

Milestone project 4 additional write-up

Table of Contents

[User Stories 2](#_Toc3152814)

[Stories 2](#_Toc3152815)

[As a Lego enthusiast, I… 2](#_Toc3152816)

[As owner of the store, I… 3](#_Toc3152817)

[Derived stories yet to be implemented 4](#_Toc3152818)

[Mockups and UX considerations 4](#_Toc3152819)

[Mockups 4](#_Toc3152820)

[UX considerations 4](#_Toc3152821)

[Testing 10](#_Toc3152822)

[Manual Testing 10](#_Toc3152823)

[Automated Testing 10](#_Toc3152824)

[Travis CI 10](#_Toc3152825)

[Deployment 11](#_Toc3152826)

[Quick deployment on local environment 11](#_Toc3152827)

[Environment variables 12](#_Toc3152828)

[Group creation in Django Admin 13](#_Toc3152829)

[Running on hosted environments 14](#_Toc3152830)

[Heroku 14](#_Toc3152831)

[AWS 16](#_Toc3152832)

# User Stories

The user stories here after are accompanied by the resulting developer tasks.

The overall Epic for this project is:

*As a user, I want a place where I can get stickers to complete the Lego sets I have built, but am missing the stickers for.*

This overall ‘desire’ was then broken down to the following stories and their accompanying developer tasks related to them.

## Stories

### As a Lego enthusiast, I…

1. Want to have a landing page, where I can see the latest stickers
   1. Build a landing page with some general information
   2. Create a back end able to hold a catalog of products
   3. Show the most recent additions to the catalog on the landing page
2. Want to be able to search for specific sticker sets
   1. Create a search field
      1. Allow free text searches
      2. Have searches cover categories and descriptions for maximum response
      3. Return the results, along with the search string they’re based on
   2. Show the search field on the products page
3. Want to be able to view a single Lego category for sticker sets
   1. Create a pulldown menu with categories
      1. Populate the list from the unique categories in the database
      2. When clicked, return the results specific to that category and show the category selected
4. Want to know how the stickers are made
   1. Create an FAQ section covering various questions
   2. Base the FAQ page on database entries, allowing easy adding of items, and no need to update page HTML
5. Want some way to be kept updated on new items or any other news
   1. Allow users to register an account, so their email address can be used to send out newsletters to them
   2. Upon registration, directly direct users to the main page
   3. Create an option for users to view their information
   4. Create a way for users to sign out of their account
   5. Create a way for users to delete their own account
6. Want to know how my personal data is treated
   1. Create a page with a privacy statement
7. Want to have a way to stop receiving news from the site
   1. This is covered in 5c.
8. Want to know how I can go about issuing questions
   1. Create a page where customers can contact the site’s owner through a contact form
   2. Provide a directly visible link to the customer to the contact page
   3. When submitted, email the form contents to the site owner
9. Want a notification when I place an order with information on my order
   1. When a user places an order, send them a confirmation mail containing their order information

### As owner of the store, I…

1. Want to have a way to add products without having to go into the admin pages
   1. Create an admin section where staff members can add items to the database
   2. Shield the page from non-staff users that manipulate the URL to the page
   3. Redirect unwanted individuals to the main page
   4. Throw an alert box when successfully adding an item or on error
2. Want to be able to quickly expand the FAQ area when certain questions are asked often
   1. Create the same overall solution for the addition of FAQ items as was made for new products to maintain look and feel
3. Don’t want regular customers to be able to add products, or FAQ items
   1. Create views to check for staff membership and render view based on true or false
4. Want to be able to process order from within the site
   1. Create a view showing all orders that are not yet processed
   2. Allow checking an order as processed to remove it from the view
   3. Make sure orders are not deleted, just marked as processed
5. Want customers to be able to leave comments on products
   1. Create a product detail page where customers can leave comments on the product
   2. Show the comments from newest to oldest
6. Want customers to get the content from their sent contact form as well
   1. Expand the view for the contact form to also send an email to the customer on the address they entered
7. Want customers that registered to have the option to change their password
   1. Provide the user with links on the login page to a password reset page
   2. Have users receive an email from which they can reset their password
   3. Upon successful change, give users option to log in, or go straight to the store
8. Don’t want staff members to be able to remove their accounts
   1. Create a check in the profile page to disable the delete button for staff members
9. Want to have an easy way of getting all email addresses of registered users for the newsletter
   1. Create a download button for a csv file with mail addresses

## Derived stories yet to be implemented

The stories below came about as the site was being built. Due to time constraints, these have not yet made the cut and are effectively on the backlog at the time of writing.

