

Final Project Stream Three

motorholics django project

Christine Wilde | Full Stack Software Developer | 15.09.2018

Table of Contents

**Project Brief**3

Keywords3

Statistics4

**Technical Inclusions4**

Back-end4

Front-end5

Project Deployment5

**The Prototype**6

Charts Required6

**Back-end Setup**9

**Front-end Setup**12

Index.html12

Graph.js13

Custom.css13

**Testing Results15**

**The Complete Product**29

The nav bar29

Line Chart – Amount of Fish Types Through the Years29

Pie Chart – Fish by Region30

Pie Chart – Main Regional Ports30

Select Menu – Fish Names and Row Chart – Fish Types31

Conclusion32

**Heroku Deployment**33

Heroku new app setup33

Requirements file34

Procfile34

Runtime.txt35

**Appendix A - References**36

## Project Brief

The goal of the stream three project is to build a Django project composed of multiple applications (apps). The Django project should include an ecommerce element to it, establish a login/logout authentication mechanism for users to the Website and connect to a database to store Website data.

After considering a few ideas the Motorholics Website was born, aiming to connect all users and car enthusiasts to car mechanics for advice and assistance with any car issues or subject interest.

This meant that the Website should contain a discussion feature for all users to comment on. It also allows scope for the mechanics to write any articles of interest to further educate users. Therefore, a forum and a blog was decided on.

The mechanics can respond to users’ issues via the forum and in a real world example can be ‘searched’ via search engine such as Google. However, since the mechanics would be spending the time and offering valuable car advice which normally users would obtain a car garage it was decided that a subscription facility should be established so that those users requiring repeated assistance can obtain first hand quick advice on a prolonged basis.

Therefore, the main features of this Django Website are:

* Car Blog (mechanics car topics)
* Car Forum (mechanics and users discussion forum – help site!)
* User Registration (register to ask for advice via a small subscription)

## Technical inclusions

The list below presents the technologies used to accomplish the Django Motorholics Website from the initial prototype versions to the completed Website product.

### Frameworks

Django 1.11.15

* The Django framework allowed creation of:

**Models** – this allowed classes, functions and tables to be established that could then be updated to the database.

**Views** – this established the business logic (e.g. the user registration and how to handle email and password validation). The views link to the models and html templates

**Templates** – here is where all the html templates are kept which presents the visual representation of the codes and it’s functions.

### Back-end

SQLite

* The open source SQL database allowed all the models and their data to be stored. This is included with the Python so conveniently no further installations are required.

### Front-end

(Semantic) HTML5

* Website HTML structure – DOM

CSS

* + General and advanced Website display including animation (external style sheet)

JavaScript

* JavaScript language was used to include a responsive ‘best car’ feature

Bootstrap

* The free front-end framework in order to contribute to designing an eye-catching Website

### Other

Disqus

* Disqus was used to include a global comments platform for the Motorholic blog

Stripe

* Stripe API’s allow payment processing and this was used for the user registration/subscription model

Tinymce

* Tinymce is a WYSIWYG HTML editor and allows users to update threads in a pretty table on the Motorholics forum

Font Awesome

* Font Awesome was used for add icons to make the functions more enjoyable such as the car icon on the buttons

### Project deployment

GitHub, GitBash, GitHub Pages

* + Hosting the software project files and version control
  + Creating repositories and branches for project files

Heroku

* Heroku was used to host the project’s files and application

## The Prototype

As with the other projects the prototype was first roughly sketched on paper – this approach is very useful to get ideas down quickly and can be easily discarded and started again!

Mockplus prototyping free limited edition was chosen for it’s ease of use and convenient tool belt. The focus was on a smart useful Website with intelligent coding.

The below pictures show the first sketches for the Website. It was initially thought that a MOT/Service booking would be helpful although since there are many available the idea turned into a more blog/forum help site later on.

Figure 1. Early sketch of Motorholic components

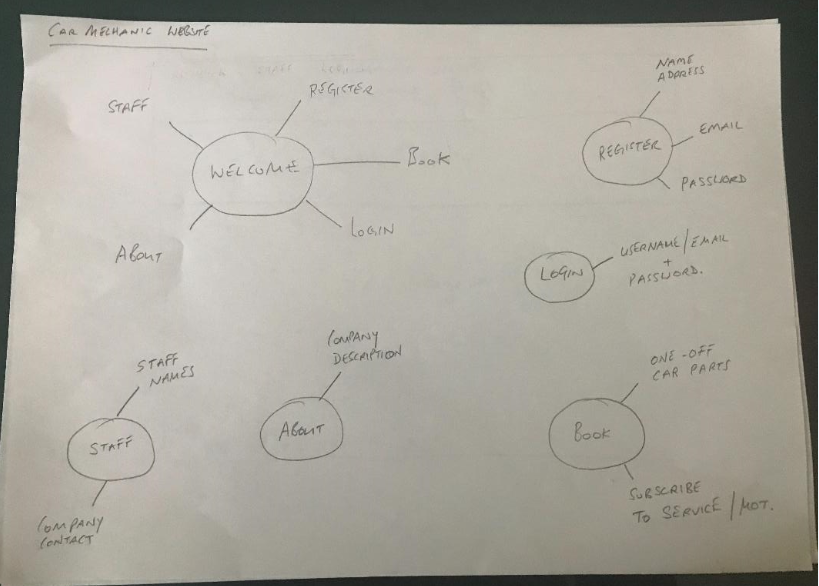
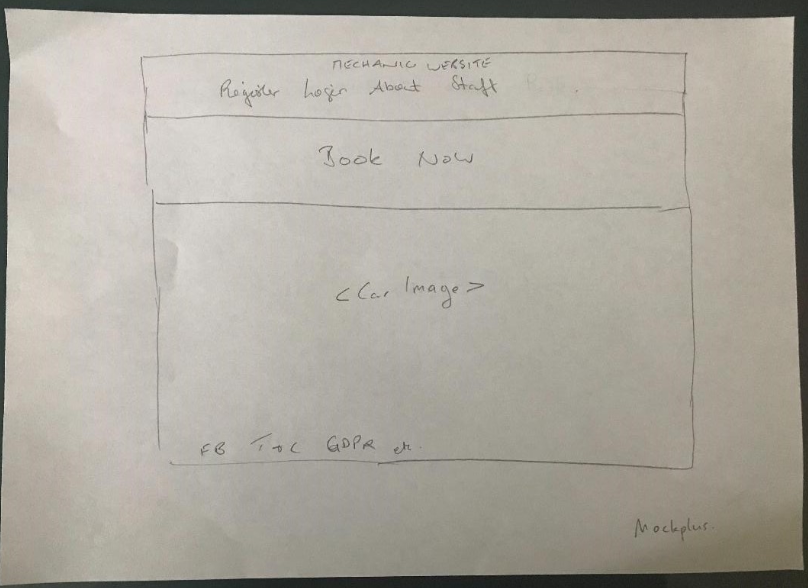


Figure 2. Simple Welcome page sketch



After the paper sketches were done and some ideas put down the drawings were transferred to Mockplus as a prototype project.

The Welcome page was eventually built close to the prototype as was the blog. However the design was updated based on payment being taken when users register. This was originally to be separated i.e. Register on one page and if you decided on a subscription this could be done under another link. However it made more sense to put the both together and make it more “compact”.

Figure 3. Mockplus Welcome Page

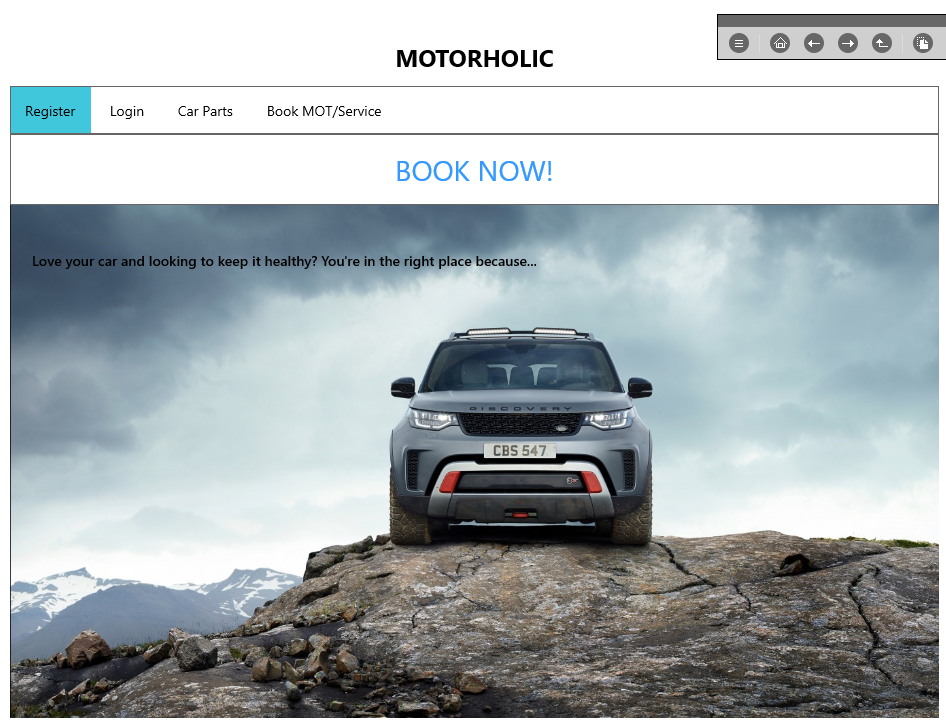
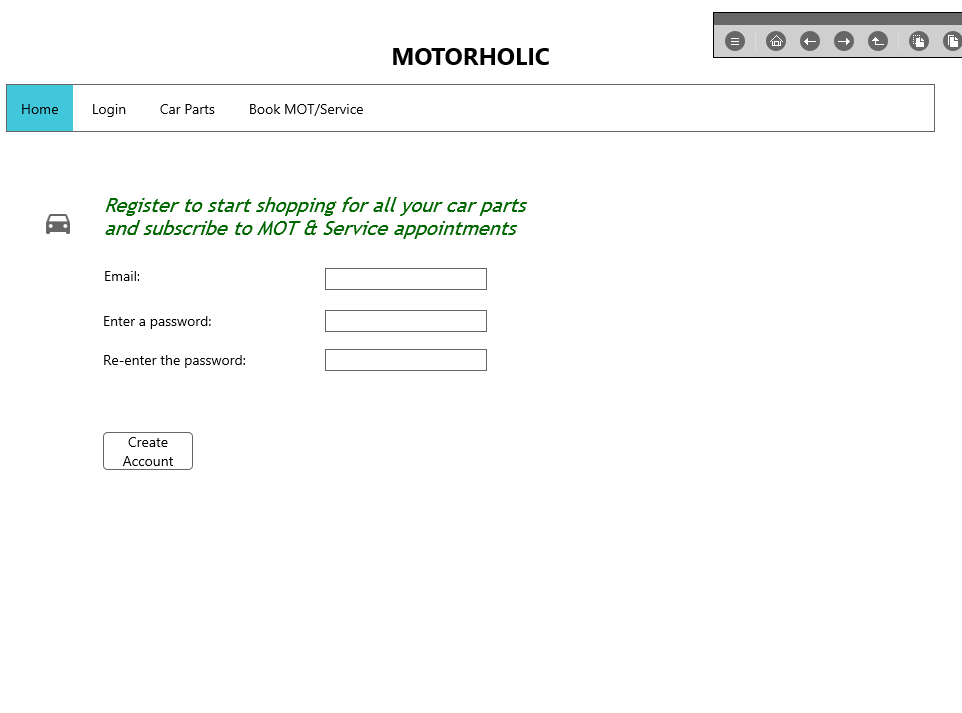


