Synapse ("기억의 비서") 프로젝트: 시스템 설계 명세서 및 AI 기반 개발 플레이북

제 1장: 'Synapse: 기억의 비서'의 기초 청사진

본 장에서는 Synapse 프로젝트의 철학적, 개념적 핵심을 정립합니다. 이는 이후의 모든 기술적 결정을 뒷받침하는 근본적인 '왜'에 대한 답변이 될 것입니다.

1.1. 핵심 가치 제안: 지능적으로 연결된 두 번째 뇌

문제 정의
현대인은 정보 과부하와 지식 파편화라는 딜레마에 직면해 있습니다. 정보는 북마크, 노트 앱, 문서, 그리고 사용자의 머릿속에 흩어져 있습니다. 이러한 정보 조각들 사이의 맥락과 연결고리는 종종 유실되어, 지식은 '밑 빠진 독'처럼 새어 나갑니다. 중요한 것은 개별 정보를 저장하는 것이 아니라, 그 정보들이 어떻게 서로 관계를 맺고 있는지를 이해하는 것입니다.
Synapse의 해결책
Synapse는 단순한 노트 필기 앱이 아닙니다. 이것은 개인의 지식 그래프(Knowledge Graph)를 구축하는 도구입니다. 그 목적은 정보가 어떻게 연결되는지에 대한 사용자의 정신적 모델을 외부화하는 데 있습니다. Synapse는 평면적인 데이터 목록을 역동적이고 조회 가능한 통찰력의 네트워크로 변환합니다. 이 시스템의 핵심 가치는 정보를 저장하는 데 있는 것이 아니라, 정보 조각들 사이의 관계를 표면으로 드러내는 것에 있습니다. 사용자는 흩어진 점들을 의미 있는 선으로 연결함으로써, 단편적인 지식을 종합적인 이해로 승화시킬 수 있습니다.

1.2. 개념적 데이터 모델: 노드, 엣지, 그리고 속성

Synapse의 데이터 구조는 지식의 본질을 모방합니다. 즉, 개별 정보 단위와 그들 사이의 관계를 명확히 구분하여 표현합니다.

노드(Node)
노드는 정보의 기본 단위입니다. 각 노드는 고유한 정체성을 가지며, 다양한 유형으로 분류될 수 있습니다. 초기 MVP 단계에서는 다음과 같은 Node 유형을 정의합니다: Note(사용자 생각), WebClip(웹 아티클), Document(업로드된 파일), Image(이미지 자료), Concept(추상적 개념). 모든 노드는 공통적으로 핵심 속성(id, title, content, source_url, createdAt, updatedAt)을 가지며, 유형에 따라 특화된 속성을 추가로 가질 수 있습니다.
엣지(Edge)
엣지는 관계의 표현입니다. 엣지는 이 시스템의 '시냅스'이며, 노드만큼이나 중요합니다. Synapse의 엣지는 단순한 연결선이 아니라, 그 자체로 정보를 담는 일급 객체(First-class citizen)입니다. 엣지는 방향성을 가집니다(A에서 B로의 연결은 B에서 A로의 연결과 다릅니다). 그리고 가장 중요한 특징으로, 엣지는 유형화되어 있으며 자체적인 속성을 가질 수 있습니다.
엣지 유형(Edge Types)
의미론적 관계를 표현하기 위해 다음과 같은 초기 엣지 유형 집합을 정의합니다. 이 유형들은 사용자가 지식 사이의 미묘한 맥락을 명시적으로 기록할 수 있게 합니다.

- REFERENCES: 한 노드가 다른 노드를 인용하거나 참조함. (예: 논문 A가 통계 자료 B를 REFERENCES한다.)
- EXPANDS_ON: 한 노드가 다른 노드의 내용을 상세히 설명하거나 확장함. (예: 요약 노트 A가 원본 아티클 B를 EXPANDS_ON한다.)
- CONTRADICTS: 한 노드가 다른 노드에 대해 반대되는 관점을 제시함. (예: 비평 C가 주장 A를 CONTRADICTS한다.)
- SUPPORTS: 한 노드가 다른 노드에 대한 근거를 제공함. (예: 실험 데이터 D가 가설 E를 SUPPORTS한다.)
- IS_A: 특정 사례가 일반적인 개념에 속함을 나타냄. (예: 'GPT-4 논문 노트'는 '머신러닝'이라는 Concept 노드의 IS_A 인스턴스이다.)

