Complete MFA Implementation Guide for Angular + ASP.NET Core

This guide provides a step-by-step implementation of Multi-Factor Authentication (MFA) using Time-based One-Time Passwords (TOTP) in an Angular frontend with ASP.NET Core backend.

Table of Contents

Backend Implementation

1. Install Required NuGet Packages

```
<PackageReference Include="OtpNet" Version="1.3.0" />
<PackageReference Include="QRCoder" Version="1.4.3" />
```

2. Update User Model

Add MFA properties to your User model:

```
public class User
{
    public int Id { get; set; }
    public string Username { get; set; } = string.Empty;
    public string Email { get; set; } = string.Empty;
    public string FirstName { get; set; } = string.Empty;
    public string LastName { get; set; } = string.Empty;
    public string PasswordHash { get; set; } = string.Empty;
    public string? Token { get; set; }

    // MFA Properties
    public bool IsMfaEnabled { get; set; } = false;
    public string? MfaSecret { get; set; }
    public DateTime? MfaSetupDate { get; set; }
    public List<string> MfaBackupCodes { get; set; } = new();
}
```

3. Create MFA Endpoints

Create a comprehensive MFA endpoints class:

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using OtpNet;
using QRCoder;
using System.Drawing;
using System.Drawing.Imaging;
using System.Security.Claims;
using System.Text;

public static class MfaEndpoints
{
    public static void MapMfaEndpoints(this IEndpointRouteBuilder app)
    {
        var group = app.MapGroup("/mfa").RequireAuthorization();

        group.MapPost("/setup", SetupMfa);
        group.MapPost("/enable", EnableMfa);
        group.MapPost("/disable", DisableMfa);
        group.MapPost("/verify", VerifyMfa);
        group.MapGet("/status", GetMfaStatus);
    }

    private static async Task<IResult> SetupMfa(
        ClaimsPrincipal user,
        IUserService userService)
    {
        try
        {
            var userId = GetUserId(user);
            var currentUser = await userService.GetUserByIdAsync(userId);

            if (currentUser == null)
                return Results.NotFound("User not found");

            // Generate secret
            var secret =
Base32Encoding.ToString(KeyGeneration.GenerateRandomKey(20));
            var issuer = "ListKeeper";
            var userEmail = currentUser.Email;

            // Generate QR code
            var qrCodeUrl = $"otpauth://totp/{issuer}:{userEmail}?secret=
{secret}&issuer={issuer}";
            var qrGenerator = new QRCodeGenerator();
            var qrCodeData = qrGenerator.CreateQrCode(qrCodeUrl,
QRCodeGenerator.ECCLevel.Q);
            var qrCode = new QRCode(qrCodeData);
            var qrCodeImage = qrCode.GetGraphic(20);

            // Convert to base64
            using var stream = new MemoryStream();
            qrCodeImage.Save(stream, ImageFormat.Png);
            var qrCodeBase64 = Convert.ToBase64String(stream.ToArray());
```

```
            // Generate backup codes
            var backupCodes = GenerateBackupCodes();

            // Store secret temporarily (not enabled yet)
            currentUser.MfaSecret = secret;
            currentUser.MfaBackupCodes = backupCodes;
            await userService.UpdateUserAsync(currentUser);

            return Results.Ok(new
            {
                secret,
                qrCode = $"{qrCodeBase64}",
                backupCodes
            });
        }
        catch (Exception ex)
        {
            return Results.Problem($"Error setting up MFA: {ex.Message}");
        }
    }

    private static async Task<IResult> EnableMfa(
        ClaimsPrincipal user,
        EnableMfaRequest request,
        IUserService userService)
    {
        try
        {
            var userId = GetUserId(user);
            var currentUser = await userService.GetUserByIdAsync(userId);

            if (currentUser == null ||
string.IsNullOrEmpty(currentUser.MfaSecret))
                return Results.BadRequest("MFA setup not found");

            // Verify the code
            var secretBytes = Base32Encoding.ToBytes(currentUser.MfaSecret);
            var totp = new Totp(secretBytes);
            var isValid = totp.VerifyTotp(request.Code, out _,
VerificationWindow.RfcSpecifiedNetworkDelay);

            if (!isValid)
                return Results.BadRequest("Invalid verification code");

            // Enable MFA
            currentUser.IsMfaEnabled = true;
            currentUser.MfaSetupDate = DateTime.UtcNow;
            await userService.UpdateUserAsync(currentUser);

            return Results.Ok(new { message = "MFA enabled successfully" });
        }
        catch (Exception ex)
        {
```

