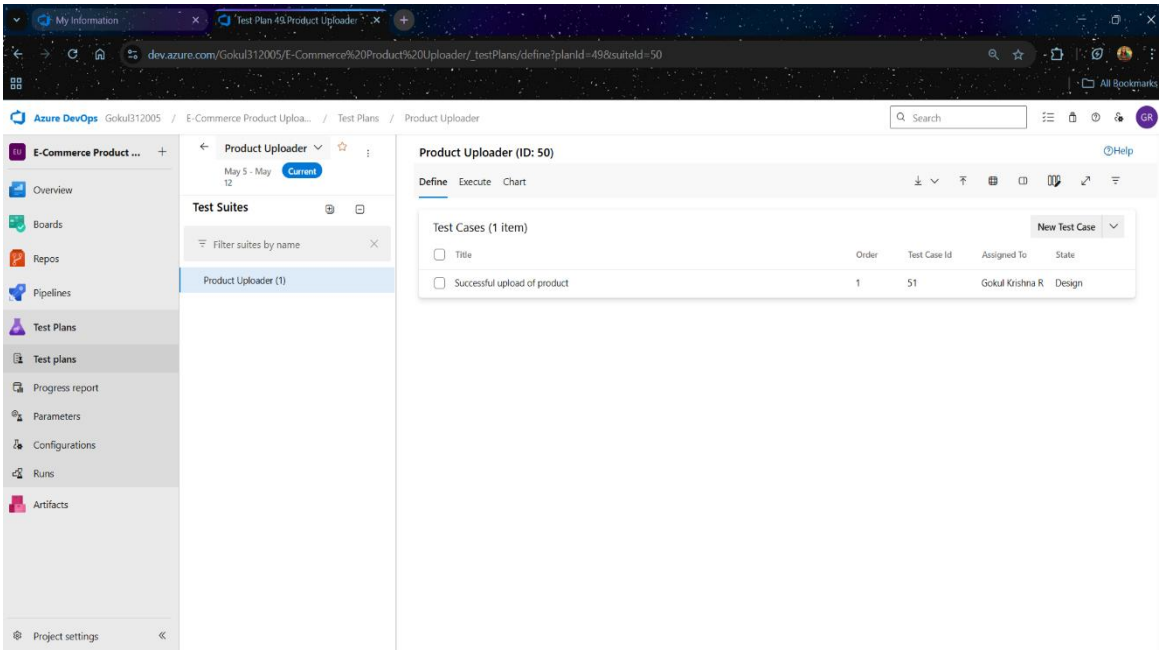
8. Prioritize and Review

- Critical user actions are marked high-priority.
- Reviewed for completeness and traceability against feature requirements.

1.New test plan



2.Test suite



3.Test case

Give two test cases for at least five user stories showcasing the happy path and error scenarios in Azure DevOps platform.

E-Commerce Product Uploader – Test Plans

USER STORIES

- As a seller, I want to upload a new product with complete details (ID: 101).
- As a seller, I should be able to see all my listed products (ID: 102).
- As a seller, I should be notified of upload success or failure (ID: 103).
- As a seller, I should be able to edit product information (ID: 104).
- As a seller, I should not be able to upload a product with missing mandatory fields (ID: 105).

Test Suites

Test Suit: TS01 – Product Upload (ID: 106)

1. TC01 – Successful Product Upload

- **Action:**
  - Go to the product upload page.
  - Fill in product name, description, price, image, category, and stock quantity.
  - Click “Upload Product”.
- **Expected Results:**
  - Product form is submitted successfully.
  - Notification "Product uploaded successfully" is displayed.
  - Product appears in seller’s product list.
- **Type:** Happy Path

2. TC02 – Upload with Missing Fields

- **Action:**
  - Go to the product upload page.
  - Leave the “Product Name” and “Price” fields empty.
  - Click “Upload Product”.
- **Expected Results:**
  - Validation fails.
  - Error message "Product Name and Price are required" is shown.
  - Product is not uploaded.
- **Type:** Error Path

3. TC03 – Upload with Invalid Image Format

- **Action:**
  - Upload a text file instead of a product image.
  - Click “Upload Product”.
- **Expected Results:**
  - Image validation fails.
  - Error “Only image formats (jpg, png) are allowed” is shown.
- **Type:** Error Path

4. TC04 – Upload with Duplicate Product Name

- **Action:**
  - Enter a product name that already exists in the seller's list.
  - Fill out the remaining details and upload.
- **Expected Results:**
  - System accepts submission.
  - Warning "This product already exists, do you want to continue?" is shown.
- **Type:** Error Path (with optional override)

Test Suit: TS02 – View & Edit Products (ID: 107)

1. TC05 – View Uploaded Products

- **Action:**
  - Log in as a seller.
  - Navigate to "My Products".
- **Expected Results:**
  - All uploaded products are listed with name, price, and image.
- **Type:** Happy Path

2. TC06 – Edit Existing Product

- **Action:**
  - Select a product and click "Edit".
  - Change the price and stock quantity.
  - Click "Save Changes".
- **Expected Results:**
  - Product updates are saved.
  - Message "Product updated successfully" is shown.
- **Type:** Happy Path

3. TC07 – Edit with Invalid Price

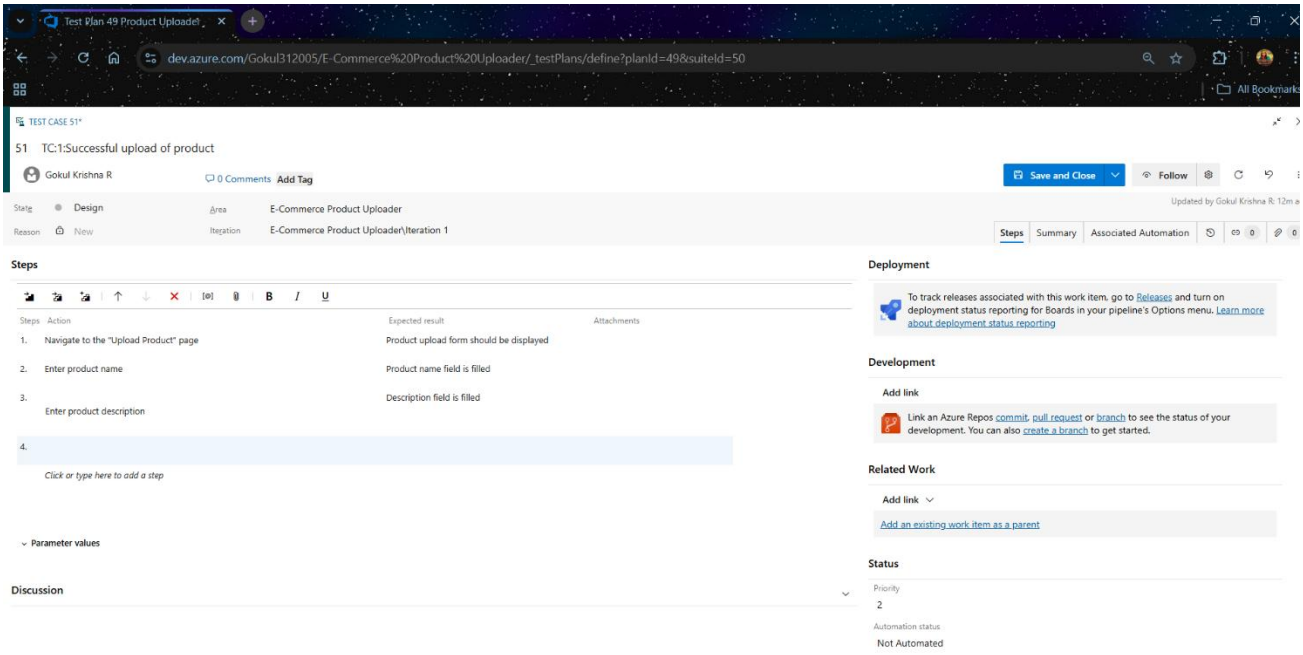
- **Action:**
  - Edit a product and enter a negative number in the Price field.
  - Click "Save Changes".
- **Expected Results:**
  - Validation fails.
  - Error "Price must be a positive number" is shown.
- **Type:** Error Path

