Generated at 2026-02-22T11:39:06.362Z | Variant: 성과중심형

신주용 | 성과 중심 백엔드 이력서

연락처

- Contact: 010-8392-1580
- Email: sea71511@gmail.com
- GitHub: https://github.com/SJY0917032
- Blog: https://sjy0917032.github.io/

요약

OTA, O2O 도메인에서 4년차 백엔드 엔지니어입니다. 71개국 / 400만 대 규모 해외 렌터카 인벤토리 시스템을 설계·운영했고, Google Sheets로 돌아가던 스타트업 업무를 NestJS 기반으로 전환하는 작업을 주도하고 있습니다. 비즈니스 요구를 기술 구조로 옮기고, 운영에서 측정 가능한 성과를 내는 걸 중요하게 생각합니다.

핵심 키워드

문제해결 · 정량성과 · 재현가능성

성과 지표 하이라이트

- 발송 실패 월 2~3회 → 0건 / 수기 작업 완전 제거
- 수기 대응 월 3~5회 → 0회 / 장애 인지 시점: 다음 날 → 즉시
- 월 3회 수기 작업 제거 / PG 추가 시 Adapter 1개 구현으로 완료
- 연간 약 400만 원 인프라 비용 절감
- 7개국 → 71개국 / 400만 대
- 월 140만 → 12만 원 (91% 절감) / 데이터 유실 없음
- 패키지 상품 출시 / 호텔 매출 내 패키지 비중 60%
- 타깃 지역 일 평균 예약 200 → 400건
- 발급 소요 30분 → 1분 / 수기 처리 월 5건 → 0건
- 업체별 SQS 메시지 독립 처리로 전환. 한 업체 실패가 다른 업체로 전파되지 않음

경력 상세 (문제-해결-성과)

뽀득 | 백엔드 개발자 | 2025.04 ~

- 조직/도메인 맥락: 식기세척 O2O 스타트업. 입사 당시 내부 업무가 구글 시트와 스크립트를 통해 운영 됐으며 일부 시스템은 레거시 시스템으로써 유지보수가 정상적으로 진행 되고 있지 않았습니다.

인수증 발행 시스템 이관

- 문제: GAS 기반 동기 파이프라인이라 업체 하나가 실패하면 뒤 업체 전부 발송이 멈추는 구조였습니다. 실패 알림도 없어서 다음 날 업체 전화로 알게 되는 상황이 월 2~3회 반복됐습니다.
- 해결: 업체별 SQS 메시지 독립 처리로 전환. 한 업체 실패가 다른 업체로 전파되지 않음
- 성과: 발송 실패 월 2~3회 → 0건 / 수기 작업 완전 제거
- 사용 기술: NestJS, SQS, Lambda(Python), NHN Cloud SMS

TMS 레거시 개편

- 문제: 이전 담당자 퇴사로 API 문서도 테스트도 없는 TMS 연동 코드를 인수했습니다. 타임아웃이 나도 재시도가 없어서 운영팀이 직접 TMS 콘솔에 들어가 수기로 처리하는 일이 월 3~5회, 건당 30분씩 발생했습니다.

- 해결: 인터페이스 추상화 + MockAdapter 분리로 로컬 테스트 환경 확보
- 성과: 수기 대응 월 3~5회 → 0회 / 장애 인지 시점: 다음 날 → 즉시
- 사용 기술: NestJS, MySQL, Slack Webhook

결제 자동화 및 PG 추상화

- 문제: API가 없는 CMS라 담당부서와 개발팀이 매월 3번씩 수기업무를 통해 CMS 데이터를 갱신하고 있엇습니다. 데이터가 늘어날수록 수기 작업 시간이 늘어났고, 휴먼에러가 지속 발생하여 이를 PG 연동을 통한 정기결제 시스템으로 변경 했습니다.
- 해결: 토스페이먼츠 빌링키 기반 정기결제로 자동화
- 성과: 월 3회 수기 작업 제거 / PG 추가 시 Adapter 1개 구현으로 완료
- 사용 기술: NestJS, 토스페이먼츠, 이니시스

NestJS 어드민 시스템화

- 문제: 운영 업무 전체가 Google Sheets 기반이었습니다. 시트 수십 개가 서로 참조 걸려 있어서 하나 바꾸면 다른 시트가 깨지고, 누가 데이터를 바꿨는지 추적할 방법이 없었습니다.
- 해결: 팀·파트·인원 3계층 RBAC 설계. 도메인별로 NONE/READ/WRITE/MANAGE 4단계 권한 부여
- 성과: 장애 대응 시간 단축 및 운영 자동화 달성
- 사용 기술: NestJS, TypeORM, MySQL, RBAC

AWS 인프라 개편

- 문제: 인수증·TMS 이관 완료 후 불필요해진 전용 Windows EC2와 RDBMS를 운영 종료하고 메인 RDS로 통합했습니다.
- 해결: 도메인 모델링과 인터페이스 분리로 안정적인 처리 구조를 구현
- 성과: 연간 약 400만 원 인프라 비용 절감
- 사용 기술: AWS EC2, RDS

기타

- 문제: 운영 과정에서 반복되는 병목/장애가 발생
- 해결: **업체 데이터 동기화**: Google Sheets → DB 주간 수기 동기화를 NestJS Cron + Sheets API로 자동화
- 성과: 장애 대응 시간 단축 및 운영 자동화 달성
- 사용 기술: NestJS, TypeScript

팀오투 (카모아) | 백엔드 개발자 → 파트 리드 | 2022.07 ~ 2025.03

- 조직/도메인 맥락: 해외 렌터카 OTA. 입사 시점 해외 인벤토리 7개국 / 2천 대에서 퇴사 시점 71개국 / 400만 대까지 확장됐습니다. 공급사 통합 아키텍처 설계, 인프라 비용 최적화, 패키지 상품 출시 등을 담당했습니다.

