# Drexel Chatbot Design Document

Hoa Vu <htv27@drexel.edu>
Tom Amon <tpa27@drexel.edu>
Daniel Fitzick <dwf35@drexel.edu>
Aaron Campbell <ajc382@drexel.edu>
Nanxi Zhang <nz66@drexel.edu>
Shishir Kharel <sk3432@drexel.edu>

Version: 1.0

# 1. Revision History

| Name(s) | Date | Comment(s) | Version |
|---|---|---|---|
| Everyone | 01/22/17 | Preliminary Document Structure | 0.1 |
| Everyone | 01/29/17 | First version | 0.2 |
| Everyone | 2/1/17 | Added component design | 0.3 |
| Everyone | 2/5/17 | Added UML | 0.4 |
| Everyone | 2/6/17 | Added most of the information to component design | 0.5 |
| Everyone | 2/9/17 | Finalize Rough Draft | 0.6 |
| Everyone | 2/12/17 | Review first four sections | 0.7 |
| Everyone | 2/13/17 | Finalize document | 1.0 |

# 2. Introduction

## 2.1 Purpose

This software design document describes the architecture and system design of Drexel Chatbot, a question answering system for the Drexel community. It is intended to outline the system structure for the project manager and stakeholder, and provide technical guidance to the development team.

## 2.2 Scope

Drexel Chatbot (Drexel natural language query service) is an AI chatbot that receives questions from users, tries to understand the question, and provides appropriate answers. It does this by converting an English sentence into a machine-friendly query, looking up in the database for necessary information to answer the question, and finally returning the answer in a natural language sentence.

The main objective is to develop a Web API that provides such natural language query service. In addition, sample web, mobile, and text messaging interfaces will be created to demonstrate the use of the API.

The goal is to provide Drexel students and faculty a quick and easy way to have their questions answered, as well as to offer other developers the means to incorporate Drexel Chatbot into their projects.

## 2.3 Overview

**1. Revision History**: Provides the date of, reason for, and people who were involved with the modification of this document.

**2. Introduction**: Provides an overview of the application, explains the objectives and goal of the project and describes the document structure.

**3. System Overview**: Gives a general description of the functionality, context and design of Drexel Chatbot.

**4. System Architecture**: Breaks the project down into various subsystems, defines how those subsystems interact, and provides UML for each subsystem.

**5. Data Design**: Describes the organization of data in the Stardog database implemented for Drexel Chatbot.

**6. Component Design**: Summarizes the functions of each component listed in system architecture.

**7. Human Interface Design**: Describes the user interface of the sample applications.

**8. Requirements Matrix**: Provides a cross reference that traces components and data structures to the requirements in Software Requirements Specifications document.

## 2.4 Definitions and Acronyms

**Chatbot**: An interface, usually text based, specializing in the mimicry of natural language conversation. AKA “artificial conversational entity.”

**Generic representation**: A generic word that is used to replace a more definite word in a sentence. I.e. In the sentence “who is $professor”, $professor is a generic representation for an actual professor’s name, like Augenblick.

**GUI**: Graphic User Interface, a type of user interface that allows users to interact with the software through graphical icons (e.g. buttons, etc.).

**HTML**: Hypertext Markup Language, a standardized system for tagging text files to achieve font, color, graphic, and hyperlink effects on webpages.

**JSON**: JavaScript Object Notation, a data-interchange format that is commonly used in exchanging data over the Internet.

**Pagerank**: PageRank is an algorithm used by Google to rank websites. It works by counting the number and quality of links to a page to determine a rough estimate of how important the website is. The underlying assumption is that more important websites are likely to receive more links from other websites.

**SMS**: Short Message Service, the text messaging protocol of cellular telephones.

**Standard English**: the language that can be understood by English-speaking high school graduates.

**Stardog**: an enterprise data unification platform leveraging smart graph technology.

**UML**: Unified Modeling Language is a general-purpose, developmental, modeling language in the field of software engineering, that is intended to provide a standard way to visualize the design of a system.

**URL**: Uniform Resource Locator, an address to a resource on the Internet.

**URL Parameter**: parameters whose values are set in a webpage's URL.
Web API: an application programming interface (API) for either a web server.

**Web Scraping**: web scraping is a technique employed to extract large amounts of data from websites whereby the data is extracted and saved to a local file in your computer or to a database.

# 3. System Overview

## 3.1 Context

Most of the search engines today, like Google, use a system (The Pagerank Algorithm) to rank different web pages. When a user enters a query, the query is interpreted as keywords and the system returns a list of highest ranked web pages which may have the answer to the query. Then the user must go through the list of webpages to find the answer they are looking for. Drexel Chatbot, however, will try to understand the query and provide a definitive answer.

## 3.2 Product Features

The major features for Drexel Chatbot will be the following:

**Web API**: an API call will include a question in the form of a query string url parameter and the service will reply in JSON.
**Natural Language Processing**: the system will take in questions written in standard English.
**Natural Language Responses**: the answer to the question will be written in standard and understandable English.
**Information Extraction**: there will be a database containing all the information needed, populated using information extraction techniques.

## 3.3 User Classes and Characteristics

The two classes of users for this system are described below:

**External Developers**: these users consist of application developers who want to incorporate Drexel Chatbot API into other software applications.

**Non-technical Users**: these users consist of non-technical users who want to get answers for their questions. These users ask questions and get answers with mobile, web, or text messaging interfaces. This class of users include Drexel's current and prospective students, teaching faculty, and staff.

## 3.4 Technologies Used

The Web API will be developed in Java, and the backend in Python. The front-end web application will be developed using HTML and Javascript, and the Android application using Java and the Android SDK.

The system has a few external libraries that it relies on:

**Spring MVC**: Java library that dispatches requests to handlers based around the model-view-controller type of architecture. The controller mechanism allows for the creation of RESTful web services and applications alike.
**RDFLib**: Python library used to represent the database as an object that facilitates the execution of queries.
**NLTK**: Natural language toolkit. A Python Library used to parse sentences.
**Keras**: Python deep learning library which runs on top of Theano or Tensorflow.
**BeautifulSoup**: Python library used for parsing HTML.
**Requests**: Python library used for getting webpages.

# 4. System Architecture

## 4.1 Architectural Design

### 4.1.1 Subsystems

**Front End Application**: web app and android app which receives question from the user and talks to the Drexel Chatbot API Service to give the answer.

**Drexel Chatbot API Service**: receives HTTP GET requests containing user queries and forwards them to Drexel Chatbot Main.

**Drexel Chatbot Main**: main process called by the API, which coordinates the other subsystems.

**Generic Question Construction**: takes the question from the user, and creates a generic question template by replacing certain nouns with generic representations.

**Generic Answer Construction**: takes in a generic question template and outputs a generic answer template.

**Generic Answer Population**: takes a generic answer template and populates it with information from the database to form an answer.

**Error Handler**: if an error occurs during execution of the system, an exception will be thrown, and the error handler will decide how to respond to the user.

**Database**: stores information about Drexel needed by generic question construction and generic answer population.

**Information Extraction**: finds information through structured or unstructured websites, and stores that information into the database. This is separate from the other subsystems, as it must be ran before the others in order to populate the database.