```
            return Results.Problem($"Error enabling MFA: {ex.Message}");
        }
    }

    private static async Task<IResult> DisableMfa(
        ClaimsPrincipal user,
        DisableMfaRequest request,
        IUserService userService)
    {
        try
        {
            var userId = GetUserId(user);
            var currentUser = await userService.GetUserByIdAsync(userId);

            if (currentUser == null)
                return Results.NotFound("User not found");

            // Verify password
            if (!BCrypt.Net.BCrypt.Verify(request.Password,
currentUser.PasswordHash))
                return Results.BadRequest("Invalid password");

            // If MFA is enabled, verify the MFA code
            if (currentUser.IsMfaEnabled &&
!string.IsNullOrEmpty(currentUser.MfaSecret))
            {
                if (string.IsNullOrEmpty(request.MfaCode))
                    return Results.BadRequest("MFA code required");

                var secretBytes = Base32Encoding.ToBytes(currentUser.MfaSecret);
                var totp = new Totp(secretBytes);
                var isValidMfa = totp.VerifyTotp(request.MfaCode, out _,
VerificationWindow.RfcSpecifiedNetworkDelay);

                if (!isValidMfa)
                    return Results.BadRequest("Invalid MFA code");
            }

            // Disable MFA and clear all MFA data
            currentUser.IsMfaEnabled = false;
            currentUser.MfaSecret = null;
            currentUser.MfaSetupDate = null;
            currentUser.MfaBackupCodes.Clear();

            await userService.UpdateUserAsync(currentUser);

            return Results.Ok(new { message = "MFA disabled successfully" });
        }
        catch (Exception ex)
        {
            return Results.Problem($"Error disabling MFA: {ex.Message}");
        }
    }
```

```
    private static async Task<IResult> VerifyMfa(
        ClaimsPrincipal user,
        VerifyMfaRequest request,
        IUserService userService)
    {
        try
        {
            var userId = GetUserId(user);
            var currentUser = await userService.GetUserByIdAsync(userId);

            if (currentUser == null || !currentUser.IsMfaEnabled ||
string.IsNullOrEmpty(currentUser.MfaSecret))
                return Results.BadRequest("MFA not enabled for this user");

            // Check backup codes first
            if (currentUser.MfaBackupCodes.Contains(request.Code))
            {
                // Remove used backup code
                currentUser.MfaBackupCodes.Remove(request.Code);
                await userService.UpdateUserAsync(currentUser);
                return Results.Ok(new { message = "MFA verified with backup code"
});
            }

            // Verify TOTP code
            var secretBytes = Base32Encoding.ToBytes(currentUser.MfaSecret);
            var totp = new Totp(secretBytes);
            var isValid = totp.VerifyTotp(request.Code, out _,
VerificationWindow.RfcSpecifiedNetworkDelay);

            if (!isValid)
                return Results.BadRequest("Invalid MFA code");

            return Results.Ok(new { message = "MFA verified successfully" });
        }
        catch (Exception ex)
        {
            return Results.Problem($"Error verifying MFA: {ex.Message}");
        }
    }

    private static async Task<IResult> GetMfaStatus(
        ClaimsPrincipal user,
        IUserService userService)
    {
        try
        {
            var userId = GetUserId(user);
            var currentUser = await userService.GetUserByIdAsync(userId);

            if (currentUser == null)
                return Results.NotFound("User not found");

            return Results.Ok(new
```

```
            {
                isMfaEnabled = currentUser.IsMfaEnabled,
                mfaSetupDate = currentUser.MfaSetupDate,
                backupCodesCount = currentUser.MfaBackupCodes?.Count ?? 0
            });
        }
        catch (Exception ex)
        {
            return Results.Problem($"Error getting MFA status: {ex.Message}");
        }
    }

    private static int GetUserId(ClaimsPrincipal user)
    {
        var userIdClaim = user.FindFirst(ClaimTypes.NameIdentifier)?.Value;
        return int.Parse(userIdClaim ?? "0");
    }

    private static List<string> GenerateBackupCodes(int count = 10)
    {
        var codes = new List<string>();
        var random = new Random();

        for (int i = 0; i < count; i++)
        {
            var code = random.Next(100000, 999999).ToString();
            codes.Add(code);
        }

        return codes;
    }

    public record EnableMfaRequest(string Code);
    public record DisableMfaRequest(string Password, string? MfaCode);
    public record VerifyMfaRequest(string Code);
}
```

4. Update Authentication Endpoint

Modify your authentication logic to handle MFA:

```
public static async Task<IResult> Authenticate(
    LoginRequest request,
    IUserService userService,
    IConfiguration configuration)
{
    try
    {
        var user = await userService.GetUserByUsernameAsync(request.Username);

        if (user == null || !BCrypt.Net.BCrypt.Verify(request.Password,
```

```
user.PasswordHash))
        {
            return Results.BadRequest("Invalid username or password");
        }

