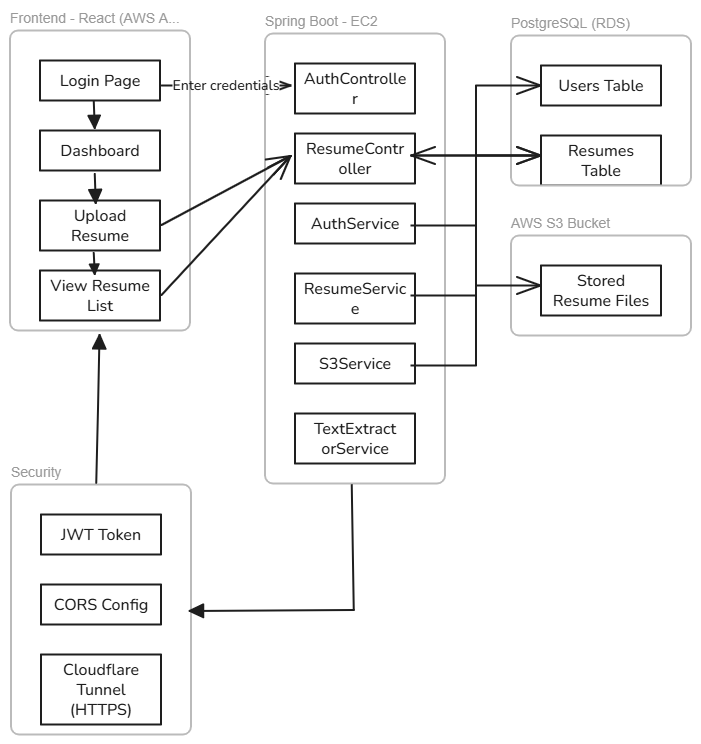
**AWS Solution Architecture**

# 1. Overview

This document describes the architecture, components, and design rationale of the CV Processing Portal.   
The portal enables users to securely upload, process, and view CVs through a cloud-native full-stack application built using   
React (frontend), Spring Boot (backend), PostgreSQL (database), and AWS services for compute, storage, and deployment.

# 2. High-Level Architecture Diagram

Below is the high-level diagram of the complete system architecture: 

# 3. Architecture Components

## 3.1 Frontend Layer (React)

Framework: React.js  
Hosted On: AWS Amplify Hosting  
Responsibilities:  
- User login  
- JWT token storage and authorization  
- Resume upload form  
- Display processed CV data  
- Resume download via pre-signed URL

## 3.2 Backend Layer (Spring Boot)

Framework: Spring Boot (Java 17)  
Deployment: AWS EC2   
Responsibilities:  
- User authentication & authorization (JWT)  
- File upload to S3  
- Generate Pre-Signed URLs  
- CV text extraction  
- Interaction with PostgreSQL database

## 3.3 CV Processing Layer

Storage: AWS S3 (Resume files)  
Processing: Backend Java service uses Apache Tika to extract text  
Purpose:  
- Centralized, scalable storage  
- File type validation & content extraction

## 3.4 Database Layer

Database: PostgreSQL  
Deployment Options: Amazon RDS (preferred), EC2 fallback  
Tables:  
- users: Stores user credentials, roles  
- resumes: Stores resume metadata (name, upload date, s3\_url, extracted\_text)

## 3.5 Security Layer

Feature Implementation  
Authentication JWT (JSON Web Token) using Spring Security  
Authorization Role-based (e.g., ROLE\_USER, ROLE\_ADMIN)  
HTTPS Enabled via Cloudflare tunnel to EC2 or Amplify  
CORS Configured in backend to allow frontend origin  
Token Storage Stored in browser's local storage

# 4. AWS Services Used

Service Purpose  
EC2 Hosting Spring Boot backend  
Amplify Hosting React frontend  
S3 Resume file storage, pre-signed URL generation  
RDS PostgreSQL database  
Cloudflare Tunnel Enables HTTPS without domain name

# 5. Security and Access Flow

User logs in via React frontend → sends credentials to /api/auth/login  
Backend validates via Spring Security and returns JWT  
JWT is stored on the frontend and attached to all protected API requests  
CV files are uploaded via secure multipart/form-data to /api/resumes  
Backend uploads the file to S3 and stores metadata  
When the user clicks “View Resume,” backend generates a time-limited pre-signed S3 URL  
Frontend fetches and displays/downloads the resume via the signed URL

# 6. Deployment Strategy

Component Deployment Method  
React Frontend AWS Amplify → Connected to GitHub repo  
Spring Boot EC2 instance (Amazon Linux 2), exposed via port 8080 with Cloudflare HTTPS tunneling  
Database Amazon RDS PostgreSQL  
S3 Buckets Public access disabled, access via signed URL only  
HTTPS Cloudflare Tunnel (no domain required)

# 8. Additional Services/Configurations Used

Spring Cloud AWS SDK (v2.x): For S3 client  
Apache Tika or PDFBox: For resume text extraction  
Cloudflare : For HTTPS tunneling  
Postman: API Testing  
tmux: For background tunnel session on EC2  
JWT Debugger: To test and validate tokens  
GitHub Actions (optional): For CI/CD with Amplify