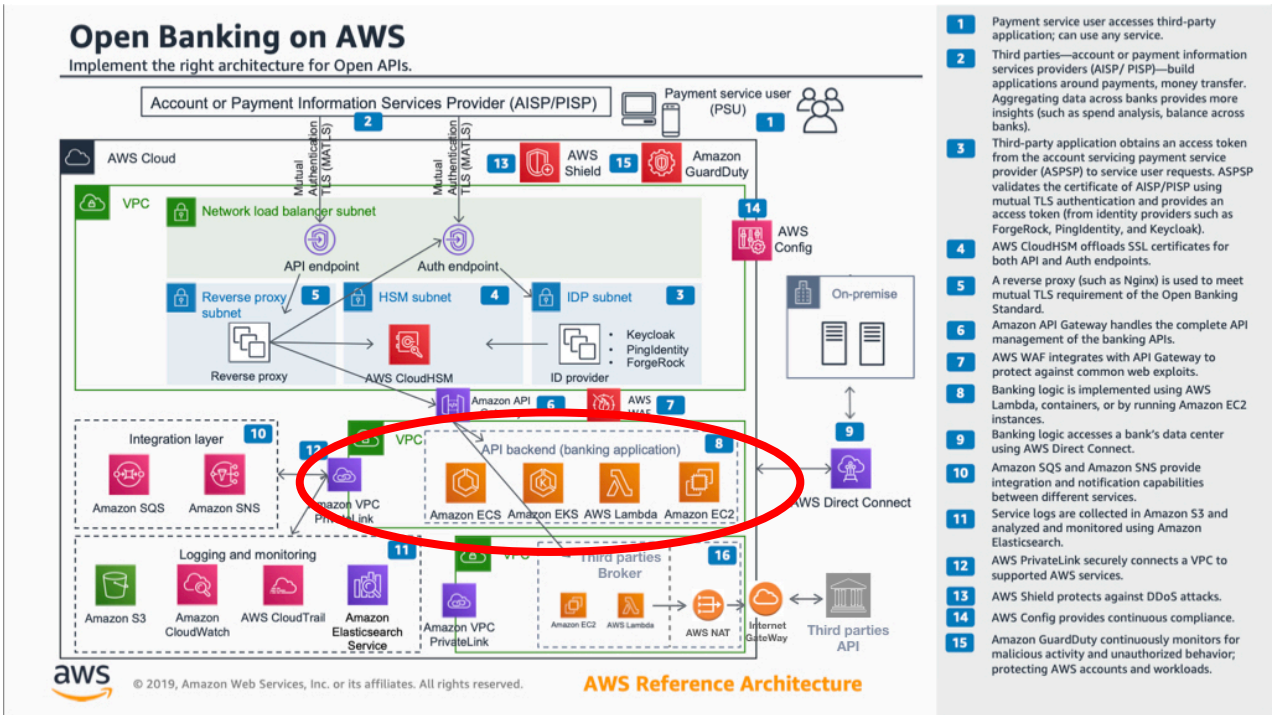
HW 4. Software Architecture Term Project Proposal

Open Banking on AWS

소프트웨어전공
20152791
강길웅

1 목표

1.1 Vision



[그림 1-1] 아키텍처 다이어그램

전체 시스템 중 API backend(banking application) 에 대한 구현을 본 프로젝트의 목표로 한다. 서버로 구동하여 정상적으로 통신하는 것까지를 목표로 한다. 단, AWS 기반 시스템 이나 AWS에 배포하지는 않으며, 로컬 서버를 이용하여 통신이 성공하는 것을 목표로 한다. 은행에서 계좌를 통해 이용할 수 있는 서비스(이체 및 조회)의 제공을 목표로 한다.

1.2 Scope

구현 할 기능은 계좌(예금)을 통한 서비스로 제한하며 보험 및 연금, 공제 등 계좌 범위를 넘어선 것은 제한한다. 또한 일체의 보안 절차와 보안관련 사항(사용자 인증 및 공인 증서 등)은 시스템 내부 타 레이어에서 제공함을 전제로 하기 때문에 보안 관련 사항은 배제한 채 구현을 진행한다.

