Automatic Profile Generation

Andrew Allen
David Ratliff
Suqian Wang

MIDTERM REPORT

REVISION – 0.1
4 October 2018

Table of Contents

1 Execution Plan

Milestones are listed under the date they will be completed. Current builds of each submodule will be presented and distributed to Shell POC on dates marked 'Deliverable'.

Date	Data Extraction and Profile Generation	Data Storage and Management	Search Engine and User Interface
10/11	NLP Research		UI design and mockup
10/18	Implement data scraping module, testing on predetermined test sources, profile generation algorithm functional breakdown	Azure DB service setup	Search UI implementation
10/25 (Deliverable)	Test scraping module on real sources, profile generation algorithm implementation 50%	Relation setup, JDBC setup, populate with dummy data	Elasticsearch setup, search API integration
11/1	Incorporate scraping module sponsor feedback, profile generation algorithm 100% (verify using team members' data)	Incorporate sponsor feedback, integrate with front-end	Incorporate sponsor feedback, DB integration
11/8 (Deliverable)	Verify profile generation algorithm with company data	Implement search filters, fuzzy search, profile relevance rankings	
11/15	Incorporate sponsor feedback, integrate profile generation/scraping	Incorporate sponsor feedback, clean up UI/UX	
11/22 (Deliverable)	Verify profile generation algorithm		

	with public figures' data	Implement manual profile-modifying UI, login system	
11/29	Incorporate sponsor feedback, documentation final pass	Incorporate sponsor feedback, documentation final pass	
12/6	Demo & Report	<=	<=

Automatic Profile Generation

Andrew Allen
David Ratliff
Suqian Wang

CONCEPT OF OPERATIONS

REVISION – 0.1
4 October 2018

# Concept of Operations
for
Automatic Profile Generation

Team Shellfish

Approved by:

____
Project Leader   Date

____
Prof. S. Kalafatis   Date

____
T/A   Date

Change Record

Rev.	Date	Originator	Approvals	Description
0.0	09/25/2018	Suqian Wang		Draft Release
0.1	10/4/2018	David Ratliff		Midterm Draft

List of Tables

List of Figures

2 Executive Summary

The company sponsor, Shell, has requested improvements to its internal talent-finding system, Shell Expertise Profiles. The current system relies on employees to voluntarily complete their own profiles. As a result, many profiles are incomplete and the system is ineffective. The improved system must extract relevant data (skills/experience) from various structured and unstructured sources and automatically generate employee profiles. It must also provide search functionality which ranks employee profiles based on the user's desired qualifications. The project utilizes data scraping and Natural Language Processing techniques to acquire the data sources and extract employee information. The user interface and search tool is developed with the proven and mature technologies, Django and Elasticsearch. The goal of this project is to increase the efficiency of matching experts with projects by reducing reliance on word-of-mouth recommendations.

3 Introduction

Shell has a large number of employees with a wide range of skills which are constantly changing. They also have a large number of complex problems for their employees to solve. Pairing the capable with the appropriate challenges is no small feat. Engineering Asset and Project teams have a constant need to find the right person with the right skills to assist with projects, and at certain times solve very specific problems. The objective of this project is to increase the efficiency of finding relevant staff, which increases asset production through better resolution of issues and quality of work.

3.1 Background

The profile generation tool improves the Shell Expertise Profiles platform. Shell Engineering division has a formal list of experts, but the taxonomy does not cover all niches or projects. People are encouraged to capture their skills and expertise in the Shell Expertise Profiles platform, but even that is not covering all of their experience and skills (especially with people moving around).

3.2 Overview

While there is a current tool in place, it relies on employees self-reporting their skills. The large amount of unstructured information in Shell's possession already has the data required to automatically generate and update the experts' profiles. Here lies the focus of our solution. In automatically querying these sources, Shell can adapt with the business's needs and deploy their resources efficiently and in a useful manner. The expected outcomes provide clear benefits: reduce the amount of time needed to fully document all expertise in Expertise Profile, increased visibility of skills of all staff, faster allocation of staff to solve problems, and improved execution of tasks (faster/better quality) due to having right staff on activities.

4 Operating Concept

4.1 Scope

The profile generation system proposed in this document is designed for Shell project leaders to look for employee profiles. These profiles allow employers to find employees with specific expertise and skills. The employer shall be able to review an employee's profile by simply searching his/her name. The employer shall also be able to find a list of experts (ranked by relevance) by searching for certain requirements.

4.2 Operational Description and Constraints

4.2.1 Operational Description

To operate the system, a user must open a web browser and navigate to a designated search page. The specific domain and host of this website has yet to be determined. The user enters parameters corresponding to the expert needed. The search returns a list of experts along with the link to their profiles matching the query that the user typed into the search bar.