1. As a user I want to be able to see my order history
2. As a user I want to be able to save my cart for later
3. As a user I want to be able to talk with others about Lego
4. As a user I want my address information to be saved for later use
5. As site owner I want to modify products outside of Django Admin
6. As site owner I want to download a full order history for administrative purposes
7. As site owner I want to have more payment options available
8. As site owner I want to be able to process the newsletter within the site

## Mockups and UX considerations

### Mockups

As with every project so far, mocking up was done the old-school way using pen and paper. Mostly because of the extremely gratifying feeling of crumpling up any mistakes and drawing again. Maybe that’s just me though ☺

<IMAGES>

### UX considerations

Initially the idea was to create the site using the Bootstrap 4 framework. This progressed up to the point where I personally felt a distinct dislike for the way Bootstrap presents on screen.

At that point I decided to redo the entire site using MaterializeCSS. The original Bootstrap versions of the site pages, as well as a write-up of how to go back to using those, can be found in the root of the repository.

The initial idea, based on the user stories thought up, was to create a site with a category list on the left hand side of the page, and the items per category to the right of that. However, while that would work well enough on desktops, my experience is that sites using that tend to scale less well.

With that in mind, I decided to have a simple navigation bar that scales well across devices, providing the most important links for the site.

These links are: **Home**, **Products**, **Contact**, **Sign In / Sign Out** and of course, the **Cart**. Limiting the numbers of items directly in the navbar this way allows for there not being a need to collapse the menu, making it identical across devices (albeit less wide on mobile devices).

The colours used throughout the site are based on the original logo for the actual shop currently out there, with additions in milder tones so not make the site scream at you.

To avoid too much white on the pages, an off-white is used extensively as well to be easier on the eyes.

The colour red used throughout is #c9190a, which is the same red as in the logo.

The colour yellow used is #faec06, which is the same yellow as in the logo.

The off-white is #f7f7f7

The lighter yellow is the lighter tone of the yellow used, #fcf69c

The light blue was selected as alternative for what would be a sickly pink, #d0e8ff.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| #c9190a  Red | #faec06  Yellow | #f7f7f7  Off-white | #fcf69c  Light Yellow | #d0e8ff  Light Blue |

These colours persist throughout the site, only adding plain white and black into the pages.

Links in the navbar are accompanied by an icon, the standard composition is yellow for the icon, white for the text. The same applies to the buttons on pages:

Red background, yellow icon and white text.

In order to create a sharp contrast, cards used throughout the site are given a 1px red border.

Pulldown menus for categories and user menu, change colour when hovering over items in the list, to the lighter yellow. This is less intrusive than the hard yellow, but still clear to see.

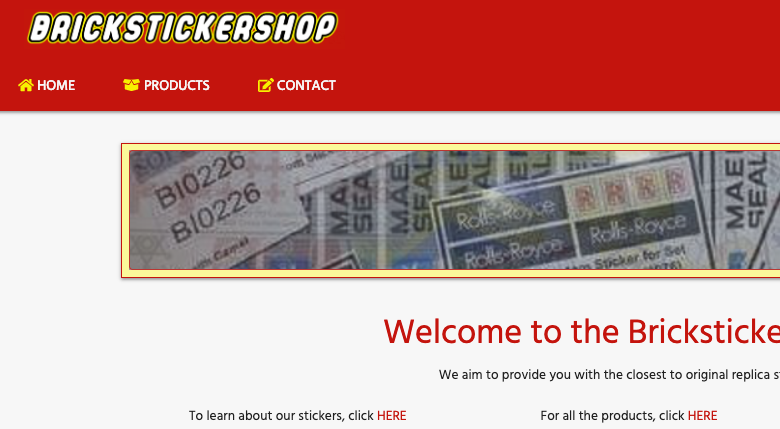
Overall, the idea was to have a site that is as straight-forward as possible, where adding items to the cart changes the button to show the item is in the cart.

