

CS179G Report #1

Requirements: Data must be at least 5GB. Data must be processed in Spark and output must be stored in Cassandra.

Design:

None. We did not implement a web crawler

Implementation:

Direct download from University of Washington

(<http://homes.cs.washington.edu/~thickstn/musicnet.htmls>)

Evaluation:

11 GB of public domain music files, timing of notes, instrument of note, and notes' position in the metrical structure of the score. Time taken to collect the data was five weeks searching for the perfect data set and five minutes to download it thanks to Brian.

ScreenShots:

Contribution:

Clayton: I searched for the data set appropriate for our proposal. There was a lot of googling and checking with the team to make sure they were satisfied with the data presented.

Brian: I downloaded the data set of 11GB. This was important because the download was direct. It would have taken three days for either Clayton or David to do it because I had a high download speed.

<https://homes.cs.washington.edu/~thickstn/start.html>

David: I helped Clayton search for data sets appropriate for our proposal. I also tried to create a web crawler because we were having trouble finding a data set that matches exactly what we want. In the end, I did not make a web crawler because we were able to find a data set.

Report 2

Requirements

Our requirement for this phase of the project was to analyze our data in a parallel processing way using Spark, and store the output in Cassandra.

Design

For our project, specifically, we took a dataset containing the note data for 330 different pieces of classical music, and used PySpark to analyze each song (in parallel) to determine both the [key signature](#) of the song, and its [chord progression](#).

Implementation

We used PySpark to implement our code. The key signature analysis was implemented by creating 12 hardcoded lists for each possible key signature. These lists each contain the 7 possible notes that can appear in their respective key signature. The key is determined by looking at how often each note appears in the song, and sorting them by how often they appear (in descending order).

After this, our code goes down the sorted list of notes, and “blacklists” any key signature that does not contain the note. After this process, the final key signature of the song is chosen from whatever key signatures remain. A similar process is done for determining the chords, except this is done for each [bar](#) of the song, rather than the entire song as a whole.

Evaluation

Varying # of Workers (Full data size):

1 worker:

```
ala:37) finished in 0.231 s
17/03/01 21:29:41 INFO DAGScheduler: Job 662
ala:37, took 0.234895 s
Saved metadata
Total time: 148.989553928
17/03/01 21:29:48 INFO CassandraConnector: D
spark_group_09
17/03/01 21:29:49 INFO SparkContext: Invokin
```

2 workers:

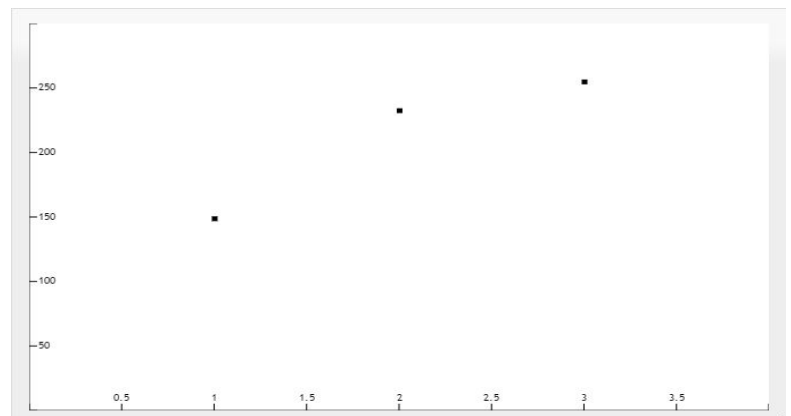
```
la:37) finished in 0.083 s
17/03/01 21:13:42 INFO DAGScheduler: Job 662
ala:37, took 0.088613 s
Saved metadata
Total time: 232.766702175
Exception in thread Thread-1 (most likely ra
Traceback (most recent call last):
  File "/usr/lib64/python2.7/threading.py",
  File "/usr/lib64/python2.7/threading.py"
```

3 workers:

```
17/03/01 20:56:32 INFO DAGScheduler: Results
la:37) finished in 0.077 s
17/03/01 20:56:32 INFO DAGScheduler: Job 662
ala:37, took 0.080218 s
Saved metadata
Total time: 255.078511953
17/03/01 20:56:33 INFO CassandraConnector: D
spark_group_09
Exception in thread Thread-1 (most likely ra
```

Graph of Execution Time vs. # of Workers:

Interestingly, the more workers we added, the *slower* the execution time got, despite the fact that our code analyzes the songs in parallel. This is likely due to the added overhead of more workers outweighing the benefits of parallelism. As seen on the graph, the execution time is not a linear increase, so it is likely that with much larger datasets, we would probably see the opposite result.



