Resume Checker Application - Project Documentation

📋 Table of Contents

---

1. Tech Stack & Algorithms

🔧 Backend Technologies

Technology	Purpose
Python 3.13	Core programming language
Django 6.0	Web framework
Django REST Framework	API development
SQLite/PostgreSQL	Database
OpenAI GPT API	Resume analysis & NLP
Sentence Transformers	Text embeddings
ChromaDB	Vector database for semantic search
PyPDF2 & python-docx	Resume parsing
openpyxl	Excel export functionality

🎨 Frontend Technologies

Technology	Purpose
React 19	UI framework
Material UI (MUI) 7	Component library
Axios	HTTP client
React Router 7	Navigation
Recharts	Data visualization
jsPDF	PDF generation
js-cookie	Authentication tokens

🧠 Algorithms Used

1. Cosine Similarity Algorithm

```
Similarity = (A · B) / (||A|| × ||B||)
```

Used for calculating semantic similarity between resume and job description embeddings.

2. Weighted Scoring Algorithm

```
final_score = (
    hard_skills × 0.30 +
    soft_skills × 0.15 +
    experience × 0.25 +
    education × 0.15 +
    semantic_similarity × 0.15
)
```

3. TF-IDF Based Skill Extraction

- Extracts keywords from resumes and job descriptions
- Matches skills using NLP techniques

4. Vector Embedding Search

- Converts text to 384-dimensional vectors
- Uses ChromaDB for efficient similarity search

---

2. Features

👤 Authentication System

- ☑ User registration with email validation
- ☑ Password strength validation (8+ chars, uppercase, lowercase, number, special char)
- ☑ Show/Hide password toggle
- ☑ Role-based access (Student / Placement Team)
- ☑ Existing user detection during registration
- ☑ Secure login with proper error messages
- ☑ JWT Token-based authentication

📄 Resume Management

- ☑ Upload resumes (PDF, DOCX)
- ☑ Automatic text extraction
- ☑ Resume storage with Cloudinary

- ☑ View uploaded resumes
- ☑ Delete resumes

💼 Job Management (Placement Team)

- ☑ Create job postings
- ☑ Upload job descriptions
- ☑ Set positions required
- ☑ View all job postings
- ☑ Edit/Delete jobs

📊 AI-Powered Evaluation

- ☑ LLM-based resume analysis using OpenAI GPT
- ☑ Semantic similarity scoring
- ☑ Skill matching (matched & missing skills)
- ☑ Multi-dimensional scoring:
  - Hard Skills Score
  - Soft Skills Score
  - Experience Score
  - Education Score
- ☑ Recommendations generation
- ☑ Strengths & Areas for improvement

📈 Candidate Shortlisting

- ☑ View matched candidates per job
- ☑ First Round Shortlisting feature
- ☑ Export to Excel (All / Matched 50%+ / Shortlist)
- ☑ Candidate ranking by score

🧮 Dashboard

- ☑ Student dashboard with evaluation results
- ☑ Placement team dashboard with analytics
- ☑ Visual charts using Recharts

---

3. Most Difficult Part & Solution

🔴 The Challenge: Silent Registration/Login Failures

Problem Description: When users attempted to register or login with incorrect credentials, the application displayed a generic "Registration failed" message or no error at all - just a silent failure with the form appearing to do nothing.

Why It Was Difficult:

1. **Multiple layers to debug:**

```
React Component → AuthContext → API Service → Axios → Django Backend
```

2. **No visible errors:**
   - UI showed nothing
   - Backend never received requests
   - No console errors appeared
3. **Root cause was hidden:**
   - The issue was **CORS (Cross-Origin Resource Sharing)**
   - Frontend ran on `localhost:3001` or `localhost:3002`
   - Backend only allowed `localhost:3000`

☑ The Solution

Step 1: Diagnosed the Request Flow

```
User clicks Login → React → AuthContext → API Service
                                               ↓
                                     ✖ BLOCKED BY CORS
                                               ↓
                                    Django (never received request)
```

Step 2: Found the Configuration Issue

```
# BEFORE (in settings.py)
CORS_ALLOWED_ORIGINS = [
    "http://localhost:3000",  # Only this port allowed!
]
```

