

## **1.0 IMPLEMENTATION**



# **1. IMPLEMENTATION**

## **1.1 FEATURES**

The implementation of this project is divided into two parts. The first part consists of server side implementation and the second part is client side implantation.

### **1.1.1 Part 1**

The first part scrapes the data from California State University - Fullerton catalog, insert the data in PostgreSQL on google cloud and create endpoints to have the data access over HTTP. The first part does the following:

1. Crawl over university catalog and scrape the useful information.
2. Build object relationship model (ORM) to build the relationship between entities.
3. Open connection with PostgreSQL database on Google App Engine.
4. Insert the data into the PostgreSQL database.
5. Once the data is available on Google Cloud, it creates google endpoints (RESTful services) to make the data available on client.

### **Steps to setup**

For development purposes - In order to have the local development environment setup, following steps needs to be performed:

#### **1. Prerequisites**

The project scrapes data from the California State University catalog [6] and insert it on google cloud PostgreSQL. Therefore, this project requires you to have a google cloud project with PostgreSQL service enabled. Refer google cloud documentation [7] to create a google cloud project and enable required services.

#### **2. Clone the repository**

Once you have satisfied the prerequisites, clone this repository (puneetjaincsuf/chatbotcsuf) on your local machine.

---

### 3. Set class path

Set the following variables into your class path:

**Unix:**

```
export  
SQLALCHEMY_DATABASE_URI=postgresql+psycopg2://[USER_NAME]:  
[PASSWORD]@127.0.0.1:5432/[DATABASE_NAME]
```

**Windows:**

```
SET  
SQLALCHEMY_DATABASE_URI=postgresql+psycopg2://[USER_NAME]:  
[PASSWORD]@127.0.0.1:5432/[DATABASE_NAME]
```

### 4. Virtual Environment setup

Create a virtual environment and install all the dependencies given in requirements.txt file in the project. Refer [8] to know how to setup and install dependencies in a virtual environment.

### 5. Create schema and insert data on google cloud

Run *python scraperdboperations.py* from console/terminal by going into the project directory. This will create the schema and insert all data in the database on google cloud.

### 6. OpenAPI.yaml changes

Replace the value of *host:* variable with the google cloud project URL.

### 7. Deploy services on google cloud

Run *gcloud endpoints services deploy openapi-appengine.yaml* command from project home directory. This will deploy your endpoints on google cloud. Please make sure you have google cloud SDK installed and configured on your local machine. Refer to google cloud documentation [9]. Execution of this command will generate the project name and configuration id. Please note them as you will use them in the next step.

### 8. Now you are ready to deploy the project on google cloud

In order to deploy, you need to change the endpoint API services configuration in api.yaml file. Open the api.yaml file and change the following with the project name and configuration id you got in step 7.

---

```
endpoints_api_service:  
  name: [Name]  
  config_id: [CONFIG_ID]
```

After the successful deployment hit following endpoints, and you should be able to see the college data:

```
https://[Project URL]/colleges/<college_name>
```

```
https://[Project URL]/departments/<department_name>
```

```
https://[Project URL]/programs/<program_name>
```

```
https://[Project URL]/specificcourses/<course_name>
```

```
https://[Project URL]/generalcourses/<course_name>
```

### 1.1.2 Part 2

The second part of this project is the client side implementation. It consumes the google endpoints created in part 1. These endpoints can be called over HTTP using `httprequest`. For this particular project, I am using dialogflow and cloud functions for firebase. Part 2 expect you to have a agent created on dialogflow. Refer [10] to create an agent on dialogflow and then refer the following steps to setup part 2 in your local development environment.

#### 1. Upload intents on your Dialogflow agent

The first step in part 2 requires to upload intents to dialogflow agent. Click on upload intents on your dialog flow console and upload all the intent present under `\client\google-assistant\intents` in the repository.

