PROJECT ARCHIVE / 2026

# 이혜연
# 포트폴리오

아이디어를 플레이 가능한 경험으로 만듭니다

| GAME DEVELOPMENT | GENERAL DEVELOPMENT |
|---|---|


서울여자대학교 소프트웨어융합학과 재학 중 (2023.03 -)
디지털미디어학과 부전공 (2025 -)
크래프톤 정글 SW-AI Lab 12기 (2026.03 - 07)

포트폴리오 사이트 바로가기 ↗
hyeyeon030206@gmail.com / github.com/HyeYeonLee0112 ↗


---

CONTENTS

01 / GAME DEVELOPMENT

# EXIPAND PROJECT

연습생을 선택해 훈련과 리듬 경연을 거치며 최종 데뷔에 도전하는 아이돌 서바이벌 게임


| | |
|---|---|
| 기간 | 2025.05.02 - 2025.12.09 |
| 인원 | 5명 · 개발 3명, 아트 2명 |
| 담당 | 게임 핵심 프레임워크 및 진행 루틴 · 리듬 게임·판정·결과 시스템 · JSON 데이터·육성·스케줄 시스템 |
| 기술 | Unity 2022.3.60f1 · C# · URP · UGUI · TextMeshPro · JSON · DOTween |
| 링크 | GitHub ↗ / 전시 페이지 ↗ |

| OVERVIEW | |
|---|---|
| | 6명의 연습생 중 한 명을 선택해 훈련과 스토리, 리듬 경연을 진행하고 최종 데뷔조에 도전하는 아이돌 서바이벌 육성 게임입니다.<br><br>요일별 스케줄로 능력치를 성장시키고 곡의 콘셉트와 연습생의 능력치를 조합해 경연 점수를 만듭니다. |


01 연습생의 상태와 활동을 확인하는 메인 화면


02 능력치 성장을 계획하는 주간 스케줄


03 데이터에 따라 전개되는 캐릭터 스토리

01 / PROJECT DETAIL

# EXIPAND PROJECT

사용자가 경험하는 기능과 제가 구현한 방식을 실제 사용 상황과 함께 설명합니다.

## 01 주요 기능

01 주간 육성 스케줄
월요일부터 금요일까지 보컬, 댄스 등 훈련 일정을 선택합니다. 활동 결과가 6개 능력치에 반영되어 이후 경연과 순위에 영향을 줍니다.

02 데이터 기반 분기 스토리
공통·캐릭터·탈락·데뷔 이야기를 JSON 데이터로 관리합니다. 플레이 상태에 맞는 대사와 장면을 불러와 선택한 연습생의 흐름을 이어갑니다.

03 4키 리듬 경연
D, F, J, K 네 키로 노트를 판정합니다. Perfect, Good, Bad, Miss 판정과 콤보, 곡 점수를 계산하고 육성 능력치와 합산해 최종 베네핏을 결정합니다.

04 순위와 탈락
경연 점수에 따라 66명의 순위를 갱신합니다. 라운드별 생존 인원이 줄어들고 마지막 6명이 데뷔하는 서바이벌 흐름을 구현했습니다.

## 02 구현 기능

01 중앙 진행 루틴
GameManagerEx가 날짜, 훈련, 스토리, 경연, 순위 발표를 하나의 진행 루틴으로 연결합니다. 각 단계가 끝나면 다음 상태로 넘어가도록 구성해 전체 게임 흐름을 한곳에서 추적할 수 있게 했습니다.

02 공통 매니저 프레임워크
Managers를 진입점으로 Game, Data, UI, Resource, Story, Rank, Sound, Input 기능을 초기화했습니다. 여러 화면에서 필요한 기능에 같은 방식으로 접근하도록 프로젝트의 기본 구조를 만들었습니다.

03 데이터 기반 콘텐츠 처리
연습생 정보와 대사, 곡, 순위 데이터를 코드에서 분리해 JSON으로 읽습니다. 콘텐츠가 변경되어도 핵심 로직을 직접 수정하는 범위를 줄였고, Google Sheet 데이터를 게임용 데이터로 변환하는 작업 흐름도 구성했습니다.

