COSC2820/2815 Advanced Programming for Data Science
Week 11 Exercise

In this exercise, you will continue to develop a web application using Python Flask. This week, you will take the code from the Exercise 1 of Week 10 and continue to develop from there.

To get started, please make sure you have downloaded the *COSC2820_Week10_exercise_simple.zip* file from Week 10 exercise to your local computer. This zip file is under Week 10 Module/Practical Material/Exercise 1 Solution. When extracting this zip file, you will find:

- the folder *templates* which consists of all the templates that we developed in Week 10.
- the folder *static* which consists of the *jquery-3.6.0.js* Javascript file and the *style.css* file.

Besides, you can also download the *COSC2820_Week11_exercise_simple.zip* file that provides a full web application for this exercise. Note that although we provide you with this zip file, you should still follow all the tasks below, do one by one so that you can understand the web application process. Your goal is to make sure you understand the Flask web application process, understand the meaning behind each step, understand what each function does, so that in the future, you can develop a similar web application. The zip file is just for you to take some special functions (e.g. Javascrip scripts, classification functions) that can help you to develop the web application.

If you have trouble with understanding any part of the exercise, please ask your tutor, your fellow classmates, or post the questions on the Canvas discussion forum.

Also, note that in this exercise, apart from the flask package, you will need to install other packages like gensim, BeautifulSoup, pandas in your new virtual environment (e.g. projectFlask) so that you can run some necessary functions from these packages in this virtual environment.

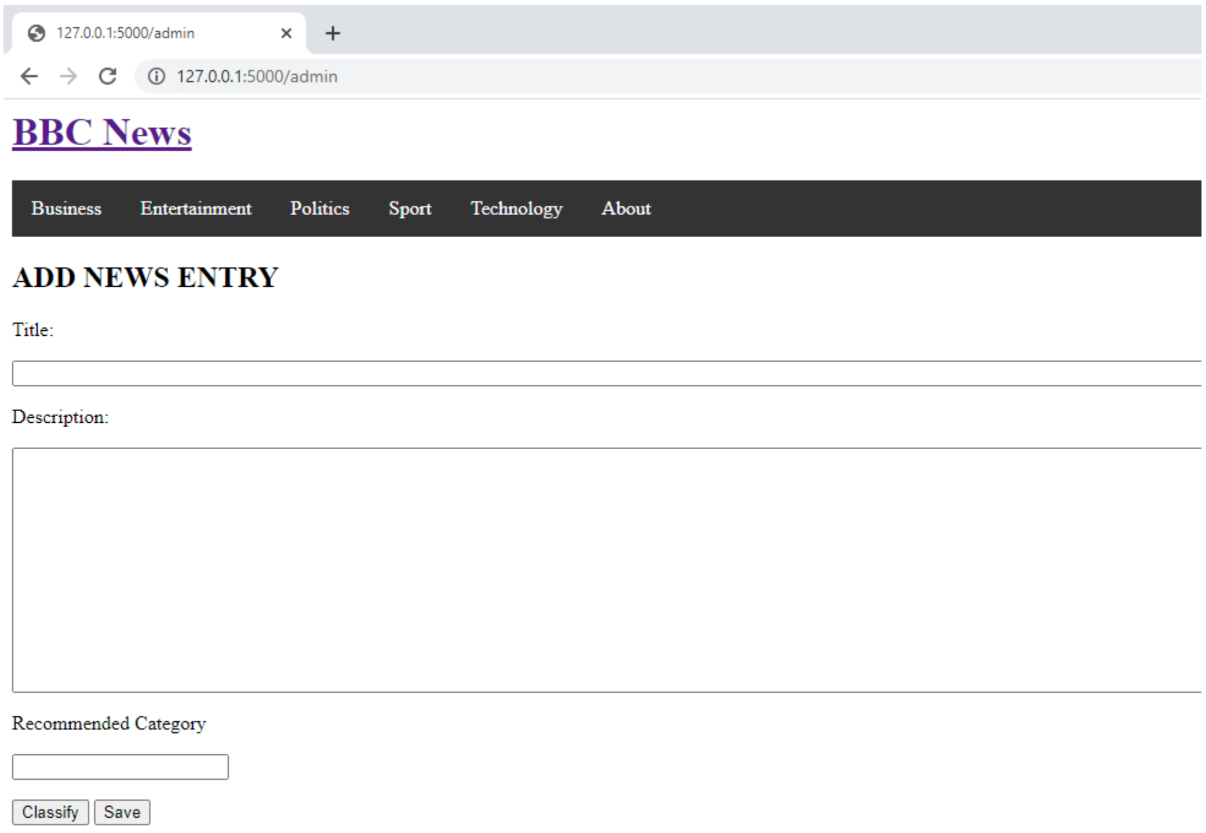
Task 1. Add a new route /admin to classify news articles

In this task, you will add a new route /admin to allow special users (admins) to perform some special tasks, e.g. add a new news article to the web application, perform classification to identify the category of the new news article. You are asked to perform the following steps to create this route:

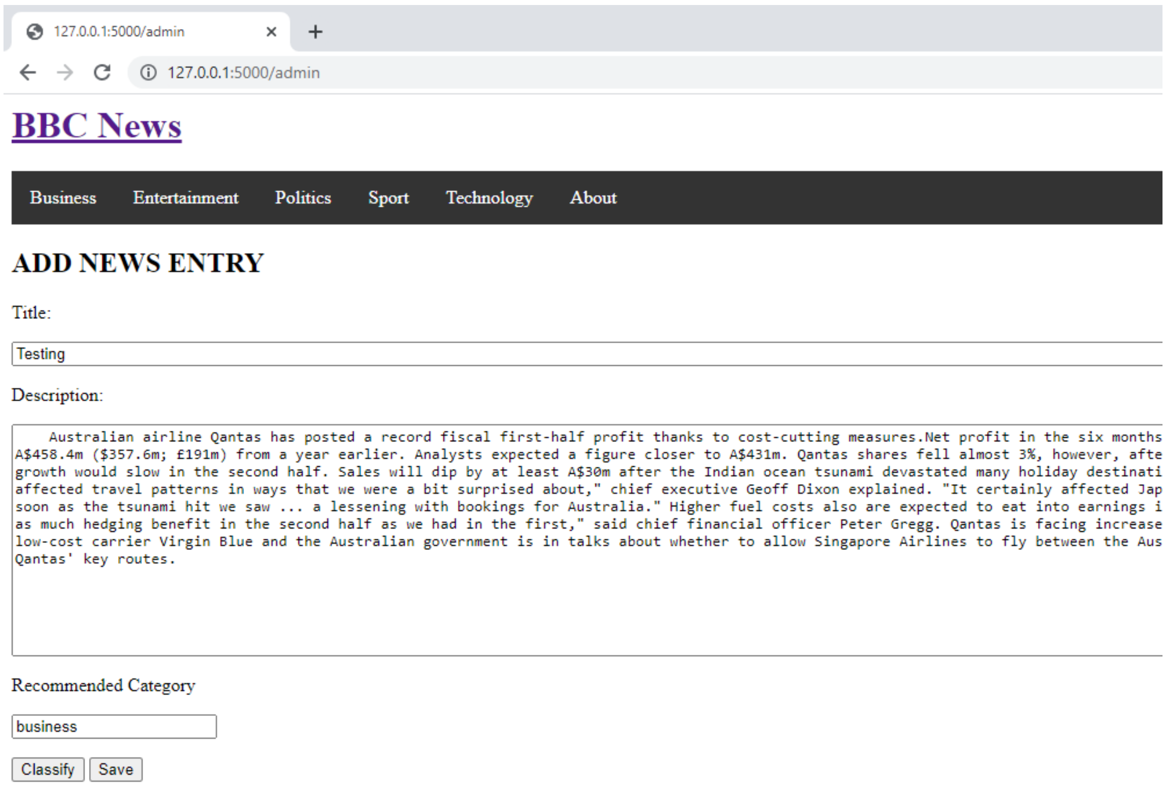
1. Create a new html file named *admin.html* which serves as the template for the route */admin*. Inside this file, you need to:
    a. create a web form with 3 fields (Title, Description, Recommended Category) and two submit buttons (Classify and Save). Action of the web form will be to redirect to the route */admin*. And you need to set the method of this form to be POST.
    b. add Jinja variables inside the fields Title, Description, Recommended Category such that later when this template (*admin.html*) is loaded via a POST request, the title, description, predicted category will be appeared in the corresponding fields of the template.
    c. Hint: you can follow the material in Section 1 Module 11.3 to design the form inside the *admin.html* file but note in this exercise you will need to have 2 buttons to classify and save the news article. Also, you can look at the below screenshots to understand the expected behaviour of this route.
2. Create a route */admin* in the *app.py* file. You need to perform the following task:

a. import the request function into the file *app.py* (from flask import request)
b. implement your route/admin such that:
    i. it supports both GET and POST requests;
    ii. if the request.method == 'POST' and the button Classify is clicked, then the view function will perform the classification, i.e. read the content from the form including both title and description, tokenize, load a language model and the logistic regression model to perform classification on the description. Finally, the view function will return the template *admin.html* that shows the predicted classification. Note, you can take the language model and the logistic regression model that we provide you in the *COSC2820_Week11_exercise_simple.zip* file or you can use other models from the previous activities or your own models on the BBC News dataset.
    iii. if the request.method is not 'POST', then the view function will simply return the blank *admin.html* file.
c. Hint: you can follow the material in Section 2 - Module 11.3 to implement this route.

At the end of this task, when loading the route /admin, you should see something like this.



Then, after entering some information in the Title and Description fields, and click the Classify button, you will then see the predicted category appears inside the field Recommended Category as follows.



127.0.0.1:5000/admin

BBC News

Business Entertainment Politics Sport Technology About

ADD NEWS ENTRY

Title:

Testing

Description:

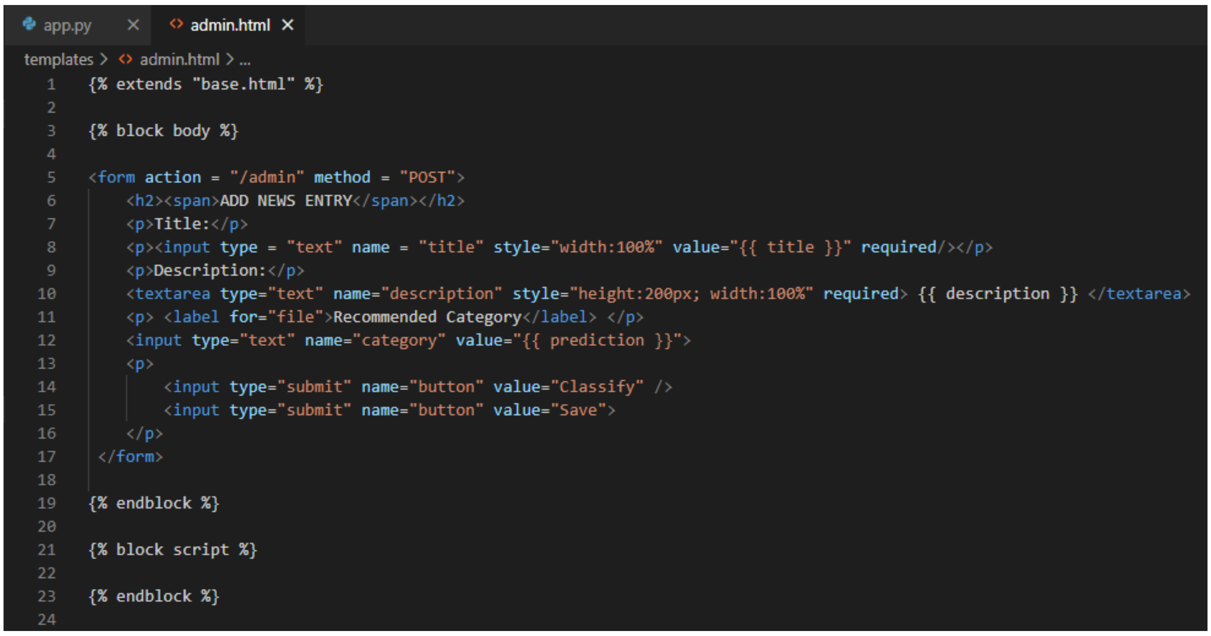
```
     Australian airline Qantas has posted a record fiscal first-half profit thanks to cost-cutting measures.Net profit in the six months
A$458.4m ($357.6m; £191m) from a year earlier. Analysts expected a figure closer to A$431m. Qantas shares fell almost 3%, however, afte
growth would slow in the second half. Sales will dip by at least A$30m after the Indian ocean tsunami devastated many holiday destinati
affected travel patterns in ways that we were a bit surprised about," chief executive Geoff Dixon explained. "It certainly affected Jap
soon as the tsunami hit we saw ... a lessening with bookings for Australia." Higher fuel costs also are expected to eat into earnings i
as much hedging benefit in the second half as we had in the first," said chief financial officer Peter Gregg. Qantas is facing increase
low-cost carrier Virgin Blue and the Australian government is in talks about whether to allow Singapore Airlines to fly between the Aus
Qantas' key routes.
```

Recommended Category

business

Classify Save

Answers of Task 1: This is how the *admin.html* file looks like. You need to make sure you set the action for the form to be '/admin' so that you perform any action within this form (click the Classify or Save button), it will be routed to the route /admin. Also, you need to set the method of the form to be POST so that you can send the data from the form to the webserver. Finally, another thing you need to pay attention is that you can specify the name of the submit input (Lines 14 & 15) to be button, so that later, you can call request.form['button'] to refer to these fields. The same with other fields like Title or Description, you can also name them, and then later inside the app.py file, you can simply call request.form['title'] or request.form['description'] to take the data in these fields.