Step 3: Applied the Fix

```
# AFTER (in settings.py)
CORS_ALLOWED_ORIGINS = [
    "http://localhost:3000",
    "http://localhost:3001",
    "http://localhost:3002",
    "http://127.0.0.1:3000",
    "http://127.0.0.1:3001",
    "http://127.0.0.1:3002",
]
```

Step 4: Enhanced Error Handling

```
// AuthContext.js - Improved error extraction
const login = async (credentials) => {
  try {
    Cookies.remove('authToken'); // Clear stale sessions
    const response = await authService.login(credentials);
    return { success: true };
  } catch (error) {
    const errorMessage = error.response?.data?.error
                      || error.message
                      || 'Login failed';
    return { success: false, error: errorMessage };
  }
};
```

🎯 Key Lesson Learned

> When debugging silent failures in full-stack apps, **always check CORS first** if the backend never receives requests. Browsers block cross-origin requests silently.

---

4. System Architecture & Flowchart

High-Level Architecture

```
┌──────────────────────────────────────────────────────────────────────┐
│                           FRONTEND (React)                           │
│  ┌─────────────┐  ┌─────────────┐  ┌─────────────┐  ┌────────────┐  │
│  │   Login/    │  │   Resume    │  │    Jobs     │  │ Dashboard  │  │
│  │  Register   │  │   Upload    │  │   Listing   │  │ & Charts   │  │
│  └──────┬──────┘  └──────┬──────┘  └──────┬──────┘  └─────┬──────┘  │
│         │                │                │               │          │
│         └────────────────┴────────────────┴───────────────┘          │
│                                   │                                   │
│                                   ▼                                   │
│                          ┌───────────────┐                           │
│                          │  AuthContext  │                           │
│                          │  API Services │                           │
│                          └───────┬───────┘                           │
└──────────────────────────────────┼───────────────────────────────────┘
                                   │ HTTP/REST API
                                   │ (Axios)
┌──────────────────────────────────┼───────────────────────────────────┐
│                           BACKEND (Django)                           │
│                          ┌───────▼───────┐                           │
│                          │  Django REST  │                           │
│                          │   Framework   │                           │
│                          └───────┬───────┘                           │
│         ┌────────────────────────┼────────────────────────┐          │
│         │                        │                        │          │
```

```
  │  ┌──────▼──────┐  ┌──────────────▼──────────────┐  ┌─────▼──────┐  │
  │  │    Auth     │  │        Evaluations          │  │    Jobs    │  │
  │  │   Module    │  │  ┌─────────────────────┐    │  │   Module   │  │
  │  │             │  │  │   LLM Services      │    │  │            │  │
  │  │  - Login    │  │  │  ┌───────────────┐  │    │  │ - Create   │  │
  │  │  - Register │  │  │  │ OpenAI GPT    │  │    │  │ - List     │  │
  │  │  - Profile  │  │  │  │ Analysis      │  │    │  │ - Export   │  │
  │  └──────┬──────┘  │  │  └───────────────┘  │    │  └─────┬──────┘  │
  │         │         │  │  ┌───────────────┐  │    │        │         │
  │         │         │  │  │ Embedding     │  │    │        │         │
  │         │         │  │  │ Service       │  │    │        │         │
  │         │         │  │  │ (ChromaDB)    │  │    │        │         │
  │         │         │  │  └───────────────┘  │    │        │         │
  │         │         │  └─────────────────────┘    │        │         │
  │         │         └──────────────┬──────────────┘        │         │
  │         │                        │                       │         │
  │         └────────────────────────┼───────────────────────┘         │
  │                                  │                                  │
  │                          ┌───────▼───────┐                          │
  │                          │   Database    │                          │
  │                          │   (SQLite/    │                          │
  │                          │  PostgreSQL)  │                          │
  │                          └───────────────┘                          │
  └────────────────────────────────────────────────────────────────────┘
```

