Complete AWS Integration for Replit Upload Center

🎯 YOUR EXISTING AWS INFRASTRUCTURE

You have everything set up:

- ✅ **S3 Buckets**: `familyportal-docs-prod` & `familyportal-photos-prod`
- ✅ **IAM Users**: `family-portal-storage`, `FamilyPortalTextractS3Read`
- ✅ **Lambda Function**: With OpenAI API integration
- ✅ **Textract Access**: For document analysis

🔧 FOR REPLIT AGENT - COMPLETE INTEGRATION

Tell Replit:

"I have complete AWS infrastructure set up. Integrate my Upload Center with my existing S3 buckets and Lambda function for real document processing. Here's the exact integration:"

Step 1: S3 Upload Integration

JSX

```jsx
const uploadToS3 = async (file, fileType) => {
  // Determine bucket based on file type
  const bucket = fileType.startsWith('image/')
    ? 'familyportal-photos-prod'
    : 'familyportal-docs-prod';

  const formData = new FormData();
  formData.append('file', file);
  formData.append('bucket', bucket);
  formData.append('key', `uploads/${Date.now()}-${file.name}`);

  try {
    const response = await fetch('/api/s3-upload', {
      method: 'POST',
      body: formData
    });

    const result = await response.json();
    return {
      id: generateId(),
```

```
      name: file.name,
      size: file.size,
      type: file.type,
      s3Url: result.s3Url,
      s3Key: result.s3Key,
      bucket: bucket,
      status: 'uploaded',
      uploadedAt: new Date().toISOString()
    };

  } catch (error) {
    console.error('S3 upload failed:', error);
    throw error;
  }
};
```

Step 2: Complete Upload Workflow

JSX

```
const handleFileUpload = async (files) => {
  try {
    // 1. Upload files to appropriate S3 buckets
    const uploadPromises = Array.from(files).map(file =>
      uploadToS3(file, file.type)
    );

    const uploadedDocuments = await Promise.all(uploadPromises);

    // 2. Open LEFT sidebar immediately
    setDocuments(uploadedDocuments);
    setSidebarOpen(true);

    // 3. Process each document with Lambda + Textract
    uploadedDocuments.forEach(async (doc) => {
      // Update status to analyzing
      updateDocumentStatus(doc.id, 'analyzing');

      // Call your Lambda function with S3 details
      const analysisResult = await callLambdaAnalysis(doc);

      // Update with real AI results
      updateDocumentStatus(doc.id, 'analyzed', analysisResult);
    });

  } catch (error) {
    console.error('Upload workflow failed:', error);
```

```
  }
};
```

Step 3: Lambda + Textract Analysis

JSX

```jsx
const callLambdaAnalysis = async (document) => {
  try {
    const response = await fetch('YOUR_LAMBDA_ENDPOINT', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
        'Authorization': 'Bearer YOUR_API_KEY'
      },
      body: JSON.stringify({
        s3Bucket: document.bucket,
        s3Key: document.s3Key,
        documentType: document.type,
        fileName: document.name
      })
    });

    const result = await response.json();

    return {
      // Textract OCR results
      extractedText: result.textractText,

      // OpenAI analysis results
      documentType: result.aiAnalysis.documentType,
      keyValuePairs: result.aiAnalysis.keyValuePairs,
      summary: result.aiAnalysis.summary,
      confidence: result.aiAnalysis.confidence,

      // Additional metadata
      pageCount: result.pageCount,
      language: result.language,
      processingTime: result.processingTime
    };

  } catch (error) {
    console.error('Lambda analysis failed:', error);
    return null;
  }
};
```

Step 4: File Type Routing

JSX

```jsx
const handleUploadByType = async (files, uploadMethod) => {
  const processedFiles = [];

  for (const file of files) {
    let processedFile;

    if (file.type.startsWith('image/')) {
      // Photos go to photos bucket
      processedFile = await uploadToS3(file, 'image');
      processedFile.category = 'photo';
      processedFile.icon = '📷';

    } else if (file.type === 'application/pdf' ||
               file.type.includes('document')) {
      // Documents go to docs bucket
      processedFile = await uploadToS3(file, 'document');
      processedFile.category = 'document';
      processedFile.icon = '📄';

    } else {
      // Default to docs bucket
      processedFile = await uploadToS3(file, 'document');
      processedFile.category = 'file';
      processedFile.icon = '📁';
    }

    processedFile.uploadMethod = uploadMethod;
    processedFiles.push(processedFile);
  }

  return processedFiles;
};
```

