**Django Notes**

**Django Setup**

Text

Description automatically generated

(command 1) cd desktop

(command 2) mkdir django\_project\_folder

(command 3) cd django\_project\_folder

(command 4) python -m venv virtual\_environment\_name

(command 5) virtual\_environment\_name\scripts\activate

(command 6) pip install django

(command 7) django-admin --version

(command 8) django-admin startproject start\_project\_name

(command 9) cd start\_project\_name

(command 10) python manage.py runserver

the above commands:

- (command 1) goes to the desktop

- (command 2) create a django project folder

- (command 3) enters the django project folder

- (command 4) creates a virtual environment so that this project is isolated from the rest of your computer

- (command 5) activates the virtual environment

- (command 6) installs django

- (command 7) displays django version (env must be activated if django was installed with env activated)

- (command 8) creates a startproject folder

- (command 9) enters the startproject folder (now inside the main folder)

- (command 10) runs the website server at <http://127.0.0.1:8000/>

If the setup is successful, the website server at <http://127.0.0.1:8000/> should display the following:

Text

Description automatically generated

A text-editor such as Atom will be needed.

**Tutorial Link:** <https://www.youtube.com/playlist?list=PL-osiE80TeTtoQCKZ03TU5fNfx2UY6U4p>

**Applications and Routes**

* An application is a section/component of our website
* Within a website, we can have multiple apps. For example, we can have apps such as a blog section or a store section of the website.
* A single project can contain several apps. As well, an app could be used in many projects.

Creating an application:

* open the command prompt with inside the main folder (where manage.py is) with the virtual environment activated
* use the command: python manage.py startapp app\_name



* a new app\_name directory is created inside the main folder
* go apps.py file inside the app\_name directory and copy the class name

Graphical user interface, text, application

Description automatically generated

* go to the settings.py file inside the main folder and paste app\_name.apps.ClassName

Text

Description automatically generated

* A route is the path towards a certain webpage URL within a website
* Creating routes:
  + Go to blog/views.py and write the following code

Text

Description automatically generated

* + create a new file in the blog directory called urls.py

Text

Description automatically generated

* + Go to blog/urls.py and write the following code

Text

Description automatically generated

* + Go to urls.py file in the main folder and write the following code

Text

Description automatically generated

Handling routes:

* + When we go to <http://127.0.0.1:8000/about/>, django first looks in the urls.py file within the main folder.
  + Inside the urls.py file, it searches if the URL contains '' or 'admin' within the path
  + Notice <http://127.0.0.1:8000/about/> can be thought of as <http://127.0.0.1:8000/>'' about/
  + Since <http://127.0.0.1:8000/about/> has a path of '' which matches path('', include('blog.urls')), we now go to blogs/urls.py and the remaining part of the URL is sent. Thus, the URL string sent to blogs/urls.py is 'about/'
  + Inside the blogs/urls.py, it searches if the URL contains 'about/' or '' path
  + Since the URL string sent to blogs/urls.py is 'about/' which matches path('about/', views.about, name = 'blog-about'), we run the about method within the views.py file
  + The about method within the views.py file returns HttpResponse("<h1>About Page</h1>") and displays the following webpage.

Graphical user interface, text, application, chat or text message

Description automatically generated

* + To recap, <http://127.0.0.1:8000/about/> --> (main folder)/urls.py and satisfies path('', include('blog.urls')) --> blog/urls.py and satisfies path('about/', views.about, name = 'blog-about') --> blog/views.py and executes the home method

**Templates**

- Recall in the views.py file, we have a home method that returns HttpResponse('<h1>Blog Home</h1>')

- Theoretically, we could write all our html in that one line with brackets but that would be horrible.

- We can solve this issue by importing html files instead which are called templates

Creating Templates:

- create a templates directory inside the app\_name directory

- create an app\_name directory inside the templates directory

- inside the templates directory, create html files for each method inside the views.py file

Graphical user interface, text, application

Description automatically generated

- inside these html files, write the html code for the website

- go to views.py and change return HttpResponse('<h1>Home Page</h1>') to return render(request, 'blog/home.html' )

Template Inheritance:

- Like class inheritance, template inheritance allows one html file to inherit tags and elements from other html files

- For example, all html files need the basic html skeleton which can be inherited from.

