Ex.no.1

Date: 31-03

Design a good looking website for your college containing a description of the courses, department, faculty, etc.,

AIM:

To design a website for my college with the given description.

PROCEDURE:

* Start the Program.
* Determine the Purpose of the website.
* identify the target audience.
* Create a basic website structure with main sections such as Home, about us, Academics, Faculty, Departments, Admissions, Student Life, Alumni, News & Events.
* Design the layout using a colour scheme and images that reflect the college brand and style.
* Develop the website using a website builder, such as wix or hire a web developer to create the site.
* Test the website and ensure that it's working properly and that all links are functioning.
* End the Program.

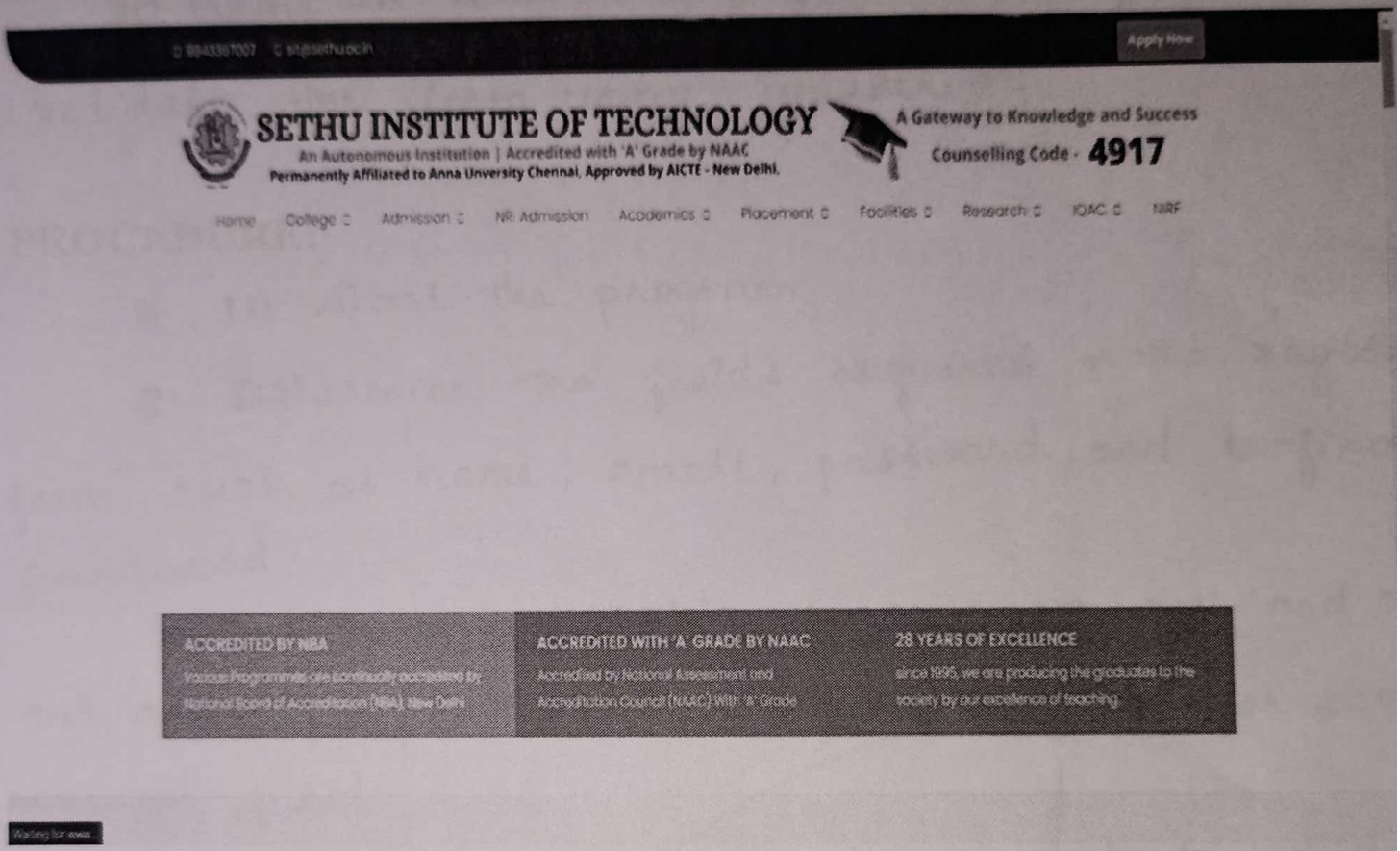
PROGRAM:

```
import React from 'react';
import { useState, useEffect } from 'react';

interface Course {
  title: string;
  description: string;
  instructor: string;
  credits: number;
}

interface Department {
  name: string;
  courses: Course[];
}

interface FacultyMember {
  name: string;
  department: string;
  courses: Course[];
}

const CollegeWebsite = () => {
  const [departments, setDepartments] = useState<Department[]>([]);
  const [facultyMembers, setFacultyMembers] = useState<FacultyMember[]>([]);

  useEffect(() => {
    // Fetch departments and faculty data from API endpoints
    const fetchDepartments = async () => {
      const response = await fetch('/api/departments');
      const data = await response.json();
      setDepartments(data);
    };

    const fetchFacultyMembers = async () => {
      const response = await fetch('/api/facultyMembers');
      const data = await response.json();
      setFacultyMembers(data);
    };

    fetchDepartments();
    fetchFacultyMembers();
  }, []);

  return (
    <div>
      <header>
        <h1>Welcome to CollegeName</h1>
        <nav>
          <ul>
            <li><a href="#courses">Courses</a></li>
            <li><a href="#departments">Departments</a></li>
            <li><a href="#faculty">Faculty</a></li>
          </ul>
```

```
 </nav>
</header>
<main>
 <section id="courses">
  <h2>Courses</h2>
  {departments.map((department) => (
   <div key={department.name}>
    <h3>{department.name}</h3>
    <ul>
     {department.courses.map((course) => (
      <li key={course.title}>
       <h4>{course.title}</h4>
       <p>{course.description}</p>
       <p>Instructor: {course.instructor}</p>
       <p>Credits: {course.credits}</p>
      </li>
     ))}
    </ul>
   </div>
  ))}
 </section>
 <section id="departments">
  <h2>Departments</h2>
  <ul>
   {departments.map((department) => (
    <li key={department.name}>
     <h3>{department.name}</h3>
     <p>{department.description}</p>
    </li>
   ))}
  </ul>
 </section>
 <section id="faculty">
  <h2>Faculty</h2>
  <ul>
   {facultyMembers.map((facultyMember) => (
    <li key={facultyMember.name}>
     <h3>{facultyMember.name}</h3>
     <p>{facultyMember.department}</p>
     <ul>
      {facultyMember.courses.map((course) => (
       <li key={course.title}>
        <h4>{course.title}</h4>
        <p>{course.description}</p>
        <p>Instructor: {course.instructor}</p>
        <p>Credits: {course.credits}</p>
       </li>
      ))}
     </ul>
    </li>
   ))}
  </ul>
 </section>
</main>
<footer>
 <p>&copy; CollegeName {new Date().getFullYear()}</p>
```

```
    </footer>
  </div>
 );
};
```

OUTPUT:



RESULT:

Thus Program was executed successfully.

Ex.no.2

Date: 10.1.23

Create an aesthetic Registration Form and validate the form using Typescript

AIM:

TO create an aesthetic Registration form and validata the form using Typescript.

PROCEDURE:

* TO start the program
* Determine the fields required in the registration form. such as name, email, password, and confirm password.
* Design an aesthetic form using HTML and CSS, and add appropriate labels and input fields for each required field.
* Add event listners to the form inputs to validate user input as it's entered. such as ensuring the emails is in a valid format and the password meets minimum requirement.
* Create a Typescript class to represent the form data and define appropriate validation rules for each field.
* If the validation fails. display error message, next to each invalid field and prevent form submission.
* End the program.