#### 2. Deploy firebase functions on Dialogflow

Before you upload firebase functions, make sure you have webhook enabled for your dialogflow agent. Once you have that, do the following:

1. Open `/client/google-assistant/firebase/functions/config/config.js` and change the `config.serviceURL` to your google cloud project URL.
  2. Open terminal and navigate to `/client/google-assistant/firebase/functions`. Run the following command to deploy the firebase function: `firebase deploy --only functions`
-

Now you can test the app by asking questions like:

"does csuf has business course"

"what is the prerequisite for CPSC 481"

## 1.2 Entity Relationship Diagram

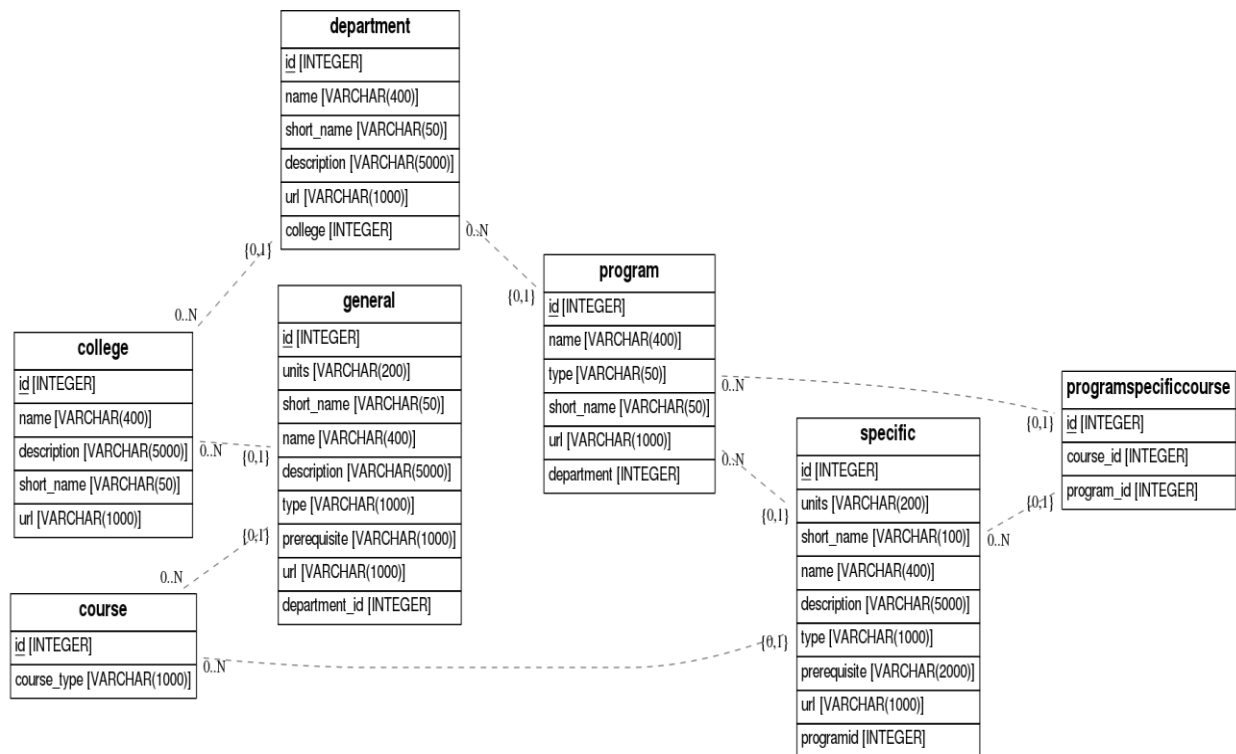


Figure-1 Entity Relationship Diagram

System is very modular in designed in object oriented approach.

The whole system consists of 7 major entities having various relationship with each other.

- 1. College:** Each college entity has one-to-many relationship with department. In short, a college can have many departments, but a department can have only one college. For example, College of Engineering and Computer Science has many departments such as, Computer Science, Civil Engineering, Computer Engineering, Electrical Engineering, Mechanical Engineering.

