

GymFit Deployment Document

To develop this project, a repository was setup in Github, using the Code Institute Gitpod Template (<https://github.com/Code-Institute-Org/gitpod-full-template>). This was then used to start a Gitpod workspace which was primarily used as the IDE.

The live projects is hosted on Heroku (<https://dashboard.heroku.com>) with images and static files hosted on Amazon Web Services (AWS) using their Simple Storage Service (S3). Stripe payment system (<https://stripe.com/gb>) has been integrated to process payments and Gmail is being used to send real world emails.

Local Deployment

Prerequisites: To deploy this project you require the following installed on you local computer or IDE:
Python 3 or higher - Programming Language
Pip3 - Python Package Installer
Git - Version control

Cloning the Repository using Git

1. Open the repository, click the code button, which is located on the right above all the repository file names.
2. Select HTTPS and copy the clone URL.
3. In your command line type "git clone" and then paste the URL that you just copied.
4. Press enter to create your local clone

Cloning the Repository using GitPod

To clone the repository, you first need to:

1. Install the GitPod Browser Extension for Chrome (including restarting the browser).
2. Log into your GitHub or create an account.
3. Find the GitHub Repository that you want to clone.
4. Click the green "GitPod" button in the top right corner of the repository. This will trigger a new GitPod workspace to be created.

The cloned repository includes the 'requirements.txt' file. To install all of the packages required for this project to run, use the following command in the terminal:

```
pip3 install -r requirements.txt
```

Once that installation is successful, setup a superuser entering the command below into the terminal and following the prompts:

```
python 3 manage.py createsuperuser
```

Development Environment Variables

The following environment variables must be set within your development environment for the application to function.

| Variable | Value |
|-------------|-------|
| DEVELOPMENT | True |

| | |
|-------------------|---|
| SECRET_KEY | A randomly generated django secret key |
| STRIPE_PUBLIC_KEY | Value attained from Stripe Account |
| STRIPE_SECRET_KEY | Value attained from Stripe Account |
| STRIPE_WH_SECRET | Value attained from stripe webhook endpoint |

For more information on how to find STRIPE_PUBLIC_KEY, STRIPE_SECRET_KEY and STRIPE_WH_SECRET, see the Stripe Setup section below.

To generate the SECRET_KEY I used miniwebtool's Django Secret Key Generator (<https://miniwebtool.com/django-secret-key-generator/>).

Development Server Database Setup

To set up your projects local database you will need to run migrations by entering the following commands into the CLI and following any prompts, if there are any:

```
python3 manage.py migrate --plan
```

```
python3 manage.py migrate
```

You should now commit all these changes to the repository.

To run the application locally just enter the command below into the CLI:

```
python3 manage.py runserver
```

Heroku Deployment

To deploy the project to Heroku you need to:

1. Go to Heroku and log into your account. If you do not have one, create one.
2. Click create an app, enter a valid app name, choose the region closest to your location and click "Create App".
3. When the app is created, add a Postgres database to it by selecting it from the addons list
4. You will then need to set the following variables in the Heroku "Config Vars". If you do not have the variable yet, continue on the deployment until you create them.

| Variable | Value |
|-------------------|--|
| SECRET_KEY | A randomly generated django secret key |
| STRIPE_PUBLIC_KEY | Value attained from Stripe Account (see Stripe Setup) |
| STRIPE_SECRET_KEY | Value attained from Stripe Account (see Stripe Setup) |
| STRIPE_WH_SECRET | Value attained from stripe webhook endpoint (see Stripe Setup) |
| DATABASE_URL | automatically setup during Heroku deployment and can be found by viewing your Postgres database within the Heroku dashboard, under Settings Database Credentials |
| USE_AWS | True |
| AWS_ACCESS_KEY_ID | Generated key from AWS (see AWS Setup section) |
| EMAIL_HOST_PASS | Password provided by the email provider (see Email Setup section) |

To have your project automatically deploy to Heroku when you push the repository to Github you need to set up automatic deployment. To do this you need to:

1. Click the deploy tab on your Heroku App.
2. Select GitHub as the deployment method
3. Search for and select the GitHub repo you want to connect to
4. Click “Connect” and then “Enable Automatic Deploys”

Deployed Server Database Setup

If you already have a database set up, or you want to use the fixtures file included in the repository, then you can use the command below in the CLI to copy it to the deployed database.

```
python3 manage.py loaddata db.json
```

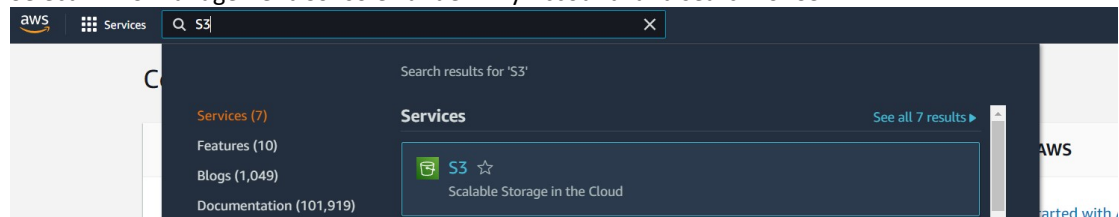
If you do not copy any data in, then you will have to set up a superuser account as in the Local Deployment section.

Finally to complete deploying to your database, you will need to run migrations, as in the Local Deployment section.

Amazon Web Services (AWS) Setup

S3 Bucket Setup

1. Navigate to Amazon Web Services and Create or Sign into your account.
2. Select “AWS Management Console” under “My Account” and search for S3



3. Create a new bucket and select the region closest to you.

4. Select ACLs enabled and Bucket Owner Preferred

Object Ownership [Info](#)

Control ownership of objects written to this bucket from other AWS accounts and the use of access control lists (ACLs). Object ownership determines who can specify access to objects.

☐ **ACLs disabled (recommended)**

All objects in this bucket are owned by this account. Access to this bucket and its objects is specified using only policies.

☒ **ACLs enabled**

Objects in this bucket can be owned by other AWS accounts. Access to this bucket and its objects can be specified using ACLs.

Object Ownership

☒ **Bucket owner preferred**

If new objects written to this bucket specify the bucket-owner-full-control canned ACL, they are owned by the bucket owner. Otherwise, they are owned by the object writer.

☐ **Object writer**

The object writer remains the object owner.

i If you want to enforce object ownership for new objects only, your bucket policy must specify that the bucket-owner-full-control canned ACL is required for object uploads. [Learn more](#)

5. Un-check block all public access and confirm that the bucket will be public

Block Public Access settings for this bucket

Public access is granted to buckets and objects through access control lists (ACLs), bucket policies, access point policies, or all. In order to ensure that public access to this bucket and its objects is blocked, turn on Block all public access. These settings apply only to this bucket and its access points. AWS recommends that you turn on Block all public access, but before applying any of these settings, ensure that your applications will work correctly without public access. If you require some level of public access to this bucket or objects within, you can customize the individual settings below to suit your specific storage use cases. [Learn more](#)

☐ **Block all public access**

Turning this setting on is the same as turning on all four settings below. Each of the following settings are independent of one another.

☐ **Block public access to buckets and objects granted through *new* access control lists (ACLs)**

S3 will block public access permissions applied to newly added buckets or objects, and prevent the creation of new public access ACLs for existing buckets and objects. This setting doesn't change any existing permissions that allow public access to S3 resources using ACLs.

☐ **Block public access to buckets and objects granted through *any* access control lists (ACLs)**


S3 will ignore all ACLs that grant public access to buckets and objects.

☐ **Block public access to buckets and objects granted through *new* public bucket or access point policies**

S3 will block new bucket and access point policies that grant public access to buckets and objects. This setting doesn't change any existing policies that allow public access to S3 resources.

☐ **Block public and cross-account access to buckets and objects through *any* public bucket or access point policies**

S3 will ignore public and cross-account access for buckets or access points with policies that grant public access to buckets and objects.

