CS5200 Database Management System Project Report (Summer 2014)  
CareerMedley

Group members:

Chi Zhang

ID: 001957029 Email: [czhang79@ccs.neu.edu](mailto:czhang79@ccs.neu.edu)  
  
Onyeka Igabari   
ID: Email: Onyeka.igabari@gmail.com

Summary

Problem to solve

Our goal in this project is to build database management system to well-organize our job applications and facilitate our job search. In current job market, we normally apply to multiple positions online in order to get an ideal intern/coop/job. The number of applications could range from tens to several hundreds. A typical naïve way to organize so many applications is: create a folder for each company and create a sub folder for each position in the company, then you put your resume/cv, login id/password, email address to that specific subfolder. It’s easy to get lost in many folders. What’s worse, after several weeks or months, you’ll have no idea what you did before. And it’s really embarrassing that sometimes you receive an interview invitation email, but you don’t remember when and where you apply it.

The important difference between our job application system and existing job application services provided by indeed/monster is: this is for user-oriented. And the emphasis is put on the job that the user is interested, the effort the user puts on the position, and related info aggregation. E.g. if the user plans to apply the data scientist positions, the system can list the core skills that most data scientist positions require. In terms of info aggregation, for instance, the applicant may search for a job on Monster.com, read reviews about that job on indeed.com and find out more about the company and salaries ranges on glassdoor.com. The applicant may also get more details about the company's stocks and news from Bloomberg or yahoo finance, etc. Won't it be nice to have a one-stop site to have all these info?

Flask – a Python web development framework

Flask introduction

Flask is an open source **micro** web development framework for Python based on Werkzeung, Jinjia2 and other components. Flask has built-in development server and debugger, integrated unit testing support, it’s 100% WSGI1.0 compliant (Web Server Gateway Interface) and Unicode based. The **micro** means Flask aims to keep the core simple but extensible. Flask won’t make many decisions for you, such as what database to use. Those decisions that it does make, such as what template engine to use, are easy to change. Everything else is up to you. By default, Flask doesn’t include a database abstraction layer, form validation or anything else where different libraries already exist that can handle that. Instead, Flask supports extensions to add such functionality to your application as if it was implemented in Flask itself. Numerous extensions provide database integration, form validation, uploading handling, various open authentication technologies, and more. Flask may be “micro”, but it’s ready for production use on a variety of needs.

In terms of Flask’s major components, Werkzeung is a Python WSGI Utility Library. Jinjia2 is a modern and designer friendly template language for Python, modeled after Django (another popular Python web development framework)’s template. It’s fast, widely used and secure with optional sandboxed template execution environment.

Further info about Flask can be found on it’s official website:  
<flask.pocoo.org>

SQLAlchemy – The database toolkit for Python

SQLAlchemy introduction

In our project, SQLAlchemy is chosen as the database abstraction layer, which is the bridge that Flask uses to talk to backend database.

SQLAlchemy is the Python SQL toolkit and Object Relational Mapper that gives application developers the full power and flexibility of SQL. It provides a full suite of well-known enterprise-level persistence patterns, designed for efficient and high-performing database access, adapted into a simple and Pythonic domain language.

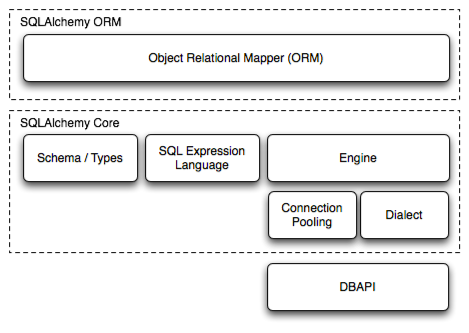


Fig 1. SQLAlchemy internal structure

One tip in installation: after SQLAlchemy is installed, Flask wrapper extension Flask-SQLAlchemy can be installed to provide an friendlier programming API for developers.

**How to solve**in order to build this system, we should aggregate information from several aspects:

1. Job positions info

- Info about the position itself:   
Title, location, job description, dept. or division, required skill sets, sponsorship, clearance, benefit (401k etc.), deadline, application link.

(To get the latest job postings, there are two options: extract job positions from indeed/monster and populate our database with them with provided APIs; manually input the job position)  
  
- Info of applicant for this position:

Customized resume, cover letter, status (interested/applied/rejected/withdrawn), id and password pair, referrer provided by user.

1. The company info: category (internet/mobile/software/hardware), main product, size, stock price, ratings, related news (such as hiring or layoff news), reviews, possible interview questions. (To get the info about the company: retrieve info from Glassdoor, Bloomberg, CrunchBase, etc.)

**What we want to deliver (tentative)**

* Architecture: web-based client/server database management system
* Services:
* The user can view the history of his/her applications, such as which material has been submitted and which are not (E.g. resume submitted/cover letter missing etc.), what phrase now(E.g. pass phone screen/second round interview/onsite etc.), the incoming deadlines ranked by date
* For a specific position, the user can view its related aggregated info as described above: title, location, deadline,
* The overall data analysis: what is the skill set required in those jobs that the user is interested? Which language is most wanted among those jobs? (E.g. the swift programming language recently released by Apple?) How may open positions of those jobs now?

**Strategy**The project implementation is broken into several steps:

1. **Investigation**:   
   In this stage, we would start by analyzing different web technologies such as Ruby on Rails, Flask (Python) etc. to figure out which would be easier and faster to use.
2. **API Analysis:**   
   In this stage, we would investigate the API options we have from different job sites and the possibility of obtaining data from these sites using the APIs.
3. **Prototype Development:**   
   - In this stage, we would design the database that is needed for our application and write a little application to test the API functionality.- Try to figure out how much data we should store in our database and how much data we should get and display dynamically.   
   - Develop a little test application for proof of concept. We would be using Northeastern Husky career service's website as a form of template for our development.
4. **Detail Implementation and Test Cycles:**   
   In this phase we would be continuing the design process towards a full product and fixing issues as we go. We will be using agile process of development to help keep us on track.

References:  
indeed: <http://www.indeed.com/>  
monster: <http://www.monster.com/>  
simplyhired: <http://www.simplyhired.com/>  
Glassdoor: <http://www.glassdoor.com/index.htm>  
Ruby on Rails: <http://rubyonrails.org/>  
Python Flask web framework: <http://flask.pocoo.org/>