BackEnd

- Java, Kotlin
- Spring Boot
  - Web, Security, Data JPA
- Python
- FastAPI
- JUnit5, Kotest, Mockito
- JPA, QueryDSL, Kotlin-JDSL, Mybatis
- Gradle
- IntelliJ
- Git
- Kafka (이벤트 발행/구독, Lag 자동 처리)
- Serverless (AWS Lambda 기반 자동화 개발)
- Scheduler / Batch 처리 (Cron 기반 동기화 및 대량 데이터 Retry)

FrontEnd

- jsp
- vue.js

Infra	• Docker • MS-SQL, MySQL, Druid DB • RabbitMQ • NIFI • AWS • Jenkins • Openshift • Kibana
API / Integration	• Meta Open API • TikTok Open API • Slack API • Jira API • Confluence API
Data / Pipeline	• 이벤트 기반 데이터 파이프라인 설계 • 상품 데이터 동기화 및 마이그레이션 • 대량 API 벌크 처리 및 실패 재시도 로직 구성

💼 업무 경험

카페24 쇼핑몰 플랫폼 개발/운영 (2025.01 ~ 2025.09)

> 💡 카페24 판매자 상품 이벤트와 외부 커머스 플랫폼(Meta, TikTok)을 연동하는 백엔드 개발을 담당했습니다.
>
> Python + FastAPI 기반의 API 서버 개발과 Kafka 이벤트 스트림 처리,
>
> 그리고 Slack·Jira·Confluence·AI 기반 자동화 시스템 개발 등을 수행했습니다.

- **사용 기술**
  - **Python**, **FastAPI**, **Kafka**, **MySQL**, **Redis**
  - **카페24 자체 물리 서버**, **카페24 클라우드 서버**
  - Slack API, Jira REST API, Confluence API
  - 카페24 서버리스, Cron 기반 배치

- 주요 성과
  - **Meta Shops 연동 시스템 개발**
    - 카페24 판매자 페이지에서 발생하는 **상품 등록/수정/삭제 이벤트를 Meta API와 자동 동기화**하는 서비스 개발
    - Meta 신규 정책에 따라 샵에즈 **이용자와 기존 Meta Shops 이용자를 구분하는 식별자 기반 마이그레이션 작업 수행**
      - Meta Throttle(대량 요청 제한)을 해결하기 위해:
        - 대량 이벤트를 **1000건 단위 벌크 처리**
        - 실패 건 **Retry 기록 DB 구축 → 자동 재처리 로직 구현**
        - 일정 기준 초과 시 자동 백오프 및 큐 재조정 기능 개발
        - 결과적으로 **안정적인 동기화율 확보 및 동기화 실패율 대폭 감소**
  - **TikTok 쇼핑 연동 개발**
    - 카페24 상품 이벤트를 Kafka로 발행·구독하는 구조에서
      이벤트 가공 → 어그리게이션 → 후처리 로직 전체 설계 및 개발
    - 상품 품절/상태변경 등 조건에 따라 **TikTok Open API 호출을 자동으로 수행**
    - 대량 데이터 병목 해결:
      - 상품을 A~Z 구간 기반으로 분할 처리
      - Cron 기반 “TikTok 상품 승인 상태 조회 → DB Sync 프로세스” 개발
    - 그 결과 **대량 이벤트에도 안정적으로 동기화되는 연동 구조 완성**
  - **AI 기반 자동 배포계획서 생성 시스템 개발**
    - Slack → Jira → Confluence API 연동하여

      배포 버전 감지 → 관련 Jira 티켓 자동 수집 → AI 요약 → Confluence 문서 자동 생성
    - 팀의 반복적인 배포계획서 문서 작업을 자동화하여

      배포 문서 작성에 소요되던 시간을 실질적으로 절감
  - **Slack 스레드 AI 요약 자동화**
    - Slack Thread에서 대량 메시지가 쌓였을때 특정 메시지를 보내면 그걸 감지해서

