🔧 DOKUMENTACJA TECHNICZNA: SOCIAL MEDIA PLANNER

Kompletny Przewodnik API dla Wszystkich Platform 2026

📋 SPIS TREŚCI

---

1. META PLATFORMS (Facebook, Instagram, Threads) {#meta-platforms}

📱 INSTAGRAM GRAPH API

Podstawowe Informacje

- **Base URL**: `https://graph.instagram.com/`
- **Dokumentacja**: https://developers.facebook.com/docs/instagram-api/
- **Typ Autoryzacji**: OAuth 2.0
- **Wymagania**: Instagram Business lub Creator Account + Facebook Page

Setup & Autoryzacja

Krok 1: Stwórz Facebook App

1. Przejdź do: https://developers.facebook.com/apps
2. Utwórz nową aplikację
3. Dodaj produkty: "Facebook Login" + "Instagram Graph API"
4. Skonfiguruj OAuth Redirect URIs

Krok 2: Uzyskaj Access Token

javascript

```javascript
// Authorization URL
const authUrl = `https://www.facebook.com/v19.0/dialog/oauth?
  client_id=${APP_ID}&
  redirect_uri=${REDIRECT_URI}&
  scope=instagram_basic,instagram_content_publish,pages_show_list,pages_read_engagement&
  response_type=code`;

// Exchange code for token
const tokenResponse = await fetch(
  `https://graph.facebook.com/v19.0/oauth/access_token?
    client_id=${APP_ID}&
    client_secret=${APP_SECRET}&
    redirect_uri=${REDIRECT_URI}&
    code=${CODE}`
);

// Zamień na long-lived token (60 dni)
const longLivedToken = await fetch(
  `https://graph.facebook.com/v19.0/oauth/access_token?
    grant_type=fb_exchange_token&
    client_id=${APP_ID}&
    client_secret=${APP_SECRET}&
    fb_exchange_token=${SHORT_LIVED_TOKEN}`
);
```

Publikowanie Treści

Feed Post (Image)

javascript

```javascript
// Krok 1: Utwórz media container
const createContainer = await fetch(
  `https://graph.instagram.com/v19.0/${IG_USER_ID}/media`,
  {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({
      image_url: 'https://example.com/image.jpg',
      caption: 'Your caption here #hashtags',
      access_token: ACCESS_TOKEN
    })
  }
);

const { id: containerId } = await createContainer.json();

// Krok 2: Opublikuj container
const publish = await fetch(
  `https://graph.instagram.com/v19.0/${IG_USER_ID}/media_publish`,
  {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({
      creation_id: containerId,
      access_token: ACCESS_TOKEN
    })
  }
);
```

Carousel Post

javascript

```javascript
// Krok 1: Utwórz kontenery dla każdego item
const item1 = await createMediaItem(imageUrl1);
const item2 = await createMediaItem(imageUrl2);

// Krok 2: Utwórz carousel container
const carousel = await fetch(
  `https://graph.instagram.com/v19.0/${IG_USER_ID}/media`,
  {
    method: 'POST',
    body: JSON.stringify({
      media_type: 'CAROUSEL',
      children: [item1.id, item2.id],
      caption: 'Carousel caption',
      access_token: ACCESS_TOKEN
    })
  }
);

// Krok 3: Opublikuj
await publishMedia(carousel.id);
```

Reels

```javascript
const reelContainer = await fetch(
  `https://graph.instagram.com/v19.0/${IG_USER_ID}/media`,
  {
    method: 'POST',
    body: JSON.stringify({
      media_type: 'REELS',
      video_url: 'https://example.com/video.mp4',
      caption: 'Reel caption',
      share_to_feed: true, // Opcjonalne
      access_token: ACCESS_TOKEN
    })
  }
);

// Następnie publish jak zwykle
```

Stories (Od 2023 dostępne)

```javascript
```

```
const storyContainer = await fetch(
  `https://graph.instagram.com/v19.0/${IG_USER_ID}/media`,
  {
    method: 'POST',
    body: JSON.stringify({
      media_type: 'STORIES',
      image_url: 'https://example.com/story.jpg', // lub video_url
      access_token: ACCESS_TOKEN
    })
  }
);
```

Rate Limits

- 200 API calls per user per hour
- 4800 API calls per app per day
- Media upload: Max 25 photos/videos per request
- Carousel: Max 10 items

Media Requirements

javascript

```javascript
const mediaSpecs = {
  images: {
    formats: ['JPG', 'PNG'],
    aspectRatio: '4:5, 1.91:1, 1:1',
    minWidth: 320,
    maxWidth: 1440,
    maxSize: '8MB'
  },
  videos: {
    formats: ['MP4', 'MOV'],
    codec: 'H.264',
    aspectRatio: '4:5, 16:9, 1:1',
    minDuration: '3s',
    maxDuration: '60s (feed), 90s (reels)',
    maxSize: '100MB (feed), 1GB (reels)',
    fps: '23-60',
    bitrate: 'max 5 Mbps'
  },
  reels: {
    duration: '90s max',
    resolution: '1080x1920 (recommended)',
    aspectRatio: '9:16'
  }
};
```

Získávání Insights

```javascript
// Post insights
const insights = await fetch(
  `https://graph.instagram.com/${MEDIA_ID}/insights?
    metric=engagement,impressions,reach,saved&
    access_token=${ACCESS_TOKEN}`
);

// Account insights
const accountInsights = await fetch(
  `https://graph.instagram.com/${IG_USER_ID}/insights?
    metric=follower_count,impressions,reach&
    period=day&
    access_token=${ACCESS_TOKEN}`
);
```

Error Handling

javascript

```javascript
const errorCodes = {
  100: 'Invalid parameter',
  190: 'Access token expired - refresh needed',
  200: 'Permission denied',
  368: 'Temporarily blocked for spam',
  9001: 'Media upload failed',
  9004: 'Media type not allowed'
};

// Auto-retry logic
async function postWithRetry(data, maxRetries = 3) {
  for (let i = 0; i < maxRetries; i++) {
    try {
      return await postToInstagram(data);
    } catch (error) {
      if (error.code === 190) {
        await refreshAccessToken();
      } else if (error.code === 368) {
        await delay(3600000); // Wait 1 hour
      } else {
        throw error;
      }
    }
  }
}
```

📘 FACEBOOK PAGES API

Publikowanie na Page

javascript

```
// Post with image
const fbPost = await fetch(
  `https://graph.facebook.com/v19.0/${PAGE_ID}/photos`,
  {
    method: 'POST',
    body: JSON.stringify({
      url: 'https://example.com/image.jpg',
      caption: 'Post caption',
      published: true, // false = draft
      scheduled_publish_time: 1234567890, // Unix timestamp (opcjonalne)
      access_token: PAGE_ACCESS_TOKEN
    })
  }
);

// Video post
const videoPost = await fetch(
  `https://graph.facebook.com/v19.0/${PAGE_ID}/videos`,
  {
    method: 'POST',
    body: formData, // multipart/form-data with video file
    params: {
      description: 'Video description',
      published: true,
      access_token: PAGE_ACCESS_TOKEN
    }
  }
);

// Reels
const reels = await fetch(
  `https://graph.facebook.com/v19.0/${PAGE_ID}/video_reels`,
  {
    method: 'POST',
    body: {
      upload_phase: 'start',
      access_token: PAGE_ACCESS_TOKEN
    }
  }
);
```