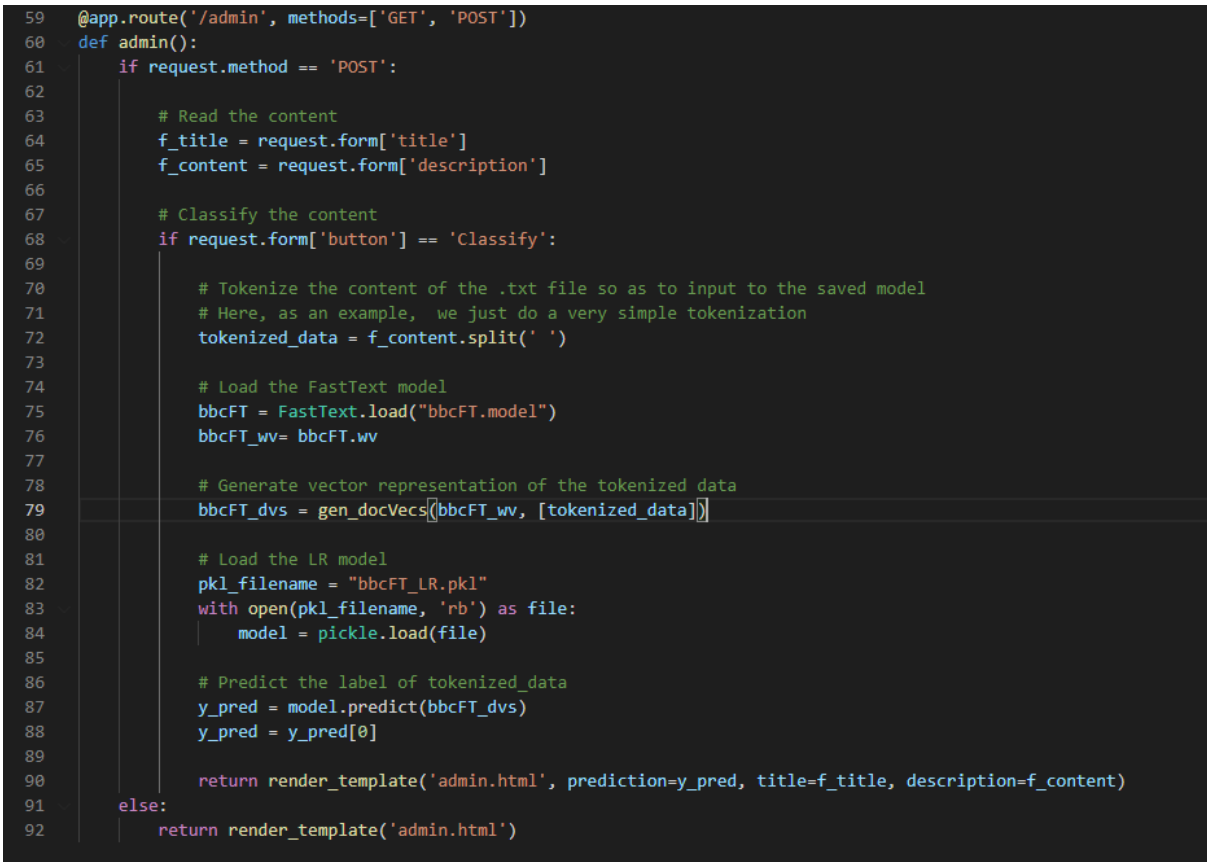
app.py admin.html

templates > admin.html > ...

```
1   {% extends "base.html" %}
2
3   {% block body %}
4
5   <form action = "/admin" method = "POST">
6       <h2><span>ADD NEWS ENTRY</span></h2>
7       <p>Title:</p>
8       <p><input type = "text" name = "title" style="width:100%" value="{{ title }}" required/></p>
9       <p>Description:</p>
10      <textarea type="text" name="description" style="height:200px; width:100%" required> {{ description }} </textarea>
11      <p> <label for="file">Recommended Category</label> </p>
12      <input type="text" name="category" value="{{ prediction }}">
13      <p>
14          <input type="submit" name="button" value="Classify" />
15          <input type="submit" name="button" value="Save">
16      </p>
17   </form>
18
19  {% endblock %}
20
21  {% block script %}
22
23  {% endblock %}
24
```

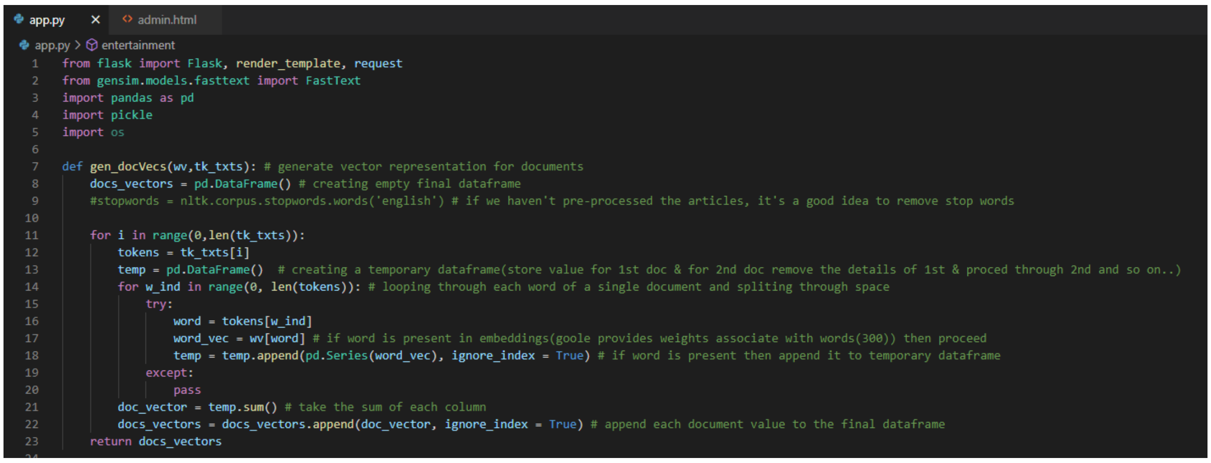
And this is how the file *app.py* looks like for the route /admin. Again, note that you can, and you should perform a more sophisticated tokenize process and use better language model (we leave this to you to enhance the accuracy of the classification process). In this example, we only demonstrate the simplest tokenize process and use a simple language model and logistic regression model.

You can see that this route accepts both GET and POST methods. When the method is POST, we will perform the classification, otherwise, we will return the blank admin.html file. This implies that only when we click the Classify button, the view function will perform classification; otherwise, it does nothing. And finally, by looking at the code, you can see that we can extract data from the fields in the file *admin.html* by using the request object (Lines 64, 65, 68).



```
59  @app.route('/admin', methods=['GET', 'POST'])
60  def admin():
61      if request.method == 'POST':
62  
63          # Read the content
64          f_title = request.form['title']
65          f_content = request.form['description']
66  
67          # Classify the content
68          if request.form['button'] == 'Classify':
69  
70              # Tokenize the content of the .txt file so as to input to the saved model
71              # Here, as an example,  we just do a very simple tokenization
72              tokenized_data = f_content.split(' ')
73  
74              # Load the FastText model
75              bbcFT = FastText.load("bbcFT.model")
76              bbcFT_wv= bbcFT.wv
77  
78              # Generate vector representation of the tokenized data
79              bbcFT_dvs = gen_docVecs(bbcFT_wv, [tokenized_data])
80  
81              # Load the LR model
82              pkl_filename = "bbcFT_LR.pkl"
83              with open(pkl_filename, 'rb') as file:
84                  model = pickle.load(file)
85  
86              # Predict the label of tokenized_data
87              y_pred = model.predict(bbcFT_dvs)
88              y_pred = y_pred[0]
89  
90              return render_template('admin.html', prediction=y_pred, title=f_title, description=f_content)
91      else:
92          return render_template('admin.html')
```

And this is how the file *app.py* looks like at the first several lines. Basically, we need to import/define necessary packages and functions to be used later. Here, we use the gen_docVecs function developed in the previous activities to generate document vector for the description.



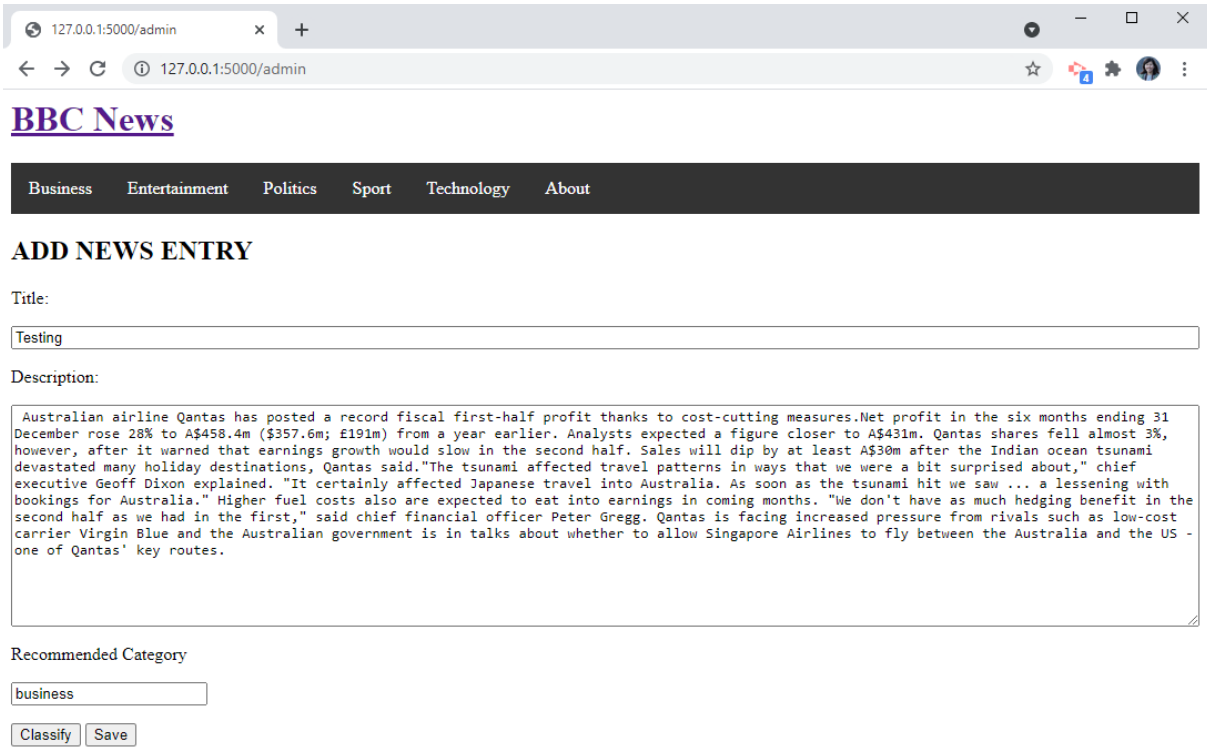
```
from flask import Flask, render_template, request
from gensim.models.fasttext import FastText
import pandas as pd
import pickle
import os

def gen_docVecs(wv,tk_txts): # generate vector representation for documents
    docs_vectors = pd.DataFrame() # creating empty final dataframe
    #stopwords = nltk.corpus.stopwords.words('english') # if we haven't pre-processed the articles, it's a good idea to remove stop words

    for i in range(0,len(tk_txts)):
        tokens = tk_txts[i]
        temp = pd.DataFrame()  # creating a temporary dataframe(store value for 1st doc & for 2nd doc remove the details of 1st & proced through 2nd and so on..)
        for w_ind in range(0, len(tokens)): # looping through each word of a single document and spliting through space
            try:
                word = tokens[w_ind]
                word_vec = wv[word] # if word is present in embeddings(goole provides weights associate with words(300)) then proceed
                temp = temp.append(pd.Series(word_vec), ignore_index = True) # if word is present then append it to temporary dataframe
            except:
                pass
        doc_vector = temp.sum() # take the sum of each column
        docs_vectors = docs_vectors.append(doc_vector, ignore_index = True) # append each document value to the final dataframe
    return docs_vectors
```

Task 2. Add the save functionality into the route /admin

In this task, you are asked to modify the view function for the route /admin such that when users click on the Save button, we will save the content in the fields Title and Description, along with the Category, into a html template of the web app. After saving, the web app will automatically redirect to the route of the new article. To do this task, you need to perform the following action:

- Create an *article_template.html* file and put inside the folder *templates*. The purpose of this template is that later you can use BeautifulSoup to copy the template and insert the title and description from the webform into the template. For this step, you can take the *article_template.html* file in the *COSC2820_Week11_exercise_simple.zip* file we provide.
- Modify the view function of the route /admin such that when the Save button is clicked, it is then read the article_template.html and then append the content from the fields Title and Description of the webform and save to the folder corresponding the category of the article. Finally, redirect to the route of the newly generated news article. For this step, you can use the code we provide to you in the *COSC2820_Week11_exercise_simple.zip* file, but make sure you understand what each statement does.
- In the file *admin.html*, add a Jinja variable after all the three above fields such that later when user clicks on the Save button, the page will display a message indicating errors when the specified category does not belong to entertainment, business, politics, sports, technology.