2. **Department:** Department maintain a one-to-many relationship with Program. Each department can have many programs, but a program must belong to one department only. For example, Computer Science department has Master of Science, Bachelor of Science, Software Engineering programs.
3. **Program:** Programs has one-to-many relationship with Specific Courses. Each program can have multiple specific courses. For example, Bachelor of Science in Computer Science can have courses such as, Cloud Computing, Computer Communication, Computer Networks, Parallel and Distributed Computing and many more.
4. **Course:** Course is basically and interface which defines the category of a course. In general, a course can either be a specific course – a course which belongs to a *program* – or it can be a general course – a course which belongs to *college*.
5. **General:** General is a category of course. A general course belongs to a college rather than a program. For example, EGGN 495 Professional Practice.
6. **Specific:** Specific is another category of course. A specific course belongs to a program. For example, CPSC 454 Cloud Computing and Security belongs to Bachelor of Science in Computer Science.
7. **ProgramSpecificCourse:** ProgramSpecificCourse is basically a placeholder which tells which courses belongs to program.

### 1.3 Workflow/Architecture

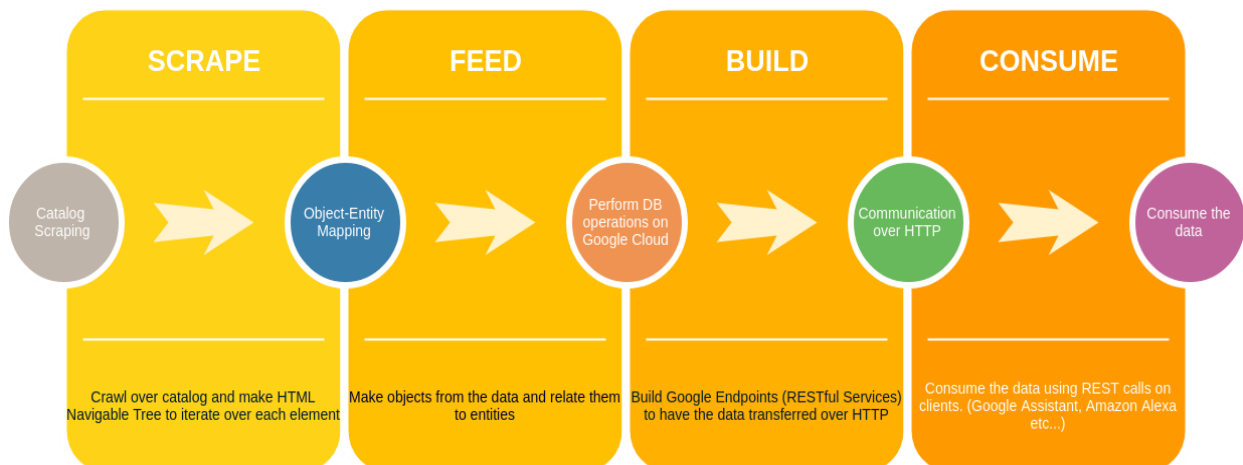


Figure-2 Workflow

The workflow of the project starts from scraping the university catalog. As part of scraping it crawl over the catalog and find relevant information and build navigable tree of HTML elements. Once the scraping is done, it builds relationships between the objects and creates object relational models (ORM). After building the ORM models, it performs database operations in PostgreSQL database on google cloud. Eventually, it builds google endpoint (RESTful Services) to consume the data on client over HTTP.

