# LinkedIn-First Social Media Scheduling Application

### Complete Product & Technical Specification Document

---

## 1. Problem Statement

LinkedIn rewards consistency over virality. Professionals, founders, and creators understand this fundamental truth, yet most fail to post regularly. The barrier is not content creation—it's sustaining the habit.

**Core Problems:**

- **Daily Decision Fatigue**: Users must decide what to post every single day
- **Timing Issues**: Remembering to post at optimal times during busy work hours
- **Momentum Loss**: Missing 2-3 days breaks the habit entirely, motivation drops to zero
- **Tool Complexity**: Existing tools (Buffer, Hootsuite, Later) are multi-platform swiss army knives, not LinkedIn-optimized
- **Context Switching**: Opening LinkedIn during work creates distraction and time loss
- **Content Paralysis**: Staring at a blank post composer kills productivity

**Why Existing Solutions Fail:**

- Buffer/Hootsuite: Built for Twitter/Facebook era, LinkedIn is an afterthought
- Native LinkedIn: No scheduling on personal profiles, only company pages
- Complex tools: Require team permissions, analytics overload, learning curve
- Generic approach: Don't understand LinkedIn's professional content patterns

**The Real Insight:** Consistency is a systems problem, not a motivation problem. Users need to batch-create content in focused sessions, then let automation handle the daily publishing rhythm.

---

## 2. Solution Statement

A LinkedIn-first web application that removes daily posting friction through intelligent batch scheduling and automated publishing, making professional content consistency effortless and reliable.

**Core Value Proposition:** Spend 30 minutes once a week planning content, then show up professionally on LinkedIn every day without thinking about it.

**Key Principles:**

1. **LinkedIn-First Design**: Every feature optimized for LinkedIn's professional context
2. **Batch Over Daily**: Create multiple posts in one focused session
3. **Automation Over Reminders**: System publishes automatically, no daily login required
4. **Clarity Over Features**: Simple, obvious UI that eliminates decision paralysis
5. **Reliability Over Complexity**: Posts publish on time, every time

---

## 3. How We Are Going To Build It

### 3.1 Product Features & User Workflow

**Core User Journey**

**Step 1: Authentication & Onboarding**

- User lands on landing page explaining the value proposition
- User signs up with email (NextAuth.js)
- User connects LinkedIn account via OAuth 2.0
- System requests necessary LinkedIn API permissions:
  - `w_member_social` - Post content to LinkedIn
  - `r_liteprofile` - Read basic profile info
  - `r_emailaddress` - User email for account matching
- System stores OAuth tokens securely in database
- User is redirected to dashboard

**Step 2: Dashboard Overview** User sees at a glance:

- **Today's Status**: "Post published at 9:00 AM" or "Next post: Tomorrow 9:00 AM"
- **Calendar View**: Visual month view showing scheduled post dots
- **Quick Stats**: Posts published this month, current streak, upcoming posts count
- **Quick Actions**: "Create New Post", "Schedule Multiple Posts", "View Queue"

**Step 3: Creating Content**

*Option A: Single Post Creation*

- User clicks "Create New Post"
- Opens LinkedIn-style composer with character count (3,000 limit)

- User writes post in rich text editor (supports line breaks, emojis)
- Live preview shows exactly how post will appear on LinkedIn
- User chooses:
    - "Publish Now" - Immediate posting
    - "Schedule" - Opens date/time picker
    - "Add to Queue" - Adds to auto-schedule queue
- User saves as draft or schedules

*Option B: Batch Post Creation (Power Feature)*

- User clicks "Batch Create Posts"
- Interface shows 3-5 post composers stacked vertically
- User writes multiple posts in one session
- User clicks "Smart Schedule" which:
    - Analyzes user's preferred posting frequency (daily, alternate days, weekdays)
    - Suggests optimal time slots based on timezone
    - Distributes posts evenly across selected period
- User reviews auto-generated schedule in calendar view
- User confirms or adjusts individual times
- All posts scheduled in one click

