Capstone

System Architecture

Sentiment Analysis and NLP

Matthew E. Miller, Manisha Kumari, Lucki Ratsavong
10-14-2019

Table of Contents

System Architecture:

Our system architecture is simple and concise. It implements the basic features of a web app. It has a web interface, administrative interface, remote database system, processing server, and third-party packages.

At a high level, the system receives user input (csv), sends that data via a JSON message, logs the data at a remote database, triggers a set of data-processing procedures, and then returns a processed response (JSON) to the user. The system's primary objective is to process users' data with natural language processing (NLP) and sentiment analysis, then return the results in a properly formatted report.

System Architecture:

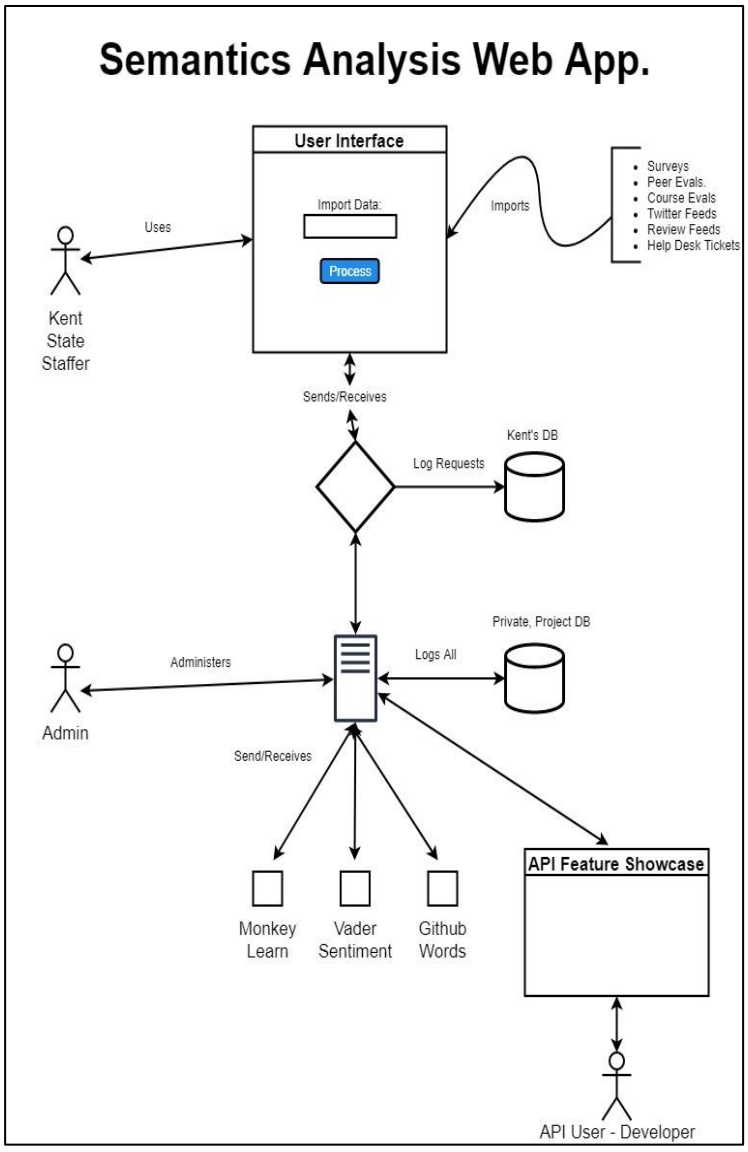

Figure 1

Specifications of Interfaces:

Here is a high-level specification of the system's interfaces. These interfaces will be described further in the next section.

Overview:

- The database systems are accessed and modified via SQL queries which are generated by Python scripts.
- The user interface is driven by HTML, CSS, and JavaScript. It's accessed via a web browser and relays formatted messages in JSON.
- The processing server has a listener script which waits for JSON messages, authenticates them, and logs them in the database system. When the processing server detects an incoming message, it creates, and submits the appropriate SQL queries to the database system.
- Administrators interface with the system via PythonAnywhere's user interface and remote terminals.
- The API interface is a series of python scripts which handle JSON requests and output JSON responses.

