

**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 and supporting libraries**

- Install Django and Gunicorn (Gunicorn is the server that will be used to run Django on) Heroku:
  - a. Type the following command in the gitpod terminal:  
`pip3 install Django==3.2 gunicorn`
- Install the main Django supporting libraries required to get the project running:
  - b. Install the library for PostgreSQL - type the following command in the gitpod terminal:  
`pip3 install dj_database_url psycopg2`
  - c. Install library for Cloudinary – type the following command in the gitpod terminal:  
`pip3 install dj3-cloudinary-storage`
- Create the requirements.txt file; this will add all the above installed packages to the file:
  - d. Type the following command in the gitpod terminal:  
`pip3 freeze --local > requirements.txt`

**C) Create a project directory and project applications**

- Create a project directory (root app):
  - a. Type the following command in the gitpod terminal, entering the desired project name at the end of the command followed by a space and a full stop:  
`django-admin startproject PROJ_NAME .`

This will create a project directory in the main directory. This project directory will contain a settings.py file and a urls.py file along with other files.

In the main directory, a manage.py file will also be created.
- Create project apps:
  - b. Type the following command in the gitpod terminal, entering the desired project app name at the end of the command:  
`python3 manage.py startapp APP_NAME`

This will create an app in the main directory; in the same directory as the root directory.
- Add the app to the settings.py file of the root directory.
  - c. In the settings.py file, navigate through to 'INSTALLED\_APPS' and add the app name, enclosed between a pair of single quotes, at the end of the list of django installed apps. After adding the app name, leave a comma at the end.  
E.g. 'services',
  - d. Save the changes.

**D) Migrate changes to the database.**

- Type the following command in the gitpod terminal:

```
python3 manage.py migrate
```

- If we make changes to an app model, we should run a 'makemigrations' command followed by the 'migrate' command. The makemigrations command can be run by following the below command:

```
python3 manage.py makemigrations
```

Note: The following commands can be used to display migration file information before we submit anything to the database:

- a) The makemigrations command is typically used when we make changes to a model. To do a practice run of what would happen if we ran makemigrations in real life, type the following into the command line:

```
python3 manage.py makemigrations --dry-run.
```

- b) The showmigrations command shows what migrations need to be submitted to the database. Type the following into the command line:

```
python3 manage.py showmigrations
```

The very first migration will contain a list of migration files from the django built-in apps (e.g. authentication and admin) that need migrating to the database.

- c) To make migrations to the database, type the following into the command line:  
Python3 manage.py migrate
  - d) After running this command – any migrations that were displaying with an empty square bracket (e.g. [ ]), will now display with a cross (e.g. [X]), when the showmigrations command is run after the migrate command. This indicates that these migrations have been made to the database.
- Run project on the server to test it is working:
    - e) Type the following command in the gitpod terminal:  
python3 manage.py runserver
    - f) Open up the browser on port 8000, it should open up a screen that displays that the "install worked successfully"

**E) Connect the gitpod terminal to Heroku:**

- Open up Heroku login command from the gitpod terminal - type in the following into the gitpod terminal:

heroku login -i

- Login in with your heroku email and password
  - a. Create an app on heroku for the EU region – type the following into the gitpod terminal:

Heroku create **APP-NAME** --region=eu

**F) Connect a server-based relational database to the heroku app**

- In Heroku, open up the app
- Click on the 'Resources' tab
  - a. In the 'add-ons' box search for 'Heroku Postgres' and select it.
  - b. A pop-up box will appear to confirm the installation of heroku postgres in your app – select 'submit order form'. Heroku Postgres will now appear in the 'add-ons' list
- Click on the 'Settings' tab
  - c. Scroll to 'Config Vars' then click on 'Reveal Config Vars' to retrieve our database URL
  - d. Next to the config var called 'DATABASE\_URL' copy the postgres URL in order to add into our project in gitpod as per steps below.

**G) Create and set up an env.py file**

- In the gitpod terminal, create a new file called 'env.py' in the top level directory. This file will store our secret environment variables.
- In the env.py:
  - a. Import the os library at the top of the file – type the following into the file:  
Import os
  - b. Set up environment variables – type the following into the file:  
os.environ["DATABASE\_URL"] = "**Paste in Heroku DATABASE\_URL Link**"
  - c. Create and add a secret key to encrypt session cookies. This should also not be visible on Github – type the following into the file:  
os.environ["SECRET\_KEY"] = "**Make up your own randomSecretKey**"
- Save all the changes in the env.py file.

