# Final Report

*Implementation of an ontology-driven conversational agent*

**Author:** Jay Rainey
**Supervisor:** Irena Spasić
**Moderator:** Yukun Lai
**Module number:** CM3203
**Module title:** One semester individual project
**Credits:** 40

# Table of Contents

# Abstract

Health behaviors that negatively affect an individual, such as lack of physical activity or an unhealthy diet are responsible for the most deaths in the United Kingdom per year [1]. It is possible to elicit behavioral change through reflection by openly discussing user habits [2, 3], though limited research has been undertaken that focuses on diet and exercise habits.

This successful project designed and developed a conversational system that uses structured conversation to administer interventions techniques to invoke user reflection. It is proposed that users will evaluate their health decisions based on conversations with the system.

User participation through these short conversations and their feedback can gradually improve the system's performance over time in two ways:

1. **Context detection:** each user message is used to improve the knowledge base, which is utilized during message-selection on the server.
2. **Conversation structure:** each message sent by the service is manually written to abide by the intervention techniques, and is evaluated by the user through a voting system.

This project concludes that administrating independent interventions through automated free-form conversations is feasible, though further research should be undertaken to assess the effectiveness of this.

## Acknowledgements

Thanks to Irena Spasić for project supervision and guidance, and users for their participation.

# Introduction

This section provides an outline and overview of the objectives of this project including a general outline that excludes technical details, project goals, and the limitations of the solution.

## Project outline

This projects aims to develop an autonomous chat service to act as a health counselor to users. The conversation is structured using intervention techniques in an attempt to elicit behavioral change and reflection in users by openly discussing their health choices and habits. It is proposed that conversations will motivate user actions and invoke independent change [4].

## Project objectives

The following is a broad overview of the goals in each stage of the project. These differ from the initial report as I have only included a general overview of the functional objectives below. The system requirements are expanded in detail in the design requirements section below.

### Modeling domain knowledge

- Design, develop, construct, and implement a domain-specific ontology.
- Develop utilities to facilitate automation and simplification of this process.

### System design

- Manually write system responses using specific intervention approaches.
- Develop algorithms that use the ontology to select context-appropriate responses.
- Develop algorithms to improve the system through user feedback:
    - Improve knowledge base and context detection using user messages.
    - Introduce a voting mechanism to improve the selection of written responses.

### Graphical user interface design

- Design an efficient chat interface where users can enter free-form messages, and provide feedback to all aspects of the system including the usefulness of system messages, usability of the designed system, and general opinions of the system.

### Testing

- Quantify and evaluate the effectiveness of the system including:
    - Functionality testing
    - Usability design
    - Ontological development over time using user messages
    - Manually written intervention questions

## Limitations and risks

This section describes the limitations and risks that were expected to be faced during the project, and identifies techniques incorporated to reduce or overcome each.

### User participation

The greatest risk in this project was the dependency on user participation to improve several areas of the system. User messages improve the systems ontology; long, accurate, and detailed responses are ideal. The contextual response selection algorithm depends on user feedback (voting) to identify the most efficient and useful messages sent by the system.

Consequently, it was imperative that mass user participation was achieved as a means of evaluating these aspects of the system. To achieve this, the system test website should be advertised on social media, and across various other outlets.

### Counseling techniques

Counseling techniques were used to structure the systems conversations. Consequently, the responses were written to abide by these techniques, which required: invoking personal reflection in users by being open-ended and empathetic. However, as I am not a trained counselor I cannot be certain that the questions I have written will achieve this goal.

To identify the questions that are inefficient I introduced a voting mechanism as an integral part of the user interface, which allows them to rate the automated responses sent by the system. The questions can be formally evaluated to identify the qustions that are well written, and improve those that are not.

### System effectiveness

Each conversation required the user to be anonymous due to the intervention technique used, which did not enforce strict controlled conditions. This and time constraints restricted the potential scope of the project as an effective, systematic evaluation on ontological development and the intervention technique used was not possible.

Instead, the system was developed to achieve the defined system objectives, rather than a controlled and research oriented project. This provides a framework for future developers or researches to begin carrying out experiments to test and evaluate the hypothesis of the research questions proposed by this project.

# Background

The section introduces the core techniques used to achieve the system objectives, a discussion of current research using these techniques, and how the system developed attempts to address a hole in current research in a novel way.

## Conversational structure

Brief motivational interventions (BMI) are a strategy used to explore a patient's motivation to change their behavior rather than prescribing a specific course of action [4]. BMI's conversation structure consists of four strategies that the counselor (this system) must abide to be effective:

1. **Open-ended questions:** invoke user reflection and reasoning for patients' choices.
2. **Affirmations:** highlight patients' strengths, values, and goals using compliments.
3. **Reflections:** restatements of patients' thoughts to demonstrate understanding.
4. **Summaries:** combines patients' statements to ensure that the counselor has the correct understanding of users ideas, and encourages further exploration of these.

The applications conversation structure follows the BMI approach by using open-ended questions to invoke reflection, and combines the other three strategies in the response to the user. This process facilitates user reflection, while demonstrating understanding of the user's particular problem by the system, and is similar to how counselors use the BMI framework [4].

## Current research

To date, there is limited research using ontologies to power automated interventions, though ontologies have been used extensively in various domains to store, access, and generate suitable responses, which is the desired goal of the ontology for this project.

Existing research uses alternative intervention techniques and adds restrictions to user input by providing pre-defined responses for the user to select from [5]. This application attempts to address these issues by providing a free-form user input where messages can be written, and analysis of these used in the application to select an appropriate response to send to the user.

Research establishes that BMIs are effective and have positive change in participants after a single conversation [4], that interventions can be more effective than control groups [6], are effective in patients of all ages and varying stages of change [7]. These unique properties make BMI best suited over alternatives for an automated system.

## Research questions

1. Can BMIs be administered effectively using an automated service?
2. Are free-form messages suitable to select appropriate intervention responses?
3. To what extent is an ontological representation suitable for selecting responses?
4. Can user participation drive the development of such a system?

# Specification and design

This section covers the requirements for what the system being developed should achieve, and how I planned to do this effectively. This includes the four stages as outlined in the introduction.

## System architecture, tools and methods

This project focuses on abiding by software engineering best practices to develop a solution that was reusable, maintainable, and well documented. This section discusses and justifies the selection of the specific architectures, tools, and methods used to achieve the systems goals.

### Flask micro-framework

Python was chosen as the main programming language due to the extensive range of libraries available suitable for this project, and my personal and professional experience using it[1].

The Python Flask micro-framework was chosen to power the web application, as it is simple, lightweight, and extensible. Its "micro" aspect aims to *"keep the core simple but extensible"* [8], and is structured to promote modularity and simplicity by using design-patterns that have been incorporated throughout the project as discussed in the system design section.

The *"micro"* aspect is why Flask was selected over alternatives, such as Django and Pyramid, which provide extensive functionality outside the scope of this project, such as admin control, which can be added to Flask through extensions if the application requires this in the future.

### Flask-SQLAlchemy

Object Relational Mapper (ORM) provides an abstraction for database access by mapping Python classes to database tables and access of data through these. ORMs were chosen over database specific extensions as they provide database vendor independence, and abstract database access to class implementation. This removes the need to write hard-coded queries, which simplifies code readability, implementation, and testing.

### Version control

Version control is a tool that saves snapshots (commits) of a working directory, which simplifies viewing, modification, or removal of previous commits. Using version control provides backup functionality as a copy of the codebase existed on external remote servers, which ensured project progress would continue if any major technical issues occurred.

Git was chosen over alternatives, such as SVN or Mercurial, due to its decentralized nature and branching features. Branching allows a copy of the local repository to be made into isolated branch where features can be implemented without affecting the core implementation. Features can then be developed and tested before pushing the changes to end-users.

[1] All code written abides by PEP8 coding standards: https://www.python.org/dev/peps/pep-0008/

### Deployment

To deploy and host the application I selected Heroku, a cloud platform as a service (PaaS), which provides an environment where code could be deployed, configured and run with ease. Within this project, deployment was automated; when Heroku detected changes to the remote Github repository it merged those into its own repository, and updated the live website respectively. Heroku provides simplicity of deployment over alternatives, such as Amazon Web Service or Digital Ocean, which offer infrastructure configuration outside project's scope.

### Python virtual environment

A python virtual environment allows the creation and management of dependencies in isolation. This project depends on several external libraries; using a virtual environment streamlines development, testing, and deployment for future developers by installing the project dependencies into a folder within the project on the working machine.

### Flask-SocketIO

Flask-SocketIO was chosen to achieve the client-server conversational aspect of the, which provides bidirectional communications through web-sockets. It uses a Python library to access messages on the server, and a JavaScript library to access client messages. This library was preferred over alternative networking libraries, such as Twisted or Tornado as they do not come pre-packaged with the conversational functionality that this project required.

### Flask-Script

Flask-Script is an extension that provides support for writing simple command-line interface scripts that access Python libraries and tasks outside the scope of the Flask application. This application uses Flask-Script to run utility methods, such as database and ontology creation and population, and to run the SocketIO server.

Although it is possible to write command-line applications in Python using alternative libraries, such as argparse, these would not have access to the Flask application context. Although there are workarounds to this (such as invoking desired methods on application creation), these would introduce tight coupling and restrictions into the application, which are not desirable.

### Bootstrap and jQuery

These libraries are well designed and tested to promote consistency across browsers and devices via responsive design, and are well documented. Using these instead of writing plain CSS design and JavaScript ensures that the web application front-end functionality remain consistent, maintainable, and that a usable system running could be developed quickly, which afforded more time spent on other requirements of the system.

## System requirements

The following are the essential requirements of the application that outline the key functionality and interactivity that must be achieved in order for the application to be a success. A description of each requirement and acceptance criteria to verify their achievement is provided.

### Functional requirements

**Requirement:** develop web-scraping algorithm to obtain domain knowledge via online sources.
**Acceptance:** data acquired must be relevant to ontological terms, and consist of at least **10,000** unique words to be used to assign to the ontology.

**Requirement:** develop an algorithm to clean data and insert concepts into the ontology.
**Acceptance:** only unknown words (terms of concepts) are produced from cleaned data.

**Requirement:** develop an algorithm to insert concepts from cleaned data to the ontology.
**Acceptance:** a command line interface that outputs *single terms* (from cleaned data) that can be assigned to known concepts in the ontology, which will update the ontology respectively.

**Requirement:** design and develop algorithms to select context appropriate responses.
**Acceptance:** manually written responses are selected based on the context of user messages. Determined by the frequency of words in user messages in comparison with the ontology.

**Requirement:** design and develop an algorithm to transfer ontology from file to database.
**Acceptance:** transfers ontology concepts from file to a hierarchical structure in an RDBMS.

**Requirement:** design a database to store ontology, system and user messages, and feedback.
**Acceptance:** each table stores relevant data and is queryable for system use and evaluation.

**Requirement:** design chat interface where users can enter free-form messages.
**Acceptance:** user can chat with system without restrictions in a free-form way.

**Requirement:** design and develop mechanisms to improve the system through user feedback.
**Acceptance:** the following criteria should be met for each

- **System messages:** users vote to improve future selection of system messages.
- **User messages:** improves knowledge base and context detection.
- **Usability:** relevant usability form incorporated into design to evaluate system.
- **General:** free-form field where users can share their opinion on all aspects of system.

