

# College of Engineering, Construction and Living Sciences Bachelor of Information Technology

IN608: Intermediate Application Development Concepts Level 6, Credits 15

## Practical 08 Django 2: View & Template

**Due Date:** 24/08/2020 at 5pm

In this practical, you will complete a series of tasks covering today's lecture. This practical is worth 1% of the final mark for the IN608: Intermediate Application Development Concepts course.

Before you start, in your practicals repository, create a new branch called **08-practical**.

## Task 1

Create a Django project called quiz. cd to quiz, create a virtual environment & install Django. Create an app called practical08quiz. Please ensure you configure your app in quiz/settings.py & quiz/urls.py. cd to the practical08quiz directory & create a Python file called urls.py. In urls.py, set the app\_name to practical08quiz & create a URL which maps to the index function in views.py. In the practical08quiz directory, create a directory called templates & sub-directory called practical08quiz. In templates/practical08quiz, create an HTML file called index.html.

In views.py, create a function called index. In this function, you will make a GET request to the OpenTDB API using the Requests Python module. Note: You will need to install Requests Python module in your virtual environment. Please ensure correct error checking, for example, not being able to make a GET request to the OpenTDB API. In this instance, you would raise a ConnectionError using the Requests Exceptions interface. Create a dictionary called context. This will be a dictionary of values (either the response contents from the GET request or an error message) to add to the template context.

The response contents should be in a JSON (JavaScript Object Notation) format. For example, if I make a GET request to https://opentdb.com/api.php?amount=5&type=multiple, the response contents would look like the following:



{"response\_code":0, "results":[{"category":"Geography", "type":"multiple", "difficulty":"hard", "question":"The mountainous Khyber Pass connects which of the two following countries?", "correct\_answer":"Afghanistan and Pakistan", "incorrect\_answers":["India and Nepal", "Pakistan and India", "Tajikistan and Kyrgyzstan"]},{"category":"Entertainment: Video Games", "type":"multiple", "difficulty":"medium", "question":"Mhich of the following characters were considered for inclusion in Super Smash Bros.

Melee?", "correct\_answer": "Lucas", "incorrect\_answers":[Mega Man", "Meta Knight", "Diddy Knog"]},{"category": "Science: Computers", "type": "multiple", "difficulty":"hard", "question": "What is the name of the process that sends one qubit of information using two bits of classical information?", "correct\_answer": "Quantum Teleportation", 'incorrect\_answers':["Super Dense Coding", "Quantum Entanglement", "Quantum Programming"]), ("category": "Politics", "type": "multiple", "difficulty": "medium", "question": "Which former US president used " Lett#039;s America Great Again" as his campaign slogan before Donald Trump#639;s campaign?", "correct\_answers': "incorrect\_answers': "Jimmy Carter", "Gerald Ford", "Richard Nixon"]}, "category": "Entertainment: Video Games", "type": "multiple", "difficulty": "medium", "question": "In the PAYDAY series, who is the iconic leader of the PAYDAY gang?", "correct\_answers': "Dallas", "incorrect\_answers': "Wolf", "Chains", "Noxton"]}}}

**Note:** If the page is reloaded, the response contents will be different.

Pay careful attention to the response\_code key in the response content. This is appended to each API call to help tell developers what the API is doing. Below is a description of each response\_code:

- response\_code: 0 or Success returned results successfully.
- response\_code: 1 or No results could not return results. The API does not have enough questions for your query.
- response\_code: 2 or Invalid parameter contains an invalid parameter. Parameters passed in are not valid, i.e, https://opentdb.com/api.php?amount=five
- response\_code: 3 or Token not found session token does not exist.
- response\_code: 4 or Token empty session token has returned all possible questions for the specified query. Resetting the token is necessary.

Please ensure you check for all response codes. If response\_code is 1-4, add an error message to the context dictionary.

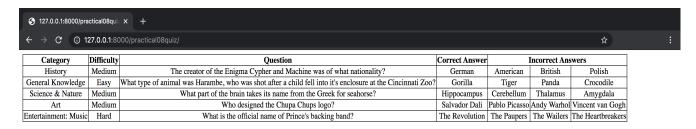
In index.html, display context in a nicely formatted HTML table. Use the capfirst filter to capitalise the first letter of each difficulty value. You may notice that some questions & answers contain character entities, i.e., ". Use the safe filter to mark a value as not requiring further HTML escaping before outputting. For example, the question value Who is the founder of " The Lego Group"? would be marked as not requiring further HTML escaping. Instead, the output would be Who is the founder of "The Lego Group"? & not contain quot; or other character entities.

Next week, we will look at how to serve static files, i.e., CSS, JavaScript, images, etc. Until then, internal styling will be accepted.