PROGRAM:

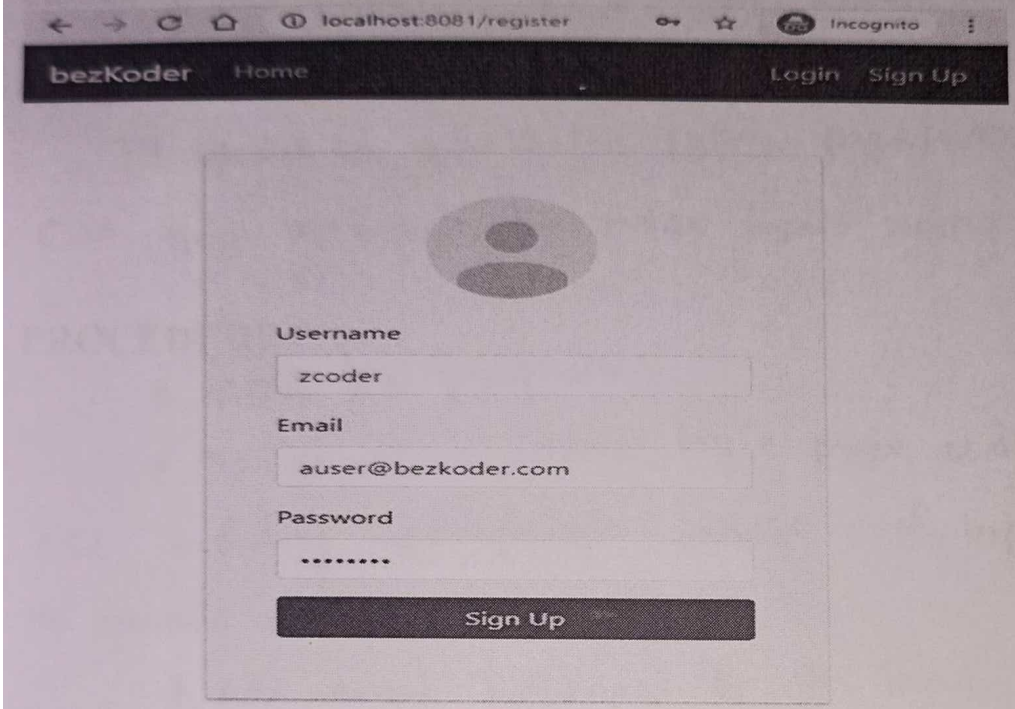
```
<!DOCTYPE html>
<html>
 <head>
  <meta charset="UTF-8" />
  <title>Registration Form</title>
  <style>
   * {
    box-sizing: border-box;
    font-family: Arial, Helvetica, sans-serif;
   }

   body {
    background-color: #f1f1f1;
   }

   h1 {
    text-align: center;
   }

   .container {
    background-color: #fff;
    margin: auto;
    width: 50%;
    padding: 20px;
    border-radius: 10px;
    box-shadow: 0px 0px 10px #888;
   }

   input[type="text"], input[type="email"], input[type="password"], select {
    width: 100%;
    padding: 12px;
    border: 1px solid #ccc;
    border-radius: 4px;
    resize: vertical;
   }

   label {
    padding: 12px 12px 12px 0;
    display: inline-block;
   }

   button[type="submit"] {
    background-color: #4CAF50;
    color: white;
    padding: 12px 20px;
    border: none;
    border-radius: 4px;
    cursor:  pointer;
    float: right;
   }

   button[type="submit"]:hover {
```

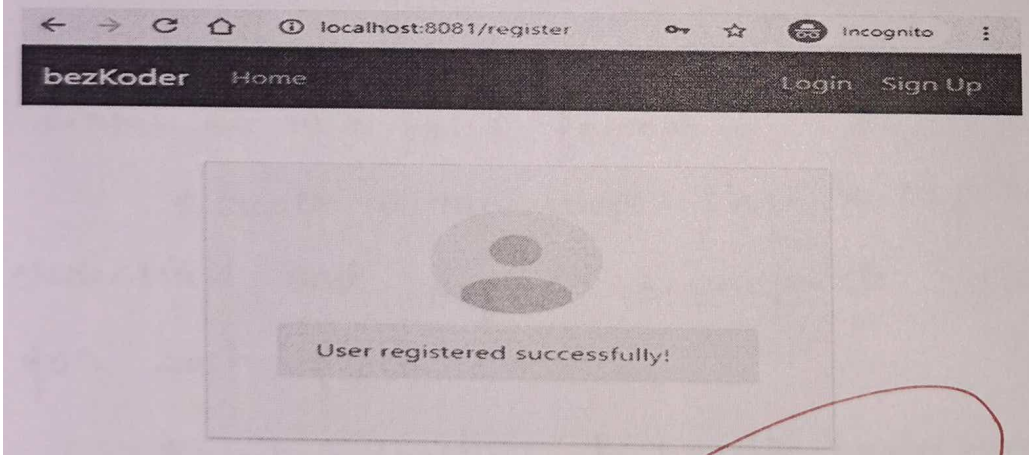
```
    background-color: #45a049;
   }

   .clearfix::after {
    content: "";
    clear: both;
    display: table;
   }

   @media screen and (max-width: 600px) {
    .container {
     width: 100%;
    }
   }
  </style>
</head>

<body>
  <h1>Registration Form</h1>

  <div class="container">
    <form id="registration-form">
      <label for="name">Name</label>
      <input type="text" id="name" name="name" placeholder="Enter your name" required>

      <label for="email">Email</label>
      <input type="email" id="email" name="email" placeholder="Enter your email" required>

      <label for="password">Password</label>
      <input type="password" id="password" name="password" placeholder="Enter your password"
required>

        <label for=" confirm password"> Confirm Password</label>
      <input type="password" id="password" name=" confirm password" placeholder="Enter your
password again to confirm" required>

      <label for="country">Country</label>
      <select id="country" name="country" required>
        <option value="">Select your country</option>
        <option value="USA">USA</option>
        <option value="Canada">Canada</option>
        <option value="UK">UK</option>
        <option value="Australia">Australia</option>
      </select>

      <div class="clearfix">
        <button type="submit">Submit</button>
      </div>
    </form>
   </div>

   <script src="registration-form-validation.js"></script>
  </body>
</html>
```

typescript code for form validation

```
interface FormValues {
  name: string;
  email: string;
  password: string;
  confirmPassword: string;
}

interface FormErrors {
  name?: string;
  email?: string;
  password?: string;
  confirmPassword?: string;
}

function validateForm(values: FormValues): FormErrors {
  const errors: FormErrors = {};

  if (!values.name) {
    errors.name = 'Name is required';
  }

  if (!values.email) {
    errors.email = 'Email is required';
  } else if (!/\S+@\S+\.\S+/.test(values.email)) {
    errors.email = 'Email is invalid';
  }

  if (!values.password) {
    errors.password = 'Password is required';
  } else if (values.password.length < 6) {
    errors.password = 'Password must be at least 6 characters';
  }

  if (!values.confirmPassword) {
    errors.confirmPassword = 'Confirm Password is required';
  } else if (values.password !== values.confirmPassword) {
    errors.confirmPassword = 'Passwords do not match';
  }

  return errors;
}
const values: FormValues = {
  name: 'John Doe',
  email: 'johndoe@example.com',
  password: 'secret123',
  confirmPassword: 'secret123',
};

const errors = validateForm(values);
console.log(errors); // output: {}
```

OUTPUT:





RESULT:

The program was executed successfully.

Ex.no.3

Date: 24.1.23

Create an aesthetic LOGIN page using HTML and CSS that authenticates user input using Typescript

AIM:

To create an aesthetic LOGIN page using HTML and CSS that authenticates user input using Typescript.

PROCEDURE:

* start the program.
* Design an aesthetic login page using HTML and CSS, including appropriate labels and input fields for the email and password.
* Add event listeners to the input fields to valida -te user input as it's entered, such as ensuring the email is in a valid format.
* create a Typescript class to represent the login credentials and define appropriate validation rules for each field.
* Write a function to handle Login submission, which creates an instance of the Typescript class and calls its validation methods to ensure both fields are valid.
* If the authentication fails, display an error message to the user.
* end the program.

