

A  
PROJECT REPORT ON

# **Online Blog System**

By

**DHRUVI SHERATHIYA (CE - 126) (20CEUOS006)**

**B. Tech CE Semester VI  
Subject: Web Service Development**

**Guided by:**  
Prof. Ankit P. Vaishnav



**Faculty of Technology  
Department of Computer Engineering  
Dharm Singh Desai University**



**Faculty of Technology  
Department of Computer Engineering  
Dharmsinh Desai University**

## **CERTIFICATE**

This is to certify that the practical / **term work** carried out in the  
subject of

**Web Service Development** and recorded in this

journal is the bonafide work of

**DHRUVI SHERATHIYA (CE - 126) (20CEUOS006)**

of B. Tech semester **VI** in the branch of **Computer Engineering**  
during the academic year **2022-2023**.

Prof. Ankit P. Vaishnav  
Assistant Professor,  
Dept. of Computer Engg.,  
Faculty of Technology  
Dharmsinh Desai University,  
Nadiad

Dr. C. K. Bhensdadia,  
Head,  
Dept. of Computer Engg.,  
Faculty of Technology  
Dharmsinh Desai University,  
Nadiad

# Contents

ABSTRACT .....	4
INTRODUCTION .....	5
SOFTWARE REQUIREMENTS SPECIFICATIONS (SRS) .....	6
DESIGN	
ER DIAGRAM .....	8
USE CASE DIAGRAM .....	8
DATA DICTIONARY .....	9
IMPLEMENTATION DETAILS .....	11
TESTING .....	15
SCREENSHOTS .....	21
CONCLUSION .....	26
LIMITATIONS AND FUTURE ENHANCEMENT .....	26
REFERENCES .....	26

## Abstract

The **Blogque** project is designed to provide a platform for users to create, share, and read blog articles. The site includes a user registration and login system to ensure security and privacy of user data. Users can view blog articles posted by other users and can also create, edit, and delete their own blog articles. The site is built with a user-friendly interface, allowing users to easily navigate and search for content. The project is ideal for bloggers who want to showcase their writing skills and connect with like-minded individuals in an online community.

# Introduction

Blogque is built using ASP .NET Web API and React, and provides a platform for users to engage with various topics and interact with like-minded individuals through blogging. Users can easily register, login, and browse through a wide range of blog posts covering diverse topics, including technology, business, travel, lifestyle, and more. The site also enables users to create, edit, and publish their own blog posts, complete with multimedia content and commenting features. User security and privacy are prioritized through robust authentication mechanisms. Overall, the platform offers a seamless and user-friendly experience for individuals seeking to express themselves through writing and connect with a community of like-minded individuals.

## Technology

### Backend

- ASP .NET Core WEB API(C#)

### Frontend

- React.js
- Bootstrap
- CSS

### Database

- MS SQL Server

## Tools

- Visual Studio Code
- Visual Studio
- GitHub
- Any Browse

# Software Requirement Specification (SRS)

## Functional Requirements

### 1. Authentication

#### 1.1 Create Account

Description: Only those who already have accounts on our site are eligible to create new blogs. Yet, anyone may read blogs without logging in. To sign up as a new user, the user must enter their username and password. A unique id would be generated and these details would be kept in a database.

Input: Username, Password

Output: Unique user id

#### 1.2 Login into System

Description: Registered user can login into the site by entering their email address and password. After successful login users would be redirected to the homepage of the site.

Input: Username, password

Output: Home Screen

#### 1.3 Logout

Description: For security reasons, users can log out of their sessions after browsing a website.

Output: Logout successful message.

### 2. Manage Blog

#### 2.1 View Blog

Description: Any user can read the available blogs on this site. There is no need to login for view blogs.

Output: Blogs List

#### 2.2 Create a blog

Description: With this website, users may easily establish new blogs by entering the blog's title, category, banner url, and content.

Input: Blog details

Output: Success message

## 2.3 Edit blog

Description: Author of the blog can edit blog in future if he/she thinks that there was a mistake in it.

Input: Details to be update

Output: Blog List

## 2.4 Delete a blog

Description: On this website, users can delete the blog which was previously created by him/her.

Input: Delete confirmation

## Non-functional Requirements

**N.1: Database:** A data base management system that is available free of cost in the public domain should be used.

**N.2: Data integrity:** Data integrity refers to maintaining and assuring data accuracy and consistency over its entire lifecycle. If this factor is corrupted, data is lost due to a database error.

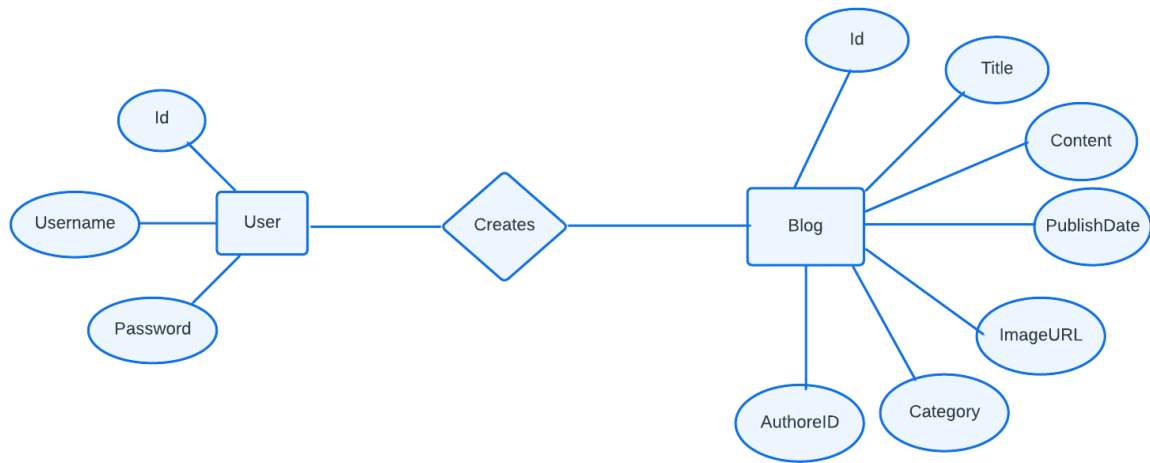
**N.3: Platform:** Windows, Unix and Mac versions of software need to be developed.