At the end of this task, when you entered the following content inside the route /admin



BBC News

Business Entertainment Politics Sport Technology About

ADD NEWS ENTRY

Title:

Testing

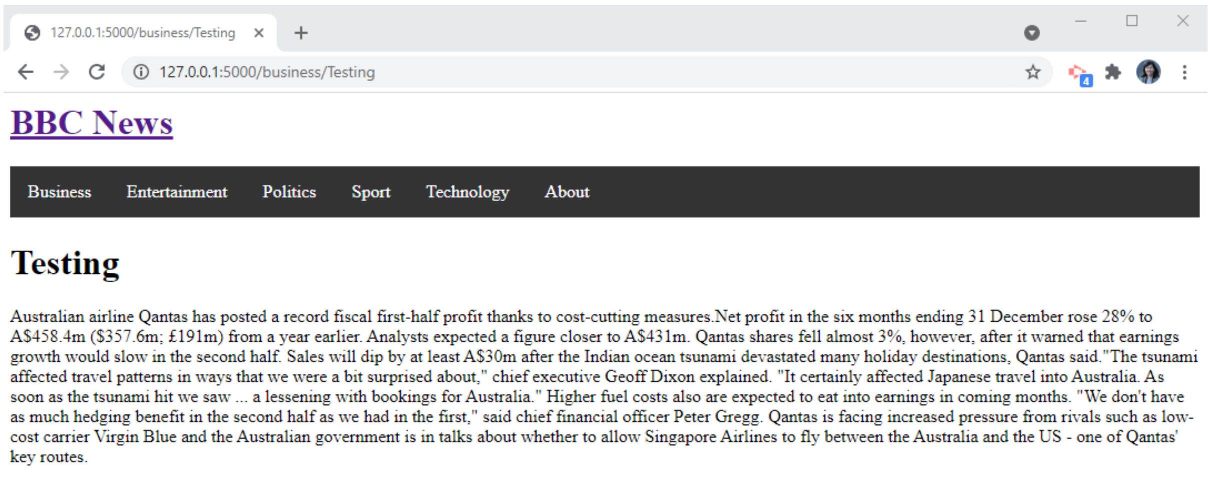
Description:

```
 Australian airline Qantas has posted a record fiscal first-half profit thanks to cost-cutting measures.Net profit in the six months ending 31
December rose 28% to A$458.4m ($357.6m; £191m) from a year earlier. Analysts expected a figure closer to A$431m. Qantas shares fell almost 3%,
however, after it warned that earnings growth would slow in the second half. Sales will dip by at least A$30m after the Indian ocean tsunami
devastated many holiday destinations, Qantas said."The tsunami affected travel patterns in ways that we were a bit surprised about," chief
executive Geoff Dixon explained. "It certainly affected Japanese travel into Australia. As soon as the tsunami hit we saw ... a lessening with
bookings for Australia." Higher fuel costs also are expected to eat into earnings in coming months. "We don't have as much hedging benefit in the
second half as we had in the first," said chief financial officer Peter Gregg. Qantas is facing increased pressure from rivals such as low-cost
carrier Virgin Blue and the Australian government is in talks about whether to allow Singapore Airlines to fly between the Australia and the US -
one of Qantas' key routes.
```

Recommended Category

business

And then when clicking the Save button, the webpage will be redirected to this route:



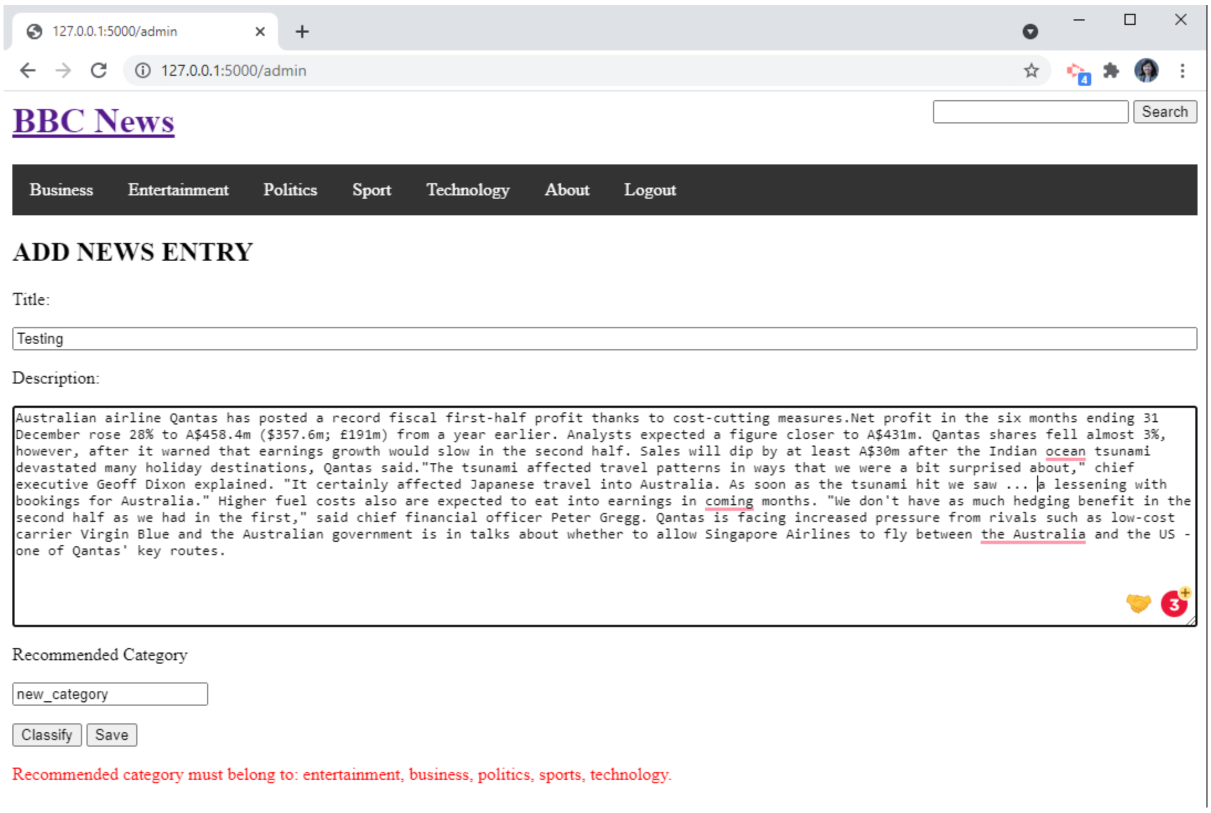
BBC News

Business Entertainment Politics Sport Technology About

Testing

Australian airline Qantas has posted a record fiscal first-half profit thanks to cost-cutting measures.Net profit in the six months ending 31 December rose 28% to A$458.4m ($357.6m; £191m) from a year earlier. Analysts expected a figure closer to A$431m. Qantas shares fell almost 3%, however, after it warned that earnings growth would slow in the second half. Sales will dip by at least A$30m after the Indian ocean tsunami devastated many holiday destinations, Qantas said."The tsunami affected travel patterns in ways that we were a bit surprised about," chief executive Geoff Dixon explained. "It certainly affected Japanese travel into Australia. As soon as the tsunami hit we saw ... a lessening with bookings for Australia." Higher fuel costs also are expected to eat into earnings in coming months. "We don't have as much hedging benefit in the second half as we had in the first," said chief financial officer Peter Gregg. Qantas is facing increased pressure from rivals such as low-cost carrier Virgin Blue and the Australian government is in talks about whether to allow Singapore Airlines to fly between the Australia and the US - one of Qantas' key routes.

If the recommended category (e.g. new_category) does not belong to the 5 specified categories, the page will not save the article, instead, it will show the following response:



BBC News

Business Entertainment Politics Sport Technology About Logout

ADD NEWS ENTRY

Title:

Testing

Description:

```
Australian airline Qantas has posted a record fiscal first-half profit thanks to cost-cutting measures.Net profit in the six months ending 31
December rose 28% to A$458.4m ($357.6m; £191m) from a year earlier. Analysts expected a figure closer to A$431m. Qantas shares fell almost 3%,
however, after it warned that earnings growth would slow in the second half. Sales will dip by at least A$30m after the Indian ocean tsunami
devastated many holiday destinations, Qantas said."The tsunami affected travel patterns in ways that we were a bit surprised about," chief
executive Geoff Dixon explained. "It certainly affected Japanese travel into Australia. As soon as the tsunami hit we saw ... a lessening with
bookings for Australia." Higher fuel costs also are expected to eat into earnings in coming months. "We don't have as much hedging benefit in the
second half as we had in the first," said chief financial officer Peter Gregg. Qantas is facing increased pressure from rivals such as low-cost
carrier Virgin Blue and the Australian government is in talks about whether to allow Singapore Airlines to fly between the Australia and the US -
one of Qantas' key routes.
```

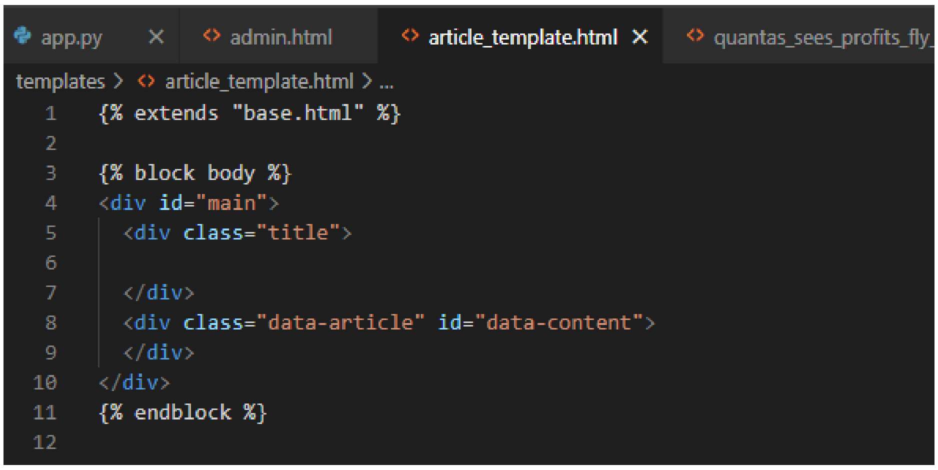
Recommended Category

new_category

Classify Save

Recommended category must belong to: entertainment, business, politics, sports, technology.

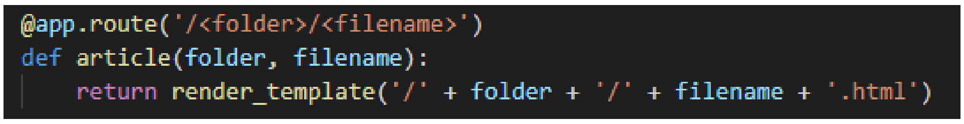
Answer of Task 2: Below is how the file *article_template.html* looks like. Basically, we define some empty <div> and we specify the classes of the empty divs as either "title" or "data-article" so that later we can use BeautifulSoup to detect these locations and append the corresponding data into these locations. Note that here, we still have some id like "main" or "data-content", this is same with other manual html files we created (e.g. quantas_sees_profits_fly_to_record.html, ect.).


app.py
admin.html
article_template.html
quantas_sees_profits_fly_
templates > article_template.html > ...

```
{% extends "base.html" %}

{% block body %}
<div id="main">
  <div class="title">

  </div>
  <div class="data-article" id="data-content">
  </div>
</div>
{% endblock %}

```

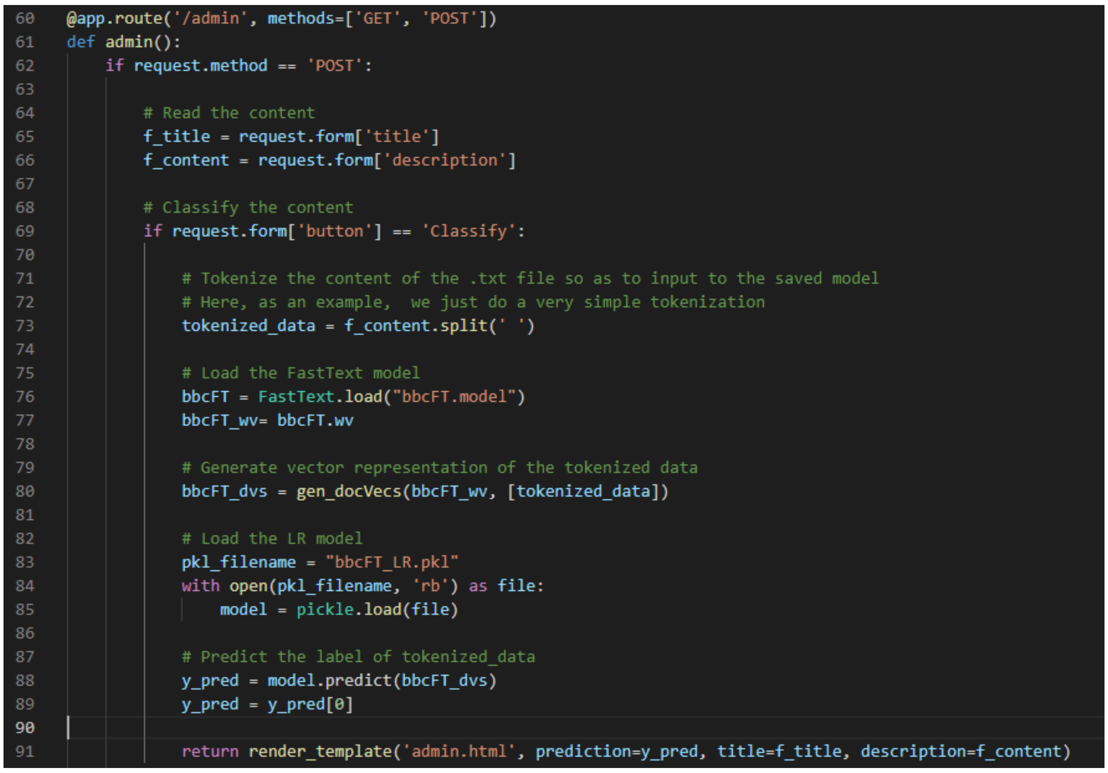
The file *admin.html* does not change compared to the previous Task. On the other hand, the route */admin* inside the app.py file will look like in the screenshots in the next page. Basically, compared to the view function of this route in the previous task (Task 1), here, we add an elif statement to check when the request.form['button'] is Save, if yes, then we specify the following actions:

- If the value in the field category to be empty, then we show the *admin.html* template again with a message saying "Recommeded category must not be empty."
- If the value in the field Recommended category is not in the 5 categories *entertainment, business, politics, sport, technology*, then we show the admin.html template again with a message saying, "Recommended category must belong to: entertainment, business, politics, sports, technology."
- Else, we will then use BeautifulSoup to read the *article_template.html*, and append the content within the fields Title and Description to the template and save to a new html file that is stored inside the folder of the corresponding category (inside the folder *templates*).
- Finally, we will redirect to the newly generated article. You can see that here, we can redirect to the newly generated article because the newly generated article is put into one of the folder business, entertainment, etc. and we already defined a route to display each article in the previous week (see the below screenshot). So, when we generate a new article and save it in the templates folder, we can route to it.

