**Developer Assessment – Coding Assignment**

Application Requirements

* Write a restful api that takes in a name and address.
* Persist those inputs.
* Name field should be considered as a unique set.
* Return proper messages for validation, duplicates, and errors.
* Allow updates to those inputs.
* Allow deletes to those inputs.
* Return all inputs persisted.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  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 | **using** **System**;  **using** **System.Collections.Generic**;  **using** **System.Linq**;  **using** **System.Threading.Tasks**;  **using** **Microsoft.AspNetCore.Mvc**;  **using** **Microsoft.EntityFrameworkCore**;  **namespace** **RestfulAPI.Models**  {  **public** **class** **User**  {  **public** **int** Id { **get**; **set**; }  **public** **string** Name { **get**; **set**; }  **public** **string** Address { **get**; **set**; }  }  **public** **class** **UserContext** : DbContext  {  **public** **UserContext**(DbContextOptions<UserContext> options) : **base**(options)  {  }  **public** DbSet<User> Users { **get**; **set**; }  }  [Route("api/[controller]")]  [ApiController]  **public** **class** **UserController** : ControllerBase  {  **private** **readonly** UserContext \_context;  **public** **UserController**(UserContext context)  {  \_context = context;  }  [HttpGet]  **public** **async** Task<ActionResult<IEnumerable<User>>> GetUsers()  {  **return** **await** \_context.Users.ToListAsync();  }  [HttpPost]  **public** **async** Task<ActionResult<User>> PostUser(User user)  {  **var** existingUser = **await** \_context.Users.SingleOrDefaultAsync(u => u.Name == user.Name);  **if** (existingUser != **null**)  {  **return** **BadRequest**("A user with the same name already exists.");  }  \_context.Users.Add(user);  **await** \_context.SaveChangesAsync();  **return** **CreatedAtAction**("GetUser", **new** { id = user.Id }, user);  }  [HttpPut("{id}")]  **public** **async** Task<IActionResult> PutUser(**int** id, User user)  {  **if** (id != user.Id)  {  **return** **BadRequest**();  }  \_context.Entry(user).State = EntityState.Modified;  **await** \_context.SaveChangesAsync();  **return** **NoContent**();  }  [HttpDelete("{id}")]  **public** **async** Task<IActionResult> DeleteUser(**int** id)  {  **var** user = **await** \_context.Users.FindAsync(id);  **if** (user == **null**)  {  **return** **NotFound**();  }  \_context.Users.Remove(user);  **await** \_context.SaveChangesAsync();  **return** **NoContent**();  }  }  } |

Technical Requirements

* A .NET Core restful api.
* Use of Dependency Injection.
* Make use SOLID principals.
* Incorporate swagger to test endpoints.
* The persistence can be an in-memory database or a simple no sql db.
* Use a caching layer to retrieve persisted data and keep cache current for all updates.
* Write unit tests making use of mocks.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  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  90  91  92  93  94  95  96  97  98  99  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 | **using** **Microsoft.AspNetCore.Builder**;  **using** **Microsoft.AspNetCore.Hosting**;  **using** **Microsoft.Extensions.DependencyInjection**;  **using** **Microsoft.Extensions.Hosting**;  **using** **Microsoft.OpenApi.Models**;  **using** **System**;  **using** **System.Collections.Generic**;  **using** **System.Linq**;  **namespace** **MyAPI**  {  **public** **class** **Startup**  {  **public** **void** **ConfigureServices**(IServiceCollection services)  {  services.AddControllers();  // Register the in-memory database  services.AddSingleton<IDatabase>(sp => **new** InMemoryDatabase());  // Register the caching layer  services.AddSingleton<ICache>(sp => **new** MemoryCache(TimeSpan.FromMinutes(**30**)));  // Register the repository, making use of dependency injection  services.AddTransient<IRepository, Repository>();  // Add Swagger for API testing  services.AddSwaggerGen(c =>  {  c.SwaggerDoc("v1", **new** OpenApiInfo { Title = "My API", Version = "v1" });  });  }  **public** **void** **Configure**(IApplicationBuilder app, IWebHostEnvironment env)  {  **if** (env.IsDevelopment())  {  app.UseDeveloperExceptionPage();  }  // Use Swagger to test endpoints  app.UseSwagger();  app.UseSwaggerUI(c =>  {  c.SwaggerEndpoint("/swagger/v1/swagger.json", "My API V1");  });  app.UseRouting();  app.UseEndpoints(endpoints =>  {  endpoints.MapControllers();  });  }  }  // Implement the IDatabase interface as an in-memory database  **public** **class** **InMemoryDatabase** : IDatabase  {  **private** **readonly** List<**object**> \_data = **new** List<**object**>();  **public** **void** **Add**(**object** item)  {  \_data.Add(item);  }  **public** IEnumerable<**object**> GetAll()  {  **return** \_data;  }  }  // Implement the ICache interface as an in-memory cache  **public** **class** **MemoryCache** : ICache  {  **private** **readonly** TimeSpan \_expiration;  **private** **readonly** Dictionary<**string**, CacheItem> \_cache = **new** Dictionary<**string**, CacheItem>();  **public** **MemoryCache**(TimeSpan expiration)  {  \_expiration = expiration;  }  **public** T Get<T>(**string** key) **where** T : **class**  {  **if** (\_cache.TryGetValue(key, **out** **var** item) && item.Expiration > DateTime.Now)  {  **return** item.Value **as** T;  }  **return** **null**;  }  **public** **void** Set<T>(**string** key, T **value**) **where** T : **class**  {  \_cache[key] = **new** CacheItem  {  Expiration = DateTime.Now.Add(\_expiration),  Value = **value**  };  }  }  // Implement the IRepository interface, making use of SOLID principles and dependency injection.  **public** **class** **Repository** : IRepository  {  **private** **readonly** IDatabase \_database;  **private** **readonly** ICache \_cache;  **public** **Repository**(IDatabase database, ICache cache)  {  \_database = database;  \_cache = cache;  }  **public** T Get<T>(**string** key) **where** T : **class**  {  **var** item = \_cache.Get<T>(key);  **if** (item != **null**)  {  **return** item;  }  item = \_database.GetAll().OfType<T>().FirstOrDefault();  **if** (item != **null**)  {  \_cache.Set(key, item);  }  **return** item;  }  **public** **void** Add<T>(T item) **where** T : **class**  {  \_database.Add(item);  \_cache.Set(item.GetType().Name, item);  }  }  // Example interface for the database  **public** **interface** IDatabase  {  **void** **Add**(**object** item);  IEnumerable<**object**> GetAll();  }  // Example interface for the cache  **public** **interface** ICache  {  T Get<T>(**string** key) **where** T : class;  **void** Set<T>(**string** key, T **value**) **where** T : class;  }  // Example cache item class  **public** **class** **CacheItem**  {  **public** DateTime Expiration { **get**; **set**; }  **public** **object** Value { **get**; **set**; }  }  // Example repository interface  **public** **interface** IRepository  {  T Get<T>(**string** key) **where** T : class;  **void** Add<T>(T item) **where** T : class;  }  } |

Tooling Requirements

* Use Visual Studio, latest version if possible.
* Keep all projects in one solution.
* The solution should compile with no errors.
* The application should run cleanly.
* Easy local setup to ensure solution can be executed.
* If nosql db used, preference is RavenDb .
* Use .NET Core 6.
* Caching layer can be In Memory
* Unit tests can be written using MSTests, NUnit, or Xunit.
* Use Mocks when unit testing persisted data.
* All tests should be rerunnable.

Please post solution to a public GitHub or any shareable file location.