Database Schema:

The database schema is comprised of four tables. The tables are: Transaction, Body, User, and Data Type. These fields and their corresponding relationships are depicted below (*Figure 2*). Note, each database from *Figure 1* implement these schemas.

Current Version (Active):

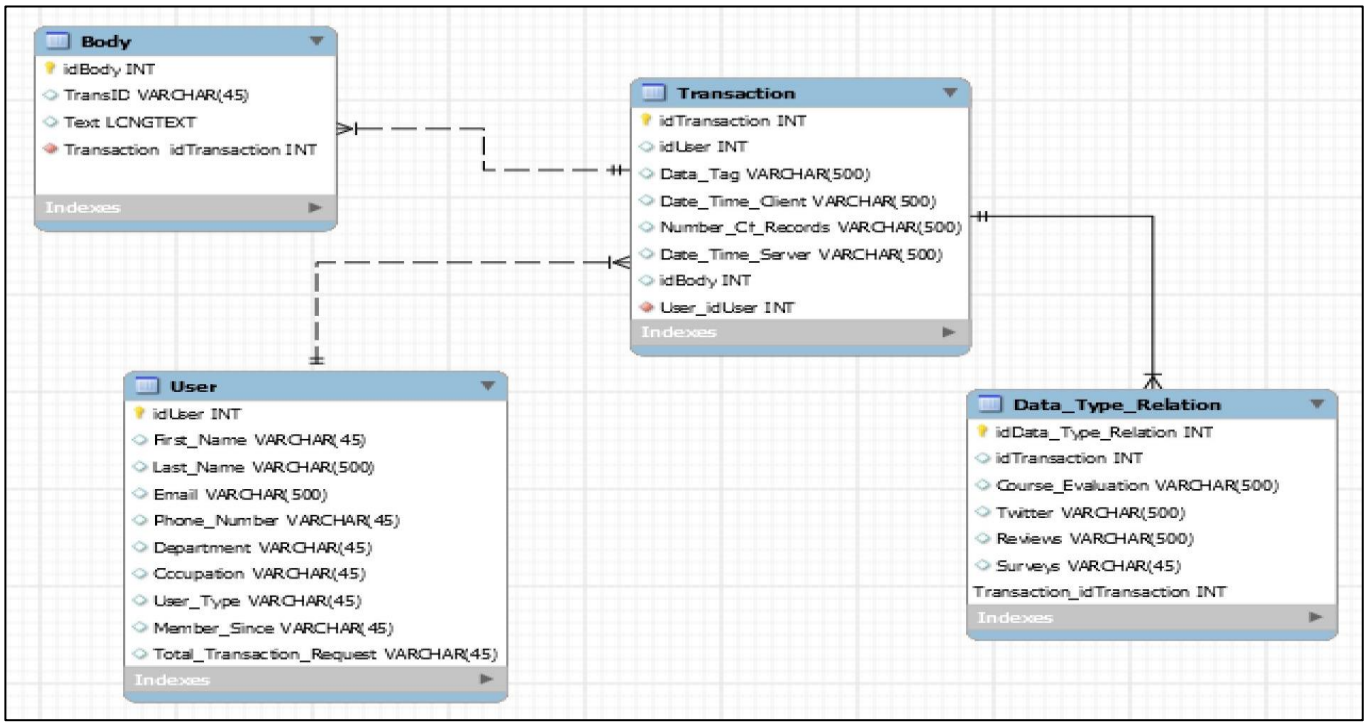

Figure 2

Previous Version (Deprecated):