2 개요

2.1 개요

은행에서 제공하는 계좌 서비스를 제공한다. 주요 서비스는 이체와 조회이다.

1. 전체 계좌 조회

   전체 계좌 조회 시 계좌번호 및 잔액과 계좌 종류(또는 이름)이 표기되도록 한다.

2. 계좌 세부 조회

   해당 계좌 번호와 계좌 종류(이름), 잔액이 표기되며 세부 내용에는 날짜(년월일시분초), 금액, 잔액, 거래 유형, 설명이 표기 되도록 한다. 또한 세부 조회는 다음과 같은 조회가 가능하도록 한다.

   A. 기간별 조회
      i. 당일
      ii. 1주일
      iii. 1개월
      iv. 3개월
      v. 6개월
      vi. 조건 검색
   B. 종류별 조회
      i. 입금
      ii. 출금

3. 이체

   이체와 관련된 정보를 가지고 이체를 실시한다. 보안과 관련된 사항은 스킵한다.

   A. 출금계좌번호
   B. 입금 계좌 정보
   C. 이체 금액
   D. 통장 표시 정보

2.2 특성

A. 실시간

   은행 서비스이기 때문에 금전적인 요소와 직결되므로 사용자의 요청에 대해 즉각적으로 반응하여 시스템에 적용 되어야 한다.

B. 데이터 무결성

   데이터 베이스 내부 데이터가 일관성있게 저장되어야 하며 데이터의 안정성이 보장 되어야 한다.

