Kevin Hind

Kevin.hind@clarivate.com

Holiday Chat Agent

Contents

[Introduction 2](#_Toc130566762)

[Requirements Analysis and design 2](#_Toc130566763)

[‘Minimum of two questions to identify possible holiday destinations’ 2](#_Toc130566764)

[‘User feedback to questions the Chat Agent cannot process the response to’ 3](#_Toc130566765)

[‘Recommended set of holidays based on responses’ 3](#_Toc130566766)

[Handing the data 4](#_Toc130566767)

[User interface 4](#_Toc130566768)

[Database design 5](#_Toc130566769)

[Development 7](#_Toc130566770)

[Development environment 7](#_Toc130566771)

[Database work 8](#_Toc130566772)

[Database creation 8](#_Toc130566773)

[Populating the *holiday* table 10](#_Toc130566774)

[Creating the backend code 11](#_Toc130566775)

[Introduction and development approach 11](#_Toc130566776)

[Function to obtain different types of data from the database table 13](#_Toc130566777)

[Output when there are no recommendations 18](#_Toc130566778)

[Building the front end application 20](#_Toc130566779)

[Setting up a new Django project and application 20](#_Toc130566780)

[Adding interactivity to retrieve data 23](#_Toc130566781)

[Creating additional questions 32](#_Toc130566782)

[Creating the ‘location’ question 39](#_Toc130566783)

[User acceptance testing 40](#_Toc130566784)

[Bug fixing: user selects more than one option in each category 43](#_Toc130566785)

[Non-functional requirements 46](#_Toc130566786)

[Suggestions for other functional requirements 47](#_Toc130566787)

[Conclusion 48](#_Toc130566788)

# Introduction

I have been assigned a project by First Holiday Ltd (“the Company”) which is a holiday business selling package holidays. They would like to create a chat agent (“the Application”) providing customers with a recommended set of holidays based on their responses to questions. The high-level requirements are as follows:

* The component will be a functional prototype used to demonstrate to shareholders and test, initially with a smaller number of customers;
* The component may be run on any execution environment of my choice in a browser, or as an application on a desktop computer or a mobile computing device;
* The component should launch as a stand-alone application in this environment;
* The component should be designed and built to production standards.

From these high-level requirements, I understand that the prototype will need to be deployed in such a way that it can be used by customers as opposed to developers (a command-line interface is not acceptable). However, I have a choice over the execution environment for the prototype. As the prototype is required to launch as a stand-alone application, it needs to be able to work offline without a network connection and not part of an existing process.

# Requirements Analysis and design

## ‘Minimum of two questions to identify possible holiday destinations’

The Application must ask the user a minimum of two questions to identify possible holiday destinations and this must be based on the holiday data set. Here is a use case for this requirement:

*Use case*

As a user of the Chat Agent, I want to be able to answer at least two questions before receiving holiday recommendations.

The Company will want to ensure that the chat agent tailors recommendations as closely as possible to the preferences of individual customers. Therefore, it will be necessary for the Application to filter out prospective holidays dependent on a series of questions.

The Company has provided a number of criteria which can be used to work out the holiday preferences of an individual user. The Company provides the following criteria:

* Category: The type of holiday with a choice of ‘active’ or ‘lazy’;
* Star Rating: The type of accommodation with a choice from 3 to 5;
* Temp Rating: The temperature of the resort, with a choice of ‘cold’, ‘mild’, or ‘hot’;
* Location: The type of location, with a choice of ‘sea’, ‘city’, or ‘mountain’;
* The price per night.

While the Company has also provided information about cities, continents and countries I feel that for the purposes of the Application it will be sufficient to provide recommendations based on the answers to the questions about category, temperature, star-rating, and location.

## ‘User feedback to questions the Chat Agent cannot process the response to’

The Application must provide user feedback to questions that the Application cannot process the response to. Here is a use case for this requirement:

*Use case*

As a user of the Chat Agent, if the Chat Agent cannot process a response, I would like to receive helpful feedback.

However, I think the likelihood of non-processable responses could be reduced by basing the questions asked on the data set and limiting the possible answers to those options within each of the criteria.

While it will require more detailed analysis of the data to determine whether this is the case, it could be that a particular set of customer choices produces no recommendations. If this is the case then I have decided to make the application default to a standard response which will give users the option of speaking to a travel agent in person. I have not been provided with any contact details so this will need to be discussed with the Company. As part of the prototype, I will supply a dummy telephone number.

## ‘Recommended set of holidays based on responses’

Provided that the answers to the questions produce a set of results from the existing data, the Application will provide a set of holidays based on user responses. Here is a use case for this requirement:

*Use case*

As a user of the Chat Agent, I would like to receive a set of holiday recommendations based on my responses to the questions.

I have decided that the end results will display in the browser on separate lines giving the city, the name of the hotel, the country and the price. The name of the city and the price per night will be in larger fonts than the other information for the purposes of drawing the customer’s attention to them:

Your holiday recommendations

|  |  |  |  |
| --- | --- | --- | --- |
| Marrakech | (Morocco) | £50 per night | **Hotel name:** Twighlight Hotel |
| New York | (USA) | £27 per night | **Hotel name:** AppleCity Hotel |

## Handing the data

I have been told that the data is to be loaded from a local file into the Application and can be converted into another format for the Application to load. Furthermore, responses to question will need to be stored temporarily in order to provide holiday recommendations. Here is a use case for the requirement:

*Use case*

As an internal employee of the Company, I want to be able to update the recommendations that are available for prospective customers.

My proposal is to store the data in a database so it can be retrieved and filtered more efficiently at runtime. Given that I will be using Python to perform the data filtering, I think it would be easier for the code to interact with a database as opposed to an alternative structure such as a list of dictionaries, particularly if the dataset is likely to grow in size. I also think it will be easier for the company to update a database as opposed to an alternative data structure.

## User interface

The Graphic User Interface (GUI) should show a trail of the chat conversation for the session. I have interpreted this requirement as meaning that the answers to each question should be displayed on the screen so the user can see the full trail of the conversation before receiving the recommendations. Here is a sketch of how this could look:

Hello. This is [name of Chat Agent].

First of all, what type of holiday are you looking for? Please choose from ‘lazy’ or ‘active’

active

Thanks. What sort of climate would you like to travel to? Please choose from ‘cold’, ‘mild’, or ‘hot’.

hot

Thanks. What sort of location would you like to travel to? Please choose from ‘sea’, ‘city’, or ‘mountain’.

sea

Thanks. Thinking about your accommodation for the stay, what star rating would you prefer? Please choose from ‘3’, ‘4’, or ‘5’.

3

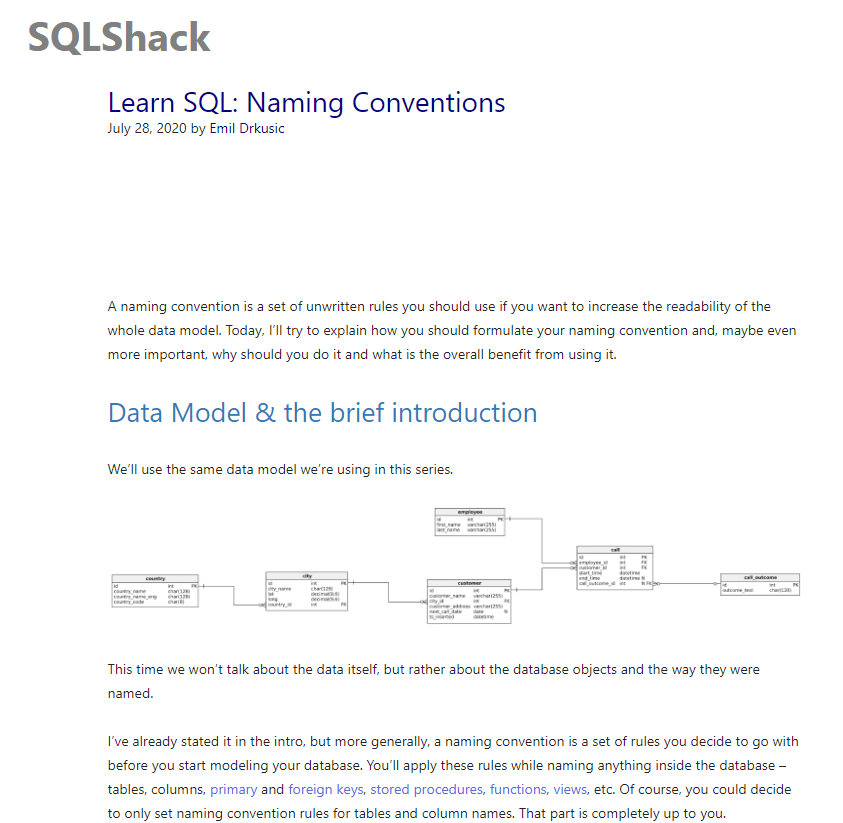
[Holiday recommendations appear below]

In terms of the questions, I will need to work out whether restricting the number of user inputs makes the Application less helpful to users. For example, it could be that providing a particular set of answers produces no recommendations. For this reason, as part of considering non-functional requirements, it could be that there should be the option to select ‘don’t mind’ as a potential option so that possible recommendations are not filtered out.

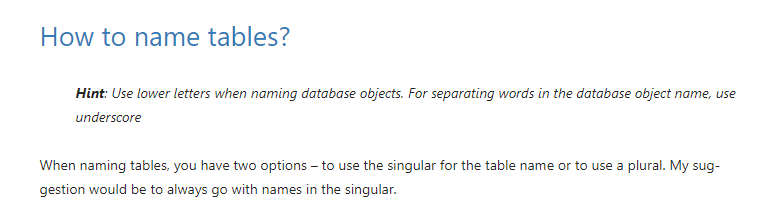
## Database design

I will represent the data in a relational database because this will allow me to link different entities together as part of filtering out the options based on user feedback. Furthermore, Python can be used as part of the Django framework to link to a database using SQL.

