# SYSTEM REQUIREMENTS DOCUMENT

## PROJECT DETAILS

PROJECT NAME: YouBank
CREATOR: SZ TECH LLC

## 1. EXECUTIVE SUMMARY SNAPSHOT

The Bank Sharing Application is an innovative financial technology solution designed to empower users with easy and secure access to banking services, enabling them to manage their accounts, conduct transactions, and stay informed about their financial activities. This application serves as a digital bridge between users, their accounts, and the Federal Reserve Bank (FRB) network. This document outlines the key technical requirements for the development and implementation of this application.

**Objective**:
The primary goal of the Bank Sharing Application is to provide a comprehensive and user-friendly digital platform for users to interact with their bank accounts, facilitate transactions, and access essential information about their financial institution. This application also includes an interface for bank administrators to manage user accounts, transactions, and FRB information.

**Key Features:**
**1. User Management:**
User registration with unique usernames and email addresses.
User authentication and login capabilities.
User profile management for updating personal information.

**2. Account Management:**
Creation and management of various types of bank accounts (e.g., savings, checking).
Real-time access to account balances, transaction history, and account statements.
Secure transaction processing, including deposits, withdrawals, and transfers.

**3. Transaction Processing:**
Handling deposits, withdrawals, and inter-account transfers.
Ensuring transaction security, integrity, and accuracy.
Transaction history and transaction receipts for users.

**4. Notifications:**
Sending notifications to users for important account events.
Supporting various notification channels (e.g., email, SMS, in-app notifications).

**5. Admin Panel:**
Admin interface for bank employees to manage user accounts and transactions.
Comprehensive audit trails for admin actions and financial oversight.

**6. FRB Integration:**
Storage and retrieval of FRB information, including routing numbers and contact details.

Admin capabilities to add, modify, and delete FRB data.

**7. Security and Compliance:**
Implementation of robust security measures, including data encryption and multi-factor authentication.
Compliance with financial regulations such as Know Your Customer (KYC) and Anti-Money Laundering (AML) requirements.
Protection of user data and privacy in accordance with data protection laws.

## 5. TECHNICAL REQUIREMENTS
These technical requirements form the foundation for building a secure, scalable, and compliant Bank Sharing

### Application.
**Programming Language**: Java
Why? Java is a mature and widely adopted language with strong support for building secure and scalable applications.
It offers excellent performance and a vast ecosystem of libraries and tools.

**Web Framework**: Spring Boot (Java-based)
Why? Spring Boot is well-suited for building robust and scalable web applications. It provides a comprehensive framework for creating RESTful APIs and offers features like dependency injection, security, and integration with databases.

**Database**: PostgreSQL (Relational Database Management System)
Why? PostgreSQL is a robust, open-source RDBMS known for data security and ACID compliance. It supports complex data models and is suitable for financial applications with relational data requirements.

**RESTful API**:
The application should expose a RESTful API to facilitate communication between the frontend and backend. API endpoints should be designed according to industry best practices and follow a clear, consistent structure.

**Security Measures**:
**1. Authentication and Authorization:**
Implement user authentication using JWT (JSON Web Tokens) to securely manage user sessions.
Define user roles (customer, employee, admin) and assign appropriate permissions.
**2. Data Encryption:**
Use HTTPS to encrypt data in transit and protect sensitive information during communication between clients and the server.
Implement encryption for sensitive data at rest in the database.
**3. Rate Limiting:**
Apply rate limiting to API endpoints to prevent abuse or misuse.
**4. Multi-Factor Authentication (MFA):**
Offer MFA options to enhance user account security.

**5. OWASP Top Ten:**
Conduct regular security assessments to address vulnerabilities listed in the OWASP Top Ten.
**6. Security Compliance:**
Ensure compliance with financial regulations, such as Know Your Customer (KYC) and Anti-Money Laundering (AML) requirements.
**7. Logging and Monitoring:**
Implement robust logging to track user activities and system behavior.
Set up monitoring tools to detect and respond to security incidents.