Varying Data Sizes (3 Workers):

82 songs analyzed:

```
ala:37) finished in 0.030 s
17/03/01 21:07:14 INFO DAGScheduler: Job
ala:37, took 0.102786 s
Saved metadata
Total time: 87.2972199917
17/03/01 21:07:21 INFO CassandraConnector
spark_group_09
17/03/01 21:07:21 INFO SparkContext:
17/03/01 21:07:21 INFO SparkUI: Stopp
```

165 songs analyzed:

```
ala:37) finished in 0.073 s
17/03/01 21:04:53 INFO DAGScheduler: Job
ala:37, took 0.078759 s
Saved metadata
Total time: 139.521975994
17/03/01 21:05:00 INFO CassandraConnector
spark_group_09
17/03/01 21:05:01 INFO SparkContext:
17/03/01 21:05:01 INFO SparkUI: Stopp
```

247 songs analyzed:

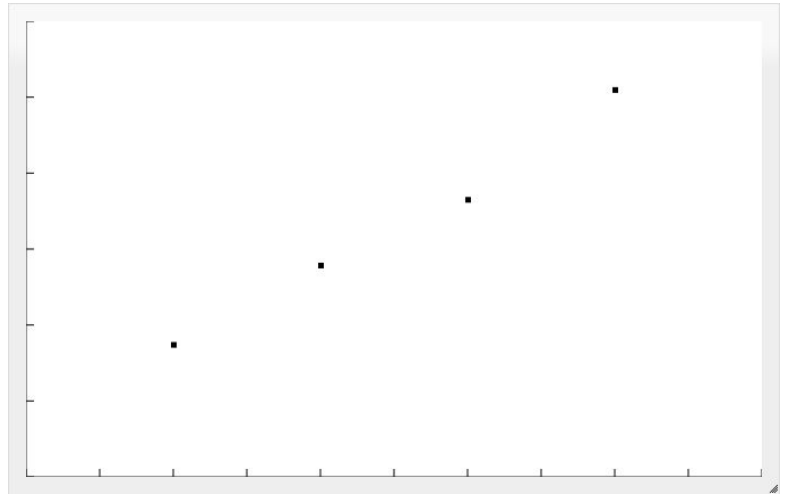
```
ala:37) finished in 0.092 s
17/03/01 21:01:40 INFO DAGScheduler: Job
ala:37, took 0.098193 s
Saved metadata
Total time: 182.798993826
Exception in thread Thread-1 (most likely
Traceback (most recent call last):
  File "/usr/lib64/python2.7/threading.p
  File "/usr/lib64/python2.7/threading.p
```

330 songs analyzed:

```
17/03/01 20:56:32 INFO DAGScheduler: Resu
ala:37) finished in 0.077 s
17/03/01 20:56:32 INFO DAGScheduler: Job
ala:37, took 0.080218 s
Saved metadata
Total time: 255.078511953
17/03/01 20:56:33 INFO CassandraConnector
spark_group_09
Exception in thread Thread-1 (most likely
Traceback (most recent call last):
```

Graph of Execution Time vs. Data Size:

As expected, this graph follows a linear increase with respect to data size.



Team Contributions

David Simon

I wrote most of the code involved with the key signature and chord progression analysis of the songs. I also did the timing of our code with the various data sizes and number of workers.

Brian Nguyen

I had to convert the numpy file we downloaded into smaller csv files so that Spark can read the data. I then cleaned up the csv files so that any rows with elements less than the ones in the cql table were deleted because they would cause null primary keys. I set up some code in which would allow Spark to read all the csv files.

Clayton Su

Cleaned up the csv files so that each dataset was put on a newline. I also made it so that each element in a dataset had quotes around them so that Spark would not throw an error saying that the element inputted in was not the right type. I also figured out how primary keys work.

Report 3

Requirements

Our requirement for this phase of the project was to make a basic website that connects with our Cassandra cluster and displays data from the cluster on the website.