Creating templates:

- create a base.html file within the templates/blog directory

- create a block tag and provide a name for this block which child templates can override

Text

Description automatically generated

- modify the html with the home.html and about.html files

Text

Description automatically generated

Parameters for HTML Files and Accessing Parameters within the HTML Files:

- to pass this dummy data into our html templates, we need to create a context dictionary and pass it into the render method

Text

Description automatically generated

- we can access this the value 'About' by referencing the key name in {{ }} brackets. Also notice we can use if statements and for loops in the html files.

Text

Description automatically generated

Text

Description automatically generated

Adding CSS Files to Templates:

- static files like css and javascript need to be in a static directory within our app

- static files don’t change like css and js files, but html files can change depending on the context parameters

- create a static directory in the blog directory

- create a blog directory inside the static directory. This directory should have the same name as the app.

- create a css file within the blog directory

Graphical user interface, application

Description automatically generated

- open the template where the css is being linked

- add {% load static %} at the top of the template

- create a link to the css file like normal

- when adding the css, the href = '{% static 'blog/main.css'%}'

Text

Description automatically generated

# The html and css files have been changed to follow the tutorial

Links in Templates

- notice we can link another template by letting the href = "{% url 'path-name'%}"

Text

Description automatically generated

Forms

- forms allow users to enter data

- we’ll create a new app called forms (below are the images for all the code)

- note that pressing submit in the form at forms/add doesn’t add any data, and that we had to create the form from scratch.

Graphical user interface

Description automatically generated with medium confidence

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Django has default forms

- django has an alternative way of doing the same thing

- go to views.py

Text

Description automatically generated

- go to add.html

Text

Description automatically generated

- now when go to <http://127.0.0.1:8000/forms/add/> we see

Graphical user interface, text, application

Description automatically generated

- This validation is called client-side validation. The browser makes sure the inputs are valid. However, client-side validation can be easily turned off or if the webpage is old(didn’t click refresh), the client may be able to bypass the new restraints in the refreshed page since it’s still stuck on the old page. Thus, we should also do client-side validation.

Thus, when to do client-side validation, we want to check if the request is a post

Text

Description automatically generated

Text

Description automatically generated

- however, since task is a global variable, all users see the same list of tasks

Graphical user interface, text, application, Word

Description automatically generated

to solve this, we use sessions

Sessions

- sessions let you store and retrieve data on a per-site-visitor basis

- django stores data about sessions in tables(talk more about it later)

Text

Description automatically generated

Text

Description automatically generated

- since the session table isn’t yet created, we have to run: python manage.py migrate

Graphical user interface, text, application, Word

Description automatically generated

**Admin**

- the admin page provides a GUI for creating/updating/deleting the backend data.

- to create an admin, we run the following commands in the main folder:

- python manage.py migrate

- python manage.py createsuperuser

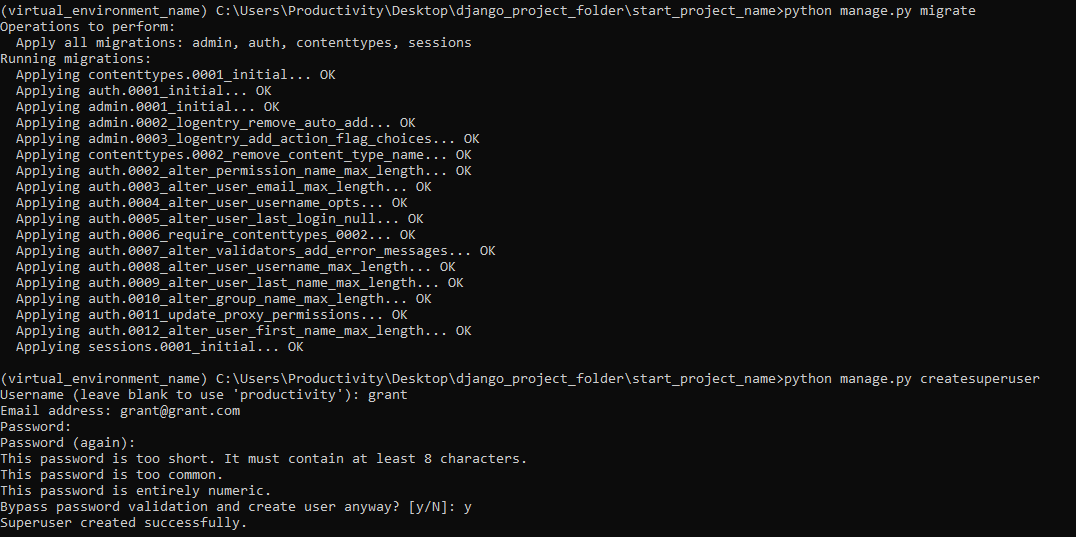
an admin is created, but the login info for the admin needs to be created as well

