C# and Angular Full-Stack Developer Interview Test

Section 1: C# (Back-End)

1. Object-Oriented Programming:
   * Write a C# class that demonstrates inheritance, polymorphism, encapsulation, and abstraction.

The answer is below.

For runnable code, please go to ShapeAreaDiameterDemo and run the ShapeAreaDiameterDemo.sln

|  |  |
| --- | --- |
| 001  002  003  004  005  006  007  008  009  010  011  012  013  014  015  016  017  018  019  020  021  022  023  024  025  026  027  028  029  030  031  032  033  034  035  036  037  038  039  040  041  042  043  044  045  046  047  048  049  050  051  052  053  054  055  056  057  058  059  060  061  062  063  064  065  066  067  068  069  070  071  072  073  074  075  076  077  078  079  080  081  082  083  084  085  086  087  088  089  090  091  092  093  094  095  096  097  098  099  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119 | using System;  using System.Collections.Generic;  using System.Linq;  using System.Runtime.InteropServices;  using System.Security.Cryptography.X509Certificates;  using System.Text;  using System.Threading.Tasks;    namespace ShapeAreaDiameterDemo  {      public abstract **class** Shape      {  **//** Abstract properties **for** Area **and** Diameter  **//** These properties must be implemented by derived classes  **//** to provide specific calculations **for** each shape.            public const double PI **=** Math.PI; **//** Constant **for** PI          protected double \_x, \_y;            protected Shape()          {          }            public Shape(double x, double y)          {              \_x **=** x;              \_y **=** y;          }            public virtual double CalculateArea()          {  **return** \_x **\*** \_y; **//** Default implementation, can be overridden          }          public virtual double CalculateDiameter()          {  **return** (\_x **\*** 2 **+** \_y **\*** 2); **//** Default implementation, can be overridden          }      }        public **class** Circle : Shape      {  **//** Circle specific properties          private double \_radius;            public double Radius          {              get { **return** \_radius; }              set { \_radius **=** value; }          }          public Circle(double radius): base(radius, 0) **//** Assuming \_x **and** \_y are both radius **for** Circle          {              Radius **=** radius;          }            public override double CalculateArea()          {  **//** Area **=** π **\*** r^2  **return** PI **\*** Radius **\*** Radius; **//** Assuming \_x **is** the radius          }            public override double CalculateDiameter()          {  **//** Diameter **=** 2 **\*** r  **return** 2 **\*** Radius **\*** PI; **//** Assuming \_x **is** the radius          }        }          public **class** Rectangle : Shape      {          private double \_width;          private double \_height;    **//** Rectangle specific properties          public double Width          {              get { **return** \_width; }              set { \_width **=** value > 0 ? value : throw new ArgumentException("Width must be positive"); }          }            public double Height          {              get { **return** \_height; }              set { \_height **=** value > 0 ? value : throw new ArgumentException("Height must be positive"); }          }            public Rectangle(double width, double height)          {              Width **=** width;              Height **=** height;          }            public override double CalculateArea()          {  **return** Width **\*** Height; **//** Area **=** width **\*** height          }            public override double CalculateDiameter()          {  **return** Width **\*** 2 **+** Height **\*** 2; **//** Diameter **=** 2 **\*** (width **+** height)          }      }    **class** Program      {          static void Main(string[] args)          {  **//** Create a Circle **and** calculate its area **and** diameter              Circle circle **=** new Circle(5);              Console.WriteLine($"Circle Area: {circle.CalculateArea()}");              Console.WriteLine($"Circle Diameter: {circle.CalculateDiameter()}");  **//** Create a Rectangle **and** calculate its area **and** diameter              Rectangle rectangle **=** new Rectangle(4, 6);              Console.WriteLine($"Rectangle Area: {rectangle.CalculateArea()}");              Console.WriteLine($"Rectangle Diameter: {rectangle.CalculateDiameter()}");          }      }  } |

1. Data Structures and Algorithms:
   * Implement a binary search tree in C#. Include methods for insertion, deletion, and searching.

* Please look at the answer below or run the code in BinarySearchTree folder’s BinarySearchTree.sln with visual studio or vscode.