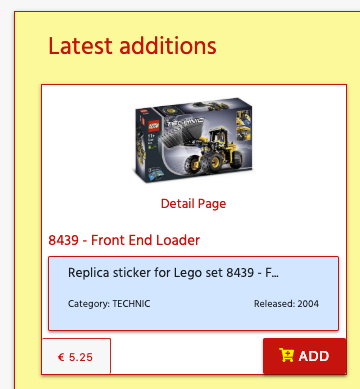
In the cart itself, items can be changed for quantity, or just deleted from the cart.

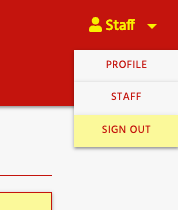
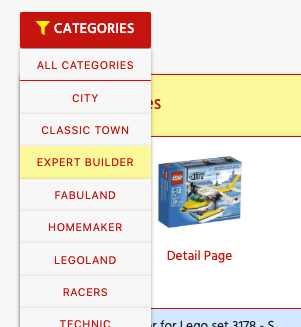
This extends to the checkout, where customers can still modify the contents of the cart.

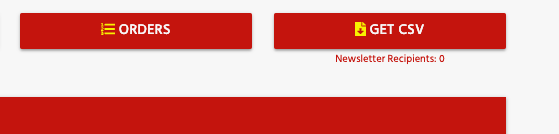
The footer, present on every page, allows access to general information at all times.