**Turning off block all public access might result in this bucket and the objects within becoming public**

AWS recommends that you turn on block all public access, unless public access is required for specific and verified use cases such as static website hosting.

☒ I acknowledge that the current settings might result in this bucket and the objects within becoming public.

6. Select "Create Bucket" at the bottom of the page.

7. Under the bucket properties tab, select to 'edit' Static website hosting and select 'enable'.

| | | | |
|---------|-------------------|-------------|---------|
| Objects | Properties | Permissions | Metrics |
|---------|-------------------|-------------|---------|

Static website hosting

Use this bucket to host a website or redirect requests. [Learn more](#)

Edit

Static website hosting

Enabled

Hosting type

Bucket hosting

- Ensure that host a static website is enabled and enter default values for the index and error documents and click save.
- On the Permissions tab, at the top of the page, paste in this CORS Configuration.

```
[
  {
    "AllowedHeaders": [
      "Authorization"
    ],
    "AllowedMethods": [
      "GET"
    ],
    "AllowedOrigins": [
      "*"
    ],
    "ExposeHeaders": [ ]
  }
]
```

- Next scroll to the Bucket Policy section and click policy generator.
- In the policy generator, choose “S3 Bucket Policy” for the type of policy, enter a star (*) into the “Principal” input to select all and choose “Get Object” from the Actions Checkbox Drop down.

Step 1: Select Policy Type

A Policy is a container for permissions. The different types of policies you can create are an IAM Policy, an S3 Bucket Policy, an SNS Topic Policy, a VPC Endpoint Policy, and an SQS Queue Policy.

Select Type of Policy S3 Bucket Policy

Step 2: Add Statement(s)

A statement is the formal description of a single permission. See a description of elements that you can use in statements.

Effect ☒ Allow ☐ Deny

Principal

Use a comma to separate multiple values.

AWS Service Amazon S3 ☐ All Services (*)

Use multiple statements to add permissions for more than one service.

Actions -- Select Actions -- ☐ All Actions (*)

Amazon Resource Name (ARN)

☐ GetLifecycleConfiguration
☐ GetMetricsConfiguration
☐ GetMultiRegionAccessPoint
☐ GetMultiRegionAccessPointPolicy
☐ GetMultiRegionAccessPointPolicyStatus
☒ GetObject
☐ GetObjectAcl

(BucketName)/\${KeyName}.
must select at least one Action

- Copy the Amazon Resource Number (ARN) from the previous tab and paste it into the Amazon Resource Number (ARN) box.
- Click “Add Statement” and then “Generate Policy”.
- Copy this policy and paste it into the Bucket Policy Editor and add a slash star (/*) onto the end of the resource key and click save.
- Finally, scroll down to the Access Control List (ACL) and click edit. Here you will need to check Everyone(Public Access) and confirm that you understand the effects of these changes.

Access control list (ACL)

Grant basic read/write permissions to other AWS accounts. [Learn more](#)

Edit

AWS doesn't recommend granting access to the Everyone grantee
Anyone in the world can access the objects in this bucket.

[Learn more](#)

Access control list (ACL)

Grant basic read/write permissions to other AWS accounts. [Learn more](#)

| Grantee | Objects | Bucket ACL |
|--|---|---|
| Bucket owner (your AWS account) Canonical ID: 9be4fc20e4d501401e7dbe56117f84072695f7e5d89ea7ecae71f93fb4895c39 | <input checked="" type="checkbox"/> List <input checked="" type="checkbox"/> Write | <input checked="" type="checkbox"/> Read <input checked="" type="checkbox"/> Write |
| Everyone (public access) Group: http://acs.amazonaws.com/groups/global/AllUsers | <input checked="" type="checkbox"/> List <input type="checkbox"/> Write | <input type="checkbox"/> Read <input type="checkbox"/> Write |
| Authenticated users group (anyone with an AWS account) Group: http://acs.amazonaws.com/groups/global/AuthenticatedUsers | <input type="checkbox"/> List <input type="checkbox"/> Write | <input type="checkbox"/> Read <input type="checkbox"/> Write |
| S3 log delivery group Group: http://acs.amazonaws.com/groups/s3/LogDelivery | <input type="checkbox"/> List <input type="checkbox"/> Write | <input type="checkbox"/> Read <input type="checkbox"/> Write |

When you grant access to the Everyone or Authenticated users group grantees, anyone in the world can access the objects in this bucket.