The UI is implemented using the Django web framework. The search functionality is implemented using an open-source enterprise search application, Elasticsearch. A database, using Microsoft Azure DB services, is implemented and managed to organized the profiles and any relations between objects. Experts are able to enhance profiles manually, while the system periodically updates them with data from new sources. Data extraction techniques are used in order to acquire unstructured from different files and web sources. NLP algorithms are used to extract the profile fields from the unstructured data.

4.2.2 Operational Constraints

The system only generates profiles from certain predetermined data sources. Other data sources, and sources emerging in the future will have to be incorporated manually into the system.

4.3 System Description

The profile generation system is divided into three key subsystems: the Data Extraction and Profile Generation, Data Storage and Management, and the Search Engine and User interface. The Data Extraction and Profile Generation subsystem is responsible for taking in all forms of data and extracting all important information related to a person's profile. The subsystem then takes the stream of data and formats the data in a form to be processed and kept in the Data Storage and Management subsystem. The Data Storage and Management subsystem is responsible for creating relations for the data and storing the personnel profiles into the database and keeping the data organized and accessible for the User Interface and Search Engine. The User Interface and Search Engine subsystem is in charge allowing the user to search the database for information designated by the user and return information to the user.

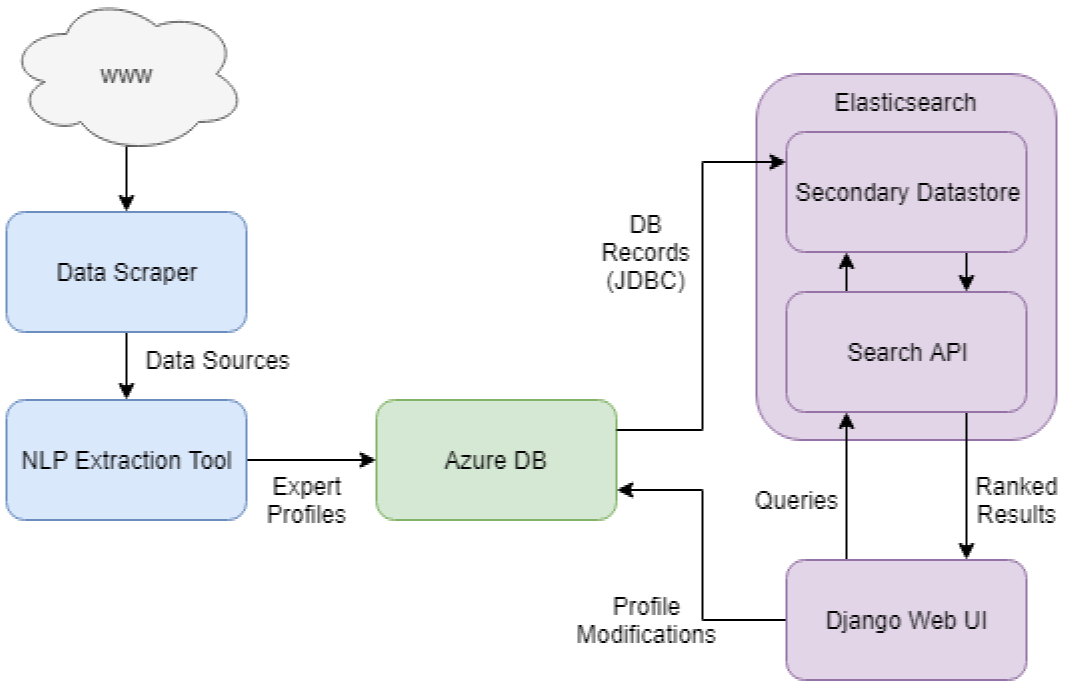


Figure 1: System Description Block Diagram

4.3.1 Data Extraction and Profile Generation

An open source Python library, Scrapy, is used as a web scraper. Its function is to acquire data from the web-based data sources utilized by the system. NLP (natural language processing) is used to acquire useful data from unstructured data files. Profiles for each personnel are formatted using all the acquired sources.

Several Python libraries (NLTK, spaCy) provide different modules to the project to aid in processing text, classifying, tokenizing, and parsing. Several algorithms provided by these libraries are utilized by this project: Summarizer algorithm, AutoTag algorithm, Named Entity Recognition algorithm, PorterStemmer algorithm, and Tokenizer algorithm.