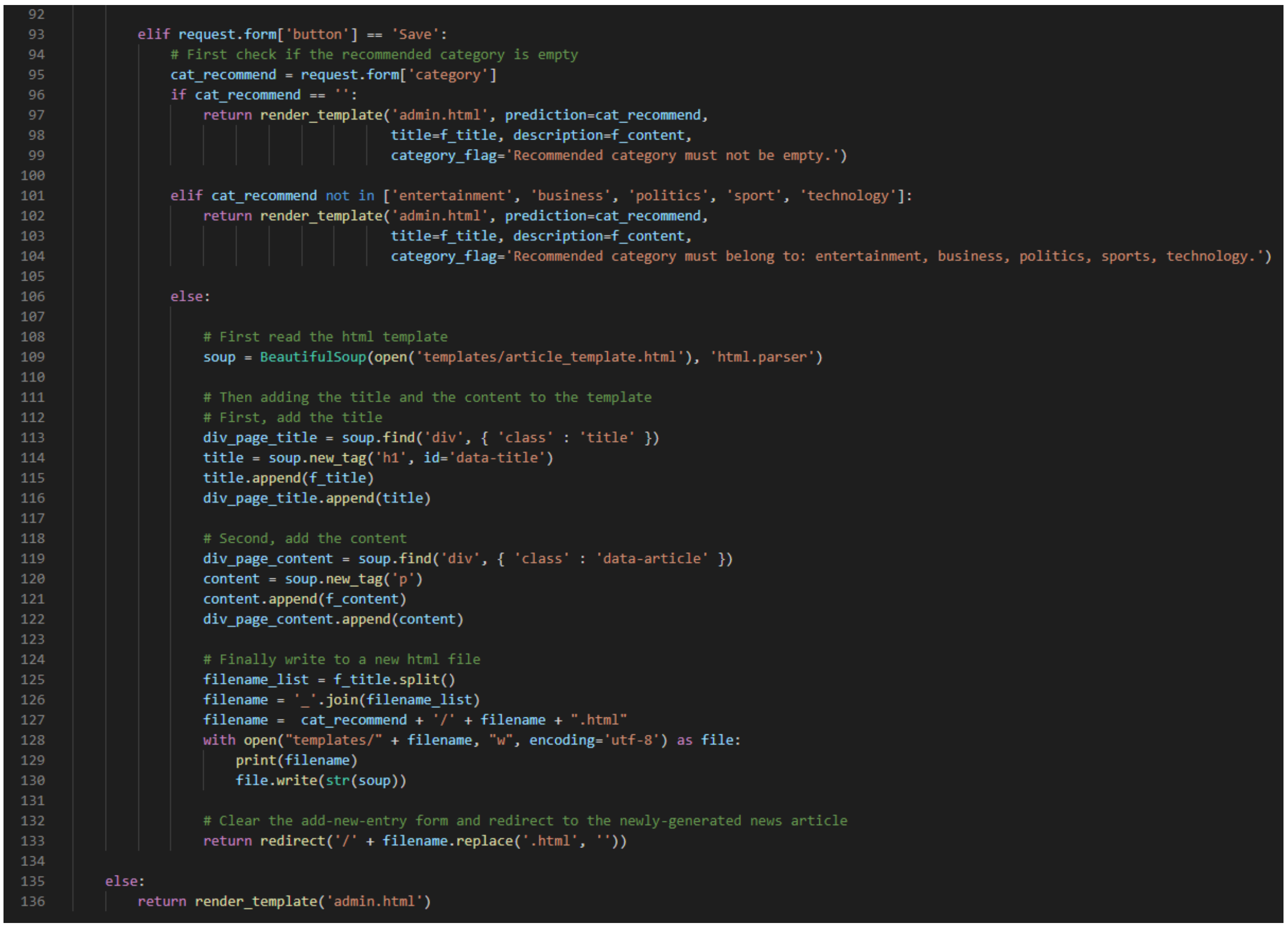


```
@app.route('/<folder>/<filename>')
def article(folder, filename):
    return render_template('/' + folder + '/' + filename + '.html')
```

The route /admin inside the app.py file.



```python
@app.route('/admin', methods=['GET', 'POST'])
def admin():
    if request.method == 'POST':

        # Read the content
        f_title = request.form['title']
        f_content = request.form['description']

        # Classify the content
        if request.form['button'] == 'Classify':

            # Tokenize the content of the .txt file so as to input to the saved model
            # Here, as an example,  we just do a very simple tokenization
            tokenized_data = f_content.split(' ')

            # Load the FastText model
            bbcFT = FastText.load("bbcFT.model")
            bbcFT_wv= bbcFT.wv

            # Generate vector representation of the tokenized data
            bbcFT_dvs = gen_docVecs(bbcFT_wv, [tokenized_data])

            # Load the LR model
            pkl_filename = "bbcFT_LR.pkl"
            with open(pkl_filename, 'rb') as file:
                model = pickle.load(file)

            # Predict the label of tokenized_data
            y_pred = model.predict(bbcFT_dvs)
            y_pred = y_pred[0]

            return render_template('admin.html', prediction=y_pred, title=f_title, description=f_content)
```



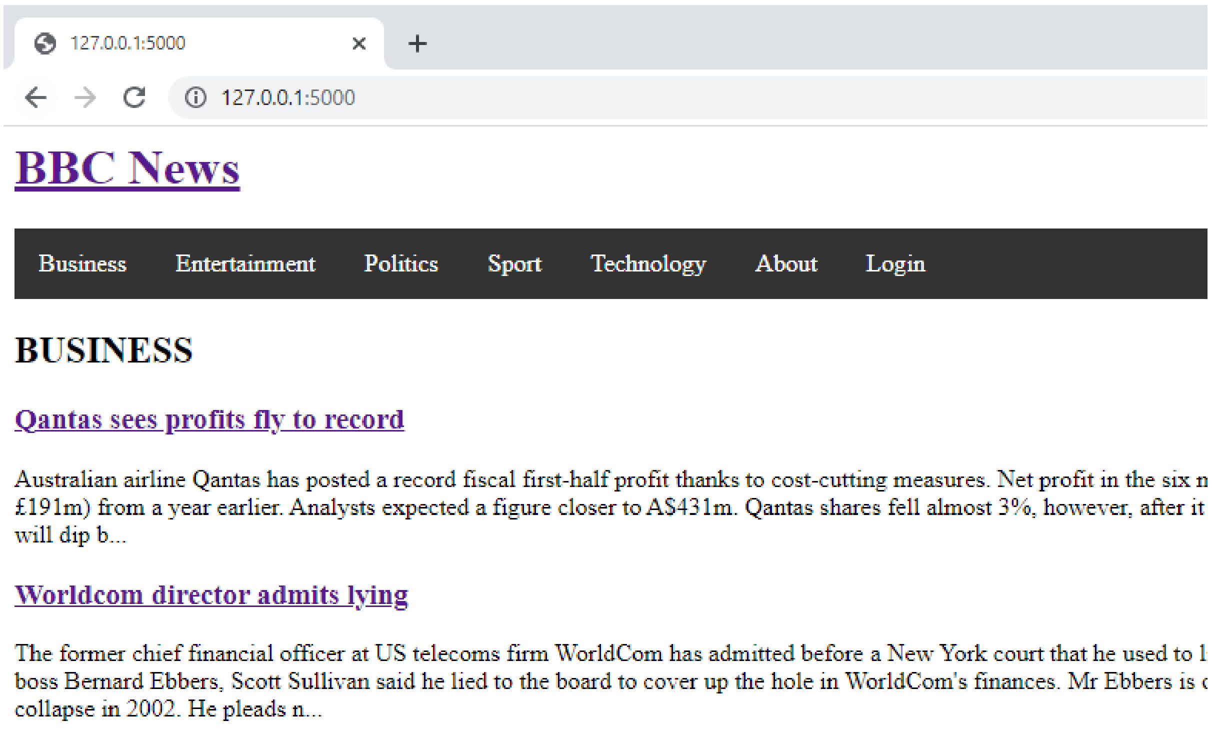
```python

        elif request.form['button'] == 'Save':
            # First check if the recommended category is empty
            cat_recommend = request.form['category']
            if cat_recommend == '':
                return render_template('admin.html', prediction=cat_recommend,
                                       title=f_title, description=f_content,
                                       category_flag='Recommended category must not be empty.')

            elif cat_recommend not in ['entertainment', 'business', 'politics', 'sport', 'technology']:
                return render_template('admin.html', prediction=cat_recommend,
                                       title=f_title, description=f_content,
                                       category_flag='Recommended category must belong to: entertainment, business, politics, sports, technology.')

            else:

                # First read the html template
                soup = BeautifulSoup(open('templates/article_template.html'), 'html.parser')

                # Then adding the title and the content to the template
                # First, add the title
                div_page_title = soup.find('div', { 'class' : 'title' })
                title = soup.new_tag('h1', id='data-title')
                title.append(f_title)
                div_page_title.append(title)

                # Second, add the content
                div_page_content = soup.find('div', { 'class' : 'data-article' })
                content = soup.new_tag('p')
                content.append(f_content)
                div_page_content.append(content)

                # Finally write to a new html file
                filename_list = f_title.split()
                filename = '_'.join(filename_list)
                filename =  cat_recommend + '/' + filename + ".html"
                with open("templates/" + filename, "w", encoding='utf-8') as file:
                    print(filename)
                    file.write(str(soup))

                # Clear the add-new-entry form and redirect to the newly-generated news article
                return redirect('/' + filename.replace('.html', ''))

    else:
        return render_template('admin.html')
```

Task 3. Adding login and logout routes

In this task, we will create login and logout routes for our web application. You will need to perform the following actions in order to create the login and logout routes:

- First, create a *login.html* template consisting two fields (username and password) and a submit button named Login. Hint: you can follow Section 2 in Module 11.4 to create a *login.html* file.
- Second, add a route /login to your web application. You can hardcode the username and password. Hint: you can follow Section 2 in Module 11.4 to create the login route. Remember to import the session function of flask, and set the secret key of your web app.
- Third, add a route /logout to your web application. Hint: you can follow Section 3 in Module 11.4 to create the logout route.
- Modify the *base.html* file such that now all the child templates (except the *admin.html*) have the link to the /login route. On the other hand, the *admin.html* template will have the link to /logout page. Note here, we just try to simplify things, so we just display the /logout page in the *admin.html* file.
- Modify the view function corresponding to the route /admin such that when the user is not logged in, it will redirect to the route /login.

Below is how your web app's home page looks like at the end of this task:



BBC News

Business Entertainment Politics Sport Technology About Login

BUSINESS

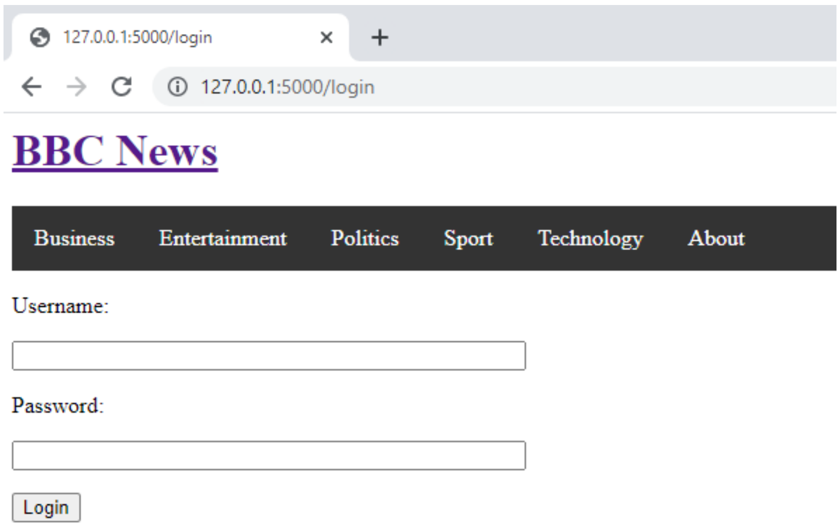
Qantas sees profits fly to record

Australian airline Qantas has posted a record fiscal first-half profit thanks to cost-cutting measures. Net profit in the six n
£191m) from a year earlier. Analysts expected a figure closer to A$431m. Qantas shares fell almost 3%, however, after it
will dip b...