이러한 의미론적 계층(Semantic Layer)의 도입은 Synapse를 다른 지식 관리 도구와 근본적으로 차별화합니다. 대부분의 도구는 단순하고 유형이 없는 백링크(A가 B에 연결됨)만을 제공합니다. 이러한 단순한 연결은 관계의 존재 여부만 알려줄 뿐, 그 관계의 성격에 대해서는 아무런 정보를 주지 않습니다. EXPANDS_ON이나 CONTRADICTS와 같은 유형화된 엣지를 도입함으로써, 우리는 지식 그래프에 기계가 읽고 해석할 수 있는 의미의 층을 추가합니다. 이는 지식 베이스를 단순한 네트워크에서 사용자의 이해를 반영하는 모델로 변환시킵니다.

이러한 구조는 훨씬 더 강력한 질의(Query)를 가능하게 합니다. 사용자는 "노드 A에 연결된 모든 것을 보여줘"라는 단순한 요청을 넘어, "노드 A의 주장을 반박하는 모든 논거를 찾아줘" 또는 "가설 B를 뒷받침하는 모든 증거를 요약해줘"와 같은 고차원적인 질문을 던질 수 있게 됩니다. 따라서 링크를 생성하는 UI/UX는 이러한 엣지 유형을 손쉽게 할당할 수 있도록 설계되어야 하며, 시스템 아키텍처는 엣지를 자체 속성(예: 관계에 대한 사용자 코멘트)을 가질 수 있는 독립적인 개체로 취급해야 합니다.

1.3. 사용자 여정**:** 정보 포착에서 통찰 생성까지

이상적인 사용자 워크플로우는 다음과 같은 연속적인 순환 과정으로 매핑됩니다.

1. 포착**(Capture):** 사용자는 정보(생각, 웹 아티클, PDF)를 마주치고 최소한의 마찰로 Synapse에 수집합니다. 예를 들어, 전역 단축키로 새 노트를 생성하거나 브라우저 확장 프로그램을 통해 웹 페이지를 클리핑합니다.
2. 처리**(Process):** 사용자는 포착된 노드를 다듬고, 태그를 추가하고, 핵심 구절을 하이라이트하며, 자신만의 요약을 작성합니다.
3. 연결**(Connect):** 사용자(또는 AI 어시스턴트)가 유형화된 엣지를 생성하여 새 노드를 지식 그래프 내의 기존 노드들과 연결합니다. 이것이 바로 맥락을 구축하는 핵심 단계입니다.
4. 검색 및 발견**(Retrieve & Discover):** 시간이 흐른 뒤, 사용자는 특정 주제를 검색합니다. 검색 결과에는 직접적인 키워드 일치 항목뿐만 아니라, 그래프 탐색을 통해 발견된 관련 노드들이 함께 표시되어 예상치 못한 연결(Serendipity)을 발견하게 합니다.
5. 종합**(Synthesize):** 사용자는 여러 노드와 그 연결 관계를 하나의 캔버스나 새로운 요약 노트에 모아, 상호 연결된 정보로부터 새로운 통찰력을 생성합니다.

제 **2**장**:** 핵심 아키텍처 및 기술 스택

본 장에서는 프로젝트의 '어떻게'를 정의합니다. 모든 주요 기술적 결정에 대한 명확한 근거와 함께 엔지니어링 청사진을 제공합니다.

2.1. 시스템 아키텍처 개요**:** 실용적인 서비스 지향 접근법

MVP(Minimum Viable Product) 단계에서는 엄격한 마이크로서비스 아키텍처(MSA) 대신, 모듈성과 개발 단순성 사이의 균형을 제공하는 서비스 지향 아키텍처(SOA)를 채택합니다. 시스템은 잘 정의된 API를 통해 통신하는 여러 개의 독립적으로 배포 가능한 서비스로 구성됩니다.