Rate Limits Facebook

- 200 calls per hour per user
- 4800 calls per day per app
- Page posts: unlimited (w ramach rate limits)

🧵 THREADS API (Meta)

Informacje Podstawowe

- **Base URL**: https://graph.threads.net/v1.0/
- **Status**: Limited Beta (2026)
- **Wymagania**: Threads Business Account

```javascript
// Publikowanie na Threads
const threadsPost = await fetch(
  `https://graph.threads.net/v1.0/${THREADS_USER_ID}/threads`,
  {
    method: 'POST',
    body: JSON.stringify({
      text: 'Post content (max 500 chars)',
      media_url: 'https://example.com/image.jpg', // opcjonalne
      access_token: ACCESS_TOKEN
    })
  }
);
```

---

2. TIKTOK API {#tiktok-api}

🎵 TIKTOK CONTENT POSTING API

Podstawowe Informacje

- **Base URL**: https://open.tiktokapis.com/
- **Dokumentacja**: https://developers.tiktok.com/doc/
- **Autoryzacja**: OAuth 2.0
- **Review Required**: TAK - aplikacja musi przejść audyt

Setup & Autoryzacja

Krok 1: Rejestracja App

1. Przejdź do: https://developers.tiktok.com/
2. Utwórz aplikację
3. Dodaj produkt: "Content Posting API"
4. Skonfiguruj redirect URI
5. Wypełnij formularz audytu (OBOWIĄZKOWE)

Krok 2: OAuth Flow

javascript

```javascript
// Authorization URL
const authUrl = `https://www.tiktok.com/v2/auth/authorize/?
  client_key=${CLIENT_KEY}&
  redirect_uri=${REDIRECT_URI}&
  response_type=code&
  scope=user.info.basic,video.list,video.upload`;

// Get Access Token
const tokenResponse = await fetch(
  'https://open.tiktokapis.com/v2/oauth/token/',
  {
    method: 'POST',
    headers: { 'Content-Type': 'application/x-www-form-urlencoded' },
    body: new URLSearchParams({
      client_key: CLIENT_KEY,
      client_secret: CLIENT_SECRET,
      code: CODE,
      grant_type: 'authorization_code',
      redirect_uri: REDIRECT_URI
    })
  }
);
```

Publikowanie Video

Metoda 1: File Upload

javascript

```
// Krok 1: Inicjalizacja uploadu
const initUpload = await fetch(
  'https://open.tiktokapis.com/v2/post/publish/video/init/',
  {
    method: 'POST',
    headers: {
      'Authorization': `Bearer ${ACCESS_TOKEN}`,
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({
      post_info: {
        title: 'Video title with #hashtags',
        privacy_level: 'PUBLIC_TO_EVERYONE', // lub MUTUAL_FOLLOW_FRIENDS, SELF_ONLY
        disable_duet: false,
        disable_comment: false,
        disable_stitch: false,
        video_cover_timestamp_ms: 1000
      },
      source_info: {
        source: 'FILE_UPLOAD',
        video_size: 50000123, // bytes
        chunk_size: 10000000,
        total_chunk_count: 5
      }
    })
  }
);

const { publish_id, upload_url } = await initUpload.json();

// Krok 2: Upload video do TikTok
const uploadVideo = await fetch(upload_url, {
  method: 'PUT',
  headers: { 'Content-Type': 'video/mp4' },
  body: videoFileBuffer
});

// Krok 3: Sprawdź status
const checkStatus = await fetch(
  'https://open.tiktokapis.com/v2/post/publish/status/fetch/',
  {
    method: 'POST',
    headers: {
      'Authorization': `Bearer ${ACCESS_TOKEN}`,
      'Content-Type': 'application/json'
    },
```

```
    body: JSON.stringify({ publish_id })
  }
);
```

Metoda 2: URL Pull

javascript

```
const initFromUrl = await fetch(
  'https://open.tiktokapis.com/v2/post/publish/video/init/',
  {
    method: 'POST',
    headers: {
      'Authorization': `Bearer ${ACCESS_TOKEN}`,
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({
      post_info: {
        title: 'Video title',
        privacy_level: 'PUBLIC_TO_EVERYONE'
      },
      source_info: {
        source: 'PULL_FROM_URL',
        video_url: 'https://your-verified-domain.com/video.mp4'
      }
    })
  }
);
```

Publikowanie Photos (Carousel)

javascript

```
const photoPost = await fetch(
  'https://open.tiktokapis.com/v2/post/publish/content/init/',
  {
    method: 'POST',
    headers: {
      'Authorization': `Bearer ${ACCESS_TOKEN}`,
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({
      post_info: {
        title: 'Photo post caption',
        privacy_level: 'PUBLIC_TO_EVERYONE',
        disable_comment: false
      },
      source_info: {
        source: 'PULL_FROM_URL',
        photo_cover_index: 0,
        photo_images: [
          'https://your-domain.com/photo1.jpg',
          'https://your-domain.com/photo2.jpg'
        ]
      },
      post_mode: 'DIRECT_POST',
      media_type: 'PHOTO'
    })
  }
);
```

Rate Limits

- 6 requests per minute per user (video init)
- 30 requests per minute per user (status check)
- Daily limit: varies by partnership tier

Media Requirements

javascript

```
const tiktokSpecs = {
  video: {
    formats: ['MP4', 'MOV', 'WEBM'],
    codec: 'H.264 or H.265',
    resolution: 'min 720p, recommended 1080p',
    aspectRatio: '9:16 (vertical), 16:9 (horizontal), 1:1 (square)',
    duration: 'min 3s, max 10min',
    maxSize: '4GB',
    fps: '23-60',
    bitrate: 'recommended 16Mbps'
  },
  photos: {
    formats: ['JPG', 'JPEG', 'PNG', 'WEBP'],
    minResolution: '1080x1080',
    maxSize: '10MB per image',
    maxCount: '35 photos per post'
  }
};
```

Webhook Events

javascript

```
// Skonfiguruj webhook w developer portal
const webhookHandler = (req, res) => {
  const event = req.body;

  if (event.event === 'video.publish.complete') {
    const { publish_id, status, post_id } = event.data;
    // Update database
    updatePostStatus(publish_id, status, post_id);
  }

  if (event.event === 'video.publish.failed') {
    const { publish_id, fail_reason } = event.data;
    // Handle failure
    handleFailure(publish_id, fail_reason);
  }
};
```

Ważne Uwagi TikTok

⚠️ CRITICAL:

1. Przed audytem wszystkie posty są PRIVATE
2. Musisz zweryfikować domenę dla URL pulls
3. Commercial content disclosure OBOWIĄZKOWY
4. Music usage confirmation WYMAGANY
5. Privacy level nie może być default - user musi wybierać

---

3. LINKEDIN API {#linkedin-api}

💼 LINKEDIN POSTS API

Podstawowe Informacje

- **Base URL**: `https://api.linkedin.com/`
- **Dokumentacja**: https://learn.microsoft.com/en-us/linkedin/
- **Autoryzacja**: OAuth 2.0
- **Wymagania**: LinkedIn Page (dla organization posts)