Design

For our project, we created a virtual environment where our local website resides. The website is connected to the Cassandra cluster through the ssh tunneling command that is provided by the TAs. The website has basic functions to help the user search for a specific song based on a single piece of data that they provide.

Implementation

We used Django as our web framework for the website front end. Cassandra is the back end. We structured the website as a single page that links to other pages with the actual search form on it. E.g. Main page

- > Search by key
- > Search by composer
- > Search by chord
- > etc...

The website makes queries to Cassandra and Cassandra returns the results which the website displays.

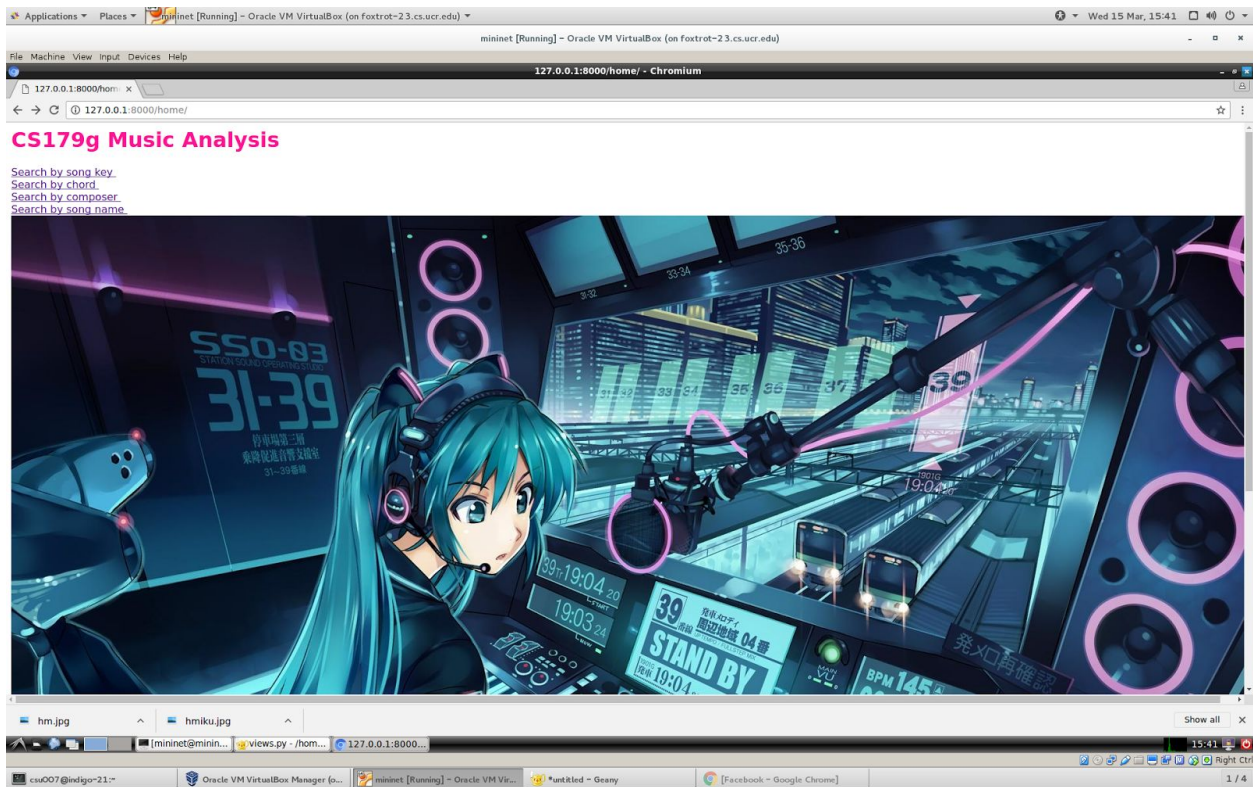
Evaluation

There are no revisions for part 1.

