UltraNetDemo - Project Documentation

Overview

UltraNetDemo is a clean, modular, and extensible .NET API framework designed for enterprise-grade applications. It includes features such as:

- JWT Authentication & Refresh Token
- Rate Limiting (Token Bucket Algorithm)
- OTP Delivery via Strategy Pattern (SMS / Email / Custom)
- Clean Code Structure with DI, Interface-based Abstractions
- Logging via Serilog (using Composite Pattern)
- Caching (Hybrid: In-Memory & Redis)

How to Run

1. Restore packages:
   - Run `dotnet restore` in the solution root.

2. Update appsettings.json (if needed):
   - Ensure the `SecretKey` is at least 32 characters for JWT to work properly.
   - Adjust token expiry, issuer, and audience in `TokenOptions`.

3. Run the API:
   - Use Visual Studio or run `dotnet run` in the API project directory.
   - Swagger UI will be available at `https://localhost:{port}/swagger`

JWT Setup

- JWT is configured in `Program.cs` using `AddAuthentication().AddJwtBearer()`.
- Token is generated using `TokenGenerator` with claims.
- You can inject and use the `ITokenGenerator` interface in any service.

Configuration (TokenOptions):

```
{
  "SecretKey": "Your_32char_minimum_secret_key",
  "Issuer": "UltraNet.Issuer",
  "Audience": "UltraNet.Audience",
  "AccessTokenExpiryMinutes": 30,
  "RefreshTokenExpiryDays": 7
}
```

OTP Strategy

- Uses Strategy Pattern to support multiple OTP channels.
- Two strategies implemented: `SmsOtpStrategy`, `EmailOtpStrategy`
- Interface: `IOTPStrategy`
- Based on `OtpOptions`, multiple strategies can be triggered at once.

Example:

```
{
  "CodeLength": 6,
  "Strategies": ["sms", "email"]
}
```

You can easily add your own strategies by implementing `IOTPStrategy`.

Rate Limiting

- Uses Token Bucket Algorithm.
- Interface: `IRateLimiter`, implemented via `InMemoryTokenBucket`.
- Rate-limiting can be easily applied to any action like OTP or login.

Logging

- Logs written to console and to rolling files via Serilog.
- Path: `Logs/log-.txt`
- Composite Pattern is used to support multiple simultaneous log outputs.

Caching (In-Memory & Redis Hybrid)

- Provides flexible caching with two built-in providers: `MemoryCacheProvider` and `RedisCacheProvider`.
- `ICacheProviderFactory` allows runtime selection of provider.

Example:

```
var cache = _cacheFactory.GetProvider("memory"); // or "redis"
await cache.SetAsync("key", yourObject);
```

```
  "Cache": {
    "Type": "memory"
  },
```

Swagger

- Swagger UI enabled with full endpoint testing.

Pro Tips for Enhancement

- Add distributed Redis cache support for rate-limiter
- Use IP-based throttling for public APIs
- Add retry mechanism for OTP failure
- Extend token storage to support blacklist
- Enable Health Checks and Prometheus metrics