Evaluation Process Flowchart

```
  ┌─────────────────┐
  │ Student Uploads │
  │     Resume      │
  └────────┬────────┘
           │
           ▼
  ┌─────────────────┐
  │  Extract Text   │
  │  (PDF/DOCX)     │
  └────────┬────────┘
           │
           ▼
  ┌─────────────────┐     ┌─────────────────┐
  │  Get Job        │────▶│ Extract Job     │
  │  Description    │     │ Requirements    │
  └────────┬────────┘     └────────┬────────┘
           │                       │
           └───────────┬───────────┘
                       │
                       ▼
           ┌───────────────────────┐
           │   LLM Analysis        │
           │   (OpenAI GPT)        │
           │                       │
```

```
         │  - Skill Matching     │
         │  - Experience Check   │
         │  - Education Match    │
         │  - Generate Feedback  │
         └───────────┬───────────┘
                     │
                     ▼
         ┌───────────────────────┐
         │  Embedding Service    │
         │  (Sentence Transform) │
         │                       │
         │  - Generate Vectors   │
         │  - Cosine Similarity  │
         │  - Store in ChromaDB  │
         └───────────┬───────────┘
                     │
                     ▼
         ┌───────────────────────┐
         │  Calculate Final      │
         │  Weighted Score       │
         │                       │
         │  Hard Skills: 30%     │
         │  Experience:  25%     │
         │  Soft Skills: 15%     │
         │  Education:   15%     │
         │  Semantic:    15%     │
         └───────────┬───────────┘
                     │
                     ▼
         ┌───────────────────────┐
         │  Store Evaluation     │
         │  Results in Database  │
         └───────────┬───────────┘
                     │
                     ▼
         ┌───────────────────────┐
         │  Return Results to    │
         │  Frontend Dashboard   │
         └───────────────────────┘
```

---

5. Efficient Code Samples

5.1 LLM-Enhanced Evaluation Service (Most Efficient)

This code combines multiple AI services for comprehensive resume analysis:

```python
# llm_services.py - EnhancedScoringService

class EnhancedScoringService:
    """
    Combines LLM analysis with semantic similarity for accurate scoring.
    This is the core algorithm that powers the resume evaluation.
    """

    def __init__(self):
        self.llm_service = LLMService()
        self.embedding_service = EmbeddingService()

    def comprehensive_evaluation(self, resume_text: str, job_description: str) ->
AnalysisResult:
        """
        Perform comprehensive evaluation combining:
        1. LLM-based skill and experience analysis
        2. Semantic similarity using embeddings
        3. Weighted scoring for final result
        """
        # Step 1: Get LLM analysis
        llm_result = self.llm_service.analyze_resume(resume_text, job_description)

        if not llm_result:
            return self._fallback_analysis(resume_text, job_description)

        # Step 2: Calculate semantic similarity
        semantic_score = self.embedding_service.calculate_semantic_similarity(
            resume_text, job_description
        )

        # Step 3: Apply weighted scoring
        final_score = self._calculate_weighted_score(
            llm_result.hard_skills_score,
            llm_result.soft_skills_score,
            llm_result.experience_score,
            llm_result.education_score,
            int(semantic_score * 100)
        )

        llm_result.overall_score = final_score
        llm_result.semantic_similarity_score = semantic_score

        return llm_result
```

Why This Is Efficient:

- Combines multiple AI techniques in a single pipeline
- Has fallback mechanism if LLM fails
- Uses caching through ChromaDB for embeddings
- Modular design allows easy testing and maintenance

5.2 Semantic Similarity with Cosine Distance

```
# llm_services.py - EmbeddingService

class EmbeddingService:
    def __init__(self):
        self.model = SentenceTransformer('all-MiniLM-L6-v2')  # 384-dim vectors

    def calculate_semantic_similarity(self, text1: str, text2: str) -> float:
        """
        Calculate semantic similarity using cosine similarity.

        Efficiency: Uses pre-trained transformer model that generates
        embeddings in milliseconds. Cosine similarity is O(n) where
        n is the embedding dimension (384).
        """
        embedding1 = self.model.encode(text1)
        embedding2 = self.model.encode(text2)

        # Cosine similarity formula: (A·B) / (||A|| × ||B||)
        similarity = cosine_similarity([embedding1], [embedding2])[0][0]

        return float(similarity)
```

Complexity Analysis:

- Embedding generation: O(sequence_length)
- Cosine similarity: O(embedding_dimension) = O(384)
- Total: Linear time complexity

5.3 Efficient Excel Export with Streaming

```
# jobs/views.py - export_job_candidates_excel

@api_view(['GET'])
@permission_classes([permissions.IsAuthenticated])
def export_job_candidates_excel(request, pk):
    """
    Export candidates to Excel with optimized query and streaming.
    """
    job = get_object_or_404(JobDescription, pk=pk)

    # Efficient query with select_related to avoid N+1 problem
    evaluations = Evaluation.objects.filter(
        job_description=job
    ).select_related(
        'resume',
        'resume__user'
    ).order_by('-overall_score')

    # Apply filters
    export_type = request.query_params.get('type', 'matched')
    min_score = int(request.query_params.get('min_score', 0))
    limit = request.query_params.get('limit')

    if min_score > 0:
        evaluations = evaluations.filter(overall_score__gte=min_score)

    if limit:
        evaluations = evaluations[:int(limit)]

    # Create Excel with openpyxl (memory efficient)
    wb = openpyxl.Workbook()
    ws = wb.active

    # Stream to response
    output = io.BytesIO()
    wb.save(output)
    output.seek(0)

    response = HttpResponse(
        output.read(),
        content_type='application/vnd.openxmlformats-
officedocument.spreadsheetml.sheet'
    )
    return response
```

Efficiency Features:

- `select_related()` prevents N+1 database queries
- Streaming response for large files
- BytesIO for memory-efficient file handling

5.4 Authentication with Secure Error Handling

```
# authentication/views.py - LoginView

class LoginView(ObtainAuthToken):
    def post(self, request, *args, **kwargs):
        username = request.data.get('username')
        password = request.data.get('password')
        login_type = request.data.get('login_type')

        # Input validation
        if not username or not password:
            return Response({
                'error': 'Both username and password are required',
            }, status=status.HTTP_400_BAD_REQUEST)

        # Check user existence
        try:
            user_exists = CustomUser.objects.get(username=username)
        except CustomUser.DoesNotExist:
            return Response({
                'error': 'User does not exist. Please register.',
                'field': 'username'
            }, status=status.HTTP_400_BAD_REQUEST)

        # Authenticate
        user = authenticate(username=username, password=password)
        if user is None:
            return Response({
                'error': 'Password is incorrect.',
                'field': 'password'
            }, status=status.HTTP_400_BAD_REQUEST)

        # Role validation
        if login_type == 'student' and user.role != 'student':
            return Response({
                'error': 'This is not a student account.',
                'field': 'role'
            }, status=status.HTTP_400_BAD_REQUEST)

        # Success - generate token
        token, _ = Token.objects.get_or_create(user=user)
        return Response({
            'user': UserSerializer(user).data,
            'token': token.key
        })
```

Security Features:

- Field-specific error messages for UX
- Role-based access validation
- Token-based authentication

5.5 Frontend API Interceptor with Auto-Refresh

```
// services/api.js

const api = axios.create({
  baseURL: 'http://127.0.0.1:8000/api',
  headers: { 'Content-Type': 'application/json' },
});

// Request interceptor - adds auth token automatically
api.interceptors.request.use(
  (config) => {
    const token = Cookies.get('authToken');
    if (token) {
      config.headers.Authorization = `Token ${token}`;
    }
    return config;
  },
  (error) => Promise.reject(error)
);

// Response interceptor - handles token expiration
api.interceptors.response.use(
  (response) => response,
  (error) => {
    if (error.response?.status === 401) {
      Cookies.remove('authToken');
      window.location.href = '/login';
    }
    return Promise.reject(error);
  }
);
```

Efficiency Features:

- Automatic token injection on every request
- Centralized error handling
- Auto-redirect on session expiry

📑 Summary

Aspect	Details
Architecture	Full-stack with React + Django REST API
AI/ML	OpenAI GPT + Sentence Transformers + ChromaDB
Key Algorithm	Weighted scoring with cosine similarity
Hardest Challenge	CORS configuration causing silent failures
Most Efficient Code	EnhancedScoringService combining LLM + Embeddings

---

Document Generated: January 27, 2026 Resume Checker Application v1.0