**Scalability:**
Design the application to be horizontally scalable, allowing it to handle increased user loads and demand. Consider containerization (e.g., Docker) and orchestration (e.g., Kubernetes) for managing scalability.

**Cloud Services:**
AWS or AZURE

**Data Protection and Privacy:**
Adhere to data protection laws and regulations, such as GDPR, to safeguard user data and privacy.

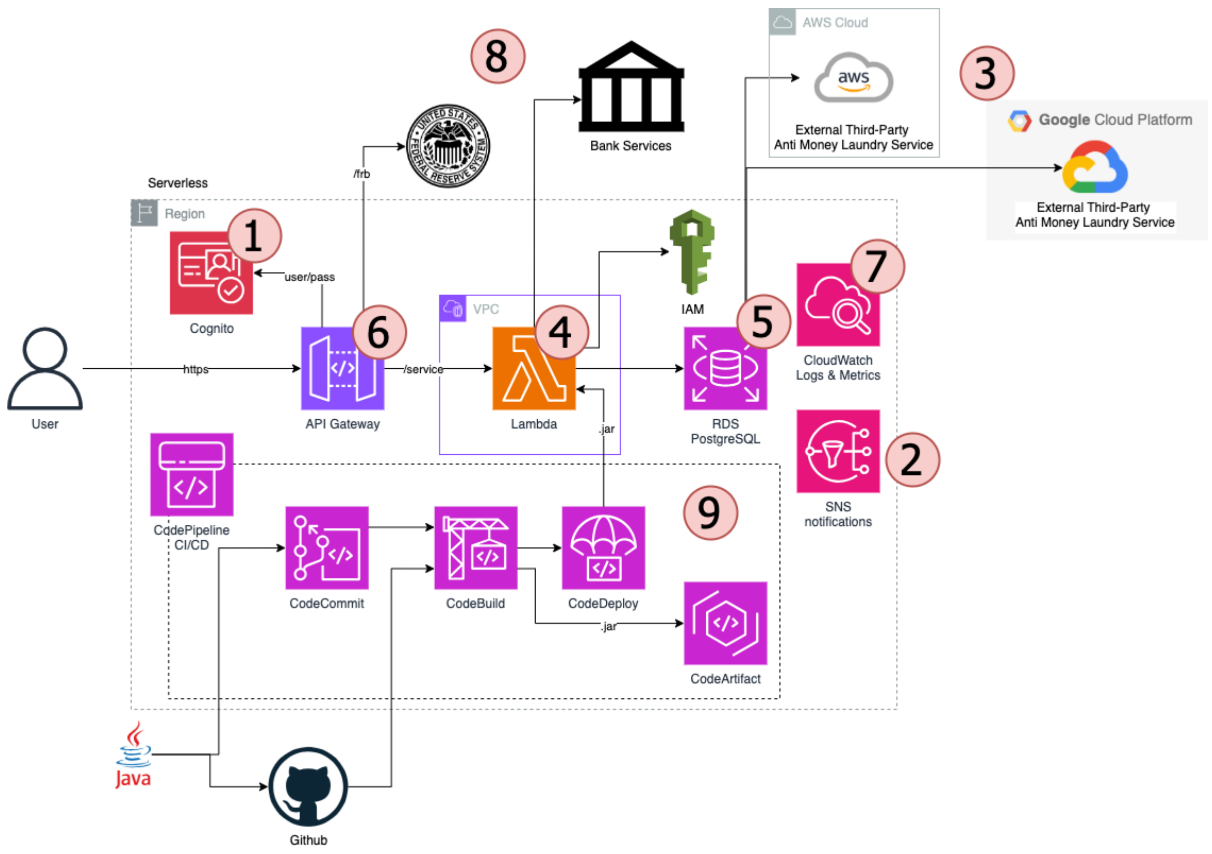
**APIs:**
API documentation for OpenAPI to is above.

