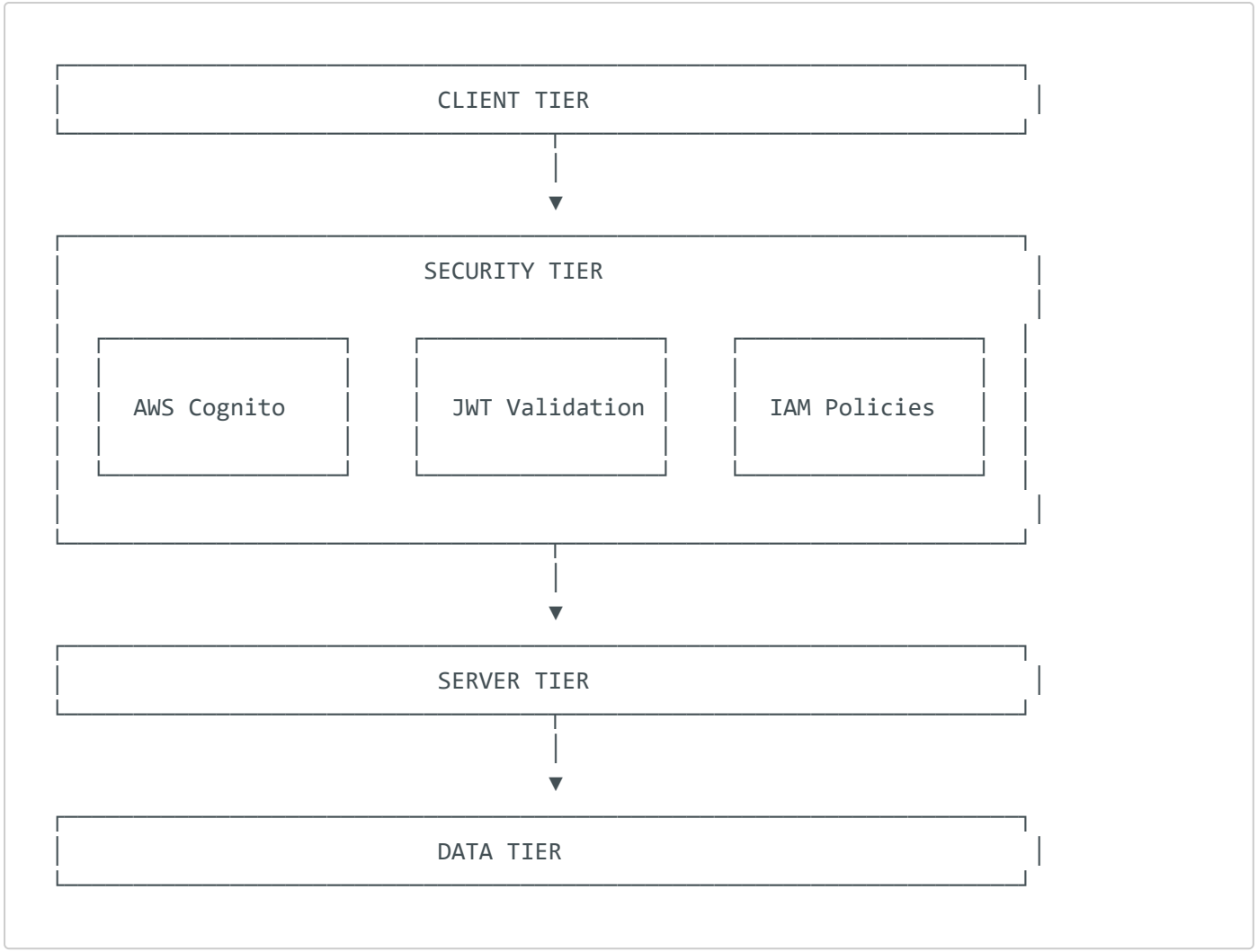
- Security Architecture
  - Security Architecture Overview
  - Authentication System
    - Development Environment
    - Production Environment
    - Authentication Flow
  - Authorization Model
    - JWT-Based API Authorization
    - Fine-Grained Access Control
    - User Permissions Model
  - Data Protection
    - Data-at-Rest Encryption
    - Data-in-Transit Encryption
    - Sensitive Data Handling
  - Network Security
    - API Protection
    - VPC Integration (Optional)
  - Secure Development Practices
    - Application Security
    - Dependency Management
    - Secure Configuration
  - Monitoring and Incident Response
    - Security Monitoring
    - Incident Response Capabilities
  - Compliance Considerations
  - Authentication Implementation Details
    - Frontend Authentication Service
    - API Security Integration
  - Security Best Practices for Developers

Security Architecture

This document outlines the security architecture of the Document Processing Accelerator, detailing the approach to authentication, authorization, data protection, and secure communications.

