Platform API Specifications - Social Media Integration Guide

Supported Social Media Platforms

Platform Priority & Implementation Order

1. **Twitter/X API** (Phase 1) - Text-focused, API-friendly
2. **Instagram Graph API** (Phase 2) - Visual content, automation-friendly
3. **LinkedIn API** (Phase 3) - Professional networking
4. **Facebook Graph API** (Phase 4) - Broader social reach
5. **TikTok Business API** (Phase 5) - Video content, younger audience

Twitter/X API Integration

API Configuration

```
interface TwitterAPIConfig {
  apiVersion: 'v2'; // Current Twitter API version
  baseUrl: 'https://api.twitter.com/2';
  authentication: TwitterAuth;
  rateLimits: TwitterRateLimits;
}

interface TwitterAuth {
  // OAuth 2.0 PKCE (Recommended for automation)
  authType: 'oauth2_pkce' | 'oauth1_user_context' |
'bearer_token';
  credentials: {
    clientId: string; // Twitter app client ID
    clientSecret: string; // Encrypted in database
    bearerToken?: string; // App-only authentication
    accessToken?: string; // User context token (per
persona)
    refreshToken?: string; // Token refresh capability
  };
  scopes: TwitterScope[];
}
```

```
type TwitterScope =
  | 'tweet.read' | 'tweet.write' | 'tweet.moderate.write'
  | 'users.read' | 'follows.read' | 'follows.write'
  | 'offline.access' | 'space.read' | 'mute.read' |
'mute.write'
  | 'like.read' | 'like.write' | 'list.read' |
'list.write';
```

Tweet Management Operations

```
interface TwitterOperations {
  // Core posting functionality
  createTweet: (content: TweetContent) =>
Promise<TweetResponse>;
  deleteTweet: (tweetId: string) => Promise<void>;

  // Thread management
  createThread: (tweets: TweetContent[]) =>
Promise<ThreadResponse>;

  // Engagement operations
  likeTweet: (tweetId: string) => Promise<void>;
  unlikeTweet: (tweetId: string) => Promise<void>;
  retweetTweet: (tweetId: string) => Promise<void>;
  replyToTweet: (tweetId: string, content: string) =>
Promise<TweetResponse>;

  // Social operations
  followUser: (userId: string) => Promise<void>;
  unfollowUser: (userId: string) => Promise<void>;

  // Data retrieval
  getUserTweets: (userId: string, options?:
TweetQueryOptions) => Promise<Tweet[]>;
  getTweet: (tweetId: string) => Promise<Tweet>;
  searchTweets: (query: string, options?: SearchOptions) =>
Promise<Tweet[]>;
}

interface TweetContent {
  text: string; // Max 280 characters
```

```
  media?: MediaAttachment[];
  poll?: PollOptions;
  reply?: {
    in_reply_to_tweet_id: string;
  };
  quote_tweet_id?: string;
  geo?: GeotagOptions;
  reply_settings?: 'everyone' | 'mentionedUsers' |
'following';
}

interface MediaAttachment {
  media_id: string; // From media upload endpoint
  alt_text?: string; // Accessibility description
  tagged_users?: string[]; // User IDs to tag in media
}
```