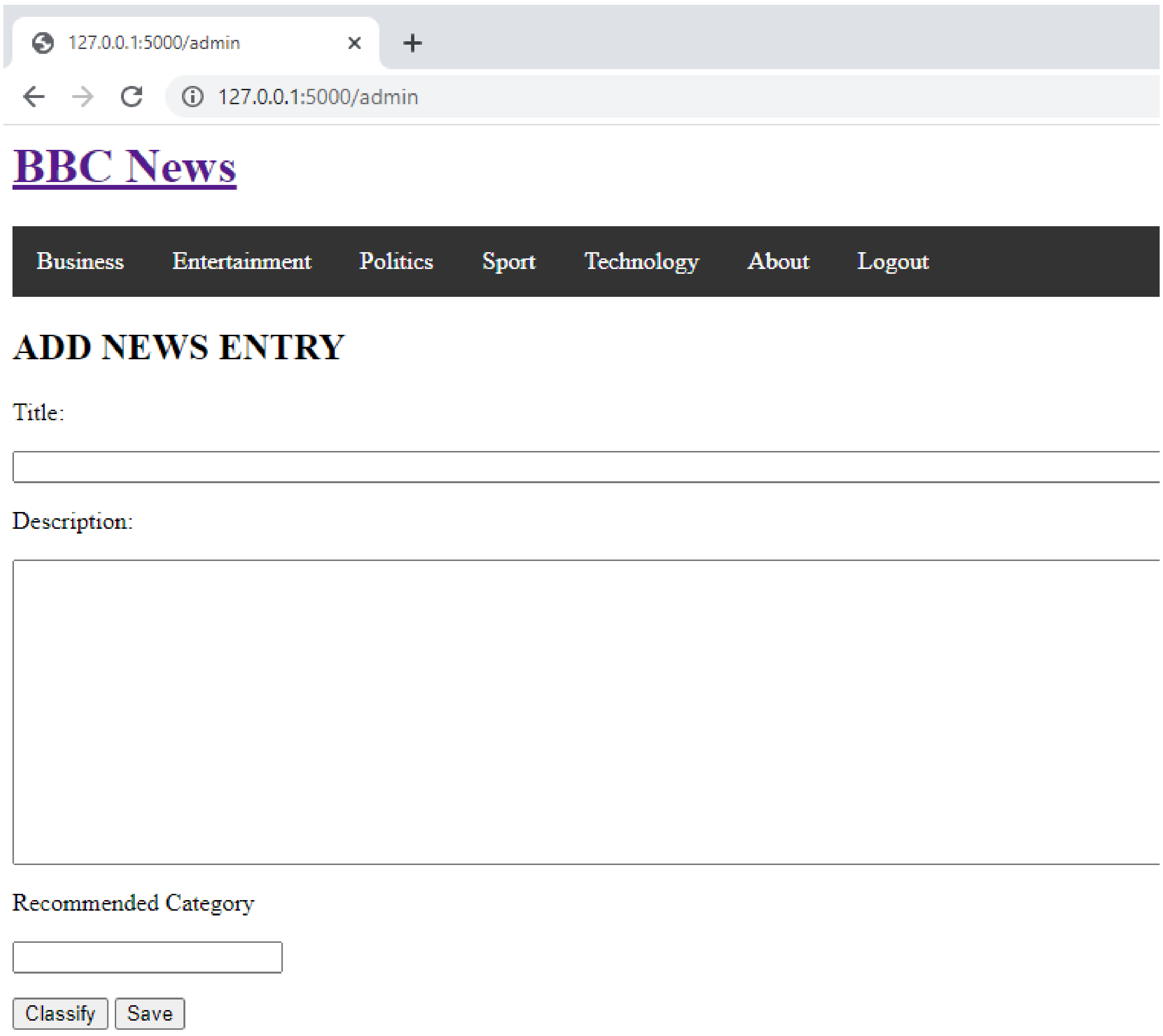
Worldcom director admits lying

The former chief financial officer at US telecoms firm WorldCom has admitted before a New York court that he used to l
boss Bernard Ebbers, Scott Sullivan said he lied to the board to cover up the hole in WorldCom's finances. Mr Ebbers is c
collapse in 2002. He pleads n...

When click on the Login link, we will see this login form:

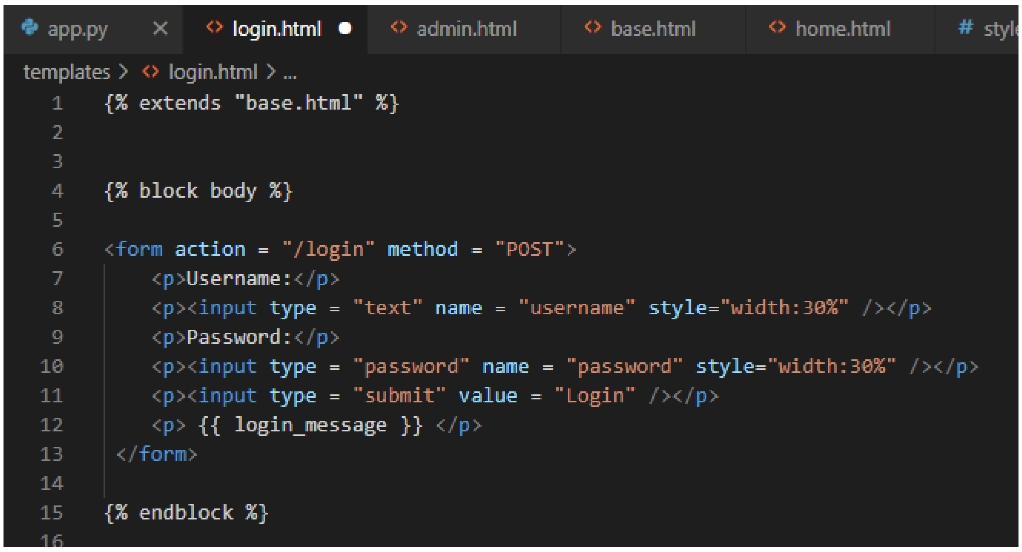


When entering correct Username and password, it will redirect to the /admin route.



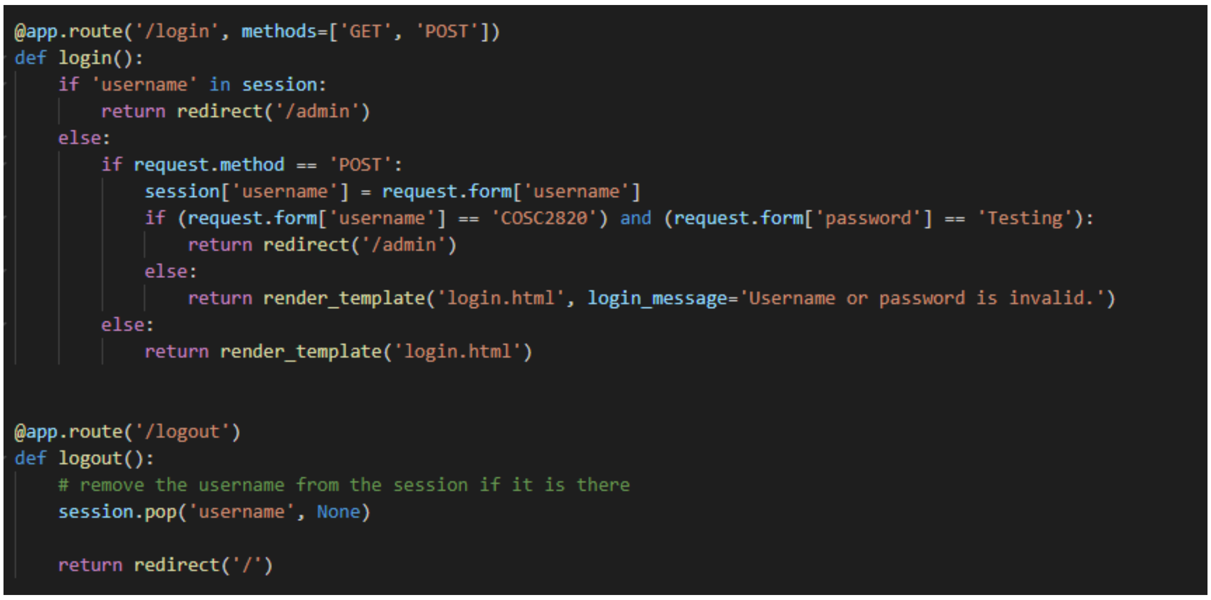
Finally, when we click the Logout link inside the route /admin, we will be redirected to the homepage. Note that if a user already logged in, and hasn't logged out yet, the web application will allow the user to visit the route /admin. But for a user who doesn't log in, then if that user visits the route /admin, it will redirect to the route /login.

Answer of Task 3: Below is how the file *login.html* looks like. It is similar with how we implement the file *admin.html* (e.g. action to be link to the route /login, method to be POST). One note here is that we set the type of the password field to be password, so that later, when users enter their passwords, they is automatically hidden.



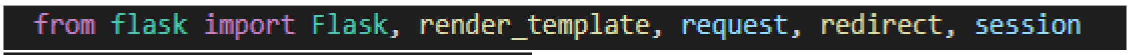
```
{% extends "base.html" %}


{% block body %}

<form action = "/login" method = "POST">
    <p>Username:</p>
    <p><input type = "text" name = "username" style="width:30%" /></p>
    <p>Password:</p>
    <p><input type = "password" name = "password" style="width:30%" /></p>
    <p><input type = "submit" value = "Login" /></p>
    <p> {{ login_message }} </p>
 </form>

{% endblock %}
```

Below is how we the routes /login and /logout are implemented in the app.py file. You can follow the explanation in Module 11.4. Basically, we check if user is logged in ('username' in session) if yes, we can let user to visit the route /admin. Otherwise, we need to ask users to log in.

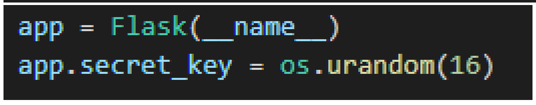


```
@app.route('/login', methods=['GET', 'POST'])
def login():
    if 'username' in session:
        return redirect('/admin')
    else:
        if request.method == 'POST':
            session['username'] = request.form['username']
            if (request.form['username'] == 'COSC2820') and (request.form['password'] == 'Testing'):
                return redirect('/admin')
            else:
                return render_template('login.html', login_message='Username or password is invalid.')
        else:
            return render_template('login.html')


@app.route('/logout')
def logout():
    # remove the username from the session if it is there
    session.pop('username', None)

    return redirect('/')
```

Note, we need to import session at the beginning of the app.py file and set the secret key.