To design my database, I will follow SQL conventions as described on the ‘SQLShack’ website:



I will name each attribute using lowercase letters and underscores where the names consist of more than one word. I will also be using names in their singular form (see the screenshot on SQLShack below):



Here is an entity relationship diagram showing how the database table will be structured and the attributes that each entity will have:

Table

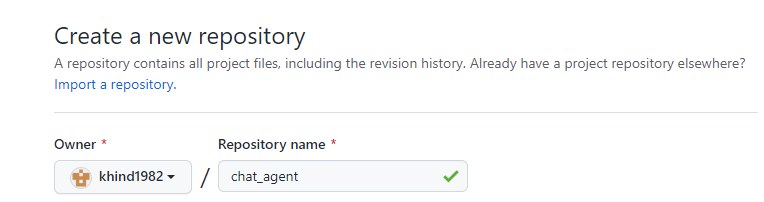
Description automatically generated

For each attribute, I have used the same name as the spreadsheet headers apart from ‘HolidayReference’ which I have changed to ‘HolidayID’ and ‘PricePerPerNight’ which I have changed to ‘Price’. For ‘HolidayID’ and ‘Price’ I have selected the Type value of ‘int’ because the values will always be numerical. For all the other attributes I have selected ‘varchar’ because the values can be strings of variable length.

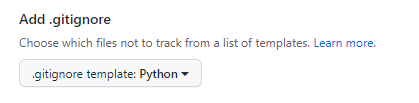
# Development

## Development environment

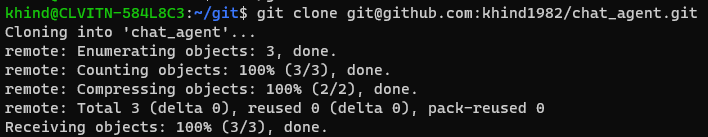
I have decided to set up a git repository to hold my code for the purposes of version control and development. I will use my personal github account and begin by setting up a bare repository:



I will also add a ‘.gitignore’ file using the ‘Python’ template:



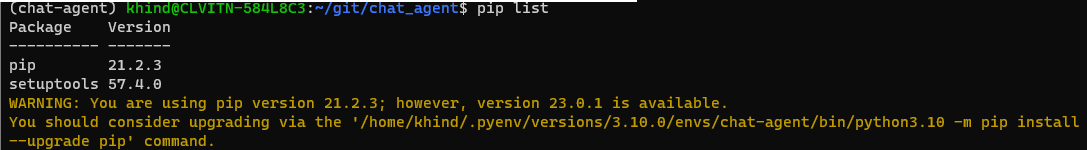
I now clone the git repository into my WSL Linux distribution. I will not be using my own company’s internal Linux distribution to protect our own company’s data for the purposes of this task:



As I am developing within Python, I will create a Python virtual environment within my Linux distribution so I can import any packages I need for the project. This will also make deployment easier. In my company, ‘pyenv’ is used to set up and deploy Python virtual environments using multiple Python versions. This is because the pyenv-virtualenv plugin makes it easier to work with multiple virtual environments:



As the pyenv version is not up-to-date, I will update it to be benefit from the latest updates and security patches:



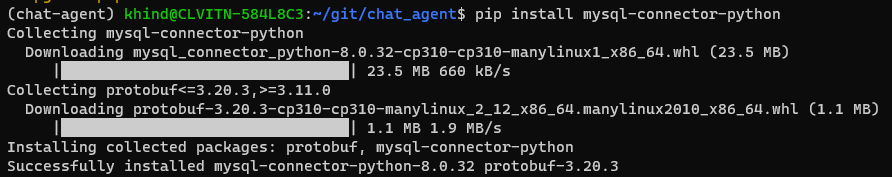
## Database work

### Database creation

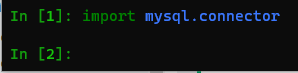
To create the database I need for the holiday data, I will be using the Python mysql module and following the W3 Schools documentation:



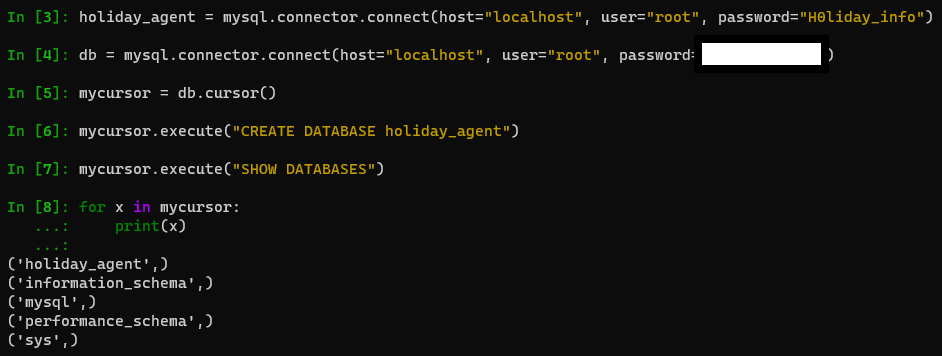
I start by installing the ‘MySQL Connector’ into my virtual environment:



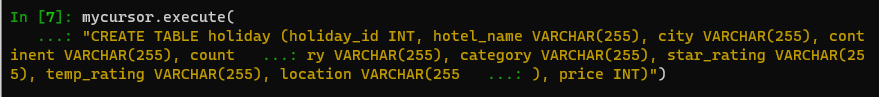
I will now activate an ipython session by attempting to import the SQL connector. This appears to work:



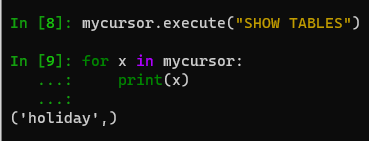
I create a new database called ‘holiday\_agent’ and check it exists:



Now I create a new database table for the holiday data:



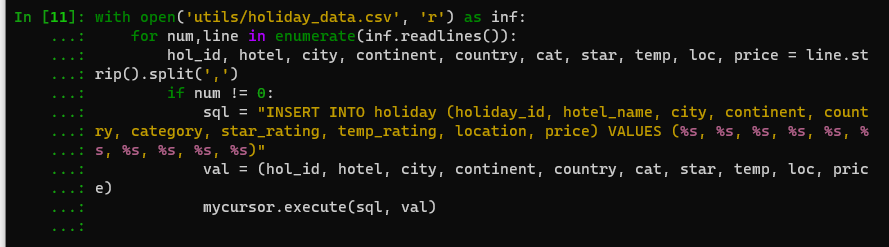
I then check to see whether the table exists:



### Populating the *holiday* table

The holiday data currently resides in comma-separated CSV file. In order to populate the database, I need to transfer the data from the spreadsheet into the table. To transfer the data into the table, I will use Python to extract the data.

I have copied the CSV file into my git repository for the holiday agent project so I can access it more easily, naming it ‘holiday\_data’ to distinguish it from other files in the repository. I now use commands in ipython to transfer the data into the ‘holiday’ table:



I then check the content of the table to ensure everything has been added:



I have now created a database from which I can retrieve the data I need to build the Application.

## Creating the backend code

### Introduction and development approach

I am now going to turn to writing a Python function that will perform one of the Chat Agent functions.

As I write the code, I will be using a test-driven development approach to produce this part of the Application. The reason for choosing this approach is that our team have found that using TDD produces cleaner code which is more easily maintainable and understandable by third parties.

According to the ‘Agile Alliance’, an online community supporting Agile software development methodologies, the TDD process is as follows:

* Write a test: a single unit test which describes an aspect of the program which has not yet been implemented in code
* Run the test: Observe that the test will fail because the program lacks the feature being tested
* Write code: The code written should be just enough to make the test pass
* Refactor: Make changes to the code until it conforms to a degree of simplicity, for example no duplication and each section of code expresses a distinct idea or responsibility.
* Repeat the process for each new feature

From the Agile Alliance website:

A screenshot of a computer

Description automatically generated with low confidence

It is a combination of working iteratively and refactoring code at every stage which leads to applications which are more easily maintainable and understandable. In effect, the tests tell the ‘story’ of how the app was built.

The reason for using pytest over and above other testing frameworks like unittest is that in pytest there are inbuilt features which require less code when compared to frameworks like unittest. In unittest we create classes that are derived from the unittest.TestCase module and define test methods inside the class. In the case of pytest we only have to define test functions and assert conditions inside them. I will be referring to the pytest documentation below using screenshots from the website:

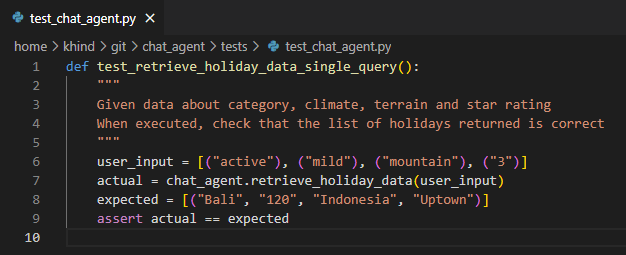
Table

Description automatically generated with medium confidence

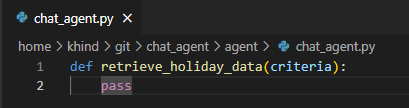
### Function to obtain different types of data from the database table

My first goal is to write a Python function that will customer choices and retrieve the records from the database that relate to those choices. At this point I am assuming that the customer has already entered their choices into the Application so the aim of this function is to take those choices and look in the database. I will be using VS Code as my IDE because VS Code contains plugins and linters for working with Python that also shows Python syntax.

