1. 주요 UI 요구사항

키워드 중심 인터페이스

사용자가 키워드를 입력하면 해당 키워드의 수집 결과(뉴스, 논문, 표준, 커뮤니티 글 등)를 한 화면에서 조회할 수 있어야 합니다.

키워드별 히스토리 페이지에서 연도별 또는 사건별로 묶어 보여주는 '타임라인' 뷰를 제공합니다.

다양한 자료 유형 표시

각 자료를 "뉴스", "논문/보고서", "표준/정책", "커뮤니티/토론" 등으로 유형별 구분하여 탭 또는 필터로 분류합니다.

유형별로 핵심 메타데이터를 다르게 표현합니다. 예: 논문은 저자/발표 연도, 표준 문서는 번호/발행 기관, 커뮤니티 글은 추천수/댓글수.

품질 및 신뢰도 시각화

QG/FC 점수 및 키워드 스코어를 직관적인 배지나 색상으로 표시해 편집자가 빠르게 우선순위를 파악할 수 있게 합니다.

각 자료의 신뢰도, 최신성, 소스 권위를 시각적으로 강조합니다.

편집·승인 워크플로우

편집자는 자료별로 승인, 보류, 제외 등을 클릭하여 상태를 변경할 수 있습니다.

승인된 자료만 발행 페이지에 포함되며, 한 줄 코멘트(편집장 의견)를 입력하는 필드가 필요합니다.

키보드 단축키(예: ↑↓ 이동, Enter 승인, Space 고정)를 지원하여 작업 효율을 높입니다.

검색 및 필터링

목록 상단에 실시간 검색창을 두어 제목, 요약, 출처를 키워드로 필터링합니다.

날짜 범위, 신뢰도 등 추가 필터를 제공하여 자료를 빠르게 좁힐 수 있게 합니다.

발행 결과 미리보기

편집자는 발행 직전에 "미리보기" 버튼을 통해 키워드별 최종 페이지가 어떻게 보일지 확인할 수 있어야 합니다.

미리보기에서는 타임라인, 유형별 섹션, 요약/번역, 편집장 코멘트가 모두 포함되어야 합니다.

---

2. 프론트엔드 구조 및 기술 스택

SPA 프레임워크

Vue.js나 React.js로 단일 페이지 애플리케이션(SPA)을 구축해 동적인 데이터 로딩과 상태 관리를 구현합니다.

Vue의 경우 Vue 3 + Composition API, React의 경우 Create React App 또는 Vite 기반을 추천합니다.

상태 관리

키워드별로 수집된 자료를 저장하고 필터링 상태를 관리하기 위해 Vuex/Pinia(또는 Redux) 같은 상태 관리 라이브러리를 사용합니다.

컴포넌트 설계

검색/키워드 입력 컴포넌트: 키워드 입력, 검색 버튼, 최근 검색 목록.
타입 필터 컴포넌트: 체크박스 또는 탭 형식으로 자료 유형을 필터링.
자료 리스트 컴포넌트: v-for/map으로 각 자료를 카드/테이블 형태로 표현하고, 상태를 변경할 수 있는 액션 버튼 포함.
요약/메타데이터 모달: 자료를 클릭하면 상세 정보와 한국어 요약, 원문 링크, 출처 등을 보여주는 모달창.
미리보기/발행 컴포넌트: 승인된 자료를 기반으로 최종 페이지를 렌더링하며, HTML/Markdown 변환 결과를 보여줌.
UI 프레임워크
디자인 일관성을 위해 Tailwind CSS나 Vuetify, Chakra UI 같은 UI 라이브러리를 사용할 수 있습니다.
Dark/Light 모드와 반응형 레이아웃을 지원하도록 설계합니다.
에디터 노트 입력
각 자료 카드 내에 한 줄 코멘트 입력 필드를 제공하거나, 모달 내에서 코멘트를 입력할 수 있는 입력창을 준비합니다.
코멘트에는 기본값(예: 첫 문장 요약)을 표시하고, 편집자가 수정할 수 있게 합니다.

---

3. 백엔드 API 설계

새 UI를 지원하기 위해 Flask/FastAPI 백엔드에 다음 API를 추가합니다.

메서드	경로	설명
GET	/api/keyword/&kw&	키워드별 자료 목록 조회. 파라미터로 type(뉴스/논문 등), start_date, end_d 있음.
POST	/api/keyword/&kw&/collect	지정 키워드에 대해 모든 유형의 자료를 즉시 수집하고 DB/JSON에 저장.
POST	/api/keyword/&kw&/approve	편집자가 승인·보류·제외 결과를 전달. 요청 바디에 자료 ID와 상태, 한 줄 코
GET	/api/preview/&kw&	해당 키워드의 발행용 HTML/MD 미리보기 데이터를 반환.
POST	/api/publish/&kw&	승인된 자료를 최종 발행하고 파일을 저장.

기존 jjippa_admin.py와 비슷한 API 구조를 유지하되, 키워드 매개변수를 통해 자료를 구분합니다.

---

4. 개발 단계 제안

요구사항 상세화 및 와이어프레임
편집자와 협의하여 화면 구성과 기능 요구를 확정하고, Figma/Adobe XD 등으로 간단한 UI 와이어프레임을 제작합니다.
백엔드 API 확장
키워드별 수집·조회·승인·발행 API를 Flask/FastAPI에 추가합니다.
기존 데이터 모델을 키워드 중심 구조로 수정하고, DB 설계도 업데이트합니다.

프론트엔드 SPA 구축

Vue/React 프로젝트 초기화 후, 공통 컴포넌트와 라우팅을 설정합니다.

API 호출을 위한 Axios/fetch 래퍼를 구현하고, 테스트용 더미 데이터를 사용해 뷰를 구성합니다.

기능 구현 및 통합

목록 조회, 필터링, 승인/보류 버튼, 한 줄 코멘트 입력 기능 구현.

한국어 요약/메타데이터 표시 및 모달창 구현.

발행 미리보기 페이지 구현 후, 백엔드와 통합하여 실데이터 테스트.

피드백 반영 및 개선

편집자 시범 사용 후 UI/UX 피드백을 수집하고 반복 개선합니다.

Dark 모드, 키보드 단축키, 반응형 레이아웃 등을 추가합니다.

배포 및 운영 문서화

개발·운영 서버 분리, 환경설정(.env) 관리, 도메인/SSL 설정 등 배포 준비를 완료합니다.

편집자용 사용 설명서와 개발자용 설치·운영 매뉴얼을 작성합니다.

새로운 퀄리뉴스 UI는 키워드 기반 자료를 효과적으로 관리하고, 편집자가 다양하고 방대한 자료를 손쉽게 검토·선택·발행할 수 있도록 설계되어야 합니다. 위 가이드를 토대로 설계와 구현을 진행하면, 키워드 역사를 한눈에 보여주는 큐레이션 플랫폼으로 퀄리뉴스를 새롭게 탈바꿈할 수 있을 것입니다.