        // Check if MFA is enabled
        if (user.IsMfaEnabled)
        {
            // Return MFA required response (no token yet)
            return Results.Ok(new
            {
                mfaRequired = true,
                userId = user.Id,
                message = "MFA verification required"
            });
        }

        // No MFA required, generate token and return user
        var token = GenerateJwtToken(user, configuration);
        user.Token = token;

        return Results.Ok(user);
    }
    catch (Exception ex)
    {
        return Results.Problem($"Authentication error: {ex.Message}");
    }
}

public static async Task<IResult> VerifyMfaLogin(
    VerifyMfaLoginRequest request,
    IUserService userService,
    IConfiguration configuration)
{
    try
    {
        var user = await userService.GetUserByIdAsync(request.UserId);

        if (user == null || !user.IsMfaEnabled ||
string.IsNullOrEmpty(user.MfaSecret))
        {
            return Results.BadRequest("Invalid MFA verification request");
        }

        // Check backup codes first
        if (user.MfaBackupCodes.Contains(request.MfaCode))
        {
            // Remove used backup code
            user.MfaBackupCodes.Remove(request.MfaCode);
            await userService.UpdateUserAsync(user);
        }
        else
        {
            // Verify TOTP code
```

```
            var secretBytes = Base32Encoding.ToBytes(user.MfaSecret);
            var totp = new Totp(secretBytes);
            var isValid = totp.VerifyTotp(request.MfaCode, out _,
VerificationWindow.RfcSpecifiedNetworkDelay);

            if (!isValid)
                return Results.BadRequest("Invalid MFA code");
        }

        // Generate token and return user
        var token = GenerateJwtToken(user, configuration);
        user.Token = token;

        return Results.Ok(user);
    }
    catch (Exception ex)
    {
        return Results.Problem($"MFA verification error: {ex.Message}");
    }
}

public record VerifyMfaLoginRequest(int UserId, string MfaCode);
```

Frontend Implementation

1. Update User Model

Update your TypeScript User model:

```
export interface User {
  id: number;
  firstname: string;
  lastname: string;
  username: string;
  email: string;
  token?: string;
  isMfaEnabled?: boolean;
  mfaSetupDate?: Date;
}
```

2. Create MFA Service

Create a dedicated MFA service:

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';
import { environment } from '../../environments/environment';
```

```
export interface MfaSetupResponse {
  secret: string;
  qrCode: string;
  backupCodes: string[];
}

export interface MfaStatusResponse {
  isMfaEnabled: boolean;
  mfaSetupDate?: Date;
  backupCodesCount: number;
}

@Injectable({
  providedIn: 'root'
})
export class MfaService {
  private baseApiUrl = environment.baseApiUrl;

  constructor(private http: HttpClient) {}

  setupMfa(): Observable<MfaSetupResponse> {
    return this.http.post<MfaSetupResponse>(`${this.baseApiUrl}/mfa/setup`, {});
  }

  enableMfa(code: string): Observable<any> {
    return this.http.post(`${this.baseApiUrl}/mfa/enable`, { code });
  }

  disableMfa(password: string, mfaCode?: string): Observable<any> {
    return this.http.post(`${this.baseApiUrl}/mfa/disable`, {
      password,
      mfaCode
    });
  }

  verifyMfa(code: string): Observable<any> {
    return this.http.post(`${this.baseApiUrl}/mfa/verify`, { code });
  }

  getMfaStatus(): Observable<MfaStatusResponse> {
    return this.http.get<MfaStatusResponse>(`${this.baseApiUrl}/mfa/status`);
  }
}
```

3. Update User Service

Enhance the UserService with MFA support and localStorage abstraction:

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { BehaviorSubject, Observable, tap } from 'rxjs';
```

```
import { User } from '../models/user.model';
import { environment } from '../../environments/environment';

@Injectable({
  providedIn: 'root'
})
export class UserService {
  private currentUserSubject: BehaviorSubject<User | null>;
  public currentUser: Observable<User | null>;
  private baseApiUrl = environment.baseApiUrl;
  private readonly USER_STORAGE_KEY = 'user';

  constructor(private http: HttpClient) {
    const storedUser = this.getUserFromStorage();
    this.currentUserSubject = new BehaviorSubject<User | null>(storedUser);
    this.currentUser = this.currentUserSubject.asObservable();
  }

  public get currentUserValue(): User | null {
    return this.currentUserSubject.value;
  }