핵심 서비스**(Core Services):**

- Gateway Service: 모든 클라이언트 요청의 단일 진입점입니다. 라우팅, 인증, API 속도 제한(Rate Limiting)을 처리합니다.
- User & Auth Service: 사용자 프로필, 회원가입, 로그인, JWT(JSON Web Token) 생성 및 검증을 관리합니다.
- Knowledge Graph Service: 애플리케이션의 심장부입니다. 노드와 엣지에 대한 모든 CRUD(Create, Read, Update, Delete) 작업을 처리하고, 복잡한 그래프 탐색 쿼리를 실행합니다.

- Ingestion Service: 웹 클리퍼나 파일 업로드로부터 들어오는 데이터를 비동기적으로 처리하는 워커(Worker) 집합입니다. (예: 웹페이지 전문 추출, 썸네일 생성, PDF 파싱)
- Search Service: 초기에는 전문 검색(Full-text Search)을 담당하며, 향후 시맨틱/벡터 검색 통합을 고려하여 설계됩니다.

웹 클리핑이나 파일 업로드와 같은 작업은 시간이 오래 걸리고 리소스를 많이 소모할 수 있습니다. 만약 메인 API 핸들러가 웹 페이지 스크래핑, 정제, 저장이 완료될 때까지 기다린다면, 사용자에게는 애플리케이션이 매우 느리게 느껴질 것입니다. 이는 포착 행위와 처리 로직 간의 강한 결합(Tight Coupling)을 만들어 시스템을 취약하게 만듭니다.

이 문제를 해결하기 위해, RabbitMQ나 AWS SQS와 같은 메시지 큐(Message Queue)를 도입합니다. API 엔드포인트는 요청을 검증한 후 "이 URL을 처리하라"는 메시지를 큐에 넣는 역할만 수행하고, 즉시 사용자에게 202 Accepted 응답을 반환합니다. Ingestion Service의 워커들은 이 큐를 구독하고 있다가 작업을 가져와 백그라운드에서 독립적으로 처리합니다. 이 비동기적 수집(Asynchronous Ingestion) 아키텍처는 애플리케이션의 체감 속도를 향상시키고, 워커 하나가 실패하더라도 메시지를 재처리할 수 있어 장애 허용성(Fault Tolerance)을 높이며, 수집 파이프라인을 사용자 대면 API와 독립적으로 확장할 수 있게 해줍니다. 이는 사용자 경험과 시스템 확장성 모두에 핵심적인 설계 결정입니다.

2.2. 기술 스택 선정

각 기술은 Synapse 프로젝트의 특성에 맞춰 신중하게 선택되었으며, 그 결정 과정과 이유는 다음 표에 요약되어 있습니다.

구성 요소	선택된 기술	고려된 대안	선정 근거 (성능, 생태계, 확장성, 프로젝트 적합성)
프론트엔드	React (with Vite) + TypeScript	Vue.js, Svelte	압도적인 생태계, 대규모 애플리케이션 유지보수를 위한 강력한 타입 시스템, 우수한 상태 관리 라이브러리(Zustand), 방대한 개발자 풀.

백엔드	Node.js + NestJS (TypeScript)	Python (FastAPI), Go	웹 API에 필수적인 I/O 집약적 작업에 대한 뛰어난 성능, 프론트엔드와 언어 통일(TypeScript), 확장 가능하고 유지보수 용이한 서비스를 위한 구조화된 프레임워크 제공.
그래프 **DB**	Neo4j	PostgreSQL (w/ extensions), AWS Neptune	고도로 연결된 데이터의 저장 및 쿼리에 특화된 네이티브 그래프 성능, 표현력 풍부한 Cypher 쿼리 언어, 성숙한 도구 생태계. 'Synapse' 개념의 핵심.
관계형 **DB**	PostgreSQL	MySQL, MariaDB	사용자 계정, 메타데이터 등 정형 데이터 관리를 위한 타의 추종을 불허하는 신뢰성, 강력한 기능(FTS, JSONB).
API 스타일	하이브리드 (REST + GraphQL)	REST only, GraphQL only	작업에 가장 적합한 도구 사용: 단순 리소스 관리는 REST, 복잡한 그래프 데이터 조회는 GraphQL을 사용하여 오버페칭(Over-fetching) 방지.
배포	Docker +	Docker Swarm, Serverless	컨테이너 오케스트레이션의

	Kubernetes	(Lambda)	산업 표준으로, 자동 확장(Auto-scaling) 및 복원력 있는 인프라 구축 가능.