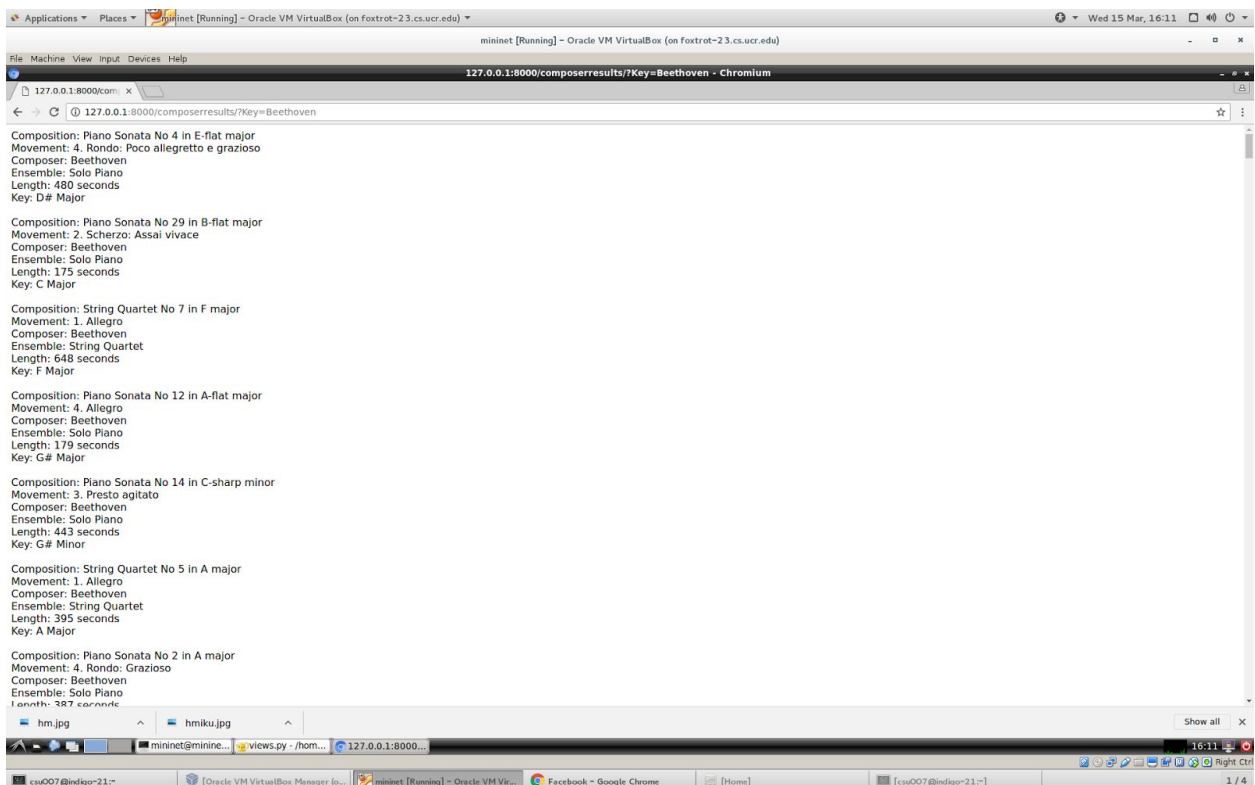
For the revisions of part 2, as suggested, we removed collect and only used Spark to process the data instead of preprocessing the data.

Although all the images inside the website are the same except for the homepage, the search form at the top right of every picture is different. You can see that the website searches based on criteria that the user provides such as "A minor", "Beethoven", and "1. Allegro".