**N.4: Web-support:** It should be possible to invoke the query book functionality from any place by using web browser.

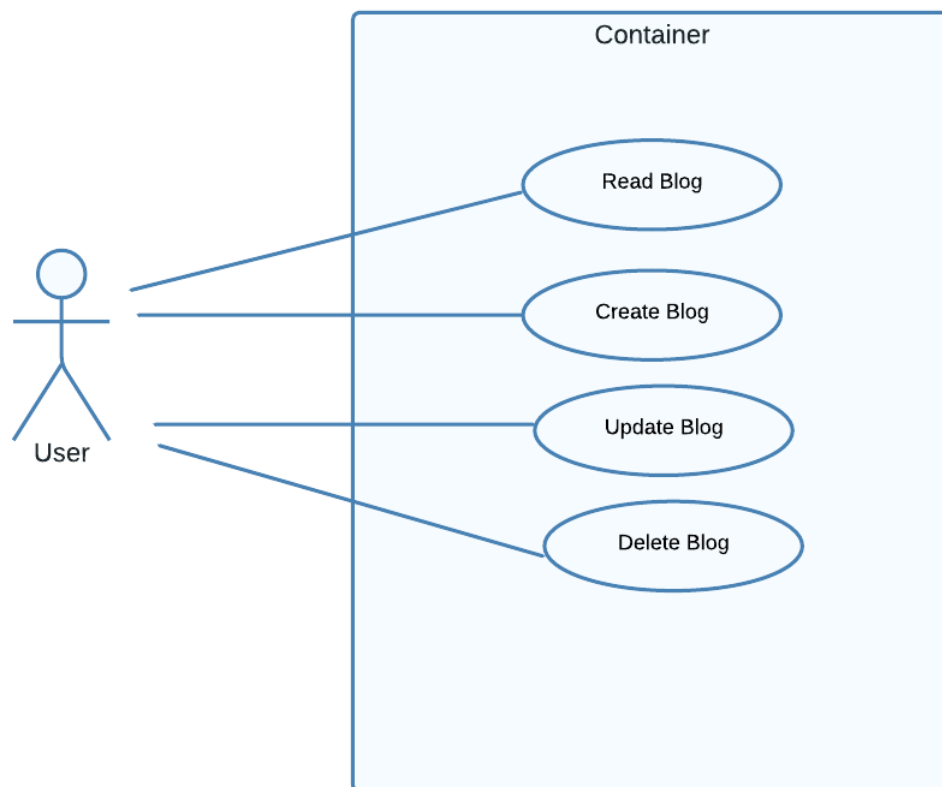
**N.5: Performance:** System performance defines how fast a system can respond to a particular user's action under a certain workload.

# Design

## ER DIAGRAM



## USE CASE DIAGRAM





# Data Dictionary

dbo.Users [Design]				
Update	Script File: dbo.Users.sql			
Name	Data Type	Allow Nulls	Default	
Id	int	<input type="checkbox"/>		
Username	nvarchar(MAX)	<input type="checkbox"/>		
PasswordHash	varbinary(MAX)	<input type="checkbox"/>		
PasswordSalt	varbinary(MAX)	<input type="checkbox"/>		
		<input type="checkbox"/>		

**Keys (1)**  
PK\_Users (Primary Key, Clustered: Id)  
**Check Constraints (0)**  
**Indexes (0)**  
**Foreign Keys (0)**  
**Triggers (0)**

dbo.BlogPosts [Design]				
Update	Script File: dbo.BlogPosts.sql			
Name	Data Type	Allow Nulls	Default	
Id	int	<input type="checkbox"/>		
Title	nvarchar(MAX)	<input type="checkbox"/>		
Content	nvarchar(MAX)	<input type="checkbox"/>		
PublishDate	datetime2(7)	<input type="checkbox"/>		
ImageUrl	nvarchar(MAX)	<input checked="" type="checkbox"/>		
Category	nvarchar(MAX)	<input checked="" type="checkbox"/>		
AuthorId	int	<input type="checkbox"/>		
		<input type="checkbox"/>		

**Keys (1)**  
PK\_BlogPosts (Primary Key, Clustered: Id)  
**Check Constraints (0)**  
**Indexes (1)**  
IX\_BlogPosts\_AuthorId (AuthorId)  
**Foreign Keys (1)**  
FK\_BlogPosts\_Users\_AuthorId (Id)  
**Triggers (0)**

## User Model

```
namespace BlogAPI.Models
{
    11 references
    public class User
    {
        6 references
        public int Id { get; set; }
        5 references
        public string Username { get; set; }
        2 references
        public byte[] PasswordHash { get; set; } = Array.Empty<byte>();
        2 references
        public byte[] PasswordSalt { get; set; } = Array.Empty<byte>();
        1 reference
        public List<BlogPost>? BlogPosts { get; set; }
    }
}
```

## BlogPost Model

```
using System.ComponentModel;
using System.ComponentModel.DataAnnotations;

namespace BlogAPI.Models
{
    9 references
    public class BlogPost
    {
        3 references
        public int Id { get; set; }
        0 references
        public string Title { get; set; }
        0 references
        public string Content { get; set; }
        [DataType(DataType.Date)]
        0 references
        public DateTime PublishDate { get; set; }
        [DefaultValue("https://tse2.mm.bing.net/th?id=OIP.wsVlPGCwZ8_WwYZmtpKbAAHaCZ&pid=Api&P=0")]
        0 references
        public string? ImageURL { get; set; }
        0 references
        public string? Category { get; set; }

        2 references
        public int AuthorId { get; set; }
        1 reference
        public User? Author { get; set; }
    }
}
```

## BlogDbContext

```
using Microsoft.EntityFrameworkCore;
using System.Collections.Generic;

namespace BlogAPI.Models
{
    13 references
    public class BlogDbContext : DbContext
    {
        0 references
        public BlogDbContext(DbContextOptions<BlogDbContext> options) : base(options)
        {
        }

        11 references
        public DbSet<BlogPost> BlogPosts { get; set; }

        13 references
        public DbSet<User> Users { get; set; }

        0 references
        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.Entity<BlogPost>()
                .HasOne(p => p.Author)
                .WithMany(u => u.BlogPosts)
                .HasForeignKey(p => p.AuthorId);
        }
    }
}
```

# Implementation Details

This project consists of 2 major modules.

- User Module
- Blog Module

Each module consists of several methods to implement the required functionality.

## User Module

There are basically two functionalities in this module. Create a new user and login to existing user. This module is the base for authentication and authorization. Here we have used JWT token for the authentication.

```
[HttpPost("Register")]
0 references
public async Task<ActionResult> Register(UserRegisterDTO userDTO)
{
    var res = await _authRepo.Register(new User() { Username = userDTO.Username }, userDTO.Password);
    if (res == 0)
    {
        return BadRequest($"Cannot register {userDTO.Username}");
    }
    return Ok($"User registered successfully!");
}
```

```
[HttpPost("Login")]
0 references
public async Task<ActionResult> Login(UserLoginDTO userDTO)
{
    var res = await _authRepo.Login(userDTO.Username, userDTO.Password);
    Console.WriteLine(res);
    if (res == null)
    {
        return BadRequest($"Incorrect username or password!");
    }
    return Ok(new { token = res, status = 200 });
}
```