Setup & Autoryzacja

```javascript
```

```javascript
// Authorization URL
const authUrl = `https://www.linkedin.com/oauth/v2/authorization?
  response_type=code&
  client_id=${CLIENT_ID}&
  redirect_uri=${REDIRECT_URI}&
  state=${RANDOM_STRING}&
  scope=w_member_social,r_liteprofile,w_organization_social`;

// Get Access Token
const tokenResponse = await fetch(
  'https://www.linkedin.com/oauth/v2/accessToken',
  {
    method: 'POST',
    headers: { 'Content-Type': 'application/x-www-form-urlencoded' },
    body: new URLSearchParams({
      grant_type: 'authorization_code',
      code: CODE,
      client_id: CLIENT_ID,
      client_secret: CLIENT_SECRET,
      redirect_uri: REDIRECT_URI
    })
  }
);

// Token ma 60 dni ważności
```

Publikowanie Postów

Text Post (Personal Profile)

javascript

```javascript
const textPost = await fetch(
  'https://api.linkedin.com/rest/posts',
  {
    method: 'POST',
    headers: {
      'Authorization': `Bearer ${ACCESS_TOKEN}`,
      'X-Restli-Protocol-Version': '2.0.0',
      'LinkedIn-Version': '202601', // Format YYYYMM
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({
      author: `urn:li:person:${PERSON_ID}`,
      commentary: 'Your post text here',
      visibility: 'PUBLIC',
      distribution: {
        feedDistribution: 'MAIN_FEED',
        targetEntities: [],
        thirdPartyDistributionChannels: []
      },
      lifecycleState: 'PUBLISHED',
      isReshareDisabledByAuthor: false
    })
  }
);
```

Post with Image

javascript

```
// Krok 1: Upload image
const registerUpload = await fetch(
  'https://api.linkedin.com/rest/images?action=initializeUpload',
  {
    method: 'POST',
    headers: {
      'Authorization': `Bearer ${ACCESS_TOKEN}`,
      'X-Restli-Protocol-Version': '2.0.0',
      'LinkedIn-Version': '202601'
    },
    body: JSON.stringify({
      initializeUploadRequest: {
        owner: `urn:li:person:${PERSON_ID}`
      }
    })
  }
);

const { uploadUrl, image } = await registerUpload.json();

// Krok 2: Upload binary
await fetch(uploadUrl, {
  method: 'PUT',
  headers: { 'Content-Type': 'image/jpeg' },
  body: imageBuffer
});

// Krok 3: Create post with image
const imagePost = await fetch(
  'https://api.linkedin.com/rest/posts',
  {
    method: 'POST',
    headers: {
      'Authorization': `Bearer ${ACCESS_TOKEN}`,
      'X-Restli-Protocol-Version': '2.0.0',
      'LinkedIn-Version': '202601',
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({
      author: `urn:li:person:${PERSON_ID}`,
      commentary: 'Post with image',
      visibility: 'PUBLIC',
      distribution: {
        feedDistribution: 'MAIN_FEED'
      },
      content: {
```

```
      media: {
        title: 'Image title',
        id: image // URN z kroku 1
      }
    },
    lifecycleState: 'PUBLISHED'
  })
 }
);
```

Video Post

javascript

```
// Podobny flow jak image, ale używasz /videos endpoint
const registerVideo = await fetch(
 'https://api.linkedin.com/rest/videos?action=initializeUpload',
 {
  method: 'POST',
  headers: {
   'Authorization': `Bearer ${ACCESS_TOKEN}`,
   'X-Restli-Protocol-Version': '2.0.0'
  },
  body: JSON.stringify({
   initializeUploadRequest: {
    owner: `urn:li:person:${PERSON_ID}`,
    fileSizeBytes: videoSize,
    uploadCaptions: false,
    uploadThumbnail: false
   }
  })
 }
);

// Upload video chunks
// Potem create post z video URN
```

Article Post

javascript

```
const articlePost = await fetch(
  'https://api.linkedin.com/rest/posts',
  {
    method: 'POST',
    headers: {
      'Authorization': `Bearer ${ACCESS_TOKEN}`,
      'X-Restli-Protocol-Version': '2.0.0',
      'LinkedIn-Version': '202601',
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({
      author: `urn:li:person:${PERSON_ID}`,
      commentary: 'Check out this article',
      visibility: 'PUBLIC',
      distribution: {
        feedDistribution: 'MAIN_FEED'
      },
      content: {
        article: {
          source: 'https://example.com/article',
          thumbnail: 'urn:li:image:...',
          title: 'Article Title',
          description: 'Article description'
        }
      },
      lifecycleState: 'PUBLISHED'
    })
  }
);
```

Organization Page Post

javascript

```javascript
// Najpierw potrzebujesz w_organization_social scope
// I approval dla "Marketing Developer Platform"

const orgPost = await fetch(
  'https://api.linkedin.com/rest/posts',
  {
    method: 'POST',
    headers: {
      'Authorization': `Bearer ${ACCESS_TOKEN}`,
      'X-Restli-Protocol-Version': '2.0.0',
      'LinkedIn-Version': '202601',
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({
      author: `urn:li:organization:${ORG_ID}`,
      commentary: 'Company post',
      visibility: 'PUBLIC',
      distribution: {
        feedDistribution: 'MAIN_FEED'
      },
      lifecycleState: 'PUBLISHED'
    })
  }
);
```

Rate Limits

- Różne limity per partnership tier
- Standardowo: ~100 calls per user per day
- Media uploads liczą się osobno

Media Specs

javascript

```javascript
const linkedinSpecs = {
  images: {
    formats: ['JPG', 'PNG', 'GIF'],
    maxSize: '10MB',
    minDimensions: '200x200',
    maxDimensions: '7680x4320'
  },
  videos: {
    formats: ['MP4', 'MOV'],
    codec: 'H.264',
    maxSize: '5GB',
    maxDuration: '10 minutes',
    aspectRatio: '16:9, 1:1, 9:16',
    fps: '10-60'
  }
};
```

Uzyskanie Person/Org ID

```javascript
// Get Person ID
const profile = await fetch(
  'https://api.linkedin.com/v2/userinfo',
  {
    headers: {
      'Authorization': `Bearer ${ACCESS_TOKEN}`
    }
  }
);
// Returns sub which is person ID

// Get Organization ID
const orgs = await fetch(
  'https://api.linkedin.com/v2/organizationAcls?q=roleAssignee&projection=(elements*(organization~))',
  {
    headers: {
      'Authorization': `Bearer ${ACCESS_TOKEN}`
    }
  }
);
```

---

4. TWITTER/X API {#twitter-api}

X (TWITTER) API v2

Podstawowe Informacje

- **Base URL**: `https://api.twitter.com/2/`
- **Dokumentacja**: https://developer.twitter.com/en/docs
- **Autoryzacja**: OAuth 2.0 lub OAuth 1.0a
- **Pricing**: Free tier bardzo ograniczony, Basic $100/mies, Pro $5000/mies

Setup & Autoryzacja (OAuth 2.0)

```javascript
```

```javascript
// OAuth 2.0 PKCE Flow (recommended)
const crypto = require('crypto');

// Krok 1: Generate code verifier & challenge
const codeVerifier = crypto.randomBytes(32).toString('base64url');
const codeChallenge = crypto
  .createHash('sha256')
  .update(codeVerifier)
  .digest('base64url');

// Krok 2: Authorization URL
const authUrl = `https://twitter.com/i/oauth2/authorize?
  response_type=code&
  client_id=${CLIENT_ID}&
  redirect_uri=${REDIRECT_URI}&
  scope=tweet.read tweet.write users.read offline.access&
  state=${STATE}&
  code_challenge=${codeChallenge}&
  code_challenge_method=S256`;

// Krok 3: Exchange code for token
const tokenResponse = await fetch(
  'https://api.twitter.com/2/oauth2/token',
  {
    method: 'POST',
    headers: {
      'Content-Type': 'application/x-www-form-urlencoded',
      'Authorization': `Basic ${btoa(`${CLIENT_ID}:${CLIENT_SECRET}`)}`
    },
    body: new URLSearchParams({
      code: CODE,
      grant_type: 'authorization_code',
      client_id: CLIENT_ID,
      redirect_uri: REDIRECT_URI,
      code_verifier: codeVerifier
    })
  }
);

// Response includes refresh_token (użyj do odświeżania)
```

