Profile Image Upload - Frontend Implementation Guide

Overview

This guide helps frontend developers implement user profile image upload functionality using the Convex backend APIs. The implementation follows a simple two-step process that's secure and efficient.

Quick Start

Required Convex Hooks

```
import { useMutation, useQuery } from "convex/react";
import { api } from "../convex/_generated/api";

const generateUploadUrl =
useMutation(api.users.generateProfileImageUploadUrl);
const updateProfileImage = useMutation(api.users.updateProfileImage);
const removeProfileImage = useMutation(api.users.removeProfileImage);
const getImageUrl = useQuery(api.users.getProfileImageUrl,
  user?.image ? { imageStorageId: user.image } : "skip"
);
```

Basic Implementation

```
import { useState } from "react";

function ProfileImageUpload({ user }) {
  const [uploading, setUploading] = useState(false);

  const generateUploadUrl =
useMutation(api.users.generateProfileImageUploadUrl);
  const updateProfileImage = useMutation(api.users.updateProfileImage);
  const removeProfileImage = useMutation(api.users.removeProfileImage);

  // Get current image URL if user has one
  const imageUrl = useQuery(api.users.getProfileImageUrl,
    user?.image ? { imageStorageId: user.image } : "skip"
  );

  const handleFileUpload = async (file) => {
    if (!file) return;

    setUploading(true);

    try {
      // Step 1: Get upload URL
      const uploadUrl = await generateUploadUrl();
```

```
      // Step 2: Upload file to Convex storage
      const uploadResponse = await fetch(uploadUrl, {
        method: "POST",
        headers: { "Content-Type": file.type },
        body: file,
      });

      if (!uploadResponse.ok) {
        throw new Error("Upload failed");
      }

      const { storageId } = await uploadResponse.json();

      // Step 3: Update user profile with storage ID
      await updateProfileImage({ imageStorageId: storageId });

    } catch (error) {
      console.error("Upload failed:", error);
      // Handle error (show toast, etc.)
    } finally {
      setUploading(false);
    }
  };

  const handleRemoveImage = async () => {
    try {
      await removeProfileImage();
    } catch (error) {
      console.error("Remove failed:", error);
    }
  };

  return (
    <div className="profile-image-upload">
      {/* Display current image */}
      {imageUrl && (
        <div>
          <img src={imageUrl} alt="Profile" className="w-32 h-32 rounded-
full" />
          <button onClick={handleRemoveImage}>Remove Image</button>
        </div>
      )}

      {/* Upload new image */}
      <input
        type="file"
        accept="image/*"
        onChange={(e) => handleFileUpload(e.target.files[0])}
        disabled={uploading}
      />

      {uploading && <p>Uploading...</p>}
    </div>
```

```
    );
}
```

Available APIs

1. Generate Upload URL

```
const uploadUrl = await generateUploadUrl();
```

- **Purpose**: Gets a temporary URL for uploading files
- **Auth**: Requires user authentication
- **Returns**: String URL for uploading

2. Update Profile Image

```
await updateProfileImage({ imageStorageId: "storage_id_here" });
```

- **Purpose**: Saves the uploaded image to user profile
- **Auth**: Requires user authentication
- **Params**: `imageStorageId` - The storage ID returned from upload
- **Returns**: Updated user object

3. Get Image URL

```
const imageUrl = useQuery(api.users.getProfileImageUrl, { imageStorageId:
"storage_id" });
```

- **Purpose**: Gets public URL for displaying image
- **Auth**: No auth required
- **Params**: `imageStorageId` - The storage ID from user.image
- **Returns**: Public URL string

4. Remove Profile Image

```
await removeProfileImage();
```

- **Purpose**: Removes user's profile image
- **Auth**: Requires user authentication
- **Returns**: Updated user object (with image set to undefined)

Error Handling

Common Errors and Solutions

```
const handleFileUpload = async (file) => {
  try {
    // ... upload logic
  } catch (error) {
    if (error.message.includes("Not authenticated")) {
      // Redirect to login
      router.push("/login");
    } else if (error.message.includes("Upload failed")) {
      // Show user-friendly error
      setError("Failed to upload image. Please try again.");
    } else {
      // Generic error
      setError("Something went wrong. Please try again.");
    }
  }
};
```

File Validation

Recommended Validations

```
const validateFile = (file) => {
  const maxSize = 5 * 1024 * 1024; // 5MB
  const allowedTypes = ['image/jpeg', 'image/png', 'image/webp'];

  if (!file) {
    throw new Error("Please select a file");
  }

  if (file.size > maxSize) {
    throw new Error("File size must be less than 5MB");
  }

  if (!allowedTypes.includes(file.type)) {
    throw new Error("Only JPEG, PNG, and WebP images are allowed");
  }

  return true;
};

const handleFileUpload = async (file) => {
  try {
    validateFile(file);
    // ... rest of upload logic
  } catch (error) {
    setError(error.message);
  }
};
```