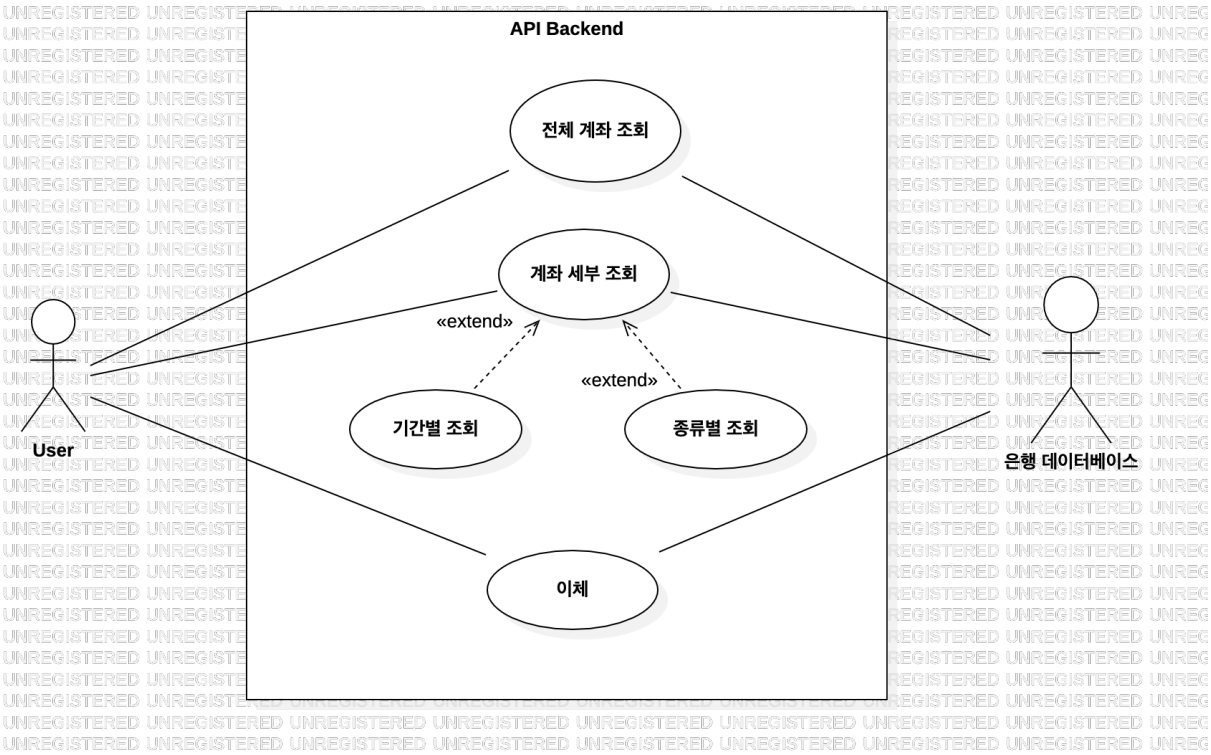
C. 정확성

   사용자의 정보와 데이터베이스 내부의 변경할 데이터가 정확히 일치 하여야 한다.

D. 보안

금전적 변경에 대한 서비스이므로 외부에 노출 되어선 안된다.

2.3 Use case Diagram



[그림 2-1] Use Case Diagram

2.4 FR

유스케이스명	전체 계좌 조회	
액터	유저, 은행 DB	
개요	은행 데이터베이스로부터 해당 유저의 계좌를 모두 조회한다.	
사전조건	유저 정보를 입력 받는다.	
사후조건	유저 정보에 알맞은 모든 계좌를 반환한다.	
기본흐름	유저	시스템
	1. 유저의 정보를 시스템에 입력한다.	2. 입력된 유저 정보의 존재를 확인한다. 3. 유저 정보가 존재한다면, 해당 정보에 관한 계좌 정보를 로드한다. 4. 계좌 정보를 반환한다.
대체흐름		2.a 유저 정보가 존재하지 않을 경우, 에러를 반환한다.

유스케이스명	계좌 세부 조회
액터	유저, 은행 DB

개요	은행 데이터베이스로부터 해당 계좌의 세부 내역을 조회한다.	
사전조건	계좌 전체 조회 이후, 사용자가 계좌 정보를 입력 한다.	
사후조건	해당 계좌의 세부 정보 및 거래 내역을 반환한다.	
기본흐름	유저	시스템
	1. 계좌를 선택한다.	2. 선택된 계좌에 대해 세부 정보 및 거래 내역을 은행 DB에 조회한다. 3. 조회한 데이터를 반환한다.
대체흐름		2.a 거래 내역이 없을경우 '거래 내역이 없습니다'를 반환한다.

유스케이스명	계좌 세부 조회 – 기간별 조회	
액터	유저, 은행 DB	
개요	은행 데이터베이스로부터 해당 계좌의 기간별 세부 내역을 조회한다.	
사전조건	계좌 전체 조회 이후, 사용자가 계좌 정보와 기간을 입력 한다.	
사후조건	해당 계좌의 기간별 세부 정보 및 거래 내역을 반환한다.	
기본흐름	유저	시스템
	1. 계좌와 기간을 선택한다.	2. 선택된 계좌에 대해 세부 정보 및 해당 기간의 거래 내역을 은행 DB에 조회한다. 3. 조회한 데이터를 반환한다.
대체흐름		2.a 거래 내역이 없을경우 '거래 내역이 없습니다'를 반환한다.

유스케이스명	계좌 세부 조회 – 종류별 조회	
액터	유저, 은행 DB	
개요	은행 데이터베이스로부터 해당 계좌의 종류별 내역을 조회한다.	
사전조건	계좌 전체 조회 이후, 사용자가 계좌 정보와 종류를 입력 한다.	
사후조건	해당 계좌의 세부 정보 및 종류별 거래 내역을 반환한다.	
기본흐름	유저	시스템
	1. 계좌와 거래 종류를 선택한다.	2. 선택된 계좌에 대해 세부 정보 및 거래 종류에 맞는 내역을 은행 DB에 조회한다. 3. 조회한 데이터를 반환한다.
대체흐름		2.a 거래 내역이 없을경우 '거래 내역이 없습니다'를 반환한다.

유스케이스명	이체
액터	유저, 은행 DB
개요	유저의 계좌에서 금액을 차감하여 지정된 계좌의 금액을 증가시킨다.