|  |  |
| --- | --- |
| 001  002  003  004  005  006  007  008  009  010  011  012  013  014  015  016  017  018  019  020  021  022  023  024  025  026  027  028  029  030  031  032  033  034  035  036  037  038  039  040  041  042  043  044  045  046  047  048  049  050  051  052  053  054  055  056  057  058  059  060  061  062  063  064  065  066  067  068  069  070  071  072  073  074  075  076  077  078  079  080  081  082  083  084  085  086  087  088  089  090  091  092  093  094  095  096  097  098  099  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196 | public **class** BinarySearchTree  {      static void Main(string[] args)      {          BinarySearchTree bst **=** new BinarySearchTree();          bst.Insert(10);          bst.Insert(5);          bst.Insert(15);          bst.Insert(3);          bst.Insert(7);          bst.Insert(12);          bst.Insert(18);            Console.WriteLine("Binary Search Tree created and values inserted.");            Console.WriteLine("== Display List: ==");          bst.traverseInOrder();            Node minNode **=** bst.FindMin(bst.root);          Console.WriteLine($"Minimum value in the BST: {minNode.Value}");            Node maxNode **=** bst.FindMax(bst.root);          Console.WriteLine($"Maximum value in the BST: {maxNode.Value}");            int searchValue0 **=** 10;          Node searchResult0 **=** bst.Search(searchValue0);          Console.WriteLine(searchResult0 !**=** null ? $"Found node with value: {searchResult0.Value}" : $"Node with {searchValue0} not found.");            bst.Delete(10);          Console.WriteLine("Deleted node with value 10.");            Console.WriteLine("== Display List: ==");          bst.traverseInOrder();            int searchValue1 **=** 7;          int searchValue2 **=** 44;          int searchValue3 **=** 10;          Node searchResult1 **=** bst.Search(searchValue1);          Node searchResult2 **=** bst.Search(searchValue2);          Console.WriteLine(searchResult1 !**=** null ? $"Found node with value: {searchResult1.Value}" : $"Node with {searchValue1} not found.");          Console.WriteLine(searchResult2 !**=** null ? $"Found node with value 2: {searchResult2.Value}" : $"Node with {searchValue2} not found.");          Node searchResult3 **=** bst.Search(searchValue3);          Console.WriteLine(searchResult3 !**=** null ? $"Found node with value 3: {searchResult3.Value}" : $"Node with {searchValue3} not found.");    **//** Console.WriteLine("Max Value", bst.FindMax(bst.root));  **//** Console.WriteLine("Min Value", bst.FindMin(bst.root));        }        public **class** Node      {          public int Value;          public Node Left;          public Node Right;            public Node(int value)          {              Value **=** value;              Left **=** null;              Right **=** null;          }      }        private Node root;        public BinarySearchTree()      {          root **=** null;      }        public void Insert(int value)      {          root **=** InsertRec(root, value);      }        private Node InsertRec(Node root, int value)      {  **if** (root **==** null)          {  **return** new Node(value);          }    **if** (value < root.Value)          {              root.Left **=** InsertRec(root.Left, value);          }  **else** **if** (value > root.Value)          {              root.Right **=** InsertRec(root.Right, value);          }    **return** root;      }        public Node FindMin(Node node)      {  **while** (node.Left !**=** null)          {              node **=** node.Left;          }  **return** node;      }          public Node FindMax(Node node)      {  **while** (node.Right !**=** null)          {              node **=** node.Right;          }  **return** node;      }          public Node Delete(int value)      {  **return** DeleteRec(root, value);      }        private Node DeleteRec(Node root, int value)      {  **if** (root **==** null)          {  **return** root;          }    **if** (value < root.Value)          {              root.Left **=** DeleteRec(root.Left, value);          }  **else** **if** (value > root.Value)          {              root.Right **=** DeleteRec(root.Right, value);          }  **else**          {  **//** Node with only one child **or** no child  **if** (root.Left **==** null)              {  **return** root.Right;              }  **else** **if** (root.Right **==** null)              {  **return** root.Left;              }    **//** Node with two children: Get the **in-**order successor (smallest **in** the right subtree)              Node temp **=** FindMin(root.Right);              root.Value **=** temp.Value; **//** Copy the **in-**order successor's value to this node              root.Right **=** DeleteRec(root.Right, temp.Value); **//** Delete the inorder successor          }  **return** root;      }          public Node Search(int value)      {  **return** SearchRec(root, value);      }      private Node SearchRec(Node root, int value)      {  **if** (root **==** null || root.Value **==** value)          {  **return** root;          }    **if** (value < root.Value)          {  **return** SearchRec(root.Left, value);          }  **else**          {  **return** SearchRec(root.Right, value);          }      }        public Node traverseInOrder()      {  **return** traverseInOrderRec(root);      }        public Node traverseInOrderRec(Node node)      {  **if** (node **==** null)          {  **return** null;          }           traverseInOrderRec(node.Left);          Console.WriteLine(node.Value);          traverseInOrderRec(node.Right);  **return** node;      }  } |

1. ASP.NET Core:
   * Create a simple RESTful API using ASP.NET Core. The API should have endpoints for CRUD operations on a "Product" entity.

- minimum code for Web Api without using separate the Controller and Services is bellowing.