Security Architecture Overview



Authentication System

The Document Processing Accelerator implements a dual authentication system:

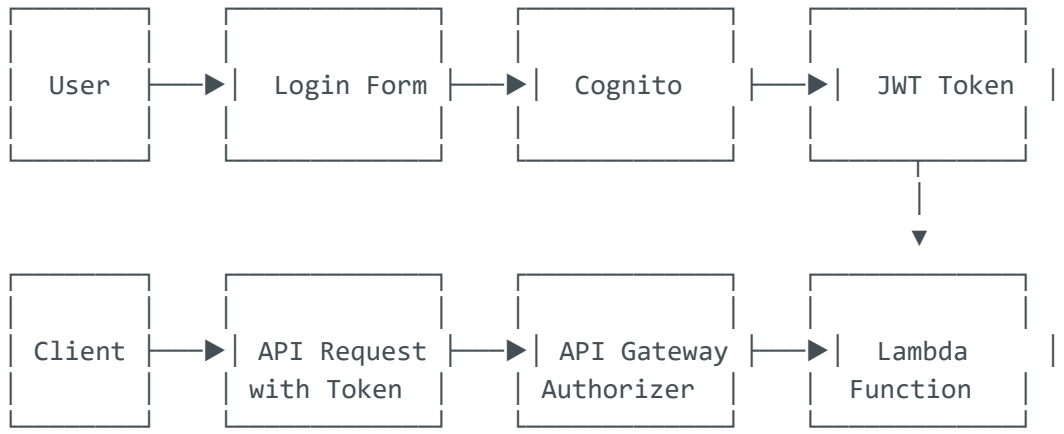
Development Environment

- **Mock Authentication Service** (mockAuthService.ts)
  - Simulates authentication flows without external dependencies
  - Stores authentication state in local storage
  - Implements the same interface as the production authentication service

Production Environment

- **AWS Cognito Authentication** (amplifyAuthService.ts)
  - Uses AWS Amplify v5 API for integration with Cognito
  - Supports user sign-up, sign-in, and password recovery
  - Implements MFA capability for enhanced security
  - Issues JWT tokens for API authorization

Authentication Flow



Authorization Model

JWT-Based API Authorization

- Tokens issued by Cognito are validated at the API Gateway level
- Custom authorizer Lambda function validates token signatures
- Token claims determine user permissions

Fine-Grained Access Control

- IAM roles and policies restrict Lambda function permissions
- DynamoDB conditional expressions enforce user-based data isolation
- S3 bucket policies limit access to user-specific paths

User Permissions Model

Role	Document Access	Administrative Functions
Regular User	Own documents only	None
Admin	All documents	User management
Super Admin	All documents, system config	System configuration

Data Protection

Data-at-Rest Encryption

- **S3 Buckets**: Server-side encryption (SSE-S3) for document storage
- **DynamoDB**: Encryption enabled for all tables
- **Lambda Environment Variables**: Encrypted with KMS
- **Secrets**: Stored in AWS Secrets Manager with automatic rotation

Data-in-Transit Encryption

- **API Gateway**: HTTPS enforcement for all endpoints
- **S3 Pre-signed URLs**: HTTPS with temporary credentials
- **Internal AWS Service Communications**: TLS 1.2+

Sensitive Data Handling

- PII data is encrypted at the application level before storage
- Document content is processed in memory and not stored in plaintext
- Encryption/decryption operations use AWS KMS for key management