## 6. Cloud architecture proposal

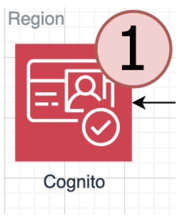
According to the elements described in the system requirements document, a set of cloud services were proposed to meet the needs of the application at POC level, with the possibility of scaling to a production system reliable, scalable and secure.

The proposed architecture is serverless AWS based, so the services used are consumed as SaaS, with no need of wasting time maintaining the infrastructure for updates, scalability, provisioning, etc. It is not Docker, it is based on cloud functions Lambda. Most services used have a free tier, which can be used for POC purposes.

The core functionality of the Java Spring Boot application is to be implemented in a combination of API Gateway and Lambda.



**Key Features:**
**1. User Management:**
**Amazon Cognito:** number 1 in the diagram.



By leveraging Amazon Cognito, you can incorporate user registration and login functionalities while managing access to your web and mobile applications. Amazon

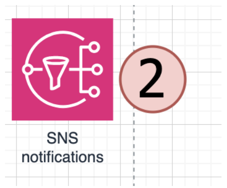
Cognito offers a scalable identity store catering to millions of users, supports social and enterprise identity federation, and provides advanced security features to safeguard your consumers and business. Built upon open identity standards, Amazon Cognito adheres to various compliance regulations and seamlessly integrates with both frontend and backend development resources.

For the spring boot application, there are implementations using Spring Security's OAuth 2.0 support to authenticate with Amazon Cognito, well described in internet articles It is strongly recommended to research about it to design the app accordingly.

This would require changes in the app dependencies. For example, it would be required to add the **spring-security-oauth2-client** and **spring-security-oauth2-jose** dependencies to the application and to configure **Cognito Authentication Provider**, as well as its dependencies.

From the cloud point of view, Amazon Cognito is an identity platform for web and mobile apps. It's a user directory, an authentication server, and an authorization service for OAuth 2.0 access tokens and AWS credentials. With Amazon Cognito, you can authenticate and authorize users from the built-in user directory, from your enterprise directory, and from consumer identity providers like Google and Facebook. Fully managed by AWS, is easy to provision and maintain.
**Pricing**: You pay for Amazon Cognito user pools based on your monthly active users (MAUs), Amazon Cognito user pools has a **free tier** of 50,000 MAUs per account for users who sign in directly to Amazon Cognito user pools and 50 MAUs for users federated through SAML 2.0 based identity providers.

**4. Notifications:**
**Amazon Simple Notification Service (SNS)**: number 2 in the diagram.

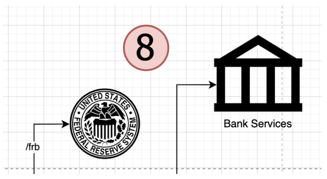


Is a managed service that provides message delivery from publishers to subscribers (also known as producers and consumers). Publishers communicate asynchronously with subscribers by sending messages to a topic, which is a logical access point and communication channel. Clients can subscribe to the SNS topic and receive published messages using a supported endpoint type, such as Amazon Kinesis Data Firehose, Amazon SQS, AWS Lambda, HTTP, email, mobile push notifications, and mobile text messages (SMS).

From the app standpoint, it should be implemented as a Spring Boot publisher for AWS SNS topic, which is widely documented in internet.

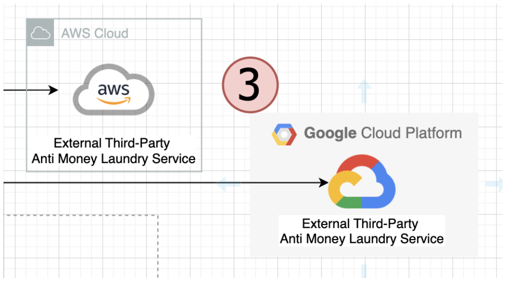
Pricing: Amazon Simple Notification Service (Amazon SNS) Standard topic pricing is based on the number of monthly API requests made, and the number of deliveries to various endpoints (the cost of the delivery depends on the endpoint type). The **free tier** includes the first 1 million Amazon SNS requests per month are free, $0.50 per 1 million requests thereafter

