A) Create a repository and workspace

- Create a repository in Github – using the Code Institute Gitpod Full template.
- Click on the green Gitpod button to load the repository workspace in Gitpod.

B) Install Django

- Install Django and Gunicorn (Gunicorn is the server that will be used to run Django on) Heroku:
    - Type the following command in the Gitpod terminal:
      pip3 install Django==3.2 gunicorn

C) Add python code to gitignore file, that are not required in version control

- Add the following code:
    - *.sqlite3
    - *.pyc
    - __pycache__

D) Check that project installed ok by running it on the server

- Type the following code into the command line:
    - python3 manage.py runserver
    - Open up Port 8000
    - Server should say that "The install worked successfully"

E) Run initial migrations

- Type the following code into the command line:
    - python3 manage.py migrate

F) Create a Superuser

- Type the following code into the command line:
    - python3 manage.py createsuperuser
    - Add username, email and password

G) Make initial commit to Github repository:

- Type the following code into the command line:
    - git add .
    - git commit –m "Initial commit"
    - git push

H) Install allauth for account registrations:

- Type the following code into the command line:
    - pip3 install django-allauth==0.41.0
    - In settings.py, at the top of the file, add the following code: import os
    - Add authentication backends code in settings.py, under "templates"
    - Below authentication backends, add code: "SITE_ID = 1"
    - Install allauth apps (messages, staticfiles, sites, allauth, account and socialaccount) under "installed apps"
    - All allauth urls to project level urls.py file

- Run migrations to update the database after the installation of new apps.
- Add "Email backend" setting to settings.py
- Add account authentication settings to tell allauth what we would allow for emails and usernames, in settings.py
- Test allauth is working properly by opening up the server and running /accounts/login – test out using the superuser account details
- Commit changes to Github

I) <u>Create a Heroku App:</u>

- Open up Heroku:
  - Select create a new app – give it a name and select a region
  - On the resources tab search for "Heroku Postgres" and use the free plan
- In the Gitpod terminal:
  - Install dj_database_url, type: pip3 install dj_database_url==0.5.0
  - Install psycopg2-binary, type: pip3 install pyscopg2-binary
  - Freeze requirements to ensure Heroku installs all app requirements when the project is deployed, type: pip3 freeze > requirements.txt

J) <u>Connect project to Heroku Postgres, in settings.py:</u>

- Import dj_database_url
- Add the following code – to allow project to connect to either the SQLite database or Heroku Postgres, depending on the development environment:

```
if 'DATABASE_URL' in os.environ:
    DATABASES = {
        'default': dj_database_url.parse(os.environ.get('DATABASE_URL'))
    }

else:
    DATABASES = {
        'default': {
            'ENGINE': 'django.db.backends.sqlite3',
            'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
        }
    }
```

K) <u>Connect Gitpod to Heroku Postgres:</u>

- In Gitpod terminal workspace, add a variable called "DATABASE_URL" and take the value from the Heroku app "DATABASE_URL", under settings.py > Config Vars

L) <u>Dump key data into JSON files:</u>

- Comment out the code to connect Gitpod to Heroku Postgres database and leave the code to connect Gitpod to the SQLite database, in settings.py and save.
- Dump the data from "categories" and "bouquets" model into a JSON file, using the following command:

  python3 manage.py dumpdata [app or model name] > [filename].json

M) Load data onto the Heroku Postgres database:

- Uncomment out the code to connect Gitpod to Heroku Postgres in settings.py and save.
- Upload "categories" and "bouquets" JSON files to Heroku Postgres – uploading the categories file first followed by the bouquets file, using the following command:

  python3 manage.py loaddata [filename].json

- Make migrations, by typing in the Gitpod terminal: python3 manage.py makemigrations
- Run migrations , by typing in the Gitpod terminal: python3 manage.py migrate
- Upload JSON files to Postres, using the following command:

  python3 manage.py loaddata [filename.json]

  If it all works well, should see following outcome in the Gitpod terminal:
  e.g. Installed x object(s) from x fixture(s)

N) Create Superuser again (as now Gitpod is connected to Postgres database):

- Use the following command: python3 manage.py createsuperuser

O) Remove DATABASE_URL from Gitpod as data transfer to Heroku has been completed:

- Remove DATABASE_URL variable from Gitpod workspace settings.

P) Install Gunicorn:

- Type the following command: pip3 install Django==3.2 gunicorn
- Freeze this in the requirements file, by typing: pip3 freeze requirements.txt

Q) Create Procfile to tell Heroku to create a web dyno to run Gunicorn and serve the Django app:

- Right click in the workspace, where all the files are listed and create a file in the main directory called "Procfile" (capital P)
- Add the following code in the file:

  web: gunicorn blossoming_bouquets.wsgi:application

R) Set up Heroku app and Gitpod settings:

- In Heroku app config vars – add a variable called disable collectstatic (as below), to stop Django collecting static files when it comes to deployment:

  Key: DISABLE_COLLECTSTATIC, Value: 1

- In Gitpod settings.py – set up allowed hosts as Heroku and Gitpod, as below:

  ALLOWED_HOSTS = ['[PROJ-NAME].herokuapp.com', 'localhost'].