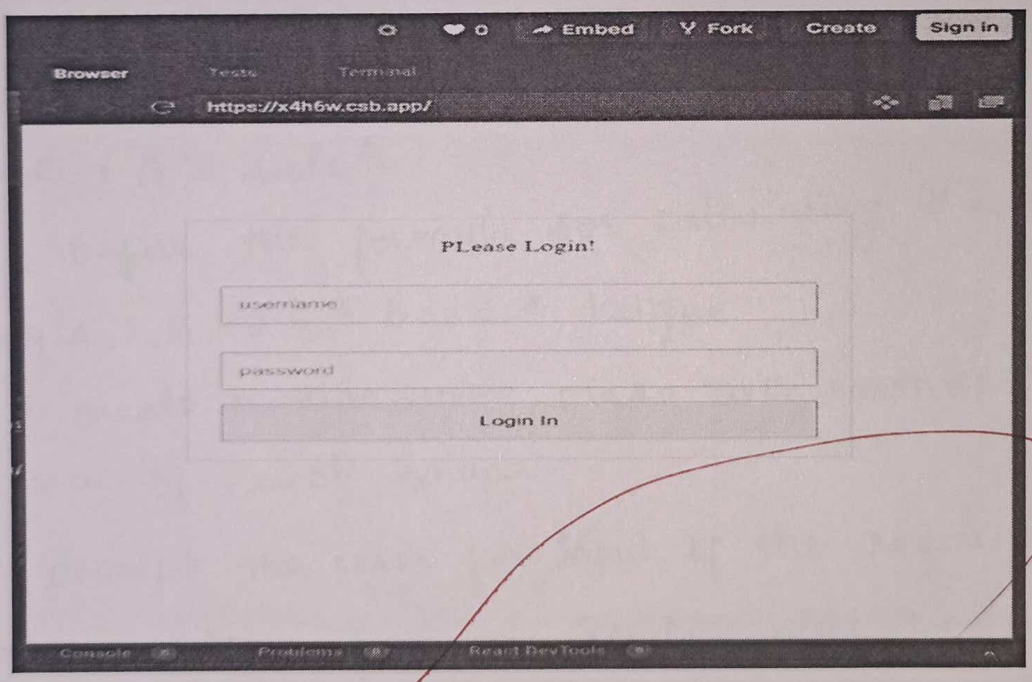
PROGRAM:

```
<!DOCTYPE html>
<html>
<head>
<title>Login Page</title>
<style>
body {
margin: 0;
padding: 0;
background: linear-gradient(to bottom right, #fc5c7d, #6a82fb);
font-family: sans-serif;
}

.login {
position: absolute;
top: 50%;
left: 50%;
transform: translate(-50%, -50%);
width: 400px;
padding: 40px;
background: #fff;
box-shadow: 0 0 10px rgba(0,0,0,0.3);
}

h1 {
margin: 0 0 30px;
padding: 0;
color: #333;
text-align: center;
font-size: 28px;
text-transform: uppercase;
letter-spacing: 2px;
}

label  {
display: block;
margin-bottom: 10px;
color: #333;
font-size: 18px;
text-transform: uppercase;
letter-spacing: 2px;
}

input[type="text"],
input[type="password"] {
width: 100%;
padding: 10px;
border: none;
background: #eee;
margin-bottom: 20px;
font-size: 16px;
}

input[type="submit"] {
```

```
width: 100%;
background: #fc5c7d;
color: #fff;
border: none;
padding: 10px;
font-size: 18px;
text-transform: uppercase;
letter-spacing:  2px;
cursor: pointer;
}

input[type="submit"]:hover {
background: #6a82fb;
}

#error-message {
color: red;
font-size: 14px;
margin-top: 10px;
text-align: center;
}
</style>
</head>
<body>
<div class="login">
<h1>Login</h1>
<form id="login-form">
<label for="username">Username:</label>
<input type="text" id="username" name="username" placeholder="Enter username">
<label for="password">Password:</label>
<input type="password" id="password" name="password" placeholder="Enter password">
<input type="submit" value="Login">
</form>
<div id="error-message"></div>
</div>
<script>
interface User {
  username: string;
  password: string;
}

const users: User[] = [
  { username: "john", password: "password123" },
  { username: "jane", password: "password456" },
];

function login(username: string, password: string): boolean {
  const user = users.find(user => user.username === username);

  if (!user) {
    return false;
  }

  return user.password === password;
}
```

```
const form = document.querySelector("form");

form.addEventListener("submit", (event) => {
 event.preventDefault();

 const usernameInput = document.querySelector<HTMLInputElement>("#username");
 const passwordInput = document.querySelector<HTMLInputElement>("#password");

 const username = usernameInput.value;
 const password = passwordInput.value;

 const isAuthenticated = login(username, password);

 if (isAuthenticated) {
  alert("Login successful!");
 } else {
  alert("Incorrect username or password.");
 }
});
 </script>
</body>
</html>
```

OUTPUT:



RESULT:

The program was executed successfully.

EX.no.4

Date: 31.1.23

Write a program using Typescript that Calculates the Area of circle, square and triangular

AIM:

To Write a program using Typescript that calculates the Area of circle, square and triangular.

PROCEDURE:

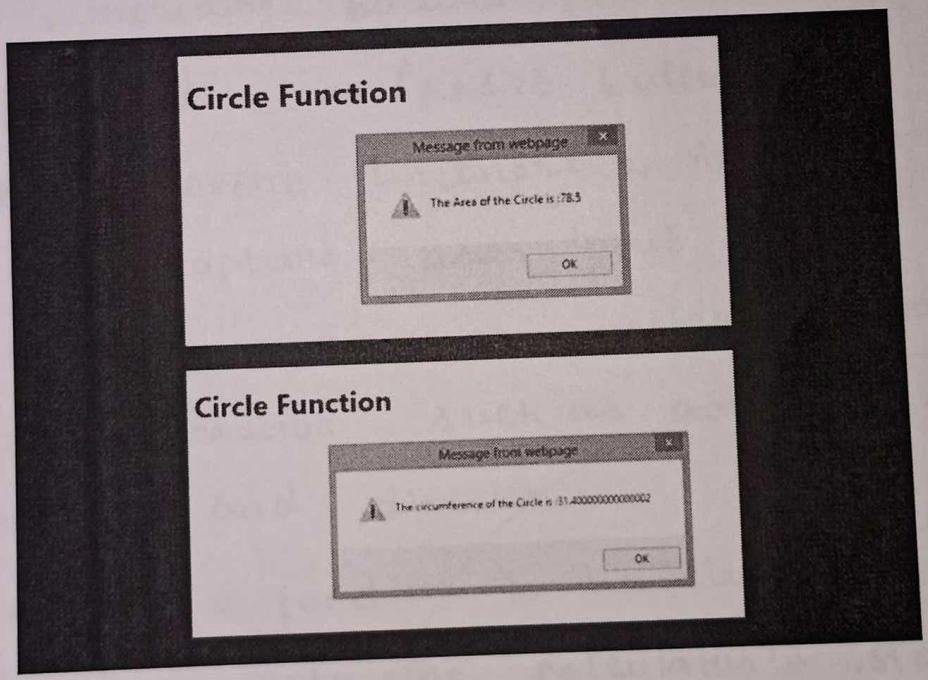
* start the program.
* Define the formula for calculating the area of a circle : $A = \pi r^2$
* Define the formula for calculating the area of a square : $A = side^2$
* Define the formula for calculating the area of a triangle : $A = 0.5 * base * height$.
* create a typescript class with methods to calculate the area of each shape.
* prompt the user for input of the required variables such as radius for the circle, side length for the square, and base and height for the square and base and height for the triangle.
* call the appropriate method based on the user's choice of shape and output the calculated area to the user.
* End the program.

PROGRAM:

```
class Shape {
  constructor(private readonly type: string) {}

  getType(): string {
    return this.type;
  }

  getArea(): number {
    return 0;
  }
}

class Circle extends Shape {
  constructor(private readonly radius: number) {
  super('circle');
  }

  getRadius(): number {
    return this.radius;
  }

  getArea(): number {
    return Math.PI * Math.pow(this.radius, 2);
  }
}

class Square extends Shape {
  constructor(private readonly sideLength: number) {
    super('square');
  }

  getSideLength(): number {
    return this.sideLength;
  }

  getArea(): number {
    return Math.pow(this.sideLength, 2);
  }
}

class Triangle extends Shape {
  constructor(private readonly base: number, private readonly height: number) {
    super('triangle');
  }

  getBase(): number {
    return this.base;
  }

  getHeight(): number {
    return this.height;
  }
```

```
  getArea(): number {
    return 0.5 * this.base * this.height;
  }
}

// Example usage
const circle = new Circle(5);
console.log(`Area of ${circle.getType()} with radius ${circle.getRadius()} is ${circle.getArea()}`);

const square = new Square(7);
console.log(`Area of ${square.getType()} with side length ${square.getSideLength()} is
${square.getArea()}`);

const triangle = new Triangle(4, 9);
console.log(`Area of ${triangle.getType()} with base ${triangle.getBase()} and height
${triangle.getHeight()} is ${triangle.getArea()}`);
```

OUTPUT:



RESULT:

The program was executed successfully.

Ex.no.5

Date: 14.2.23

Create a simple aesthetic Calculator using HTML, CSS and Typescript

AIM:

TO create a simple aesthetic calculator using HTML, CSS and Typescript.

PROCEDURE:

* Start the program.
* Design an aesthetic calculator UI using HTML and CSS, including buttons for each digit, arithmetic operation, and clear/reset button.
* Add event listeners to the digit and operation buttons to capture user input as it's entered.
* Write Typescript functions to handle each arithmetic operation, such as addition, subtraction, multiplication, and division.
* Write a function to handle the clear/reset button, which resets the calculator's state to its default value.
* update the display with the result of each calculation.
* End the program.

