Career Profile Builder Project Report

Formatting Specification (to apply in Word processor)

1. Margins: top 1", bottom 1", left 1", right 0.7"
2. Paper size: A4
3. Font: Times New Roman
4. Abstract: one page, normal, size 14, line spacing 1.5, justify
5. Contents: include labels with page numbers
6. Chapter title: size 14, line spacing 1.15, justify
7. Sub title: size 12, line spacing 1.15, justify
8. Chapter heading: bold, size 14, line spacing 1.5, center
9. Section heading: bold, size 12, line spacing 1.5, left
10. Sub-section heading: bold, size 12, line spacing 1.5, left
11. Body content: normal, size 12, line spacing 1.5, justify
12. Figure/Table labels: bold, size 10, line spacing 1.5, center, chapter-based numbering
13. References: size 12, line spacing 1.15, justify
14. Main content target: at least 10 pages

Abstract (Page i)

Career Profile Builder is a full-stack web application developed to automate resume parsing, evaluate resume quality, and generate reusable professional profile outputs. The system addresses common challenges faced by students and job seekers: manually rewriting resume content for different platforms, identifying weak resume sections, and maintaining consistent profile information across CV documents, GitHub README pages, and LinkedIn summaries. The project combines document ingestion, text extraction, rule-based natural language processing, score-based health assessment, and authenticated user workflows in one integrated platform.

The backend is implemented using Django and Django REST Framework. It supports secure JWT-based authentication, resume upload, text parsing, structured data extraction, and profile export generation. Users can upload resume files in PDF or Word format, and the system extracts contact information, skills, education, experience, and projects. A resume health module computes strengths, warnings, and practical suggestions by scoring completeness and quality indicators such as quantified impact statements. Parsed data is stored per authenticated user, enabling resume history and incremental improvements.

The frontend is built with React, TypeScript, Vite, and Tailwind CSS. It provides account registration/login, resume upload and preview, result visualization, profile dashboard, parsed data editing, and export tools. The latest enhancement includes direct editing of current extracted information, automatic regeneration of CV output, and export features for generated CV markdown, GitHub README markdown, and LinkedIn-ready JSON. This reduces repeated manual work and improves user control over final output quality.

From a software engineering perspective, the project demonstrates modular service design, API-driven architecture, maintainable type-safe client integration, and practical UX workflows for document-processing applications. It can be extended with OCR, AI-assisted rewriting, ATS keyword optimization, and

template-based PDF generation. Overall, Career Profile Builder offers a practical and scalable foundation for career document management and profile automation.

Table of Contents (Page ii)

List of Figures (Page iii)

List of Tables (Page iv)

Chapter 1: Introduction

1.1 Background

Preparing career documents is repetitive and error-prone. A single person typically maintains multiple versions of resume content for job applications, LinkedIn profile summaries, project portfolios, and public repositories. Manual copy-paste leads to inconsistencies, outdated details, and low productivity. Recruiters and ATS systems also expect concise, keyword-rich, measurable experience statements. This project was developed to solve these practical issues through automated parsing and profile generation.

1.2 Problem Statement

Users need a system that can:

- Accept common resume files.
- Extract relevant information reliably.
- Evaluate quality and identify improvement areas.
- Reuse extracted content across multiple output formats.
- Allow users to edit extracted content and regenerate outputs without re-uploading the original file.

Existing tools often provide either parsing only or template output only. They rarely provide an integrated workflow that combines parsing, quality analysis, editing, and multi-platform exports in one authenticated application.

1.3 Objectives

Primary objectives of Career Profile Builder are:

- Build a full-stack system for resume parsing and profile export.
- Provide secure user-specific resume storage.
- Generate meaningful resume health feedback.
- Produce platform-oriented exports (GitHub README and LinkedIn profile JSON).
- Provide editable parsed data and CV regeneration/export.
- Maintain clean modular architecture for extension and maintenance.

1.4 Scope and Limitations

Scope includes user authentication, resume upload, structured extraction, health scoring, result view, profile history, edit/regenerate workflows, and text-based exports. Current limitations include:

- Parsing quality depends on input resume structure.
- OCR for scanned image resumes is not yet integrated.
- CV export is currently markdown text; PDF template rendering is future work.
- Automated NLP is rule-based and can be improved with ML/LLM enhancements.

