



ECE528 Cloud Computing
Winter 2021

TUTORIAL:
DEPLOYING DJANGO IN AWS

March 14th, 2021
Version: 1.0

Daniel A. Zajac
DanZajac@umich.edu
College of Electrical Engineering and Computer Science
University of Michigan



Contents

1	Introduction	3
1.1	Prerequisites	3
1.2	Resources	3
2	Django Background	3
2.1	Tutorial Objectives	4
2.2	Project Organization	4
2.3	Project Organization	5
3	Tutorial1: Creating and Deploying to AWS	6
3.1	Creating the Project and Install Perquisites	6
3.2	Getting the Project Ready to Deploy	9
3.3	Setting up the AWS EB Project and Environment	11
3.4	Setting up a Database and Admin Setup	14
3.4.1	Assignment 1.1: Use the RDS CLI	20
4	Tutorial 2: Creating and Deploying an App	20
4.1	Creating the First View	20
4.1.1	Assignment 2.1: Add a View that Renders A Template	22
4.2	Adding and Using Models	23
4.2.1	Assignment 2.2: Modify and Migrating Models	29
4.2.2	Assignment 2.3: Create a View with a Form	29
4.3	Loading Data from CSV	29
4.4	Using a Template and Creating Pie-Chart	31
4.4.1	Assignment 2.4: Create a view to display a Pie Chart of total sales by Manufacturer	34
4.4.2	Assignment 2.5: Modify the Template to use Modal Restaurant Template	35
4.5	Creating Bar-Chart and Self Referencing from JS	36
4.5.1	Assignment 2.6: Modify the View to serve a Dynamic Picture	38
5	Cleaning Up	38

1 Introduction

This tutorial will show the user how to create a basic Django project, setup an AWS Elastic Beanstalk project and environment, and connect the system to the AWS RDB database.

1.1 Prerequisites

This tutorial assumes you are running on windows 10. To complete this tutorial will require the following:

- Pycharm Professional 2020
- Python 3.8 or later
- AWS educate or AWS Account with billing configured
- [OpenSSH and SSH-Keygen](#)
- Project file and Tutorial From GitHub Repo: [DjangoAWSTutorial](#)

1.2 Resources

The following tutorials and resources were used in the construction of this tutorial. Be aware, some are outdated and will not work with the most recent version of Django (versions higher than 2.2):

- [Django Tutorial](#)
- [Deploying a Django application to Elastic Beanstalk](#)
- [Adding an Amazon RDS DB instance to your Python application environment](#)
- [How to Use Chart.js with Django](#)
- [Kaggle: US Cars Dataset](#)

2 Django Background

Is a server-side python web framework that offers a highly integral model based database interface. It offers a fast and easy framework for creating dynamic web content way to create database backed content without directly interacting with SQL.

2.1 Tutorial Objectives

This tutorial will focus on creating a basic Django project from scratch and deploying it through elastic beanstalk. Figure 43 shows a high level deployment of the end project. Elastic beanstalk will stand up a load balancer and EC2 compute nodes for Django and the Nginx webserver will run. Further, the compute nodes will use S3 buckets for storage and an Amazon RDB Database will be connected for storage.

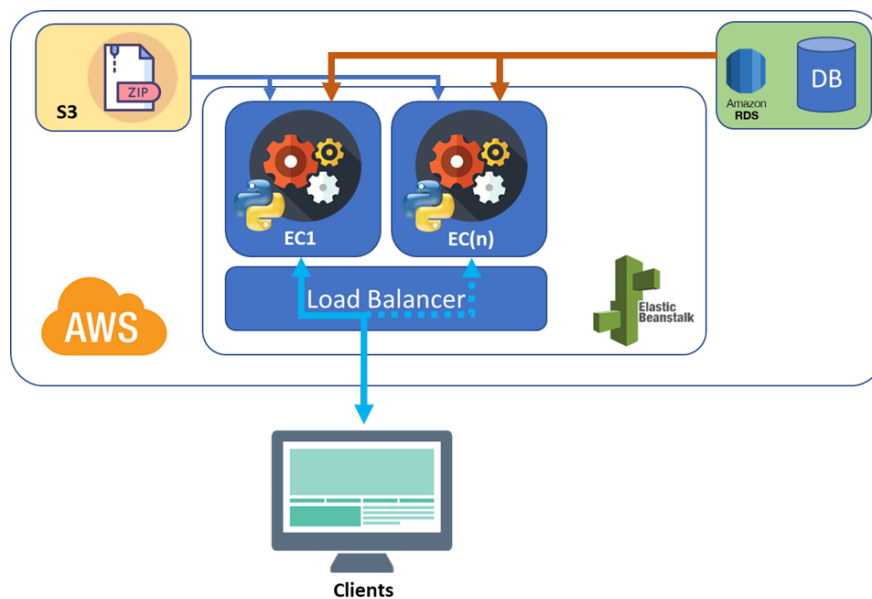


Figure 1: Typical AWS Django Deployment

Note: Securing a Django deployment is beyond the scope of this tutorial. Refer to the Django documentation for recommended best practices on deploying a Django project in a production environment.

2.2 Project Organization

The basic Django Project (Figure 2) consists of Applications, a database, and typically an Admin portal. Clients are routed to their desired application by URL matches. URLs can be cascaded

for multiple levels and typically the root project imports individual app urls.py. This allows the system design to be modular.

2.3 Project Organization

Views represent the python code that renders the individual endpoints defined in the urls.py file. Again, both the root project can have views (uncommon) but each app will define it's own views. The views can call on templates to render view responses.

The database is interfaced through python objects defined by models. A series of management scripts will create and modify the database tables and fields based on the models. It's important to run these scripts (*makemigrations*, *migrate*) after modifying the models or the relationships to the tables may break. The database is shared across the applications.

Finally, the admin is a special application that allows direct interaction with the database content and users. The models must be reflected and registered with the admin app. There is a series of scrips provided to manage creating superusers (*createsuperuser*, *changepassword*) that can be used to add and modify users for the admin app.

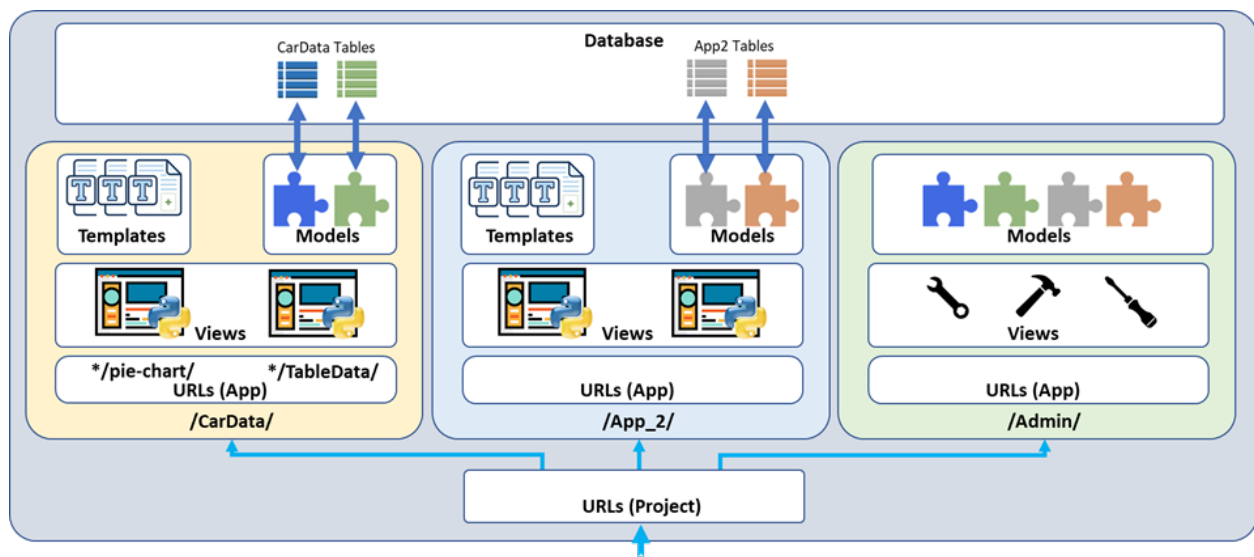


Figure 2: Typical Django Project Organization

3 Tutorial1: Creating and Deploying to AWS

This Tutorial will:

- Create a new Django project from scratch using Pycharm.
- Configure and Test the installation locally.
- Install the EB CLI and Prerequisites
- Launch an EB Environment
- Create and Configure an AWS RDB and configure the Project
- Configure the Admin and Test the Deployment on AWS EB and RDS

3.1 Creating the Project and Install Perquisites

1. Open Pycharm and create a new project:
 - (a) Select “Django” from the Left menu.
 - (b) Give the project a name (Location)
 - (c) Name the application “CarData” in (Application Name)
 - (d) Use Vritualenv
 - (e) Ensure the interpreter is set to python 3.8
 - (f) Click Create button.

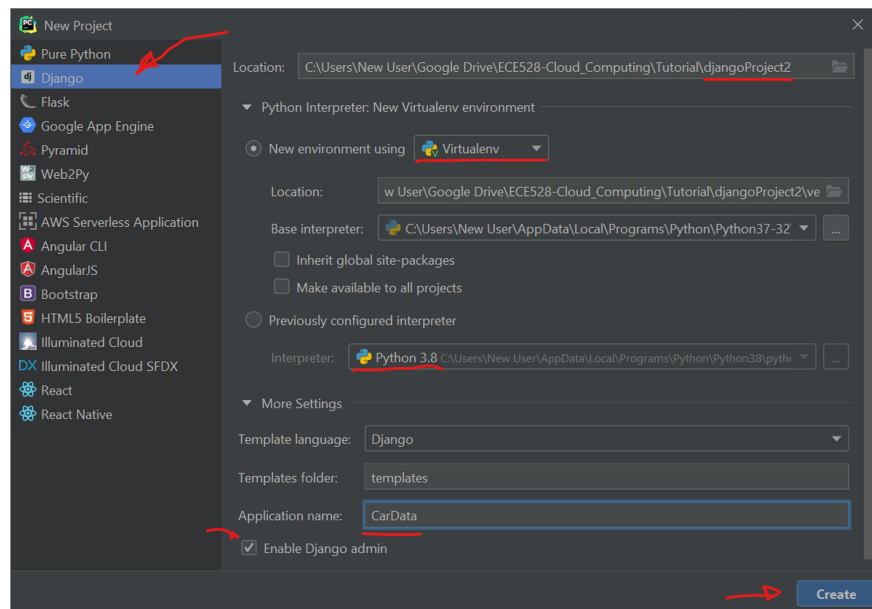


Figure 3: Creating a New Django Project

2. Start the Django CLI:

- (a) Open the project folder
- (b) Open the djangoProject2 folder
- (c) Open the settings.py in the editor
- (d) Open the Django Console by Pressing <ctrl>+<alt>+r