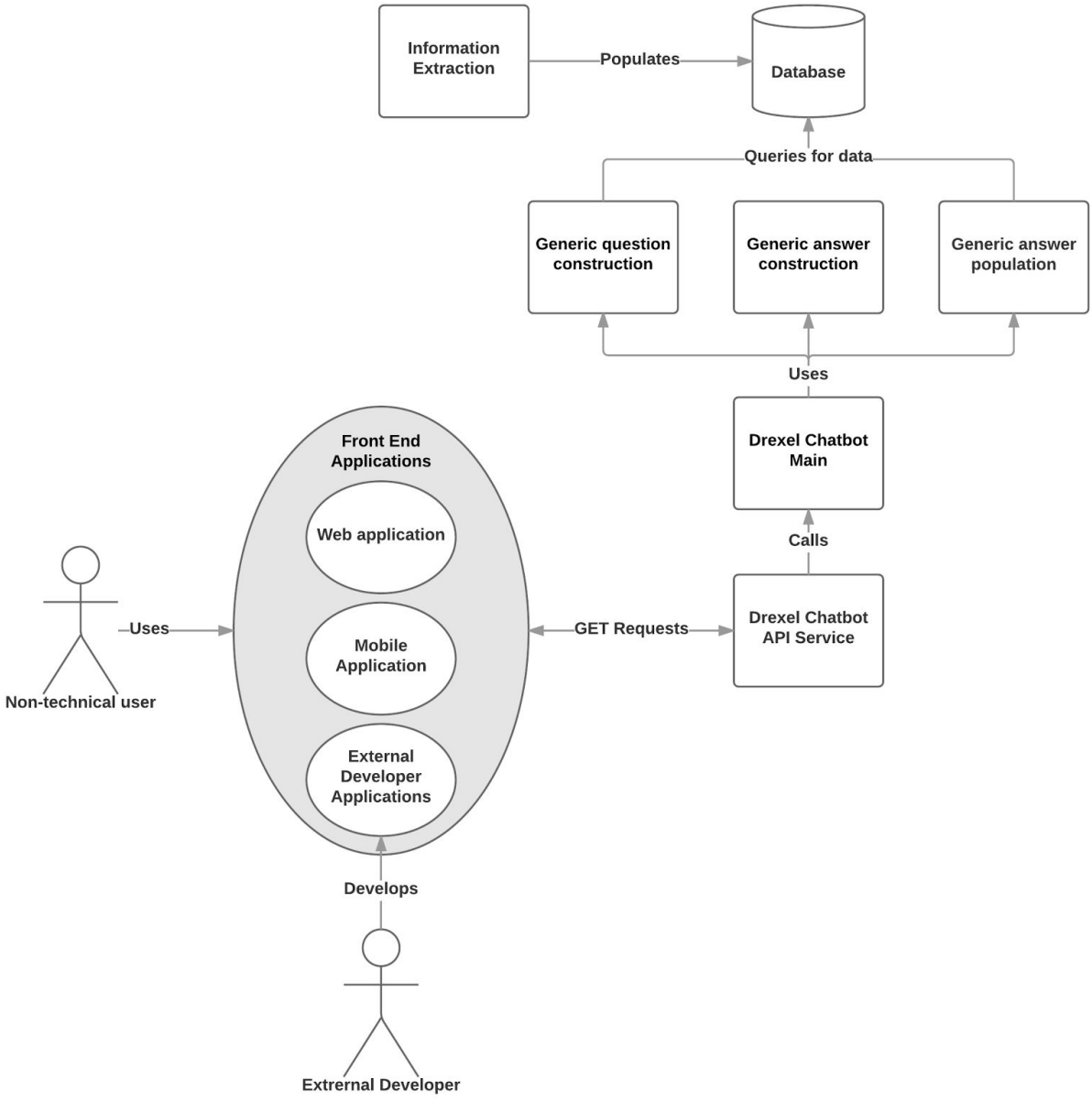



Figure 1. Use case diagram

## 4.1.2 System Operation

Figure 2 is a sequence diagram that illustrates the interactions of the system's subsystems during the typical use of a front end application.