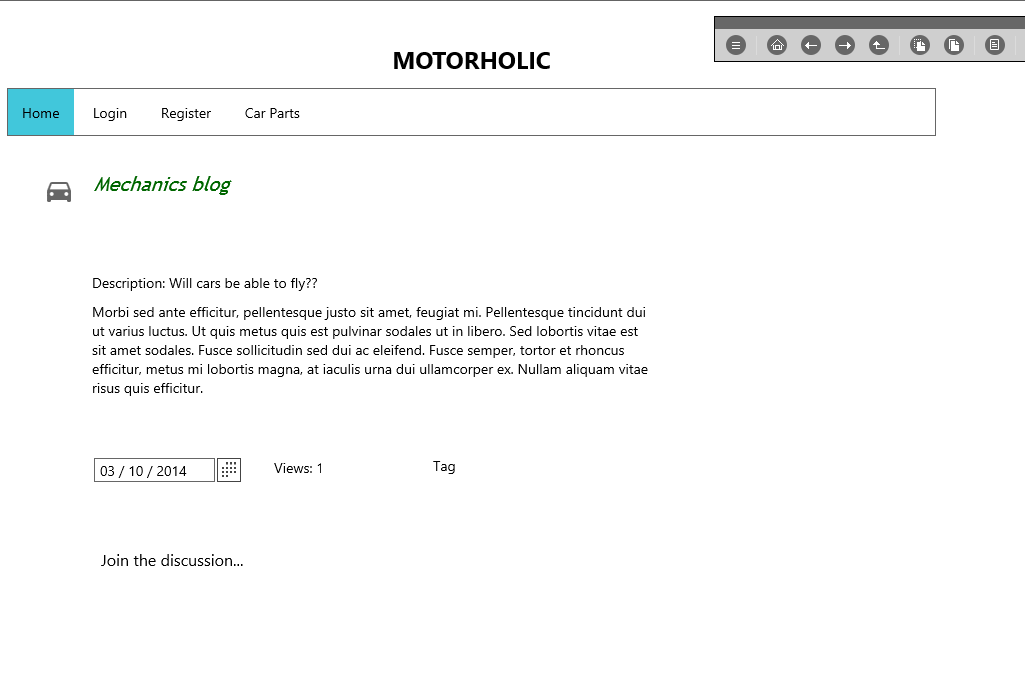
Figure 4. Mockplus Register Page



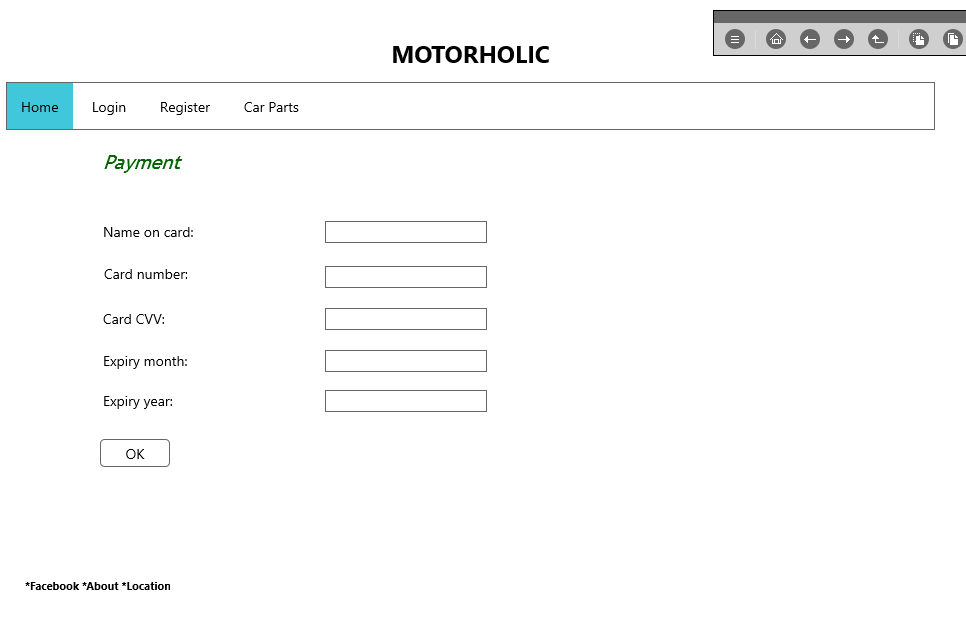
**Figure 5. Mockplus Login Page**

****

**6. Mockplus Blog Page**



**7. Mockplus Payment Page**



## Django Project Start Up

The very first steps to start the project was to create a directory (mkdir mechanic) and create a virtual environment. Once the virtual environment was activated Django was installed. Using the Django admin command a new project was created and the first few apps that were going to be worked on created as well. It made sense to start with the “home” and “accounts” app. The home app would be used for designing the navigation on the Welcome page and linking in some flat pages. The accounts app was hugely important because this was the app providing the user authentication across several Website pages.

Bootstrap features heavily in the Website and is a brilliant front-end framework to use to make Websites look professional. The Django Bootstrap forms was therefore also installed.

The settings.py file was updated so that Django could find the root templates directory. Additionally the home app was add to INSTALLED\_APPS at this stage.

1. Install Django under virtual environment

Machine generated alternative text:
(enu) install däango 
01 lecting däango 
Using cached https ://F iles-pythonhosted.org/packages/F8/Ic 
01 lecting pytz (From däango) 
Using cached https ://F iles-pythonhosted.org/packages/3Ø/4e 
Installing collected packages: pytz. däango 
uccessFuIIy installed däango—l .11 .15 pytz—2Ø18.5 
(enu) 

1. Start a new project with Django

Machine generated alternative text:
: startproäect mechanic 
: \mechan 

1. Check server

Machine generated alternative text:
: manage . py runseruer 
erForming system checks... 
ystem check identif ied no issues (O silenced). 
ou have 13 unapplied migration(s). Your project may not wm 
un ' python manage . py migrate' to apply them. 
ugust 11. 2018 - 
jango version I .11 .15. using settings ' mechanic -settings' 
tarting development server at http://127.Ø.Ø.I:8øøø/ 
uit the server with CTRL—BREAK. 

1. Start the first apps – home and accounts

Machine generated alternative text:
: manage . py startapp home 
: manage . py startapp accounts 
: \mechan ic \mechan ic 

1. Bootstrap install (under virtual environment)

Machine generated alternative text:
(enu) C: install 
01 lecting 
Using cached https ://F iles-pythonhosted.org/packages/F2/bd/6bFF32d77e 
Installing collected packages: 
uccessFuIIy installed .1.ø 
(enu) 

1. INSTALLED\_APPS

Machine generated alternative text:
30 
31 
32 
33 
34 
35 
36 
37 
38 
39 
41 
# Application definition 
INSTALLED_APPS = [ 
django. contrib. admin ' 
django. contrib. auth' , 
django. contrib. contenttypes ' 
django. contrib. sessions 
django. contrib. messages ' 
django. contrib. staticfiles ' 
home ' 

1. TEMPLATES update

Machine generated alternative text:
TEMPLATES = [ 
BACKEND' : 
' django. template. backends . django. DjangoTemp1ates ' , 
DIRS': [os. path . join 
(BASE DIR, •templates 
'APP DIRS•: lrue, 
OPTIONS : 
context_processors ' • 
django. template . context_processors. debug ' 
django. template . context_processors. request ' , 
django. contrib. auth . context_processors . auth ' 
django. contrib. messages . context_processors . messages ' 

1. Browser view

Machine generated alternative text:
Motorholics 
Welcome to Motorholics Website 
Home to the famous car blog 
C Wilde - Project 3 
Home 
Motornolics Blog 
Car Shop 
Register 
Login 
Logout 

**Figure 8. Django Project Layout**

Machine generated alternative text:
FOLDERS 
env 
accounts 
templates 
db.sqIite3 
manage.py 

An index file was created so that the Welcome message to the site could be highlighted for the landing/base page

1. Index.html file – for Home app

Machine generated alternative text:
base. html 
index. html 
views.py 
2 
4 
extends base. html' X} 
block content X} 
<h2>We1come to Motorholics Website 
<p>Home to the famous car blog 
endblock X} 

## The django Accounts app

It was decided that that registration would be via email and password validation. The username that most Websites asked for nowadays would be the email address. Therefore a “backend.py” file had to be created as well so the custom email authentication can be used instead of the default.

Models.py

The first step is to create the models and for the accounts app an AccountUserManager class was created first.

To create user objects Django should then refer back to the AccountUserManager class. Adding the User class achieved this. Under settings.py the standard AUTH\_USER\_MODEL was updated to reflect the accounts.User. This could then be used acrossed the Website to authenticate users according to the AccountUserManager class in the models.py file. The AccountUserManager requires that the user’s user name is the email address of that person (username=email).

1. AccountUserManager class

Machine generated alternative text:
3 
4 
6 
7 
8 
9 
10 
12 
14 
16 
17 
18 
19 
20 
22 
24 
26 
27 
28 
29 
30 
from 
from 
from 
from 
future 
import unicode literals 
django.db import models 
django. contrib. auth . models import AbstractUser, 
django. utils import timezone 
UserManager 
# Create your models here. 
class AccountUserManager(UsenManager 
def _create user (self, username, email, password, 
is_staff, is_superuser, 
now = timezone. now 
if not email : 
raise VaLueError(' The given username must be set' ) 
self. normalize email (email) 
email 
self. model (username=email, email=email, 
user — 
is staff=is staff, is active=True, 
IS superuser=ls_superuser, 
date_joined=now, 
set _ password (password) 
user. 
. save (using—self. 
db) 
user 
return user 
class User(AbstractUser) 
objects 
AccountUserManager 

1. Settings.py

Machine generated alternative text:
30 
32 
34 
36 
37 
38 
39 
43 
# Application definition 
INSTALLED_APPS = [ 
django. contrib. admin ' 
django. contrib. auth' , 
django. contrib. contenttypes ' 
django. contrib. sessions 
django. contrib. messages ' 
django. contrib. staticfiles ' 
home ' 
AUTH USER MODEL 
MIDDLEWARE 
' accounts. User 
dianeo. middleware. security. SecurityMidd1ewal 

1. Backend.py file

Machine generated alternative text:
base.html 
backends.py 
2 
4 
6 
7 
8 
9 
10 
11 
12 
13 
14 
15 
16 
17 
18 
19 
20 
21 
22 
23 
from models import User 
class EmailAuth(object) : 
def 
def 
authenticate (self, email —None, password=None) 
try: 
user = User. objects. 
if user. check_password(password) 
return user 
except User. DoesNotExist : 
return None 
get_user(self, user_id): 
try: 
user = User. objects. 
if user. is active: 
return user 
return None 
except User. DoesNotExist : 
return None 

Views.py

The views.py file was then updated. This is the business logic to link the templates to the models created. The views.py file used the POST method which enables the HTTP communication between the user (client) and the server, in this case using the localhost.

Here is where the Register.html file could be referred to and so the next step was to create the Register.html file and save this file under the templates folder where eventually all html templates would reside.

1. Views.py file

Machine generated alternative text:
from 
from 
from 
from 
from 
from 
coding: utf-8 -*- 
future 
import unicode literals 
django. contrib import messages, auth 
django. core . urlresolvers import reverse 
django. shortcuts import render, redirect 
django. template . context_processors import csrf 
accounts . forms import UserRegistrationForm 
def register(request) : 
if request. method 
'POST': 
UserRegistrationForm( request. POST) 
form 
o: 
if form. is valid 
form. save 
user — 
auth. authenticate (email=request. POST. get( ' email ' ) , 
password—request . POST. get( ' passwordl ' )) 
if user: 
. success(request, "Thank you. You have successfully registered") 
messages 
return redirect(reverse( profile 
else 
(request, 
"Sorry, we are unable to log you!" ) 
messages 
. error 
else 
UserRegistrationForm( ) 
form 
{'form': form} 
args 
update(cs of (request)) 
args . 
return render(request, register . html ' 
args) 

1. Register.html file

Machine generated alternative text:
settings-w 
forms. py 
views.py 
reg i ster.htm I 
2 
4 
6 
7 
8 
9 
10 
12 
extends "base. html " X} 
load bootstrap_tags X} 
block content X} 
(form role="form" method="post" action=" {X ur1 
<1egend>Create a new account<," legend> 
{X csrf_token X} 
form I 
as_bootstrap 
(div class="form-group"> 
< / form> 
endblock % 
' register ' 
(button type="submit" class="btn btn-primary">Create an account</button> 

After adding the accounts app to the settings.py file (and additionally the Django bootstrap forms) the next step was to migrate the model changes which would update the SQLite database.

A profile.html file was then created for the logged in user. This would feedback to the user that they are logged in successfully on the Website. Note this was updated later once Stripe was installed and used to adjust the user profile.

After each html file was added the urls.py file was updated. This file tells Django how to look-up each file via it’s url.

1. Migrating the changes

Machine generated alternative text:
mana e -py makemigrations 
igrations For ' accounts' • 
accounts \migrat ions _ in it iaI . py 
Create model User 
: manage . py migrate 
perations to perform: 
Apply all migrations: accounts. admin. auth. content types. 
unning migrations: 
sessions 
Applying 
Applying 
Applying 
Applying 
Applying 
Applying 
Applying 
Applying 
Applying 
Applying 
Applying 
Applying 
Applying 
Applying 
content types .øøøl_initial. 
content types . 
auth. "01 _ initial. 
auth . 
auth. 
auth . . 
auth. 
auth . 
auth . 
auth . 
accounts . øøøl_initial. 
admin . 
admin . 
sessions . øøøl_initial. 

1. Profile.html file

Machine generated alternative text:
reg ister.htmL 
views.py 
profile. h tm 
2 
4 
extends "base. html " X} 
block content X} 
<h2>Success ! </h2> 
<p>You are logged in as {{ user. email } } </p> 
endblock % 

1. urls.py file

Machine generated alternative text:
urlpatterns 
url(r' Aadmin/', 
admin . site. urls) 
url(r' , home views .get_index, name=' index' ) , 
url(r' Aregister/$' , 
accounts_views . register, name= ' register ' ) 
url(r' Aprofile/$• , 
accounts _ views . profile, name= profile ) 

Next the views.py needed to be updated for both login/logout functions. The code was updated to reflect that if you have successfully logged in then you're redirected to the user profile page. To login a form was created and set under the form.py file. An html template was then added for the logged in view on the Website.

No new html template was required for the logout as this logic would just redirect back to the index view.