3. Install Prerequisites:

- (a) Open the “Terminal” on the bottom strip
- (b) Run: ***pip install awsebcli --upgrade***
- (c) Allow all the scripts to install and index
- (d) Run: ***pip install pymysql***

4. Test the project:

- (a) Switch to the “manage.py@djangoProject2” tab console
- (b) Run: ***makemigrations***
- (c) Run: ***migrate***
- (d) Run: ***runserver***
- (e) Launch a web browser from the link in the term <http://127.0.0.1:8000/>
- (f) If all goes well, you should be greeted with the default page

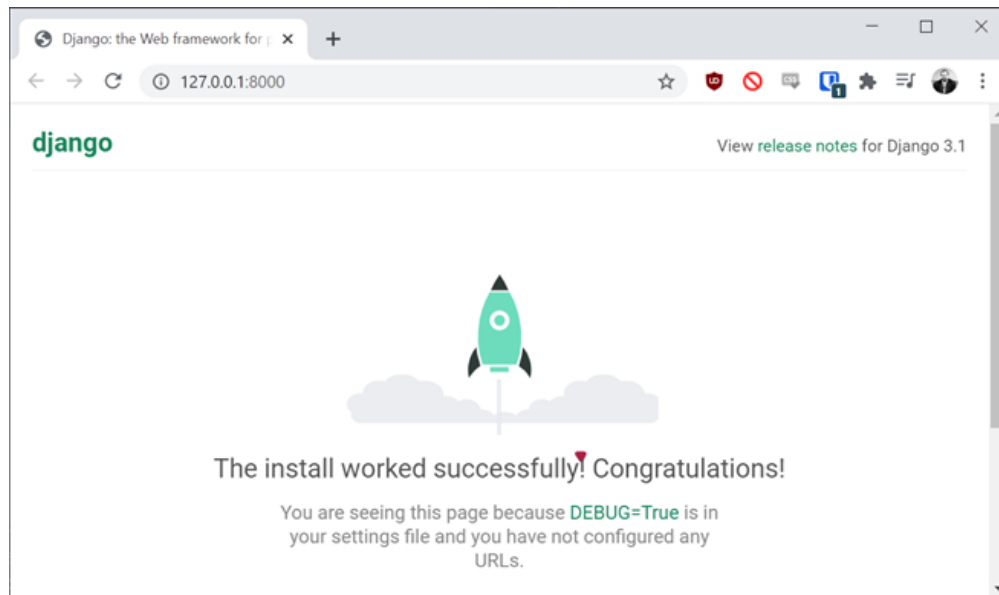


Figure 4: Success! Django Running Locally

5. Stop the server by pressing the Red stop button on to the left of the Django term.
6. You should have the following project structure

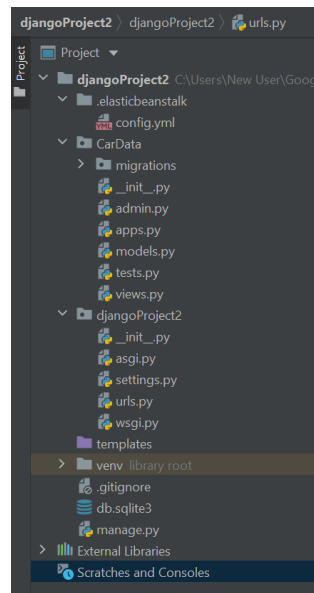


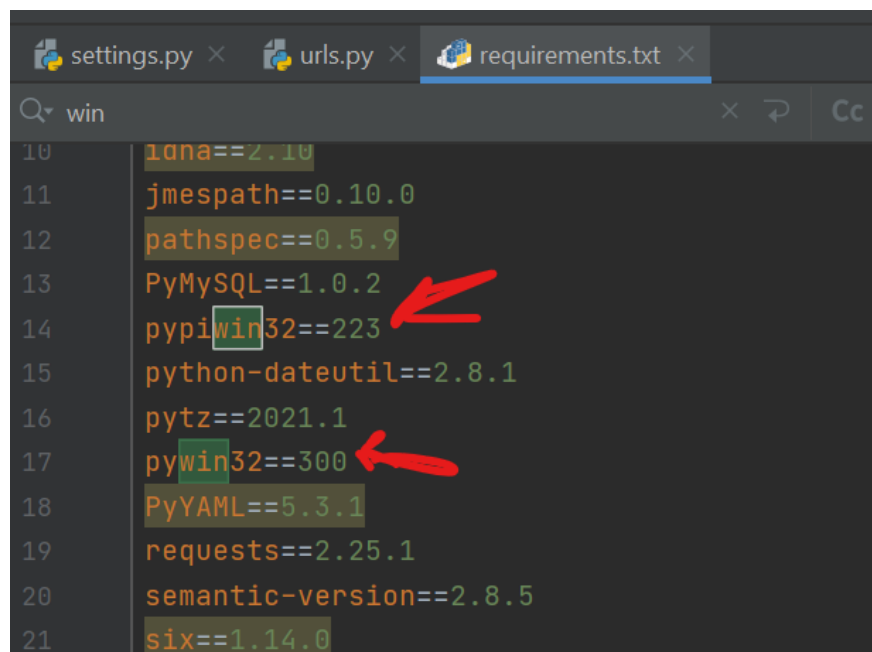
Figure 5: File Structure of Initialized Project

Congratulations, you have launched your first Django project locally.

Note: The default project uses a local sqlite database. If deployed to an AWS EB environment this will fail on Django releases newer than 2.2. AWS Linux 2 EC2 instances ship with a very old version of sqlite which is incompatible with most modern installations of Django.

3.2 Getting the Project Ready to Deploy

1. Create the requirements file:
 - (a) In the terminal migrate to the root of the project
 - (b) run: ***pip freeze > requirements.txt***
 - (c) IMPORTANT: Ensure no *win* dependencies are in the generated requirement.txt
 - i. Open the file
 - ii. Search for “win”
 - iii. Remove any entries containing win (e.g. pywin)



The screenshot shows a code editor with three tabs: settings.py, urls.py, and requirements.txt. The requirements.txt tab is active, and a search bar at the top contains the text 'win'. The search results show two lines in the requirements.txt file: 'pypiwin32==223' on line 14 and 'pywin32==300' on line 17. Red arrows point to these two lines, indicating they should be removed. The rest of the requirements.txt file contains the following dependencies: 'idna==2.10', 'jmespath==0.10.0', 'pathspec==0.5.9', 'PyMySQL==1.0.2', 'python-dateutil==2.8.1', 'pytz==2021.1', 'PyYAML==5.3.1', 'requests==2.25.1', 'semantic-version==2.8.5', and 'six==1.14.0'.

```
10 idna==2.10
11 jmespath==0.10.0
12 pathspec==0.5.9
13 PyMySQL==1.0.2
14 pypiwin32==223
15 python-dateutil==2.8.1
16 pytz==2021.1
17 pywin32==300
18 PyYAML==5.3.1
19 requests==2.25.1
20 semantic-version==2.8.5
21 six==1.14.0
```

Figure 6: Important! Remove all references to Windows Libraries

- (d) Note: you may need to periodically re-run this step if new pip dependencies get added to the project.

2. Create the EB deployment configs

- (a) In the root of the project, create a new directory called “**.ebextensions**”
- (b) Create a file called “**Django.config**” with the following contents:

```
option_settings:
  aws:elasticbeanstalk:container:python:
    WSGIPath: djangoProject2.wsgi:application
  aws:elasticbeanstalk:environment:proxy:staticfiles:
    /static: static
```

3. Create the static files folder:

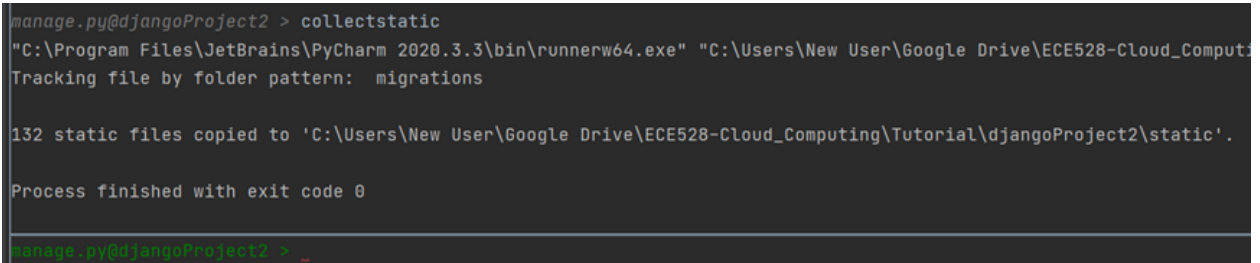
- (a) In the root of the project create a new folder called “static”
- (b) Open settings.py in the djangoProject2 folder
- (c) Under “STATIC_URL = '/static/'” **add** the following line:

```
STATIC_ROOT = 'static'
```

- (d) Find the “ALLOWED_HOSTS = []” and **update** it to:

```
ALLOWED_HOSTS = ['*']
```

- (e) From the Django terminal run: **collectstatic**



```
manage.py@djangoProject2 > collectstatic
"C:\Program Files\JetBrains\PyCharm 2020.3.3\bin\runnerw64.exe" "C:\Users\New User\Google Drive\ECE528-Cloud_Computing\Tutorial\djangoProject2\static"
Tracking file by folder pattern: migrations

132 static files copied to 'C:\Users\New User\Google Drive\ECE528-Cloud_Computing\Tutorial\djangoProject2\static'.

Process finished with exit code 0

manage.py@djangoProject2 > _
```

Figure 7: Collecting and Archiving References to /static/

3.3 Setting up the AWS EB Project and Environment

1. For AWS educate users – Make sure the credentials are up to date:
 - (a) Create a file called “credentials” in the c:\users\<user name>\.aws folder.
 - (b) From the Vocareum portal, click Account Details
 - (c) Click the “Account details” button
 - i. Expand the AWS CLI: by click the “Show” button
 - ii. Copy the complete contents into the “credentials” file and save
2. **Important: Make sure you have a default SSH keygen before executing the next steps**
 - (a) Open a terminal <win>+<r> and type *cmd*
 - (b) Run: *ssh-keygen*
 - (c) Follow the prompts