04 리듬 노트 제작과 판정
노트 생성, 이동, 입력 판정, 콤보, 점수 계산과 결과 화면을 구현했습니다. Unity Editor에서 곡의 타이밍에 맞춰 노트를 기록하고 저장할 수 있는 제작 도구를 추가해 반복 입력 작업을 줄였습니다.

## 03 실제 작동 예시

| 상황 | 예시: 월요일에 보컬 훈련을 선택한 경우 |
|---|---|
| 01 행동 선택 | 플레이어가 스케줄 화면에서 월요일 활동으로 보컬 훈련을 선택합니다. |
| 02 정보 확인 | 게임은 선택한 연습생, 현재 요일, 선택한 훈련 종류를 함께 확인합니다. |
| 03 결과 계산 | 보컬 훈련 규칙에 따라 능력치 변화를 계산하고 연습생의 현재 상태에 반영합니다. |
| 04 다음 콘텐츠 연결 | 변경된 능력치는 이후 스토리 조건과 경연 점수를 계산할 때 다시 사용됩니다. |
| 05 화면 반영 | 훈련 결과와 오른 능력치를 보여준 뒤 날짜를 진행하고 다음 일정을 선택할 수 있게 합니다. |

01 / CASE STUDY & RETROSPECTIVE

# EXIPAND PROJECT

| | |
|---|---|
| **PROJECT OUTCOME** | ■ 6명의 연습생 선택과 육성, 스토리, 경연, 순위 발표가 이어지는 전체 플레이 흐름 완성<br>■ 4키 리듬 게임과 Unity Editor용 노트 제작 도구 구현<br>■ 66명 참가자의 순위와 라운드별 탈락, 최종 6명 데뷔 로직 구현 |

## 대표 문제 해결

실제 작업 중 마주친 상황과 해결 과정을 한 사례로 정리했습니다.

| CASE 01 | **화면 전환 때 UI가 중복 생성되는 문제** |
|---|---|
| **상황** | 진행 단계가 바뀔 때 홈 또는 리듬 게임 UI가 두 번 생성되고 이전 팝업이 남는 경우가 있었습니다. |
| **원인** | 씬 전환 코드와 게임 진행 루틴이 각각 같은 UI 생성을 요청해 화면 생성 책임이 겹쳤습니다. |
| **도입한 방법** | UI 생성은 씬 초기화 과정 한곳에서 담당하도록 중복 호출을 제거했습니다. 다음 루틴을 실행하기 전에 씬 UI와 팝업 UI를 함께 정리하도록 종료 순서도 통일했습니다. |
| **해결 결과** | 전환 뒤 화면이 중복되거나 이전 팝업이 남는 현상을 없애고, 단계마다 같은 초기 상태에서 시작하도록 만들었습니다. |

## 회고

프로젝트에서 얻은 판단과 다음 작업에서 바꿀 점을 구분했습니다.

| | |
|---|---|
| **잘된 점** | ■ 게임 진행에 필요한 공통 기능을 먼저 구성해 팀원이 같은 흐름 위에서 기능을 나누어 개발할 수 있었습니다.<br>■ 스토리와 캐릭터 정보를 데이터로 분리해 콘텐츠 추가와 수정이 핵심 코드에 미치는 영향을 줄였습니다. |
| **아쉬운 점** | ■ 여러 시스템이 공통 매니저와 정적 이벤트에 직접 연결되어 기능이 늘어날수록 의존 관계를 추적하기 어려워졌습니다.<br>■ 일부 매니저가 게임 규칙과 화면 전환을 함께 담당해 클래스의 책임이 커졌습니다. |
| **다시 만든다면** | ■ 진행 규칙, 데이터, 화면 표현을 더 작은 단위로 나누고 인터페이스를 통해 연결하겠습니다.<br>■ 순위 계산과 경연 점수처럼 규칙이 명확한 로직에는 자동화 테스트를 추가하겠습니다. |