**Requirement:** quantify and evaluate effectiveness of the system.
**Acceptance:** highlight improvements through evaluations including: ontological development, response-selection algorithms, usability design, and user participation.

**Non-functional requirements**

**Requirement:** manually write responses to ontology concepts using intervention technique.
**Acceptance:** 5 unique responses for the initial question, and open-ended question must be written for each parent term in the ontology.

**Requirement:** crowd source user participation via social networking websites.
**Acceptance:** a minimum of 100 unique users must use the application.

**Requirement:** usability – system design
**Acceptance:** at least 12 users providing feedback and achieving system usability score (SUS) above 68 percent. This would demonstrate that the usability is above average [9].

**Requirement:** cross-browser support
**Acceptance:** system must look and function the same across all major browsers.

**Requirement:** reliability – chat performance
**Acceptance:** appropriate context specific system messages must be chosen and sent within *10 seconds* of receiving a user message.

## Modeling domain knowledge

Formally modeling domain knowledge is critical to the success of the application, as it is the driving force of the system's conversational algorithms. Therefore, having a systematic and reusable approach to developing and updating the ontology as a means of formal model of domain knowledge is of utmost importance. The steps taken to achieve this, including the dependencies of each are discussed in the following section.

### Ontology structure

An ontology is an abstract understanding and simplified view of the world that we wish to represent [10]. Ontologies consist of concepts (objects of reality) and the relationships between these terms, e.g. a fruit and a grape would be terms in an ontology, linked by *"is a"* relationship.

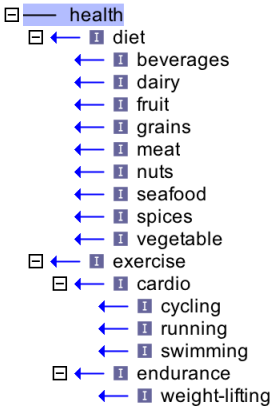
Various ontological representations exist, e.g. RDF [11], OBO [12], and OWL [13]. OBO was selected in this project as the complex features of alternatives are not required (quantified relations, cardinality, and inferences), and OBO has a simple syntax, which is illustrated in the following table where a term (diet) shares a single relationship to another term (health).

| **OBO term** |
| --- |
| [Term] |
| id: ID:2 |
| name: diet |
| is_a: ID:1 ! health |

The ontology is used within the system to detect the content of user messages by performing word frequency analysis of them and comparing the results with terms in the ontology. Appropriate term selection for the ontology is thus fundamental to the applications success.

Diet and exercise were selected as the parent terms of the ontology as they are the focus of the conversational system. To ensure suitable terms were selected, current literature and health focused ontologies in the biomedical repository [14] were examined.

Terms for this applications ontology were selected based on the commonality between ontologies in current research that provided the best scope for conversations. The resulting structure of the ontology is presented in the diagram to the right.




Diet was further sub-divided into sub-terms using criteria based on food properties and nutritional information. Other literature subcategorized ontologies by food type, such as fruit, and vegetables [15, 16]. Having compared several ontologies that use food branches, I selected those that occurred most often, and were within the boundaries of the expected conversations with the system. This analysis resulted in *nine* core diet sub-terms, which would act as parents and contain specific sub-terms related to their parent.

The major difference between the exercise ontologies examined depends on their use, which explains the sub-divisions of the categories in current literature [17, 18, 19]. This analysis resulted in the selection of two core branches of exercise: cardio and endurance. Four sub-terms were selected that were common amongst all research and ontologies analyzed, which were chosen as they provide suitable detail for the conversational scope of this project.

**Data acquisition**

Contextual data was required to populate the parent-terms of the ontology described above with suitable children terms, e.g. the "diet" term has a child term of "fruit", which has a child term of "apple". An algorithm for scrapping relevant data for each parent-term (diet and exercise) must be developed. This section discusses and justified the data sources chosen.

Reddit is an online forum where users can share their opinions, and thoughts regarding specific topics of personal interest. It is the 24th most popular website on the Internet [20], and has 168 million unique visitors from over 208 countries per month [21]. Each topic on the website is sub-divided into sections where users can discuss that topic, which is known as a "subreddit".

Reddit was chosen for data acquisition due to the diversity of users and free-form nature of the conversations held there, which resemble the desired conversational aspect of this application. Due to Reddit's popularity, the algorithm developed could be re-run to acquire additional data of conversations that had taken place since it had been previously run, which would further promote diversity of the known terms in the ontology.

To select context appropriate subreddits to be used a manual popularity analysis was performed using redditlist [22], which is a website that publishes statistics for subreddits, including their rank, growth, and number of subscribers. A limit of five subreddits of the 250 most popular was selected, as coverage of three million subscribers for each parent term will provide sufficient data to achieve the system requirements. The results of this analysis, and selected subreddits for each parent-term are provided in the tables observed on 09/02/15:

| **Diet** | | |
|---|---|---|
| **Rank** | **Subreddit** | **Subscribers** |
| 48 | Food | 2,436,847 |
| 94 | Cooking | 284,692 |
| 142 | EatCheapAndHealthy | 199,076 |
| 221 | Keto | 133,376 |
| 224 | Slowcooking | 132,973 |
| | **Total:** | *3,186,964* |

| **Exercise** | | |
|---|---|---|
| **Rank** | **Subreddit** | **Subscribers** |
| 29 | Fitness | 2,556,378 |
| 105 | Loseit | 256,849 |
| 192 | Bicycling | 142,842 |
| 199 | Running | 148,686 |
| 212 | bodyweightfitness | 138,000 |
| | **Total:** | *3,242,755* |

### Data cleansing

An algorithm must be written that loads data (Reddit data or user messages), and reduces it in-memory to a suitable format to add to the ontology. The algorithm will split the data into a list of unique words, comparing against a separate file of words that are known to the ontology. The result of this algorithm will be read by the developer and manually added to the ontology. It is therefore vital to reduce the result to a minimum. The outline for the algorithm is as follows:

```
1) Load Reddit data into memory
2) Split Reddit data into a set of unique words:
       1. Convert data to lowercase and remove punctuation for comparison.
       2. Remove stop words.
       3. Remove words greater than a length 12 as they are likely to be
          reddit waffle.
3) Return the difference of the words in the above set to the words in a file
   that contains words the ontology already knows, or the developer has
   previously disregarded.
```

### Ontology development

Data acquired, and cleansed must be parsed and allow the developer to manually assign each word to known terms in the ontology file if it is related. Otherwise, it should be skipped and added to a “known words” file to improve data cleansing of future data.

To achieve this, a command-line interface algorithm must be developed that outputs each word of the input data to the user, including the possible terms in the ontology as available options. The developer must select an available option, which generates and writes an OBO term to the ontology for that specific word. An external library will be used to write OBO terms to the file.

A possible barrier when developing this algorithm and ontology is the loss of time due to manually assigning words to terms as there may exist spelling errors, inconsistencies, or unknown words to the developer, which they have to look up. The manual assignment of words is required as there is currently no way to automatically achieve this.

### Database development

A relational database was used to store and manage the project data. Two separate database engines were used as they provide advantages depending on the development environment.

SQLite was chosen for local development due to its file system nature, and installing of dependencies was not required. PostgreSQL was chosen for production as SQLite cannot take heavy traffic [23], and is *not* recommended for production code due to lock issues [23].

As an ORM was used, the choice of production database engine was not relevant besides data storage, and support for high traffic, which PostgreSQL provides. I will now discuss the purpose behind of each table, including a description of each attribute and relationships.

### Ontological representation

The health ontology must be accessible and queryable from within the application to determine the appropriate intervention question to send to the user. To achieve this, the health ontology must be represented in a database format to enable efficient querying and comparison of terms from within the ontology against each word in the users message.

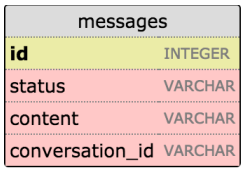
Closure tables are the simplest and most efficient way to represent a hierarchical ontology in a database [24]. They were selected over alternatives as they provide referential integrity, the ability to easily query children and sub-trees, which alternatives do not [24].

### Transitive closure

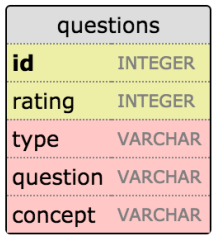
A transitive closure consists of two tables: “nodes”, which stores a parent node, and a human-readable name of the parent, and “closure”, which stores all parent-child relations and depths. These tables share relationships; the parent row in nodes references id (self-referential), and both the parent and child in closure reference nodes.id as illustrated below:


### User conversations

Two tables are used to facilitate the conversation, and are illustrated to the right in entity relationship diagrams (ERD). The “messages” table stores all messages sent or received during a conversation. The “status” attribute is used to differentiate between sent and received messages.

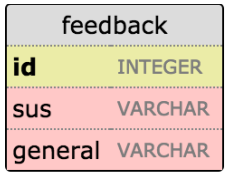



The “questions” stores all of the questions to send to the user. The “type” attribute is used to differentiate between intervention questions, e.g. initial question or clarification, the “concept” (term) and “rating” of a question are used to facilitate response selection, e.g. to select a question related to a parent of the highest rating known.




### User feedback

This table stores results from user feedback, which consists of a System Usability Scale (SUS) questionnaire form, and a text-area for general feedback by users. Both rows are represented in the ERD to the right.



### Populating database with ontology terms

An OBO file containing all ontological terms and their relationships is populated using the ontology development algorithm below. This allows the ontology to be shared between developers and applications, and is used to verify the relationships are correct through testing.

To determine the ontological terms that exist in user messages the ontology must be converted to a database representation as discussed above. The transitive closure tables will be populated from using the contents of the OBO file as outlined in the following algorithm:

1. Reads the OBO ontology to memory.
2. For each term in the OBO ontology that is not known to the database:
    1. Create a new Node for the term
    2. For each parent of the Node
        I. Collect ancestors of parent and insert each into the Closure table.
    3. Insert the Node into the Closure table

## System design

This section provides an outline of each system design choice made that was fundamental to developing the implementation for the system.

### Design patterns

The Flask framework provides several design-patterns that promote modularity and simplicity that has been incorporated within this application. This section describes each, and the benefits of their use to this project and future developers.

#### *Blueprints*

Blueprints are a technique to divide a Flask application into separate modules. This provides extensibility and isolation of testing, as related view methods, templates, and classes are grouped together. The benefits of such modularity enable additional functionality to be added to the application without affecting other components, which allow each to be tested individually.

Within this application, one blueprint is used to represent the chat components, and is assigned to the Flask application in the application factory as illustrated below. The templates or view methods have not been separated into their own modules due to the small-scale scope of this application. A blueprint was used to support such extensibility if desired by future developers.

#### *Application factory*

An application factory enables multiple instances of the application to be created by assigning all related extensions and application settings on creation. This provides control over the creation of the application, and allows multiple applications and services within the application to share these extensions.