#### Sample site images



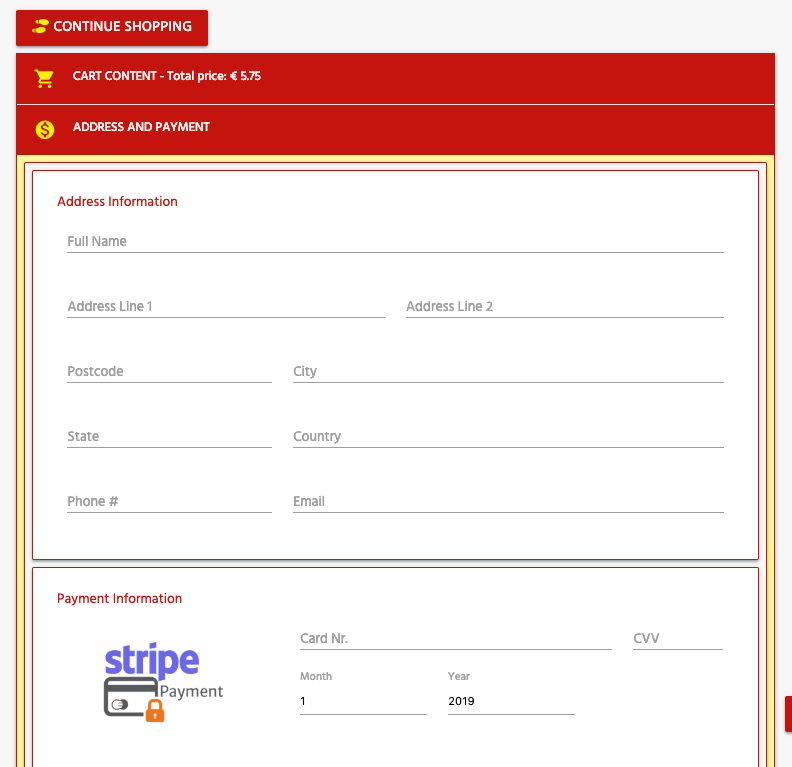


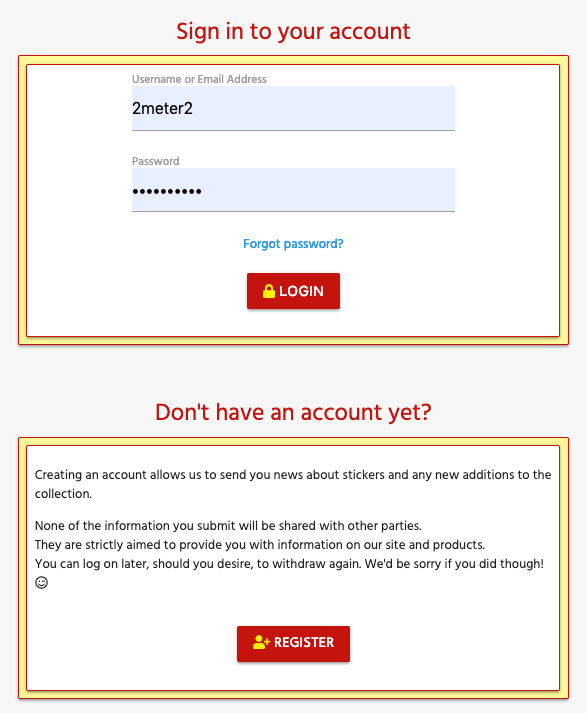
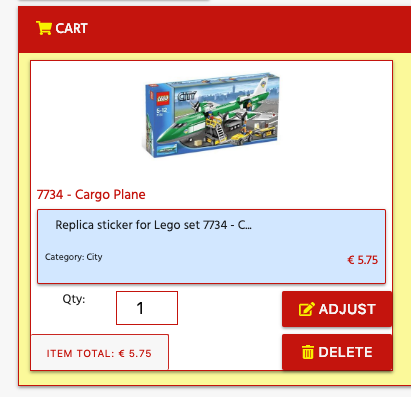


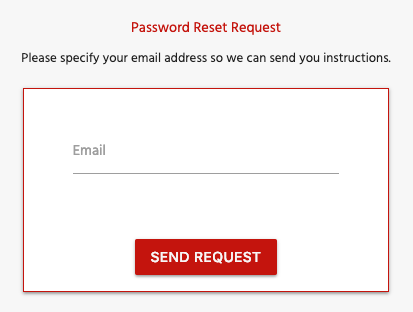
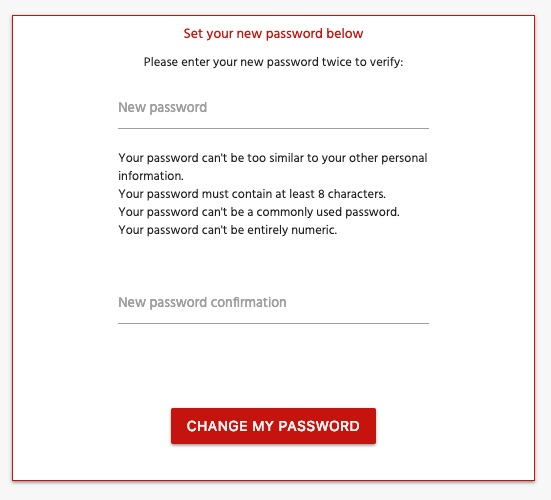


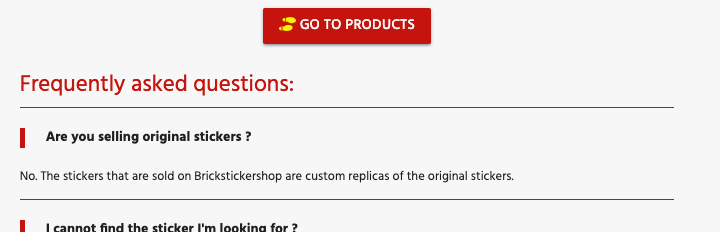


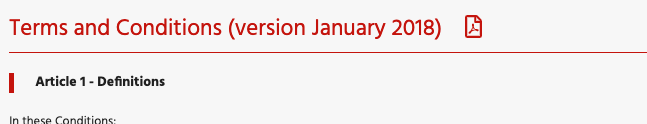










# Testing

## Manual Testing

## Automated Testing

## Travis CI

# Deployment

## Quick deployment on local environment

To do a fast installation of the site perform the following steps:

* Download and unpack, or clone the repository from Github
* Set the environment variables below in **~/.bashrc** using your favorite editor   
  (be it vi or nano, I don’t judge.) For development, the yellow lines are optional.  
  After, make sure to refresh using **. ~/.bashrc** from the prompt, or closing and opening the terminal again!

|  |
| --- |
| export SECRET\_KEY='I-am-a-development-server'  export EMAIL\_HOST\_USER='fake@example.com'  export EMAIL\_HOST\_PASSWORD='fake-password-this'  export STRIPE\_PK='pk\_test\_key'  export STRIPE\_SK='sk\_test\_key'  export DATABASE\_URL='fake-db-url'  export AWS\_ACCESS\_KEY\_ID='fake-AWS-key'  export AWS\_SECRET\_ACCESS\_KEY='fake-AWS-secret-key'  export ENVTYPE='development' |