- create a username, ex: grant

- create an email, ex: [grant@grant.com](mailto:grant@grant.com)

- create a password, ex: 1

- type y and note that we want a strong username and password in a real-world application



- after creating the admin, we can go to <http://127.0.0.1:8000/admin> and log in with the admin credentials.

- There, we have an intuitive GUI which allows us to do backend work

Graphical user interface, website

Description automatically generated

- We can create a new user (not admin) by clicking Users

- Click add user

- Create login info and click save

- Now, when we go back to the users page, we can see the new user

Graphical user interface, text, application

Description automatically generated

**SQL Database and Migrations**

- databases allow us to create our own posts instead of relying on dummy data

- SQLite is a database language used to interact with databases and is the default database for django

In relational databases, we store data in tables where each table has rows and columns

A screenshot of a computer

Description automatically generated with low confidence

- to work with these databased, Django has its own ORM

- ORM stands for object relational mapper and allows us to access our database in an OOP way instead of having to learn SQL.

- You can use different databases without changing your code to query the database, you just change the settings file.

- data bases information is structured as classes which we call models, each class is going to be its own table in the database

Creating Models:

- open the models.py file in the blog folder

- the models.py file lets us think of things we want to save to the database

- we want to save posts for the blog into the database

- we will make a post model which will be a class that inherits from the Django model class

- each attribute of the class will be a different field in the database, ex: title of a post, content of post

Text

Description automatically generated

Updating Models:

- We should run the following code commands whenever models are changed

- go to the command prompt

- python manage.py makemigrations (creates the changes that I would like to apply to a database)

- python manage.py migrate (take those changes and actually apply them to the database)

Text

Description automatically generated

- notice we created a file at blog\migrations\0001\_inital.py

Text

Description automatically generated

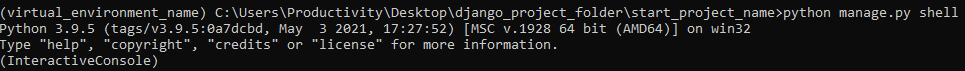
- these are instructions that reflect the changes made to the database

Django Python Shell

- the django python shell allows us to work with models

Shell Commands:

- to activate the shell, go to the command prompt and type python manage.py shell



- to use the User models as seen in the admin page, import the Users via from django.contrib.auth.models import User



- to see all the users in the database, type User.objects.all()



- to return the first user in the database, type User.objects.first()



- to return the last user in the database, type User.objects.last()



- to search for a certain user, type User.objects.filter(user\_attribute = 'desired\_user\_ attribute)



- to declare a desired user to be a variable, type user\_var = User.objects.filter(user\_ attribute = 'desired\_user\_ attribute).first()



- to view the desired user, type user\_var



- to view the attributes of the desired user, type user\_var.attributes() (notice the password is hashed)

Text

Description automatically generated

- to use the Post models as seen in the admin page, import the Users via from blog.models import Post



- to create a new post, type post\_var = Post(title = 'Blog1', content = 'First Post Content!', author = user1)



- to save the new post, type post\_var.save()



- to view all posts, type Post.objects.all()

- features like User.objects.first() and user\_var.attributes() are also available for Post models



- notice that the user\_var.author() returns a User, and we can see the user’s email with previous commands

Text

Description automatically generated

- to delete a post, type post\_var.delete()

Text

Description automatically generated

- to quit the shell, type quit()



Using data within databases

- go to views.py, import Post so that we can get all the Post objects. Whenever we have to use the Post class, import it.

Text

Description automatically generated

- go to admin.py

Text

Description automatically generated

Graphical user interface, application

Description automatically generated

**User Registration**

- we want to allow users to create an account, login, and make posts

Creating a Registration Page:

- create a new users app

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

- if we create a user, nothing happens since it creates a post request which the register(request) method doesn’t handle

Handling Registration Form Data:

- Since we set method = 'POST', we need to put a condition in views.register to handle the post request and validate our form data.

