Smart HR Management System - Complete Troubleshooting Guide

Problems Encountered During Development

Based on the conversation history, here's a comprehensive troubleshooting guide covering all the issues faced during the Smart HR Management System development:

1. AWS Cognito Authentication Issues

Primary Problem: Authentication Flow Failures

Error Messages Encountered:

- "Please sign in to continue to your HR dashboard"
- "⚠️Not found"
- "🔐 Sign in with AWS Cognito"

Root Cause Analysis

- Missing authentication implementation in Node.js backend
- Cognito App Client configuration issues
- Incorrect callback URL configuration
- Missing OAuth scopes or flow configurations

Detailed Solutions

Backend Authentication Implementation

javascript

```
const jwt = require('jsonwebtoken');
const jwksClient = require('jwks-rsa');

// JWKS client for token verification
const client = jwksClient({
  jwksUri:
`https://cognito-idp.${process.env.COGNITO_REGION}.amazonaws.com/${process.env.COGNI
TO_USER_POOL_ID}/.well-known/jwks.json`
});

// Authentication middleware
const authenticateToken = async (req, res, next) => {
  const authHeader = req.headers['authorization'];
  const token = authHeader && authHeader.split(' ')[1];

  if (!token) {
    return res.status(401).json({ error: 'Access token required' });
  }

  try {
    const decoded = jwt.verify(token, getKey, {
      algorithms: ['RS256']
```

```
    });
    req.user = decoded;
    next();
  } catch (error) {
    return res.status(403).json({ error: 'Invalid token' });
  }
};
```

Missing Environment Variables

Add these to your .env file:
bash

```
COGNITO_USER_POOL_ID=us-east-1_YourPoolId
COGNITO_ISSUER=https://cognito-idp.us-east-1.amazonaws.com/us-east-1_YourPoolId
```

Cognito App Client Configuration Checklist

- ✅ Identity providers enabled: Ensure "Cognito User Pool" is checked
- ✅ Callback URLs: Verify http://localhost:3000/callback is listed
- ✅ Sign out URLs: Confirm http://localhost:3000/ is configured
- ✅ OAuth 2.0 flows: Enable "Authorization code grant" and "Implicit grant"
- ✅ OAuth scopes: Select "openid", "email", "profile"

Authentication Route Implementation

javascript

```
// Login route
app.get('/auth/login', (req, res) => {
  const cognitoLoginUrl = `https://${process.env.COGNITO_DOMAIN}/oauth2/authorize?` +
    `response_type=code&` +
    `client_id=${process.env.COGNITO_CLIENT_ID}&` +
    `redirect_uri=${encodeURIComponent(process.env.COGNITO_REDIRECT_URI)}&` +
    `scope=openid email profile`;

  res.redirect(cognitoLoginUrl);
});
```

2. Security Vulnerabilities

Critical Security Issues Identified

Exposed AWS Credentials

Problem: AWS credentials were exposed in .env file

bash

```
# EXPOSED - SECURITY RISK
AWS_ACCESS_KEY_ID=AKIA6ODU4OEZTXM2UOFH
AWS_SECRET_ACCESS_KEY=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
```

Immediate Actions Required:

1. Rotate these credentials immediately
2. Use IAM roles instead of hardcoded keys for production
3. Never commit .env files to version control
4. Implement proper secret management using AWS Secrets Manager

Security Best Practices Implementation

- Enable MFA for Cognito user pool
- Implement rate limiting on authentication endpoints
- Use HTTPS in production environments
- Regularly audit user permissions and access logs

3. Document Processing Pipeline Issues

Resume Upload and Processing Problems

File Format Support

Challenge: Supporting multiple resume formats (PDF, DOC, DOCX)
Solution:

- Implement format validation before S3 upload
- Configure Textract for optimal document processing
- Add error handling for unsupported formats

S3 Integration Configuration

javascript

```
// S3 upload configuration
const s3Upload = multer({
  storage: multerS3({
    s3: s3Client,
    bucket: process.env.S3_BUCKET,
    key: function (req, file, cb) {
      cb(null, `resumes/${Date.now()}-${file.originalname}`)
    }
  })
});
```

Textract Processing Optimization

- Implement batch processing for multiple resumes
- Add retry mechanisms for processing failures
- Configure proper IAM permissions for Textract access

4. AI Question Generation Issues

Bedrock Integration Challenges

DeepSeek Model Configuration

Problem: Inconsistent AI question generation quality
Solutions:

- Implement custom prompt engineering for consistent output
- Configure model parameters for optimal performance
- Add validation for generated question relevance

Prompt Engineering Best Practices

javascript

```
const generateL1Questions = (candidateData) => {
  const prompt = `
    Based on the following candidate profile:
    ${candidateData}

    Generate 5 technical questions and 3 behavioral questions.
    Format: Technical questions should test specific skills mentioned.
    Behavioral questions should assess cultural fit and experience.
  `;
  return bedrock.generateContent(prompt);
};
```

