Database Module Documentation

Aim:

- To store FAQ questions and answers.
- To store user queries which could not be answered successfully.
- To provide feature to populate, edit and download database.

Technology used:

• Django web framework.

• MySQL

• pip

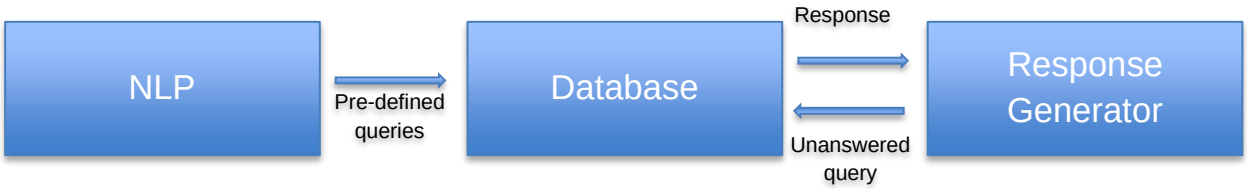
Description of technology used:

• Django is a Python-based free and open-source web framework that follows the model-template-view architectural pattern. It is maintained by the Django Software Foundation.

• MySQL is an open-source relational database management system. MySQL is free and open- source software under the terms of the GNU General Public License, and is also available under a variety of proprietary licenses.

• **PIP** is a package management system used to install and manage software packages written in Python.

Architecture Diagram:



Directory structure in database module:

1. .vscode:

    Visual Studio code settings are stored here in settings.json.

    1. settings.json:

        "python.pythonPath" is the path to the python interpreter used for debugging or running the code.

2. database_chatbot:

    1. __pycache__:

        When a program in python is run, the interpreter compiles it to bytecode

first and stores it in the __pycache__ folder. It makes the program start a little faster. When the scripts change, they will be recompiled, and if the files or the whole folder is deleted the program is run again, they will reappear.

1. __init__.cpython-36.pyc
2. settings.cpython-36.pyc
3. urls.cpython-36.pyc
4. wsgi.cpython-36.pyc

These are bytecode-compiled and optimized bytecode-compiled versions of the program's files.

2. __init__.py:

The __init__.py file makes Python treat directories containing it as modules. Furthermore, this is the first file to be loaded in a module, so it can be used to execute code that has to be run each time a module is loaded, or specify the submodules to be exported.

3. settings.py:

A Django settings file contains all the configuration of your Django Installation.

4. urls.py:

This module is pure Python code and is a mapping between URL path expressions to Python functions..

5. wsgi.py:

WSGI is a specification for a standardized interface between Web servers and Python Web frameworks/applications. The goal is to provide a relatively simple yet comprehensive interface capable of supporting all (or most) interactions between a Web server and a Web framework.

3. queries:

1. __pycache__:
    1. __init__.cpython-36.pyc
    2. admin.cpython-36.pyc
    3. apps.cpython-36.pyc
    4. models.cpython-36.pyc
2. __init__.py
3. admin.py:

This file is used to display the models in django admin panel. It is also allows us to customise the admin panel.

Classes:

1. class QueryResource(resources.ModelResource):

To integrate django-import-export with our Query model, we will create a QueryResource class in admin.py that will describe how this resource can be imported or exported.

2. class Meta:

By default QueryResource introspects model fields and creates Field-attributes with an appropriate Widget for each field. To affect which model fields will be included in an import-export resource, use the fields option to whitelist fields.

Attributes -

model : model in use.

skip_unchanged : skip import of unchanged records
report_skipped : Report what columns are skipped

exclude : What fields to exclude

import_id_fields : What fields to include in import

fields : What fields to include in import export resource.

3. class QueryAdmin(ImportExportModelAdmin):

Admin integration is achieved by subclassing ImportExportModelAdmin or one of the available mixins. It registers our QueryResource in Admin panel.

4. apps.py:

This file is created to help the user include any application configuration for the app. Using this, you can configure some of the attributes of the application.

5. models.py:

A model is a class that represents table or collection in the DB, and where every attribute of the class is a field of the table or collection. Models are defined in the app/models.py.

Classes:

1. class Query(models.Model):

Class for storing the fields as class attributes.

intent and response are database field of the model. They are specified as class attributes, and map to a database column.

2. class Unanswered_Query(models.Model):

Class for storing the fields as class attributes.

unanswered_query are database field of the model.

unanswered_query is the only attribute of the model.

6. nlp_module.py:

This python script return the most similar sentence for a query dynamically from the database.

7. response_generator_1.py:

With the intent identified, the response generator returns the response from the database.

8. serializers.py:

Serializers allow complex data such as querysets and model instances to

be converted to native Python datatypes that can then be easily rendered into JSON, XML or other content types. Serializers also provide deserialization, allowing parsed data to be converted back into complex types, after first validating the incoming data.

Classes:

1. class query1Serializer(serializers.ModelSerializer):

Class for serializing user query.

9. tests.py:

Testing a website is a complex task, because it is made of several layers of logic – from HTTP-level request handling, queries models, to form validation and processing, and template rendering. Django provides a test framework with a small hierarchy of classes that build on the Python standard unittest library.

10. urls.py:

This module is pure Python code and is a mapping between URL path expressions to Python functions.

11. views.py:

A view sources data from your database (or an external data source or service) and delivers it to a template. A view function is a Python function that takes a Web request and returns a Web response to the frontend.

Methods:

1. def message_list(request, sender=None, receiver=None):

List all required messages, or create a new message.

Args:

request(Json): Contains the JSON of the user input

sender:To store the id of the sender, default = None

receiver : To store the id of the reciever, default = None

Returns:

body(dictionary): Returned as JSON to the frontend using JsonResponse subclass

4. Readme.md

5. manage.py:

Every Django project starts with a manage.py file in its root. It's a convenience script that allows you to run administrative tasks like Django's included django-admin.

6. requirements.txt:

    Specifies what python packages are required for running the project.