AKTA Phase 1 - Project Specification

Searchable Proposal Archive System

Version: 1.0
Date: May 2025
Author: Jonas Hörter
Status: Draft

---

1. Executive Summary

AKTA (Assistent für Kreierung von TextAnträgen) Phase 1 implements a searchable digital archive for political proposals within RCDS organizations. This system enables members to efficiently search through historical proposals using AI-powered semantic search, eliminating the current challenges of institutional knowledge loss and manual document management.

1.1 Key Objectives

- Create a centralized, searchable repository of all RCDS proposals
- Implement AI-powered search capabilities for finding relevant historical decisions
- Establish automated PDF ingestion for meeting protocols
- Provide a foundation for future proposal management features

1.2 Success Metrics

- Store and index 10+ historical proposals
- Achieve <2 second search response time
- 90%+ search relevance accuracy
- Successfully process 95%+ of uploaded PDFs

---

2. Project Scope

2.1 In Scope

- **Proposal Storage System**: Database for structured proposal data
- **AI Search Engine**: Semantic and full-text search capabilities
- **PDF Processing Pipeline**: Automated extraction from meeting protocols
- **Web Interface**: Search UI with filtering and result display
- **Manual Entry Option**: Form-based proposal input
- **Basic Analytics**: Search usage statistics

2.2 Out of Scope (Future Phases)

- Proposal creation/editing tools
- Live voting functionality
- Meeting management features

- User authentication beyond basic access control
- Mobile applications
- Integration with external systems

---

3. Functional Requirements

3.1 Search Functionality

FR-001: Semantic Search

- **Description**: Users can search using natural language queries
- **Acceptance Criteria**:
  - Returns relevant proposals for conceptual queries (e.g., "Digitalisierung der Lehre")
  - Ranks results by semantic similarity
  - Processes queries in <500ms

FR-002: Advanced Filtering

- **Description**: Filter search results by multiple criteria
- **Filters**:
  - Date range (submitted/decided)
  - Proposal status (passed/rejected/withdrawn)
  - Proposal type (Positionsantrag/Satzungsänderung)
  - Tags/categories
  - Submitting organization

FR-003: Full-Text Search

- **Description**: Traditional keyword search with German language support
- **Features**:
  - German stemming and stop words
  - Phrase search with quotation marks
  - Boolean operators (AND, OR, NOT)

3.2 Data Ingestion

FR-004: PDF Upload and Processing

- **Description**: Extract proposals from PDF meeting protocols
- **Supported Formats**:
  - Native PDF with selectable text
  - Scanned PDFs (via OCR)
- **Output**: Structured proposal data with metadata

FR-005: Manual Proposal Entry

- **Description**: Web form for entering individual proposals
- **Required Fields**:

    - Title
    - Full text
    - Date
    - Status
    - Submitting group

FR-006: Batch Import

- **Description**: Process multiple PDFs simultaneously
- **Features**:
    - Progress tracking
    - Error reporting
    - Partial success handling

3.3 Data Display

FR-007: Search Results Display

- **Description**: Present search results in scannable format
- **Information Shown**:
    - Title with highlighting
    - Proposal number and date
    - Summary (first 200 characters)
    - Status badge
    - Relevance score

FR-008: Proposal Detail View

- **Description**: Full proposal information page
- **Sections**:
    - Complete proposal text
    - Metadata (dates, authors, meeting)
    - Voting results
    - Similar proposals
    - Source document link

3.4 Administration

FR-009: Data Quality Dashboard

- **Description**: Monitor system health and data quality
- **Metrics**:
    - Total proposals indexed
    - Recent uploads
    - Failed processing jobs
    - Search query analytics

---

4. Non-Functional Requirements

4.1 Performance

- **NFR-001**: Search response time <2 seconds for 95% of queries
- **NFR-002**: Support 50 concurrent users
- **NFR-003**: Process 50-page PDF in <30 seconds

4.2 Reliability

- **NFR-004**: 99.5% uptime during business hours
- **NFR-005**: Automated daily backups
- **NFR-006**: Graceful degradation if AI services unavailable