02 / GAME DEVELOPMENT

# 따끈따끈 붕어빵

손님의 주문을 맞추며 작은 붕어빵 가게를 5일 동안 운영하는 2D 타이쿤 게임

| | |
|---|---|
| 기간 | 2025.05.04 - 2025.06.22 |
| 인원 | 1명 · 개인 프로젝트 |
| 담당 | 게임 기획 및 전체 클라이언트 개발 · 조리·주문·손님 시스템 · UI·일일 정산·멀티 엔딩 |
| 기술 | Unity 6000.0.39f1 · C# · URP 2D · UGUI · TextMeshPro |
| 링크 | GitHub ↗ / 플레이 영상 ↗ |

| OVERVIEW | 손님의 주문을 확인하고 반죽과 속재료를 조합해 붕어빵을 구운 뒤 제한 시간 안에 제공하는 2D 가게 경영 게임입니다.<br><br>매일 오후 6시부터 11시까지 가게를 운영하고 매출에서 재료비를 뺀 순이익을 정산합니다. |
|---|---|

01 주문을 확인하며 붕어빵을 만드는 가게 운영 화면


02 목표 금액을 달성했을 때의 성공 엔딩


03 5일간의 운영 결과에 따라 달라지는 엔딩

02 / PROJECT DETAIL

# 따끈따끈 붕어빵

사용자가 경험하는 기능과 제가 구현한 방식을 실제 사용 상황과 함께 설명합니다.

## 01 주요 기능

### 01 단계별 붕어빵 조리

틀에 아래 반죽, 팥·슈크림·피자 중 선택한 속재료, 위 반죽을 순서대로 넣습니다. 알맞게 구운 붕어빵은 손님에게 전달하거나 진열대에 보관할 수 있고, 너무 오래 구우면 타버립니다.

### 02 손님 주문과 대기 시간

손님마다 붕어빵 종류와 수량이 무작위로 정해집니다. 기다리는 시간이 길어질수록 화남 수치가 올라가며, 주문을 모두 제공하기 전에 한계에 도달하면 손님이 떠납니다.

### 03 드래그 앤 드롭 상호작용

완성된 붕어빵을 마우스로 끌어 손님이나 진열대에 놓습니다. 주문 종류와 일치할 때만 판매가 처리되도록 조리 상태와 목적지를 함께 검사합니다.

### 04 5일 운영과 멀티 엔딩

영업 종료 후 판매 개수, 손님 수, 매출, 재료비와 순이익을 정산합니다. 5일째 보유 금액을 기준으로 세 가지 결말 중 하나를 보여줍니다.

## 02 구현 기능

### 01 조리 상태 머신

FishBunController에서 아래 반죽, 속재료, 위 반죽, 굽는 중, 완성 상태를 순서대로 관리합니다. 각 상태에서 가능한 클릭과 드래그 동작을 제한해 잘못된 조리 순서를 막았습니다.

### 02 주문 생성과 처리

CustomerController가 전체 주문 수를 속재료별 수량으로 나누어 저장합니다. 붕어빵을 전달할 때 주문과 조리 상태를 확인하고, 일치하면 남은 수량과 결제 금액, 판매 통계를 갱신합니다.

### 03 시간·날짜·경제 시스템

GameManagerEx가 영업 시간, 날짜, 보유 금액과 당일 판매 기록을 관리합니다. 영업 종료 시 매출과 재료비를 계산하고 정산 화면을 거쳐 다음 날 또는 엔딩으로 이동시킵니다.

### 04 공통 UI 프레임워크

UI 기본 클래스에서 버튼과 텍스트 같은 요소를 일관된 방식으로 연결합니다. 인트로, 상점, 게임, 정산, 엔딩 화면을 각각 분리해 게임 상태에 맞는 UI를 표시하도록 구성했습니다.

## 03 실제 작동 예시

| 상황 | 예시: 손님이 팥 붕어빵 두 개를 주문한 경우 |
|---|---|
| 01 주문 생성 | 손님이 등장하면 붕어빵 종류와 수량, 기다릴 수 있는 시간이 정해집니다. |
| 02 조리 | 플레이어가 틀에 반죽과 팥을 넣으면 붕어빵의 조리 상태가 굽는 중에서 완성으로 바뀝니다. |
| 03 주문 확인 | 완성된 붕어빵을 손님에게 놓을 때 주문 종류, 수량, 완성 상태가 모두 맞는지 검사합니다. |
| 04 판매 처리 | 주문이 맞으면 판매 금액과 사용한 재료비를 기록하고 손님을 다음 상태로 이동시킵니다. |
| 05 정산 | 영업이 끝나면 하루 매출에서 재료비를 뺀 순이익을 계산하고 누적 금액과 엔딩 조건에 반영합니다. |