## 1.4 Source Code

The complete source code of the project is available on my GitHub Repository - *puneetjaincsuf*. The repository is currently set to private. Send an email to get access. Following are some of the important snippets and business logics.

```
def create_colleges():
    try:
        colleges = College.get_colleges();
        for url, clg in colleges.items():
            college = models.College(name=clg.get()[0],
description=clg.get()[1], short_name="", url = url)
            models.db.session.add(college)
        except Exception as e:
            print(e)
```

Above snippet get the college data from the scrapper and insert it on google cloud. Each entity (Department, Course, Program) has a separate method which gets relevant data from scraper and insert it on google cloud.

```
class College(db.Model):
    id = db.Column(db.Integer, nullable=False, primary_key=True)
    name = db.Column(db.String(400))
    description = db.Column(db.String(5000))
    short_name = db.Column(db.String(50))
    url = db.Column(db.String(1000))
    department = db.relationship('Department', backref='department_owner',
lazy='dynamic')
    general_course = db.relationship('General',
backref='general_course_owner', lazy='dynamic')
```

Above snippet is a model class for college. Each entity (Department, Course, Program) has a separate model class that builds the object relationship mapping between the entities.

```
def get_colleges():
```

---

```

college_dict = {}
try:
    page = requests.get(const.BASE_URL)
    tree = html.fromstring(page.content)
    colleges = tree.xpath(const.COLLEGE_XPATH)
    pool = mp.Pool()
    for i in range(len(colleges)):
        college_text = colleges[i].text
        if const.COLLEGE in college_text:
            college_url = colleges[i].attrib[const.HREF]
            args = (college_url, college_text)
            college_dict[college_url] = pool.apply_async(__scrape_colleges,
args)
    except Exception as e:
        print(e)
    return college_dict

```

```

def __scrape_colleges(college_url, college_name):
    try:
        college_list = []
        college_page = requests.get(const.BASE_URL + const.SLASH +
college_url)
        college_tree = html.fromstring(college_page.content)
        desc = college_tree.xpath(const.COLLEGE_DESC_XPATH)
        if len(desc) > 0:
            description = desc[0].text
            college_list.append(college_name)
            college_list.append(description[0:description.find('.')])
        return college_list
    except Exception as e:
        print(e)

```

Above snippet is scrapper for college entity. Each entity (Department, Course, Program) has its own scrapper. These scrappers are data independent and runs in parallel to make use of all the system cores.

```

def get_college_by_name(college_name):
    looking_for = '%{0}%'.format(college_name)
    data =
models.College.query.filter(models.College.name.ilike(looking_for))
    college_schema = models.CollegeSchema(many=True)
    college = college_schema.dump(data).data
    logging.debug(college)
    return college

```

---



Above snippet is a typical example of a service method for college entity. This service finds the college by a given name and returns it to the user. Each entity (Department, Course, Program) has its own service. Essentially these services get called from client as HTTP request.

## **2.0 CHALLENGES AND PERFORMANCE IMPROVEMENTS**

## 2. CHALLENGES AND PERFORMANCE IMPROVEMENTS

Overall the project was quite challenging especially scraping the catalog. Since the university catalog was inconsistent and all the colleges have different format of rendering data, it was quite difficult to write a generic code to scrape the whole catalog.

Initially, the scraper was taking about 60 minutes to scrape the whole catalog and insert the data in PostgreSQL on Google Cloud. Later, I implemented multiprocessing on code to have the scraper run on all the cores. After the multiprocessing implementation, the scraper finished its execution in about 20 minutes.