사전조건	계좌에 잔액이 존재한다.	
사후조건	올바르게 금액이 증감된다.	
기본흐름	유저	시스템
	1. 계좌를 선택한다. 2. 목표로 하는 계좌의 정보와 보낼 금액, 표기 내용 등 정보를 입력한다.	3. 올바른 계좌인지 파악한다. 4. 유저의 계좌에 잔액이 존재하는지 파악한다. 5. 유저의 계좌에서 금액을 차감시켜 목표 계좌의 금액에 더한다. 6. 이체 내역을 반환한다.
대체흐름		3.a 올바른 계좌가 아닐경우 오류를 표기한다. 4.a 잔액이 부족한 경우 이체 할 수 없다는 오류를 표기한다.

2.5 NFR & Quality Attribute

이름	설명	중요도
즉각성	제공하는 서비스가 즉각적으로 이루어져야 한다.	중
정확성	계좌 정보에 대해 유저 정보와 정확히 일치 하는 것이여야 한다.	상
보안성	해당 서비스는 외부에 노출되어서는 안된다. 시스템 내부 통신으로만 통신하여야 한다.	상
신뢰성	데이터베이스의 데이터가 항상 신뢰할 수 있어야 한다.	상
상호운용성	내부 모듈 및 DB와 상호 운용에서 문제가 없어야 한다.	중
법적 요구사항	유저의 정보를 수집/이용 할 때 법적 요구사항에 맞는 것이여야 한다.	하

3 사용 기술 & Framework

3.1 시스템 내부 사용 모듈

AWS환경의 적용은 본 프로젝트의 진행 목표가 아니므로, AWS 의 서비스를 대체하여 내부 모듈을 사용한다. 사용하는 모듈 및 대체 내용은 다음과 같다.

3.1.가 API Gateway

API Backend로의 접근을 내부 시스템 만으로 한정 짓도록 한다. 해당 모듈을 이용하여 보안 레이어와 직접적으로 연결 되지 않게 하여 독립성을 유지한다. 단, 본 프로젝트의 진행에서는 외부 접근을 허용하지 않고 로컬에서의 접근 만을 허용하는 것으로 대체한다.

3.1.나 Integration Layer

내부 시스템에서 긴급하게 문제가 생길 경우 이를 파악하기 위한 모듈이다. 서비스의 제공에서 문제가 발생할 경우 알려주는 서비스이나, 본 프로젝트에서는 로그로 대신한다.

3.1.다 Logging and monitoring

S3와 같은 저장소에 로그를 주기적으로 남기고 CloudWatch등을 이용하여 모니터링을 실시한다. 서비스의 유지 보수를 위해 로그를 남긴다. 본 프로젝트에서는 서버 로그를 일단위로 기록하고 실시간 로그를 확인하는 것으로 대체한다.

3.1.라 데이터 베이스

해당 시스템 외부의 데이터베이스가 존재하고 이는 데이터베이스 서버를 통해 접근, 이용하는 것으로 한다. 단, 본 프로젝트 진행에서는 이것을 로컬 데이터베이스로 대체하며 직접 연결하며 이용한다. 환경은 MySQL로 한다.

3.2 Framework 및 외부 모듈

3.2.가 Spring boot

빌드 시 내장 서버로 제공되는 간편함, 강력한 라이브러리, 개발자의 스택 등을 고려하여 Spring Boot 기반 WAS(Web Service Application)를 제작한다. Spring Boot는 Bean을 이용하며 Annotation을 이용한 각종 강력한 기능을 제공하기 때문에 구조 및 코드의 퀄리티 향상을 기대 할 수 있다.

3.2.나 Gradle

최근 Maven에서 Gradle로의 추세 변화가 이어지고 있다. Maven은 XML기반의 설정 방식 때문에 의존성 관리가 Gradle에 비해 복잡하며, 빌드 시간 또한 Gradle이 우세하여 Gradle을 이용하여 개발한다.

3.2.다 Inteli J

최근 많이 이용되는 IDE이다. 컨스트럭터나 get, set 메소드 등 개발 환경에서 많이 작성해야 하는 코드의 Auto generation이 가능하며, 개발자의 편의에 맞추어 시각적 환경 변경, Code Assistant제공 등의 기능을 갖추고 있다.