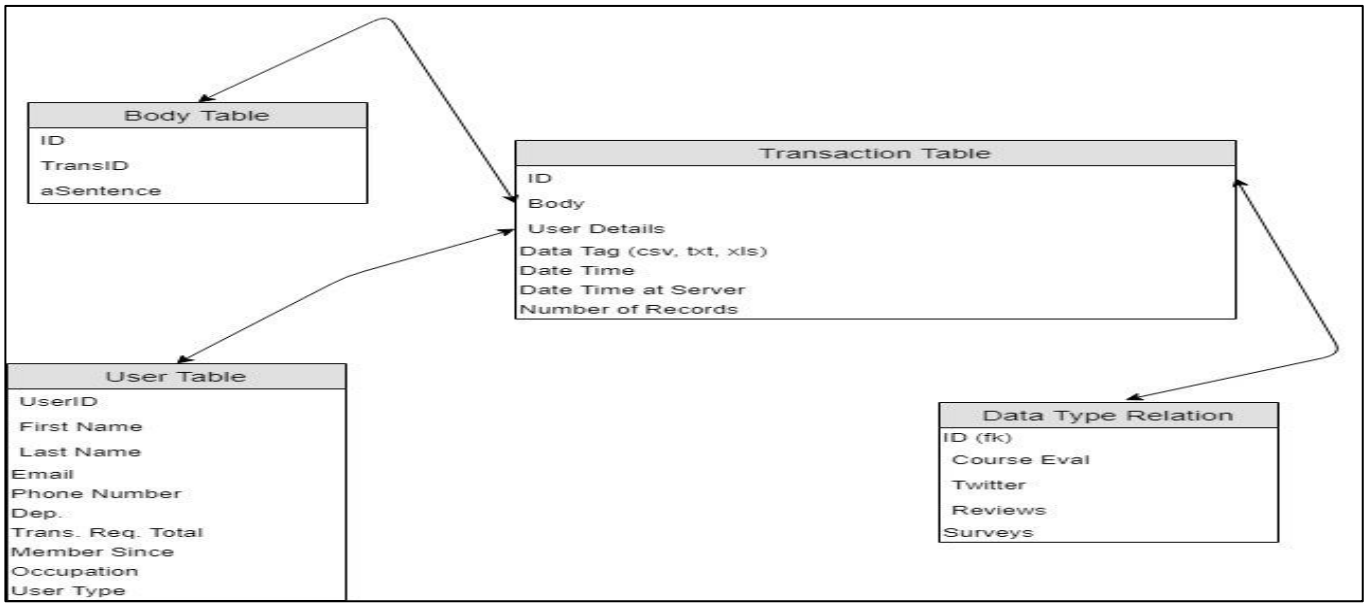

Figure 3

User Interface:

The user interface is structured across two pages, a landing page and an import page. The landing page is made of full-page segments which provide the basic information a user will need to accomplish their NLP processing task. The import page has a drag and drop zone for importing. It also has a file browser.

Here is a series of user interface screenshots.

Landing Page, Section One:

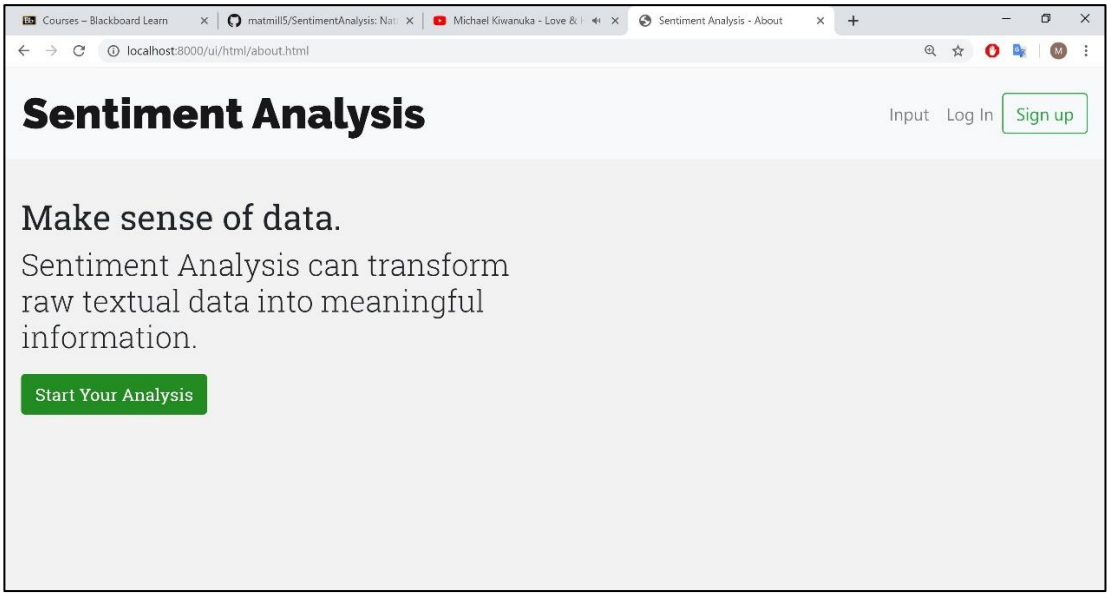

Figure 4

Landing Page, Section Two:

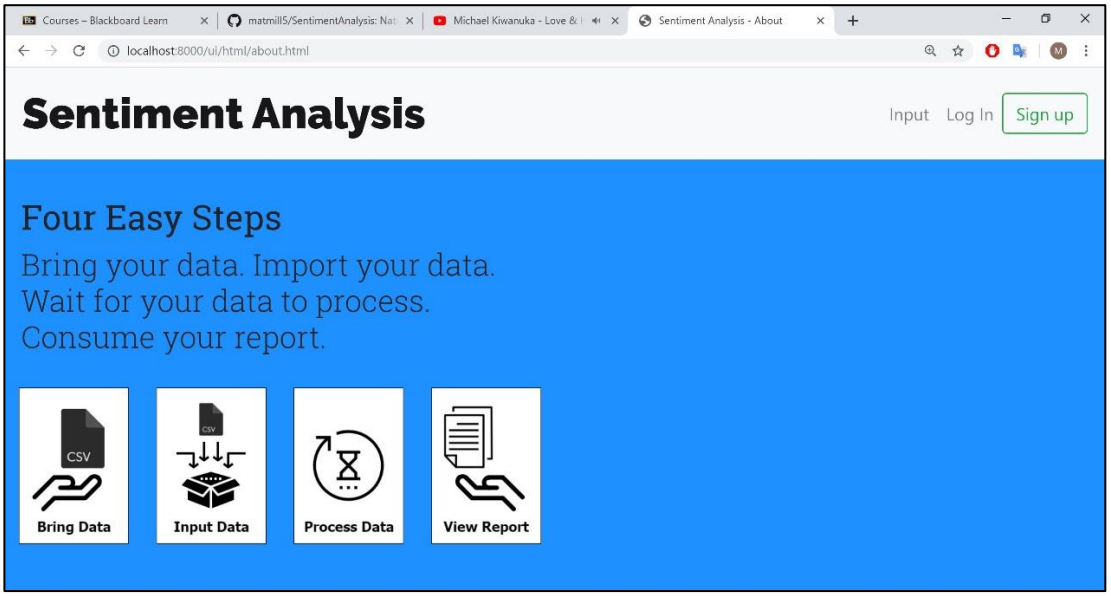

Figure 5

Landing Page, Section Three:

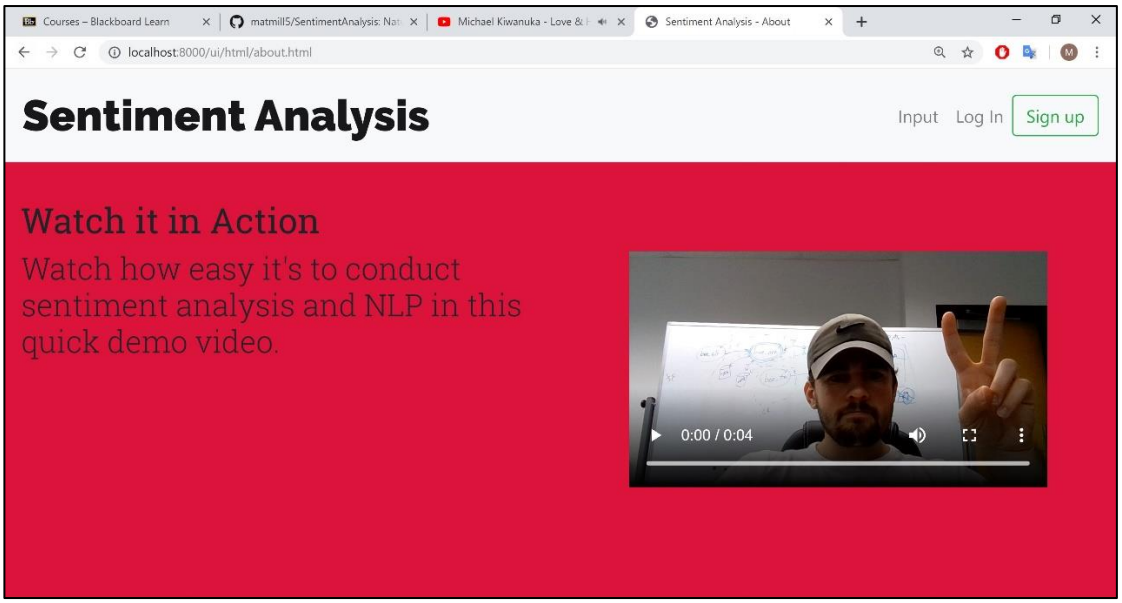

Figure 6

Landing Page, Section Four:

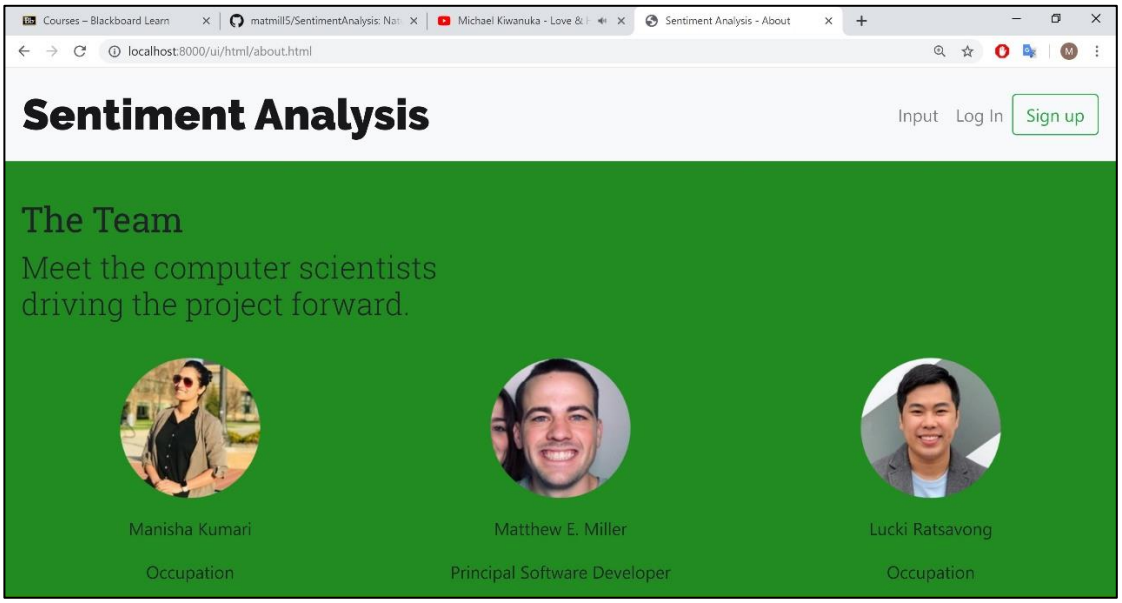

Figure 7

Landing Page, Section Final:

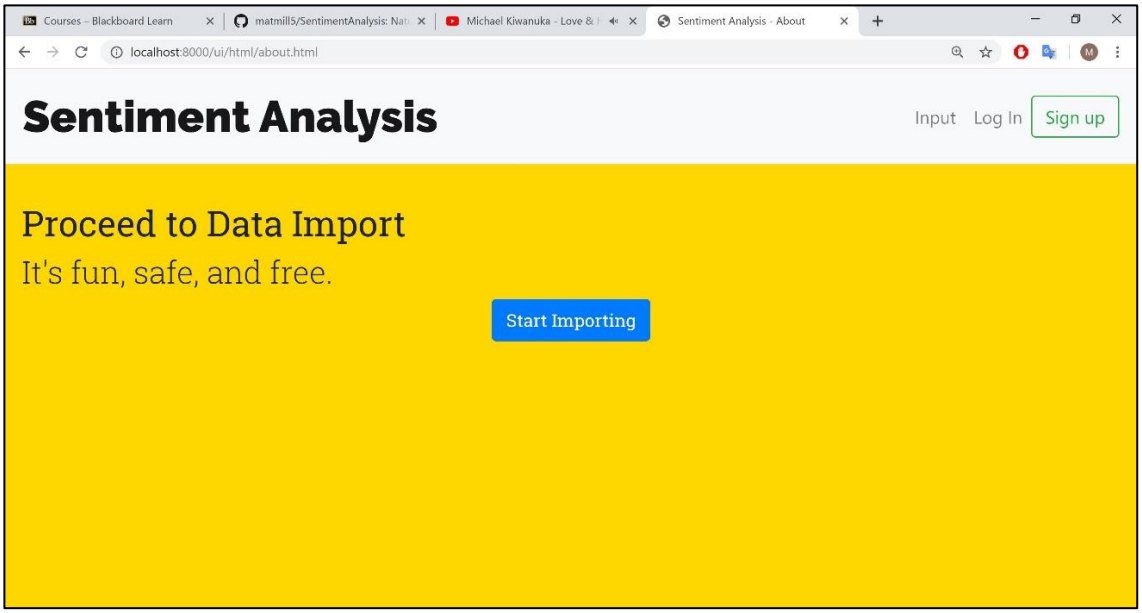

Figure 8

Import Page:

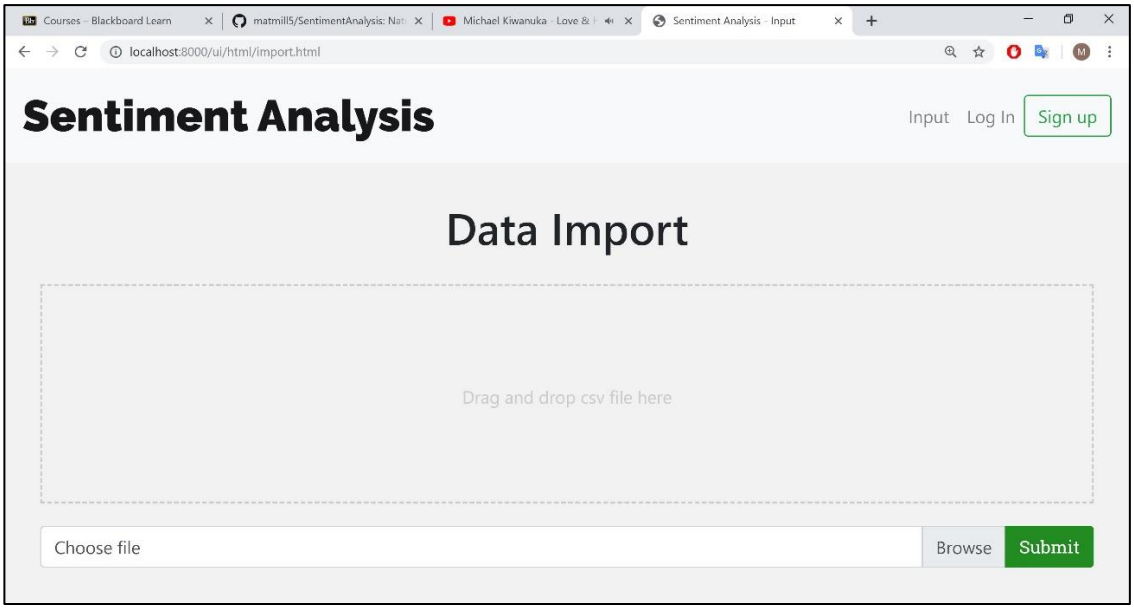

Figure 9

Message streams (Structure/Format):

Figure 10

Description:

In *Figure 10*, the message stream's structure and format are defined for the connection between our user interface and the processing server.