1. Login logic on view.py file

Machine generated alternative text:
from django. template . context_processors import csrf 
from accounts . forms import UserRegistrationForm, UserLoginForm 
from django. contrib. auth . decorators import login_required 
def register(request) : 
if request. method 
'POST': 
UserRegistrationForm( request. POST) 
form 
if form. is_valid(): 
form. save 
user — 
auth. authenticate (email=request. POST. get( ' email ' ) , 
password—request . POST. get( ' pas swordl ' )) 
if user: 
. success(request, "You have successfully registered" 
messages 
return redirect(reverse( profile 
else 
(request, 
"unable to log you in at this time! " 
messages 
. error 
else 
UserRegistrationForm( ) 
form 
{ • form': form} 
args 
update(cs of (request)) 
args . 
return render(request, 
' register . html ' , 
' /login/ ' ) 
def profile(request) 
return render(request, 
profile. html • ) 
args) 

1. Login form (forms.py)

Machine generated alternative text:
class UserLoginForm(forms. Form 
email = forms . EmailFie1d 
password = forms . CharFie1d(widget=forms. Passwordlnput) 

1. Logout logic (views.py)

Machine generated alternative text:
def logout(request): 
auth . logout( request) 
. success(request, 'You have successfully logged out') 
messages 
return redirect(reverse( index')) 

1. urls.py file

Machine generated alternative text:
urlpatterns 
url(r' Aadmin/', 
admin . site. urls) 
url(r' , home views .get_index, name=' index' ) , 
ur1(r'Aregister/$' , 
accounts_views . register, name= ' register ' ) 
url(r' Aprofile/$• , 
accounts _ views . profile, name= profile ) 
url(r' Alogin/$', 
accounts _ views. login, name= login') 
url(r' Alogout/$' , 
accounts _ views . logout, name= ' logout ' ) , 

1. Udated base.html file with links

Machine generated alternative text:
<div class="container"> 
(div class="masthead"> 
<u1 class="nav nav-pills pull-right" > 
href="#" / /1i> 
href="#" Notorholics 
>Car Shop</a></li> 
" {X ur1 ' register' 
href= 
"{X ur1 'login' In</a></li> 
href= 
" {X ur1 logout' " >Log 
href= 
<hl><span class="fui-settings-16 muted" 
</div> 

Stripe

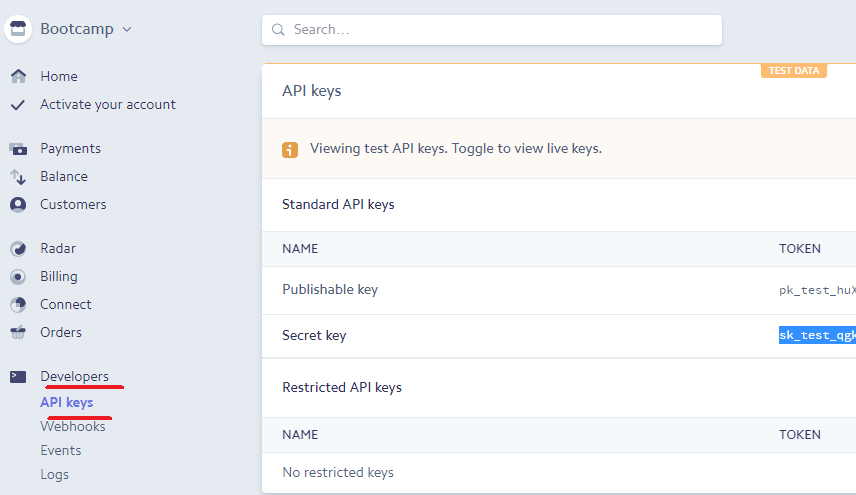
Stripe was used for the subscription to the Motorholics blog/forum requirements.

Firstly using pip, Python’s install package command, Stripe was installed under the virtual environment. A Stripe account on the stripe Website was then created so that configurations on their API could be done.

1. Pip install Stripe (under virtual environment)

Machine generated alternative text:
Admin cmd prompt 
(enu) install stripe 
30 Ilecting stripe 
Downloading https : //F 
løøz :uuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuu: 215kB 416kB/s 
301 lecting requests python_uersion < "3 -W' (From stripe) 
Using cached https://FiIes-pythonhosted.org/packages/65/47/7eØ2164a2a3db5Øed6d8a6abId6d6Øb69c4c3FdF57a284257925dFc12bda/requests—2.19.1 
301 lecting (From requests python_uersion < "3 
• py3 
-py2 
—none—any. whI (208kB) 
. . whI 
Down loading https : //Files-pythonhosted.org/packages/16/IF/5Ød729cIØ4b21cIØ42aa5156Øda6141dIcab476ba7Ø15d92b2111c8db841/certiF i—2ø18 .8 .13—py2 whI 
løøz :uuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuu: 153kB 1.3MB/s 
301 lecting urIIib3<I .24. .21 .1 (From requests python_uersion < "3 
Using cached https://FiIes-pythonhosted.org/packages/bd/c9/6Fdd99ØØ19Ø71a4a32a5e7cb78aId92c53851eF4F56F62a3486e6a7d8FFb/urIIib3—I.23—py2.py3—none—any.whI 
301 lecting chardet<3 .ø.2 (From requests python_uersion < "3 
Using cached https://FiIes-pythonhosted.org/packages/bc/a9/ØIFFebFb562e4274b6487b4bbIddec7ca55ec751Øb22e4c51F14Ø98443b8/chardet—3.Ø.4—py2.py3—none—any.whI 
301 lecting idna<2 (From requests python_uersion < "3 
Using cached https://FiIes-pythonhosted.org/packages/4b/2a/Ø276479a4b3caeb8a8cIaF2F8e4355746a97FabØ5a372e4a2c6a6b876165/idna—2.7—py2.py3—none—any.whI 
301 lecting -3.4; extra "security" (From requests python_uersion < "3 
Down loading https : . whI 
løøz :uuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuu: 1.1MB 1.2MB/s 
301 lecting extra "security" (From requests python_uersion < "3 
Using cached https://FiIes-pythonhosted.org/packages/96/aF/9d29e6bd4Ø823Ø61aea2eØ574ccb2FcF72bFd613Øce53d32773ec375458c/pyOpenSSL—18.Ø.Ø—py2.py3—none—any.whI 
301 lecting cFFi! (From -3.4; extra python_uersion < "3 
Using cached https://Fi1es.pythonhosted.org/packages/3a/84/9d1bF29b212853Ø8?318696c38241a3aF65b33ea2edaabad11F26826ecF4/cFFi-1.11.5-cp2?-cp2?m-win32.wh1 
301 lecting enum34; python_uersion < "3" (From -3.4; extra python_uersion < "3 
Using cached https://FiIes-pythonhosted.org/packages/c5/db/e56e6b4bbac7c4aØ6deIc5Øde6FeIeF381ØØ18ae11732a5ØF15F62c7dØ5Ø/enum34—I.1.6—py2—none—any.whI 
301 lecting -4.1 (From -3.4; extra python_uersion < "3 
Using cached https://FiIes-pythonhosted.org/packages/67/4b/141a5811Ø4bIF6397bFa78ac9d43d8ad29a7ca43ea9Øa2d863Fe3Ø56e86a/six—I.11.Ø—py2.py3—none—any.whI 
301 lecting (From -3.4; extra python_uersion < "3 
Using cached https://FiIes-pythonhosted.org/packages/ea/cd/35485615F45F3Øa51Ø576FIa56dIeØa7ad7bd8ab5ed7cdc6ØØeF7cdØ6222/asnIcrypto—Ø.24.Ø—py2.py3—none—any.whI 
301 lecting ipaddress; python_uersion < "3" (From -3.4; extra python_uersion < "3 
Using cached https://FiIes-pythonhosted.org/packages/Fc/dØ/7Fc3a811eØIId4b388be48aØe381db8d99ØØ42dF54aa4eF4599a31d39853/ipaddress—I.Ø.22—py2.py3—none—any.whI 
301 lecting pycparser (From cFFi! -3.4; extra python_uersion < "3 
Installing collected packages: certifi- urIIib3. chardet. idna. pycparser. cFFi. enum34. six. asnlcrypto. ipaddress. cryptography. pyOpenSSL. requests. stripe 
(146 
(1. 
installed asnIcrypto—Ø.24.Ø certifi—2ø18.8.13 cFFi—I .11 .5 chardet—3.Ø.4 cryptography—2.3.I enum34—I.I.6 idna—2.7 ipaddress—l .ø.22 pyOpenSSL—18.Ø.Ø pycp 
(enu) 

1. Stripe Website



1. Updated settings.py file with Stripe keys

Machine generated alternative text:
apps. py 
backends.py 
forms. py 
models.py 
tests. py 
views. py 
init_.py 
settings .py 
urls.py 
80 
81 
82 
83 
84 
85 
86 
87 
88 
89 
90 
91 
92 
93 
94 
WSGI APPLICATION 
django. contrib. messages . context_processors . messages ' 
' mechanic . wsgi . application ' 
# Stripe environment variables 
STRIPE PUBLISHABLE = os.getenv( 'STRIPE PUBLISHABLE' , 
STRIPE SECRET = os.getenv('STRIPE SECRET' , 
' sk_test_qgkn8rHXvHGEtPaqreOCeXi7 ' ) 
Database 
httDs : / /docs . di angooroi ect . com/en/l.11/ref/settings/#databases 

Stripe was added as an import to the top of the views.py file (import stripe) including arrow API which helps significantly with date formats. The forms.py was updated to reflect the relevant fields for payment such as credit\_card\_number, expiry\_month and stripe\_id.

Regarding the latter ‘stripe\_id’ in order to retain this the User class under the models.py file was updated:

Stripe\_id = models.CharField….

Again once all changes are made to the model a migration was done to update the database.

1. Forms.py file with fields to accommodate stripe payment

Machine generated alternative text:
documentation 
accounts 
'Nov', 'Dec' 
password2' , ' stripe_id ' 
incex.html 
X default.html 
2 
4 
6 
7 
8 
9 
10 
11 
12 
13 
14 
15 
16 
17 
18 
19 
20 
21 
22 
23 
24 
25 
26 
27 
28 
29 
30 
31 
32 
33 
34 
35 
from django import forms 
from django. contrib. auth . forms import UserCreationForm 
from accounts . models import User 
from django. core . exceptions import ValidationError 
class UserRegistrationForm(UserCreationForm) 
migrations 
init_.py 
admin.py 
apps. py 
backends.py 
forms .py 
models.py 
tests. py 
views. py 
init_.py 
settings. py 
urls.py 
wsgl.py 
templates 
flatpages 
< > default.html 
base.html 
> index.html 
login.html 
< > profile.html 
register.html 
db.sqIite3 
manage.py 
MONTH ABBREVIATIONS 
'Jan', 'Feb', 
'Mar', 'Apr', 
'May , 
'July', 'Aug', 
'Sept 
'Oct', 
' June 
ABBREVIATIONS, 1)) 
MONTH CHOICES 
[ (i, i) for i in range(2015, 206)] 
YEAR CHOICES 
credit_card_number = forms .CharFie1d(1abe1='Credit card number') 
forms. Security code (CVV) ) 
cvv — 
expiry_month = forms . , 
choices=MONTH CHOICES) 
forms. ChoiceF , 
choices=YEAR CHOICES) 
exp 1 ry_year - 
stripe_id 
passwordl 
password2 
class Meta: 
forms . CharF ield(widget=forms . Hiddenlnput ) 
def 
forms . CharF ield( 
label= 'Password , 
widget=forms . Passwordlnput 
forms . CharF ield( 
label= 'Password Confirmation ' 
widget=forms . Passwordlnput 
model 
User 
[ ' email • , 
fields 
' passwordl ' 
[ username ' 
exclude 
clean_password2(se1f) • 
asswordl 
set 
. cleaned data. 
asswordl 

1. Models.py file updated to retain stripe id per user

Machine generated alternative text:
24 
25 
26 
27 
28 
29 
class User(AbstractLlser 
stripe_id = models . 
objects 
AccountUserManager 
default=• ' ) 

1. Migrate models.py changes

Machine generated alternative text:
: manage . py makemigrations 
igrations For ' accounts' • 
accoun ts \migrat ions _user_s tripe _ id . py 
Add Field stripe_id to user 
: manage . py migrate 
perations to perform: 
Apply all migrations: accounts. admin. auth. 
unning migrations: 
Applying accounts 
: \mechan ic \mechan 
content types. 
F Iatpages. 
sessions. 
sites 

## Stripe.js file

JavaScript was used to help validate the stripe payment form.

The .js file was saved under the “static” folder. A new folder was created simply called “js” and the stripe.js file stored under here.

Using JavaScript the validations confirmed that if the various fields (card number, expiry date, cvc) were completed and correct then a stripe token would be created and a stripe id returned otherwise the user would receive an error.

After this was established the register.html file was updated to link in the JavaScript code found in stripe.js by adding the “head\_js” block to the top of the file.