Publikowanie Tweetów

Text Tweet

javascript

```
const tweet = await fetch(
  'https://api.twitter.com/2/tweets',
  {
    method: 'POST',
    headers: {
      'Authorization': `Bearer ${ACCESS_TOKEN}`,
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({
      text: 'Your tweet here (max 280 chars for free, 4000 for Premium)'
    })
  }
);
```

Tweet with Media

javascript

```javascript
// Krok 1: Upload media (wymaga OAuth 1.0a)
const FormData = require('form-data');
const formData = new FormData();
formData.append('media', imageBuffer, { filename: 'image.jpg' });

const mediaUpload = await fetch(
  'https://upload.twitter.com/1.1/media/upload.json',
  {
    method: 'POST',
    headers: {
      'Authorization': generateOAuth1Header('POST', uploadUrl, {})
    },
    body: formData
  }
);

const { media_id_string } = await mediaUpload.json();

// Krok 2: Tweet with media_id
const tweetWithMedia = await fetch(
  'https://api.twitter.com/2/tweets',
  {
    method: 'POST',
    headers: {
      'Authorization': `Bearer ${ACCESS_TOKEN}`,
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({
      text: 'Tweet with image',
      media: {
        media_ids: [media_id_string]
      }
    })
  }
);
```

Thread (Multiple Tweets)

javascript

```javascript
// Pierwszy tweet
const tweet1 = await createTweet({ text: 'First tweet in thread' });

// Kolejne tweety z reply_to
const tweet2 = await createTweet({
  text: 'Second tweet',
  reply: {
    in_reply_to_tweet_id: tweet1.data.id
  }
});

const tweet3 = await createTweet({
  text: 'Third tweet',
  reply: {
    in_reply_to_tweet_id: tweet2.data.id
  }
});
```

Poll

```javascript
const poll = await fetch(
  'https://api.twitter.com/2/tweets',
  {
    method: 'POST',
    headers: {
      'Authorization': `Bearer ${ACCESS_TOKEN}`,
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({
      text: 'Question here?',
      poll: {
        options: ['Option 1', 'Option 2', 'Option 3', 'Option 4'],
        duration_minutes: 1440 // 1-10080 (7 days)
      }
    })
  }
);
```

Rate Limits (2026 API v2)

FREE TIER:
- 1500 tweets/month CREATE limit
- 10,000 tweets/month READ limit

BASIC ($100/month):
- 3000 tweets/month
- 50,000 reads/month

PRO ($5000/month):
- 300,000 tweets/month
- 1,000,000 reads/month

Media Specs

```javascript
const twitterSpecs = {
  images: {
    formats: ['JPG', 'PNG', 'GIF', 'WEBP'],
    maxSize: '5MB (photos), 15MB (GIFs)',
    maxCount: 4
  },
  videos: {
    formats: ['MP4', 'MOV'],
    maxSize: '512MB',
    maxDuration: '140s (2:20)',
    aspectRatio: '1:1, 16:9',
    frameRate: '40 fps max',
    resolution: '1920x1200 max'
  }
};
```

---

5. YOUTUBE API {#youtube-api}

📺 YOUTUBE DATA API v3

Podstawowe Informacje

- **Base URL**: `https://www.googleapis.com/youtube/v3/`
- **Dokumentacja**: https://developers.google.com/youtube/v3
- **Autoryzacja**: OAuth 2.0
- **Quota**: 10,000 units/day (upload = 1600 units)

Setup Google Cloud Project

1. https://console.cloud.google.com/
2. Utwórz projekt
3. Włącz "YouTube Data API v3"
4. Utwórz OAuth 2.0 credentials
5. Dodaj scopes: youtube.upload, youtube.force-ssl

Autoryzacja

javascript

```javascript
const { google } = require('googleapis');

const oauth2Client = new google.auth.OAuth2(
  CLIENT_ID,
  CLIENT_SECRET,
  REDIRECT_URI
);

// Authorization URL
const authUrl = oauth2Client.generateAuthUrl({
  access_type: 'offline',
  scope: [
    'https://www.googleapis.com/auth/youtube.upload',
    'https://www.googleapis.com/auth/youtube.force-ssl'
  ]
});

// Exchange code for token
const { tokens } = await oauth2Client.getToken(CODE);
oauth2Client.setCredentials(tokens);
```

Upload Video

javascript

```javascript
const youtube = google.youtube({ version: 'v3', auth: oauth2Client });

const videoUpload = await youtube.videos.insert({
  part: 'snippet,status',
  requestBody: {
    snippet: {
      title: 'Video Title',
      description: 'Video description',
      tags: ['tag1', 'tag2'],
      categoryId: '22', // People & Blogs
      defaultLanguage: 'en',
      defaultAudioLanguage: 'en'
    },
    status: {
      privacyStatus: 'public', // public, unlisted, private
      selfDeclaredMadeForKids: false,
      publishAt: '2026-03-01T10:00:00Z' // Schedule (ISO 8601)
    }
  },
  media: {
    body: fs.createReadStream('video.mp4')
  }
});

console.log('Video ID:', videoUpload.data.id);
```

Community Post (Shorts)

```javascript
```

```javascript
// YouTube Shorts API (Beta 2026)
const short = await youtube.videos.insert({
  part: 'snippet,status',
  requestBody: {
    snippet: {
      title: 'Short Title #shorts',
      description: 'Short description',
      tags: ['shorts']
    },
    status: {
      privacyStatus: 'public',
      selfDeclaredMadeForKids: false
    }
  },
  media: {
    body: fs.createReadStream('vertical-video.mp4')
  }
});
```

Rate Limits & Quota

Daily Quota: 10,000 units

Quota Costs:
- Upload video: 1,600 units
- Update video: 50 units
- Delete video: 50 units
- List videos: 1 unit

⚠️ To znaczy ~6 uploads/day na darmowym limicie

Video Specs

javascript

```javascript
const youtubeSpecs = {
  video: {
    formats: ['MP4', 'MOV', 'AVI', 'WMV', 'FLV', '3GPP', 'MPEG-PS', 'WebM'],
    maxSize: '256GB or 12 hours',
    resolution: 'up to 8K (7680x4320)',
    fps: '24, 25, 30, 48, 50, 60',
    codec: 'H.264, MPEG-2, MPEG-4'
  },
  shorts: {
    duration: 'max 60s',
    aspectRatio: '9:16 (vertical)',
    resolution: '1080x1920 recommended'
  }
};
```

---

6. PINTEREST API {#pinterest-api}

📌 PINTEREST API v5

Podstawowe Info

- **Base URL**: `https://api.pinterest.com/v5/`
- **Dokumentacja**: https://developers.pinterest.com/docs/
- **Autoryzacja**: OAuth 2.0