Tokenizer is used first to break up text into words. Porter Stemmer reduces words to their root word, which benefits us by giving users more accurate search results. Summarizer extracts the most important and central ideas from a body of text. It is helpful when extracting useful information from files that have much irrelevant information. AutoTag extracts keywords contained within a body of text; these tags are useful for our keywords search feature in the user interface. Named Entity

Recognition identifies those named within the unstructured data, which allows mapping between other data sources and an individual's profile.

4.3.2 Data Storage and Management

Microsoft Azure Cosmos DB is used to store our profiles. This database scales and replicates data easily, and supports both NoSQL and SQL APIs. As the employee profiles easily adhere to a relational model, this application utilizes the SQL API.

4.3.3 Search Engine and User Interface

Elasticsearch is the search engine for our database. It is responsible for gathering data from the database and accurately displaying that information to the user. Elasticsearch is a distributed, RESTful search and analytics engine. This search engine provides scalability, resilience, and flexibility to the system. Elasticsearch handles any type of search quickly, is reliable, and has APIs in Python for easy integration.

Django web framework serves web pages to users, handles user inputs, and simplifies communication of queries and search results. The webpage is constructed using Python, HTML, CSS, and JavaScript.

4.4 Modes of Operation

Active

The system is active when a user makes a search using the system. The user inputs a request to the database and the system outputs a list of matching experts, ranked by relevance.

Idle

The system is in idle when there is no input being received from the user. In this mode, the profile generator waits for users to input data for it to complete.

Automated update

The system is in this mode when the profile generator periodically acquires new raw data from the employer's company to be processed and updated into the database.

4.5 Users

The targeted user of the profile generation system is for employers looking for candidates for certain tasks and jobs. Basic computer literacy is all that is required of users to operate the system. The user interface will allow the system to be intuitive and easy to navigate for employers using this system. Software developers will be maintaining the system. Command line arguments on how to run applications will be given in the user manual. If changes need to be made for the system or the system is to be used for integration, whoever performing these operations should be proficient in Python in addition to having experience with NLP.

4.6 Support

Instructions on how to set up, run, and stop the system will be included in the user manual. The user manual will go over instructions on how to operate different functions of the system and help troubleshoot different problems within the software. Documentation in the code provides the purpose of each function along with its input and output. Each subsystem contains a README file as a detailed description.

5 Scenarios

5.1 Find Expert

When assembling the team for a project, or during the course of the project, managers may need to contact an expert in a certain field. To accomplish this, the manager simply enters the desired qualifications in the search engine, and checks the first few resulting profiles to see who they would like to speak to. The system ensures the completeness and relevance of these profiles.

5.2 Update Profile

Employees may be inclined to include additional information in their profile that the automatic tool cannot generate (ex. biographies, interests). The employee can simply log in to the system, and manually enter information in a manner similar to how Shell Expertise Profiles currently works.

6 Analysis

6.1 Summary of Proposed Improvements

- Automatically generates employee expertise profiles
- Allows manual enhancement of an individual's profile
- Provides easy search functionality, returning list of experts adhering to user's queries, ranked by relevance
- Multiple search and filter options

6.2 Disadvantages and Limitations

- Unable to automatically generate biographies and other complex profile information
- Unable to incorporate subjective traits into profiles (e.g. personality, soft skills)

6.3 Alternatives

- Word-of-mouth
    - De-facto current method of expert-finding at Shell
    - Not an engineered system, so requires no development or maintenance costs
    - Incorporates personal biases and anecdotal information into professional recommendations
    - Reduces visibility of experts more than a few degrees outside one's professional network
- Shell Expertise Profiles
    - De-jure current method of expert-finding at Shell
    - Already implemented, requires only minor upkeep
    - Relies on employees to voluntarily update profiles, resulting in incomplete profiles

Automatic Profile Generation

Andrew Allen
David Ratliff
Suqian Wang

Functional System Requirements

REVISION – 0.1
4 October 2018

FUNCTIONAL SYSTEM REQUIREMENTS
FOR
Automatic Profile Generation

APPROVED BY:

Project Leader	Date
Prof. S. Kalafatis	Date
T/A	Date

Change Record

Rev.	Date	Originator	Approvals	Description
0.0	09/25/2018	Suqian Wang		Draft Release
0.1	10/4/2018	David Ratliff		Midterm Draft

List of Tables

List of Figures

7 Introduction

7.1 Purpose and Scope

This document covers the detailed system requirements for the project. The purpose of this project is to improve upon and replace the Shell Expertise Profiles system currently used by Shell to locate internal experts. The improved system will increase operational efficiency by improving expert visibility and expert/project matching. The system shall automatically generate employee profiles using various structured and unstructured data sources. The system shall include search functionality so users can query the profiles and find a suitable expert. The system shall rank search results according to relevance to the user's query. The system shall use Azure DB services for its primary datastore. The automatically generated profiles should be at least as complete and useful as those presently in the Shell Expertise Profiles system. The system should automatically acquire and incorporate new data sources into employee profiles on a regular basis.