*Option C: AI-Assisted Creation (Future Enhancement)*

- User enters topic/bullet points: "Launched new product feature, increased user retention by 40%"
- AI generates 3 post variations:
    - **Storytelling Style**: "Three months ago, our retention was struggling..."
    - **Data-Driven Style**: "New feature impact: 40% retention increase, here's what we learned..."
    - **Question Style**: "What if one feature could change your retention by 40%? Here's our journey..."
- User selects preferred version, edits, and schedules

**Step 4: Content Calendar Management**

- Calendar view shows all scheduled posts
- Color coding:
    - Green: Successfully published
    - Blue: Scheduled (pending)

    - Red: Failed to publish
    - Gray: Draft
- Click any post to:
    - Edit content
    - Reschedule time
    - Delete from queue
    - Duplicate and reschedule
- Drag-and-drop to change posting dates

**Step 5: Automated Publishing**

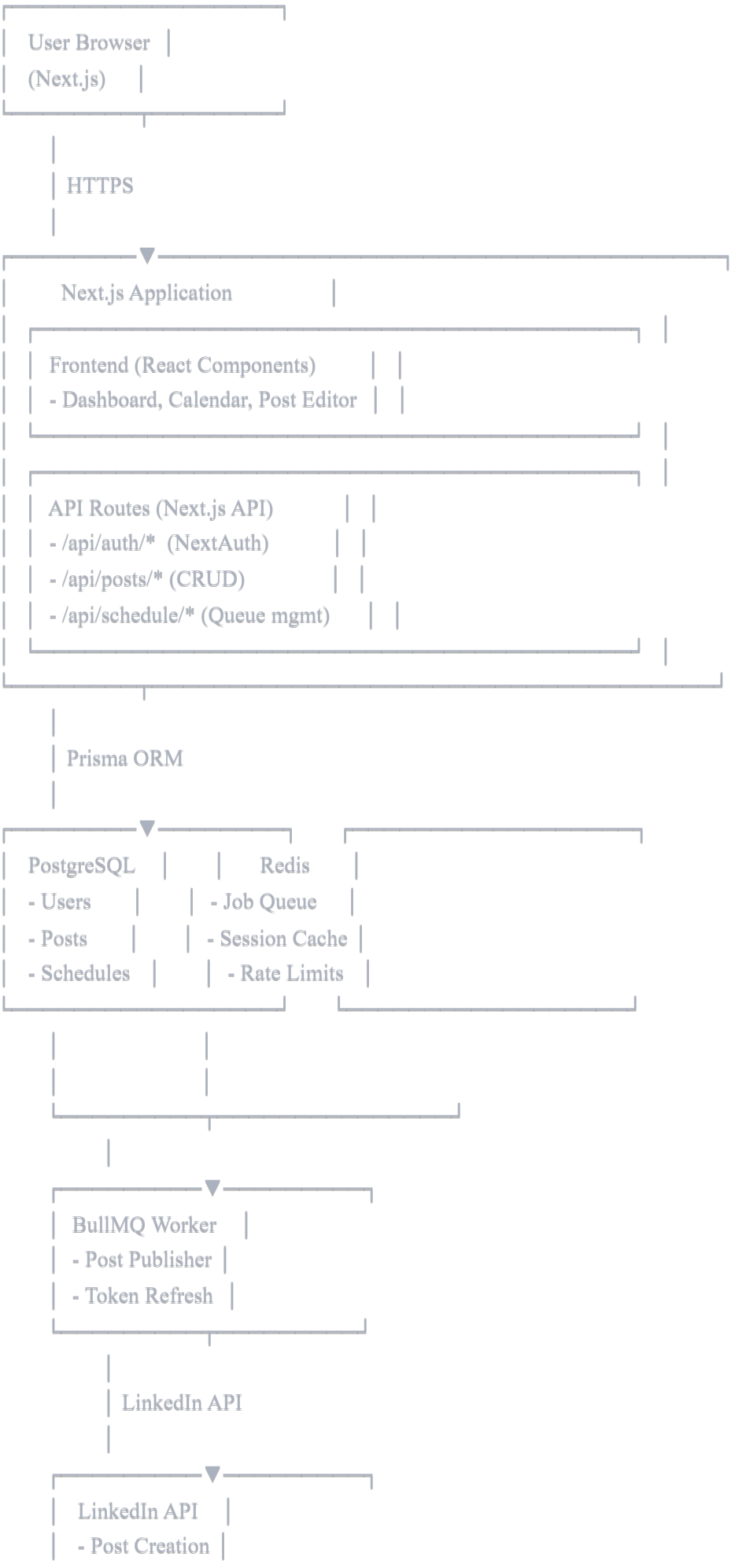
- System runs background job every minute checking for posts due to publish
- When post time arrives:
    - System retrieves post content and user's LinkedIn OAuth tokens
    - System calls LinkedIn API to publish post
    - System updates post status to "published"
    - System sends notification to user: "Your post 'How to build...' was published"
- If publishing fails:
    - System retries 3 times with exponential backoff (1min, 5min, 15min)
    - System marks post as "failed"
    - System sends notification: "Post failed to publish. LinkedIn token may have expired."
    - User can manually retry from dashboard