Chapter 2: Requirement Analysis

2.1 Functional Requirements

Table (2.1) Functional Requirements Matrix

ID	Requirement	Priority	Status
FR-01	User registration and login	High	Implemented
FR-02	JWT access/refresh authentication	High	Implemented
FR-03	Resume upload (.pdf/.doc/.docx)	High	Implemented
FR-04	Text extraction and preprocessing	High	Implemented
FR-05	Structured parsing (contact, skills, projects, education, experience)	High	Implemented
FR-06	Resume health scoring and feedback	High	Implemented
FR-07	Resume list and detail retrieval	High	Implemented
FR-08	Export generation (README, LinkedIn, CV markdown)	High	Implemented
FR-09	Edit current parsed data and save	High	Implemented
FR-10	Regenerate and export updated CV	High	Implemented

2.2 Non-Functional Requirements

Table (2.2) Non-Functional Requirements Matrix

ID	Requirement	Target
NFR-01	Security	Token-based protected API routes
NFR-02	Usability	Clear upload-result-edit-export flow
NFR-03	Performance	Responsive API and UI for normal resume sizes
NFR-04	Maintainability	Modular service-based backend architecture
NFR-05	Portability	Local development via Python + Node toolchains
NFR-06	Reliability	Validation for file type/size and API errors

2.3 User Roles and Use Cases

Primary user role is authenticated end user (job seeker/student/professional). Main use cases:

- Sign up and sign in.
- Upload resume and parse.
- Review extracted sections and health score.
- Edit extracted details if parser misses/incorrectly interprets data.
- Regenerate outputs.
- Export generated CV/README/LinkedIn payload.

Chapter 3: System Design

3.1 Overall Architecture

Figure (3.1) High-Level System Architecture Client (React) communicates with Django REST APIs over HTTP. Authentication uses JWT. Resume files are processed by backend services; parsed outputs and metadata are persisted in SQLite for development. Exports are generated from parsed JSON and returned to the frontend for preview/download.

3.2 Backend Design

Backend components:

- Django project configuration (`cpb_api`).
- Parser app (`parser`) with models, serializers, views, and services.
- Service layer for extraction, parsing, scoring, and profile export.

Table (3.1) Core Backend Modules

Module	Responsibility
`extract_text.py`	Extract plain text from PDF/DOC/DOCX
`preprocess.py`	Normalize text into processable lines
`build_output.py`	Parse normalized lines into structured fields
`resume_health.py`	Score resume quality and create suggestions
`profile_export.py`	Build CV markdown, README markdown, LinkedIn JSON
`resume_workflow.py`	Coordinate upload processing and export generation

3.3 Frontend Design

Frontend is organized by routes and reusable API helpers:

- Authentication pages (register/login).
- Upload page with file validation and preview.
- Result page with parsed sections, score panel, export panel.
- Profile page listing historical uploads.

State is handled through React hooks. API calls are centralized in `src/lib/api.ts` with timeout, token injection, and auto-refresh logic.

3.4 Data Model and API Design

Main model: `Resume`

- `user`, `file_name`, `raw_text`, `parsed_data`, `resume_health`, `is_confirmed`, timestamps.

Table (3.2) API Endpoint Summary

Method	Endpoint	Purpose
POST	`/api/register/`	Create user account
POST	`/api/auth/login/`	Obtain JWT tokens
POST	`/api/auth/refresh/`	Refresh access token
POST	`/api/parse-resume/`	Upload and parse resume
GET	`/api/resumes/`	List user resumes
GET	`/api/resumes/<id>/`	Get one resume
PATCH	`/api/resumes/<id>/edit/`	Update parsed info and health
GET	`/api/resumes/<id>/exports/`	Generate exports from parsed data

Figure (3.2) Resume Processing Pipeline Upload -> Validation -> Text Extraction -> Preprocess -> Section Parsing -> Health Scoring -> Persist -> Export Generation -> Frontend Display.

Chapter 4: Implementation

4.1 Resume Parsing Workflow

The parsing workflow receives multipart uploads, validates extension and size, extracts text based on file type, preprocesses lines, and maps content into structured sections. This data structure is designed for direct rendering and downstream export generation.

4.2 Resume Health Scoring

Health scoring evaluates profile completeness and impact quality. Criteria include:

- Presence of professional email.
- Number and distribution of skills.
- Presence of education and experience.
- Quantified achievements using numeric-pattern detection.
- Presence of professional links.

Figure (4.1) Resume Health Scoring Flow Structured profile input -> Criterion checks -> Score aggregation -> strengths/warnings/suggestions output.

4.3 Export Generation

Exports are generated from the same `parsed_data` source:

- `cv_markdown`: concise CV-ready markdown.
- `github_readme`: repository profile style summary.
- `linkedin_profile`: structured JSON for LinkedIn section filling.

This approach enforces consistency: one source of truth, multiple platform views.

4.4 Edit-and-Regenerate CV Feature

A key enhancement is the new edit flow:

1. Load existing `parsed_data` into editable JSON in Result page.
2. User modifies extracted information.
3. Frontend submits `PATCH /api/resumes/<id>/edit/`.
4. Backend validates JSON and recalculates `resume_health`.
5. Frontend requests `/api/resumes/<id>/exports/`.
6. Updated CV/README/LinkedIn outputs are shown and downloadable.

