E-Buy Shopping

Assignment-4

Team – 2:

Shirish Srinivasa

Kavya Raviprakash

Luke Henkenius

Akhila Pillalamarri

ISQA 8380

Enterprise Architecture and System Integration

Index

Team Members and Roles:

Shirish Srinivasa – Developer / Tester

Kavya Raviprakash – Project Manager / Developer

Luke Henkenius – Developer / Tester

Akhila Pillalamarri - Scrum master / Developer

Business use and description of the web application

Ecommerce or electronic commerce refers to a business model that involves sales transactions being done on the web. Virtually every online shopping website - big or small - follows this structure. Any site where you can obtain items for sale over the internet is considered an ecommerce website.

The purpose for designing this ecommerce (shopping-cart) application is because most of the people nowadays prefer shopping online rather than spending a lot of time at physical markets. It basically saves the shoppers time, and they can easily browse through hundreds of products from the place of their convenience.

Through the use of an ecommerce application, a retailer can sell his products to customers who might be living at distant places and a customer can easily browse through products, can also compare their prices and can buy the products that would suit their need.

This web application will be built using the python and Django Framework integrated with various APIs to improve the web application's functionality.

Admin Features:

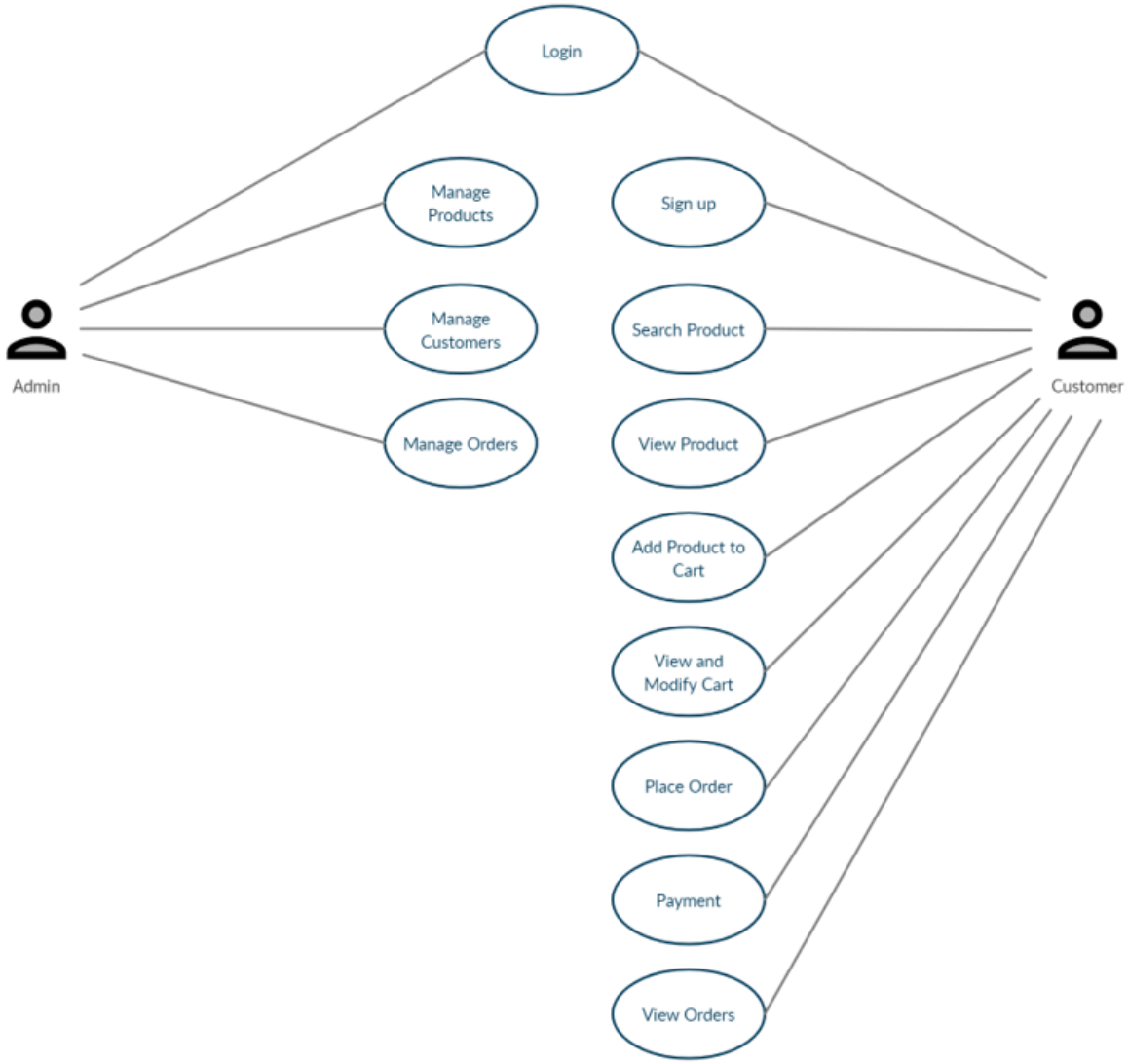
- Admin can login to the web application
- Admin can manage registered customers and also add new users to the system if needed
- Admin can View, Add, edit, and delete product categories.
- Admin can View, Add, edit, and delete product
- Admin can View, Add, edit, and delete orders and order details

Customer Features:

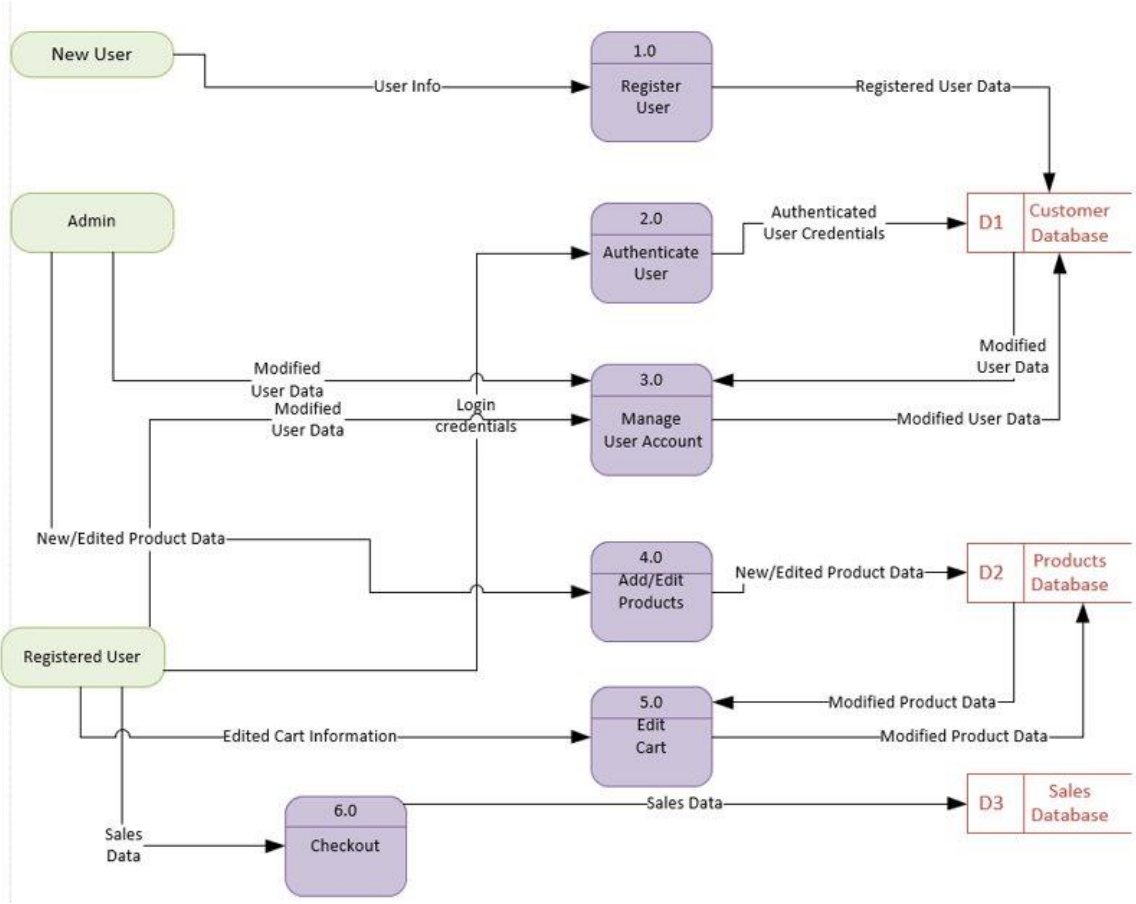
- Customer can sign up to the web application through filling a signup page or by authenticating through google sign up.
- Customer once registered can login to the web application.
- Customer can search products based on categories.

- Customer can view product description and perform currency conversion to view the prices of the product in different currency.
- Customer can add product to the and once he is completed can checkout and place the order.
- Customer will receive email and text notification, on successful order placement.
- Customer can view his order information on the orders that he has placed.
- Customer can download invoice copy of his orders.