**Step 6: Monitoring & Notifications**

- Email notifications for:
    - Post published successfully
    - Post failed to publish (with reason)
    - LinkedIn token expiring soon (7 days before)
    - Weekly summary: "You published 5 posts this week, 12-day streak!"
- Dashboard notifications:
    - Red banner if any posts failed in last 24 hours
    - Yellow banner if LinkedIn token expires in < 7 days
    - Success toast when new post is scheduled

## 3.2 Technical Architecture

### System Architecture Overview


User Browser
(Next.js)
HTTPS
Next.js Application
Frontend (React Components)
- Dashboard, Calendar, Post Editor
API Routes (Next.js API)
- /api/auth/* (NextAuth)
- /api/posts/* (CRUD)
- /api/schedule/* (Queue mgmt)
Prisma ORM
PostgreSQL
- Users
- Posts
- Schedules
Redis
- Job Queue
- Session Cache
- Rate Limits
BullMQ Worker
- Post Publisher
- Token Refresh
LinkedIn API
LinkedIn API
- Post Creation


**Technology Stack (Detailed)**

**Frontend**

- **Framework**: Next.js 14+ (App Router)
  - Why: Server-side rendering, API routes, optimized builds, best React DX
  - File structure: `/app` directory for routes, `/components` for UI
- **Styling**: Tailwind CSS 3+
  - Why: Rapid development, consistent design system, small bundle size
  - Custom theme for LinkedIn brand colors
- **UI Components**: shadcn/ui (built on Radix UI)
  - Pre-built accessible components: Calendar, Dialog, Dropdown, Toast
  - Customizable with Tailwind
- **State Management**:
  - React Query (TanStack Query) for server state
  - Zustand for client state (UI state, draft posts)
- **Forms**: React Hook Form + Zod validation
  - Type-safe form handling
  - Real-time validation
- **Rich Text Editor**:
  - Simple textarea with character count for MVP
  - Future: Lexical or Tiptap for formatting support
- **Date/Time**: date-fns for timezone handling
  - Critical: Always store UTC in DB, display in user's timezone

**Backend**

- **Runtime**: Node.js 20+ LTS
- **Framework**: Next.js API Routes
  - RESTful API design
  - `/api/auth/*` - Authentication endpoints
  - `/api/posts/*` - Post CRUD operations

    - `/api/schedule/*` - Scheduling logic
    - `/api/linkedin/*` - LinkedIn API proxy
- **Authentication**: NextAuth.js v5
    - Email provider (magic links or password)
    - LinkedIn OAuth provider
    - JWT session strategy
    - Secure token storage
- **Database ORM**: Prisma 5+
    - Type-safe database queries
    - Automatic migrations
    - Schema-first development

