Image Upload Flow in the Property Listing Wizard

Overview

This document details the complete image upload process in the Bhoomitalli Real Estate Platform's Property Listing Wizard, including frontend interaction, backend storage, and database mappings between properties and their images.

1. Image Upload Mechanics

The image upload process is primarily handled by three key components:

1. **ImageUploadSection.tsx** - The main UI component that displays the upload area and image grid
2. **UploadArea.tsx** - Handles the drag-and-drop and file input functionality
3. **useImageUpload.ts** - The hook that manages all image uploading, fetching, and management operations

Upload Flow

The flow works as follows:

- Users can upload images either by clicking on the upload area or by dragging and dropping files
- The `UploadArea` component handles drag events and file selection
- When files are selected, they're passed to the `handleFileSelect` function in the `useImageUpload` hook
- The hook validates the files (size, type) and then uploads them directly to Supabase storage
- After successful upload, the file information is stored in the `property_images` table with a reference to the property ID

Key Constraints

- Maximum 10 photos per property
- Maximum 5MB per image
- Only JPG and PNG formats are supported

Code Highlights: File Selection in UploadArea.tsx

```javascript
const handleFileInputChange = useCallback((e: React.ChangeEvent<HTMLInputElement>) =>
  const files = Array.from(e.target.files || []);
  onFileSelect(files);
}, [onFileSelect]);
```

Code Highlights: Drag and Drop Handling

```javascript
const handleDrop = useCallback((e: React.DragEvent<HTMLDivElement>) => {
  e.preventDefault();
  e.stopPropagation();

  setIsDragging(false);
  setIsGlobalDragging(false);

  if (disabled) return;

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

  onFileSelect(droppedFiles);
}, [disabled, onFileSelect]);
```

2. Transition from Review Tab & Property ID Linkage

Before users can upload images, they need to create a property record. This happens when:

1. The user completes the property details form and clicks "Save and Upload Photos" button
2. The `handleSaveAsDraft` or `handleSaveAndPublish` functions in `usePropertyFormOperations.ts` are triggered
3. These functions:
   - Collect all form data
   - Add essential fields for property type (sale or rental)
   - Insert or update the property record in the database
   - Store the property ID (from the newly created record) in state

Property ID Flow

The critical part is how the property ID is passed to the Image Upload section:

- In `PropertyForm/index.tsx`, when the form submission is successful, the `savedPropertyId` is stored in state
- This ID is then passed to the `ImageUploadSection` component via the `FormContent` component
- The `ImageUploadSection` component receives the `propertyId` as a prop
- The `useImageUpload` hook uses this ID to associate uploaded images with the property

Code Highlights: Property Saving in usePropertyFormOperations.ts

javascript

Copy

```javascript
const { data: newProperty, error: createError } = await supabase
  .from('properties')
  .insert([propertyData])
  .select()
  .single();

if (createError) throw createError;
savedPropertyId = newProperty.id;
setSavedPropertyId(newProperty.id);
```

3. Database Table Interactions

The database uses the following tables for property and image management:

Properties Table

- Stores all property details including owner_id, price, status, etc.
- Contains a JSONB `property_details` field that stores all form data
- Key fields include:
  - `id` - Primary key
  - `owner_id` - Foreign key to profiles table
  - `title` - Property title
  - `price` - Price of the property
  - `status` - 'draft' or 'published'
  - `property_details` - JSONB data with all form fields

Property_Images Table

- Linked to properties via a foreign key constraint (`property_id` references `properties.id`)
- Stores image metadata including:
  - `id` - Unique identifier for the image

- `property_id` - Reference to the property
- `url` - Public URL to the image in Supabase storage
- `is_primary` - Boolean flag to mark the primary (featured) image
- `display_order` - Integer that determines the display order

Database-Storage Workflow

1. Images are uploaded to Supabase storage in the "property-images" bucket
2. The file path follows a pattern `${propertyId}/${timestamp}-${random}.${fileExt}`
3. After successful upload, a public URL is generated
4. This URL, along with the property ID, is stored in the `property_images` table
5. When fetching property data, the images are also fetched and returned with the property