3.2.라 MyBatis

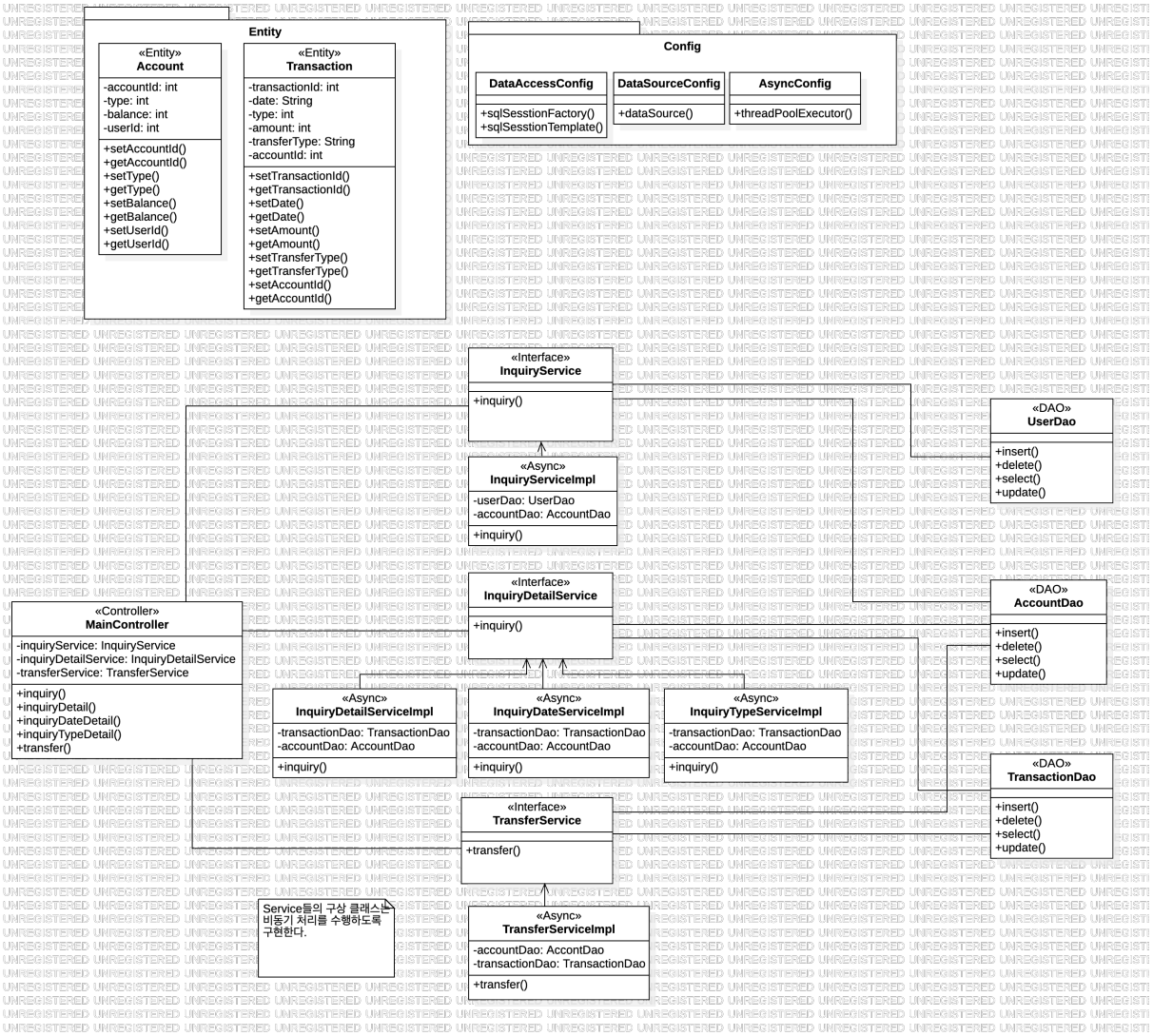
Insert, select 등과 같은 SQL문을 코드 내부에 넣는 것이 아닌 XML형태로 따로 관리 할 수 있도록 한다.