1. Stripe.js file

Machine generated alternative text:
FOLDERS 
documentation 
accounts 
stripe.js 
templates 
db.sqIite3 
manage.py 
index.html 
default.html 
X base.html 
settings-py 
2 
4 
6 
7 
8 
9 
10 
11 
12 
13 
14 
15 
16 
17 
18 
19 
20 
21 
22 
23 
24 
25 
26 
27 
$ ( "#register- form") 
var form 
this ; 
var card 
number : 
exp"onth : 
expYear : 
cvc: 
val(), 
'v ) . val( ) 
.val(), 
$("#id_cvv ) 
.val() 
) . attr( " disabled" , 
true 
Stripe. createToken(card, function(status, response) { 
if (status 
= 200) { 
console. Log(status, response); 
$("#credit-card-errors") . hide() 
. val (response. id) • 
o; 
form. submi t 
else { 
"#stripe-error-message") . text (response. error . message) 
"#credit -card -errors" ) . show( ) 
$("#validate_card_btn") faLse) 
return false; 

1. Register.html file with head\_js block

Machine generated alternative text:
FOLDERS 
documentation 
accounts 
stripe.js 
templates 
flatpages 
< > default.html 
base.html 
> index.html 
login.html 
< ) profile.html 
register. html 
db.sqIite3 
manage.py 
index.htm X 
default.html X 
base.html 
settings-w 
stripe.js 
reg i ster.htm I 
2 
4 
6 
7 
8 
9 
10 
11 
12 
13 
14 
15 
16 
17 
18 
19 
20 
21 
22 
23 
24 
25 
26 
27 
28 
forms. py 
value="Va1idate 
models.g 
{ % extends "base. html " X} 
{ % load bootstrap_tags X} 
{ % load staticfiles X} 
{ % block head_js X} 
<script src="https://js.stripe.com/v2/" type="text/javascript"></script> 
<script type="text/javascript"> 
//<! CCDATA[ 
Stripe . publishableKey 
publishable 
</script> 
<script type="text/javascript" src="{% static js/stripe.js' 
{ % endblock X} 
{ % block content X} 
<form role="form" method="post" id="register-form" action="{% ur1 register' 
(div id="credit-card-errors" style="dispLay:none"> 
<div class="alert-message block-message error" 
<1egend>Create a new account</legend> 
{X csrf_token X} 
{{ form las_bootstrap Y} 
(div class="form-group"> 
<input class="btn btn-primary" 
Credit Card "> 
{X endblock X} 
id="validate card btn" 
name= 
" commit " 
type=" submit" 

## Subscribing Monthly

As mentioned the Website is to take a subscription from users who which to contribute to the Motorholics blog and forum.

In order to do this a Stripe “product” was created on the Stripe Website which was configured to take a £9.99 fee per month.

The views.py file was updated to reflect a POST method based on a valid form that obtains the email, card and the Stripe monthly product plan. The stripe\_id becomes the customer.id and a subscription based on adding 4 weeks. If validation fails then the users receives an error and feedback that the card was not accepted.

Note csrf is used to avoid malicious 3rd party intervention. The token code producing such a difficult random number that 3rd parties would not be able to guess and then dupe the user’s requested site with.

1. Stripe product plan

Machine generated alternative text:
Bootcamp 
Home 
Activate your account 
Payments 
Balance 
Customers 
Radar 
Billing 
Invoices 
Subscriptions 
Products 
Coupons 
Settings 
Connect 
Orders 
Developers 
Viewing test data 
Q Search... 
bootcamp 
{9.99 Gap / month 
+ Add pricing plan 
Details 
ID 
Created 
Name 
Statement descriptor 
Pricing plans 
Monthly Subscription 
0.99 Ggp / month 
prod_CuMxv8eKRgT4N 
2018/05/22 11:59 
boot cam p 
None 

1. Updated views.py file

Machine generated alternative text:
7 
8 
9 
10 
11 
12 
13 
14 
15 
16 
17 
18 
19 
20 
21 
22 
23 
24 
25 
26 
27 
28 
29 
30 
31 
32 
33 
user. save 
34 
35 
36 
37 
38 
39 
from 
from 
from 
from 
accounts . forms import UserRegistrationForm, UserLoginForm 
django. contrib. auth . decorators import login_required 
django. template . context_processors import csrf 
django. conf import settings 
import datetime 
import stri 
import arrow 
stripe. api_key = settings . STRIPE SECRET 
def register(request) : 
if request. method 
'POST': 
UserRegistrationForm( request. POST) 
form 
if form. is_valid(): 
try: 
stripe. Customer. create 
customer 
email=form. cleaned_data[ email ' ] 
card—form. ' stripe_id ' ] , 
plan='REG MONTHLY' , 
customer : 
# this is currently the card token/id 
() . replace(weeks=+4) . datetime 
user — 
user 
user 
user — 
form. save 
. stripe_id 
customer. id 
. subscription_end 
arrow. now 
auth. authenticate (email=request. POST. get( ' email ' ) , 
password—request . POST. get( ' passwordl ' )) 
if user: 
auth . login(request, user) 
messages . success (request, "You have successfully registered") 

1. Updated User class (models.py)

Machine generated alternative text:
20 
21 
22 
23 
24 
25 
26 
27 
28 
29 
set _ password (password) 
user. 
. save (using=seLf. 
db) 
user 
return user 
class User(AbstractUser) 
stripe_id = models . default=• ) 
subscription_end = models now) 
objects = AccountUserManager 

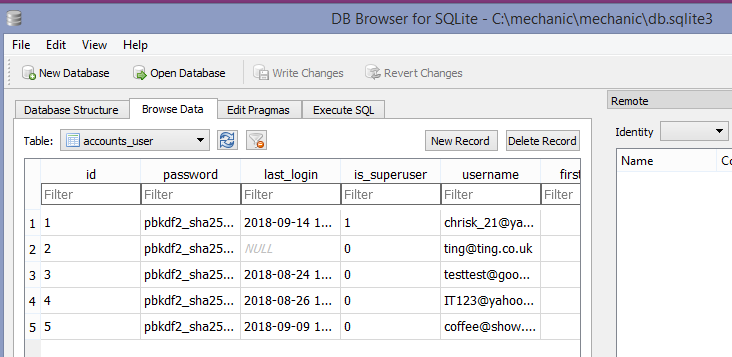
1. Migration of changes

Machine generated alternative text:
: manage . py makemigrations 
igrations For ' accounts' • 
accoun ts \migrat ions _user_subscript ion _end . py 
Add Field subscription_end to user 
: manage . py migrate 
perations to perform: 
Apply all migrations: accounts. admin. auth. content types. 
unning migrations: 
Applying accounts 
: \mechan ic \mechan 
F Iatpages. 
sessions. 
sites 

1. Customer monthly charges (Stripe Website)

Machine generated alternative text:
Invoices 
AMOUNT 
{9.99 
{9.99 
{9.99 
GBP 
GBP 
GBP 
Paid 
Paid 
Paid 
INVOICE NUMBER 
407E23s-0003 
407E23s-0002 
407E23s-0001 
CUSTOMER 
coffee@showco.u k 
coffee@showco.u k 
coffee@showco.u k 
PAYMENT DUE 
CREATED 
2018/08/17 
2018/07/17 
2018/06/17 

1. Registered accounts\_user view on SQLite database



Lastly for users to cancel a subscription the views.py file was updated. In order to cancel the Stripe ID that was previously saved on registration is retrieved and then cancel subscription function entered based on the “at\_period\_end=True”. The urls.py and profile.html were subsequently updated.

So now once users log in they will not see the register or login tabs.

Once users logout however the Register and Login tabs will appear on the Website.

1. Updated views.py for cancellation

Machine generated alternative text:
update(cs (request)) 
args . 
return render(request, register . html ' 
args) 
def cancel _ subscription(request) 
try: 
stripe . Customer. retrieve( request. user. stripe_id) 
customer - 
customer. cancel subscription(at_period_end=True) 
except Exception, 
messages. error(request, e) 
return redirect( profile ) 

1. Updated urls.py file

Machine generated alternative text:
from 
from 
from 
from 
from 
django. conf. ur1s import ur1 
django. contrib import admin 
accounts import views as accounts _ views 
home import views as home views 
django. conf. ur1s import ur1, include 
urlpatterns 
url(r' Aadmin/', 
admin . site. urls) 
url(r' , home views .get_index, name=' index' ) , 
url(r' Aregister/$' , 
accounts_views . register, name= ' register ' ) 
url(r' Aprofile/$• , 
accounts _ views . profile, name= profile ) 
url(r' Alogin/$', 
accounts _ views. login, name= login') 
url(r' Alogout/$' , 
accounts _ views . logout, name= ' logout ' ) , 
url(r' Apages/', 
include( django. contrib. flatpages . urls ' )) 
url(r' Acancel_subscription/$ , 
accounts views . cancel_subscription 
name= 
cancel subscription ' 

1. Updated profile.html file

Machine generated alternative text:
Views..y• 
profile.html 
{ % extends "base. html " X} 
{ % block content X} 
<h2>Success ! </h2> 
<p>You are logged in as {{ user. email } } </p> 
<p>Your stripe id is { { user. stripe_id } } 
<h2>Subscription Details</h2> 
<p>You have {{ user. subscription_end Itimeuntil}} left on your 
subscription</p> 
<p>To cancel your subscription click 'Cancel Subscription' below</p> 
<a href="{% ur1 "cancel _ subscription" X}" 
onclick=" return cancelSubscriptionCheck() 
class="btn btn-danger">Cance1 Subscription</a> 
(script type="text/javascript"> 
function cancelSubscriptionCheck() { 
if (confirm( 'Are you sure you want to cancel?')) { 
return true; 
return false; 
</script> 
{ % endblock X} 

## flatpages

Flatpages was used for the “about us” page. Flat HTML pages are displayed as they are stored which in this case is the Django Admin site. This is an easy way to simply format pages such as about us, security notifications, certain policies etc.

Since these are created on the Django Admin site the first step was to create a superuser for the Admin login.

The settings.py file was updated so that the Django flatpages feature was set under the INSTALLED\_APPS and the SITE\_ID set once the correct ID sequence was checked under SQLite database. Due to these changes a migration was then done. The middleware was then updated for the flatpages routing.

The flatpage default html template was saved under a new flatpages folder in the templates project root folder. Once the extend base.html code was added to the top the flatpage formatting was applied across all the pages.

1. Flatpages folder structure

Machine generated alternative text:
FOLDERS 
env 
accounts 
templates 
flatpages 
< > default.html 
< > base.html 
< > index.html 
login.html 
< > profile.html 
register.html 
db.sqIite3 
manage.py 
incex.html 
default.html 
{ % extends base . html' X} 
{ % block content X} 
<h2>{{ flatpage.title 
{{ flatpage. content } } 
{ % endblock % 

Then it was just a matter of adding the flatpage URL to the urls.py file and then new flatpages could be created quite easily on the Django Admin site.

1. INSTALLED\_APPS update for Flatpages

Machine generated alternative text:
INSTALLED_APPS = [ 
django. contrib. admin ' 
django. contrib. auth' , 
django. contrib. contenttypes ' 
django. contrib. sessions 
django. contrib. messages ' 
django. contrib. staticfiles ' 
django_forms_bootstrap ' 
django. contrib. sites ' 
django. contrib. flatpages ' 
home ' 
accounts ' 

1. SITE\_ID added

Machine generated alternative text:
# SECURITY WARNING: 
DEBUG = True 
ALLOWED HOSTS 
don't run with 
# Application definition 
SITE ID = 2 
INSTALLED_APPS = [ 
django. contrib. admin ' 
django. contrib. auth' , 
django. contrib. contenttypes ' 
django. contrib. sessions 
django. contrib. messages ' 
django. contrib. staticfiles ' 
django_forms_bootstrap ' 
django. contrib. sites ' 
django. contrib. flatpages ' 
home ' 
accounts ' 

1. Migrate changes – flatpages

Machine generated alternative text:
: manage . py migrate 
perations to perform: 
Apply all migrations: accounts. admin. auth. 
unning migrations: 
Applying sites .øøøl_initial. 
Applying Flatpages .0001 _ initial. 
Applying sites 
: \mechan ic \mechan ic 
content types. 
F Iatpages. 
sessions. 
sites 

1. Middleware updated

Machine generated alternative text:
MIDDLEWARE 
django. middleware. security. SecurityMidd1eware ' , 
django. contrib. sessions . middleware. SessionMidd1eware ' 
django. middleware. common . CommonMidd1eware ' 
django. middleware. csrf. Cs ' , 
django. contrib. auth . middleware. AuthenticationMidd1eware ' 
django. contrib. messages . middleware. MessageMidd1eware ' 
django. middleware. clickj acking. XF rameOptionsMidd1eware ' 
django. middleware. security. SecurityMidd1eware ' 
django. contrib. flatpages . middleware . FlatpageFa11backMidd1eware ' , 

1. Update urls.py file – flatpages

Machine generated alternative text:
16 
17 
18 
19 
20 
22 
24 
26 
27 
28 
29 
30 
from 
from 
from 
from 
from 
django. conf. ur1s import ur1 
django. contrib import admin 
accounts import views as accounts _ views 
home import views as home views 
django. conf. ur1s import ur1, include 
urlpatterns 
url(r' Aadmin/', 
admin . site. urls) 
url(r' , home views .get_index, name=' index' ) , 
url(r' Aregister/$' , 
accounts_views . register, name= ' register ' ) 
url(r' Aprofile/$• , 
accounts _ views . profile, name= profile ) 
url(r' Alogin/$', 
accounts _ views. login, name= login') 
url(r' Alogout/$' , 
accounts _ views . logout, name= ' logout ' ) , 
url(r' Apages/', 
include( django. contrib. flatpages . urls ' )) 
• nclude(ar 

1. Django Admin site – adding About us page

Machine generated alternative text:
Django administration 
Home Flat Pages Flat pages Add fiat page 
Add flat page 
URL: 
Title: 
Content: 
Sites: 
/pages/about/ 
Example: '/about/contact/l Make sure to have leading and trailing slashes. 
About us 
Motorholics.com are one of the largest online car and van parts stores in the UK We have a complete 
selection of high quality parts for a wide range of vehicle makes and models offered at the lowest 
prices. 
We conduct our business online and we ship direct from our warehouse, this means we don* have to 
charge you for expenses on office maintenance and store personnel. This way, we pass the savings on to 
you. 
In addition to our car parts shop we are running a blog where local mechanics also join the conversation 
and can provide online advice on various car issues. 
example.com 
localhost 

## The django blog app

The Motorholics Website is to feature a blog that mechanics can access as the administrators and start writing on various car topics. If a user is interested in contributing then they will need to subscribe and become part of the Motorholics “team”. They will not be permitted to start topics however will be allow to contribute to the blogs.

