Table of Contents

Introduction

Problem Context

With the rise of the Internet and social media, a company's online presence is becoming increasingly important. However, from Yelp to Twitter to Facebook, the number of websites that a company must constantly monitor is only increasing. To mitigate this, companies spend thousands of dollars each year hiring people to search the Internet and identify sources of both positive and negative reviews. We have developed a web application that allows users to simply add social media APIs that they would like to pull review data from. Once the review data is obtained, we analyze the review text for its sentiment and display the resulting information in a user-friendly format.

Shareholder and Customer Requirements

Working with our sponsors, we developed several shareholder and customer requirements that our project had to meet: (1) open source, (2) extensible, (3) user friendly, (4) search functionality, (5) customizable, and (6) understanding of existing Social Media APIs.

Open Source

Our project is intended to be open source so that any developer can continue development on the project and customize the project to suit their needs. As a result, our project is currently hosted in a public GitHub repository.

Extensible

Since our project was designed to be open source, we made each layer of our application modular and extensible. This means that future developers can easily replace a portion of our application with one that they have personally created. Additionally, developers can easily add new information, such as a social media source, by simply adding a new configuration file.

User Friendly

Our project was designed to have an intuitive user interface so that new users can quickly learn how to use the platform. We measured this metric using user testing data on navigability, learnability, and satisfaction.

Search Functionality

We implemented a search functionality to allow users to be able to search all previous reviews to find posts that matched the keywords entered in the search bar. We validated this requirement by examining the accuracy of the search results as well as the speed with which the search is completed.

Customizable

The user interface of our web application can be customized by the user to display only the information that the user cares about.

Understanding Existing APIs for Social Media Platforms

For this project, we spent a significant amount of time researching the APIs provided by various social media platforms. Our findings can be found in Appendix _____.

System Architecture

Design Overview

Our software consists of four layers: a user interaction (UI) layer, a middle or business layer, a service layer, and a data layer. We also built social media grabbers to retrieve review information from social media websites.

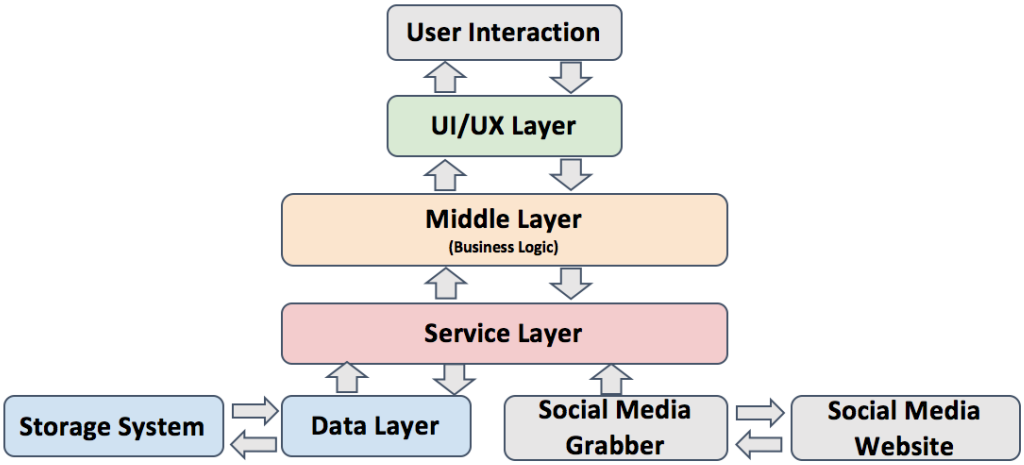


Figure 1: System architecture overview.

The user interacts with our web application through the UI layer by performing actions such as creating an account, logging into the application, changing configurations, or viewing the reviews. The data that the user passes to the application, such as login information or user settings, are then passed to the middle layer, which takes that information and decides what to do next. This layer will call different APIs in the service layer to complete tasks as required by our business logic. If these jobs involve data storage, the data layer will process the request. On the other hand, if more social media information is needed, the social media grabber will be responsible.

UI Layer

We utilized the React.js library as our primary tool. By using React.js, it allowed us to easily modularize our JavaScript code and give the user a rich and seamless experience in the dashboard page.

React.js

The React.js is a library that generates all the UI content in front end to make the web page more dynamic. Check out http://facebook.github.io/react/ to learn more about React.

UI Components References

`dashboard-login.js`

This component is our whole single-page dashboard app after the user logs in. It connects to all four sections: summary, review feeds, charts and settings. The default section is the summary section.

`dashboard-navbar.js`

This component is at the top of our dashboard. Due to time limitations, it currently has no functionality.

`dashboard-sidebar.js`

This component is to the left of the dashboard. It consists of the five icon buttons that allow the user to change between the four sections of the application and to logout. It also retrieves user information from Django server using the function `loadUserInfoFromServer()`.

`dashboard-summary.js`

This component is higher level component for the summary sections.

`dashboard-summary-review-division.js`

This component is the three boxes in the summary section, showing how many positive, neutral and negative reviews the company has.

`dashboard-summary-review-pick.js`

This component consists of the 10 randomly selected reviews that are displayed in the summary section. If the user refreshes the page, different reviews will appear.

`dashboard-review-feeds.js`

This component is the higher level component for the review feeds section. It contains two variables as components in the file: `SearchBar` and `ReviewFeed`. The `SearchBar` is the search tool on top of the review feeds section. The `ReviewFeed` renders review cards in different styles according which platform the review is from.

`dashboard-review-card.js`

This component contains three kinds of review cards, one for each of the provided API options. They are: `CityGridReviewCard`, `TweetCard`, and `YelpReviewCard`. They are rendered in `dashboard-review-feeds.js`.

`dashboard-charts.js`

This component is a higher level component for charts section. It renders the three kinds of charts that we support: a time series showing number of reviews by month, a bar chart showing ratings on a 5-point scale, and a pie chart of customer sentiments.

`dashboard-charts-barchart.js`

This component is the module for bar chart ratings. It is a bar chart showing how many reviews are there for the ratings ranging from 1-5.

`dashboard-charts-piechart.js`

This component is the module for sentiment pie chart. It shows the percentage for positive, neutral and negative reviews.

`dashboard-charts-timechart.js`

This component is the module for number of reviews by month. It shows the distribution of reviews along the timeline.