Twitter Rate Limits & Constraints

```
interface TwitterRateLimits {
  // Posting limits (per 24 hours)
  tweets: {
    limit: 300; // Tweets per day
    window: '24h';
    includesReplies: true;
    includesRetweets: true;
  };

  // API call limits (per 15 minutes)
  apiCalls: {
    tweetCreation: { limit: 300; window: '15m' };
    tweetDeletion: { limit: 300; window: '15m' };
    userLookup: { limit: 300; window: '15m' };
    followManagement: { limit: 50; window: '15m' };
    tweetSearch: { limit: 180; window: '15m' };
  };

  // Engagement limits
  engagement: {
    likes: { limit: 1000; window: '24h' };
    follows: { limit: 400; window: '24h' };
```

```
    unfollows: { limit: 400; window: '24h' };
  };
}

// Rate limiting implementation
class TwitterRateLimiter {
  private buckets = new Map<string, RateLimitBucket>();

  async checkRateLimit(operation: string, personaId:
string): Promise<boolean> {
    const key = `${personaId}:${operation}`;
    const bucket = this.buckets.get(key) ||
this.createBucket(operation);

    return bucket.hasCapacity();
  }

  async waitForRateLimit(operation: string, personaId:
string): Promise<void> {
    const key = `${personaId}:${operation}`;
    const bucket = this.buckets.get(key);

    if (bucket && !bucket.hasCapacity()) {
      const waitTime = bucket.getResetTime() - Date.now();
      await new Promise(resolve => setTimeout(resolve,
waitTime));
    }
  }
}
```

Instagram Graph API Integration

Instagram API Configuration

```
interface InstagramAPIConfig {
  apiVersion: 'v20.0'; // Current Graph API version
  baseUrl: 'https://graph.facebook.com/v20.0';
  authentication: InstagramAuth;
  accountType: 'business' | 'creator'; // Required for API
access
}
```

```
interface InstagramAuth {
  authType: 'oauth2';
  credentials: {
    appId: string; // Facebook App ID
    appSecret: string; // Encrypted app secret
    accessToken: string; // Long-lived user access token
(60 days)
    refreshToken?: string;
    pageId: string; // Instagram Business Account ID
  };
  permissions: InstagramPermission[];
}

type InstagramPermission =
  | 'instagram_basic' | 'instagram_content_publish'
  | 'instagram_manage_comments' |
'instagram_manage_insights'
  | 'pages_read_engagement' | 'pages_show_list';
```

Instagram Content Operations

```
interface InstagramOperations {
  // Content publishing
  publishPhoto: (content: PhotoContent) =>
Promise<MediaResponse>;
  publishVideo: (content: VideoContent) =>
Promise<MediaResponse>;
  publishCarousel: (content: CarouselContent) =>
Promise<MediaResponse>;
  publishStory: (content: StoryContent) =>
Promise<StoryResponse>;

  // Content management
  getMediaList: (options?: MediaQueryOptions) =>
Promise<Media[]>;
  getMediaDetails: (mediaId: string) =>
Promise<MediaDetails>;
  deleteMedia: (mediaId: string) => Promise<void>;

  // Engagement operations
```

```
  getComments: (mediaId: string) => Promise<Comment[]>;
  replyToComment: (commentId: string, reply: string) =>
Promise<CommentResponse>;
  hideComment: (commentId: string) => Promise<void>;

  // Analytics
  getInsights: (mediaId: string) => Promise<MediaInsights>;
  getAccountInsights: (metrics: string[], period: 'day' |
'week' | 'days_28') => Promise<AccountInsights>;
}

interface PhotoContent {
  image_url: string; // Publicly accessible image URL
  caption?: string; // Max 2,200 characters
  location_id?: string; // Facebook location ID
  user_tags?: UserTag[]; // Tag other Instagram users
  alt_text?: string; // Accessibility text
  thumb_offset?: number; // For video thumbnails
}

interface VideoContent extends PhotoContent {
  video_url: string; // Publicly accessible video URL
  thumbnail_offset?: number; // Video thumbnail timestamp
  media_type: 'REELS' | 'VIDEO'; // Instagram Reels vs
regular video
}

interface CarouselContent {
  children: (PhotoContent | VideoContent)[];
  caption?: string;
  location_id?: string;
  user_tags?: UserTag[];
}
```