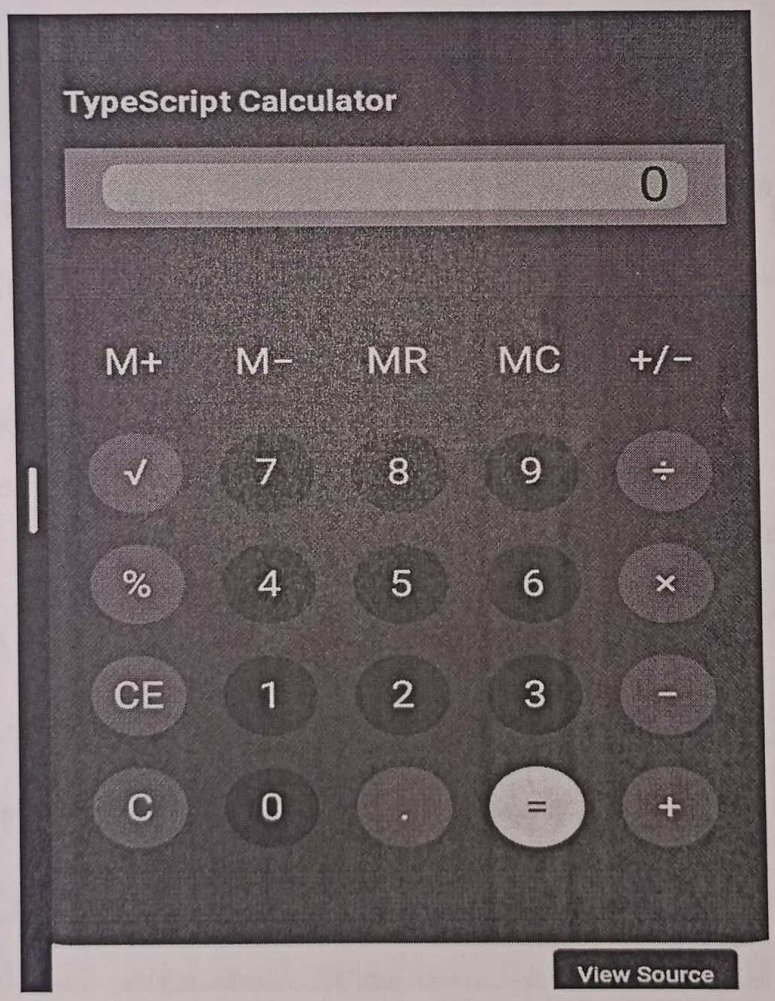
PROGRAM:

```
<!DOCTYPE html>
<html>
 <head>
  <meta charset="UTF-8" />
  <title>Simple Calculator</title>
  <style>
   body {
    background-color: #f5f5f5;
    font-family: Arial, sans-serif;
   }
   .calculator {
    margin: 0 auto;
    width: 250px;
    background-color: #eee;
    border: 1px solid #ccc;
    border-radius: 5px;
    box-shadow: 0px 0px 5px #ccc;
    padding: 10px;
   }
   .calculator button {
    background-color: #fff;
    border: 1px solid #ccc;
    border-radius: 5px;
    box-shadow: 0px 0px 3px #ccc;
    color: #333;
    font-size: 20px;
    padding: 10px;
    width: 50px;
    height: 50px;
    margin: 5px;
    outline: none;
   }
   .calculator button:hover {
    background-color: #f5f5f5;
    cursor: pointer;
   }
   .calculator button:active {
    background-color: #ccc;
   }
   .result {
    font-size: 30px;
    text-align: right;
    margin-bottom: 10px;
   }
  </style>
 </head>
 <body>
  <div class="calculator">
   <div class="result"></div>
   <button id="btnClear">C</button>
   <button id="btnNeg">+/-</button>
   <button id="btnPercent">%</button>
```

```
    <button id="btnDivide">/</button>
    <button id="btn7">7</button>
    <button id="btn8">8</button>
    <button id="btn9">9</button>
    <button id="btnMultiply">*</button>
    <button id="btn4">4</button>
    <button id="btn5">5</button>
    <button id="btn6">6</button>
    <button id="btnSubtract">-</button>
    <button id="btn1">1</button>
    <button id="btn2">2</button>
    <button id="btn3">3</button>
    <button id="btnAdd">+</button>
    <button id="btn0">0</button>
    <button id="btnDecimal">.</button>
    <button id="btnEqual">=</button>
  </div>
  <script src="./calculator.ts"></script>
 </body>
</html>

typescript code
class Calculator {
  private display: HTMLElement;

  constructor(display: HTMLElement) {
    this.display = display;
  }

  public clearDisplay() {
    this.display.textContent = "";
  }

  public updateDisplay(value: string) {
    this.display.textContent += value;
  }

  public evaluate() {
    const expression = this.display.textContent;
    if (expression) {
      try {
        const result = eval(expression);
        this.display.textContent = result.toString();
      } catch (error) {
        this.display.textContent = "Error";
      }
    }
  }
}

// Create a new calculator instance
const calculator = new Calculator(document.getElementById("calculator-display"));

// Attach event listeners to the calculator buttons
document.getElementById("btn-clear").addEventListener("click", () => {
calculator.clearDisplay();
```

```
});

document.getElementById("btn-equal").addEventListener("click", () => {
  calculator.evaluate();
});

document.querySelectorAll(".btn-number").forEach((btn) => {
  btn.addEventListener("click", () => {
  calculator.updateDisplay(btn.textContent);
  });
});
```

OUTPUT:



RESULT:

The program was executed successfully.

Ex.no.6

Date: 21-2-23

Create a responsive Navigation Menu using HTML, CSS and Typescript

AIM:

TO create a responsive Navigation Menu using HTML, CSS and Typescript

PROCEDURE:

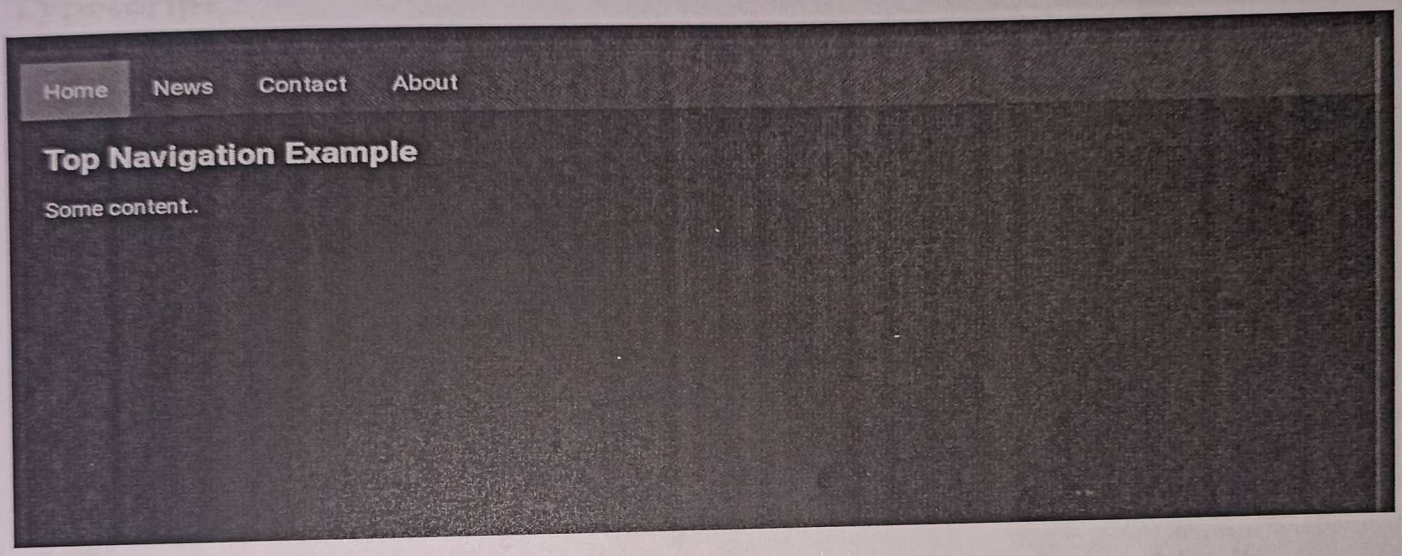
* start the program.
* Design a navigation menu using HTML and CSS, including a container div and a List of menu items.
* Add event listeners to the Menu items to capture user clicks.
* create a Typescript class to represent the navigation menu and define appropriate methods for handling user inputs and updating the display.
* use media queries in the CSS to adjust the appearance of the navigation menu based on the device screen size.
* update the display with the active menu item based on user input.
* End the program.