02 / CASE STUDY & RETROSPECTIVE

# 따끈따끈 붕어빵

| | |
|---|---|
| **PROJECT OUTCOME** | ■ 기획부터 게임 로직과 UI까지 개인 프로젝트로 전체 제작<br>■ 조리, 주문, 판매, 정산이 이어지는 5일간의 완전한 플레이 흐름 구현<br>■ 운영 결과에 따른 파산, 일반, 성공의 3가지 엔딩 구현 |

## 대표 문제 해결

실제 작업 중 마주친 상황과 해결 과정을 한 사례로 정리했습니다.

| CASE 01 | **일일 정산 금액이 서로 맞지 않는 문제** |
|---|---|
| **상황** | 정산 화면의 매출, 재료비, 순이익과 다음 날에 반영되는 금액이 서로 다르게 계산됐습니다. |
| **원인** | 판매 금액과 재료비를 계산하고 차감하는 시점이 나뉘어 있었고, 화면에서도 별도 계산을 사용했습니다. |
| **도입한 방법** | 당일 매출, 재료비, 순이익을 각각 분리하고 영업 종료 처리에서 한 번만 계산하도록 모았습니다. 정산 UI는 계산된 값을 받아 표시하도록 변경했습니다. |
| **해결 결과** | 화면에 표시되는 정산 내역과 실제 보유 금액이 같은 계산 결과를 사용하도록 통일했습니다. |

## 회고

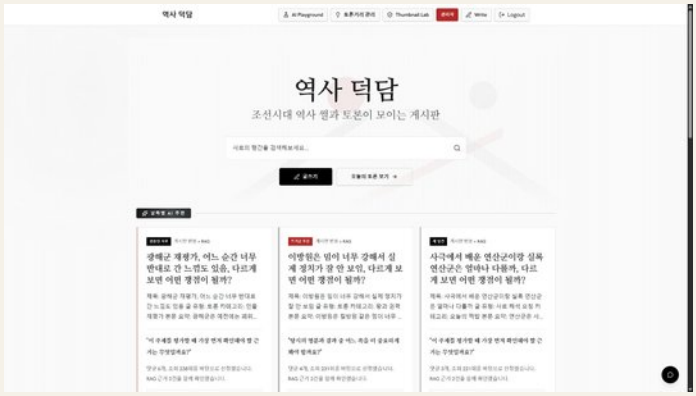
프로젝트에서 얻은 판단과 다음 작업에서 바꿀 점을 구분했습니다.

| | |
|---|---|
| **잘된 점** | ■ 붕어빵의 조리 단계를 상태로 나누어 복잡한 클릭과 드래그 조건을 명확하게 관리했습니다.<br>■ 하루 영업과 정산, 다음 날, 엔딩까지 게임의 시작과 끝을 혼자 완성했습니다. |
| **아쉬운 점** | ■ 게임 상태와 화면 전환이 공통 매니저에 집중되어 기능이 커질수록 수정 범위가 넓어질 수 있습니다.<br>■ 가격과 조리 시간 같은 설정값이 코드 여러 곳에 있어 밸런스를 반복해서 조정하기 어렵습니다. |
| **다시 만든다면** | ■ 가격, 조리 시간, 손님 성향을 ScriptableObject 데이터로 분리해 에디터에서 쉽게 조정하겠습니다.<br>■ 주문 생성과 일일 정산처럼 경계값 오류가 생기기 쉬운 규칙에는 자동화 테스트를 추가하겠습니다. |

03 / GENERAL DEVELOPMENT

# 역사덕담

역사 자료의 근거를 찾고 검증해 글쓰기에 반영하는 RAG·AI Agent 기반 커뮤니티




| | |
|---|---|
| 기간 | 2026.06.06 - 2026.06.18 |
| 인원 | 1명 · 개인 프로젝트 |
| 담당 | 게시판·인증 기반과 프론트/백엔드 연동 · 역사 자료 RAG 파이프라인 및 검색 품질 개선 · LangGraph Agent·MCP·Redis 캐시 구현 |
| 기술 | Next.js · TypeScript · FastAPI · Python · LangGraph · PostgreSQL · pgvector · Redis |
| 링크 | GitHub ↗ |

**OVERVIEW**

조선시대 인물과 사건을 주제로 글을 쓰고 토론하면서, AI가 답변에 사용한 역사 자료까지 함께 확인할 수 있는 웹 커뮤니티입니다.