**H) Update config vars in heroku app with secret key**

- In the heroku app, click on the 'Settings' tab
  - a. Scroll to 'Config Vars' then click on 'Reveal Config Vars'
  - b. Type the following in the 'Key' as: 'SECRET\_KEY'
  - c. For the 'Value' - Copy and paste the secret key created in gitpod
  - d. Click 'Add' – once both entries have been made

**I) Update root app with secret key**

- In gitpod, open the 'settings.py' file from the root app
  - a. Navigate to where it says:  

```
from pathlib import Path
```
  - b. Directly below this import, add the following code:  

```
import os
import dj_database_url
if os.path.isfile("env.py"):
    import env
```
  - c. Update the 'SECRET\_KEY' value in the settings file, with:

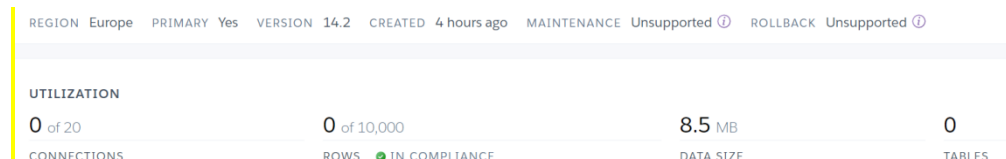
```
os.environ.get('SECRET_KEY')
```

**J) Connect the root app to the Postgres database**

- In gitpod, open up the 'settings.py' file from the root app
  - a. Navigate to the 'DATABASES' section
  - b. Highlight all of the section and comment it out (using ctrl + /)
  - c. Add a new 'DATABASES' section – as below:

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

- To test the root app has been linked to the database:
  - d. Open up the heroku app > Resources > Heroku Postgres – should say that there are zero tables/rows

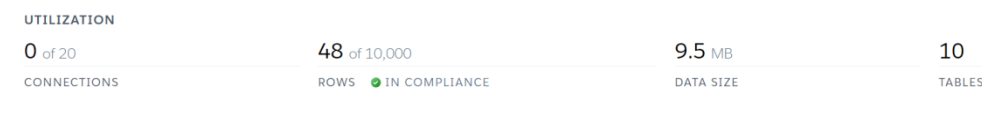


REGION	Europe	PRIMARY	Yes	VERSION	14.2	CREATED	4 hours ago	MAINTENANCE	Unsupported ⓘ	ROLLBACK	Unsupported ⓘ
UTILIZATION											
0 of 20		0 of 10,000		8.5 MB		0					
CONNECTIONS		ROWS		DATA SIZE		TABLES					

- e. In the gitpod terminal – type:

```
python3 manage.py migrate
```

- f. All the initial django built apps will migrate again – but this time to the heroku postgres database rather than the sqlite3 database on gitpod.
- g. Refresh the 'Heroku Postgres' page in Heroku – it should now show that there are more tables and rows. This shows our github project is now connected to the heroku database.



UTILIZATION											
0 of 20		48 of 10,000		9.5 MB		10					
CONNECTIONS		ROWS		DATA SIZE		TABLES					

**K) Push to changes to Github (optional)**

- Push the changes made so far to Github by typing the following into the command line:  
git add .  
git commit -m "initial commit"  
git push

**L) Set up a Cloudinary account and link it to the project**

- Create a Cloudinary account by visiting the Cloudinary website
- From your Cloudinary dashboard – scroll to the 'API Environment Variable' and copy the link. This link will be used to connect our app to Cloudinary.
- In gitpod, in the top level directory, open up the env.py file:
  - a. Add the following code and paste the Cloudinary API Environment Variable link as the value (removing the 'CLOUDINARY=' part from the front of the code:

```
os.environ["CLOUDINARY_URL"] = "cloudinary://*****"
```

- Copy the edited cloundinary API link
- Open up the Heroku app > Settings > Config Vars > Reveal Config Vars
- Add in a new Config Var key of CLOUDINARY\_URL and paste in the API link as the value. Then, select 'Add'
- Add a temporary environment variable – as below:  
Key: DISABLE\_COLLECTSTATIC

