

권동하

Backend Engineer / 경력기술서

ansqhr13037@gmail.com github.com/ssyy3034 ssyy3034.dev

주어진 요구를 구현하는 데서 멈추지 않고, 운영 중인 서비스의 개선점을 먼저 찾아 실행하는 백엔드 개발자입니다.

현재 업무

CRM 청구·인력 관리와 제품 이관

2026.06 - 현재 · 담당 기능 구현·검증 · 회사 업무 · 팀 협업

회사에서 청구·인력 관리 기능의 구현과 검증을 맡고 있습니다. 팀이 제시한 청구 스케줄과 인력 계획 요구를 업무 화면·상태 전이·저장 조건으로 구체화했습니다. 발행 상태, 배정 기간, 역할, 공수처럼 서로 연결된 규칙을 다룹니다.

화면의 읽기 전용 처리에만 기대지 않고, 생성·월 이동·수정에서 같은 서버 규칙을 적용하도록 정리했습니다. 청구 저장 경로를 일원화하고, 종료한 계획의 이력을 보존하며 중복 호출을 구분했습니다. 회회 허용 조건을 검토할 때도 편리한 기본값보다 확인 가능한 권한을 기준으로 판단했습니다.

제품 이관에서는 데이터가 옮겨졌다는 사실과 사용자가 실행·저장할 수 있다는 사실을 구분했습니다. DB 정의부터 API, 화면, 실행, 저장 후 재조회까지 단계별로 확인하고 기존 업무 데이터의 보존도 점검했습니다. 구현에는 AI 도구를 활용했고, 제 기여는 요구를 서버 조건으로 정리한 판단과 담당 경로의 검증입니다.

청구·인력 관리의 규칙을 서버에 옮기기

청구 발행 상태와 인력 배정의 기간·역할·공수는 서로 연결된 규칙입니다. 화면에서 입력을 막아도 생성·월 이동·수정 경로마다 저장 로직이 달라, 같은 데이터가 경로에 따라 다른 규칙으로 저장될 수 있었습니다. 청구에는 별도 쓰기 경로와 폴백 경로까지 있었습니다.

화면의 읽기 전용 처리만으로는 경로가 늘 때마다 규칙이 달라질 수 있었습니다. **그래서 화면은 안내만 하고, 최종 판단은 서버 저장 경로 한 곳에서 하도록 바꿨습니다.** 인력 생성·월 이동·공수 수정에 같은 조건을 적용하고 청구의 별도 쓰기·폴백 경로는 없었습니다. 종료한 계획은 삭제하는 대신 완료 상태로 남겨 이력을 보존했고, 이미 종료된 계획의 재요청은 따로 구분했습니다.

어느 경로로 저장해도 같은 규칙을 따르고, 종료한 계획은 이력으로 남습니다. 변경은 담당 코드와 업무 기록으로 확인했습니다.

MariaDB에서 통과하던 패치가 PostgreSQL에서 막히는 네 가지

제품을 MariaDB에서 PostgreSQL로 옮기면서 구버전에서 만든 패치를 적용하자 INSERT와 테이블 생성이 연이어 거부됐습니다. MariaDB가 관대하게 받아 주던 값을 PostgreSQL은 엄격하게 거부했고, 원인이 제각각이었습니다.

거부된 사례를 네 유형으로 나눠 유형마다 기준을 정했습니다. 멀티테넌시 키가 빠진 NOT NULL 위반은 공통 데이터와 테넌트 데이터를 구분해 값을 채웠고, MEDIUMTEXT·VARCHAR2·CLOB 같은 전용 타입은 PostgreSQL 타입으로 바꿨습니다. 숫자 컬럼의 빈 문자열은 일괄 NULL로 바꾸면 원래 0이던 데이터의 합계·평균·= 0 조건 결과가 달라져, 원래 0이던 값은 0으로, 미입력은 NULL로 나눴습니다. 날짜 함수는 PostgreSQL 전용 문법으로 우회하면 다른 DB에서 다시 깨지므로, **전용 함수 대신 플랫폼 표준 날짜 함수와 CAST를 유지했습니다.**

보정한 패치를 다시 적용해 정상 반응을 확인했습니다. 네 유형과 보정 방법을 확인 순서로 정리해 사내에 공유했고, 이후 다른 팀원의 이관 기록에서 이 정리를 참조했습니다.

