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: 001150568 Email: [Onyeka.igabari@gmail.com](mailto:Onyeka.igabari@gmail.com)

Introduction

**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?

**Information Required**

In order to build this system, we should collect 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 on the employer’s website. (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:

Resumes, CVs, status (interested/applied/rejected/withdrawn), id and password pair for that position, referrer provided by user.

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

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

* Architecture:   
  A web-based client/server database management system that be hosted on a public clouding computing platform, such as Amazon AWS, Red Hat OpenShift, Salesforce Heroku.
* 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 info, such as: title, location, deadline, and short description of the job, etc.

For a specific employer, the user can view its related info: number of reviews, ratings by its employees, CEO, CEO approval rate, etc. The user can deduce the employer’s reputation from the info.

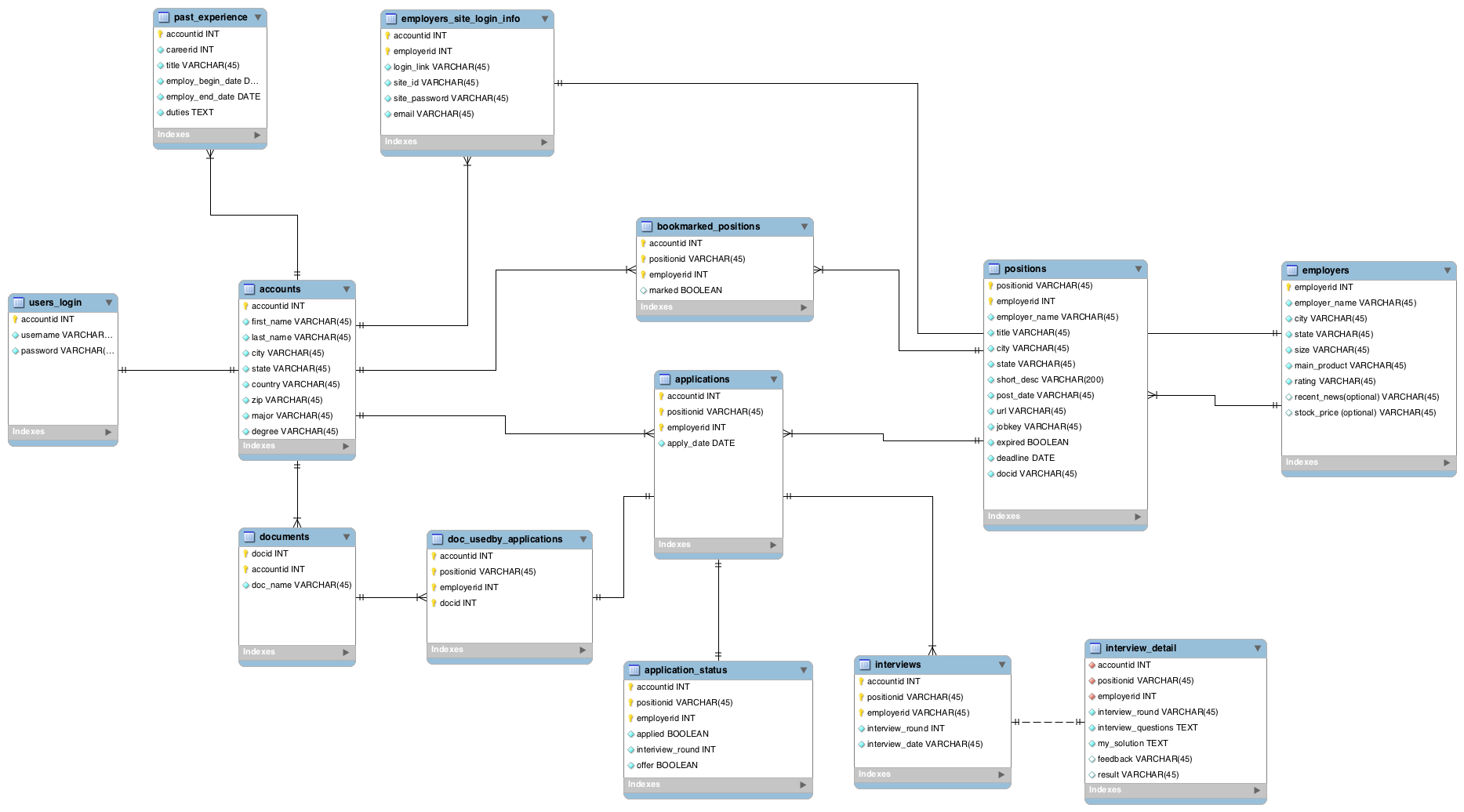
The overall data analysis: what kind of skill set is required in those jobs that the user saves? Which language is most wanted among those jobs? (E.g. the swift programming language recently released by Apple?) How may open positions of those jobs now?

