Survay Basket Project

• Add Polls Controller:

Notes

→ ControllerBase: abstract class every controller in API inherit from it.

→ Controller [used in MVC]: inherit from ControllerBase, and add Some Properities and Methods.

⇒ in Program.cs we have :

1. builder.Services.AddControllers();

which means we will use Controllers in this API, Some APIs could be built without controllers

2. app.MapControllers();

هنا بيبدأ الـ Scan لكل الـ Controllers اللي موجودة في البروجيكت وياخدهم كلهم الـ Routes اللي موجودة جواهم (كدا هو عنده List بالـ Routes دي)، ولما بيجي request عليها بيكون عارف من الـ Controller اللي هيروح ليها.

→ [Route("")]

بتتكتب على الـ Controllers والـ Action واللي موجودة جوا الـ Controllers

```
[Route("api/[controller]")]   // api/Polls
[APIController]

Public class PollsController : ControllerBase
{
}
```

• Add GetAll EndPoint:

IActionResult ⇒ (... string, StatusCode) لازم ترجع نوع

بنبدأ الأول بـ Domain

1. هنكون Poll

```
Public class Poll
{
    Public int Id {get; Set;}
    Public String Title {get; set;} = String.Empty;
    Public String Description {get; set;} = String.Empty;
}
```

```
[HttpGet("{id}")]
Public IActionResult Get(int id)
{  var Poll = _PollService.Get(id);
   return Poll is null ? NotFound() : Ok(Poll);
}
}
```

هنروح نعمل register لـ Service في الـ Program

```
builder.Services.AddScoped<IPollService, PollService>();
```

Add new EndPoint:

① هنضيف في الـ Interface (IPollService)

```
Poll Add(Poll poll);
```

② هنروح في الـ Class (PollService) implement

```
Public Poll Add(Poll poll)
{  poll.Id = _Polls.Count + 1;
   _Polls.Add(poll);
   return poll;
}
```

③ هنروح على الـ PollsController

Domain Model.

```
[HttpPost("")]
Public IActionResult Add(Poll poll)
{
  var newPoll = _pollService.Add(poll);
                  → 201 (StatusCode
  return CreatedAtAction(nameof(Get), new { id = newPoll.Id }, newPoll);
```

الـ Action لاسم الـ

الـ routes اللي هو محتاجها الـ Id عندي

الـ value الجديدة اللي حصلها creation بالكامل.

هيشوف هنا الـ URI الخاص بال resource اللي حصله creation

Update Poll EndPoint:

Interface [IPollService] ①

```
bool Update (int id, Poll Poll);
```

Class [PollService] ②

```
Public bool Update (int id, Poll Poll)
{
    var CurrentPoll = Get (id);
    if ( CurrentPoll is null)
      return false

    current Poll .Title = Poll.Title;
    Current Poll. Description = Poll. Description;
    return true;
}
```

PollsController ③

```
[HttpPut ("{id}")]
Public IActionResult Update (int id, Poll request)
{
  var isUpdated = _PollService .Update (id, request);
  if (! isUpdated)
      return NotFound ();

  return NoContent ();      // 204
}
```

Delete Poll EndPoint:

Interface [IPollService] ①

```
bool Delete (int id);
```

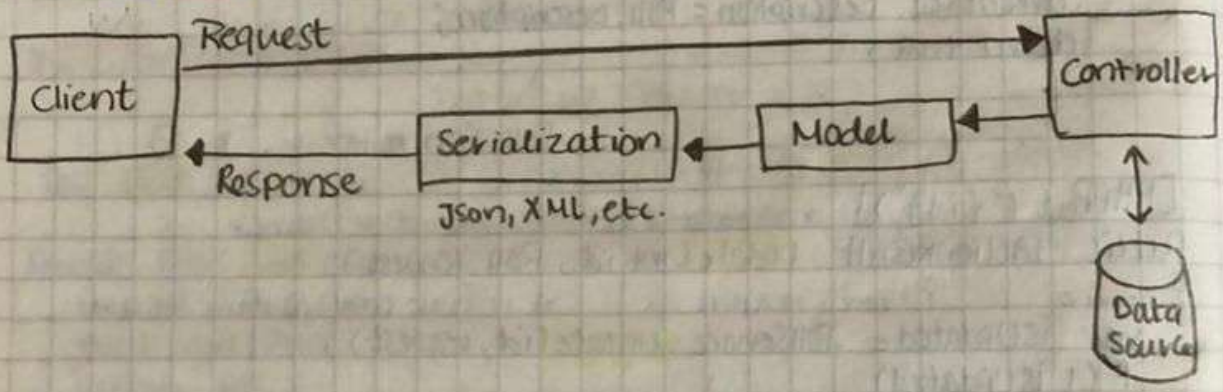
Class [PollService] ②

```
Public bool Delete (int id)
{
  var currentPoll = Get (id);
  if (currentPoll is null)
      return False;

  polls. Remove (currentPoll);
  return true;
}
```