**Database**

- **Primary DB**: PostgreSQL 15+
    - Why: ACID compliance, JSON support, reliable, scalable
    - Hosted on: Neon, Supabase, or Railway for easy setup

**Database Schema (Prisma)**

```prisma
```

```
// User model
model User {
  id               String    @id @default(cuid())
  email            String    @unique
  name             String?
  emailVerified    DateTime?
  image            String?
  createdAt        DateTime  @default(now())
  updatedAt        DateTime  @updatedAt

  accounts         Account[]
  sessions         Session[]
  posts            Post[]
  schedules        Schedule[]
}

// OAuth Account (LinkedIn connection)
model Account {
  id               String  @id @default(cuid())
  userId           String
  type             String
  provider         String  // "linkedin"
  providerAccountId String  // LinkedIn user ID
  refresh_token    String? @db.Text
  access_token     String? @db.Text
  expires_at       Int?
  token_type       String?
  scope            String?
  id_token         String? @db.Text
  session_state    String?

  user User @relation(fields: [userId], references: [id], onDelete: Cascade)

  @@unique([provider, providerAccountId])
  @@index([userId])
}

// NextAuth session
model Session {
  id           String   @id @default(cuid())
  sessionToken String   @unique
  userId       String
  expires      DateTime
  user         User     @relation(fields: [userId], references: [id], onDelete: Cascade)

  @@index([userId])
```

```
}

// Post content and metadata
model Post {
  id             String   @id @default(cuid())
  userId         String
  content        String   @db.Text
  status         PostStatus @default(DRAFT)
  scheduledFor   DateTime?
  publishedAt    DateTime?
  linkedinPostId String?  // LinkedIn's post ID after publishing
  failureReason  String?  @db.Text
  retryCount     Int      @default(0)
  createdAt      DateTime @default(now())
  updatedAt      DateTime @updatedAt

  user User @relation(fields: [userId], references: [id], onDelete: Cascade)

  @@index([userId, status])
  @@index([scheduledFor])
}

enum PostStatus {
  DRAFT
  SCHEDULED
  PUBLISHED
  FAILED
}

// User scheduling preferences
model Schedule {
  id             String   @id @default(cuid())
  userId         String   @unique
  preferredTime  String   // "09:00" in user's timezone
  timezone       String   // "America/New_York"
  frequency      Frequency @default(DAILY)
  weekdaysOnly   Boolean  @default(true)
  createdAt      DateTime @default(now())
  updatedAt      DateTime @updatedAt

  user User @relation(fields: [userId], references: [id], onDelete: Cascade)
}

enum Frequency {
  DAILY
  ALTERNATE_DAYS
  WEEKDAYS_ONLY
```

```
  CUSTOM
}
```

**Job Queue & Background Processing**

- **Queue System**: BullMQ
  - Redis-backed job queue
  - Retry logic with exponential backoff
  - Job scheduling (cron-like)
  - Job progress tracking
- **Worker Processes**:
  - **Post Publisher Worker**: Runs every minute, checks for posts due to publish
  - **Token Refresh Worker**: Runs daily, checks for expiring OAuth tokens
  - **Notification Worker**: Sends email notifications

**Redis**

- **Use Cases**:
  - BullMQ job queue storage
  - Session storage (NextAuth)
  - Rate limiting (prevent API abuse)
  - Cache for LinkedIn API responses
- **Hosted on**: Upstash (serverless Redis) or Redis Cloud

**LinkedIn API Integration**

- **API Version**: LinkedIn REST API v2
- **Required APIs**:
  - **UGC Posts API**: Create posts as authenticated user
  - **OAuth 2.0**: User authentication and authorization
- **Endpoints Used**:
  - `POST /ugcPosts` - Create a new post
  - `GET /me` - Get authenticated user's profile
- **Rate Limits**:
  - 100 API calls per user per day
  - Must implement rate limiting in our app