## Blog Module

There are mainly four things can be done with blogs. First, we can fetch all the blogs, create a new blog, edit existing blog and delete blog.

```
// GET: api/BlogPosts
[AllowAnonymous]
[HttpGet]
0 references
public async Task<ActionResult<IEnumerable<BlogPost>>> GetBlogPosts()
{
    if (_context.BlogPosts == null)
    {
        return NotFound();
    }
    return await _context.BlogPosts.ToListAsync();
}
```

```
// GET: api/BlogPosts/5
[AllowAnonymous]
[HttpGet("{id}")]
0 references
public async Task<ActionResult<BlogPost>> GetBlogPost(int id)
{
    if (_context.BlogPosts == null)
    {
        return NotFound();
    }
    var blogPost = await _context.BlogPosts.FindAsync(id);

    if (blogPost == null)
    {
        return NotFound();
    }

    return blogPost;
}
```

```

// get blog by user id
[AllowAnonymous]
[HttpGet("/blog/{id}")]
0 references
public async Task<ActionResult<IEnumerable<BlogPost>>> GetBlogsById(long id)
{
    return await _context.BlogPosts.Where(x => x.AuthorId == id).ToListAsync();
}

```

```

// PUT: api/BlogPosts/5
[HttpPut("{id}")]
[Authorize]
0 references
public async Task<IActionResult> PutBlogPost(int id, BlogPost blogPost)
{
    if (id != blogPost.Id)
    {
        return BadRequest();
    }
    _context.Entry(blogPost).State = EntityState.Modified;
    try
    {
        await _context.SaveChangesAsync();
    }
    catch (DbUpdateConcurrencyException)
    {
        if (!BlogPostExists(id))
        {
            return NotFound();
        }
        else
        {
            throw;
        }
    }

    return NoContent();
}

```

```
// POST: api/BlogPosts
// To protect from overposting attacks, see https://go.microsoft.com/fwlink/?linkid=2123754
[HttpPost]
[Authorize]
0 references
public async Task<ActionResult<BlogPost>> PostBlogPost(BlogPost blogPost)
{
    if (_context.BlogPosts == null)
    {
        return Problem("Entity set 'BlogDbContext.BlogPosts' is null.");
    }
    _context.BlogPosts.Add(blogPost);
    await _context.SaveChangesAsync();

    return CreatedAtAction("GetBlogPost", new { id = blogPost.Id }, blogPost);
}
```

```
// DELETE: api/BlogPosts/5
[HttpDelete("{id}")]
[Authorize]
0 references
public async Task<IActionResult> DeleteBlogPost(int id)
{
    if (_context.BlogPosts == null)
    {
        return NotFound();
    }
    var blogPost = await _context.BlogPosts.FindAsync(id);
    if (blogPost == null)
    {
        return NotFound();
    }

    _context.BlogPosts.Remove(blogPost);
    await _context.SaveChangesAsync();

    return NoContent();
}
```

```
1 reference
private bool BlogPostExists(int id)
{
    return (_context.BlogPosts?.Any(e => e.Id == id)).GetValueOrDefault();
}
```

# Testing

Test Case Id	Test Case Objective	Input Data	Expected Output	Actual Output	Status
TC-01	Add new User into system	Credential	User Added	User Added	Pass
TC-02	Add user with already exists username	Credential	User already exists	User already exists	Pass
TC-03	Login into System	Credential	Token	Token	Pass
TC-04	Login into System with wrong credential	Wrong Credential			Pass
TC-05	Logout from system	Logout Button	Home Screen	Home Screen	Pass
TC-06	Blog List	-	Blogs	Blogs	Pass
TC-07	Blog Detail	Select Blog	Blog Content	Blog Content	Pass
TC-08	Create Blog	Blog Details	Blog added on home screen	Blog added on home screen	Pass
TC-09	Update Blog	Blog ID along with new data	Updated Blog	Updated Blog	Pass
TC-10	Delete Blog	Blog ID	Blog removed from site	Blog removed from site	Pass

## Testing our backend with Swagger

### TC-01: Add new user into system

The screenshot shows the Swagger UI for the **Auth** API. The selected endpoint is **POST /api/Auth/Register**. The request body is a JSON object: `{ "username": "demo", "password": "demo123" }`. The response is a 200 status code with the body `User registered successfully!`. The response headers include `access-control-allow-credentials: true`, `access-control-allow-origin: https://localhost:7079`, `content-type: text/plain; charset=utf-8`, `date: Tue, 28 Mar 2023 09:46:09 GMT`, `server: Kestrel`, and `vary: Origin`.

**Auth**

**POST** /api/Auth/Register

Parameters

No parameters

Request body

application/json

```
{
  "username": "demo",
  "password": "demo123"
}
```

Responses

Curl

```
curl -X 'POST' \
  'https://localhost:7079/api/Auth/Register' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
    "username": "demo",
    "password": "demo123"
  }'
```

Request URL

https://localhost:7079/api/Auth/Register

Server response

Code Details

200

Response body

User registered successfully!

Response headers

```
access-control-allow-credentials: true
access-control-allow-origin: https://localhost:7079
content-type: text/plain; charset=utf-8
date: Tue, 28 Mar 2023 09:46:09 GMT
server: Kestrel
vary: Origin
```

### TC-02: Trying to register with already exists username

The screenshot shows the Swagger UI for the **Auth** API. The selected endpoint is **POST /api/Auth/Register**. The request body is a JSON object: `{ "username": "demo", "password": "demo123" }`. The response is a 400 status code with the body `Cannot register demo`. The response headers include `access-control-allow-credentials: true`, `access-control-allow-origin: https://localhost:7079`, `content-type: text/plain; charset=utf-8`, `date: Tue, 28 Mar 2023 10:01:15 GMT`, `server: Kestrel`, and `vary: Origin`.

Responses

Curl

```
curl -X 'POST' \
  'https://localhost:7079/api/Auth/Register' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
    "username": "demo",
    "password": "demo123"
  }'
```

Request URL

https://localhost:7079/api/Auth/Register

Server response

Code Details

400

Error: response status is 400

Response body

Cannot register demo

Response headers

```
access-control-allow-credentials: true
access-control-allow-origin: https://localhost:7079
content-type: text/plain; charset=utf-8
date: Tue, 28 Mar 2023 10:01:15 GMT
server: Kestrel
vary: Origin
```

### TC-03: Login into system



```
Curl
curl -X 'POST' \
      'https://localhost:7079/api/Auth/Login' \
      -H 'accept: */*' \
      -H 'Content-Type: application/json' \
      -d '{
        "username": "demo",
        "password": "demo123"
      }'
```

Request URL

```
https://localhost:7079/api/Auth/Login
```

Server response

Code Details

---

200

Response body

```
{
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJuYWQlOiWQlOIiwSIjoiInVuaXF1ZV9uYmQlIjo1IjZGVbyIsIm5kaXIiOiE1IGMTY3OTk5NzkxS29uZXhpdjoxMjg0eWZAcjB7YXQlOjE2ZnkiS0tC5WDF9.Jjya56fysH6s8HPGAyyes-1H0R6BuNr-BEDTyv71h6jl2Vs8DR2UAks3w2f6n5p8t_M8Rth5VD-BAZOEGIA/11",
  "status": 200
}
```

Available authorizations

oauth2 (apiKey)

Standard Authorization using Bearer scheme. Example: bearer  
Name: Authorization  
In: header  
Value:  

bearer eyJhbGciOiJIUzUxMi...