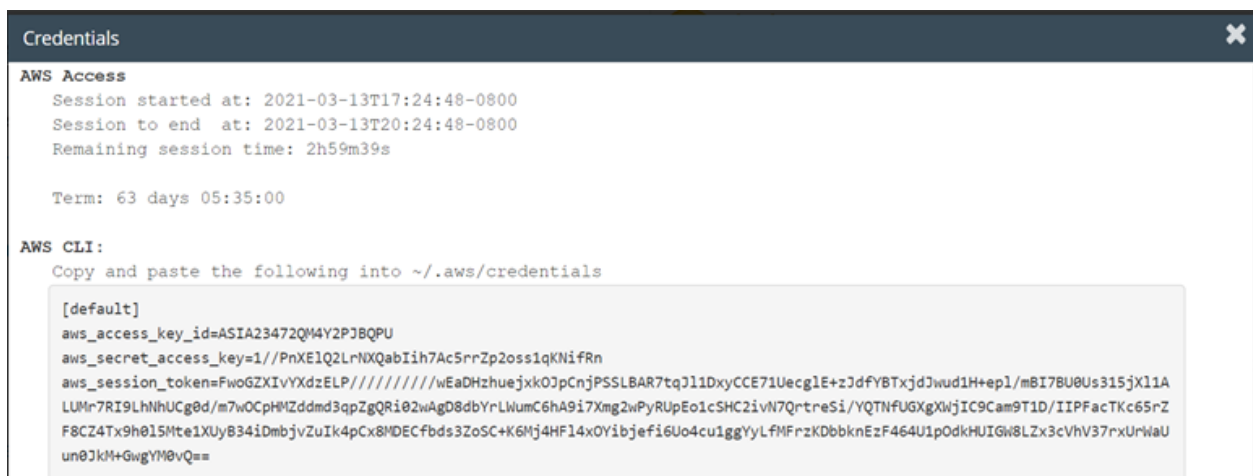


Figure 8: Importing the AWS Educate Credentials

- (d) Create another file called “config” in the .aws folder
- (e) Put the following into the file (adjust region as required) and save.

```
[default]
region = us-east-1
```

```
output = json
```

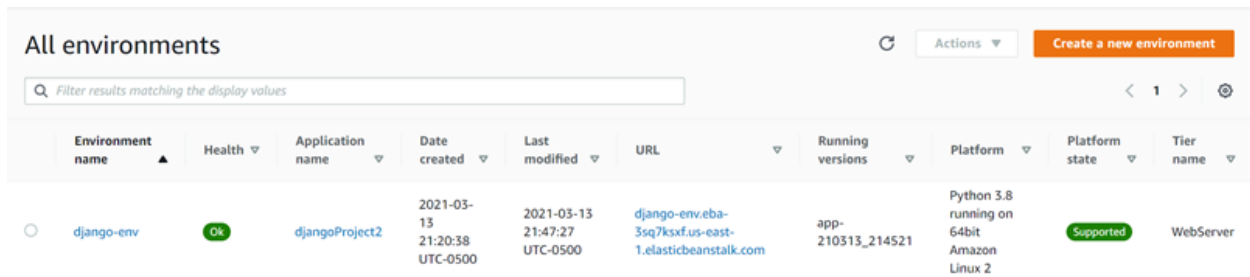
3. Create the Environment:

- (a) From the terminal tab run: ***eb init***
 - i. Select the region
 - ii. Give the project a name “djangoProject2”
 - iii. Select the Python 3.8 running on 64 bit Amazon Linux 2
 - iv. Y for SSH setup (helpful for debugging if necessary)
 - v. Create a new keypair
- (b) Run: ***eb create django-env***
- (c) The script should run and generate new instances. This will take a while.

```
2021-03-14 01:42:37 INFO Created CloudWatch alarm named: awseb-e-psvft8ma7r-stack-AWSEBCloudwatchAlarmHigh-5ZUQPCV97MVQ
2021-03-14 01:42:37 INFO Created CloudWatch alarm named: awseb-e-psvft8ma7r-stack-AWSEBCloudwatchAlarmLow-5RNYQ4B3WBRR
2021-03-14 01:42:44 INFO Instance deployment successfully generated a 'Procfile'.
2021-03-14 01:42:46 INFO Instance deployment completed successfully.
2021-03-14 01:43:51 INFO Successfully launched environment: django-env
```

Figure 9: Environment Successfully Created

- 4. Ensure you are using the created environment by running: ***eb use django-env***
- 5. Check to see if your application is running:
 - (a) Open the AWS Console
 - (b) Navigate to the EB dashboard: [AWS Console Home](#)
 - (c) Your environment should be OK



Environment name	Health	Application name	Date created	Last modified	URL	Running versions	Platform	Platform state	Tier name
django-env	OK	djangoProject2	2021-03-13 21:20:38 UTC-0500	2021-03-13 21:47:27 UTC-0500	django-env.eba-3sq7ksxf.us-east-1.elasticbeanstalk.com	app-210313_214521	Python 3.8 running on 64bit Amazon Linux 2	Supported	WebServer

Figure 10: Environment Up and Healthy

(d) Click the URL to see if it has successfully deployed.

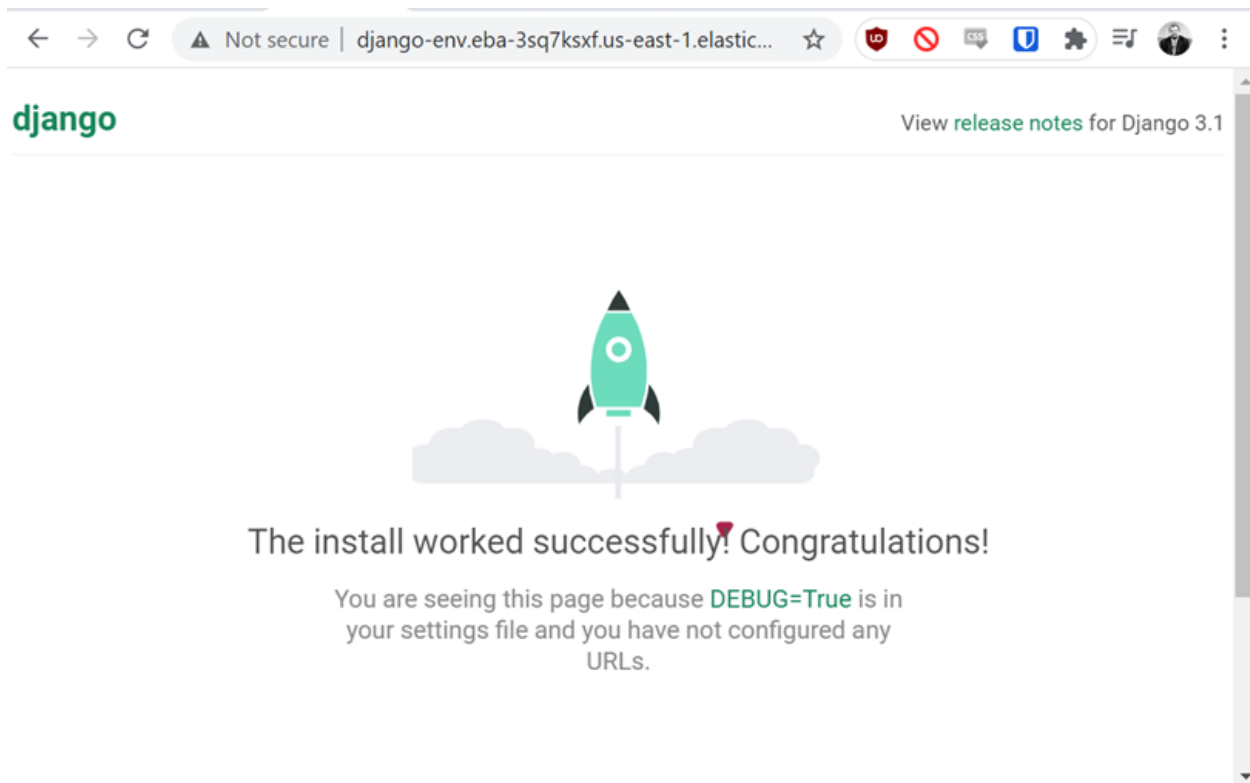
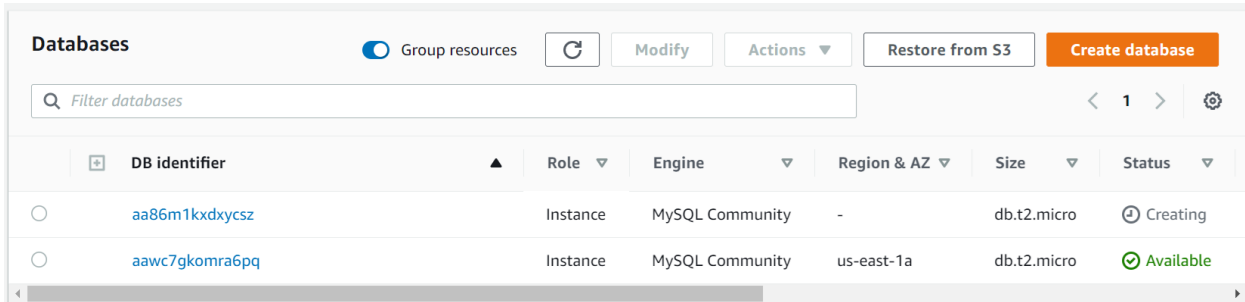


Figure 11: Project Successfully Deployed

Congratulations, you have launched your first Django project on AWS.

3.4 Setting up a Database and Admin Setup

1. Add an RDB database to the Django-env environment
 - (a) Navigate to the EB environments: [AWS EB Home](#)
 - (b) Click on django-env
 - (c) Click Configuration on the left hand side
 - (d) Scroll down to “Database” and click Edit
 - (e) Add a username “djangouser”
 - (f) Add a password “djangopw”
 - (g) Wait for the RDB to be built and deployed
 - i. You can check the status here: [AWS RDB](#)
 - ii. Click on DB instances
 - iii. Wait for the database to become “Available”



Databases						
<input type="checkbox"/> Group resources <input type="button" value="Refresh"/> <input type="button" value="Modify"/> <input type="button" value="Actions"/> <input type="button" value="Restore from S3"/> <input type="button" value="Create database"/>						
<input type="text" value="Filter databases"/>						
	DB identifier	Role	Engine	Region & AZ	Size	Status
<input type="radio"/>	aa86m1kxdxycsz	Instance	MySQL Community	-	db.t2.micro	Creating
<input type="radio"/>	aawc7gkomra6pq	Instance	MySQL Community	us-east-1a	db.t2.micro	Available

Figure 12: Database Creating

2. Update the Database Connection Details
 - (a) In Pycharm, open the project settings.py file in the project folder.

- (b) Replace the “DATABASES = ...” entry with the following:

```
DATABASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.mysql',  
        'NAME': 'ebdb',  
        'USER': 'djangouser',  
        'PASSWORD': 'djangopw',  
        'HOST': 'aawc7gkomra6pq.cgf0r2etlzn5.us-east-1.rds.amazonaws.com',  
        'PORT': '3306',  
        'OPTIONS': {  
            'init_command': "SET sql_mode='STRICT_TRANS_TABLES'",  
        },  
    },  
}
```

- (c) Open the RDS interface in your browser: [AWS RDS Home](#)
- (d) Select the DB created for your environment
- (e) Under Connectivity Security
- (f) In settings.py replace the ‘HOST’ value with the endpoint value.
- (g) Ensure the ‘PORT’ also matches

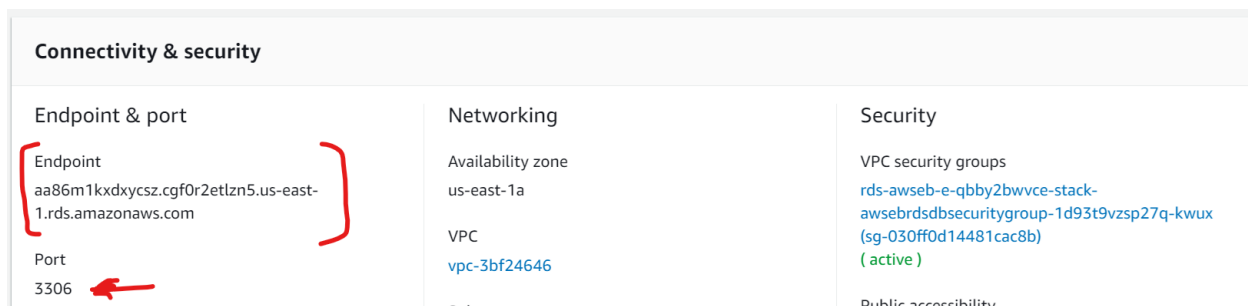


Figure 13: Get the Endpoint Information for Django DB Config

- (h) Click on the Configuration Tab
- (i) Ensure the DB name matches the ‘NAME’ field.