Test Suit: TS03 – Upload Notifications (ID: 108)

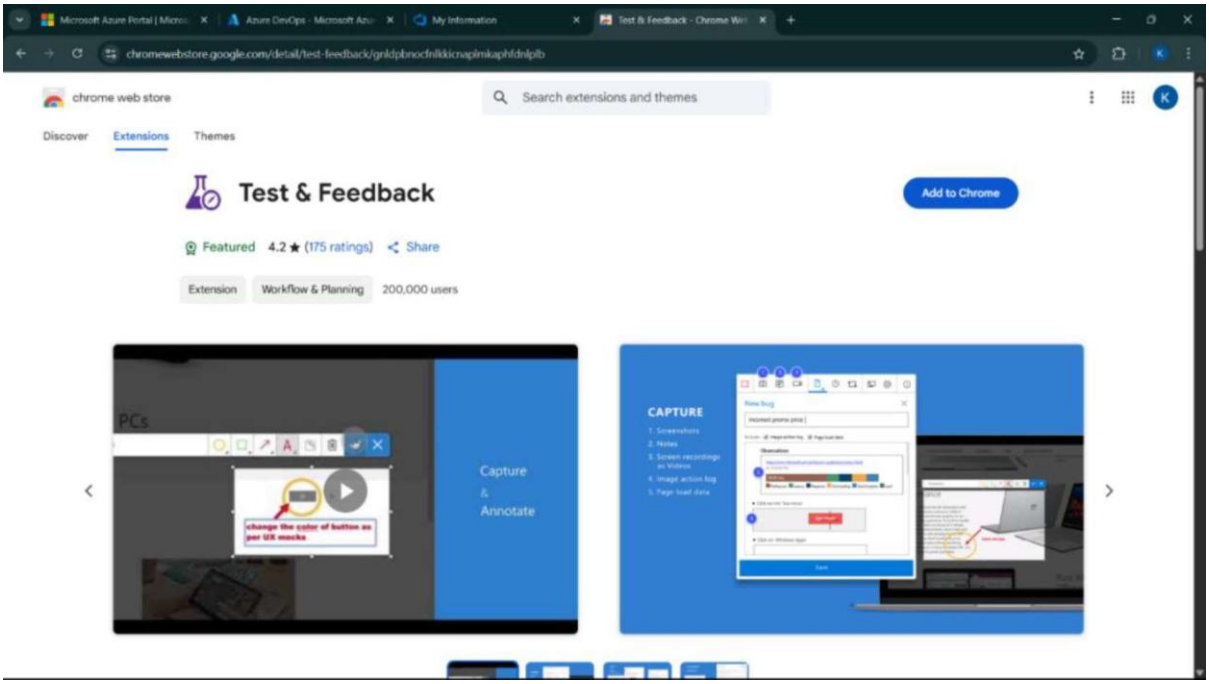
1. TC08 – Upload Failure Notification

- **Action:**
  - Simulate backend failure (e.g., disconnect from server).
  - Try uploading a product.
- **Expected Results:**
  - Upload fails.
  - Message "Upload failed. Please try again later." is shown.
- **Type:** Error Path