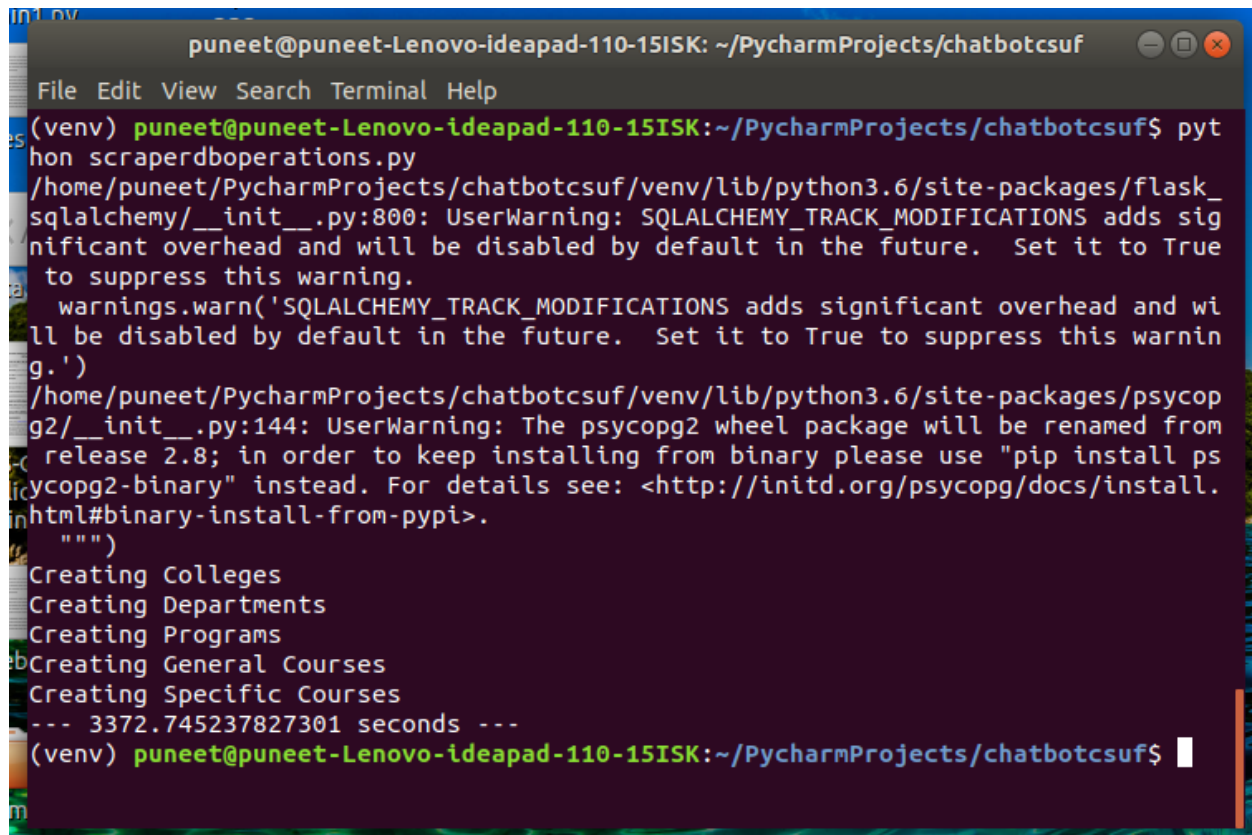
### Sequential Execution

Processor: Intel Core i3 6<sup>th</sup> Generation

Memory: 6 GB

Cores: 4

Total Execution Time: 56 Minutes



```
puneet@puneet-Lenovo-ideapad-110-15ISK: ~/PycharmProjects/chatbotcsuf
File Edit View Search Terminal Help
(venv) puneet@puneet-Lenovo-ideapad-110-15ISK:~/PycharmProjects/chatbotcsuf$ python scraperdboperations.py
/home/puneet/PycharmProjects/chatbotcsuf/venv/lib/python3.6/site-packages/flask_sqlalchemy/__init__.py:800: UserWarning: SQLALCHEMY_TRACK_MODIFICATIONS adds significant overhead and will be disabled by default in the future. Set it to True to suppress this warning.
  warnings.warn('SQLALCHEMY_TRACK_MODIFICATIONS adds significant overhead and will be disabled by default in the future. Set it to True to suppress this warning.')
/home/puneet/PycharmProjects/chatbotcsuf/venv/lib/python3.6/site-packages/psycpg2/__init__.py:144: UserWarning: The psycopg2 wheel package will be renamed from release 2.8; in order to keep installing from binary please use "pip install psycopg2-binary" instead. For details see: <http://initd.org/psycopg/docs/install.html#binary-install-from-pypi>.
  """
Creating Colleges
Creating Departments
Creating Programs
Creating General Courses
Creating Specific Courses
--- 3372.745237827301 seconds ---
(venv) puneet@puneet-Lenovo-ideapad-110-15ISK:~/PycharmProjects/chatbotcsuf$
```