- **Token Management**:
    - Access tokens expire after 60 days
    - Refresh tokens do not exist for LinkedIn (must re-authenticate)
    - System must detect expired tokens and prompt re-authentication

**Hosting & Deployment**

- **Frontend + API**: Vercel
    - Why: Zero-config Next.js deployment, edge functions, great DX
    - Auto-deploy from Git (main branch)
    - Preview deployments for PRs
- **Database**: Neon PostgreSQL or Supabase
    - Serverless Postgres with generous free tier
    - Built-in connection pooling
- **Redis**: Upstash
    - Serverless Redis
    - Pay-per-request pricing
- **Background Workers**:
    - Option 1: Vercel Cron Jobs (for simple scheduled tasks)
    - Option 2: Railway or Render (for persistent BullMQ workers)

**Monitoring & Logging**

- **Error Tracking**: Sentry
    - Catches frontend and backend errors
    - Tracks failed post publishes
- **Logging**:
    - Structured JSON logs
    - Log levels: error, warn, info, debug
    - Store in cloud logging service (Vercel logs, Better Stack)
- **Uptime Monitoring**:
    - UptimeRobot or Checkly
    - Alerts if API goes down

**Email Service**

- **Provider**: Resend or SendGrid
    - Transactional emails for:
        - Magic link authentication
        - Post published notifications
        - Post failed notifications
        - Weekly summary emails
- **Email Templates**: React Email
    - Type-safe email templates
    - Renders to HTML

---

### 3.3 Detailed Implementation Plan

#### Phase 1: Foundation (Week 1-2)

**Step 1: Project Setup**

```bash
# Create Next.js project
npx create-next-app@latest linkedin-scheduler --typescript --tailwind --app

# Install core dependencies
npm install prisma @prisma/client
npm install next-auth@beta
npm install @tanstack/react-query
npm install zod react-hook-form @hookform/resolvers
npm install date-fns
npm install bullmq ioredis

# Install UI dependencies
npm install @radix-ui/react-dialog @radix-ui/react-dropdown-menu
npm install lucide-react
npm install class-variance-authority clsx tailwind-merge

# Setup Prisma
npx prisma init
```

**Step 2: Database Setup**

- Create PostgreSQL database on Neon/Supabase
- Define Prisma schema (see schema above)
- Run migrations: `npx prisma migrate dev --name init`
- Generate Prisma Client: `npx prisma generate`

**Step 3: Authentication Setup**

- Configure NextAuth.js with:
  - Email provider (for MVP)
  - LinkedIn OAuth provider
- Create auth API routes: `/app/api/auth/[...nextauth]/route.ts`
- Create login page: `/app/login/page.tsx`
- Create protected route middleware
- Test full auth flow

**Step 4: Basic UI Framework**

- Create layout with navigation
- Create dashboard page (empty state)
- Create post creation page (basic form)
- Setup Tailwind theme with LinkedIn colors
- Create reusable button, input, dialog components

**Phase 2: Core Features (Week 3-4)**

**Step 5: Post Creation**

- Build post editor component:
  - Textarea with character count (3,000 limit)
  - Real-time validation
  - Draft auto-save to local state
- Create API route: `POST /api/posts`
  - Validates post content
  - Saves to database with status: DRAFT
- Create API route: `GET /api/posts`
  - Returns user's posts (paginated)

- Test creating and retrieving posts

**Step 6: Post Scheduling**

- Build date/time picker component
- Create API route: `PUT /api/posts/[id]/schedule`
    - Updates post with scheduledFor timestamp
    - Changes status to SCHEDULED
- Build calendar view component:
    - Shows posts by date
    - Color-coded by status
- Test scheduling posts for future dates

**Step 7: LinkedIn API Integration**

- Register app on LinkedIn Developer Portal
- Configure OAuth redirect URLs
- Add LinkedIn provider to NextAuth
- Create API route: `POST /api/linkedin/publish`
    - Accepts post ID
    - Retrieves user's LinkedIn access token from Account table
    - Calls LinkedIn UGC Posts API
    - Handles success/failure responses
- Test manual post publishing

