Post Editing Feature Implementation Plan

Overview & Purpose

This document outlines a strategy for implementing post editing functionality by extending the existing post creation flow. Currently, users can create posts but cannot edit scheduled or failed posts. By implementing a unified flow that handles both creation and editing, we can:

1. **Maximize code reuse** - Leveraging the existing creation flow components
2. **Maintain UI consistency** - Users experience the same interface for related actions
3. **Simplify maintenance** - Changes to the flow affect both creation and editing
4. **Reduce development time** - Building on existing components rather than creating parallel functionality

The approach modifies the existing post creation components to conditionally render or behave differently based on whether we're creating a new post or editing an existing one.

Component Functionality in Different Modes

Component (Current → Proposed Name)	Description	Create Mode	Edit Mode	Reasoning for Edit Mode Behavior
Step 1: ImageAnalyser → PostImageSelector	Allows uploading images and performs AI detection of content	Active	**Partially Active**	Image upload needed for changes, but AI analysis is unnecessary when editing existing content; confirmation modal also disabled
Step 2: PostDetails	Collects business info, categories, and additional context for AI caption generation	Active	**Disabled**	This step primarily gathers inputs for AI caption generation, which isn't needed when editing existing content
Step 3: CaptionSelection → CaptionEditor	Contains two main sections: AI suggestions (top) and platform-specific editors (bottom)	Active	**Partially Active**	AI suggestions section hidden in edit mode, platform-specific editors shown by default
Step 4: PostReviewStep	Shows final preview before posting/updating	Active	**Modified**	Same functionality but with "Update" button instead of "Post"; scheduled time can be modified

Implementation Requirements

Feature	Files to Modify	Changes Required
Mode Parameter	Frontend: `src/components/post/create/PostCreationFlow.tsx` → `PostEditorFlow.tsx`	Add new `mode` enum parameter (`'create'` or `'edit'`)
	Frontend: `src/app/(protected)/posts/page.tsx`	Update Modal implementation to pass mode and post data
Data Population	Frontend: `src/components/post/create/PostEditorFlow.tsx`	Add logic to populate state with existing post data when in edit mode
Conditional Rendering	Frontend: `src/components/post/create/ImageAnalyser.tsx` → `PostImageSelector.tsx`	Disable analysis button and confirmation modal in edit mode
	Frontend: `src/components/post/create/CaptionSelection.tsx` → `CaptionEditor.tsx`	Add conditional to hide AI suggestion section and show platform editors by default in edit mode
API Integration	Frontend: `src/constants/api.ts`	Update `api.ts` to use a consistent endpoint for both update and delete operations
	Backend: `backend/posts/urls.py`	Add a new URL pattern for `PostDetailView`
	Backend: `backend/posts/views.py`	Implement a `PostDetailView` class to handle editing as well as deletion

Post Attributes Editability by Status

Post Attribute	Scheduled Post	Posted Post	Failed Post
Image	✅ Editable	❌ Read-Only	✅ Editable
Caption	✅ Editable	❌ Read-Only	✅ Editable
Categories	✅ Editable	❌ Read-Only	✅ Editable
Scheduled Time	✅ Editable	N/A	✅ Editable
Platform	✅ Editable	❌ Read-Only	✅ Editable

Note: Posted content editing is disabled as most social media APIs require paid plans for editing functionality. While we could implement this in the future, it's not a current priority. The current UI already

prevents users from attempting to edit posted content by not showing edit buttons for these items.

User Experience Workflow for Edit Mode

1. User clicks "Edit" on a scheduled or failed post in the Posts dashboard
    - The edit button is only available for scheduled or failed posts, not for posted content
2. System opens the `PostEditorFlow` with `mode="edit"` and passes the existing post data
3. Flow begins with the image already loaded in Step 1:
    - User can change the image if desired
    - AI analysis button is disabled to prevent unnecessary processing
    - Confirmation modal for skipping analysis is disabled
4. System skips Step 2 (PostDetails) entirely as it's not needed for editing
5. System opens Step 3 (CaptionEditor) with:
    - AI suggestion section hidden
    - Platform-specific caption editors open by default with existing captions loaded
6. User can proceed to Step 4 to review changes before updating
    - User can modify scheduled time if needed
    - Update button replaces Post button

Design Considerations and Selected Approach

1. CaptionEditor Component Improvement Options:

1. **Option 1: Split into separate components:**
    - `AICaptionSuggestions` - Handles only the AI-generated content section
    - `PlatformCaptionEditors` - Handles the platform-specific editing section
    - Pros: Cleaner separation of concerns, easier to conditionally render
    - Cons: Requires more refactoring, may complicate state management
2. **Option 2: Keep as single component with improved conditional rendering:**
    - Add clear section identifiers
    - Use a mode parameter to control visibility
    - Pros: Less refactoring, maintains current state management
    - Cons: Component remains responsible for multiple concerns

Selected Approach: For the current implementation, I will proceed with Option 2 (keeping a single component with conditional rendering). This allows me to implement the editing functionality with minimal changes to the existing codebase.

2. Post Editing Strategy for Multi-Platform Posts:

1. **Option 1: Platform-specific editing:**
    - Edit each platform's post individually
    - Simpler approach with clearer database model
    - Better matches actual social media behavior where posts on different platforms are truly separate
    - May be repetitive for users if same change is needed across platforms

2. **Option 2: Group editing with creation ID:**
    - Add a "creation_group_id" to database model linking related posts
    - Allow editing all posts from same creation session simultaneously
    - More convenient for users making same change across platforms
    - More complex database model and state management
    - Doesn't reflect real-world separation between platforms

Selected Approach: For the current implementation, I will proceed with Option 1 (platform-specific editing) for simplicity and alignment with how social platforms actually work. This approach requires minimal database changes and matches user expectations that posts on different platforms are independent entities.

Reflection

Challenges in the Current Approach

1. **Component Overloading:** The current components handle multiple responsibilities, making them harder to adapt for different use cases.
2. **Step-Based Flow:** The rigid step-based flow doesn't easily accommodate skipping steps or showing different content based on mode.
3. **Tight Coupling:** The AI-driven features are tightly integrated with the UI components, making it difficult to disable just those features.
4. **Missing Edit Consideration:** The original design didn't account for future editing functionality, creating technical debt that we now need to address.

Learnings for Future Development

1. **Component Design:** Design components with single responsibilities from the start, making them more adaptable to different contexts.
2. **Feature Flagging:** Build in feature flags or mode parameters early on to easily toggle functionality.
3. **State Management:** Consider using a more structured state management approach (like reducers) to handle complex flows with different modes.
4. **Progressive Disclosure:** Design UIs with progressive disclosure in mind, where additional options appear only when needed.
5. **Anticipate Future Needs:** Consider how features might evolve and be extended when creating initial designs. If we had built the creation flow with editing in mind from the start, we could have saved significant refactoring effort.
6. **Database Design:** Our current approach ties posts directly to specific platforms without grouping related posts. For future projects, considering a more flexible model that accommodates editing groups of related content would be beneficial.

This edit implementation is a practical solution given the current architecture, but future versions should consider a more modular approach where components are designed from the ground up to support multiple modes and use cases.