🏗️ Architecture & Component Structure

System Architecture Diagram

```
┌─────────────────────────────────────────────────────────────────────────┐
│                            USER INTERFACE                               │
│                            (Web Browser)                                │
└─────────────────────────────────────────────────────────────────────────┘
                                 │
                                 │ HTTP/REST
                                 ▼
┌─────────────────────────────────────────────────────────────────────────┐
│                          FRONTEND LAYER                                 │
│                       (React + Vite + Nginx)                            │
│                          Port: 3000                                     │
├─────────────────────────────────────────────────────────────────────────┤
│                                                                         │
│  ┌──────────────┐  ┌──────────────┐  ┌──────────────┐                   │
│  │  StacksList  │  │ StackBuilder │  │  ChatModal   │                   │
│  │    Page      │  │    Page      │  │  Component   │                   │
│  └──────────────┘  └──────────────┘  └──────────────┘                   │
│                                                                         │
│  ┌──────────────┐  ┌──────────────┐  ┌──────────────┐                   │
│  │ CustomNodes  │  │ StackSidebar │  │CreateStack   │                   │
│  │  Component   │  │  Component   │  │    Modal     │                   │
│  └──────────────┘  └──────────────┘  └──────────────┘                   │
│                                                                         │
│  ┌───────────────────────────────────────────────────────────────────┐  │
│  │              API Client Layer (Axios)                             │  │
│  │  • chat.js   • documents.js   • workflows.js   • client.js        │  │
│  └───────────────────────────────────────────────────────────────────┘  │
│                                                                         │
└─────────────────────────────────────────────────────────────────────────┘
                                 │
                                 │ REST API (JSON)
                                 ▼
┌─────────────────────────────────────────────────────────────────────────┐
│                          BACKEND LAYER                                  │
│                       (FastAPI + Python)                                │
│                          Port: 8000                                     │
├─────────────────────────────────────────────────────────────────────────┤
│                                                                         │
│  ┌─────────────────────────────────────────────────────────────────┐    │
│  │                     API ENDPOINTS                               │    │
│  │  ┌──────────────┐  ┌──────────────┐  ┌──────────────┐           │    │
│  │  │    Chat      │  │  Documents   │  │  Workflows   │           │    │
│  │  │  Endpoints   │  │  Endpoints   │  │  Endpoints   │           │    │
│  │  └──────────────┘  └──────────────┘  └──────────────┘           │    │
│  └─────────────────────────────────────────────────────────────────┘    │
│                                                                         │
│  ┌──────────────────────────────────────────────────────────────┐   │
│  │                  BUSINESS LOGIC (Services)                   │   │
│  │  ┌──────────────┐  ┌──────────────┐  ┌──────────────┐       │   │
│  │  │  Workflow    │  │     LLM      │  │   Vector     │       │   │
│  │  │   Service    │  │   Service    │  │   Service    │       │   │
│  │  └──────────────┘  └──────────────┘  └──────────────┘       │   │
│  │  ┌──────────────┐                                            │   │
│  │  │  Document    │                                            │   │
│  │  │   Service    │                                            │   │
│  │  └──────────────┘                                            │   │
│  └──────────────────────────────────────────────────────────────┘   │
│                                                                      │
│  ┌──────────────────────────────────────────────────────────────┐   │
│  │                  DATA ACCESS LAYER                           │   │
│  │  ┌──────────────┐  ┌──────────────┐                          │   │
│  │  │  SQLAlchemy  │  │   Database   │                          │   │
│  │  │    Models    │  │   Session    │                          │   │
│  │  └──────────────┘  └──────────────┘                          │   │
│  └──────────────────────────────────────────────────────────────┘   │
│                                                                      │
└────────────┬──────────────────────────────┬─────────────────────────┘
             │                              │
             │                              │
             ▼                              ▼
┌──────────────────────────┐   ┌──────────────────────────┐
│   PostgreSQL Database    │   │   ChromaDB Vector Store  │
│      Port: 5432          │   │    (Embedded/Local)      │
├──────────────────────────┤   ├──────────────────────────┤
│ • documents              │   │ • Document embeddings    │
│ • workflows              │   │ • Semantic search        │
│ • chat_sessions          │   │ • Vector similarity      │
│ • chat_messages          │   │                          │
└──────────────────────────┘   └──────────────────────────┘
             │
             │
             ▼
┌──────────────────────────┐
│   External Services      │
├──────────────────────────┤
│ • Google Gemini API      │
│   - Text Generation      │
│   - Embeddings           │
│ • SerpAPI (Optional)     │
│   - Web Search           │
└──────────────────────────┘
```