**Phase 3: Automation (Week 5-6)**

**Step 8: Background Job Queue Setup**

- Setup Redis connection (Upstash)
- Create BullMQ queue configuration
- Create worker file: `/workers/post-publisher.ts`
- Deploy worker as separate process (Railway/Render)
- Test job creation and processing

**Step 9: Automated Post Publishing**

- Create scheduled job that runs every minute
- Job logic:

typescript

```typescript
// Pseudo-code
async function publishScheduledPosts() {
  const now = new Date();
  const posts = await prisma.post.findMany({
    where: {
      status: 'SCHEDULED',
      scheduledFor: { lte: now }
    },
    include: { user: { include: { accounts: true } } }
  });

  for (const post of posts) {
    await queue.add('publish-post', { postId: post.id });
  }
}
```

- Worker processes job:

typescript

```
worker.process('publish-post', async (job) => {
  const { postId } = job.data;
  const post = await getPostWithUser(postId);

  try {
    const linkedinPostId = await publishToLinkedIn(
      post.content,
      post.user.linkedinAccessToken
    );

    await prisma.post.update({
      where: { id: postId },
      data: {
        status: 'PUBLISHED',
        publishedAt: new Date(),
        linkedinPostId
      }
    });

    await sendSuccessNotification(post.user.email);
  } catch (error) {
    await handlePublishError(post, error);
  }
});
```

- Implement retry logic with exponential backoff
- Test end-to-end automated publishing

**Step 10: Error Handling & Notifications**

- Implement failed post detection
- Create email notification system:
  - Success: "Your post was published"
  - Failure: "Post failed - LinkedIn token expired"
- Display failures on dashboard with retry button
- Test error scenarios (expired token, rate limit, network error)

**Phase 4: Polish & Launch (Week 7-8)**

**Step 11: Batch Creation Feature**

- Create "Batch Create" page with multiple editors

- Create "Smart Schedule" algorithm:
  - Takes user's frequency preference
  - Distributes posts evenly across timeframe
  - Respects user's preferred time and timezone
- Test batch scheduling flow

**Step 12: Dashboard Improvements**

- Add stats: posts published this month, current streak
- Add quick actions: Create post, View calendar
- Add status indicators: Today's post, Next post
- Improve calendar with drag-to-reschedule
- Add post preview modal

**Step 13: Token Expiration Handling**

- Create daily job to check token expiration
- Send email 7 days before expiration: "Reconnect your LinkedIn"
- Show banner on dashboard when token expired
- Create "Reconnect LinkedIn" flow
- Test re-authentication process

**Step 14: Landing Page & Onboarding**

- Create landing page with:
  - Hero section explaining value prop
  - Feature highlights
  - Pricing (if applicable)
  - Call-to-action: "Start Scheduling"
- Create onboarding flow:
  - Welcome screen
  - LinkedIn connection
  - Set preferred time & frequency
  - Create first post tutorial
- Add empty states for new users

**Step 15: Testing & Deployment**

- Write integration tests for critical paths:
  - User signup → LinkedIn connect → Create post → Schedule → Publish
- Test timezone handling thoroughly
- Load test with multiple users and posts
- Deploy to production (Vercel + Railway)
- Setup monitoring and alerts
- Soft launch to 10 beta users

---

### 3.4 Critical Technical Considerations

#### OAuth Token Management

LinkedIn access tokens expire after 60 days. This is the biggest technical challenge.

**Solution**:

- Store `expires_at` timestamp in Account table
- Create daily cron job that checks: `expires_at < now + 7 days`
- Send email notification: "Your LinkedIn connection expires in X days. Reconnect now."
- Show persistent banner on dashboard when token expired
- Create "Reconnect LinkedIn" button that triggers OAuth flow
- After reconnect, update access_token and expires_at
- Before every publish, check token validity, fail gracefully if expired

#### Timezone Handling

User schedules post for "9:00 AM" but they travel or move. Post publishes at wrong time.