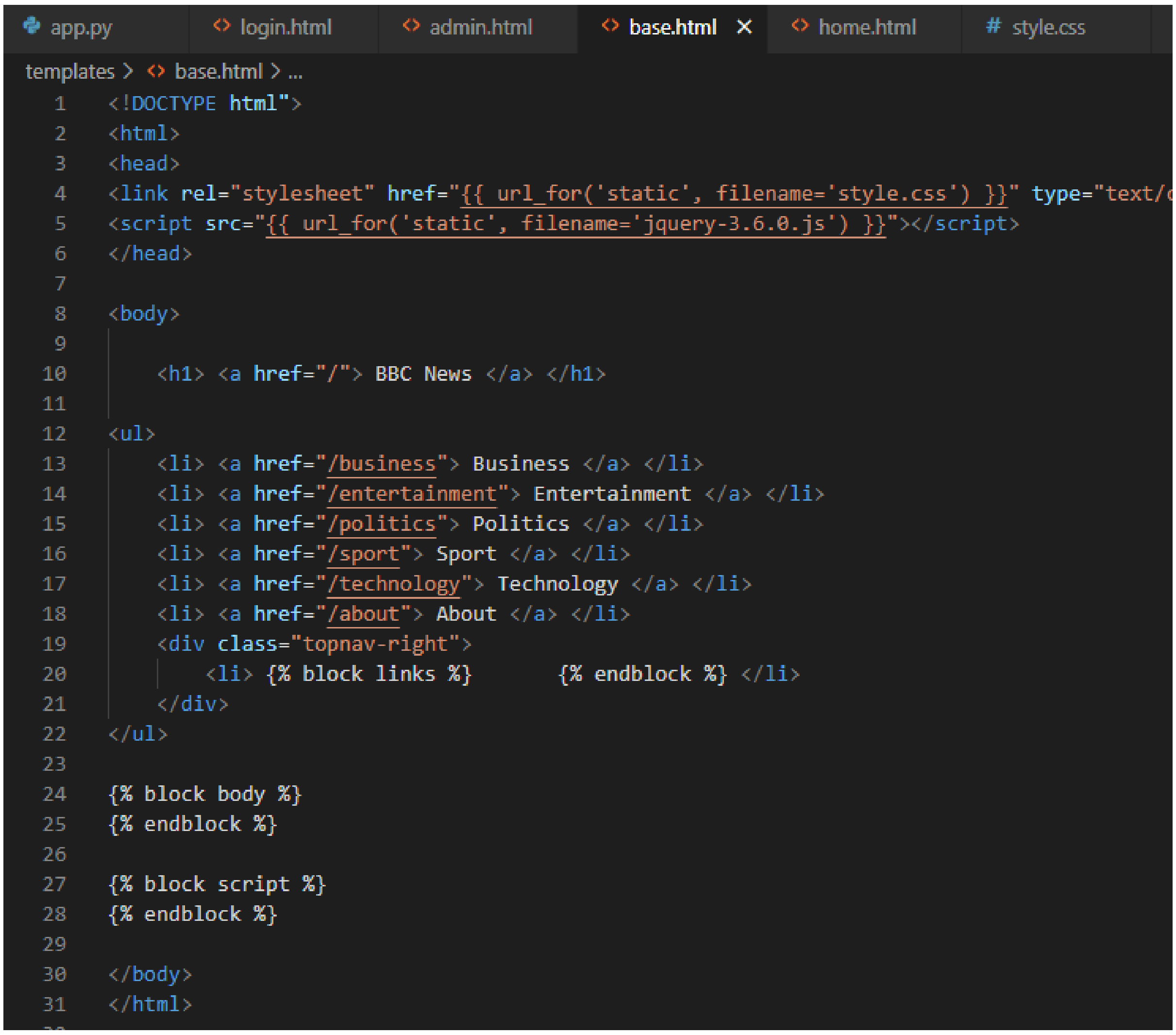


```
from flask import Flask, render_template, request, redirect, session
```



```
app = Flask(__name__)
app.secret_key = os.urandom(16)
```

Below is how the file *base.html* looks like. Note that compared to the previous tasks, we add a {% block links %}   {% endblock %} tag into the navigation bar, for the child templates to customize the link. For example, the child templates (home.html, business.html) to choose if they will display the login or logout link.

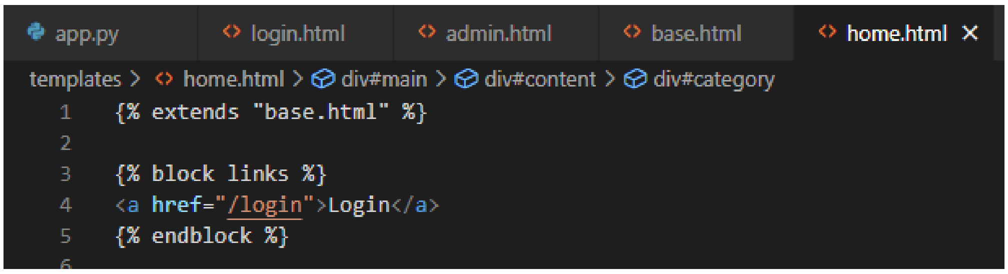


app.py | login.html | admin.html | base.html | home.html | style.css

templates > base.html > ...

```html
<!DOCTYPE html">
<html>
<head>
<link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}" type="text/c
<script src="{{ url_for('static', filename='jquery-3.6.0.js') }}"></script>
</head>

<body>

    <h1> <a href="/"> BBC News </a> </h1>

<ul>
    <li> <a href="/business"> Business </a> </li>
    <li> <a href="/entertainment"> Entertainment </a> </li>
    <li> <a href="/politics"> Politics </a> </li>
    <li> <a href="/sport"> Sport </a> </li>
    <li> <a href="/technology"> Technology </a> </li>
    <li> <a href="/about"> About </a> </li>
    <div class="topnav-right">
        <li> {% block links %}       {% endblock %} </li>
    </div>
</ul>

{% block body %}
{% endblock %}

{% block script %}
{% endblock %}

</body>
</html>
```

Below is how the file *home.html* looks like (for the first several lines, the rest doesn't change compared to previous tasks). We set the link to be login.

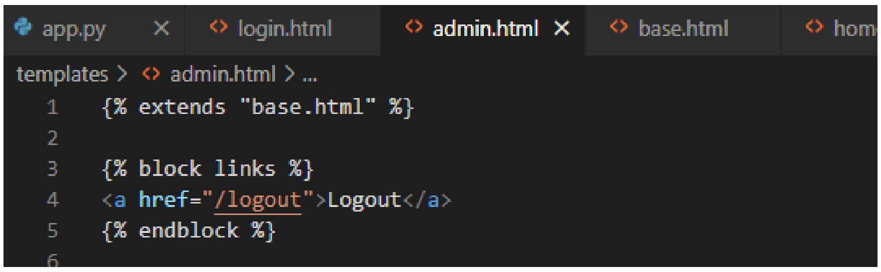


app.py | login.html | admin.html | base.html | home.html

templates > home.html > div#main > div#content > div#category

```html
{% extends "base.html" %}

{% block links %}
<a href="/login">Login</a>
{% endblock %}
```

The same with other files such as *business.html, entertainment.html*, etc. Only within the file *admin.html*, we specify the link to be logout:

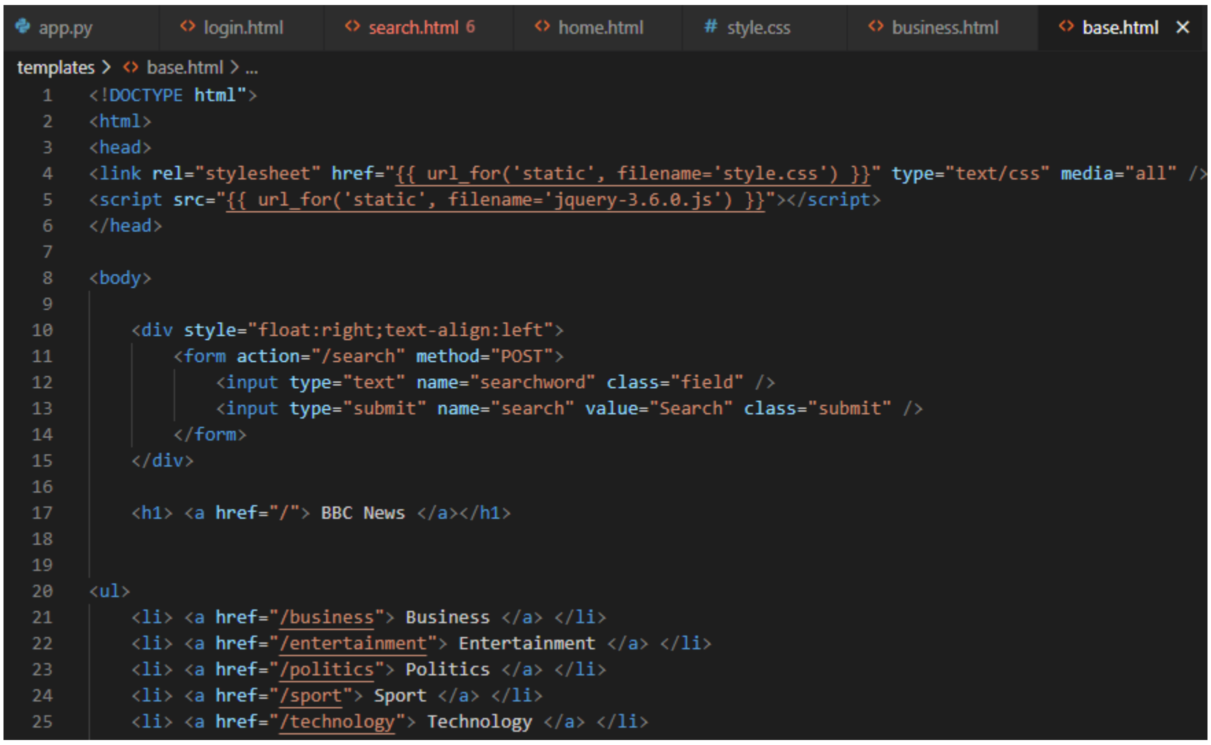


```
{% extends "base.html" %}

{% block links %}
<a href="/logout">Logout</a>
{% endblock %}
```

Task 4. Perform search on the news articles

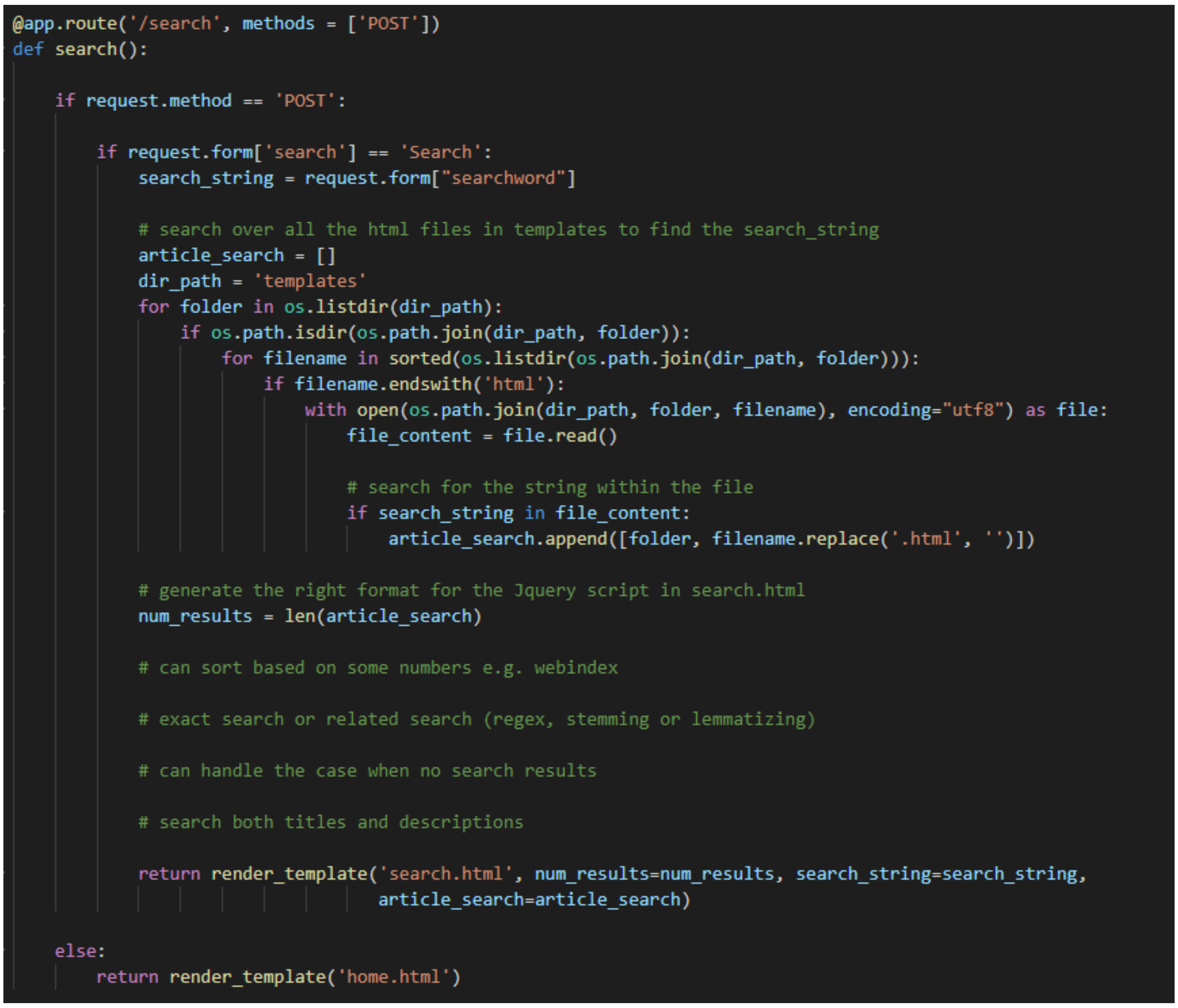
In this task, we will create a search bar, and create the corresponding route to search for all the articles based on a keyword. You need to follow the below actions in order to create a search bar and the corresponding route /search:

- Modify the file *base.html* in order to include a search bar. One solution is to modify the file *base.html* to include a webform as below (Lines 10-15). Note the screenshot only shows the first half of the file *base.html*. The importnat point is that you need to specify the action of the form to be '/search' so that when we click on the Search button, it will redirect to the route /search. The second important key point is to set the method to be POST.