2.3. 데이터베이스 스키마 및 데이터 영속성 전략

Synapse는 두 종류의 데이터를 다루며, 각 데이터의 특성에 최적화된 데이터베이스를 사용하는 하이브리드 저장 전략을 채택합니다.

PostgreSQL 스키마 (User & Auth 및 Node Content용)
관계형 데이터베이스는 사용자 정보와 노드의 대용량 콘텐츠 본문을 저장하는 데 사용됩니다.

- users 테이블: id, email, password_hash, created_at 등 사용자 계정 정보.
- nodes_content 테이블: id (UUID, Primary Key), content_body (TEXT), raw_source (HTML/Markdown), version_history (JSONB).

Neo4j 데이터 모델 (Knowledge Graph용)
그래프 데이터베이스는 노드 간의 관계, 즉 지식의 구조를 저장하는 데 사용됩니다.

- 노드**(Nodes):** (:InformationNode {id: UUID, title: String, nodeType: String, createdAt: DateTime}), (:ConceptNode {id: UUID, name: String})
- 관계**(Relationships):** , 등

이 하이브리드 데이터 저장 전략은 성능 최적화를 위한 핵심적인 결정입니다. 그래프 데이터베이스는 대용량 텍스트나 바이너리 데이터를 속성으로 저장하는 데 항상 최적은 아닙니다. 수 메가바이트에 달하는 HTML 콘텐츠를 Neo4j 노드 속성에 직접 저장하면 그래프 저장소의 크기가 비대해져 순회 성능이 저하될 수 있습니다. 그래프 저장소는 위상(Topology)과 메타데이터에 집중하여 가볍게 유지해야 합니다. 반면, PostgreSQL은 대용량 TEXT 필드를 저장하고 검색하는 데 매우 최적화되어 있습니다.

따라서, 그래프의 구조는 Neo4j에, 노드의 부피가 큰 콘텐츠는 PostgreSQL에 저장합니다. Neo4j의 InformationNode는 UUID를 가지며, 해당 노드의 전체 콘텐츠는 PostgreSQL의 nodes_content 테이블에서 동일한 UUID를 기본 키로 하는 행에 저장됩니다. 사용자가 특정 노드를 요청하면, Knowledge Graph Service는 먼저 Neo4j에 노드의 메타데이터와 연결 정보를 질의한 다음, 반환된 UUID를 사용하여 PostgreSQL에서 전체 콘텐츠를 가져옵니다. 이 방식은 각 데이터베이스의 장점을 최대한 활용하는 효율적인 접근법입니다.

2.4. API 설계 철학**: RESTful** 엔드포인트와 **GraphQL** 스키마

Synapse는 작업의 성격에 따라 두 가지 API 스타일을 혼용하여 클라이언트-서버 통신의 효율성을 극대화합니다.

REST API (리소스 관리용)
리소스 중심의 명확한 CRUD 작업에는 REST가 이상적입니다.

- POST /auth/register: 신규 사용자 등록
- POST /auth/login: 사용자 로그인
- POST /nodes: 새 노드 생성
- GET /nodes/{id}: 특정 노드의 전체 데이터 조회

GraphQL API (그래프 탐색용)
클라이언트가 복잡하고 중첩된 데이터 구조를 한 번의 요청으로 가져와야 하는 그래프 탐색에는 GraphQL이 월등한 성능을 보입니다.

- **Query** 타입**:** node(id: ID!): Node, neighbors(nodeId: ID!, hops: Int = 1): [Node]
- **Mutation** 타입**:** createLink(fromId: ID!, toId: ID!, type: EdgeType!, properties: JSON): Edge

이러한 GraphQL 스키마를 통해 클라이언트는 다음과 같이 강력하고 유연한 쿼리를 작성할 수 있습니다. 이는 프론트엔드에서 그래프 데이터를 시각화할 때 여러 번의 API 호출 없이 필요한 모든 정보를 효율적으로 가져올 수 있게 해줍니다.