Instance			
Configuration	Instance class	Storage	Performance Insights
DB instance ID aa86m1kxdxycsz	Instance class db.t2.micro	Encryption Not enabled	Performance Insights enabled No
Engine version 8.0.20	vCPU 1	Storage type General Purpose (SSD)	
DB name ebdb 	RAM 1 GB	IOPS -	

Figure 14: Getting the DB name from RDB Instance

(j) Save the settings.py file

3. Adding

- (a) From the Connectivity Settings tab under the database, click on the “VPC Settings Groups” link
- (b) Click on the security group ID link for the database.


Security Groups (1/1) Info					
<input type="text" value="Filter security groups"/>					
search: sg-030ff0d14481cac8b <input type="text" value="X"/> <input type="button" value="Clear filters"/>					
<input checked="" type="checkbox"/>	Name	Security group ID	Security group name	VPC ID	Description
<input checked="" type="checkbox"/>	-	 sg-030ff0d14481cac8b	rds-awseb-e-qbby2bw...	vpc-3bf24646 VPC ID	Security group for RDS...

Figure 15: Navigate to the Security Group Details

- (c) Under “Inbound Rules” click on the “Edit Inbound Rules”

Inbound rules (1)				
 <input type="button" value="Edit inbound rules"/>				
Type	Protocol	Port range	Source	Description - optional
MYSQL/Aurora	TCP	3306	sg-04af385d821d28235 / awseb-e-qbby2bwvce-stack-AWSEBSecurityGroup-1TBNL16S94WVK	-

Figure 16: Select Edit Inbound Rules from Security Rules Page

- (d) Click “Add Rule”
 - i. Type: MYSQL/Aurora
 - ii. Source: My IP
- (e) Click “Save Rule”

Edit inbound rules [Info](#)

Inbound rules control the incoming traffic that's allowed to reach the instance.

Type Info	Protocol Info	Port range Info	Source Info	Description - optional Info	
MYSQL/Aurora	TCP	3306	Custom		Delete
				sg-04af385d821d28235	
MYSQL/Aurora	TCP	3306	My IP		Delete
				75.128.89.116/32	
Add rule					

NOTE: Any edits made on existing rules will result in the edited rule being deleted and a new rule created with the new details. This will cause traffic that depends on that rule to be dropped for a very brief period of time until the new rule can be created.

Cancel Preview changes **Save rules**

Figure 17: Configuring Inbound Rules for the Database

4. Importing the MSQL connector

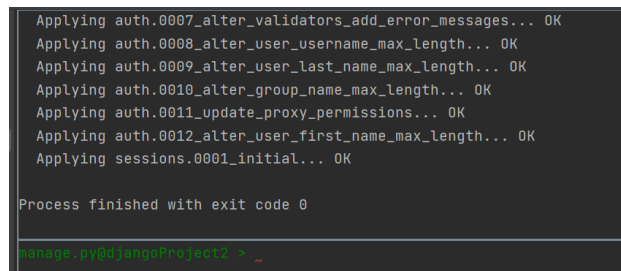
- (a) Open the `__init__.py` file inside the `djangoProject2` folder
- (b) Add the following lines:

```
import pymysql
pymysql.install_as_MySQLdb()
```

5. Testing the connection Locally

- (a) From the project root, delete the `db.sqlite3`. While not necessary, this will make sure the next steps are talking to the AWS RDB.
- (b) From the Django terminal, run: ***makemigrations***

- (c) Run: **migrate**
- (d) This should build the tables in the RDB database for the Django project.
- (e) If there are errors now, debug/fix them as they will prevent creating/updating tables from the local Django console and the deployment will not work.



```
Applying auth.0007_alter_validators_add_error_messages... OK
Applying auth.0008_alter_user_username_max_length... OK
Applying auth.0009_alter_user_last_name_max_length... OK
Applying auth.0010_alter_group_name_max_length... OK
Applying auth.0011_update_proxy_permissions... OK
Applying auth.0012_alter_user_first_name_max_length... OK
Applying sessions.0001_initial... OK

Process finished with exit code 0

venv@ip-10-10-10-10: ~$
```

Figure 18: Successful Connection and Migration of Database

6. Creating an Admin user

- (a) From the Django terminal, run: **createsuperuser**
 - i. Username: admin
 - ii. Email: <some email address>
 - iii. Password: <password you will remember>
- (b) Run: **makemigrations**
- (c) Run: **migrate**
- (d) Run: **collectstatic**
 - i. Answer 'yes' to overwrite the files
- (e) From the Terminal, run: **eb deploy**

7. Testing the Admin interface

- (a) Launch your environment endpoint (can be found on the [AWS EB Home page](#))
- (b) Append /admin/ to the url
- (c) You should be greeted with the admin interface

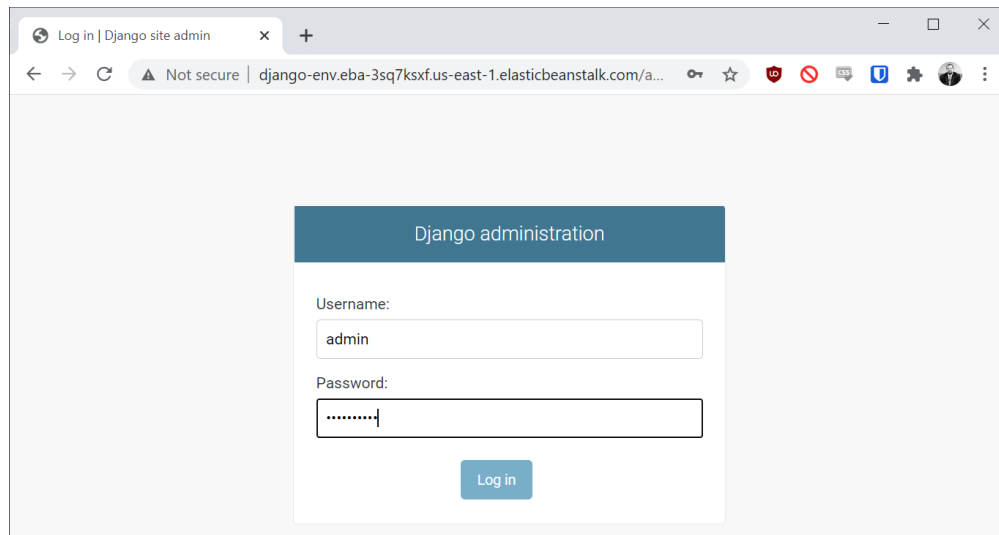


Figure 19: Django Admin on AWS

- (d) Enter the username and password created in the step above and click “Log in”
- (e) If all goes well, you should be greeted with the main Administration page.

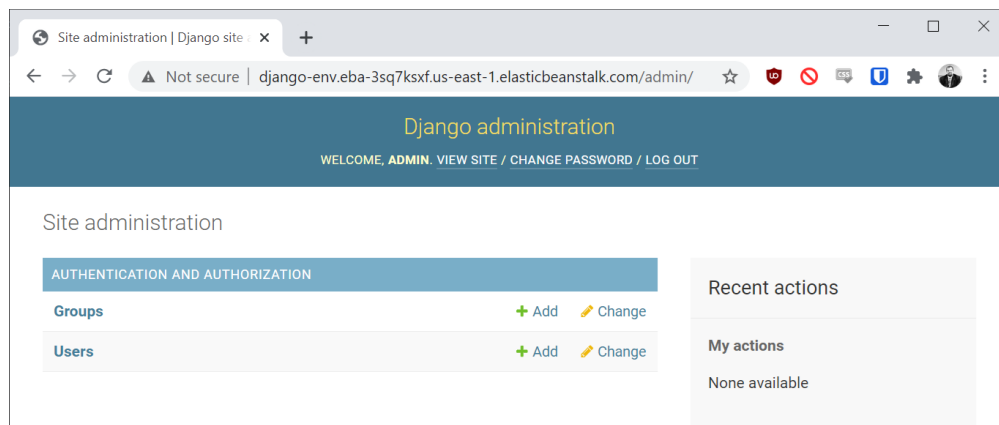


Figure 20: Django Admin Page Running in our Environment

Congratulations, you have connected your Django project both locally and deployed to the AWS RDB Database.

3.4.1 Assignment 1.1: Use the RDS CLI

Install and use the AWS RDS CLI interface to create and configure the database instead of the web portal.

4 Tutorial 2: Creating and Deploying an App

At this point, you can follow the instructions over at the [Django Tutorial](#) for creating the polls app. This section will create slightly more sophisticated App that:

- Loads car sales data from a CSV
- Uses a CSS template
- Creates a page with a Pie Chart using javascript
- Creates a page with a bar chart using javascript and a self-referencing endpoint

The rest of this tutorial assumes you have downloaded the content from the [GitHub Repo](#). Full source will not longer be provided for each item added to the project. You may continue using your currently established project and use the resources from the Repo.