사용자의 제목·본문·질문을 이해한 Agent가 내부 역사 자료를 검색하고 외부 자료로 근거를 보강합니다.


01 역사 글과 AI 추천 토론을 탐색하는 커뮤니티 홈


02 글쓰기 흐름 안에서 제목·본문·태그를 제안하는 에디터 Agent

03 채택한 내부 근거와 외부 자료를 분리해 보여주는 근거 패널

03 / PROJECT DETAIL

# 역사덕담

사용자가 경험하는 기능과 제가 구현한 방식을 실제 사용 상황과 함께 설명합니다.

## 01 주요 기능

01 근거 중심 에디터 Agent
게시글 작성 화면 안에서 질문 답변, 새 본문 작성, 기존 본문 수정을 요청할 수 있습니다. Agent가 현재 글의 맥락과 대화 기록을 함께 보고 제목·본문·태그·토론 질문을 제안하며, 사용자는 필요한 결과만 에디터에 적용합니다.

02 내부 RAG와 외부 자료 검색
조선왕조실록과 역사 개요 자료를 의미 검색하고, 내부 근거가 부족하면 네이버 검색과 실록 검색을 연결해 외부 후보를 찾습니다. 인물과 사건이 실제로 일치하는지 판정한 뒤 관련 없는 자료를 제외합니다.

03 역사 커뮤니티
회원가입과 로그인, 게시글·댓글 작성 및 수정·삭제, 카테고리·태그·검색·페이지네이션을 제공합니다. Markdown 작성과 미리보기를 지원하고 작성자 권한을 서버에서 확인합니다.

04 AI 확장 기능
로그인 사용자를 위한 전역 역사 챗봇, 최근 글과 사료를 조합한 오늘의 토론거리, 글의 내용을 반영한 썸네일 후보 생성, 관리자용 AI Playground를 구현했습니다.

## 02 구현 기능

01 LangGraph 기반 글쓰기 흐름
사용자 요청을 intent → retrieve → external_search → respond 단계로 나눴습니다. 질문 답변·본문 생성·본문 수정 의도를 분류하고 내부 RAG, 외부 자료, 근거 부족 상태를 하나의 구조화된 응답으로...

02 역사 자료 수집과 검색 파이프라인
수집한 사료를 Markdown으로 정규화하고 문서와 청크를 데이터베이스에 동기화한 뒤 임베딩을 생성했습니다. 저장소 기준 3,709건의 자료를 실록 원문, 역사 개요, 실록 v2 검증 자료로 나누고 질문 성격에 따라 검색 우선순위를...

03 MCP 외부 도구 연결
JSON-RPC 2.0 기반 MCP 서버에 실록 검색, 네이버 검색, 웹 검색, 썸네일 생성 도구를 등록했습니다. Agent가 도구를 호출하면 사용한 검색어와 결과 상태, 근거 후보를 함께 기록하도록 구성했습니다.

04 캐시와 실패 대체 경로
외부 검색과 게시글 목록에 Redis 캐시를 적용했습니다. 임베딩 결과가 없거나 API 키를 사용할 수 없을 때는 키워드 검색으로 전환해 핵심 커뮤니티 기능과 검색 흐름이 모두 멈추지 않도록 했습니다.