      서버리스 환경에서 AI 요약 → Slack 자동 응답 기능 개발
    - 회의/이슈 논의 내용을 따로 정리할 필요 없이

      실시간 자동 요약 프로세스 구축

- **문서 작업**
  - 연동 시스템 구조도 및 데이터 흐름 문서 관리
  - 자동화 도구 개발 시 API 스펙 또는 사용 방법 문서화
  - 장애 발생 시 Slack Bot 로그 기반 장애 이력 정리 및 운영 대응 기록
- **회고(~ing) – Kafka 기반 이벤트 처리 구조**
  - **가장 고민이 많았던 부분**
    - 카페24에서 Kafka는 팀 전용이 아닌 **공용 Kafka 환경**이라 → 토픽 생성 권한이 없고, 카프카 운영팀을 통해서만 생성/변경이 가능했습니다.
    - 이 제한 때문에 컨슈머 구현시 **한 토픽에 몰려 있는 모든 메시지를 소비하면서 이벤트 타입을 조건으로 필터링**하는 방식으로 구현해야 했습니다. 그러나 이 방식은 유지보수성과 안정성이 떨어져 어려움이 많았습니다.
    - 또한, 기존에는 **비동기 Python Kafka 라이브러리**를 사용했는데, 처리 속도는 빠르지만 **로그 순서가 뒤죽박죽**으로 찍히는 문제가 발생했습니다. 여러 쓰레드로 메시지를 비동기로 처리하다보니 메시지가 순서대로 처리되지 않아, 성능 로그 측정과 문제 추적이 어려웠습니다.
  - **구체적인 해결 과정**
    - 빠른 시일 내에 요구사항을 맞추기 위해 기존 비동기 구조를 그대로 사용하되, 메시지 처리 시 이벤트 타입을 조건으로 필터링하여 필요한 데이터만 소비하도록 구현했습니다.
    - 향후 같은 개발을 맡는다면, **비동기 라이브러리 대신 동기 라이브러리를 사용하고**, 파티션과 컨슈머 수를 늘려 **순서 보장과 처리 속도**를 동시에 확보하는 방안을 고려하고 있습니다.
    - 또한 가능하다면 팀 전용 Kafka 서버(클라우드 또는 별도 물리 서버)를 분리하여, 토픽 생성과 관리가 자유로운 환경을 만드는 것도 개선 방안으로 고려할 수 있습니다.
  - **얻은 교훈**
    - **트레이드오프를 이해하고 설계할 것**: 비동기 라이브러리를 쓰면 처리 속도는 빨라지지만 로그와 메시지 순서가 뒤섞일 수 있다는 것을 경험했습니다. 요구사항에 따라 **속도와 안정성 중 무엇을 우선할지 판단하고, 구조를 선택**해야 한다는 점을 배웠습니다.
    - **전용 환경 확보의 중요성**: 공용 Kafka 환경에서는 급한 대응이 어렵고 유지보수 부담이 크므로, 가능하다면 **팀 전용 인프라**를 확보하는 것이 효율적임을 체감했습니다.

- **회고 (~ing) – AI 자동화 시스템**
  - **가장 고민이 많았던 부분**
    - 카페24에서는 “AI 자동화”를 강하게 요구하여 빠른 일정에 맞추다 보니 여러 어려움이 있었습니다.
    - 팀원마다 Jira 티켓 정보 입력 방식이 달라 패턴 추출이 어려웠고, Confluence 문서를 중간에 수정하면 DB 기록과 엇갈리는 문제가 발생했습니다.
    - 배포계획서 자동 생성 스케줄도 모든 팀원이 동일한 날짜 기준으로 Jira를 정리해야 했지만, 이 규칙이 애매하게 관리되었습니다.
  - **구체적인 해결 과정**
    - 초반에는 개인이 임시로 규칙을 정하고 시스템을 구현하여 자동화 기능을 작동시켰습니다.
    - 배포 계획서 자동 생성 기능은 **DB에 기록하고 추가된 티켓만 반영**하는 방식으로 설계했습니다.
    - 시스템 완성 후에는 팀원과 소통하며 **규칙 강제화**를 시도했지만, 충분한 초기 합의가 이루어지지 않아 일부 혼선이 있었습니다.
  - **얻은 교훈**
    - **초기 규칙 합의의 중요성**: 자동화를 시작하기 전에 Jira 티켓 제목, 라벨/이슈 타입, Confluence 수정 금지, 배포 일정, API 호출 시점 등 규칙을 명확히 정하고 팀 전체 합의가 필요함을 배웠습니다.
    - **유지보수성을 고려한 설계 필요성**: 시스템 기능 자체가 뛰어나더라도, 팀원들이 규칙을 준수하지 않으면 자동화 효과가 제한된다는 점을 경험했습니다.