Autoryzacja

```javascript
```

```
const authUrl = `https://www.pinterest.com/oauth/?
  client_id=${CLIENT_ID}&
  redirect_uri=${REDIRECT_URI}&
  response_type=code&
  scope=boards:read,boards:write,pins:read,pins:write`;

// Get token
const tokenResponse = await fetch(
  'https://api.pinterest.com/v5/oauth/token',
  {
    method: 'POST',
    headers: {
      'Authorization': `Basic ${btoa(`${CLIENT_ID}:${CLIENT_SECRET}`)}`,
      'Content-Type': 'application/x-www-form-urlencoded'
    },
    body: new URLSearchParams({
      grant_type: 'authorization_code',
      code: CODE,
      redirect_uri: REDIRECT_URI
    })
  }
);
```

Create Pin

javascript

```
const pin = await fetch(
  'https://api.pinterest.com/v5/pins',
  {
    method: 'POST',
    headers: {
      'Authorization': `Bearer ${ACCESS_TOKEN}`,
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({
      board_id: BOARD_ID,
      title: 'Pin Title',
      description: 'Pin description',
      link: 'https://example.com',
      media_source: {
        source_type: 'image_url',
        url: 'https://example.com/image.jpg'
      }
    })
  }
);

// Video pin
const videoPin = await fetch(
  'https://api.pinterest.com/v5/pins',
  {
    method: 'POST',
    headers: {
      'Authorization': `Bearer ${ACCESS_TOKEN}`,
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({
      board_id: BOARD_ID,
      title: 'Video Pin',
      media_source: {
        source_type: 'video_url',
        url: 'https://example.com/video.mp4',
        cover_image_url: 'https://example.com/cover.jpg'
      }
    })
  }
);
```

Rate Limits

- 300 requests per hour per user
- 1000 requests per day per user

---

7. GOOGLE BUSINESS PROFILE API {#google-business-api}

🏢 GOOGLE MY BUSINESS API

Base Info

- **Base URL**: `https://mybusinessbusinessinformation.googleapis.com/v1/`
- **Docs**: https://developers.google.com/my-business

Create Local Post

javascript

```
const localPost = await fetch(
  `https://mybusiness.googleapis.com/v4/accounts/${ACCOUNT_ID}/locations/${LOCATION_ID}/localPosts`,
  {
    method: 'POST',
    headers: {
      'Authorization': `Bearer ${ACCESS_TOKEN}`,
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({
      languageCode: 'en-US',
      summary: 'Post summary',
      media: [{
        mediaFormat: 'PHOTO',
        sourceUrl: 'https://example.com/image.jpg'
      }],
      topicType: 'STANDARD', // STANDARD, EVENT, OFFER, ALERT
      callToAction: {
        actionType: 'LEARN_MORE',
        url: 'https://example.com'
      }
    })
  }
);
```

---

8. TELEGRAM API {#telegram-api}

✈️ TELEGRAM BOT API

Base Info

- **Base URL**: `https://api.telegram.org/bot<TOKEN>/`
- **Docs**: https://core.telegram.org/bots/api

Send Message to Channel

```javascript
```

```
const message = await fetch(
  `https://api.telegram.org/bot${BOT_TOKEN}/sendMessage`,
  {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({
      chat_id: '@your_channel',
      text: 'Message text',
      parse_mode: 'HTML' // lub 'Markdown'
    })
  }
);

// Photo
const photo = await fetch(
  `https://api.telegram.org/bot${BOT_TOKEN}/sendPhoto`,
  {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({
      chat_id: '@your_channel',
      photo: 'https://example.com/image.jpg',
      caption: 'Photo caption'
    })
  }
);

// Video
const video = await fetch(
  `https://api.telegram.org/bot${BOT_TOKEN}/sendVideo`,
  {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({
      chat_id: '@your_channel',
      video: 'https://example.com/video.mp4',
      caption: 'Video caption'
    })
  }
);
```

Rate Limits

- 30 messages per second to different chats
- 20 messages per minute to same chat
- File upload: 50MB max

---

9. REDDIT API {#reddit-api}

🔴 REDDIT API

Base Info

- **Base URL**: `https://oauth.reddit.com/`
- **Docs**: https://www.reddit.com/dev/api

OAuth Flow

```javascript
// Get token (application-only)
const token = await fetch(
  'https://www.reddit.com/api/v1/access_token',
  {
    method: 'POST',
    headers: {
      'Authorization': `Basic ${btoa(`${CLIENT_ID}:${CLIENT_SECRET}`)}`,
      'Content-Type': 'application/x-www-form-urlencoded'
    },
    body: 'grant_type=password&username=USERNAME&password=PASSWORD'
  }
);
```

Submit Post

```javascript
```

```
// Text post
const post = await fetch(
  'https://oauth.reddit.com/api/submit',
  {
    method: 'POST',
    headers: {
      'Authorization': `Bearer ${ACCESS_TOKEN}`,
      'Content-Type': 'application/x-www-form-urlencoded',
      'User-Agent': 'YourApp/1.0'
    },
    body: new URLSearchParams({
      sr: 'subreddit_name',
      kind: 'self',
      title: 'Post title',
      text: 'Post body (Markdown supported)',
      sendreplies: true
    })
  }
);

// Link post
const linkPost = await fetch(
  'https://oauth.reddit.com/api/submit',
  {
    method: 'POST',
    headers: {
      'Authorization': `Bearer ${ACCESS_TOKEN}`,
      'Content-Type': 'application/x-www-form-urlencoded',
      'User-Agent': 'YourApp/1.0'
    },
    body: new URLSearchParams({
      sr: 'subreddit_name',
      kind: 'link',
      title: 'Post title',
      url: 'https://example.com'
    })
  }
);

// Image/video - wymaga uploadu do Reddit media server
```

Rate Limits

- 60 requests per minute
- 600 requests per 10 minutes
- Bardzo strict na spam

---

10. BLUESKY API {#bluesky-api}

🦋 BLUESKY AT PROTOCOL

Base Info

- **Base URL**: `https://bsky.social/xrpc/`
- **Docs**: https://atproto.com/

Autoryzacja

```javascript
const session = await fetch(
  'https://bsky.social/xrpc/com.atproto.server.createSession',
  {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({
      identifier: 'username.bsky.social',
      password: 'app-password' // Generate in settings
    })
  }
);

const { accessJwt, did } = await session.json();
```

Create Post

```javascript
```

```
const post = await fetch(
  'https://bsky.social/xrpc/com.atproto.repo.createRecord',
  {
    method: 'POST',
    headers: {
      'Authorization': `Bearer ${accessJwt}`,
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({
      repo: did,
      collection: 'app.bsky.feed.post',
      record: {
        text: 'Post text (300 chars max)',
        createdAt: new Date().toISOString(),
        $type: 'app.bsky.feed.post'
      }
    })
  }
);
```

Post with Image

javascript

```
// Upload blob
const blob = await fetch(
  'https://bsky.social/xrpc/com.atproto.repo.uploadBlob',
  {
    method: 'POST',
    headers: {
      'Authorization': `Bearer ${accessJwt}`,
      'Content-Type': 'image/jpeg'
    },
    body: imageBuffer
  }
);

const blobData = await blob.json();

// Create post with embed
const imagePost = await fetch(
  'https://bsky.social/xrpc/com.atproto.repo.createRecord',
  {
    method: 'POST',
    headers: {
      'Authorization': `Bearer ${accessJwt}`,
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({
      repo: did,
      collection: 'app.bsky.feed.post',
      record: {
        text: 'Post with image',
        createdAt: new Date().toISOString(),
        embed: {
          $type: 'app.bsky.embed.images',
          images: [{
            image: blobData.blob,
            alt: 'Image description'
          }]
        }
      }
    })
  }
);
```