Test Cases

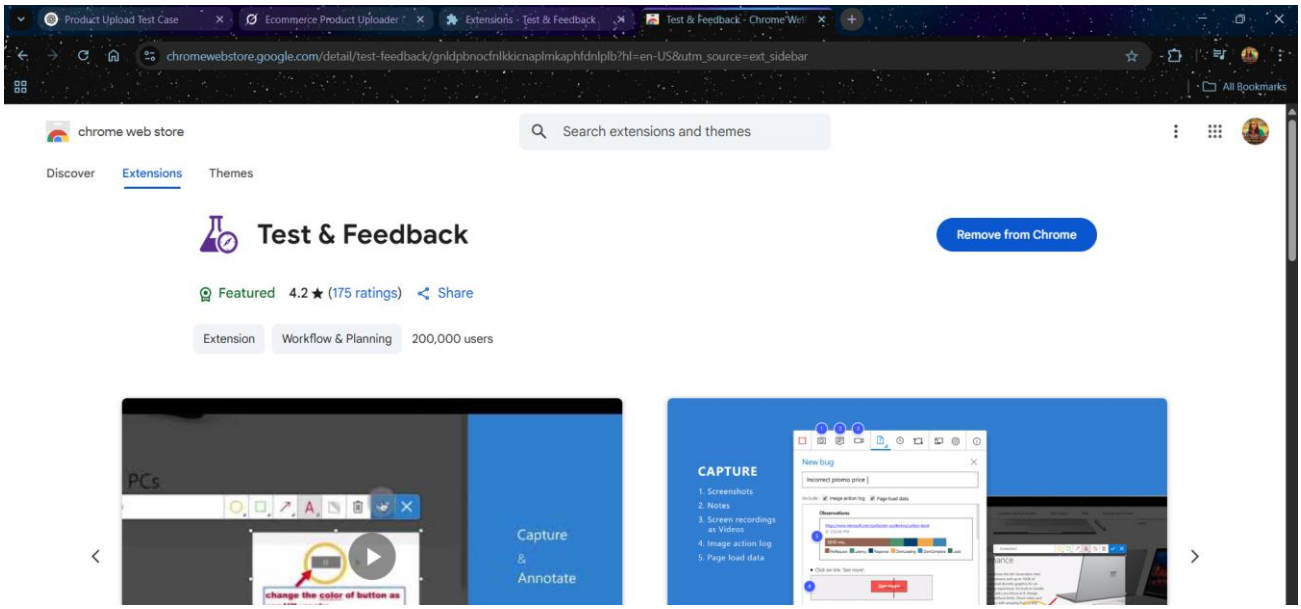


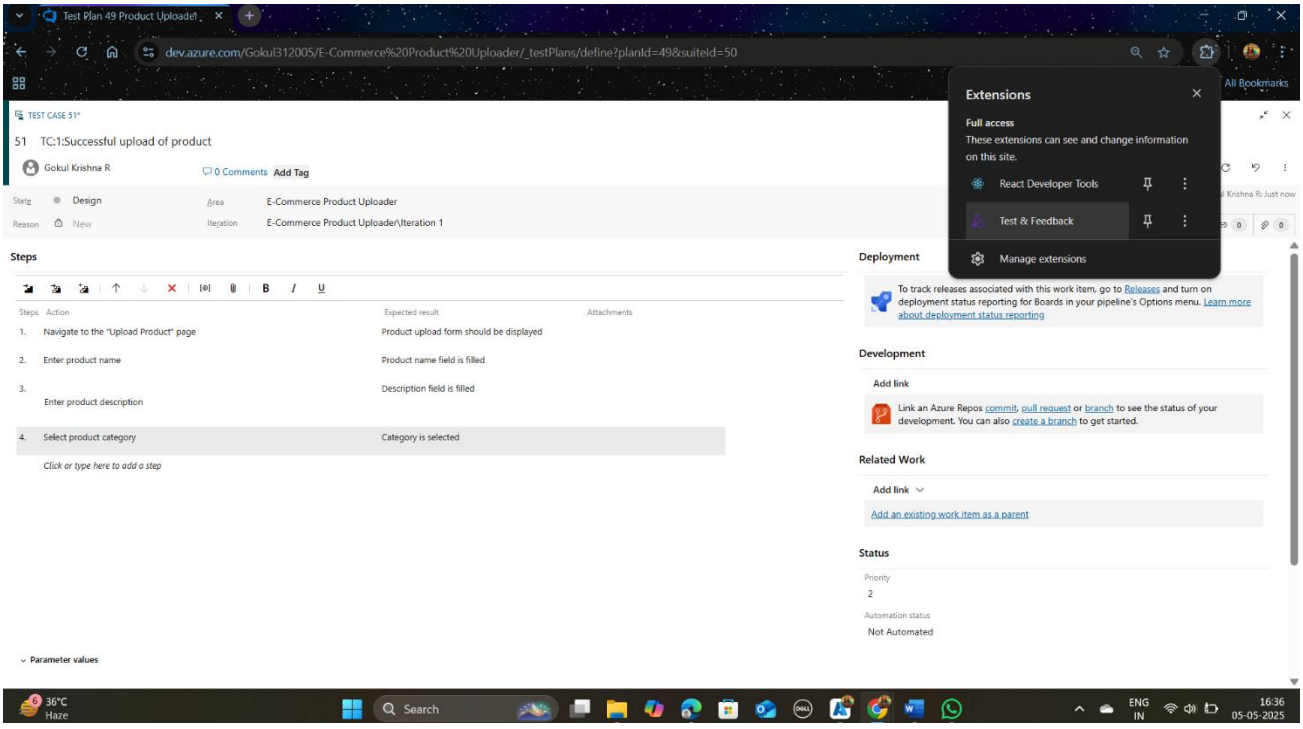
4.Installation of test



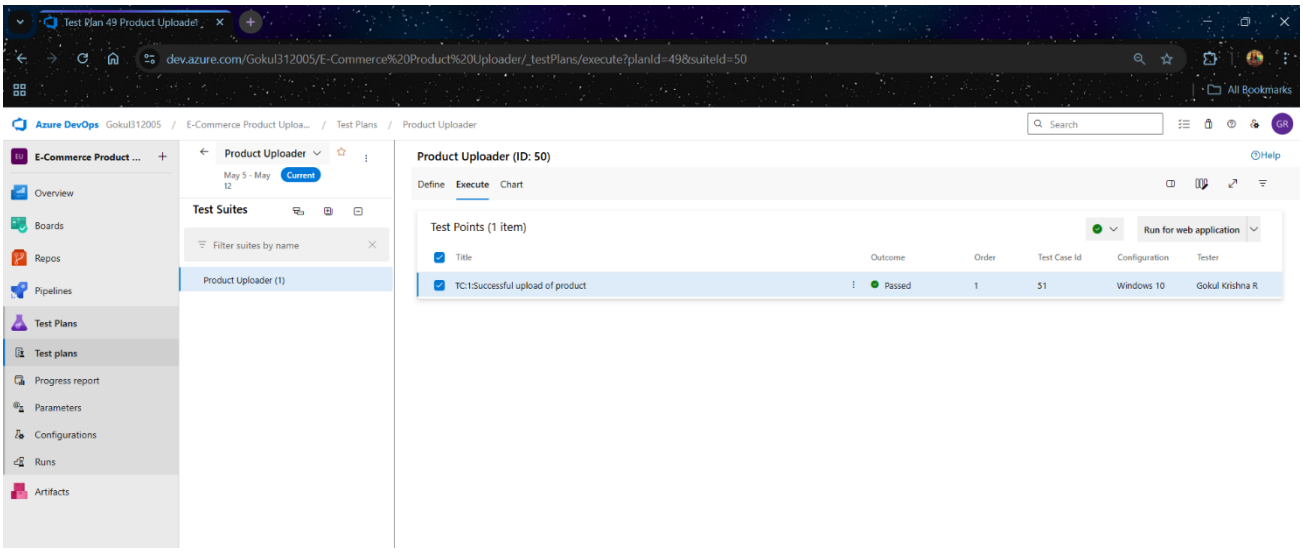
Test and feedback