4.1 Creating the First View

1. Including our Application views in the root URL:
 - (a) In the Project Folder (djangoProject2), open urls.py
 - (b) In the includes, add “include” to the from Django.urls import: from django.urls import path, include
 - (c) To the url patterns, add: path('CarData/', include('CarData.urls')),
 - (d) This points the Django Project to look at our App for additional routes
 - (e) In the App folder (CarData), create a new file called urls.py

- (f) In the new file, add the following;

```
from django.urls import path

from . import views

urlpatterns = [
    path('', views.hello, name='hello'),
]
```

- (g) This tells Django that when someone lands on <address>/CarData/ to route to our view called 'Hello'

2. Create the Hello View:

- (a) In the App folder (CarData), open views.py
(b) Add the following content to views.py

```
from django.http import HttpResponse

# Our first Endpoint
def hello(request):
    return HttpResponse('Success!')
```

- (c) From the Django terminal, run: **runserver**
(d) Launch a web browser and navigate to: <http://127.0.0.1:8000/CarData/>
(e) You should be greeted with "Success!"
(f) Stop now and fix any errors and get the URL pattern working before continuing.

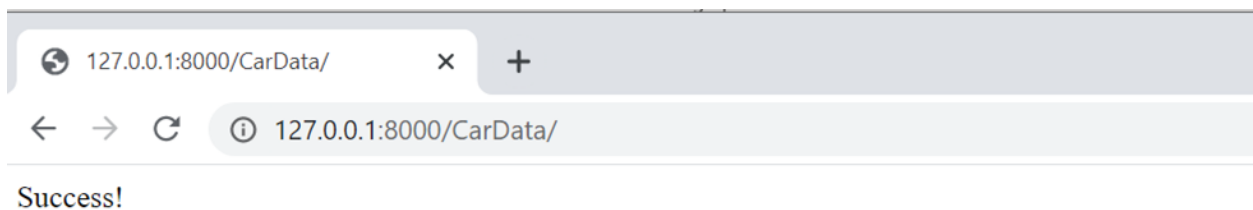


Figure 21: Successful first Endpoint Running Locally

3. Deploying the Project to AWS

- (a) From the Terminal, run: ***eb deploy***
- (b) Navigate to the environment endpoint
- (c) Note when Debug is turned on, you will get a list of the currently installed URLs if you land on one that does not exist:

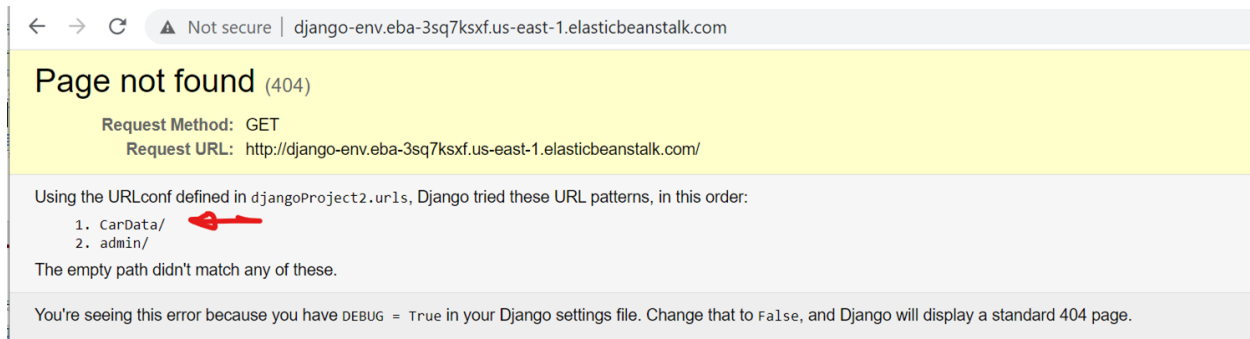


Figure 22: Using DEBUG to Help Navigate Django

- (d) Append `/CarData/` to your endpoint
- (e) You should be greeted with “Success!”

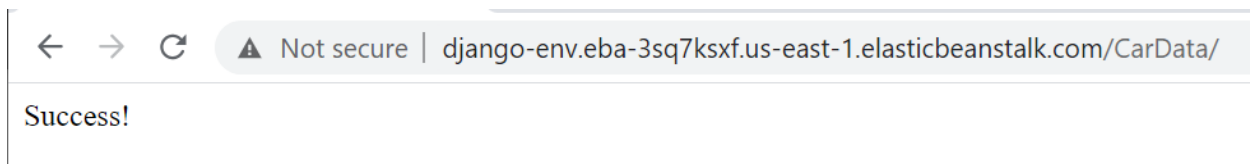


Figure 23: Successfully Deployed to AWS

Congratulations you have created and deployed your first View

4.1.1 Assignment 2.1: Add a View that Renders A Template

Update the hello view to use `render()` instead of the `HttpResponse()` function and pulls in a simple HTML template stored in the templates folder.

Hint: Modify the form in the instructions from the [Django Tutorial: Writing your first Django app, part 3](#) page.

4.2 Adding and Using Models

In this tutorial, we will be using data from the [Kaggle: US Cars Dataset](#). A copy is provided in the root folder of the GitHub project or can be downloaded directly from Kaggle. The data is formatted in the way below:

	A	B	C	D	E	F	G	H	I	J	K	L	M
1		price	brand	model	year	title_status	mileage	color	vin	lot	state	country	condition
2	0	6300	toyota	cruiser	2008	clean vehicle	274117	black	jtezu11f88k007763	159348797	new jersey	usa	10 days left
3	1	2899	ford	se	2011	clean vehicle	190552	silver	2fmdk3gc4bbb02217	166951262	tennessee	usa	6 days left
4	2	5350	dodge	mpv	2018	clean vehicle	39590	silver	3c4pdcgg5jt346413	167655728	georgia	usa	2 days left
5	3	25000	ford	door	2014	clean vehicle	64146	blue	1ftfw1et4efc23745	167753855	virginia	usa	22 hours left

Figure 24: CSV Data to be Loaded

In this exercise, we will create relational database elements in the following organization. A make and model is an abstract of an individual vehicle. A specific vehicle is one instance of a Make/Model. An individual vehicle can have multiple sales.

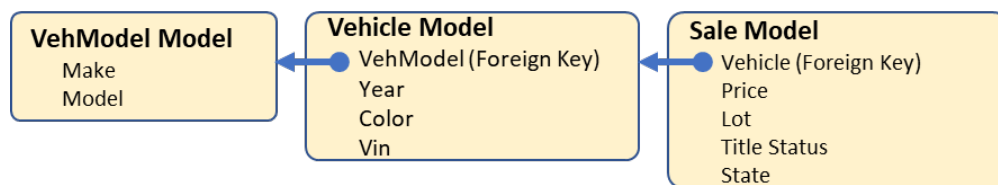


Figure 25: Database Created for this Exercise

1. Creating New Models

- In the App folder (CarData), open models.py
- Add the following models describing our database:

```
from django.db import models

# Database model for a Vehicle Model
class VehModel(models.Model):
    brand = models.CharField(max_length=200)
    model = models.CharField(max_length=200)

    def __str__(self):
        return self.brand + ', ' + self.model
```

```
# Database model for a Specific Vehicle
class Vehicle(models.Model):
    VehModel = models.ForeignKey(VehModel, on_delete=models.CASCADE)
    title_status_salvage = models.BooleanField(default=False)
    year = models.IntegerField(default=1900)
    color = models.CharField(max_length=200)
    vin = models.CharField(max_length=200)

    def __str__(self):
        return self.VehModel.__str__() + ' Vin:' + self.vin

# Database model for a Specific Sale transaction
class Sale(models.Model):
    Vehicle = models.ForeignKey(Vehicle, on_delete=models.CASCADE)
    mileage = models.IntegerField(default=0)
    lot = models.CharField(max_length=200)
    state = models.CharField(max_length=200)
    country = models.CharField(max_length=200)
    price = models.IntegerField(default=0)

    def __str__(self):
        return self.Vehicle.__str__() + ' Lot:' + self.lot
```

- (c) Note the definition of ‘__str__’ override. This is so we can return pretty printed data in the admin terminal later.

2. Registering the models with the Admin

- (a) In the App folder (CarData), open admin.py
- (b) Add the following that describes and registers our new models with the database:

```
from django.contrib import admin
from .models import VehModel, Vehicle, Sale

# Create an Admin Interface for Vehicle Model
class AdminVehModel(admin.ModelAdmin):
    fields = ['brand', 'model']

# Create an Admin Interface for Vehicle
class AdminVehicle(admin.ModelAdmin):
    fields = [
        'VehModel',
        'title_status_salvage',
        'year',
        'color',
        'vin',
    ]
```



```
# Create an Admin Interface for Sale
class AdminSale(admin.ModelAdmin):
    fields = [
        'Vehicle',
        'mileage',
        'lot',
        'state',
        'country',
        'price'
    ]

# Register all three with the Admin Interface
admin.site.register(VehModel, AdminVehModel)
admin.site.register(Vehicle, AdminVehicle)
admin.site.register(Sale, AdminSale)
```

2. Creating a View to Add a single entry

- (a) This View will create a single vehicle, A 2013 Ford Ranger with a single sale and save it to the Database.
- (b) In the App folder (CarData), open views.py
- (c) Add Imports to support our new view:

```
from django.shortcuts import render
from .models import VehModel, Vehicle, Sale
```

- (d) Add the following new view:

```
# Adding our first vehicle
def createOne(request):

    #Create the Model (Since we are starting with an Empty DB)
    theModel = VehModel(brand="Ford",model="Ranger")
    theModel.save()

    #Create a new Vehicle
    theVehicle = Vehicle(
        VehModel = theModel,
        title_status_salvage = False,
        year = 2013,
        color = "Blue",
        vin = "1FTCR11AXDUB12345")
    theVehicle.save()

    #Generate a new Sale entry
    theSale = Sale(
        Vehicle = theVehicle,
```

```
        mileage = 12345,
        lot = "9607-892",
        state = "Michigan",
        country = "USA",
        price = 1900)
    theSale.save()

    # get the total number of vehicles from the DB
    count = len(Vehicle.objects.all())

    # generate a response to the browser.
    return HttpResponse("Added: " + str(theVehicle) + " "
        + str(theSale) + "<br>Total Vehicles in DB: " + str(count))
```

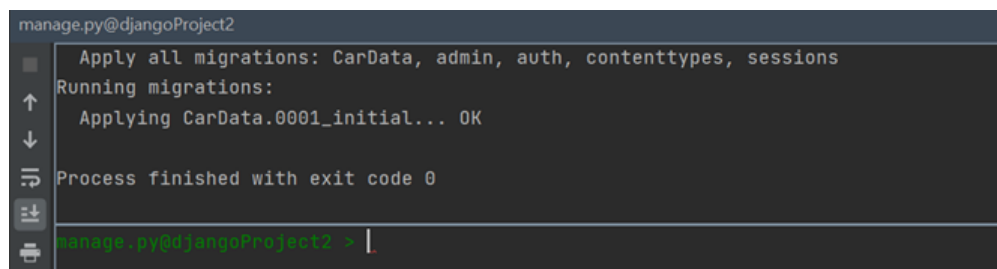
3. Registering the new View

- (a) In the App folder (CarData), open `urls.py`
- (b) Add the following line to the `urls`:

```
path('csvReader', views.csvReader, name='csvReader'),
```

4. Migrate the database

- (a) From the Django Terminal, run: ***makemigrations***
- (b) Run: ***migrate***



```
manage.py@djangoProject2
Apply all migrations: CarData, admin, auth, contenttypes, sessions
Running migrations:
  Applying CarData.0001_initial... OK
Process finished with exit code 0
manage.py@djangoProject2 > |
```

Figure 26: Successful Migration Adding the Vehicle Models

- (c) From the Terminal, run: ***eb deploy***

5. Inspecting through Admin and running the new view

- (a) Navigate to your environment endpoint

- (b) Add '/admin/' to the url
- (c) Login using the credentials created earlier
- (d) You should now see the new tables for the VehData, Vehicle, and Sale.