S) <u>Add Heroku CLI as the deployment method in the Heroku app and deploy to Heroku</u>

- In the Heroku > open app > 'deploy' tab
- Scroll to the 'Deployment method' and select 'Heroku CLI'
- In the Gitpod terminal type:
    - heroku login –i
    - Enter your Heroku email address and password
    - To link the Heroku app to the Gitpod terminal, type the following command: heroku git:remote –a your-app-name (this only needs to be done the first time you connect your Heroku app to Gitpod)
    - In the Gitpod terminal type:
        - git add .
        - git commit –m "Set up Heroku deployment settings – add gunicorn, procfile, disable collectstatic and allowed hosts"
        - git push (push to Github)
        - git push heroku main (to push to heroku)

  At this stage, by clicking on the Heroku app link – the site should have all the key items displaying however the static files should not be applied (as we disabled collectstatic in the Heroku app config vars settings)

T) <u>Set the Heroku app so that it deploys automatically when pushed to Github:</u>

- In Heroku > open app > deploy tab – select "Github" as the deployment method
- Under "Connect to Github" – search for the appropriate Github repository - in this case, "blossoming_bouquets", select it and click "connect"
- Under "Automatic deploys" select "Enable Automatic Deploys"

U) <u>Remove secret key and set debug in Gitpod settings:</u>

- Can use a website e.g miniwebtool.com to generate a secret key and copy it
- In Heroku app config vars add a key called "SECRET_KEY" and copy the secret key generated as the value.
- Remove "SECRET_KEY" value in Gitpod settings.py and replace with value as below:

  SECRET_KEY = os.environ.get('SECRET_KEY', '')

- Replace "DEBUG = True" in Gitpod settings.py, with value as below:

  DEBUG = 'DEVELOPMENT' in os.environ

- Commit changes to Github (this should automatically update on Heroku app as well).

V) <u>Create Amazon Web Services (AWS) S3 bucket to store media and static files:</u>

- Create an account with AWS and login
- Search for S3
- Create a bucket for your project and ensure it is set to public to allow public access to the static files
    - Properties > Turn on static web hosting
    - Permissions > Add CORS configuration
    - Permissions > Add bucket policy

- Permissions > Access control > Public access > set list objects permission to everyone

W) <u>Create user to access S3 bucket:</u>

- In AWS account, open IAM service
- Create a group for the user – ideally use a name that is indicative of the project
- Create a policy used to give full access to the S3 bucket and attach it to the group
- Create a user and add them to the group
- Download the csv file which contains the users access key and secret access key which will be used to authenticate them from the Django app

X) <u>Connect Django to S3 bucket and static files to S3 bucket:</u>

- In the Gitpod terminal, install boto3 and django-storages
- Freeze them into requirements.txt file so they get installed on Heroku upon deployment.
- Add “storages” to “installed apps” in settings.py
- To connect Django to S3 – add the below code in settings.py checking if there is an environmental variable called “USE_AWS” in the environment and if so defining a cache control setting, bucket configuration and static and media file settings:

```
if 'USE_AWS' in os.environ:
    # Cache control
    AWS_S3_OBJECT_PARAMETERS = {
        'Expires': 'Thu, 31 Dec 2099 20:00:00 GMT',
        'CacheControl': 'max-age=94608000',
    }

    # Bucket Config
    AWS_STORAGE_BUCKET_NAME = 'blossoming-bouquets'
    AWS_S3_REGION_NAME = 'us-west-2'
    AWS_ACCESS_KEY_ID = os.environ.get('AWS_ACCESS_KEY_ID')
    AWS_SECRET_ACCESS_KEY = os.environ.get('AWS_SECRET_ACCESS_KEY')
    AWS_S3_CUSTOM_DOMAIN = f'{AWS_STORAGE_BUCKET_NAME}.s3.amazonaws.com'

    # Static and media files
    STATICFILES_STORAGE = 'custom_storages.StaticStorage'
    STATICFILES_LOCATION = 'static'
    DEFAULT_FILE_STORAGE = 'custom_storages.MediaStorage'
    MEDIAFILES_LOCATION = 'media'

    # Override static and media URLs in production
    STATIC_URL = f'https://{AWS_S3_CUSTOM_DOMAIN}/{STATICFILES_LOCATION}/
    MEDIA_URL = f'https://{AWS_S3_CUSTOM_DOMAIN}/{MEDIAFILES_LOCATION}/'
```

- In Heroku app config vars – add the following keys: USE_AWS (value=True), AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY (values are taken from the csv file)
- Remove “disable collectstatic” variable - as now Django will collectstatic files automatically and upload them to S3.
- In the main project directory in Gitpod, create a file called “custom_storages.py” and add the below code to tell Django to use S3 to store static and media files:

```
custom_storages.py > MediaStorage > location
1   from django.conf import settings
2   from storages.backends.s3boto3 import S3Boto3Storage
3
4
5   class StaticStorage(S3Boto3Storage):
6       location = settings.STATICFILES_LOCATION
7
8
9   class MediaStorage(S3Boto3Storage):
10      location = settings.MEDIAFILES_LOCATION
```

- Commit changes to Github
- Heroku build log should show all static files were collected successfully

- AWS S3 bucket will have a static folder which will contain all the project static files

Y) <u>Add media files to S3 bucket:</u>

- Open up AWS project bucket
- Create a new folder called “media”
- Upload all images used on site and grant public read access

Z) <u>Final bits:</u>

- Add Stripe keys (obtained from stripe account > developers> API keys) to Heroku app config vars
- Create a new webhook endpoint that sends webhooks to the Heroku app rather than to the Gitpod workspace.
- Add webhook signing secret to Heroku app config vars.