This application uses this pattern for testing using separate configurations as described below, and to access extension functionality to initialize the database and to run the socketio server through a command line interface. The pattern used for this application is illustrated to the right.

```
from flask import Flask
from flask.ext.sqlalchemy import SQLAlchemy
from flask.ext.socketio import SocketIO

socketio = SocketIO()
db = SQLAlchemy()


def create_app(config):
    app = Flask(__name__)
    app.config.from_object(config)

    socketio.init_app(app)
    db.init_app(app)

    from app.views import chat
    app.register_blueprint(chat)

    return app
```

### *Configuration*

Configuration settings contain different settings depending on the application environment that the application is being used in, e.g. debug mode, secret keys, database configurations, etc. This application has two configurations that inherit shared configuration settings. One contains local development settings, and the other for production. This enables specific settings to be toggled with ease (i.e. debug) and different configurations to be attached to an application.

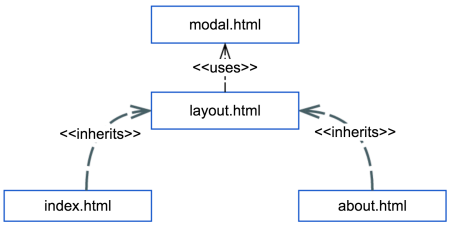
To achieve this, an environmental variable is set on the production sever to load the configuration on application initialization, which is illustrated in the following image:

```
if os.environ.get("ENV") == 'prod':
    app = create_app(ProdConfig)
else:
    app = create_app(DevConfig)
```

### *Shared templates*

Flask uses Jinja2 as the template engine, which provides *template inheritance* that allows a base *"skeleton"* template to be created that contains all common elements for the website, and defines *"blocks"* that children templates can override [25]. This prevents code duplication by promoting reusability of HTML elements, which simplifies application development.

Within this application, a base template is used (layout.html), which includes the navigation, CSS styles, JavaScript, feedback form (modal.html), and the HTML structure. An inheritance diagram of the web application templates is illustrated to the right:

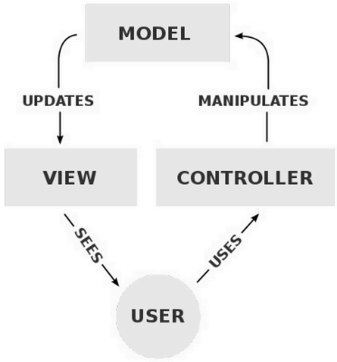



Index.html and about.html inherit the base template and override the content block to represent applicable content on that webpage to the user, which is illustrated in the image below:

```
{% extends "layout.html" %}
{% block title %}Health chat &middot; How can I help?{% endblock %}
{% block content %}
    <h1>Health chat</h1>
    <textarea id="user_message" class="form-control"
      placeholder="Type your response here and press enter to submit your message."></textarea>
    <div id="messages"></div>
{%- endblock %}
```

### Architecture

Flask does not use any framework paradigms due to its “micro” nature. Instead, the developer adapts techniques suited to their project using the foundations flask provides.




For this project, the model-view-controller (MVC) architecture was chosen as Flask provides models for data representation, and views for displaying the data to the user. Whereas controller functionality is not explicitly provided, which allows the developer to manipulate data as required by their application.

Flask models and views implemented do not depend on a class structure typical to UML modeling. Instead, the implementation details and advantages of each are discussed below.

#### *Models*

To achieve database access with an ORM a Python class must be written for each database table and must extend the Flask-SQLAlchemy base class. The resulting class will be mapped to the respective database tables at runtime to expose database access [26]. An example of this declarative definition for the application’s Message table is illustrated below:

```
class Message(db.Model):
    '''
    Read/Write the content of the message sent/received (type) from system.
    '''
    __tablename__ = 'messages'

    id = db.Column(db.Integer, nullable=False, unique=True, primary_key=True)
    status = db.Column(db.String, nullable=False)
    content = db.Column(db.String, nullable=False)
    conversation_id = db.Column(db.String, nullable=False)
```

Extending the base database model (db.Model) allows operations on the models respective tables to be performed. Each flask instance creates a database object on initialization that maps known models automatically. This enables object queries to be written, for example, to obtain the name of a term based on an id using the Nodes model:

```
row = models.Nodes.query.filter_by(id=str(parent_id)).first()
```

To create and insert a new row into the respective table for the current database session:

```
db.session.add(models.Feedback(sus=data['sus'], general=data['general']))
```

***Views***

A view function is where an HTTP request (from the user) meets the application logic (performing actions on data and displaying the results). Each view generated binds the view function to a URL route, which corresponds to the blueprint "route" method parameter as illustrated below: when a user visits '/about' the method 'about()' is invoked.

```
@chat.route('/about')
def about():
    '''
    Contains detailed information on the project, including:
        application purpose, user participation, instructions, & privacy policy.
    '''
    return render_template('about.html')
```

A view function exists for each webpage, which is used to render the corresponding templates to the user. A brief description of each:

- **index:** the application root where the conversation and user interaction occurs.
- **about:** project information, including purpose, user participation, and instructions.
- **404:** a user-friendly error message and redirects users to about.

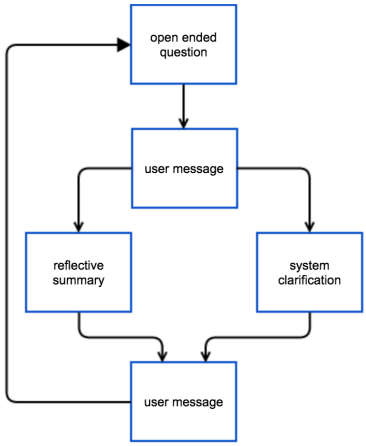
Traditionally, application logic is known as the "controller" as it performs operations on data. However, controllers within this application perform responding actions to user actions using SocketIO, while views explicitly render HTML to the user as illustrated above.

### Conversational structure

BMIs have been selected to structure the conversation between users and the system due to their advantages as described in the background section. The purpose of the conversational structural components is described below:

1. **Initial question:** provides the user with the purpose of the application and concludes with a general open-ended question to steer the conversation towards health.
2. **Open-ended questions:** used to invoke user reflection and reasoning for choices that affect their health and wellbeing.
3. **Reflective summaries:** combines reflection, affirmations (a restatement of terms in the user's message), and summaries (using previous messages to determine context).

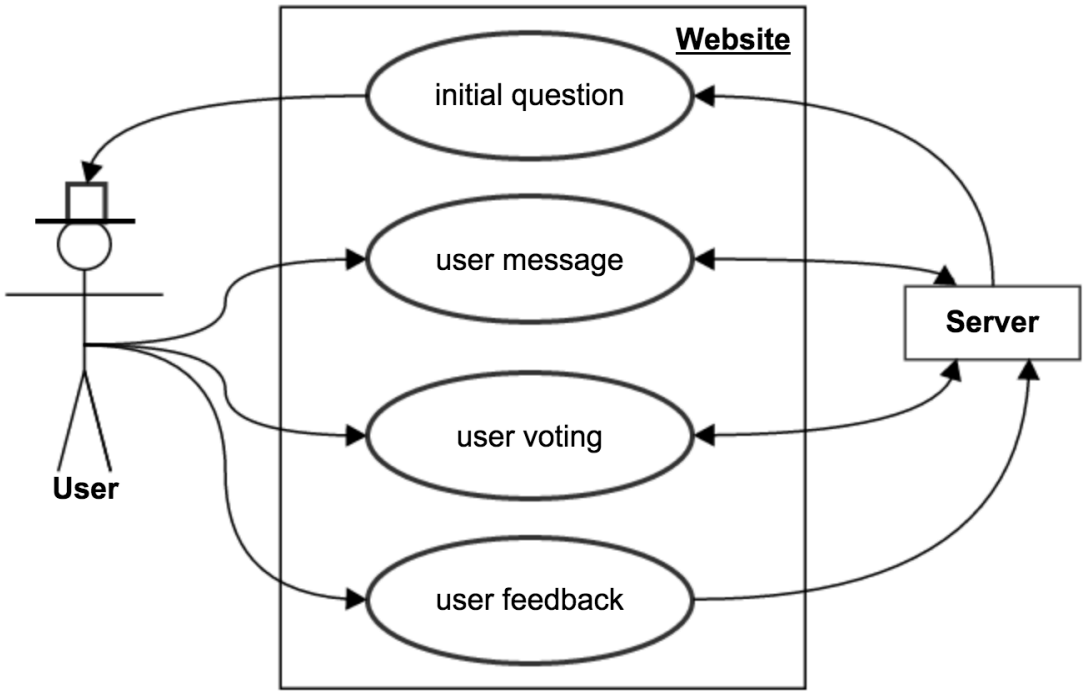
Conversations were structured to adhere to BMI techniques as discussed above with one difference; a clarification message is sent when no terms are detected in a user message.




This provides the user with the option to write a detailed response, which would bring the conversation forward and facilitate further reflection, as they have to consider the previous question.

**Conversation use cases**

Although I have written a strict set of requirements above, a use case diagram was created to identify the user interactions and data flow between the users and the system (the actors) when conversing with the website. The use case diagram below illustrates the interactions and data flow between these. The server is an actor as it dynamically processes and responds to user messages, and interacts with the core components of the system to achieve this.




***Description of use cases***

A detailed describing of each use case including the required conditions to facilitate it, and the data flow of each are described below. The actors of each use case are: user, website, server.

**Use case:** *initial question*
**Description:** the server provides the initial opening question to begin conversation with user.
**Preconditions:** the website is online.
**Basic flow:**

1. A user visits the website.
2. The interface displays the initial question to the user.

**Use case:** *user message (conversation)*
**Description:** the user sends a message to the server through the website. The system generates a response automatically and replies to the user.
**Preconditions:** the initial message has been sent, and the server and website are online.
**Basic flow:**

1. The user types a message in the free-form text box and presses enter.
2. The server sends an appropriate response.

**Use case:** *user voting*
**Description:** the user casts a vote (rates) a server response
**Preconditions:** the rating system interface appears alongside a server response.
**Basic flow:**

1. The user clicks the down-vote button.

**Alternative flow:**

1. The user clicks the up-vote button.

**Use case:** *user feedback*
**Description:** the user completes and submits the feedback form.
**Preconditions:** the website and server are online.
**Basic flow:**

1. The user completes and submits the feedback form.

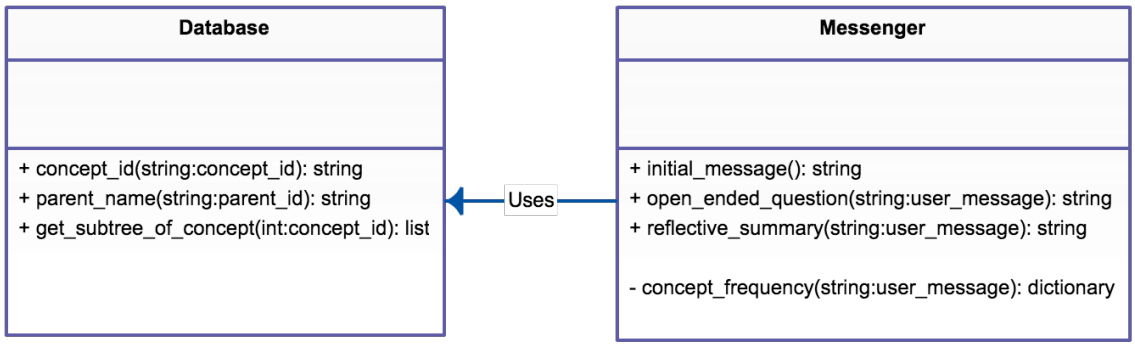
## Class diagrams

A class diagram was designed for each class to describe the functionality required to achieve the desired behavior of the system. This section describes the purpose of each class modeled and their relationships.