GraphQL

```
query ExploreNode {
  node(id: "some-uuid") {
    title
    connectedTo(type: "EXPANDS_ON") {
      edge { comment }
      node {
        title
        connectedTo(type: "REFERENCES") {
          node { title, sourceUrl }
        }
      }
    }
  }
}
```

제 **3**장**:** 사용자 경험 및 기능 요구사항 정의

본 장에서는 Synapse의 비전을 구체적인 기능 명세로 변환합니다. 각 기능은 모듈별로 그룹화되어 정의됩니다.

3.1. 모듈 **1:** 수집 엔진 **(Ingestion Engine)**

- **FR-1 (**수동 노트 생성**):** 마크다운을 지원하는 리치 텍스트 에디터(예: TipTap, Slate.js)를 통해 사용자가 직접 노트를 생성하고 편집할 수 있어야 합니다.
- **FR-2 (**웹 클리퍼**):** Chrome/Firefox용 브라우저 확장 프로그램을 제공하며, 다음 세 가지 모드를 지원해야 합니다.
  - Clip Article: 페이지의 광고나 불필요한 요소를 제거하고 본문 내용만 추출합니다.
  - Clip Selection: 사용자가 하이라이트한 텍스트만 캡처합니다.
  - Clip Bookmark: URL과 메타데이터(제목, 설명)만 저장합니다.
- **FR-3 (**파일 업로드**):** PDF 및 일반 텍스트 파일을 업로드할 수 있어야 합니다. 텍스트 추출은 백엔드의 Ingestion Service가 담당합니다.

3.2. 모듈 **2:** 지식 그래프 인터페이스

- **FR-4 (**노드 뷰**):** 단일 노드의 콘텐츠를 깨끗하고 가독성 높게 보여주는 뷰를 제공해야 합니다.
- **FR-5 (**링크 생성**):** 현재 보고 있는 노드를 다른 노드와 연결하는 직관적인 UI를 제공해야 합니다. 대상 노드를 쉽게 찾을 수 있도록 자동 완성 검색 기능이 포함되어야 합니다.
- **FR-6 (**유형화된 엣지 명시**):** 링크 생성 시, 사용자는 반드시 드롭다운 메뉴에서 엣지 유형(예: REFERENCES, SUPPORTS)을 선택하고, 선택적으로 관계에 대한 코멘트를 추가할 수 있어야 합니다. 이는 Synapse의 핵심인 의미론적 모델을 강제하는 필수 기능입니다.
- **FR-7 (**그래프 시각화**):** D3.js나 react-force-graph와 같은 라이브러리를 사용하여 현재 노드와 그 직접적인 이웃 노드들을 보여주는 시각적이고 상호작용적인 그래프 뷰를 제공해야 합니다.

3.3. 모듈 **3:** 검색 및 발견 인터페이스

- **FR-8 (**키워드 검색**):** PostgreSQL에 저장된 모든 노드 콘텐츠에 대해 전문 검색을 수행하는

전역 검색창을 제공해야 합니다.

- **FR-9 (**백링크 패널**):** 특정 노드를 볼 때, 해당 노드를 가리키는 다른 모든 노드(백링크) 목록을 사이드바에 표시해야 합니다. 이 목록은 엣지 유형별로 분류되어야 합니다.
- **FR-10 (**그래프 순회 쿼리 **- Post-MVP):** "'domain.com'을 REFERENCES하는 WebClip을 EXPANDS_ON하는 모든 Note를 보여줘"와 같은 복잡한 쿼리를 구성할 수 있는 고급 검색 인터페이스를 제공합니다 (MVP 이후 기능).

3.4. 모듈 **4:** 사용자 인증 및 데이터 보안

- **FR-11 (**사용자 등록 및 로그인**):** 표준적인 이메일/패스워드 기반의 인증 기능을 제공해야 합니다.
- **FR-12 (JWT** 기반 세션 관리**):** 백엔드는 API 요청을 인증하기 위해 상태 비저장(Stateless) 방식의 JSON 웹 토큰을 발급해야 합니다.
- **FR-13 (**데이터 격리**):** 모든 데이터베이스 쿼리는 반드시 현재 인증된 사용자의 범위로 제한되어야 합니다. 한 사용자가 다른 사용자의 데이터에 접근할 수 있는 가능성은 원천적으로 차단되어야 합니다.

제 **4**장**:** 비기능적 요구사항**:** 성능**,** 보안**,** 확장성

본 장에서는 애플리케이션의 품질과 전문성을 보장하는 비기능적 속성들을 정의합니다.