## 03 실제 작동 예시

| 상황 | 예시: 사용자가 장녹수에 관해 질문한 경우 |
|---|---|
| 01 질문 이해 | Agent가 질문에서 장녹수라는 핵심 인물과 사용자가 알고 싶은 내용을 먼저 찾습니다. |
| 02 내부 자료 검색 | 정리해 둔 역사 자료에서 관련 문서를 찾고 질문과 가까운 근거 후보를 모읍니다. |
| 03 근거 검증 | 문서 제목, 시대, 요약에 장녹수가 실제로 포함되는지 확인하고 다른 인물의 자료는 제외합니다. |
| 04 외부 자료 보강 | 내부 근거가 부족하면 실록과 외부 검색을 연결하고 주요 근거와 참고 자료를 구분합니다. |
| 05 답변 표시 | 검증된 자료로 답변을 만들고 사용한 출처를 함께 보여줍니다. 근거가 약하면 그 한계도 표시합니다. |

03 / CASE STUDY & RETROSPECTIVE

# 역사덕담

| | |
|---|---|
| **PROJECT OUTCOME** | ■ 개인 프로젝트에서 게시판 기반과 RAG·Agent·MCP·캐시를 잇는 핵심 기능 구현<br>■ Markdown 역사 자료 3,709건의 수집·정규화·검색 파이프라인 구성<br>■ 게시글 목록 캐시 적용 후 평균 응답 시간 42.5% 감소, 처리량 74.8% 증가 |

## 대표 문제 해결

실제 작업 중 마주친 상황과 해결 과정을 한 사례로 정리했습니다.

| CASE 01 | 유사도는 높지만 다른 인물의 근거가 선택되는 문제 |
|---|---|
| **상황** | 장녹수를 질문했는데 세종이나 양녕대군 자료처럼 검색 점수만 높은 다른 인물의 문서가 근거로 채택될 수 있었습니다. |
| **원인** | 기존 판정은 유사도 임계값만 확인해 질문의 핵심 인물·사건이 문서 제목과 요약에 실제로 포함됐는지 검사하지 않았습니다. |
| **도입한 방법** | 질문에서 핵심 개체를 추출하고 citation의 제목·시대·요약과 대조했습니다. 내부 근거와 외부 자료를 주요 근거와 대중문화 보조 자료로 다시 분류하고, 직접 연결되지 않는 결과는 제외했습니다. |
| **해결 결과** | 관련 없는 내부 요약이 본문에 섞이지 않게 되었고, 내부 자료가 부족할 때는 외부 자료 후보라는 한계를 명확히 표시하게 됐습니다. |

## 회고

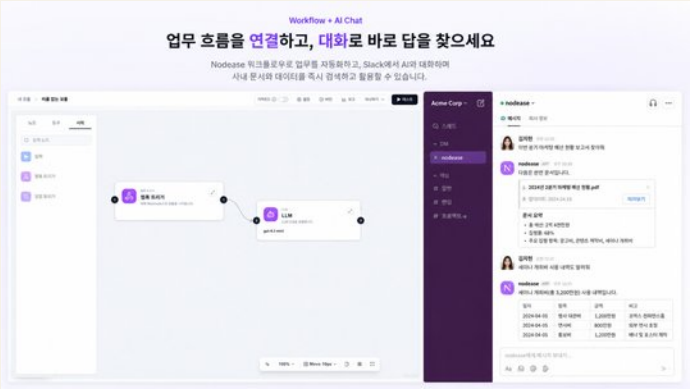
프로젝트에서 얻은 판단과 다음 작업에서 바꿀 점을 구분했습니다.

| | |
|---|---|
| **잘된 점** | ■ AI가 문장을 잘 생성하는 것보다 어떤 자료를 근거로 삼았는지 확인하는 경험을 먼저 설계했습니다.<br>■ 게시판부터 AI 글쓰기까지 프론트와 백엔드를 함께 구현해 사용자의 작성 흐름과 서버의 근거 처리 흐름을 연결했습니다. |
| **아쉬운 점** | ■ pgvector 컬럼과 인덱스는 준비했지만 현재 유사도 계산 일부가 Python에서 실행되어 데이터가 커질 때 성능을 다시 검증해야 합니다.<br>■ 사료·문헌형 외부 검색은 실록 검증 때문에 6~9초까지 걸릴 수 있어 짧은 인물 질문만큼 빠르지 않습니다. |
| **다시 만든다면** | ■ 벡터 검색을 데이터베이스 쿼리로 완전히 옮기고 검색 품질과 지연 시간을 함께 측정하겠습니다.<br>■ 느린 실록 검증은 첫 후보를 먼저 보여준 뒤 비동기로 보강하는 방식으로 분리하겠습니다. |