Data Flow Diagram

```
┌──────────────────────────────────────────────────────────────────────┐
│                    USER WORKFLOW EXECUTION FLOW                      │
└──────────────────────────────────────────────────────────────────────┘
```

```
1. USER CREATES WORKFLOW
   │
   ├─► User drags components to canvas (StackBuilder)
   │   └─► User Input → Knowledge Base → LLM Engine → Output
   │
   ├─► User configures each component
   │   └─► Sets model, prompts, temperature, etc.
   │
   ├─► User connects components
   │   └─► Creates edges between nodes
   │
   └─► User saves workflow
       └─► POST /api/v1/workflows/ → Stored in PostgreSQL


2. USER UPLOADS DOCUMENT (Optional)
   │
   ├─► User selects PDF/TXT file
   │   └─► POST /api/v1/documents/upload
   │
   ├─► Backend extracts text (PyMuPDF)
   │   └─► Document Service processes file
   │
   ├─► Backend generates embeddings (Gemini)
   │   └─► Vector Service creates embeddings
   │
   └─► Embeddings stored in ChromaDB
       └─► Ready for semantic search


3. USER CHATS WITH WORKFLOW
   │
   ├─► User opens chat modal
   │   └─► Creates new chat session
   │       └─► POST /api/v1/chat/sessions
   │
   ├─► User types query: "What is machine learning?"
   │   └─► Sent to backend
   │
   ├─► Backend executes workflow:
   │   │
   │   ├─► Step 1: User Query Component
   │   │   └─► Receives: "What is machine learning?"
   │   │
   │   ├─► Step 2: Knowledge Base Component
   │   │   ├─► Generates query embedding (Gemini)
   │   │   ├─► Searches ChromaDB for similar content
   │   │   └─► Returns: Top 3 relevant document chunks
   │   │
   │   ├─► Step 3: LLM Engine Component
   │   │   ├─► Receives: Query + Context from Knowledge Base
   │   │   ├─► Builds prompt with system instructions
   │   │   ├─► Calls Google Gemini API
   │   │   └─► Returns: AI-generated response
   │   │
   │   └─► Step 4: Output Component
   │       └─► Formats response for display
   │
   ├─► Response sent to frontend
   │   └─► Displayed in chat interface
   │
   └─► Message saved to database
       └─► Stored in chat_messages table
```

Component Structure

Frontend Components

```
frontend/src/
│
├── pages/                          # Page-level components
│   ├── StacksList.jsx             # Main landing page
│   │   ├── Lists all workflows
│   │   ├── Create new workflow button
│   │   └── Navigate to StackBuilder
│   │
│   └── StackBuilder.jsx           # Workflow builder page
│       ├── ReactFlow canvas
│       ├── Component drag-and-drop
│       ├── Workflow save/load
│       └── Chat modal trigger
│
├── components/                     # Reusable components
│   ├── CustomNodes.jsx            # All 4 workflow node types
│   │   ├── UserQueryNode         # User input component
│   │   ├── KnowledgeBaseNode     # Document retrieval
│   │   ├── LLMEngineNode         # AI processing
│   │   └── OutputNode            # Response display
│   │
│   ├── ChatModal.jsx              # Chat interface
│   │   ├── Message history
│   │   ├── Session management
│   │   ├── Delete session
│   │   └── Real-time messaging
│   │
│   ├── StackSidebar.jsx           # Component library
│   │   ├── Draggable components
│   │   └── Component descriptions
│   │
│   └── CreateStackModal.jsx       # New workflow modal
│       ├── Name input
│       ├── Description input
│       └── Create action
│
```

```
└── api/                            # API client layer
    ├── client.js                  # Axios configuration
    ├── chat.js                    # Chat API calls
    ├── documents.js               # Document API calls
    └── workflows.js               # Workflow API calls
```