상세 기록: ssyy3034.dev/projects/business-systems

사내 지식 검색 MCP 서버

2026.06 - 현재 · 구축·검색 평가·운영 · 개인 구축 · 동료 7명 사용

사내 용어를 잘못 이해하는 개발 도구를 보며 자료 수집·검색·MCP 서버를 구축했고, 일부 동료의 개발 도구에 연결했습니다. AI 보조 구현을 포함하며, 서버의 책임과 검증 기준을 정한 판단을 중심으로 씁니다.

답변을 만드는 서버에서 근거를 반환하는 서버로

처음에는 E-GENE 화면에 붙는 챗봇으로 만들어, 서버가 검색 결과를 LLM에 넘겨 답변까지 생성했습니다. 요청마다 LLM을 불러야 해서 개인 비용으로는 저가 모델을 쓸 수밖에 없었고, 사용자가 늘수록 비용도 늘었습니다. 서버 답변의 정답 일치 점수는 검색과 프롬프트를 고쳐도 0.5 부근에 머물렀고, 근거를 무시하고 없는 컬럼명으로 SQL을 만드는 경우도 있었습니다.

더 좋은 모델로 바꾸면 품질은 오르지만 비용이 사용자 수에 비례해 늘었습니다. 그런데 팀원들은 이미 Claude Code·Codex 같은 상용 코딩 에이전트로 개발하고 있었고, 같은 근거를 저가 모델과 상용 에이전트에 주고 블라인드로 비교하자 상용 에이전트가 분명히 앞섰습니다. 그래서 답변은 각자의 에이전트에 맡기고, 서버는 검색과 근거 반환만 하는 **MCP 서버로 역할을 좁혔습니다**. 답변 품질을 서버가 직접 통제할 수 없게 된 대신 근거마다 출처 링크를 함께 돌려줘 원문을 확인할 수 있게 했습니다. 상위 20건을 LLM이 다시 정렬하는 리랭커도 시험했지만 recall@5가 0.770에서 0.516으로 떨어져 뺏습니다.

서버의 LLM 호출 비용이 0원이 됐고, 답변은 동료들이 쓰던 에이전트의 작업 맥락 안에서 근거와 출처를 받아 만들어집니다.

호출 횟수보다 연결 수명을 추적하기

하루 백여 건, 건당 약 0.5초면 CPU 사용은 하루 1분 미만이어야 했는데, 청구 내역에는 주 약 319K CPU초가 찍혀 있었습니다. 요청이 없을 때 인스턴스가 0으로 줄어드는 Cloud Run을 고른 이유가 사라지는 숫자였습니다.

요청 수가 아니라 연결 수명을 의심하고 Cloud Run 지표와 요청 기록을 맞춰 보니, Claude Code 등 에이전트 클라이언트가 도구 목록 변경 같은 서버 알림을 받으려고 여는 GET 스트림(SSE)이, 실제 검색(POST)과 별개로 약 5분마다 재연결하며 하루 종일 열려 있었습니다. Cloud Run은 열린 스트림을 처리 중인 요청으로 보기 때문에, 검색이 없어도 인스턴스가 내려가지 못하고 과금이 이어졌습니다. 최소 인스턴스는 이미 0이었고, 요청 타임아웃을 줄여도 클라이언트가 끊기자마자 다시 연결해 열린 연결이 늘 남았습니다. MCP 사양을 보니 이 스트림은 선택 기능이었고 제 서버는 보낼 알림이 없어, **단독 GET 요청은 405로 거부하고 POST 도구 호출은 그대로 뒀습니다**.