Instagram Publishing Workflow

```
class InstagramPublisher {
  private apiClient: InstagramAPIClient;

  async publishContent(personaId: string, content:
InstagramContent): Promise<PublishResult> {
```

```
    // Step 1: Create media container
    const container = await
this.createMediaContainer(content);

    // Step 2: Wait for container processing (required for
videos)
    if (content.type === 'video' || content.type ===
'reel') {
      await this.waitForProcessing(container.id);
    }

    // Step 3: Publish the container
    const publishResult = await
this.publishContainer(container.id);

    // Step 4: Track publication for rate limiting
    await this.trackPublication(personaId,
publishResult.id);

    return publishResult;
  }

  private async waitForProcessing(containerId: string):
Promise<void> {
    const maxWaitTime = 60000; // 1 minute max
    const checkInterval = 5000; // Check every 5 seconds
    const startTime = Date.now();

    while (Date.now() - startTime < maxWaitTime) {
      const status = await
this.checkContainerStatus(containerId);

      if (status.status_code === 'FINISHED') {
        return; // Ready to publish
      }

      if (status.status_code === 'ERROR') {
        throw new Error(`Media processing failed: $
{status.error_message}`);
      }
```

```
      await new Promise(resolve => setTimeout(resolve,
checkInterval));
    }

    throw new Error('Media processing timeout');
  }
}
```

Instagram Rate Limits & Best Practices

```
interface InstagramRateLimits {
  // Content publishing limits
  publishing: {
    photos: { limit: 25; window: '24h' };
    videos: { limit: 25; window: '24h' };
    stories: { limit: 100; window: '24h' };
    carousels: { limit: 25; window: '24h' }; // Count as
single post
  };

  // API call limits (per hour)
  apiCalls: {
    standard: { limit: 200; window: '1h' };
    insights: { limit: 200; window: '1h' };
    messaging: { limit: 1000; window: '1h' };
  };

  // Media processing constraints
  mediaSpecs: {
    photo: {
      formats: ['JPEG', 'PNG'];
      maxSize: '8MB';
      minResolution: '320x320';
      maxResolution: '1440x1800';
      aspectRatio: '4:5 to 1.91:1';
    };
    video: {
      formats: ['MP4', 'MOV'];
      maxSize: '100MB';
      maxDuration: '60s';
```

```
      minResolution: '720p';
      frameRate: '30fps max';
    };
  };
}
```

LinkedIn API Integration

LinkedIn API Configuration

```
interface LinkedInAPIConfig {
  apiVersion: 'v2';
  baseUrl: 'https://api.linkedin.com/v2';
  authentication: LinkedInAuth;
  accountType: 'personal' | 'company'; // Persona type
}

interface LinkedInAuth {
  authType: 'oauth2';
  credentials: {
    clientId: string;
    clientSecret: string; // Encrypted
    accessToken: string; // 60-day token
    refreshToken: string;
    personUrn?: string; // LinkedIn member ID
    organizationUrn?: string; // Company page ID
  };
  scopes: LinkedInScope[];
}

type LinkedInScope =
  | 'r_liteprofile' | 'r_emailaddress' | 'w_member_social'
  | 'r_organization_social' | 'w_organization_social'
  | 'rw_organization_admin';
```

