

피드 작성, 피드 열람 UI form 에 대한 tsx 코드 소개_ 정동진.

1. 피드 작성 UI Form.

"build me the feed writing form UI design with tsx script which includes feed pictures inserting component and feed introduction writing form."

UI design 은 아래와 같습니다 😊

Create Your Post
Share your thoughts, add images, and engage with your audience

B I # @ 😊

What's on your mind?

Use # for hashtags Use @ for mentions 0/2200

Add Images (0/10)

Drop images here
or click to browse (0/10)

Browse Files Image URL

Saved 오전 9:44:47

Copy Prompt Edit in chat Save Draft Publish

위 tsx 파일의 소스 코드는 아래와 같습니다.

Paste below code into '

/components/ui' folder and name this file as ' component.tsx'.

```
import React, { useState, useRef, useCallback, useEffect } from 'react';
import { Button } from '@components/ui/button';
import { Card } from '@components/ui/card';
import { Input } from '@components/ui/input';
import { Label } from '@components/ui/label';
import { Textarea } from '@components/ui/textarea';
import { Badge } from '@components/ui/badge';
import { Progress } from '@components/ui/progress';
import { Separator } from '@components/ui/separator';
import { Alert, AlertDescription } from '@components/ui/alert';
import { Tooltip, TooltipContent, TooltipProvider, TooltipTrigger } from '@components/ui/tooltip';
import { Dialog, DialogContent, DialogHeader, DialogTitle, DialogTrigger } from '@components/ui/dialog';
import { Popover, PopoverContent, PopoverTrigger } from '@components/ui/popover';
import {
  Upload,
  X,
  Image as ImageIcon,
  Link,
  Bold,
  Italic,
  Hash,
  AtSign,
  Smile,
  Save,
  Send,
  AlertCircle,
  CheckCircle,
  Crop,
  Move,
  RotateCw,
```

```

    Eye,
    EyeOff,
    Loader2
  } from 'lucide-react';

interface ImageData {
  id: string;
  file?: File;
  url: string;
  altText: string;
  preview: string;
  uploadProgress?: number;
  error?: string;
}

interface FeedFormData {
  content: string;
  images: ImageData[];
  hashtags: string[];
  mentions: string[];
}

interface FeedWritingFormProps {
  onSubmit?: (data: FeedFormData) ⇒ void;
  onSave?: (data: FeedFormData) ⇒ void;
  maxImages?: number;
  maxCharacters?: number;
  theme?: 'light' | 'dark';
  autoSave?: boolean;
  enableHashtags?: boolean;
  enableMentions?: boolean;
  enableEmojis?: boolean;
  placeholder?: string;
  className?: string;
}

const ImageCropDialog: React.FC<{
  image: ImageData;

```

```

isOpen: boolean;
onClose: () ⇒ void;
onSave: (croppedImage: string) ⇒ void;
}> = ({ image, isOpen, onClose, onSave }) ⇒ {
  const canvasRef = useRef<HTMLCanvasElement>(null);
  const [rotation, setRotation] = useState(0);

  const handleSave = () ⇒ {
    if (canvasRef.current) {
      const croppedData = canvasRef.current.toDataURL();
      onSave(croppedData);
      onClose();
    }
  };

  return (
    <Dialog open={isOpen} onOpenChange={onClose}>
      <DialogContent className="max-w-4xl">
        <DialogHeader>
          <DialogTitle>Edit Image</DialogTitle>
        </DialogHeader>
        <div className="space-y-4">
          <div className="flex justify-center bg-muted rounded-lg p-4">
            <img
              src={image.preview}
              alt="Crop preview"
              className="max-h-96 object-contain"
              style={{ transform: `rotate(${rotation}deg)` }}
            />
          </div>
          <div className="flex items-center gap-2">
            <Button
              variant="outline"
              size="sm"
              onClick={() ⇒ setRotation(r ⇒ r + 90)}
            >
              <RotateCw className="w-4 h-4 mr-2" />
              Rotate

```

```

    </Button>
    <Separator orientation="vertical" className="h-6" />
    <Button onClick={handleSave}>
      <Save className="w-4 h-4 mr-2" />
      Save Changes
    </Button>
  </div>
</div>
</DialogContent>
</Dialog>
);
};

const EmojiPicker: React.FC<{
  onEmojiSelect: (emoji: string) => void;
}> = ({ onEmojiSelect }) => {
  const emojis = ['😊', '😂', '😍', '😞', '👍', '❤️', '🔥', '💯', '🎉', '🚀', '💪', '☀️'];

  return (
    <div className="grid grid-cols-6 gap-2 p-2">
      {emojis.map((emoji, index) => (
        <button
          key={index}
          onClick={() => onEmojiSelect(emoji)}
          className="p-2 hover:bg-muted rounded text-lg transition-colors"
        >
          {emoji}
        </button>
      ))}
    </div>
  );
};

const ImageUploadArea: React.FC<{
  images: ImageData[];
  onImagesAdd: (files: File[]) => void;
  onImageRemove: (id: string) => void;

```

```

onImageReorder: (fromIndex: number, toIndex: number) ⇒ void;
onImageEdit: (id: string) ⇒ void;
onAltTextChange: (id: string, altText: string) ⇒ void;
maxImages: number;
}> = ({
  images,
  onImagesAdd,
  onImageRemove,
  onImageReorder,
  onImageEdit,
  onAltTextChange,
  maxImages
}) ⇒ {
  const [isDragging, setIsDragging] = useState(false);
  const [urlInput, setUrlInput] = useState('');
  const [draggedIndex, setDraggedIndex] = useState<number | null>(null);
  const fileInputRef = useRef<HTMLInputElement>(null);

  const handleDragOver = useCallback((e: React.DragEvent) ⇒ {
    e.preventDefault();
    setIsDragging(true);
  }, []);

  const handleDragLeave = useCallback((e: React.DragEvent) ⇒ {
    e.preventDefault();
    setIsDragging(false);
  }, []);

  const handleDrop = useCallback((e: React.DragEvent) ⇒ {
    e.preventDefault();
    setIsDragging(false);

    const files = Array.from(e.dataTransfer.files).filter(file ⇒
      file.type.startsWith('image/'))
    );

    if (files.length > 0) {
      onImagesAdd(files);
    }
  }, [onImagesAdd]);