KT 광고 CMS 개발/운영 (2023.02 ~ 2024.12)

KT 광고 CMS 플랫폼의 개발 및 고도화를 담당하며 다양한 기능을 개선하고 운영 중입니다.
매일 약 600만건 이상의 데이터를 처리하여 관리하고 있으며
매월 새로운 개발 과제를 수행하고, 동시에 사용자 지원(CS)도 처리하고 있습니다.

- **사용 기술**
  - Java 8, Spring Boot 2.X, MyBatis, MS-SQL, Redis, Kubernetes, Jenkins, Openshift

- 주요 성과
  - **API 성능 최적화**
    - 기존 10초에 달하던 API 응답 시간을 1초 이하로 단축했습니다. 불필요한 API 호출을 줄이고 쿼리를 최적화했습니다. 특히, 실행 계획을 분석하여 필요한 인덱스를 추가하고, 대용량 테이블의 비정규화를 통해 조회 속도를 크게 개선했습니다.
  - **조회속도 향상을 위한 테이블 파티셔닝**
    - **문제 상황**
      - 배치 서버에서 시간마다 FTP로 전송된 CSV파일을 이용해 하루평균 **600만 건**의 노출 로그 데이터 적재.
      - 3년 동안 축적된 데이터는 약 **60억 건**으로 대용량 테이블로 인해 **데이터 적재 및 인덱스 생성 속도 저하** 발생.
      - 실제 **정책 상 6개월치 데이터**만 필요하지만, 복구 및 연간 데이터 분석을 위해 **연 단위로 데이터 유지**가 필요한 상황.
    - **개선 방향 및 구현 방안**
      1. **데이터 삭제 프로세스 개선**
         - 연 단위로 데이터를 관리하도록 설계.
         - **최신 연도인 2024년의 연 단위 데이터**를 유지하고, 오래된 연도의 데이터는 주기적으로 삭제하도록 배치 프로그램 구현.
         - **최신 연도의 데이터만 유지**되도록 관리.
      2. **테이블 파티셔닝 적용**
         - **6개월 단위 파티션**을 추가 적용하여 최신 데이터를 효율적으로 관리.
           - **파티션 함수**를 생성하여 데이터를 분리
           - **파티션 스킴**을 생성하여 파티션 데이터가 저장될 파일 그룹을 지정
           - 데이터 삽입 시 LOG_DT를 기준으로 자동으로 적절한 파티션에 데이터가 저장
           - where 문에 log_dt를 조건으로 주면 자동으로 적절한 파티션만 접근
         - 조회 시 **최신 연도의 데이터**에서 필요한 **6개월 데이터가 포함된 파티션**만 접근하도록 설정.
      3. **결과 및 효과**
         - **데이터 생성 및 조회 성능 개선**: 대용량 데이터에서 특정 연도나 기간만 빠르게 조회 가능.
         - **공간 최적화**: 필요하지 않은 데이터를 주기적으로 삭제

- **파일 검증 로직 개선**
  - 파일 업로드 시 각기 다른 부분에서 유효성 검사를 수행하던 문제를 AOP로 일원화하여 코드의 유지보수성을 높였습니다. 이를 통해 파일 확장자와 MIME 타입 검증이 보다 효율적이고 일관되게 이루어졌습니다.
- **프로시저의 Java 전환**
  - 프로시저의 긴 동작시간과 중간 상태 파악의 어려움을 해결하기 위해 Java로 변환하여 로그를 통해 진행 단계를 추적할 수 있도록 했습니다. 또한 조회 쿼리의 실행계획을 분석해 빠져있는 인덱스를 추가하고 bulk insert를 활용하여 동작 시간을 절반으로 줄였습니다.
  - 배치 실패 시 날짜를 넣어 다시 동작시킬수 있도록 API 구현
- **CS 처리 자동화**
  - 반복적이고 정형화된 CS 작업을 자동화하여 운영 효율성을 크게 향상시켰습니다. 이를 통해 인수인계 과정이 간소화되고, 작업 시간도 대폭 줄어들었습니다.
- **비동기 이벤트 처리 도입**
  - 광고 저장과 메일 전송이 동기로 이루어져 메일 전송 실패 시 전체 작업이 롤백되는 문제를 비동기 이벤트 처리로 개선하여 시스템 안정성을 높이고 관심사를 분리하였습니다.
- **엑셀 다운로드 성능 개선**
  - 기존 45초가 걸리던 엑셀 다운로드 시간을 CompletableFuture를 이용한 비동기 처리로 15~18초로 단축했습니다.
- **더미 데이터 생성**
  - 개발 환경에서 다양한 시나리오를 테스트할 수 있도록 주기적인 더미 데이터 생성 및 자동 삭제 기능을 구현했습니다.
- **화면 데이터의 비동기 호출**
  - 여러 탭의 데이터를 동기 방식으로 불러오는 기존 방식을 비동기로 전환하여, 사용자가 빠르게 조회된 데이터를 먼저 확인할 수 있도록 개선했습니다.
- **JS, JSP 파일의 모듈화**
  - 각 페이지에서 중복되는 기능을 모듈화하여 코드의 유지보수성을 높였습니다.
- **조회 성능 최적화**
  - mybatis resultMap 매핑 시간이 오래걸려 쿼리를 분리한 뒤 애플리케이션상에서 메모리 조인을 통해 매핑해줌으로써 페이지 조회하는데 총 25초 걸리던시간을 5초로 단축하였습니다.