Token Optimization Strategies

- Implement token counting to manage costs
- Cache frequently used prompts
- Optimize prompt length for efficiency

5. Database and Storage Issues

DynamoDB Configuration Problems

Table Design Optimization

Challenge: Efficient candidate data storage and retrieval
Solution:
javascript

```
// Optimized DynamoDB table structure
const candidateSchema = {
  TableName: 'SmartHR_Candidates',
```

```
  KeySchema: [
    { AttributeName: 'candidateId', KeyType: 'HASH' }
  ],
  AttributeDefinitions: [
    { AttributeName: 'candidateId', AttributeType: 'S' }
  ],
  GlobalSecondaryIndexes: [
    {
      IndexName: 'SkillsIndex',
      KeySchema: [{ AttributeName: 'primarySkill', KeyType: 'HASH' }]
    }
  ]
};
```

Real-time Update Synchronization

Problem: Dashboard updates not reflecting in real-time
Solution:

- Implement WebSocket connections for live updates
- Configure DynamoDB streams for change notifications
- Add proper error handling for update operations

6. Frontend Integration Challenges

React Component State Management

Dashboard Rendering Issues

Problem: Candidate data not displaying correctly
Solutions:

- Implement proper state management using React hooks
- Add loading states for asynchronous operations
- Configure error boundaries for component failures

File Upload Component Optimization

jsx

```
const ResumeUpload = () => {
  const [uploading, setUploading] = useState(false);
  const [error, setError] = useState(null);

  const handleUpload = async (file) => {
    setUploading(true);
    try {
      await uploadToS3(file);
      // Success handling
    } catch (err) {
```

```
      setError(err.message);
    } finally {
      setUploading(false);
    }
  };
};
```

7. Performance and Scalability Issues

System Performance Optimization

Node.js Backend Performance

Challenges:

- Slow API response times
- Memory leaks during high-volume processing
- Timeout issues with AI processing

Solutions:

- Implement connection pooling for database connections
- Add caching mechanisms for frequently accessed data
- Configure proper memory management and garbage collection

AWS Service Optimization

- Configure auto-scaling for Lambda functions
- Implement proper error handling and retry mechanisms
- Optimize DynamoDB read/write capacity units

8. Cross-Region Deployment Issues

Regional Service Availability

Service Selection Challenges

Problem: Initially considered Mumbai region (ap-south-1) but switched to North Virginia (us-east-1)
Reason: Better service availability and AI/ML capabilities
Solution:

- Standardize on us-east-1 for comprehensive service access
- Plan for data residency requirements if needed
- Consider multi-region deployment for disaster recovery

9. Cost Management Problems

Unexpected Cost Escalation

Service Usage Optimization

Issues Identified:

- High Bedrock token consumption

- Excessive Textract processing costs
- Inefficient S3 storage management

Cost Control Measures:

- Implement usage monitoring and alerts
- Configure S3 lifecycle policies for archive management
- Optimize AI model usage with caching and batching

10. Testing and Validation Issues

System Integration Testing

End-to-End Testing Challenges

Problems:

- Inconsistent test data for AI processing
- Authentication flow testing complexity
- Real-time feature validation difficulties

Testing Strategy:

- Implement comprehensive unit and integration tests
- Create mock data for consistent AI testing
- Configure automated testing pipelines

Prevention Strategies

Proactive Monitoring

- Set up CloudWatch alarms for all critical metrics
- Implement comprehensive logging across all components
- Configure automated backup and recovery procedures

Documentation and Knowledge Management

- Maintain updated API documentation
- Document all configuration changes
- Create operational runbooks for common issues

Security Hardening

- Regular security assessments and penetration testing
- Implement least privilege access principles
- Monitor for security vulnerabilities and patches

Performance Monitoring

- Continuous performance monitoring and optimization
- Regular load testing for capacity planning
- Proactive scaling based on usage patterns

Quick Reference Troubleshooting Commands

System Health Checks

bash

```
# Check Cognito configuration
curl
https://cognito-idp.us-east-1.amazonaws.com/YOUR_POOL_ID/.well-known/openid-configuratio
n

# Test S3 bucket access
aws s3 ls s3://your-bucket-name

# Verify DynamoDB table status
aws dynamodb describe-table --table-name SmartHR_Candidates

# Check Node.js backend health
curl http://localhost:5000/health
```

Log Analysis

bash

```
# Check CloudWatch logs
aws logs describe-log-groups --log-group-name-prefix /aws/lambda/hr-function

# Monitor system metrics
aws cloudwatch get-metric-statistics --namespace AWS/DynamoDB --metric-name
ConsumedReadCapacityUnits
```

This comprehensive troubleshooting guide covers all the major issues encountered during the Smart HR Management System development and provides practical solutions for prevention and resolution of similar problems in the future.