```

```

    }
  }, [onImagesAdd]);

  const handleFileSelect = useCallback((e: React.ChangeEvent<HTMLInput
  Element>) => {
    const files = Array.from(e.target.files || []);
    if (files.length > 0) {
      onImagesAdd(files);
    }
  }, [onImagesAdd]);

  const handleUrlAdd = useCallback(() => {
    if (urlInput.trim()) {
      const mockFile = new File([''], 'url-image', { type: 'image/jpeg' });
      onImagesAdd([mockFile]);
      setUrlInput('');
    }
  }, [urlInput, onImagesAdd]);

  const handleImageDragStart = (e: React.DragEvent, index: number) => {
    setDraggedIndex(index);
    e.dataTransfer.effectAllowed = 'move';
  };

  const handleImageDragOver = (e: React.DragEvent, index: number) => {
    e.preventDefault();
    if (draggedIndex !== null && draggedIndex !== index) {
      onImageReorder(draggedIndex, index);
      setDraggedIndex(index);
    }
  };

  const canAddMore = images.length < maxImages;

  return (
    <div className="space-y-4">
      {canAddMore && (
        <Card

```

```

        className={`border-2 border-dashed transition-colors ${
          isDragging ? 'border-primary bg-primary/5' : 'border-muted-foreground
und/25'
        }}
        onDragOver={handleDragOver}
        onDragLeave={handleDragLeave}
        onDrop={handleDrop}
      >
      <div className="p-8 text-center space-y-4">
        <div className="flex justify-center">
          <div className="p-4 bg-muted rounded-full">
            <Upload className="w-8 h-8 text-muted-foreground" />
          </div>
        </div>
        <div>
          <p className="text-lg font-medium">Drop images here</p>
          <p className="text-sm text-muted-foreground">
            or click to browse ({images.length}/{maxImages})
          </p>
        </div>
        <div className="flex flex-col sm:flex-row gap-2 justify-center">
          <Button
            variant="outline"
            onClick={() => fileInputRef.current?.click()}
          >
            <ImageIcon className="w-4 h-4 mr-2" />
            Browse Files
          </Button>
          <div className="flex gap-2">
            <Input
              placeholder="Image URL"
              value={urlInput}
              onChange={(e) => setUrlInput(e.target.value)}
              className="w-48"
            />
            <Button
              variant="outline"
              onClick={handleUrlAdd}

```



```

        disabled={!urlInput.trim()}
      >
        <Link className="w-4 h-4" />
      </Button>
    </div>
  </div>
</div>
</Card>
)}

<input
  ref={fileInputRef}
  type="file"
  multiple
  accept="image/*"
  onChange={handleFileSelect}
  className="hidden"
/>

{images.length > 0 && (
  <div className="grid grid-cols-1 sm:grid-cols-2 lg:grid-cols-3 gap-4">
    {images.map((image, index) => (
      <Card key={image.id} className="overflow-hidden">
        <div
          className="relative group cursor-move"
          draggable
          onStartDrag={(e) => handleImageDragStart(e, index)}
          onDragOver={(e) => handleImageDragOver(e, index)}
          onDragEnd={() => setDraggedIndex(null)}
        >
          <div className="aspect-square bg-muted flex items-center justify-center">
            {image.preview ? (
              <img
                src={image.preview}
                alt={image.altText || `Image ${index + 1}`}
                className="w-full h-full object-cover"

```

```

        />
      ) : (
        <ImageIcon className="w-12 h-12 text-muted-foreground" />
      )}
    </div>

    {image.uploadProgress !== undefined && image.uploadProgress
    < 100 && (
      <div className="absolute inset-0 bg-black/50 flex items-center justify-center">
        <div className="text-white text-center">
          <Loader2 className="w-6 h-6 animate-spin mx-auto mb-2"
        />
          <Progress value={image.uploadProgress} className="w-24"
        />
        </div>
      </div>
    )}

    <div className="absolute top-2 right-2 opacity-0 group-hover:opacity-100 transition-opacity">
      <div className="flex gap-1">
        <TooltipProvider>
          <Tooltip>
            <TooltipTrigger asChild>
              <Button
                size="sm"
                variant="secondary"
                onClick={() => onImageEdit(image.id)}
              >
                <Crop className="w-3 h-3" />
              </Button>
            </TooltipTrigger>
            <TooltipContent>Edit image</TooltipContent>
          </Tooltip>
        </TooltipProvider>

        <Button

```

```

        size="sm"
        variant="destructive"
        onClick={() ⇒ onImageRemove(image.id)}
      >
        <X className="w-3 h-3" />
      </Button>
    </div>
  </div>

  <div className="absolute bottom-2 left-2 opacity-0 group-hover:opacity-100 transition-opacity">
    <Badge variant="secondary" className="text-xs">
      <Move className="w-3 h-3 mr-1" />
      {index + 1}
    </Badge>
  </div>
</div>

<div className="p-3 space-y-2">
  <Input
    placeholder="Alt text (accessibility)"
    value={image.altText}
    onChange={(e) ⇒ onAltTextChange(image.id, e.target.value)}
    className="text-sm"
  />
  {image.error && (
    <Alert variant="destructive">
      <AlertCircle className="w-4 h-4" />
      <AlertDescription className="text-xs">
        {image.error}
      </AlertDescription>
    </Alert>
  )}
</div>
</Card>
)}}
</div>
)}