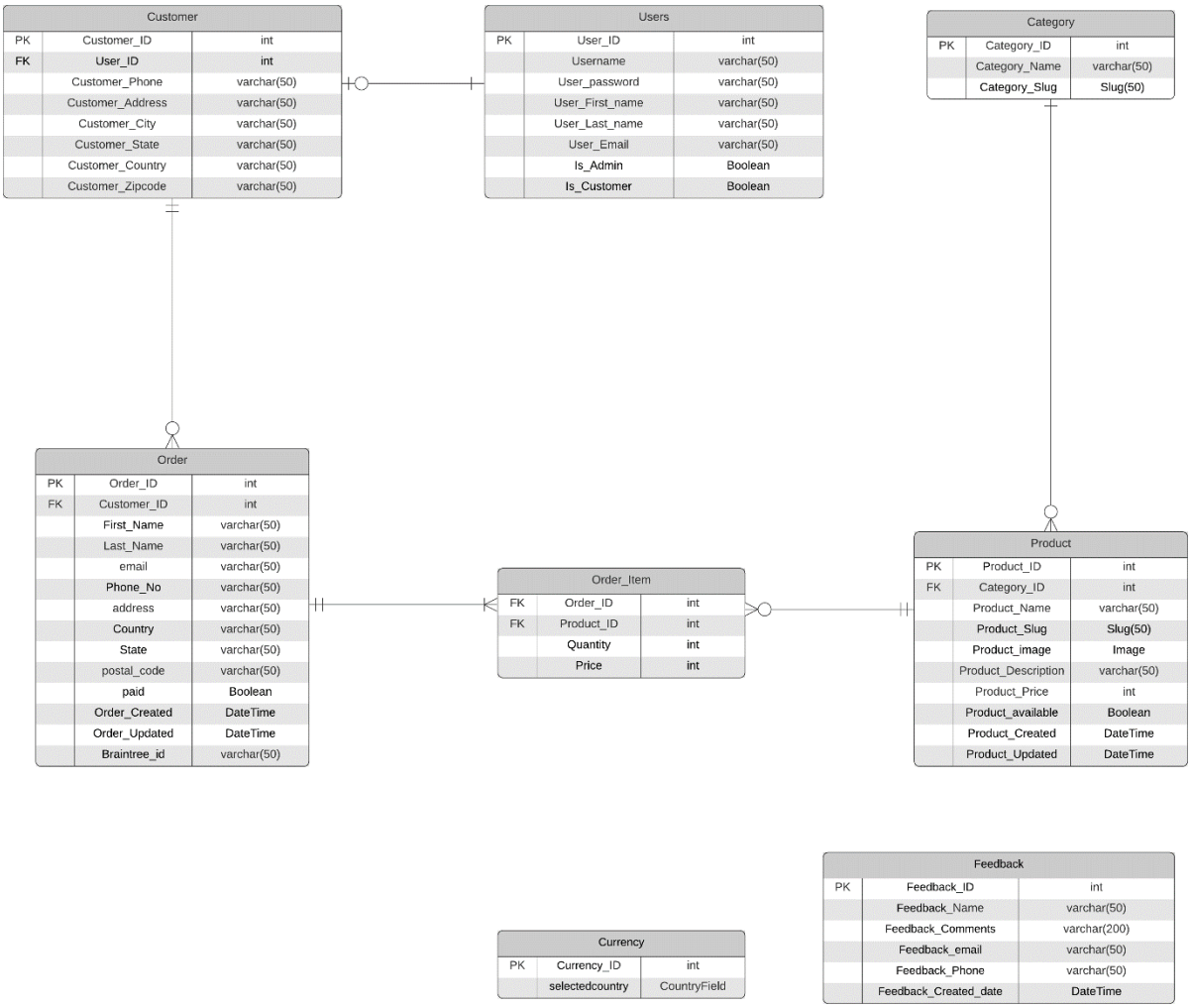
Use Case Diagram



Data Flow Diagram



Data Model


Customer
PK Customer_ID int
FK User_ID int
Customer_Phone varchar(50)
Customer_Address varchar(50)
Customer_City varchar(50)
Customer_State varchar(50)
Customer_Country varchar(50)
Customer_Zipcode varchar(50)
Users
PK User_ID int
Username varchar(50)
User_password varchar(50)
User_First_name varchar(50)
User_Last_name varchar(50)
User_Email varchar(50)
Is_Admin Boolean
Is_Customer Boolean
Category
PK Category_ID int
Category_Name varchar(50)
Category_Slug Slug(50)
Order
PK Order_ID int
FK Customer_ID int
First_Name varchar(50)
Last_Name varchar(50)
email varchar(50)
Phone_No varchar(50)
address varchar(50)
Country varchar(50)
State varchar(50)
postal_code varchar(50)
paid Boolean
Order_Created DateTime
Order_Updated DateTime
Braintree_id varchar(50)
Order_Item
FK Order_ID int
FK Product_ID int
Quantity int
Price int
Product
PK Product_ID int
FK Category_ID int
Product_Name varchar(50)
Product_Slug Slug(50)
Product_image Image
Product_Description varchar(50)
Product_Price int
Product_available Boolean
Product_Created DateTime
Product_Updated DateTime
Currency
PK Currency_ID int
selectedcountry CountryField
Feedback
PK Feedback_ID int
Feedback_Name varchar(50)
Feedback_Comments varchar(200)
Feedback_email varchar(50)
Feedback_Phone varchar(50)
Feedback_Created_date DateTime

System Requirements:

Hardware Requirements:

Minimum requirements:

- Dual core processor
- 4GB RAM
- 10GB Storage

Maximum requirements

- Quad Core processor
- 8GB RAM
- 50GB Storage

Software Requirements:

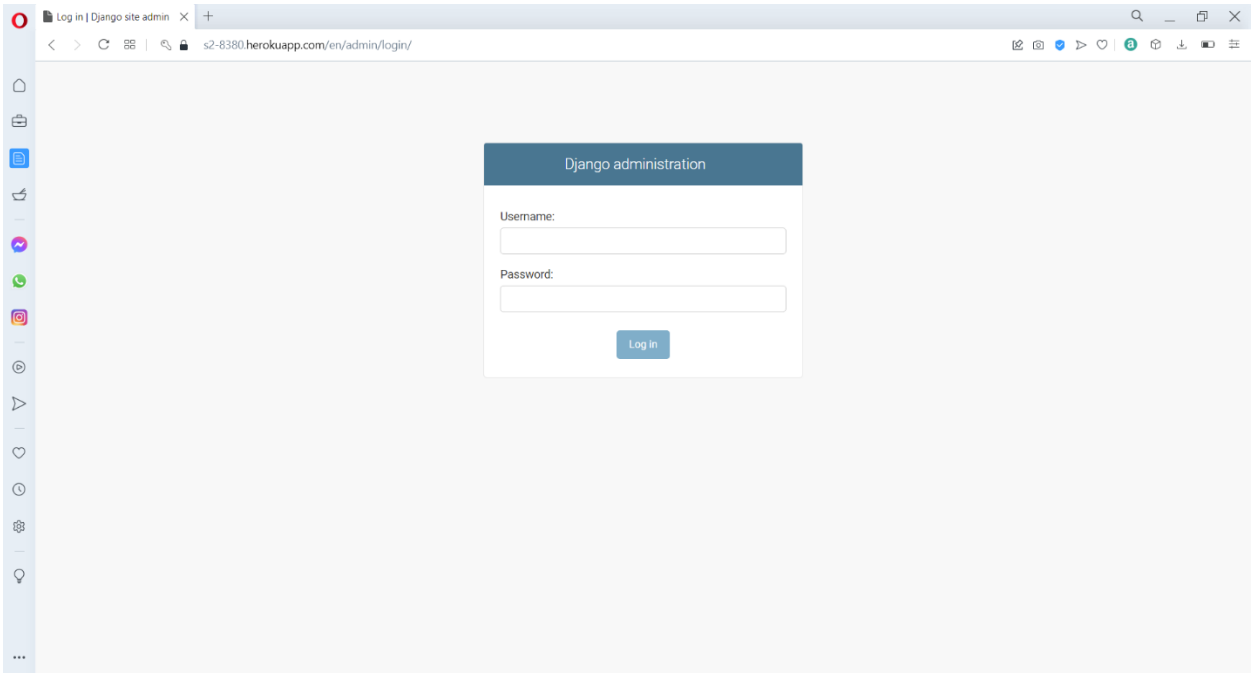
- PyCharm
- Python 3.9
- GitHub Desktop
- Windows 7 or later
- Web browser: Chrome or Firefox

Heroku Requirements:

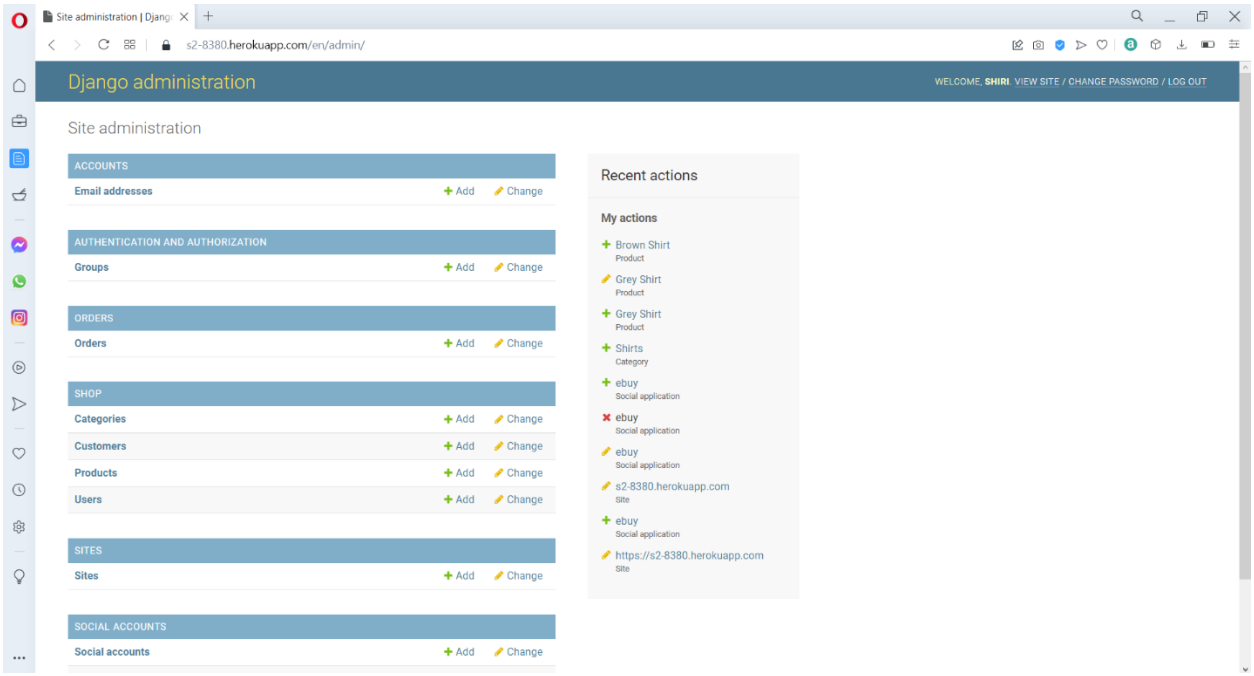
- Internet Connected devices
- Web browser: Chrome or Firefox
- GitHub account and access
- Heroku account and app access

Screenshots

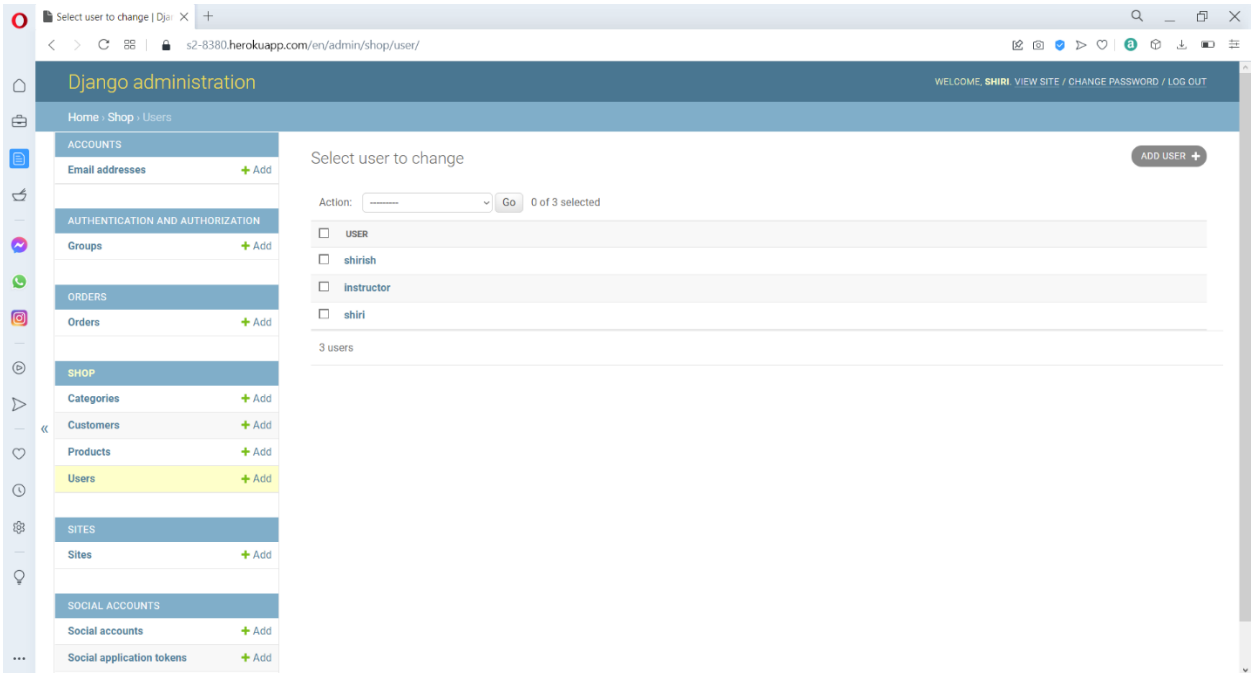
Admin



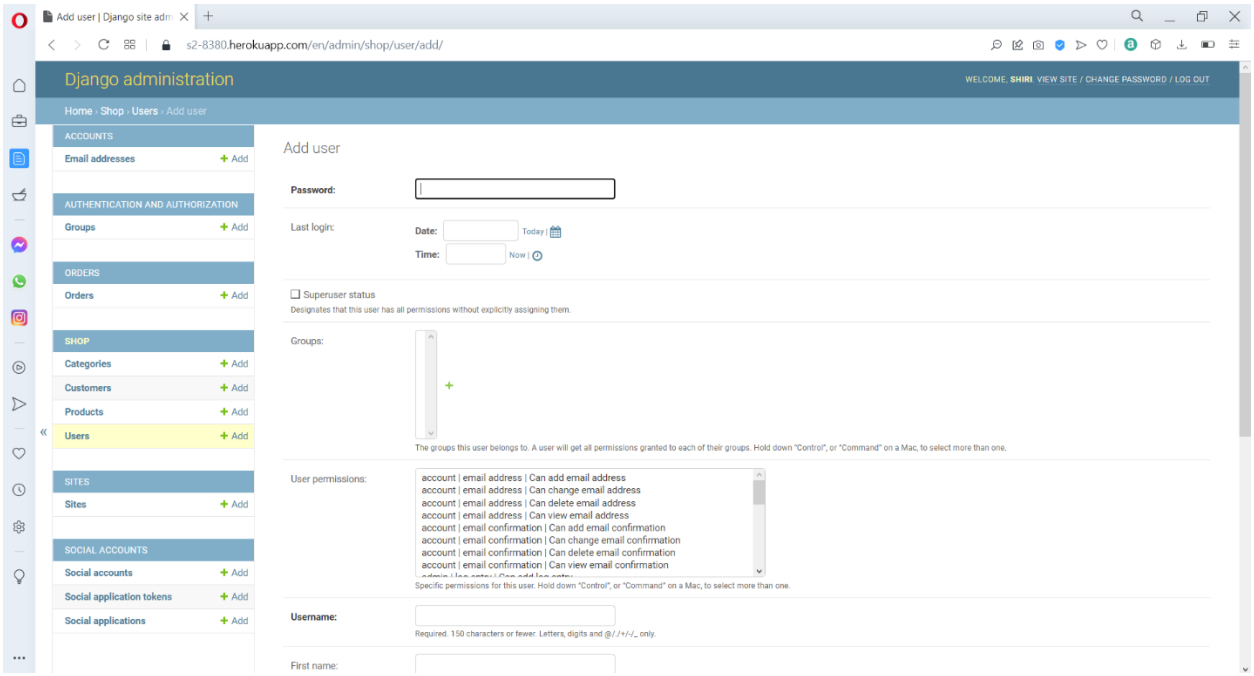
This sis the login screen of the web application for admin.



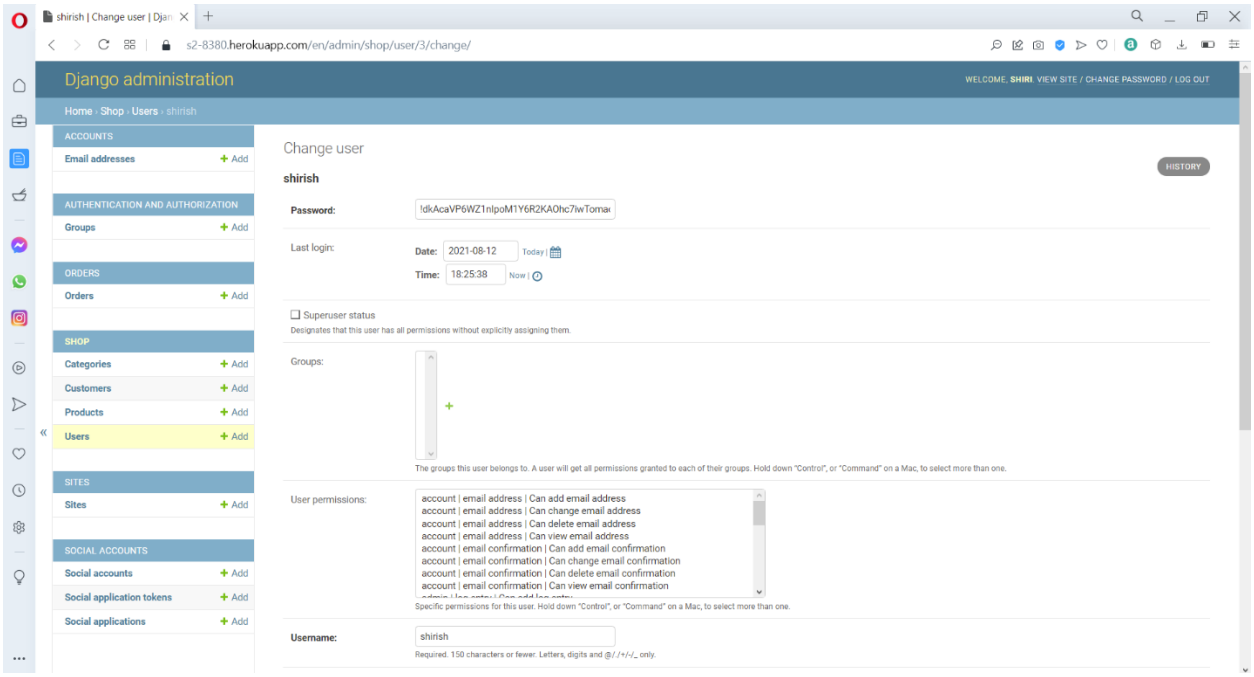
This is the Site administration panel for the ebuy shooing web application.



