Deployment

The master branch of this repository has been used for the deployed version of this application.

Using Github & Gitpod

Used the Code Institute Python Essentials Template.

- Click the Use This Template button.
- Add a repository name and brief description.
- Click the Create Repository from Template to create your repository.
- To create a Gitpod workspace you then need to click Gitpod, this can take a few minutes.
- When you want to work on the project it is best to open the workspace from Gitpod (rather than Github) as this will open your previous workspace rather than create a new one. You should pin the workspace so that it isn't deleted.
- Committing your work should be done often and should have clear/explanatory messages, use the following commands to make your commits:
    - git add .: adds all modified files to a staging area
    - git commit -m "A message explaining your commit": commits all changes to a local repository.
    - git push: pushes all your committed changes to your Github repository.

Forking the GitHub Repository

If you want to make changes to your repository without affecting it, you can make a copy of it by 'Forking' it. This ensures your original repository remains unchanged.

1. Find the relevant GitHub repository
2. In the top right corner of the page, click the Fork button (under your account)
3. Your repository has now been 'Forked' and you have a copy to work on

Cloning the GitHub Repository

Cloning your repository will allow you to download a local version of the repository to be worked on. Cloning can also be a great way to backup your work.

1. Find the relevant GitHub repository
2. Press the arrow on the Code button

3. Copy the link that is shown in the drop-down
4. Now open Gitpod & select the directory location where you would like the clone created
5. In the terminal type 'git clone' & then paste the link you copied in GitHub
6. Press enter and your local clone will be created.

Gmail SMTP

Used Gmail SMTP to send confirmation emails and all AllAuth related emails when the deployed version is used. settings.py file:

```
if 'DEVELOPMENT' in os.environ:
    EMAIL_BACKEND = 'django.core.mail.backends.console.EmailBackend'
    DEFAULT_FROM_EMAIL = 'hdyd@example.com'
else:
    EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
    EMAIL_USE_TLS = True
    EMAIL_PORT = 587
    EMAIL_HOST = 'smtp.gmail.com'
    EMAIL_HOST_USER = os.environ.get('EMAIL_HOST_USER')
    EMAIL_HOST_PASSWORD = os.environ.get('EMAIL_HOST_PASS')
    DEFAULT_FROM_EMAIL = os.environ.get('EMAIL_HOST_USER')
```

AWS static and media storage

All static and media files are stored in the cloud using Amazon AWS S3; Would need to create a bucket, user group and user that can access this site and the relevant files. In order for the files to be correctly served the following settings have to be added to settings.py file:

```
STATIC_URL = '/static/'
STATICFILES_DIRS = (os.path.join(BASE_DIR, 'static'),)

MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')

STATIC_ROOT = os.path.join(BASE_DIR, 'staticfiles')
```

```
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 = 'hdyd'
    AWS_S3_REGION_NAME = 'eu-west-1'
    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}/'
```

Creating an Application with Heroku

Followed the below steps using the Code Institute tutorial and Django Blog cheatsheat

- The following command in the Gitpod CLI will create the relevant files needed for Heroku to install your project dependencies pip3 freeze --local > requirements.txt. Please note this file should be added to a .gitignore file to prevent the file from being committed. A Procfile is also required that specifies the commands that are executed by the app on startup.

1. Go to Heroku.com and log in; if you do not already have an account then you will need to create one.
2. Click the New dropdown and select Create New App.
3. Enter a name for your new project, all Heroku apps need to have a unique name, you will be prompted if you need to change it.
4. Select the region you are working in.

Heroku Settings You will need to set your Environment Variables - this is a key step to ensuring your application is deployed properly.

- In the Settings tab, click on Reveal Config Vars and set the following variables:

AWS_ACCESS_KEY_ID	AWS access Key
AWS_SECRET_ACCESS_KEY	AWS secret Key
DATABASE_URL	postgres://
EMAIL_HOST_PASS	16 character password given from Gmail
EMAIL_HOST_USER	Gmail account
SECRET_KEY	Application secret key
SITE_ID	1
USE_AWS	True

Debug

ensure you don't have DEVELOPMENT as a key in your Heroku config variables DEBUG = 'DEVELOPMENT' in os.environ. This means DEBUG is set to whatever 'DEVELOPMENT' in os.environ evaluates to. If 'DEVELOPMENT' exists in the environment, DEBUG = True, if 'DEVELOPMENT' does NOT exist, then DEBUG = False. In another word, If 'DEVELOPMENT' is in your Heroku Config Vars, then DEBUG = True. If it's NOT there, then DEBUG = False. To collect static files, enable DEBUG in your local app, just add this to env.py: os.environ['DEVELOPMENT'] = 'Yes!!'

Heroku Deployment using website In the Deploy tab:

1. Connect your Heroku account to your Github Repository following these steps:
    i. Click on the Deploy tab and choose Github-Connect to Github.
    ii. Enter the GitHub repository name and click on Search.
    iii. Choose the correct repository for your application and click on Connect.
2. You can then choose to deploy the project manually or automatically, automatic deployment will generate a new application every time you push a change to Github, whereas manual deployment requires you to push the Deploy Branch button whenever you want a change made.
3. Once you have chosen your deployment method and have clicked Deploy Branch your application will be built and you should see the below View button, click this to open your application:

Heroku CLI deployment Whilst building this project there was a security breach on Heroku which caused issues with deployment via their website. Due to this, I had to add a runtime.txt file specifying

which version of Python to build the app with and used the following commands to push the code to Heroku:

1. Login to Heroku via the CLI using heroku login -i
2. Enter your email and password
3. Connect to the Heroku git remote using the heroku git:remote -a YOURAPPNAME
4. Push to the Heroku git remote using git push heroku main