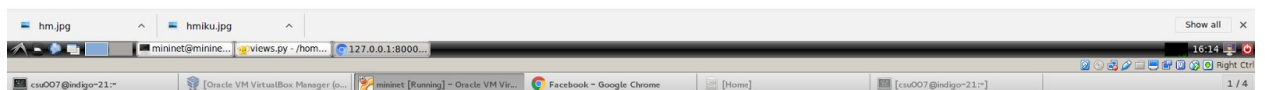
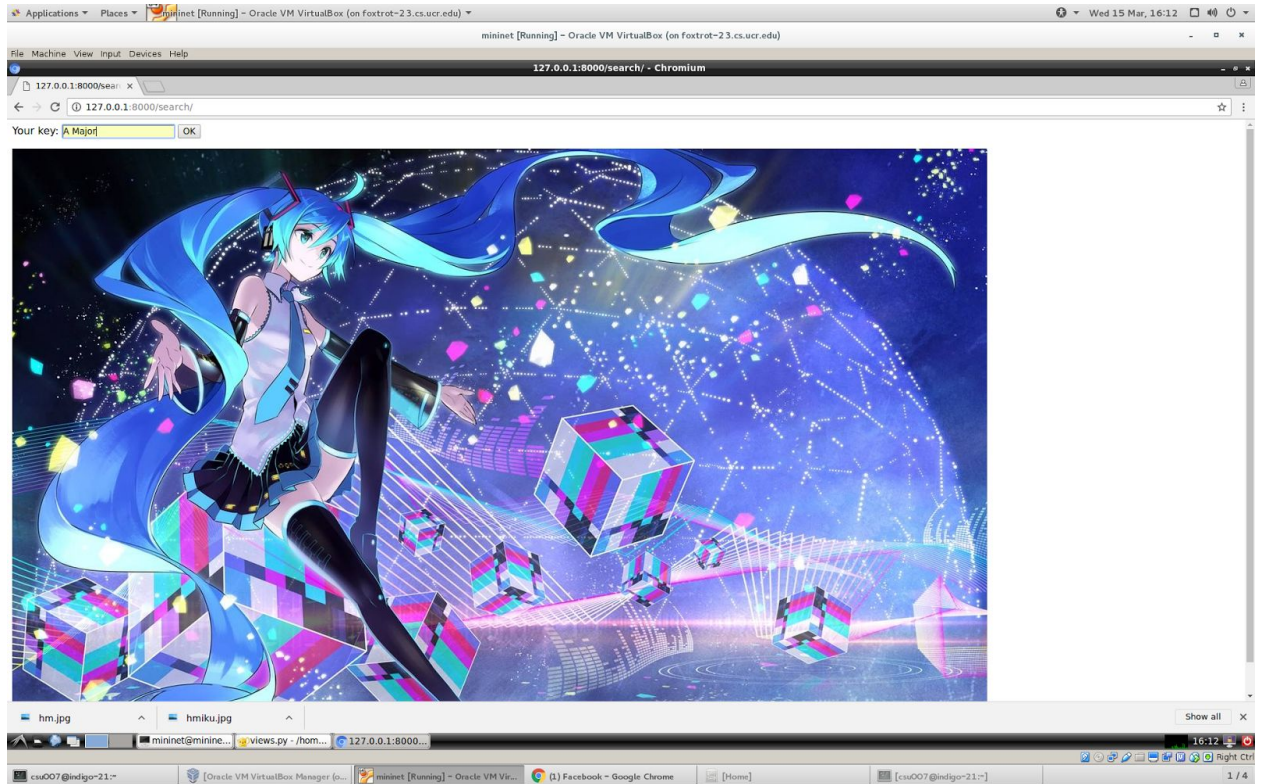
HomePage:



Search by composer page and results:



Search by song key and results:



Search by chord and results:


Applications ▾ Places ▾ mininet [Running] - Oracle VM VirtualBox (on foxtrot-23.cs.ucr.edu) Wed 15 Mar, 16:16

mininet [Running] - Oracle VM VirtualBox (on foxtrot-23.cs.ucr.edu)

127.0.0.1:8000/chord/ - Chromium

127.0.0.1:8000/chord/

Chord: Fm OK



hm.jpg hmiku.jpg

mininet@mininet... views.py /hom... 127.0.0.1:8000

csu007@indigo-21 ~ Oracle VM VirtualBox Manager (su... mininet [Running] - Oracle VM Vir... Facebook - Google Chrome [Home] csu007@indigo-21 ~ 1 / 4

Applications ▾ Places ▾ mininet [Running] - Oracle VM VirtualBox (on foxtrot-23.cs.ucr.edu) Wed 15 Mar, 16:17

mininet [Running] - Oracle VM VirtualBox (on foxtrot-23.cs.ucr.edu)