AuthorizeClose

POST

/api/Auth/Login

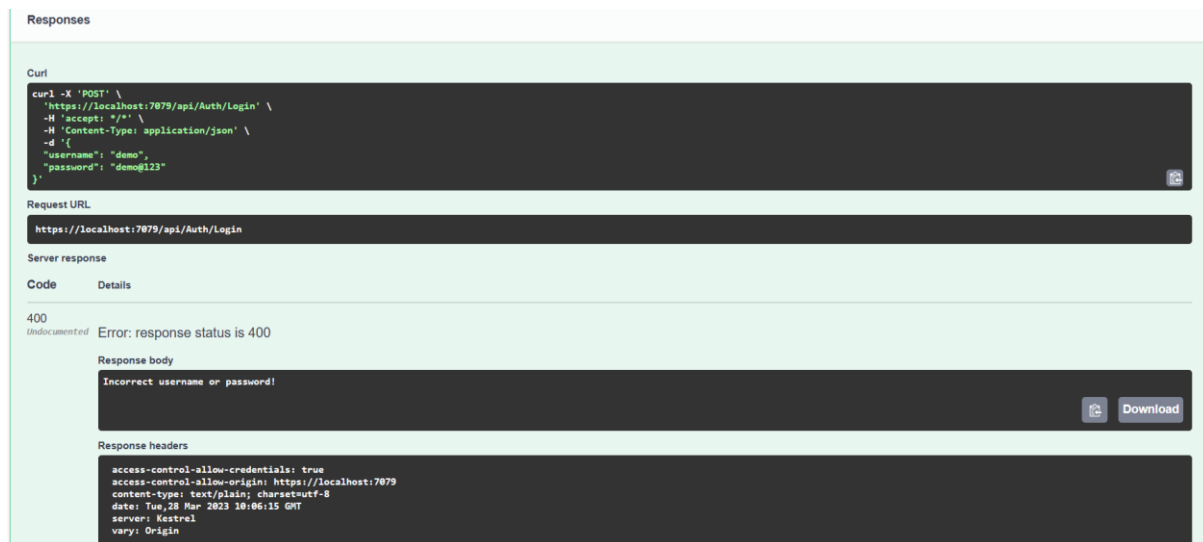
Parameters

No parameters

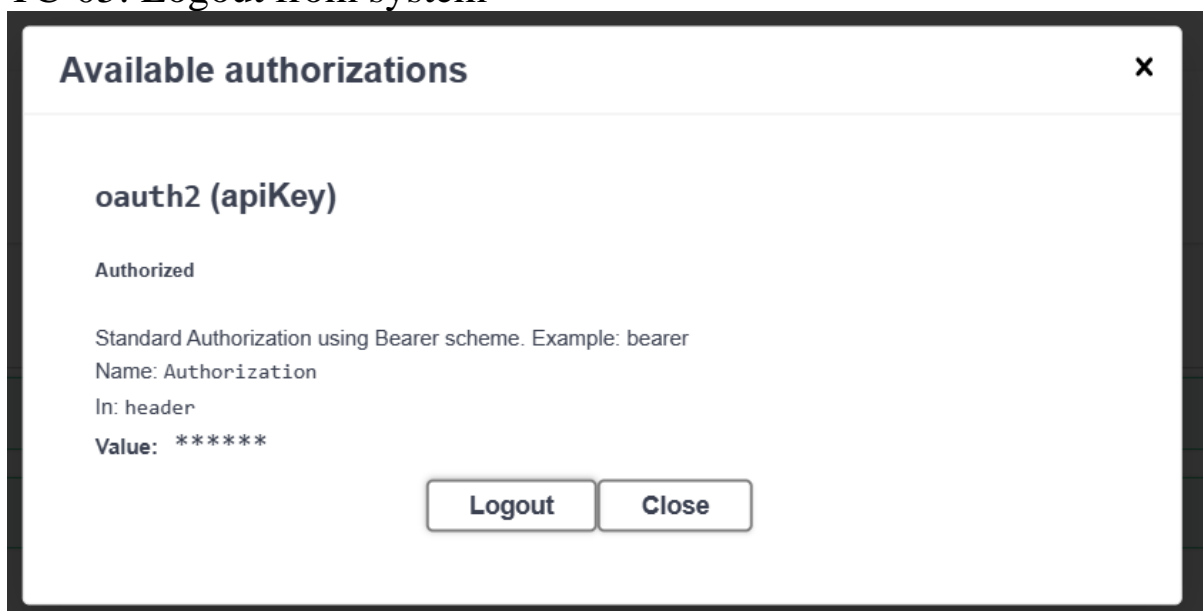
Request body

application/json

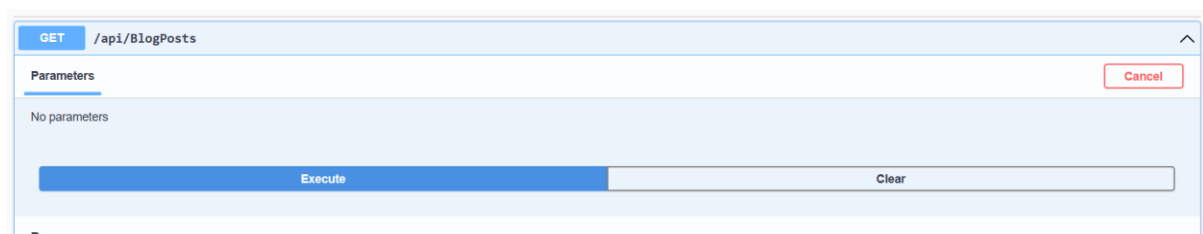
```
{
  "username": "demo",
  "password": "demo@123"
}
```