```

```

    </div>
  );
};

const RichTextEditor: React.FC<{
  content: string;
  onChange: (content: string) ⇒ void;
  onHashtagAdd: (hashtag: string) ⇒ void;
  onMentionAdd: (mention: string) ⇒ void;
  maxCharacters: number;
  placeholder: string;
  enableHashtags: boolean;
  enableMentions: boolean;
  enableEmojis: boolean;
}> = ({
  content,
  onChange,
  onHashtagAdd,
  onMentionAdd,
  maxCharacters,
  placeholder,
  enableHashtags,
  enableMentions,
  enableEmojis
}) ⇒ {
  const textareaRef = useRef<HTMLTextAreaElement>(null);
  const [cursorPosition, setCursorPosition] = useState(0);
  const [showSuggestions, setShowSuggestions] = useState(false);
  const [suggestionType, setSuggestionType] = useState<'hashtag' | 'mention' | null>(null);
  const [suggestions] = useState({
    hashtags: ['trending', 'viral', 'photography', 'nature', 'lifestyle', 'travel'],
    mentions: ['john_doe', 'jane_smith', 'photographer_pro', 'travel_blogger']
  });

  const characterCount = content.length;
  const isNearLimit = characterCount > maxCharacters * 0.8;
  const isOverLimit = characterCount > maxCharacters;

```

```

const handleTextChange = (e: React.ChangeEvent<HTMLTextAreaElement>) => {
  const newContent = e.target.value;
  const position = e.target.selectionStart;

  setCursorPosition(position);
  onChange(newContent);

  // Check for hashtag or mention triggers
  const textBeforeCursor = newContent.slice(0, position);
  const lastWord = textBeforeCursor.split(/\s/).pop() || '';

  if (enableHashtags && lastWord.startsWith('#') && lastWord.length > 1) {
    setSuggestionType('hashtag');
    setShowSuggestions(true);
  } else if (enableMentions && lastWord.startsWith('@') && lastWord.length > 1) {
    setSuggestionType('mention');
    setShowSuggestions(true);
  } else {
    setShowSuggestions(false);
    setSuggestionType(null);
  }
};

const insertText = (text: string) => {
  if (!textareaRef.current) return;

  const textarea = textareaRef.current;
  const start = textarea.selectionStart;
  const end = textarea.selectionEnd;
  const newContent = content.slice(0, start) + text + content.slice(end);

  onChange(newContent);

  // Set cursor position after inserted text
  setTimeout(() => {

```

```

    textarea.focus();
    textarea.setSelectionRange(start + text.length, start + text.length);
  }, 0);
};

const formatText = (format: 'bold' | 'italic') => {
  if (!textareaRef.current) return;

  const textarea = textareaRef.current;
  const start = textarea.selectionStart;
  const end = textarea.selectionEnd;
  const selectedText = content.slice(start, end);

  let formattedText = '';
  switch (format) {
    case 'bold':
      formattedText = `**${selectedText}**`;
      break;
    case 'italic':
      formattedText = `*${selectedText}*`;
      break;
  }

  const newContent = content.slice(0, start) + formattedText + content.slice(end);
  onChange(newContent);
};

const handleSuggestionSelect = (suggestion: string) => {
  if (!textareaRef.current || !suggestionType) return;

  const textarea = textareaRef.current;
  const textBeforeCursor = content.slice(0, cursorPosition);
  const textAfterCursor = content.slice(cursorPosition);
  const words = textBeforeCursor.split(/\s/);
  const lastWordIndex = textBeforeCursor.lastIndexOf(words[words.length - 1]);

```

```

const prefix = suggestionType === 'hashtag' ? '#' : '@';
const newContent =
  content.slice(0, lastWordIndex) +
  prefix + suggestion + ' ' +
  textAfterCursor;

onChange(newContent);
setShowSuggestions(false);
setSuggestionType(null);

if (suggestionType === 'hashtag') {
  onHashtagAdd(suggestion);
} else {
  onMentionAdd(suggestion);
}
};

return (
  <div className="space-y-3">
    <div className="flex items-center gap-2 p-2 border rounded-lg bg-muted/50">
      <Button
        variant="ghost"
        size="sm"
        onClick={() => formatText('bold')}
        className="h-8 w-8 p-0"
      >
        <Bold className="w-4 h-4" />
      </Button>
      <Button
        variant="ghost"
        size="sm"
        onClick={() => formatText('italic')}
        className="h-8 w-8 p-0"
      >
        <Italic className="w-4 h-4" />
      </Button>
    </div>
  </div>
);

```

```

<Separator orientation="vertical" className="h-6" />

{enableHashtags && (
  <Button
    variant="ghost"
    size="sm"
    onClick={() => insertText('#')}
    className="h-8 w-8 p-0"
  >
    <Hash className="w-4 h-4" />
  </Button>
)}

{enableMentions && (
  <Button
    variant="ghost"
    size="sm"
    onClick={() => insertText('@')}
    className="h-8 w-8 p-0"
  >
    <AtSign className="w-4 h-4" />
  </Button>
)}

{enableEmojis && (
  <Popover>
    <PopoverTrigger asChild>
      <Button
        variant="ghost"
        size="sm"
        className="h-8 w-8 p-0"
      >
        <Smile className="w-4 h-4" />
      </Button>
    </PopoverTrigger>
    <PopoverContent className="w-64">
      <EmojiPicker onEmojiSelect={(emoji) => insertText(emoji)} />
    </PopoverContent>
  </Popover>
)}

```



```

    </Popover>
  )}
</div>

<div className="relative">
  <Textarea
    ref={textareaRef}
    value={content}
    onChange={handleTextChange}
    placeholder={placeholder}
    className={`min-h-32 resize-none ${isOverLimit ? 'border-destructive' : ''}`}
    maxLength={maxCharacters}
  />

  {showSuggestions && suggestionType && (
    <Card className="absolute top-full left-0 right-0 z-10 mt-1 max-h-40 overflow-y-auto">
      <div className="p-2">
        {suggestions[suggestionType === 'hashtag' ? 'hashtags' : 'mentions'].map((suggestion) => (
          <button
            key={suggestion}
            onClick={() => handleSuggestionSelect(suggestion)}
            className="w-full text-left px-2 py-1 hover:bg-muted rounded text-sm"
          >
            {suggestionType === 'hashtag' ? '#' : '@'}{suggestion}
          </button>
        ))}
      </div>
    </Card>
  )}
</div>

<div className="flex justify-between items-center text-sm">
  <div className="flex gap-2">
    {enableHashtags && (