4.3 Security

- **NFR-007**: HTTPS encryption for all communications
- **NFR-008**: Input validation and SQL injection prevention
- **NFR-009**: Rate limiting on API endpoints

4.4 Usability

- **NFR-010**: Mobile-responsive design
- **NFR-011**: Support for modern browsers (Chrome, Firefox, Safari, Edge)
- **NFR-012**: Accessible UI meeting WCAG 2.1 Level AA

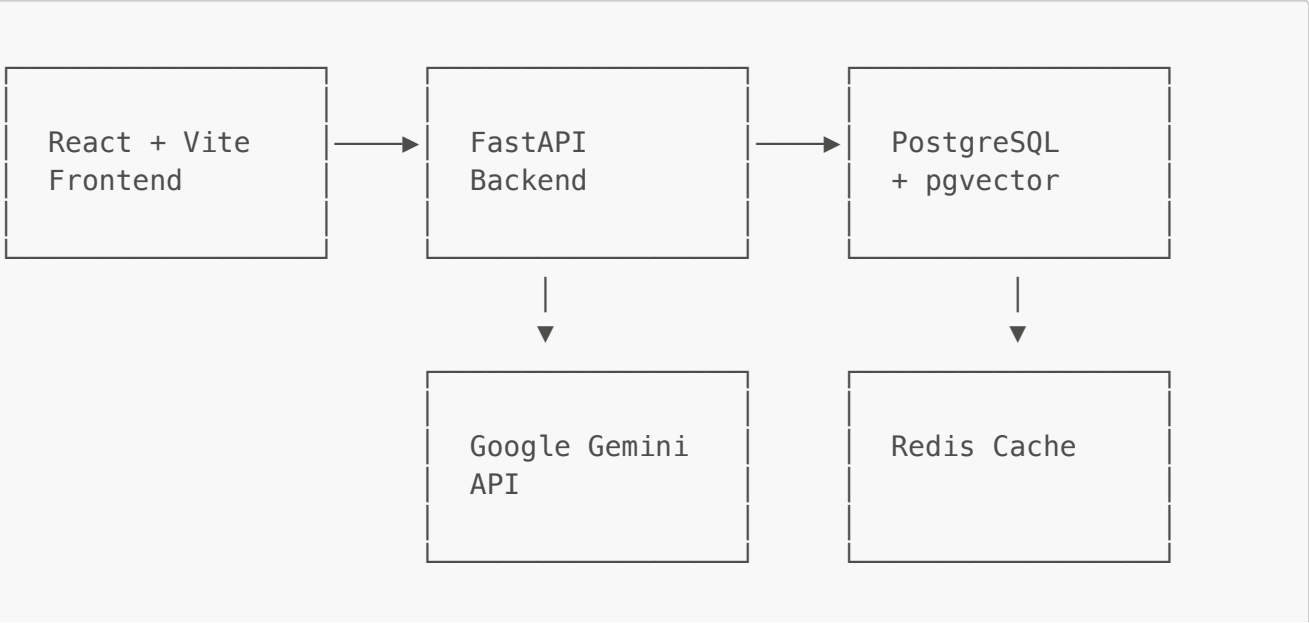
4.5 Scalability

- **NFR-013**: Support up to 50,000 proposals
- **NFR-014**: Horizontal scaling capability for API layer

---

5. Technical Architecture

5.1 System Components



5.2 Technology Stack

Backend

- **Language**: Python 3.11+
- **Framework**: FastAPI
- **Database**: PostgreSQL 16 with pgvector extension
- **Cache**: Redis
- **Task Queue**: Celery with Redis broker
- **PDF Processing**: pdfplumber, pytesseract
- **AI/ML**: Google Gemini API (gemini-pro), langchain

Frontend

- **Build Tool**: Vite 5.x
- **Framework**: React 18.x
- **Language**: JavaScript
- **Styling**: Tailwind CSS + shadcn/ui
- **State Management**: Zustand
- **API Client**: React Query (TanStack Query)
- **Routing**: React Router v6
- **Development**: Vite HMR, ESLint, Prettier

---

6. Data Model

6.1 Core Entities

Proposal

```
proposals
├── id: UUID (PK)
├── title: VARCHAR(500)
├── proposal_number: VARCHAR(50)
├── proposal_type: VARCHAR(100)
├── full_content_text: TEXT
├── full_explanation_text: TEXT
├── summary: TEXT
├── primary_author: VARCHAR(200)
├── co_authors: TEXT[]
├── created_at: TIMESTAMP
└── updated_at: TIMESTAMP
```

---

7. API Specification (tentative - needs to be revisited during actual implementation)

7.1 Search Endpoints

GET /api/v1/search

Search for proposals using various methods.