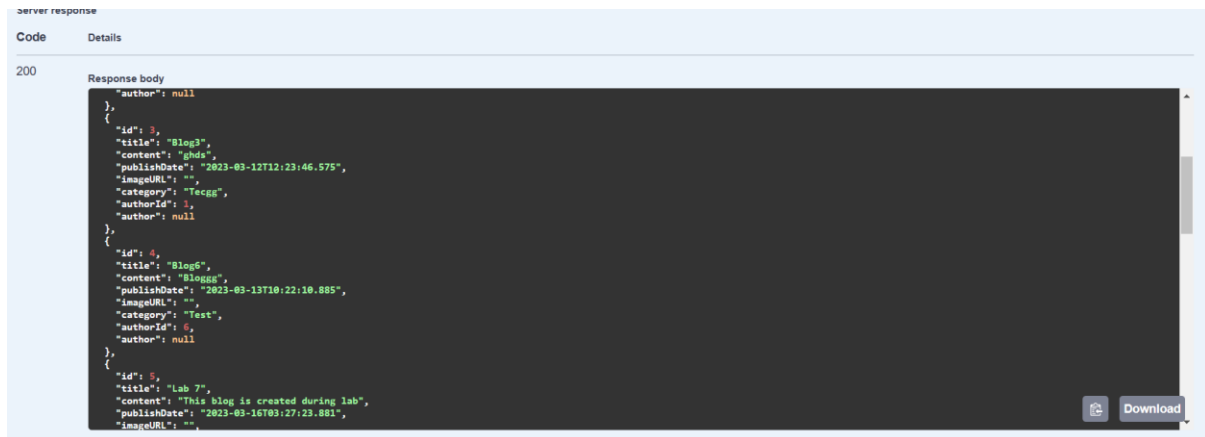


## TC-05: Logout from system

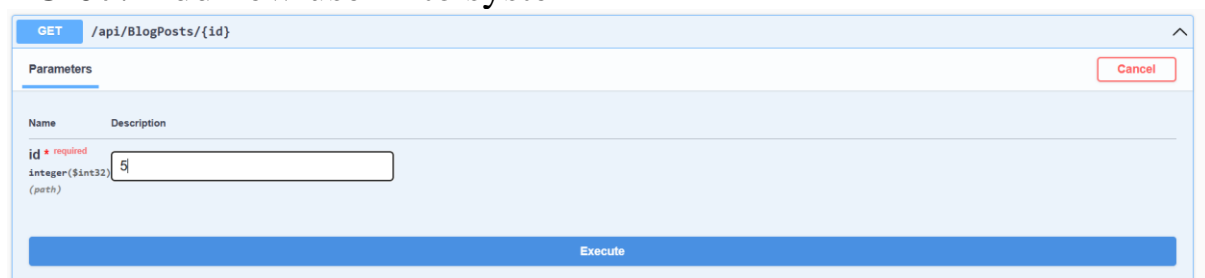


## TC-06: Get all Blogs

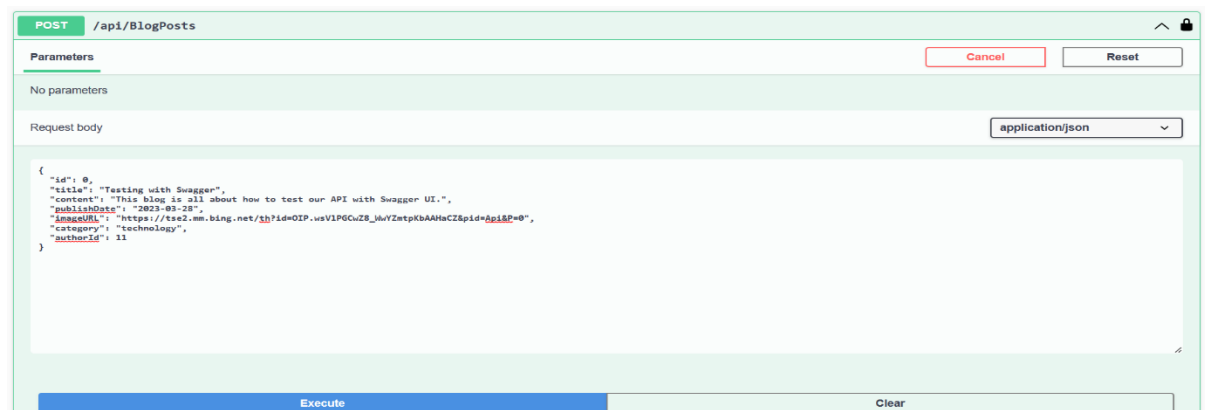




## TC-07: Add new user into system



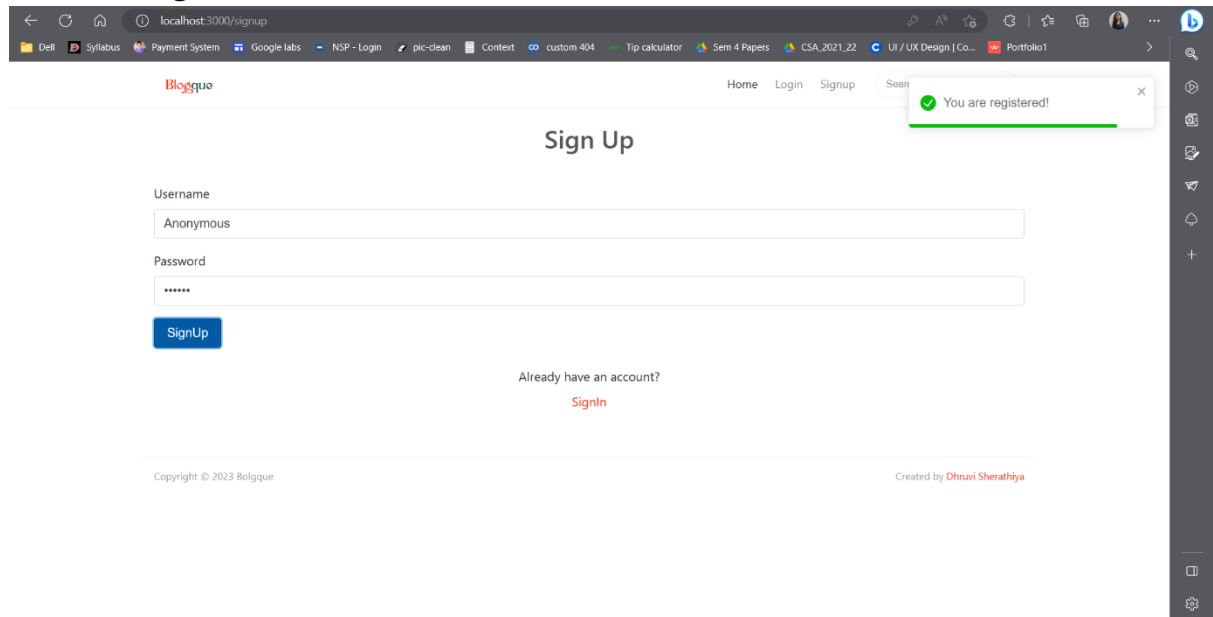