Backend Components

```
backend/app/
│
├── api/                            # API layer
│   ├── endpoints/
│   │   ├── chat.py                # Chat endpoints
│   │   │   ├── POST /sessions    # Create session
│   │   │   ├── GET /sessions     # List sessions
│   │   │   ├── GET /sessions/{id}/messages
│   │   │   └── DELETE /sessions/{id}
│   │   │
│   │   ├── documents.py           # Document endpoints
│   │   │   ├── POST /upload      # Upload file
│   │   │   ├── POST /{id}/process # Process document
│   │   │   ├── GET /             # List documents
│   │   │   └── DELETE /{id}      # Delete document
│   │   │
│   │   └── workflows.py           # Workflow endpoints
│   │       ├── POST /            # Create workflow
│   │       ├── GET /             # List workflows
│   │       ├── GET /{id}         # Get workflow
│   │       ├── PUT /{id}         # Update workflow
│   │       ├── POST /execute     # Execute workflow
│   │       └── DELETE /{id}      # Delete workflow
│   │
│   └── routes.py                  # Route registration
│
├── services/                       # Business logic layer
│   ├── workflow_service.py        # Workflow orchestration
│   │   ├── validate_workflow()   # Check workflow validity
│   │   ├── execute_workflow()    # Run workflow steps
│   │   └── topological_sort()    # Order components
│   │
│   ├── llm_service.py             # LLM integration
│   │   ├── generate_response()   # Call Gemini API
│   │   ├── build_prompt()        # Construct prompts
│   │   └── stream_response()     # Handle streaming
│   │
│   ├── vector_service.py          # Vector operations
│   │   ├── generate_embeddings() # Create embeddings
│   │   ├── store_embeddings()    # Save to ChromaDB
│   │   └── search_similar()      # Semantic search
│   │
│   └── document_service.py        # Document processing
│       ├── extract_text()        # Extract from PDF/TXT
│       ├── chunk_text()          # Split into chunks
│       └── process_document()    # Full pipeline
│
├── database/                       # Data access layer
│   ├── models.py                  # SQLAlchemy models
│   │   ├── Document              # Document metadata
│   │   ├── Workflow              # Workflow definition
│   │   ├── ChatSession           # Chat session
│   │   └── ChatMessage           # Individual messages
│   │
│   └── database.py                # Database connection
│       ├── engine                # SQLAlchemy engine
│       ├── SessionLocal          # Session factory
│       └── get_db()              # Dependency injection
│
├── schemas/                        # Pydantic schemas
│   ├── chat.py                    # Chat request/response
│   ├── document.py                # Document schemas
│   └── workflow.py                # Workflow schemas
│
├── core/                           # Core configuration
│   └── config.py                  # Settings management
│       ├── DATABASE_URL
│       ├── GOOGLE_API_KEY
│       └── Other environment vars
│
└── main.py                         # Application entry point
    ├── FastAPI app initialization
    ├── CORS middleware
    ├── Route registration
    └── Startup/shutdown events
```

Modular Design Principles

1. Separation of Concerns

```
┌─────────────────────────────────────────────────────────────┐
│                    LAYERED ARCHITECTURE                     │
├─────────────────────────────────────────────────────────────┤
│  Presentation Layer  │  React Components, Pages, UI         │
├─────────────────────────────────────────────────────────────┤
│  API Layer           │  REST Endpoints, Request Handling    │
├─────────────────────────────────────────────────────────────┤
│  Business Logic      │  Services, Workflow Execution        │
├─────────────────────────────────────────────────────────────┤
│  Data Access         │  Models, Database Operations         │
├─────────────────────────────────────────────────────────────┤
│  Data Storage        │  PostgreSQL, ChromaDB                │
└─────────────────────────────────────────────────────────────┘
```