```

```

        <span className="text-muted-foreground">Use # for hashtags</span>
      )}
      {enableMentions && (
        <span className="text-muted-foreground">Use @ for mentions</span>
      )}
    </div>
    <div className={`font-medium ${isOverLimit ? 'text-destructive' : isNearLimit ? 'text-warning' : 'text-muted-foreground'}`}>
      {characterCount}/{maxCharacters}
    </div>
  </div>
);
};

```

```

const FeedWritingForm: React.FC<FeedWritingFormProps> = ({
  onSubmit,
  onSave,
  maxImages = 10,
  maxCharacters = 2200,
  theme = 'light',
  autoSave = true,
  enableHashtags = true,
  enableMentions = true,
  enableEmojis = true,
  placeholder = "What's on your mind?",
  className = ''
}) => {
  const [formData, setFormData] = useState<FeedFormData>({
    content: '',
    images: [],
    hashtags: [],
    mentions: []
  });

  const [isSubmitting, setIsSubmitting] = useState(false);

```

```

const [isSaving, setIsSaving] = useState(false);
const [lastSaved, setLastSaved] = useState<Date | null>(null);
const [notification, setNotification] = useState<{
  type: 'success' | 'error';
  message: string;
} | null>(null);
const [editingImage, setEditingImage] = useState<ImageData | null>(null);

// Auto-save functionality
useEffect(() => {
  if (!autoSave) return;

  const timer = setTimeout(() => {
    handleSave();
  }, 2000);

  return () => clearTimeout(timer);
}, [formData, autoSave]);

const showNotification = (type: 'success' | 'error', message: string) => {
  setNotification({ type, message });
  setTimeout(() => setNotification(null), 3000);
};

const generateImageId = () => Math.random().toString(36).substr(2, 9);

const handleImagesAdd = async (files: File[]) => {
  const remainingSlots = maxImages - formData.images.length;
  const filesToProcess = files.slice(0, remainingSlots);

  if (files.length > remainingSlots) {
    showNotification('error', `Only ${remainingSlots} more images can be added`);
  }

  const newImages: ImageData[] = [];

  for (const file of filesToProcess) {

```

```

if (!file.type.startsWith('image/')) {
  showNotification('error', `${file.name} is not a valid image file`);
  continue;
}

if (file.size > 10 * 1024 * 1024) { // 10MB limit
  showNotification('error', `${file.name} is too large (max 10MB)`);
  continue;
}

const imageId = generateImageId();
const preview = URL.createObjectURL(file);

const imageData: ImageData = {
  id: imageId,
  file,
  url: preview,
  altText: '',
  preview,
  uploadProgress: 0
};

newImages.push(imageData);

// Simulate upload progress
const uploadInterval = setInterval(() => {
  setFormData(prev => ({
    ...prev,
    images: prev.images.map(img =>
      img.id === imageId
        ? { ...img, uploadProgress: Math.min((img.uploadProgress || 0) + 1
0, 100) }
        : img
    )
  }));
}, 100);

setTimeout(() => {

```

```

clearInterval(uploadInterval);
setFormData(prev ⇒ ({
  ...prev,
  images: prev.images.map(img ⇒
    img.id === imageId
    ? { ...img, uploadProgress: 100 }
    : img
  )
}));
}, 1000);
}

setFormData(prev ⇒ ({
  ...prev,
  images: [...prev.images, ...newImages]
}));
};

const handleImageRemove = (id: string) ⇒ {
  setFormData(prev ⇒ ({
    ...prev,
    images: prev.images.filter(img ⇒ img.id !== id)
  }));
};

const handleImageReorder = (fromIndex: number, toIndex: number) ⇒ {
  setFormData(prev ⇒ {
    const newImages = [...prev.images];
    const [movedImage] = newImages.splice(fromIndex, 1);
    newImages.splice(toIndex, 0, movedImage);
    return { ...prev, images: newImages };
  });
};

const handleImageEdit = (id: string) ⇒ {
  const image = formData.images.find(img ⇒ img.id === id);
  if (image) {
    setEditingImage(image);
  }
};

```

```

    }
  };

const handleImageCropSave = (croppedImage: string) => {
  if (!editingImage) return;

  setFormData(prev => ({
    ...prev,
    images: prev.images.map(img =>
      img.id === editingImage.id
        ? { ...img, preview: croppedImage, url: croppedImage }
        : img
    )
  }));
  setEditingImage(null);
};

const handleAltTextChange = (id: string, altText: string) => {
  setFormData(prev => ({
    ...prev,
    images: prev.images.map(img =>
      img.id === id ? { ...img, altText } : img
    )
  }));
};

const handleContentChange = (content: string) => {
  setFormData(prev => ({ ...prev, content }));
};

const handleHashtagAdd = (hashtag: string) => {
  setFormData(prev => ({
    ...prev,
    hashtags: [...new Set([...prev.hashtags, hashtag])]
  }));
};

const handleMentionAdd = (mention: string) => {

```

```

setFormData(prev => ({
  ...prev,
  mentions: [...new Set([...prev.mentions, mention])]
}));
};

const handleSave = async () => {
  if (isSaving) return;

  setIsSaving(true);
  try {
    await new Promise(resolve => setTimeout(resolve, 500)); // Simulate API
    call
    onSave?.(formData);
    setLastSaved(new Date());
    showNotification('success', 'Draft saved');
  } catch (error) {
    showNotification('error', 'Failed to save draft');
  } finally {
    setIsSaving(false);
  }
};

const handleSubmit = async () => {
  if (isSubmitting) return;

  if (!formData.content.trim() && formData.images.length === 0) {
    showNotification('error', 'Please add some content or images');
    return;
  }

  if (formData.content.length > maxCharacters) {
    showNotification('error', 'Content exceeds character limit');
    return;
  }

  setIsSubmitting(true);
  try {