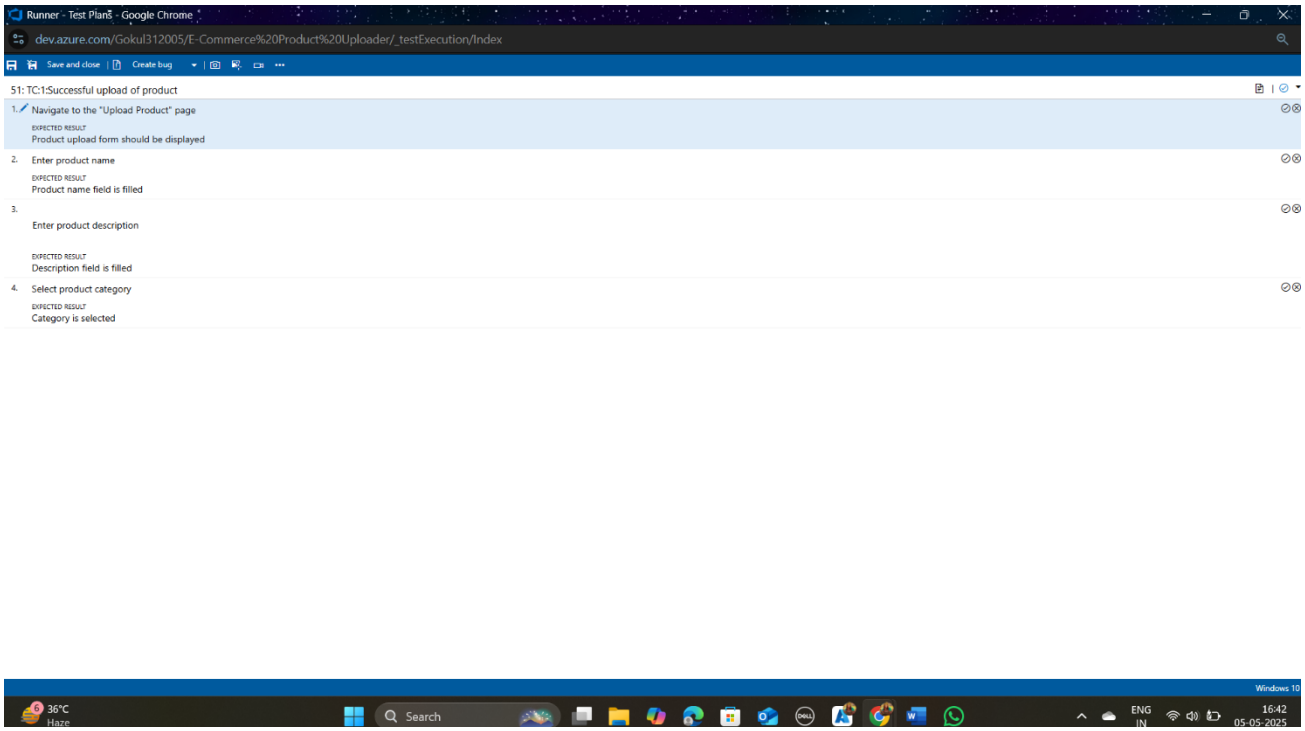
Showing it as an extension



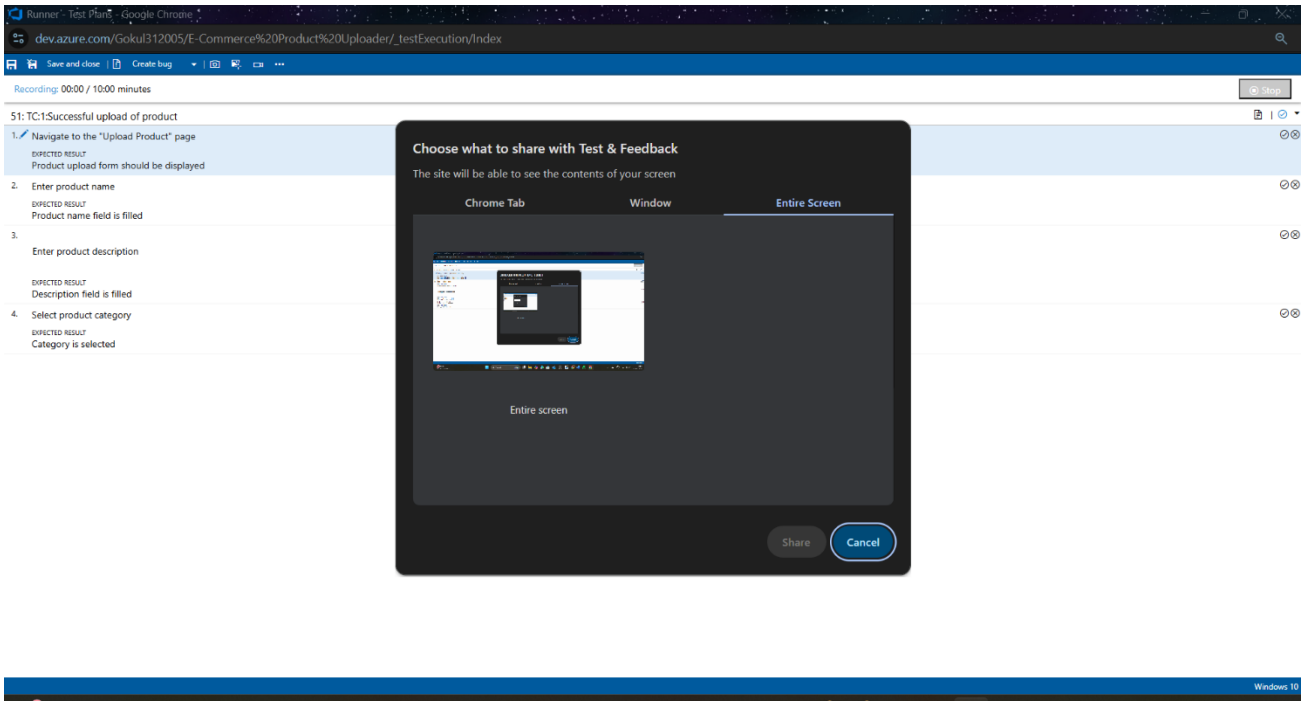


5.Running the test cases

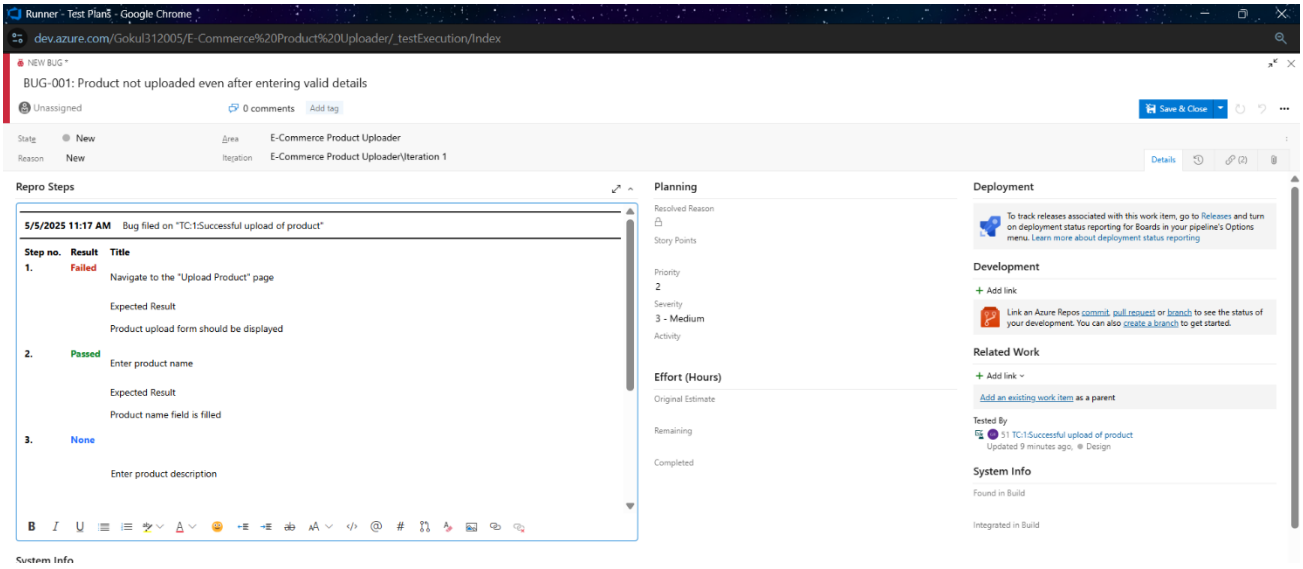