LinkedIn Content Operations

```
interface LinkedInOperations {
  // Content sharing
  shareUpdate: (content: ShareContent) =>
Promise<ShareResponse>;
```

```
  shareArticle: (article: ArticleContent) =>
Promise<ShareResponse>;

  // Media uploads
  uploadImage: (imageData: Buffer, filename: string) =>
Promise<MediaUploadResponse>;
  uploadVideo: (videoData: Buffer, filename: string) =>
Promise<VideoUploadResponse>;

  // Profile management
  getProfile: () => Promise<LinkedInProfile>;
  updateProfile: (updates: ProfileUpdates) =>
Promise<void>;

  // Network operations
  getConnections: () => Promise<Connection[]>;
  sendConnectionRequest: (personUrn: string, message?:
string) => Promise<void>;

  // Content analytics
  getShareStatistics: (shareUrn: string) =>
Promise<ShareStatistics>;
  getFollowerStatistics: () => Promise<FollowerStatistics>;
}

interface ShareContent {
  author: string; // Person or organization URN
  lifecycleState: 'PUBLISHED' | 'DRAFT';
  specificContent: {
    'com.linkedin.ugc.ShareContent': {
      shareCommentary: {
        text: string; // Max 3,000 characters
        attributes?: TextAttribute[];
      };
      shareMediaCategory: 'NONE' | 'ARTICLE' | 'IMAGE' |
'VIDEO';
      media?: MediaContent[];
    };
  };
  visibility: {
```

```
    'com.linkedin.ugc.MemberNetworkVisibility': 'PUBLIC' |
'CONNECTIONS';
  };
}

interface MediaContent {
  status: 'READY';
  description: {
    text: string;
    attributes?: TextAttribute[];
  };
  media: string; // Media URN from upload
  title?: {
    text: string;
  };
}
```

Facebook Graph API Integration

Facebook API Configuration

```
interface FacebookAPIConfig {
  apiVersion: 'v20.0';
  baseUrl: 'https://graph.facebook.com/v20.0';
  authentication: FacebookAuth;
  pageType: 'personal' | 'business'; // Profile vs Page
}

interface FacebookAuth {
  authType: 'oauth2';
  credentials: {
    appId: string;
    appSecret: string; // Encrypted
    userAccessToken: string; // Short-lived (1 hour)
    pageAccessToken: string; // Long-lived page token
    longLivedToken?: string; // 60-day user token
    pageId: string; // Facebook page ID
  };
  permissions: FacebookPermission[];
}
```

```
type FacebookPermission =
  | 'pages_manage_posts' | 'pages_read_engagement'
  | 'publish_to_groups' | 'user_posts' | 'user_photos'
  | 'pages_show_list' | 'pages_manage_metadata';
```

Facebook Content Operations

```
interface FacebookOperations {
  // Page posting
  publishPost: (content: FacebookPost) =>
Promise<PostResponse>;
  publishPhoto: (photo: PhotoPost) =>
Promise<PostResponse>;
  publishVideo: (video: VideoPost) =>
Promise<PostResponse>;

  // Story publishing
  publishStory: (story: StoryContent) =>
Promise<StoryResponse>;

  // Content management
  getPagePosts: (options?: PostQueryOptions) =>
Promise<Post[]>;
  updatePost: (postId: string, updates: PostUpdates) =>
Promise<void>;
  deletePost: (postId: string) => Promise<void>;

  // Engagement
  getPostComments: (postId: string) => Promise<Comment[]>;
  replyToComment: (commentId: string, reply: string) =>
Promise<CommentResponse>;
  likePost: (postId: string) => Promise<void>;

  // Analytics
  getPageInsights: (metrics: string[], period: string) =>
Promise<PageInsights>;
  getPostInsights: (postId: string) =>
Promise<PostInsights>;
}

interface FacebookPost {
```

```
  message?: string; // Post text content
  link?: string; // Shared link URL
  published: boolean; // true for immediate, false for
draft
  scheduled_publish_time?: number; // Unix timestamp for
scheduling
  targeting?: {
    geo_locations?: {
      countries?: string[];
      regions?: string[];
      cities?: string[];
    };
    locales?: string[];
    age_min?: number;
    age_max?: number;
  };
  call_to_action?: {
    type: 'LEARN_MORE' | 'SHOP_NOW' | 'BOOK_TRAVEL' |
'LISTEN_MUSIC';
    value: {
      link: string;
      link_caption?: string;
    };
  };
}
```