PollsController

```
[HttpDelete("{id}")
public IActionResult Delete(int id)
{
    var isDeleted = _pollService.Delete(id);
    if(!isDeleted)
        return NotFound();

    return NoContent();
}
```

what happen?



Route Constraints:

لو إني عندي في الـ Routes وعوزه يكون int بس ↩ Id

```
[HttpGet("{id:int}")]
```

Constraints:	Example:	Notes:
- int	{id:int}	→ Matches any integer.
- bool	{Active:bool}	→ Matches true or false [case insensitive]
- datetime	{dob:datetime}	→ Matches a valid DateTime value in the invariant culture.
- decimal	{Price:decimal}	→ Match the valid decimal value
- double	{weight:double}	
- float	{weight:float}	
- guid	{id:guid}	
- long	{ticks:long}	

datetime ⇒ Must be in fromate: ↳ 2016-12-31 Or
↳ 2016-12-31 7:32 Pm.

we can add another group of Constvaints.

String:
- minlength (value)
- maxlength (value)
- length (value)
- length (min, max)

Integers:
- min (value)
- max (value)
- range (min, max)

- alpha

{username : minlength(4)} ⇒ String must be at least 4 characters.

{Filename : length(12)} ⇒ String must be exactly 12

{age : min(18)} ⇒ integer value must be atleast 18

{name : alpha} ⇒ String must consist of one or more alphabetical characters, a-Z and case-insensitive.

Regular Expression
- regex (expression)

{ssn : regex(^\d{3}-\d{2}-\d{4}$)}
↳ 123-45-6789
↳ String must match the regular expression.

- required {name : required} ⇒ used to enforces a non-nullable Parameter.

[HttpGet("{id:int:min(10)}")] ← نقدر نستخدم أكتر من constraint في نفس الوقت

Parameter Binding:

← الـ Frame work (.NET) هو اللي هيأخذ الـ Parameter اللي جايه مع الـ HttpRequest ، وبيأ يدورلها الـ endPoint المسئولة عن أنها تهندل الـ request دا في بتاع الـ DataType اللي الـ endPoint مستنيها.

← عشان الـ End Point تستقبل الـ Parameter من أكتر من مكان:
لو عايزين نعرف الـ Parameter دا جاي منين بنضيف قبله:

1) Routes
↳ [HttpGet("{id}")] ← [FromRoute]

2) Query String عشان يستقبل array of values ← [FromQuery]

3) Header ← [FromHeader] ← عشان يستقبل حاجه جاية في الـ Header زي current language أو user id أو user location

4) Body (json) ← [FromBody]

5) Body (Key/Value Pair) ← [FromForm]

الـ Body بيكون more Secure عن Query عشان في Query ممكن تشوف الـ Parameter في الـ Url (بالمقابل الـ Body والـ Auth بالـ Header)

في الـ endPoint اللي اسمها Add كانت بتستقبل Parameter من نوع Poll واسمه Poll

Public IActionResult Add (Poll poll) { }

لو جينا نروح على الـ Swagger هناك إنه هيعرف إن المستقبل الـ Values هيكون JSon طب هو عرف حاجة اسمها Body { description, title, id } منين؟

attribute [Api Controller]

لو شيلنا الـ attribute دي الـ Swagger هيستقبل الـ values في الـ Body على شكل JSon

Binding من الـ Query String جوا

Ex

Public IActionResult Update ([FromRoute] int id, [FromBody] Poll request)
{ }

لو عايز تقوله Complex value عادي.