**Solving Strategies**Project implementation can be broken down into several stages:

1. **Framework Investigation**:   
   In this stage, we start by analyzing different web technologies such as Ruby on Rails, Spring+Hibernate (Java), Django (Python), Flask (Python) etc. to figure out which would be advanced and fun to use nowadays.
2. **API Investigation:**   
   In this stage, we would investigate the API options we have from different job search engine sites and the possibility of obtaining data from these sites using their web services.
3. **Prototype Development:**   
   - In this stage, we would design the database that is needed for our application and write small applications 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.

Database Design

**E-R Diagram**

 Fig. E-R Diagram of our project

The figure above shows the diagram of our project. Two figures below show the zoom-in left and right side of the same diagram.



Fig. Left side of E-R Diagram

Tables on figure above:

* users\_login
* account
* past\_experience
* employers\_site\_login\_info
* document
* doc\_usedby\_applications



Fig. Right side of E-R Diagram

Tables on figure above:

* positions
* bookmarked\_positions
* applications
* application\_status
* interviews
* interview\_detail
* employers

Implementation

In our implement, we picked Flask and SQLAlchemy (ORM). It’s a typical MVC application.

**Architecture**  
~/careermedley

|-- run.py # bring the application up  
 |-- config.py # configuration

|-- db\_create.py # database creation

|-- /env # virtual environment

|-- /application # application module

|-- \_\_init\_\_.py # general application setup

|-- /static # store statics files: images, js and css

|-- /css

|-- /img

|-- /js

|-- /templates # holds jinjia2 templates

|-- base.html

|-- login.html

|-- user.html

|-- main\_page.html

|-- …

|-- forms.py # input data mappers and validators

|-- models.py # classes that be mapped to tables

|-- view.py # router and controller

Other miscellaneous technologies we used:

* OpenId
* Full text search based on table index
* bootstrap

Technology stack:

**Web development framework- Flask (Python)**

[Flask](http://flask.pocoo.org/) 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>

**Data access layer: SQLAlchemy (Python database toolkit)**

In our project, [SQLAlchemy](http://www.sqlalchemy.org/) 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.

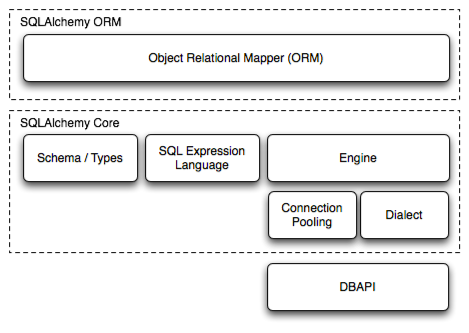


Fig 1. SQLAlchemy internal structure

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

**Web services**Indeed  
<https://ads.indeed.com/jobroll/xmlfeed>  
 - Case 1: dynamically retrieve a bunch of job openings by submitting a query to indeed;

- Case 2: dynamically retrieve the detail of a specific job by submitting a jobkey.  
  
Glassdoor

<http://www.glassdoor.com/api/index.htm>

- Case 1: dynamically retrieve the detail of a specific employer by submitting the employer name.

**Use cases (which reflects CRUD operations)**CareerMedley use case analysis

- List all open positions

- Update open positions

- List all employers

- Update employers

- List/Upload/Delete resumes or Cover Letter

- Register on employer’s official website

- List all my job applications

- Create views for job applications

- View a specific job application view

- Apply a specific position

- Update application status  
--------------------------------------------------------------------------------------------

Use Case: applicant lists all open positions

Actors: applicant, IDE

Description: when an applicant logs in, he/she can see a list of open positions.  
Precondition: the applicant has logged in

Steps:

1. IDE queries table: positions  
2. IDE provides a list of all open positions (software engineer, system engineer, etc.): the title of the position, location, employer, etc.

--------------------------------------------------------------------------------------------Use Case: applicant updates open positions:

Actors: applicant, IDE

Description: applicant can import new positions from external sources and if there are some new employers emerge, add them to table: employers

Steps:   
there are two mechanisms to update open positions:

a. import by invoking INDEED api

b. manually type in related infos, like title, location, employer name.

and then, if there are new employers, insert them to table: employers

--------------------------------------------------------------------------------------------Use Case: applicant lists all employers

Actors: applicant, IDE

Description: when an applicant logs in, he/she can see a list of all employers.  
Precondition: the applicant has logged in

Steps:

1. IDE queries table: employers  
2. provides a list of all open positions (Google, Facebook, etc.)

--------------------------------------------------------------------------------------------Use Case: applicant updates employers

Actors: applicant, IDE

Description: applicant can import new positions from external sources and if there are some new employers emerge, add them to table: employers

Steps:   
there are two mechanisms to update open positions:

a. import by invoking GLASSDOOR api

b. manually input in related information, like title, location, employer name, ratings.