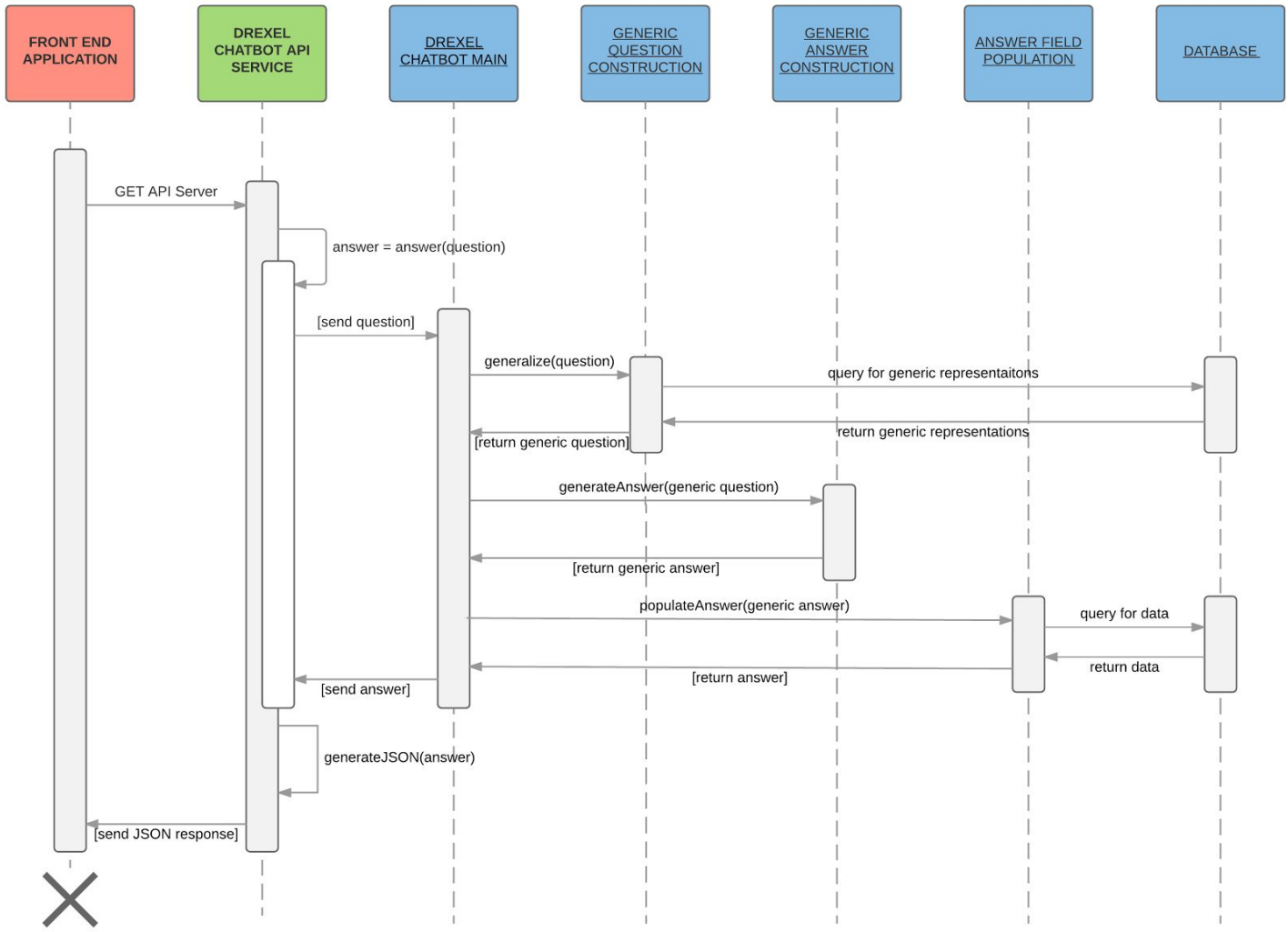



Figure 2. Drexel Chatbot Sequence Diagram

## 4.2 Decomposition Description

### 4.2.1 Drexel Chatbot API Service

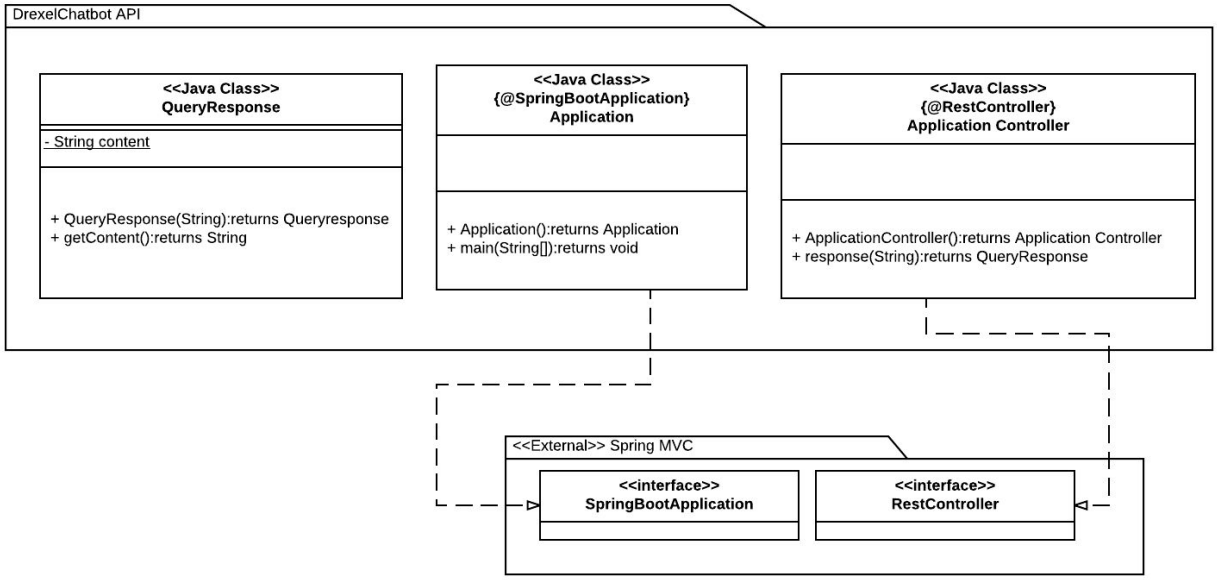



Figure 3. Drexel Chatbot API Service UML

### 4.2.2 Drexel Chatbot Main

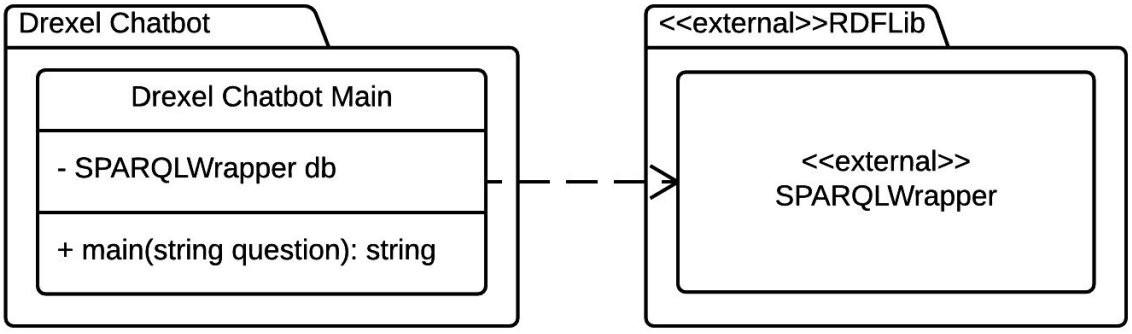



Figure 4. Drexel Chatbot Main UML