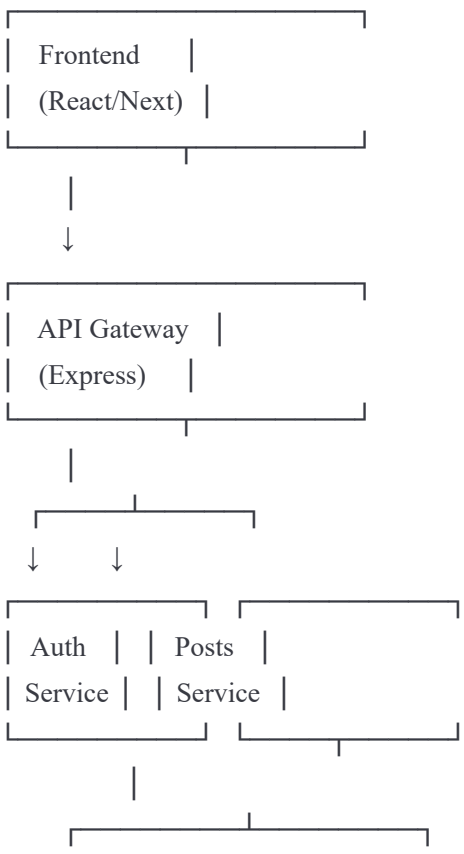
11. ARCHITEKTURA SYSTEMU {#architektura-systemu}

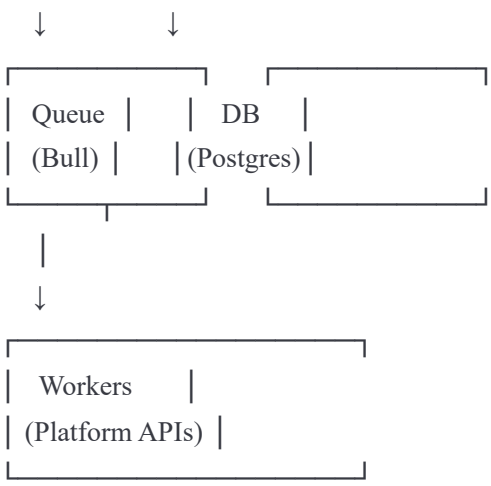
🏗️ RECOMMENDED TECH STACK

```javascript
// Backend
{
  runtime: 'Node.js 20+',
  framework: 'Express.js / Fastify',
  queue: 'BullMQ (Redis-based)',
  database: 'PostgreSQL 15+',
  cache: 'Redis 7+',
  storage: 'AWS S3 / Cloudflare R2',
  monitoring: 'Sentry + DataDog'
}

// Frontend
{
  framework: 'React 18+ / Next.js 14+',
  state: 'Zustand / Redux Toolkit',
  ui: 'shadcn/ui + Tailwind',
  calendar: 'FullCalendar / react-big-calendar'
}
```

📐 SYSTEM ARCHITECTURE





---

12. DATABASE SCHEMA {#database-schema}

sql

```sql
-- Users table
CREATE TABLE users (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  email VARCHAR(255) UNIQUE NOT NULL,
  password_hash VARCHAR(255) NOT NULL,
  created_at TIMESTAMP DEFAULT NOW(),
  updated_at TIMESTAMP DEFAULT NOW()
);

-- Social accounts (OAuth connections)
CREATE TABLE social_accounts (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  user_id UUID REFERENCES users(id) ON DELETE CASCADE,
  platform VARCHAR(50) NOT NULL, -- 'instagram', 'tiktok', etc.
  platform_user_id VARCHAR(255),
  platform_username VARCHAR(255),
  access_token TEXT,
  refresh_token TEXT,
  token_expires_at TIMESTAMP,
  scopes TEXT[], -- Array of granted scopes
  account_data JSONB, -- Additional platform-specific data
  is_active BOOLEAN DEFAULT true,
  created_at TIMESTAMP DEFAULT NOW(),
  updated_at TIMESTAMP DEFAULT NOW(),
  UNIQUE(user_id, platform, platform_user_id)
);

-- Posts table
CREATE TABLE posts (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  user_id UUID REFERENCES users(id) ON DELETE CASCADE,
  social_account_id UUID REFERENCES social_accounts(id),

  -- Content
  content_type VARCHAR(50) NOT NULL, -- 'text', 'image', 'video', 'carousel'
  caption TEXT,
  media_urls TEXT[], -- Array of media URLs

  -- Metadata
  platforms TEXT[], -- ['instagram', 'facebook']
  platform_specific_data JSONB, -- Different data per platform

  -- Scheduling
  status VARCHAR(50) DEFAULT 'draft', -- draft, scheduled, publishing, published, failed
  scheduled_at TIMESTAMP,
  published_at TIMESTAMP,
```

```
  -- Platform responses
  platform_post_ids JSONB, -- {instagram: 'id123', facebook: 'id456'}
  errors JSONB,

  created_at TIMESTAMP DEFAULT NOW(),
  updated_at TIMESTAMP DEFAULT NOW()
);

-- Media files
CREATE TABLE media (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  user_id UUID REFERENCES users(id) ON DELETE CASCADE,
  post_id UUID REFERENCES posts(id) ON DELETE SET NULL,

  filename VARCHAR(255),
  original_url TEXT,
  cdn_url TEXT,
  storage_key TEXT, -- S3/R2 key

  file_type VARCHAR(50), -- image/jpeg, video/mp4
  file_size INTEGER, -- bytes
  width INTEGER,
  height INTEGER,
  duration INTEGER, -- for videos (seconds)

  metadata JSONB,

  created_at TIMESTAMP DEFAULT NOW()
);

-- Queue jobs (dla tracking)
CREATE TABLE queue_jobs (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  post_id UUID REFERENCES posts(id),
  platform VARCHAR(50),

  job_id VARCHAR(255), -- BullMQ job ID
  status VARCHAR(50), -- waiting, active, completed, failed
  attempts INTEGER DEFAULT 0,
  max_attempts INTEGER DEFAULT 3,

  data JSONB,
  result JSONB,
  error JSONB,

  started_at TIMESTAMP,
```

```
  completed_at TIMESTAMP,
  created_at TIMESTAMP DEFAULT NOW()
);

-- Analytics (basic)
CREATE TABLE post_analytics (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  post_id UUID REFERENCES posts(id),
  platform VARCHAR(50),
  platform_post_id VARCHAR(255),

  likes INTEGER DEFAULT 0,
  comments INTEGER DEFAULT 0,
  shares INTEGER DEFAULT 0,
  views INTEGER DEFAULT 0,
  impressions INTEGER DEFAULT 0,
  reach INTEGER DEFAULT 0,

  engagement_rate DECIMAL(5,2),

  fetched_at TIMESTAMP DEFAULT NOW(),
  created_at TIMESTAMP DEFAULT NOW()
);

-- Indexes
CREATE INDEX idx_posts_user_id ON posts(user_id);
CREATE INDEX idx_posts_scheduled_at ON posts(scheduled_at);
CREATE INDEX idx_posts_status ON posts(status);
CREATE INDEX idx_social_accounts_user_platform ON social_accounts(user_id, platform);
CREATE INDEX idx_queue_jobs_post_platform ON queue_jobs(post_id, platform);
```