2026년 7월 실측에서 유틸 스트림은 Cloud Run CPU 과금의 82%를 차지했고, 차단 후 연결당 점유가 약 301초에서 3ms로 줄었습니다. 이후 사용량은 실제 검색 처리 시간에 비례해, 사용자가 늘어난 지금도 서버 비용은 월 몇천 원에서 몇만 원 사이입니다.

검색 평가: Recall·MRR 회귀셋과 RAGAS

검색 엔진을 새 구성으로 바꾸자 p95 응답 시간은 2,972ms에서 642ms로 줄었지만, 순위 지표 MRR@10은 0.851에서 0.721로 떨어졌습니다. 처음에는 기존 엔진 결과로 만든 정답지가 새 엔진에 불리하다고 설명했지만, 검증하지 않은 가설이었습니다.

속도만 보고 바꾸거나 지표만 보고 되돌리는 대신, 무엇을 재고 있는지부터 나눴습니다. 검색은 Recall·Hit·MRR로, 답변은 RAGAS로 따로 재고, 사람이 정답 문서를 라벨링한 골든셋(25문항)과 변경 감지용 회귀셋(149문항)의 역할을 나눴습니다. 정답지를 보완해 다시 채점하자 격차는 오히려 -0.130에서 -0.159로 벌어져($p=0.002$) 가설이 틀렸음을 확인하고 기록을 정정했습니다. 순위 지표는 정답이 몇 번째에 오는지만 재기 때문에, 결과 묶음의 쓸모는 두 엔진의 결과를 가린 채 LLM 심판에게 비교시켰습니다. 심판 판정의 22.8%가 내용 대신 위치를 따라, **순서를 뒤집어 두 번 판정하고 같은 쪽을 고른 경우만 승리로 셧습니다**.

블라인드 비교(38문항 × 3회)에서 새 엔진이 매회 18승, 기존 엔진이 6~9승이었고, 상위 10건 내 정답 포함률(hit@10)은 두 엔진 모두 0.96이었습니다. 에이전트는 상위 여러 건을 함께 읽으므로 1위 정확도보다 결과 묶음의 질과 응답 속도가 중요하다고 판단해, 1위 순위가 낮아지는 것과 메모리를 2GB로 올리는 비용을 감수하고 새 엔진으로 전환했습니다.

테스트 통과가 실제 코드의 회귀를 잡는지 확인하기

주간 워크플로의 검색 회귀 검사는 저장소에 커밋된 검색 결과 파일을 다시 채점할 뿐 실제 검색을 돌리지 않아, 랭킹 코드가 바뀌어도 숫자가 그대로였습니다. 감시 대상도 운영 중인 엔진이 아니라 이전 엔진이었습니다.

매번 실제 검색을 돌리면 임베딩 API와 벡터 DB를 불러야 해 자격 증명과 네트워크, 비용이 들었습니다. 그래서 **벡터 검색 후보는 파일로 고정하고, 그 뒤의 BM25-정확 일치-융합-감점 단계는 운영 엔진 코드를 그대로 실행하는 검사로 바꿨습니다.** LLM 호출 없이 약 4초에 끝나 PR마다 돌릴 수 있는 대신, 임베딩 모델과 벡터 검색 자체의 회귀는 잡지 못하는 것을 감수했습니다. 엔진을 일부러 망가뜨려 검사를 시험하자 홍보글 감점을 지운 변경은 지표가 그대로라 통과했고, **그래서 지표와 함께 문항별 상위 10건 순위를 기준값과 대조하게 했습니다.**

BM25 정규화를 지운 변경(MRR@10 0.917 → 0.732)과 홍보글 감점을 지운 변경(순위가 바뀐 문항 31개)이 모두 실패하고, 되돌리면 통과합니다. 의도한 순위 변경은 기준값 파일의 문항별 diff를 확인한 뒤 반영합니다.