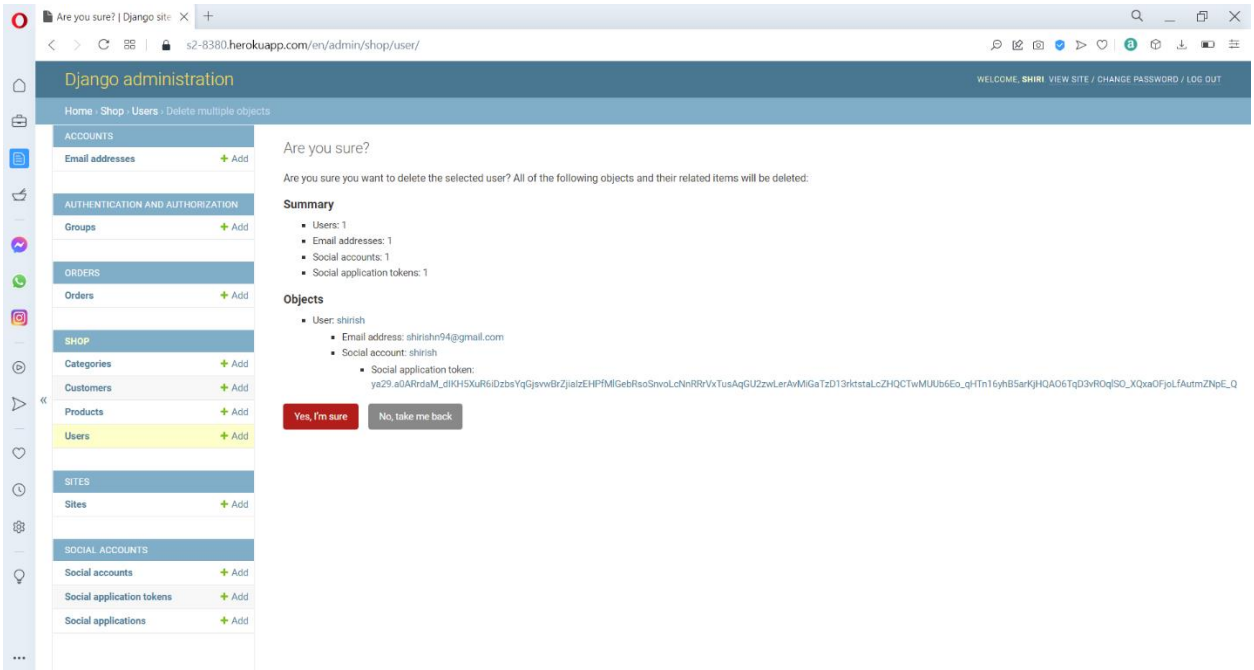
This is the User information page where admin can add, create, and edit new or existing users.



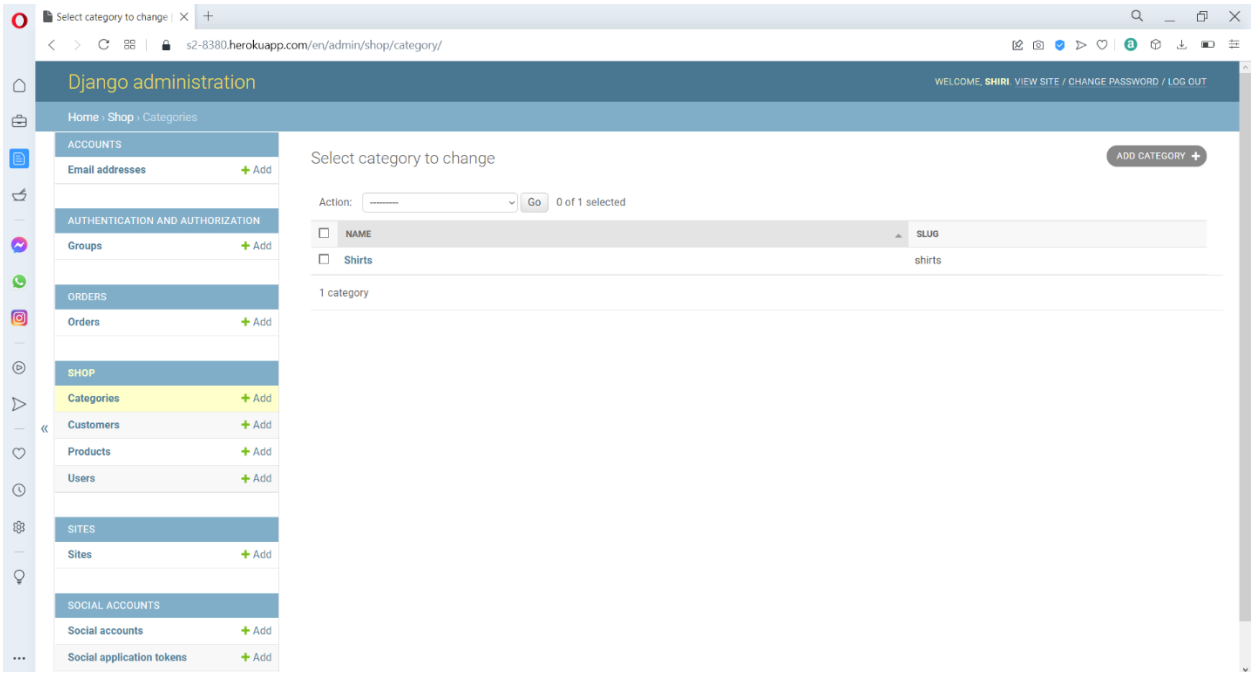
This is the add user page, where the admin can enter a username and password information for creating a new user account.



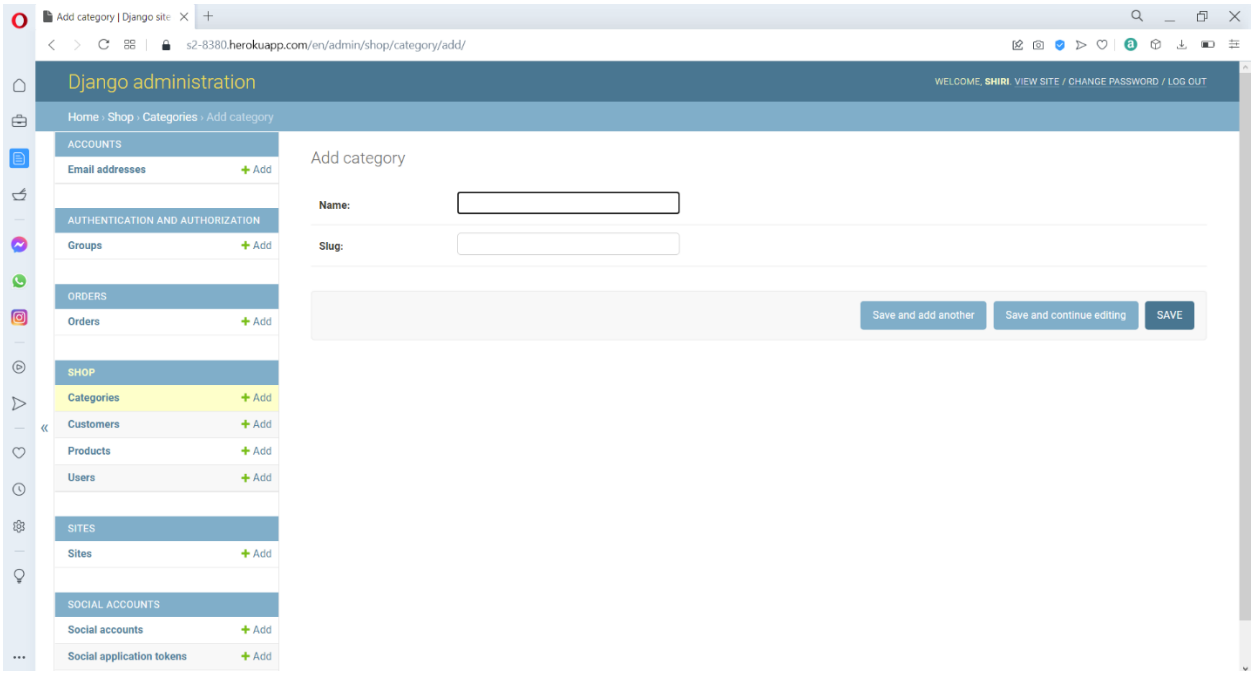
This is the change user page, where admin can edit existing user, and we must also add out email address, so that that application can send the forgot password link to recover the account.