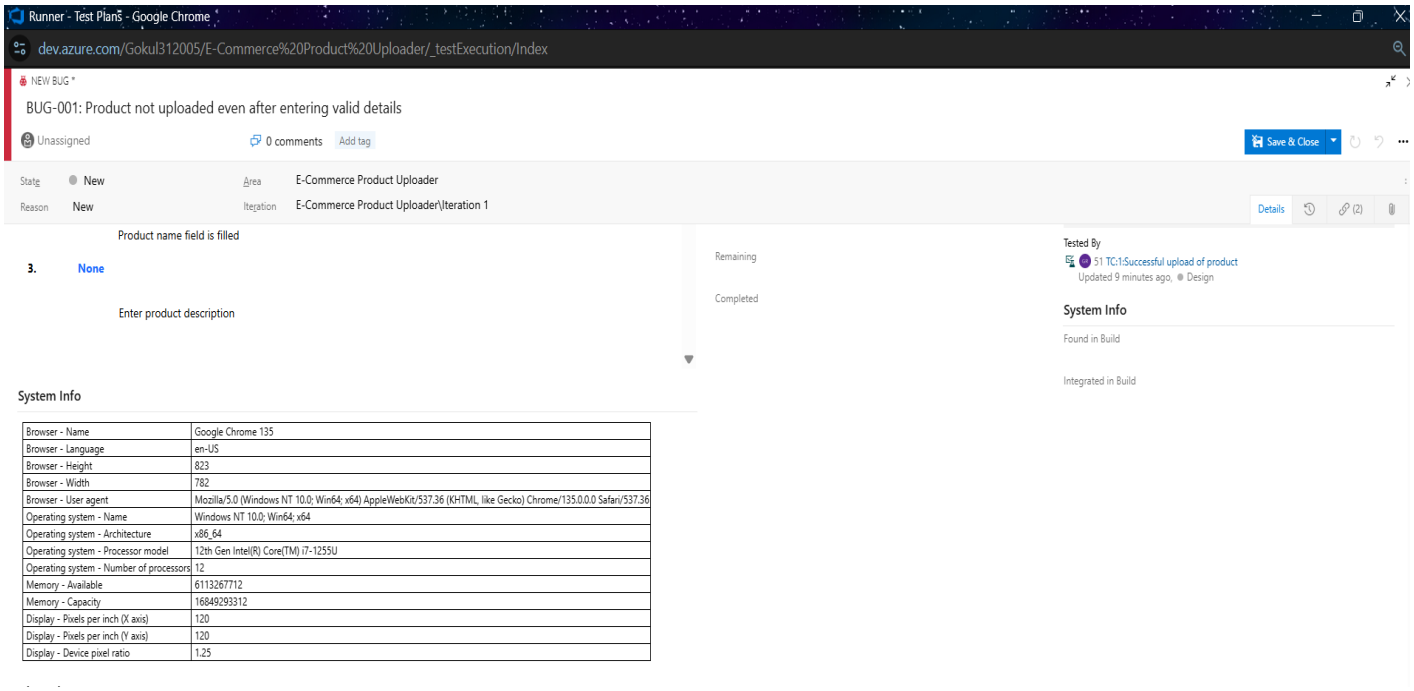


6.Recording the test case

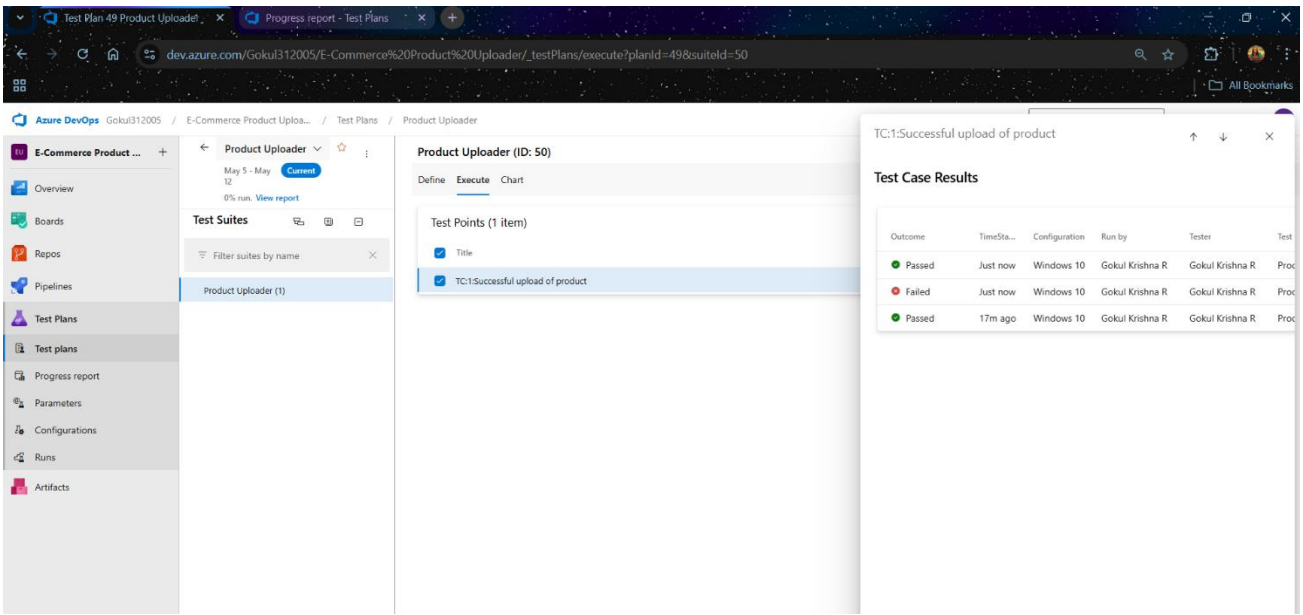


7.Creating the bug

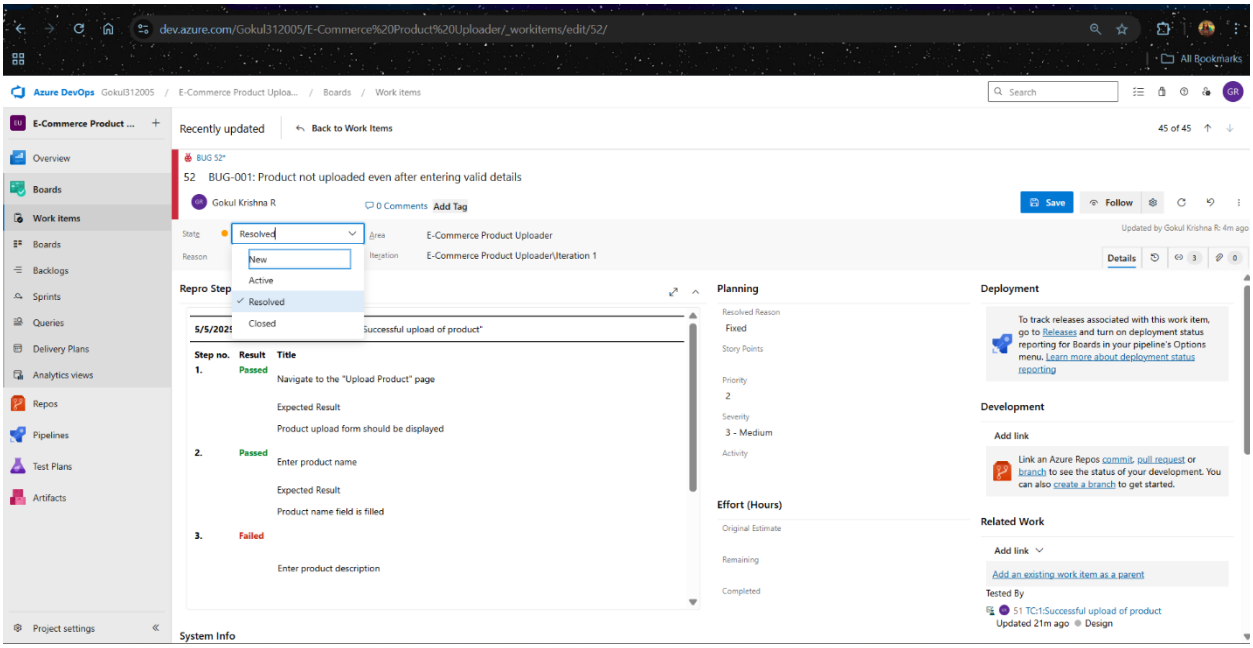




8.Test case results