After that, add the following alias into **~/.bash\_aliases**, again using your favorite editor again:

|  |
| --- |
| alias run="python3 ~/workspace/manage.py runserver $IP:$PORT" |

*Substitute ~/workspace/ with whatever working directory you are using.*  
Make sure to refresh aliases as well: **. ~/.bash\_aliases**

* Run the following commands in succession and follow prompts where needed:

|  |
| --- |
| sudo pip3 install -r requirements.txt  python3 manage.py makemigrations  python3 manage.py migrate  python3 manage.py createsuperuser |

* At this point you have a very pristine working version of the site
* Running the server is as as easy as entering **run** on the command prompt now. Logging in using the superuser you created allows you to add products to the database through the staff pages. The same goes for FAQ items.  
  Comments can then be added to the new products on the product detail pages as well.   
  There will not be any orders, nor can they be placed in this state. For that, Stripe and working email are required.

## Environment variables

The table below shows every environment variable used:

|  |  |
| --- | --- |
| SECRET\_KEY | Required by Django. For security reasons, this variable should never be present in files, so it was moved to live in an environment variable. Can be whatever your heart desires, but shouldn’t be too simple. |
| EMAIL\_HOST\_USER | Mail address used to send order confirmation, contactforms and password reset related mails. For the purpose of the course, I created a Gmail account and gave it the right access to be used by the application. |
| EMAIL\_HOST\_PASSWORD | Seems obvious enough. Don’t want that in a file |
| STRIPE\_PK | Stripe Publishable Key, needs to be obtained at Stripe.com. For the project, the only keys used are the test keys provided by Stripe, as this site is not actually live in the sense of selling items. |
| STRIPE\_SK | Stripe Secret Key, same as above, obtained from Stripe.com. Again, for the projet, only the test keys are in use. |
| DATABASE\_URL | While not needed for development, it is required for production and referenced in the settings.py file. Contains the link to the PostGres database at AWS for this project. |
| AWS\_ACCESS\_KEY\_ID | Not needed for development, required for production for communication with AWS |
| AWS\_SECRET\_ACCESS\_KEY | Same as above, only needed in production for communication with AWS |

The colours used indicate the variables that are sets when used.

If accounts at Stripe and AWS aren’t readily available, the payment can be tested on the Heroku hosted version of the site.

Get working keys to add to the environment variables:

* In order to get mail working, provide user and password to be used. Gmail is an easy option.
* In order to get Stripe working visit [Stripe.com](https://stripe.com/docs/development) and create a developer account.
* In order to get AWS up and running, visit [AWS](https://aws.amazon.com/) and create an account and/or bucket for the site to use.

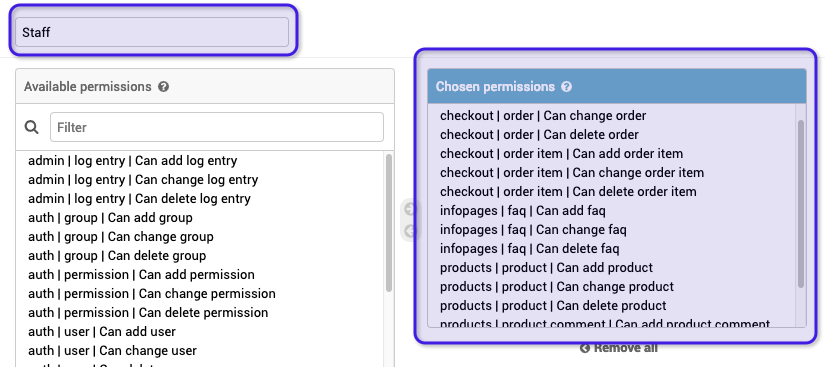
Alternatively, test things out on the live URL provided in the README.md

The only thing you will not receive there, is the site admin mail for the contact form.