4.1. 성능 벤치마크

- **NFR-1 (API** 응답 시간**):** 사용자 대면 작업에 대한 모든 API의 95번째 백분위수(P95) 응답 시간은 250ms 미만이어야 합니다.
- **NFR-2 (**그래프 쿼리 시간**):** 2-홉(2-hop) 그래프 순회 쿼리의 P95 응답 시간은 750ms 미만이어야 합니다.
- **NFR-3 (**프론트엔드 로드 시간**):** 표준 광대역 연결 환경에서 애플리케이션의 초기 로드 시간(Largest Contentful Paint)은 2.5초 미만이어야 합니다.

4.2. 보안 프로토콜 및 위협 모델링

- **NFR-4 (**인증**):** 패스워드는 반드시 Argon2나 bcrypt와 같은 강력한 솔트(Salt) 기반 해싱 알고리즘을 사용하여 해시 처리되어야 합니다.
- **NFR-5 (**인가**):** 모든 API 엔드포인트는 보호되어야 하며, JWT 서명과 요청된 리소스에 대한 사용자 소유권을 검증해야 합니다.
- **NFR-6 (**데이터 전송**):** 모든 통신 트래픽은 TLS 1.2 이상을 사용하여 전송 중 암호화되어야 합니다.
- **NFR-7 (**데이터 저장**):** 데이터베이스에 저장되는 민감한 데이터는 저장 시 암호화(Encryption at Rest)되어야 합니다.
- **NFR-8 (OWASP Top 10):** 애플리케이션은 XSS, CSRF, SQL/Cypher Injection을 포함한 일반적인 웹 취약점으로부터 보호되어야 합니다. 이는 모든 데이터베이스 쿼리에 파라미터화된 쿼리(Parameterized Queries)를 사용하여 달성합니다.

4.3. 확장성 및 유지보수성 전략

- **NFR-9 (**상태 비저장 서비스**):** 데이터베이스를 제외한 모든 백엔드 서비스는 수평적 확장을 용이하게 하기 위해 상태 비저장(Stateless)으로 설계되어야 합니다.
- **NFR-10 (**비동기 처리**):** 웹 스크래핑, PDF 파싱, 향후 AI 임베딩 생성과 같은 장기 실행 작업은 반드시 메시지 큐를 통해 비동기 Ingestion Service로 오프로드되어야 합니다.
- **NFR-11 (**로깅 및 모니터링**):** 모든 서비스에 구조화된 로깅 프레임워크(예: Pino)를 구현해야 합니다. 로그는 ELK Stack이나 Datadog과 같은 중앙 집중식 플랫폼으로 집계되어야 하며, 주요 애플리케이션 메트릭(응답 시간, 에러율)은 지속적으로 모니터링되어야 합니다.

제 **5**장**: Claude Code** 개발 플레이북**:** 프롬프트 기반 구현 가이드

본 장은 이 문서의 실행 가능한 핵심 부분으로, Claude Code를 사용하여 Synapse를 구축하기 위한 구조화된 프롬프트 시퀀스를 제공합니다. 각 프롬프트는 이전 섹션에서 정의된 설계 명세를 컨텍스트로 제공하여 구체적이고 명확하게 작성되었습니다.

5.1. 1단계**:** 프로젝트 스캐폴딩 및 환경 설정

- 프롬프트 **5.1.1:**'Synapse' 프로젝트를 위한 루트 디렉토리를 생성하세요. 내부에 docker-compose.yml 파일을 만들고, gateway, users-service, graph-service, postgres-db, neo4j-db 서비스를 정의합니다. PostgreSQL(버전 15)과 Neo4j(버전 5)에는 표준 이미지를 사용하세요. 각 서비스 간 통신을 위한 네트워크와 데이터 영속성을 위한 볼륨을 정의해야 합니다.
- 프롬프트 **5.1.2:**새로운 /services/users 디렉토리 내에서 NestJS CLI (nest new)를 사용하여 새로운 TypeScript 프로젝트를 스캐폴딩하세요. 이 프로젝트는 2.2절에 정의된 아키텍처를 따라야 합니다.
- 프롬프트 **5.1.3:**새로운 /frontend 디렉토리 내에서 Vite를 사용하여 React + TypeScript 프로젝트를 스캐폴딩하세요. 코드 스타일링 및 포맷팅을 위해 ESLint, Prettier, 그리고 Tailwind CSS를 설정합니다.