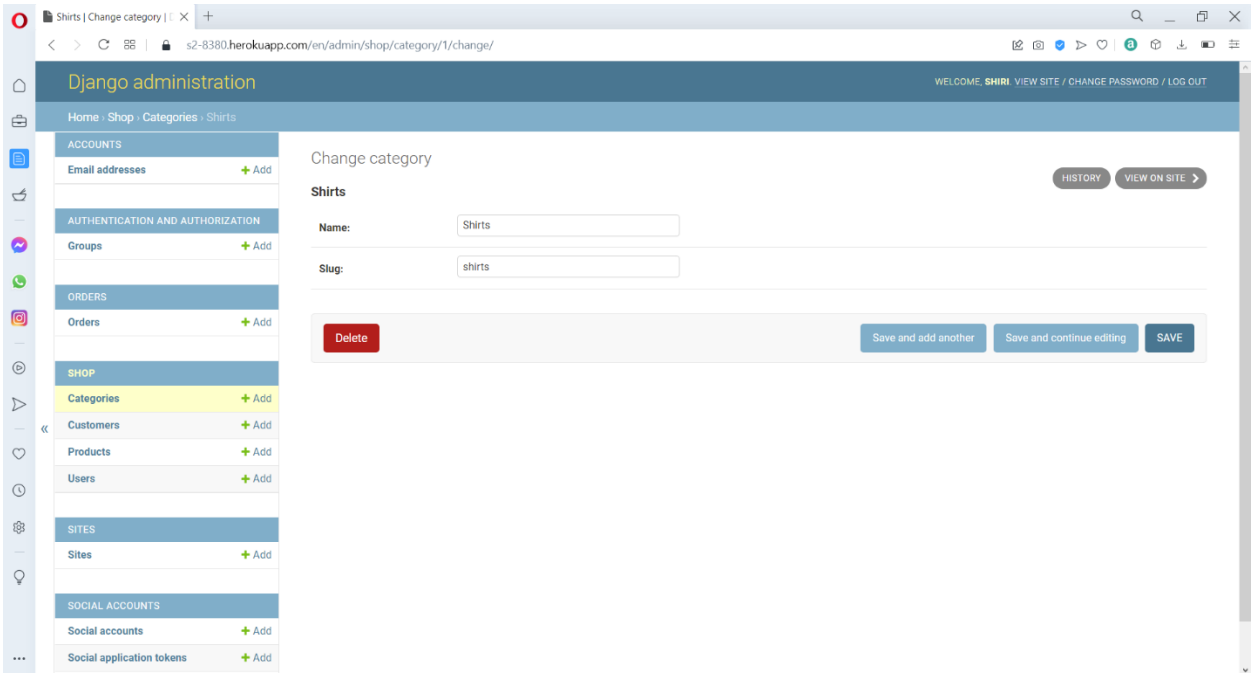
This is the user delete confirmation page, when admin click on delete user, we will be redirected to a web page to confirm our decision to delete the user.



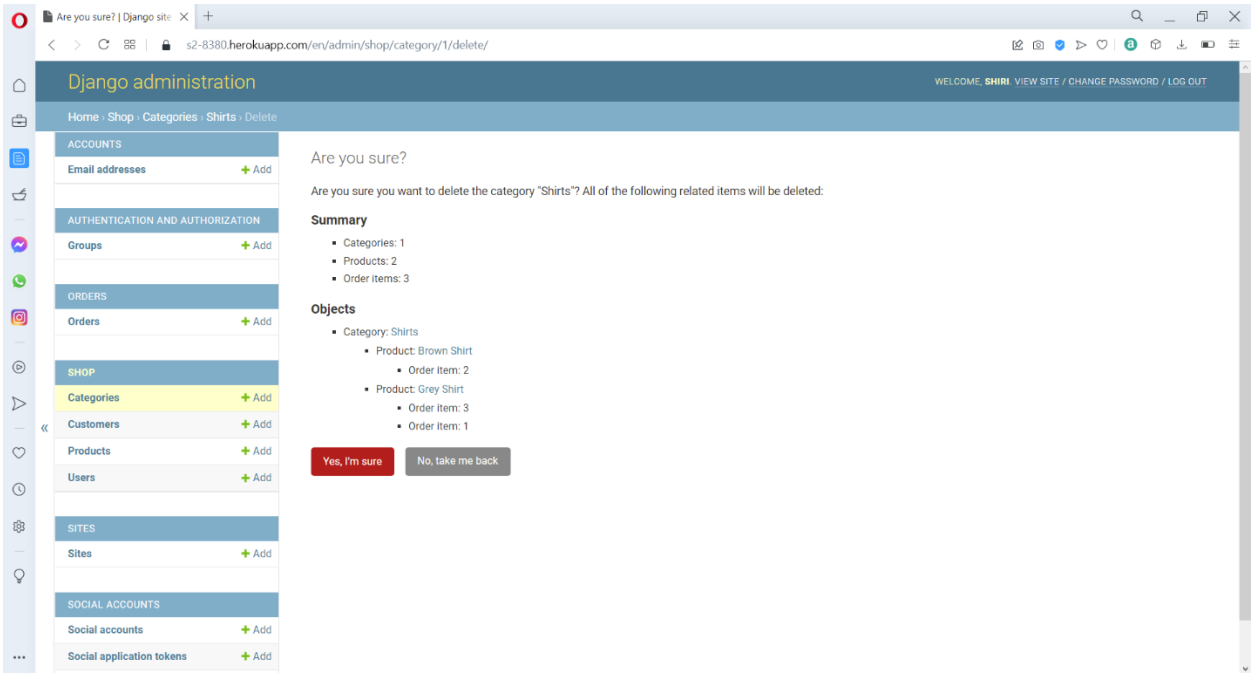
This is the category list page where admin can add, create, and edit new or existing categories.



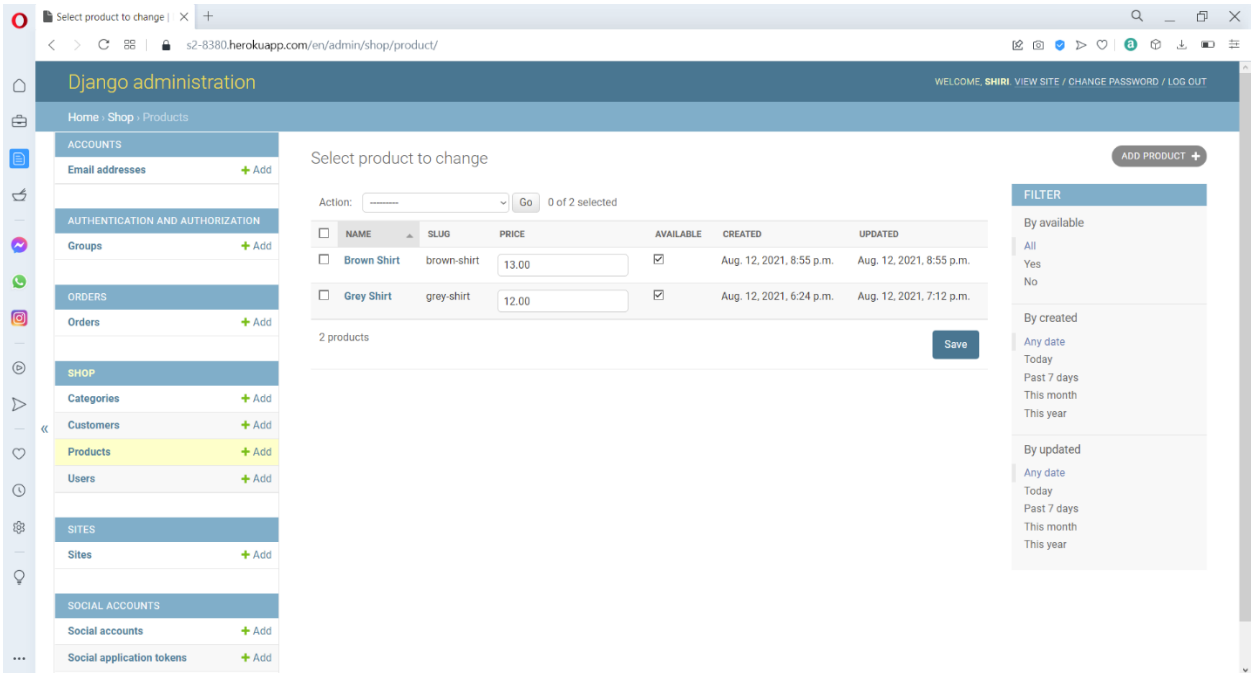
This is the Add category page, where an admin can Add or create new category for the web application.