`dashboard-settings.js`

This component is the higher level component for settings sections. It contains modules from `config-account.js` and `config-chart.js`.

`config-account.js`

This component is the module that allows users to change their configurations for different social platforms.

`config-chart.js`

This component is the module that allows users to customize what charts they want to see in their dashboard.

`add-platform-block.js`

This component is mounted on a `div` dom component with the `id` attribute `dynamic-api-config` in `register.html` (the sign up page). It is rendered as a green button to allow user add their account information on different social platforms.

Middle Layer

Django was chosen as our web framework. It is a powerful and battle-tested framework that is used by a lot of well-known websites, such as Instagram and Uber. Most importantly, this framework is also used by our sponsor and therefore would be easy for them to maintain.

Django

To learn more about Django, please visit their website: www.djangoproject.com.

Business Logic

The `business.py` file contains the business logic required for our application, including logging out the user (`log_out_user_logic`), checking whether the user is logged in and redirecting them to the dashboard if they are (`check_status_redirect`), allowing the user to sign up for a new account (`signup_logic`), and allowing the user to login (`login_auth_logic`).

Services

The services.py file makes the different calls to the service layer as needed by the business logic.

Views

The `views.py` file returns the various pages of our web application based on the actions of the user. We have several pages, including the index page, the login page, and the dashboard page.

Service Layer

The service layer is written in Java, serving in another server, a design is suggested by our sponsor.

RESTful API

To communicate between the frontend and backend we created a REST Service. The basic functionality of the REST service is to ask the backend for data, or tell the backend to pull data for a user. The APIs are defined as follows:

`/pull`

Description: Pulls data for a specific company from all API's, but we recommend you use `/updateAPI` or `/init` instead.

Type: `GET`

Parameters: company, location, yelpName, twitterName, citygridName

Example:

`GET` http://localhost:3456/pull?company=zingerman%27s&location=ann%20arbor&yelpName=zingermans-delicatessen-ann-arbor-2

`/init`

Description: Initializes data for a company by pulling from all API's (for signup)

Type: `POST`

Parameters:

```
{
  "company": <company_name>,
  "twitterName": <twitter_name>,
  "yelpName":<yelp_name>,
  "citygridName": <citygrid_name>,
  "location": <location>
}
```

Example: `POST` http://localhost:3456/init

```
{
  "company": "zingermans",
  "twitterName": "zingermans-twitter",
  "yelpName": "zingermans-delicatessen-ann-arbor-2"
  "citygridName: "zingermans-citygrid",
  "location": "ann arbor, mi"
}
```

`/updateAPI`

Description: Pulls data from specific APIs to update data for a company (same as `/init` but can specify API's as a list)

Type: `POST`

Parameters:

```
{
```

```
  "company": <company_name>,
  "twitterName":<twitter_name>,
  "yelpName":<yelp_name>,
  "citygridName": <citygrid_name>,
  "location": <location>,
  "apis": <comma-separated-list-of-apis>
}
```

Example:

```
POST http://localhost:3456/updateAPI
{
  "company": "zingerman's",
  "citygridName": "zingermans",
  "twitterName": "zingermans-twitter",
  "yelpName":"zingermans-ann-arbor",
  "location":"ann arbor, mi",
  "apis" : "twitter"
}
```

`/search`

Description: Grabs reviews for a specific company from database, specifying the APIs from which to pull. Not specifying any APIs will grab reviews from all the APIs. Not specifying a query will return all reviews.

Type: `GET`

Parameters: company, query (optional), twitter (optional), yelp (optional), citygrid (optional)

Examples:

```
GET http://localhost:3456/search?company=zingerman's
```

```
GET http://localhost:3456/search?company=zingerman's&query=tasty
```

```
GET
http://localhost:3456/search?company=zingerman's&query=tasty&twitter=yes&citygrid=yes
```

Service Sublayers

Service

Provides core functionality, interfaces with other sublayers, data layer and front end. The service code is responsible for hosting a running service so as to receive requests from business layer and provide core data processing functionalities.

`Class - Server.java:`

The actual server implementation that provides service.

`Member - dbAddr = "127.0.0.1"`

This member specifies the Cassandra database IP address, 127.0.0.1 as default.

`Member - keyspaceName = "review_keyspace"`

`Member - tableName = "review_table"`

These two members specifies the keyspace name and table name to store the reviews.

`Function - void main(String argv[])`

This function provides the interface to spin up the service, and it is also the entry point for our back-end service program. One integer indicating the port number on which the service will be running is required as a parameter.

`Function - SentimentStruct sentimentAnalyze(String review)`

This function perform sentiment analysis for the provided review text, and a SentimentStruct is returned as result.

```
Function - void pullAllAPIAndStoreForUsers(List<CompanyStruct>
companyNameList, List<String> locationList)
```

This function grabs data from social APIs for the given list of companies.

```
Function - void pullSpecificAPIforUsers(List<String> APIs, List<CompanyStruct>
companyNameList)
```

This function grabs data from given social APIs for the given list of companies.

```
Function - List<ResponseStruct> pullAPIsForUsers(List<CompanyStruct>
companyNameList, List<String> locationList, List<DataGrabberGeneric>
listGrabber)
```

This function is an internal implementation to grab data from social APIs for the given list of companies.

Class - ServerGeneric.java:

The core internal implementation of the service.

`Function - void handle(HttpExchange exchange)`

This function is the core of the class, which handles the requests from business layer. This function is implemented by different HttpHandler to deal with corresponding requests.

`Function - void serve()`

This function initializes the internal structures including the query handlers.

Class - CompanyStruct.java:

The internal structure of a company representation, which specifies the company name, Twitter account name, Yelp business name, Citygrid business name and its location.

Class - ResponseStruct.java:

An agreed structure to exchange information between service layer and data layer, and the information includes API names, company names and the responses returned by the specific API.

Class - SentimentStruct.java:

An agreed internal structure to describe sentiment analysis result that includes sentiment feeling and sentiment score. The sentiment feeling can take the following values: "positive", "negative", "neutral", or "mixed". While the sentiment score describes the strength of the sentiment feeling. Initially, the sentiment feeling is set to “neutral” and the sentiment score is initialized as 0.

Grabber

The grabber code is responsible for calling social media APIs to pull data for companies.