해외 공급사 API 통합 — GOTAR

- 문제: 공급사마다 API 형식이 달랐고(REST / SOAP), 기존 모놀리스 구조에서는 공급사 하나가 느려지면 서버 전체가 영향을 받았습니다. 해외 공급사가 계속 늘어날 예정이라 구조를 바꿔야 했습니다.
- 해결: 도메인별(차종·차량·업체·지점·주문·예약) 공통 인터페이스(GOTAR 스키마)를 정의하고 npm private 패키지로 배포. 래핑 서버와 코어 서버가 같은 타입을 참조해서 타입 불일치를 원천 차단
- 성과: 7개국 → 71개국 / 400만 대
- 사용 기술: NestJS, TypeScript, ECS

DocumentDB 스토리지 비용 91% 절감

- 문제: AWS 비용을 항목별로 보다가 DocumentDB가 월 140만 원 나오는 걸 발견했습니다. 하루 최대 1,500건 예약 규모에 비해 과한 비용이었고, 원인은 서비스 초기부터 쌓인 수년치 데이터를 그대로 두고 있었던 것이었습니다.
- 해결: 사업부에 보존 기간 확인 후 TTL Index로 1개월 초과 데이터 자동 만료
- 성과: 월 140만 → 12만 원 (91% 절감) / 데이터 유실 없음
- 사용 기술: AWS DocumentDB, TTL Index

렌터카+호텔 패키지 — 보상 트랜잭션

- 문제: 렌터카 예약할 때 호텔도 같이 팔자는 사업 요구였습니다. 기존에 렌터카 예약과 결제가 하나의 DB 트랜잭션에 묶여 있어서 호텔 도메인을 끼울 수 없었습니다.
- 해결: 사가 패턴은 두 단계짜리 흐름에 오케스트레이터까지 만드는 건 과하다고 판단해서 보상 트랜잭션 선택
- 성과: 패키지 상품 출시 / 호텔 매출 내 패키지 비중 60%
- 사용 기술: NestJS, MySQL, REST Webhook, DLQ

지점 마스터 시스템

- 문제: 6개 공급사의 지점 데이터가 전부 달랐습니다. 같은 인천공항인데 "Incheon Airport", "ICN Int'l", "인천공항 T1"으로 제각각 들어오니 통합 검색이나 가격 비교가 안 됐습니다.
- 해결: 마스터 지점 테이블 + 공급사별 매핑 테이블 구조로 6개 공급사 데이터를 정규화. 신규 공급사는 매핑만 추가하면 됨
- 성과: 타깃 지역 일 평균 예약 200 → 400건
- 사용 기술: NestJS, MySQL, OpenAI Batch API

쿠폰 도메인 분리

- 문제: 쿠폰 로직이 렌터카 예약 서비스 코드에 직접 박혀 있었습니다. 호텔이나 패키지에 쿠폰을 적용하려면 같은 코드를 복사해야 했고, 발급은 개발팀이 직접 PHP 스크립트를 통해 진행하여 수기 오류가 발생할 수 있는 구조 였습니다.
- 해결: 쿠폰을 독립 도메인으로 분리해서 전 상품군에서 동일하게 사용할 수 있게 만들었음
- 성과: 발급 소요 30분 → 1분 / 수기 처리 월 5건 → 0건
- 사용 기술: NestJS, MySQL, Event-driven

파트 리드 (2025.01 ~)

- 문제: 운영 과정에서 반복되는 병목/장애가 발생
- 해결: PR이 3~4일씩 방치되는 문제를 해결하기 위해 코드 리뷰 SLA 도입 (긴급 D-1 / 일반 D-2 / 리팩토링 D-3)
- 성과: 장애 대응 시간 단축 및 운영 자동화 달성
- 사용 기술: NestJS, TypeScript

기술 역량

- Backend: Node.js, NestJS, TypeScript
- Database: MySQL, MongoDB, Redis, PostgreSQL
- Infrastructure: AWS, GCP

오픈소스 기여

Claw-Empire — AI 에이전트 오케스트레이션 플랫폼

- 주요 기여: #23 미팅 프롬프트 토큰 절감 / #26 안전 자동 업데이트 시스템 (Merged)
- 참고: GreenSheep01201/claw-empire (https://github.com/GreenSheep01201/claw-empire) (★106) — AI 코딩 에이전트를 가상 회사처럼 오케스트레이션하는 프로젝트. 직접 쓰다가 불편한 부분을 고쳐서 PR을 올렸습니다. 미팅이 길어지면 전체 task description과 과거 transcript를 프롬프트에 그대로 넣어서 토큰이 계속 늘어나는 문제가 있었습니다. Task context에 bounded head/tail compaction을 적용하고, transcript는 최근 N턴만 유지하면서 중복 발언을 제거했습니다. budget 값은 환경변수로 조절 가능하게 했고, compaction 로직은 meeting-prompt-utils.ts로 분리해서 테스트를 붙였습니다. 로컬 퍼스트 앱이라 자동 업데이트 시 작업 코드가 날아갈 위험이 있었습니다. "안전하지 않으면 건너뛴다"는 원칙으로 Safety gates 4개(dirty tree / non-main branch / busy server / channel policy)를 설계하고, 환경변수 기반 opt-in(기본 off)으로 구현했습니다. POST /api/update-apply(dry_run/force)와 Status API를 추가했고, v1.1.4 릴리즈에 반영됐습니다.

학력

- 코드캠프 Node.js 백엔드 부트캠프 (2022.03 - 2022.07): 성적 우수자 기업협업: 비대면 강의 시스템 어드민 서버 개발