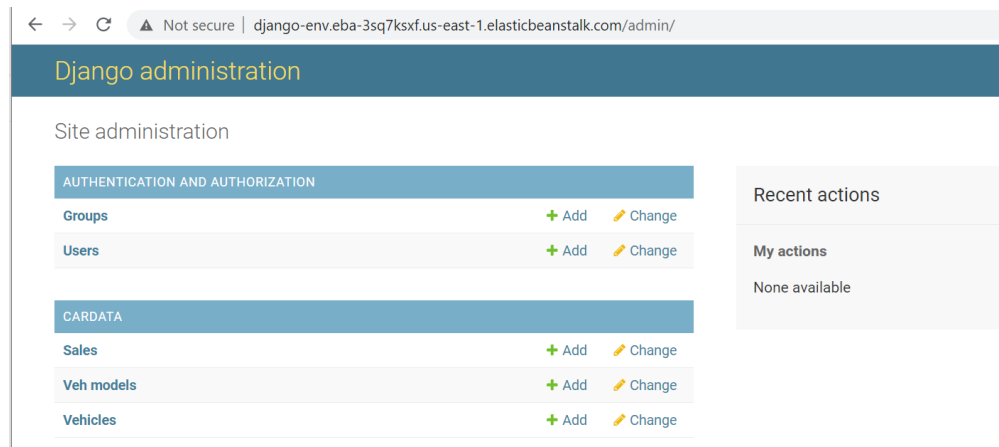


Figure 27: Models Registered in the Admin Interface

- (e) If you enter them, however, they are empty.

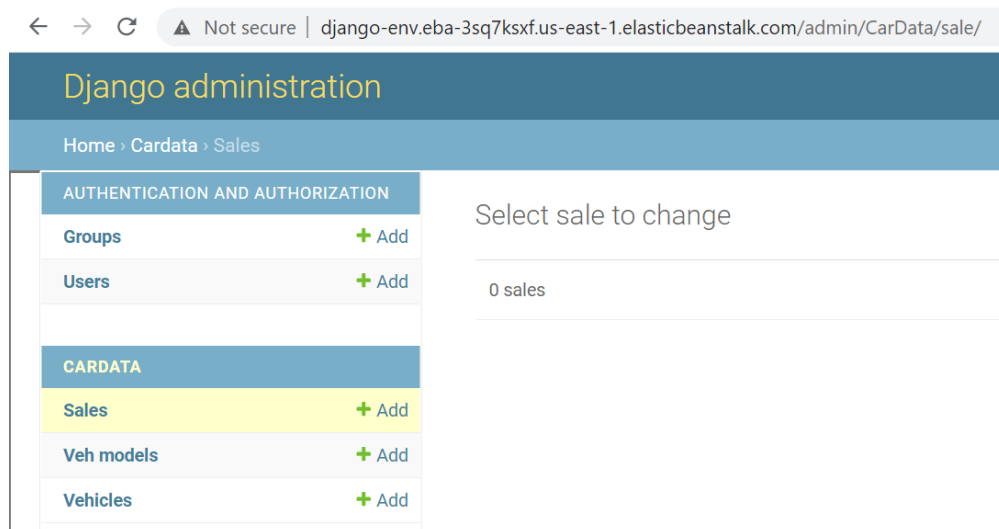


Figure 28: Empty Sales Table

6. Using a view to create new DB entries

- (a) Navigate to your URL endpoint and add '/CarData/createOne'
- (b) You should see a summary indicating the entry was created from our new view

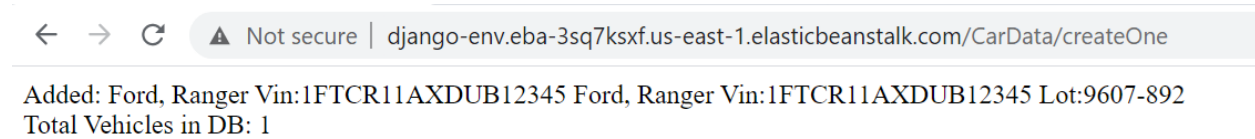


Figure 29: Successfully Added one entry by the View

- (c) Go back to the admin panel `<env endpoint>/Admin/`
- (d) Now inspecting into the vehicles, Models, or Sales, we should see our new entry.

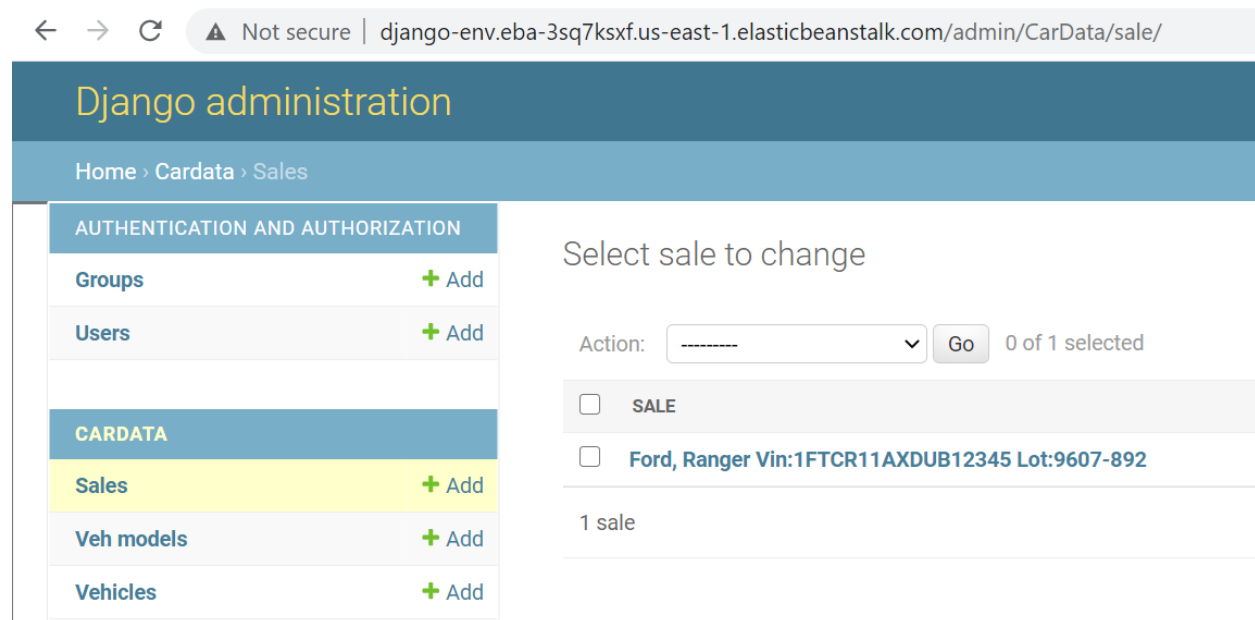


Figure 30: One Entry Programmatically Added to the DB

Congratulations, you have successfully used views to interact with the database.

4.2.1 Assignment 2.2: Modify and Migrating Models

Edit the Sale Model to include the condition field in the CSV.

- Add the 'Condition' textfield to the sale model.
- Update and Migrate the database.
- Update the createOne view to add the condition.
- Register the changes in the admin console

4.2.2 Assignment 2.3: Create a View with a Form

Create a new view with a form that lets you add a new record directly from the web.

Hint: Modify the form in the instructions from the [Django Tutorial: 4 Forms](#) page.

4.3 Loading Data from CSV

There are lots of ways to load data into the database. For example, you can create SQL commands to import them from static files or restore and migrate another database. For simplicity, in this example, we will write a basic view that when called will iterate through the CSV provided from the Car Sales data and create the entries. The source CSV will be uploaded through the project deployment. There are many resources available to help you design an upload mechanism but those are not covered here. For example: [How to Upload Files With Django](#)

1. From the github copy the USA_cars_datasets.csv into the root directory.
2. Creating the ReadCSV view
 - (a) In the App folder (CarData), open views.py
 - (b) Paste in the function for csvReader from the github project
 - (c) Update the imports as necessary
3. Adding the URL
 - (a) In the App folder (CarData), open urls.py
 - (b) To the urlpatterns, add the following:

```
path('csvReader', views.csvReader, name='csvReader'),
```

- (c) Save the file
- (d) From the terminal, run: ***eb deploy***
- (e) Navigate to your URL endpoint and add '/CarData/csvReader'
- (f) This will take several seconds as the system ingests the new models.
- (g) Note, this can also be run locally by using Django> runserver and pointing to the local server instead.
- (h) Once complete, navigate back to the admin url and you should see all the data loaded into the database:

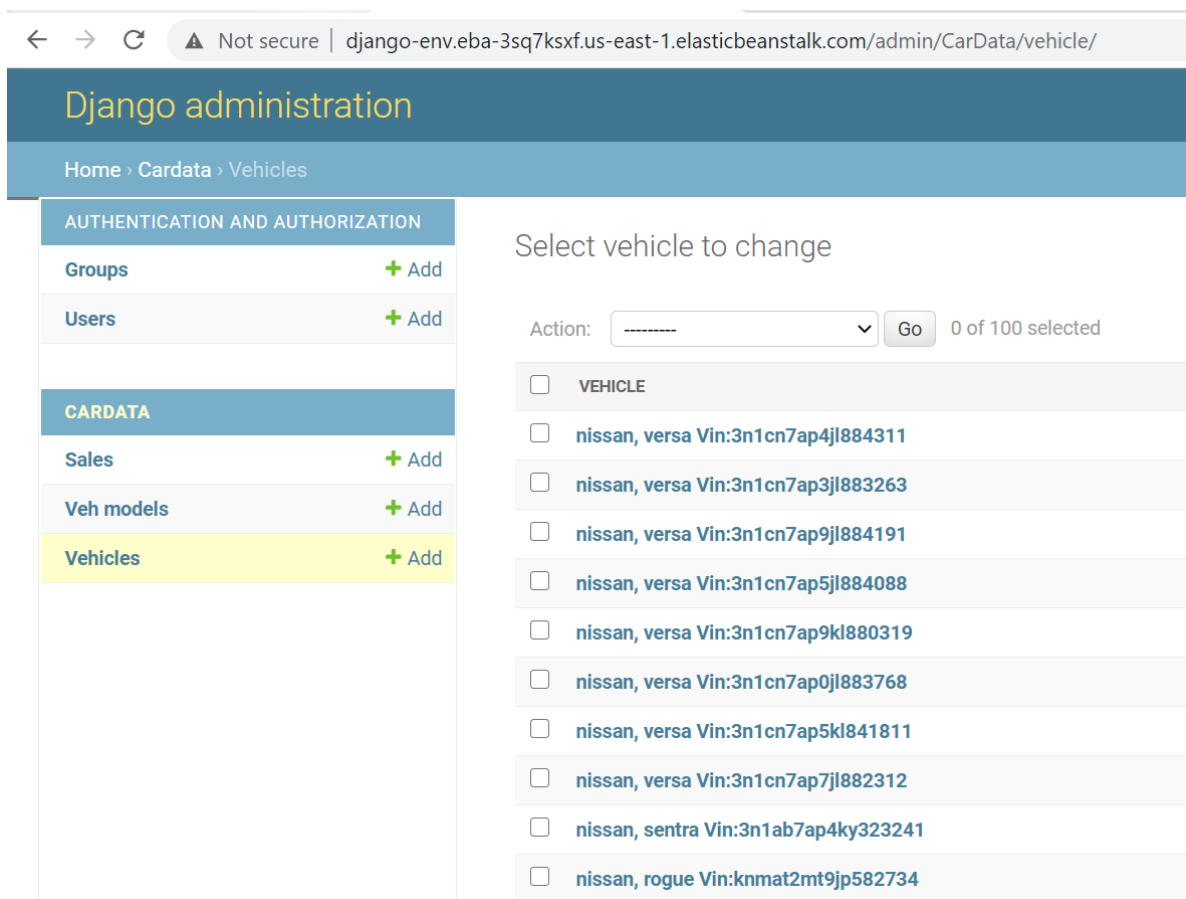


Figure 31: Database Populated

Congratulations, you have successfully loaded some useful data into your database.

4.4 Using a Template and Creating Pie-Chart

In this section we explain templates and then make use of a Pie-chart Javascript script to display data returned from the database.

1. From the github copy the contents of the templates folder into your projects Templates Folder.

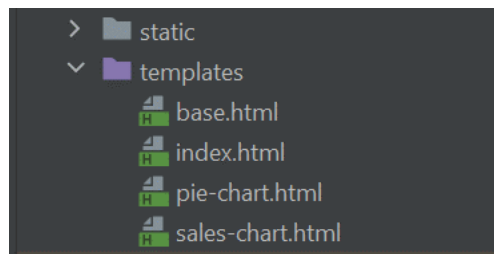


Figure 32: Templates loaded into the templates folder

- (a) The templates pull in CSS templates. Special `{%}` and `{% endblock %}` fields are used to reference functions that will be added by Django when render is called.
- (b) For example, inspect the base.html. The highlighted regions will be replaced by Django's rendering engine by the child template.

```
and is wrapped around the whole page content, except for the footer in this exam
<div class="w3-content" style="..." >

  <!-- Header -->
  <header class="w3-container w3-center w3-padding-32">
    <h1><b>{% block title %}{% endblock title %}</b></h1>
    <p>{% block subtitle %}{% endblock subtitle %}</p>
  </header>

  <!-- Grid -->
  <div class="w3-row" align="center">
    {% block content %}{% endblock content %}
  </div><br>

  <!-- END w3-content -->
</div>
```

Figure 33: Inspecting the base.html Template

- (c) Looking inside `pie-chart.html`, the replacement regions are defined. The content between these blocks will be inserted into base at runtime:

```
{% extends 'base.html' %}
{% block title %}Cars by Color{% endblock title %}
{% block subtitle %}Number of Cars Sold by Color{% endblock subtitle %}

{% block content %}
<div id="container" style="...">
  <canvas id="pie-chart"></canvas>
</div>

<script src="https://cdn.jsdelivr.net/npm/chart.js@2.9.3/dist/Chart.min.js"></script>
</script>
```

Figure 34: Inspecting the `pie-chart.html` Template

- (d) Further down, you can see the fields that will be replaced by the python variables passed into the `render()` function call at the end of the view.

```
var config = {
  type: 'pie',
  data: {
    datasets: [{
      data: {{ data|safe }},
      backgroundColor:
        {{ colors|safe }},
      label: 'Population'
    }],
    labels: {{ labels|safe }}
  },
  options: {
    responsive: true
  }
};

window.onload = function() {
  var ctx = document.getElementById('pie-chart').getContext('2d');
  window.myPie = new Chart(ctx, config);
};
```

Figure 35: Templated data members to be replaced by Django during Rendering.

2. Creating the `pie_chart` view

- (a) In the App folder (CarData), open views.py.
- (b) Paste in the function for pie_chart() from the github project.
- (c) This view pulls all the vehicles from the database and then puts counts into buckets depending on the color of the vehicle
- (d) The page then uses a bit of javascript to render that into a nice Pie Chart.
- (e) Update the imports as necessary.

3. Adding the URL

- (a) In the App folder (CarData), open urls.py
- (b) To the urlpatterns, add the following: `path('pie-chart', views.pie_chart, name='pie-chart')`,
- (c) Save the file
- (d) From the terminal, run: ***eb deploy***
- (e) Navigate to your URL endpoint and add `'/CarData/pie-chart'`
- (f) Note, this can also be run locally by using Django> ***runserver*** and pointing to the local server instead.

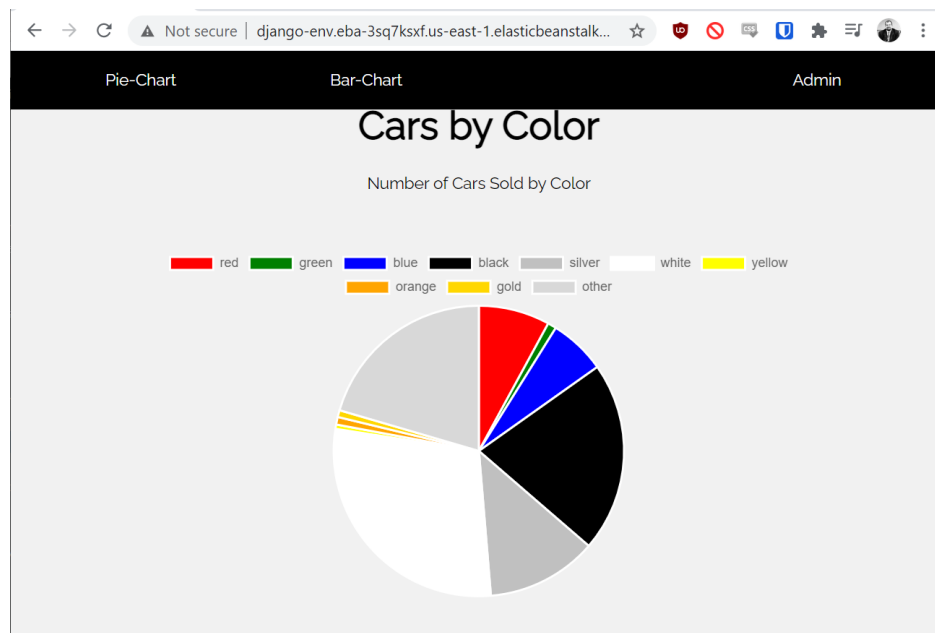


Figure 36: Completed Rendered Page with Pie Chart

4. Inspect the rendered page and see the source complete with data from the python code:

```
<script src="https://cdn.jsdelivr.net/npm/chart.js@2.9.3/dist/Chart.min.js"></script>
<script>
  var config = {
    type: 'pie',
    data: {
      datasets: [{
        data: [198, 24, 157, 528, 308, 718, 12, 21, 19, 511],
        backgroundColor:
          ['red', 'green', 'blue', 'black', 'silver', 'white', 'yellow', 'orange',
'gold'],
        label: 'Population'
      }],
      labels: ['red', 'green', 'blue', 'black', 'silver', 'white', 'yellow', 'orange', 'gold',
'other']
    },
    options: {
      responsive: true
    }
  };

  window.onload = function() {
    var ctx = document.getElementById('pie-chart').getContext('2d');
    window.myPie = new Chart(ctx, config);
  };
</script> == $0
<!-- END GRID -->
```

Figure 37: Source from rendered page with populated data fields.

Congratulations, You have successfully used templates, views, and javascript to make dynamic interface to the data.

4.4.1 Assignment 2.4: Create a view to display a Pie Chart of total sales by Manufacturer

Modify the view created in this exercise to display pie slices that represent the total sales for the top 9 manufacturers. Hint: You will need to iterate the Sale objects instead of the Vehicle. Note: this may take a long time if running locally.

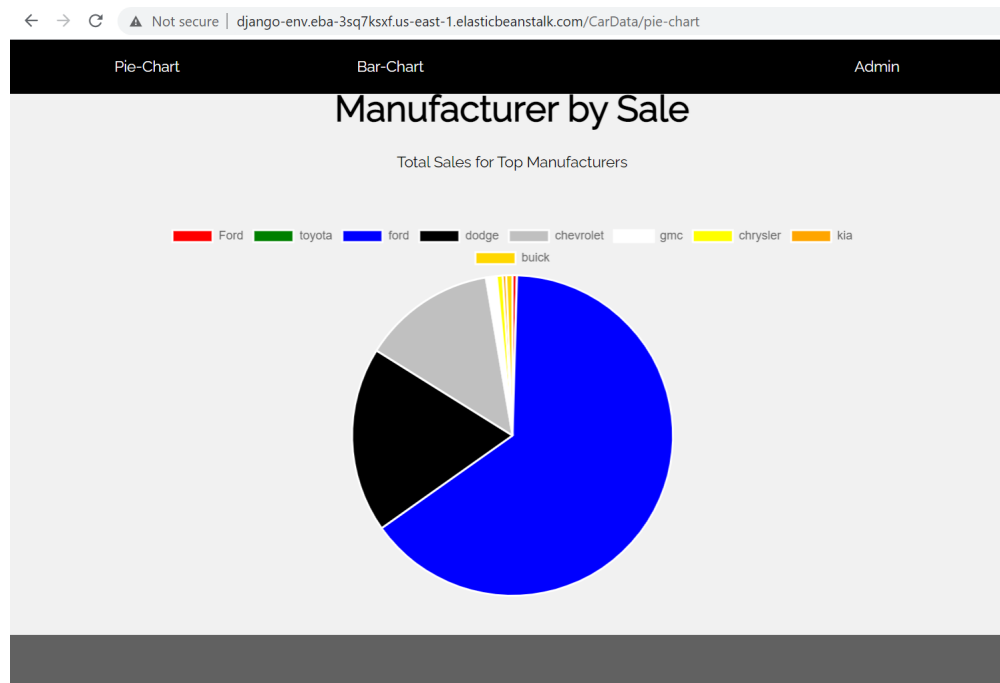


Figure 38: Modify the View to Render Sales by Manufacturer

4.4.2 Assignment 2.5: Modify the Template to use Modal Restaurant Template

Modify the html templates to use the [Modal Restaurant Template](#) or another template from the [W3c Templates](#) page.