---

13. QUEUE MANAGEMENT {#queue-management}

🔄 BULLMQ IMPLEMENTATION

```
javascript
```

```
// queue.js
const { Queue, Worker } = require('bullmq');
const Redis = require('ioredis');

const connection = new Redis({
  host: 'localhost',
  port: 6379,
  maxRetriesPerRequest: null
});

// Create separate queues per platform
const createPlatformQueue = (platform) => {
  return new Queue(`${platform}-posts`, {
    connection,
    defaultJobOptions: {
      attempts: 3,
      backoff: {
        type: 'exponential',
        delay: 60000 // 1 minute
      },
      removeOnComplete: {
        age: 86400, // Keep for 24h
        count: 1000
      },
      removeOnFail: {
        age: 604800 // Keep failed for 7 days
      }
    }
  });
};

const instagramQueue = createPlatformQueue('instagram');
const tiktokQueue = createPlatformQueue('tiktok');
const linkedinQueue = createPlatformQueue('linkedin');
const twitterQueue = createPlatformQueue('twitter');
const facebookQueue = createPlatformQueue('facebook');

// Schedule post
async function schedulePost(postData, scheduledTime) {
  const delay = scheduledTime.getTime() - Date.now();

  for (const platform of postData.platforms) {
    const queue = getQueueForPlatform(platform);

    await queue.add(`publish-${platform}`, {
      postId: postData.id,
```

```
      userId: postData.userId,
      content: postData.content,
      media: postData.media,
      platformData: postData.platformSpecific[platform]
    }, {
      delay: delay > 0 ? delay : 0,
      jobId: `${postData.id}-${platform}` // Idempotency
    });
  }
}

// Worker for Instagram
const instagramWorker = new Worker('instagram-posts', async (job) => {
  const { postId, userId, content, media } = job.data;

  try {
    // Update status to publishing
    await updatePostStatus(postId, 'publishing');

    // Get fresh access token
    const socialAccount = await getSocialAccount(userId, 'instagram');
    const accessToken = await refreshTokenIfNeeded(socialAccount);

    // Publish to Instagram
    const result = await publishToInstagram({
      accessToken,
      content,
      media,
      igUserId: socialAccount.platformUserId
    });

    // Update post with platform ID
    await updatePostPlatformId(postId, 'instagram', result.id);
    await updatePostStatus(postId, 'published');

    // Schedule analytics fetch (after 1 hour)
    await scheduleAnalyticsFetch(postId, 'instagram', result.id);

    return { success: true, platformPostId: result.id };

  } catch (error) {
    // Update with error
    await updatePostError(postId, 'instagram', {
      code: error.code,
      message: error.message,
      details: error.response?.data
    });
```

```
    // If token expired, refresh and retry
    if (error.code === 190) {
      await refreshAccessToken(userId, 'instagram');
      throw new Error('Token expired, retrying...');
    }

    // Rate limited - retry with longer delay
    if (error.code === 32 || error.code === 368) {
      throw new Error('Rate limited');
    }

    throw error;
  }
}, {
  connection,
  concurrency: 5, // Process 5 jobs at once
  limiter: {
    max: 200, // Instagram limit
    duration: 3600000 // 1 hour
  }
});

// Handle job events
instagramWorker.on('completed', (job) => {
  console.log(`✅ Instagram job ${job.id} completed`);
});

instagramWorker.on('failed', (job, err) => {
  console.error(`❌ Instagram job ${job.id} failed:`, err.message);
});

// Retry logic with exponential backoff
instagramWorker.on('error', (err) => {
  console.error('Worker error:', err);
});
```

📊 MONITORING QUEUES

javascript

```
// queue-monitor.js
const { QueueEvents } = require('bullmq');

const queueEvents = new QueueEvents('instagram-posts', { connection });

queueEvents.on('waiting', ({ jobId }) => {
  console.log(`Job ${jobId} is waiting`);
});

queueEvents.on('active', ({ jobId }) => {
  console.log(`Job ${jobId} is active`);
});

queueEvents.on('completed', ({ jobId, returnvalue }) => {
  console.log(`Job ${jobId} completed:`, returnvalue);
});

queueEvents.on('failed', ({ jobId, failedReason }) => {
  console.error(`Job ${jobId} failed:`, failedReason);
});

// Get queue stats
async function getQueueStats(queueName) {
  const queue = getQueueForPlatform(queueName);

  const [waiting, active, completed, failed, delayed] = await Promise.all([
    queue.getWaitingCount(),
    queue.getActiveCount(),
    queue.getCompletedCount(),
    queue.getFailedCount(),
    queue.getDelayedCount()
  ]);

  return { waiting, active, completed, failed, delayed };
}
```

---

14. ERROR HANDLING {#error-handling}

⚠ CENTRALIZED ERROR HANDLER

javascript

```
// errors.js
class APIError extends Error {
  constructor(platform, code, message, details) {
    super(message);
    this.platform = platform;
    this.code = code;
    this.details = details;
    this.retryable = this.determineRetryable(code);
  }

  determineRetryable(code) {
    const retryableCodes = {
      instagram: [190, 1, 2], // Token expired, API error, temporary
      tiktok: [10002, 10003], // Rate limit, service error
      linkedin: [401, 429], // Unauthorized, rate limited
      twitter: [88, 130, 131] // Rate limited, server errors
    };

    return retryableCodes[this.platform]?.includes(code) || false;
  }
}

// Platform-specific error handling
async function handlePlatformError(platform, error, context) {
  const errorHandlers = {
    instagram: handleInstagramError,
    tiktok: handleTikTokError,
    linkedin: handleLinkedInError,
    twitter: handleTwitterError
  };

  const handler = errorHandlers[platform];
  if (handler) {
    return await handler(error, context);
  }

  throw new APIError(platform, 'UNKNOWN', error.message, error);
}

async function handleInstagramError(error, context) {
  const errorCode = error.response?.data?.error?.code;

  switch (errorCode) {
    case 190: // Token expired
      await refreshAccessToken(context.userId, 'instagram');
      throw new APIError('instagram', 190, 'Token expired, retrying', { retryable: true });
```

```
    case 368: // Temporarily blocked
      throw new APIError('instagram', 368, 'Temporarily blocked for spam', {
        retryable: true,
        retryAfter: 3600000 // 1 hour
      });

    case 100: // Invalid parameter
      throw new APIError('instagram', 100, 'Invalid parameters', {
        retryable: false,
        details: error.response.data
      });

    default:
      throw new APIError('instagram', errorCode, error.message, error.response?.data);
  }
}

// Retry strategy
async function executeWithRetry(fn, options = {}) {
  const {
    maxAttempts = 3,
    backoffMs = 1000,
    backoffMultiplier = 2,
    onRetry = null
  } = options;

  let lastError;

  for (let attempt = 1; attempt <= maxAttempts; attempt++) {
    try {
      return await fn();
    } catch (error) {
      lastError = error;

      if (!error.retryable || attempt === maxAttempts) {
        throw error;
      }

      const delay = backoffMs * Math.pow(backoffMultiplier, attempt - 1);

      if (onRetry) {
        onRetry(attempt, delay, error);
      }

      await sleep(delay);
    }
```

```
  }

  throw lastError;
}
```

