Resume Matcher - Next.js Project Documentation

Table of Contents

1. Project Overview

Resume Matcher is a Next.js application that uses AI-powered semantic matching to connect job postings with candidate resumes. The system leverages OpenAI's embedding technology and Pinecone's vector database to perform intelligent matching based on skills, experience, and job requirements.

Purpose

- **For Recruiters:** Post job listings and automatically find matching candidates
- **For Candidates:** Upload resumes and get matched with relevant job opportunities
- **For HR Teams:** Streamline the hiring process with AI-powered candidate screening

Key Benefits

- **Semantic Matching:** Goes beyond keyword matching to understand context and meaning
- **Scalable:** Can handle large volumes of resumes and job postings
- **Real-time:** Instant matching results as new content is added
- **User-friendly:** Clean, modern interface with drag-and-drop functionality

2. Architecture & Technology Stack

Frontend

- **Next.js 13.4.0:** React framework with App Router
- **React 18.2.0:** UI library
- **TypeScript:** Type-safe development
- **CSS-in-JS:** Inline styles for component styling

Backend

- **Next.js API Routes:** Server-side API endpoints
- **Formidable:** File upload handling
- **PDF-Parse:** PDF text extraction

AI & Database

- **@xenova/transformers:** Local text embeddings using `all-MiniLM-L6-v2` model (384-dimensional vectors)
- **Pinecone:** Vector database for similarity search
- **Vector Similarity:** 384-dimensional embeddings for semantic matching
- **No OpenAI API or payment required:** Model is downloaded on first use (~90MB)

Development Tools

- **ESLint:** Code linting
- **TypeScript:** Static type checking

3. Project Structure

```
resume-matcher-nextjs-final/
├── app/                          # Next.js App Router
│   ├── layout.tsx               # Root layout component
│   ├── page.tsx                 # Home page with navigation
│   ├── upload-resume/
│   │   └── page.tsx            # Resume upload interface
│   ├── job-listings/
│   │   └── page.tsx            # Job posting interface
│   └── matches/
│       └── page.tsx            # Match results display
├── pages/
│   └── api/                     # API routes
│       ├── resume.ts           # Resume upload & processing
│       ├── job.ts              # Job posting & processing
│       └── match.ts            # Match retrieval & scoring
├── lib/                         # Utility libraries
│   ├── xenova.ts              # Local embedding integration (@
│   └── pinecone.ts            # Pinecone database connection
├── package.json                # Dependencies & scripts
├── next.config.js             # Next.js configuration
└── tsconfig.json              # TypeScript configuration
```

4. Core Components

1. Home Page (`app/page.tsx`)

Purpose: Main navigation hub for the application

Key Features:

- Clean, centered layout with gradient background
- Three main navigation links:
  - Job Listings: Post new job opportunities
  - Matches: View AI-generated matches
  - Upload Resume: Add candidate resumes
- Uses inline CSS with modern design elements

2. Resume Upload (`app/upload-resume/page.tsx`)

Purpose: Interface for uploading and processing candidate resumes

Key Features:

- **Drag & Drop:** Modern file upload with visual feedback
- **PDF Validation:** Ensures only PDF files are accepted
- **Progress Feedback:** Loading states and success/error messages
- **File Preview:** Shows selected file name and size

3. Job Listings (`app/job-listings/page.tsx`)

Purpose: Interface for posting new job opportunities

Key Features:

- **Form Validation:** Ensures both title and description are provided
- **Rich Text Input:** Large textarea for detailed job descriptions
- **Real-time Feedback:** Loading states and success/error messages

4. Matches Display (`app/matches/page.tsx`)

Purpose: Displays AI-generated matches between jobs and resumes

Key Features:

- **Match Scoring:** Shows percentage-based match scores
- **Keyword Highlighting:** Displays relevant keywords from both jobs and resumes
- **Job Details:** Shows job titles and descriptions
- **Responsive Design:** Adapts to different screen sizes

5. API Endpoints

1. Resume Upload (`/api/resume`)

Method: POST
Purpose: Process and store resume files

Process Flow:
File Validation → Text Extraction → Text Enhancement → Embedding Generation → Vector Storage

Key Functions:

- `sanitizeFilename()`: Cleans filename for safe storage
- `enhanceResumeText()`: Prioritizes important resume sections
- Vector creation and storage with metadata

2. Job Posting (`/api/job`)

Method: POST

Purpose: Process and store job postings

Process Flow:
Text Enhancement → Embedding Generation → Vector Storage

3. Match Retrieval (`/api/match`)

Method: GET

Purpose: Retrieve and score matches between jobs and resumes

Process Flow:
Data Retrieval → Similarity Calculation → Keyword Extraction → Result Formatting

6. Data Flow

Resume Upload Flow

User Uploads PDF → File Validation → Text Extraction → Text Enhancement → Embedding Generation → Pinecone Storage

Job Posting Flow

User Submits Job → Text Enhancement → Embedding Generation → Pinecone Storage

Matching Flow

Request Matches → Fetch All Vectors → Calculate Similarity → Extract Keywords → Return Scored Results

7. Key Features

1. Semantic Matching

- **Vector Embeddings:** 384-dimensional vectors capture semantic meaning
- **Context Understanding:** Goes beyond exact keyword matches
- **Similarity Scoring:** Percentage-based match scores

2. Intelligent Text Processing

- **Section Prioritization:** Focuses on skills, experience, and requirements

- **Noise Reduction:** Removes common resume/job noise words
- **Keyword Extraction:** Identifies relevant technical terms