Value: 1

**Note: This will allow our skeleton project to deploy as initially we have no static files.  
When we deploy our full project – this should be removed.**

**M) Add Cloudinary libraries to the project**

- In gitpod, open up settings.py file from the root app
  - a. In the 'INSTALLED\_APPS' section install 'cloudinary\_storage' and 'cloudinary', such that they are placed in the following order:

```
INSTALLED_APPS = [  
    ...,  
    'cloudinary_storage',  
    'django.contrib.staticfiles',  
    'cloudinary',  
    ...,  
]
```

**N) Instruct Django to use Cloudinary to store media and static files for project**

- In gitpod, open up settings.py file from the root app, navigate to 'STATIC\_URL= '/static/'
  - a. Below this code, add the following code:

```
STATICFILES_STORAGE =  
'cloudinary_storage.storage.StaticHashedCloudinaryStorage'  
STATICFILES_DIRS = [os.path.join(BASE_DIR, 'static')]  
STATIC_ROOT = os.path.join(BASE_DIR, 'staticfiles')  
  
MEDIA_URL = '/media/'  
DEFAULT_FILE_STORAGE = 'cloudinary_storage.storage.MediaCloudinaryStorage'
```

**O) Instruct Django on where templates will be stored**

- In gitpod, open up settings.py file from root app, navigate to 'BASE\_DIR'
- Add in a template directory, using the below code:

```
TEMPLATES_DIR = os.path.join(BASE_DIR, 'templates')
```

- In main app> settings.py, navigate to 'TEMPLATES':
  - a. Change the 'DIRS' Key value to 'TEMPLATES\_DIR'

**P) Set up allowed hosts for the project**

- In gitpod, open up settings.py file from the root app
  - a. In the 'ALLOWED\_HOSTS' section, type the following:

```
ALLOWED_HOSTS = ['PROJ_NAME.herokuapp.com', 'localhost'].
```

This will add the heroku app to the project, to allow the project to be run from there and also add the localhost to allow the project to be run locally.

**Q) Create media, static and templates directories**

- In gitpod, in the top level directory, add the following new directories:

```
media  
static  
templates
```

### **R) Create and set up a Procfile**

This tells Heroku how to run the project.

- In gitpod, in the top level directory, add the following new file:

Procfile (Capital 'P')

- In the Procfile – add the following code:

web: gunicorn PROJ\_NAME.wsgi

This tells Heroku that this process should accept http traffic and that it should allow Python servers to interact with web servers

### **S) Push code to Github**

- In gitpod terminal, push the changes made so far to Github by typing the following into the command line:

```
git add .  
git commit -m "deployment commit"  
git push
```

### **T) Deploying to Heroku from gitpod**

- Deployment to Heroku can be done by staging and committing any changes to git, using the following commands:

```
git add .  
git commit -m "An appropriate commit message"  
git push
```

- In the heroku app, click on the 'Deploy' tab
  - a. Scroll to the 'Deployment method' and select 'Heroku CLI'

- In the gitpod terminal type:

```
heroku login -i
```

- Push the changes to Heroku, using the below command:

```
git push heroku main
```

Note: Before deploying to Heroku – always commit to github then login to heroku using heroku login -i then 'git push heroku main'. Once you have logged in once, then you can just use git push heroku main – don't have to keep logging in.

**U) Final deployment steps**

- In gitpod workspace open up the settings.py file from the root app:
  - a. set DEBUG = False
  - b. under DEBUG = False, add:  
  
`X_FRAME_OPTIONS = 'SAMEORIGIN'`
- Git add, git commit and git push to Github
- In Heroku app > Settings > Config Vars
  - c. Navigate to the “DISABLE\_COLLECTSTATIC” environmental variable and delete it.
  - d. Add the following Config Var:  
  
Key: PORT  
  
Value: 8000
- Push code to heroku:
  - e. In gitpod terminal, login to heroku, by typing:  
  
`heroku login -l` – enter email and password
  - f. Push code to Heroku by typing:  
  
`git push heroku main`
  - g. Open up app in Heroku to make sure everything is loading as expected.