Complete React Component Example

```
import { useState } from "react";
import { useMutation, useQuery } from "convex/react";
import { api } from "../convex/_generated/api";

function ProfileImageUpload({ user }) {
  const [uploading, setUploading] = useState(false);
  const [error, setError] = useState("");

  const generateUploadUrl =
useMutation(api.users.generateProfileImageUploadUrl);
  const updateProfileImage = useMutation(api.users.updateProfileImage);
  const removeProfileImage = useMutation(api.users.removeProfileImage);

  const imageUrl = useQuery(api.users.getProfileImageUrl,
    user?.image ? { imageStorageId: user.image } : "skip"
  );

  const validateFile = (file) => {
    const maxSize = 5 * 1024 * 1024; // 5MB
    const allowedTypes = ['image/jpeg', 'image/png', 'image/webp'];

    if (!file) throw new Error("Please select a file");
    if (file.size > maxSize) throw new Error("File size must be less than
5MB");
    if (!allowedTypes.includes(file.type)) {
      throw new Error("Only JPEG, PNG, and WebP images are allowed");
    }
  };

  const handleFileUpload = async (file) => {
    setError("");
    setUploading(true);

    try {
      validateFile(file);

      const uploadUrl = await generateUploadUrl();

      const uploadResponse = await fetch(uploadUrl, {
        method: "POST",
        headers: { "Content-Type": file.type },
        body: file,
      });

      if (!uploadResponse.ok) {
        throw new Error("Upload failed");
      }
```

```
        const { storageId } = await uploadResponse.json();
        await updateProfileImage({ imageStorageId: storageId });

      } catch (error) {
        setError(error.message);
      } finally {
        setUploading(false);
      }
    };

    const handleRemoveImage = async () => {
      try {
        await removeProfileImage();
      } catch (error) {
        setError("Failed to remove image");
      }
    };

    return (
      <div className="space-y-4">
        {/* Current Image Display */}
        {imageUrl && (
          <div className="flex flex-col items-center space-y-2">
            <img
              src={imageUrl}
              alt="Profile"
              className="w-32 h-32 rounded-full object-cover border-4
border-gray-200"
            />
            <button
              onClick={handleRemoveImage}
              className="text-red-600 text-sm hover:text-red-800"
            >
              Remove Image
            </button>
          </div>
        )}

        {/* Upload Interface */}
        <div className="flex flex-col items-center space-y-2">
          <label className="cursor-pointer bg-blue-500 text-white px-4 py-2
rounded hover:bg-blue-600">
            {uploading ? "Uploading..." : "Choose Image"}
            <input
              type="file"
              accept="image/*"
              onChange={(e) => handleFileUpload(e.target.files[0])}
              disabled={uploading}
              className="hidden"
            />
          </label>

          {error && (
            <p className="text-red-600 text-sm">{error}</p>
```

```
          )}
        </div>
      </div>
    );
}

export default ProfileImageUpload;
```

Integration Tips

1. Using with Forms

```
// In a profile edit form
const ProfileForm = () => {
  const { user } = useAuthContext();

  return (
    <form>
      <ProfileImageUpload user={user} />
      {/* Other form fields */}
    </form>
  );
};
```

2. Loading States

```
// Show skeleton while image URL is loading
if (user?.image && imageUrl === undefined) {
  return <div className="w-32 h-32 bg-gray-200 rounded-full animate-pulse"
/>;
}
```

3. Responsive Design

```
// Different sizes for different screen sizes
<img
  src={imageUrl}
  alt="Profile"
  className="w-16 h-16 md:w-32 md:h-32 rounded-full object-cover"
/>
```

Testing

Manual Testing Checklist

- [ ] Upload image works
- [ ] Image displays correctly
- [ ] Remove image works
- [ ] File validation works (size, type)
- [ ] Error messages display
- [ ] Loading states work
- [ ] Works on mobile devices
- [ ] Works with slow internet

Error Scenarios to Test

- [ ] Upload without authentication
- [ ] Upload oversized file
- [ ] Upload invalid file type
- [ ] Network failure during upload
- [ ] Remove image without permission

Performance Considerations

1. **Image Optimization**: Consider resizing images on the client before upload
2. **Lazy Loading**: Use `"skip"` parameter for useQuery when no image exists
3. **Caching**: Convex handles URL caching automatically
4. **File Size**: Recommend users keep images under 5MB

Security Notes

1. **Authentication**: All mutations require user authentication
2. **File Validation**: Always validate on frontend AND backend
3. **Storage IDs**: Never expose storage IDs publicly - they're internal references
4. **URLs**: Public URLs from `getProfileImageUrl` are safe to use anywhere