상세 기록: ssyy3034.dev/projects/knowledge-mcp

영업 업무 멀티 에이전트

2026.09 · 에이전트 구성·실행 검증 · 회사 업무 · 팀 협업

영업 담당자의 미팅 준비, 조건 대응, 초안 작성, 활동 기록을 처리하는 업무 에이전트를 자사 솔루션의 에이전트 런타임에서 구성했습니다. 팀 협업 결과이며 역할·도구 구성과 실행 결과 검증에 참여했습니다. 구현에는 AI 보조를 활용했습니다.

에이전트별 도구 범위와 승인 후 쓰기

영업 담당자의 미팅 준비부터 활동 기록까지 처리하는 업무 에이전트를 팀으로 만들었습니다. 하나의 에이전트에 조회와 쓰기 도구를 모두 주면 단순 조회 요청에서도 쓰기 도구가 호출될 수 있었고, '저장했다'는 답만으로는 실제 저장 여부를 알 수 없었습니다.

프롬프트로 쓰기를 자제시키면 결국 모델의 판단에 기대야 했습니다. 그래서 **오케스트레이터가 분류·준비·코칭·초안·기록·검증 6개 역할 에이전트에 위임하고, 역할마다 쓸 수 있는 도구 자체를 나눴습니다.** 유사 사례 비교는 자유 텍스트 필터 대신 대상 ID에서 SQL 조건을 도출하게 했고, 기록 도구는 저장할 내용을 먼저 보여 주고 사용자가 승인해야 실행됩니다. 단계가 하나 늘어나는 대신 사람이 확인한 내용만 저장됩니다.

조회·초안·저장 경로가 분리됐고, 답변에는 내부 근거와 공개 자료가 구분돼 표시됩니다. 저장 여부는 DB 행 수로 판정합니다.

상세 기록: ssyy3034.dev/projects/sales-agents

세일즈 네비게이터

2026.09 - 현재 · 네이티브 이관 설계·구현·운영 배포 · 회사 업무 · 솔루션 내재화 담당

외근 영업 담당자가 휴대폰에서 오늘 일정 브리핑, 장소 검색과 길찾기, 음성 일정 초안, 미팅 녹음과 회의록, 딜 준비를 처리하는 CRM 모듈입니다. 원래는 다른 구성원이 만든 사내 해커톤 1위 작품(Express.js 브릿지 서버와 브라우저 앱)이었고, 서비스 도메인이 CRM과 맞아 제품에 이식하는 업무를 맡았습니다.

이관 전에 원본을 분석해 같은 검색 로직이 서버·브라우저·플랫폼 규칙에 3벌로 있고, 사용자 식별과 LLM 키가 브라우저 저장소에 있어 플랫폼의 인증·테넌트 밖에서 동작한다는 점을 확인했습니다. 플랫폼 세션을 인증 주체로 삼는 단일 서버 경로로 옮긴 뒤 녹음·회의록·딜 준비 기능을 더했고, 운영 배포까지 맡았습니다. 모듈 커밋 157개 중 155개를 작성했고, 구현에는 AI 보조를 활용했습니다.

해커톤 구조를 플랫폼 안으로 옮기기

동료의 해커톤 1위 작품은 Express 브릿지 서버, 브라우저 앱, 플랫폼 규칙 6개로 나뉘 7,770줄이었고, 플랫폼 안에서 도는 것은 규칙 6개뿐이었습니다. 같은 검색 로직이 서버·브라우저·규칙에 3벌로 있어 프롬프트 하나를 고치려면 세 곳을 고쳐야 했습니다. 사용자 식별은 브라우저가 만든 UUID와 Google 로그인뿐이었고 LLM 키를 포함한 설정 28개가 브라우저 저장소에 있어, 영업팀 계정과 권한으로는 쓸 수 없는 구조였습니다.