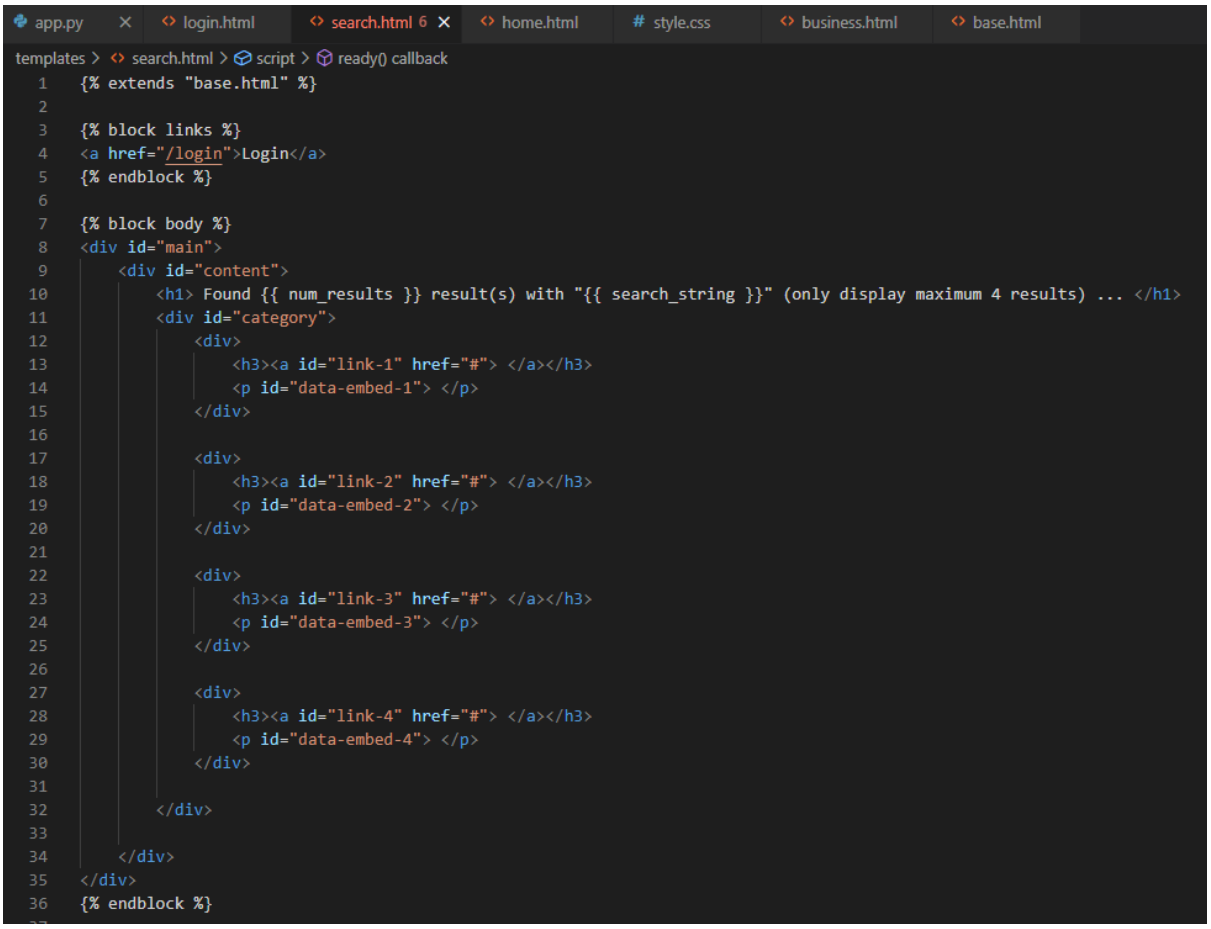
```
1   <!DOCTYPE html">
2   <html>
3   <head>
4   <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}" type="text/css" media="all" />
5   <script src="{{ url_for('static', filename='jquery-3.6.0.js') }}"></script>
6   </head>
7
8   <body>
9
10      <div style="float:right;text-align:left">
11          <form action="/search" method="POST">
12              <input type="text" name="searchword" class="field" />
13              <input type="submit" name="search" value="Search" class="submit" />
14          </form>
15      </div>
16
17      <h1> <a href="/"> BBC News </a></h1>
18
19
20  <ul>
21      <li> <a href="/business"> Business </a> </li>
22      <li> <a href="/entertainment"> Entertainment </a> </li>
23      <li> <a href="/politics"> Politics </a> </li>
24      <li> <a href="/sport"> Sport </a> </li>
25      <li> <a href="/technology"> Technology </a> </li>
```

- Create a route /search in the file *app.py*. The main idea is to simply go through all the articles in the templates folder and search if any article contains the search keyword. The search keyword can be obtained by request.form["searchword"] (based on how we define the web form of the search bar in base.html). At the end, the view function returns the template *search.html* and pass into this template some useful information such as the number of search results, the list of matched articles, etc. Below is how the route /search looks like in the file *app.py*. Note for this route, we set the web application in a way that it can be only called through the POST method. If users directly access to the route /search, it will show an error page. Also, here we implement a very simple search method (e.g find a string within the articles, you can and you should do more sophisticated search).

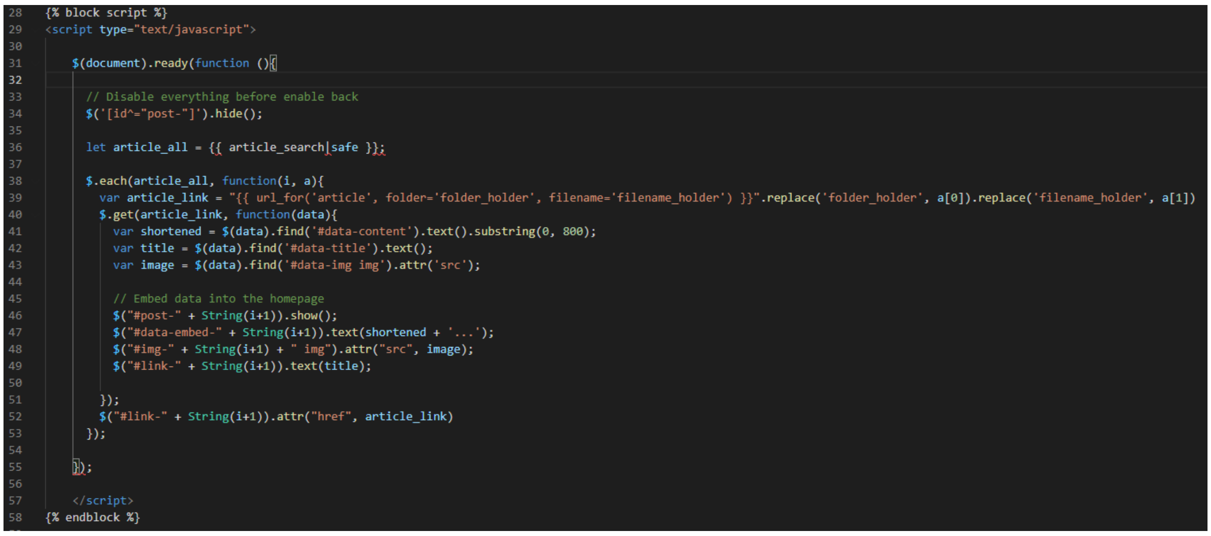


```python
@app.route('/search', methods = ['POST'])
def search():

    if request.method == 'POST':

        if request.form['search'] == 'Search':
            search_string = request.form["searchword"]

            # search over all the html files in templates to find the search_string
            article_search = []
            dir_path = 'templates'
            for folder in os.listdir(dir_path):
                if os.path.isdir(os.path.join(dir_path, folder)):
                    for filename in sorted(os.listdir(os.path.join(dir_path, folder))):
                        if filename.endswith('html'):
                            with open(os.path.join(dir_path, folder, filename), encoding="utf8") as file:
                                file_content = file.read()

                                # search for the string within the file
                                if search_string in file_content:
                                    article_search.append([folder, filename.replace('.html', '')])

            # generate the right format for the Jquery script in search.html
            num_results = len(article_search)

            # can sort based on some numbers e.g. webindex

            # exact search or related search (regex, stemming or lemmatizing)

            # can handle the case when no search results

            # search both titles and descriptions

            return render_template('search.html', num_results=num_results, search_string=search_string,
                                   article_search=article_search)

    else:
        return render_template('home.html')
```

- Create a file *search.html* to obtain the passed-in arguments from the view function of the route /search. After having a list of articles passed in by the view function, we can use Jinja variables in order to display those results. We already prepare for you a Javascript script (based on jquery) to read all the content in those passed-in articles from the view function and display on the search page. This method is similar to when we use Javascript script to automatically display the content of the news articles on the home page. The file *search.html* will look as follows.

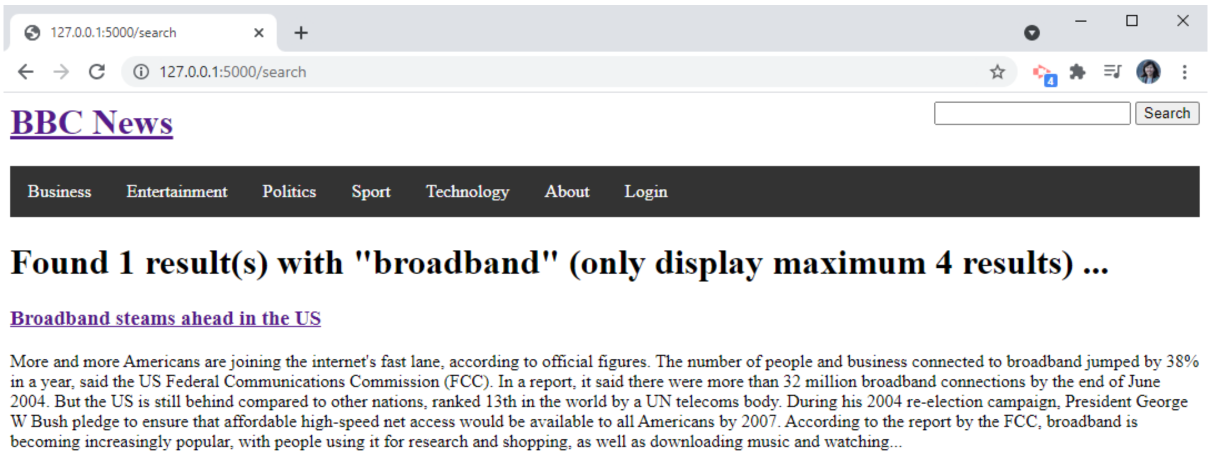


```
{% extends "base.html" %}

{% block links %}
<a href="/login">Login</a>
{% endblock %}

{% block body %}
<div id="main">
    <div id="content">
        <h1> Found {{ num_results }} result(s) with "{{ search_string }}" (only display maximum 4 results) ... </h1>
        <div id="category">
            <div>
                <h3><a id="link-1" href="#"> </a></h3>
                <p id="data-embed-1"> </p>
            </div>

            <div>
                <h3><a id="link-2" href="#"> </a></h3>
                <p id="data-embed-2"> </p>
            </div>

            <div>
                <h3><a id="link-3" href="#"> </a></h3>
                <p id="data-embed-3"> </p>
            </div>

            <div>
                <h3><a id="link-4" href="#"> </a></h3>
                <p id="data-embed-4"> </p>
            </div>

        </div>

    </div>
</div>
{% endblock %}
```



```
{% block script %}
<script type="text/javascript">

    $(document).ready(function (){

      // Disable everything before enable back
      $('[id^="post-"]').hide();

      let article_all = {{ article_search|safe }};

      $.each(article_all, function(i, a){
        var article_link = "{{ url_for('article', folder='folder_holder', filename='filename_holder') }}".replace('folder_holder', a[0]).replace('filename_holder', a[1])
        $.get(article_link, function(data){
          var shortened = $(data).find('#data-content').text().substring(0, 800);
          var title = $(data).find('#data-title').text();
          var image = $(data).find('#data-img img').attr('src');

          // Embed data into the homepage
          $("#post-" + String(i+1)).show();
          $("#data-embed-" + String(i+1)).text(shortened + '...');
          $("#img-" + String(i+1) + " img").attr("src", image);
          $("#link-" + String(i+1)).text(title);

        });
        $("#link-" + String(i+1)).attr("href", article_link)
      });

    });

    </script>
{% endblock %}
```

At the end of this task, when you type in any search word in the search bar, it will redirect the route /search and display all the related articles. For example, when you type 'broadband' in the search bar, you will see the following response:



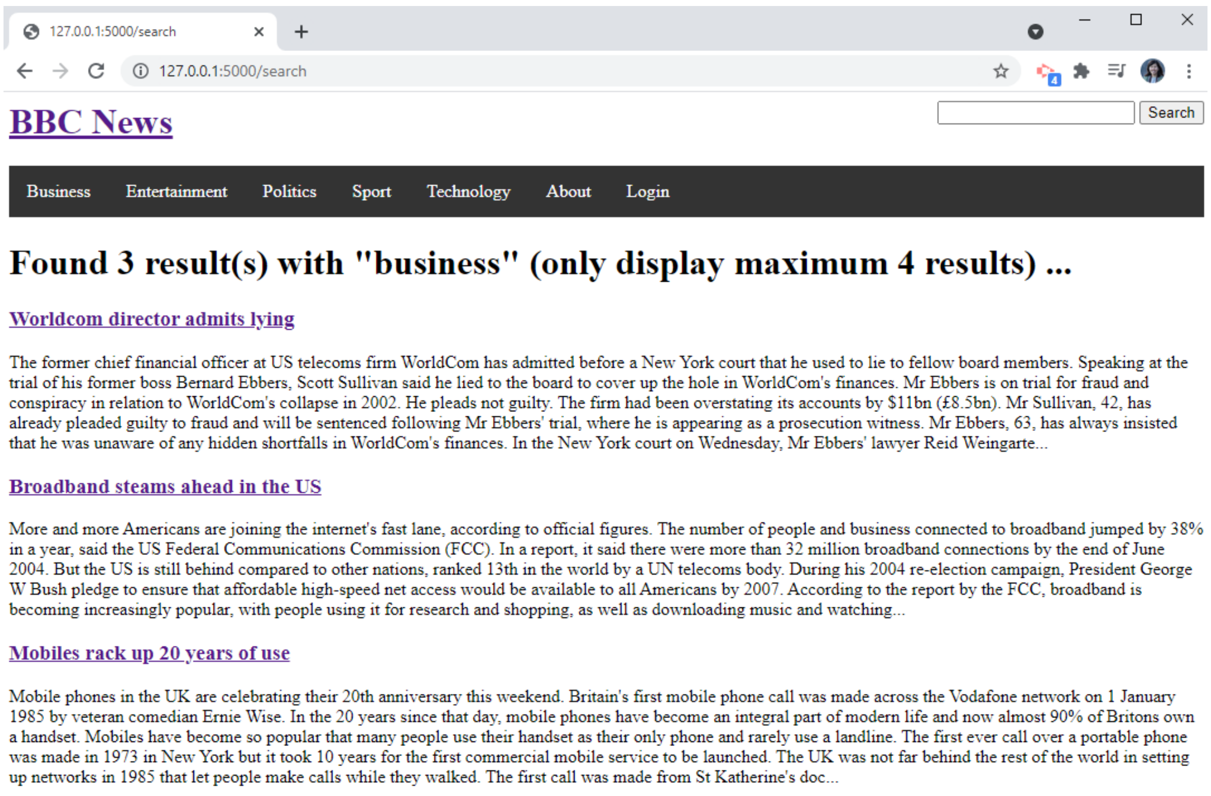
BBC News

Found 1 result(s) with "broadband" (only display maximum 4 results) ...