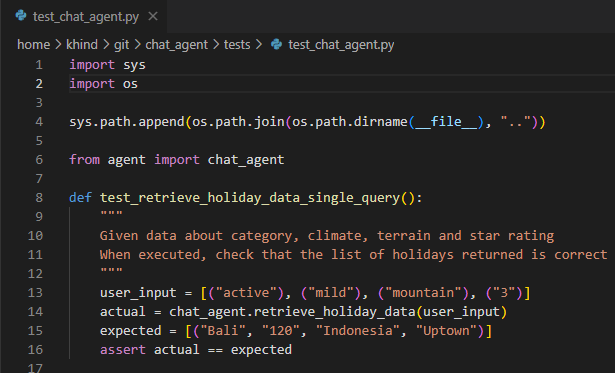
First of all, I create a test that uses the function I intend to create to retrieve a single record from the database. Looking back at my Application design, I explained that in retrieving data for the customer I wanted to get the city, price, country and hotel information. Therefore, the return value of my function will be a list containing city, price, country and hotel information for each holiday.



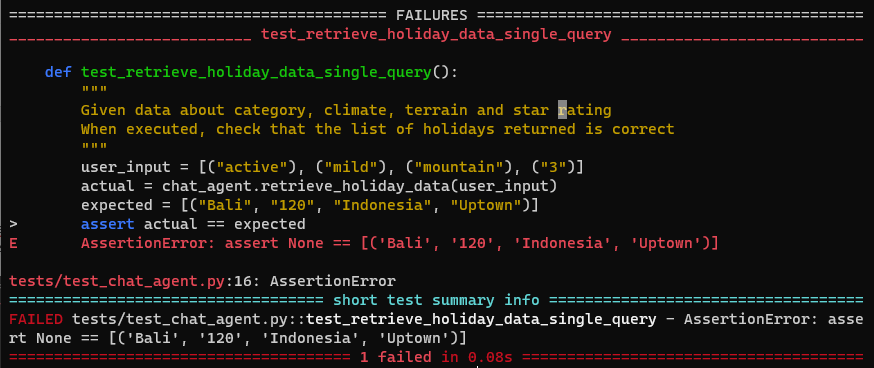
I run the test suite from my command line and the single test fails because there is no object defined called ‘chat\_agent’. Therefore, I need to create the module ‘chat\_agent’ and add a function called ‘retrieve\_holiday\_data()’:



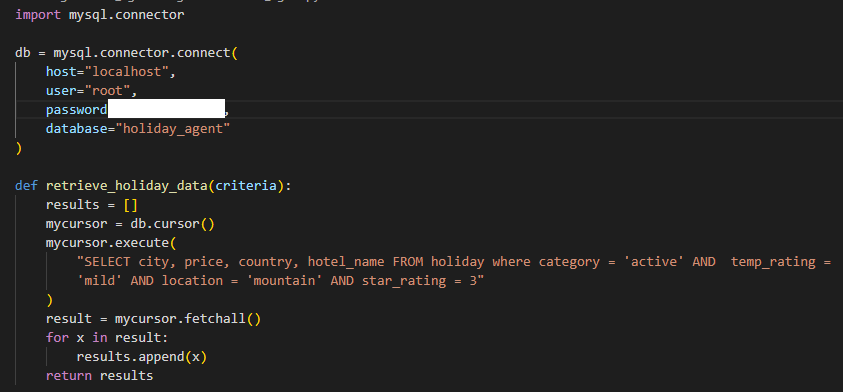
I rewrite the test so that it can retrieve the new ‘chat\_agent’ module and its single function. I also refactor the test by adding docstrings to the test function:



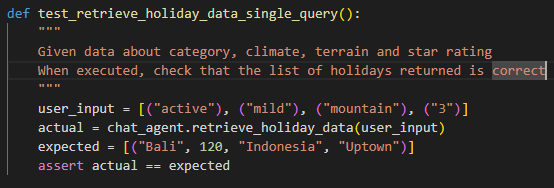
When I re-run the test suite, I get a failing test but this time I receive an AssertionError because I have not yet written any code in the new function to obtain database results:



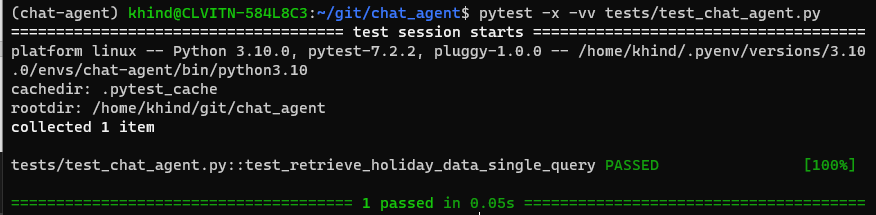
I modify the function to call the database object then use the Python mysql module to obtain results from the database as a list of tuples:



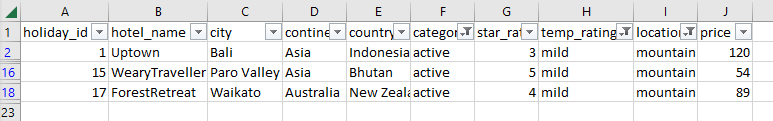
When I run the test, I get another AssertionError but this time it is because the actual output is not exactly the same as the expected output. I can see that while the expected output represents the hotel price per night as a string, the database is producing an integer. For the moment, I will change the value in my test so that the price per night in the expected value is a string. I do not want to manipulate the output from the database at this point:



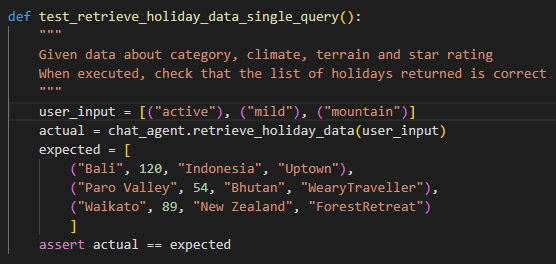
I now re-run the test suite and the test passes:



Unfortunately, the code I have written will only work where the user’s criteria match the contents of the SQL statement. If a user decides they want recommendations for 4- and 5-star hotels as well as 3-star hotels then an expanded set of recommendations will be given:



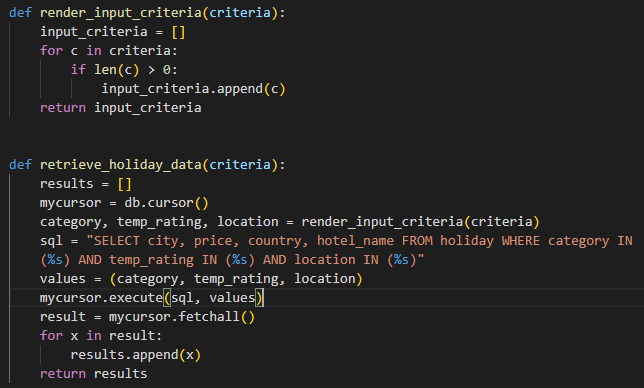
I will refactor my test to cater for the additional results:



I re-run the test and obtain an AssertionError because the expected output is no longer the same as the actual output:

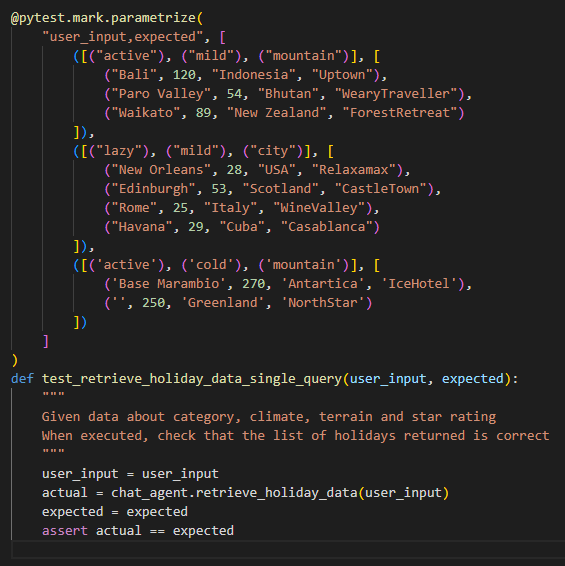


I change the retrieve\_holiday\_data() function to make the test pass. I will need to add code to manipulate the contents of the ‘criteria’ parameter the values can be passed into the SQL statement. I use a helper function called ‘render\_input\_criteria()’ to enable a list of values to be passed into the SQL query:

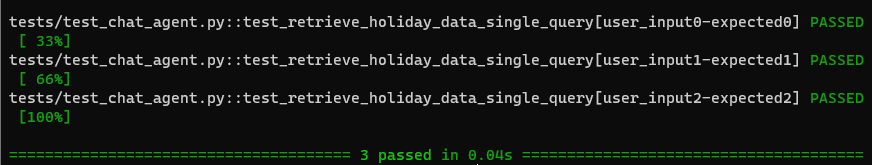


I re-run the tests and everything returns green as expected. However, this code now only works if the user has not specified a star rating. I make the design calculation that the existing filters provide enough results for the user, so the Application will no longer ask the user what rating of hotel they want. The existing filters should narrow the options sufficiently for the user not to be bombarded with data.