5.2. 2단계**:** 백엔드 개발 **(API** 및 데이터베이스 계층**)**

- 프롬프트 **5.2.1 (User Service -** 데이터베이스**):**users-service 프로젝트에 TypeORM과 pg 드라이버를 설치하세요. 2.3절의 PostgreSQL 스키마에 따라 User 엔티티를 정의합니다. Docker Compose 파일에 정의된 PostgreSQL 컨테이너에 연결하는 데이터베이스 모듈을 생성하세요.
- 프롬프트 **5.2.2 (User Service -** 인증 로직**):**AuthService를 구현하세요. register (bcrypt로 패스워드 해싱)와 login (해시 비교 및 JWT 생성) 메소드를 만듭니다. @nestjs/jwt 패키지를 사용하고, JWT 시크릿 키는 환경 변수에서 가져오도록 설정하세요.
- 프롬프트 **5.2.3 (Graph Service - Neo4j** 연결**):**graph-service 프로젝트에 공식 neo4j-driver를 설치하세요. Neo4j 컨테이너와의 연결을 설정하는 Neo4jService를 생성합니다. 연결 상태를 확인하는 헬스 체크 메소드를 구현하세요.
- 프롬프트 **5.2.4 (Graph Service -** 노드 생성**):**GraphService에 createInformationNode 메소드를 생성하세요. 이 메소드는 title과 nodeType을 인자로 받습니다. 2.3절에 정의된 대로, UUID와 타임스탬프를 설정하여 Neo4j에 새로운 :InformationNode를 생성하는 파라미터화된 Cypher 쿼리를 실행해야 합니다.
- 프롬프트 **5.2.5 (API** 게이트웨이**):**gateway 서비스를 위한 NestJS 프로젝트를 설정하세요. /auth로 시작하는 요청은 users-service로, /graph로 시작하는 요청은 graph-service로 전달하는 리버스 프록시를 구현합니다.

5.3. 3단계**:** 프론트엔드 개발 **(UI** 및 상태 관리**)**

- 프롬프트 **5.3.1 (**상태 관리**):**React 애플리케이션의 전역 상태 관리를 위해 Zustand를 설정하세요. 사용자의 JWT와 프로필 정보를 보관하는 인증 스토어(store)를 생성합니다.
- 프롬프트 **5.3.2 (API** 클라이언트**):**axios를 사용하여 API 클라이언트를 생성하세요. Zustand

스토어의 JWT를 모든 발신 요청의 Authorization 헤더에 자동으로 첨부하는 인터셉터를 구성합니다.

- 프롬프트 **5.3.3 (**로그인 페이지**):**이메일과 패스워드 필드를 가진 반응형 로그인 페이지 컴포넌트를 생성하세요. 제출 시, API 클라이언트를 통해 /auth/login 엔드포인트를 호출하고, 성공 시 JWT를 상태에 저장한 후 대시보드로 리디렉션해야 합니다.
- 프롬프트 **5.3.4 (**노드 에디터 컴포넌트**):**TipTap 라이브러리를 사용하여 리치 텍스트 에디터 컴포넌트를 생성하세요. 볼드, 이탤릭, 리스트와 같은 기본 서식을 지원해야 하며, 내용이 변경될 때마다 컴포넌트의 상태에 저장해야 합니다.

5.4. 4단계**:** 통합**,** 테스트**,** 및 리팩토링

- 프롬프트 **5.4.1 (**단위 테스트**):**users-service에서 Jest를 사용하여 AuthService에 대한 단위 테스트를 작성하세요. 데이터베이스 리포지토리를 모킹(mocking)합니다. register 메소드가 패스워드 해싱 함수와 사용자 생성 메소드를 올바르게 호출하는지 테스트하세요.
- 프롬프트 **5.4.2 (**통합 테스트**):**Cypress를 사용하여 사용자 로그인 플로우에 대한 엔드-투-엔드(E2E) 테스트를 작성하세요. 테스트는 로그인 페이지를 방문하여 시드된 테스트 사용자의 자격 증명을 입력하고, 제출 버튼을 클릭한 후, URL이 대시보드 페이지로 변경되는지 확인해야 합니다.
- 프롬프트 **5.4.3 (**코드 리팩토링**):**GraphService의 createInformationNode 메소드를 검토하세요. 입력값 검증을 위해 class-validator 데코레이터가 있는 전용 데이터 전송 객체(DTO)를 사용하도록 리팩토링합니다. 모든 데이터베이스 쿼리가 에러 핸들링을 위해 try-catch 블록으로 감싸져 있는지 확인하세요.