PROGRAM:

```
<!DOCTYPE html>
<html>
 <head>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>Responsive Navigation Menu</title>
  <script src="script.js"></script>
  <style>
   /* Internal CSS */
   * {
    box-sizing: border-box;
   }

   body {
    margin: 0;
    font-family: Arial, sans-serif;
   }

   .navbar  {
    overflow: hidden;
    background-color: #333;
   }

   .navbar a {
    float: left;
    display: block;
    color: #f2f2f2;
    text-align: center;
    padding: 14px 16px;
    text-decoration: none;
   }

   .navbar a:hover {
    background-color: #ddd;
    color: black;
   }

   .navbar a.active {
    background-color: #4CAF50;
    color: white;
   }

   .icon {
    display: none;
   }

   @media screen and (max-width: 600px) {
```

```
    .navbar a:not(:first-child) {
      display: none;
    }
    .navbar a.icon {
      float: right;
      display: block;
    }
  }

  @media screen and (max-width: 600px) {
    .navbar.responsive {
      position: relative;
    }
    .navbar.responsive a.icon {
      position: absolute;
      right: 0;
      top: 0;
    }
    .navbar.responsive a {
      float: none;
      display: block;
      text-align: left;
    }
  }
 </style>
</head>
<body>
 <div class="navbar" id="myNavbar">
  <a href="#" class="active">Home</a>
  <a href="#">About</a>
  <a href="#">Contact</a>
  <a href="#" class="icon" onclick="myFunction()">&#9776;</a>
 </div>

 <script>
  function myFunction() {
    var x = document.getElementById("myNavbar");
    if (x.className === "navbar") {
      x.className += " responsive";
    } else {
      x.className = "navbar";
    }
  }
 </script>
</body>
</html>
```

OUTPUT:



RESULT:

The program was executed successfully.

Ex.no.7

Date: 7.3.23

Create a good-looking E-commerce website using HTML, CSS and Typescript

AIM:

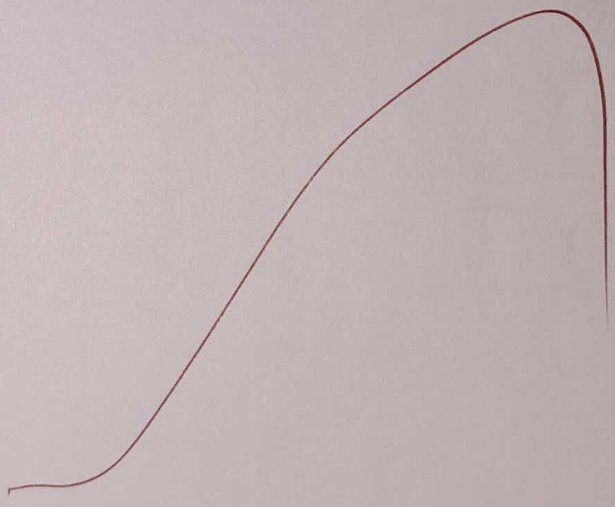
To create a good-looking E-commerce website using HTML, CSS and Typescript.

PROCEDURE:

* Start the program.
* plan and design your website using wireframes, sketches, or design software.
* create an HTML layout for your website, including a header, navigation menu, product listings, shopping cart, and footer.
* use css to style the HTML layout with appropriate colors, fonts, and spacing.
* Integrate a payment gateway for processing customer transactions, such as stripe or paypal.
* Write server-side code to handle data base queries and other server-side tasks, such as sending confirmation emails to customers.
* End the program.

PROGRAM:

```
<!DOCTYPE html>
<html>
 <head>
  <meta charset="UTF-8">
  <title>My E-commerce Website</title>
  <style>
/* Global styles */
body {
 margin: 0;
 padding: 0;
 font-family: Arial, sans-serif;
}

/* Header styles */
header {
 background-color: #333;
 color: #fff;
 padding: 20px;
}

nav ul {
 list-style: none;
 margin: 0;
 padding: 0;
}

nav ul li {
 display: inline-block;
 margin-right: 20px;
}

nav ul li:last-child {
 margin-right: 0;
}

nav ul li a {
 color: #fff;
 text-decoration: none;
}

/* Main styles */
main {
 max-width: 800px;
 margin: 0 auto;
 padding: 20px;
}

#products {
 margin-top: 20px;
}

#products h2 {
```



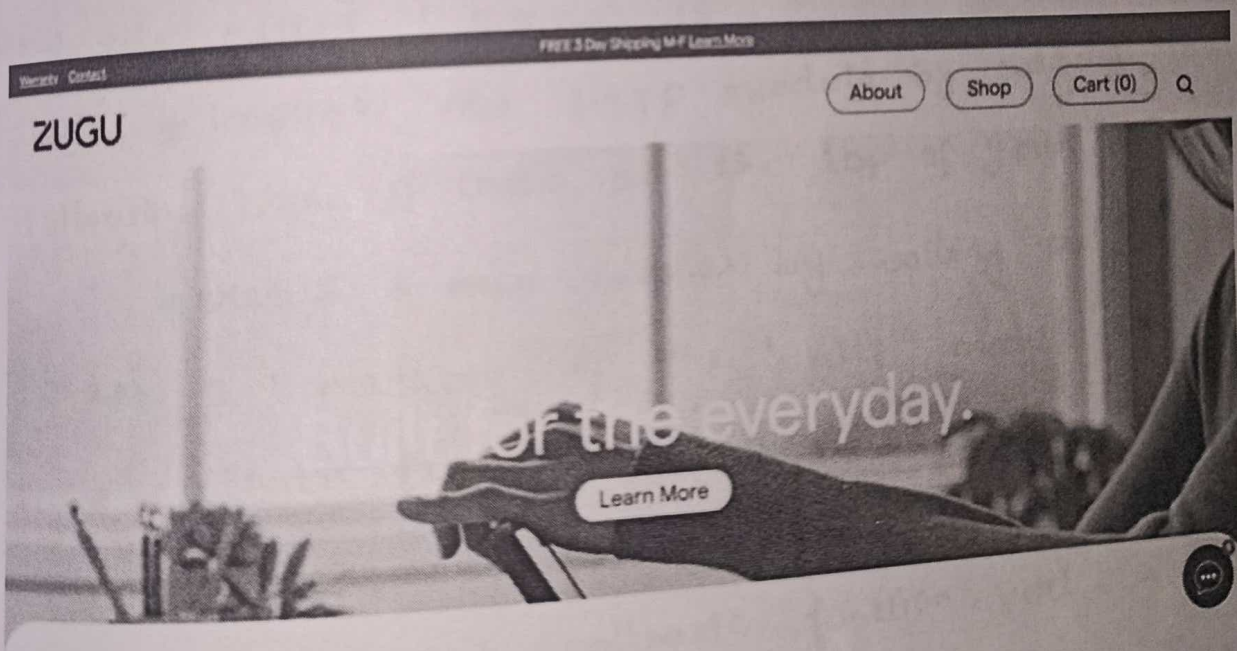
```
  margin-bottom: 10px;
}

#products ul {
  list-style: none;
  margin: 0;
  padding: 0;
}

#products li {
  border: 1px solid #ccc;
  padding: 10px;
  margin-bottom: 10px;
}

#products li h3 {
  margin-top: 0;
}

#products li button {
  background-color: #333;
  color: #fff;
  border: none;
  padding: 5px 10px;
  cursor: pointer;
}

#products li button:hover {
  background-color: #555;
}

/* Footer styles */
footer {
  background-color: #333;
  color: #fff;
  padding: 20px;
  text-align: center;
}
</style>
  </head>
  <body>
    <header>
      <nav>
        <ul>
          <li><a href="#">Home</a></li>
          <li><a href="#">Products</a></li>
          <li><a href="#">Contact Us</a></li>
        </ul>
      </nav>
    </header>
<script>
// Define a Product interface
interface Product {
  id: number;
  name: string;
  price: number;
```

```
}

// Define a ShoppingCartItem interface
interface ShoppingCartItem {
  product: Product;
  quantity: number;
}

// Define a ShoppingCart class
class ShoppingCart {
  private items: ShoppingCartItem[] = [];

  // Add a product to the cart
  addProduct(product: Product, quantity: number = 1) {
    // Check if the product is already in the cart
    const item = this.items.find(item => item.product.id === product.id);
    if (item) {
      item.quantity += quantity;
    } else {
      this.items.push({ product, quantity });
    }
  }

  // Remove a product from the cart
  removeProduct(product: Product) {
    const index = this.items.findIndex(item => item.product.id === product.id);
    if (index !== -1) {
      this.items.splice(index, 1);
    }
  }

  // Get the total price of all items in the cart
  getTotalPrice() {
    return this.items.reduce((total, item) => total + item.product.price * item.quantity, 0);
  }

  // Get the items in the cart
  getItems() {
    return this.items;
  }
}

// Define a User class
class User {
  private shoppingCart: ShoppingCart = new ShoppingCart();

  // Add a product to the shopping cart
  addToCart(product: Product, quantity: number = 1) {
  this.shoppingCart.addProduct(product, quantity);
  }

  // Remove a product from the shopping cart
  removeFromCart(product: Product) {
  this.shoppingCart.removeProduct(product);
  }
```

```
  // Get the total price of all items in the shopping cart
  getTotalPrice() {
   return this.shoppingCart.getTotalPrice();
  }

  // Get the items in the shopping cart
  getItemsInCart() {
   return this.shoppingCart.getItems();
  }
}

// Define a ProductCatalog class
class ProductCatalog {
  private products: Product[] = [
   { id: 1, name: 'Product 1', price: 10 },
   { id: 2, name: 'Product 2', price: 20 },
   { id: 3, name: 'Product 3', price: 30 },
  ];

  // Get all products
  getAllProducts() {
  return this.products;
  }

  // Get a product by ID
  getProductById(id: number) {
   return this.products.find(product => product.id === id);
  }
}

// Example usage
const productCatalog = new ProductCatalog();
const user = new User();

const products = productCatalog.getAllProducts();
const product1 = productCatalog.getProductById(1);
const product2 = productCatalog.getProductById(2);

user.addToCart(product1);
user.addToCart(product2, 2);

console.log('Items in cart:', user.getItemsInCart());
console.log('Total price:', user.getTotalPrice());
</script>
   <main>
    <h1>Welcome to My E-commerce Website</h1>
    <p>Here you can find all sorts of amazing products at great prices.</p>
    <section id="products">
     <h2>Our Products</h2>
     <ul>
      <li>
       <h3>Product 1</h3>
       <p>Description of product 1</p>
       <button>Add to Cart</button>
      </li>
      <li>
```

```
      <h3>Product 2</h3>
      <p>Description of product 2</p>
      <button>Add to Cart</button>
     </li>
     <li>
      <h3>Product 3</h3>
      <p>Description of product 3</p>
      <button>Add to Cart</button>
     </li>
    </ul>
   </section>
  </main>
  <footer>
   <p>Copyright © My E-commerce Website.
   All rights reserved.</p>
  </footer>
 </body>
</html>
```