### 4.2.3 Generic Question Construction

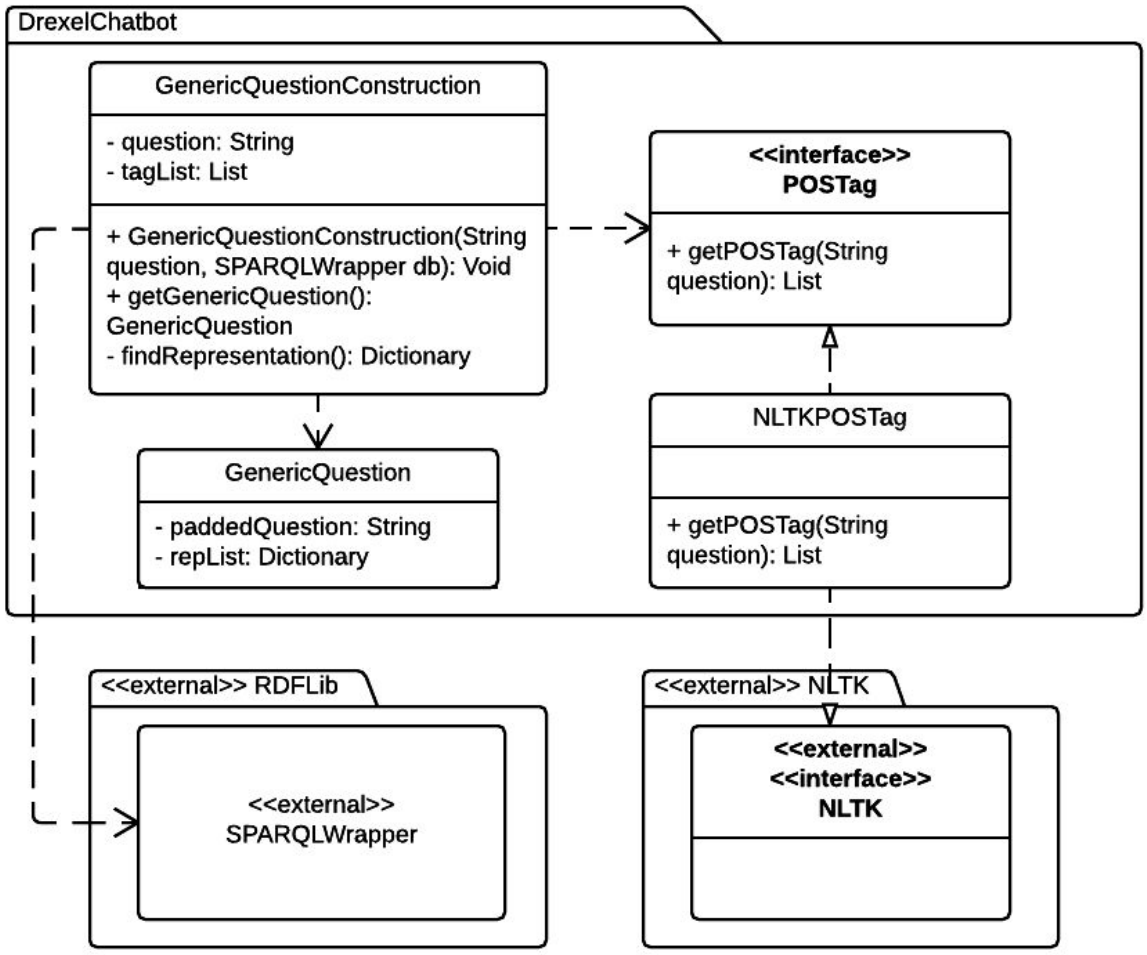



Figure 5. Generic Question Construction UML

### 4.2.4 Generic Answer Construction

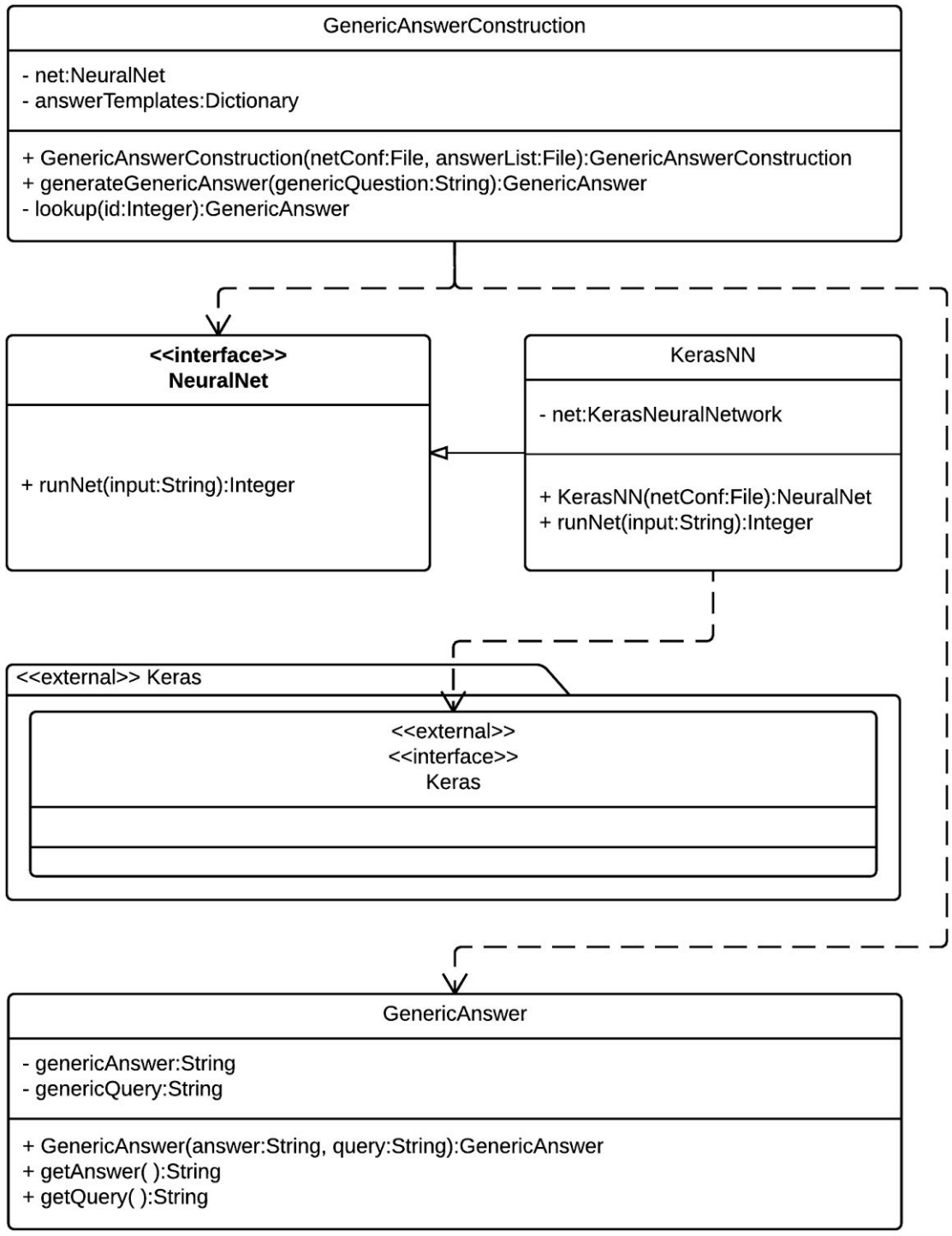



Figure 6. Generic Answer Construction UML

### 4.2.5 Generic Answer Population

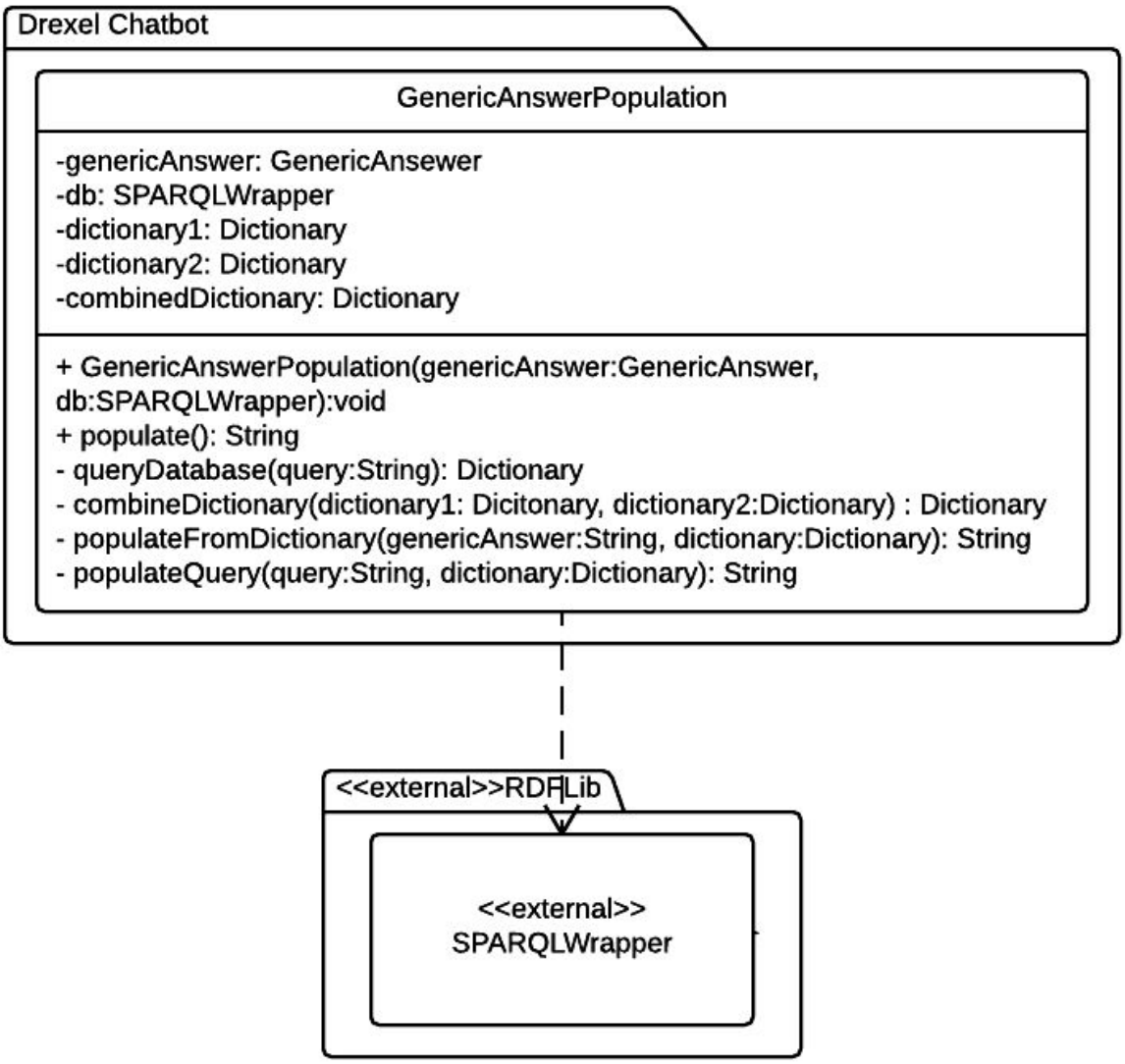
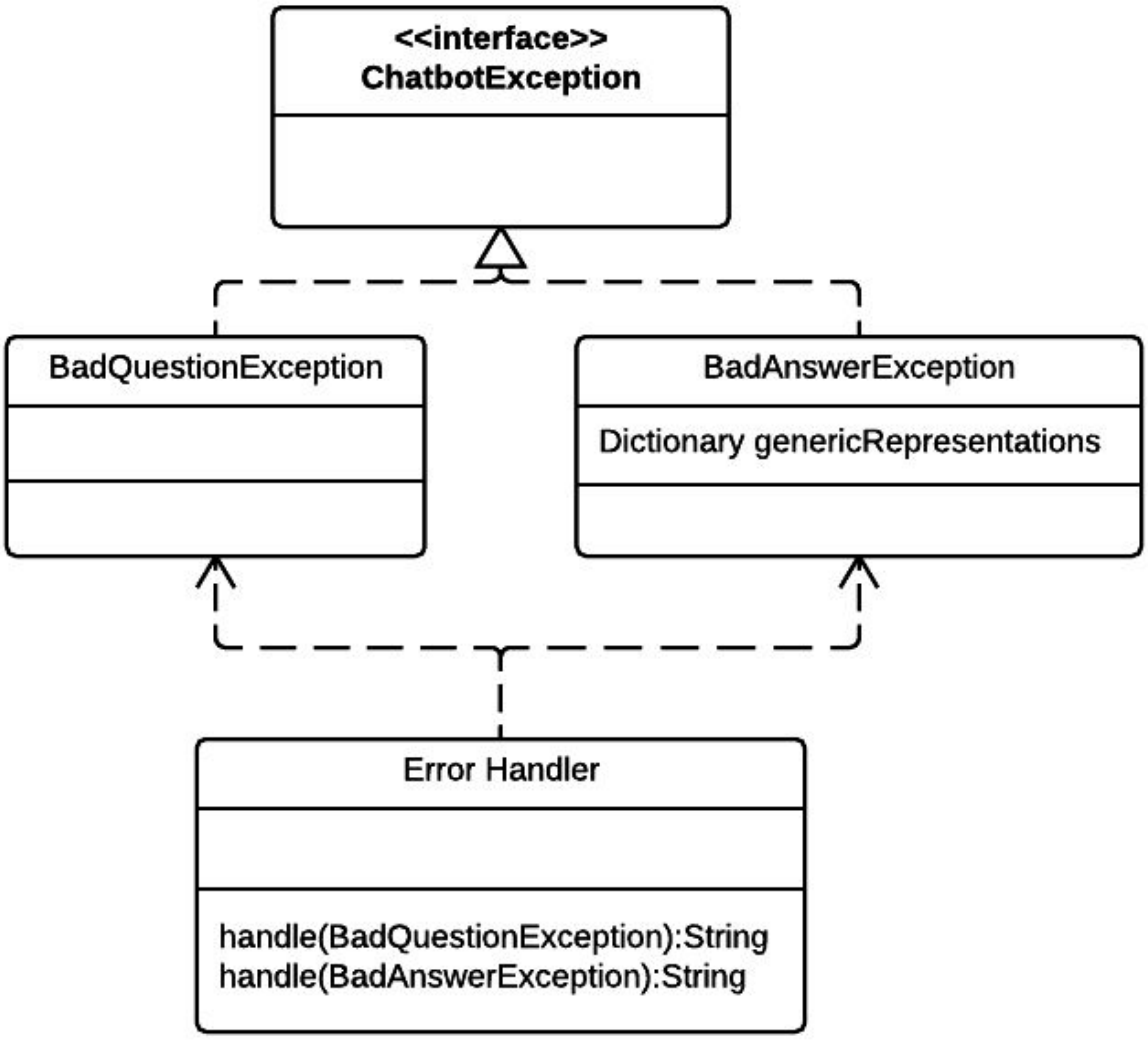



