Internet Banking System Architecture

Name: Freddy Sumba

Date: 26/04/2023

The following is a summary of the C4 Model for the Internet Banking System, which utilizes services in Amazon Web Services (AWS) and external services to deliver a scalable, secure, resilient, high-availability, and fault-tolerant architecture for the solution. In the sections that follow, I will address various architectural aspects and provide a comprehensive solution to the challenges at hand.

1. Regulatory elements:
    - Ecuadorian Personal data protection: Ensure privacy and security of customer data according to the Personal Data Protection Law.
    - Adherence to PCI DSS standard to ensure security in payment processing.
2. High availability (HA) and fault tolerance (DR):
    - Use of AWS cloud services that offer high availability and scalability.
    - Implementation of load balancing and server redundancy.
    - Backup and data recovery strategies.
3. Security and monitoring:
    - Implementation of Amazon WAF, Amazon Security Hub, and intrusion detection systems.
    - Encryption in transmission and storage of sensitive data.
    - Monitoring and auditing system and user activities through Amazon CloudWatch and CloudWatch Logs.
4. Front-end:
    - SPA application: Development with frameworks such as Angular or React.
    - Mobile application: Use of cross-platform frameworks like Flutter and Ionic.
5. Authentication and authorization:
    - Implementation of OAuth 2.0 with the "Authorization Code" flow and Keycloak for authentication and authorization management.
    - Integration of facial recognition in the onboarding and authentication process in the mobile application.
    - Offering multiple authentication methods with Keycloak architecture
6. System integration:
    - API Gateway: Implementation of Kubernetes Ingress as API Gateway to manage and control access to services.
    - Core services: Basic data query, movement query, and transfers.
    - Additional services: Inclusion of notification services using Twilio for SMS and Outlook for email, as well as interbank payments.
7. Information persistence and design patterns:
    - Use of Repository and Factory Pattern to abstract data access and allow the use of different data sources.
    - Implementation of an audit database that records all client actions.

    - Use of cache to temporarily store frequent customer information and speed up queries.
8. Cloud infrastructure and architecture patterns:
    - Use of AWS cloud services, such as EC2, S3, RDS, and Route 53 to ensure low latency and scalability.
    - Configuration of availability zones and auto-scaling groups to ensure high availability and fault tolerance.
    - Application of Architecture Pattern to organize and structure system components.
9. Scalability and performance:
    - Design of architecture and components with scalability and performance in mind.
    - Use of optimization techniques, such as data compression and static resource minification.
    - Monitoring and performance tuning over time to adapt to changing business and user needs.
10. Testing and software quality:
    - Establishment of automated and manual testing processes, including unit, integration, and acceptance tests.
    - Implementation of development practices such as Continuous Integration (CI) and Continuous Deployment (CD) to ensure software quality and accelerate releases.
    - Use of static and dynamic code analysis tools to detect quality and security issues, such as SonarQube (Java), ESLint (Angular).
    - Automated tests: JUnit and Mockito (Java), Jasmine and Karma (Angular).
    - Continuous Integration (CI) and Continuous Deployment (CD): Jenkins, GitLab CI/CD.
11. Documentation and training:
    - Creation and maintenance of up-to-date and accessible technical documentation for development, operations, and support teams.
    - Technical documentation generation: Javadoc (Java), Compodoc (Angular).
    - Establishment of a training program to ensure the team is familiar with technologies, design patterns, and architecture used; and could be solved with a partner or some training company.
12. Cost Management :
    - We can use AWS cloudwatch and AWS cost explorer to control de use and the cost associate to the solution.
    - Use a control cost solution software for manage the AWS cost stimation and eternal services like AWS Lambda Pricing and Calculator of AWS for services.

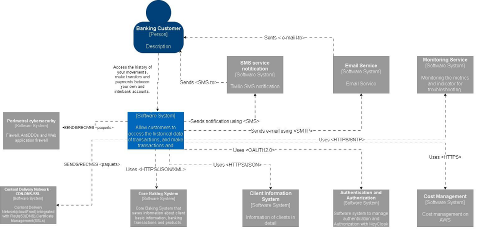
I am utilizing a methodology to define and specify the product or concept by posing four essential questions that offer clarity and foster a deeper understanding of each component or feature within the system. In the following table, I identify the most crucial components and features that need to be specified at the outset of the solution architecture.