분기 로직을 둘 곳은 두 가지였습니다. 원본처럼 플랫폼 규칙 엔진의 노드로 옮기면 플랫폼 기능만으로 끝나지만, 조건 노드는 단일 비교만 지원하고 에이전트 노드는 도구 호출을 지원하지 않았습니다. 원본도 자연어 검색 분기를 노드로 옮기다 30개 노드짜리 플로우가 된 기록이 있었습니다. 그래서 **분기는 서버 코드에 두고 규칙은 호출 진입점으로만 썼습니다.** 공급자 키는 브라우저 대신 플랫폼의 기존 AI Portal 키 저장소에서 읽게 했습니다.

3벌이던 검색 로직은 서버 경로 하나로 합쳐졌고, 모든 외부 API·LLM 호출이 플랫폼 인증을 거치게 되어 영업팀 계정으로 쓰는 운영 배포까지 진행했습니다.

LLM을 기다리는 동안 바뀐 녹음을 덮어쓰지 않기

회의록 요약은 녹음 전사본 전체를 LLM에 보내고 결과를 저장합니다. LLM 응답은 수십 초 걸릴 수 있어 트랜잭션 안에서 기다리면 그동안 DB 커넥션을 붙잡고, 그사이 녹음이나 전사본이 바뀌면 오래된 입력으로 만든 요약이 최신 데이터를 덮어쓸 수 있었습니다. 원본은 LLM 결과 중 맞지 않는 부분을 조용히 버리고 나머지를 썼습니다.

트랜잭션 안에서 행을 잠근 채 기다리면 정합성은 쉽게 지켜지지만, 외부 대기가 DB 커넥션을 묶는 것은 StoLink에서 전체 API를 멈추게 했던 원인과 같았습니다. 그래서 **LLM 호출은 트랜잭션 밖에서 하고, 저장할 때만 짧은 트랜잭션을 열어 녹음을 다시 읽고 호출 전과 비교했습니다.** 다르면 덮어쓰지 않고 409로 알려, 사용자가 다시 요청해야 하는 수고를 감수하는 대신 최신 데이터를 지켰습니다. LLM 응답은 종로 사유·형식·크기를 검사하고, 장소 재정렬은 원래 후보의 완전한 순열일 때만, 요약은 원문 구간과 정확히 일치하는 발췌일 때만 저장합니다.

외부 호출을 기다리는 동안 DB 커넥션을 붙잡지 않게 됐고, 그사이 바뀐 녹음은 덮어쓰지 않고 충돌로 드러납니다. 요약에는 원문에 없는 문장이 저장되지 않습니다.

상세 기록: ssyy3034.dev/projects/field-assistant

이전 프로젝트

자연어 업무 시스템 생성 에이전트

2026.06 - 2026.08 · 설계·구현·검증 · 개인 프로젝트

자사 솔루션은 엔티티·폼·리스트·워크플로우·메뉴 같은 메타데이터로 업무 시스템을 구성합니다. 반복되는 CRUD·승인 업무 시스템을 요구사항 문서나 자연어 한 줄로 만들 수 있도록, 의도를 명세로 바꾸는 단계와 플랫폼 객체를 생성하는 단계를 나눈 에이전트를 개인 프로젝트로 설계하고 구현했습니다. 실사용자는 없고, 검증용 인스턴스에 빌드해 동작을 확인했습니다. 구현에는 AI 보조를 썼습니다.

빈 DB인데 모든 객체가 '이미 있다'고 나온 이유

자연어 명세로 업무 시스템을 만드는 에이전트를 검증하려고 빈 DB와 WAS를 따로 세워 반영 계획을 돌렸더니, 생성되어야 할 20건이 전부 '수정'으로 나왔습니다. 그대로 반영하면 없는 객체를 고치려 하거나, 빌드 중 일부를 조용히 건너뛸 수 있었습니다.

계획 로직보다 응답 처리부터 따라갔습니다. 세션이 없으면 플랫폼의 `/api/` 경로는 HTTP 200과 HTML 로그인 화면을 돌려주는데, 파서는 이를 문자열로 넘겼고 존재 확인은 '비어 있지 않으면 있음'으로 판정했습니다. 같은 확인을 반영 엔진과 빌더도 쓰고 있어 존재 확인 함수만 고치면 다른 곳에 같은 구멍이 남았습니다. 그래서 **응답 파서에서 `/api/` 경로가 로그인 화면을 주면 인증 오류로 터뜨리고, 존재 확인이 인증 오류를 '존재함'으로 읽지 않게 했습니다.** 계획만 세우는 모드도 인증을 거치게 했습니다.