  /**
   * Get user data from localStorage
   */
  private getUserFromStorage(): User | null {
    try {
      const storedUser = localStorage.getItem(this.USER_STORAGE_KEY);
      return storedUser ? JSON.parse(storedUser) : null;
    } catch (error) {
      console.error('Error parsing user from storage:', error);
      this.clearUserFromStorage();
      return null;
    }
  }

  /**
   * Store user data in localStorage
   */
  private setUserInStorage(user: User): void {
    try {
      localStorage.setItem(this.USER_STORAGE_KEY, JSON.stringify(user));
    } catch (error) {
      console.error('Error storing user in storage:', error);
    }
  }

  /**
   * Clear user data from localStorage
   */
  private clearUserFromStorage(): void {
    localStorage.removeItem(this.USER_STORAGE_KEY);
  }
```

```
  /**
   * Get current user's token
   */
  public getCurrentUserToken(): string | null {
    const user = this.currentUserValue;
    return user?.token || null;
  }

  /**
   * Check if user is authenticated
   */
  public isAuthenticated(): boolean {
    const user = this.currentUserValue;
    return !!(user && user.token);
  }

  /**
   * Update current user and storage
   */
  public updateCurrentUser(user: User | null): void {
    if (user) {
      this.setUserInStorage(user);
    } else {
      this.clearUserFromStorage();
    }
    this.currentUserSubject.next(user);
  }

  login(username: string, password: string): Observable<any> {
    return this.http.post<any>(`${this.baseApiUrl}/users/authenticate`, {
username, password })
      .pipe(tap(response => {
        // If it's a complete user login (no MFA required)
        if (response.token && !response.mfaRequired) {
          this.updateCurrentUser(response);
        }
        // If MFA is required, don't update current user yet
      }));
  }

  /**
   * Verify MFA during login process
   */
  verifyMfaLogin(userId: number, mfaCode: string): Observable<User> {
    return this.http.post<User>(`${this.baseApiUrl}/users/verify-mfa-login`, {
      userId,
      mfaCode
    }).pipe(tap(user => {
      if (user.token) {
        this.updateCurrentUser(user);
      }
    }));
  }
```

```
  register(userData: any): Observable<User> {
    return this.http.post<User>(`${this.baseApiUrl}/users/register`, userData)
      .pipe(tap(user => this.updateCurrentUser(user)));
  }

  logout(): void {
    this.updateCurrentUser(null);
  }

  getCurrentUser(): Observable<User> {
    return this.http.get<User>(`${this.baseApiUrl}/users/current`);
  }

  updateUser(user: User): Observable<User> {
    return this.http.put<User>(`${this.baseApiUrl}/users/${user.id}`, user);
  }

  deleteUser(id: number): Observable<any> {
    return this.http.delete(`${this.baseApiUrl}/users/${id}`);
  }
}
```

4. Update Login Component

Implement two-stage login with MFA support:

```
import { Component, OnInit } from '@angular/core';
import { FormBuilder, FormGroup, Validators, ReactiveFormsModule } from
'@angular/forms';
import { Router } from '@angular/router';
import { CommonModule } from '@angular/common';
import { UserService } from '../../../services/user.service';

@Component({
  selector: 'app-login',
  standalone: true,
  imports: [CommonModule, ReactiveFormsModule],
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.css']
})
export class LoginComponent implements OnInit {
  loginForm!: FormGroup;
  mfaForm!: FormGroup;
  errorMessage = '';
  successMessage = '';
  isLoading = false;
  showMfaForm = false;
  pendingUserId: number | null = null;

  constructor(
    private formBuilder: FormBuilder,
```

```
    private userService: UserService,
    private router: Router
  ) {}

  ngOnInit(): void {
    this.loginForm = this.formBuilder.group({
      username: ['', [Validators.required]],
      password: ['', [Validators.required]]
    });

    this.mfaForm = this.formBuilder.group({
      mfaCode: ['', [Validators.required, Validators.pattern(/^\d{6}$/)]]
    });
  }

  onSubmit(): void {
    if (this.loginForm.valid) {
      this.isLoading = true;
      this.errorMessage = '';

      const { username, password } = this.loginForm.value;

      this.userService.login(username, password).subscribe({
        next: (response) => {
          this.isLoading = false;

          if (response.mfaRequired) {
            // Show MFA form
            this.showMfaForm = true;
            this.pendingUserId = response.userId;
            this.successMessage = 'Please enter your MFA code to complete login';
          } else if (response.token) {
            // Direct login success
            this.successMessage = 'Login successful!';
            this.router.navigate(['/notes']);
          }
        },
        error: (error) => {
          this.isLoading = false;
          this.errorMessage = error.error?.message || 'Login failed. Please try
again.';
          console.error('Login error:', error);
        }
      });
    } else {
      this.markFormGroupTouched(this.loginForm);
    }
  }

  onMfaSubmit(): void {
    if (this.mfaForm.valid && this.pendingUserId) {
      this.isLoading = true;
      this.errorMessage = '';
```