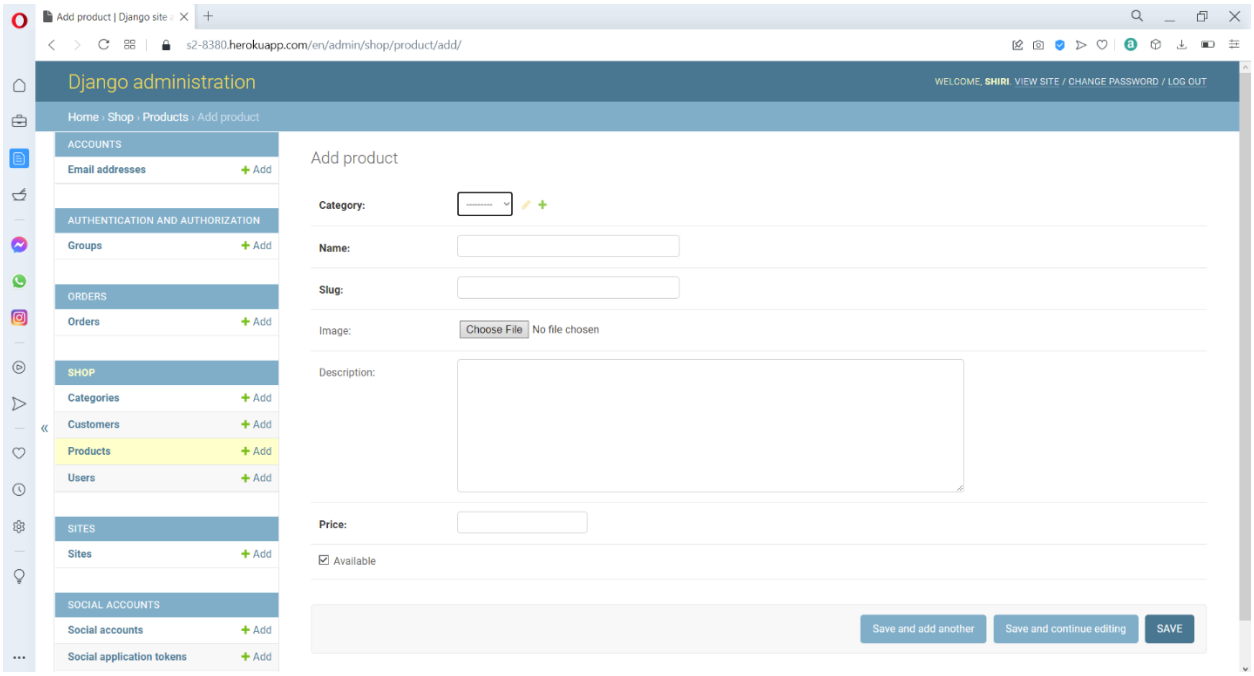
This is the change category page, where admin can edit and change details of existing customers.



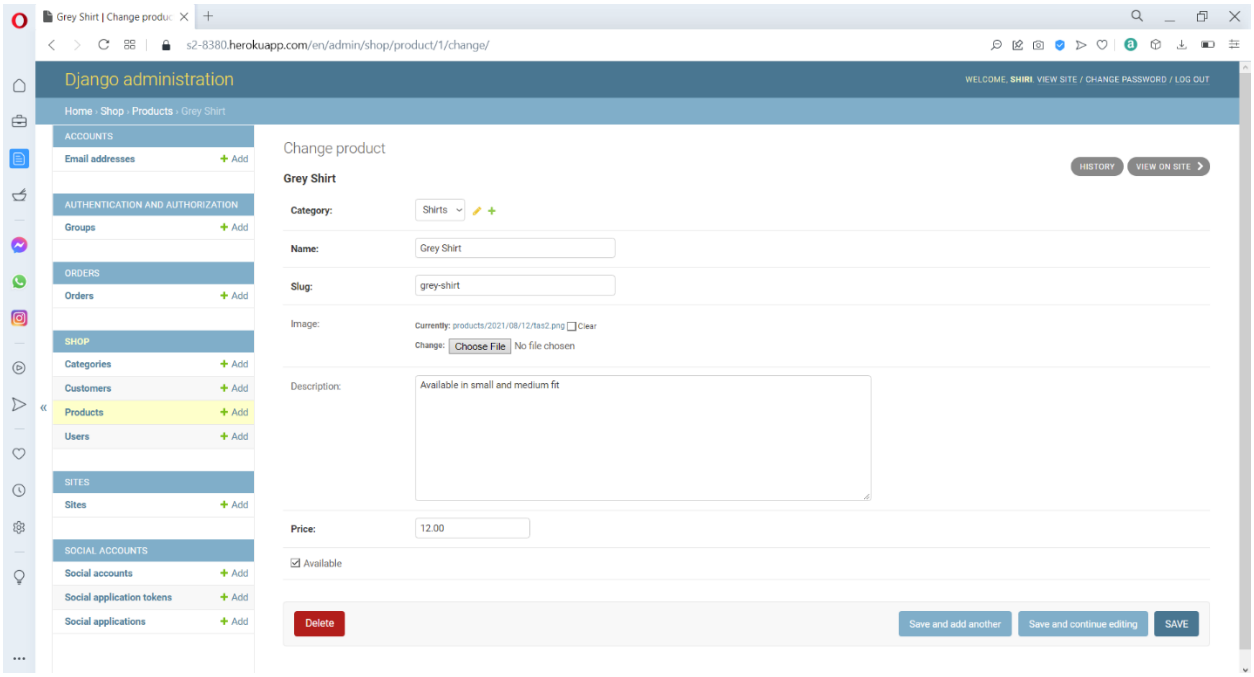
This is the customer delete confirmation page, where when admin click on delete customer, we will be redirected to a web page to confirm our decision to delete the customer, and when that customer is deleted, all the related information in orders will also be deleted.



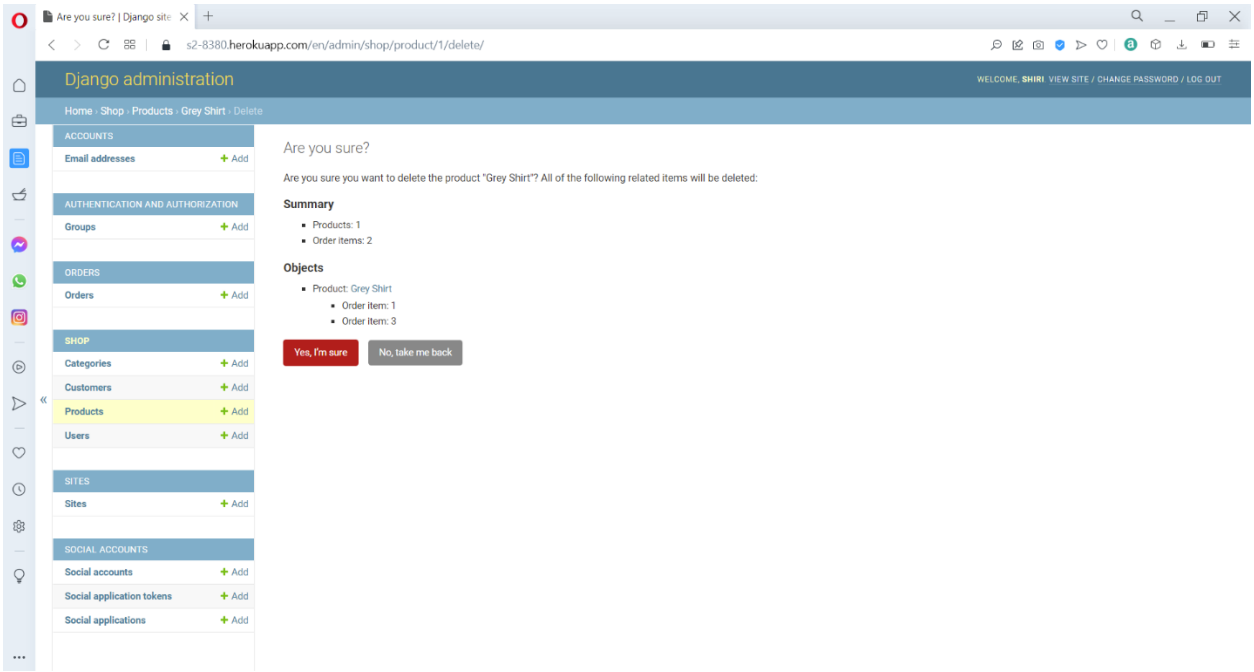
This is the product list page where admin can add, create, and edit new or existing products.