This specification defines the technical requirements for the development items and support subsystems delivered to the client for the project. Figure 1 shows a representative integration of the project in the proposed CONOPS. The verification requirements for the project are contained in a separate Verification and Validation Plan.

7.2 Responsibility and Change Authority

Changes to performance requirements can be made only by Shell, or by the consensus of all team members with the consent of Shell. Changes to implementation requirements can be made by the consensus of the team. The team leader, David Ratliff, is responsible for the fulfillment of all project requirements. Subsystem owners are also responsible for the fulfillment of their subsystem's requirements. Subsystem ownership is as follows:

- David: Data Extraction and Profile Generation
- Suqian: Data Storage and Management
- Andrew: Search Engine and User Interface

8 Applicable and Reference Documents

8.1 Applicable Documents

Reference Appendix C

8.2 Reference Documents

Reference Appendix C

8.3 Order of Precedence

In the event of a conflict between the text of this specification and an applicable document cited herein, the text of this specification takes precedence without any exceptions.

All specifications, standards, exhibits, drawings or other documents that are invoked as "applicable" in this specification are incorporated as cited. All documents that are referred to within an applicable document are considered to be for guidance and information only, with the exception of ICDs that have their applicable documents considered to be incorporated as cited.

9 Requirements

This section defines the minimum requirements that the development item must meet. The requirements and constraints that apply to performance, design, interoperability, reliability, etc., of the system are covered.

9.1 System Definition

The profile generation system is divided into three key subsystems: the Data Extraction and Profile Generation, Data Storage and Management, and the Search Engine and User interface. The Data Extraction and Profile Generation subsystem is responsible for abstracting useful information from the various unstructured data sources. The Data Storage and Management subsystem is responsible for creating an instance of a secure database and inserting profile entries into the database indexed for search. The User Interface and Search Engine subsystem focuses on allowing functional, full-text search on the profiles in the database. The user interface allows employees to manage their profile, search for other personnel, and search for other employees based on desired skills or other qualifications.

When a search is made, a request is sent to the search engine which queries the database and returns the result to the client in a functional UI. The database is populated with auto-generated profiles for all requested personnel. A single profile is generated by extracting useful information from the unstructured data sources which have been scraped from the web or internal documents.