### *Conversation controller*

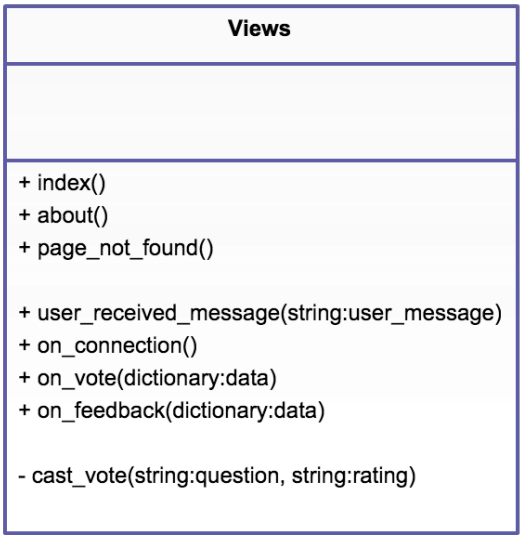
The messenger class acts as a controller in this application as it manipulates model data, and generates an appropriate response to display in the views. Each public method in Messenger adheres to the BMI conversation structure. The methods that take a user message as input (open_ended_question and reflective_summary) use the private method for context detection.

The database class contains methods that query the ontological database representation. This class is used within Messenger's private method to acquire ontological terms from the user's message. These methods are fundamental to the application and have been grouped together in isolation to accommodate unit testing. Both classes, their methods and relationship is illustrated in the following class diagram:




### *Views*

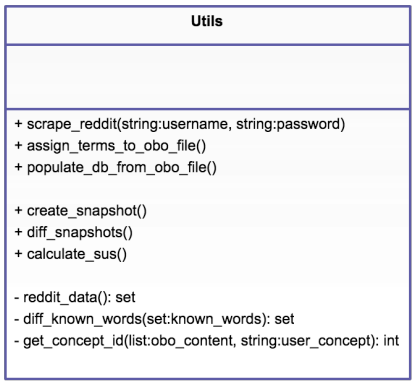
Views in Flask do not require a class, but instead blueprint views are contained within the same module. The Views module for this application contains the webpages (index, about, page_not_found), and associated messaging methods, which are illustrated to the right.




The private method (cast_vote) will be used by "on_vote" to modify the rating of a particular question in the database, and was abstracted to a separate method to simplify logic within "on_vote".

### *Utils*

The first three methods in the UML diagram to the right contain the algorithms for modeling domain knowledge for this application, and the following three are used for evaluating the system. The implementation details of each are discussed in the design and evaluation sections below.




All three private methods will be used within "assign_terms_to_obo_file", which provides a command-line interface for updating the ontology from scraped data.

### *Context detection*

This algorithm attempts to determine the context of a users message to bring the conversation forward, i.e. the topic (exercise, diet, or children of these) a user is discussing. This is achieved by determining the frequency of ontological terms in the user message. This algorithm is used by all selection algorithms to select an appropriate response, and is where the queries against the transitive closure ontological representation are made.

To achieve this, each word in the users message must be queried against the ontological representation to identify if any parent terms exist for it. A dictionary will be used to store the counter for each term. The proposed design of this algorithm is illustrated below:

```
1. terms_in_user_message = dict(key:term_name, value:counter)
2. user_words_list = split user message by unique words
3. for each word in user_words_list:
       a.  term_id = obtain the term id for the word
       b.  if term_id exists:
              i.  parent_name = obtain the parent name using term_id
             ii.  known_terms = list of terms from the Question model
            iii.  If parent name in known_terms:
                      1.  Increment parent term value in terms dictionary
4. return terms_in_user_message
```

### Response selection

Automatically generating responses is outside the scope of this project. Instead, responses for each parent term must be manually written, and stored for use by the selection algorithms discussed below (Appendix A), e.g. open-ended questions for the fruit term are illustrated:

```
fruit : [
  What do you think are the health benefits to eating fruit?,
  What impact do you think fruits have on your health?,
  What do you consider the benefits and drawbacks to eating fruit in your diet?,
  How do you incorporate fruits into your current diet?,
  How would your health differ if you incorporated more fruits into your diet?
],
```

As I am not a trained clinician the questions manually written may not adhere to the BMI principles of empathy, open-endedness, and reflection. This may limit user participation and responses to the system, which would limit the effect of the system overall.

To overcome this limitation a voting system was developed to allow users to "rate" each question, which the selection algorithm uses to choose the most suitable response. This ensures poorly written questions will be used last, which will improve conversations.

This algorithm selects a suitable response to a user message by determining the most frequent concept in the users message using the context detection algorithm described above. The response is selected based on the rating of a question, where the best is selected until no best messages exist, and then unsent responses are sent that have less rating. If all messages have been sent, then one is chosen at random to be sent. This prevents bias being introduced by the selection algorithm as other users may down-vote a response that would be helpful to other users. An outline of the desired algorithm with these characteristics is illustrated below:

1. terms_in_message = frequency dictionary of terms in user message
2. if terms_in_message is not empty:
    - unsent_questions = all questions that have not been sent
    - all_questions = all questions in database for *most frequent term*
    - unsent_highest_rated = all unsent questions for most frequent term with the highest rating
    - if unsent_highest_rated is not empty:
        i. Return a question from unsent_highest_rated at random
    - else if unsent_questions is not empty:
        i. Return a question from unsent_questions at random
    - else:
        i. Return a question from all_questions at random
3. else:
    - Return a clarification message as no terms detected in user message

## Graphical user interface design

This section describes and justifies the interface design choices for all aspects of the system.

### Responsive design

The number of devices with Internet access is increasing per year [27], which implies mobile users are likely to visit and interact with this application. It was therefore vital to design the application to function accordingly across all devices. Responsive web design forces a website to respond to the user's device. This eliminates different design and development phases for each device and reduces the burden of maintaining these on the developer.

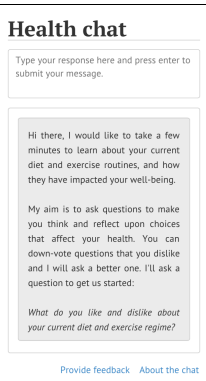
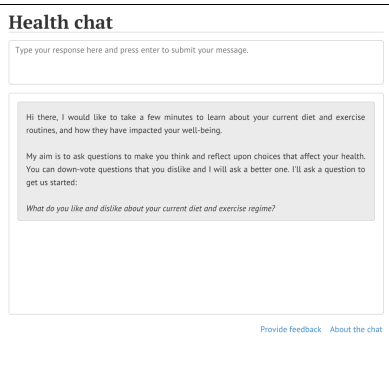
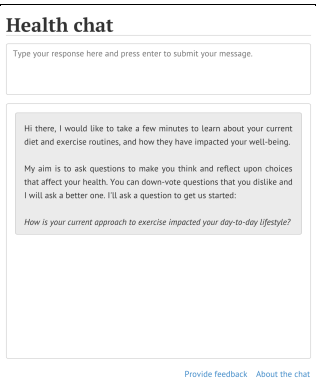
Responsive design is achieved using CSS media-queries where styles can be written within each media-query that uses a specific width. Minimum width refers to everything greater than or equal to the amount given, so in the below example styles written within it would only work on small devices, such as phones which do not have a width greater than 320 pixels.

```
@media only screen and (min-width: 480px) { /* Styles in here */ }
```

A "mobile first" approach was taken, which focuses on designing the application for a mobile first to allow the design to be centered around the key features of the application while remaining simple. Bootstrap supports this by default, and to use it I applied a set-width to the container that holds the entire application. This is expanded automatically until the application's CSS is applied, which limits the size of the container to 600 pixels on large devices:

```
@media (min-width: 1200px) { .container { max-width: 600px } }
```

The result of using this approach is that the application will have a high level of consistent usability across all devices and browsers, which will afford user interaction with the application. The design across several devices is illustrated below:

| Mobile (iPhone 6) | Tablet (Google Nexus 7) | Computer (MacBook) |
|---|---|---|
|  |  |  |

### Browser compatibility

The design of the web application remains consistent across all modern browsers due to the use of Bootstrap. This provides support for modern web features (CSS3 and HTML5) on all browsers and devices by using JavaScript on older browsers to ensure this functionality works. The client/server functionality implemented using SocketIO is supported across all browsers.

### Chat design

As the conversational aspect is fundamental to the system, research was undertaken to ensure the design of the chat system would achieve a high level of usability, readability and be aesthetically pleasing to users.

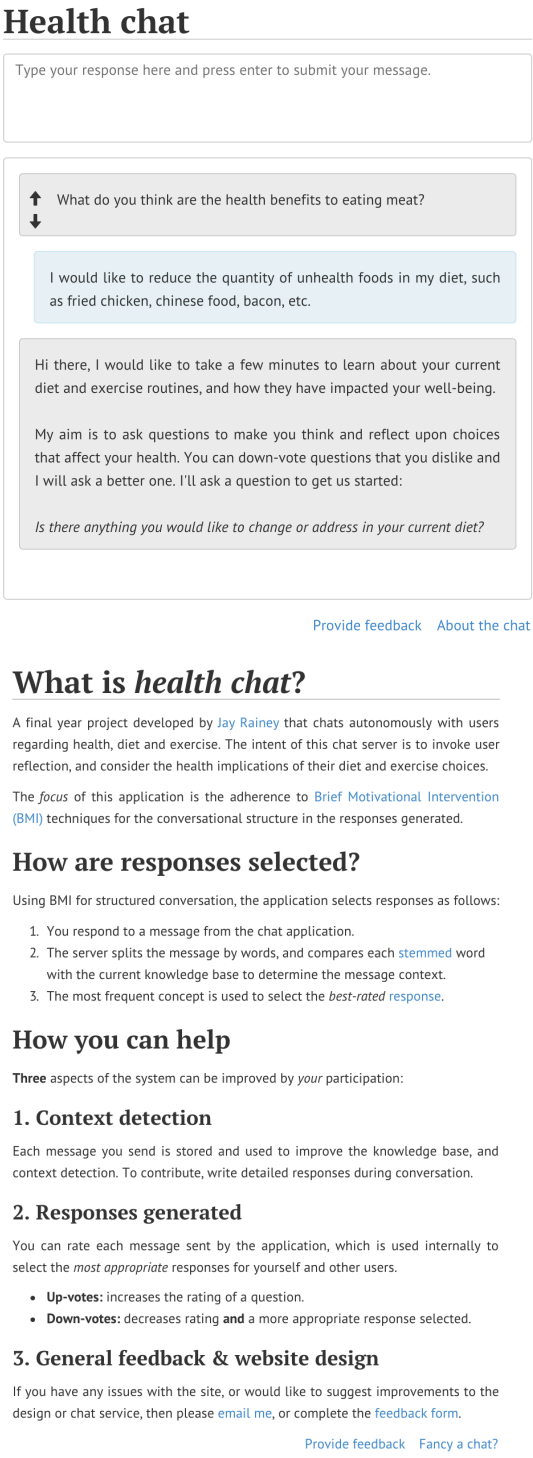
Messages have been separated to easily differentiate between each by color coordinating sent and received messages, and indenting user sent messages. These techniques allow each message to be viewed in isolation, improving readability and the context of the current conversation.