Class - DataGrabberGeneric.java:

A general base template for specific data grabber implementations, which provides universal interfaces to make http requests to social media API and receive response data.

`Function - abstract List<ResponseStruct> pullData(CompanyStruct companyName, String location)`
This function specifies the interface of pulling data for specific company from all social media APIs.

`Function - List<ResponseStruct> pullDataForAll(List<CompanyStruct> companyNameList, List<String> locationList)`

This function pulls data for a list of companies from all social media APIs.

`Function - String sendGet(String url)`
This function makes a "GET" http request using the given url.

Class - GrabberCitygrid.java:

The actual grabber implementation for Citygrid API.

`Function - String generateURL(String companyName, String location, int pageNum, int rpp)`
This function generates a Citygrid formatted URL using the given parameters.

`Function - String processResponse(String response)`
This function serves as a helper to fix the invalid character encodings in Citygrid response.

`Function - List<ResponseStruct> pullData(CompanyStruct companyName, String location)`
This function overrides the "pullData" function specifically for Citygrid.

Class - GrabberTwitter.java:

The actual implementation for Twitter APi, and overrides the pullData interface.

Class - ImportIO.java:

The generic ImportIO interface.

Class - ImportYelp.java:

The actual implementation for ImportIO Yelp API, and overrides the pullData interface.

Utility

This utility code/lib contains a set of frequently used classes and functions that are used by other classes. Specifically, a large portion of the utilities provided are IO related such as parsing configuration files.

Class - Parser.java:

This class contains frequently used file parser tools/functions.

`Member - String RESOURCE_PATH`

The path of the resources folders.

`Member - String API_KEY_PATH`

The path of the file that contains the API key for Alchemy API.

`Member - String GRABBER_PATH`

The path of the folder that contains the set of grabbers, and those grabbers will be loaded dynamically.

`Member - Map<String, Map<String, String>> path_map, Map<String, Double> conversion_map, List<String> business_attributes, List<String> sources`

There members contains the mapping between the format of the data returned by social media API and our canonical attributes set.

`Function - Map<String, String> getCityGrid(), Map<String, String> getYelp(), Map<String, String> getTwitter(), Map<String, String> getAPIMap(String APIName)`

These functions automatically parses the configuration file to obtain the path to attributes mapping for each of the social media APIs.

Data Layer

A relational database is adopted for user information and a non-relational one for different formatted reviews from various social media websites. Cassandra is chosen for our non-relational database, while SQLite3 is used now for storing user information.

Relational Database Schema (Primary Key Omitted)

Userprofile

Attribute name	Schema type	Foreign key
user	NUMBER	YES
area	VARCHAR(100)	NO
api_config	TEXT	NO
chart_config	TEXT	NO
my_company	NUMBER	YES

Company

Attribute name	Schema type	Foreign key
company_name	VARCHAR(100)	NO

Important Design Details

Whenever a new Grabber API is added, a new class with the name of the API must be added to the database layer, and must implement the 2 abstract methods defined in `database.API`. For example, notice how the 3 APIs that are in the grabber package—`GrabberImportMagicYelp`, `GrabberCitygrid`, and `GrabberTwitter`—have corresponding classes in the database package—`ImportMagicYelp`, `Citygrid`, and `Twitter`. For consistency, the class names should be identical to the "Grabber" added to the class in the grabber package (i.e. "ImportMagicYelp" has the same camel case format as "GrabberImportMagicYelp").

To parse the reviews for each API, the data layer uses the configurations specified in `/resources/API<API-name>.txt`. See documentation on the resources folder for more details.

The Data Layer is designed in a flexible format so that it is easy to add new Data Sources. Adding a new data source consists of a few steps:

1. Adding a Grabber class that pulls the data.
2. Adding a configuration file in the resources folder that allows for parsing of the data.
3. Adding a class in the database package with the API name.

Java Data Layer

To interface with the database, we implemented a Java Data Layer. This data layer provides 2 main functions, to insert new reviews into the database, and to select reviews upon a search. This section describes in detail the Data API for the Java Data Layer.

`public interface database.Data`

```
void insertData(List<ResponseStruct> responses) {
  // Takes a list of 'responses' (reviews) from various APIs and inserts into the
database.  See service.ResponseStruct for more detail about 'responses'.
}
```

```
String select(SearchStruct search, List<String> attributes) {
  // Selects reviews from database for a company with a given 'search'. Formats each
review as a JSONObject with the requested 'attributes'. Puts all the reviews
(formatted as JSONObject) into a JSONArray. Returns the string format of the
JSONArray. See service.SearchStruct for more detail about 'search'. Entries in
'attributes' can be 0+ of the attributes found in /resources/business.txt
}
```

database.AccessData implements Data

```
public static void initializeDatabase(String host_in, String keyspace_name_in, String
review_table_in, String inverted_table_in) {
  // initializes connection with database with the given parameters. Called by
service.Server upon startup. Must be called before select() or insertData()
}
```

```
public static void truncateTables() {
  // removes all data from the review table and inverted table (inverted table
currently not being used).  Dangerous operation, should only be used in special
circumstances
}
```

```
public static void truncateReviewTable() {
  // removes all data from review table. Dangerous operation.
}
```

```
public static void truncateInvertedTable() {
  // removes all data from inverted table. Dangerous operation.
}
```

```
public static void close() {
  // closes connection with database. Should be called when server stops running
}
```

```
public static Session getConnection() {
  // returns connection to database
}
```

public abstract class database.API extends AccessData

```
public abstract String getContent(JSONObject currentReview) {
  // Function that must be implemented whenever a new API Grabber is added. Parses
the 'currentReview' for the review content (i.e. review text) and returns it as a
string.
}
```

```
public abstract String getId(JSONObject currentReview) {
  // Function that must be implemented whenever a new API Grabber is added. Parses
the 'current' review' for the review ID and returns it as a string.
}
```

```
protected void insert(ResponseStruct responses) {
  // Takes 'responses', parses out each individual response, and inserts into
database. Called by insertData()
}
```

```
private void insertReview(String review_id, String company_name, String full_review)
{
  // function that actually inserts a review into the database.
}
```

```
public JSONObject formatReview(Row current_row, List<String> attributes) {
 // takes a row that has been selected from the database and formats it to be
returned to the front end in a recognizable format.  Called by select().
}
```

`database.ImportMagicYelp`

A class created for inserting data from the ImportMagicYelp API into database. Implements getContent(…) and getId(…), the two abstract methods of API.