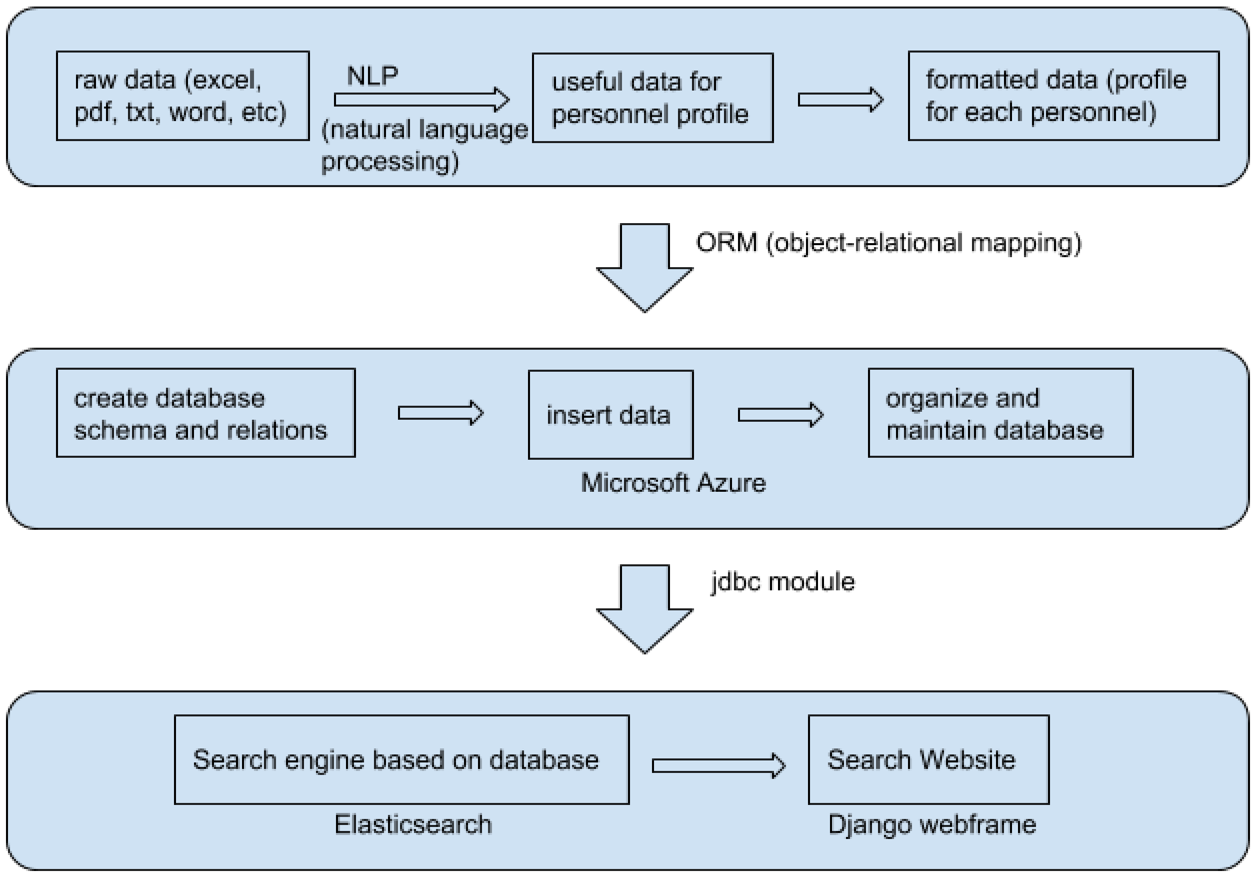

raw data (excel, pdf, txt, word, etc)
NLP
(natural language processing)
useful data for personnel profile
formatted data (profile for each personnel)
ORM (object-relational mapping)
create database schema and relations
insert data
organize and maintain database
Microsoft Azure
jdbc module
Search engine based on database
Elasticsearch
Search Website
Django webframe

Figure 2: Profile Generation System Functional Block Diagram

9.1.1 Data Extraction and Profile Generation

9.1.1.1 Web Data Extraction

A Python library, Scrapy, is used to scrape information from potentially useful data sources (eg. LinkedIn). Python is used to parse text from the various formats provided: reports, forum text, internal documentation, etc.

9.1.1.2 Natural Language Processing

To obtain relevant information about the employees, NLP is used to analyze the the text from the previous stage. NLTK, a Python library, is used in processing, classifying, and tokenizing the extracted text. Upon success, the now structured data is sent to the database subsystem for storage, possible modification, and retrieval.

9.1.2 Data Storage and Management

A secure, reliable, and compatible method for storing employees profiles is a necessity. Microsoft's Azure database services are leveraged so that securing, scaling, and transitioning infrastructure to the production environment will be a smooth and relatively painless process. Upon storing profiles on the database, connecting the database to Elasticsearch via a JDBC driver allows for search indexing.

9.1.3 Search Engine and User Interface

9.1.3.1 Elasticsearch

Elasticsearch is the main search engine for our database. It is responsible for indexing and allowing functional full-text search across all profiles and data entries. This search engine provides scalability, resilience, and flexibility. The query request is processed and returned for the Django backend to serve to the client.

9.1.3.2 Django Server

The Django web framework serves web pages to users, handles user inputs, and simplifies communication of queries and search results. It communicates with users via the built-in Django framework network protocols. The Django backend receives requests from the client and return results for the user.

9.2 Characteristics

9.2.1 Functional / Performance Requirements

9.2.1.1 Data Size Requirement

The Profile Generation System should be able to process 60 - 100 employees' data and generate corresponding profiles for each employee. It should be capable of scaling to a large number of employees per Shell's use.

> *Rationale: According to the company, we will be given 60 - 100 employees' data to work with. However, we are hoping our system could be used for the entire enterprise in the future.*

9.2.1.2 Profile Generation Speed

The Profile Generation System should be able to generate result based on the query user input into the system within 120s.

> *Rationale: The time for systems to interact with each other is non-negligible. Specific time will be measure in the future. 120s is just an estimate.*

9.2.1.3 Accuracy

The Profile Generation system should generate no more than 20% false data suggestions.

Rationale: NLP algorithms have an inherent margin of error, which increases when the sample size of the input data is small. The scope of data supplied by the company sponsor limits the algorithm's accuracy. However, the since the algorithm will operate in the production environment by suggesting profile modifications and will not act without a user's consent, user verification easily filters this incorrect data.

9.2.1.4 Completeness

The Profile Generation system results in at least 60% completeness in the following profile sections:

- Current Role
- Location
- Business Unit
- Skills
- Previous Projects
- Certifications/Recognitions

9.2.2 Software Requirements

This section specifies the runtime environment requirements of the project.