3.2.마 workbench

MySQL기반의 DB를 생성/관리 하기 위해 널리 이용되는 MySQL Workbench를 이용한다.

4 시스템 구성도 한페이지 이상

4.1 클래스 다이어그램



[그림 4-1] 클래스 다이어그램

4.1.가 Entity

시스템에서 이용하게 될 Entity. Account와 Transaction이 있으며 각 계좌 정보와 거래 정보를 입력하여 이용한다. Controller에서 생성한 뒤 Service등에 이용 할 때에는 DTO로 변경하여 이용한다.

4.1.나 Config

시스템 세팅에 관한 패키지이다. DataAccessConfig와 DataSourceConfig는 Mybatis를 이용하기 위함이고 AsyncConfig는 서비스를 비동기 처리하기 위한 스레드 생성 관련 Config이다.

4.1.다 Controller

요청을 파악하고 올바른 서비스를 호출하도록 한다. 이후 처리된 결과를 반환한다.

4.1.라 Service

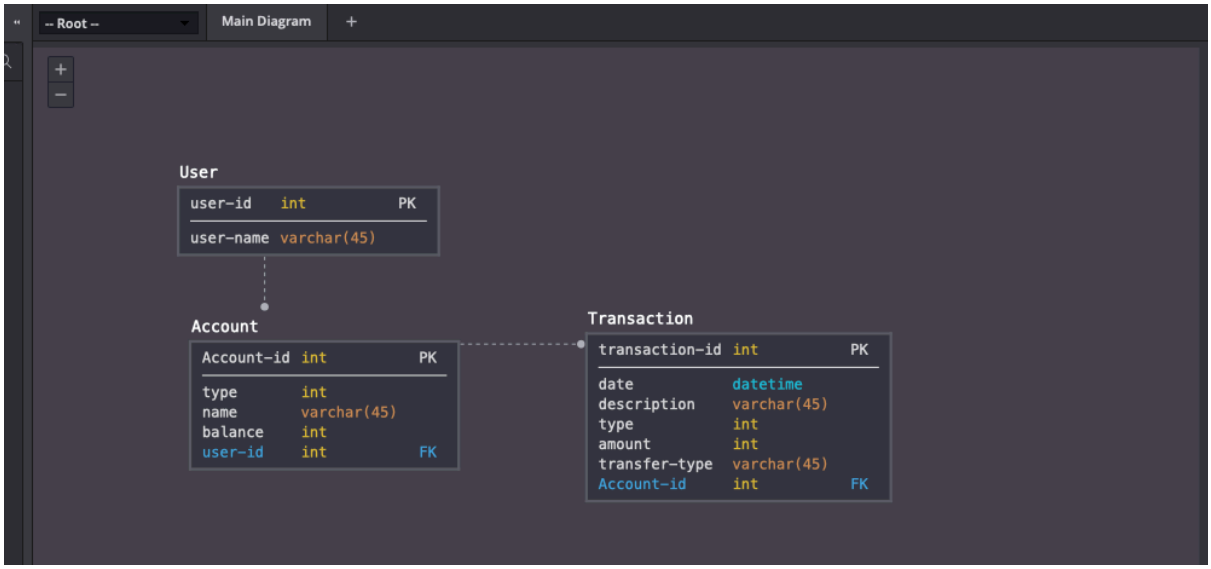
주요 서비스들의 로직을 포함하는 레이어. 전체 조회, 세부 조회, 이체 세가지로 나뉘며 세부 조회의 경우 다시 일반 조회, 날짜별 조회, 종류별 조회 세가지로 나뉜

다. 인터페이스로 타 레이어와 통신하며, 각 구상 클래스는 비동기처리를 하도록 하여 서비스가 동시에 여러 작업을 수행 할 수 있도록 한다.