다시 돌린 계획은 '생성 20, 수정 0'으로 빈 DB와 일치했고, 인증 확인 테스트 5케이스를 추가했습니다.

'미리보기'가 운영에 적용되지 않게 막기

플랫폼 코어를 바꾸지 않고 미리보기를 제공하려면 격리 인스턴스에 실제로 적용했다가 되돌려야 했습니다. 이 기능이 '미리보기'라는 이름으로 운영 서버에 쓰이는 것이 가장 큰 위험이었습니다.

안전장치를 기능 본체로 다뤘습니다. 허용 대상 목록에 없는 서버면 아무것도 하지 않고 거부하며, 목록의 기본값은 비워 두었습니다. 미리보기 URL은 HMAC 서명과 만료 시간을 갖고, 만료 검사는 서명 검사 뒤에 합니다. 서명이 틀린 토큰의 만료 시각을 믿으면 만료 검사를 우회당하기 때문이며, 이 순서는 테스트로 고정했습니다. 모든 상태 변화는 감사 로그에 남깁니다.

검증 환경에서 미리보기 시작 시 2건이 생성되고 종료 후 0건으로 돌아오는 것을 확인했습니다. 미리보기 테스트 13케이스를 추가했습니다(전체 449개 통과).

상세 기록: ssyy3034.dev/projects/agentic-modeler

StoLink

2025.12 - 2026.02 (후속 개선 2026.03) · 장애 분석·결제 구현 · 5인 팀 · 이전 프로젝트

크래프톤 정글 5인 팀 프로젝트에서 백엔드와 프론트엔드 구현을 맡았습니다. 여기에는 장애 분석과 결제 경계 설계만 남겼습니다.

쓰레드 덤프에서 커넥션 점유까지 추적한 장애

전체 API가 간헐적으로 응답하지 않았지만 CPU-메모리는 정상이었습니다. 자원 부족으로는 설명되지 않는 장애였습니다.

서버를 키우는 대신 멈춘 순간의 쓰레드 덤프를 뚫습니다. 외부 헬스체크가 Socket Read에서 기다리고 있었고, REQUIRES_NEW 트랜잭션이 DB 커넥션을 쥔 채 그 응답을 기다리는 흐름이었습니다. 타임아웃 설정이 있는데도 무한 대기한 이유는 @Value 필드 주입이 생성자 이후에 이뤄져 초기화 시점에 값이 비어 있었기 때문이었습니다. **타임아웃과 트랜잭션 경계를 따로 고쳤습니다.** 생성자 주입으로 바뀌 외부 응답 대기를 5초로 제한하고, 결제는 검증 → 트랜잭션 밖 PG 호출 → 결과 반영으로 나눠 외부 대기 동안 커넥션을 잡지 않게 했습니다. 타임아웃만으로는 5초 동안의 점유가 남기 때문입니다.

외부 연결 차단과 PG 지연을 주입한 검증에서도 일반 API가 정상 응답하는 것을 확인했습니다.

동시 차감과 중복 요청을 다른 실패로 다루기

크레딧 결제에서 같은 잔액을 동시에 읽고 저장하면 한쪽 차감이 사라질 수 있고, 같은 요청이 재시도되면 두 번 차감될 수 있었습니다. AI 코드 리뷰가 제기한 위험을 먼저 테스트로 재현해 실제 문제임을 확인했습니다.