I now need to write further tests to ensure that different input values return the expected results from the database. To facilitate this, I use the @pytest.mark.parametrize decorator on the test to pass in more options. This will also enable another developer to add other test cases more easily:

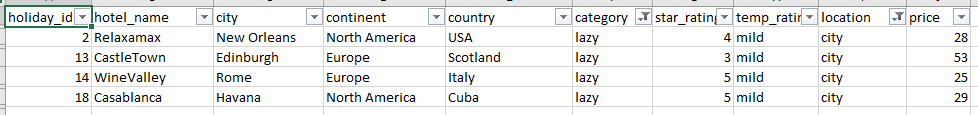


All the test cases pass:

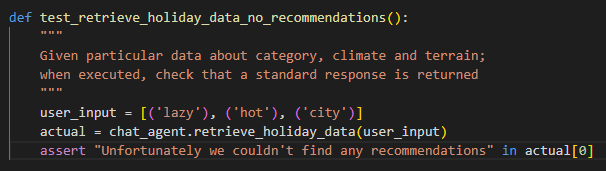


### Output when there are no recommendations

When a user selects choices that produce no recommendations, they should receive a standard response giving users the option of speaking to a travel agent in person. I will write a test asserting that a standard response will be returned if a particular set of options are given. For example, I have determined that there are no recommendations left in the data if a user selects a category of ‘lazy’, a location of ‘city’, and a temperature rating of ‘hot’ or ‘cold’:



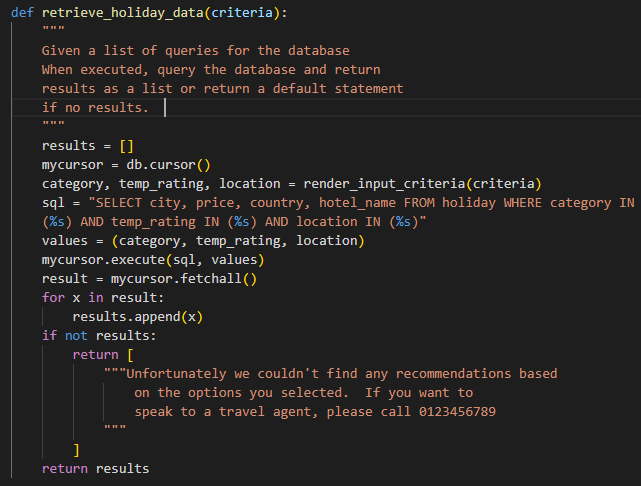
Therefore, I will use this combination of choices in my test input:



I run the tests and retrieve an AssertionError because the retrieve\_holiday\_data() function is just returning an empty list when there are no recommendations.



I change the function to return the standard response if the list of database results is empty. I add a docstring to the function describing what it does:



I re-run the test suite and there are no errors.

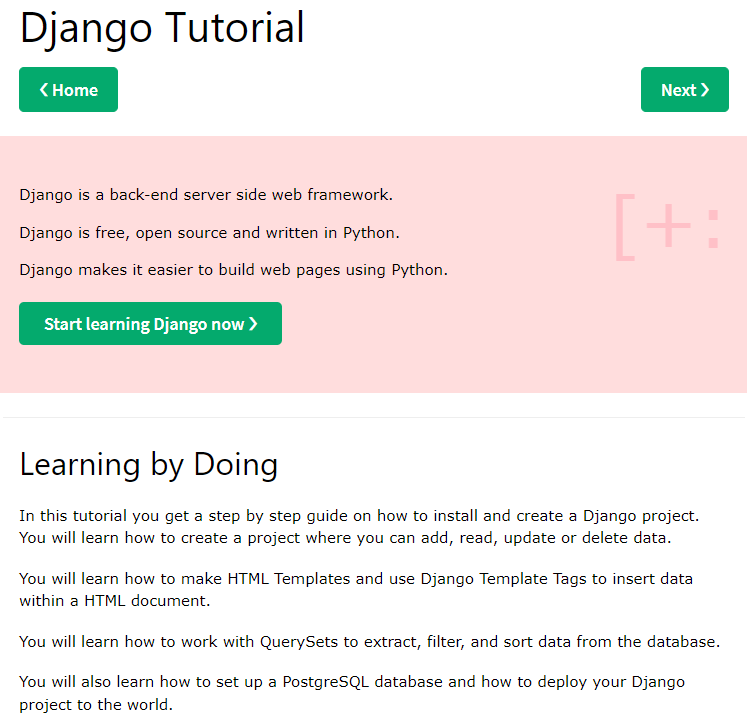
## Building the front end application

### Setting up a new Django project and application

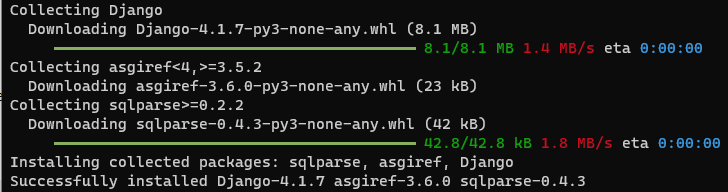
Now that I have a database storing the holiday data and a Python application that retrieves data from the database, I now need to create the front end web application. I will use the Django web framework that is a recognized way of building web applications in a Python environment.

I will continue to use a test-driven development approach to writing the code for the front-end application, however I will not use the pytest package as this does not have functionality for checking results in a browser. Therefore, I will simply implement the code incrementally, checking the results in a web browser at each stage of testing.

I will be using the W3 Schools documentation on using Django:



I begin by installing Django in my virtual environment then creating a new project within the git repository:

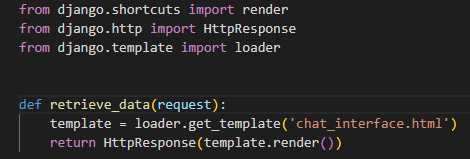




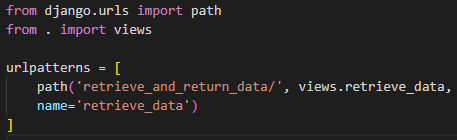
I create a new application within the Django project called ‘retrieve\_and\_return\_data’:



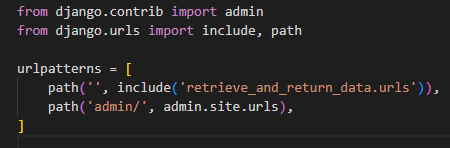
I am now ready to create a Django ‘view’ for the retrieve\_and\_return\_data application (“the Retrieve Application”). A ‘view’ takes http requests and returns http responses like HTML documents. A view must be called via a URL. I create a new view within the ‘views.py’ file which refers to a template HTML document called ‘chat\_interface.html’:



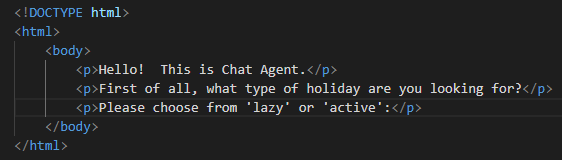
I then modify the ‘urls.py’ file within the Retrieve Application to include the view I have just created:



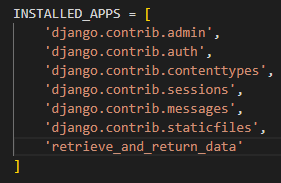
I then modify the ‘urls.py’ file at the project level to locate the URL patterns within the Retrieve Application:



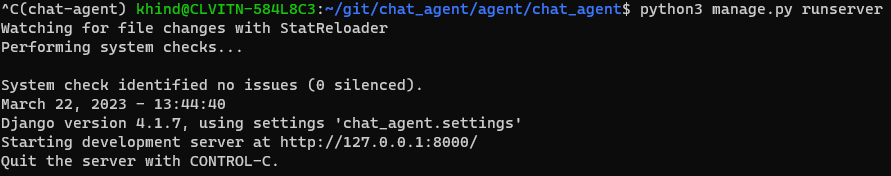
Finally, I create the HTML for the first part of the chat agent interface where the agent introduces themselves and asks the user the first question:

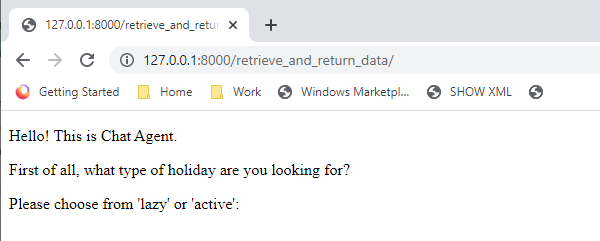


Finally, I need to update the project settings to include the Retrieve Application. This is done by updating ‘INSTALLED\_APPS’ within the ‘settings.py’ file:



I now test the Retrieve Application by running ‘manage.py’ from my command line and checking the URL within a web browser:





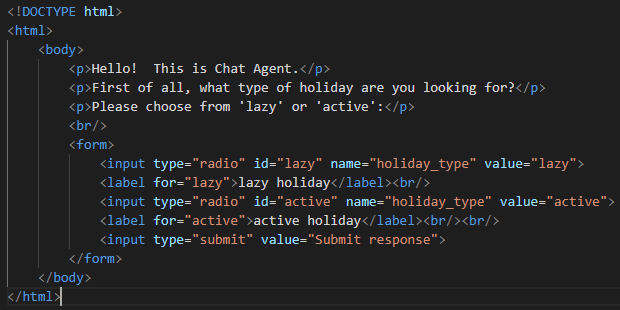
I have now completed the first part of the work to build the chat agent itself.

### Adding interactivity to retrieve data

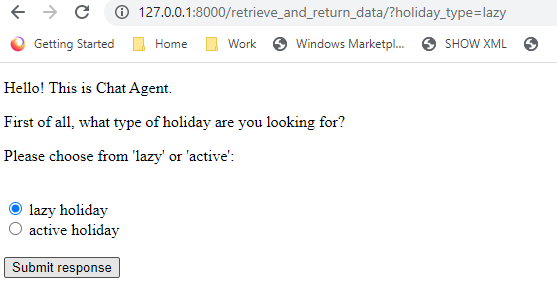
I now want to write code that will enable the user to answer the first question by entering a response into the web browser. Once the user has entered a response, I will write code to present all the options for that type of holiday. I will then check the output against the listed results before adding any more questions.

