Asp.TotpKit – TOTP (Time-based One-Time Password) Validation Library

Table of contents

1. Purpose

Asp.TotpKit is a Google Authenticator-compatible time-based one-time password (TOTP) verification library **designed for ASP.NET developers** . It allows you to integrate a secure two-factor authentication (2FA) system in your project without writing it from scratch, ready for the **production environment**.

This library collects the following repetitive operations in one place:

- QR code generation,
- Secret key production,
- Code verification (at 30-second intervals),
- Cookie or user-based temporary session management,
- ASP.NET Compatibility with Identity or custom user models.

2. Features

- TOTP generation in accordance with RFC 6238 standard
- Google Authenticator and Authy support
- 30-second auto-cycle encodings
- Tolerated validation (±30 seconds difference)
- Optional cookie-based temporary session
- AppUser (Identity) or custom model support
- DI compatible with IOptions<T> configuration
- QRCoder & .NET dependency only
- Full logging and error management (ILogger)
- Optional linking of user information such as email and ID

3. Architectural Structure

Asp.TotpKit consists of 3 main components:

1. **Interface** → ITotpService<TUser> and ITotpUserInfo
    - Contains all method signatures (Generate, Validate, Verify, Disable...)
2. **Model** → TotpSetupResponse, TotpVerifyResult, TotpOptions, TotpInstructions
    - Data transfer and configuration models
3. **Service** → TotpService<TUser>
    - Real implementation. It manages HMAC-based TOTP generation, QR code generation, cookie transactions, and user updates.

4. Installation

Asp.TotpKit is designed to be easily integrated into ASP.NET Core projects. The setup process is straightforward, whether or not the Identity structure is used. The following steps cover use cases with both AppUser (IdentityUser)-based and **custom user model** .

4.1. Add NuGet Package

```
dotnet add package Asp.TotpKit
```

4.2. Service Registration via Program.cs

Use the AddTotpKit<TUser>() method to save **the library to the** Dependency Injection (DI) container.

```
builder.Services.AddTotpKit<AppUser>(options =>
{
    options.AppName = "MyProject";
    options.Issuer = "MyProject.Auth";
    options.GetUserEmail = u => u.Email!;
    options.GetUserId = u => u.Id!;
});
```

Program.cs Integration (Manual Scoped Registration):

```
builder.Services.AddScoped<ITotpService<AppUser>, TotpService<AppUser>>(sp =>
{
    var service = new TotpService<AppUser>(
        sp.GetRequiredService<UserManager<AppUser>>(),
        sp.GetRequiredService<IConfiguration>(),
        sp.GetRequiredService<ILogger<TotpService<AppUser>>>(),
        sp.GetRequiredService<IOptions<TotpOptions<AppUser>>>()
    );

    // 🔷 Local ortam (Swagger, Postman) için varsayılan cookie ayarları
    service.httpOnlyCookie = false;
    service.secureCookie = false;
    service.sameSiteMode = SameSiteMode.Lax;

    // 🔷 Production ortamı için önerilen güvenli ayarlar:
    // service.httpOnlyCookie = true;
    // service.secureCookie = true;
    // service.sameSiteMode = SameSiteMode.Lax;

    return service;
});
```

If you are using a custom model:

```
builder.Services.AddTotpKit<MyUser>(options =>
{
    options.AppName = "MyProject";
    options.Issuer = "MyCompany.Security";
    options.GetUserEmail = u => u.EmailAddress!;
    options.GetUserId = u => u.Uuid.ToString();
});
```

Note: Your AppUser or MyUser class must implement the ITotpUserInfo interface. This allows TOTP-owned fields (Secret, Enabled, VerifiedDate, etc.) to be stored.

4.3. AppUser / Custom User Model Definition

Asp.TotpKit uses an interface to record user-specific TOTP information (secret, verification time, etc.):
ITotpUserInfo.

Every class that implements this interface supports TOTP operations. If you're using ASP.NET Identity by default, simply add the following fields to the AppUser class:

```
public class AppUser : IdentityUser, ITotpUserInfo
{
    public string? ToptSecret { get; set; }                // Base32 Secret Key
    public bool IsToptEnabled { get; set; }                // TOTP aktif mi
    public DateTime? ToptVerifiedDate { get; set; }        // Son doğrulama tarihi
    public bool IsTotpResetRequested { get; set; }         // Reset isteği
    public DateTime? TotpResetRequestedAt { get; set; }    // Reset zamanı
}
```

If you don't use Identity and manage your own user table, you can integrate the same interface into your own model:

```
public class MyUser : ITotpUserInfo
{
    public Guid Uuid { get; set; }
    public string EmailAddress { get; set; } = "";
    public string? ToptSecret { get; set; }
    public bool IsToptEnabled { get; set; }
    public DateTime? ToptVerifiedDate { get; set; }
    public bool IsTotpResetRequested { get; set; }
    public DateTime? TotpResetRequestedAt { get; set; }
}
```

In this way, the library, whether it uses Identity or your own ORM model, recognizes the user object and the TotpService automatically uses that structure.