- If we get a get request, the method runs as before.

- add form.save() inside the form.is\_valid() if-statement (forgot to add in the picture)

Text

Description automatically generated

- update our base template to have the flash messages

Text

Description automatically generated

- If we submit two passwords that do not match, the username isn’t removed so the user doesn’t have to rewrite everything.

- An invalid submission would go into the if request.method == "POST": and then create a form = UserCreationForm(request.POST) with the username data and 2 password fields. Then we check if its valid and we see that it is not since the passwords don’t match. Thus, it falls to the return render(request, 'users/register.html', {'form': form}). Remember our form was instantiated with the request data so its still going to have those form fields fill in along with added error messages.

Changing Form Fields:

- create a new file in users directory called forms.py and add additional fields such as email

Text

Description automatically generated

- go to views.py and import UserRegisterForm. Replace all instances of UserCreationForm with UserRegisterForm

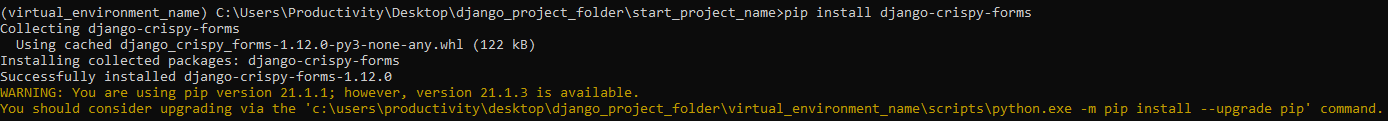
Text

Description automatically generated

Crispy Forms:

- crispy forms is a third-party Django application which allows us to put style our form in a bootstrap fashion inside the templates

- we need to install crispy forms in main folder with the command: pip install django-crispy-forms



- then go to settings module in main and add crispy forms

Text

Description automatically generated

Text

Description automatically generated

- add crispy forms to the register.html template

Text

Description automatically generated

**Login/Logout System**

- go to urls.py in main folder

Text

Description automatically generated

- runserver

Graphical user interface, text, application

Description automatically generated

- change directory paths

- notice we get an error since its looking for a template at registration/login.html. We can change the path and make the html files

Text

Description automatically generated

Graphical user interface, application

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

- try to log in and receive the following error of trying to access a url that doesn’t have a view attached to it. The url it’s trying to access is /accounts/profile/ which we don’t have right now, we also don’t want the user to be sent to their profile, but rather the home page

Text

Description automatically generated

- we can modify that location using our settings to redirect users to the home page

Text

Description automatically generated

- now when we try to log in, it works. Even though there isn’t any visual feedback, admin page is only accessible when u log in via admin. As well, logout now works and doesn’t display the admin logout

- Now, we need to assign URLs to the login/register links in the nav bar if the user is not logged in. If they are, they can go to their profile(we will make later) or logout. Go to base template

Text

Description automatically generated

Restricted Routes:

- now we will put a restriction on certain routes so that routes are accessible only if you are logged in

- ex: you can only change your fb profile if you are logged in

- go to users views and create a profile view

Text

Description automatically generated

- create the profile template inside the same folder as the login and logout templates

Text

Description automatically generated

- add view to urls.py in main folder

Text

Description automatically generated

- we have a problem since although everything works, nothing is stopping us from manually typing in /profile to bypass the authentication. To resolve this problem, we have to use a login required decorator that Django provides

Text

Description automatically generated

- however, when a user who is not logged in tries to go to the profile/ view, they are met with this error. Notice the url is <http://127.0.0.1:8000/login/?next=/profile/>. The next has a parameter of profile which means that after u login, the website takes u to the profile page and not some other random page

Graphical user interface, text, application

Description automatically generated

- Django is trying to redirect the user to the default login route, but we changed it. To resolve this problem, go to settings

Text

Description automatically generated

**Adding a Profile Picture**

- the default user model doesn’t have a field for a profile picture.

