Team Dynamix Web Api SDK

A fluent, opinionated C# SDK for interacting with the TeamDynamix Web API.

✨ Features

- ✅ Fluent-style request builders
- ✅ Strongly typed models and builders
- ✅ Supports both basic and complex ticket operations
- ✅ Extensible and customizable
- ✅ Designed for testability and clean layering

📦 Installation

Install via NuGet:

```
dotnet add package TdxClient
```

Or reference the project directly in your solution:

```
<ProjectReference Include="..\TdxClient\TdxClient.csproj" />
```

🔧 Setup

🚀 Quick Start

Option 1: Minimal Setup (Not Recommended for ASP.NET Core)

```
var client = new TdxClient(
    tenant: "mytenant",
    webServicesBeId: "abc123",
    webServicesKey: "super-secret-key"
);

await client.AuthenticateAsync();

var ticket = await client.Tickets("244")["123456"].GetAsync();
```

Option 2: Recommended Setup with IHttpClientFactory

In Program.cs or Startup.cs:

```
builder.Services.AddHttpClient<TdxClient>((provider, httpClient) =>
{
    // Optional: Add custom handlers, policies, etc.
})
.ConfigurePrimaryHttpMessageHandler(() => new HttpClientHandler
{
    AutomaticDecompression = DecompressionMethods.GZip
| DecompressionMethods.Deflate
});
```

Then register your config:

```
builder.Services.AddSingleton(new TdxClientOptions
{
    EnableThrottleCountdownLogging = true,
    OnThrottleWait = remaining => Console.WriteLine($"Waiting
{remaining.TotalSeconds} seconds due to rate limit..."),
    OnThrottleContinue = () => Console.WriteLine("Continuing after wait.")
});

builder.Services.AddTransient<TdxClient>(provider =>
{
    var httpClient = provider.GetRequiredService<IHttpClientFactory>
().CreateClient(nameof(TdxClient));
    var options = provider.GetRequiredService<TdxClientOptions>();

    return new TdxClient(
        httpClient,
        tenant: "mytenant",
        webServicesBeId: "abc123",
        webServicesKey: "super-secret-key",
        options
    );
});
```

Then inject it in your service:

```
public class MyService
{
    private readonly TdxClient _tdx;

    public MyService(TdxClient tdx)
```

```
    {
        _tdx = tdx;
    }

    public async Task RunAsync()
    {
        await _tdx.AuthenticateAsync();

        var newTicket = new Ticket
        {
            Title = "My New Ticket",
            Description = "Created via fluent API",
            RequestorUid = Guid.Parse("..."),
            StatusID = 1234,
            TypeID = 5678,
            AccountID = 9876,
            PriorityID = 1111
        };

        var created = await _tdx.Tickets("244").Create(newTicket);
    }
}
```

Inject Your Own HttpClient With Custom Delegating Handlers

```
// Custom logging handler example
public class LoggingHandler : DelegatingHandler
{
    protected override async Task<HttpResponseMessage> SendAsync(HttpRequestMessage
request, CancellationToken cancellationToken)
    {
        Console.WriteLine($"Request: {request.Method} {request.RequestUri}");
        var response = await base.SendAsync(request, cancellationToken);
        Console.WriteLine($"Response: {response.StatusCode}");
        return response;
    }
}

// Create a list of custom handlers
var handlers = new List<DelegatingHandler> { new LoggingHandler() };

// Create HttpClient with custom handlers using the factory
var httpClient = TdxClientFactory.CreateHttpClient(handlers, tenant);
```

```
// Inject your HttpClient into TdxClient
var client = new TdxClient(httpClient, tenant, beId, webServicesKey);

// Authenticate and use
await client.AuthenticateAsync();

var tickets = await client.Tickets("244")["12345"].GetAsync();
```

Advanced: Adding Multiple Handlers (Logging + Retry + Telemetry)

```
var handlers = new List<DelegatingHandler>
{
    new LoggingHandler(),
    new RetryHandler(),  // Your custom retry handler if you have one
    new TelemetryHandler() // Your telemetry handler for metrics
};

var httpClient = TdxClientFactory.CreateHttpClient(handlers, tenant);

var client = new TdxClient(httpClient, tenant, beId, webServicesKey);

await client.AuthenticateAsync();

// Now all requests go through the pipeline of your handlers
var tickets = await client.Tickets("244")["12345"].GetAsync();
```