Message flow (from top to bottom) was chosen to force the user to focus on the most recent messages received as previous messages that may no longer be relevant due to the nature of the intervention used.

An adapting font-size was chosen to adhere to 45-75 characters standard across all devices [28], which improves the readability and accessibility of the chat service.

### About page

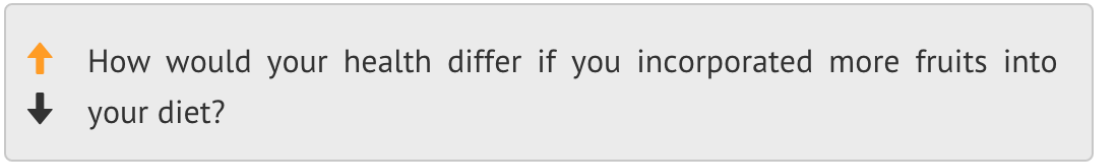
A separate webpage was developed that explains the applications functionality, purpose, and how users can participate. This was used to provide context for the application when advertising online. A screenshot is illustrated to the right.



**User participation: voting system**

A voting system must be developed to evaluate the effectiveness of the manually written responses, and to enable the response selection algorithms to choose best-known responses.

From a design perspective, directional arrows were used beside each response the system sends to the user to accommodate feedback. A two choice voting mechanism was selected over alternatives as this is proven to be effective for rating and evaluating a system [29]. The design of the voting system in use is illustrated below when a user has up-voted a response:




Accompanying algorithms must be developed both on the client and server to achieve this. On the client, an algorithm must notify the server of each vote and send the related question and vote type. The server to update the rating for that question in the database uses this.

The client must also detect when a down-vote response from the system is received and replace the related question on the interface with the new one received from the server. This provides the user with conversation control, and facilitates evaluation of system responses.

On the server, an algorithm must exist that increases/decreases the rating by .1 with each user vote until a minimum (0) or maximum (1) threshold is met. If a user has down-voted a question, then a new question is selected using the response selection algorithm (therefore, another best-rated question will be selected) will be sent to the user after the vote has been applied. The interactions between the client and server are highlighted in the following diagram:

| Client | Flow | Server |
|---|---|---|
| User casts vote on GUI for a question. | → | Updates question rating in database. |
| If “vote message” received:<br>Replace question on GUI | ← | If “down-vote” cast by user:<br>Send a new question |

# Implementation

All algorithms designed in the previous section were implemented (Appendix A) and used within the project. This section discusses the algorithms that deviate from their design, are interesting in nature, or where problems were encountered during implementation.

## Modeling domain knowledge

This section covers specific implementation details of the algorithms used to develop the system ontology, challenges faced during implementation, and how these were overcome.

### Data acquisition

It was had planned to acquire all existing data (threads and comments) from selected subreddits obtained from analysis as discussed in the design section above. However, it was discovered that Reddit uses aggressive caching mechanisms that enforce a strict limit on their API to a maximum of 1000 “most recent” comment downloads per request [30].

The Python Reddit API wrapper [31] was used to simplify access to Reddit data. The algorithm developed downloads 1000 comments for each subreddit related to a parent term, and stores the result into a file labeled with the run date and parent time, and stored in the *“data”* folder within the project. This algorithm can then be run on other days, and developers can easily differentiate between data if they wish to perform separate analysis using these files.

### Data cleansing

This algorithm generates a set of unique words from all scrapped Reddit data from the above algorithm. This is performed in memory as the result is passed to the *‘ontology development’* algorithm, and prevents files being saved that would not be used otherwise.

To reduce the size of the dataset (as each word in the resulting set is manually parsed and added to the ontology using the algorithm below) several techniques have been implemented:

1. Markdown removal: regex is used to replace none English characters with nothing.
2. Converts all letters to lowercase for set comparison later.
3. Remove English stopwords using the Natural Language ToolKit (NLTK) module.
4. Remove words greater than length 12 (double the average English word length) as words larger than this are likely to be the result of the above Regex replace.
5. Converts result to set to remove duplicates.

A file that contains words previously manually assigned to terms in the ontology (or thrown away) is used to perform a set difference to further reduce the resulting dataset.

### Ontology development

This algorithm was developed to parse the cleansed dataset of words, and to provide a command-line interface for developers to assign each word to a term in the ontology. If a word were not relevant to the ontology, then it would be inserted into a file that is later used in the data-cleansing algorithm to reduce the dataset size for future iterations of scrapped data.

An open-source OBO module was used to read, and insert OBO terms into the ontology. To build a term to insert into the ontology Python's string formatting capabilities are used as illustrated to the right.

```
term = ('\n[Term]\n'
        'id: ID:%s\n'
        'name: %s\n'
        'is_a: ID:%s ! %s\n')
```

A *"skip"* option was incorporated that adds the word output on the command-line to a separate file that stores a list of words not suitable for the ontological representation (Appendix A). This file is re-used in this data-cleansing algorithm to reduce the number of words output to the user within this algorithm. The command-line output when running this algorithm is illustrated below:

```
(venv)on master in [healthchat] » python manage.py update_ontology
Select a concept for comically from:
    beverages, dairy, fruit, grains, nuts, meat, seafood, vegetable, spices, exercise,
 endurance, weight-lifting, cardio, swimming, running, cycling or "skip"
```

This algorithm was run twice against two separate datasets[2] (Appendix A) to add terms to the ontology. No additional runs were carried out, as the ontology size was sufficient to begin user testing, and manually assigning words took a considerable amount of time that was better sent elsewhere in the project. The known words file simplified this process significantly after a single iteration, which is illustrated in the table below:

| | **Iteration one** | **Iteration two** |
|---|---|---|
| Words in cleansed dataset | 18,213 | 13,405 |
| Words assigned to known words file | 17,866 | 13,362 |
| Words assigned to ontology | 347 | 43 |

Several limitations were encountered during this process; words that I did not know, such as "molasses" had to be manually looked up to verify if they belonged in the ontology, and if yes, where. Words often contained spelling errors and had to subsequently be 'skipped', or plurals of alternative versions of words that already existed in the ontology had to also be 'skipped'.

These limitations could be overcome in the future by improving the ontology-cleansing algorithm to use natural language processing techniques, such as stemming, to reduce the dataset size output by parsing the 'know words' list, stemming each word and removing and related words from the reddit dataset.

[2] The datasets exist inside the "data" folder in the project source.
Terms added to the ontology can be viewed inside *structure.obo* in the same folder

**Database development**

The “concept_id” is arguably the most significant aspect of the system as it takes as input a word from the users message, and attempts to detect if it exists in the ontology.

Stemming was used to reduce the input word to their root word to improve comparison and detection of user words in the ontology. This overcomes a one-to-one comparison with the input word and all terms in the ontology, which is inefficient as the user may use plurals, or have spelling errors. The Snowball stemming algorithm from NLTK was chosen over alternatives (Lancaster and Porter), as it’s faster, and less aggressive [32].

To enhance this comparison a “LIKE” query was used against the ontology using the stemmed word. This produces a higher comparison rate across, which improves overall context detection in the system. The implementation details of this method are illustrated in the image below:

```
from nltk import SnowballStemmer
concept = SnowballStemmer("english").stem(str(concept))
row = models.Nodes.query.filter(
    models.Nodes.name.like("%" + concept + "%")).first()
return row.id if row else None
```

**Selecting parent term name**

This method takes the term id, and uses it to obtain the parent name. It is used in the context detection algorithm to assign parent names as keys in the counter dictionary, which is used to select a manually written response. This is achieved using two queries: the first obtains the parent id for the term, and the second obtains the name for the parent id as illustrated below:

```
# Obtains the node for the term, e.g. id, parent, name => (22, 5, apple)
row = models.Nodes.query.filter_by(id=str(term_id)).first()
if row:
    # Obtains the parent name for the parent ID of the above row.
    # Performing another query reduces the need for a complex join.
    parent = models.Nodes.query.filter_by(id=str(row.parent)).first()
    return parent.name
else:
    return None
```

**Populating database with ontology terms**

An external module[3] was used to facilitate adding OBO terms to the local OBO ontological file. This was chosen over manually writing the functionality for simplicity; the library has extensive tests that demonstrate it is correct. The contents of the applications OBO ontology are loaded into memory, which is accommodated by the module as illustrated below:

```
import obo
# Note: structure.obo MUST contain the initial concepts.
obo_content = [i for i in obo.Parser(Config.ONTOLOGY)]
```

Each term in the OBO ontology is read, and the attributes of the term are stored in variables to be used to populate the transitive closure as illustrated below. As the top-level attribute is "health", for which there are no manually written questions the parent id ("_pid") had to be zero rather than one, otherwise PostgreSQL would throw an error as closure terms could not be generated for it.

```
for i in obo_content:
    _id = int(str(i.tags['id'][0]).split(':')[1])
    # The root element does not have a parent.
    # Assign it a one as PostgreSQL does not accept zero (no parent).
    _pid = (int(str(i.tags['is_a'][0]).split(':')[1])
            if 'is_a' in str(i) else 1)
    _name = str(i.tags['name'][0])
```

A new Node is then added to the database if the term is known and generates all ancestors of the parent (as illustrated below) to populate the transitive closure with their values.

```
# Collect ancestor of parent, and insert into closure table.
values = [(i.parent, _id, i.depth + 1) for i in
          models.Closure.query.filter_by(child=_pid).all()]
```

[3] This module was obtained from a popular open-source biological gene library: https://github.com/ntamas/gfam/blob/master/gfam/go/obo.py

### Application manager

Several of the utility methods and the application server must be run through the command line interface (CLI). An external library (Flask-Script) was used to simplify the process of writing a CLI to handle tasks outside the scope of the web application.

An instance of the Flask application instance is passed to the Flask-Script constructor, which exposes access to all Flask variables. Methods that are decorated with *"command"* become command line arguments. The initialization of the database command is is illustrated to the right.

```
@manager.command
def init_db():
    '''
    Create initial database tables.
    '''
    db.drop_all()
    db.create_all()
    db.session.commit()
```

Flask-Script provides a 'runserver' argument by default that will invoke the Flask application server. Instead, this application must construct the SocketIO web server to gain exposure to asynchronous behavior. This was achieved by overriding the 'runserver' command by providing my own implementation, which calls the SocketIO run method. Methods that have been made as arguments for this CLI script are illustrated below:

```
(venv)on master in [healthchat] » python manage.py
usage: manage.py [-?]

                {update_ontology,shell,questions_to_sql,runserver,populate_ontology,init_db}
                ...

positional arguments:
  {update_ontology,shell,questions_to_sql,runserver,populate_ontology,init_db}
    update_ontology     Manually assign children to parent terms.
    shell               Runs a Python shell inside Flask application context.
    questions_to_sql    Saves pre-defined responses to database with rating 1.
    runserver           Required to run socketio rather than default manager.
    populate_ontology   Add the initial ontology data to the database.
    init_db             Create initial database tables.

optional arguments:
  -?, --help            show this help message and exit
```

## System implementation

The following sub-section discusses the implementation details of core algorithms designed above, which have deviated from their design, or where issues were encountered.