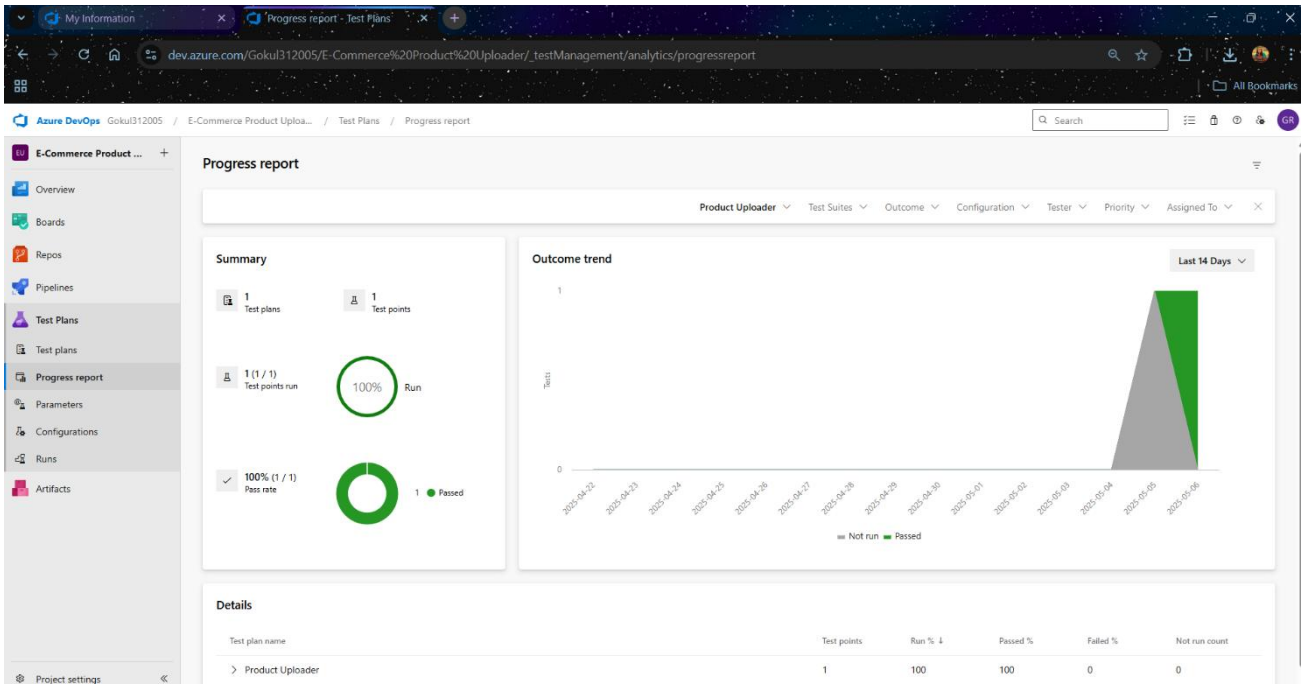


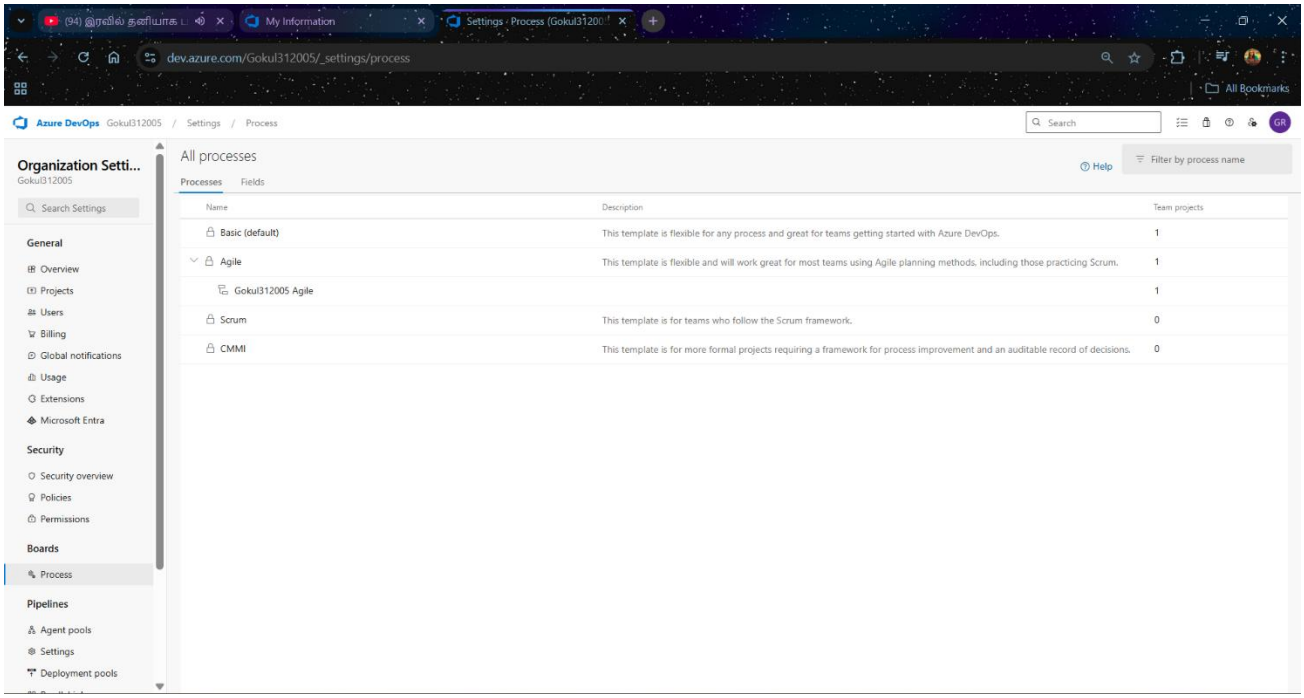
9.Test report summary



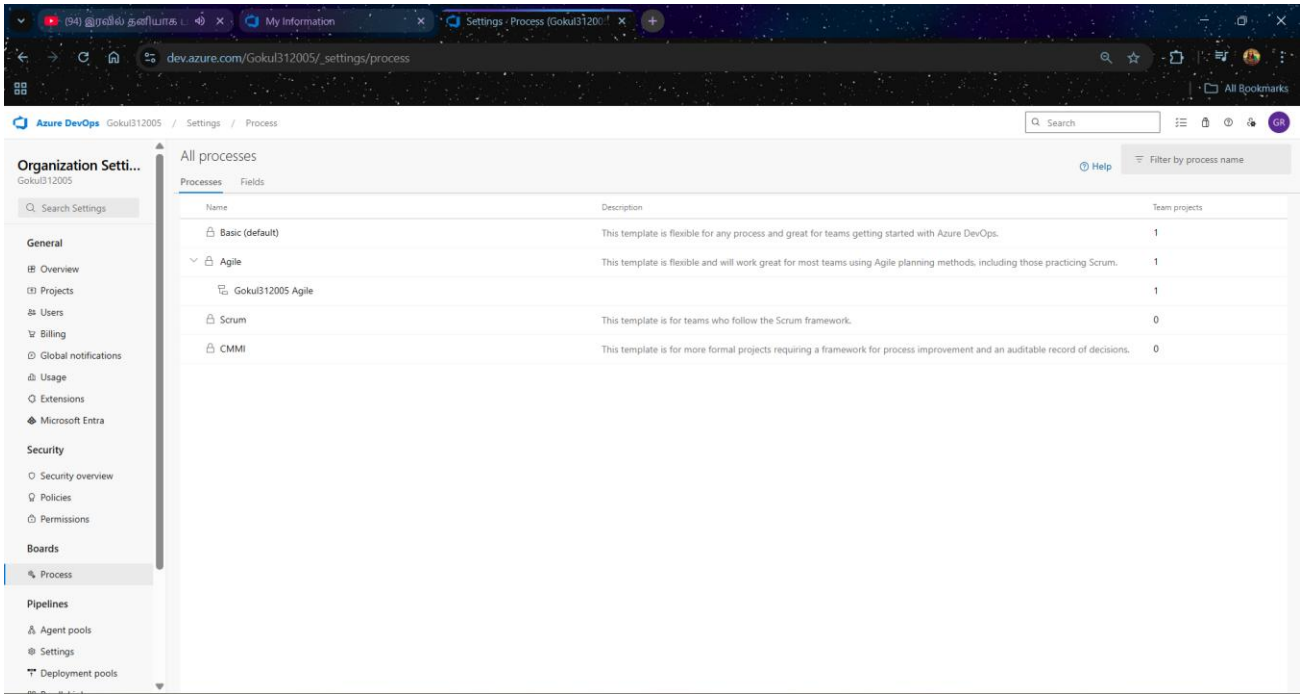
- Assigning bug to the developer and changing state