OUTPUT:



RESULT:

The program was executed successfully.

Ex.no.8

Date: 16.3.23

Build Node Js using HTTP module

AIM:

TO Bulid Node js using HTTP Module.

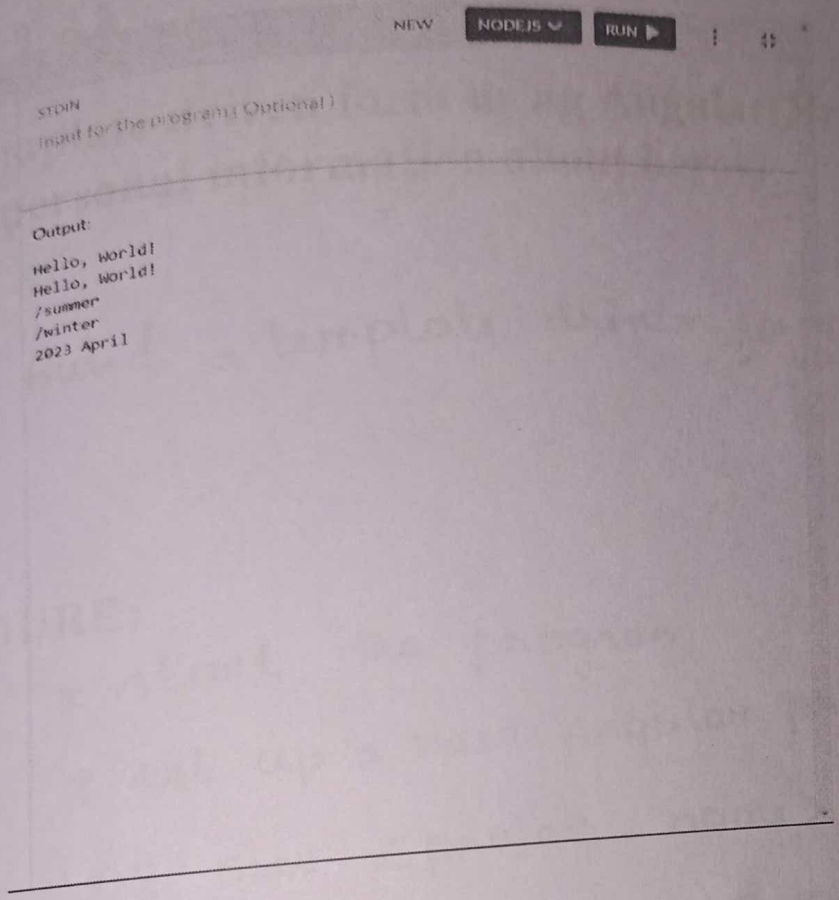
PROCEDURE:

* start the program.
* open your command prompt or terminal and create a new directory for your project.
* Navigate into the directory and create a new file called 'server.js'.
* import the HTTP module by adding the following line of code at the top of your 'server.js'
* create a new server by calling the 'create server()' method of the 'http' module, and pass in a callback functions what will be executed every times the server receives a request.
* inside the callback function, you can write logic to handle different types of requests. for example, you can write logic to handle GET request as follows.
* start the server by calling the 'listen()' method of the server object, and pass in a port number.
* End the program.

PROGRAM:

```
var http = require('http');
http.createServer(function (req, res) {
res.write('Hello World!');
 res.end();
}).listen(8080);

//Add an http header

var http = require('http');
http.createServer(function (req, res) {
 res.writeHead(200, {'Content-Type': 'text/html'});
 res.write('Hello World!');
 res.end();
}).listen(8080);

//Read url query string

var http = require('http');
http.createServer(function (req, res) {
 res.writeHead(200, {'Content-Type': 'text/html'});
 res.write(req.url);
 res.end();
}).listen(8080);


//Split the query string

var http = require('http');
var url = require('url');

http.createServer(function (req, res) {
 res.writeHead(200, {'Content-Type': 'text/html'});
 var q = url.parse(req.url, true).query;
 var txt = q.year + " " + q.month;
 res.end(txt);
}).listen(8080);
```

OUTPUT:



executed successfully.

Ex.no.9

Date: 21.3.23

Build a template driven form using Angular(Hero employment agency to maintain personal information about heros)

AIM: To Bulid a template driven form using Angular.

PROCEDURE:

* Start the program.
* set up a new Angular project by running the command 'ng new <project_name>' in your terminal.
* Navigate into the project directory by running the command 'cd <project_name>'.
* Create a new component by running the command 'ng generate component <component_name>'.
* In the component file, import the 'FormsModule' by adding the following line at the top.
* Add the 'FormsModule' to the 'imports' array in the '@NgModule' decorator.
* Add the '(submit)' event binding to the form element and bind it to the 'onSubmit()' method.
* Run the project by running the command 'ng serve' in your terminal, and navigate to 'http://localhost:4200' in your web browser to see the form in action.
* End the program.

Ex.no.9

Date: 21.3.23

Build a template driven form using Angular(Hero employment agency to maintain personal information about heros)

AIM: TO Bulid a template driven form using Angular.