Broadband steams ahead in the US

More and more Americans are joining the internet's fast lane, according to official figures. The number of people and business connected to broadband jumped by 38% in a year, said the US Federal Communications Commission (FCC). In a report, it said there were more than 32 million broadband connections by the end of June 2004. But the US is still behind compared to other nations, ranked 13th in the world by a UN telecoms body. During his 2004 re-election campaign, President George W Bush pledge to ensure that affordable high-speed net access would be available to all Americans by 2007. According to the report by the FCC, broadband is becoming increasingly popular, with people using it for research and shopping, as well as downloading music and watching...

Or when you type in some words like "business", you will see the following response.



BBC News

Found 3 result(s) with "business" (only display maximum 4 results) ...

Worldcom director admits lying

The former chief financial officer at US telecoms firm WorldCom has admitted before a New York court that he used to lie to fellow board members. Speaking at the trial of his former boss Bernard Ebbers, Scott Sullivan said he lied to the board to cover up the hole in WorldCom's finances. Mr Ebbers is on trial for fraud and conspiracy in relation to WorldCom's collapse in 2002. He pleads not guilty. The firm had been overstating its accounts by $11bn (£8.5bn). Mr Sullivan, 42, has already pleaded guilty to fraud and will be sentenced following Mr Ebbers' trial, where he is appearing as a prosecution witness. Mr Ebbers, 63, has always insisted that he was unaware of any hidden shortfalls in WorldCom's finances. In the New York court on Wednesday, Mr Ebbers' lawyer Reid Weingarte...

Broadband steams ahead in the US

More and more Americans are joining the internet's fast lane, according to official figures. The number of people and business connected to broadband jumped by 38% in a year, said the US Federal Communications Commission (FCC). In a report, it said there were more than 32 million broadband connections by the end of June 2004. But the US is still behind compared to other nations, ranked 13th in the world by a UN telecoms body. During his 2004 re-election campaign, President George W Bush pledge to ensure that affordable high-speed net access would be available to all Americans by 2007. According to the report by the FCC, broadband is becoming increasingly popular, with people using it for research and shopping, as well as downloading music and watching...

Mobiles rack up 20 years of use

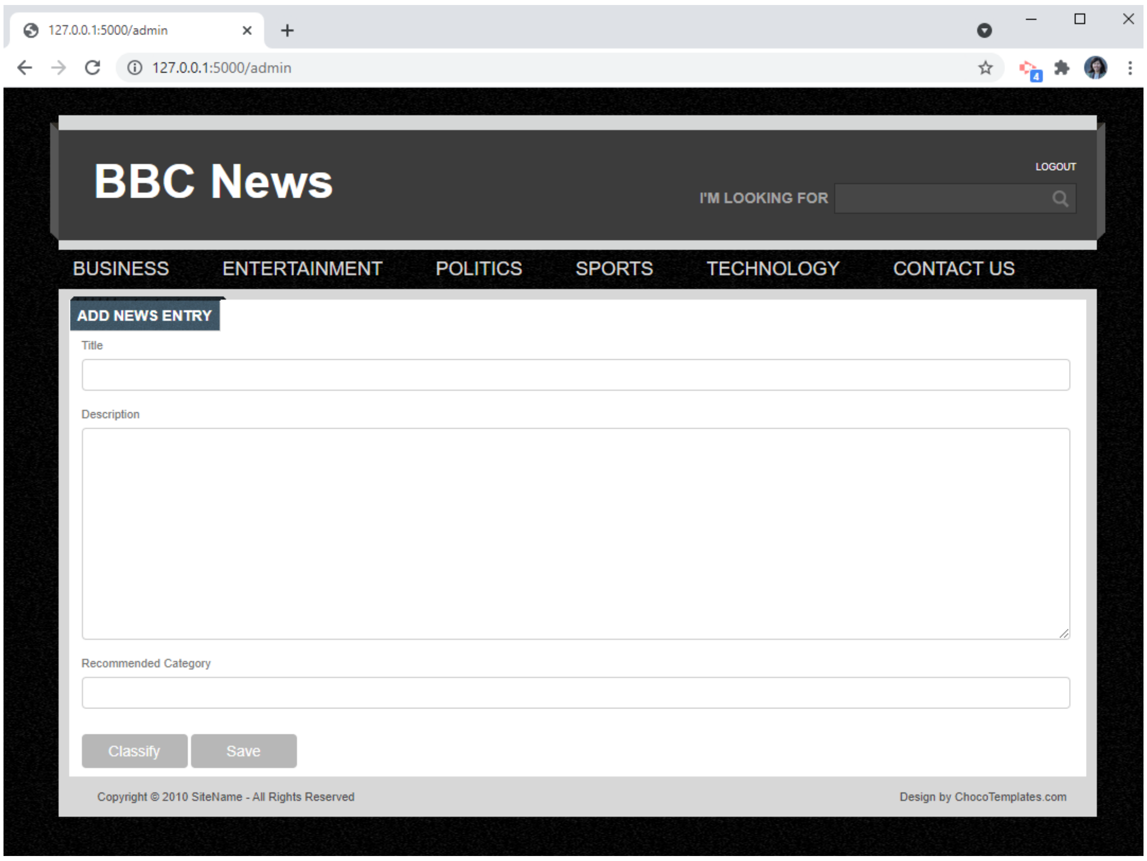
Mobile phones in the UK are celebrating their 20th anniversary this weekend. Britain's first mobile phone call was made across the Vodafone network on 1 January 1985 by veteran comedian Ernie Wise. In the 20 years since that day, mobile phones have become an integral part of modern life and now almost 90% of Britons own a handset. Mobiles have become so popular that many people use their handset as their only phone and rarely use a landline. The first ever call over a portable phone was made in 1973 in New York but it took 10 years for the first commercial mobile service to be launched. The UK was not far behind the rest of the world in setting up networks in 1985 that let people make calls while they walked. The first call was made from St Katherine's doc...

Task 5. Make the design of the web app to be more beautiful

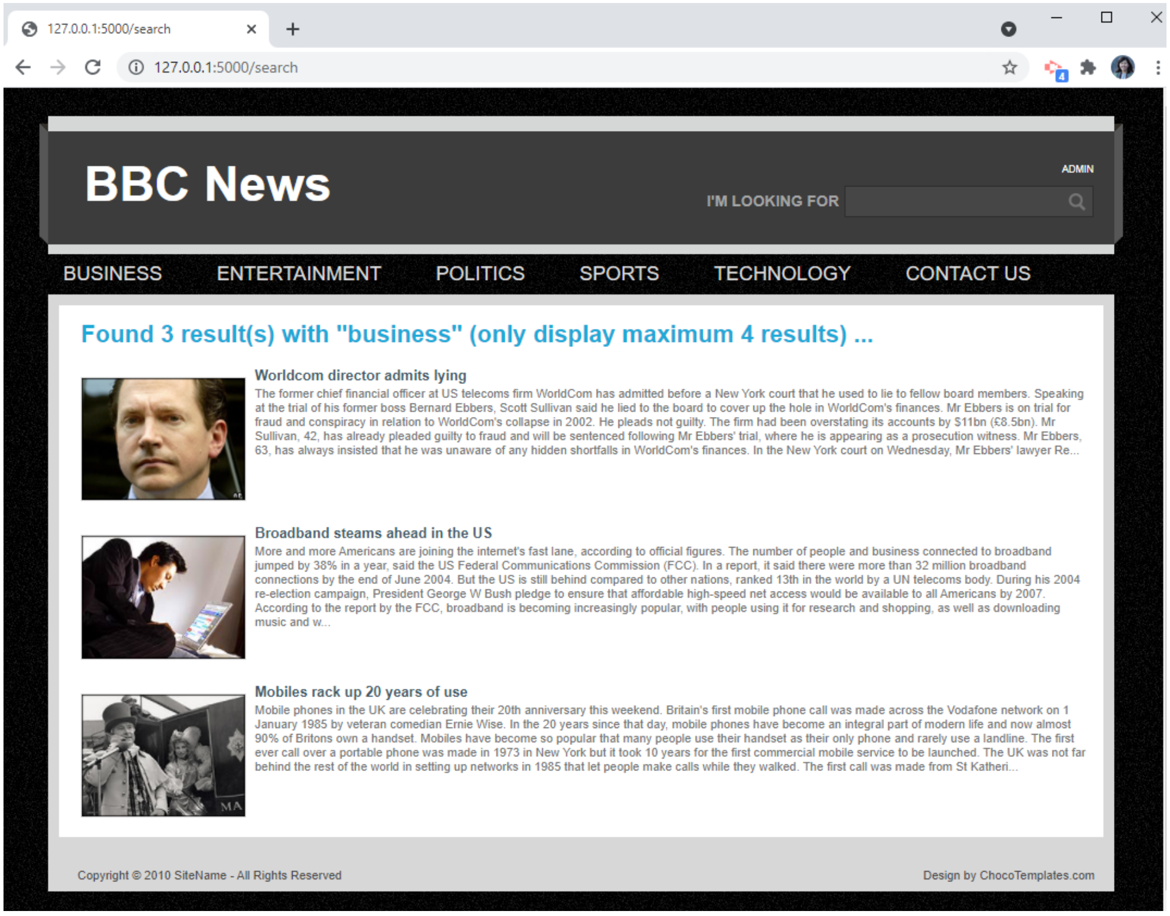
In the previous tasks, we have used the simple Flask web application we developed in the previous exercise of Week 10. In this task, you can use the advance Flask web application and perform the similar tasks as in Tasks 1 to 4 in order to modify the advance web application to support classify, save news article, log in, log out, and search articles.

We will make the code of both the simple and advance web app to be available at the end of Week 11.

Below is how the Admin tab looks like for the advance version:



This is how the Search results look like.



Found 3 result(s) with "business" (only display maximum 4 results) ...

Worldcom director admits lying
The former chief financial officer at US telecoms firm WorldCom has admitted before a New York court that he used to lie to fellow board members. Speaking at the trial of his former boss Bernard Ebbers, Scott Sullivan said he lied to the board to cover up the hole in WorldCom's finances. Mr Ebbers is on trial for fraud and conspiracy in relation to WorldCom's collapse in 2002. He pleads not guilty. The firm had been overstating its accounts by $11bn (£8.5bn). Mr Sullivan, 42, has already pleaded guilty to fraud and will be sentenced following Mr Ebbers' trial, where he is appearing as a prosecution witness. Mr Ebbers, 63, has always insisted that he was unaware of any hidden shortfalls in WorldCom's finances. In the New York court on Wednesday, Mr Ebbers' lawyer Re...

Broadband steams ahead in the US
More and more Americans are joining the internet's fast lane, according to official figures. The number of people and business connected to broadband jumped by 38% in a year, said the US Federal Communications Commission (FCC). In a report, it said there were more than 32 million broadband connections by the end of June 2004. But the US is still behind compared to other nations, ranked 13th in the world by a UN telecoms body. During his 2004 re-election campaign, President George W Bush pledge to ensure that affordable high-speed net access would be available to all Americans by 2007. According to the report by the FCC, broadband is becoming increasingly popular, with people using it for research and shopping, as well as downloading music and w...

Mobiles rack up 20 years of use
Mobile phones in the UK are celebrating their 20th anniversary this weekend. Britain's first mobile phone call was made across the Vodafone network on 1 January 1985 by veteran comedian Ernie Wise. In the 20 years since that day, mobile phones have become an integral part of modern life and now almost 90% of Britons own a handset. Mobiles have become so popular that many people use their handset as their only phone and rarely use a landline. The first ever call over a portable phone was made in 1973 in New York but it took 10 years for the first commercial mobile service to be launched. The UK was not far behind the rest of the world in setting up networks in 1985 that let people make calls while they walked. The first call was made from St Katheri...

Copyright © 2010 SiteName - All Rights Reserved
Design by ChocoTemplates.com