Figure (4.2) Edit and Regenerate CV Workflow User edit -> Save -> Server validation -> Health recompute -> Export regeneration -> Download.

Implementation highlights:

- Added `updateResume()` API helper in frontend.
- Added `cv_markdown` field in export payload type.
- Added download helpers for markdown and JSON.
- Updated backend serializer update behavior to auto-rescore data.
- Updated resume edit endpoint to return full updated resume object.

Chapter 5: Testing and Validation

5.1 Test Strategy

Testing combines API-level checks, frontend build validation, manual scenario walkthroughs, and data validation checks.

5.2 Manual and Automated Checks

Table (5.1) Testing Checklist and Results

Test Case	Expected Result	Outcome
Upload valid PDF resume	Parsed result saved and displayed	Pass
Upload unsupported format	Validation error message	Pass
Access protected endpoint without token	Unauthorized response	Pass

Test Case	Expected Result	Outcome
View profile resume history	List sorted by upload date	Pass
Generate exports	README + LinkedIn + CV markdown returned	Pass
Edit parsed data and save	Data updated and score recalculated	Pass
Download generated CV	Markdown file downloaded	Pass
Build frontend project	Production bundle created	Pass

5.3 Results and Observations

Observed strengths:

- Stable, modular backend service design.
- Good separation of concerns between parsing, scoring, and export generation.
- Fast UI iteration due to typed API and route-based pages.

Observed risks:

- Parsing quality varies by resume formatting styles.
- Large or highly graphical resumes may reduce extraction quality.
- Current output quality depends on rule coverage rather than semantic ML models.

Chapter 6: Deployment and Maintenance

6.1 Environment Setup

Backend setup:

- Create and activate virtual environment.
- Install dependencies using `pip install -r requirements.txt`.
- Apply migrations.
- Run Django server.

Frontend setup:

- Install Node dependencies via `npm install`.
- Run development server using `npm run dev`.

6.2 Dependency Management

Backend dependency list is maintained in `backend/requirements.txt` and includes Django, DRF, JWT auth library, and document parsing libraries.

Recommended maintenance practices:

- Pin minimum stable versions in production branches.
- Run periodic dependency audit.
- Document upgrade and migration checklist.

6.3 Security and Operational Considerations

Security controls:

- JWT-authenticated endpoints.
- User-scoped queryset filters to prevent cross-user resume access.
- File type/size validation before processing.

Operational considerations:

- Add structured logging and request IDs.
- Add CI pipeline with unit tests and lint checks.
- Use PostgreSQL and object storage in production.
- Introduce rate limiting and upload scanning for hardened deployment.

Chapter 7: Conclusion and Future Enhancements

7.1 Conclusion

Career Profile Builder successfully integrates resume ingestion, parsing, quality analysis, editable profile management, and multi-platform export generation in a single full-stack application. The solution reduces repetitive profile writing effort, improves consistency, and gives users actionable guidance to improve resume quality. The recently added edit-and-regenerate capability addresses practical user needs by allowing iterative profile refinement without re-uploading documents.

7.2 Future Work

Planned enhancements:

- OCR integration for scanned resumes.
- AI-assisted rewriting for bullets and summaries.
- ATS keyword gap analysis by job description.
- Styled CV template generator (PDF/Docx exports).
- Versioned history and diff view for resume edits.
- Admin analytics dashboard for usage and quality metrics.

References

1. Django Software Foundation. Django Documentation. https://docs.djangoproject.com/
2. Django REST Framework. Official Documentation. https://www.django-rest-framework.org/
3. Simple JWT for DRF. Documentation. https://django-rest-framework-simplejwt.readthedocs.io/
4. React Documentation. https://react.dev/
5. Vite Documentation. https://vite.dev/
6. Tailwind CSS Documentation. https://tailwindcss.com/docs
7. pdfplumber Documentation. https://github.com/jsvine/pdfplumber
8. python-docx Documentation. https://python-docx.readthedocs.io/
9. Pressman, R. S. Software Engineering: A Practitioner’s Approach. McGraw-Hill.
10. Sommerville, I. Software Engineering. Pearson.

Appendix A: Suggested Page Distribution (for 10+ main pages)

- Chapter 1: 1.5 pages
- Chapter 2: 1.5 pages
- Chapter 3: 2 pages
- Chapter 4: 2 pages
- Chapter 5: 1.5 pages
- Chapter 6: 1 page
- Chapter 7: 1 page Total main content: approximately 10.5 pages (excluding abstract, contents, lists, references)