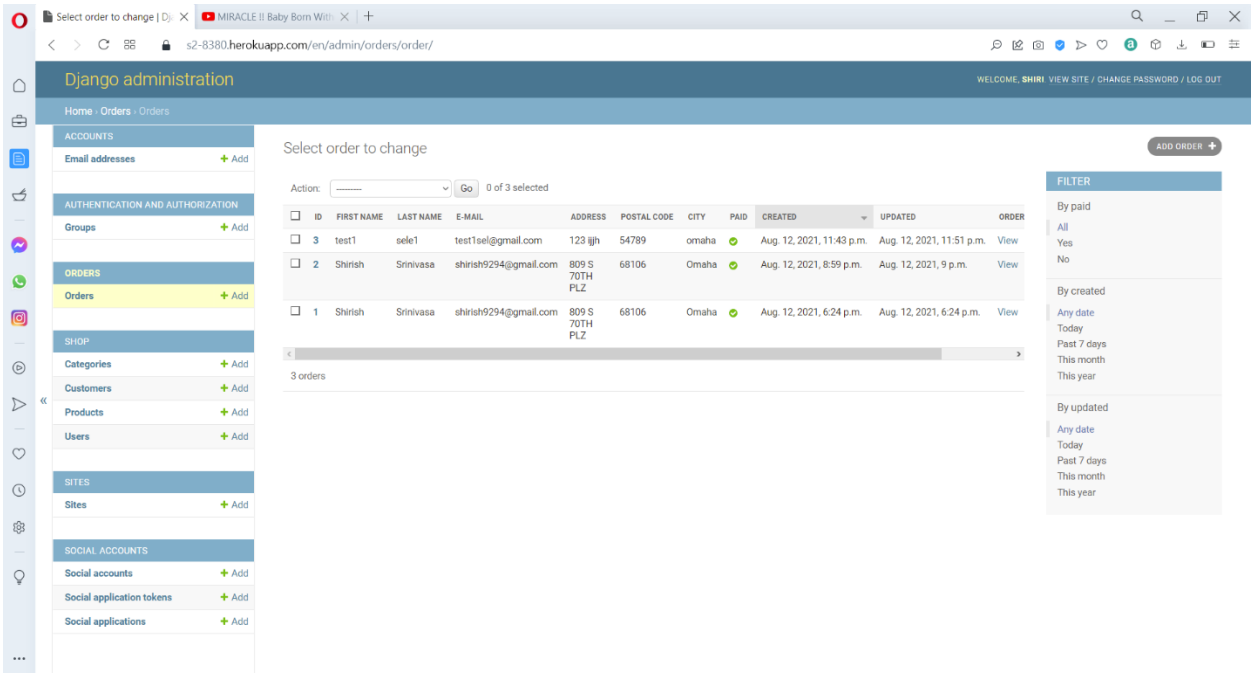
This is the Add product page, where an admin can Add or create new product for the web application.



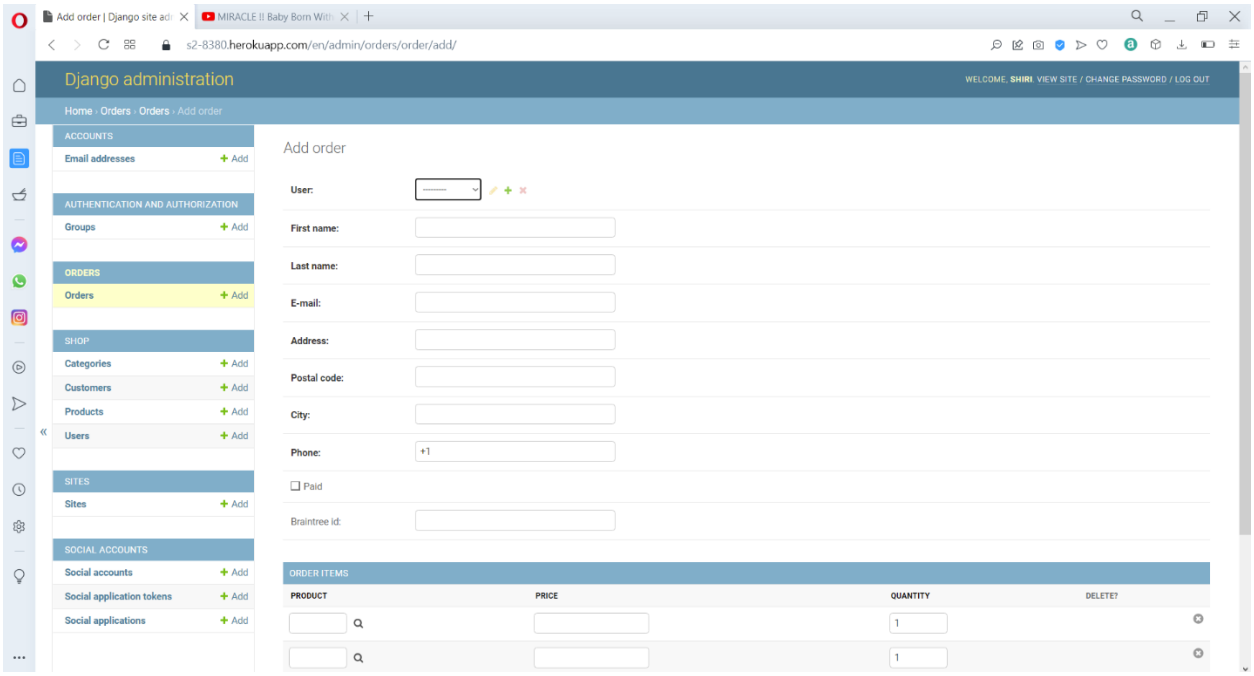
This is the change product page, where admin can edit and change details of existing products.



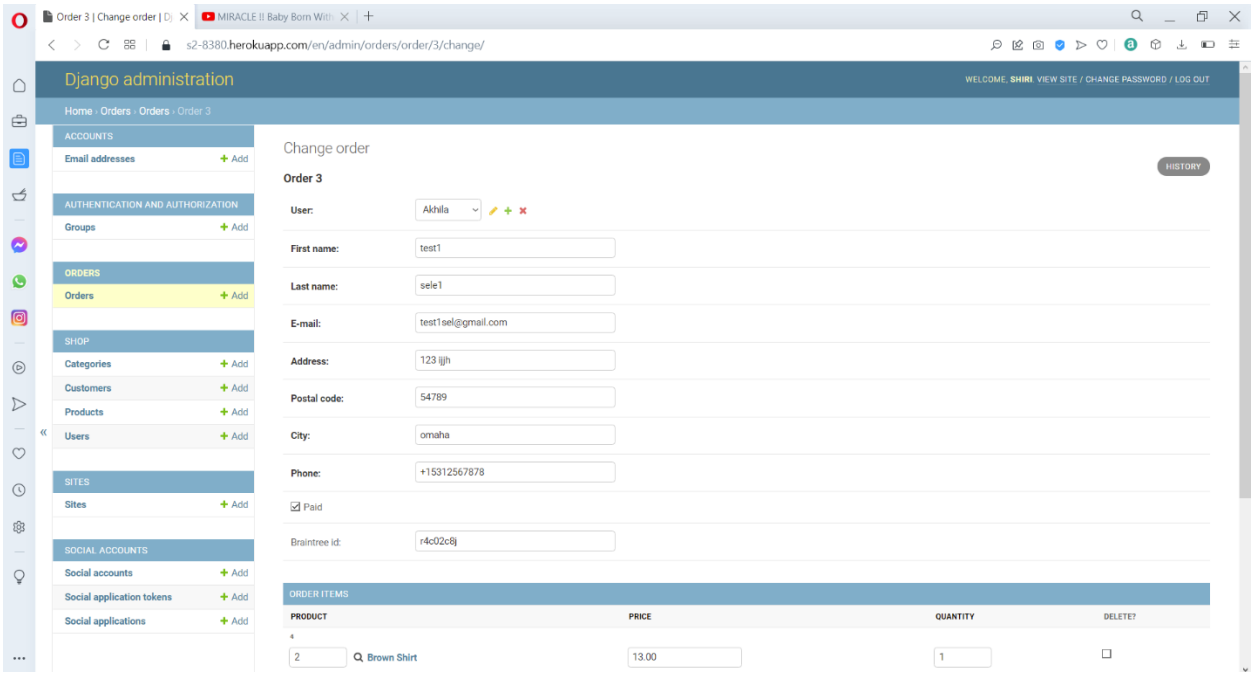
This is the product delete confirmation page, when we click on delete product, we will be redirected to a web page to confirm our decision to delete the product.



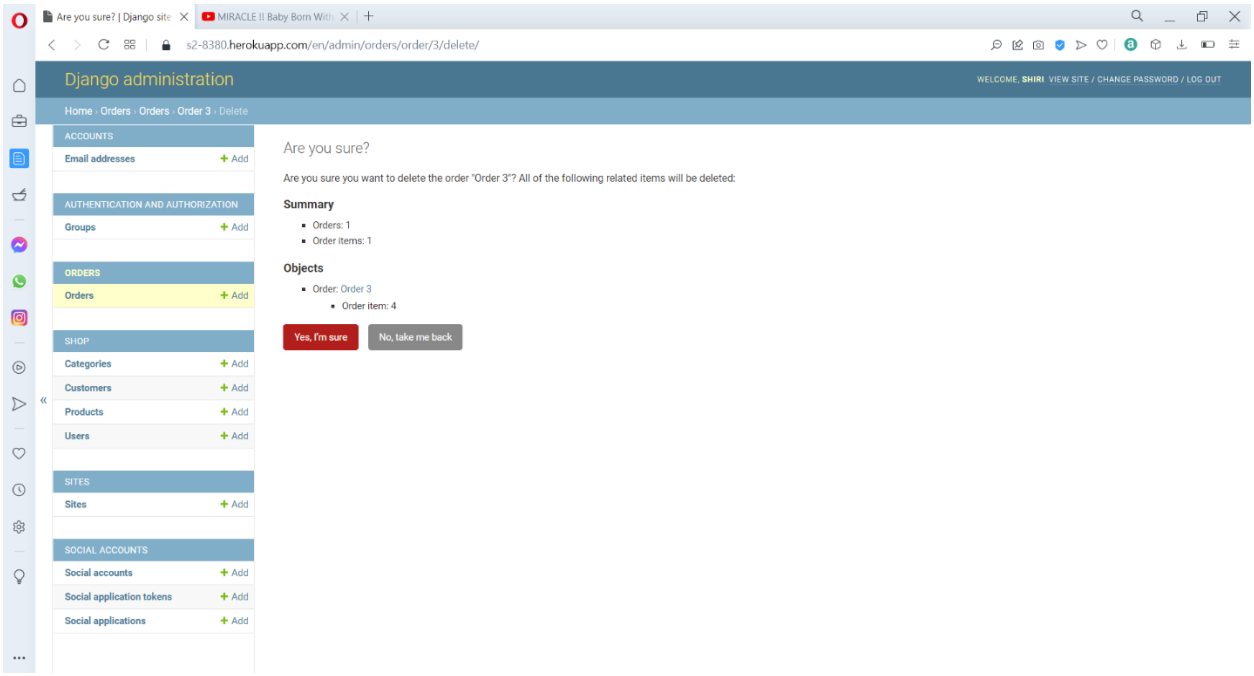
This is the order list page where admin can add, create, and edit new or existing orders.