## TC-08: Create new blog



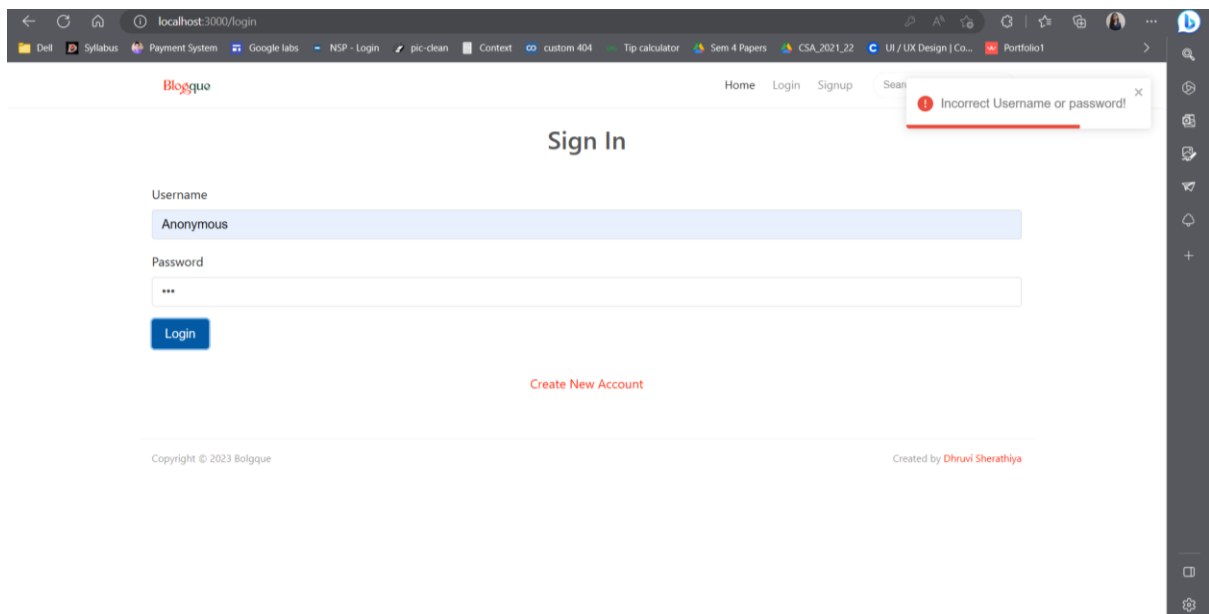


# Screenshots

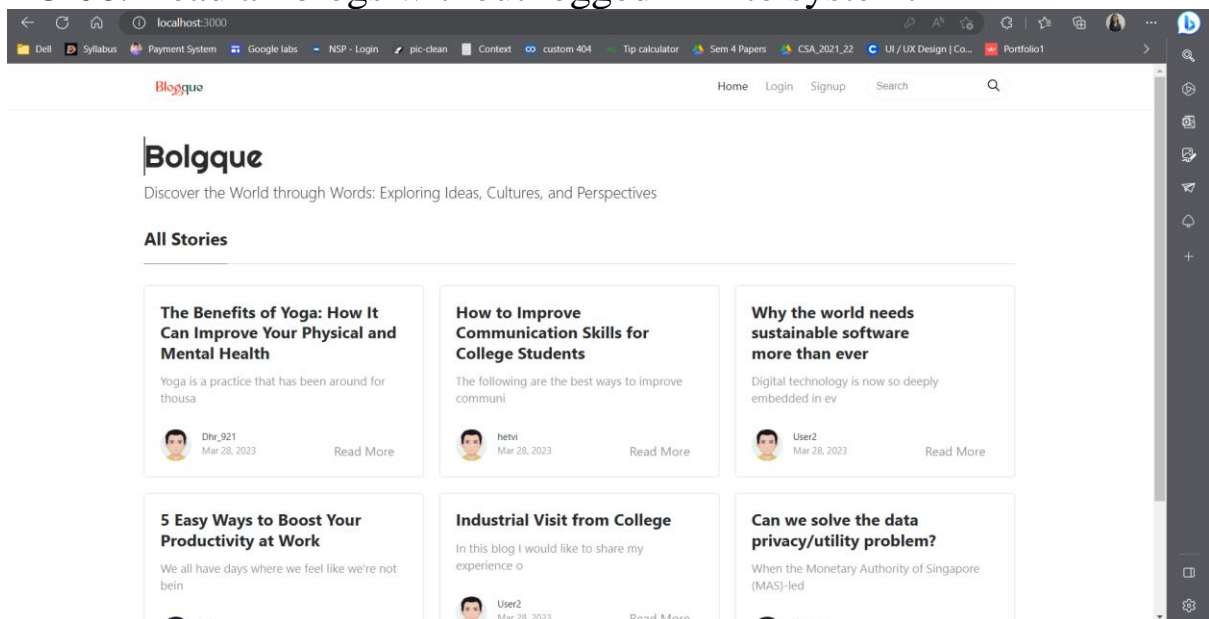
## TC-01: Register new user into site



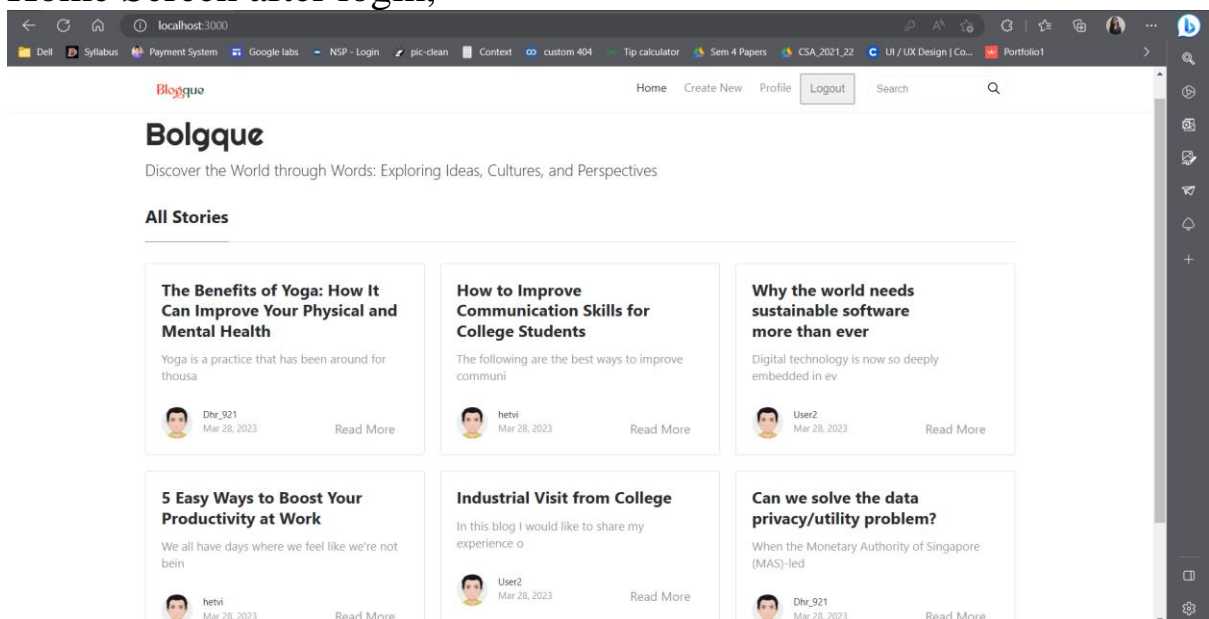
## TC-04: Trying to login into system with wrong credentials