9.2.2.1 Installation

This project is built and run on an virtual machine instance of Ubuntu Server 18.04 running on Azure. Operating system compatibility should be in place so long as proper versions of all Python packages and other software are installed correctly. To install the prerequisites reference README.md on the main branch of our GitHub repository.

9.2.2.2 Python Packages and Environments

Specifications with versions of all Python modules are in requirements.txt. To install the prerequisites reference README.md on the main branch of our GitHub repository. NLTK and Scrapy in particular are heavily used so the same version is highly advised. If wanting to run locally, set up a virtual Python 3 environment in order to ensure the modules installed are the versions described in requirements.txt.

9.2.3 Communication Requirements

9.2.3.1 Database to Search System

Elasticsearch's SQL JDBC driver is used to connect with the database on Azure. According to documentation, "it is a platform independent, stand-alone, Direct to Database, pure Java driver that converts JDBC calls to Elasticsearch SQL". The JDBC driver queries the data source and updates values and indexes within.

9.2.3.2 Search System to and from Django Backend

Django forwards queries to the Elasticsearch API. Results from a given query are then posted to the frontend by Django.

9.2.3.3 Django Backend to and from Frontend

The search functionality on the client-side sends an HTTP GET request to the server. Django receives this request and returns results by posting its response via the same protocol.

9.2.4 Failure Propagation

By subscribing to safe coding practices, any exceptions are caught within the scripts of origin and throw errors to be read by the user if the system is unable to recover gracefully. All data remains safe on the database and if a system crashes a simple restart should place a user in a functioning environment again. System failures result in exceptions which will be caught and analyzed.

Automatic Profile Generation

Andrew Allen
David Ratliff
Suqian Wang

INTERFACE CONTROL DOCUMENT

REVISION – 0.1
4 October 2018

INTERFACE CONTROL DOCUMENT
FOR
Automatic Profile Generation

APPROVED BY:

Project Leader	Date
Prof. Stavros Kalafatis	Date
T/A	Date

Change Record

Rev.	Date	Originator	Approvals	Description
0.0	09/25/2018	Suqian Wang		Draft Release
0.1	10/4/2018	David Ratliff		Midterm Draft

List of Tables

List of Figures

10 Overview

This document describes the interfaces between the subsystems of the project. There are three interface sections, one for each subsystem of the project.

11 Data Extraction and Profile Generation

This module acquires and processes data sources relevant for profile generation. It then assembles employee profiles and submits them to the database.

11.1 WWW-Data Extraction

Data sources are acquired from the WWW and company sources using the Python library Scrapy. More documentation on the Scrapy API is contained in the *Scrapy Documentation* reference.

11.2 Profile Generation-Data Storage

The NLP module assembles an object-oriented representation of an employee's profile. These profile objects are converted to relational form and submitted to the database through the Django ORM interface, described in detail in the *Django Documentation* reference. The fields of the profile object are given in the list below:

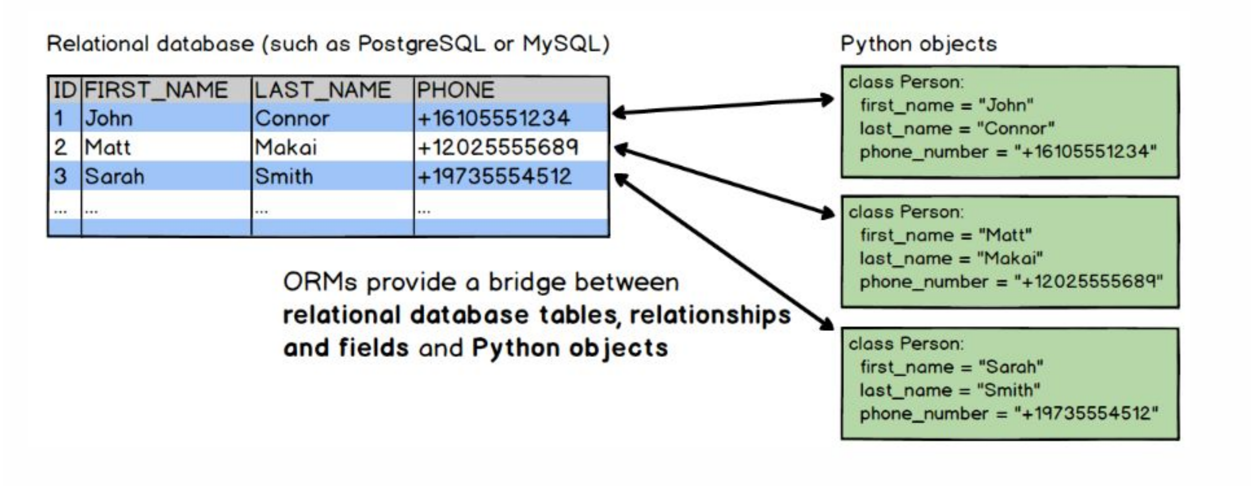
- Name
- Email
- Phone #
- Discipline
- Shell Business
- Business Unit
- Department
- Region
- Current Role
- Job Title
- Skills
- Bio
- Previous Projects
- Certifications/Recognitions
- Interests
- Profile Completeness (%)