```
      const { mfaCode } = this.mfaForm.value;

      this.userService.verifyMfaLogin(this.pendingUserId, mfaCode).subscribe({
        next: (user) => {
          this.isLoading = false;
          this.successMessage = 'MFA verification successful!';
          this.router.navigate(['/notes']);
        },
        error: (error) => {
          this.isLoading = false;
          this.errorMessage = error.error?.message || 'Invalid MFA code. Please
try again.';
          console.error('MFA verification error:', error);
        }
      });
    } else {
      this.markFormGroupTouched(this.mfaForm);
    }
  }

  backToLogin(): void {
    this.showMfaForm = false;
    this.pendingUserId = null;
    this.mfaForm.reset();
    this.errorMessage = '';
    this.successMessage = '';
  }

  private markFormGroupTouched(formGroup: FormGroup): void {
    Object.keys(formGroup.controls).forEach(key => {
      formGroup.get(key)?.markAsTouched();
    });
  }
}
```

5. Login Component Template

```
<div class="container mt-5">
  <div class="row justify-content-center">
    <div class="col-md-6">
      <div class="card">
        <div class="card-header">
          <h4 class="mb-0">{{ showMfaForm ? 'Two-Factor Authentication' : 'Login'
}}</h4>
        </div>
        <div class="card-body">

          <!-- Success Message -->
          <div *ngIf="successMessage" class="alert alert-success">
            {{ successMessage }}
          </div>
```

```
          <!-- Error Message -->
          <div *ngIf="errorMessage" class="alert alert-danger">
            {{ errorMessage }}
          </div>

          <!-- Standard Login Form -->
          <form *ngIf="!showMfaForm" [formGroup]="loginForm"
(ngSubmit)="onSubmit()">
            <div class="mb-3">
              <label for="username" class="form-label">Username</label>
              <input
                type="text"
                class="form-control"
                id="username"
                formControlName="username"
                [class.is-invalid]="loginForm.get('username')?.invalid &&
loginForm.get('username')?.touched">
              <div *ngIf="loginForm.get('username')?.invalid &&
loginForm.get('username')?.touched" class="invalid-feedback">
                Username is required
              </div>
            </div>

            <div class="mb-3">
              <label for="password" class="form-label">Password</label>
              <input
                type="password"
                class="form-control"
                id="password"
                formControlName="password"
                [class.is-invalid]="loginForm.get('password')?.invalid &&
loginForm.get('password')?.touched">
              <div *ngIf="loginForm.get('password')?.invalid &&
loginForm.get('password')?.touched" class="invalid-feedback">
                Password is required
              </div>
            </div>

            <button type="submit" class="btn btn-primary" [disabled]="isLoading">
              <span *ngIf="isLoading" class="spinner-border spinner-border-sm me-
2"></span>
              {{ isLoading ? 'Logging in...' : 'Login' }}
            </button>
            <a routerLink="/signup" class="btn btn-link">Don't have an account?
Sign up</a>
          </form>

          <!-- MFA Verification Form -->
          <form *ngIf="showMfaForm" [formGroup]="mfaForm"
(ngSubmit)="onMfaSubmit()">
            <div class="mb-3">
              <p class="text-muted">
                Enter the 6-digit code from your authenticator app or use a backup
```

```
code.
              </p>
              <label for="mfaCode" class="form-label">Authentication Code</label>
              <input
                type="text"
                class="form-control"
                id="mfaCode"
                formControlName="mfaCode"
                placeholder="000000"
                maxlength="6"
                [class.is-invalid]="mfaForm.get('mfaCode')?.invalid &&
mfaForm.get('mfaCode')?.touched">
              <div *ngIf="mfaForm.get('mfaCode')?.invalid &&
mfaForm.get('mfaCode')?.touched" class="invalid-feedback">
                Please enter a valid 6-digit code
              </div>
            </div>

            <div class="d-flex gap-2">
              <button type="submit" class="btn btn-primary"
[disabled]="isLoading">
                <span *ngIf="isLoading" class="spinner-border spinner-border-sm
me-2"></span>
                {{ isLoading ? 'Verifying...' : 'Verify' }}
              </button>
              <button type="button" class="btn btn-secondary"
(click)="backToLogin()" [disabled]="isLoading">
                Back to Login
              </button>
            </div>
          </form>