Instead of using free text input for the Application, I will implement the Application with a check box form. One purpose of using check boxes is to build greater security into the Application and avoid the possibility of injection attacks.

I start by modifying the HTML file to enable user input, at this point using a text box and button:



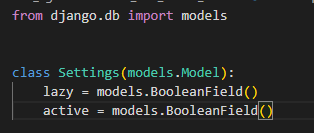
I test this code in my browser and can see it works:



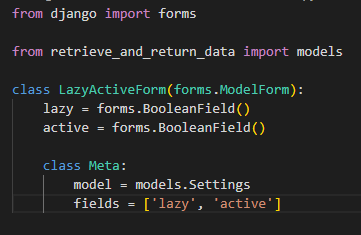
I now need to write code that will interact with the ‘chat\_agent’ module I created to handle responses.

In Django, data is created in objects called Models. Django has functionality to render forms with checkbox values like those used above. Furthermore, Django allows us to store the output from a POST request as a value which can then be used more generally in other Python processes. This will be useful for storing the user’s response and then retrieving results from the database using that response.

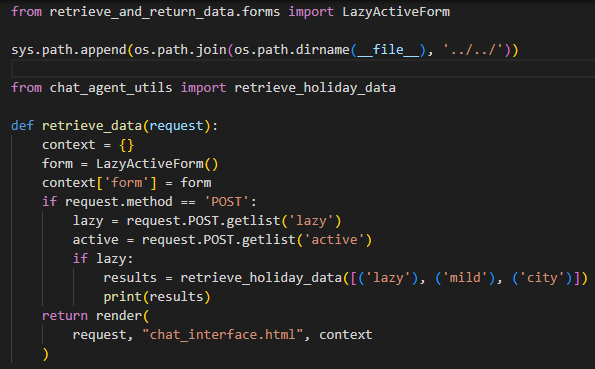
First of all I modify the content of ‘models.py’ to create settings for the ‘lazy’ and ‘active’ options based on Boolean values:



Next I create a new file ‘forms.py’ in my Retrieve Application and add a new form class using the ‘ModelForm’ class. This form will have two fields: ‘lazy’ and ‘active’. Using the BooleanField renders check box input to the user:



Next I modify the ‘retrieve\_data’ view so that it passes in the LazyActiveForm class to the request. Furthermore, if the ‘POST’ method returns a value of ‘on’ for the ‘lazy’ option then I make a call to the ‘retrieve\_holiday\_data()’ function to retrieve data from the database:

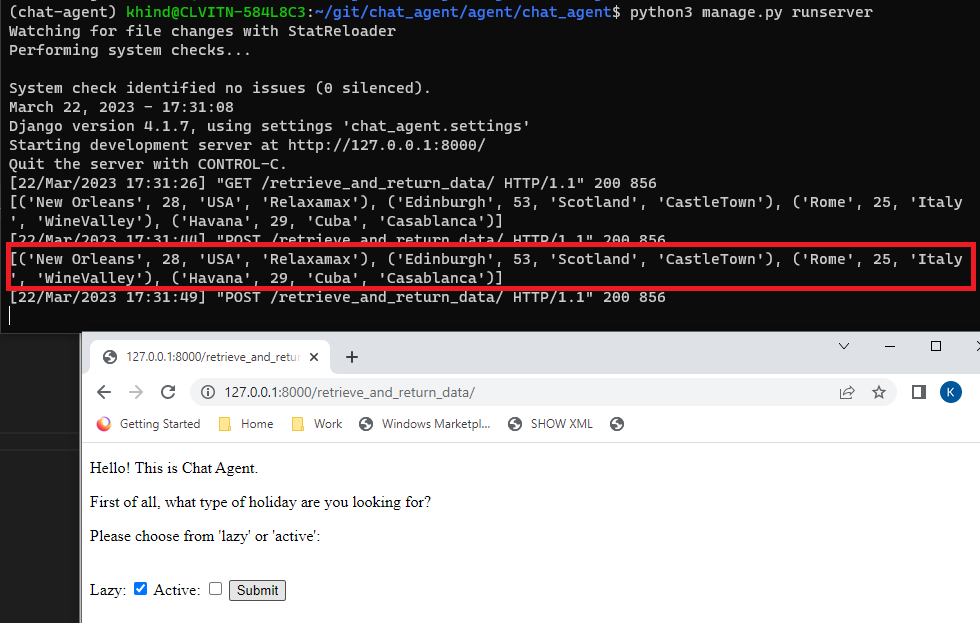


I now test the Retrieve Application by running ‘manage.py’ from my command line and checking the URL within a web browser:

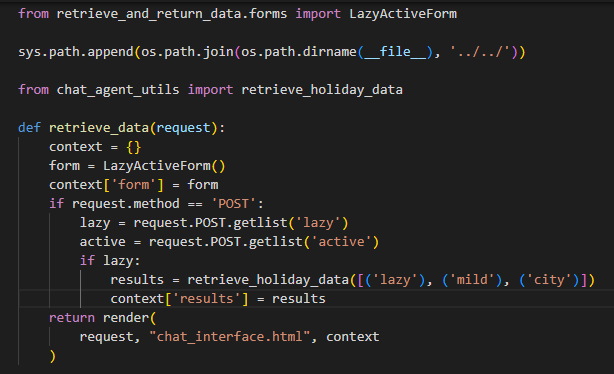
Text

Description automatically generated

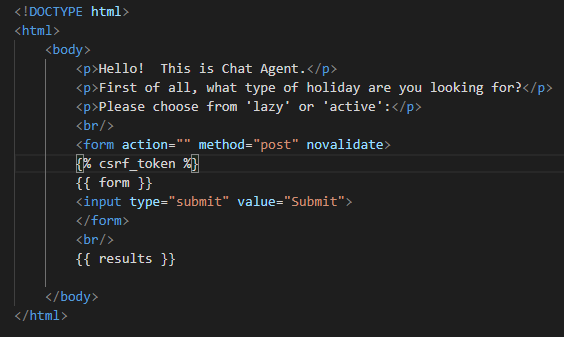
In the browser, I type the URL and then select ‘lazy’:



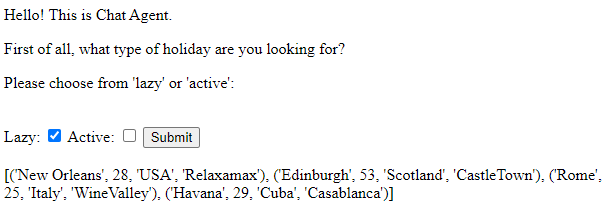
I can see that a set of recommendations from the database are being returned to the screen in my command line. Now I want to make sure those results are displayed back to the user in the browser. Firstly, I stash the database results as a new value in the context dictionary which is passed to the browser:



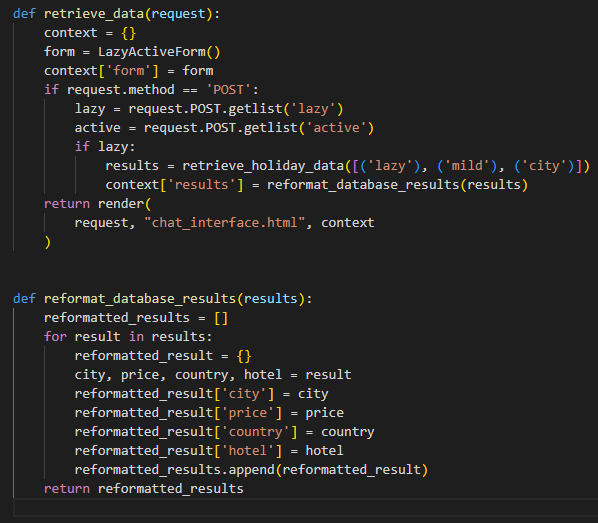
I then modify the chat\_interface template so that the value of ‘results’ will display once the form has been submitted (provided that the user selects ‘lazy’, for the purposes of testing the Retrieve Application:



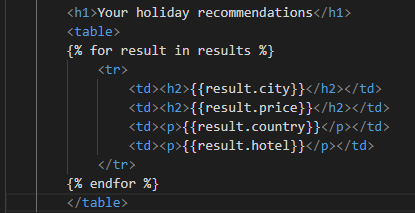
I then test the code by running the Application and I can see the correct data being returned to the browser:



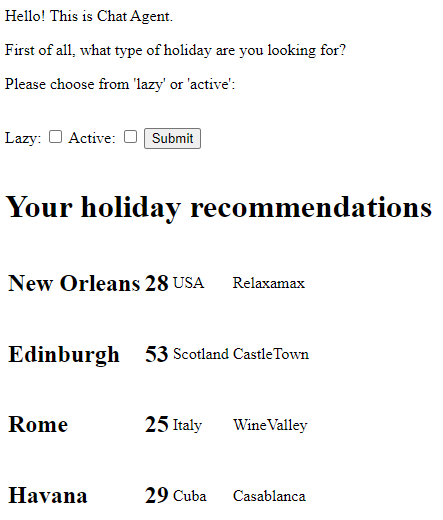
The data is poorly formatted and will not make sense to the user, so I need to write code that will display the data as sketched in the ‘Design’ section above. To get closer to the sketch, I reformat the results in my ‘retrieve\_data’ view so that a list of dictionaries is passed back to the browser in the list of results:



I now reformat the HTML in the ‘chat\_interface’ template so that each recommendation is displayed on a separate line. I have chosen to use a table format for the data at this point:



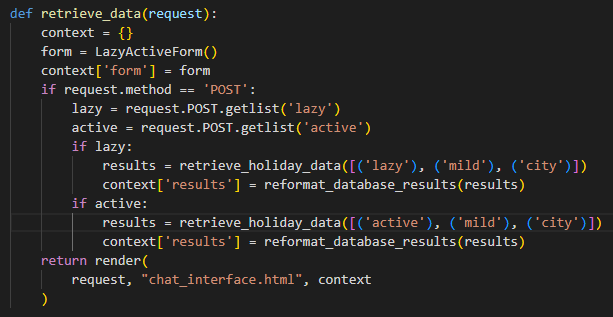
I now refresh the browser and re-run the project. The results now display as follows:



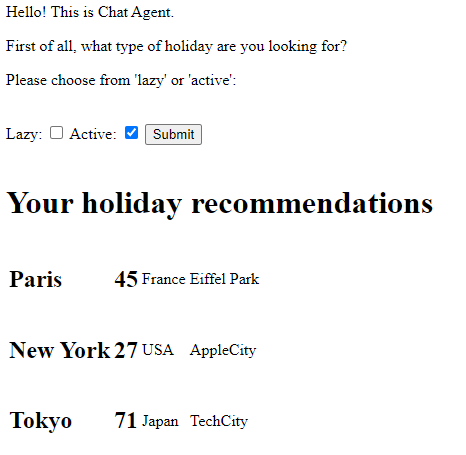
I will now write the code necessary to return results where the type of holiday is ‘active’ rather than ‘lazy’. If I keep the location and temperature rating as ‘city’ and ‘mild’ respectively, I would expect to retrieve the following results:

|  |  |  |  |
| --- | --- | --- | --- |
| Paris | 45 | France | Eiffel Park |
| New York | 27 | USA | AppleCity |
| Tokyo | 71 | Japan | TechCity |

I write code to handle the case where the user chooses ‘active’ rather than ‘lazy’:



If I now run the Application and select ‘active’, I retrieve the following recommendations, which matches what I expected:



I now have an application where the user can meet the Chat Agent, select a type of holiday and then receive some recommendations. However, the Application needs to be able to tailor the recommendations beyond the assumptions used as part of the tests above (i.e. that all users will choose a city location and a mild climate). Therefore, we now turn to creating additional questions for the user before recommendations are given. Also, we need to ensure that the standard message displays where the set of options produces no recommendations.

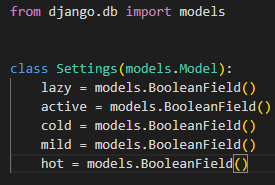
Before I do this, I will add and commit my code to the git repository before I begin further development work:

Text

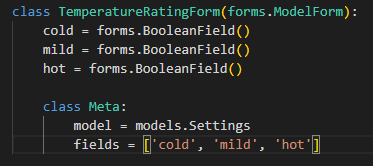
Description automatically generated

### Creating additional questions

Turning to the temperature rating, I first need to add new attributes to the Settings class in the models module so that these options can be added to a new form for the temperature rating:



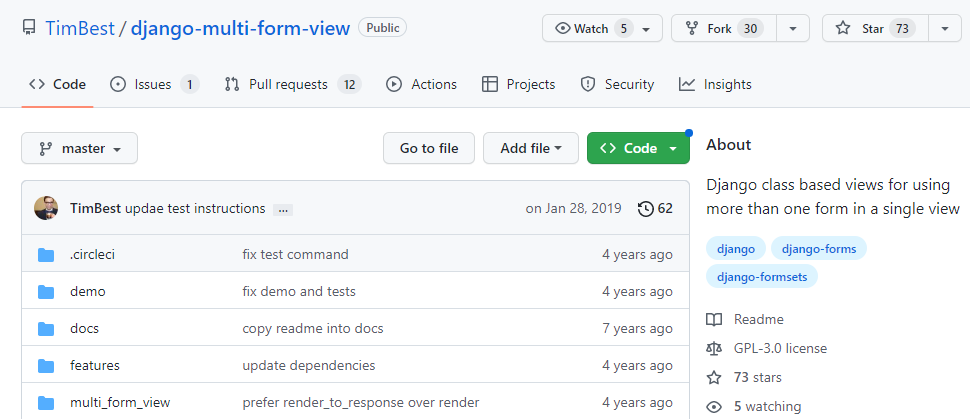
I now create a new class in the forms module that will implement the temperature rating form:



I now need to implement the new form in the ‘retrieve\_data()’ view within the views module. There is a design decision to make because the current implementation of the ‘retrieve\_data()’ view relies on only one form (the lazy/active form) and makes assumptions about the remaining data points. Depending on how I implement the code to handle the climate rating, it is possible that I might end up with multiple if/else statements in the function which would make the code difficult to maintain. Therefore, I need to work out how to ensure that the ‘context’ argument passed to the render function as part of the return statement caters for different types of form and different results sets.

I do some research into ‘multi model form views’ using the internet and I find resources explaining the role of the ‘multi\_form\_view’ package for Django as well as class-based views. Here are screenshots of the websites I visited:

On the multi\_form\_view package

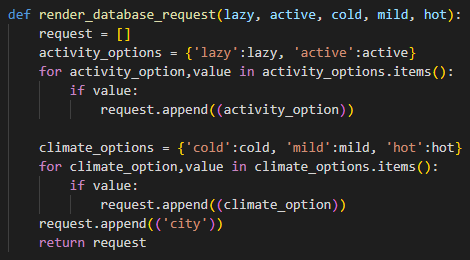


On class-based views

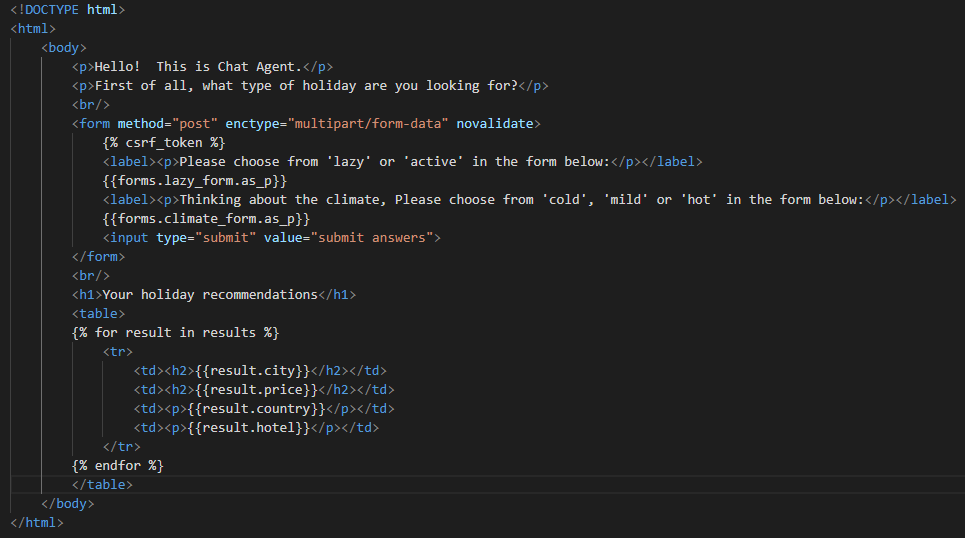


On the basis of this research, I replace my ‘retrieve\_data()’ function view in the views module with a ‘RetrieveData’ class that inherits from the MultiModelFormView class. I then create ‘get’ and ‘post’ methods to handle the form display and the user input. I create a helper function ‘render\_database\_request()’ to process the user input into a list of tuples that can be passed to the ‘retrieve\_holiday\_data()’ function. The results are then passed back to the web browser:

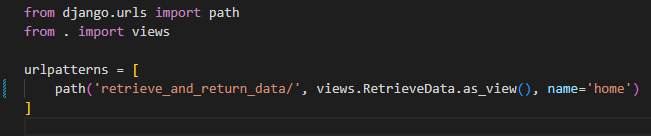




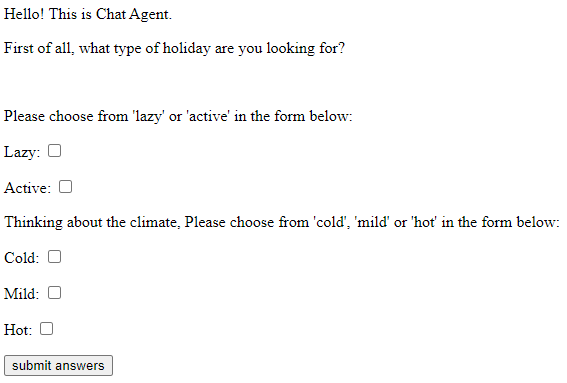
I make changes to the ‘chat\_interface’ template so that I can call multiple forms and ask multiple questions before the user submits their answers:



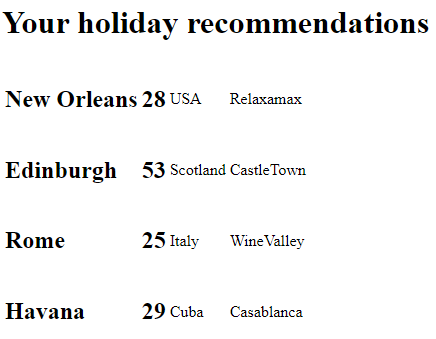
Finally, I update the ‘urls.py’ file within the Retrieve Application to take account of the fact that the functional view has been replaced with a class-based view:



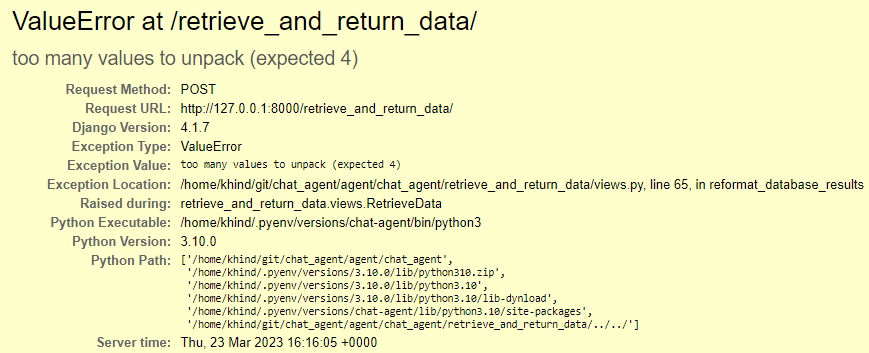
I now re-run the Application interface and see that a multi-option form is now displaying:



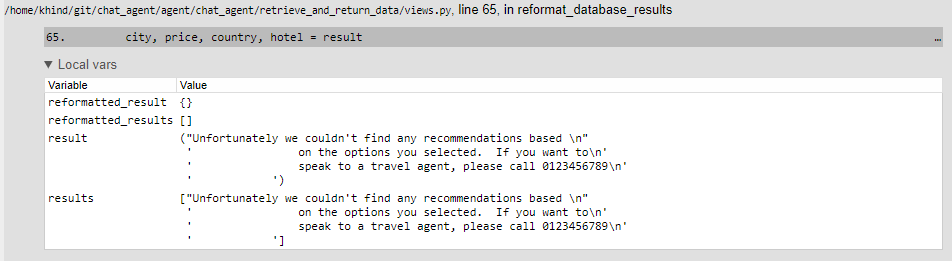
If I select ‘Lazy’ and ‘Mild’ from the form, I retrieve the following results:



To conduct further tests, I refresh the web page in my browser and select ‘Active’ and ‘Hot’. When I submit the answers, I retrieve a ValueError exception:

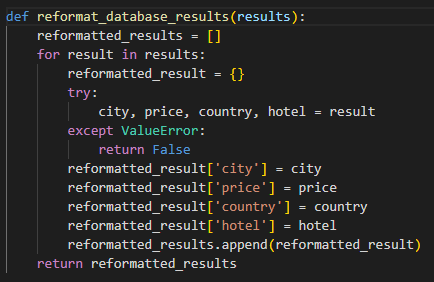


If I look at the details of the Traceback, I can see that I have selected options which produce the default response when there are no recommendations:

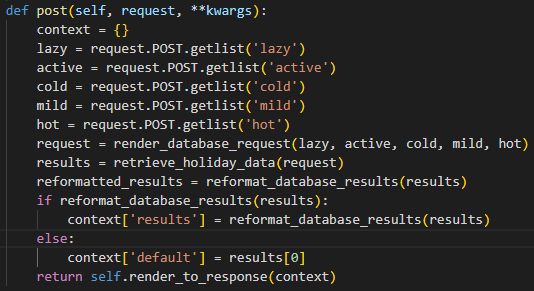


The error occurs because I have not written code to handle the case where a default response is given when there are no recommendations.

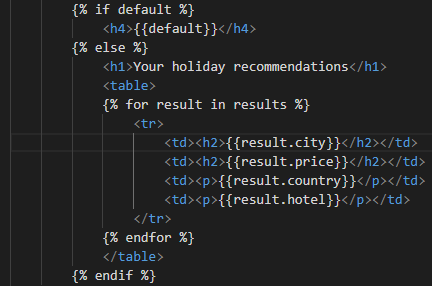
To implement the changes needed to the code, I change the ‘reformat\_database\_results()’ function so that it returns a Boolean value of ‘False’ if a ValueError is raised:



In the ‘post’ method of the RetrieveData class I specify different contexts depending on the output of the reformatting function. If there is a reformatted list, I create a ‘results’ variable in the context data. If there is no reformatted list (i.e. the reformatting function returns a value of ‘False’, I create a ‘default’ variable in the context data:



In the ‘chat\_interface’ template I use an ‘if’ statement to implement different output behaviours depending on whether there are holiday recommendations or a default message:



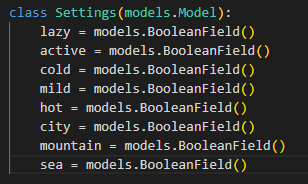
I re-run the Application and enter values into the form that I know will produce no recommendations. Here is the output:



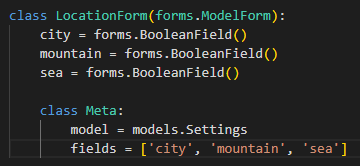
I now have an Application which will ask two questions, returning formatted results if there are holiday recommendations and a default response if there are no recommendations. I will now complete the functionality by writing code to add a ‘location’ form producing expected recommendations.

### Creating the ‘location’ question

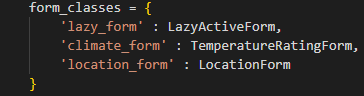
Firstly, I add new models to the Settings class in the ‘models.py’ file which account for the new option in the location form:

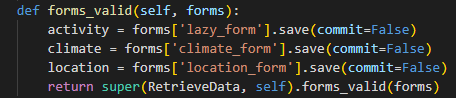


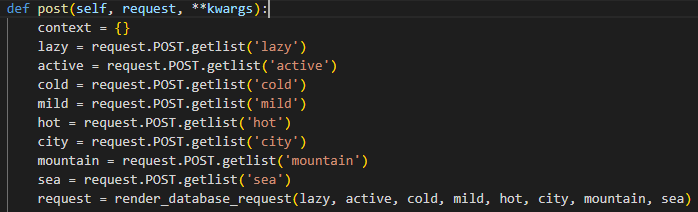
I then create a new form class in the ‘forms.py’ file using the new settings created above:



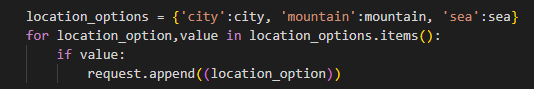
I update the ‘RetrieveData’ class to take account of the new form:



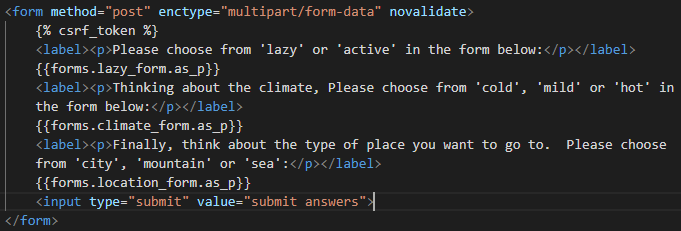




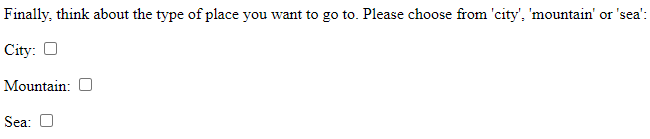
I also make a change to the ‘render\_database\_request()’ function to take account of the new options from the location form:



Finally, I add new HTML code into the ‘chat\_interface’ template to take account of the new form:



I re-run the Application and see that there is now a third option in the form:



### User acceptance testing

I then select options from all three forms to see the results. These tests will take the form of user acceptance tests. In describing the tests below, I will give the ‘category’, ‘temperature rating’ and ‘location’ values followed by the expected hotel names and cities:

#### Test 1: active, mild, sea

|  |  |
| --- | --- |
| Expected results | Actual results |
|  |  |

Verdict: Pass

#### Test 2: lazy, hot, sea

|  |  |
| --- | --- |
| Expected results | Actual results |
|  |  |

Verdict: Pass

#### Test 3: lazy, mild, sea

|  |  |
| --- | --- |
| Expected results | Actual results |
|  |  |

Verdict: Pass

#### Test 4: active, cold, city

|  |  |
| --- | --- |
| Expected results | Actual results |
| No results – default response | Default response |



Verdict: Pass

#### Test 5: active, cold, mountain

|  |  |
| --- | --- |
| Expected results | Actual results |
|  |  |

Verdict: Pass

#### Test 6: lazy, hot, sea

|  |  |
| --- | --- |
| Expected results | Actual results |
|  |  |

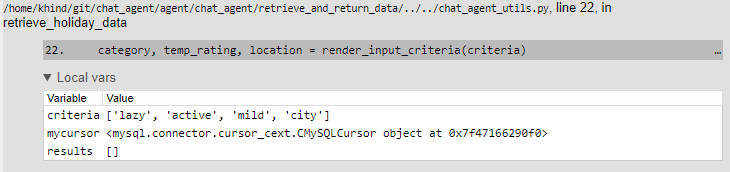
Verdict: Pass

#### Test 7: lazy and active, hot, sea

|  |  |
| --- | --- |
| Expected results | Actual results |
|  | Produces ValueError – too many values to unpack (expected 3) |

Verdict: Fail

Here is the traceback message. The Application fails when more than one value is entered for each of the categories.



#### Overall verdict

Having conducted some tests, I have demonstrated that the Application will produce results for given combinations of user inputs. However, the final test failed when more than two values were entered in one category.

I will fix this bug before completing the Application as a prototype because it should be possible for a user to do this without obtaining an error.