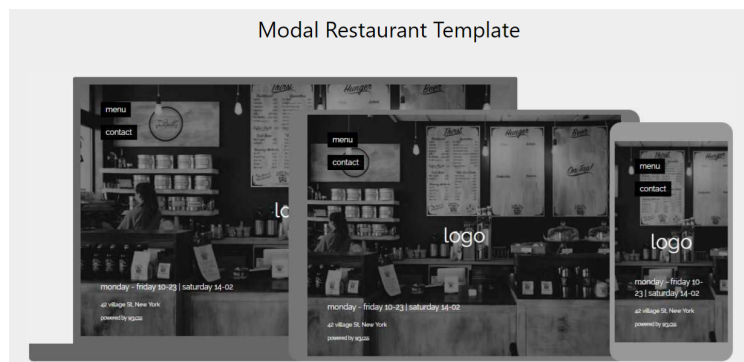


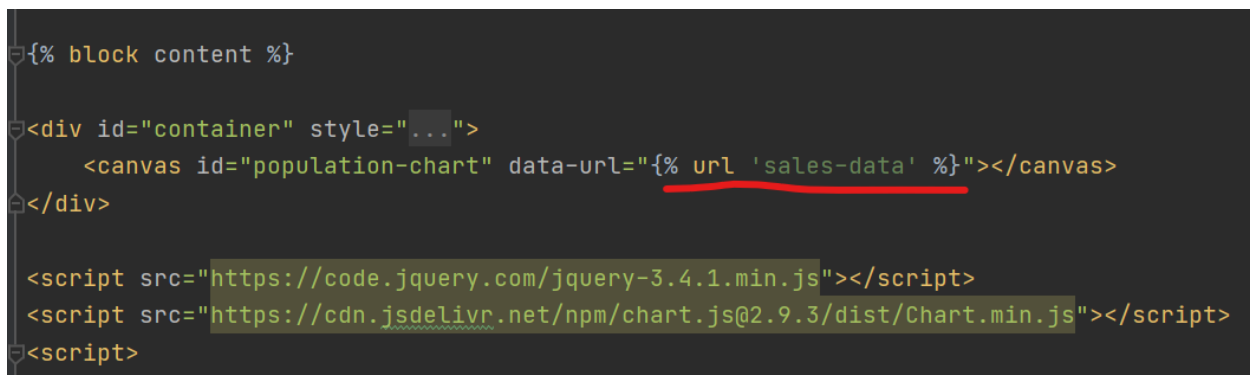
Figure 39: Modify templates to use new CSS template.

4.5 Creating Bar-Chart and Self Referencing from JS

In this section we will create a served page that points to a separate URL on the server to serve up the data in a JSON format to the javascript file. Thereby making the page more responsive when the backend needs to operate longer on the data.

1. Inspect the sales-chart template

- (a) Notice that the data-url argument for the javascript points to the Django url for sales-data



```
{% block content %}
<div id="container" style="...">
  <canvas id="population-chart" data-url="{% url 'sales-data' %}"></canvas>
</div>

<script src="https://code.jquery.com/jquery-3.4.1.min.js"></script>
<script src="https://cdn.jsdelivr.net/npm/chart.js@2.9.3/dist/Chart.min.js"></script>
<script>
```

Figure 40: sales-chart Template Referencing sales-data url

- (b) This lets the page load while the back-end is gathering and preparing the data.
- (c) This can method can be useful when fetching the data takes a long time. You could add a cycling animation while the data loads or serve other content dynamically as the user scrolls over it to avoid unnecessary hits to the database.

2. Creating the pie_chart view

- (a) In the App folder (CarData), open views.py.
- (b) Paste in the function for sales_chart(request) and sales_data(request) from the github project.
- (c) Update the imports as necessary.
- (d) Sales_chart renders the template with the javascript that renders the graph.

- (e) The sales_chart page will make a second request to sales_data.
- (f) Sales_Data then generates a json formatted data element and serves it back to the javascript request.
- (g) The javascript then renders the data into a nice bargraph. Adding the URL
- (h) In the App folder (CarData), open urls.py
- (i) To the urlpatterns, add the following lines:

```
path('sales-chart/', views.sales\_chart, name='sales-chart'),  
path('sales-data/', views.sales\_data, name='sales-data'),
```

- (j) Save the file
- (k) From the terminal, run: **eb deploy**
- (l) Navigate to your URL endpoint and add '/CarData/sales-data/'
- (m) You will get the raw json data which takes a few seconds.
- (n) Now navigate to your URL endpoint and add '/CarData/sales-chart/'
- (o) Note, its not recommended to run this one locally as the database will take a long time to respond to all the queries.

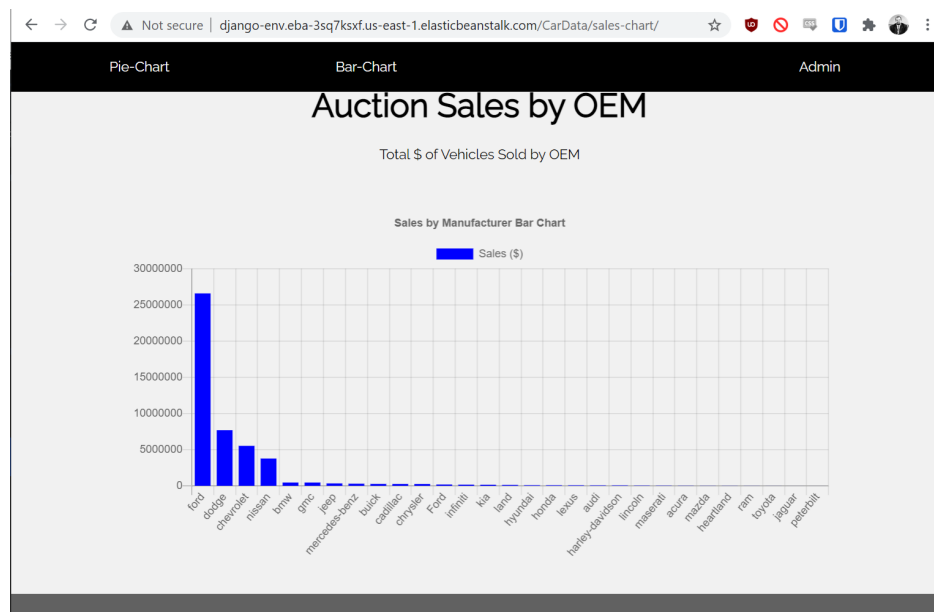


Figure 41: The Rendered Sales-Chart Page

4.5.1 Assignment 2.6: Modify the View to serve a Dynamic Picture

Modify the sales-chart view to display an image along with the data. The image should be source from another API and served to the page through a JSON object.

5 Cleaning Up

This section includes instructions on how to tear down your environment to minimize charges to your account:

- Unfortunately EC3 instances cannot be terminated individually. The load balancer will spin up new instances to replace the terminated nodes.
- Further, this means there is no way to “park” an environment in an offline state. Removing the project/environment will remove the database and all.
- Once removed, the environment cannot be restarted. Any data added to the database must be rebuilt.

1. To remove an environment with the cli:

- (a) From the terminal run: ***eb terminate django-env***
- (b) The environment will tear down databases and EC3 instances which will take several minutes
- (c) You can check the status of the removal by running: ***eb status***

2. To remove an application from the AWS Web interface:

- (a) Navigate to the [EB Applications Page](#)
- (b) Select the project to delete with the radio button
- (c) Click Actions -> Delete Application
- (d) Type the name of the application and confirm the delete.
- (e) The environment will tear down databases and EC3 instances which will take several minutes

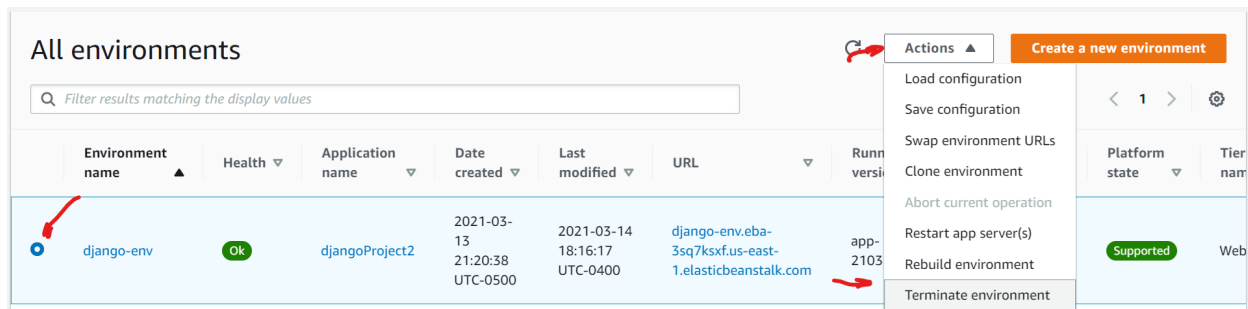


Figure 42: Removing an Applicaiton through the AWS Web Management

3. To remove an application from the AWS Web interface:

- Navigate to the [EB Environments Page](#)
- Select the project to delete with the radio button
- Click Actions -> Terminate environment
- The environment will tear down databases and EC3 instances and databases which will take several minutes

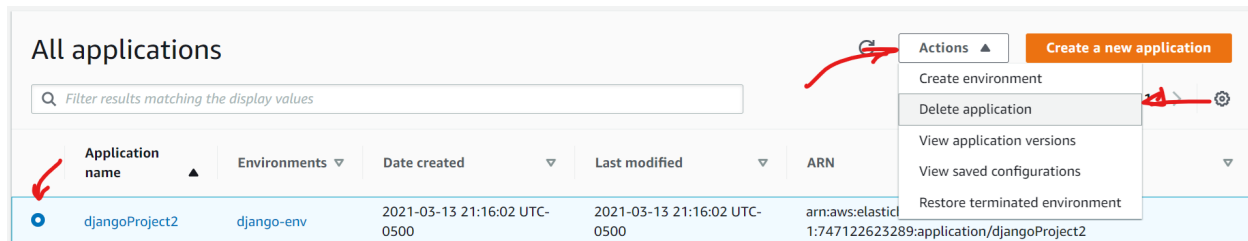


Figure 43: Removing an Environment through the AWS Web Management