4.1.마 Dao

DB에 접근하기 위한 레이어로 인터페이스로 구성하며 세부 SQL문은 XML형태의 Mapper를 작성하여 이용한다. Mapper와의 연결은 Config들을 통해 이루어지며 객체의 주입은 Bean을 AutoWired하게 주입하여 이용한다. User, Account, Transaction세가지 테이블을 위해 세가지 인터페이스가 존재한다.

4.2 DB 다이어그램



4.2.가 User

유저 정보를 담는 테이블. 보안관련 사항을 처리하지는 않기 때문에 패스워드 같은 정보는 담지 않으며 유저 파악 및 Account에서의 외래키를 위함.

4.2.나 Account

계좌 정보를 담고있는 테이블. Id는 계좌 번호이며, Type의 경우 적금, 예금과 같은 계좌 종류가 담겨지며 Name은 예금의 이름, balance는 잔액을 표기한다.

4.2.다 Transaction

거래 내역에 대한 정보를 가지는 테이블. 거래 일시, 설명, 거래 종류, 잔액 등이 표기 된다. Type은 입금인지 출금인지 내용이며, transfer-type의 경우 인터넷 이체, ATM이체 등의 정보가 담긴다.

5 아키텍처 스타일

5.1 Master-Slave

스레드를 이용하여 Master-Slave 아키텍처 스타일과 유사하게 설계하였다. 본래 Master-Slave 스타일과 같이 Slave가 이미 존재하는 것이 아니고, 요청에 따라 Slave가 생성되는 형태이다. 요청이 들어오면 해당 요청을 분석하여 Controller가 Service를 콜하는데, Service는 스레드를 이용하여 작업하게 되므로 Master의 스레드에서는 다시 요청을 받아 다른 스레드에게 작업을 주는 형태로 되어있다. 스레드 버젼의 Master-Slave 아키텍처 스타일이라고 할 수 있겠다.

5.2 Layered Architecture

책임에 따라 크게 Controller, Service, Dao세가지 레이어로 구분하여 설계 하였다. 컨트롤러는 요청을 받아 적절하게 핸들링하여 서비스에게 작업을 분배하고, 서비스는 작업을 수행한다. Dao는 DB와 관련된 SQL문을 수행한다. 각 레이어는 Interface로 통신하여 커플링을 낮추었으며 책임에 따라 클래스를 세부적으로 나누었다.

5.3 SOA (Service Oriented Architecture)

본래 SOA와는 조금 다르게 스레드를 이용한 SOA를 설계 하였다. SOA는 하나의 서버가 하나의 서비스를 담당하여 처리하는 방식을 취하는데 본 프로젝트에서는 하나의 스레드가 하나의 서비스 작업을 처리하는 것으로 설계 하였다. 하나의 서비스 클래스는 하나의 서비스만을 처리하며 이는 스레드로 처리된다. 컨트롤러에서 마치 SOA의 방식처럼 서버에 요청을 보내고 응답 받듯이, 스레드에 처리 요청을 보내고 스레드가 완료하면 값을 내보내도록 한다.

6 고려사항

6.1 스레드 문제

조회의 경우 스레드로 처리하더라도 결국 반환 해주기까지 메인 스레드가 멈춰있어야 정상 값을 반환해 줄 수 있다는 문제점이 있다.

➔ 리턴을 스레드에서 할 수 있도록 하는 방안을 찾는 중임.

6.2 보안

외부에서 오는 요청에 대해 블락시켜야 하므로 포트, 패스, 헤더 등을 파악하여 핸들링하도록 컨트롤러를 신경써서 구현해야 한다.

7 참고 문헌

AWS 기반 오픈 뱅킹(2019). AWS 참조 아키텍쳐 다이어그램 ,

https://d1.awsstatic.com/architecture-diagrams/ArchitectureDiagrams/open- banking-on-aws.pdf?did=wp_card&trk=wp_card

AWS 설명서(2020). AWS 설명서,

https://docs.aws.amazon.com/index.html?nc2=h_ql_doc_do_v

자바와 스프링의 비동기 기술(2019). KimJongMin, Git hub.io,

https://jongmin92.github.io/2019/03/31/Java/java-async-1/