04 / GENERAL DEVELOPMENT

# Nodease

기업의 AI 업무 흐름을 시각적으로 설계하고 권한·비용·실행 기록까지 운영하는 LLMOps 플랫폼




| | |
|---|---|
| 기간 | 2026.06.24 - 2026.07.31 |
| 인원 | 5명 · 개발 5명 |
| 담당 | 시각적 워크플로우 편집·테스트 실행 UX · LLM 비용 최적화와 실행 비교 기능 · Judge 기반 자동 모델 라우팅 및 실험 |
| 기술 | Next.js 16 · React 19 · TypeScript · React Flow · FastAPI · Python · Celery · PostgreSQL |
| 링크 | GitHub ↗ |

**OVERVIEW**

기업 내부 사용자가 AI 업무 흐름을 노드로 설계하고 테스트·배포하며, 조직의 권한과 지식 문서, 실행 비용과 감사 기록까지 한곳에서 운영하는 오픈소스 플랫폼입니다.

저는 시각적 워크플로우를 실제로 편집하고 실행하는 경험을 중심으로 개발했고, LLM 노드의 후보 설정을 비교해 비용을 줄이는 Cost Optimizer와 요청 난이도에 맞는 모델을 고르는 자동 라우팅 기능을 프론트엔드부터 실행 엔진, 실험 도구까지 연결했습니다.


01 노드 기반 업무 자동화와 사내 AI 채팅을 연결하는 제품 화면

02 업무 흐름의 방향을 찾는다는 의미를 담은 Nodease 심볼

04 / PROJECT DETAIL

# Nodease

사용자가 경험하는 기능과 제가 구현한 방식을 실제 사용 상황과 함께 설명합니다.

## 01 주요 기능

01 시각적 워크플로우 편집
입력, LLM, RAG, 조건, 외부 도구, 응답 노드를 캔버스에서 연결합니다. 변수 출력을 다음 노드로 드래그하고, 단축키와 복사·붙여넣기, 실행 결과 비교를 이용해 복잡한 흐름을 구성할 수 있습니다.

02 자연어 Agent Builder
업무 설명을 자연어로 입력하면 필요한 노드와 연결을 제안합니다. 사용자는 생성된 흐름과 필수 설정을 확인한 뒤 워크플로우 편집기에 적용할 수 있습니다.

03 권한 기반 RAG
조직과 팀, 사용자 권한을 기준으로 접근 가능한 Knowledge Base만 검색 후보에 포함합니다. 검색에 사용한 문서와 citation을 실행 기록에 남깁니다.

04 LLMOps와 자동 모델 선택
노드별 토큰, 비용, 지연 시간과 실행 결과를 비교하고 더 적절한 모델 설정을 추천합니다. 자동 라우팅은 요청의 요구 수준과 운영에서 학습한 결과를 바탕으로 후보 모델을 선택합니다.

## 02 구현 기능

01 워크플로우 편집 경험
React Flow 캔버스에 grid snap, 노드 번호, 복사·붙여넣기·복제, undo·redo와 상세 패널 크기 조절을 구현했습니다. 이전 노드의 출력 변수를 칩으로 보여주고 클릭 또는 드래그로 다음 노드 설정에 삽입하도록 만들었습니다.

02 그래프 검증과 테스트 실행
노드 연결과 필수 입력, 응답 필드 이름을 실행 전에 검사하고 오류 위치를 사용자에게 안내합니다. 테스트 실행 결과와 각 노드 로그, 최종 응답, 이전 실행 비교를 사이드 패널에서 확인하고 다시 열어도 결과를 복원할 수 있게 했습니다.

03 Cost Optimizer
기준 LLM 노드와 후보 모델·파라미터를 같은 입력으로 실행해 비용, 지연 시간, 출력 계약과 품질을 비교합니다. 추천 설정을 바로 적용하지 않고 미리보기와 재검증을 거치며, 권한과 비용 상한을 통과한 결과만 워크플로우에 반영하도록...

04 Judge-first 자동 모델 라우팅
요청의 난이도와 필요한 능력을 Judge가 판정해 모델을 고르고, 확정된 운영 결과를 로컬 분류기의 학습 데이터로 축적했습니다. 충분히 학습된 경우 로컬 분류기가 먼저 선택하고 자신이 없으면 다시 Judge로 보내는 경로를 구현했습니다.