락은 동시 차감을 막지만 같은 요청의 재시도는 막지 못해, 두 문제를 따로 다뤘습니다. 잔액 경합에는 낙관적 락도 검토했지만, 같은 잔액에 요청이 몰리면 충돌한 요청들의 재시도도가 더 큰 부하가 된다고 판단했습니다. **그래서 대기 지연을 감수하고 잔액 행에는 비관적 락을, 같은 요청의 재처리에는 멍등키와 DB 제약을 적용했고,** 락을 쥐는 시간이 길어지지 않게 PG 호출은 트랜잭션 밖으로 뺐습니다. 락과 제약의 동작은 인메모리 DB로는 확인할 수 없어, Testcontainers로 실제 PostgreSQL을 띄워 검증했습니다.

100개 스레드가 동시에 차감해도 잔액이 맞았고, 잔액이 절반뿐이면 정확히 50건만 성공해 음수가 되지 않았습니다.

동시성 개선-검증 PR #80

https://github.com/stolink/storead_spring/pull/80

중복 결제 예외 경계 PR #87

https://github.com/stolink/storead_spring/pull/87

노드 50개에서 느려지던 관계도를 Canvas로 옮기기

핵심 기능인 인물 관계도는 D3.js SVG로 그렸는데, 노드가 50개만 넘어도 클릭·드래그 응답이 420ms까지 느려지고 FPS가 10 아래로 떨어졌습니다. 인물이 많은 소설일수록 쓸 수 없는 기능이 되는 상황이었습니다.

처음 맡은 프론트엔드라 원인부터 몰랐는데, DevTools Performance로 보니 병목은 스타일 재계산이었습니다. SVG는 노드마다 DOM 요소가 생겨 프레임마다 수백 개 요소의 스타일 재계산과 리플로우가 반복되는 구조였습니다. **DOM 요소 수에 비례하는 비용을 없애려고 Canvas로 전면 전환했습니다.** 대신 DOM 이벤트를 쓸 수 없어 클릭과 드래그는 좌표 기반 히트 테스트로 직접 구현했고, React의 불변 데이터와 물리 엔진의 가변 좌표가 충돌하는 문제는 물리 연산용 복사본과 useRef로 분리했습니다. 캐릭터 이미지는 Offscreen Canvas에 비트맵으로 캐싱했습니다.

INP는 420ms에서 64ms로, FPS는 10 미만에서 60으로 올랐고, 노드 650개 이상에서도 끊김 없이 동작했습니다.

상세 기록: ssyy3034.dev/projects/stolink

Aidiary

2025.04 - 2025.06 (개인 개선 2026.02) · 백엔드·인프라 개선 · 2인 팀 · 이전 프로젝트

2인 캡스톤 프로젝트 이후 개인적으로 백엔드와 인프라를 개선했습니다.

JVM 안의 비동기 작업을 메시지 큐로 옮기기

일기로 아기 캐릭터를 만드는 AI 합성은 한 건에 약 30초가 걸렸고, 그동안 웹 서버 자원을 붙잡았습니다.

먼저 @Async와 JVM 내부 상태 저장으로 요청을 빨리 반환해 봤지만, 부하가 물리면 자원 관리가 어렵고 재배포하면 진행 중인 작업이 사라졌으며 인스턴스끼리 상태를 공유할 수 없었습니다. **작업의 수명을 웹 서버와 분리하려고 RabbitMQ를 택했습니다.** 운영할 구성 요소가 하나 늘어나는 대신, WAS는 접수 응답만 반환하고 Python Worker가 수동 ACK로 작업을 처리합니다. 재전달에 대비해 멍등성 가드와 실패 메시지 경로를 뒀습니다.

외부 AI 서버를 내린 상태에서도 코어 API가 응답하는 것을 확인했습니다.

개인 프로젝트 저장소

<https://github.com/ssyy3034/aidiary-main>

상세 기록: ssyy3034.dev/projects/aidiary

사례에 사용한 기술

Java · Python · JavaScript / Spring Boot · React / PostgreSQL / RabbitMQ / AWS · GCP / MCP · RAGAS

회사 업무는 공개 가능한 범위로 정리했습니다. 담당 기간과 검증 범위는 각 사례에 표시했으며, 구현에는 AI 보조를 활용했습니다.