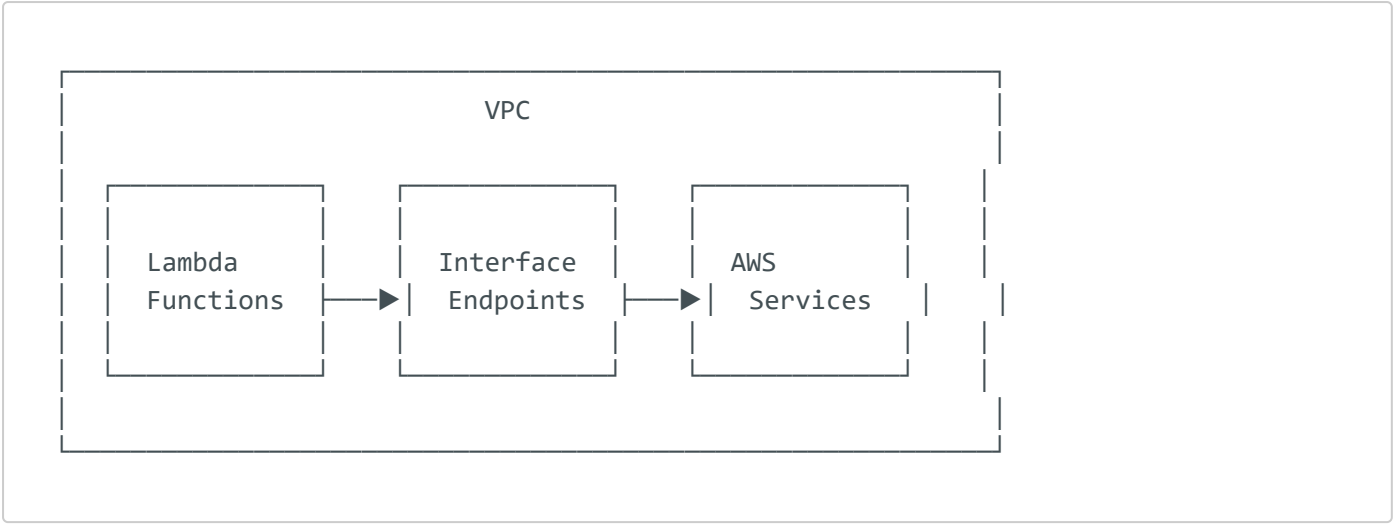
Network Security

API Protection

- API Gateway provides front-end protection from attacks
- Rate limiting prevents abuse and DoS attacks

- WAF integration blocks common attack patterns

VPC Integration (Optional)



- Lambda functions can be configured to run within a VPC
- Private network paths to AWS services via VPC endpoints
- Security groups restrict traffic flow

Secure Development Practices

Application Security

- Input validation on both client and server
- Output encoding to prevent XSS attacks
- Proper error handling that doesn't leak sensitive information
- Content Security Policy implementation

Dependency Management

- Regular dependency scanning for vulnerabilities
- Automated updates for critical security patches
- Dependency pinning to prevent supply chain attacks

Secure Configuration

- Environment-specific configuration with proper separation
- No hardcoded secrets or credentials
- Infrastructure as Code with version control

Monitoring and Incident Response

Security Monitoring

- CloudWatch Logs for Lambda function monitoring
- CloudTrail for AWS API activity logging
- Custom alerting for suspicious activities

Incident Response Capabilities

- Automated alerts for security anomalies
- Predefined response playbooks for common scenarios
- Regular security incident drills

Compliance Considerations

The Document Processing Accelerator is designed with the following compliance frameworks in mind:

- GDPR: Data privacy and protection measures
- SOC 2: Controls for security, availability, and confidentiality
- HIPAA: If configured for healthcare document processing

Authentication Implementation Details

Frontend Authentication Service

The authentication service is designed with a provider pattern that allows seamless switching between development and production implementations:

```
// authServiceProvider.ts
import { AuthService } from '../types/auth';
import { mockAuthService } from './mockAuthService';
import { amplifyAuthService } from './amplifyAuthService';

// Determine which auth implementation to use
const useRealAuth = process.env.REACT_APP_USE_REAL_AUTH === 'true';

// Export the appropriate auth service implementation
export const authService: AuthService = useRealAuth
  ? amplifyAuthService
  : mockAuthService;
```

API Security Integration

The API client automatically includes authentication tokens in requests:

```
// apiClient.ts
import { authService } from './authServiceProvider';

export const apiClient = {
  async get(endpoint: string) {
    const token = await authService.getToken();

    return fetch(`${API_BASE_URL}${endpoint}`, {
      headers: {
        'Authorization': `Bearer ${token}`,
        'Content-Type': 'application/json'
      }
    }).then(handleResponse);
  },

  // Additional methods (post, put, delete) follow the same pattern
};
```

Security Best Practices for Developers

When working on the Document Processing Accelerator, developers should:

1. **Never disable authentication** in production environments
2. **Always validate input** from users and external systems
3. **Use parameterized queries** when accessing DynamoDB
4. **Follow the principle of least privilege** for IAM roles

5. **Keep dependencies updated** to mitigate security vulnerabilities
6. **Enable MFA** for their AWS console access
7. **Use pre-signed URLs** for all S3 operations exposed to end users