PROCEDURE:

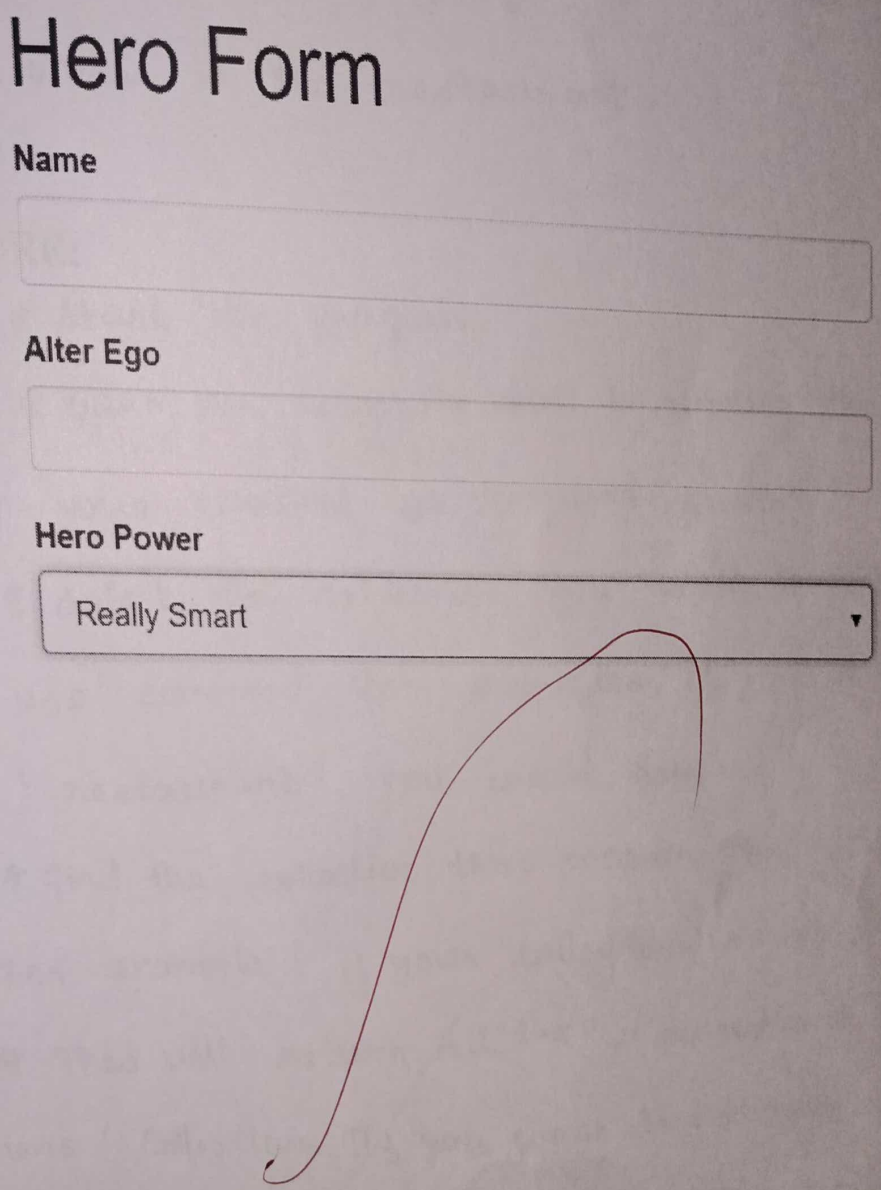
* Start the program.
* set up a new Angular project by running the command 'ng new <project-name>' in your terminal.
* Navigate into the project directory by running the command 'cd <project-name>'.
* Create a new component by running the command 'ng generate component <component-name>'.
* In the component file, import the 'FormsModule' by adding the following line at the top.
* Add the 'FormsModule' to the 'imports' array in the '@NgModule' decorator.
* Add the '(submit)' event binding to the form element and bind it to the 'onSubmit()' method.
* Run the project by running the command 'ng serve' in your terminal, and navigate to 'http://localhost:4200' in your web browser to see the form in action.
* End the program.

PROGRAM:

```
export class Hero {

 constructor(
  public id: number,
  public name: string,
  public power: string,
  public alterEgo?: string
 ) { }

}
import { Component } from '@angular/core';

import { Hero } from '../hero';

@Component({
 selector: 'app-hero-form',
 templateUrl: './hero-form.component.html',
 styleUrls: ['./hero-form.component.css']
})
export class HeroFormComponent {

  powers = ['Really Smart', 'Super Flexible',
        'Super Hot', 'Weather Changer'];

  model = new Hero(18, 'Dr. IQ', this.powers[0], 'Chuck Overstreet');

  submitted = false;

  onSubmit() { this.submitted = true; }

 }
 const myHero =  new Hero(42, 'SkyDog',
                  'Fetch any object at any distance',
                  'Leslie Rollover');
 console.log('My hero is called ' + myHero.name); // "My hero is called SkyDog"
 import { NgModule } from '@angular/core';
 import { BrowserModule } from '@angular/platform-browser';
 import { CommonModule } from '@angular/common';
 import { FormsModule } from '@angular/forms';

 import { AppComponent } from './app.component';
 import { HeroFormComponent } from './hero-form/hero-form.component';

 @NgModule({
   imports: [
   BrowserModule,
   CommonModule,
   FormsModule
   ],
   declarations: [
     AppComponent,
```

```
    HeroFormComponent
  ],
  providers: [],
  bootstrap: [ AppComponent ]
})
```

OUTPUT:



RESULT:

The program was executed successfully.

Ex.no.10

Date: 16.3.23

Write a MongoDB query to get the details about the customers in the restaurant

AIM: To write a MongoDB query to get the details about the customers in the restaurant.

PROCEDURE:

* Start the program.
* open the MongoDB shell by running the 'mongo' command in your terminal or command prompt.
* Select the database you want to work with using the 'use' command. For examples, if your database is named 'restaurant', you would run:
* find the collection that contains the customers details. for example: if your collection is named 'customers'
* This will return all the documents in the 'customers' Collection. If you want to retrieve specific details about a customer, you can use the 'find()' method with a query object that specifies the criteria to match.
* you can also use the 'findone()' method to retriere only the first document that matches the query criteria.
* End the program.

PROGRAM:

//Create Database

use restaurant

```
MongoDB Enterprise > use restaurant
switched to db restaurant
```

Create Collection

db.createCollection(“re”)

```
MongoDB Enterprise > db.createCollection("re")
{ "ok" : 1 }
MongoDB Enterprise >
```

Insert Documents inside the collection

db.re.insert([{"address":

{"building": "1007",
"coord": [-73.856077, 40.848447],
"street": "Morris Park Ave", "zipcode": "10462"}, "borough": "Bronx", "cuisine": "Bakery",
"grades": [
{"date": {"$date": 1393804800000},
"grade": "A", "score": 2},
{"date": {"$date": 1378857600000},
"grade": "A", "score": 6},
{"date": {"$date": 1358985600000},
"grade": "A", "score": 10},
{"date": {"$date": 1322006400000},
"grade": "A", "score": 9},
{"date": {"$date": 1299715200000},
"grade": "B", "score": 14}], "name":
"Morris Park Bake Shop",
"restaurant_id": "30075445"}])

```
MongoDB Enterprise > db.re.insert([{"address": {"building": "1007", "coord": [-73.856077, 40.848447], "street": "Morris Park Ave", "zipcode": "10462"}, "borough": "Bronx", "cuis
ine": "Bakery", "grades": [{"date": {"$date": 1393804800000}, "grade": "A", "score": 2}, {"date": {"$date": 1378857600000}, "grade": "A", "score": 6}, {"date": {"$date": 1358985
600000}, "grade": "A", "score": 10}, {"date": {"$date": 1322006400000}, "grade": "A", "score": 9}, {"date": {"$date": 1299715200000}, "grade": "B", "score": 14}], "name": "Morri
s Park Bake Shop", "restaurant_id": "30075445"}] )
BulkWriteResult({
    "writeErrors" : [ ],
    "writeConcernErrors" : [ ],
    "nInserted" : 1,
    "nUpserted" : 0,
    "nMatched" : 0,
    "nModified" : 0,
    "nRemoved" : 0,
    "upserted" : [ ]
})
```

Display all the documents in the collection

db.re.find().pretty()


```
{
        "_id" : ObjectId("6356a127c03d1a7d9b3b0bcc"),
        "address" : {
                "building" : "1007",
                "coord" : [
                        -73.856077,
                        40.848447
                ],
                "street" : "Morris Park Ave",
                "zipcode" : "10462"
        },
        "borough" : "Bronx",
        "cuisine" : "Bakery",
        "grades" : [
                {
                        "date" : {
                                "$date" : 1393804800000
                        },
                        "grade" : "A",
                        "score" : 2
                },
                {
                        "date" : {
                                "$date" : 1378857600000
                        },
                        "grade" : "A",
                        "score" : 6
                },
                {
                        "date" : {
                                "$date" : 1358985600000
                        },
                        "grade" : "A",
                        "score" : 10
                },
                {
                        "date" : {
                                "$date" : 1322006400000
                        },
                        "grade" : "A",
                        "score" : 9
                },
                {
                        "date" : {
                                "$date" : 1299715200000
                        },
                        "grade" : "B",
                        "score" : 14
```