Figure 7. Generic Answer Population UML

4.2.6 Error Handler



### 4.2.7 Information Extraction

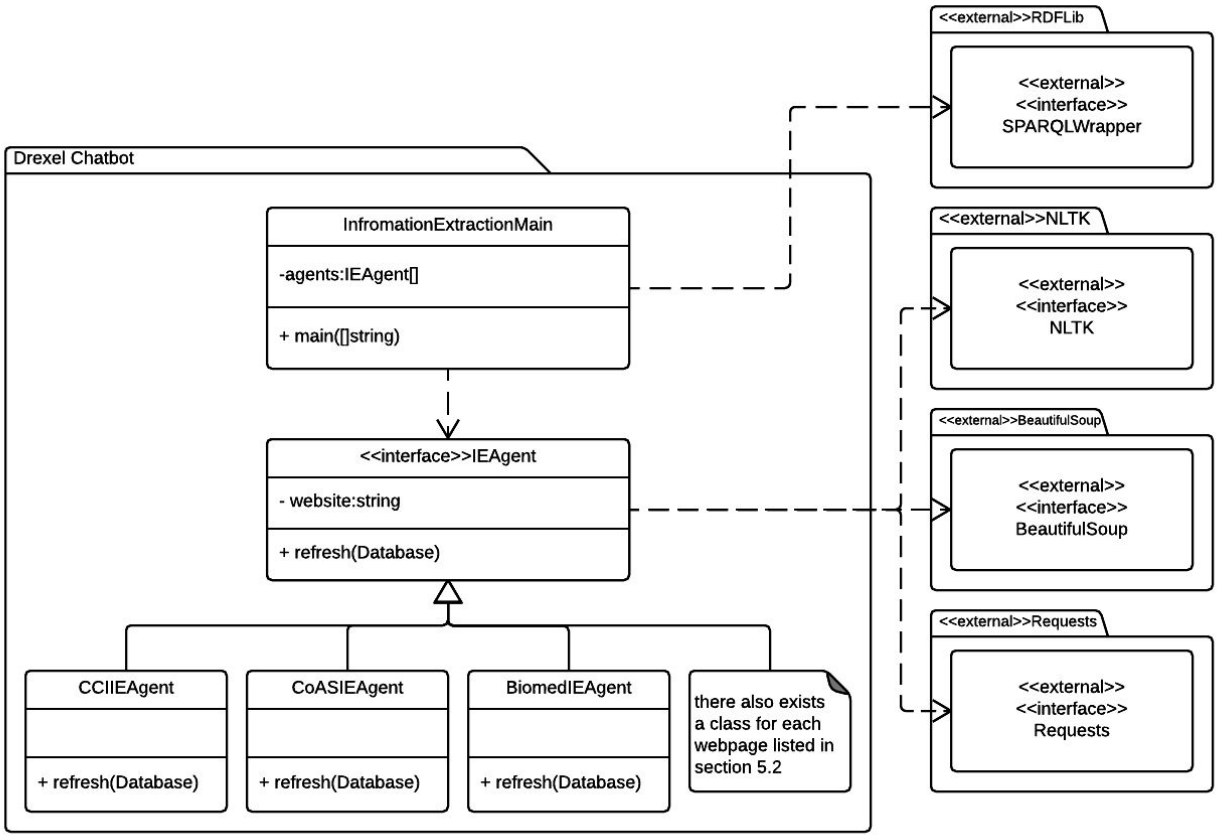



Figure 8. Information Extraction UML

## 4.3 Design Rationale

In this section, we explain our choices for the following designs and systems with our reasonings behind them.

### 4.3.1 Generic Question Answering System

We opted for a generic question answering system instead of specific questions and answers to eliminate the amount training neural network has to perform. In addition, if some answers change, we would not have to train the neural network all over again.

### 4.3.2 API

Since Drexel Chatbot focuses on researching on question answering system, we decided to create a web API that demonstrates our findings and benefits other application developers. Sample web and mobile applications will be developed to

showcase our result, but we are not focused on developing any comprehensive, robust front-end systems.

### 4.3.3 Neural Net

There are two ways to create a chatbot. First, we write all the logic that would be able take any question and give an appropriate response. This would be adding a lot of rules manually and it will take a lot of time and effort. It will also not be reusable for any other domain. The second method is machine learning. Using neural network would allow us train the neural net with a set of question and answers and if a new question is passed to the neural net, the neural net will try to provide an appropriate answer.

### 4.3.4 Python

We selected Python as our main programming language because it has the highest number of libraries in the Artificial Intelligence field, as well as each member on the development team are reasonably familiar with the language. However, the web API portion will be developed in Java since Spring MVC enables easy handling of multiple API users, while the rest of our application will stay in Python.

### 4.3.5 Modularize

Since we are not sure how well some algorithms will perform, modularizing subsystems will allow us to retire poorly-functioning algorithms and introduce new ones without breaking the entire system.

### 4.3.6 Error Handler

We added error handler to process ambiguous questions entered by the user and unexpected responses returned by the neural network. In addition, the error handler lets us provide support for additional error cases efficiently, by just creating a new exception type and handle function.

# 5. Data Design

## 5.1 Database

Drexel Chatbot utilizes a Stardog database table to store information needed by generic answer construction and generic answer population. The database essentially serves as an all-encompassing cluster that provides the data needed for all questions that Drexel Chatbot can handle. Sample columns in the table include the name, open time, close time, and location of an academic building. Each column corresponds with a generic representation.

## 5.2 Data Collection and Upkeep

The project team pre-populate and periodically update the table with information collected from www.drexel.edu and associated domains listed below:

Faculty:

- CoAS - http://drexel.edu/coas/faculty-research/faculty-directory/
- Biomed - http://drexel.edu/biomed/faculty/
- LeBow - http://www.lebow.drexel.edu/faculty-and-research/faculty-directory
- CCI - http://drexel.edu/cci/contact/Faculty/
- SoE- http://drexel.edu/soe/faculty-research/faculty/
- EngCBE - http://drexel.edu/cbe/contact/faculty/
- EngCAE - http://www.drexel.edu/cae/contact/faculty/
- EngECE - http://drexel.edu/ece/contact/faculty-directory/
- EngMgmt - http://drexel.edu/engmgmt/contact/faculty/
- EngTech - http://drexel.edu/engtech/contact/faculty/
- EngMat - http://www.drexel.edu/materials/contact/faculty/
- EngMEM - http://www.drexel.edu/mem/contact/faculty-directory/
- Close - http://www.drexel.edu/close/academics/faculty/
- Hsm - http://drexel.edu/hsm/about/faculty/
- LawCore - http://drexel.edu/law/faculty/fulltime_fac/
- LawAffil - http://drexel.edu/law/faculty/clinical_affiliated/
- LawAdjunct - http://drexel.edu/law/faculty/adjunct/
- Westphal - http://www.drexel.edu/westphal/about/directory/
- Medicine - http://www.drexel.edu/medicine/Faculty/Profiles/
- Nursing - http://www.drexel.edu/cnhp/faculty/profiles/
- Goodwin - http://drexel.edu/goodwin/about/directory/
- Dornsife - http://drexel.edu/dornsife/academics/faculty/

Facilities:

- https://www.library.drexel.edu/drexel-buildings-1891-present

Policies:

- http://drexel.edu/provost/policies/overview/
- http://drexel.edu/it/about/policies/policies/
- http://drexel.edu/undergrad/apply/freshmen-instructions/

Events:

- http://drexel.edu/events/

This list may not be exhaustive, and more sites may need to be added at a later date if need be. Additionally, information may be added to the list manually using Stardog's command line tools, if no easily parsable website exists with the information.