```

```

    await new Promise(resolve => setTimeout(resolve, 1000)); // Simulate A
    PI call
    onSubmit?.(formData);
    showNotification('success', 'Post published successfully!');

    // Reset form
    setFormData({
      content: '',
      images: [],
      hashtags: [],
      mentions: []
    });
  } catch (error) {
    showNotification('error', 'Failed to publish post');
  } finally {
    setIsSubmitting(false);
  }
};

const isValidForm = formData.content.trim() || formData.images.length >
0;

return (
  <TooltipProvider>
    <div className={`max-w-4xl mx-auto p-6 space-y-6 ${className}`}>
      {notification && (
        <Alert variant={notification.type === 'error' ? 'destructive' : 'default'}
      >
        {notification.type === 'success' ? (
          <CheckCircle className="w-4 h-4" />
        ) : (
          <AlertCircle className="w-4 h-4" />
        )}
        <AlertDescription>{notification.message}</AlertDescription>
      </Alert>
    )}

    <Card className="p-6">

```



```

<div className="space-y-6">
  <div>
    <Label htmlFor="content" className="text-lg font-semibold">
      Create Your Post
    </Label>
    <p className="text-sm text-muted-foreground mt-1">
      Share your thoughts, add images, and engage with your audience
    </p>
  </div>

  <RichTextEditor
    content={formData.content}
    onChange={handleContentChange}
    onHashtagAdd={handleHashtagAdd}
    onMentionAdd={handleMentionAdd}
    maxCharacters={maxCharacters}
    placeholder={placeholder}
    enableHashtags={enableHashtags}
    enableMentions={enableMentions}
    enableEmojis={enableEmojis}
  />

  <Separator />

  <div>
    <Label className="text-base font-medium mb-4 block">
      Add Images ({formData.images.length}/{maxImages})
    </Label>
    <ImageUploadArea
      images={formData.images}
      onImagesAdd={handleImagesAdd}
      onImageRemove={handleImageRemove}
      onImageReorder={handleImageReorder}
      onImageEdit={handleImageEdit}
      onAltTextChange={handleAltTextChange}
      maxImages={maxImages}
    />
  </div>

```

```

    {(formData.hashtags.length > 0 || formData.mentions.length > 0) &&
  (
    <>
    <Separator />
    <div className="space-y-3">
      {formData.hashtags.length > 0 && (
        <div>
          <Label className="text-sm font-medium">Hashtags</Label>
        >
          <div className="flex flex-wrap gap-2 mt-2">
            {formData.hashtags.map((hashtag, index) => (
              <Badge key={index} variant="secondary">
                #{hashtag}
              </Badge>
            ))}
          </div>
        </div>
      )}
    </div>
    {formData.mentions.length > 0 && (
      <div>
        <Label className="text-sm font-medium">Mentions</Label>
      >
        <div className="flex flex-wrap gap-2 mt-2">
          {formData.mentions.map((mention, index) => (
            <Badge key={index} variant="outline">
              @{mention}
            </Badge>
          ))}
        </div>
      </div>
    )}
  </div>
</>
)}

<Separator />

```

```

    <div className="flex flex-col sm:flex-row justify-between items-st
art sm:items-center gap-4">
      <div className="flex items-center gap-4 text-sm text-muted-fore
ground">
        {autoSave && (
          <div className="flex items-center gap-2">
            {isSaving ? (
              <>
                <Loader2 className="w-4 h-4 animate-spin" />
                <span>Saving...</span>
              </>
            ) : lastSaved ? (
              <>
                <CheckCircle className="w-4 h-4 text-green-500" />
                <span>Saved {lastSaved.toLocaleTimeString()}</span>
              </>
            ) : null}
          </div>
        )}
      </div>

      <div className="flex gap-3">
        <Button
          variant="outline"
          onClick={handleSave}
          disabled={isSaving || !isFormValid}
        >
          {isSaving ? (
            <Loader2 className="w-4 h-4 animate-spin mr-2" />
          ) : (
            <Save className="w-4 h-4 mr-2" />
          )}
          Save Draft
        </Button>

        <Button
          onClick={handleSubmit}

```

```

        disabled={isSubmitting || !isFormValid}
        className="min-w-24"
      >
        {isSubmitting ? (
          <Loader2 className="w-4 h-4 animate-spin mr-2" />
        ) : (
          <Send className="w-4 h-4 mr-2" />
        )}
        Publish
      </Button>
    </div>
  </div>
</div>
</Card>

{editingImage && (
  <ImageCropDialog
    image={editingImage}
    isOpen={!editingImage}
    onClose={() => setEditingImage(null)}
    onSave={handleImageCropSave}
  />
)}
</div>
</TooltipProvider>
);
};

export default FeedWritingForm;

```


2. 피드 스크롤하면서 열람하는, 좋아요 표시가 가능한 UI form tsx 파일.

"build me the feed viewing form UI design with tsx script which includes feed pictures enabling scrolling to side direction to see all the photos of each feed

component and feed introduction below of it for each feed component. User can see the various feeds through scrolling down of the UI."