Code Highlights: Image Upload and Database Storage

javascript

Copy

```javascript
// Upload to storage
const { error: uploadError } = await supabase.storage
  .from('property-images')
  .upload(fileName, file, {
    cacheControl: '3600',
    upsert: false
  });

// Get public URL
const { data: { publicUrl } } = supabase.storage
  .from('property-images')
  .getPublicUrl(fileName);

// Store image metadata in database
const { error: dbError, data: newImage } = await supabase
  .from('property_images')
  .insert([{
    property_id: propertyId,
    url: publicUrl,
    is_primary: startIndex + idx === primaryImageIndex,
    display_order: startIndex + idx
  }])
  .select()
  .single();
```

4. Upload Limit Enforcement

The 10-photo limit is enforced in several places:

Frontend Validation

In the `useImageUpload.ts` hook:

javascript Copy

```javascript
const totalImages = images.length + existingImages.length + newFiles.length;
if (totalImages > MAX_IMAGES) {
  setError(`Maximum ${MAX_IMAGES} images allowed`);
  return;
}
```

UI Feedback

In the `UploadArea.tsx` component:

javascript Copy

```javascript
{images.length === 0
  ? "Click or drag images here"
  : `Add more images (${MAX_IMAGES - images.length} remaining)`}
```

Input Disabling

The file input is also disabled when the limit is reached:

javascript Copy

```javascript
<input
  type="file"
  multiple
  accept="image/*"
  onChange={handleFileInputChange}
  className="absolute inset-0 w-full h-full opacity-0 cursor-pointer"
  disabled={images.length >= MAX_IMAGES || disabled}
/>
```

5. Primary Image Selection

The system also allows setting a primary image that will be displayed first:

1. When images are uploaded, the first image is automatically set as primary
2. Users can change the primary image by clicking the star icon on any image

3. This triggers the `handleSetPrimaryImage` function in `useImageUpload.ts`
4. The function updates the database, setting all other images to non-primary and the selected one to primary

Code Highlights: Setting Primary Image

javascript

Copy

```javascript
const updatePrimaryImage = async (imageId: string) => {
  try {
    // First, set all images as non-primary
    await supabase
      .from('property_images')
      .update({ is_primary: false })
      .eq('property_id', propertyId);

    // Then set the selected image as primary
    const { error } = await supabase
      .from('property_images')
      .update({ is_primary: true })
      .eq('id', imageId);

    if (error) throw error;
  } catch (err) {
    console.error('Error updating primary image:', err);
    setError('Failed to update primary image');
  }
};
```

6. Error Handling and User Feedback

The system provides comprehensive error handling and user feedback:

1. **Upload Progress**: Shows a progress bar during upload
2. **Error Messages**: Displays clear error messages if something goes wrong
3. **Image Count**: Shows how many more images can be uploaded
4. **Loading State**: Shows a loading spinner when fetching existing images

Code Highlights: Error Display in ImageUploadSection.tsx

javascript

Copy

```javascript
{error && (
  <div className="flex items-center gap-2 text-destructive bg-destructive/10 rounded-l
    <AlertCircle className="h-5 w-5 flex-shrink-0" />
    <p className="text-sm">{error}</p>
  </div>
)}
```

7. UI Components and Interactions

Image Grid

- Displays all uploaded images in a responsive grid
- Shows image number and primary status
- Provides hover actions for setting primary and removing images

Image Upload Area

- Supports both drag-and-drop and click-to-select
- Provides visual feedback during drag operations
- Shows remaining upload capacity

Tips and Requirements

- Shows photo requirements and tips to the user
- Informs about size limits, supported formats, etc.

Conclusion

The image upload system in the Property Listing Wizard provides a comprehensive solution for:

1. **Property Creation**: User fills out property details and saves/publishes the property
2. **Property ID**: The newly created property ID is passed to the Image Upload section
3. **Image Upload**: User uploads images via drag-and-drop or file selection
4. **Storage**: Images are stored in Supabase storage
5. **Database**: Image metadata (including URLs) is stored in the property_images table with a reference to the property
6. **UI Feedback**: User receives real-time feedback during upload, can set a primary image, and can remove images

This system follows best practices for image handling, provides a good user experience, and ensures data integrity between properties and their images.