### Conversation flow

SocketIO was used to simplify message sending between the client and server. To achieve this, a method was required on the client to send data, and a matching method on the server to receive data. Methods were implemented for each conversation use case developed above.

As each implementation covers the same principles (including the user participation voting system) I have illustrated the "user feedback" use case below. When the client (end user) submits the feedback form, JavaScript is invoked that sends (emits) the form data across the websocket to the server.

The server implements the SocketIO "on" method that receives data from the client based on the namespace transported with the associated data. Once data is received for a namespace, the server calls the associated Python method, which in this use case saves the form data to the database.

Client (JavaScript)

```
socket.emit('feedback', {
  'sus': sus.toString(),
  'general': $('textarea[name=general]').val()
  });
$('#sus_feedback').modal('hide');
```

Server (Python)

```
@socketio.on('feedback', namespace='/chat')
def on_feedback(data):
    '''
    Saves user feedback form data to the database for later review.

    Args:
        data (dict): stores the System Usability Scale (SUS) results, and the
        results for a free-form general feedback.
    '''
    db.session.add(models.Feedback(sus=data['sus'], general=data['general']))
    db.session.commit()
```

### Context detection

This is the most fundamental method in the application as it builds a dictionary containing the frequency of parent terms detected in a user message, which is used to select a manually written response to send to the user in the selection algorithms. This is achieved by splitting the user message by individual words, and identifying if these exist in the ontology representation using the term detection algorithm as discussed above.

Security measures were implemented to filter user input to prevent SQL injections and other well-known security risks. To achieve this, all none English characters are removed from the users' message, and it is split by spaces (e.g. into words). The database operations are then performed using each word from the list of words generated, which limits security risks.

To prevent parents being added to the dictionary for which responses were not written, all concepts from the question model are stored in a list. This is used to verify that a parent being added to the dictionary has responses written for it; some parents do not, e.g. health, or diet.

**Response selection: open-ended questions**

This was the most fundamental algorithm in the application as it selects best-rated contextual responses based on the terms detected in the users messages. The implementation described does not differ from the design, but the techniques used to acquire the characteristics of the design are interested as illustrated below.

Firstly, if a term is detected in the users message (using the context detection algorithm described above), then a suitable response must be selected to send to users. As a best-suited response must be selected the most frequent occurring term in the user message is obtained, then all questions for that term are obtained from the database.

```
terms_in_message = self.__concept_frequency(user_message)

if terms_in_message:
    most_freq_concept = terms_in_message.most_common()[0][0]
    all_questions = (models.Question.query.filter_by(
                     concept=most_freq_concept).order_by(
                     models.Question.rating.desc()))
```

The questions are then filtered to acquire the highest rated questions, and all messages for this as illustrated below. When obtaining the highest rated questions the rating had to be routed to one decimal place as down-voted questions would differ from highest rating, and comparison of these did not provide accurate results.

```
# Rounding for distinction, e.g. .6 vs .600000000
highest_rated_questions = [i.question for i in all_questions
                           if round(i.rating, 1) ==
                           round(all_questions[0].rating, 1)]

# Made a set as we do not want to know many times each was sent.
all_sent = [i.content for i in set(models.Message.query.filter_by(
    conversation_id=request.namespace.socket.sessid,
    status='service').all())]
```

The core selection mechanism ensures the best un-sent messages are not sent (rather than repeating the previous best-sent message) until all best rated messages are sent, and then unsent messages are sent. Finally, if all messages have been sent then one is selected at random. It is thought that this selection process will prevent bias being introduced as each question has a fair chance of being sent the more the user interacts with the system.

# Results and evaluation

This section illustrates mechanisms used to verify the software developed, and the extent to which it achieved success in relation to the system requirements. Moreover, this section also highlights the results of user participation for the development of the system, which are used to validate, verify, and critically evaluate the system requirements.

## Functionality testing

This was carried out to verify that the application functioned as anticipated by end-users. The conversation use cases created in the design section were used as test cases to validate the user requirements and expected interactions with the system, which were:

1. Initial question
2. User message conversation
3. User voting
4. User feedback

A test case template was developed to provide consistency and comparison between testers:

| **Test case ID:** | | | |
|---|---|---|---|
| **Test purpose:** | | | |
| **Environment:** | | | |
| **Preconditions:** | | | |
| **Test case steps:** | | | |
| **Step number:** | **Procedure:** | **Expected response:** | **Pass or fail:** |
| | | | |
| **Tester name:** | | | |
| **Tester comments:** | | | |

Three independent testers completed each test case to verify functionality across a range of browsers, devices, and environments. All test cases passed and no errors were reported. The results of each independent test case were joined together to simplify readability as no errors were found (Appendix B). The devices and browsers tested on were:

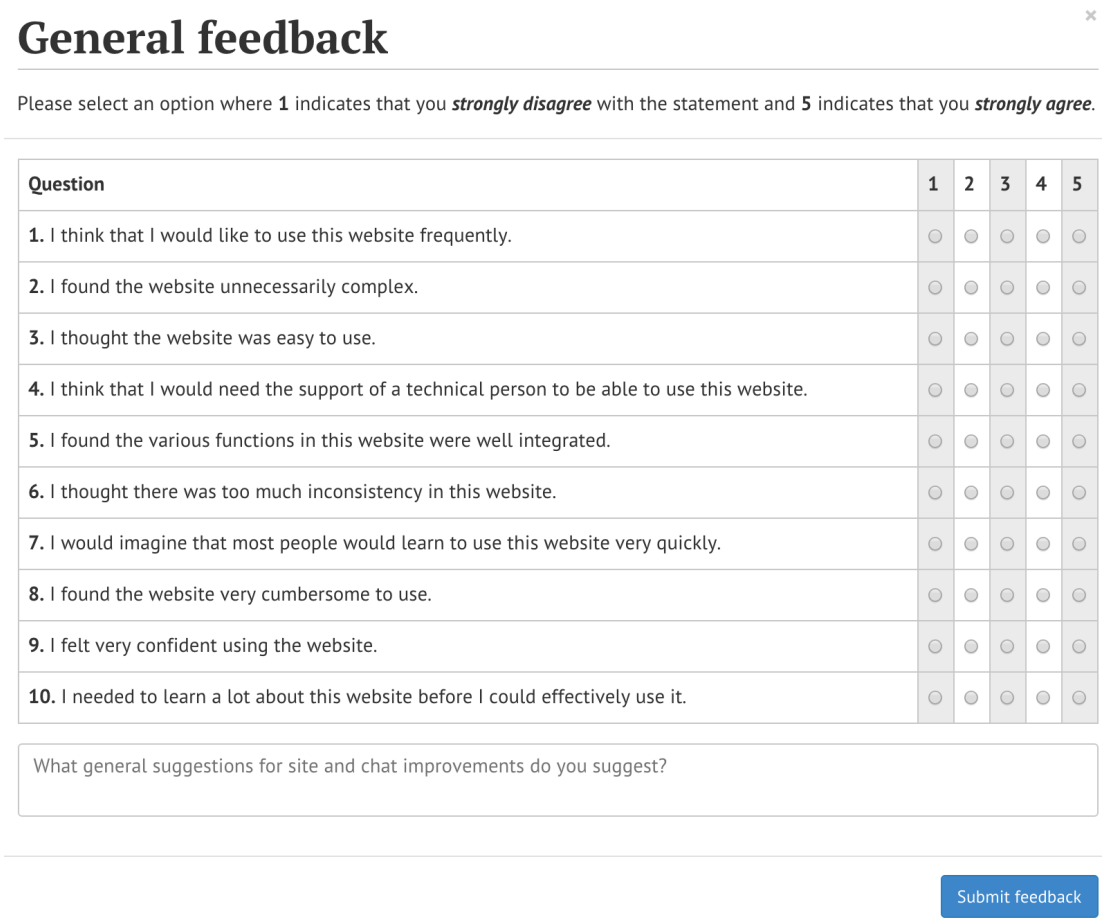
- **Windows 7:** Internet explorer 10, Firefox (version 30), and Opera (version 10).
- **OSX Mavericks:** Safari, Firefox (version 34), Google Chrome, and Opera (version 12).
- **Ubuntu 12.04:** Chromium, Firefox (version 34), and Opera (version 10).
- **Google Nexus 7** (tablet): Google Chrome.
- **iPhone 6** (phone): Google Chrome.

## Usability testing

This section describes the performance of usability testing strategies used and the measures taken to address issues raised by users. To gain user feedback a web-form was developed to acquire quantitative data through a questionnaire and qualitative data through free-form input.

### System usability scale

The System Usability Scale (SUS) is a method designed to measure the usability of a website. This is achieved using a questionnaire of ten questions on a five-point Likert scale from "Strongly agree" to "Strongly disagree" [9] as illustrated in the image below. The benefit of using SUS over alternatives is its effectiveness on small sample sizes. Furthermore, current research indicates that a sample size of 12 provides coverage for 90% of all feedback [33].



General feedback

Please select an option where **1** indicates that you ***strongly disagree*** with the statement and **5** indicates that you ***strongly agree***.

| Question | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| **1.** I think that I would like to use this website frequently. | ○ | ○ | ○ | ○ | ○ |
| **2.** I found the website unnecessarily complex. | ○ | ○ | ○ | ○ | ○ |
| **3.** I thought the website was easy to use. | ○ | ○ | ○ | ○ | ○ |
| **4.** I think that I would need the support of a technical person to be able to use this website. | ○ | ○ | ○ | ○ | ○ |
| **5.** I found the various functions in this website were well integrated. | ○ | ○ | ○ | ○ | ○ |
| **6.** I thought there was too much inconsistency in this website. | ○ | ○ | ○ | ○ | ○ |
| **7.** I would imagine that most people would learn to use this website very quickly. | ○ | ○ | ○ | ○ | ○ |
| **8.** I found the website very cumbersome to use. | ○ | ○ | ○ | ○ | ○ |
| **9.** I felt very confident using the website. | ○ | ○ | ○ | ○ | ○ |
| **10.** I needed to learn a lot about this website before I could effectively use it. | ○ | ○ | ○ | ○ | ○ |

What general suggestions for site and chat improvements do you suggest?

Submit feedback

Overall, 17 users participants provided feedback, which resulted in a SUS score of 92.5% (Appendix C). This puts the usability of this application significantly above average, and validates that this application is reliable, and fit for purpose.