제 **6**장**:** 전략적 로드맵**: MVP**에서 살아있는 아카이브까지

본 장에서는 Synapse의 미래 비전을 제시하여, MVP가 최종 목적지가 아닌 더 큰 목표를 향한 디딤돌이 되도록 합니다.

6.1. MVP 기능 우선순위

출시를 위한 절대적인 핵심 기능 루프는 다음과 같습니다.

1. 사용자 인증 (회원가입/로그인).
2. 텍스트 기반 Note 노드의 수동 생성 및 편집.

3. 두 노드 간에 유형화된 링크를 수동으로 생성하는 기능.
4. 간단한 키워드 검색.
5. 각 노드에 대한 "백링크" 뷰.

6.2. Post-MVP 향상 계획: 버전 2.0으로의 길

- **V1.1 (수집 기능 강화):** 웹 클리퍼 브라우저 확장 프로그램과 PDF/파일 업로드 기능을 구현합니다.
- **V1.2 (시각화):** 상호작용적인 그래프 시각화 인터페이스를 구축합니다.
- **V2.0 (AI 기반 통찰력):**
  - 시맨틱 검색: Pinecone이나 Weaviate 같은 벡터 데이터베이스를 통합합니다. 수집 과정에서 모든 콘텐츠에 대해 sentence-transformer 모델을 사용하여 벡터 임베딩을 생성합니다. 검색 인터페이스에 "의미 기반 검색" 옵션을 추가합니다.
  - **AI** 링크 추천: 사용자가 새 노드를 생성하면, AI 서비스가 해당 노드의 임베딩을 그래프 내 다른 모든 노드와 비교하여 연결할 만한 가장 관련성 높은 기존 노드 3~5개를 추천합니다.
  - **AI** 요약 및 질의응답: 사용자가 여러 노드를 선택하고, 선택된 콘텐츠를 기반으로 LLM에게 요약을 생성하거나 질문에 답하도록 요청할 수 있는 기능을 추가합니다.
- **V2.X (협업):** 사용자 간에 지식 그래프를 공유하고 공동으로 편집할 수 있는 기능을 도입합니다.

결론 및 권고

본 문서는 "기억의 비서 (Synapse)" 프로젝트를 성공적으로 개발하기 위한 포괄적인 청사진을 제시했습니다. 제안된 아키텍처는 확장성, 유지보수성, 그리고 성능을 고려하여 설계되었으며, 특히 의미론적 관계 표현을 위한 네이티브 그래프 데이터베이스(Neo4j)와 대용량 콘텐츠 저장을 위한 관계형 데이터베이스(PostgreSQL)의 하이브리드 접근법은 Synapse의 핵심 가치를 기술적으로 구현하는 최적의 방안입니다.

개발 과정에서는 제시된 AI 코딩 어시스턴트(Claude Code) 플레이북을 단계별로 따를 것을 강력히 권고합니다. 각 프롬프트는 이전 단계의 결과물을 기반으로 컨텍스트를 제공하므로, 순차적으로 진행할 때 가장 높은 효율성과 코드 품질을 보장할 수 있습니다.

MVP 단계에서는 핵심 기능 루프(수집-연결-발견)의 안정적인 구현에 집중하고, 이후 로드맵에 따라 AI 기반 기능들을 점진적으로 통합함으로써 Synapse를 단순한 정보 저장소를 넘어 진정한 '두 번째 뇌'이자 살아있는 지식 아카이브로 발전시켜 나갈 수 있을 것입니다. 성공적인 프로젝트 수행을 위해서는 초기 단계부터 비기능적 요구사항, 특히 데이터 보안과 격리에 대한

철저한 구현이 필수적입니다.