**6. FRB Integration:**
**External consumption of services**: number 8 in the diagram.



Able to consume external services like FRB or other banking exposing their API's

**7. Security and Compliance:**
**External provider:** number 3 in the diagram.


AWS Cloud
aws
External Third-Party Anti Money Laundry Service
3
Google Cloud Platform
External Third-Party Anti Money Laundry Service

For this, it is strongly recommended to include an external provider for verifying this type of compliance. AWS partners are highly recommended, since they provide solutions as service with easy integration with existing aws infrastructure. There are other solutions available in different forms, for example GCP offers an AML service. Providers from AWS: https://aws.amazon.com/solutions/financial-services/aml-know-your-customer/?marketplace-ppa-and-quickstart.sort-by=item.additionalFields.sortDate&marketplace-ppa-and-quickstart.sort-order=desc

**5. TECHNICAL REQUIREMENTS**

The following is a list of the requirements provided and which AWS service meet the requirements for the implementation.

**Application**.

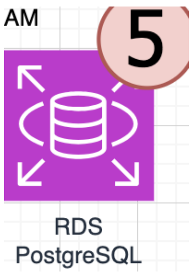
**Programming Language**: Java
**Web Framework**: Spring Boot (Java-based)
**Amazon Lambda**: number 4 in the diagram.

To meet this requirement the service chosen is AWS Lambda, which is a compute service that lets you run code without provisioning or managing servers.
It is a common implementation to migrate a local spring-boot app to AWS Lambda, well documented and with strong benefits. So even if the app was not initially designed for this serverles model, it can be easily adapted.

Lambda runs your code on a high-availability compute infrastructure and performs all of the administration of the compute resources, including server and operating system maintenance, capacity provisioning and automatic scaling, and logging. With Lambda, all you need to do is supply your code in one of the language runtimes that Lambda supports. You organize your code into Lambda functions. The Lambda service runs your function only when needed and scales automatically. You only pay for the compute time that you consume—there is no charge when your code is not running.

**Pricing**: The AWS Lambda **free tier** includes one million free requests per month and 400,000 GB-seconds of compute time per month, usable for functions powered by both x86, and Graviton2 processors, in aggregate. Additionally, the free tier includes 100GiB of HTTP response streaming per month, beyond the first 6MB per request, which are free.

**Database**: PostgreSQL (Relational Database Management System)
**Amazon Lambda**: number 5 in the diagram.

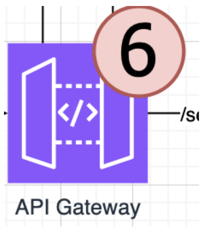


Amazon Relational Database Service (Amazon RDS) is a web service that makes it easier to set up, operate, and scale a relational database in the AWS Cloud. It provides cost-efficient, resizable capacity for an industry-standard relational database and manages common database administration tasks. Amazon RDS supports DB instances running several versions of PostgreSQL.

You can create DB instances and DB snapshots, point-in-time restores and backups. DB instances running PostgreSQL support Multi-AZ deployments, read replicas, Provisioned IOPS, and can be created inside a virtual private cloud (VPC). You can also use Secure Socket Layer (SSL) to connect to a DB instance running PostgreSQL.

**Pricing**: New AWS customers can get started with Amazon RDS for PostgreSQL for free as part of the AWS **Free Tier**. The Amazon RDS for PostgreSQL Free Tier includes 750 hours on select Single-AZ instance databases, 20 GB of General Purpose SSD (gp2) storage, and 20 GB of storage for automated database backups each month for one year.

**RESTful API**:
**Amazon API Gateway**: number 6 in the diagram.



Amazon API Gateway is an AWS service for creating, publishing, maintaining, monitoring, and securing REST, HTTP, and WebSocket APIs at any scale. API developers can create APIs that access AWS or other web services, as well as data stored in the AWS Cloud. As an API Gateway API developer, you can create APIs for use in your own client applications. Or you can make your APIs available to third-party app developers.
API Gateway creates RESTful APIs that: Are HTTP-based. Enable stateless client-server communication.Implement standard HTTP methods such as GET, POST, PUT, PATCH, and DELETE.
The integration with Lambda spring boot java, is the core of this architecture.