TikTok Business API Integration

TikTok API Configuration (Limited Availability)

```
interface TikTokAPIConfig {
  apiVersion: 'v1.3';
  baseUrl: 'https://business-api.tiktok.com/open_api/v1.3';
  authentication: TikTokAuth;
  accountType: 'business'; // Only business accounts have
API access
}

interface TikTokAuth {
  authType: 'oauth2';
  credentials: {
```

```
    appId: string;
    secret: string; // Encrypted
    accessToken: string;
    refreshToken: string;
    advertiserIds: string[]; // TikTok ad accounts
  };
  scopes: TikTokScope[];
}

type TikTokScope =
  | 'video.list' | 'video.upload' | 'user.info.basic'
  | 'video.publish' | 'video.delete';

// Note: TikTok's API is primarily for advertising and
analytics
// Content posting via API is very limited and requires
special approval
```

Cross-Platform Content Adaptation

Content Transformation System

```
interface ContentAdapter {
  adaptForPlatform: (
    content: UniversalContent,
    targetPlatform: SocialPlatform,
    persona: PenNamePersona
  ) => Promise<PlatformContent>;
}

interface UniversalContent {
  // Core content
  message: string;
  media?: MediaAsset[];

  // Metadata
  themes: string[];
  mood: 'professional' | 'casual' | 'humorous' |
'inspirational';
  callToAction?: string;
```

```
  // Targeting
  targetAudience: string[];
  objectives: ContentObjective[];
}

interface PlatformContent {
  platform: SocialPlatform;
  adaptedContent: any; // Platform-specific content
structure
  publishOptions: PublishOptions;
  schedulingRecommendations: SchedulingOptions;
}

class ContentAdapter {
  async adaptForTwitter(content: UniversalContent, persona:
PenNamePersona): Promise<TweetContent> {
    const adaptedText = await
this.adaptText(content.message, {
      maxLength: 280,
      style: persona.platformPersonas.twitter.tweetStyle,
      hashtagStrategy:
persona.platformPersonas.twitter.hashtagUsage
    });

    return {
      text: adaptedText,
      media: await this.adaptMedia(content.media,
'twitter'),
      // Additional Twitter-specific fields
    };
  }

  async adaptForInstagram(content: UniversalContent,
persona: PenNamePersona): Promise<PhotoContent> {
    return {
      image_url: content.media?.[0]?.url || await
this.generateImageForText(content.message),
      caption: await this.adaptText(content.message, {
        maxLength: 2200,
```

```
        style:
persona.platformPersonas.instagram.captionLength,
        includeHashtags: true
      }),
      // Additional Instagram-specific fields
    };
  }

  private async adaptText(text: string, options:
TextAdaptationOptions): Promise<string> {
    // Implement text adaptation logic
    // - Truncate for character limits
    // - Adjust tone and style
    // - Add platform-appropriate hashtags
    // - Include platform-specific formatting

    return adaptedText;
  }
}
```

Error Handling & Rate Limiting

Universal Error Handling

```
interface PlatformError {
  platform: SocialPlatform;
  errorType: ErrorType;
  errorCode: string | number;
  message: string;
  retryable: boolean;
  retryAfter?: number; // Seconds to wait before retry
}

type ErrorType =
  | 'rate_limit_exceeded'
  | 'authentication_failed'
  | 'content_policy_violation'
  | 'media_processing_failed'
  | 'network_error'
  | 'account_suspended'
  | 'feature_not_available';
```

```
class PlatformErrorHandler {
  async handleError(error: PlatformError, context:
OperationContext): Promise<ErrorHandlingResult> {
    switch (error.errorType) {
      case 'rate_limit_exceeded':
        return await this.handleRateLimit(error, context);

      case 'authentication_failed':
        return await this.refreshAuthentication(error,
context);

      case 'content_policy_violation':
        return await this.handleContentViolation(error,
context);

      case 'account_suspended':
        return await this.handleAccountSuspension(error,
context);

      default:
        return await this.handleGenericError(error,
context);
    }
  }

  private async handleRateLimit(error: PlatformError,
context: OperationContext): Promise<ErrorHandlingResult> {
    const waitTime = error.retryAfter ||
this.calculateBackoffTime(context.retryCount);

    // Schedule retry
    await this.scheduleRetry(context.operation, waitTime);

    // Log rate limit hit for optimization
    await this.logRateLimit(error.platform,
context.personaId, waitTime);

    return {
      action: 'retry_scheduled',
```

```
      waitTime,
      message: `Rate limit hit, retrying in ${waitTime}
seconds`
    };
  }
}
```