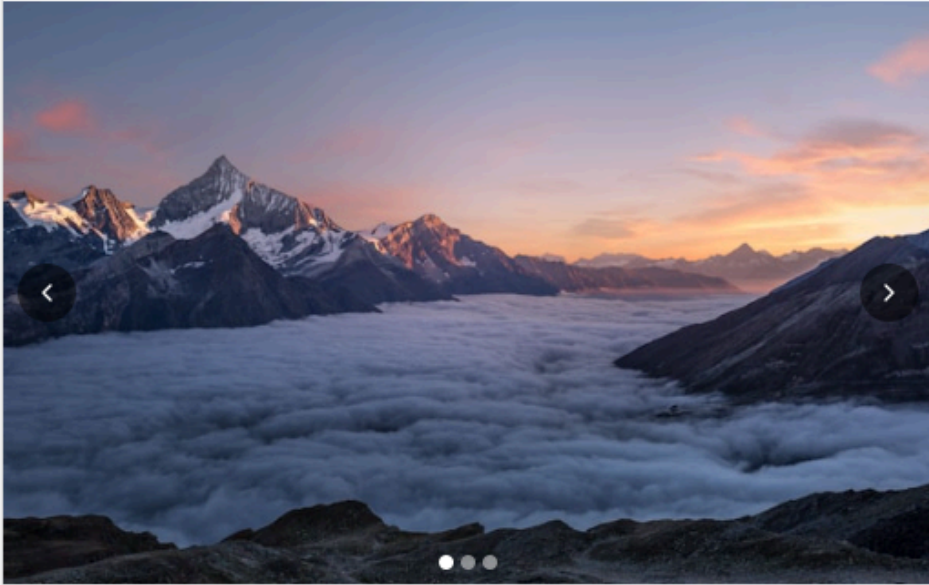
UI design 은 아래와 같아요 😊


inclu...





Sarah Johnson
@sarahj_photo · 2h ago


...




 235

 18

 12





sarahj_photo Amazing hiking trip through the mountains! The views were absolutely breathtaking. Can't wait to go back next weekend! 🏔️ #hiking #nature #adventure



Alex Chen
@alexc_food · 4h ago


...


 Delicious pizza



bolt

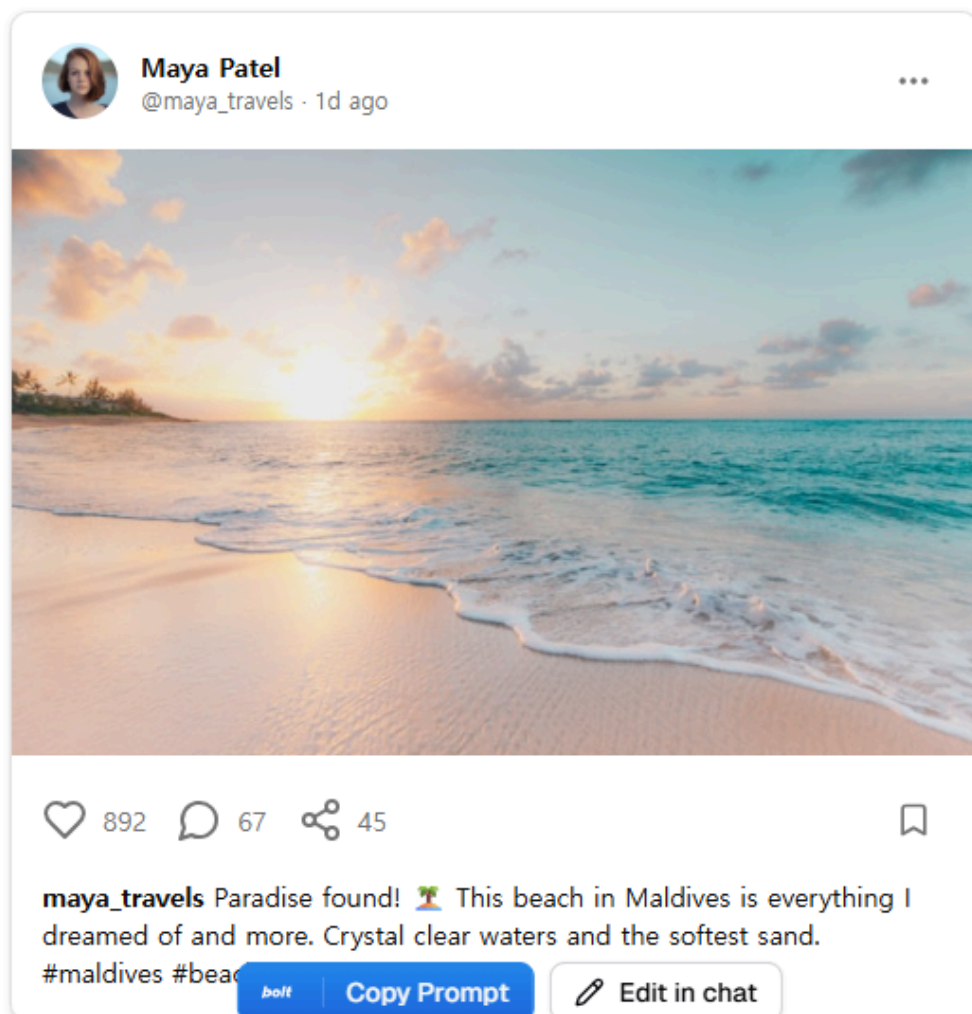
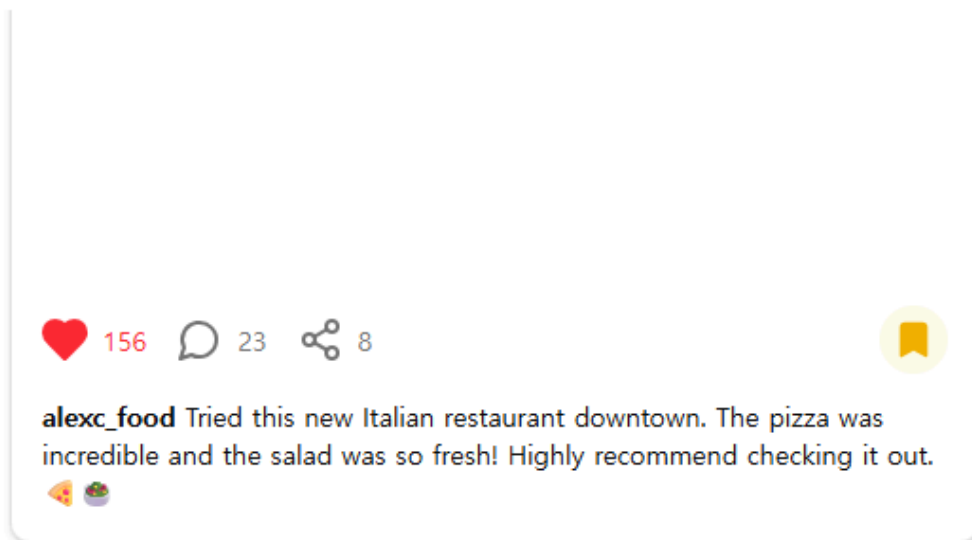
Copy Prompt