`database.Citygrid`

A class created for inserting data from the Citygrid API into database. Implements getContent(…) and getId(…).

`database.Twitter`

A class created for inserting data from the Citygrid API into database. Implements getContent(…) and getId(…).

Resources

This section contains important information about the files in `src/main/resources`.

business.txt

This file defines the attributes that will be part of the `/search` API response to the front end.

Currently, when a REST call to `/search` is made, the 'source', 'title', 'content', 'sentiment_feeling', 'sentiment_score', 'rating' and review 'date' are passed back as attributes in each json response. These attributes can be changed according to what information is needed on the front end, but this will require changing the parsing implementations in both the front and backend.

tbl_create.sql

This cql script will create the Cassandra keyspaces and tables that are needed for the application.

ServiceInit.conf

This file has a list of the API's that data is being pulled from, followed by either 'YES' or 'NO'. 'YES' indicates that that data for that API has already been pulled for all companies, and 'NO' indicates that it hasn't. Upon startup of the server, data will be pulled for all companies from each API that has a 'NO' value.

alchemy_api_key.txt

A file that contains the AlchemyAPI API key (needed for API call to AlchemyAPI, which provides the sentiment analysis). The current key has access only to the free tier of AlchemyAPI, which allows for 1,000 transactions per day.

API<Api-name>.txt

Files of this format describe the JSON path to each business attribute, so that when pulling data from this API, the reviews can be parsed properly.

The proper configuration for the file is:

```
<attribute>         $.path.to[0].attribute
reviews             $.tables[0].results
{
      “date”: “1-10-1010”
      “tables”: [{ “results”: [{ “review_content”: “very good”},
{“review_content”: “bad”}]}, {“other info”:[]}]
}
```

The path always starts with the '$' which represents the outermost JSON Object response. To specify the path, separate each attribute (i.e. a nested JSON Object) by a '.', and to specify a specific element in a nested JSON Array, use '[]' notation.

The 'reviews' and 'content' paths must be specified in this file, other attiributes from business.txt are optional. 'reviews' is a path to a JSON Array with multiple reviews. The 'content' path is a relative path from inside each 'review' JSON object. In the above example, the path for 'content' would be '$.review_content'.

The 'sentiment_feeling' and 'sentiment_score' do not need to be specified, as they will be automatically determined from a sentiment analytics service, like AlchemyAPI.

APICitygrid.txt

The configuration file for Citygrid's JSON response.

APIImportMagicYelp.txt

The configuration file for Yelp's JSON response.

APITwitter.txt

The configuration file for Twitter's JSON response.

Googleplaces.txt

The configuration file for Google Places's JSON response. We aren’t currently pulling from Google Places. If the Google Places API were to be added, then this file should be renamed to "APIGoogleplaces.txt".

End User Guide

Signing Up

To register for an account, the user should follow the steps below.

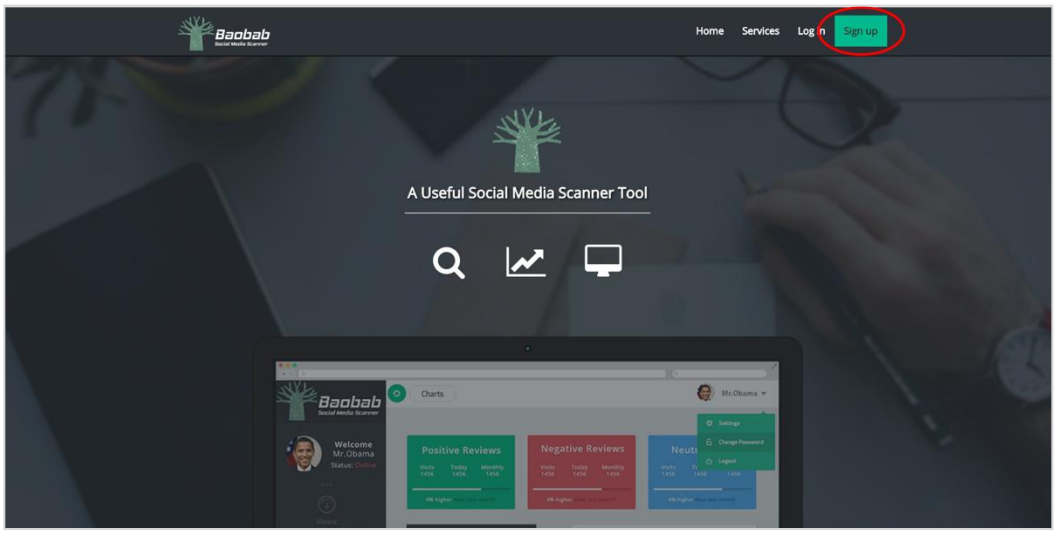


Figure 2: Sign up button location on the index page.

Clicking the "Sign up" button the index page will direct the user to a page where he can enter his company information into a form.

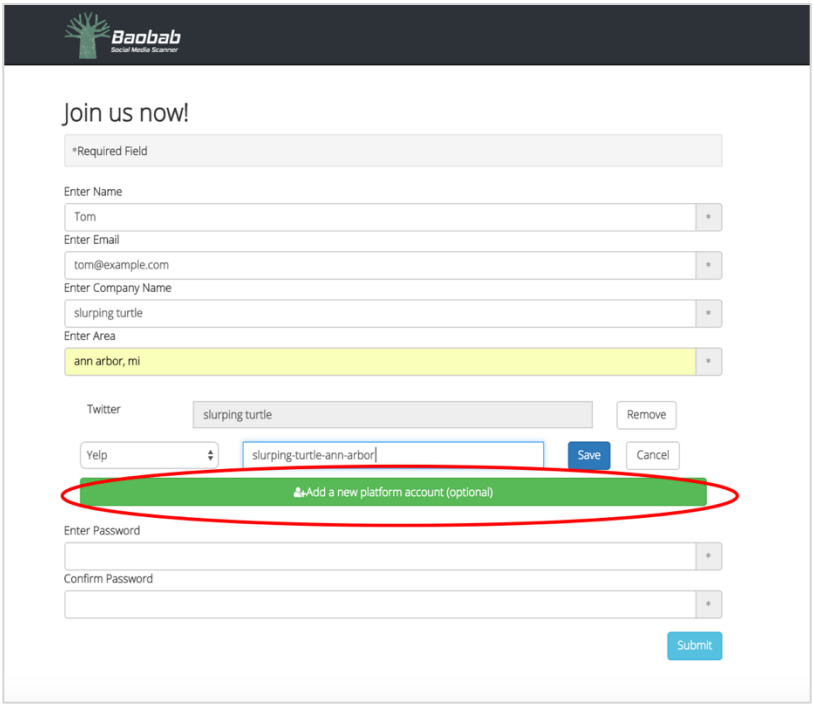


Figure 3: Entering user and company information.