127.0.0.1:8000/chordresults/?Key=Fm - Chromium

127.0.0.1:8000/chordresults/?Key=Fm

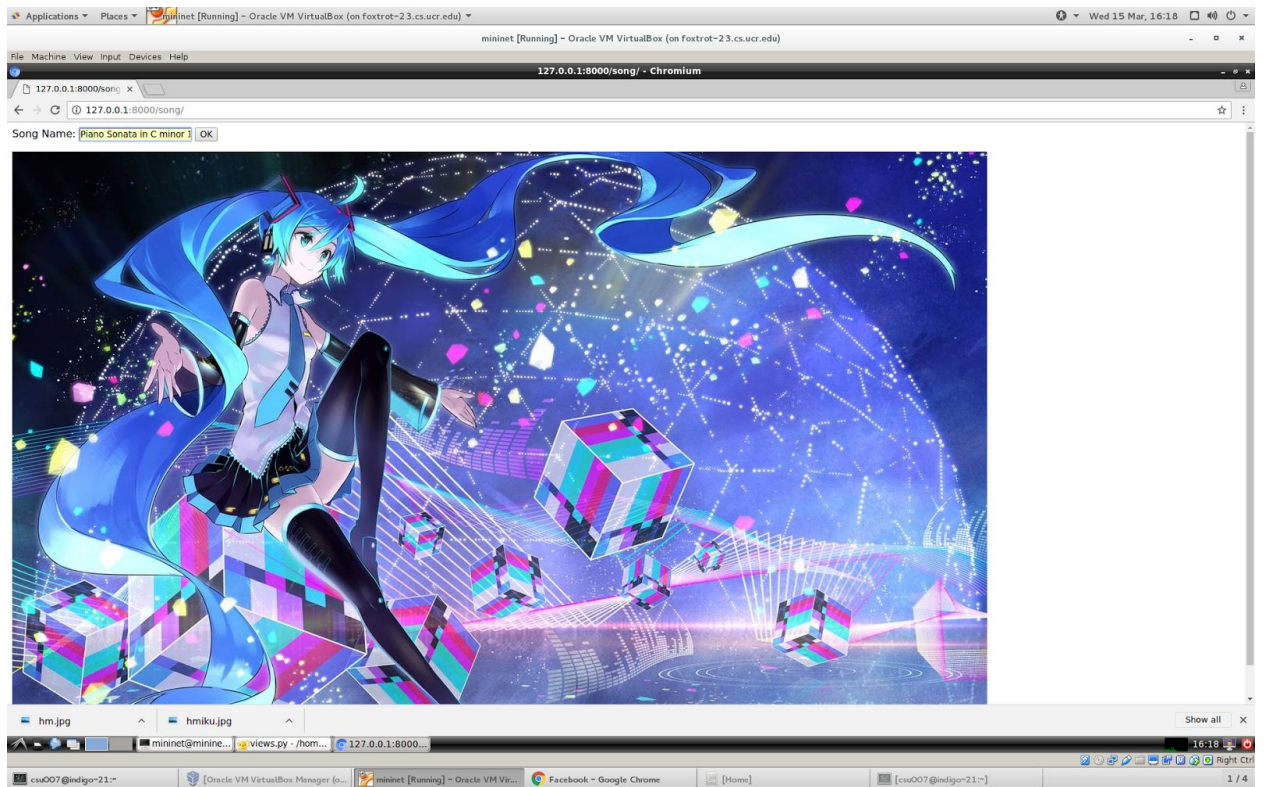
```
1739 Fm
1766 Fm
2411 Fm
2530 Fm
2161 Fm
2586 Fm
2325 Fm
1763 Fm
1766 Fm
1773 Fm
2118 Fm
2154 Fm
2322 Fm
2368 Fm
2422 Fm
2225 Fm
2116 Fm
2154 Fm
2116 Fm
2492 Fm
1923 Fm
1760 Fm
2570 Fm
1916 Fm
2411 Fm
2365 Fm
2463 Fm
2322 Fm
1819 Fm
1933 Fm
1763 Fm
1932 Fm
2078 Fm
2140 Fm
2112 Fm
2118 Fm
1923 Fm
2149 Fm
2149 Fm
2593 Fm
2403 Fm
2527 Fm
2588 Fm
2433 Fm
1919 Fm
2076 Fm
2104 Fm
```

hm.jpg hmiku.jpg

mininet@mininet... views.py /hom... 127.0.0.1:8000

csu007@indigo-21 ~ Oracle VM VirtualBox Manager (su... mininet [Running] - Oracle VM Vir... Facebook - Google Chrome [Home] csu007@indigo-21 ~ 1 / 4

Search by Song Name and Results:



Team Contributions

David Simon

-Did the revisions for part 2 which includes removing collect and doing all the processing in Spark.

- Coded the search bars for some of the webpages.
- Shared music knowledge

Brian Nguyen

- Set up the virtual environment for the website
- Created the base webpage that every other page is based on.
- Finalized and beautified the web pages

Clayton Su

- Helped research Django for the website.
- Helped program part of the website