Query String;

Ex

عايز تقول للـ header اسمك

[HttpGet ("test")]
Public IActionResult Test ([FromHeader (Name = "x-Language")] String lang)
{
  return Ok (lang)
}

اسم الـ header

لو عايز تطبيقه لاستخدام الـ header في أحيانا, اللغة مثلاً

الأفضل من ناحية الـ Performance ←

Manual Mapping Using Extension Methods;

DTOs نستخدم الأحسن ← Domain Model على شكل endPoint الـ

المفروض هنا خلق العالم الـ DTOs فهنعمل Contracts folder ← [CreatePollRequest, PollResponse] مثلاً

DTOs

Public class CreatePollRequest
{
  Public String Title { get; set; } = String.Empty;
  Public String Description { get; set; } = String.Empty;
}

PollResponse

Public class PollResponse
{
  Public int Id { get; set; }
  Public String Title { get; set; } = String.Empty;
  Public String Description { get; set; } = String.Empty;
}

جوا Folder جديد تسميه Mapping وظيفته ContractMapping

```
Public static class ContractMapping
{
  Public Static  [النوع اللي هحول ليه]  Name (This [DomainModel] Name)
  { }

  Public Static  PollResponse MapToPollResponse (this Poll poll)
  {
    return new()
          {
                    // Assign attributes.
            Id = Poll.Id,
            Title = Poll.Title,
            Description = Poll.Description
          };
  }

  Public static  Poll MapToPoll (this CreatePollRequest request)
  {
    return new()
          {
            Title= request.Title;
            Description=request.Description;
          };
  }
```

هنروح نعدل في الـ controller بتاع

```
[HttpPost(" ")]
Public IActionResult Add([FromBody] CreatePollRequest request)
{
  var newPoll = _pollService.Add(request.MapToPoll());
  return CreatedAtAction(nameof(Get), new { id = newPoll.Id }, newPoll);
}
```

IEnumerable<Polls> Polls ← احنا بنرجع عرض من الـ GetAll في الـ

mapping في الـ ←

```
Public static IEnumerable<PollResponse> MapToPollResponse (this
                                  IEnumerable<Poll> Polls)
{
  return Polls.Select(MapToPollResponse);
}
```

في الـ Controller

عادي GetAll

```
[HttpGet("")]
Public IActionResult GetAll()
{
    var Polls = _PollService.GetAll();
    return Ok(Polls.MapToPollResponse());
}
```

Manual Mapping (using implicit Conversion):

هنعمل الـ Mapping عن طريق implicit Conversion في كلاس

```
Public class Poll
{
    Public int Id {get; Set;}
    Public String Title {get; Set;} = String.Empty;
    Public String Description {get; Set;} = String.Empty;

    Public Static implicit Operator PollResponse (Poll Poll)
    {
        return new()
        {
            Id = Poll.Id,
            Title = Poll.Title,
            Description = Poll.Description
        };
    }
```

نفس الكلاس والشيء اللي هحول ليه

✓ Create PollResponse نفس الكلام في الـ

Manual Mapping (using Explicit Conversion):

- implicit مكان الـ explicit الفرق عندي ان هكتب

Mapping With Mapster:

أفضل من ال record ال DTOs مع ال class

```
Public record CreatePollRequest (
                                  String Title ,
                                  String Description
                                 );

Public record PollResponse(
                            int Id,
                            String Title,
                            String Description
                           );
```

How to use Mapster?

1. install Package
   ↳ Mapster

⇒ domainModel.Adapt<Destination>();
Ex ↳ Generic.

```
[HttpGet ("{id}")]
Public IActionResult  Get ([FromRoute] int id)
{
  var Poll = _PollService.Get(id);
  if(Poll is null)
    return NotFound();

  var response = Poll.Adapt<PollResponse>();
  return Ok(response);
}
```

response.

configuration نروح نعرفها فوق عشان نقدر نستخدمها هنا

```
var config = new TypeAdapterConfig();
config.NewConfig<Poll , PollResponse>().Map(dest ⇒ dest.Notes,
        Source ↵   ↳ Destination          src ⇒ src.Description);

var response = Poll.Adapt<PollResponse>(config);
```

another way to use Mapping:

controller ال IMapper ال inject ①
Mapster.DependencyInjection ← install Package ②
builder.Services.AddMapster() ← Program.cs ال ③

```
var response = _mapper.Map<PollResponse>(poll);
                                    ↳ Source
```

Mapster Global Configurations:

① MappingConfigurations ← Mapping جوا Folder

```
Public class MappingConfigurations : IRegister
{
    Public void Register (TypeAdapterConfig config)
    {
        Config.NewConfig<Poll, PollResponse>()
            .Map(dest => dest.Notes,
                 Src => Src.Description);
    }
}
```