4.4. IConfiguration / appsettings.json Usage

Asp.TotpKit can also retrieve configurations over appsettings.json instead of Program.cs because it supports IOptions<T> model binding.

This makes it much easier to change environment-based settings when your project is in production.

Example configuration:

```
"AppSettings": {
  "AppName": "MyEnterpriseApp",
  "TotpIssuer": "MyEnterprise.Security"
}
```

If these fields are not specified in the Program.cs:

```
options.AppName = ...
options.Issuer = ...
```

library automatically reads these values from within IConfiguration.

This is a great convenience for teams that want *to "share common configurations on many projects"*. For example, you can use separate appsettings files for staging, production, and local environments.

4.5. Cookie Behavior (Live and Local Environment Difference)

Asp.TotpKit allows you to use temporary cookies **in the user authentication process after login** . This is especially used in two-factor authentication (login → TOTP code) flows.

Default behavior:

```
_totpService.httpOnlyCookie = false;
_totpService.secureCookie = false;
_totpService.sameSiteMode = SameSiteMode.Lax;
```

These settings are appropriate for **the local development environment (Swagger/Postman).** Because HTTPS enforcement and HttpOnly restriction are disabled.

Production environment:

On live servers, you need to configure it as follows:

```
_totpService.httpOnlyCookie = true;
_totpService.secureCookie = true;
_totpService.sameSiteMode = SameSiteMode.Lax;
```

Only the server can access the HttpOnly → cookie, reducing the risk of XSS.
A Secure → cookie is only sent on an HTTPS connection.
SameSite provides → CSRF protection. "Lax" is the recommended mode.

If the frontend and backend are running on different domains (CORS scenario), you can set SameSiteMode.None. In this case, Secure = true is mandatory (browser restriction).

4.7. Setup Flow

Step	Explanation
1	Install Asp.TotpKit package (dotnet add package)
2	Apply ITotpUserInfo to AppUser or MyUser model
3	Add AddTotpKit<TUser>() service to Program.cs
4	(Optional) Type AppName & Issuer into appsettings.json
5	(Optional) Configure cookie settings
6	Inject and use ITotpService<TUser> into the constructor

5. Method Reference

Method	Explanation	Return Type
GenerateSecretKeyAsync(user)	Generates and saves new Base32 TOTP secret keys for the user.	Task<string>
GenerateQrCodeUriAsync(user, secretKey)	Generates QR code URI compatible with Google Authenticator.	Task<string>

GenerateQrCodeImageAsync(qrCodeUri)	The QR code generates bytes[] in PNG format.	Task<byte[]>
ValidateTotpCodeAsync(user, code)	It checks if the 6-digit code entered is valid.	Task<bool>
ValidateLoginCodeAsync(user, code)	It verifies the TOTP code during login.	Task<bool>
GenerateSetupAsync(user)	It returns the user's private secret and QR information.	Task<TotpSetupResponse>
VerifyCodeAsync(user, code)	It activates TOTP after the first installation.	Task<TotpVerifyResult>
EnableTotpAsync(user, code)	Enables TOTP for the verified user.	Task<bool>
DisableTotpAsync(user)	Closes TOTP, clears secret.	Task<bool>
HasTotpSetup(user)	The user's TOTP activity is checked.	Bool
SetTotpTempCookie(response, userId, expiryMinutes)	It creates a temporary cookie during the login process.	Void
GetTotpTempCookie(request)	Returns the user ID from the cookie.	String
ClearTotpTempCookie(response)	Deletes the cookie.	Void

6. Conclusion and Closing

I developed this library because I was tired of reinstalling the same TOTP validation structure on every project. My goal **was to create a 2FA system that** was simple enough for everyone to easily understand**, but** secure at a professional level.

Asp.TotpKit is not only a "code verification service", but also a non-repetitive, clean and extensible security architecture.

Use it in your own projects, test it, examine it. If you see something missing, inaccurate, or improvable, always send feedback. For me, this project is not just a library — **it's a culture of reusable security.**