        </div>
      </div>
    </div>
  </div>
</div>
```

6. Create User Status Component

Create a header component that shows security warnings:

```
import { Component, OnInit, OnDestroy } from '@angular/core';
import { CommonModule } from '@angular/common';
import { RouterModule } from '@angular/router';
import { UserService } from '../../services/user.service';
import { User } from '../../models/user.model';
import { Subscription } from 'rxjs';

@Component({
  selector: 'app-user-status',
```

```
  standalone: true,
  imports: [CommonModule, RouterModule],
  template: `
    <div *ngIf="user" class="d-flex align-items-center">
      <!-- User Icon and Name with Security Styling -->
      <a routerLink="/profile"
         class="text-decoration-none d-flex align-items-center"
         [class.text-danger]="!user.isMfaEnabled"
         [class.text-light]="user.isMfaEnabled">

        <!-- User Icon -->
        <i class="fas fa-user-circle me-2 fs-5"
           [class.text-danger]="!user.isMfaEnabled"></i>

        <!-- User Name -->
        <span class="me-2">{{ user.firstname }} {{ user.lastname }}</span>
      </a>

      <!-- Security Warning Icon -->
      <span *ngIf="!user.isMfaEnabled"
            class="text-warning me-2 pulsing-warning"
            title="Security Warning: Multi-Factor Authentication is not enabled.
Click to enable MFA for better account security."
            data-bs-toggle="tooltip"
            data-bs-placement="bottom">
        <i class="fas fa-exclamation-triangle"></i>
      </span>
    </div>
  `,
  styles: [`
    .pulsing-warning {
      animation: pulse 2s infinite;
    }

    @keyframes pulse {
      0% { opacity: 1; }
      50% { opacity: 0.5; }
      100% { opacity: 1; }
    }

    .text-danger:hover {
      text-decoration: underline !important;
    }
  `]
})
export class UserStatusComponent implements OnInit, OnDestroy {
  user: User | null = null;
  private userSubscription?: Subscription;

  constructor(private userService: UserService) {}

  ngOnInit(): void {
    this.userSubscription = this.userService.currentUser.subscribe(user => {
      this.user = user;
```

```
    });
  }

  ngOnDestroy(): void {
    this.userSubscription?.unsubscribe();
  }
}
```

7. Create MFA Setup Component

```
import { Component, OnInit } from '@angular/core';
import { CommonModule } from '@angular/common';
import { FormBuilder, FormGroup, Validators, ReactiveFormsModule } from
'@angular/forms';
import { Router } from '@angular/router';
import { MfaService, MfaSetupResponse } from '../../../services/mfa.service';
import { UserService } from '../../../services/user.service';

@Component({
  selector: 'app-mfa-setup',
  standalone: true,
  imports: [CommonModule, ReactiveFormsModule],
  templateUrl: './mfa-setup.component.html',
  styleUrls: ['./mfa-setup.component.css']
})
export class MfaSetupComponent implements OnInit {
  currentStep = 1;
  totalSteps = 4;
  setupForm!: FormGroup;
  setupData: MfaSetupResponse | null = null;
  isLoading = false;
  errorMessage = '';
  successMessage = '';

  constructor(
    private formBuilder: FormBuilder,
    private mfaService: MfaService,
    private userService: UserService,
    private router: Router
  ) {}

  ngOnInit(): void {
    this.setupForm = this.formBuilder.group({
      verificationCode: ['', [Validators.required, Validators.pattern(/^\d{6}$/)]]
    });

    this.startSetup();
  }

  startSetup(): void {
    this.isLoading = true;
```

```
    this.errorMessage = '';

    this.mfaService.setupMfa().subscribe({
      next: (response) => {
        this.setupData = response;
        this.isLoading = false;
        this.currentStep = 2;
      },
      error: (error) => {
        this.isLoading = false;
        this.errorMessage = error.error?.message || 'Failed to setup MFA. Please
try again.';
        console.error('MFA setup error:', error);
      }
    });
  }

  nextStep(): void {
    if (this.currentStep < this.totalSteps) {
      this.currentStep++;
    }
  }

  verifyAndEnable(): void {
    if (this.setupForm.valid) {
      this.isLoading = true;
      this.errorMessage = '';

      const verificationCode = this.setupForm.get('verificationCode')?.value;

      this.mfaService.enableMfa(verificationCode).subscribe({
        next: (response) => {
          this.isLoading = false;
          this.successMessage = response.message;
          this.currentStep = 4;