## 5.3 Database Organization

| Column Name | Data Type | Brief Overview | Example |
|---|---|---|---|
| Name | string | The name's of the object | Dr. Marcello |
| Property | string | The generic representation of this object | Building; faculty |
| Title | string | The title of a faculty member | Assistant Professor |
| Department | string | The department a building hosts or a faculty member belongs to | Department of Psychology |
| Address | string | The address of a faculty member's office or an academic building | 3141 Market Street |
| Room | string | The room number of a faculty member's office | Rush 231 |
| StartTime | string | Open time of a building, or start time of a faculty member's office hour | Tue 10:30 a.m. |
| EndTime | string | Close time of a building, or end time of a faculty member's office hour | Tue 12:30 p.m. |
| Education | string | Degrees received by a faculty member | Masters in Computer Science, University of Pennsylvania |
| Email | string | The email address of a faculty member or facility | nz66@drexel.edu |
| Website | string | The website of a department or faculty member | http://drexel.edu/biomed/ |
| Picture | string | The url of a picture of a building or faculty member | https://goo.gl/5ZnKHm |
| Publication | string | The 3 most recent scholarly articles a faculty published | N/A |
| Food Type | string | The main types of food a food truck serves | Indian; Chinese; Italian |

## 5.4 Structure

### 5.4.1 Generic Answer

| Name | Type | Description |
|---|---|---|
| answer | string | The generic answer that was created by the Generic Answer Construction component |

# 6. Component Design

## 6.1 Drexel Chatbot API Service

### 6.1.1 QueryResponse Class

#### 6.1.1.1 Attributes

| Name | Type | Description |
|---|---|---|
| content | String | The content of a QueryResponse Object |

#### 6.1.1.2 Methods

| Public QueryResponse(String) | | |
|---|---|---|
| | Data type | Example |
| Input | String | “Data for the query response” |
| Output | QueryResponse | QueryResponse Object |
| Description | This object is used to hold responses to be returned from the main python class | |

| Public getContent() | | |
|---|---|---|
| | Data type | Example |

| | | |
|---|---|---|
| Input | void | |
| Output | String | “Data for the query response” |
| Description | Returns the data contained in the QueryResponse object | |

### 6.1.2 Application Class

#### 6.1.2.1 Methods

| public Application() | | |
|---|---|---|
| | Data type | Example |
| Input | void | |
| Output | void | |
| Description | Instantiates the bootable Spring MVC application | |

| Public main(String[] args) | | |
|---|---|---|
| | Data type | Example |
| Input | String[] | [“array”, “of”, “arguments”] |
| Output | void | |
| Description | Runs the Spring application | |

### 6.1.3 ApplicationController Class

#### 6.1.3.1 Methods

| Public ApplicationController() | | |
|---|---|---|
| | Data type | Example |
| Input | void | |
| Output | ApplicationController | ApplicationController Object |

| | |
|---|---|
| Description | Instantiates an Application Controller to handle web requests and respond with web applications |

| Public response(String) | | |
|---|---|---|
| | Data type | Example |
| Input | String | “The query from the user to be passed to the internal application” |
| Output | QueryResponse | QueryResponse Object |
| Description | When receiving a query from the user by means of a GET request, sends the query to the internal workings of the ChatBot and waits for a response. Upon receiving a response, Returns a QueryResponse containing that response as JSON | |

# 6.2. Drexel Chatbot Main

## 6.2.1 DrexelChatbotMain Class

### 6.2.1.1 Attributes:

| Name | Type | Description |
|---|---|---|
| db | Private SPARQLWrapper | The object containing methods for interacting with the database |

### 6.2.1.2 Methods:

| public main(String question) | | |
|---|---|---|
| | Data type | Example |
| Input | String | “Where is Marcello Balduccinni’s office?” |
| Output | String | “Marcello Balduccinni’s office is located at Rush 23” |
| Description | The main method of the whole system. Calls the other subsystem’s in order, as well as handling error cases:<br>1. Try {<br>2. Call GenericQuestionConstruction.getGenericQuestion with the input question | |

| | |
|---|---|
| | 3. Call GenericAnswerConstruction.generateGenericAnswer with the output of step 2<br>4. Call GenericAnswerPopulation with outputs of step 2 and 3<br>5. Return output of step 4<br>6. } Catch ChatbotException {<br>7. Call ErrorHandler.handle(exception)<br>8. } |

## 6.3 Generic Question Construction

### 6.3.1 GenericQuestionConstruction Class

#### 6.3.1.1 Attributes:

| Name | Type | Description |
|---|---|---|
| question | Private String | The generic question that was created by the Generic Question Construction component |
| tagList | Private List | A list that contains tuples of words and their POS tags. |

#### 6.3.1.2 Methods:

| public GenericQuestionConstruction(String question, SPARQLWrapper db) | | |
|---|---|---|
| | Data type | Example |
| Input | String, SPARQLWrapper | “Where is Marcello Balduccinni’s office?” |
| Output | Void | |
| Description | Constructor of GenericQuestionConstruction class | |

| public GenericQuestion getGenericQuestion() | | |
|---|---|---|
| | Data type | Example |
| Input | Void | |
| Output | GenericQuestion | {<br>question = “Where is $(professor)’s office?”, |

| | | |
|---|---|---|
| | | rep_dict = {$(professor) :'Marcello Balduccinni'} } |
| Description | Processing method of GenericQuestionConstruction class:<br>1. Take string and get Part-of-Speech tags, by calling POSTag.getPOSTag<br>2. Make a list of all nouns in the sentence<br>3. Find possible generic representations for all nouns, by calling findRepresentation() method<br>4. Create a generic question string by replacing nouns from the original string<br>5. Create and return a GenericQuestion object containing the string with generic representations, and the dictionary returned from findRepresentation<br>6. If there was an error, then thrown BadQuestionException | |

| private Dictionary findRepresentation() | | |
|---|---|---|
| | Data type | Example |
| Input | Void | |
| Output | Dictionary | {$(professor) :'Marcello Balduccinni'} |
| Description | Create a dictionary. For each noun in the list, query the database to see if it exists in the "name" column. If so, then it is a noun we have information on, and its generic representation is in the "property" column. Add an entry to the dictionary, key=property column and value=name column. Return this dictionary at the end. | |

## 6.3.2 POSTag Interface

### 6.3.2.1 Methods

| public List getPOSTag(String question) | | |
|---|---|---|
| | Data type | Example |
| Input | String | "Where is Marcello Balduccinni's office?" |
| Output | List | [('Where', 'WRB'), ('is', 'VBZ'), ('Marcello', 'NNP'), ('Balduccinni', 'NNP'), (''s', 'POS'), ('office', 'NN'), ('?', '.')] |

| | |
|---|---|
| Description | Interface of methods for applying POS tagging techniques to a given String and returning a list of tuples, each tuple contains a word and its tag. |

### 6.3.3 GenericQuestion Class

#### 6.3.3.1 Attributes

| Name | Type | Description |
|---|---|---|
| question | Public string | The generic question that was created by the Generic Question Construction component |
| genericRepresentation | Public dictionary(string, string) | A dictionary that links the generic representations to the word they replaced. |

## 6.4 Generic Answer Construction

### 6.4.1 GenericAnswerConstruction Class

#### 6.4.1.1 Attributes

| Name | Type | Description |
|---|---|---|
| net | Private NeuralNet | The neural network interface. |
| GenericAnswers | Private Dictionary | Dictionary of GenericAnswers |