## 03 실제 작동 예시

| 상황 | 예시: 입력 노드와 LLM 노드를 연결해 테스트한 경우 |
|---|---|
| 01 흐름 설계 | 사용자가 입력, LLM, 응답 노드를 캔버스에 놓고 실행될 순서대로 연결합니다. |
| 02 실행 전 검사 | 필수 입력과 노드 연결, 응답 필드 이름을 확인해 잘못된 부분을 실행 전에 알려줍니다. |
| 03 순서대로 실행 | 검사를 통과하면 서버가 첫 노드부터 실행하고 각 결과를 다음 노드의 입력으로 전달합니다. |
| 04 기록 저장 | 노드별 입력과 출력, 오류, 실행 시간과 비용을 기록해 나중에 같은 실행을 다시 확인할 수 있게 합니다. |
| 05 결과 확인 | 최종 응답과 각 노드의 로그를 화면에 보여주고 실패했다면 문제가 발생한 위치를 함께 안내합니다. |

04 / CASE STUDY & RETROSPECTIVE

# Nodease

| | |
|---|---|
| **PROJECT OUTCOME** | ■ 5인 팀에서 워크플로우 편집 UX와 Cost Optimizer, 자동 모델 라우팅을 프론트·API·실행 엔진까지 연결<br>■ 80건 비교 실험에서 고가 모델 고정 대비 $1.367401의 실행 비용 절감 확인<br>■ 자동 라우팅 평균 품질 85.8점, JSON 출력 계약 100% 통과 |

## 대표 문제 해결

실제 작업 중 마주친 상황과 해결 과정을 한 사례로 정리했습니다.

| CASE 01 | 응답 필드 이름 때문에 실행 시점에 실패하는 문제 |
|---|---|
| **상황** | 응답 노드에서 한글, 대문자, 공백, 중복 이름처럼 JSON 필드로 안전하지 않은 값을 입력해도 편집 화면에서는 저장할 수 있어 실행 단계에서 오류가 발생했습니다. |
| **원인** | 프론트 입력 규칙과 Workflow Engine의 데이터 규칙이 분리되어 있었고, 사용자에게 허용 형식과 길이를 안내하지 않았습니다. |
| **도입한 방법** | 영문 소문자로 시작하고 소문자·숫자·언더바만 허용하는 규칙과 최대 32자 제한을 프론트와 Pydantic 스키마에 동일하게 적용했습니다. 중복 이름도 입력 즉시 검사해 오류 이유를 필드 아래에 표시했습니다. |
| **해결 결과** | 잘못된 응답 구조가 실행 엔진에 전달되기 전에 편집 화면에서 바로 수정할 수 있고, 클라이언트와 서버가 같은 계약을 사용하게 됐습니다. |

## 회고

프로젝트에서 얻은 판단과 다음 작업에서 바꿀 점을 구분했습니다.

| | |
|---|---|
| **잘된 점** | ■ 편집 화면의 작은 조작부터 서버의 실행 계약까지 함께 수정해 사용자가 만든 워크플로우가 실제로 실행되는 전체 흐름을 다뤘습니다.<br>■ 비용 최적화를 단순 추천 문구로 끝내지 않고 동일 입력의 실행 결과와 독립 품질 평가로 검증했습니다. |
| **아쉬운 점** | ■ 자동 라우팅은 고가 모델 고정보다 저렴했지만 중간 모델 고정보다 비용과 평균 시간이 높아, 모든 요청에서 항상 가장 경제적인 방식은 아니었습니다.<br>■ Runtime Judge 호출이 80건 중 50건 발생해 선택 자체의 비용과 지연 시간이 컸습니다. |
| **다시 만든다면** | ■ 초기부터 고정 평가 데이터와 비용·품질 기준선을 만들고 라우팅 전략 변경마다 같은 보고서를 자동 생성하겠습니다.<br>■ 학습된 로컬 라우터가 안전하게 처리할 수 있는 범위를 늘려 Judge 호출 비율과 지연 시간을 낮추겠습니다. |