          // Update the current user's MFA status in UserService
          this.updateUserMfaStatus(true);
        },
        error: (error) => {
          this.isLoading = false;
          this.errorMessage = error.error?.message || 'Invalid verification code.
Please try again.';
          console.error('MFA enable error:', error);
        }
      });
    } else {
      this.setupForm.get('verificationCode')?.markAsTouched();
    }
  }

  downloadBackupCodes(): void {
    if (!this.setupData?.backupCodes) return;
```

```
    const codesText = this.setupData.backupCodes.join('\n');
    const blob = new Blob([codesText], { type: 'text/plain' });
    const url = window.URL.createObjectURL(blob);

    const link = document.createElement('a');
    link.href = url;
    link.download = 'listkeeper-backup-codes.txt';
    link.click();

    window.URL.revokeObjectURL(url);
  }

  copyBackupCodes(): void {
    if (!this.setupData?.backupCodes) return;

    const codesText = this.setupData.backupCodes.join('\n');
    navigator.clipboard.writeText(codesText).then(() => {
      this.successMessage = 'Backup codes copied to clipboard!';
      setTimeout(() => this.successMessage = '', 3000);
    }).catch(() => {
      this.errorMessage = 'Failed to copy backup codes. Please manually save
them.';
    });
  }

  finishSetup(): void {
    this.router.navigate(['/profile'], {
      queryParams: { mfaEnabled: 'true' }
    });
  }

  cancel(): void {
    this.router.navigate(['/profile']);
  }

  // Helper method to check if step is current
  isStepCurrent(step: number): boolean {
    return this.currentStep === step;
  }

  /**
   * Update the current user's MFA status in UserService and localStorage
   */
  private updateUserMfaStatus(isMfaEnabled: boolean): void {
    const currentUser = this.userService.currentUserValue;
    if (currentUser) {
      // Update the user object with new MFA status
      const updatedUser = { ...currentUser, isMfaEnabled, mfaSetupDate: new Date()
};

      // Update the current user in UserService
      // This will update both the BehaviorSubject and localStorage
      this.userService.updateCurrentUser(updatedUser);
    }
```

```
  }
}
```

8. Update User Profile Component

Add MFA management to the profile:

```
import { Component, OnInit } from '@angular/core';
import { CommonModule } from '@angular/common';
import { RouterModule, Router, ActivatedRoute } from '@angular/router';
import { FormBuilder, FormGroup, Validators, ReactiveFormsModule } from
'@angular/forms';
import { UserService } from '../../../services/user.service';
import { MfaService } from '../../../services/mfa.service';
import { User } from '../../../models/user.model';

@Component({
  selector: 'app-user-profile',
  standalone: true,
  imports: [CommonModule, RouterModule, ReactiveFormsModule],
  templateUrl: './user-profile.component.html',
  styleUrls: ['./user-profile.component.css']
})
export class UserProfileComponent implements OnInit {
  user: User | null = null;
  isLoading = false;
  errorMessage = '';
  successMessage = '';
  showDisableMfaForm = false;
  disableMfaForm!: FormGroup;

  constructor(
    private userService: UserService,
    private mfaService: MfaService,
    private router: Router,
    private route: ActivatedRoute,
    private formBuilder: FormBuilder
  ) {}

  ngOnInit(): void {
    this.user = this.userService.currentUserValue;

    this.disableMfaForm = this.formBuilder.group({
      password: ['', [Validators.required]],
      mfaCode: ['', [Validators.required, Validators.pattern(/^\d{6}$/)]]
    });

    // Check for success message from MFA setup
    this.route.queryParams.subscribe(params => {
      if (params['mfaEnabled'] === 'true') {
        this.successMessage = 'MFA has been successfully enabled for your
```

```
account!';
        setTimeout(() => this.successMessage = '', 5000);
      }
    });
  }

  enableMfa(): void {
    this.router.navigate(['/profile/mfa-setup']);
  }

  promptDisableMfa(): void {
    this.showDisableMfaForm = true;
    this.errorMessage = '';
    this.disableMfaForm.reset();
  }

  disableMfa(): void {
    if (this.disableMfaForm.valid) {
      this.isLoading = true;
      this.errorMessage = '';

      const { password, mfaCode } = this.disableMfaForm.value;

      this.mfaService.disableMfa(password, mfaCode).subscribe({
        next: (response) => {
          this.isLoading = false;
          this.successMessage = response.message;
          this.showDisableMfaForm = false;

