S3 Credentials Setup Guide for Replit

🔑 STEP 1: Get Your IAM User Credentials

You need to get the **Access Key ID** and **Secret Access Key** from your `family-portal-storage` IAM user.

Go to IAM Console:

1. **AWS Console → IAM → Users**
2. Click on **`family-portal-storage`** user
3. Go to **"Security credentials"** tab
4. Under **"Access keys"** section:
   - If you see an existing key → **Copy the Access Key ID**
   - If no key exists → Click **"Create access key"**

Create New Access Key (if needed):

1. Click **"Create access key"**
2. Select **"Application running outside AWS"**
3. Add description: **"Replit Upload Integration"**
4. Click **"Create access key"**
5. ⚠️ **IMPORTANT:** Copy both keys immediately (you can't see Secret Key again)

🔧 STEP 2: Configure Replit Environment Variables

Tell Replit to add these environment variables:

```Bash
# AWS S3 Configuration
AWS_ACCESS_KEY_ID=AKIA... (your access key)
AWS_SECRET_ACCESS_KEY=... (your secret key)
AWS_REGION=us-east-2
AWS_S3_DOCS_BUCKET=familyportal-docs-prod
AWS_S3_PHOTOS_BUCKET=familyportal-photos-prod
```

📝 STEP 3: Replit Backend Code

Tell Replit to implement this S3 upload code:

JavaScript

```javascript
// Install AWS SDK
// npm install aws-sdk multer

const AWS = require('aws-sdk');
const multer = require('multer');

// Configure AWS
AWS.config.update({
  accessKeyId: process.env.AWS_ACCESS_KEY_ID,
  secretAccessKey: process.env.AWS_SECRET_ACCESS_KEY,
  region: process.env.AWS_REGION
});

const s3 = new AWS.S3();

// Multer memory storage
const upload = multer({ storage: multer.memoryStorage() });

// S3 Upload Function
const uploadToS3 = async (file, bucketName) => {
  const key = `uploads/${Date.now()}-${file.originalname}`;

  const params = {
    Bucket: bucketName,
    Key: key,
    Body: file.buffer,
    ContentType: file.mimetype,
    ACL: 'private' // Keep files private
  };

  try {
    const result = await s3.upload(params).promise();
    return {
      s3Url: result.Location,
      s3Key: result.Key,
      bucket: bucketName
    };
  } catch (error) {
    console.error('S3 upload error:', error);
    throw error;
  }
};

// Upload Endpoint
```

```javascript
app.post('/api/upload', upload.array('documents'), async (req, res) => {
  try {
    console.log('Files received:', req.files?.length || 0);

    if (!req.files || req.files.length === 0) {
      return res.status(400).json({ error: 'No files uploaded' });
    }

    const uploadPromises = req.files.map(async (file) => {
      // Determine bucket based on file type
      const isImage = file.mimetype.startsWith('image/');
      const bucketName = isImage
        ? process.env.AWS_S3_PHOTOS_BUCKET
        : process.env.AWS_S3_DOCS_BUCKET;

      // Upload to S3
      const s3Result = await uploadToS3(file, bucketName);

      return {
        id: `doc_${Date.now()}_${Math.random().toString(36).substr(2, 9)}`,
        name: file.originalname,
        size: file.size,
        type: file.mimetype,
        category: isImage ? 'photo' : 'document',
        icon: isImage ? '📷' : '📄',
        status: 'uploaded',
        uploadedAt: new Date().toISOString(),
        ...s3Result
      };
    });

    const documents = await Promise.all(uploadPromises);

    console.log('Upload successful:', documents.length, 'files');
    res.json({
      success: true,
      documents,
      message: `Successfully uploaded ${documents.length} file(s)`
    });

  } catch (error) {
    console.error('Upload failed:', error);
    res.status(500).json({
      error: 'Upload failed',
      message: error.message
    });
  }
});
```

🔗 STEP 4: Lambda Integration

After S3 upload works, add Lambda analysis:

JavaScript

```javascript
// Lambda Analysis Function
const callLambdaAnalysis = async (document) => {
  try {
    const response = await fetch(process.env.LAMBDA_ENDPOINT, {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
        'Authorization': `Bearer ${process.env.LAMBDA_API_KEY}` // if needed
      },
      body: JSON.stringify({
        documentUrl: document.s3Url,
        documentId: document.id,
        bucketName: document.bucket,
        s3Key: document.s3Key
      })
    });

    if (!response.ok) {
      throw new Error(`Lambda failed: ${response.status}`);
    }

    const result = await response.json();
    return result;

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

// Enhanced Upload Endpoint with Lambda
app.post('/api/upload', upload.array('documents'), async (req, res) => {
  try {
    // ... S3 upload code above ...

    const documents = await Promise.all(uploadPromises);

    // Start Lambda analysis for each document
    documents.forEach(async (doc) => {
      const analysisResult = await callLambdaAnalysis(doc);
      if (analysisResult) {
        // Store analysis result (in database or memory)
        doc.analysisResult = analysisResult;
        doc.status = 'analyzed';
      }
    });

    res.json({ success: true, documents });

  } catch (error) {
    console.error('Upload failed:', error);
    res.status(500).json({ error: 'Upload failed', message: error.message });
  }
});
```

🎯 STEP 5: Tell Replit Exactly What to Do

Copy this message to Replit:

"Replace the mock upload with real S3 integration. I'll provide you with:

- *AWS_ACCESS_KEY_ID: [YOUR_ACCESS_KEY]*
- *AWS_SECRET_ACCESS_KEY: [YOUR_SECRET_KEY]*
- *AWS_REGION: us-east-2*
- *AWS_S3_DOCS_BUCKET: familyportal-docs-prod*
- *AWS_S3_PHOTOS_BUCKET: familyportal-photos-prod*

Implement the S3 upload code I'm providing. Photos should go to familyportal-photos-prod, documents to familyportal-docs-prod. After upload, return the S3 URL and key so we can call Lambda analysis next."

🔐 SECURITY NOTES

1. **Environment Variables:** Store credentials in Replit's environment variables (Secrets tab)
2. **Private Files:** Keep S3 files private (ACL: 'private')
3. **CORS:** Your CORS is already configured correctly
4. **IAM Permissions:** Your `family-portal-storage` user should have S3 read/write permissions

📋 WHAT YOU NEED TO PROVIDE REPLIT

1. **AWS_ACCESS_KEY_ID** from your IAM user
2. **AWS_SECRET_ACCESS_KEY** from your IAM user
3. **Lambda endpoint URL** (when ready)
4. **Any Lambda authentication** (API keys, etc.)

Once you get the credentials from IAM, you'll be ready to integrate real S3 upload!