- **문서 작업**
  - ERD 클라우드 작성 및 매달 개발 이력 문서 작성
  - 장애 발생 시 이력 기록 및 CS 처리 문서 작성
- 회고(~ing)
  - **가장 고민이 많았던 부분**
    - 프로젝트 중 가장 큰 고민은 **API 성능 최적화**였습니다. 처음 작업을 맡았을 때 API 응답 시간이 10초이상 소요되어 사용자가 대기하는 시간이 지나치게 길었고, 시스템 성능도 저하되고 있었습니다. 문제의 원인을 정확히 파악하고 이를 어떻게 해결할지 고민이 많았습니다. DB 쿼리 성능 문제인지, 네트워크 병목인지, 또는 불필요한 API 호출 때문인지 여러 요소를 고려해야 했습니다.
  - **구체적인 해결 과정**
    - 제일 먼저 **실행 계획을 분석**하여 쿼리 최적화가 필요하다는 결론을 내렸습니다. 이를 통해 쿼리 자체를 개선하고, 대용량 테이블에 **인덱스**를 추가했습니다. 또한, 조회를 위해 대용량 테이블의 비정규화를 통해 조회 모델을 만들었고 이를 통해 데이터 조회 성능을 개선했습니다. 그 외에도 불필요한 API 호출을 줄이는 방향으로 시스템 구조를 개선했습니다.

      이 과정을 통해 기존 10초에 달하던 API 응답 시간을 1초 이하로 줄이는 데 성공했습니다. 이 경험은 단순히 성능 최적화 기법을 배우는 것뿐만 아니라, 문제를 단계적으로 분석하고 해결하는 과정을 체험할 수 있는 기회였습니다.
  - **얻은 교훈**
    - **근본적인 원인 파악의 중요성**: 이 프로젝트를 진행하면서 가장 크게 느낀 부분은 근본적인 원인을 파악하는게 얼마나 중요한지 알게 되었습니다. 문제를 단순히 성능 저하로 보고 접근했다면 일부 문제만 해결하고 근본적인 원인은 남았을 것입니다. 하지만 실행 계획을 세밀하게 분석하고, 쿼리와 데이터 구조까지 깊이 들여다본 덕분에 근본적인 성능 문제를 해결할 수 있었습니다.
    - **데이터 관리의 중요성**: 이전에는 데이터 관리가 제대로 되지 않은 상태에서 대용량 데이터를 다루다 보니, 조회나 생성 작업에 시간이 많이 걸렸는데, 이번에 데이터를 체계적으로 정리하면서 전체적으로 성능이 향상되는 걸 경험했습니다.

KIBA 광고 CMS 개발/운영 (2023.04 ~ 2024.12)

KIBA 광고 CMS 플랫폼의 운영 및 개발을 담당하고 있으며, 주요 업무는 CS 처리 및 시스템 안정화입니다.

- **사용 기술**
  - Java 8, Kotlin, Vue.js, AWS, ELK, Jenkins, MySQL, Druid, NIFI, Rabbit MQ