--------------------------------------------------------------------------------------------Use case: List/Upload/Delete resumes

Actors: applicant, IDE

Description: the applicant can list/upload/delete resumes

Precondition: the applicant logs in DB

Steps:

1. IDE queries tables: accounts and resumes tables to get available resumes version for current applicant

For list resumes: IDE return the applicant’s all resume versions

For upload resume: upload a new resume to DB, insert new entry in table: resumes to point to the new resume

For delete resume: IDE displays list of resumes, applicant picks up one to delete. IDE asks for delete confirmation, the applicant acknowledges the deletion, the IDE displays new list of resumes  
--------------------------------------------------------------------------------------------

Use case: List/Upload/Delete cover letters

Similar case as above

--------------------------------------------------------------------------------------------Use case: applicant registers on employer’s official website

Actors: applicant, IDE

Description: the applicant registers on employer’s official website then populates the fields of table: employer\_site\_login\_infos

Precondition: the applicant logs in DB

Steps:

1. applicant visits the official website of the employer

2. applicant chooses a combination of email, id, password to register on employer’s website

3. applicant also input the combination of email, id, password to the table: employer\_site\_login\_infos (Since different websites require different id/password naming rules, if we don’t store those combinations carefully, sometimes we forget the combination, and then we couldn’t login the employer’s official site again.)  
--------------------------------------------------------------------------------------------

Use Case: applicant lists all his/her applications

Actors: applicant, IDE

Description: the applicant can view all his/her applications’ status  
Precondition: the applicant has logged in

Steps:

1. IDE queries tables: accounts and positions to get job applications for an applicant

2. IDE displays basic info of those job applications：the position, location, employer, etc.

--------------------------------------------------------------------------------------------Use case: create view for job applications

Actors: applicant, IDE

Description: applicant creates view for job applications  
Precondition: applicant has logged in

Steps:

1. applicant requests to create view for job applications

2. IDE queries the tables: accounts, positions and employers to create the view for job application with related infos  
--------------------------------------------------------------------------------------------

Use case: view a specific job application view

Actors: applicant, IDE

Description: applicant view job application views  
Precondition: applicant has logged in

Steps:

1. IDE displays view for job application with related infos.  
--------------------------------------------------------------------------------------------  
Use case: apply a specific position

Actors: applicant, IDE

Description: applicant applies an open position   
Precondition: applicant has logged in

Steps:

1. applicant fills in the fields, upload resumes and CVs, as required on the official job application link

2. applicant fills in related infos, such as date, which version of resume, which version of CV to local DB tables: applications

--------------------------------------------------------------------------------------------Use case: update application status

Actors: applicant, IDE

Description: applicant can change the application status  
Precondition: applicant has logged in

Steps:

1. IDE queries tables: accounts and applications to find a specific application for a specific applicant

2. the applicant marks the job as interested, applied, interviewed, offer etc.

3. if the applicant has been interviewed, he/she input the interview questions into tables: questions. In this way, the applicant can learn from the past interviews and improve him/herself.

--------------------------------------------------------------------------------------------Use case: view deadlines

Actors: applicant, IDE

Description: applicant can check whether the incoming deadlines of positions  
Precondition: applicant has logged in

Steps:

1. IDE queries the table: deadlines to display the incoming deadlines of positions (a month after now)

(Note: This is not a complete list of use cases or final use cases. Use cases will be added or deleted depending on further function discussion and the progress of development.)

Summary

**- Number of:**Tables：  
fields：  
foreign keys：  
mapping:   
tables：  
enumerations：

classes：

CRUD operations：  
interfaces：  
pages：

* /main.html
* /login.html
* /logout.html
* /profile.html
* /user.html
* /document.html
* /applications.html
* /user/userid/jobkey/apply
* /user/userid/jobkey/applydetail
* /user/saved\_positions
* /user/employer\_search.html
* /user/favorite\_employers.html

fields：

* Key lessons you learned – anything that surprised, delighted or concerned you.
* For Team projects:
  + Your specific contributions to the project
  + Comments and Feedback for your team members.

Statistics  
Thank you for your final reports so far. Please don't forget to make your code available on github or bit bucket or any other online source control and send me the link. Please include the final report as a PDF in  the source control and images of the user interface. Also include the following statistics.

**- Number of: Tables, fields, foreign keys, mapping tables, enumerations, classes, CRUD operations, interfaces, pages, fields.**

**~~- Also include a list of the web services APIs you used, their URLs, and a one sentence description of you use them.~~**

**~~- Briefly explain (a paragraph) the architecture,~~**

**- technology stack,**

**~~- ORM (if any), or data access layer technique.~~**

Please provide the statistics, source, final report as a PDF, in a source control **by this Friday.** Do not include libraries, JAR files, compiled objects, or project or configuration files, only source code. I'll grade everything this weekend.

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/>