The user has the ability to add his or her social media account information for several different platforms by clicking on the light green button. After filling in all the information and pressing the submit button in the bottom right corner, our social media grabbers will pull the requested information from the various sources. The following page will be displayed as the data is being initialized.

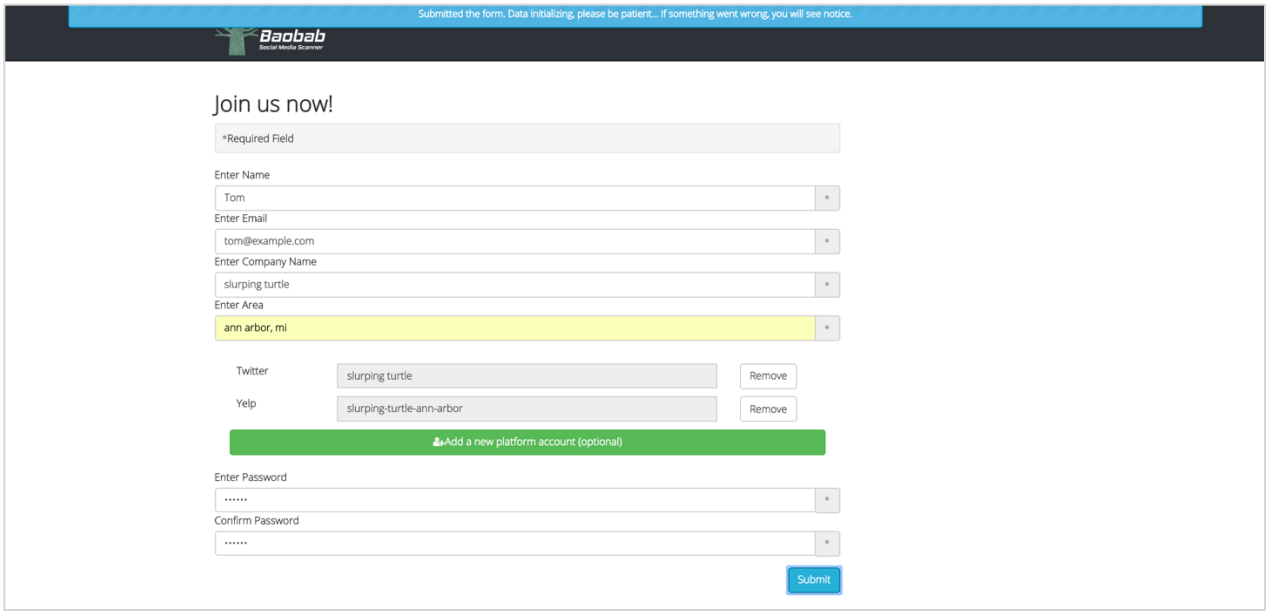


Figure 4: Screenshot after submitting user information.

After account initialization, the user will be redirected to his dashboard, and account registration is complete.

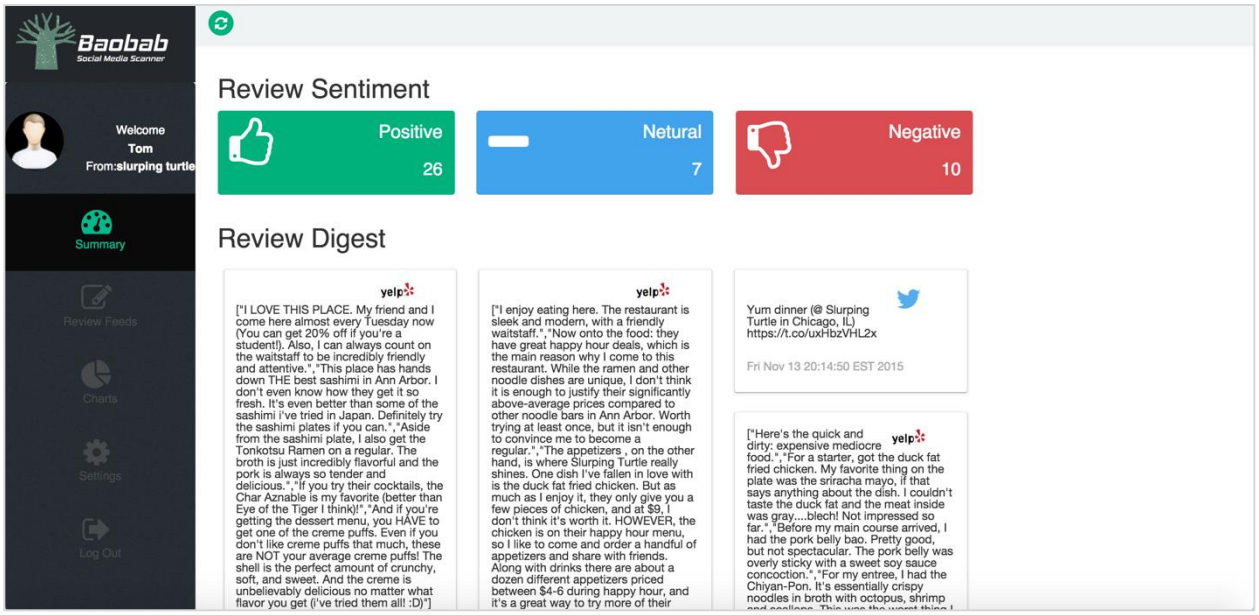


Figure 5: The user will be directed to the dashboard after registration.

Logging In

To begin the log in process, the user should first click the “Log in” button on the index page.