The first step is to use the Django startproject command.

The model was created first and once done then the views and templates could be focused on.

Update blog\models.py with Post class

As mentioned any user wishing to post will need to be an authorised user, so the model needed to specify the custom auth user settings that had previously been created via the accounts app.

In this case the accounts.User.

In order for the mechanics (staff) to add a new topic the Post model was registered on the admin.py file. Once done the Post object appeared on the Django Admin site ready for a new post to be added.

1. Model.py – blog

Machine generated alternative text:
2 
4 
6 
7 
8 
9 
0 
2 
4 
6 
7 
8 
9 
0 
2 
from 
from 
from 
from 
coding: utf-8 -*- 
future 
import unicode literals 
django.db import models 
django. utils import timezone 
django. conf import settings 
class Post(modeLs.ModeL) 
models. ForeignKey( ' accounts . User' ) 
author 
title = models 
TextFie1d() 
content = models . 
. DateT imeF iel True) 
created date 
models 
models. True, null=True) 
published_date 
def publish(self): 
self. publ ished_date 
self. save() 
(self) : 
def unicode 
return self. title 
timezone 

1. Migrate the changes – blog

Machine generated alternative text:
: manage . py makemigrations 
igrations For ' blog' : 
blog\migrat ions _ in it iaI . py 
Create model Post 
: \mechan ic >python manage 
•PY migrate 
perations to perform: 
Apply all migrations: accounts. admin. auth. 
unning migrations: 
Applying blog.øøøl_initial. 
: \mechan ic \mechan 
blog. 
content types. 
F Iatpages. 
sessions. 
sites 

1. Registering Post on the admin.py file

Machine generated alternative text:
views.py 
profile.html 
2 
3 
4 
6 
7 
8 
9 
10 
11 
# coding: utf-8 -*- 
from future 
import unicode literals 
from django. contrib import admin 
from . models import Post 
. register(Post) 
admin . site 
# Register your models here. 

1. Django Admin site – Adding a new Post

Machine generated alternative text:
Django administration 
Home Blog Posts 
The post •Car paint repair" was added successfully. 
Select post to change 
Action: 
Cl POST 
Car paint repair 
1 post 
Oof 1 selected 

## Post List and Post Detail views

A post list and post detail view was added.

For this the views.py file was updated with new car post html files. The urls.py file was updated so Django would be able to find the correct path to the views.

The blog urls were added to the blog app urls.py file this time and then referred back using the ‘include’ wording in the root urls.py file.

In order to link the detail view back to the relevant blog the PK (primary key) was used from the database. This was set in the post\_detail view and results shown the carpostdetail.html template.

If a post ID is not found then the Website displays a 404 error.

1. Views.py file – blog

Machine generated alternative text:
urls.py — blog 
2 
4 
6 
7 
8 
9 
10 
11 
12 
13 
14 
15 
16 
17 
18 
{'post' : post}) 
19 
20 
carpostdetail.' urispy— mechanic X 
coding: utf-8 -*- 
carblogposts.html 
views. py 
from 
from 
from 
from 
from 
def 
def 
future 
import unicode literals 
django. shortcuts import render 
django. utils import timezone 
. models import Post 
django. shortcuts import render, 
post_list ( request) : 
get_object_or_4Ø4 
Post . objects . filter(published_date 
posts 
. order_by( ' -published_date ' ) 
return render(request, " carblogposts. html" , 
id) • 
pk=id) 
post 
return render(request, " carpostdetail . html" , 
Ite=timezone 
{ ' posts' • 
posts}) 

1. Carblogpost.html file with PK reference

Machine generated alternative text:
carblogposts.html — templates X 
</div> 
carpostdetail.html 
carblogposts.html — blog Itemplates 
(div class="c01-md-1Ø col 
<h3>{{ post. title 
(div class=" row" > 
(div class=" 
<p>{{ post. content }}</p> 
<p><button class="btn btn-default" 
</button></p> 
<p>{{ post.published_date 
{X endfor X} 
endblock X} 
onclick= 
post. id 
'Read More 

Additionally a view counter, date and tags were added to the blog lists. After this the changes were migrated.

1. Blog list example

Machine generated alternative text:
When will cars fly? 
PAL-V showed-off its Liberty flying car in Geneva. But will it help get the flying car 
market off the ground? The PAL-V Liberty flying car made an appearance at the . 
Read More 
Aug. 2, 2018, 6 p.m. I Views0 1 Tag Flying 

## Disqus and Images

Disqus is a global communication platform that can be added to Django projects. This was added to the blog for global discussions under a particular existing thread. In order to add another thread to a topic the New Post would still need to be clicked (for authorized members).

Images were added since they made the site posts more eye catching and helped to highlight a topic.

The DISQUS\_WEBSITE\_SHORTNAME was added to the settings.py file.

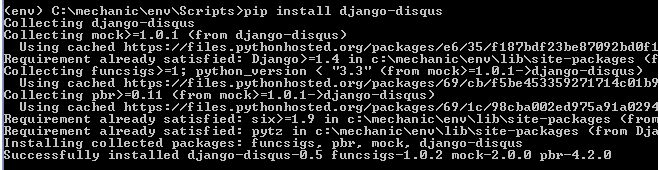
Additionally the blog app was added to the INSTALLED\_APPS under the settings.py file.

The carpostdetail.html file was updated with disqus tags: {{% load disqus\_tags %}.

1. Disqus Website account

Machine generated alternative text:
DISQUS 
Profile 
Account 
Email Notifications 
Web Notifications 
Apps 
Moderation 
Blocking 
A Home 
Apps 
Notifications 
• Channels 
Explore 
Control what apps can access your Disqus account. 
mybootcampblog 
Revoke Access 

1. Disqus pip installation



1. Settings.py file updated – blog

Machine generated alternative text:
# Application definition 
DISQUS WEBSITE SHORTNAME 
21 
SITE ID 
' mybootcampblog ' 

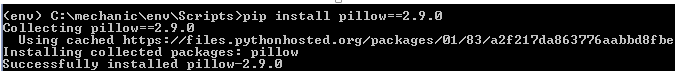
1. DISQUS\_WEBSITE\_SHORTNAME

Machine generated alternative text:
# Application definition 
DISQUS WEBSITE SHORTNAME 
21 
SITE ID 
' mybootcampblog ' 

Images was added with the help of Pillow installation, the Python imaging library.

Once migrated a media folder was created and the MEDIA\_ROOT folder was set under the settings.py file. This is so images can be uploaded dynamically by users on the Website (as opposed to the stored images under the static folder).

1. Pip Pillow installation (under virtual environment)



1. Updated settings.py – blog media path

Machine generated alternative text:
STATIC URL 
' /static/' 
45 
47 
48 
49 
50 
51 
STATICFILES DIRS 
.join(BASE DIR, 
os . path 
" static"), 
MEDIA 
MEDIA 
ROOT 
URL 
os. path . join(BASE DIR, 
/media/ 
media') 

## New Post Form

A post form was designed for the user once they clicked on New Post: 

The forms.py file was therefore updated and a new view created (new\_post). This was then directed back to the new car post form.

The POST method was again used for the new post mechanism. This then redirects the user back to the post details so they can see the results.

1. New post form – blog (with imaging enabled)

Machine generated alternative text:
carblogform.html 
16 
17 
18 
19 
20 
21 
22 
23 
24 
25 
26 
27 
28 
29 
30 
31 
32 
33 
34 
35 
36 
37 
models. py 
id) • 
views.py 
def 
def 
get_object_or_404(Post, pk=id) 
post 
post . views += 1 
post . save() 
return render(request, 
"carpostdetail . html " , 
{ ' post': 
if request. method 
"POST" : 
form = BlogPostForm(request. POST, request . FILES) 
if form. is_valid(): 
post = form. 
post . author 
request . user 
post . published date = timezone 
post . save 
return redirect(post_detail, 
post . pk) 
else 
Car810gForm() 
form 
return render(request, 
{'form' : 
' carblogform. html , 
post}) 
form} ) 

1. New post form view – blog

Machine generated alternative text:
23 
24 
25 
26 
27 
def new_post(request) 
Car810gForm() 
form 
return render(request, 
' carblogform. html , 
{'form' : 
form} ) 

1. Making use of the POST method again

Machine generated alternative text:
Preterences 
carblogform.html 
2 
4 
6 
7 
8 
9 
10 
extends "base. html " X} 
block content X} 
{X load bootstrap_tags X} 
<h1>Add A 
(form method="POST"> 
{X csrf_token X} 
form I 
as_bootstrap}} 
(button type="submit" class="btn 
< / form> 
endblock X} 

1. Non authorized user

Machine generated alternative text:
C localhost:8000/post/new/ 
ValueError at /post/new/ 
Cannot assign "<SimpleLazyObject: <django.contrib.a 
Request Method: 
Request URL: 
Django Version: 
Exception Type: 
Exception Value: 
Exception Location: 
Python Executable: 
Python Version: 
Python Path: 
Server time: 
POST 
http//localhost:8000./post\new/ 
ValueError 
Cannot assign "(SimpleLazyObject: cdjango.ccntrib.äl 
27.15 
• C : zip • , 
sun, 2 sep 2018 +0000 
Traceback 
Switch to copy-and-paste view 
in inner 

Editing was very similar to the post new view. First the object is retrieved or a 404 error shown. Then the CarBlogForm used to make an updated and POST this. The user is then again directed back to the post detail.

1. Views.py edit view - blogMachine generated alternative text:
   def edit_post(request, id): 
   pk=id) 
   post 
   if request. method 
   "POST" : 
   Car810gForm(request.POST, request . FILES, 
   form 
   if form. is_valid(): 
   post = form. 
   post . author 
   request . user 
   post . published date = timezone 
   post . save 
   return post .pk) 
   else 
   Car810gForm(instance=post) 
   form 
   return render(request, 
   ' carblogform. html , 
   {'form' : 
   instance=post) 
   form} ) 
2. Website view

Machine generated alternative text:
New lights for Mini BMW 
The new Mini family drinks less fuel, has a new dual-clutch gearbox, and is precisely 
thirty-seven per cent more British, old bean. Because the Oxford-build posh towncar 
has inherited Union Jack motif taillights. 
Back To Blog 
Edit 
Sept. 2, 2018, 10:19 a.m. I Views 6 | Tag None 

## The django Thread (forum) app

The forum app was created for users to connect to mechanics and ask questions on various car topics. Only subscribing members can response to a post and only staff can add a new subject similar to the Blog app. Registered users can also add a vote to certain subject so mechanics can see some results on their advice. For fun a further feature was added with JavaScript for users to toggle through the Auto Express best car 2018 list.

The forum also made use of font awesome and tinymce.

## Models.py – Forum

Once the new app was added under the settings.py file the first step was to add the models for the app. Three classes were created: Subject, Thread and Post.

Each model had fields added and a migration was done. As mentioned only registered users can respond to a discussion so the user field was set to the AUTH\_USER\_MODEL.

Once the migrations were done the three models were registered on the admin.py. The staff can now add new Forum subjects.

1. Models.py – Forum

Machine generated alternative text:
models. py 
2 
4 
6 
7 
8 
9 
10 
11 
12 
13 
14 
15 
16 
17 
18 
19 
20 
21 
22 
23 
24 
25 
26 
27 
28 
29 
30 
from 
from 
from 
from 
from 
from 
coding: utf-8 -*- 
future 
import unicode literals 
django.db import models 
django. utils import timezone 
tinymce . models import HTMLFie1d 
django. conf import settings 
django. utils import timezone 
class Subject(modeLs.ModeL): 
. CharF ie1d(max_1ength=255) 
models 
name 
= HTMLFie1d() 
description 
(self) : 
def unicode 
return self. name 
class Thread(modeLs .1v10deL): 
. CharF ie1d(max_1ength=255) 
models 
name 
models . ForeignKey(settings .AUTH USER I•ODEI_, related_name= 'threads ) 
user — 
subject = models . ForeignKey(Subject, 'threads ) 
created at = models. now) 
class Post(modeLs.ModeL) 
thread = models. ForeignKey(Thread, related_name= posts ' ) 
comment = HTMLFie1d(b1ank=True) 
models . ForeignKey(settings .AUTH USER I•ODEI_, related_name= ' posts ' ) 
user — 
created at = models. now) 

1. Migrate the changes – Forum

Machine generated alternative text:
: manage . py makemigrations 
igrations For ' threads' • 
threads \migrat ions _ in it iaI . py 
Create model Post 
Create model Subject 
Create model Thread 
Add Field thread to post 
Add Field user to post 
: manage . py migrate 
perations to perform: 
Apply all migrations: accounts. admin. auth. 
unning migrations: 
Applying threads .øøøl_initial... 
blog. 
content types. 
F Iatpages. 
sessions. 
sites. 
threads 

## HTML files and thread extras

Two html files were created.