Step 5: Enhanced Document Display

JSX

```jsx
const DocumentItem = ({ document }) => {
  return (
    <div className="document-item">
      <div className="document-thumbnail">
        {document.category === 'photo' ? (
```

```
          <img src={document.s3Url} alt="Thumbnail" className="photo-thumb"
/>
        ) : (
          <div className="doc-icon">{document.icon}</div>
        )}
      </div>

      <div className="document-info">
        <h4>{document.name}</h4>
        <p className="document-meta">
          {formatFileSize(document.size)} • {document.category}
        </p>
        <p className="status">
          {document.status === 'analyzing' && '🤖 AI Analysis in
progress...'}
          {document.status === 'analyzed' && '✅ Analysis complete'}
        </p>
      </div>

      {/* Lightning bolt with analysis count */}
      {document.status === 'analyzed' && document.analysisResult && (
        <button
          className="lightning-bolt"
          onClick={() => showDetailsModal(document)}
        >
          ⚡ Details {document.analysisResult.keyValuePairs?.length || 0}
        </button>
      )}
    </div>
  );
};
```

Step 6: Enhanced Details Modal

```jsx
const DetailsModal = ({ document, onClose }) => {
  const analysis = document.analysisResult;

  return (
    <div className="details-modal">
      <div className="modal-header">
        <div className="document-header">
          <span className="doc-icon">{document.icon}</span>
          <div>
            <h3>{document.name}</h3>
            <p>Stored in: {document.bucket}</p>
```

```
          </div>
        </div>
        <button onClick={onClose}>×</button>
      </div>

      <div className="modal-content">
        {/* AI Analysis Results */}
        <div className="analysis-section">
          <h4>🤖 AI Analysis</h4>
          <div className="analysis-grid">
            <div className="analysis-item">
              <span className="label">Document Type:</span>
              <span className="value">{analysis.documentType}</span>
            </div>
            <div className="analysis-item">
              <span className="label">Confidence:</span>
              <span className="value">{analysis.confidence}%</span>
            </div>
            <div className="analysis-item">
              <span className="label">Pages:</span>
              <span className="value">{analysis.pageCount}</span>
            </div>
          </div>
        </div>

        {/* Extracted Key-Value Pairs */}
        <div className="analysis-section">
          <h4>📋 Extracted Information</h4>
          <div className="key-value-grid">
            {analysis.keyValuePairs.map((pair, index) => (
              <div key={index} className="key-value-pair">
                <span className="key">{pair.key}:</span>
                <span className="value">{pair.value}</span>
                <span className="confidence">{pair.confidence}%</span>
              </div>
            ))}
          </div>
        </div>

        {/* AI Summary */}
        {analysis.summary && (
          <div className="analysis-section">
            <h4>📝 AI Summary</h4>
            <div className="summary-text">{analysis.summary}</div>
          </div>
        )}

        {/* Textract OCR Results */}
```

```
        <div className="analysis-section">
          <h4>🔍 Extracted Text (OCR)</h4>
          <div className="extracted-text">
            {analysis.extractedText.substring(0, 1000)}
            {analysis.extractedText.length > 1000 && '...'}
          </div>
        </div>

        {/* S3 Information */}
        <div className="analysis-section">
          <h4>☁️ Storage Details</h4>
          <div className="storage-info">
            <p><strong>Bucket:</strong> {document.bucket}</p>
            <p><strong>Key:</strong> {document.s3Key}</p>
            <p><strong>Upload Method:</strong> {document.uploadMethod}</p>
            <p><strong>Upload Time:</strong> {new
Date(document.uploadedAt).toLocaleString()}</p>
          </div>
        </div>
      </div>

      <div className="modal-actions">
        <button className="accept-btn" onClick={() =>
acceptDocument(document)}>
          ✅ Accept & Route to Profile
        </button>
        <button className="view-s3-btn" onClick={() =>
window.open(document.s3Url, '_blank')}>
          ☁️ View in S3
        </button>
        <button className="dismiss-btn" onClick={onClose}>
          ❌ Dismiss
        </button>
      </div>
    </div>
  );
};
```

📋 CONFIGURATION FOR REPLIT

Provide Replit with:

1. **S3 Bucket Names:**
   - Documents: `familyportal-docs-prod`
   - Photos: `familyportal-photos-prod`

2. **IAM Credentials:**
   - Access Key ID from `family-portal-storage` user
   - Secret Access Key from `family-portal-storage` user
3. **Lambda Endpoint:**
   - Your Lambda function URL
   - Any API authentication needed
4. **AWS Region:**
   - `us-east-2` (Ohio) based on your screenshot

🚀 TELL REPLIT:

"Integrate my complete AWS infrastructure into the Upload Center workflow:

*1. **Upload files** to my S3 buckets (photos → familyportal-photos-prod, docs → familyportal-docs-prod)*
*2. **Open LEFT sidebar** with uploaded documents*

*3. **Call my Lambda function** with S3 details for Textract + OpenAI analysis*
*4. **Show real AI results** in lightning bolt details modal*
*5. **Display S3 storage info** and extracted data*

I'll provide the S3 credentials, Lambda endpoint, and AWS region. This should give me production-ready document processing with real AI analysis."

This creates a complete enterprise-grade document processing system!