⚠️ Not Recommended: Manual `HttpClient` Instantiation

You will need to handle your own retry policies.

```
// Use only for quick tests or console apps
var httpClient = new HttpClient
{
    BaseAddress = new Uri("https://your-org.teamdynamix.com/TDWebApi/api/")
};

var tdxClient = new TdxClient(httpClient, "your-webservices-key", "your-beid");
```

This bypasses DI scopes, Polly retry policies, DNS refresh handling, and other best practices.

🚀 Usage Exmaple

🔍 Search for a Person

```
var results = await tdxClient.Users.Search
    .WithEmail("jdoe@yourorg.edu")
    .ExecuteAsync();
```

📄 Get a Ticket by ID

```
var ticket = await tdxClient.Tickets("244")["123456"].GetAsync();
```

📝 Create a Ticket

```
var ticketRequest = new TicketRequest
{
    Title = "New Hire Onboarding",
    RequestorUid = "abc-123",
    Type = TicketType.OnboardingOrJobUpdate,
    Description = "Provision access and equipment",
    ResponsibleGroupId = "4000"
};

var ticket = await tdxClient.Tickets("244").Create(ticketRequest.MapToTicket());
```

📝 Create ticket cont.

```
var ticket = await tdxClient.Tickets("244").Create().WithTitle("...")...
// if you attempt to send without all required fields it will error our and
notify you
```

✅ Resolve a Ticket

```
await tdxClient.Tickets("244")["123456"]
    .SetStatus(TicketStatus.Resolved)
    .WithComment("Issue resolved by help desk.")
    .UpdateAsync();
```

⚙️ TdxClientOptions

Customize retry/throttling behavior:

```
var options = new TdxClientOptions
{
    MaxRetries = 3,
    EnableThrottleCountdownLogging = true,
    OnThrottleWait = remaining => Console.WriteLine($"Waiting
{remaining.TotalSeconds}..."),
    OnThrottleContinue = () => Console.WriteLine("Continuing.")
};
```

🧱 Structure

```
TdxClient
├── People
│   └── Search
├── Tickets
│   ├── Create
│   └── [ticketId]
│       ├── Get
│       ├── SetStatus
│       └── AddFeedEntry
```

📦 Extending

You can add more fluent builders by extending BaseRequestBuilder:

```
public class AssetsRequestBuilder : BaseRequestBuilder
{
    public AssetsRequestBuilder(TdxBaseClient client)
        : base("assets", client) { }

    public Task<Asset> GetAssetAsync(string id) =>
        SendAsync<Asset>(CreateRequest(HttpMethod.Get, id));
}
```

🧪 Testing

You can mock the `TdxBaseClient` and `HttpClient` layers for unit testing. Also would recommend implementing your own httpclient purely to repoint to

---

🔐 Authentication

Supports Web Services Key-based authentication. If your API uses OAuth, you'll need to extend `TdxBaseClient` accordingly.

📄 License

MIT

🙋 Support / Contribution

Open an issue or submit a pull request — contributions welcome!

Namespace TeamDynamix.Api

Classes

BaseRequestBuilder

Abstract base class for building and sending HTTP requests to the TeamDynamix API. Provides helper methods to create requests with optional JSON bodies and send them using a TdxBaseClient instance.

TdxBaseClient

Base abstract client for interacting with the TeamDynamix API. Provides core HTTP communication features such as:

- Configurable HttpClient with base address setup.
- Retry policy handling for HTTP 429 (Too Many Requests) throttling responses.
- Token-based authorization header management.
- Generic request sending methods with built-in retry logic and deserialization.

TdxClient

Client for interacting with the TeamDynamix API, extending TdxBaseClient. Provides strongly typed request builders for People, Tickets, and Accounts endpoints, and handles authentication using provided credentials.

TdxClientFactory

Factory for creating HttpClient instances with advanced control over the message handler pipeline.