Display all the documents in the collection

db.re.find().pretty()


```
{
        "_id" : ObjectId("6356a127c03d1a7d9b3b0bcc"),
        "address" : {
                "building" : "1007",
                "coord" : [
                        -73.856077,
                        40.848447
                ],
                "street" : "Morris Park Ave",
                "zipcode" : "10462"
        },
        "borough" : "Bronx",
        "cuisine" : "Bakery",
        "grades" : [
                {
                        "date" : {
                                "$date" : 1393804800000
                        },
                        "grade" : "A",
                        "score" : 2
                },
                {
                        "date" : {
                                "$date" : 1378857600000
                        },
                        "grade" : "A",
                        "score" : 6
                },
                {
                        "date" : {
                                "$date" : 1358985600000
                        },
                        "grade" : "A",
                        "score" : 10
                },
                {
                        "date" : {
                                "$date" : 1322006400000
                        },
                        "grade" : "A",
                        "score" : 9
                },
                {
                        "date" : {
                                "$date" : 1299715200000
                        },
                        "grade" : "B",
                        "score" : 14
```

```
MongoDB Enterprise > db.re.insert([{"address": {"building": "1007", "coord": [-73.856077, 40.848447], "street": "Morris Park Ave", "zipcode": "10462"}, "borough": "Bronx", "cuis
ine": "Bakery", "grades": [{"date": {"$date": 1393804800000}, "grade": "A", "score": 2}, {"date": {"$date": 1378857600000}, "grade": "A", "score": 6}, {"date": {"$date": 1358985
600000}, "grade": "A", "score": 10}, {"date": {"$date": 1322006400000}, "grade": "A", "score": 9}, {"date": {"$date": 1299715200000}, "grade": "B", "score": 14}], "name": "Morri
s Park Bake Shop", "restaurant_id": "30075445"}] )
BulkWriteResult({
    "writeErrors" : [ ],
    "writeConcernErrors" : [ ],
    "nInserted" : 1,
    "nUpserted" : 0,
    "nMatched" : 0,
    "nModified" : 0,
    "nRemoved" : 0,
    "upserted" : [ ]
```

Find the restaurants that achieved a score, more than 10 but less than 40

db.re.find({grades : { $elemMatch:{"score":{$gt : 10 , $lt :40}}}}).pretty();

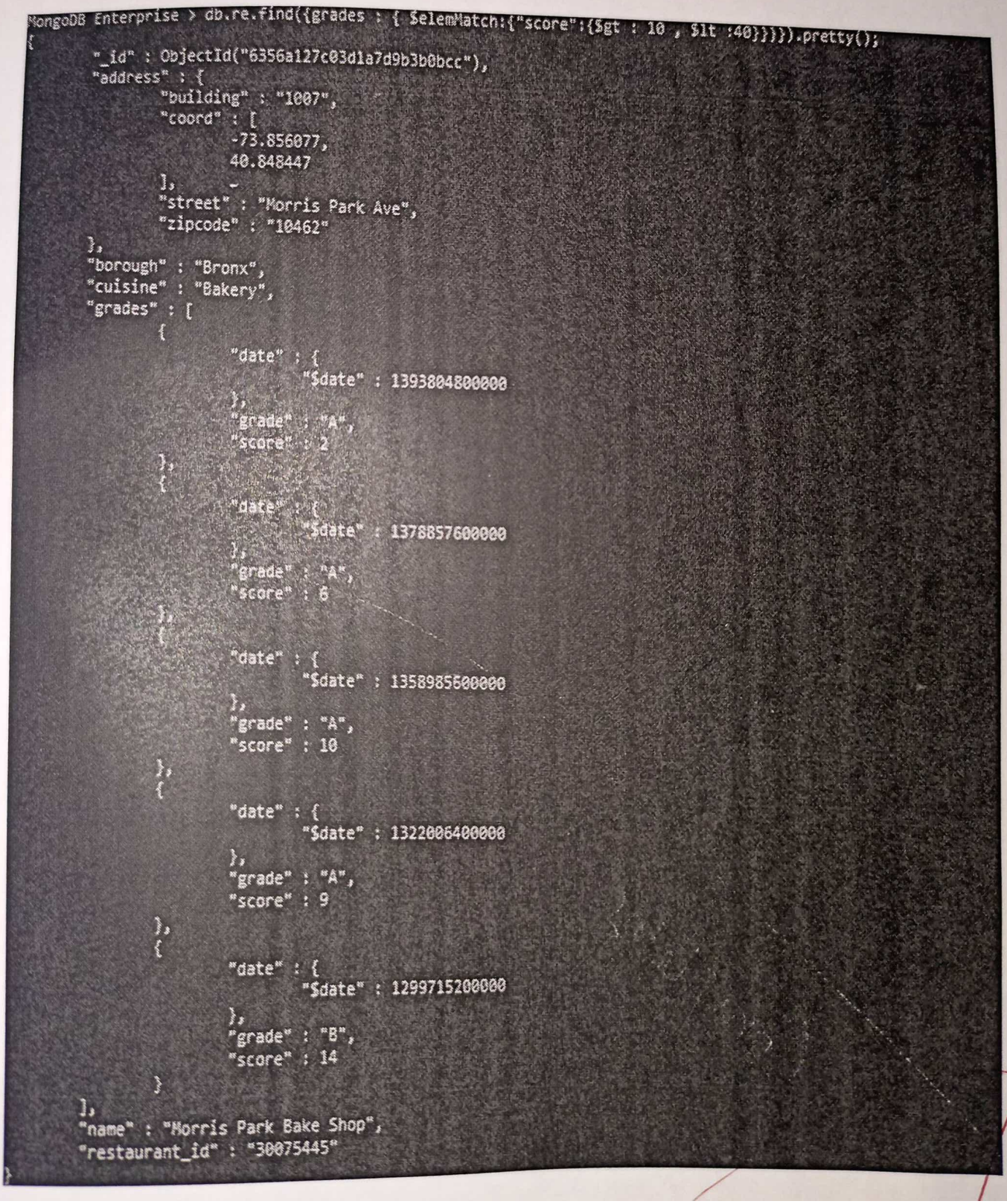
```
MongoDB Enterprise > db.re.find({grades : { $elemMatch:{"score":{$gt : 10 , $lt :40}}}}).pretty();
{
        "_id" : ObjectId("6356a127c03d1a7d9b3b0bcc"),
        "address" : {
                "building" : "1007",
                "coord" : [
                        -73.856077,
                        40.848447
                ],
                "street" : "Morris Park Ave",
                "zipcode" : "10462"
        },
        "borough" : "Bronx",
        "cuisine" : "Bakery",
        "grades" : [
                {
                        "date" : {
                                "$date" : 1393804800000
                        },
                        "grade" : "A",
                        "score" : 2
                },
                {
                        "date" : {
                                "$date" : 1378857600000
                        },
                        "grade" : "A",
                        "score" : 6
                },
                {
                        "date" : {
                                "$date" : 1358985600000
                        },
                        "grade" : "A",
                        "score" : 10
                },
                {
                        "date" : {
                                "$date" : 1322006400000
                        },
                        "grade" : "A",
                        "score" : 9
                },
                {
                        "date" : {
                                "$date" : 1299715200000
                        },
                        "grade" : "B",
                        "score" : 14
                }
        ],
        "name" : "Morris Park Bake Shop",
        "restaurant_id" : "30075445"
}
```

RESULT:

The program was executed successfully.

Find the restaurants that achieved a score, more than 10 but less than 40

db.re.find({grades : { $elemMatch:{"score":{$gt : 10 , $lt :40}}}}).pretty();



```
MongoDB Enterprise > db.re.find({grades : { $elemMatch:{"score":{$gt : 10 , $lt :40}}}}).pretty();
{
        "_id" : ObjectId("6356a127c03d1a7d9b3b0bcc"),
        "address" : {
                "building" : "1007",
                "coord" : [
                        -73.856077,
                        40.848447
                ],
                "street" : "Morris Park Ave",
                "zipcode" : "10462"
        },
        "borough" : "Bronx",
        "cuisine" : "Bakery",
        "grades" : [
                {
                        "date" : {
                                "$date" : 1393804800000
                        },
                        "grade" : "A",
                        "score" : 2
                },
                {
                        "date" : {
                                "$date" : 1378857600000
                        },
                        "grade" : "A",
                        "score" : 6
                },
                {
                        "date" : {
                                "$date" : 1358985600000
                        },
                        "grade" : "A",
                        "score" : 10
                },
                {
                        "date" : {
                                "$date" : 1322006400000
                        },
                        "grade" : "A",
                        "score" : 9
                },
                {
                        "date" : {
                                "$date" : 1299715200000
                        },
                        "grade" : "B",
                        "score" : 14
                }
        ],
        "name" : "Morris Park Bake Shop",
        "restaurant_id" : "30075445"
}
```