Carforum.html (this will list all the Subjects)

Carthreads.html (this will list a particular subject with its threads)

For user to add a new thread they need to be registered i.e. logged in. The @login\_required was therefore used on the views.py to ensure only logged in users are adding threads.

Under the apps template folder a new folder was created called forum which houses all the html files for the forum.

A new folder was created call threadextras which kept a new file called thread\_extras.py. This file contained the filtering for all the threads, for example get\_total\_subject\_posts

1. HTML files – Forum

Machine generated alternative text:
2 
4 
6 
7 
8 
9 
10 
11 
12 
13 
14 
15 
16 
17 
18 
19 
20 
21 
22 
from 
from 
from 
from 
from 
from 
from 
from 
coding: utf-8 -*- 
future 
import unicode literals 
django. shortcuts import render, get_object_or_4Ø4 
models 
import Subject, Thread, Post 
django. shortcuts import redirect 
django. contrib import messages, auth 
django. contrib. auth . decorators import login_required 
django. core . urlresolvers import reverse 
django. template . context_processors import csrf 
def forum(request): 
return render (request, 
' forum/ carforum. html ' , 
def threads (request, subject_id) 
subject = get_object_or 404(Subject, pk=subject_id) 
return render(request, 
forum/ carthreads. html ' , 
' subjects ' : 
{ ' subject' 
.all()}) 
Subject .objects 
subject}) 
@login_required 
def new_thread(request, 
pass 
subject_id) : 

1. Thread\_extras.py filtering – Forum

Machine generated alternative text:
2 
4 
6 
7 
8 
9 
10 
11 
12 
import arrow 
from django import template 
from django. core . urlresolvers import reverse 
template. Library( ) 
register 
@register.filter 
def get_total_subject_posts(subject) 
total_posts 
o: 
for thread in subject . threads . all 
total_posts thread . posts 
. count 
return total_posts 

## Views.py file – Forum

The views.py consisted of forum (directing to main forum html file), threads (thread lists), creating new threads, adding new posts to a thread, editing, saving and deleting functions.

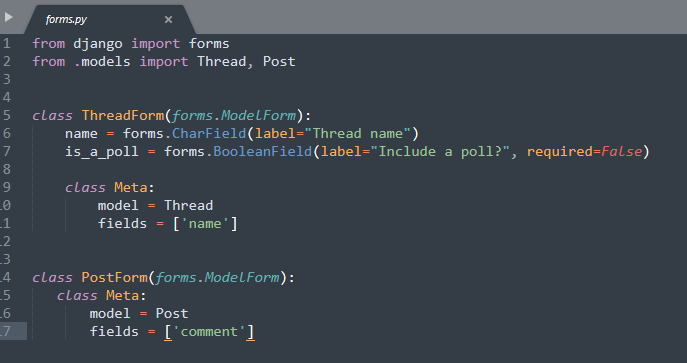
For all editing, saving and deleting the @login\_required was again used so only registered users could contribute. The {% if user.is\_authenticated %} method and {% if post.user == user or user.is\_staff %} method were both used for authenticating on the forum app. The POST feature was used for the ‘new’ item functions which is the same process as the blog app.

Two forms were designed for adding a new thread or a new post to a thread. These were then referred to in the views.py file.

1. Views.py file new thread – Forum



1. Forms.py – Forum



## New poll app and design

A new poll app was also created to be used in conjunction with the forum/thread app.

Once a user adds a new thread they can include a poll to obtain voting on certain subject from users. Via the thread\_extras.py file the calculations on how many users voted and it’s percentage. Font Awesome and Tinymce made the look of the voting and threads even better.

The Auto Express JavaScript addition was added for fun for car enthusiasts.

1. Models.py file – Polls

Machine generated alternative text:
2 
4 
6 
7 
8 
9 
10 
11 
12 
13 
14 
15 
16 
17 
18 
19 
20 
21 
22 
23 
24 
25 
26 
27 
28 
29 
30 
from 
from 
from 
from 
coding: utf-8 -*- 
future 
import unicode literals 
django.db import models 
django. conf import settings 
threads . models import Thread 
class Poll(modeLs.ModeL)• 
models . TextF ield( ) 
question - 
thread = models. OneToOneFie1d(Thread, 
(self) : 
def unicode 
return seLf.question 
class PollSubject(modeLs.ModeL): 
null=True) 
name 
poll 
def 
. CharF ength=255) 
models 
models . ForeignKey (Poll, related_name= subjects ) 
(self) : 
unicode 
return seLf. name 
class Vote(modeLs.ModeL)• 
poll = models . ForeignKey(P011, related_name="votes ") 
subject = models . ForeignKey(P011Subject, related_name="votes") 
models . ForeignKey(settings .AUTH USER I•ODEI_, related_name= 'vot 
user — 

1. Forms.py view – Forum (vote feature added)

Machine generated alternative text:
2 
4 
6 
7 
8 
9 
10 
11 
12 
13 
14 
15 
16 
17 
from django import forms 
from . models import Thread, Post 
class ThreadForm(forms. modeLForm) 
name = forms . CharFie1d(1abe1="Thread name") 
required 
class Meta: 
model 
fields 
. a poll?" 
forms 
Thread 
[ ' name 
class PostForm(forms .ModeLForm 
class Meta: 
model 
fields 
Post 
[ ' comment ' ] 

To hide the voting buttons once someone votes update the thread\_extras.py file. This ensures they can't vote again.

1. Thread\_extras.py – Voting

Machine generated alternative text:
@register.simple_tag 
def user_vote_button(thread, subject, user): 
vote = thread . poll id) . first() 
if not vote: 
if user. is _ authenticated(): 
link 
(div class="c01-md-3 btn-vote"> 
<a href="%s" class="btn btn-default btn-sm"> 
Add my vote! 
</div>""" % reverse(' cast vote' 
link 
return 
return 
' thread_id ' 
thread. id, 
' subject_id ' 
: subject . id}) 

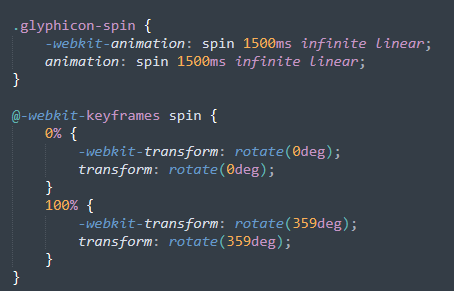
The use of Tinymce was use for the voting fields

1. Poll feature

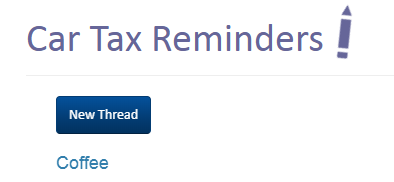
Machine generated alternative text:
poll 
DVLA Reminders 
Here's how the votes are so far: 
DVLA Reminder 
Add my vote! 
Phone App Reminder 
Add my vote! 
Post Reminder 
Add my vote! 
New post 
chrisk 21@yahoo.co.uk contact these guys for reminders 
3 POSTS 
Help with reminders 
Contact the Driver and Vehicle Standards Age 

## Glyphicons was used to design a spinning pencil to make the Website a bit more eye-grabbing.

1. Glyphicon pencil css – forum

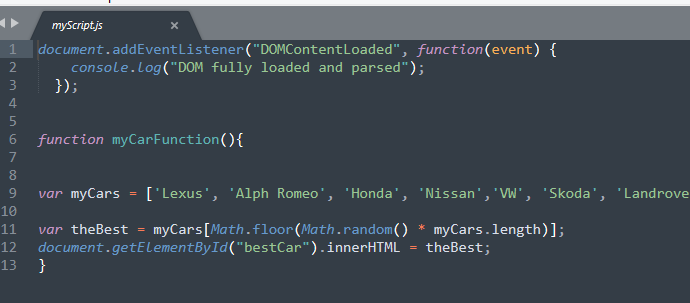


1. Glyphicon pencil html – forum

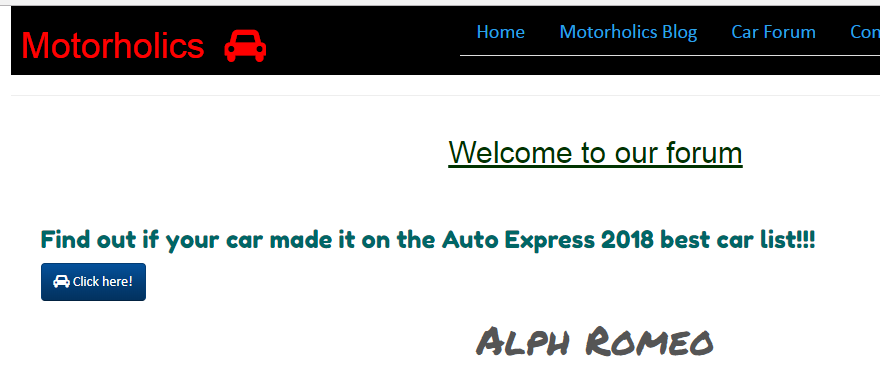


The mechanics.css file was saved under the static folder. The JavaScript for the Auto Express feature was also saved under the static folder.

1. myScript.js – forum



1. Auto Express JavaScript feature



## motorholics project testing

The below table summarizes the testing results whilst working on the Django project.

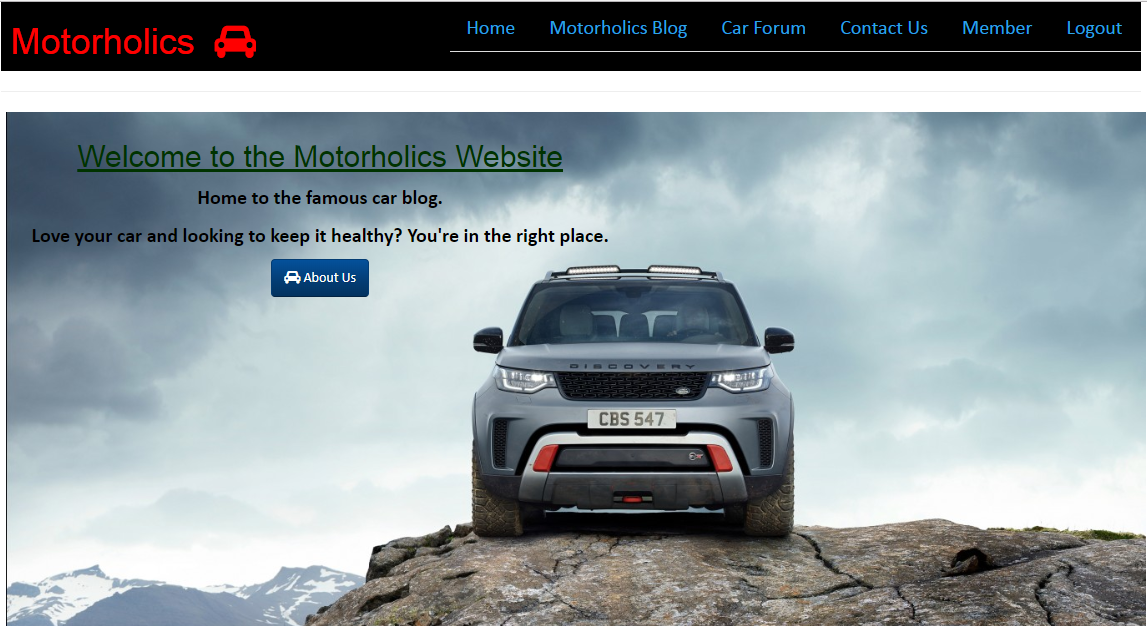
|  |
| --- |
| JavaScript – Auto Express feature |
| Tested in Code Pen |
| Python tests – Home app |
| test URL      Used assertTemplateUsed() to check template is the one that’s actually being used  render\_to\_response  to check output from URL and ensure nothing else is affecting it |
| Python tests – Threads (forum) app |
| Testing the threads real data but to test subjects and alike we need to run database tests (so tests are based on data) |
| Database testing |
| First create fixtures folder in app  Then run command per below to run a dump of data from the test database    This produces the Json format file output:    To make it work also need to collect my user data from accounts app    Output of user data    Updated threads test |
| Testing via one object class |
| Updated home app tests using one test class for all:    Error –    Added the below that were missing on imports  from django.core.urlresolvers import resolve  from home.views import get\_index    Corrected –    Complete home pages tests: |
| Model tests |
| MODEL TESTS  No need to tests models generally to did quick test on the custom user setup under accounts app |
| Form Validation |
| 1. Validation for unfilled forms   First test for form that is filled out    Result –    Now with incomplete fields      With incorrect matching passwords    Indentation error    Fixed    Missing password      Invalid card |
| Threads app form |
| Result –    No name entered for thread |
| Further URL tests |
| Blog url          Negative test on forum URL:  Updated to INCORRECT URL …..    Result – |

## motorholics – the complete product

The Motorholics Website was designed so it was viewed as clean and professional – a place where users can receive valuable car advice and support.