Figure 6: Login button on the index page.

The user will then be redirected to a page where he can enter his username and password.

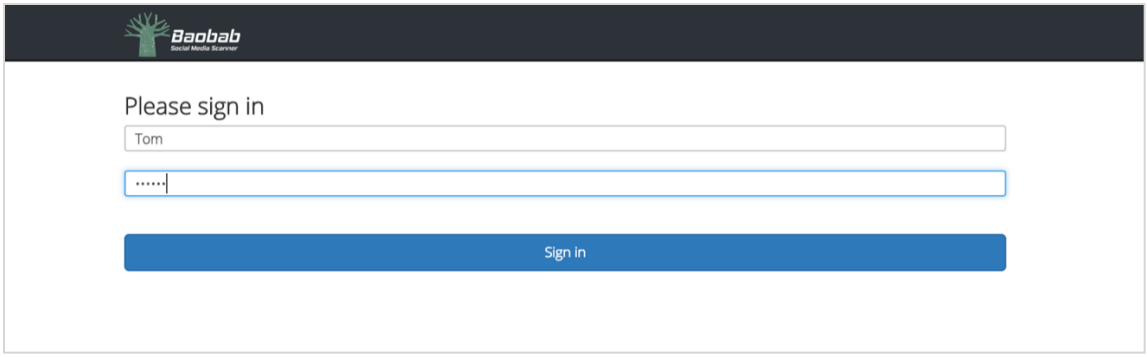
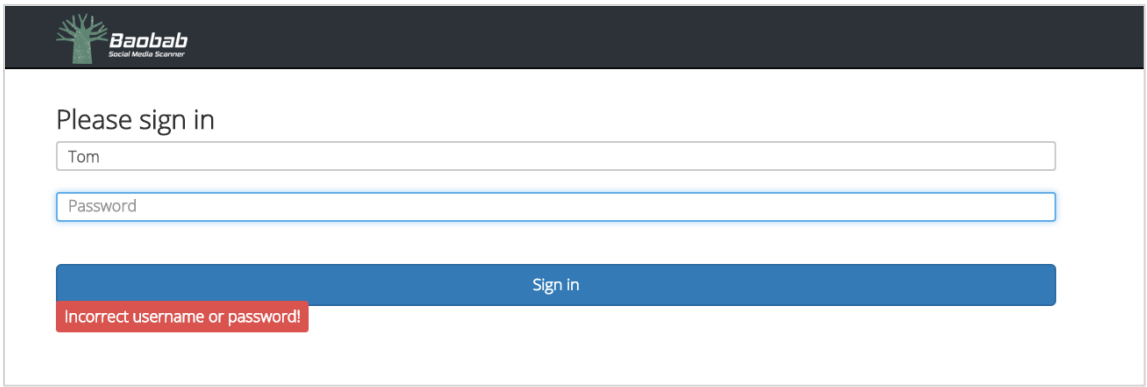


Figure 7: Sign in form.

If the wrong password or username is entered, the following error message will be displayed. Once the correct username and password are entered, the user will be directed to the user dashboard.



Dashboard Walkthrough

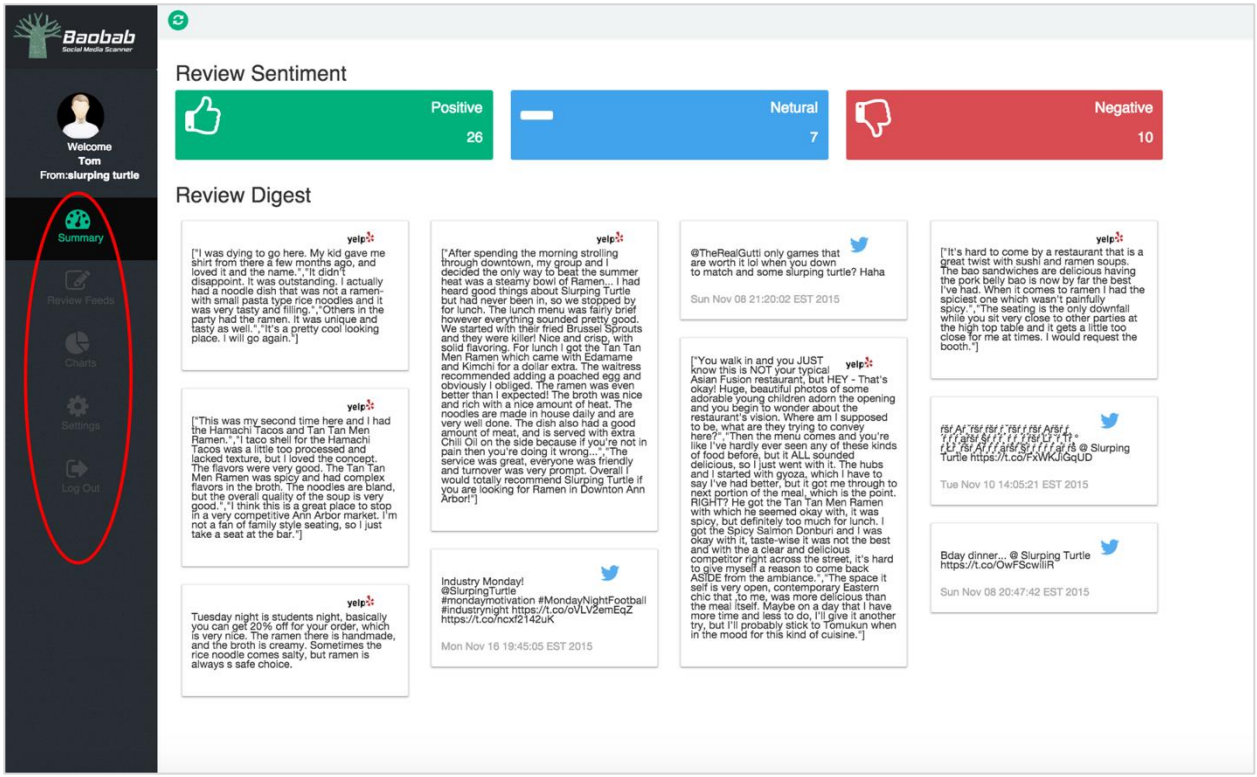


Figure 8: user dashboard

In the user dashboard, there is a sidebar placed in the left-hand side. The dashboard consists of four sections. Summary, Review feeds, charts, and settings. General statistics and random selected reviews are in this section. All the reviews for the company will be in the review feeds section. Customized statistical charts lie in the charts part. Additionally, you could update social platform configuration and what chart you want to see in the chart section in the settings section.