Component/Feature	What is it?	What is it not?	What does it do?	What does it not do?
Access the history of movements	Functionality to view transaction history	An unrelated or non-banking feature	Allows users to review their past transactions	Modify or delete transaction history
Make transfers & payments between own accounts	Functionality to move funds within the user's accounts.	Non-banking or unrelated features	Enables users to manage funds between their accounts	Transfer or make payments to other users' accounts
Interbank transfers & payments	Functionality to transfer funds between different banks	Intra-bank transfers or non-banking features	Allows users to send money to accounts in other banks	Handle transfers within the same bank or unrelated transactions
SPA Application	Web interface for the banking system	Standalone back-end or APIs without user interfaces	Provides user-friendly access to banking services through a web browser	Work without internet access or serve as standalone applications
Mobile Application	Mobile interface for the banking system	Standalone back-end or APIs without user interfaces	Provides user-friendly access to banking services	Work without internet access or serve as

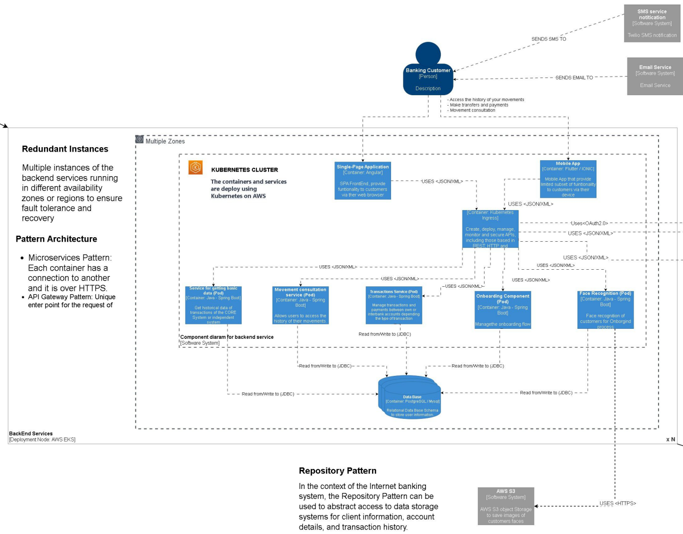
			through mobile devices	standalone applications
Data protection & regulatory compliance	Compliance with data protection laws & financial regulations in Ecuador	Non-compliance or ignoring regulations	Ensures customer data privacy and meets legal requirements	Violate data protection laws or bypass regulations in Ecuador
High availability & fault tolerance	Ensuring system availability and resilience to failures	A non-resilient system prone to downtime	Minimizes downtime and recovers quickly from failures	Guarantee 100% uptime or prevent all possible failures
Security & monitoring	Implementation of security measures and system monitoring	A system without security features and monitoring	Protects sensitive data and detects potential threats	Eliminate all security risks or detect all possible threats
Front-end applications (SPA)	Web interface for the banking system	Standalone back-end or APIs without user interfaces	Provides user-friendly access to banking services through a web browser	Work without internet access or serve as standalone applications
Front-end applications (Mobile)	Mobile interface for the banking system	Standalone back-end or APIs without user interfaces	Provides user-friendly access to banking services through mobile devices	Work without internet access or serve as standalone applications
Authentication & authorization	Secure user identity verification and access control	A system without user authentication or authorization	Manages user access and ensures secure authentication	Allow unauthorized access or bypass authentication
System integration (API Gateway, core & additional services)	Connection and management of system components	A monolithic architecture without integration	Facilitates communication and interaction between services	Operate as standalone components without connections

Data persistence & design patterns	Storage and organization of data using design patterns	A system without data storage or organization	Stores and manages data efficiently and consistently	Store data without structure or organization
Cloud infrastructure & architectural patterns	Utilization of AWS cloud services and architectural best practices	On-premise infrastructure or ad-hoc architecture	Ensures scalability, reliability, and cost-effectiveness	Rely on traditional infrastructure or use outdated architecture
Scalability & performance	Ability to grow and adapt to increasing demand	A rigid system with limited capacity	Handles increased load and maintains optimal performance	Scale indefinitely or provide constant peak performance
Cost management	Monitoring and optimizing system expenses	Ignoring or mismanaging costs	Balances system performance and budget constraints	Eliminate all costs or provide constant cost reduction
Support & maintenance	Ongoing system upkeep and issue resolution	A system without support or maintenance	Ensures business continuity and addresses system issues	Prevent all issues or

