Smarter.Codes Search

Semantic Search Web Application

Introduction

Overview:
- A semantic search application that helps users find relevant content from any website
- Efficiently processes HTML content and returns the most semantically similar matches
- Combines modern web technologies with advanced NLP capabilities

Solution Approach:
- Frontend: React-based SPA with responsive design
- Backend: FastAPI with semantic search capabilities
- Vector Database: Weaviate for efficient similarity search
- NLP: Sentence Transformers for semantic understanding

Frontend Design

UI/UX Implementation:

- Clean, Minimalist Interface
  - Two-field form: URL input and search query
  - Real-time feedback with loading states
  - Error handling with user-friendly messages

React/Next.js Features:

- Component-Based Architecture
  - SearchForm: Handles user input and validation
  - ResultList: Displays search results with relevance scores
  - Responsive design using Tailwind CSS
  - State management with React hooks

User Experience:

- Instant feedback on search progress
- Clear presentation of results with relevance scores
- Mobile-first responsive design
- Accessible and intuitive interface

Backend Logic

FastAPI Framework:
- RESTful API endpoints
- Async request handling
- Input validation with Pydantic
- CORS support for frontend integration

Content Processing Pipeline:

1. HTML Fetching & Parsing
   - BeautifulSoup4 for DOM parsing
   - Intelligent content cleaning
   - Script and style removal
2. Text Processing
   - NLTK for tokenization
   - Smart chunking algorithm (500 tokens per chunk)
   - Content normalization
3. Performance Optimizations
   - Async operations
   - Efficient batch processing
   - Error handling and retries

Vector Database Integration

Weaviate Implementation:

- Schema Design:

```
{
  "class": "HTMLChunk",
  "vectorizer": "none",
  "properties": [
    {"name": "content", "dataType": ["text"]},
    {"name": "url", "dataType": ["string"]},
    {"name": "timestamp", "dataType": ["date"]}
  ]
}
```

Semantic Search Process:

1. Content Vectorization
   - Sentence-BERT embeddings
   - 384-dimensional vectors
   - Batch processing for efficiency
3. Performance Features
   - In-memory vector operations
   - Efficient indexing
   - Fast similarity search

2. Search Optimization
- Vector similarity comparison
- Relevance scoring
- Results ranking

Conclusion

Achievements:
- Successfully implemented semantic search capability
- Scalable and maintainable architecture
- Efficient content processing pipeline
- User-friendly interface

Challenges Faced:
1. Content Processing
   - Handling diverse HTML structures
   - Efficient text chunking
   - Memory management
2. Search Quality
   - Balancing speed vs accuracy
   - Relevance scoring optimization
   - Result ranking refinement

Future Improvements:
- Caching mechanism for frequently searched URLs
- Advanced filtering options
- Multi-language support
- Search result highlighting
- Performance optimizations for large websites