The below pictures show each section of the Website and the results.

Figure 9. Home Page



The home page features a cover image with a short introduction to the Website. Unlike some other car Websites that are overloaded with information this was kept nice and tiny.

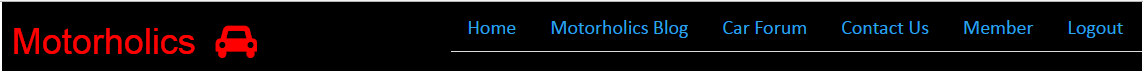
The About Us button featured a small icon using Font Awesome and this was carried over to most of the other buttons on the Website as consistent branding.

The Motorholics name was highlighted in the Bootstrap Navbar and a car icon added. The rest of the menu items on the Navbar lay neatly to the left via the nav class “"nav nav-tabs pull-right”.

When a user is logged in the Navbar shows “Logout” on menu.

However if a user is not logged in the Navbar updates to show

Figure 10. Navbar Login/Logout



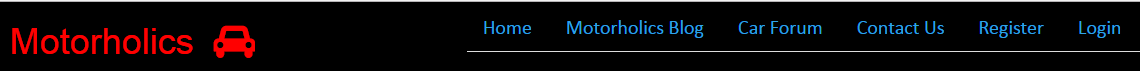
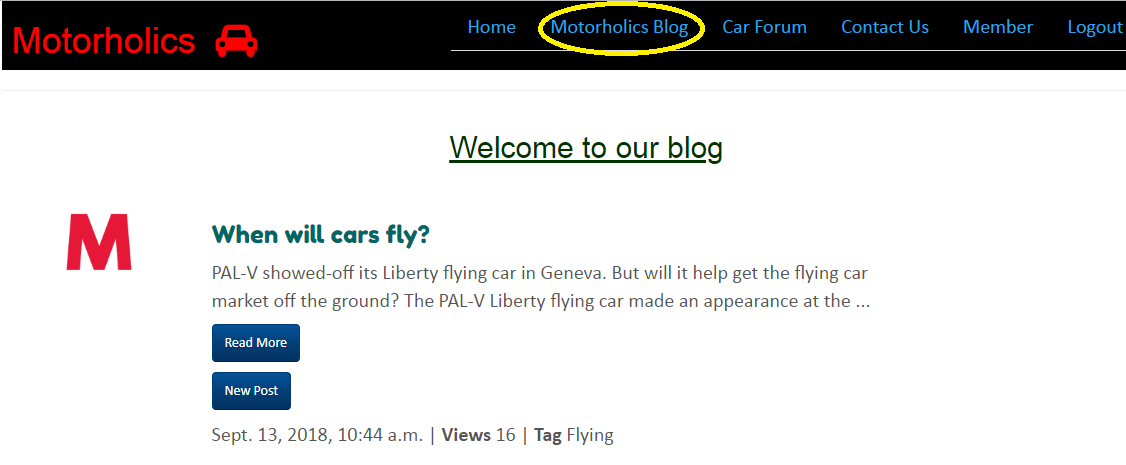


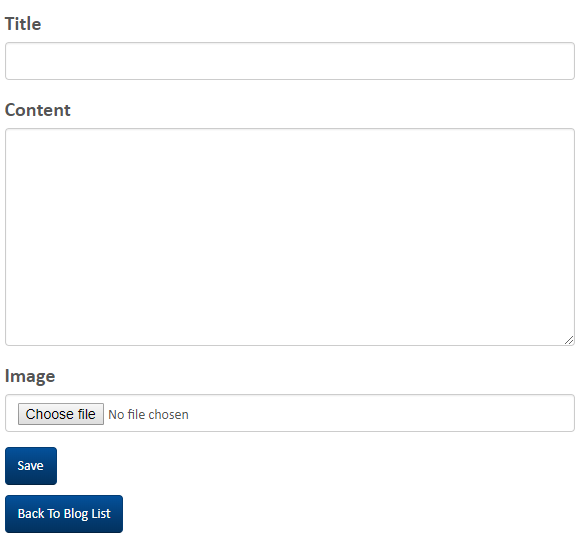
Figure 11. The Blog Page



The Blog page lists various car subjects and an M brand logo on the left. If a user is logged in they will be able to see the “New Post” button as well as the Read More button.

Under the Read More button the registered users can either edit the post or navigate back to the blog list. Here is where Disqus was also added for discussions on a global platform.

Figure 12. New Post – Blog



Adding a new post to a subject allows you to add a new title and content plus add any image you like. The image is saved to the /media/ project folder.

Figure 13. Read More – Blog

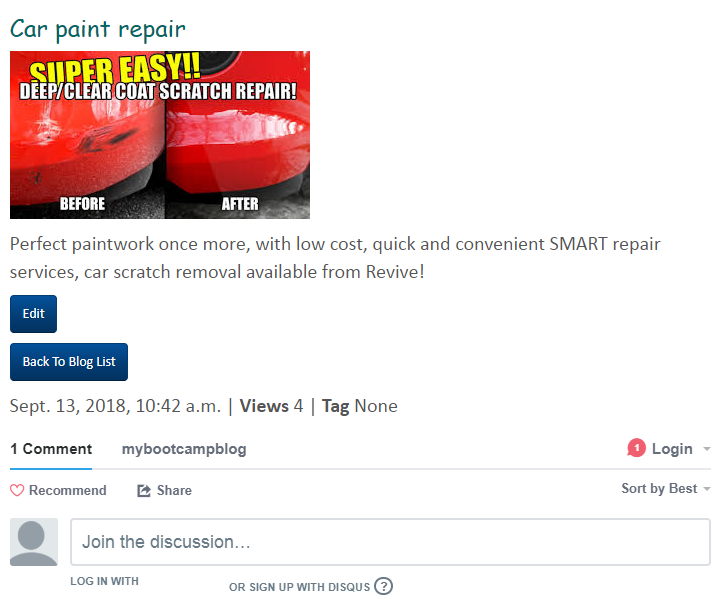
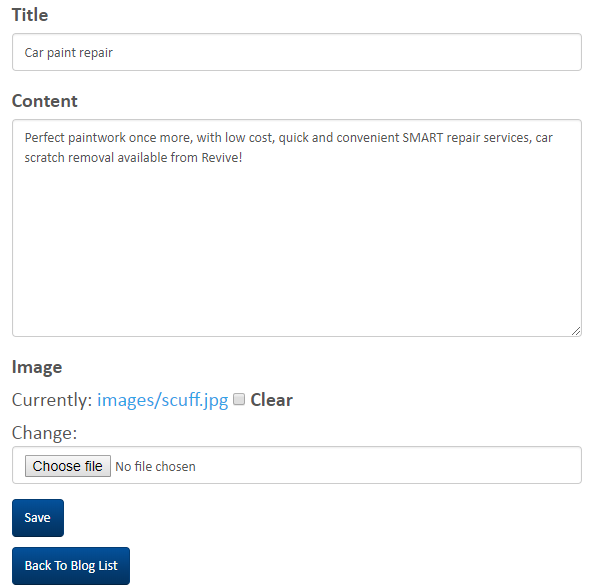


Figure 14. Editing a thread



Once a user clicks on Edit they can modify the post’s title and content. They can also delete an existing image and add a new one then save the changes.

The Motorholics car forum listed all the threads to specific subjects started by the staff – in this case the mechanics. Registered users who are logged in can see the New Thread button once they click on a subject.

The new thread form and voting mechanism both used Tinymce.

Figure 15. The Car Forum Page

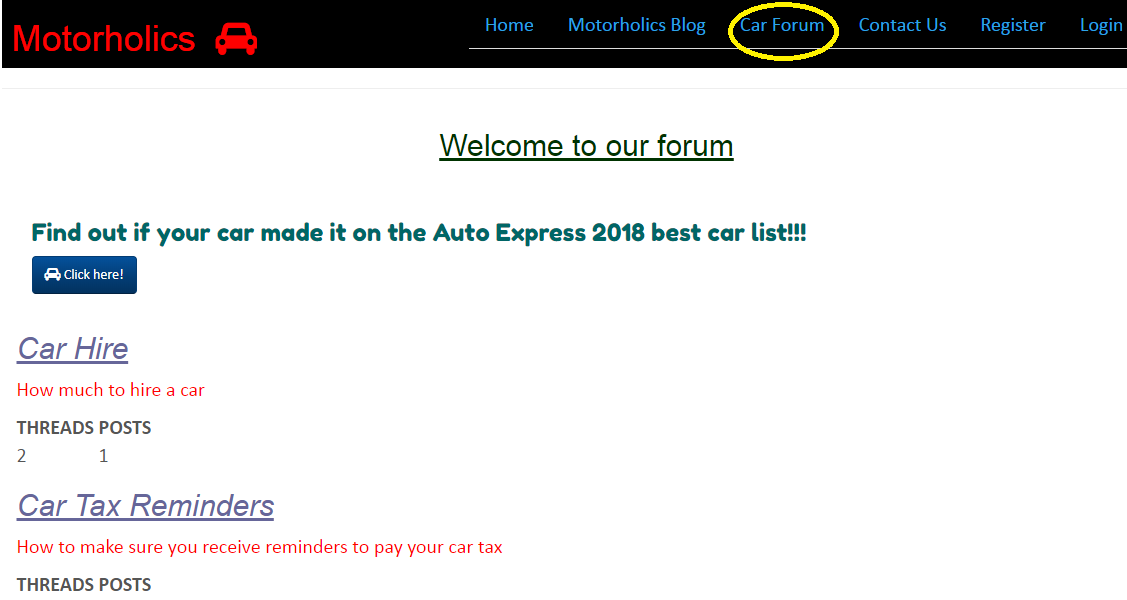


Figure 16. New Thread – Forum

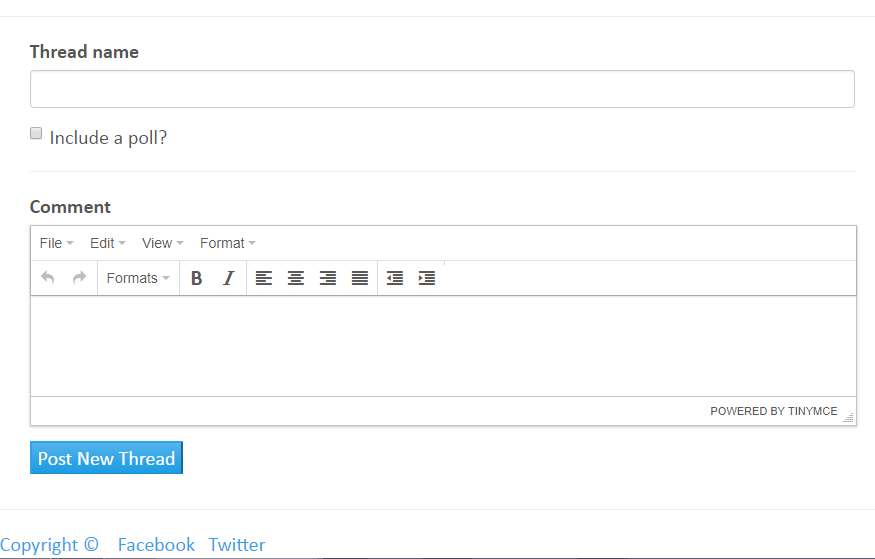
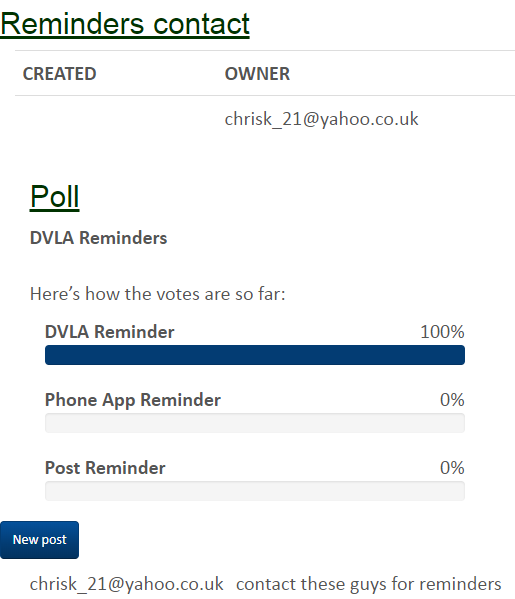
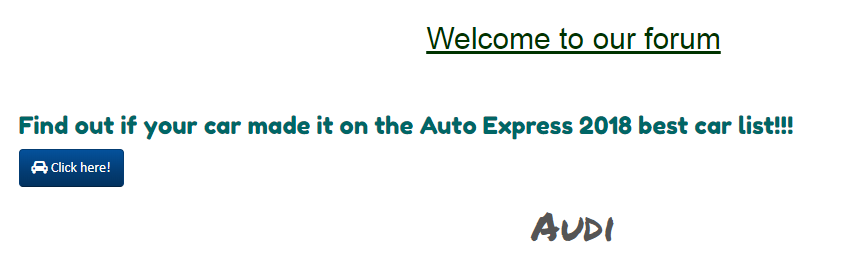


Figure 17. Voting option



The Auto Express function lists random cars that made it to the best car 2018 list.