As you can see, there is a parent 'transaction' tag and its children tags. The children tags are id, user, body, dataTag, recordCount, clientTime, and serverTime.

Some of these children elements have their own subfields.

Response JSON:

```
{
  "transaction":{
    "id": "0001",
    "user": {
      "firstName": "John",
      "lastName": "Doe",
      "email": "jdoe@kent.edu",
      "phone": "111-000-0000",
      "dep": "CompSci",
      "Occupation": "Dep. Chair",
      "userType": "Customer"
    },
    "body": {
        "sentenceList": [
          {
          "sentenceId": "1",
          "text": "blah",
          "tag": "pos/neg/neu"
          },
          {
          "sentenceId": "2",
          "text": "Blahhh",
          "tag": "pos/neg/neu"
          },
          {
          "sentenceId": "3",
          "text": "Blah Blah Blah",
          "tag": "pos/neg/neu"
          }]
    },
    "result": {
      "complete": "bool",
      "errors": {
        "errorList": {
        }
      }

    }
  }
}
```

Ln: 9 Col: 24

Description:

The response from the processing server, post-processing is comprised of a parent field, 'transaction', and four children. These children fields are id, user, body, and result.

Performance, Reliability, Design:

To ensure the web application is able to process the required amount of user requests, I have upgraded our subscription to the remote hosting service. We now have extended CPU time, increased storage capacity, SSH access to the DB system, and improved priority in the service queue.

The other elements of our system have their own ways of handling multiple sessions. For example, the DB-system employs locking mechanisms and the transaction schema ensures we can queue requests/responses effectively.

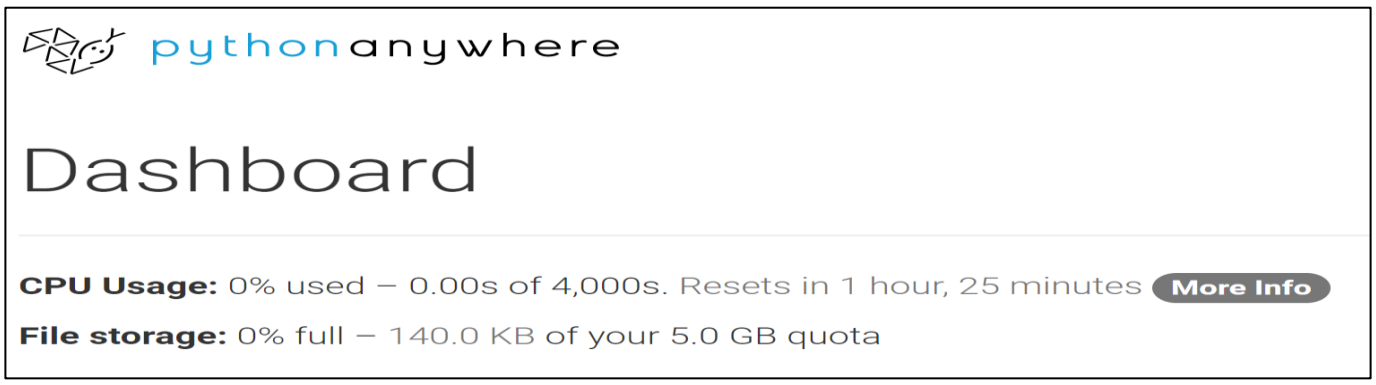

Figure 11

Conclusion:

Our system is going to have a forward-facing web interface, use JSON messages to relay information across the internet, and log transactions in our remote database system using SQL queries. Our administrators will use PythonAnywhere's user interface and remote terminal service to administrate our web application. Our remote Python scripts will listen for API-users' requests and serve up the appropriate responses.

The final result will be an interconnected system architecture based on web technologies like HTML, CSS, JavaScript, JSON, and MySQL. External packages will be accessed via RESTful APIs, CDNs, and local imports.

Thanks for reading.