- then we have to extend user model and create a new profile model that has a 1 to 1 relationship with the user (1 user has 1 profile, 1 profile belongs to 1 user)

Text

Description automatically generated

- install pillow so we can upload images and make migrations since we made changes to the models

Text

Description automatically generated

- add the Profile model to the admin page by going to users/admin.py

Graphical user interface, text

Description automatically generated

- go to admin page and create a profile

Background pattern

Description automatically generated

- we now see a new folder in called profile\_pics in the main folder, but we prob want to move it

**Graphical user interface, text, application

Description automatically generated**

- to move it, we go to settings and import os and add to the bottom



Graphical user interface, text

Description automatically generated

- add the media routes to urls.py in main folder

Text

Description automatically generated

- now, go to admin and delete all profiles and recreate the profiles

- now, we see a media folder inside the base directory of our project, and we can delete the previous pfp folder

Graphical user interface, text, application

Description automatically generated

- go to profile template and make it look better while also adding the username and email fields

Text

Description automatically generated

**Once a user is made, so is their profile**

- the following is too confusing to understand, so just do it

- create a new file our users directory called signals.py

Text

Description automatically generated

- go to users/apps.py

Text

Description automatically generated

**Updating User Profiles Through Forms**

- we will create a form to allow users to update their profile information

- we will create a new model form which allows us to create a form that will work with a specific database model (in our case, the user model)

- open users/forms.py

Graphical user interface, text

Description automatically generated

- open users/views.py

Text

Description automatically generated

- go to profile.html

Text

Description automatically generated

Graphical user interface, application

Description automatically generated

**Processing Update Form Data**

- we need to do a post request back to the same route with the data from the form

- go to users/views.py

- first, populate these regular forms with the current information with the logged in user. Remember these are model forms that are expecting to be working on a model object so we can populate the fields of the form by passing in an instance of an object it expects

Text

Description automatically generated

**Resize Images**

- go to users/models

Text

Description automatically generated

**Add Profile Picture to each Post**

- go to home.html

Text

Description automatically generated

Create/Update/Delete Posts

- before we had function-based views, now we will use class-based view

- class-based views have a lot more built-in functionality

- there are list, detail, create, update, and delete class views

- blog home lists all the blogs which would be list view

- click on a blog shows more detail so that would be a detail view

- updating and deleting blogs would be update/delete views

- class-based views will look for templates in the naming convention: <app\_name>/<model\_name>\_<viewtype>.html

Creating List View of Home Page:

- by default, the list view will look for templates in blog/post\_list.html, but we changed the path

Text

Description automatically generated

- go to urls.py in blog folder

Text

Description automatically generated

Detail view

view.py

by default, the list view will look for templates in blog/post\_detail.html, but we changed the path

Text

Description automatically generated

urls.py

Text

Description automatically generated

create post\_detail.html file in templates/blog

Text

Description automatically generated

go to templates/blog/home.html

Text

Description automatically generated

Create View

views.py

Text

Description automatically generated

urls.py

Text

Description automatically generated

you might expect the name of the path to be blog/post\_create.html, but it will share a template with the update view that we will soon create. Instead, the path name will be blog/post\_form.html and let’s create that file in the blog templates

Text

Description automatically generated

if we try to create a post, we get an error since it says the author is null which is not allowed. every post must have an author. we want the author to be the current logged in user, which we can do by overriding our form valid method for our create view.

Text

Description automatically generated

we have to redirect the user, and we’ll redirect to the detail view of the post

open up blog/models.py

Text

Description automatically generated

import LoginRequiredMixin to redirect user to login page if user is not logged in

Text

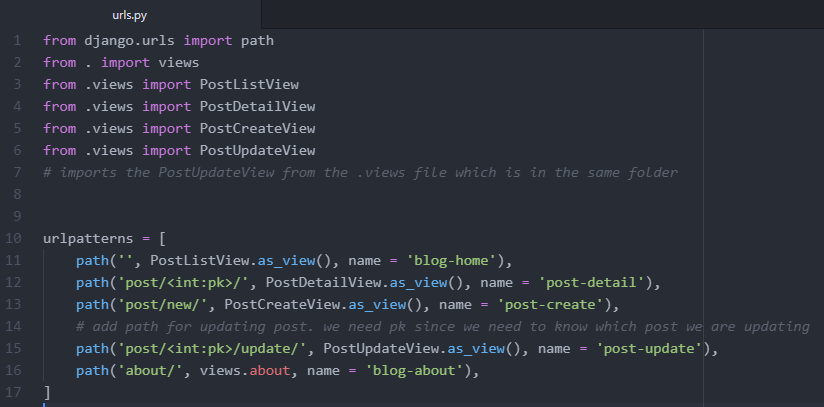
Description automatically generated

Update Views

Text

Description automatically generated

go to blog/urls.py



the template it uses is teh same tempalte as the PostCreateView which was the post\_form.html