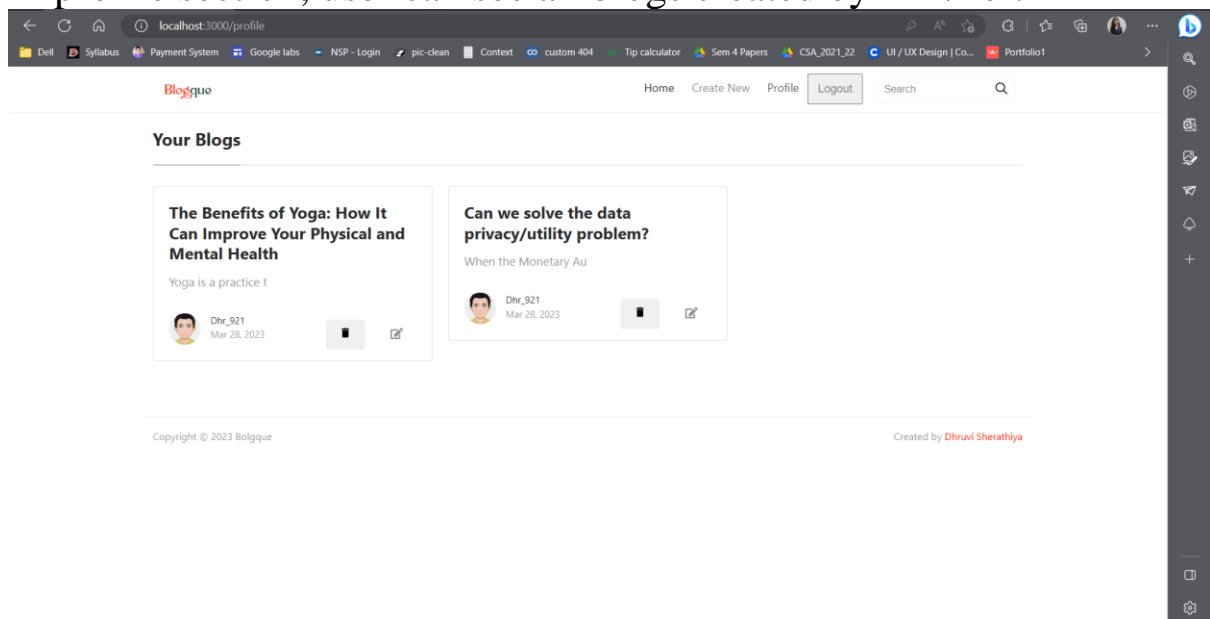
TC-06: Read all blogs without logged-in into system.



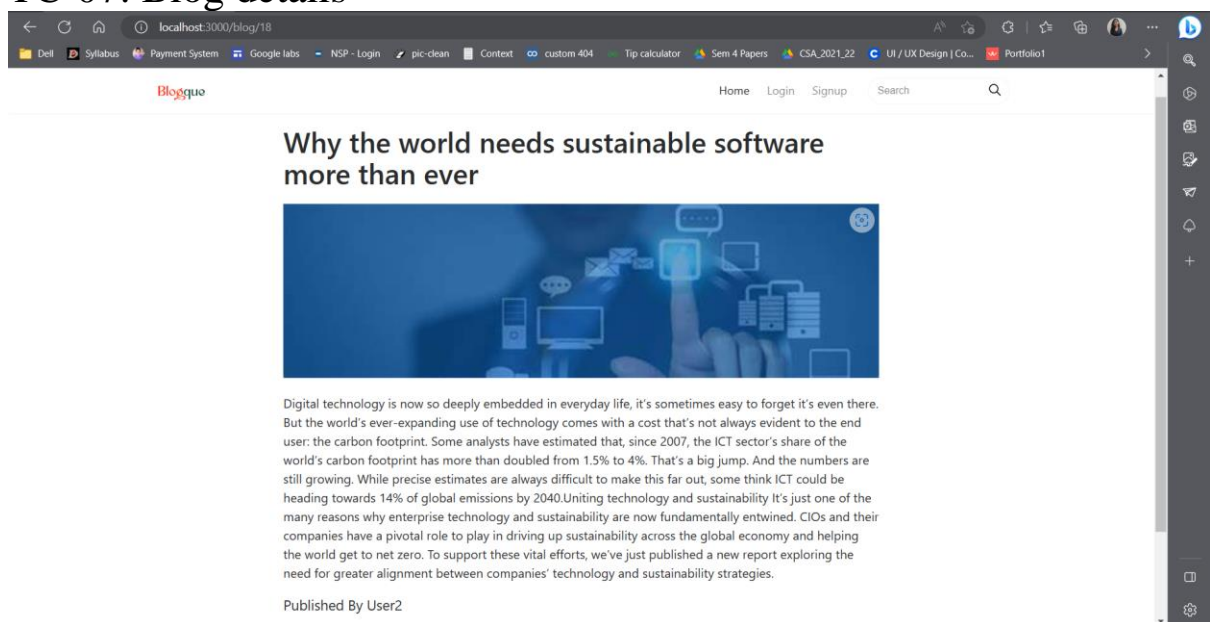
Home Screen after login,



In profile section, user can see all blogs created by him/her.



## TC-07: Blog details

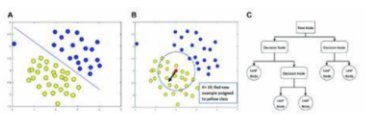


## TC-08: Create a Blog

localhost:3000/create

Blogque
Home
Create New
Profile
Logout
Search

## Create a new Blog



Blog Title

Banner Image URL

Blog Category

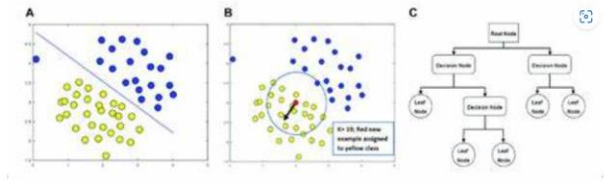
Tell your story ...