# Bug fixing: user selects more than one option in each category

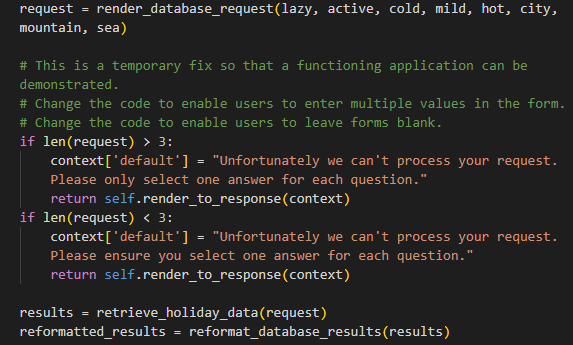
The code that executes the SQL Query is in the ‘retrieve\_holiday\_data()’ function in the ‘chat\_agent\_utils’ module. The SQL statement is written in such a way that the ‘WHERE’ statement will check for categories in a list. Therefore, it should be possible for the user to select more than one value and retrieve the correct results. However, I have a deadline for this project and we need to present a working prototype to the Company. Therefore, I will implement a temporary fix with a view to another developer creating a permanent solution.

To fix the bug, I will return to the RetrieveData class. I will create a temporary fix for this bug so that when the user selects more than one option in each category the user receives a helpful message rather than a traceback message.

In the ‘post’ method of the ‘RetrieveData’ class, I have added ‘if’ statements that will be triggered if the user:

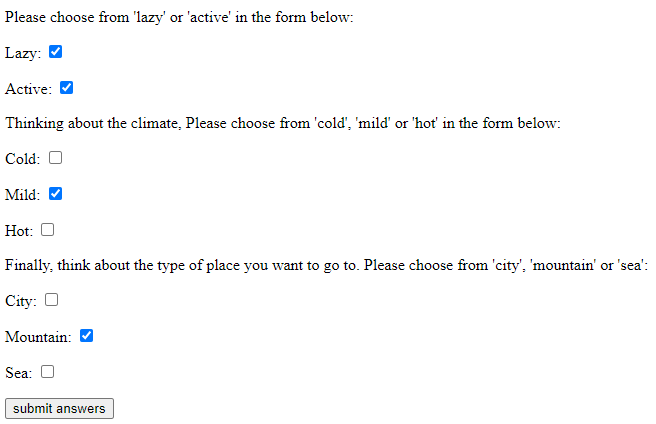
* Enters more than one value in a form, or
* Fails to enter any values in one or more of the forms.

I have added a comment to this section of the code to ensure that another programmer knows that this is a temporary fix and should not be deployed in the final version of the code.



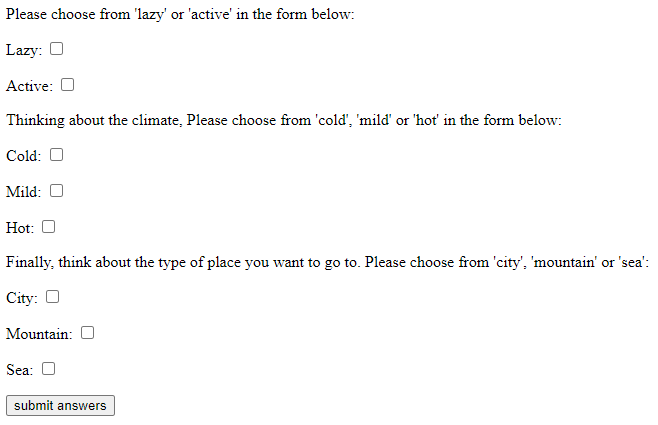
I will test this implementation by testing the use cases where a user selects more than one value or fails to make any selection.

More than one selection



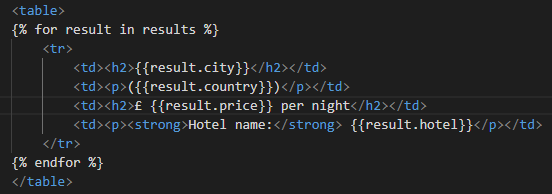


Failure to select any options

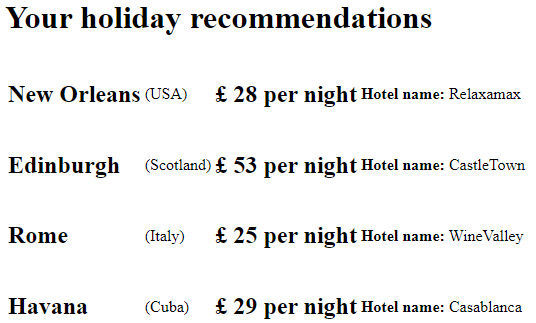




I have also added some additional HTML code to make the holiday recommendations more closely resemble my initial sketch. For example, I have added a pound sign to the price per night and I have put the words ‘Hotel name:’ before the name of the hotel in bold. I have also put the country value straight after the city name in brackets so it is clearer:



This is the new display format:



# Non-functional requirements

Now that I have a working prototype of the Application, I now turn to suggestions for how it could be improved in terms of adding non-functional requirements.

Performance

At the moment, the database only contains 21 holiday recommendations. Therefore, when a query is executed the speed of the transaction will be relatively quick. However, the more records that are added to the database, the more data will be involved in the transaction. To test the performance of the Application, many more holiday recommendations could be added to the database. It would then be possible to test the overall effect of the additions on speed. If there is a significant impact on performance, it would then be necessary to check how this could be improved by (for example) applying normalization principles to the database.

Security and Access

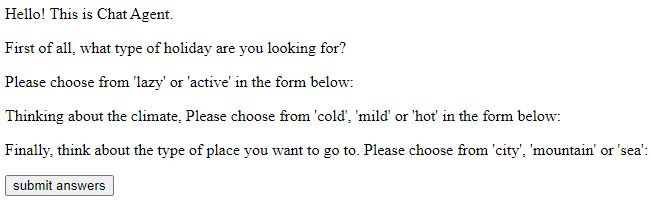
When I set up and created the database, I used a secure installation which meant that the database is secured with a password. However, the level of protection could be raised by having an encrypted password for accessing the database. However, it will be for the Company to decide who can update and add records to the database.

Robustness

I addressed some of the user error issues that could arise from using the Application, such as when the user fails to select any options and retrieves a traceback error.

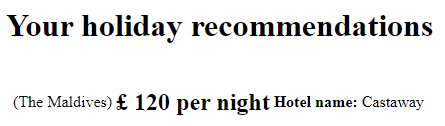
In creating the user interface, I decided to implement a ‘form’ style for the Application so that I could meet the requirement that the GUI interface should show a trail of the chat conversation for the session. However, a more conversational style could be created by ensuring that the options and checkboxes appear sequentially once the user has submitted options for each question. For example, the climate question only appears when the user answers the ‘lazy/active’ question.

Furthermore, when the user selects the checkboxes and clicks the ‘submit answers’ button, the checkboxes and options disappear:



It is possible to refresh the checkboxes by refreshing the database, but this is a bug that will clearly need to be fixed before a customer-facing version is deployed.

There are gaps in the data, for example entries 6 and 21 have no ‘city’ information. When this displays to the user, the country name is given in brackets but it looks odd. The front-end code could be changed so that if a ‘city’ name is missing, the country name appears in bold:



# Suggestions for other functional requirements

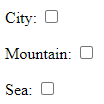
Implementing a ‘bot’ interface

The Company may decide not to implement the Application as a form but may decide to implement the Chat Agent as a ‘chat bot’. For example, my bank has a chat bot that acts as a helpdesk where the user can ask questions using free text which are then interpreted by the bot. The Company may decide that it wants a similar implementation for its Chat Agent, so that users can describe the types of holidays they want in more idiosyncratic language. For example, someone who wants a lazy holiday in a hot climate in a seaside location might say: “I’m looking for somewhere hot by the sea where I can chill out on the beach with a book”. Artificial Intelligence concepts like Natural Language Processing could be used to implement this type of Application.

User interface

Currently, the user interface consists of unstyled HTML. The UI could be improved by adding CSS styling.

Currently, the layout of the checkboxes is not in a straight line. This could be improved by using CSS to align the checkboxes and options in a more visually appealing way:



Also, the layout of the holiday recommendations still looks untidy and does not match the original sketch made as part of the design. The options could be displayed with more colour, an alternative font and an alternative arrangement. This would need to be agreed with the Company so it fitted in with any house styles or branding.

# Conclusion

I have written an Application that meets all the functional requirements defined in the Chat Agent Specification. I used Python as the primary programming language but I also demonstrated use of SQL to build the database as well as HTML to build the front-end interface. I used a Linux distribution to execute the commands necessary to build and run the Application. I used Visual Studio Code as my IDE.

I applied security principles to the creation of the database, ensuring that it was set up with username and password authentication concealed from the end user. I also avoided using free text boxes for the Application to build greater security into it and avoid the possibility of injection attacks. I ensured that my virtual environment was up-to-date before starting the development process.

I used github to build the Application, ensuring that version control was built in from the start of the development process. I wrote full documentation describing in as much detail as possible the process I used to build the Application. I demonstrated that I understood the business context by referring to the needs of the Company and anticipating how they might want to use the Application.

I wrote code using SQL commands to link user inputs to a MySQL database to retrieve holiday recommendations.

I used a test-driven development approach to build the Application using both automated tests in Python (using the pytest module) as well as manual tests to verify the user interface. In using test-driven development, I demonstrated that I could compare expected results with actual outputs. I also used this development approach to resolve problems encountered as and when they arose.

Before starting the development work, I listed all the customer requirements and described how I would implement those requirements in code. For each requirement I represented the required functionality with a use case.

In terms of quality standards, I ensured that I used relevant libraries and modules when writing my code to query the database and write the automated tests. I used a web development framework (Django) to build the Application front end.

I developed a basic user interface for the Application and gave consideration to how this could be optimized for the end user.