Figure 18. Best Car 2018 JavaScript



The “Contact Us” page was created using Flatpages

Figure 19. Contact Us page

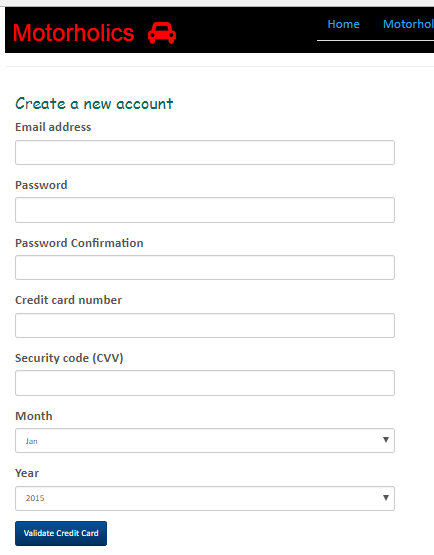


Registration

Once of the most important features was the user registration. This was designed using a custom user authentication model and meant that users registered with their email address.

Stripe was used for payment processing.

Figure 20. Registration page



Login/Logout pages

Users logged in using their email address and password. Once logged in they can then contribution to the blog and forum.

Under the Member tab users can view the account details.

Once logged out the user is presented with a green stripe on the home page confirming they have been logged out.

Figure 21. Login page

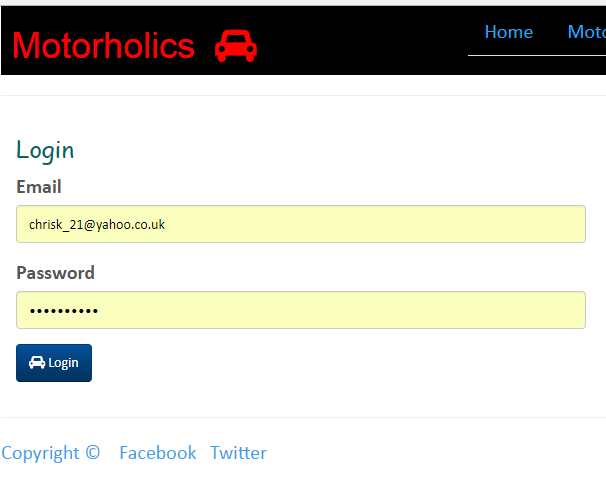
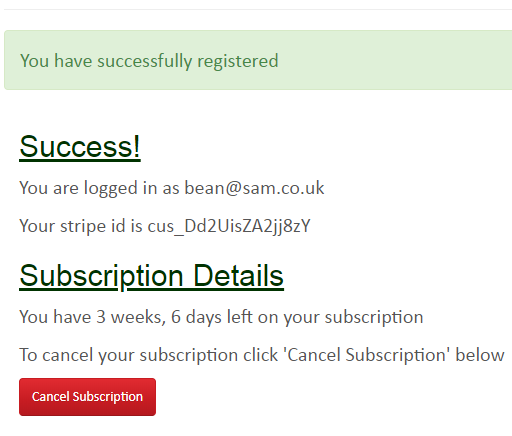


Figure 22. Logout notification

## 

Figure 23. User registration



## reusable apps

## references

The complete product

The nav bar

**Figure 13 – The nav-bar**



The navigation bar was constructed using Bootstrap and fixed to the top of the page: *navbar navbar-inverse navbar-fixed-top*

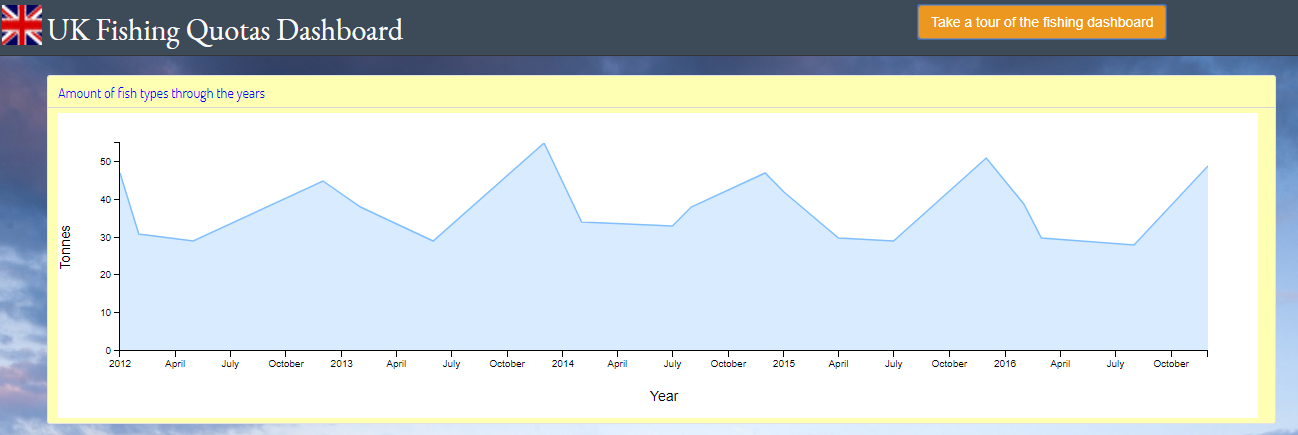
The idea was to keep the nav bar smart and more conservative – the tour button which was the tool tips functionality had to stand out so this was done using Bootstrap warning button: *btn btn-warning*

Line Chart – Amount of Fish Types Through the Years

This chart featured as the main chart at the very top of the Webpage. It pulled data from field “date” and served as the dimension (x) layout, then the metrics grouped the amounts for each region.

The data updates when selecting the region or port chart options

**Figure 14. The Line Chart – Amount of Fish Types Through the Years**

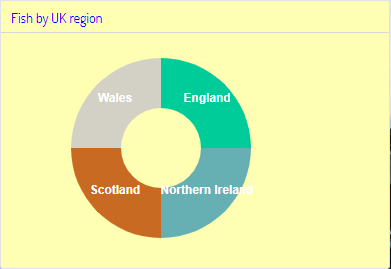


Pie Chart – Fish by UK Region

This provides the user the option of obtaining statistics for individual UK regions.

Scotland has the largest number of caught of fish with 1.88 billion tons over a 5 year span. Wales has the least with 78.3 million tons. Overall there has been healthy landings of fish throughout the last 5 years for the UK.

**Figure 15. The Pie Chart – UK Regions**



**Figure 16. The Quantity and Value Totals**

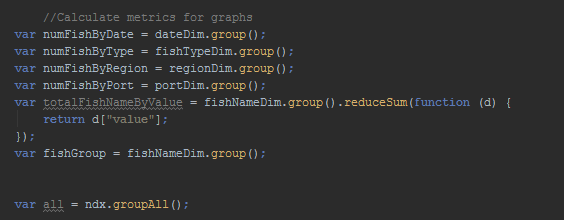


Pie Chart – Main Regional Ports

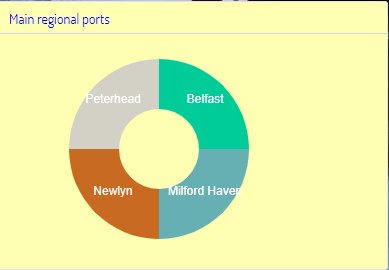
The main regional ports pie chart is there to highlight where the fish is caught in the individual UK regions. Selecting a port then presents the relevant UK region in the regional pie chart above it.

The metrics in the graph.js file allowed for this grouping and calculation of values to happen:

**Figure 17. Metrics in graph.js file**



**Figure 18. The Pie Chart – UK Ports**

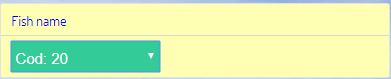


Select Menu – Fish Names and Row Chart – Fish Types

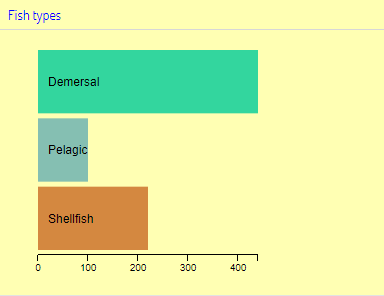
The fish name select menu was important feature to answer the project question of how well cod was doing. Once the user clicks on the name ‘cod’ under the list the quantity and values are shown highlighting that there has been no downward drop in the amount that the UK catches, in fact cod has one of the highest numbers.

The fish types row chart and information button are there for information purposes to provide further knowledge on what the UK is retrieving in terms of fish categories. Cod, haddock and crabs are among the most popular. Both cod and haddock are demersal fish while crab falls under the shellfish category.

**Figure 19. The Select Menu for fish names**



**Figure 20. The Row Chart – Fish Types**



Conclusion

With the help of the technologies used we can see that the statistics from the government show that there is no downward spiral of fish landings in the UK – quite the opposite, there is a healthy number of fish caught over a span of 5 years!

Heroku deployment

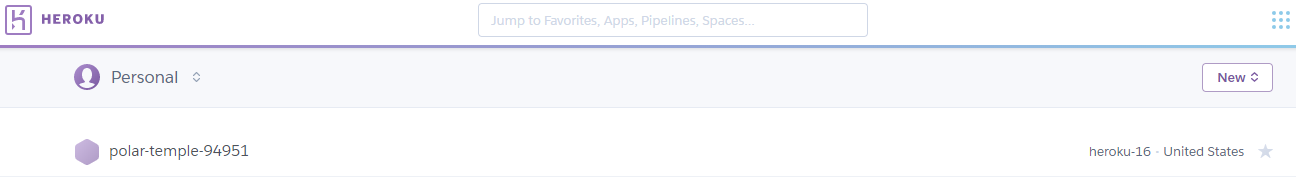
Heroku new app setup

I did an admin cmd prompt and typed **heroku create** in cmd line

**Figure 21. Heroku app creation**



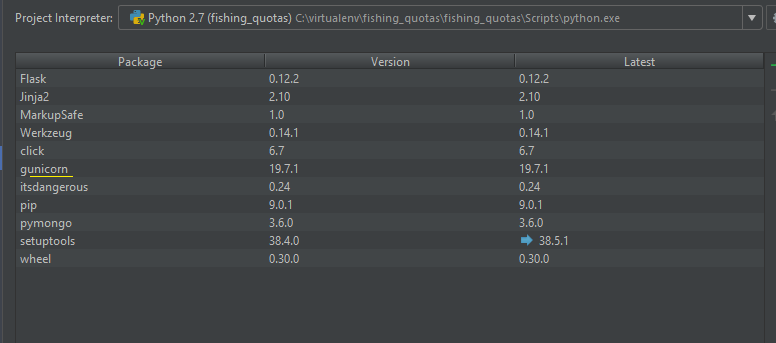
View new app on Heroku sign-on



Next I installed **gunicorn**. *Gunicorn is used for running HTTP servers on UNIX based operating systems, such as Mac OS X and Linux distros.*

Then I went to Pycharm settings and installed it.

Figure 22. Pycharm settings



Requirements file

I created a requirements txt file so that I can keep all the installed packages versions.

The following commands were run:

Activate VM : (fishing\_quotas) C:\virtualenv\fishing\_quotas\Scripts>activate

Run pip freeze --local > requirements.txt

This has applied the following output dependencies to the requirements.txt file:

click==6.7

Flask==0.12.2

gunicorn==19.7.1

itsdangerous==0.24

Jinja2==2.10

MarkupSafe==1.0

pymongo==3.6.0

Werkzeug==0.14.1

Procfile

The Procfile is a file that’s used by Heroku to tell it what to do with the application once it’s been deployed.

I created new file called procfile and saved as a txt file.

A procfile.txt file was then created and added to the pycharm project folder containing the following command: web: gunicorn fishing\_quotas:app

Heroku will then execute this command once the procfile is detected.

Since I have Windows on my machine I created another procfile called Procfile.windows containing the command: web: python fishing\_quotas.py

Since gunicorn doesn’t have Windows support the server will run directly from the app.

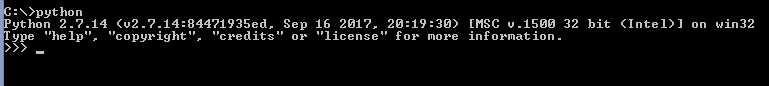
To ensure Windows users run the window procfile the following command was initiated (after ensuring Mongo DB was running)

heroku local -f Procfile.windows

Runtime.txt

Lastly a runtime.txt was created in Pycharm to tell Heroku what version of Python is running. Per below the version running on this PC is 2.7.14.

Therefore python-2.7.14 was added to the runtime.txt file.



Appendix A: References

<http://www.gov.scot/Topics/marine/marine-environment/species/fish/pelagic>

<https://data.oecd.org/fish/fish-landings.htm>

<https://www.gov.uk>

<https://fullfact.org/europe/eu-pinching-our-fish/>

<http://www.telegraph.co.uk/comment/11305123/No-end-to-the-EUs-crazy-fishing-policy.html>

<http://bl.ocks.org/zanarmstrong/05c1e95bf7aa16c4768e>