C4 Model Diagram of Internet banking System:
https://viewer.diagrams.net/?tags=%7B%7D&highlight=0000ff&edit=_blank&layers=1&nav=1&title=C4%20Model.png#Uhttps%3A%2F%2Fdrive.google.com%2Fuc%3Fid%3D17aUtkoZdsoK1KgCEmCMobwuVPuISKLXi%26export%3Ddownload



Level 1- Context



Factory Pattern

Used to create instances of different types of objects, such as notifications or authentication or another certain conditions or configurations.

Authentication and Authorization Architecture

Keycloak software has a huge use in the authentication and authorization process on legacy and cloud-native applications. It has the capability to control different authentication protocols, clients, and integrations. The support is provided by RedHat and the partners that offer it abroad in the technology field. In this use case, it could be an interesting solution to define different Realms(domain clients) for the front-end application and the mobile App controlling the authorization and authentication.

The detailed architecture of Keycloak was already defined by João Arthur Brunet Monteiro, the c4 model of Keycloak is described in the following post: https://joaoarthurbm.github.io/arqsoft-blog/posts/keycloak/

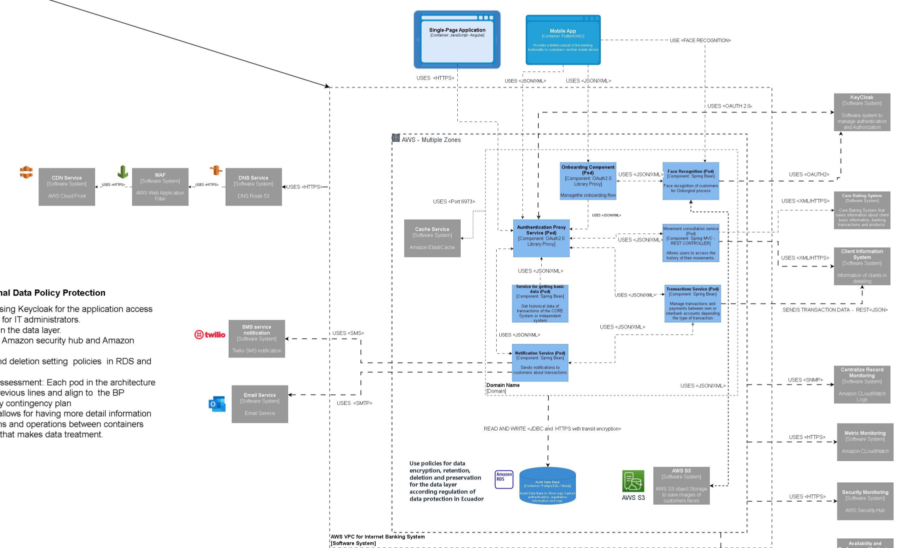
Repository Pattern

In the context of the Internet banking system, the Repository Pattern can be used to abstract access to data storage systems for client information, account details, and transaction history.

Level 2- Containers

Ecuadorian Personal Data Policy Protection

- Access control using Keycloak for the application access and IAM of AWS for IT administrators.
- Data encryption in the data layer.
- Monitoring using Amazon security hub and Amazon Cloudwatch.
- Data retention and deletion setting policies in RDS and S3 data layer
- Privacy Impact Assessment: Each pod in the architecture may follow the previous lines and align to the BP customer security contingency plan
- Audit Database allows for having more detail information about transactions and operations between containers and the process that makes data treatment.



PCI DSS

The architecture is align to PCI DSS standard to use credit cars in the payment process

Monitoring, logging, and alerting following the Observer Pattern

Incorporating these monitoring and logging solutions into the architecture, it be able to maintain visibility into your system operations, ensure optimal performance, and quickly detect and fix any security issues or threats.

Cost Management

Amazon Cost Explorer and Amazon Cloudwatch may help us to monitor and manage the cost of the architecture on AWS. In another way, the costs of external services need take into account in the total cost of the architecture.

Level 3 - Components