### **Expected Output**

Run the command python manage.py runserver then navigate to http://127.0.0.1:8000/practical08quiz/

**Note:** The incorrect answers column span is 3.





There was a problem connecting to the OpenTDB API.

Deployment link: https://int-app-dev-practical-08.herokuapp.com/practical08quiz/

#### Resources

- OpenTDB API
- Requests
- Requests JSON Response Content
- Requests Exceptions
- Django Built-In Filters

## Task 2

Create a Django project called dog. cd to dog, create a virtual environment & install Django. Create an app called practical08dog. Alternatively, you can create an app in quiz. Though, it requires additional configuration. Please ensure you configure your app in dog/settings.py & dog/urls.py. cd to the practical08dog directory & create a Python file called urls.py. In the practical08dog directory, create a directory called templates & sub-directory called practical08dog. In templates/practical08dog, create a two HTML files called index.html & details.html.

In models.py, create a class called Dog which extends models.Model. In Dog, declare the following:

```
RANGE_CHOICE = [('L', 'Low'), ('M', 'Medium'), ('H', 'High')]
```

Above is a list containing tuples used as choices for a field. The first element in each tuple is the actual value to be set on the model & the second element is the human-readable name. If choices are given to a field, they are enforced by model validation. The default form widget will be a select drop down containing choices as drop down options.

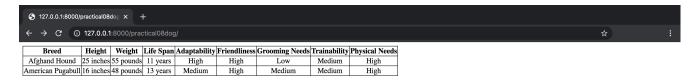
Below RANGE\_CHOICE, declare the following field names with their types & options:

```
breed = models.CharField(max_length=200, unique=True)
height = models.IntegerField(default=1)
weight = models.IntegerField(default=1)
life_span = models.IntegerField(default=1)
adaptability = models.CharField(choices=RANGE_CHOICE, max_length=200)
friendliness = models.CharField(choices=RANGE_CHOICE, max_length=200)
grooming_needs = models.CharField(choices=RANGE_CHOICE, max_length=200)
trainability = models.CharField(choices=RANGE_CHOICE, max_length=200)
physical_needs = models.CharField(choices=RANGE_CHOICE, max_length=200)
```

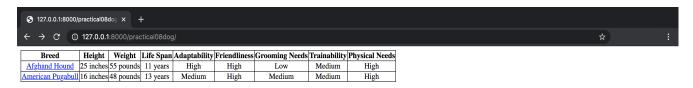
For Dog, create a \_\_str\_ method which returns breed.

In views.py, create two functions called index & details. In the index function, render the index.html template with a context dictionary containing all Dog objects in the database. In index.html, display context

in a nicely formatted HTML table. For height, weight & life\_span, use the pluralize filter which returns a plural suffix if a value is not 1, '1' or an object of length 1. By default, this suffix is 's'. height will require an alternative suffix, i.e., 'es'.



In index.html, change each breed value to a link so when clicked, gets the Dog object by its id & displays its details.



In the details function, render the details.html template with a context dictionary containing the Dog object with the primary key of, for example, 1 from Dog. Again, in details.html, display context in a nicely formatted HTML table & use the pluralize filter for height, weight & life\_span.



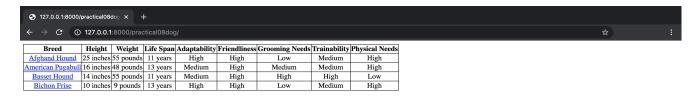
In urls.py, set the app\_name to practical08dog & create two URLs which map to the index & details functions in views.py.

#### **Fixtures**

In Django, you can pre-populate your database using migrations or fixtures. If you want to automatically load initial data, create a migration by running the command python manage.py migrate. Fixtures work slightly different as data is not automatically loaded like migrations. A fixture is a collection of data that Django knows how to import into a database. Fixtures can be written as JSON, XML (Extensible Markup Language) or YAML (Yet Another Markup Language). Feel free to use any of the three formats. To start using fixtures, create a directory called fixtures. A JSON file called dogs.json has been provided for you in the O8-django-2-view-template directory. Copy & paste dogs.json into the fixtures directory. In fixtures/dogs.json, there are two Dog objects. Create two more Dog objects then run the command python manage.py loaddata dogs.json. You should see the following message in the terminal: Installed 4 object(s) from 1 fixture(s).

## **Expected Output**

Run the command python manage.py runserver then navigate to http://127.0.0.1:8000/practical08dog/





No dogs available.

Deployment link: https://int-app-dev-practical-08.herokuapp.com/practical08dog/

## Resources

- Django Model Reference
- Django Choices
- Django Fixtures