- 주요 성과
  - **데이터 복구 및 업그레이드**
    - AWS Glue와 Nifi를 사용해 데이터 복구 작업을 수행했으며, MySQL DB의 버전 업그레이드를 성공적으로 완료했습니다.
  - **CS 처리 자동화**
    - CS 작업을 API를 통해 운영팀이 직접 처리할 수 있도록 개선하여, 처리 시간을 줄이고 작업 효율을 높였습니다.
  - **서버 관리 및 배포**
    - 개발 및 운영 서버의 정기적인 배포와 관리 작업을 수행
  - **화면 유효성 검증 기능 추가**
    - Vue.js로 작성된 화면에 유효성 검증 기능을 추가하여, 오류 발생 시 사용자에게 명확한 피드백을 제공하고 특정 조건에서는 이벤트 발생을 차단하도록 개선했습니다.
  - **모수/인벤토리 분석 페이지 구현**
    - Vue.js와 Kotlin을 사용해 모수 및 인벤토리 분석 페이지를 구현했습니다.
  - **ELK + Nginx 자동화**
    - ELK 인스턴스 실행 시 Docker Compose 파일에 Nginx 설정을 추가하여, 두 서비스를 동시에 실행할 수 있도록 자동화했습니다.
  - **모수/인벤토리 분석 쿼리 호출 최적화**
    - 분석 api 요청 시 130개 이상 조회되는 쿼리를 프로젝션을 이용하여 4개만 조회되도록 최적화했습니다
  - **GitLab Repository 이관**
  - **Cloud Watch를 이용해 AWS Glue를 이용한 데이터 복구 작업의 성공 여부를 실시간으로 모니터링하고, 자동으로 이메일 알림을 받아 복구 작업의 편의성과 관리 효율성 증대**
    - **SNS 주제 생성**
      - Amazon SNS 콘솔에서 새 주제를 생성하고, 이메일을 구독
    - **CloudWatch에서 Glue 작업 상태 감시**
      - CloudWatch Events에서 규칙을 만들고, 이벤트 패턴에 Glue 작업이 완료되었을 때를 트리거로 설정
    - **SNS 주제로 전송**
      - CloudWatch 이벤트가 발생할 때 **대상** 설정을 SNS 주제로 알림을 전송하도록 설정

    - **AWS Glue 특정 job에서 Aurora MySQL 및 S3 접근이 가능하도록 Connection 설정 및 VPC GateWay 설정**
    - **문서 작업**
        - ERD 클라우드 작성
        - 장애 발생 시 이력 기록 및 CS 처리 방법에 대한 문서 작성
        - AWS 비용 증명을 위한 CS 처리 이력 기록
- 회고(~ing)
    - **가장 고민이 많았던 부분**
        - 프로젝트에서 가장 큰 고민은 프로젝트에 대해 알고 있는 사람이 없다는 점이었습니다.
          회사에서 기존 프로젝트 외에 SM 프로젝트를 맡게 되었는데, 인수인계가 불완전해서 팀원들이 모두 나가고 저만 남게 되었습니다. 인수인계 문서에는 당장 급한 일만 처리할 수 있는 내용만 있었고, 문제 해결은 되었지만 프로젝트를 정확히 이해하지 못한 상태였습니다.
          그리고 많은 CS 작업이 수동으로 이루어지고 있어 자동화를 하고 싶지만 비즈니스 지식 부족과 검토해줄 사람이 없어 선뜻 진행하지 못했습니다.
          또한, AWS를 처음 접하다 보니 이미 구성된 상태에서 설정된 것들에 대해 이해하기가 어려웠습니다.
    - **구체적인 해결 과정**
        - 혼자서 다양한 작업을 하면서 억지로 알아가면서 진행하다 보니, 완벽하게 이해한 것은 아니지만 방법만이라도 알게 되었습니다. 그 과정에서 작업 내용을 정리하고, 플로우차트와 ERD를 작성하여 어느 정도 파악할 수 있었습니다. 그 덕분에 CS 처리 작업을 자동화할 수 있었고, AWS와 관련된 작업도 어느 정도 이해하게 되었습니다.
    - **얻은 교훈**
        - 이 프로젝트를 진행하면서 가장 크게 느낀 부분은
            - **철저한 문서화의 중요성**: 인수인계가 제대로 이루어지지 않아 정말 힘들었지만 그 과정 속에서 작업 내용과 프로세스를 체계적으로 정리하면서 다른 문제를 해결할때도 도움이 되었고 향후 들어온 팀원들에게 인수인계를 하는데 정말 큰 도움이 되었습니다.
            - **자기 주도적인 학습**: 팀원들이 떠나고 혼자 남게 되었을 때, 스스로 학습하고 문제를 해결해 나가는 과정이 중요하다는 것을 알게 되었습니다. 스스로 알아가면서도 필요한 자료를 만들어가는 것이 업무에 있어 큰 도움이 되었습니다.
            - **자동화의 가치**: 매달 최소 5회 이상씩 하던 반복적인 수동 작업을 자동화하여