**Pricing**: The Amazon API Gateway **free tier** includes one million API calls received for REST APIs, one million API calls received for HTTP APIs, and one million messages and 750,000 connection minutes for WebSocket APIs per month for up to 12 months. If you exceed this number of calls per month, you will be charged the API Gateway usage rates.

**Security Measures**:
**1. Authentication and Authorization:**
AWS implements Role based access for internal consuming of services, as well as https for incoming communications for consuming the API. With Cognito, the authentications of users have an extra layer of management.
**2. Data Encryption:**
HTTPS is used for all incoming requests, the data in transit within aws never leaves the aws network, it travels encrypted and it is encrypted at rest. In RDS this can be configured to have encrypted databases, snapshots and data in transit.

**3. Rate Limiting:**
API Gateway service includes management for limiting the connections.

**4. Multi-Factor Authentication (MFA):**

Implemented with Cognito

**5. OWASP Top Ten:**
Conduct regular security assessments to address vulnerabilities listed in the OWASP Top Ten.
This is an effort in the app design, according to the shared responsibility model from AWS, a well designed app is on the developers side.

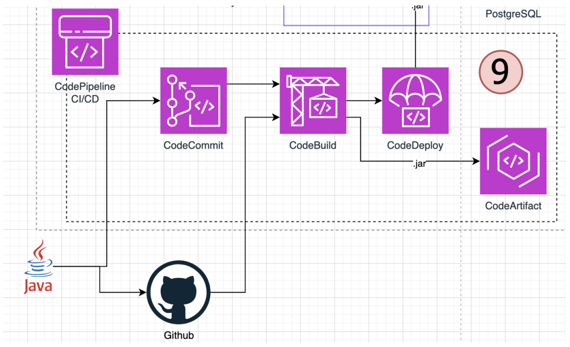
**6. Security Compliance:**
For this point, the recommendation is to use a third party service, there is a wide variety of options available in the market, which are specialized in this matter. Custom local implementations are possible, but involve a hard effort to build, maintain and implement. Which at POC level would deviate the effort dedicated to the main app development.

**7. Logging and Monitoring:**

**Cloudwatch**: Number 7 in the architecture diagram,
Amazon CloudWatch monitors your Amazon Web Services (AWS) resources and the applications you run on AWS in real time. You can use CloudWatch to collect and track metrics, which are variables you can measure for your resources and applications.
The CloudWatch home page automatically displays metrics about every AWS service you use. You can additionally create custom dashboards to display metrics about your custom applications, and display custom collections of metrics that you choose.
You can create alarms that watch metrics and send notifications or automatically make changes to the resources you are monitoring when a threshold is breached.

## CI/CD

For the continuous integration/deployment the number 9 in the diagram.



It uses a set of services provided by aws to maintain code and publish it to the respective cloud service. The proposed system consist of:

**AWS CodePipeline** is a continuous delivery service you can use to model, visualize, and automate the steps required to release your software. You can quickly model and configure the different stages of a software release process. CodePipeline automates the steps required to release your software changes continuously.

**AWS CodeCommit** is a version control service hosted by Amazon Web Services that you can use to privately store and manage assets (such as documents, source code, and binary files) in the cloud. Alternative to Github

**AWS CodeBuild** is a fully managed build service in the cloud. CodeBuild compiles your source code, runs unit tests, and produces artifacts that are ready to deploy.

**AWS CodeDeploy** is a deployment service that automates application deployments to Amazon EC2 instances, on-premises instances, serverless Lambda functions, or Amazon ECS services.

**AWS CodeArtifact** is a secure, highly scalable, managed artifact repository service that helps organizations to store and share software packages for application development.