          // Update user object
          if (this.user) {
            this.user.isMfaEnabled = false;
            this.userService.updateCurrentUser(this.user);
          }
        },
        error: (error) => {
          this.isLoading = false;
          this.errorMessage = error.error?.message || 'Failed to disable MFA.
Please try again.';
          console.error('Disable MFA error:', error);
        }
      });
    } else {
      this.markFormGroupTouched(this.disableMfaForm);
    }
  }

  cancelDisableMfa(): void {
    this.showDisableMfaForm = false;
    this.disableMfaForm.reset();
    this.errorMessage = '';
  }

  logout(): void {
```

```
    this.userService.logout();
    this.router.navigate(['/login']);
  }

  private markFormGroupTouched(formGroup: FormGroup): void {
    Object.keys(formGroup.controls).forEach(key => {
      formGroup.get(key)?.markAsTouched();
    });
  }
}
```

9. Update Main Router Configuration

Configure standalone routing in `main.ts`:

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app/app.component';
import { importProvidersFrom } from '@angular/core';
import { HttpClientModule, HTTP_INTERCEPTORS } from '@angular/common/http';
import { provideRouter } from '@angular/router';
import { AuthInterceptor } from './app/interceptors/auth.interceptor';

// Import components
import { HomeComponent } from './app/components/home/home.component';
import { NoteListComponent } from './app/components/notes/note-list/note-
list.component';
import { SignupComponent } from './app/components/users/signup/signup.component';
import { LoginComponent } from './app/components/users/login/login.component';
import { UserProfileComponent } from './app/components/users/user-profile/user-
profile.component';
import { MfaSetupComponent } from './app/components/users/mfa-setup/mfa-
setup.component';
import { MfaVerificationComponent } from './app/components/users/mfa-
verification/mfa-verification.component';
import { AuthGuard } from './app/guards/auth.guard';

const routes = [
  {
    path: '',
    component: HomeComponent,
    data: { debug: 'Root route - HomeComponent' }
  },
  {
    path: 'notes',
    component: NoteListComponent,
    canActivate: [AuthGuard],
    data: { debug: 'Notes route - NoteListComponent' }
  },
  {
    path: 'profile',
    component: UserProfileComponent,
```

```
    canActivate: [AuthGuard],
    data: { debug: 'Profile route - UserProfileComponent' }
  },
  {
    path: 'profile/mfa-setup',
    component: MfaSetupComponent,
    canActivate: [AuthGuard],
    data: { debug: 'MFA Setup route - MfaSetupComponent' }
  },
  {
    path: 'profile/mfa-verification',
    component: MfaVerificationComponent,
    canActivate: [AuthGuard],
    data: { debug: 'MFA Verification route - MfaVerificationComponent' }
  },
  {
    path: 'signup',
    component: SignupComponent,
    data: { debug: 'Signup route - SignupComponent' }
  },
  {
    path: 'login',
    component: LoginComponent,
    data: { debug: 'Login route - LoginComponent' }
  },
  {
    path: '**',
    redirectTo: '',
    data: { debug: 'Wildcard route - redirect to root' }
  }
];

bootstrapApplication(AppComponent, {
  providers: [
    importProvidersFrom(HttpClientModule),
    provideRouter(routes),
    {
      provide: HTTP_INTERCEPTORS,
      useClass: AuthInterceptor,
      multi: true
    }
  ]
}).catch(err => console.error(err));
```

Testing the Implementation

1. Backend Testing

Test the MFA endpoints using a tool like Postman:

1. **Setup MFA**: `POST /mfa/setup`
2. **Enable MFA**: `POST /mfa/enable` with verification code

3. **Verify MFA**: `POST /mfa/verify` with TOTP code
4. **Disable MFA**: `POST /mfa/disable` with password and MFA code
5. **Get Status**: `GET /mfa/status`

2. Frontend Testing

1. **Login Flow**:
    - Login with username/password
    - If MFA enabled, verify with TOTP code
    - Successfully authenticate
2. **MFA Setup**:
    - Navigate to profile
    - Click "Enable MFA"
    - Scan QR code with authenticator app
    - Verify with generated code
    - Save backup codes
3. **Security Warnings**:
    - Users without MFA should see red username and warning icon
    - Users with MFA should see normal styling
4. **Real-time Updates**:
    - After enabling MFA, UI should update immediately
    - No logout/login required

Security Considerations

1. **Secret Storage**: MFA secrets are stored securely in the database
2. **Backup Codes**: Single-use backup codes for account recovery
3. **Rate Limiting**: Implement rate limiting on MFA verification attempts
4. **Session Management**: Proper token handling and expiration
5. **HTTPS**: Always use HTTPS in production
6. **Input Validation**: Validate all MFA codes and user inputs

Conclusion

This implementation provides a complete MFA system with:

- TOTP-based authentication using industry standards
- QR code generation for easy setup
- Backup codes for account recovery
- Real-time UI updates
- Security warnings for non-MFA users
- Clean separation of concerns between frontend and backend

The system follows security best practices and provides a smooth user experience while maintaining high security standards.