Forward Inheritance SaaS Architecture with React/React Router, Node.js/Express, PostgreSQL, Lambda, and PII Protection

1. Overview

In this architecture, the SaaS web application is delivered through a seamless and nationwide optimized (with global delivery as an option) pipeline. The frontend is served through Amazon CloudFront, ensuring low-latency content delivery worldwide. CloudFront pulls content from AWS Amplify, where the React application is hosted and continuously deployed from version control. When users interact with the frontend and initiate API requests, the requests are securely routed through AWS API Gateway. API Gateway enforces security, rate limiting, and routes the requests to the backend Node.js application, which runs in Docker containers on AWS Fargate or Amazon EKS. The Node.js/Express backend performs business logic and interacts with a PostgreSQL database hosted on Amazon RDS. This layered design provides scalability, modularity, and performance for delivering SaaS applications with sensitive data handling requirements.

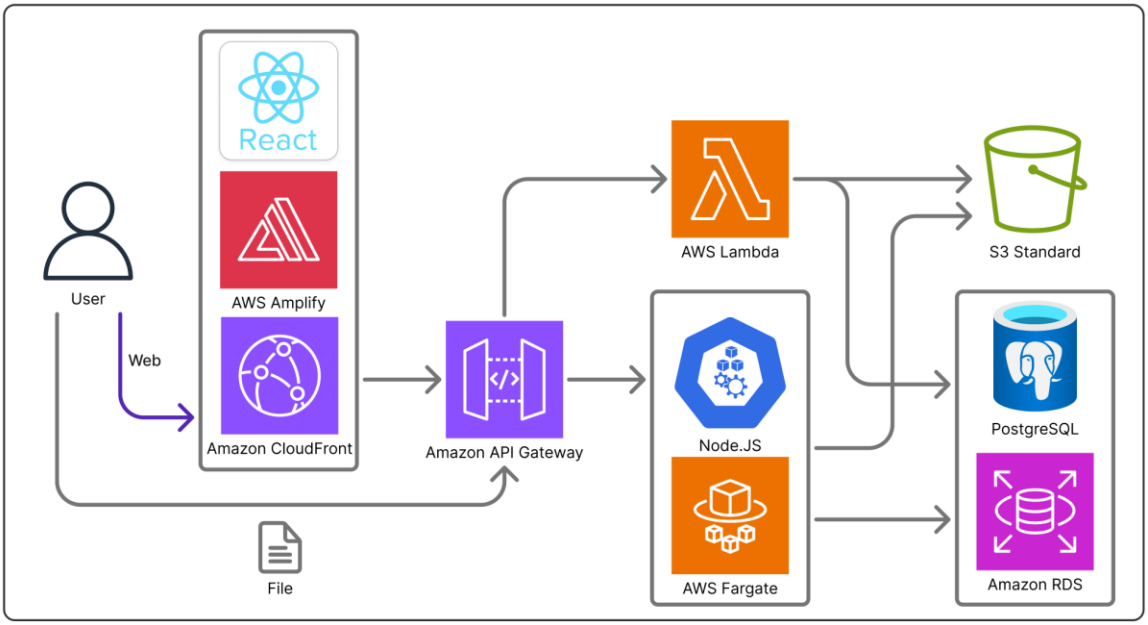
This document describes the architecture for Forward Inheritance, referred to as 'Forward', which includes a scalable, secure architecture built on AWS to support a modern full-stack application. It includes a React frontend, Node.js backend, PostgreSQL database, serverless Lambda functions for business logic, and dedicated tokenization and encryption for personally identifiable information (PII).

2. Component Breakdown

- Frontend: Hosted on AWS Amplify with CI/CD and global delivery via CloudFront.
- API Gateway: Manages API access, routes requests, applies throttling and security.
- Node.js Backend: Dockerized and deployed to AWS Fargate or Amazon EKS.
- Lambda Functions: Used for file import/export, reporting, and scheduled business logic.
- Database: Amazon RDS PostgreSQL with upgrade path to Aurora PostgreSQL.
- File Storage: Amazon S3 for storing static files and exports.
- PII Tokenization & Encryption: Sensitive documents are processed through a secure Lambda or container-based microservice, tokenized, encrypted, and stored securely in S3 with metadata in RDS.