12 Data Storage and Management

This module accepts profiles from the Data Extraction and Profile Generation module, and stores them for later search indexing. It also acts as the single source-of-truth DB for the Elasticsearch secondary datastore.

12.1 ORM Interface

The Django ORM interface connects this module to both the Profile Generation submodule, and the front-end UI submodule. In both these cases, the ORM is used to transfer object-oriented profiles from Python to the database (new profiles for the former, manually-modified profiles for the latter). The ORM maps the profile's fields to a relational format suitable for use in a RDBMS, and automatically generates the SQL query used to insert the profile into the relation. More information on the Django ORM API can be found in the *Django Documentation* reference.



Relational database (such as PostgreSQL or MySQL)
Python objects
ORMs provide a bridge between **relational database tables, relationships and fields** and **Python objects**

ID	FIRST_NAME	LAST_NAME	PHONE
1	John	Connor	+16105551234
2	Matt	Makai	+12025555689
3	Sarah	Smith	+19735554512
...	...	...	...

```
class Person:
  first_name = "John"
  last_name = "Connor"
  phone_number = "+16105551234"
```

```
class Person:
  first_name = "Matt"
  last_name = "Makai"
  phone_number = "+12025555689"
```

```
class Person:
  first_name = "Sarah"
  last_name = "Smith"
  phone_number = "+19735554512"
```

Figure 3: Python-ORM Interaction

12.2 JDBC Interface

The project's search tool, Elasticsearch, operates by indexing its own internal datastore for search, which is updated in real-time from a single source-of-truth database (this module). This module supplies updated data through the JDBC interface, a Java API prescribing how client Java programs can access databases. More information on the JDBC interface can be found in the *JDBC Documentation* reference.

13 Search Engine and User Interface

13.1 Database Querying

Elasticsearch uses the JDBC interface to query the indexed text for useful and relevant results. Django processes the HTTP GET requests from the client and then formats the text requests from the user in such a way that Elasticsearch has the best odds of finding relevant results. Reference the *Elasticsearch Documentation* for more details on the search module used.

13.2 User Interface

The user interface consists of two main pages. The search page has a simple search bar, which queries for profiles based upon user defined qualifications. Upon receiving the results the user may sort on various relevance metrics. Users may also login and edit the information on their profile. This improves the profiles' completeness in areas outside the project scope (eg. bio sections), and allows for a better self-portrayal. Reference the *Django Documentation* for more details on the web framework used to construct the UI.

Appendix A: Verification Plan

This section details specific tasks the system must pass to be considered valid.

A.1 Data Extraction and Profile Generation

A.1.1 Data Extraction Verification

- Successfully connects to predetermined web-based sources:
    - Authored reports (Shell library)
    - Forum activity
    - Sharepoint activity
    - Linkedin profile
    - More to be determined during course of development
- Successfully extracts and logs plain-text data above sources
- Correctly indexes and logs data sources already scraped
- Does not attempt to re-scrape old data

A.1.2 Profile Generation Verification

- Results in > 60% completeness in following profile fields:
    - Current Role
    - Location
    - Business Unit
    - Skills
    - Previous projects
    - Certifications/Recognitions
- Less than 20% false data suggestions
- Pipeline length < 120s from scraping end to DB submission
- DB insertion success/failure message logged

A.2 Data Storage and Management

This module is trivial to verify, as database actions always occur when interfacing with the other subsystems. Successful operation in conjunction with the Profile Generation module and the UI is verified through the success/failure flags included with the ORM. The interfaced modules simply log their interactions with the DB using these flags. Correct operation alongside the Elasticsearch secondary datastore is verified by checking that data is received by the search module in a complete and proper format (already a test for the search engine module).