3. User Experience

- **Drag & Drop:** Modern file upload interface
- **Real-time Feedback:** Loading states and progress indicators
- **Responsive Design:** Works on desktop and mobile devices
- **Error Handling:** Graceful error messages and recovery

4. Scalability

- **Vector Database:** Pinecone handles large-scale similarity search
- **Chunked Processing:** Handles large documents efficiently
- **Batch Operations:** Efficient vector storage and retrieval

8. Setup & Configuration

Prerequisites

- Node.js 18+
- npm or yarn
- Pinecone API key and index

Environment Variables

```
PINECONE_API_KEY=your_pinecone_api_key
PINECONE_INDEX=your_pinecone_index_name
USE_MOCK_EMBEDDINGS=false  # Set to true for testing with
```

Installation Steps

1. **Clone Repository**

```
git clone <repository-url>
cd resume-matcher-nextjs-final
```

2. **Install Dependencies**

```
npm install
```

3. **Configure Environment**
    - Create `.env.local` file
    - Add required API keys
4. **Run Development Server**

```
npm run dev
```

5. **Build for Production**

```
npm run build
npm start
```

9. Usage Guide

For Recruiters/HR Teams

1. Post a Job:

1. Navigate to "Job Listings"
2. Enter job title and detailed description
3. Include specific requirements and skills
4. Submit to create job posting

2. View Matches:

1. Navigate to "Matches"
2. View AI-generated candidate matches
3. Review match scores and keywords
4. Identify top candidates for interviews

For Candidates

1. Upload Resume:

1. Navigate to "Upload Resume"
2. Drag and drop PDF resume or click to browse
3. Ensure resume is in PDF format
4. Wait for upload confirmation

2. Get Matched:

- Resumes are automatically matched with job postings
- Higher scores indicate better matches
- Keywords show relevant skills and experience

10. Technical Implementation Details

Vector Embedding Process

1. Text Preprocessing:

- **Text Preprocessing:** (Planned) Remove noise words and common resume/job terms
- **Embedding Generation:** Use @xenova/transformers `all-MiniLM-L6-v2` model (384-dimensional vectors, no API key required)

2. Text Chunking:

- Split large documents into semantic chunks
- Preserve sentence and paragraph boundaries
- Maximum chunk size: 1000 characters

3. Embedding Generation:

- **Embedding Generation:** Creates vector embeddings using local model
- Generate 384-dimensional vectors
- Average multiple chunk embeddings for final vector

Similarity Matching Algorithm

1. Vector Retrieval:

- Fetch all job and resume vectors from Pinecone
- Separate jobs and resumes by ID prefix

2. Similarity Calculation:

- Compute cosine similarity between job and resume vectors
- Score range: 0-1 (0% to 100%)

3. Result Filtering:

- **Result Filtering:** Apply minimum score threshold (50%)
- Sort by similarity score (highest first)
- Include relevant keywords for context

Keyword Extraction

1. Stop Word Removal:

- Comprehensive list of common words to ignore
- Technical and job-specific stop words

2. Technical Term Prioritization:

- Bonus weight for programming languages and technologies
- Focus on skills and tools relevant to job matching

3. Frequency Analysis:

- Count word occurrences with weighting
- Return top 5 most relevant keywords

Error Handling & Fallbacks

1. OpenAI Quota Management:

- **OpenAI Quota Management:** (No longer required, embeddings are local)

2. File Processing:

- PDF validation and error handling
- Safe filename sanitization

3. API Error Handling:

- Comprehensive error messages
- Retry mechanisms for failed requests

Performance Considerations

Optimization Strategies:

- **Text Processing:** Efficient chunking algorithms and noise word filtering
- **Vector Operations:** Batch processing for multiple embeddings
- **Database Operations:** Optimized Pinecone queries with metadata filtering

Scalability Features:

- **Horizontal Scaling:** Stateless API design for multiple instances
- **Database Scaling:** Pinecone handles vector database scaling
- **Caching:** Consider Redis for caching frequent queries

Security Considerations

Data Protection:

- **File Upload Security:** PDF validation, filename sanitization, size limits
- **API Security:** Input validation, error message sanitization, rate limiting

- **Environment Variables:** Secure API key storage

Future Enhancements

Potential Improvements:

- **Advanced Matching:** Multi-modal matching, industry-specific algorithms
- **User Management:** Authentication, company-specific job boards
- **Analytics & Reporting:** Match success tracking, hiring funnel analytics
- **Integration Capabilities:** ATS integration, LinkedIn import, email notifications

Troubleshooting

Common Issues:

- **Upload Failures:** Check file format, size limits, network connectivity
- **API Errors:** Verify environment variables, API key validity, quota limits
- **Match Quality:** Improve job descriptions, ensure relevant keywords in resumes

Debug Mode:

Enable debug logging by setting:

```
DEBUG=true
USE_MOCK_EMBEDDINGS=true  # For testing without embedding
```

Conclusion

The Resume Matcher project now uses local embeddings via @xenova/transformers, removing the need for OpenAI API keys and enabling free, scalable, and private semantic matching.

The modular architecture allows for easy extension and customization, while the comprehensive error handling and fallback mechanisms ensure robust operation in production environments.

Key Takeaways:

- Semantic matching provides more accurate results than traditional keyword matching
- Vector databases enable efficient similarity search at scale
- Modern web technologies create responsive and user-friendly interfaces
- Proper error handling and fallbacks ensure system reliability

Resume Matcher Documentation
Generated on: 20/07/2025
Version: 1.0.0