**Solution**:

- Store user's timezone in Schedule table
- Always store timestamps in UTC in database
- Convert to user's timezone for display
- When scheduling: `userLocalTime → UTC → database`
- When displaying: `database UTC → user's timezone → display`

- Use date-fns-tz library: `formatInTimeZone()`, `zonedTimeToUtc()`
- Allow user to change timezone in settings

**Rate Limiting**

LinkedIn API limits: 100 calls per user per day.

**Solution**:

- Track API calls per user per day in Redis
- Before making LinkedIn API call, check Redis counter
- If at limit, fail gracefully with clear message
- Reset counter at midnight UTC
- Show rate limit usage on dashboard: "72/100 API calls used today"

**Failed Post Handling**

Network issues, API downtime, rate limits cause failures.

**Solution**:

- Implement retry with exponential backoff:
  - Retry 1: After 1 minute
  - Retry 2: After 5 minutes
  - Retry 3: After 15 minutes
- After 3 failures, mark as FAILED
- Store failure reason in database
- Send email notification immediately
- Show "Failed Posts" section on dashboard with "Retry" button
- Manual retry resets retry count

**Duplicate Post Prevention**

Job runs twice, publishes same post twice.

**Solution**:

- Use BullMQ's job ID based on post ID (idempotent)
- Check post status before publishing:

```
typescript
```

```
if (post.status === 'PUBLISHED') {
  console.log('Post already published, skipping');
  return;
}
```

- Use database transaction when updating status
- LinkedIn API is idempotent (same content posted twice creates two separate posts, but our status check prevents this)

**Scalability Considerations**

What happens at 10,000 users with 100,000 scheduled posts?

**Solution**:

- **Database**: Postgres scales to millions of rows easily
  - Add indexes on: `userId`, `status`, `scheduledFor`
  - Use pagination for all list queries
- **Job Queue**: BullMQ handles thousands of jobs per second
  - Separate queues for different priorities
  - Scale workers horizontally (run multiple worker instances)
- **LinkedIn API**: Rate limit per user, not per app
  - Each user has 100 calls/day, so 10k users = 1M calls/day capacity
- **Redis**: Upstash scales automatically
- **Next.js**: Vercel scales to millions of requests

---

### 3.5 API Endpoints Reference

**Authentication**

- `POST /api/auth/signin` - Email or LinkedIn login
- `POST /api/auth/signout` - Logout
- `GET /api/auth/session` - Get current session
- `GET /api/auth/callback/linkedin` - OAuth callback

**Posts**

- `GET /api/posts` - List user's posts (paginated, filterable by status)
- `POST /api/posts` - Create new post (draft)
- `GET /api/posts/[id]` - Get single post
- `PUT /api/posts/[id]` - Update post content
- `DELETE /api/posts/[id]` - Delete post
- `PUT /api/posts/[id]/schedule` - Schedule post for future time
- `POST /api/posts/[id]/publish-now` - Publish immediately
- `POST /api/posts/[id]/retry` - Retry failed post

**Scheduling**

- `GET /api/schedule` - Get user's schedule preferences
- `PUT /api/schedule` - Update schedule preferences
- `POST /api/schedule/batch` - Batch schedule multiple posts

**LinkedIn**

- `POST /api/linkedin/publish` - Internal endpoint to publish post
- `GET /api/linkedin/profile` - Get user's LinkedIn profile info
- `POST /api/linkedin/reconnect` - Refresh LinkedIn connection