A.3 Search Engine and UI

A.3.1 UI Verification

Feature	**Test**
Login	Successful login redirects to employee's profile
	Login for nonexistent profile redirects to profile creation
	Incorrect credentials yield error message, stays on same page
Profile Modification	All fields modifiable with 'Edit' button
	Modified fields successfully post to DB
	Modified profiles indexed by search within 5 minutes
Search Engine	Correct categorization of search terms by topic
	Search filters narrow result span
	Misspelled queries yield search suggestions
Search Results	Search terms return additional results with related terms
	Name search returns exclusively results by name
	Correct # of results count
	Correct results/page displayed
	Query, filters retained from page to page
	Relevance rankings adhere to human-predicted outcomes within 10%
Help Button	Redirects to github getting started page
Result Card	Displays expert name
	Displays expert title
	Displays business unit
	Displays matched query terms, in bold
UI	Proper scaling on devices

	Proper scaling with window resize
Cross-browser	Site functionality identical across Chrome, Firefox, and Edge browsers

A.3.2 Interface Verification

Interface	**Test**
Elasticsearch/DB	Connection failure results in nonblocking error message (user can still search)
	Profile content received properly formatted
	New profiles indexed for search < 1 minute
Django/DB	Connection failure results in error message blocking profile edits
	DB insertion success/failure messages returned and logged

Appendix B: References Documents

Doc #	Revision/Release Date	Document Title
1	3.7.1	Python Standard Library
2	1.5	Scrapy Documentation
3	3.3	NLTK documentation
4	2.1	Django documentation
5	6.4	Elasticsearch documentation
6	January 1st, 2018	ORM explanation
7	January 8th, 2018	Azure SQL Libraries for Python
8	August 24th, 2018	Shell business case

Appendix C: Acronyms & Abbreviations

API	Application Programming Interface
ConOps	Concept Of Operation
CSS	Cascading Style Sheet
DB	Database
FSR	Functional System Requirement
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
ICD	Interface Control Document
JDBC	Java Database Connectivity
NLP	Natural Language Processing
NLTK	Natural Language ToolKit
NoSQL	Not Only Structured Query Language
ORM	Object Relational Mapping
pdf	Portable Document Format
RDBMS	Relational Database Management System
REST	Representational State Transfer
SQL	Structured Query Language
txt	Plain Text
UI/UX	User Interface/User Experience
WWW	World Wide Web (public internet)

Appendix D: Definitions

Application Programming Interface	An interface used by programmers to access functionality of external tools and other programs
Cascading Style Sheet	Cascading Style Sheets (CSS) is a style sheet language used for describing the presentation of a document written in a markup language like HTML.
Database	A database is an organized collection of data, stored and accessed electronically.
Django	Django is a free and open-source web framework, written in Python, which follows the model-view-template (MVT) architectural pattern.
Elasticsearch	Elasticsearch is an enterprise, open-source search engine based on Apache Lucene
Excel	Microsoft Excel is a spreadsheet developed by Microsoft
GitHub	A website that hosts repositories (managed file systems often used for developing software)
HyperText Transfer Protocol	The standard web API used today. Includes four main methods for browsers and other network-connected applications to interact with web-based resources: GET, POST, PUT, DELETE
Hypertext Markup Language	Hypertext Markup Language (HTML) is the standard markup language for creating web pages and web applications.
JavaScript	JavaScript (JS) is a high-level, interpreted programming language.
Java Database Connectivity	An API used to prescribe how a Java client application interacts with databases.
Microsoft Azure	Microsoft Azure is a cloud computing service created by Microsoft for building, testing, deploying, and managing applications and services through a global network of Microsoft-managed data centers.
Natural Language Processing	Natural language processing (NLP) is an area of computer science and artificial intelligence concerned with the interactions between computers and human (natural) languages

Natural Language ToolKit	The Natural Language Toolkit is a suite of libraries and programs for symbolic and statistical natural language processing (NLP) for English written in the Python programming language.
Not Only Structured Query Language	A NoSQL database provides a mechanism for storage and retrieval of data that is modeled in means other than the tabular relations used in relational databases.
Object Relational Mapping	Object-relational mapping is a programming technique for converting data between incompatible type systems using object-oriented programming languages.
Python	Python is an interpreted high-level programming language for general-purpose programming.
Relational Database Management System	A “traditional” table-based database. Records adhere to a strict “schema” defining what data types they contain.
RESTful	An architectural style used to define HTTP-based web services
Scrapy	A Python library used to scrape data from web-based data sources
Structured Query Language	SQL is a domain-specific language used in programming and designed for managing data held in a relational database management system(RDBMS), or for stream processing in a relational data stream management system (RDSMS).
Ubuntu	A distribution of the popular Linux OS
Word	Microsoft Word is a word processor developed by Microsoft.