Figure-3 Sequential Scraper Execution

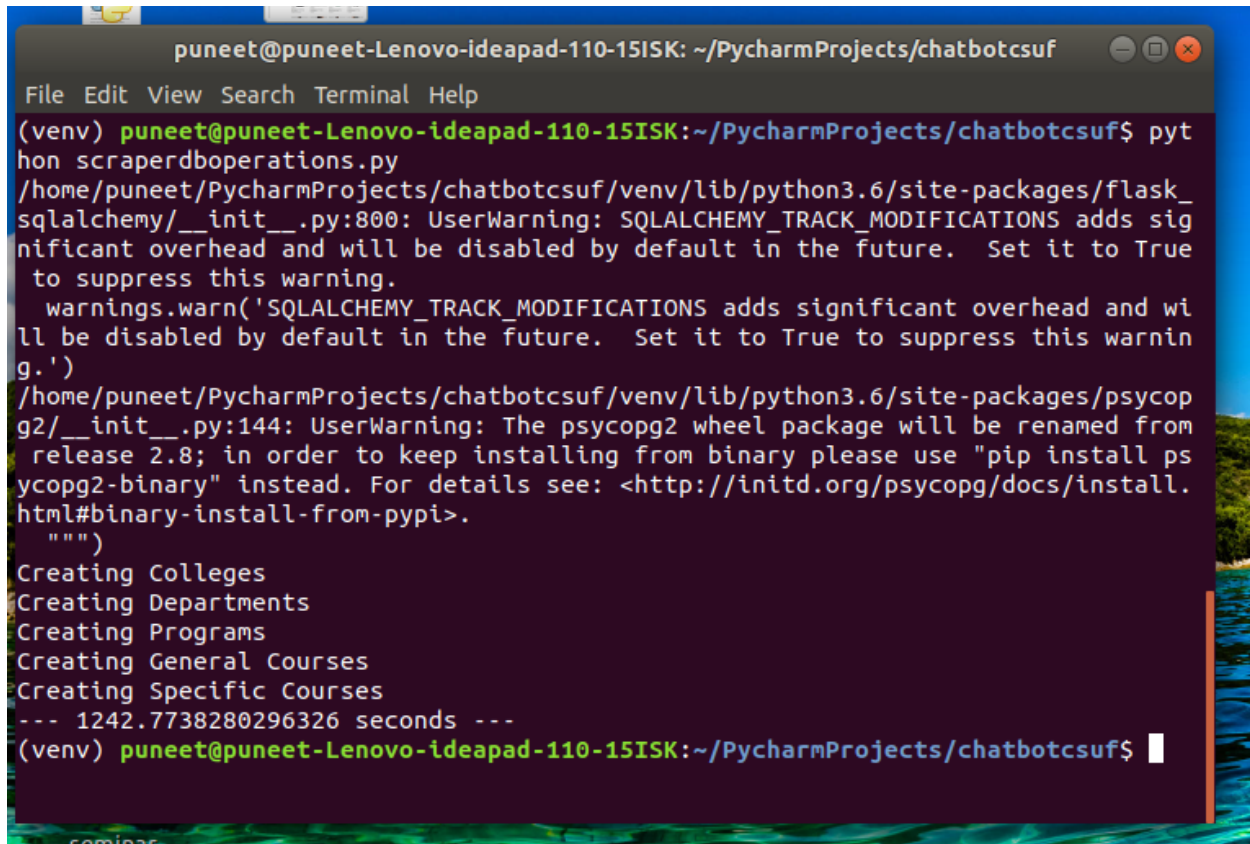
## Parallel Execution

Processor: Intel Core i3 6<sup>th</sup> Generation

Memory: 6 GB

Cores: 4

Total Execution Time: 20 Minutes



```
puneet@puneet-Lenovo-ideapad-110-15ISK: ~/PycharmProjects/chatbotcsuf
File Edit View Search Terminal Help
(venv) puneet@puneet-Lenovo-ideapad-110-15ISK:~/PycharmProjects/chatbotcsuf$ python scraperdboperations.py
/home/puneet/PycharmProjects/chatbotcsuf/venv/lib/python3.6/site-packages/flask_sqlalchemy/__init__.py:800: UserWarning: SQLALCHEMY_TRACK_MODIFICATIONS adds significant overhead and will be disabled by default in the future. Set it to True to suppress this warning.
  warnings.warn('SQLALCHEMY_TRACK_MODIFICATIONS adds significant overhead and will be disabled by default in the future. Set it to True to suppress this warning.')
/home/puneet/PycharmProjects/chatbotcsuf/venv/lib/python3.6/site-packages/psycopg2/__init__.py:144: UserWarning: The psycopg2 wheel package will be renamed from release 2.8; in order to keep installing from binary please use "pip install psycopg2-binary" instead. For details see: <http://initd.org/psycopg/docs/install.html#binary-install-from-pypi>.
  """
Creating Colleges
Creating Departments
Creating Programs
Creating General Courses
Creating Specific Courses
--- 1242.7738280296326 seconds ---
(venv) puneet@puneet-Lenovo-ideapad-110-15ISK:~/PycharmProjects/chatbotcsuf$
```

Figure-4 Parallel Scraper Execution

### **3.0 REFERENCES**

### 3. REFERENCES

- [1] Ferrari, A., Donati, B., & Gnesi, S. (2017). Detecting Domain-Specific Ambiguities: An NLP Approach Based on Wikipedia Crawling and Word Embeddings. 2017 IEEE 25th International Requirements Engineering Conference Workshops (REW). doi:10.1109/rew.2017.20