2. Component Independence

Each component is self-contained and can be:

- **Developed** independently
- **Tested** in isolation
- **Deployed** separately
- **Scaled** individually

3. Clear Interfaces

```python
# Service Interface Example
class LLMService:
    def generate_response(query: str, context: str, config: dict) -> str:
        """
        Input: Query, context, configuration
        Output: Generated response
        Dependencies: Google Gemini API
        """
        pass

class VectorService:
    def search_similar(query: str, top_k: int) -> List[str]:
        """
        Input: Search query, number of results
        Output: List of similar documents
        Dependencies: ChromaDB
        """
        pass
```

4. Dependency Injection

```python
# FastAPI Dependency Injection
def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()

@router.post("/workflows/")
def create_workflow(
    workflow: WorkflowCreate,
    db: Session = Depends(get_db)  # Injected dependency
):
    return workflow_service.create(db, workflow)
```

5. Configuration Management

```
# Centralized configuration
class Settings(BaseSettings):
    DATABASE_URL: str
    GOOGLE_API_KEY: str
    SECRET_KEY: str

    class Config:
        env_file = ".env"

settings = Settings()  # Single source of truth
```

Workflow Execution Flow

```
┌─────────────────────────────────────────────────────────────────┐
│                  WORKFLOW EXECUTION PIPELINE                    │
└─────────────────────────────────────────────────────────────────┘

1. RECEIVE REQUEST
   POST /api/v1/workflows/execute
   Body: { workflow_id: 1, query: "What is AI?" }
   │
   ▼
2. LOAD WORKFLOW
   workflow_service.get_workflow(workflow_id)
   └─► Fetch from database
       └─► Parse components and connections
   │
   ▼
3. VALIDATE WORKFLOW
   workflow_service.validate_workflow(workflow)
   ├─► Check all components are connected
   ├─► Verify no cycles exist
   └─► Ensure valid configuration
   │
   ▼
4. TOPOLOGICAL SORT
   workflow_service.topological_sort(components, connections)
   └─► Determine execution order
       └─► [User Query → Knowledge Base → LLM → Output]
   │
   ▼
5. EXECUTE COMPONENTS IN ORDER
   │
   ├─► Execute: User Query Component
   │   Input: { query: "What is AI?" }
   │   Output: { query: "What is AI?" }
   │
   ├─► Execute: Knowledge Base Component
   │   Input: { query: "What is AI?" }
   │   Process:
```

```
   │   ├─► vector_service.generate_embeddings(query)
   │   ├─► vector_service.search_similar(embedding, top_k=3)
   │   └─► Retrieve relevant document chunks
   │   Output: { context: "AI is...", sources: [...] }
   │
   ├─► Execute: LLM Engine Component
   │   Input: { query: "What is AI?", context: "AI is..." }
   │   Process:
   │   ├─► llm_service.build_prompt(query, context, config)
   │   ├─► llm_service.generate_response(prompt)
   │   └─► Call Google Gemini API
   │   Output: { response: "Artificial Intelligence is..." }
   │
   └─► Execute: Output Component
       Input: { response: "Artificial Intelligence is..." }
       Output: { formatted_response: "...", metadata: {...} }
   │
   ▼
6. SAVE TO DATABASE
   chat_service.save_message(session_id, message)
   ├─► Save user message
   └─► Save assistant response
   │
   ▼
7. RETURN RESPONSE
   Response: {
     "response": "Artificial Intelligence is...",
     "sources": [...],
     "execution_time": 2.5
   }
```