#### 6.4.1.2 Methods:

| Public GenericAnswerConstruction(File netConf, File answerDict) | | |
|---|---|---|
| | Data type | Example |
| Input | File, File | “answerDict.txt”:<br><file start><br>1: $(professor)’s office is in $(building),<br>2: $(professor) teaches $(class)<br>⋮<br><file end> |
| Output | GenericAnswerConstruction | |

| | |
|---|---|
| Description | Constructor of GenericAnswerConstruction class. Reads the answerDict file and adds them to the GenericAnswer dictionary. The dictionary is from id number to GenericAnswer objects. Also initialize the neural net using the netConf file. |

| Public GenericAnswer generateGenericAnswer(String GenericQuestion) | | |
|---|---|---|
| | Data type | Example |
| Input | String | “Where is $(professor)’s office?” |
| Output | GenericAnswer | {<br>answer = “$(professor)’s office is in $(location).”, query = “Get location where $(professor) => office => location”<br>} |
| Description | This function transforms a generic question to a generic answer:<br>1. Prepares the string to run with the NeuralNet interface<br>2. Calls NeuralNet.runNet(PreparedInput), which returns the id number of the generic answer<br>3. Call lookup(id number) for the GenericAnswer.<br>4. If there was an error, then thrown BadAnswerException | |

| Private GenericAnswer lookup(int id) | | |
|---|---|---|
| | Data type | Example |
| Input | Int | 4 |
| Output | GenericAnswer | {<br>answer = “$(professor)’s office is in $(location).”,<br>query = “Get location where $(professor) => office => location”<br>} |
| Description | This function looks up the generic answer that has the id given | |

## 6.4.2 NeuralNet Interface

### 6.4.2.1 Methods

| Private Int runNet(NetInput input) | | |
|---|---|---|
| | Data type | Example |

| | | |
|---|---|---|
| Input | String | “Where is $(professor)’s office?” |
| Output | Int | 4 |
| Description | This function runs the neural net on the generic question given | |

### 6.4.3 KerasNN Class

#### 6.4.3.1 Attributes

| Name | Type | Description |
|---|---|---|
| net | Private KerasNeuralNetwork | An instance of the Keras neural network |

### 6.4.4 Training

There will be a short script that will train and test the neural net, print out some useful statistics, and save the neural network configuration and weights to a file. This process could take up to a few days, but is only done once, not with every API call.

## 6.5 Generic Answer Population

### 6.5.1 GenericAnswerPopulation Class

#### 6.5.1.1 Attributes

| Name | Type | Description |
|---|---|---|
| genericAnswer | private GenericAnswer | The GenericAnswer object returned by GenericAnswerConstrucion class. It contains answer String and query String. |
| dictionary1 | private Dictionary | The dictionary received with Generic Representation as key and specific name as value. Eg<br>{<br>$(Professor) -> Marcello<br>} |
| dictionary2 | private Dictionary | The dictionary received after making the query to the database. Eg<br>{<br>$(Building) -> UCross |

| | | |
|---|---|---|
| | | } |
| combinedDictionary | private Dictionary | The combined dictionary from dictionary1 and dictionary2. Eg<br>{<br>$(Professor) -> Marcello<br>$(Building) -> UCross<br>} |
| db | private SPARQLWrapper | The SPARQL wrapper through which database calls can be made |

6.5.1.2 Methods

| public GenericAnswerConstruction(GenericAnswer genericAnswer, SPARQLWrapper db) | | |
|---|---|---|
| | Data type | Example |
| Input | GenericAnswer, SPARQLWrapper | {<br>answer = “$(Professer)’s office is in $(location)”<br>query = “select location where Professor = $Professor”,<br>dictionary = (Professor -> Marcello Balduccinni),<br>},<br>db |
| Output | Void | |
| Description | Constructor of GenericAnswerConstruction Class. | |

| public String populate() | | |
|---|---|---|
| | Data type | Example |
| Input | void | |
| Output | String | Marcello Balduccinni’s Office is in Rush22 |
| Description | Runs the whole process and returns the final answer:<br>1. populate database query using function populateQuery()<br>2. query the database and get new dictionary using function queryDatabase()<br>3. combined the two dictionaries using function combineDictionary() | |

| | |
|---|---|
| | 4. replace the items from the combined dictionary to the generic answer sentence using function populateFromDictionary()<br>5. If there was an error, then thrown BadAnswerException |

| private Dictionary databaseQuery(String query) | | |
|---|---|---|
| | Data type | Example |
| Input | String | Select location where Professor is “Marcello Balduccini”. |
| Output | Dictionary | (Location -> Rush 221) |
| Description | This function queries the database and gives a dictionary with the result query | |

| private Dictionary combineDictionary(Dictionary dictionary1, Dictionary dictionary2) | | |
|---|---|---|
| | Data type | Example |
| Input | Dictionary, Dictionary | (Professor -> Marcello Balduccinni), (Location -> Rush 221) |
| Output | Dictionary | (Professor -> Marcello Balduccinni, Location -> Rush 221) |
| Description | Combines two dictionary into a single dictionary. Any conflicts will result in an error. | |

| private String populateFromDictionary(String genericAnswer, Dictionary dict) | | |
|---|---|---|
| | Data type | Example |
| Input | String, Dictionary | “$(Professer)’s office is in $(location)”, (Professor -> Marcello Balduccinni, Location -> Rush 221) |
| Output | Dictionary | Marcello Balduccinni’s Office is in Rush221 |
| Description | This function populates the missing words in the String using a dictionary | |

| private String  populateQuery(String genericQuery) | | |
|---|---|---|
| | Data type | Example |
| Input | String, Dictionary | “select location where Professor = $Professor”, (Professor -> Marcello Balduccinni) |
| Output | String | “select officeName where Professor = Marcello Balduccinni” |
| Description | This function populates missing words from the query String using a dictionary | |

# 6.6 Error Handler

## 6.6.1 ErrorHandler Class

### 6.6.1.1 Methods

| Public void handle(BadQuestionException ex) | | |
|---|---|---|
| | Data type | Example |
| Input | BadQuestion Exception | |
| Output | String | “Sorry, I didn’t understand the question.” |
| Description | There was a problem with the generated generic question. In this case, we wish to return the sentence “Sorry, I didn’t understand the question.” | |

| Public void handle(BadAnswerException ex) | | |
|---|---|---|
| | Data type | Example |
| Input | BadAnswer Exception | |
| Output | String | “Sorry, I had trouble understanding, but here is the website for PISB: http://...” |
| Description | There was a problem with the generated generic answer. In this case, we wish to return the website for the nouns we found. | |

| | |
|---|---|
| | 1. For each value in the dictionary<br>2. Query the database for the websites related to those nouns<br>3. Generate a sentence such as “Sorry, I had trouble understanding, but here is the website for …” and fill insert website information.<br>4. If no website is found in the database, then return “Sorry, I didn’t understand the question.”<br>5. Return the sentence |

## 6.7 Information Extraction

### 6.7.1 InformationExtractionMain Class

#### 6.7.1.1 Attributes:

| Name | Type | Description |
|---|---|---|
| agents | Private IEAgent[] | A list that contain all classes that implement the IEAgent interface. |

#### 6.7.1.2 Methods:

| Public void main() | | |
|---|---|---|
| | Data type | Example |
| Input | void | |
| Output | void | |
| Description | Perform information extraction by initializing the connection to the database, and then calling the “refresh” method on all of the IEAgents located in the “agents” list. | |

### 6.7.2 IEAgent<<interface>>

#### 6.7.2.1 Attributes

| Name | Type | Description |
|---|---|---|
| website | string | The URL to the webpage containing the information that this Agent should extract. |

6.7.2.2 Methods

| Public void refresh(Database database) | | |
|---|---|---|
| | Data type | Example |
| Input | Database | |
| Output | void | |
| Description | An interface method that all IEAgent's will implement.This will get the webpage located at the website, and then parse it for its information. Drexel appears to have no reliable, standard format for their webpages, so each webpage will have it's own parsing algorithm. The NLTK library is also available to these classes, which can be used to extract information from unstructured data, such as paragraphs. | |

# 7. Human Interface Design

## 7.1 Overview of User Interface

### 7.1.1 Methods

| private setMessageTime() | | |
|---|---|---|
| | Data type | Example |
| Input | void | |
| Output | void | |
| Description | Sets the last sent message's timestamp to the time it was sent. | |

| private buttonOnClick() | | |
|---|---|---|
| | Data type | Example |
| Input | void | |
| Output | void | |
| Description | Calls sendMessage(message) and then sets the text area's value to empty. | |

| private void sendMessage(String message) | | |
|---|---|---|
| | Data type | Example |
| Input | String | "Where is Marcello Balduccinni's office?" |
| Output | void | |
| Description | If the text area's value is not null, sends the value to the server and waits for a response. When a response is received, the message is pushed to the user's screen. | |