- I am using .NET Web API in Memory extension to simulate the db. ProductContext.cs can connect to any database connection string with sqlClient or other ODBC driver or other db driver.  
- I don’t have a actual program for this answer. For more detail runnable Dependency Injection example, please run the ProductApi.sln within the folder ProductApi folder.

|  |  |
| --- | --- |
| 001  002  003  004  005  006  007  008  009  010  011  012  013  014  015  016  017  018  019  020  021  022  023  024  025  026  027  028  029  030  031  032  033  034  035  036  037  038  039  040  041  042  043  044  045  046  047  048  049  050  051  052  053  054  055  056  057  058  059  060  061  062  063  064  065  066  067  068  069  070  071  072  073  074  075  076  077  078  079  080  081  082  083  084  085  086  087  088  089  090  091  092  093  094  095  096  097  098  099  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145 | using System;  using System.Collections.Generic;  using System.Linq;  using System.Threading.Tasks;  using Microsoft.AspNetCore.Http;  using Microsoft.AspNetCore.Mvc;  using Microsoft.EntityFrameworkCore;  using NuGet.Versioning;  using ProductAPI.Models;    namespace ProductAPI.Controllers  {      [Route("api/[controller]")]      [ApiController]      public **class** ProductsController : ControllerBase      {          private readonly ProductContext \_context;            public ProductsController(ProductContext context)          {              \_context **=** context;          }    **//** GET: api**/**Products          [HttpGet]          public async Task<ActionResult<IEnumerable<Product>>> GetProducts()          {  **return** await \_context.Products.ToListAsync();          }    **//** GET: api**/**Products**/**5          [HttpGet("{id}")]          public async Task<ActionResult<Product>> GetProduct(int id)          {              var product **=** await \_context.Products.FindAsync(id);    **if** (product **==** null)              {  **return** NotFound();              }    **return** product;          }            [HttpGet("search/{searchTerm}")]          public async Task<ActionResult<IEnumerable<Product>>> GetProductsBySearchTerm(string searchTerm)          {              var productList **=** await \_context.Products.ToListAsync();    **//** Use LINQ to filter products based on the search term              var products **=** productList.Where(r **=**> r.Name.Contains(searchTerm, StringComparison.OrdinalIgnoreCase)).ToList();    **if** (!products.Any())              {  **return** NotFound();              }    **return** Ok(products); **//** Wrap the result **in** Ok() to **return** ActionResult<IEnumerable<Product>>          }      **//** PUT: api**/**Products**/**5  **//** To protect **from** overposting attacks, see https:**//**go.microsoft.com**/**fwlink**/**?linkid**=**2123754          [HttpPut("{id}")]          public async Task<IActionResult> PutProduct(int id, Product product)          {  **if** (id !**=** product.Id)              {  **return** BadRequest();              }                \_context.Entry(product).State **=** EntityState.Modified;    **try**              {                  await \_context.SaveChangesAsync();              }              catch (DbUpdateConcurrencyException)              {  **if** (!ProductExists(id))                  {  **return** NotFound();                  }  **else**                  {                      throw;                  }              }    **return** NoContent();          }    **//** POST: api**/**Products  **//** To protect **from** overposting attacks, see https:**//**go.microsoft.com**/**fwlink**/**?linkid**=**2123754          [HttpPost]          public async Task<ActionResult<Product>> PostProduct(Product product)          {              \_context.Products.Add(product);              await \_context.SaveChangesAsync();    **return** CreatedAtAction("GetProduct", new { id **=** product.Id }, product);          }    **//** DELETE: api**/**Products**/**5          [HttpDelete("{id}")]          public async Task<IActionResult> DeleteProduct(int id)          {              var product **=** await \_context.Products.FindAsync(id);  **if** (product **==** null)              {  **return** NotFound();              }                \_context.Products.Remove(product);              await \_context.SaveChangesAsync();    **return** NoContent();          }            private bool ProductExists(int id)          {  **return** \_context.Products.Any(e **=**> e.Id **==** id);          }            [HttpGet("AddSampleData")]          public async Task<IActionResult> AddSampleData()          {  **if** (\_context.Products.Any())              {  **return** BadRequest("Sample data already exists.");              }              var sampleProducts **=** new List<Product>              {                  new Product { Name **=** "Sample Product 1", Description **=** "Description for Sample Product 1", Price **=** 10.99, StockQty **=** 100, Category **=** "Category1", ImageUrl **=** "1.png" },                  new Product { Name **=** "Sample Product 2", Description **=** "Description for Sample Product 2", Price **=** 20.99, StockQty **=** 50, Category **=** "Category2", ImageUrl **=** "2.png" },                  new Product { Name **=** "Sample Product 1a", Description **=** "Description for Sample Product 1a", Price **=** 30.99, StockQty **=** 75, Category **=** "Category1a", ImageUrl **=** "1.png" },                  new Product { Name **=** "Sample Product 3", Description **=** "Description for Sample Product 3", Price **=** 40.99, StockQty **=** 175, Category **=** "Category3", ImageUrl **=** "3.png" },                  new Product { Name **=** "Sample Product 4", Description **=** "Description for Sample Product 4", Price **=** 30.99, StockQty **=** 75, Category **=** "Category4", ImageUrl **=** "2.png" },              };              \_context.Products.AddRange(sampleProducts);              await \_context.SaveChangesAsync();  **return** Ok();          }      }  } |

Section 2: Angular (Front-End)

* Create an Angular component that displays a list of products. Implement two-way data binding to allow users to add, edit, and delete products.