API Security & Best Practices

Authentication Security

```
interface SecurityBestPractices {
  // Token management
  tokenRotation: {
    refreshThreshold: number; // Days before expiration to
refresh
    autoRefresh: boolean;
    fallbackTokens: boolean; // Keep backup valid tokens
  };

  // Request security
  requestSecurity: {
    useHttps: true;
    validateCertificates: true;
    timeoutMs: number;
    maxRetries: number;
  };

  // Data protection
  dataProtection: {
    encryptTokens: boolean;
    logSensitiveData: false;
    sanitizeErrorMessages: boolean;
  };
}

class APISecurityManager {
  private encryptionService: EncryptionService;

  async storeCredentials(platform: SocialPlatform,
credentials: any): Promise<void> {
```

```
    const encrypted = await
this.encryptionService.encrypt(JSON.stringify(credentials))
;

    await this.database.storeCredentials({
      platform,
      personaId: credentials.personaId,
      encryptedCredentials: encrypted,
      expiresAt: this.calculateExpiration(credentials),
      createdAt: new Date()
    });
  }

  async getCredentials(platform: SocialPlatform, personaId:
string): Promise<PlatformCredentials> {
    const stored = await
this.database.getCredentials(platform, personaId);

    if (!stored || stored.expiresAt < new Date()) {
      throw new Error('Credentials expired or not found');
    }

    const decrypted = await
this.encryptionService.decrypt(stored.encryptedCredentials)
;
    return JSON.parse(decrypted);
  }

  async rotateTokens(platform: SocialPlatform, personaId:
string): Promise<void> {
    // Platform-specific token refresh logic
    const refreshResult = await
this.refreshPlatformToken(platform, personaId);

    if (refreshResult.success) {
      await this.storeCredentials(platform,
refreshResult.newCredentials);
      await this.logSecurityEvent('token_rotated',
{ platform, personaId });
    } else {
```

```
      await this.alertTokenRefreshFailure(platform,
personaId, refreshResult.error);
    }
  }
}
```

Monitoring & Analytics

API Performance Monitoring

```
interface APIMetrics {
  platform: SocialPlatform;
  personaId: string;

  // Performance metrics
  responseTime: number;
  successRate: number;
  errorRate: number;

  // Usage metrics
  apiCallsToday: number;
  rateLimitHits: number;
  contentPublished: number;

  // Quality metrics
  contentApprovalRate: number;
  engagementPerformance: number;

  // Cost metrics
  apiCostEstimate: number;
}

class APIMonitoringService {
  async collectMetrics(): Promise<PlatformMetrics[]> {
    const platforms = await this.getActivePlatforms();
    const metricsPromises = platforms.map(platform =>
this.collectPlatformMetrics(platform));

    return Promise.all(metricsPromises);
  }
```

```
  async generateDailyReport():
Promise<APIPerformanceReport> {
    const metrics = await this.collectMetrics();

    return {
      date: new Date().toISOString().split('T')[0],
      totalApiCalls: metrics.reduce((sum, m) => sum +
m.apiCallsToday, 0),
      averageResponseTime:
this.calculateAverage(metrics.map(m => m.responseTime)),
      overallSuccessRate:
this.calculateAverage(metrics.map(m => m.successRate)),
      platformBreakdown: metrics,
      recommendations: await
this.generateRecommendations(metrics)
    };
  }
}
```

This comprehensive API specification provides the foundation for integrating with all major social media platforms while maintaining security, performance, and persona authenticity across the entire system.