[Learn more](#)

☒ I understand the effects of these changes on my objects and buckets.

Creating AWS Groups, Policies and Users

- In the AWS services menu open IAM (Identity and Access Management) and click 'User groups' to create a new group.
- Give the group a logical name and click "next step" until you reach "create group" which you should click.
- On the next page click "Policies" then "Create Policy".
- On the Create Policy page, click the "Import Policy" button, search for "S3" in the search bar and import the "AmazonS3FullAccess" policy.
- Modify the policy by adding your ARN as the value for "Resource" as a list in the following formats:
 - "arn:aws:<ARN>",

- b) `"arn:aws:<ARN>/*"`,
6. Click the 'Next: Tags' button and then click 'Next: Review' button.
7. Provide a name and description for the policy and click 'Create policy'.

Attach policy to group.

1. Navigate back to IAM and click 'User groups' to view the newly created group.
2. Select the new group and click the permissions tab
3. Click the 'Add permissions' button and click 'Attach Policies' from the drop down menu.
4. Choose the policy that was just created and select 'Add permissions'

Create User and add to group.

5. Click Users in IAM and select Add User
6. Provide a logical user name, check the checkbox to give them programmatic access and click next.
7. Check the tick box next to the group just created to add the user to the group. Click through the next few pages to create user.
8. On the success page, click 'Download .csv file' which contains the user access key and secret access key which is needed to authenticate them from the Django app (See Heroku Deployment).

Connect Django to AWS s3 bucket.

1. In the settings.py file in the Project Level GymFit app, update the setting with the values you have just created the Bucket using:

```
# Bucket Config
AWS_STORAGE_BUCKET_NAME = "gym-fit-ms4"
AWS_S3_REGION_NAME = "eu-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"
```

2. In the Heroku Config Vars, add the AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY and add the USE_AWS key and set it to true. The access key and secret key are contained with the downloaded csv file.
3. Once that is done commit and push the changes. Check the Heroku build log to check that the static files were collected and there should be a 'static' folder within the s3 bucket.

Add media files to s3 bucket

1. Within the s3 bucket overview click create folder and call it media.
2. Inside that folder click on 'upload' and then 'add files'
3. Select all of the images you need to upload, stored in the repo, and click open.
4. Click next and then under permissions, check the box for 'grant public-read access':
5. Then click upload.
6. There should now be a 'media' folder within the s3 bucket containing the images.

Stripe Payment Setup

1. Create a Stripe Account or Sign In to an existing one.
2. In the Developers page (link in top right corner) copy the Test API Key and Secret Key to store in the Development Environment and the Heroku Config Variables (See Heroku Deployment section)
3. Run the server or go to the deployed site and copy the URL..
4. In Stripe, Click 'Developers', select 'webhooks' and then click 'Add endpoint'.
5. Paste the copied URL in and add '/checkout/wh/' onto the end.
6. Click 'Select Events' and select to listen for all events.
7. In the newly created webhook copy the "Signing Secret"s and add it to the value for the environment and Heroku variable STRIPE_WH_SECRET (See Heroku Deployment section).

Email Setup

The following instructions are to be implemented using a Gmail account.

1. Log in to your email account or set one up.
2. Click account settings, then 'Accounts and Import'.
3. Under 'Change account settings' click 'Other Google Account settings'
4. Click on the 'Security' option then click on '2-Step Verification' and follow through all the prompts.
5. When this is completed, a new option should show under the previous 'Signing in to Google' menu screen.
6. Click this option and select 'mail' for app and 'other' for device from the dropdown menus. Add an appropriate name and click 'generate'. You will then be given a 16 character password which you will need to enter into the Heroku Config Vars under the key EMAIL_HOST_PASS (see Heroku Deployment section)