Database Schema Relationships

```
┌─────────────────────────────────────────────────────────────────┐
│                    DATABASE RELATIONSHIPS                         │
└─────────────────────────────────────────────────────────────────┘

documents                    workflows
┌──────────────┐            ┌──────────────┐
│ id (PK)      │            │ id (PK)      │
│ filename     │            │ name         │
│ file_path    │            │ description  │
│ extracted_   │            │ components   │◄─────┐
│   text       │            │ connections  │      │
│ is_processed │            │ created_at   │      │
│ created_at   │            └──────────────┘      │
└──────────────┘                    │             │
                                    │ 1           │
                                    │             │
                                    │ *           │
                            chat_sessions         │
                            ┌──────────────┐      │
                            │ id (PK)      │      │
                            │ workflow_id  │──────┘
                            │   (FK)       │
                            │ session_name │
                            │ created_at   │
                            └──────────────┘
                                    │ 1
                                    │
                                    │ *
                            chat_messages
                            ┌──────────────┐
                            │ id (PK)      │
                            │ session_id   │
                            │   (FK)       │
                            │ message_type │
                            │ content      │
                            │ metadata     │
                            │ created_at   │
                            └──────────────┘


Relationships:
• One Workflow has Many ChatSessions (1:N)
• One ChatSession has Many ChatMessages (1:N)
• Documents are independent (no foreign keys)
```

Technology Stack Summary

```

┌─────────────────────────────────────────────────────────────────┐
│                      TECHNOLOGY STACK                           │
├─────────────────────────────────────────────────────────────────┤
│ Frontend                                                        │
│  • React 18          - UI framework                             │
│  • Vite              - Build tool                               │
│  • ReactFlow         - Workflow visualization                   │
│  • Axios             - HTTP client                              │
│  • React Router      - Navigation                               │
│  • Lucide React      - Icons                                    │
│  • Nginx             - Web server (production)                  │
├─────────────────────────────────────────────────────────────────┤
│ Backend                                                         │
│  • FastAPI           - Web framework                            │
│  • Python 3.11       - Programming language                     │
│  • SQLAlchemy        - ORM                                      │
│  • Pydantic          - Data validation                          │
│  • Uvicorn           - ASGI server                              │
│  • Alembic           - Database migrations                      │
├─────────────────────────────────────────────────────────────────┤
│ Database & Storage                                              │
│  • PostgreSQL 15     - Relational database                      │
│  • ChromaDB          - Vector database                          │
```

```
├──────────────────────────────────────────────────────────────┤
│ AI & ML                                                       │
│  • Google Gemini      - LLM & Embeddings                     │
│  • PyMuPDF            - PDF text extraction                  │
├──────────────────────────────────────────────────────────────┤
│ DevOps                                                        │
│  • Docker             - Containerization                     │
│  • Docker Compose     - Multi-container orchestration        │
│  • Git                - Version control                      │
└──────────────────────────────────────────────────────────────┘
```

Key Design Patterns

1. **Service Layer Pattern**

- Business logic separated from API endpoints
- Reusable across different endpoints
- Easy to test and maintain

2. **Repository Pattern**

- Data access abstracted through models
- Database operations centralized
- Easy to switch databases

3. **Dependency Injection**

- Loose coupling between components
- Easy to mock for testing
- Flexible configuration

4. **Factory Pattern**

- Node types created dynamically
- Component configuration flexible
- Easy to add new component types

5. **Observer Pattern**

- React state management
- Real-time UI updates
- Event-driven architecture

Extensibility Points

The architecture supports easy extension:

1. **New Component Types**: Add to `CustomNodes.jsx`
2. **New LLM Providers**: Extend `llm_service.py`
3. **New Document Types**: Extend `document_service.py`

4. **New API Endpoints**: Add to `api/endpoints/`
5. **New Database Tables**: Add to `models.py` + Alembic migration

Performance Considerations

- **Async Operations**: FastAPI async endpoints for I/O operations
- **Database Indexing**: Indexes on frequently queried columns
- **Connection Pooling**: SQLAlchemy connection pool
- **Caching**: ChromaDB persistent storage
- **Lazy Loading**: React components loaded on demand
- **Code Splitting**: Vite automatic code splitting