  - [2] Fujita, T., Bai, W., & Quan, C. (2017). Long short-term memory networks for automatic generation of conversations. 2017 18th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD). doi:10.1109/snpd.2017.8022766
  - [3] Long short-term memory. (n.d.). In Wikipedia. Retrieved November 08, 2017, from [https://en.wikipedia.org/wiki/Long\\_short-term\\_memory](https://en.wikipedia.org/wiki/Long_short-term_memory)
  - [4] Milhorat, P., Schlogl, S., Chollet, G., Boudy, J., Esposito, A., & Pelosi, G. (2014). Building the next generation of personal digital Assistants. 2014 1st International Conference on Advanced Technologies for Signal and Image Processing (ATSIP). doi:10.1109/atsip.2014.6834655
  - [5] Yu, F., & Zheng, D. (2017). Education data mining: How to mine interactive text in MOOCs using natural language process. 2017 12th International Conference on Computer Science and Education (ICCSE). doi:10.1109/iccse.2017.8085582
  - [6] Catalog.fullerton.edu. (2017). California State University Fullerton - Acalog ACMS™. [online] Available at: <http://catalog.fullerton.edu> [Accessed 16 May 2018].
  - [7] Google Cloud. (2018). Using Cloud SQL for PostgreSQL|App Engine flexible environment for Python docs|Google Cloud. [online] Available at: <https://cloud.google.com/appengine/docs/flexible/python/using-cloud-sql-postgres> [Accessed 16 May 2018].
  - [8] Installing packages using pip and virtualenv (n.d.). Retrieved from <https://packaging.python.org/guides/installing-using-pip-and-virtualenv/>
  - [9] Download and Install the SDK for App Engine (n.d.). Retrieved from <https://cloud.google.com/appengine/downloads/>
  - [10] Agents (n.d.). Retrieved from <https://dialogflow.com/docs/agents>
  - [11] Firebase CLI Reference (n.d.). Retrieved from <https://firebase.google.com/docs/cli/>
-