② في Program ال Register نعمل Mapper ونضيف الكونفيج بتاعنا

```
var mappingConfig = TypeAdapterConfig.GlobalSettings;
mappingConfig.Scan(Assembly.GetExecutingAssembly());

builder.Services.AddSingleton<IMapper>(new Mapper(mappingConfig));
```

لو عندي في Student domain Model حاجات زي: FirstName, MiddleName, LastName

وعايز في StudentResponse أعمل FullName إزاي؟ بنكتب في Configuration ال

```
Config.NewConfig<Student, StudentResponse>()
    .Map(dest => dest.FullName,
         Src => $"{Src.FirstName} {Src.MiddleName}
                  {Src.LastName}");
```

Age في StudentResponse هتجيب DateOfBirth في Student

```
    .Map(dest => dest.Age,
         Src => Src.DateTime.Now.Year - Src.DateOfBirth!.Value.Year,
         SrcCond => SrcCond.DateOfBirth.HasValue);
```

DateOfBirth ممكن تبقى Null عشان كده لازم نحط condition

configuration على Conditions خاصية لو عاوزة تضيف ← الـ Mapster عنده
اللي عندي (زي بتاعت Age كدا).

لو عندي attribute مش عاوز أعملها Mapping ← ولتكن الـ DepartmentName مثلاً (بتتعمل على نفس اعتبار إني خايف)

.Ignore (dest ⇒ dest.DepartmentName);

↩ عكسه مستعملهاش كدا

[AdaptIgnore] ← لو عايز attribute أعملها (data Annotation) وكتبتها كدا
public String DepartmentName {get; set;} = String.Empty;

⇐ لو عاوز أعكس المابينج بين الـ Student والـ StudentResponse

config.NewConfig<Student, StudentResponse>().TwoWays();

Validation Using Data Annotation:

Ex

بتحدد الحاجة اللي أنا عاوزة اليوزر يدخلها صح

→ [AllowedValues("New", "Old", ErrorMessage = "only 'new' and 'old' are allowed values")]
String title;

→ [Compare] ⇒ Password بتاعت Confirm بتقارنها مثلاً مع

→ [CreditCard] ⇒ عاوز الرقم اللي داخله يكون رقم فيزا مثلاً

→ [DeniedValues] ⇒ عكس الـ AllowedValues

→ [EmailAddress]

→ [FileExtensions]

Custom Validation Attribute:

class جديد وتضيف فيه ValidationAttributes

MinAge Attribute

DateOfBirth اسمها attribute فيه Student داخل class ← هيكون عندي
واضنا عاوزين نعمل validation إننا محتش هيتعين أقل من 18 سنة:

```
[AttributeUsage(AttributeTargets.Field | AttributeTargets.Property)]
Public class MinAgeAttribute : validationAttribute
{
   Public override bool IsValid(object? value)
   {
      if(value is not null)
        { var Date = (DateTime) value;
          if( DateTime.Today < Date.AddYears(18))
              return false;
        }
      return true;
   }
}
```

← بعدين فوق هنحدد الـ attribute اللي هيتعمل عليه validate

← عشان نقدر نستخدمه

AttributeTargets.All = هنستخدمه مع أي Attribute عادي

← هنروح فوق الـ DateOfBirth في الـ Student ونكتب [MinAge]

← هنعمل Validator للكود اللي فوق عشان نقدر نستخدمه مع أي Age ، ويطلع الـ errorMessage من الـ Prop. implement نفسه عشان كل سنة أكتبه فوق ↓

```
[AttributeUsage(AttributeTargets.Field | AttributeTargets.Property)]
Public class MinAgeAttribute (int minAge) : validationAttribute
 {
   Private readonly int _minAge = minAge;

   Protected override validationResult? IsValid(object? value,
                          validationContext validationContext)
   {
     if(value is not null)
       {
          var Date = (DateTime)value;
          if( DateTime.Today < Date.AddYears(_minAge).
             return new validationResult($" Invalid {validationContext
                .DisplayName} Age should be {_minAge} years old");
       }
       return validationResult.Success;
   }
}
```

← الشكل ده عشان نقدر نعرف الـ message مع Property

← وهنا بنعمل الـ annotation عشان نكتبها كدا

```
[MinAge(18), Display(Name = "Birth of Date")]
```

Install EntityFramwork

Packages:
- Microsoft.EntityFramworkCore
- Microsoft.EntityFramworkCore.Tools. ← Migrations
- Microsoft.EntityFramworkCore.SqlServer

Add The DbContext:

Persistance ← Folder نضيف

ApplicationDbContext ← class نضيفه

Primary Ctor

```
Public class ApplicationDbContext (DbContextOptions<ApplicationDbContext> options)
                : DbContext (options)
{
  Public DbSet<Poll> Polls {get; Set;}
}
```

هنروح نضيف الـ Connection String في الـ AppSettings

```
"connectionStrings": {
  "DefaultConnection": "Server=      ; Database=      ; Trusted_Connection
      = True; Encrypt= False"
          }
```

هنروح على DependencyInjection ونعمل Register للـ Connection String

الأول نضيف في الـ Parameters ← IConfiguration Configuration

```
var connectionString = Configuration.GetConnectionString("DefaultConnection")
?? throw new InvalidOperationException("connection string 'DefaultConnection'
      not found.");

Services.AddDbContext<ApplicationDbContext>(options => options
    .UseSqlServer(connectionString));
```

في الـ Program.cs نكتب

```
builder.Services.AddDependencies(builder.Configuration);
```

Add Polls Configurations:

نضيف جوا Persistance ← Folder Configurations عشان نعمل

هنكون فيه Entity JS Class محددة

Another Example:

```
Public class StudentValidator : AbstractValidator<Student>
{
    Public StudentValidator()
    {
        RuleFor(x => x.DateOfBirth)
            .Must(x => DateTime.Today > x.Value.AddYears(18))
            .WithMessage("{PropertyName} is invalid, age should be 18 atleast");
    }
}
```

BeMoreThan 18 years.

PlaceHolder

Property اسم الحقل

PropertyValue قيمة الحقل

Null لتجنب

.When(x => x.DateOfBirth.HasValue)

عكس الشرط السابق

```
Private bool BeMoreThan18Years(DateTime? dateOfBirth)
{
    return DateTime.Today > dateOfBirth!.Value.AddYears(18);
}
```

Move Dependencies:

Static لجعلها DependencyInjection class لنقل

Program وتعديل الاستدعاء في Services

Validation Using FluentValidation :

1. هنا Package اللي install
   FluentValidation.DependencyInjectionExtensions
2. هنضيف Folder اسمه Validations جوا ContractsFolder
3. هنضيف CreatePollRequestValidator
4. class اللي هعمل عليه ← class
   وبنرث ← : AbstractValidator<CreatePollRequest>

```
public class CreatePollRequestValidator : AbstractValidator<CreatePollRequest>
{
  public CreatePollRequestValidator()
  {
    RuleFor(x => x.Title)
        .NotEmpty();
  }
}
```

الوقتي هنروح نعمل IValidator اللي inject جوا الـ EndPoint

```
[HttpPost(" ")]
Public IActionResult Add ([FromBody] CreatePollRequest request,
   [FromServices] IValidator<CreatePollRequest> validator)
{
  var validationResult = validator.Validate(request);
  if(!validationResult.IsValid)
   {
     var modelState = new ModelStateDictionary();
     validationResult.Errors.ForEach(x => modelstate.AddModelError
                                   (x.PropertyName, x.ErrorMessage));
     return ValidationProblem(modelState);
   }
  var newPoll = _pollService.Add(request.Adapt<Poll>());
  return CreatedAtAction(nameof(Get), new {id = newPoll.Id}, newPoll);
}
```

بس كدا هنروح نعمل register جوا الـ Program

بس المشكلة حتى احنا حاطين الـ validation في الـ Controller، فنروح نشيل الـ validation من الـ Controller ونروح على الـ Program ونحطها هناك

1. هنعمل install لـ Package
   SharpGrip.FluentValidation.AutoValidation.Mvc
   FluentValidation.AspNetCore
   وهنكتب ده

```
builder.Services.
    .AddFluentValidationAutoValidation()
    .AddValidatorsFromAssembly(Assembly.GetExecutingAssembly());
```

وبالتالي بقى كل الـ validation هيروح في الـ register (اللي هيعمل كل الـ validation)

```
Public class PollConfiguration : IEntityTypeConfiguration<Poll>
{
    Public void Configure(EntityTypeBuilder<Poll> builder)
    {
        builder.HasIndex(x => x.Title).IsUnique();

        builder.Property(x => x.Title).HasMaxLength(100);

        builder.Property(x => x.Summary).HasMaxLength(1500);
    }
}
```

هنروح على ApplicationDbContext في الـ OnModelCreating نضيف الـ Configuration

```
Protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.ApplyConfigurationFromAssembly(Assembly.GetExecutingAssembly());

    base.OnModelCreating(modelBuilder);
}
```

هنشغل الـ Migration

✓ add-migration InitialCreate -o Presistence/Migrations.