 Edit in chat



피드 작성, 피드 열람 UI form 에 대한 tsx 코드 소개_ 정동진.

30



위의 디자인 ui 에 대한 소스 코드는 아래와 같습니다 😊

```
"use client";
```

```
import { useState, useRef, useEffect } from "react";
```

```
import { Heart, MessageCircle, Share2, Bookmark, MoreHorizontal, ChevronLeft, ChevronRight } from "lucide-react";
```

```
import { cn } from "@/lib/utils";
```

```
interface FeedImage {
```

```
  id: string;
```

```
  url: string;
```

```
  alt: string;
```

```
}
```

```
interface FeedAuthor {
```

```
  name: string;
```

```
  username: string;
```

```
  avatar: string;
```

```
  timeAgo: string;
```

```
}
```

```
interface FeedEngagement {
```

```
  likes: number;
```

```
  comments: number;
```

```
  shares: number;
```

```
  isLiked: boolean;
```

```
  isBookmarked: boolean;
```

```
}
```

```
interface FeedPost {
```

```
  id: string;
```

```
  author: FeedAuthor;
```

```
  images: FeedImage[];
```

```
  caption: string;
```

```
  engagement: FeedEngagement;
```

```
}
```

```
interface ImageCarouselProps {
```

```
  images: FeedImage[];
```

```
  className?: string;
```



```

}

function ImageCarousel({ images, className }: ImageCarouselProps) {
  const [currentIndex, setCurrentIndex] = useState(0);
  const scrollRef = useRef<HTMLDivElement>(null);

  const scrollToImage = (index: number) => {
    if (scrollRef.current) {
      const imageWidth = scrollRef.current.clientWidth;
      scrollRef.current.scrollTo({
        left: index * imageWidth,
        behavior: 'smooth'
      });
    }
    setCurrentIndex(index);
  };

  const nextImage = () => {
    const nextIndex = (currentIndex + 1) % images.length;
    scrollToImage(nextIndex);
  };

  const prevImage = () => {
    const prevIndex = currentIndex === 0 ? images.length - 1 : currentIndex - 1;
    scrollToImage(prevIndex);
  };

  const handleScroll = () => {
    if (scrollRef.current) {
      const imageWidth = scrollRef.current.clientWidth;
      const scrollLeft = scrollRef.current.scrollLeft;
      const index = Math.round(scrollLeft / imageWidth);
      setCurrentIndex(index);
    }
  };

  if (images.length === 0) return null;

```

```

return (
  <div className={cn("relative w-full", className)}>
    <div
      ref={scrollRef}
      className="flex overflow-x-auto scrollbar-hide snap-x snap-mandatory"
      style={{ scrollbarWidth: 'none', msOverflowStyle: 'none' }}
      onScroll={handleScroll}
    >
      {images.map((image, index) => (
        <div key={image.id} className="w-full flex-shrink-0 snap-start">
          <img
            src={image.url}
            alt={image.alt}
            className="w-full h-80 object-cover"
          />
        </div>
      ))}
    </div>

    {images.length > 1 && (
      <>
        <button
          onClick={prevImage}
          className="absolute left-2 top-1/2 -translate-y-1/2 bg-black/50 hover:
        >
          <ChevronLeft className="w-4 h-4" />
        </button>
        <button
          onClick={nextImage}
          className="absolute right-2 top-1/2 -translate-y-1/2 bg-black/50 hov
        >
          <ChevronRight className="w-4 h-4" />
        </button>

        <div className="absolute bottom-2 left-1/2 -translate-x-1/2 flex space-
          {images.map( (_, index) => (
            <button
              key={index}

```

```

        onClick={() => scrollToImage(index)}
        className={cn(
          "w-2 h-2 rounded-full transition-colors",
          index === currentIndex ? "bg-white" : "bg-white/50"
        )}
      />
    )}
  </div>
</>
)}
</div>
);
}

```

```

interface FeedCardProps {
  post: FeedPost;
  onLike?: (postId: string) => void;
  onComment?: (postId: string) => void;
  onShare?: (postId: string) => void;
  onBookmark?: (postId: string) => void;
  onMore?: (postId: string) => void;
  className?: string;
}

```

```

function FeedCard({
  post,
  onLike,
  onComment,
  onShare,
  onBookmark,
  onMore,
  className
}: FeedCardProps) {
  const [isLiked, setIsLiked] = useState(post.engagement.isLiked);
  const [isBookmarked, setIsBookmarked] = useState(post.engagement.isBookmarked);
  const [likes, setLikes] = useState(post.engagement.likes);

  const handleLike = () => {