## 7.2 Screen Images

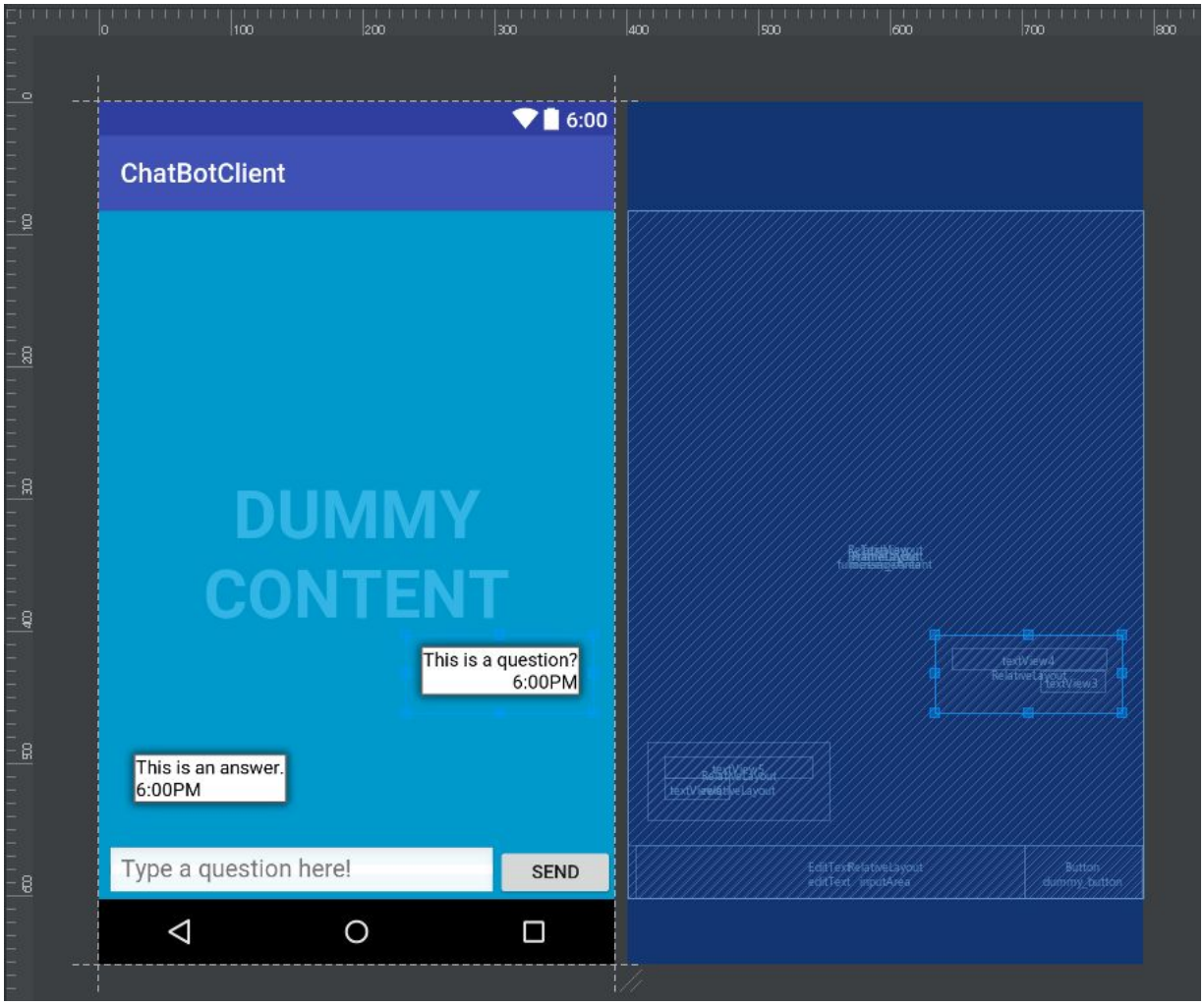



Figure 9. An example of a mobile application

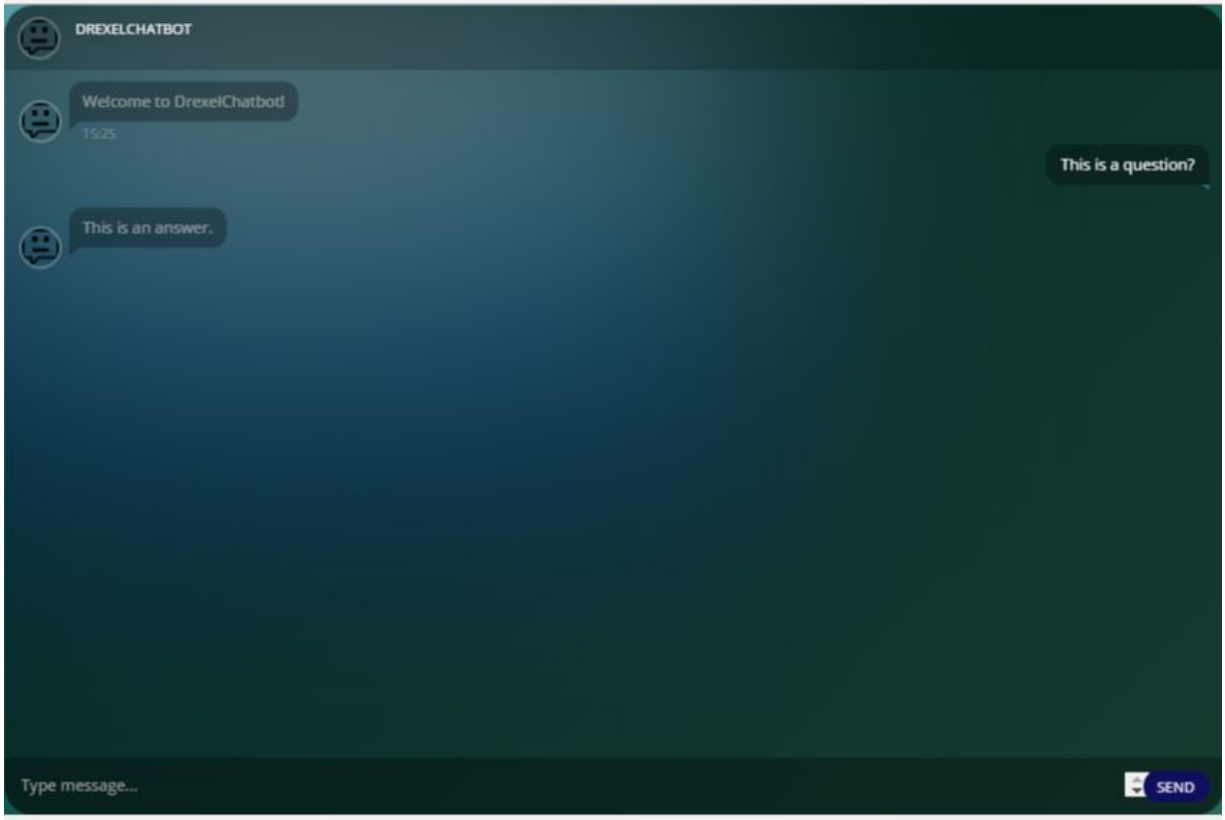



Figure 10. An example of a web application

## 7.3 Screen Objects and Actions

Client applications for the Chatbot web service will have four distinct objects separated into two sections. These sections will separate user input from the ongoing Q&A session being held by the user and chatbot.

### 7.3.1 User Input Section

The user input section will contain two objects: A textbox to enter a question; A button to send the query to the web service

### 7.3.2 Message Container

The message container section will contain two types of objects in the form of a list: User queries; Service answers

# 8. Requirements Matrix

| Requirement | Description | Design Reference |
|---|---|---|
| 4.1.*, 5.1.1.1 | API call | 6.1.* |
| 4.2.*, 5.1.1.1 | Generic Question Construction | 6.3.*, 6.6.* |
| 4.3.*, 5.1.1.1 | Generic Answer Construction | 6.4.*, 6.6.* |
| 4.4.*, 5.1.1.1 | Generic Answer Population | 6.5.*, 6.6.* |
| 4.5.2.*, 4.5.3.*, 4.6.*, 5.1.1.1 | Information Extraction | 6.7.* |
| 4.5.1.*, 5.1.1.1 | Database | 5.* |
| 4.7.* | User Interface | 7.* |