✓ update-database

Start working with DataBase:

هنغير شوية حاجات

① عاوزين نشتغل Asynchronus (هنشتغل على الـ GetAll وغير الاسم)

هنروح على الـ IPollService وغير الـ GetAll

```
Task<IEnumerable<Poll>> GetAllAsync();
```

هنروح على الـ PollService نعمل inject لـ ApplicationDbContext نشتغل عليها

```
Public class PollService(ApplicationDbContext context) : IPollService
{
    Private readonly ApplicationDbContext _context = context;

    Public Async Task<IEnumerable<Poll>> GetAllAsync() =>
        await _context.Polls.AsNoTracking().ToListAsync();
}
```

هنروح على الـ Controller

```
[HttpGet("")]
Public async Task<IActionResult> GetAll()
{
  var Polls = await _PollService.GetAllAsync();
  var response = Poll.Adapt<IEnumerable<PollResponse>>();

  return Ok(response);
}
```

CancellationToken:

⇐ كاميم نقلها مع أي EndPoint بتتعامل مع الـ DB علشان طا اليوزر يجرب يعمل أي action ويرفع بكتير في نص التنفيذ ميسمحش في الـ DB

```
[HttpPost("")]
Public async Task<IActionResult> Add([FromBody] CreatePollRequest request,
        CancellationToken cancellationToken)
{

  باقي الكود هنا زي ماهو

}
```

وبرفع نبعتها في الـ IPollService في الـ EndPoints

```
Task<Poll> AddAsync(Poll poll, CancellationToken cancellationToken = default);
```

في الـ implement في كلاس الـ PollService

```
Public async Task<Poll> AddAsync(Poll poll, CancellationToken cancellationToken
                                 = default)
{
  await _context.AddAsync(Poll, CancellationToken);
  await _context.SaveChangesAsync(cancellationToken);

  return Poll;
}
```

هنضيفها في أي كلاس بيتعامل مع الـ DataBase

Sync vs. Async

↳ Sync ⇒ Code work Sequential

- السطر الأول ← لثاني ← لتالت وهكذا ، بياخد وقت أطول في التنفيذ
- لو عندي خطوة طويلة مثلاً بتستنى حاجة، الخطوة اللي بعدها لازم تستنى لحد ما الخطوة دي تخلص

← ال UI بتاع ال app هيحصله Freeze في الوقت دا.

async ↴ لو فيه خطوة بتاخد وقت مش هتعمل للكود Block في باقي

Ex

Sync

```
Public Static class Kitchen
{
    Public Static void MakeTea()
    {
        var Water = BoileWater();
        Console.WriteLine("Take the Cup out");
        Console.WriteLine("Put tea in Cup");
        Console.WriteLine("Put Sugar in Cup");
        Console.WriteLine($"Pour {water} in the Cup");
    }

    Public static String BoileWater()
    {
        Console.WriteLine("Start the Kettle");
        Console.WriteLine("waiting for the Kettle");
        Task.Delay(5000).GetAwaiter.GetResult();
        Console.WriteLine("Kettle finished boiling");

        return "water";
    }
}
```

→ in main()

```
Kitchen.MakeTea();
```

OutPut

Start the Kettle
waiting for the kettle ← هستنى 5 ثوانى
Kettle finished boiling
Take the cup out
Put tea in cup
Put Sugar in cup
Pour water in the cup.

Ex async

Public
{
Pub
{
}
Priv
{
C
}
}

async

```
Public class Kitchen
{
    Public static ~~Stat~~ async Task MakeTeaAsync()
    {
        var boilingWater = BoileWaterAsync()

        console.writeline ("take the cup out");
        "  "          ("Put tea in cup");
        "  "          ("Put Sugar in cup")

        var water = await boiling water;

        Console.writeline ($"Pour {water} in cup");
    }

    Private Static async Task<String> BoilewateAsync()
    {
        Console.writeline ("start the kattle");
            "  "      ("waiting for the kattle");

        await Task.Delay (5000);

        Console.writeline ("Kattle finshed boiling");

        return "water";
    }
}
```

↝ Main()

```
Kitchen.MakeTeaAsync();
```

OutPut:

Start The kattle
waiting for the kattle
Take the cup out
Put tea in cup
Put Sugar in cup
kattle finished boiling
Pour water in cup.