### General user feedback

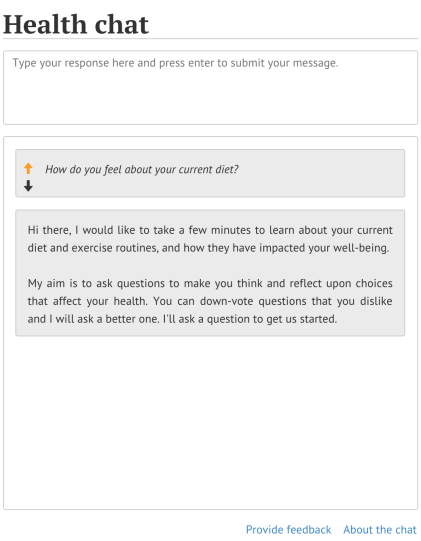
An optional free-form text field in the feedback form allows the user to suggest improves to the current website. This feedback was taken into account, and solutions to their suggestions were developed that were within the timeframe of this project, which are illustrated below:

*"When I visited the website it took quite some time to load (10 seconds), and responding to my messages took seconds at. Once his first response was sent, all others seemed to be quick."*

The free hosting provided by Heroku was used for testing the project, which automatically puts the instance of the web application to sleep when inactive. Consequently, when users visit the inactive application it takes some time to awake and begin the conversation. This could easily be overcome by using an alternative hosting service, but was outside the scope of this project.

*"This is a neat idea! One thing I would mention is that the opening question asked is a bit confusing. It would be easier if you sent it in another separate message to me. That way what I should do next is more obvious."*

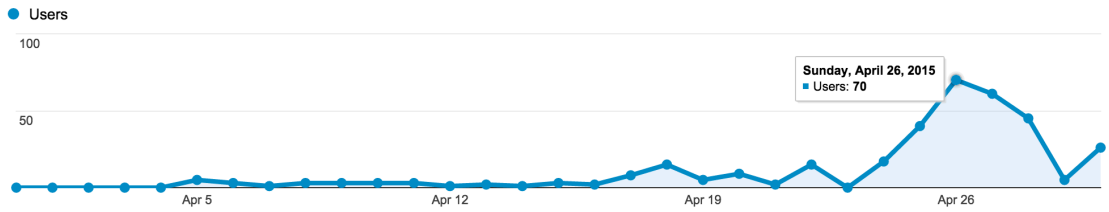
Similar comments suggesting that it would be beneficial to users if the initial question were asked in a separate message. This was incorporated as illustrated to the right, and has the advantage that users can now rate the initial question asked by the system, which improves the selection of popular questions for other users. However, as this was introduced towards the end of testing the effectiveness of this question will not be evaluated.



## User participation

This section evaluates the results of user participation and testing, and illustrates the impact they had on improving the system's performance through evaluative feedback mechanism; writing detailed responses would improve the domain-specific ontology, and rating messages would enable evaluation of intervention questions written. Although user participation was not as significant as anticipated, trends were identified as discussed in the following sections.

Google Analytics was used to capture user participation on the testing website due to its simplicity in settings up, and prior experience using it. Results conclude that 315 users participated from 45 different countries participated as illustrated below:




The website was released on April 5th to three peers who rigorously tested it for a two week period to ensure any bugs were addressed prior to releasing it to the general public.

On April 18th the application was advertised on social media (Twitter/Facebook), which increased the number of users by four. However, at this stage participants were friends, or peers due to the channels advertised on, which limited participation as they were likely to visit the website, but not participate in the application as illustrated in the results section below.

On April 24th, the application was advertised on Reddit, which significantly increased traffic for the remaining week of testing. It is stipulated that user participation at this stage was more detailed due to their unbiased with the application developer. The participants at this stage were more detailed, which provided a better opportunity to evaluate the effectiveness of BMIs.

### Evaluation of ontological development

User messages were used to improve the knowledge base in the same way as the acquired data. The ontological development algorithm was used against the user messages dataset (Appendix C). A total of 134 unique words existed in the reduced dataset, and were manually parsed using the ontological development algorithm. A total of 28 words were assigned to the ontology. This is more on average than the to the acquired dataset, which illustrates that user messages can be more effective than the acquired data for ontological development.

It is thought that the quantity of relevant terms in the user messages dataset in comparison to the acquired dataset is due to the structured conversations used. Questions asked are directly related to a parent term in the ontology, rather than general conversations that exist on the subreddits used to acquire the data.

Moreover, terms added to the ontology as a result of user participation identify a correlation between the groups of effective intervention questions as illustrated below. This demonstrates that the best intervention questions invoke better reflection from users.

The lack of detailed participation in user responses restricted the evaluative scope of this project. It is thought that the lack of participation is due to the advertising mechanisms used (social media), which may not have been the target audience for the application developed.

The detailed conversations that took place provided enough ontological development to demonstrate that the system on a large scale could be effective. This could be improved in the future through the use of controlled groups, which would focus on specific sub-domains (i.e. diet) that would allow for focused and detailed responses from users to be obtained.

### Evaluation of intervention questions

A rating system was incorporated into the system to identify the best and worst written open-ended questions based on user opinions and use of the system. This feedback mechanism was required to evaluate the effectiveness of the manually written intervention questions, and provide a means for selecting the “best” responses to respond with.

Another trend identified was that the majority of terms added to the ontology from user messages were related to the best-rated intervention questions (fruit), which are illustrated below. This could be used on a large-scale for further analysis to identify the intervention questions that are ineffective in an attempt to determine why to improve the questions written.

The majority of the intervention questions were not used, which restricts the evaluative scope of these questions. The most popular category of use was “fruit”, where all questions received ratings. One of the questions was down-voted significantly more than others, which indicates that this question was not fit for purpose as illustrated below:

```
What impact do you think fruits have on your health?, fruit, 1
How do you incorporate fruits into your current diet?, fruit, 1
What do you consider the benefits and drawbacks to eating fruit in your diet?, fruit, .7
How would your health differ if you incorporated more fruits into your diet?, fruit, .7
What do you think are the health benefits to eating fruit?, fruit, .3
```

It is observed that the majority of written responses went unused during user participation, which was especially true for the “exercise” scope of the project. This indicates that the project may have been too broad, or the sample size too small. An example of un-used term is swimming, where all ratings remained “1”, which is illustrated below:

```
What impact do you feel swimming has on your health and mental state?, swimming, 1
What do you consider are the main health benefits of regular swimming?, swimming, 1
How do you feel after having a swim?, swimming, 1
How would you describe the health benefits of swimming?, swimming, 1
Where is your favourite place to swim, and why?, swimming, 1
```

Overall, the effectiveness of the majority of the intervention questions cannot be determined due to the small-scale user testing undertaken. Although, no question received a rating below .7, which may indicate that those written were effective.

One notable limitation identified through analysis was that the stemming algorithm used to select the concept from the database would select inaccurate responses as illustrated below. This could be overcome by introducing word length limits to stem for in the ontology, or incorporating NLTK functionality to improve the context detection algorithm described above.

```
service, What impact do you think those foods (carbohydrates based) have on your health?, 798142131868
received, Essential for cell renewal (amino acids in DNA). No drawbacks unless consumed excessively., 798142131868
service, What do you consider are the main health benefits and drawbacks of eating protein heavy foods?, 798142131868
```

## Evaluation of system requirements

All functional and non-functional requirements have been met or exceeded. This section reviews the achievement of each alongside their acceptance criteria, and provides details of where their design, and implementation have been discussed elsewhere in this report.

### Functional requirements

**Requirement:** develop web-scraping algorithm to obtain domain knowledge via online sources.
**Acceptance:** data acquired must be relevant to ontological terms, and consist of at least **10,000** unique words to be used to assign to the ontology.
**Achievement:**

Two datasets of relevant data were acquired consisting of 31,968 unique words. The algorithm developed was discussed in detail the ontology development section above.

**Requirement:** develop an algorithm to clean data and insert concepts into the ontology.
**Acceptance:** only unknown words (terms of concepts) are produced from cleaned data.
**Achievement:**

The algorithm developed as discussed in the *ontology development* section above significantly reduces the acquired dataset. The mechanisms incorporated facilitate algorithm re-use to accommodate analysis of acquired datasets and user participation data.

**Requirement:** develop an algorithm to insert concepts from cleaned data to the ontology.
**Acceptance:** a command line interface that outputs *single terms* (from cleaned data) that can be assigned to known concepts in the ontology, which will update the ontology respectively.
**Achievement:**

The algorithm developed takes a set of words as input, and provide the developer with a CLI to manually assign these terms as illustrated above**.** This algorithm uses the data reduction algorithm to simplify this stage of ontology development for the developer.

**Requirement:** design and develop algorithms to select context appropriate responses.
**Acceptance:** manually written responses are selected based on the context of user messages. Determined by the frequency of words in user messages in comparison with the ontology.
**Achievement:**

A shared method was developed to promote reusability across the structured conversation selection algorithms: initial question, and open-ended question. The algorithms developed for structured conversation exceed expectations as demonstrated in the results and evaluation section above.

**Requirement:** design and develop an algorithm to transfer ontology from file to database.
**Acceptance:** transfers ontology concepts from file to a hierarchical structure in an RDBMS.
**Achievement:**

An algorithm was developed to transfer the ontology from an OBO file to a database representation (transitive closure) as discussed in the design section above.

**Requirement:** design a database to store ontology, system and user messages, and feedback.
**Acceptance:** each table stores relevant data and is queryable for system use and evaluation.
**Achievement:**

A normalized database representation was developed for each aspect of the system to accommodate user participation as discussed in the *database development* section above. Each table is used throughout the system to save or retrieve data on user interactions.

**Requirement:** design chat interface where users can enter free-form messages.
**Acceptance:** user can chat with system without restrictions in a free-form way.
**Achievement:**

The interface developed accommodates free-form user conversations, and was developed to achieve a high level of usability to afford user participation.

**Requirement:** design and develop mechanisms to improve the system through user feedback.
**Acceptance:** the following criteria should be met for each

- **System messages:** users vote to improve future selection of system messages.
- **User messages:** improves knowledge base and context detection.
- **Usability:** relevant usability form incorporated into design to evaluate system.
- **General:** free-form field where users can share their opinion on all aspects of system.

**Achievement:**

**System messages:**

A voting mechanism was designed and developed to incorporate user feedback into the system. This allowed users to rate each intervention question, which is used by the selection algorithms and discussed throughout the report.

**User messages:**

These messages were used at the end of the project to demonstrate the effectiveness of the system.

**Usability:**

Usability testing results demonstrate that the system designed was of high quality and usability by achieving a SUS of 92.5%.

**General:**

As illustrated above, a free-form field was provided to allow users to write any message to the system.

**Requirement:** quantify and evaluate effectiveness of the system.
**Acceptance:** highlight improvements through evaluations including: ontological development, response-selection algorithms, usability design, and user participation.
**Achievement:**

The results and evaluation section above discusses the effectiveness of the system through quantitative analysis of usability, functional testing, and user participation. Results indicate the effectiveness of the system, but the sample size used limited evaluative scope.

**Non-functional requirements**

***Requirement:*** *manually write responses to ontology concepts using intervention technique.*
***Acceptance:*** *5 unique responses for the initial question, and open-ended question must be written for each parent term.*
**Achievement:**

7 responses were written for initial question, and 5 responses for each parent term were written for the open-ended question and reflective summaries (Appendix A).

**Requirement:** crowd source user participation via social networking websites.
**Acceptance:** a minimum of 100 unique users must use the application.
**Achievement:**

During testing, the website had 315 unique visitors with 255 unique conversations.