To change section, just click on the icon button in the sidebar.

Updating Platform Settings

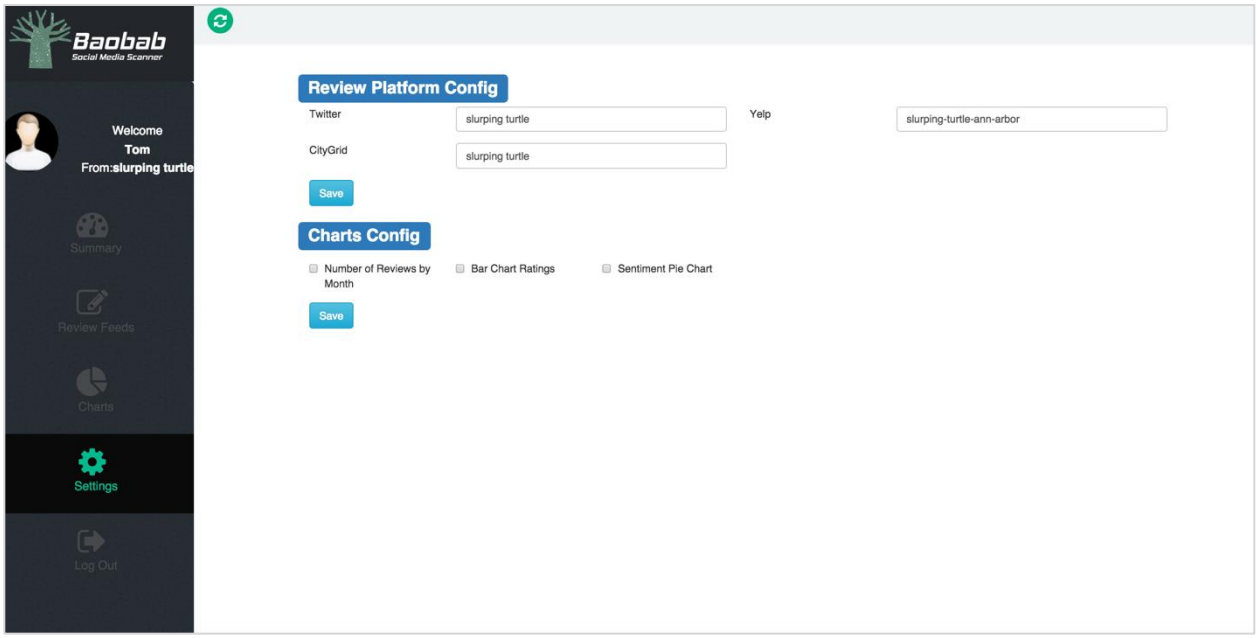


Figure 9: Settings section

In the settings section, you could update your platform configuration. For example, in the screenshot above, we update the platform account for "CityGrid". Then you could press the save button. After refreshing the page, you will see the updated reviews.

Customizing the Chart Configuration

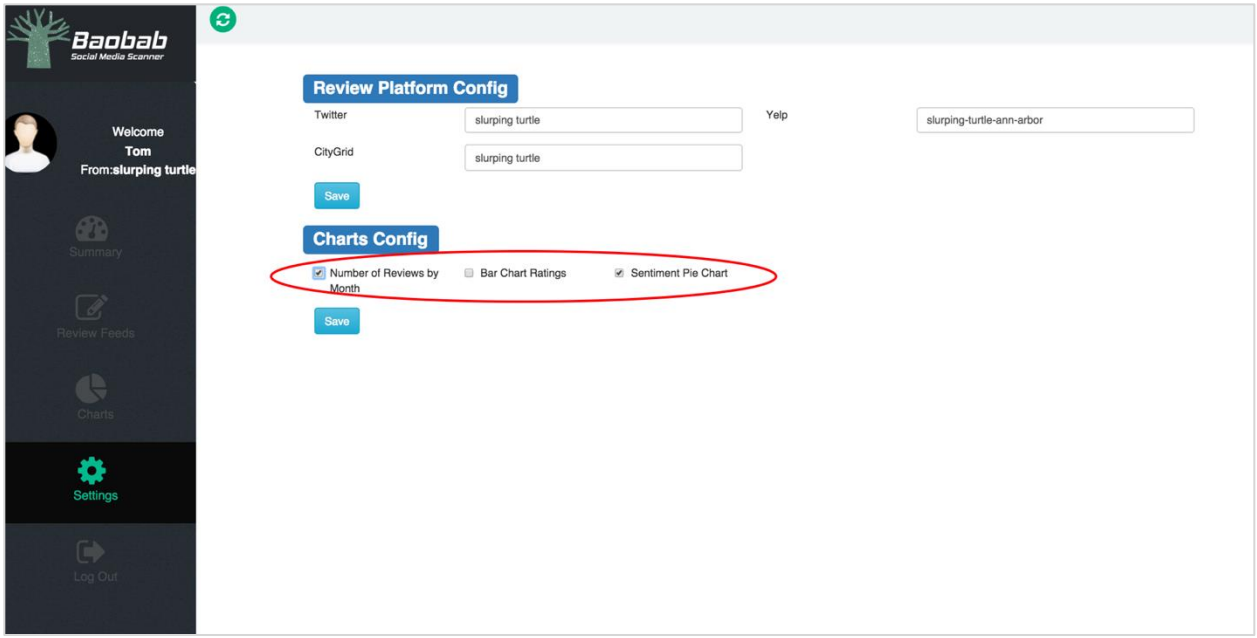


Figure 10: Customize shown charts

In the “Charts Config” of the settings, you could check the checkbox to select what kind of statistical charts you want to see in the chart section. Press the save button to confirm your selection.

