This document provides specific, step-by-step instructions for Yanga Mgudwa to implement his serverless resume parsing project. It assumes he has completed the design and coding phases and is ready to deploy the application to AWS.

Phase 1: Setting up AWS Resources

1. **Create IAM Roles:**
    - Create an IAM role for your Lambda function. Attach the AWSLambdaBasicExecutionRole, AmazonS3FullAccess (restrict to your specific buckets later!), AmazonTextractFullAccess (restrict later!), and AmazonDynamoDBFullAccess (restrict later!) managed policies, or create custom policies with least privilege. Note the role's ARN.
    - Create an IAM role (or use the same role as the Lambda function, but best practice is to create a separate role for security) for API Gateway with permission to invoke the Lambda function.
2.
3. **Create S3 Buckets:**
    - Create an S3 bucket for your website's static assets (e.g., yanga-portfolio-website). Enable static website hosting, and make it publicly accessible (with caution; understand the security implications).
    - Create a separate S3 bucket for temporarily storing uploaded resumes (e.g., resume-uploads). **Do not** make it publicly accessible. Create a bucket policy allowing only your API Gateway's execution role to upload files. Enable server-side encryption (SSE-S3 or SSE-KMS).
4.
5. **Create DynamoDB Table:**
    - Create a DynamoDB table named ParsedResumes.
    - Use ResumeID (String) as the primary key.
    - Define attributes for ApplicantName, ContactInfo, Skills (list), Experience (list of JSON objects), Education (list of JSON objects), Projects (list of JSON objects), and Keywords (list), with appropriate data types.
6.
7. **Create API Gateway Endpoint:**
    - Create a new REST API in API Gateway (e.g., ResumeParserAPI).
    - Create a resource (e.g., /resume).
    - Create a POST method for this resource, configured to integrate with your Lambda function. Select the correct IAM role and ensure multipart/form-data is accepted.
    - Deploy the API and note the Invoke URL.
8.

Phase 2: Deploying and Configuring Application Components

1. **Deploy Lambda Function:**

    - Upload your completed Lambda function code to AWS Lambda.
    - Configure environment variables (e.g., DYNAMODB_TABLE_NAME).
    - Test the function thoroughly.
2.
3. **Configure S3 Event Notification:**
    - Go to your resume-uploads S3 bucket.
    - Under "Properties" -> "Events," add an event notification.
    - Select "Object created (All)" and choose your Lambda function as the destination.
4.
5. **Set up QuickSight Data Source:**
    - Go to QuickSight, create a data source connecting to your DynamoDB table.
    - Configure the necessary IAM role for QuickSight to access your DynamoDB table.
    - Carefully map the fields and data types.
6.
7. **Create QuickSight Dashboard:**
    - Build your visualizations based on your parsed resume data.
    - Use suitable chart types (bar charts, word clouds, pie charts, etc.).
    - Ensure data accuracy and visual appeal.
8.
9. **Integrate Frontend with API Gateway:**
    - Update your frontend's resume upload form's action attribute to point to the API Gateway Invoke URL.
    - Implement appropriate feedback mechanisms (success/error messages).
10.

Phase 3: CI/CD Implementation (Optional)

1. **Create CodePipeline Pipeline:**
    - Create a CodePipeline pipeline, connecting it to your GitHub repository, CodeBuild project, and your deployment targets (S3 and Lambda).
    - Configure the buildspec.yml file for CodeBuild.
2.
3. **Configure CodeBuild Project:** Make sure it will successfully build both frontend and backend.
4. **Test CI/CD Pipeline:** Trigger the pipeline to verify functionality.

Phase 4: Testing, Documentation, and Deployment

1. **Thorough Testing:** Conduct comprehensive end-to-end testing.
2. **Documentation:** Write a detailed README.md file for your GitHub repository, clearly explaining your project's architecture, functionality, and deployment steps.
3. **Deployment:** Deploy the code to your GitHub repository and update your LinkedIn profile with details about your project.

Important Considerations:

- **Security:** Always prioritize security. Implement least-privilege IAM roles, use server-side encryption, and carefully configure bucket policies.
- **Error Handling:** Implement robust error handling in your Lambda function and frontend.
- **Testing:** Test thoroughly at each step.
- **Scalability:** Consider scaling options for DynamoDB if you anticipate large-scale usage.
- **Cost Optimization:** Monitor your AWS costs regularly using Cost Explorer.

These instructions provide a detailed plan. Remember to consult the official AWS documentation for each service for detailed information and configurations. Testing and iterative development are crucial for success. Prioritize security best practices throughout the process.