**Requirement:** usability – system design
**Acceptance:** at least 12 users providing feedback and achieving system usability score (SUS) above 68 percent. This would demonstrate that the usability is above average [9].
**Achievement:**

Usability testing results demonstrate that the system designed was of high quality and usability by achieving a SUS of 92.5% with 17 participants providing feedback during testing.

**Requirement:** cross-browser support
**Acceptance:** system must look and function the same across all major browsers.
**Achievement:**

The system was tested across 5 operating systems and devices, and on 6 different browsers. No usability or functionality issues were reported during testing.

**Requirement:** reliability – chat performance
**Acceptance:** system messages must send within *10 seconds* of receiving a user message.
**Achievement:**

Although it was observed by users that the system was slow to start (due to Heroku hibernation as noted above), each message the system responded with was within seconds.

# Future work

This section provides suggestions for improving the current application by exploiting the limitations discovered when undertaking this project. The following suggestions will also be beneficial to those undertaking similar research focused projects in the area of public health.

## Control group

As user participation was undertaken without enforcing controlled conditions the results obtained above are not an ideal representation of the capabilities of this application or the affect it can have on participants.

To achieve accurate evaluation a controlled and none-controlled group of participants could interact with the application on a defined basis and their eating habits be recorded over time. This could be used to evaluate the effectiveness of the application in invoking reflection by illustrating change in user health habits over time.

## Crowd sourcing

The small quantity of participants in this application significantly limited the evaluation of the effectiveness intervention technique used. Undertaking large-scale user testing in the future would improve the validity of current results, and provide a better way to identifying trends.

This could be achieved through crowd sourcing to acquire significant user participation for better system evaluation. Several user-driven aspects of the system (voting, user messages) would benefit from such participation, and are ideal candidates for crowd sourcing use.

## Personalization

The current application was developed with a focus on achieving the objectives set out as a proof of concept. By adding personalization options prior to beginning the conversation the system can determine qualities of the user and focus the conversation towards those.

It is proposed that this would facilitate further user participation, which would help overcome the limitation of this project where several user messages lacked in quality. Possible personalization options include: users diet type, height, weight, and last meal eaten.

## Unit testing

Unit testing provides a framework to verify the functional behavior of methods independently. This provides confidence that each method and their interactions functions as designed. It was considered implementing unit tests during the final weeks of the project. Attention was instead focused elsewhere to ensure the all of the system objectives setout in the initial report were met. As the project was a proof of concept, unit tests were not considered vital.

# Conclusion

This project successfully developed software to autonomously administer interventions through a chat service to users in an attempt to address the research questions proposed. All defined objectives were exceeded, which demonstrates that the technical solution developed is usable, and suitable to addressing the problem set out to address.

The software was developed with reusability and professionalism in mind, which was incorporated through rigorous design and testing phrases, as well as writing detailed developer documentation to begin contributing to the project as is provided in the source README. This ensures that the software can be used in the future by other developers and researchers a like.

From a research perspective, the system developed proves through user participation that automatically administering BMIs using free-form user messages is possible, which had not been previously demonstrated in other research. However, as noted above, the extent to which administers these interventions were effective was outside the scope of this project.

The feedback mechanisms incorporated in the project demonstrate that user participation improves the ontology, but is insignificant in comparison to data acquired from external sources. The ontological representation is proven suitable for selecting responses to user messages; the small sample size and lack of detailed user participation limited the evaluation in determining the effectiveness of user participation.

As noted in the limitations and risks section above, the effectiveness of the intervention techniques cannot be evaluated due to insufficient user participation, but the groundwork has been set for promising research in areas of public health by adapting this software in the future.

# Reflection on learning

This section identifies the impact the learning experiences of undertaking this project have had on my personal development, and a reflection on the decisions made that impacted the project.

## Project scope

The project undertaken enabled me to gain insight in multiple areas of interest: ontologies and their use, natural language processing, and user design, while gaining exposure to research in other disciplines (public health). This broadened my personal research interests, which ultimately motivated me to pursue postgraduate studies in this area.

The breath of exposure this project arguably limited the scope of results. If instead the project focused on gaining a deep understanding of a specific sub-area of the defined problem then more useful results may have been obtained, for example, if the ontology focused on food (or going further, a specific diet), rather than food *and* exercise.

## Project management

The work plan developed in the initial report lacked rigor and detail. This slowed progress during the project implementation phase as the low-level implementation of each component had not been considered. Leeway introduced in the initial plan facilitated moving user testing two weeks forward when the final report was being written. This prevented thorough analysis and evaluation of the results, as they had to be written in the last few days.

Moreover, when carrying out similar large-scale projects in the future I will ensure reports and documentation are written alongside each stage as writing retrospectively from notes took a considerable amount of time, which limited the detail I could go into during the evaluation stage, which was left last due to user participation.

## User participation

User participation demonstrated that the usability and feedback mechanisms developed were highly effective. However, due to the small number of participants the extent to which this analysis can be used is limited. Instead, I propose that experiments and evaluations be undertaken on a controlled group of users, to re-evaluate the effectiveness of the system in achieving the desired research objectives.

Moreover, a small group of beta-testers was used during the initial testing phase to validate the system prior to advertising it on social media. This allows a sub-set of users to test the system, and report flaws, which could be immediately fixed. This type of testing proved effective as several potential flaws were reported and addressed without impacting the majority of users.

# References


[1] Office for National Statistics. Mortality statistics: deaths registered in England and Wales. *http://www.ons.gov.uk/ons/rel/vsob1/mortality-statistics--deaths-registered-in-england-and-wales--series-dr-/2013/index.html* (accessed 29 January 2015)

[2] Hustad J, Mastroleo N. *The comparative effectiveness of individual and group brief motivational interventions for mandated college students.* 2014.

[3] Gaume J, Gmel G. *Is brief motivational intervention effective in reducing alcohol use among young men voluntarily receiving it? A randomized controlled trial.* 2011.

[4] Field C, Hungerford D, Dunn C. *Brief Motivational Interventions: An introduction.* 2005

[5] Bickmore T, Schulman D, Shaw G. *A reusable framework for health counseling dialogue systems based on a behavioral medicine ontology.* 2011.

[6] Stevens VJ, Smith KS. *One-year results from a brief, computer-assisted intervention to decrease consumption of fat and increase consumption of fruits and vegetables.* 2003.

[7] Miller W, Rollnick S. *Motivational Interviewing: preparing people for change.* 2002.

[8] Flask. *What does micro mean.* http://flask.pocoo.org/docs/0.10/foreword/#what-does-micro-mean (accessed 30th January 2015).

[9] Broke J. *SUS - A quick and dirty usability scale.* 1996.

[10] Guarino N, Oberle N, Staab S. *What Is an Ontology?* 2009.

[11] W3C. *Resource Description Framework (RDF).* http://www.w3.org/RDF/ (accessed 30th January 2015)

[12] obofoundry. *The open biological and biomedical ontologies.* http://www.obofoundry.org/ (accessed 30th January 2015).

[13] W3C. *OWL Web Ontology Language.* http://www.w3.org/TR/owl-features/ (accessed 30th January 2015).

[14] BioPortal. *The world's most comprehensive repository of biomedical ontologies.* http://bioportal.bioontology.org/ (accessed 30th January 2015).

[15] BioPortal. *Computer retrieval of information on scientific projects thesaurus.* http://bioportal.bioontology.org/ontologies/CRISP/ (accessed 1st February 2015).

[16] BioPortal. *Read Codes, Clinical Terms Version 3 (CTV3).* http://bioportal.bioontology.org/ontologies/RCD/ (accessed 1st February 2015).

[17] BioPortal. *Robert Hoehndorf Version of MeSH.* http://bioportal.bioontology.org/ontologies/RH-MESH/ (accessed 1st February 2015).

[18] BioPortal. *Medical Subject Headings.* http://bioportal.bioontology.org/ontologies/MESH/ (accessed 1st February 2015).

[19] BioPortal. *National Cancer Institute Thesaurus.* http://bioportal.bioontology.org/ontologies/NCIT/ (accessed 1st February 2015).

[20] Alexa. *How popular is reddit.com?* http://www.alexa.com/siteinfo/reddit.com (accessed 1st February 2015)

[21] Reddit. *About reddit.* http://www.reddit.com/about/ (accessed 1st February 2015).

[22] Redditlist. *Tracking the top 5000 subreddits.* http://redditlist.com/ (accessed 9th February 2015).

[23] SQLite. *Appropriate uses for SQLite.* https://www.sqlite.org/whentouse.html (accessed 10th February 2015).

[24] Karwin B. *Models for hierarchical data.* http://www.slideshare.net/billkarwin/models-for-hierarchical-data (accessed 10th February 2015).

[25] Flask. *Template inheritance.* http://flask.pocoo.org/docs/0.10/patterns/templateinheritance/#template-inheritance (accessed 10th February 2015).

[26] SQLAlchemy. *SQLAlchemy 0.8 Documentation.* http://docs.sqlalchemy.org/en/rel_0_8/orm/extensions/declarative.html (accessed 10th February 2015).

[27] ONS. *Internet Access – Households and individuals 2014.* http://www.ons.gov.uk/ons/rel/rdit2/internet-access---households-and-individuals/2014/stb-ia-2014.html (accessed 11th February 2015).

[28] Craig J. *Designing With Type.* 1971.

[29] Chen M, Singh J. *Computing and using reputations for Internet ratings.* 2001.

[30] PRAW. *Non-obvious behavior and other need to know.* https://praw.readthedocs.org/en/latest/pages/faq.html#non-obvious-behavior-and-other-need-to-know (accessed 13th February 2015)

[31] PRAW. *PRAW: The Python Reddit API Wrapper.* https://praw.readthedocs.org/ (accessed 13th February 2015).

[32] Snowball. *Snowball: A language for stemming algorithms.* http://snowball.tartarus.org/texts/introduction.html (accessed 14th February 2015).

[33] Tullis T, Stetson J. *A Comparison of Questionnaires for Assessing Website Usability.* 2004.

# Appendices

The name of each appendix folder and a description is listed below:

**Appendix A:** *source-code*

This folder contains a clean copy of the project source code. The README file describes the steps necessary to setup, run, and use the project locally and in production. The "data" folder within this folder contains the manually written responses (questions.json), the acquired data in .txt format (labeled by date), the OBO ontology (structure.OBO), and the known words file.

**Appendix B:** *test-cases*

This folder contains the results of functional testing in both PDF and DOC format.

**Appendix C:** *results*

This folder containing two sub-folders: "usability" and "user-participation" containing the results of user participation, and a script used to generate the results from the Heroku database[4]. A brief description of each file is listed below:

**db_copy.py**

Generates a copy of the database when the method exists in manage.py

**usability**

*general.txt*

The general free-form comments made by users during user feedback.

*SUS.txt*

The SUS scores of acquired from user feedback.

*SUS.py*

A script created to generate the SUS from SUS.txt

**user-participation**

*question-ratings.txt*

The rating of each open-ended question after user participation

*user-messages.txt*

The messages users sent during conversations with the system.

[4] Issues were encountered when generating a database backup on Heroku.
Instead, *db_copy.py* was written that copies the contents of the database to file.