**Calendar**

- `GET /api/calendar?month=2025-01` - Get all posts for calendar view

---

### 3.6 Environment Variables

```env
```

```
# Database
DATABASE_URL="postgresql://user:password@host:5432/dbname"

# NextAuth
NEXTAUTH_URL="https://yourdomain.com"
NEXTAUTH_SECRET="generate-with-openssl-rand-base64-32"

# LinkedIn OAuth
LINKEDIN_CLIENT_ID="your-linkedin-app-id"
LINKEDIN_CLIENT_SECRET="your-linkedin-app-secret"

# Redis
REDIS_URL="redis://default:password@host:6379"

# Email (Resend)
RESEND_API_KEY="re_xxxxxxxxxxxx"
EMAIL_FROM="noreply@yourdomain.com"

# Sentry (optional)
SENTRY_DSN="https://xxx@sentry.io/xxx"

# Feature Flags
ENABLE_AI_FEATURES="false"
```

---

### 3.7 File Structure

```
linkedin-scheduler/
├── app/
│   ├── (auth)/
│   │   ├── login/
│   │   │   └── page.tsx
│   │   └── signup/
│   │       └── page.tsx
│   ├── (dashboard)/
│   │   ├── layout.tsx
│   │   ├── page.tsx              # Main dashboard
│   │   ├── posts/
│   │   │   ├── page.tsx          # Posts list
│   │   │   ├── new/
│   │   │   │   └── page.tsx      # Create post
│   │   │   ├── [id]/
│   │   │   │   └── page.tsx      # Edit post
│   │   │   └── batch/
│   │   │       └── page.tsx      # Batch create
```

```
│   │   ├── calendar/
│   │   │   └── page.tsx          # Calendar view
│   │   └── settings/
│   │       └── page.tsx          # User settings
│   ├── api/
│   │   ├── auth/
│   │   │   └── [...nextauth]/
│   │   │       └── route.ts
│   │   ├── posts/
│   │   │   ├── route.ts          # GET, POST
│   │   │   └── [id]/
│   │   │       ├── route.ts      # GET, PUT, DELETE
│   │   │       ├── schedule/
│   │   │       │   └── route.ts
│   │   │       └── publish-now/
│   │   │           └── route.ts
│   │   ├── schedule/
│   │   │   ├── route.ts
│   │   │   └── batch/
│   │   │       └── route.ts
│   │   ├── linkedin/
│   │   │   ├── publish/
│   │   │   │   └── route.ts
│   │   │   └── profile/
│   │   │       └── route.ts
│   │   └── calendar/
│   │       └── route.ts
│   ├── layout.tsx
│   └── page.tsx                  # Landing page
├── components/
│   ├── ui/                       # shadcn components
│   │   ├── button.tsx
│   │   ├── dialog.tsx
│   │   ├── calendar.tsx
│   │   └── ...
│   ├── post-editor.tsx
│   ├── post-card.tsx
│   ├── calendar-view.tsx
│   ├── schedule-picker.tsx
│   └── navbar.tsx
├── lib/
│   ├── prisma.ts                 # Prisma client singleton
│   ├── auth.ts                   # NextAuth config
│   ├── linkedin.ts               # LinkedIn API wrapper
│   ├── queue.ts                  # BullMQ setup
│   ├── redis.ts                  # Redis client
│   ├── email.ts                  # Email sending
```

```
│   └── utils.ts              # Helper functions
├── workers/
│   ├── post-publisher.ts     # Background job worker
│   └── token-checker.ts      # Token expiry checker
├── prisma/
│   ├── schema.prisma
│   └── migrations/
├── public/
├── .env.local
├── next.config.js
├── tailwind.config.js
├── tsconfig.json
└── package.json
```

---

## 4. Success Metrics

**User Engagement**:

- Users posting 3+ times per week for 30+ days: Target 60%
- Average posts scheduled per user per session: Target 5+
- User retention after 30 days: Target 70%

**Technical Performance**:

- Post publish success rate: Target 98%+
- Average time to publish after scheduled time: < 60 seconds
- API error rate: < 1%
- Page load time: < 2 seconds

**Business Metrics**:

- User signup to first scheduled post: Target < 10 minutes
- Time saved per user per week: Target 2+ hours (vs manual posting)
- User satisfaction (NPS): Target 50+

---

## 5. Future