Create

localhost:3000/blog/20

Blogque
Home
Create New
Profile
Logout
Search

## Supervised ML Algorithm: Support Vector Machines (SVM)

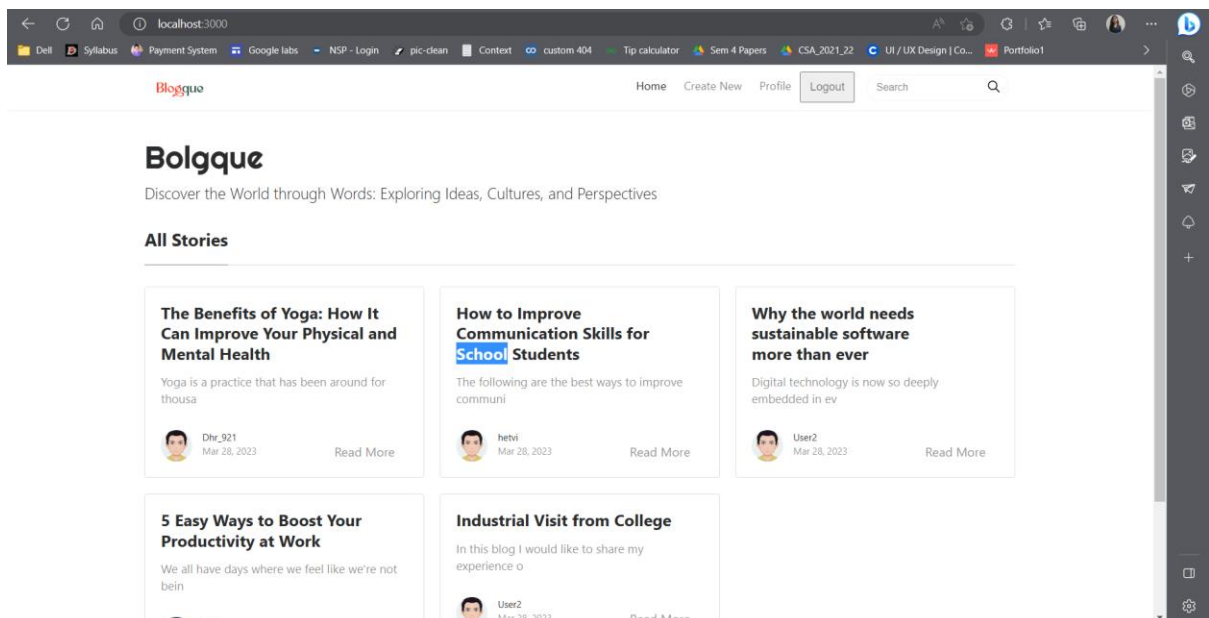
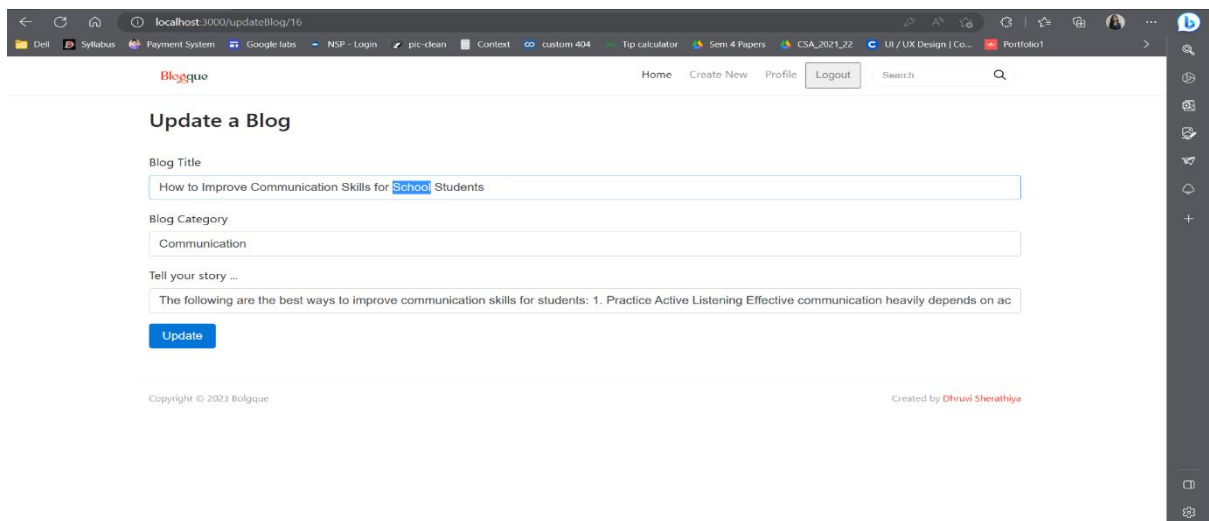


In supervised learning algorithms, labels are given to data-points (i.e- pairing a data-points with its class). In other words, models are supervised with class labels. While, in unsupervised learning algorithms, labels are not given to data-points, the model tries to find the class labels by comparing a data-point with other data-points and label similar data-points as the same class and likewise. In other words, models are not supervised explicitly. Support Vector Machine (SVM) is a popular supervised Machine Learning algorithm used for classification problems, regression problems, and outlier detection. In simple words, when all data-points plotted in n-dimensional space (dimensions are decided on the basis of the number of features in the dataset), then there exist many hyperplanes that can separate 2 classes, thus in SVM, the hyperplane that has maximum margin value to separate these 2 classes is used as the decision boundary line.

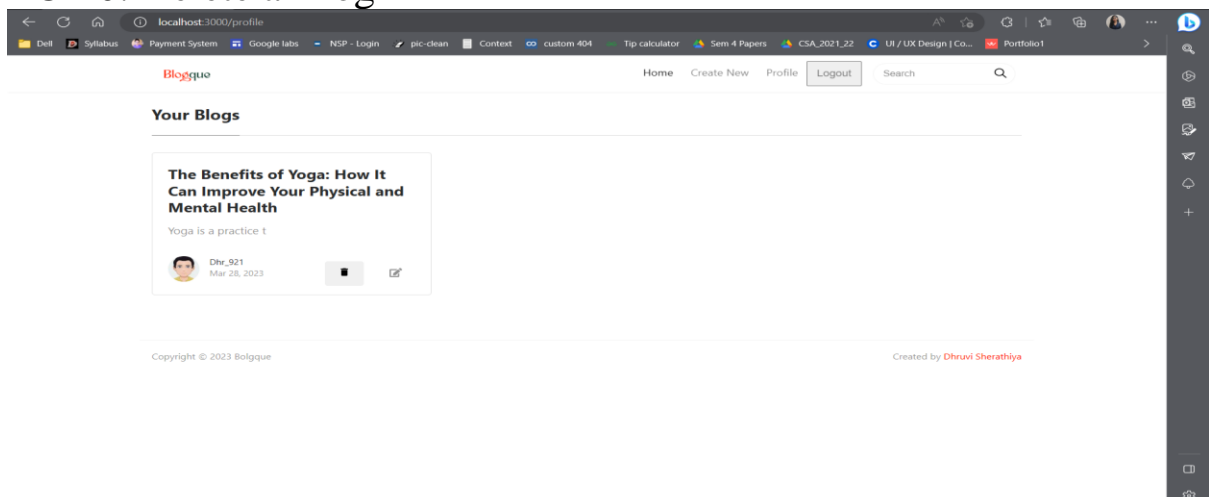
Published By Dhr\_921

TC-09: Update a blog





## TC-10: Delete a Blog



## Conclusion

The functionalities are implemented in system after understanding all the system modules according to the requirements. Functionalities that are successfully implemented in the system are:

- User Registration
- User Login / Logout
- Read all blogs
- Create new blog
- Update blog
- Delete a blog
- Read full blog

## Future Enhancement

Below are some future extensions to this project in order to improve the project.

- Commenting System
- Image Integration
- Category Filtering
- Search Functionality

## References

Following links and websites were referred during the development of this project.

- Stack overflow
- Microsoft Documentation
- React
- Bootstrap