We want to make sure a logged in user can’t update the posts of another user

Text

Description automatically generated

delete view

Text

Description automatically generated

Text

Description automatically generated

urls.py

Text

Description automatically generated

the template it expects is called post\_confirm\_delete.html so create it in the blog templates

Text

Description automatically generated

Links to update, create

link to create post in base template

Text

Description automatically generated

link to delete/update post in post\_detail

Text

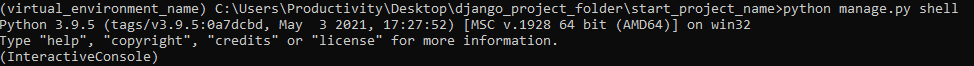
Description automatically generated

Pagination

- pagination allows us to load in a certain number of posts on one page and then display links at the bottom to go to next/previous pages

Paginator Object

- open shell



- import from django.core.paginator import Paginator to use the Paginator object



- type: posts = ['1','2','3','4','5'] to create some dummy data



- to have at most 2 posts per page, type: p = Paginator(posts, 2)



- to return the total number of pages, type: p.num\_pages

 posts 1 and 2 are on page 1, posts 3 and 4 are on page 2, and post 5 is on page 3

- to print all the pages, type:

for page in p.page\_range:

... print(page)

Text

Description automatically generated

- to get the first page in a page variable, type: p1 = p.page(1)



- return the page variable with p1



- return the page variable’s number with p1.number



- return the post in the page variable with p1.object\_list



- return if the current page has a previous page with p1.has\_previous()



- return if the current page has a next page with p1.has\_next()



- return the page number of the next page p1.next\_page\_number()



go to blog/views.py

Text

Description automatically generated

go to home.html

Text

Description automatically generated

Create Posts Only From Certain User

go to blog/views.py

Text

Description automatically generated

go to blog/urls.py

Text

Description automatically generated

create user\_post.html

Text

Description automatically generated

go to home.html

Text

Description automatically generated

go to post\_detail.html

Text

Description automatically generated

Text

Description automatically generated

**Sending Email:**

Have less secure apps on

Unlock captcha

<https://accounts.google.com/DisplayUnlockCaptcha>

Go to settings:

add to the bottom

Text

Description automatically generated

create a template such as

Text

Description automatically generated

in the view to call the template,

Text

Description automatically generated

**Deployment**

deploy links:

<https://www.youtube.com/watch?v=kBwhtEIXGII&t=1162s>

do css/bootstrap/js in same file as html

download:

<https://devcenter.heroku.com/articles/heroku-cli>

<https://git-scm.com/downloads>

create account:

<https://www.heroku.com/>

<https://imgbb.com/> upload all image files and the src for these images in the html should be links

<https://github.com/>

<https://www.godaddy.com/>

Graphical user interface, website

Description automatically generated

click create new app

Graphical user interface, text, application, email, Teams

Description automatically generated

click open app and copy domain url

in main folder with env activated, type heroku login and connect with heroku account

git init

heroku git:remote -a appNameOnHeroku

pip install gunicorn whitenoise

pip freeze > requirements.txt

create a runtime.txt file

Text

Description automatically generated

create a Procfile with no extension

Graphical user interface, text

Description automatically generated

Text

Description automatically generated

go to settings and add domain url that we got from open app on heroku, change debug to false, set static root, and add whitenoise to middleware

Graphical user interface, application

Description automatically generated with medium confidence



Text

Description automatically generated

go to github

add files within main folder to private repo so sensitive info is not leaked

Graphical user interface, text, application, email

Description automatically generated

go to heroku

Graphical user interface, application, Teams

Description automatically generated

in settings, go to add buildpack and click python

go to deploy and click github

connect to github

find repository

click deploy branch

Graphical user interface, text, application

Description automatically generated

Make sure Heroku is fast, use <https://uptimerobot.com/> and read <https://bit.ly/2WxxZXz>

change domain name

<https://www.youtube.com/watch?v=RRFxHju8pWM>

<https://www.youtube.com/watch?v=CxrzD73r6Rw&list=LL&index=2&t=532s>