```

```

    setIsLiked(!isLiked);
    setLikes(prev => isLiked ? prev - 1 : prev + 1);
    onLike?.(post.id);
  };

  const handleBookmark = () => {
    setIsBookmarked(!isBookmarked);
    onBookmark?.(post.id);
  };

  return (
    <div
      className={cn(
        "w-full max-w-lg mx-auto",
        "bg-background border border-border rounded-lg overflow-hidden",
        "shadow-sm",
        className
      )}
    >
      {/* Author Header */}
      <div className="flex items-center justify-between p-4">
        <div className="flex items-center gap-3">
          <img
            src={post.author.avatar}
            alt={post.author.name}
            className="w-10 h-10 rounded-full object-cover"
          />
          <div>
            <h3 className="text-sm font-semibold text-foreground">
              {post.author.name}
            </h3>
            <p className="text-xs text-muted-foreground">
              @{post.author.username} · {post.author.timeAgo}
            </p>
          </div>
        </div>
        <button
          onClick={() => onMore?.(post.id)}

```

```

        className="p-2 hover:bg-muted rounded-full transition-colors"
      >
        <MoreHorizontal className="w-5 h-5 text-muted-foreground" />
      </button>
    </div>

    { /* Image Carousel */ }
    <ImageCarousel images={post.images} />

    { /* Engagement Actions */ }
    <div className="p-4">
      <div className="flex items-center justify-between mb-3">
        <div className="flex items-center gap-4">
          <button
            onClick={handleLike}
            className={cn(
              "flex items-center gap-2 text-sm transition-colors",
              isLiked
                ? "text-red-500"
                : "text-muted-foreground hover:text-red-500"
            )}
          >
            <Heart
              className={cn(
                "w-6 h-6 transition-all",
                isLiked && "fill-current scale-110"
              )}
            />
            <span>{likes}</span>
          </button>
          <button
            onClick={() => onComment?.(post.id)}
            className="flex items-center gap-2 text-sm text-muted-foreground"
          >
            <MessageCircle className="w-6 h-6" />
            <span>{post.engagement.comments}</span>
          </button>
          <button

```

```

      onClick={() ⇒ onShare?.(post.id)}
      className="flex items-center gap-2 text-sm text-muted-foreground
    >
      <Share2 className="w-6 h-6" />
      <span>{post.engagement.shares}</span>
    </button>
  </div>
  <button
    onClick={handleBookmark}
    className={cn(
      "p-2 rounded-full transition-all",
      isBookmarked
        ? "text-yellow-500 bg-yellow-50 dark:bg-yellow-500/10"
        : "text-muted-foreground hover:bg-muted"
    )}
  >
    <Bookmark
      className={cn(
        "w-5 h-5 transition-transform",
        isBookmarked && "fill-current scale-110"
      )}
    />
  </button>
</div>

  { /* Caption */ }
  <div className="text-sm text-foreground">
    <span className="font-semibold">{post.author.username}</span>{" "
    {post.caption}
  </div>
</div>
</div>
);
}

interface FeedViewerProps {
  posts?: FeedPost[];
  className?: string;

```

```

}

function FeedViewer({ posts = [], className }: FeedViewerProps) {
  const handleAction = (postId: string, action: string) => {
    console.log(`Post ${postId}: ${action}`);
  };

  return (
    <div className={cn("w-full max-w-2xl mx-auto", className)}>
      <div className="space-y-6">
        {posts.map((post) => (
          <FeedCard
            key={post.id}
            post={post}
            onLike={(id) => handleAction(id, 'liked')}
            onComment={(id) => handleAction(id, 'commented')}
            onShare={(id) => handleAction(id, 'shared')}
            onBookmark={(id) => handleAction(id, 'bookmarked')}
            onMore={(id) => handleAction(id, 'more')}
          />
        ))}
      </div>
    </div>
  );
}

export default function FeedViewerDemo() {
  const samplePosts: FeedPost[] = [
    {
      id: "1",
      author: {
        name: "Sarah Johnson",
        username: "sarahj_photo",
        avatar: "https://images.unsplash.com/photo-1494790108755-2616b612bf7d",
        timeAgo: "2h ago"
      },
      images: [
        {

```

```

    id: "img1",
    url: "https://images.unsplash.com/photo-1506905925346-21bda4d32df",
    alt: "Mountain landscape"
  },
  {
    id: "img2",
    url: "https://images.unsplash.com/photo-1469474968028-56623f02e42",
    alt: "Forest path"
  },
  {
    id: "img3",
    url: "https://images.unsplash.com/photo-1441974231531-c6227db76b6e",
    alt: "River valley"
  }
],
caption: "Amazing hiking trip through the mountains! The views were abso
engagement: {
  likes: 234,
  comments: 18,
  shares: 12,
  isLiked: false,
  isBookmarked: false
}
},
{
  id: "2",
  author: {
    name: "Alex Chen",
    username: "alexc_food",
    avatar: "https://images.unsplash.com/photo-1507003211169-0a1dd7228f",
    timeAgo: "4h ago"
  },
  images: [
    {
      id: "img4",
      url: "https://images.unsplash.com/photo-1565299624946-b28f40a0ca4",
      alt: "Delicious pizza"
    },
  ],

```



```

    {
      id: "img5",
      url: "https://images.unsplash.com/photo-1571997478779-2adcbbe9ab21",
      alt: "Fresh salad"
    }
  ],
  caption: "Tried this new Italian restaurant downtown. The pizza was incred",
  engagement: {
    likes: 156,
    comments: 23,
    shares: 8,
    isLiked: true,
    isBookmarked: true
  }
},
{
  id: "3",
  author: {
    name: "Maya Patel",
    username: "maya_travels",
    avatar: "https://images.unsplash.com/photo-1438761681033-6461ffad8d",
    timeAgo: "1d ago"
  },
  images: [
    {
      id: "img6",
      url: "https://images.unsplash.com/photo-1507525428034-b723cf961d3",
      alt: "Tropical beach"
    }
  ],
  caption: "Paradise found! 🌴 This beach in Maldives is everything I dreamed",
  engagement: {
    likes: 892,
    comments: 67,
    shares: 45,
    isLiked: false,
    isBookmarked: false
  }
}

```

```
    }  
  ];  
  
  return (  
    <div className="min-h-screen bg-background p-4">  
      <FeedViewer posts={samplePosts} />  
    </div>  
  );  
}
```