## Group creation in Django Admin

To easily add users to the staff group, create a group in Django admin with the following rights. Add all entries for:

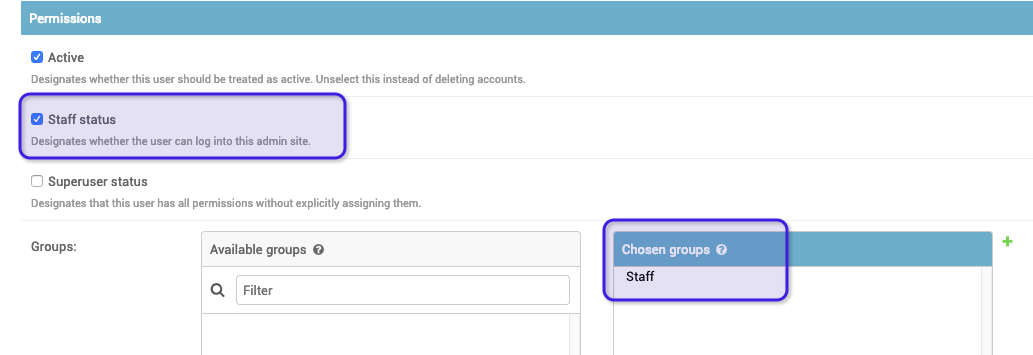
* Checkout (includes ‘orderitem’)
* Infopages
* Products (includes ‘product comment’)



Whenever a new user is made staff, they only need to be added to this group, to have full options available to them, outside of user management and other superuser tasks.

Deployment does not create this, and will need to be performed in both development and production manually. This also allows to create more granular groups in a production setting, however for the purpose of the project the choice fell to one single group.

An example of a non-superuser staff member’s permissions then would be:



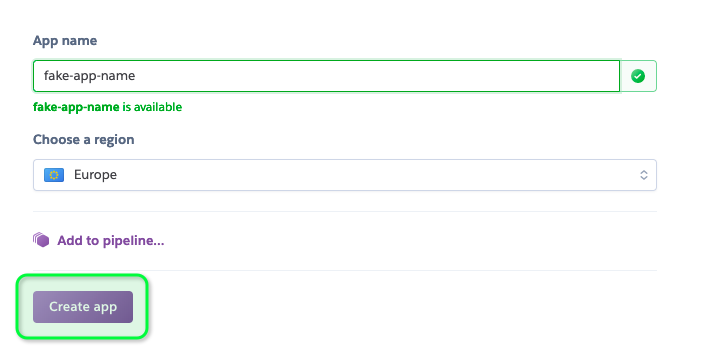
Currently there is no way to do this using the site interface itself, the superuser will need to do this through the Django Admin panel.

# Running on hosted environments

In order to run the site on a hosted environment some steps must be taken. For the project, Heroku was used, along with AWS for media and static file storage.

## Heroku

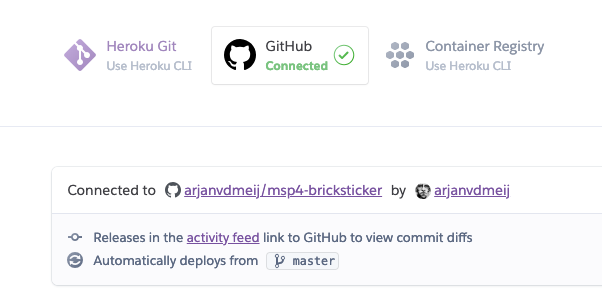
If not available, go to <https://www.heroku.com/> and create an account. Once verified, create a new app:

****

Choose any name you like, provided it is of course available.

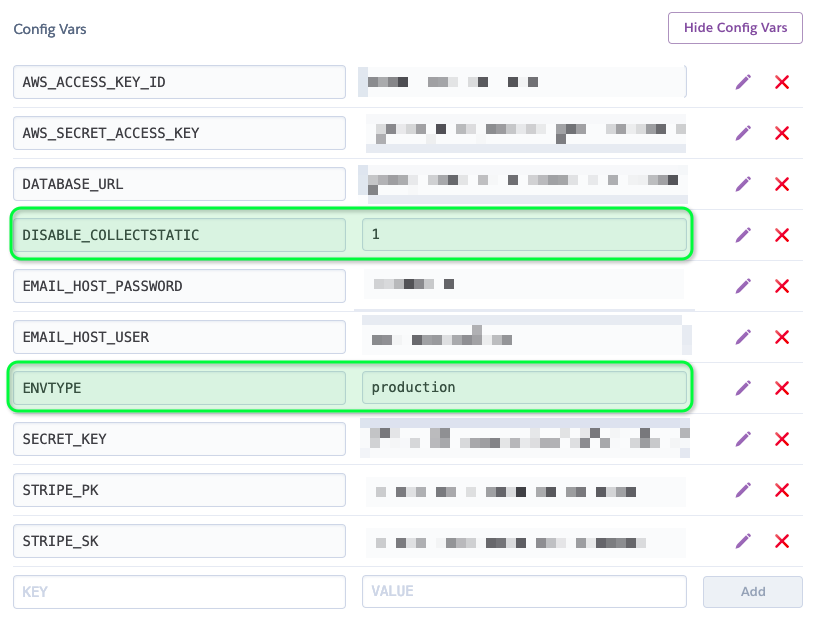
You can choose one of two ways to deploy an app, either directly with Heroku CLI, or using an existing Git repository.