Web Application Administrator Guide

Setting Up the Web Application

Installation

Please follow these steps to install the software:

Step 1:

```
pip install virtualenv
```

Note: Please visit this site to install pip http://pip.readthedocs.org/en/stable/installing/

Step 2:

```
virtualenv venv
```

Step 3:

Create a folder called **build** in **project/SocialMediaScanner/core/static,** and also create a file **Settings.py** in **project/SocialMediaScanner/SocialMediaScanner** (you could copy the following snippet)

```
"""
Django settings for SocialMediaScanner project.

For more information on this file, see
https://docs.djangoproject.com/en/1.7/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/1.7/ref/settings/
"""

# Build paths inside the project like this: os.path.join(BASE_DIR, ...)
import os
BASE_DIR = os.path.dirname(os.path.dirname(__file__))


# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/1.7/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'bykq5p!v31jyq9ee+^o1p!53)b@=vfu^cvatw45t(d2v#7=m&d'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

TEMPLATE_DEBUG = True
```

```
ALLOWED_HOSTS = []


# Application definition

INSTALLED_APPS = (
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'core',
)
#AUTH_USER_MODEL = 'core.UserProfile'


MIDDLEWARE_CLASSES = (
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.auth.middleware.SessionAuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
)

ROOT_URLCONF = 'SocialMediaScanner.urls'

WSGI_APPLICATION = 'SocialMediaScanner.wsgi.application'


# Database
# https://docs.djangoproject.com/en/1.7/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}

# Internationalization
# https://docs.djangoproject.com/en/1.7/topics/i18n/

LANGUAGE_CODE = 'en-us'
```

```
TIME_ZONE = 'UTC'

USE_I18N = True

USE_L10N = True

USE_TZ = True


# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/1.7/howto/static-files/

STATIC_URL = '/static/'
CASSANDRA_URL = 'localhost'
```

Step 4:

In folder **project/SocialMediaScanner**, run

```
source setup.sh
```

If the setup process succeeds, you will see:

=====Middle Layer Packages Successfully Installed====

====User Information Database Successfully Setup====

=====React Packages Successfully Installed====

=====React Components Successfully Generated====

If any of the installation fails, you will see explicit error message in the terminal.

```
New python executable in venv/bin/python
Installing setuptools, pip...done.
You are using pip version 6.0.8, however version 7.1.2 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
Collecting Django==1.7.2 (from -r requirements.txt (line 1))
  Using cached Django-1.7.2-py2.py3-none-any.whl
Collecting httpretty==0.8.10 (from -r requirements.txt (line 2))
  Using cached httpretty-0.8.10-py2-none-any.whl
Collecting requests==2.8.1 (from -r requirements.txt (line 3))
  Using cached requests-2.8.1-py2.py3-none-any.whl
Installing collected packages: requests, httpretty, Django



Successfully installed Django-1.7.2 httpretty-0.8.10 requests-2.8.1



=====Middle Layer Packages Successfully Installed====


Unknown command: 'makemigration'
Type 'manage.py help' for usage.
Operations to perform:
  Apply all migrations: admin, core, contenttypes, auth, sessions
Running migrations:
  No migrations to apply.



====User Information Database Successfully Setup====


npm WARN package.json static@1.0.0 No repository field.
npm WARN package.json static@1.0.0 No README data
npm WARN package.json Dependency 'noty' exists in both dependencies and devDependencies, using 'noty
@^2.3.6' from dependencies

> sleep@3.0.0 install                                  project/SocialMediaScanner/core/static/node_
modules/sleep
> node-gyp rebuild

  CXX(target) Release/obj.target/node_sleep/sleep.o
  SOLINK_MODULE(target) Release/node_sleep.node



=====React Packages Successfully Installed====



> static@1.0.0 build                                  project/SocialMediaScanner/core/static
> browserify -t reactify -t es6-destructuring src/dashboard-login.js -o build/dashboard-login.js


> static@1.0.0 build-signup                                  project/SocialMediaScanner/core/static
> browserify -t reactify -t es6-destructuring src/signup-config.js -o build/signup-config.js




=====React Components Successfully Generated====
```

Step 4:

Next, we need to install Cassandra 2.0+ (untested on earlier versions)

5.1 Go to http://www.planetcassandra.org/cassandra/ and follow the instructions under "Step Download Apache Cassandra to select the right version for your machine.

5.2 Go to the cassandra folder you downloaded, open /bin, execute ./cassandra

5.3 To check if cassandra is running, run "./cqlsh" from command line, or double click on "cqlsh" executable.

If Cassandra is running you should see cqlsh startup like this:

```
Connected to Test Cluster at localhost:9160.
[cqlsh 4.1.1 | Cassandra 2.0.12 | CQL spec 3.1.1 | Thrift protocol 19.39.0]
Use HELP for help.
cqlsh>
```

5.4 If it does not start properly, try to use chmod to change the permission of the executable

5.5 Run "cqlsh" again to confirm that cassandra has started

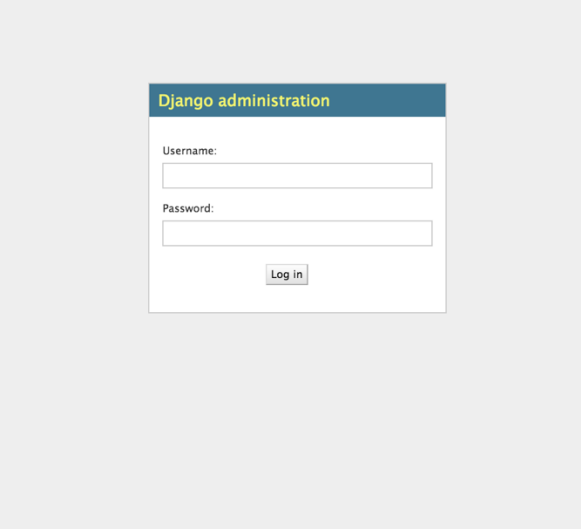
Creating the Superuser

In order to use the administrator tool, you need to create a superuser for the site. To create a superuser, run

```
python manage.py createsuperuser
```

Administrator Tool

As the administrator for the website, you could go to /admin to enter the administrator tool.



After type in the admin username and password you setup in the section "Setup the web application". You will see the screenshot as below. You could easily manage the users' profile (area, company, configuration) in the "core section". Also, the administrator could help reset users' password in the "Authentication and authorization" section.


Django administration
Welcome, renl. Change password / Log out
Site administration
Authentication and Authorization
Groups Add Change
Users Add Change
Core
User profiles Add Change
Recent Actions
My Actions
None available