Query Parameters:

- `q` (string, required): Search query
- `type` (string): Search type - `semantic`, `fulltext`, `hybrid` (default: `hybrid`)
- `limit` (integer): Maximum results (default: 20, max: 100)
- `offset` (integer): Pagination offset (default: 0)
- `status` (string): Filter by status
- `date_from` (date): Filter by submission date
- `date_to` (date): Filter by submission date
- `tags` (array): Filter by tags

Response:

```
{
  "query": "Digitalisierung",
  "type": "hybrid",
  "count": 15,
  "total": 45,
  "results": [
    {
      "id": "123e4567-e89b-12d3-a456-426614174000",
      "title": "Antrag zur Digitalisierung der Hochschullehre",
      "proposal_number": "A-2024-03-15",
      "summary": "Der RCDS fordert...",
      "submitted_date": "2024-03-15",
      "status": "passed",
      "relevance_score": 0.94,
      "tags": ["Digitalisierung", "Hochschulpolitik"]
    }
  ]
}
```

GET /api/v1/proposals/{id}

Retrieve detailed proposal information.

GET /api/v1/proposals/{id}/similar

Find proposals similar to the given proposal.

7.2 Ingestion Endpoints

POST /api/v1/ingest/pdf

Upload and process a PDF document.

Request Body: multipart/form-data

- `file`: PDF file
- `meeting_name`: Meeting name
- `meeting_date`: Meeting date
- `organization`: Organization name

POST /api/v1/ingest/manual

Manually create a proposal entry.

---

8. User Interface Design

8.1 Key Pages

Search Page (/)

- Prominent search bar with autocomplete
- Filter sidebar (collapsible on mobile)
- Results grid with cards
- Pagination controls

Proposal Detail Page (/proposals/{id})

- Full proposal text with formatting
- Metadata sidebar
- Similar proposals section
- Export/print options

Upload Page (/upload)

- Drag-and-drop PDF upload
- Upload progress indicator
- Processing status display
- Error reporting

---

9. Implementation Plan

9.1 Development Phases

Phase 1.1: Foundation

- [ ] Set up development environment
- [ ] Initialize project repositories
- [ ] Configure PostgreSQL with pgvector
- [ ] Implement basic data models

- [ ] Create initial API structure

Phase 1.2: PDF Processing

- [ ] Build PDF text extraction
- [ ] Implement AI proposal parser
- [ ] Create ingestion pipeline
- [ ] Test with sample documents
- [ ] Handle edge cases and errors

Phase 1.3: Search Implementation

- [ ] Implement embedding generation
- [ ] Build semantic search
- [ ] Add full-text search
- [ ] Create hybrid search algorithm
- [ ] Optimize search performance

Phase 1.4: Frontend Development

- [ ] Design UI components
- [ ] Implement search interface
- [ ] Build proposal display
- [ ] Add filtering capabilities
- [ ] Create upload interface

Phase 1.5: Integration & Testing

- [ ] System integration testing
- [ ] Performance optimization
- [ ] User acceptance testing
- [ ] Bug fixes and refinements
- [ ] Documentation completion

---

11. Deployment & Operations

11.1 Deployment Strategy

- **Environment**: Docker containers
- **Hosting**: Self-hosted VPS initially
- **Domain**: akta.rcds-tum.de (example)
- **SSL**: Let's Encrypt certificates

11.2 Monitoring

- **Uptime**: Uptime monitoring
- **Performance**: Response time tracking
- **Errors**: Sentry error tracking

- **Analytics**: Privacy-friendly usage analytics

11.3 Backup Strategy

- **Database**: Daily automated backups
- **Documents**: Versioned storage of PDFs
- **Configuration**: Git-based config management