Encryption across the stack is handled using AWS-native services:
- **AWS Key Management Service (KMS)**: For managing encryption keys used across Lambda, RDS, and S3.
- **RDS Encryption at Rest**: Enables storage-level encryption with KMS integration.
- **S3 Server-Side Encryption (SSE-KMS)**: Ensures secure storage of file objects.
- **Envelope Encryption**: Can be used in custom Lambda logic for per-record encryption.
All key usage is logged via CloudTrail for auditing and compliance.

3. Architecture Diagram



4. Example Use Cases

a) User Login: User navigates to the SaaS website, served through CloudFront and Amplify. The login form authenticates through AWS Cognito. Once verified, the user receives a secure JWT token. Subsequent API requests include the token and are validated by API Gateway before reaching the Node.js/Express backend. Session metadata is securely stored and updated in PostgreSQL.

b) File Import: User uploads → API Gateway → Lambda → RDS
c) File Export: API Gateway → Lambda → RDS → Generate file → S3
d) Scheduled Logic: Lambda invoked via EventBridge for periodic jobs
e) PII Tokenization: Uploaded docs pass through tokenization service before storage

5. Security and Cyber Attack Protection

This architecture emphasizes security at multiple layers:
- CloudFront: HTTPS, WAF integration, throttling
- API Gateway: IAM and token-based authentication, request validation
- Lambda & Backend: Run in secure VPCs with minimal IAM roles
- PII Tokenization Service: Encrypts sensitive content, replaces identifiers with tokens
- S3: Encrypted storage with pre-signed URLs
- RDS: Encrypted at rest and in transit, limited access

These features help protect against common threats such as:
- Injection attacks
- Unauthorized access
- Data exfiltration
- DDoS attacks
- Insider threats

This makes the system ideal for handling and protecting forward-secure personal and financial information.

6. Architecture Extension Possibilities

6.1 AI Integration with Amazon Bedrock

To enhance user interaction and business intelligence, Amazon Bedrock can be integrated for AI services such as generative chat, document summarization, and personalized recommendations. These models can be invoked via secure APIs or embedded within Lambda functions. Data used for AI interactions does not leave the secure environment of the Forward platform, and Amazon Bedrock does not use customer data for training its models. There is no external sharing or exposure of customer information—security and privacy are enforced at every level of AI interaction.

6.2 Telephony, IVR, and Call Center with AWS Connect

AWS Connect can be used to deploy a cloud-native contact center integrated with the existing architecture. It supports IVR flows, intelligent routing, and integration with AWS Lambda and Amazon Lex for AI-driven interactions. Customer data can be fetched from RDS or S3 using secure APIs. Real-time sentiment analysis and support transcripts can be stored and analyzed via Bedrock or Comprehend. This extension brings voice support to the SaaS app, allowing for a fully integrated omnichannel support system.

Addendum: CI/CD Integration and Deployment

A comprehensive CI/CD pipeline should be implemented for each tier of the system to ensure efficient, reliable deployment.

- Frontend (Amplify): GitHub/GitLab integration enables auto-build and deploy of the React app to Amplify, with preview branches and rollback support.
- Backend (Node.js): Docker containers are built via CodePipeline or GitHub Actions, pushed to Amazon ECR, and deployed to Fargate/EKS using ECS or Kubernetes manifests.
- Lambda Functions: Packaged and deployed through AWS SAM or the Serverless Framework with Git-based triggers.
- Infrastructure: Terraform or AWS CDK can be used to manage infrastructure as code across environments.
- Database: Migration tools such as Flyway or Prisma Migrate should be integrated into backend deployment pipelines.

Each pipeline includes linting, unit tests, security scans, and staged deployments to support continuous delivery while maintaining high reliability and security.

6.3 Multi-Site Protection and No Downtime Deployments

To support business continuity and application resilience, this architecture can be extended to support multi-site deployment strategies across AWS regions or Availability Zones. This provides protection against localized outages and enables near-zero downtime deployments.

- **Multi-AZ/Region Deployment**: Backend services and databases can be deployed across multiple AWS Availability Zones (AZs). PostgreSQL (Aurora) supports multi-AZ failover and read replicas across regions.
- **Route 53 + CloudFront**: Can be used for DNS-based traffic routing and failover.
- **Blue/Green Deployments**: Supported in ECS and Lambda through CodeDeploy, allowing staged deployments with automatic rollback on failure.
- **Global Datastore for Amazon ElastiCache or DynamoDB**: For real-time sync and low-latency access in distributed environments.
- **S3 Cross-Region Replication**: Ensures file assets are available across geographies.
This extension ensures service continuity even during major AWS outages or regional disruptions, and allows application updates without interrupting users.