For the project, I have linked Heroku to my Git repository:



I have also enabled Automatic deploys on detection of changes to github, in order to directly push changes live. This is entirely optional however.

Within the settings tab, the config vars entered are:



These variables are the same as they are in the development environment, with one addition and one notable change:

**ENVTYPE** here is set to production

**DISABLE\_COLLECTSTATIC** is set to **1** to prevent Heroku from adding or removing files on AWS or creating a local staticfiles directory.

The **DATABASE\_URL** is provided from within Heroku, by installing *PostGres* as an add-on to the app.

In the Resources tab, type postgres into the *Add-ons* quick search bar. Select **Heroku PostGres** and the plan as *Hobby Dev – Free*, then click **Provision**.

This will attach the database to your app.

This will automatically fill out the **DATABASE\_URL** environment variable for you, you don’t need to do anything there yourself. This is the only variable to be entered automatically however, the rest needs to be added manually.

The values for the AWS variables are those provided to you by AWS.

## AWS

In AWS, create a bucket that has public access, so uploading of images for products works without problems.

Within S3, create a bucket that holds public access and is accessible for static website hosting.

Follow the tutorials by Amazon to create the bucket, and its accompanying IAM groups, users and policies.

From a command prompt, run the command

Python3 manage.py collectstatic

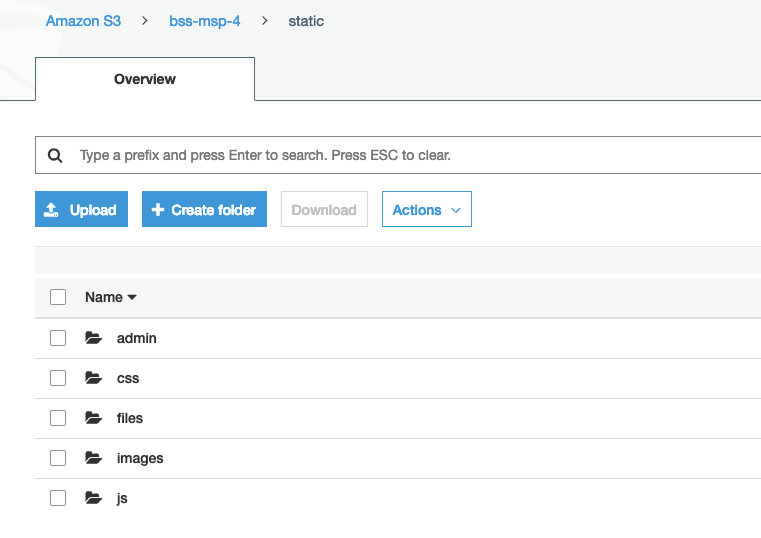
This will create a *staticfiles* folder within the root, containing everything that needs to go into AWS.

In the bucket, create two directories:

* *media* (and in it, a directory called *img*)
* *static*

In theory, the media directory would be created automatically, but given how little effort, it can be done immediately as well.

From the *staticfiles* directory locally, upload the entire contents into *static* on AWS resulting in a view like below:



\*\* continue on next page \*\*

Make sure your personal AWS\_ACCESS\_KEY\_ID and AWS\_SECRET\_ACCESS\_KEY are stored as variables for the environment.

Finally, change the following lines in *settings.py* to the values for your personal bucket:

|  |
| --- |
| AWS\_STORAGE\_BUCKET\_NAME = 'bss-msp-4'  AWS\_S3\_REGION\_NAME = 'eu-west-1' |

Once all the settings are done, deploying the application should now run it without trouble.