10.Progress report

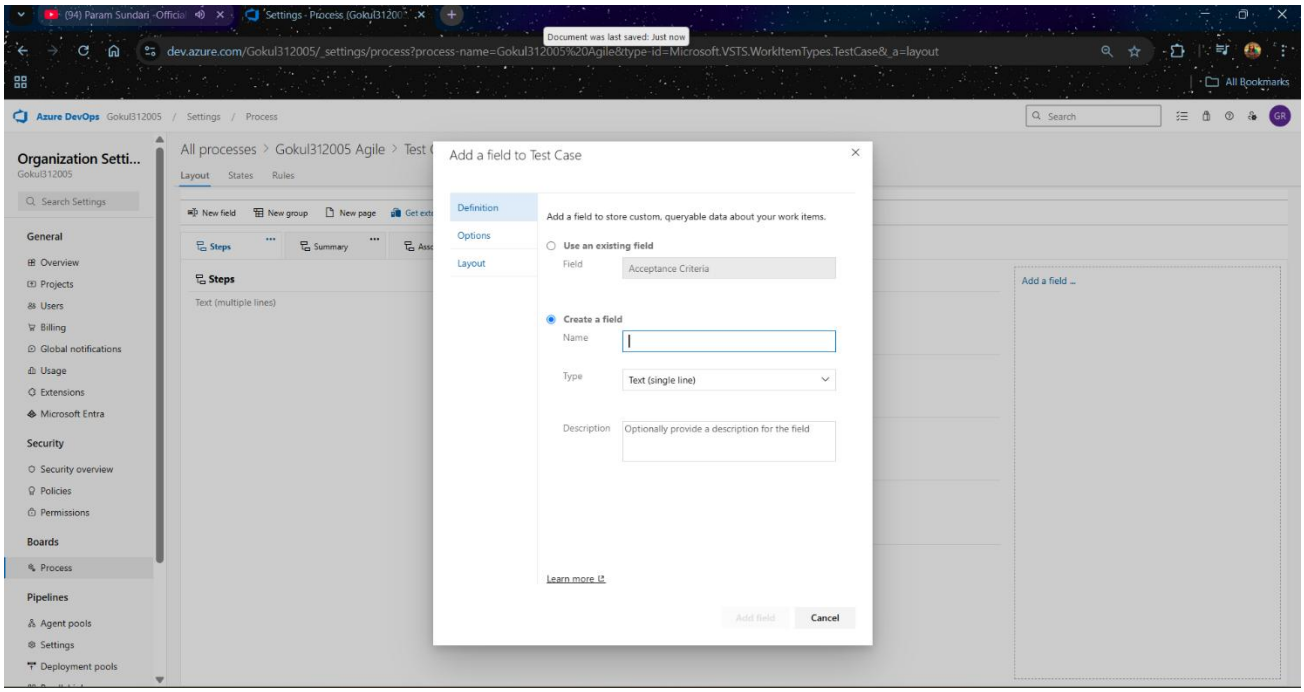


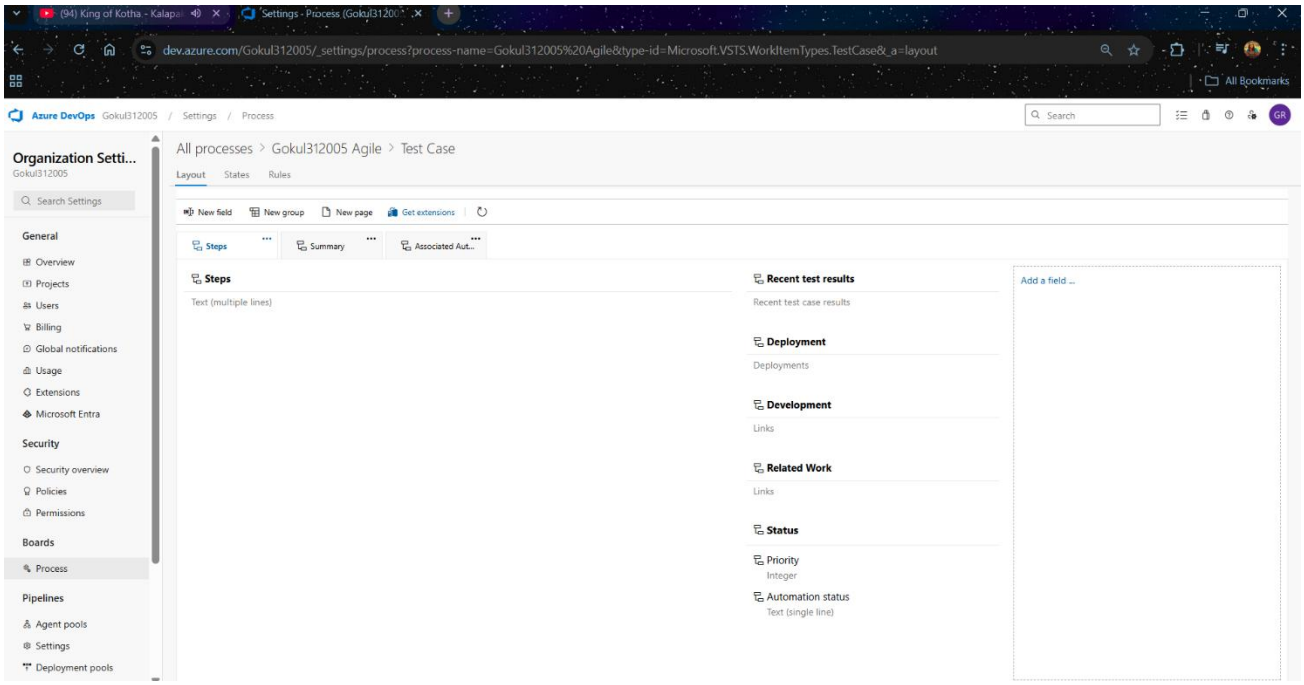


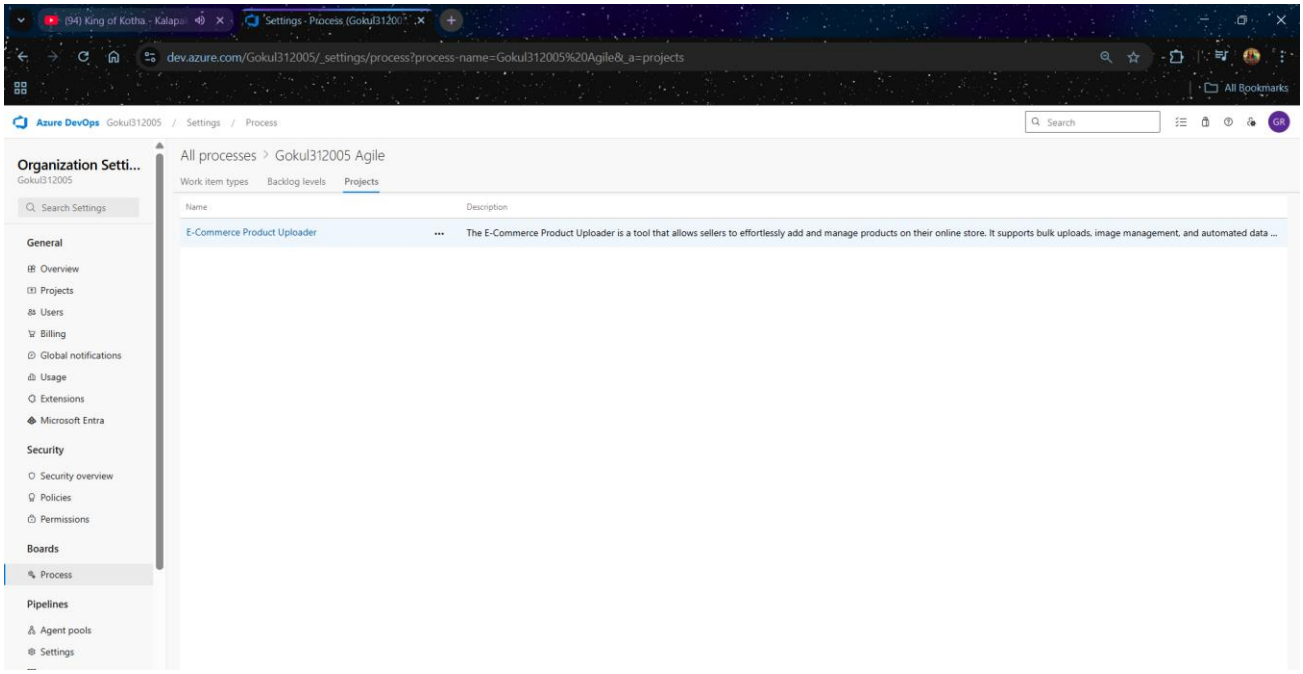
11.Changing the test template



12.View the new test case template







Result:

The test plans and test cases for the user stories is created in Azure DevOps with Happy Path and Error Path

b) Load Testing and Performance Testing

Aim:

To create an Azure Load Testing resource and run a load test to evaluate the performance of a target endpoint.

Load Testing

Steps to Create an Azure Load Testing Resource:

Before you run your first test, you need to create the Azure Load Testing resource:

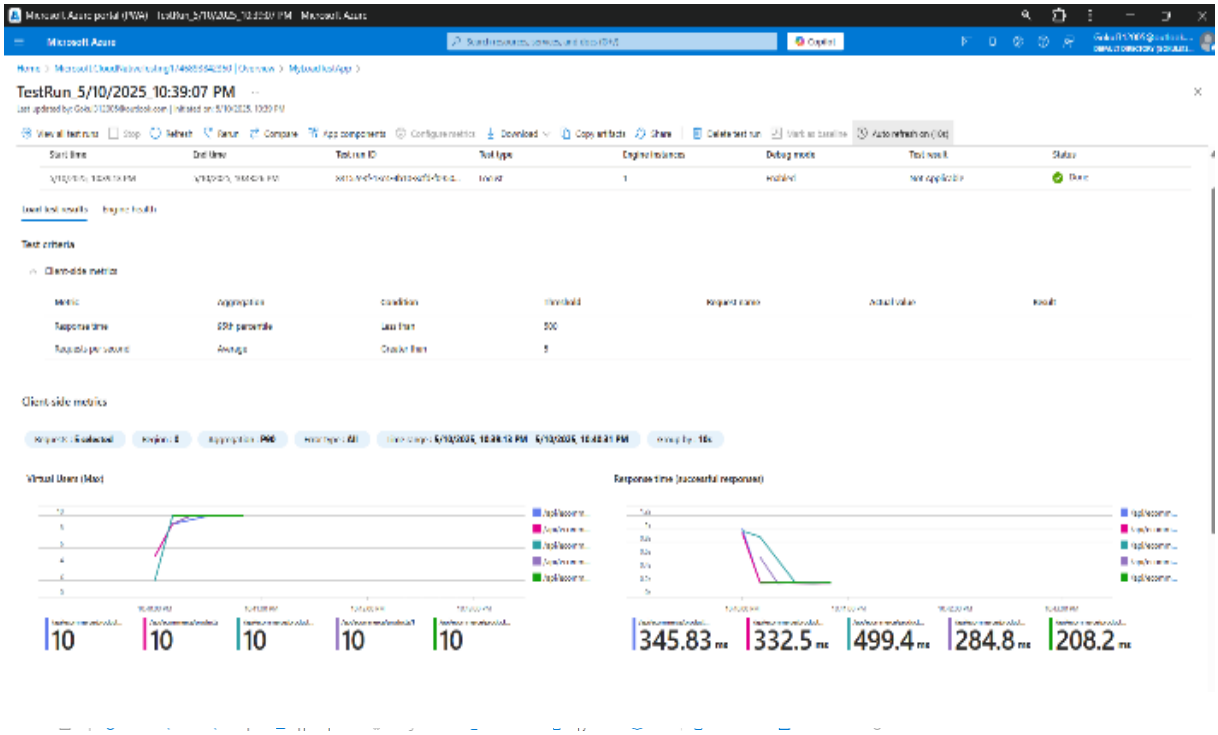
1. Sign in to Azure Portal
   Go to https://portal.azure.com and log in.
2. Create the Resource
   - Go to *Create a resource* → Search for "Azure Load Testing".
   - Select Azure Load Testing and click Create.
3. Fill in the Configuration Details
   - Subscription*:* Choose your Azure subscription.
   - *Resource Group:* Create new or select an existing one.
   - *Name:* Provide a unique name (no special characters).
   - *Location:* Choose the region for hosting the resource.
4. (Optional) Configure tags for categorization and billing.
5. Click Review + Create, then Create.
6. Once deployment is complete, click Go to resource.

Steps to Create and Run a Load Test:

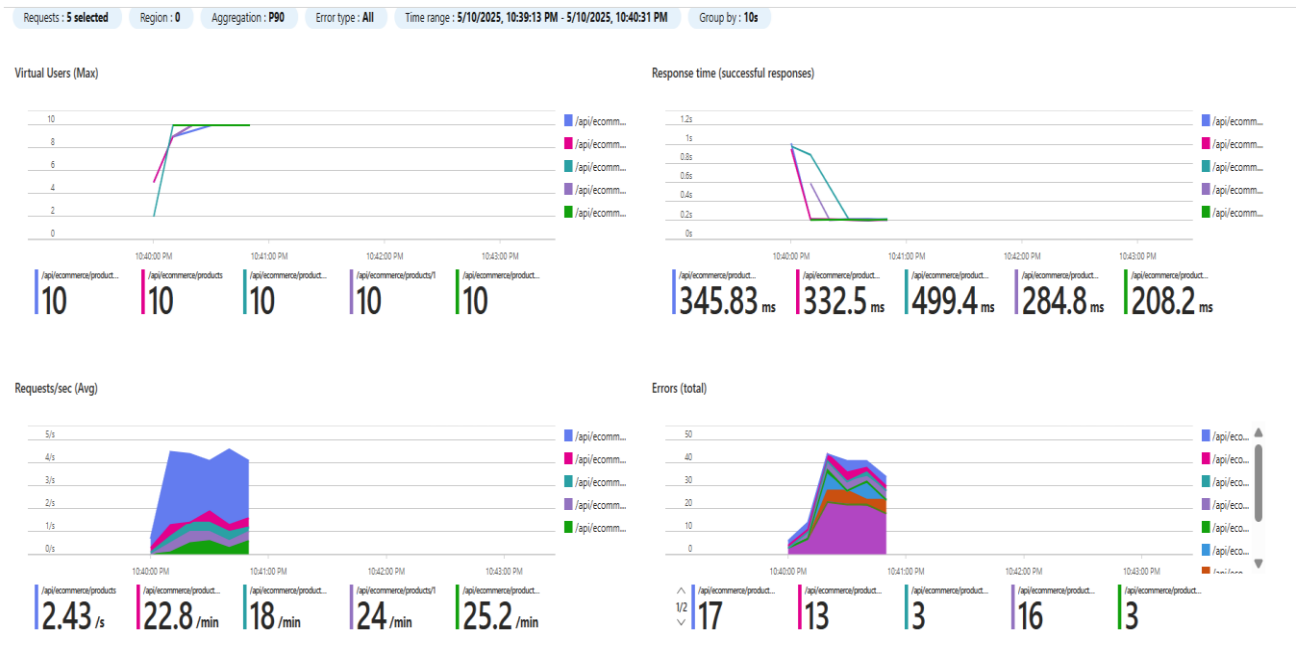
Once your resource is ready:

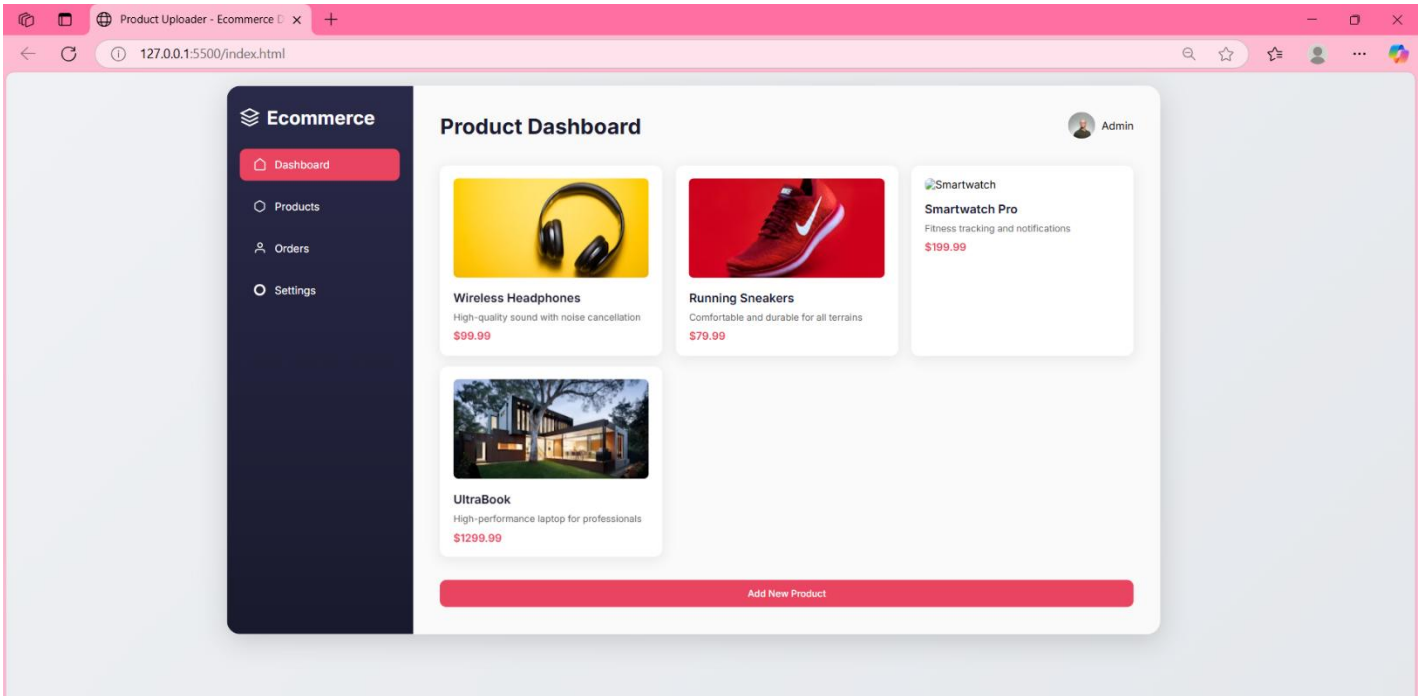
1. Go to your Azure Load Testing resource and click Add HTTP requests > Create.
2. Basics Tab

Load Testing



Z





Result:

Successfully created the Azure Load Testing resource and executed a load test to assess the performance of the specified endpoint.