---

15. SECURITY BEST PRACTICES {#security}

🔒 TOKEN STORAGE & ENCRYPTION

javascript

```
// encryption.js
const crypto = require('crypto');

const ALGORITHM = 'aes-256-gcm';
const ENCRYPTION_KEY = Buffer.from(process.env.ENCRYPTION_KEY, 'hex'); // 32 bytes

function encryptToken(token) {
  const iv = crypto.randomBytes(16);
  const cipher = crypto.createCipheriv(ALGORITHM, ENCRYPTION_KEY, iv);

  let encrypted = cipher.update(token, 'utf8', 'hex');
  encrypted += cipher.final('hex');

  const authTag = cipher.getAuthTag();

  return {
    encrypted: encrypted,
    iv: iv.toString('hex'),
    authTag: authTag.toString('hex')
  };
}

function decryptToken(encryptedData) {
  const decipher = crypto.createDecipheriv(
    ALGORITHM,
    ENCRYPTION_KEY,
    Buffer.from(encryptedData.iv, 'hex')
  );

  decipher.setAuthTag(Buffer.from(encryptedData.authTag, 'hex'));

  let decrypted = decipher.update(encryptedData.encrypted, 'hex', 'utf8');
  decrypted += decipher.final('utf8');

  return decrypted;
}

// Store token
async function storeAccessToken(userId, platform, token, expiresIn) {
  const encrypted = encryptToken(token);

  await db.query(`
    UPDATE social_accounts
    SET
      access_token = $1,
      token_iv = $2,
```

```
      token_auth_tag = $3,
      token_expires_at = NOW() + INTERVAL '${expiresIn} seconds'
    WHERE user_id = $4 AND platform = $5
  `, [encrypted.encrypted, encrypted.iv, encrypted.authTag, userId, platform]);
}

// Retrieve token
async function getAccessToken(userId, platform) {
  const result = await db.query(`
    SELECT access_token, token_iv, token_auth_tag, token_expires_at
    FROM social_accounts
    WHERE user_id = $1 AND platform = $2
  `, [userId, platform]);

  if (!result.rows[0]) {
    throw new Error('Social account not found');
  }

  const { access_token, token_iv, token_auth_tag, token_expires_at } = result.rows[0];

  // Check if expired
  if (new Date(token_expires_at) < new Date()) {
    // Refresh token
    return await refreshAndGetToken(userId, platform);
  }

  return decryptToken({
    encrypted: access_token,
    iv: token_iv,
    authTag: token_auth_tag
  });
}
```

RATE LIMITING

javascript

```
// rate-limiter.js
const Redis = require('ioredis');
const redis = new Redis();

async function checkRateLimit(key, limit, windowMs) {
  const current = await redis.incr(key);

  if (current === 1) {
    await redis.pexpire(key, windowMs);
  }

  if (current > limit) {
    const ttl = await redis.pttl(key);
    throw new Error(`Rate limit exceeded. Try again in ${Math.ceil(ttl / 1000)}s`);
  }

  return {
    current,
    limit,
    remaining: limit - current
  };
}

// Middleware for API
function rateLimitMiddleware(options) {
  return async (req, res, next) => {
    const key = `ratelimit:${req.user.id}:${req.path}`;

    try {
      const result = await checkRateLimit(
        key,
        options.limit,
        options.windowMs
      );

      res.setHeader('X-RateLimit-Limit', result.limit);
      res.setHeader('X-RateLimit-Remaining', result.remaining);

      next();
    } catch (error) {
      res.status(429).json({ error: error.message });
    }
  };
}
```

WEBHOOK VERIFICATION

javascript

```javascript
// webhook-verification.js
const crypto = require('crypto');

// Instagram/Facebook
function verifyMetaWebhook(signature, body, appSecret) {
  const expectedSignature = crypto
    .createHmac('sha256', appSecret)
    .update(body)
    .digest('hex');

  return signature === `sha256=${expectedSignature}`;
}

// TikTok
function verifyTikTokWebhook(signature, body, clientSecret) {
  const expectedSignature = crypto
    .createHmac('sha256', clientSecret)
    .update(body)
    .digest('hex');

  return signature === expectedSignature;
}

// Webhook handler
app.post('/webhooks/:platform', async (req, res) => {
  const { platform } = req.params;
  const signature = req.headers['x-hub-signature-256'] || req.headers['x-signature'];

  const verifiers = {
    instagram: () => verifyMetaWebhook(signature, req.rawBody, process.env.META_APP_SECRET),
    facebook: () => verifyMetaWebhook(signature, req.rawBody, process.env.META_APP_SECRET),
    tiktok: () => verifyTikTokWebhook(signature, req.rawBody, process.env.TIKTOK_CLIENT_SECRET)
  };

  if (!verifiers[platform]?.()) {
    return res.status(403).json({ error: 'Invalid signature' });
  }

  // Process webhook
  await processWebhook(platform, req.body);
  res.status(200).send('OK');
});
```

📚 DODATKOWE ZASOBY

API Documentation Links

Meta (Instagram/Facebook): https://developers.facebook.com/docs/
TikTok: https://developers.tiktok.com/doc/
LinkedIn: https://learn.microsoft.com/en-us/linkedin/
Twitter/X: https://developer.twitter.com/en/docs
YouTube: https://developers.google.com/youtube
Pinterest: https://developers.pinterest.com/docs/
Reddit: https://www.reddit.com/dev/api
Google Business: https://developers.google.com/my-business
Telegram: https://core.telegram.org/bots/api
Bluesky: https://atproto.com/

Przydatne Biblioteki

javascript

```
{
  "meta": "@nestjs/facebook", // Unofficial wrapper
  "instagram": "instagram-private-api",
  "tiktok": "@tiktok-api",
  "linkedin": "linkedin-api-client",
  "twitter": "twitter-api-v2",
  "youtube": "googleapis",
  "pinterest": "pinterest-api",
  "reddit": "snoowrap",
  "telegram": "node-telegram-bot-api",
  "media": "sharp", // Image processing
  "video": "fluent-ffmpeg", // Video processing
  "queue": "bullmq",
  "database": "pg", // PostgreSQL
  "cache": "ioredis"
}
```

⚡ QUICK START CHECKLIST

markdown

□ Setup developer accounts na wszystkich platformach
□ Utwórz OAuth apps i zapisz credentials
□ Skonfiguruj PostgreSQL database
□ Setup Redis dla queue i cache
□ Skonfiguruj S3/R2 dla media storage
□ Zaimplementuj OAuth flow dla każdej platformy
□ Stwórz queue workers dla każdej platformy
□ Dodaj token encryption
□ Zaimplementuj rate limiting
□ Setup webhook endpoints
□ Dodaj error handling i retry logic
□ Stwórz monitoring (Sentry/DataDog)
□ Testy end-to-end dla każdej platformy

WAŻNE UWAGI:

1. **Rate Limits są KRYTYCZNE** - zawsze implementuj rate limiting i retry logic
2. **Token Management** - tokeny wygasają, potrzebujesz refresh logic
3. **Media Processing** - każda platforma ma inne wymagania (resize, transcode)
4. **Queue System** - MUSISZ używać queue dla schedulingu (BullMQ recommended)
5. **Error Handling** - każda platforma ma inne error codes
6. **Webhooks** - używaj webhooków zamiast pollingu gdzie możliwe
7. **Security** - ZAWSZE encrypt tokeny w database
8. **Compliance** - przestrzegaj ToS każdej platformy

Ostatnia aktualizacja: Luty 2026 **Status:** Production-ready **License:** Proprietary - HardbarRecordsLab

---

Powodzenia w budowaniu plannera! 🚀