1. Services and Dependency Injection:

* Write an Angular service that communicates with the RESTful API created in the C# section. Use dependency injection to provide the service to components.
* Answer: Please check out and run code in ProductApp (angular application) for detail inform.
  + The modules folder will have all the components.
  + The service folder will contains the service to call the API in <https://localhost:7285/api/Products>
  + This service is injected into the component for each component for listing, add, edit, delete
  + Npm install is needed to install all needed nodejs modules.
  + After npm install, please open a terminal to run with command “npm run start”

**Section 3: Full-Stack Integration**

1. End-to-End Application:

* Develop a full-stack application that includes both the C# back-end and Angular front-end. The application should allow users to manage products (add, edit, delete, view).
  + Please run the ProductApp Projects with VSCode along with ProductApi .net web api application

1. Testing:

* Write unit tests for the C# API using a testing framework like xUnit or NUnit.
  + Ans: Unit Testing is for compare Search function with word “Product 1” in the name field (this search is only search by product name only)
  + \*\*Please press **control+F5** to run ProductApi without debugger to run a separate instant of the program.
  + \*\*And run another instance of Unit Test within the ProductApi project.
  + It will return 2 records of product with search term “Product 1”and it will return product name “Product 1” and the other product with name of “Product 1a”
  + The unit test is supposed to assert
    - 2 records has been returned
    - Assert.Equal(products[0].Name, result[0].Name);
      * Will assert result[0].Name = Result[0].Name
    - Assert.Equal(products[1].Name, result[1].Name);
      * Will assert result[1].Name = Result[1].Name
    - 1 record contains “Product 1”
    - 1 other record contains “Product 1a”



|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89 | using Microsoft.AspNetCore.Builder;  using Microsoft.EntityFrameworkCore;  using Microsoft.Extensions.DependencyInjection;  using ProductAPI.Controllers;  using ProductAPI.Models;  using System.Net.Http;  using System.Net.NetworkInformation;  using System.Text.Json;    namespace UnitTests  {      public **class** UnitTestForWebApi      {            private readonly HttpClient \_httpClient **=** new() { BaseAddress **=** new Uri("https://localhost:7285") };            [Fact]          public async Task searchTermTestAsync()          {  **//** 1. Arrange    **//** expected result of the search API call              List<Product> products **=** new List<Product>              {              new Product              {                      Id **=** 1,  **Name = "Sample Product 1",**                      Description **=** "Description for Sample Product 1",                      Price **=** 10.99,                      StockQty **=** 100,                      Category **=** "Category1",                      ImageUrl **=** "1.png"              },                  new Product {                      Id **=** 3,  **Name = "Sample Product 1a",**                      Description **=** "Description for Sample Product 1a",                      Price **=** 30.99,                      StockQty **=** 75,                      Category **=** "Category1a",                      ImageUrl **=** "1.png"                  },              };    **//** setup sample data              var apiUrlForAddSampleData **=** "https://localhost:7285/api/Products/AddSampleData";              var response\_AddSampleData **=** await \_httpClient.GetAsync(apiUrlForAddSampleData);      **//** 2. Act                var apiUrl **=** "https://localhost:7285/api/Products/search/Product 1";              var response **=** await \_httpClient.GetAsync(apiUrl);    **//** 3. Assert                response.EnsureSuccessStatusCode();              System.Diagnostics.Debug.WriteLine(response.StatusCode);              var content **=** await response.Content.ReadAsStringAsync();              var result **=** JsonSerializer.Deserialize<List<Product>>(content, new JsonSerializerOptions { PropertyNameCaseInsensitive **=** true });      **//** Assertions **for** the result **is** **not** null              Assert.NotNull(result);    **//** Assertions **for** the 2 counts of products **return** expected              Assert.Equal(2, result.Count);    **//** since this search API only search the "Name" field, therefore we only  **//** compare the name property of the expected name result **and** actual name result.  **if** (result.Count **==** 2)              {                  Assert.Equal(products[0].Name, result[0].Name);                  Assert.Equal(products[1].Name, result[1].Name);              }      **//** Assertions the acutal result **for** the product names  **//** of the expected result              Assert.Contains(result, p **=**> p.Name **==** "Sample Product 1");              Assert.Contains(result, p **=**> p.Name **==** "Sample Product 1a");            }      }  } |