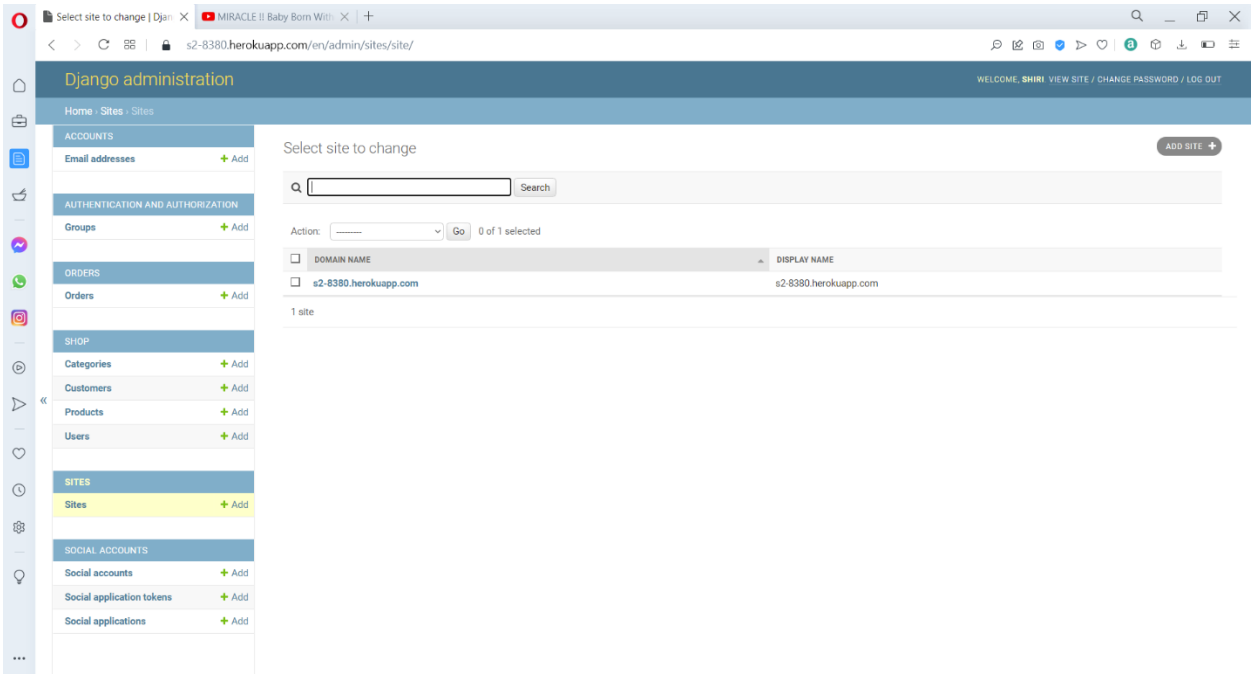
This is the Add order page, where an admin can Add or create new orders in the web application.



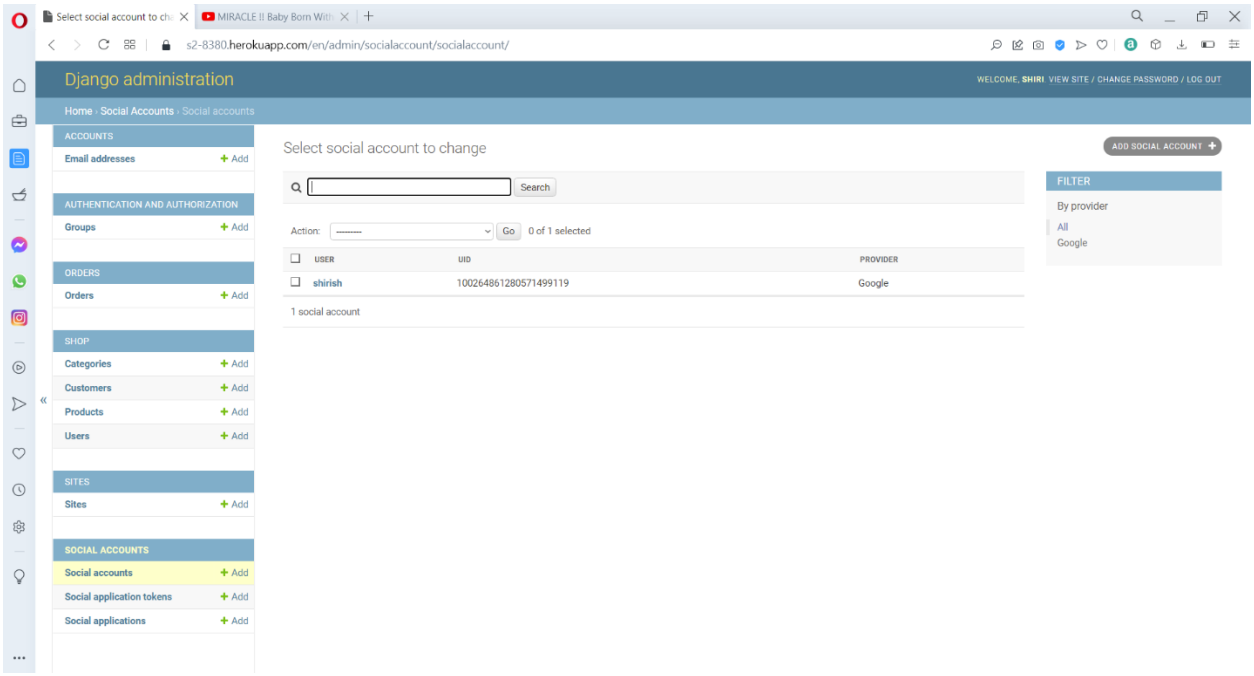
This is the change order page, where admin can edit and change details of existing orders.



This is the order delete confirmation page, when we click on delete order, we will be redirected to a web page to confirm our decision to delete the order.

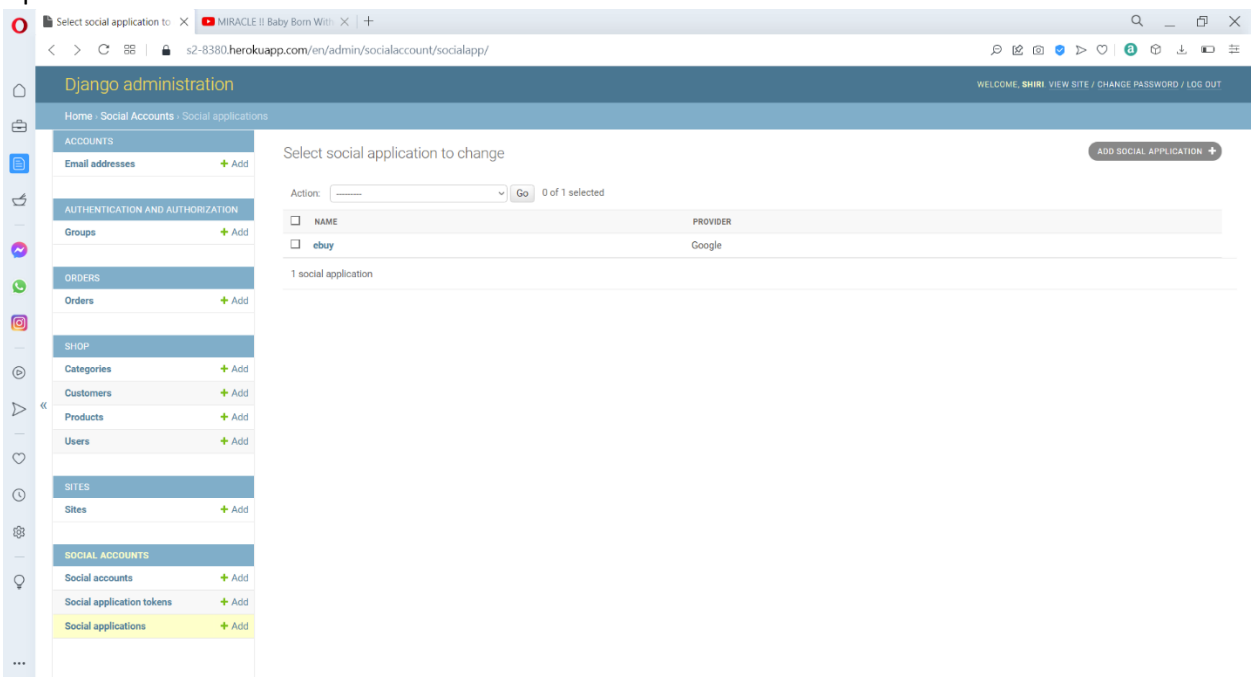


This is the site to change where it will contain the web address for application redirection.

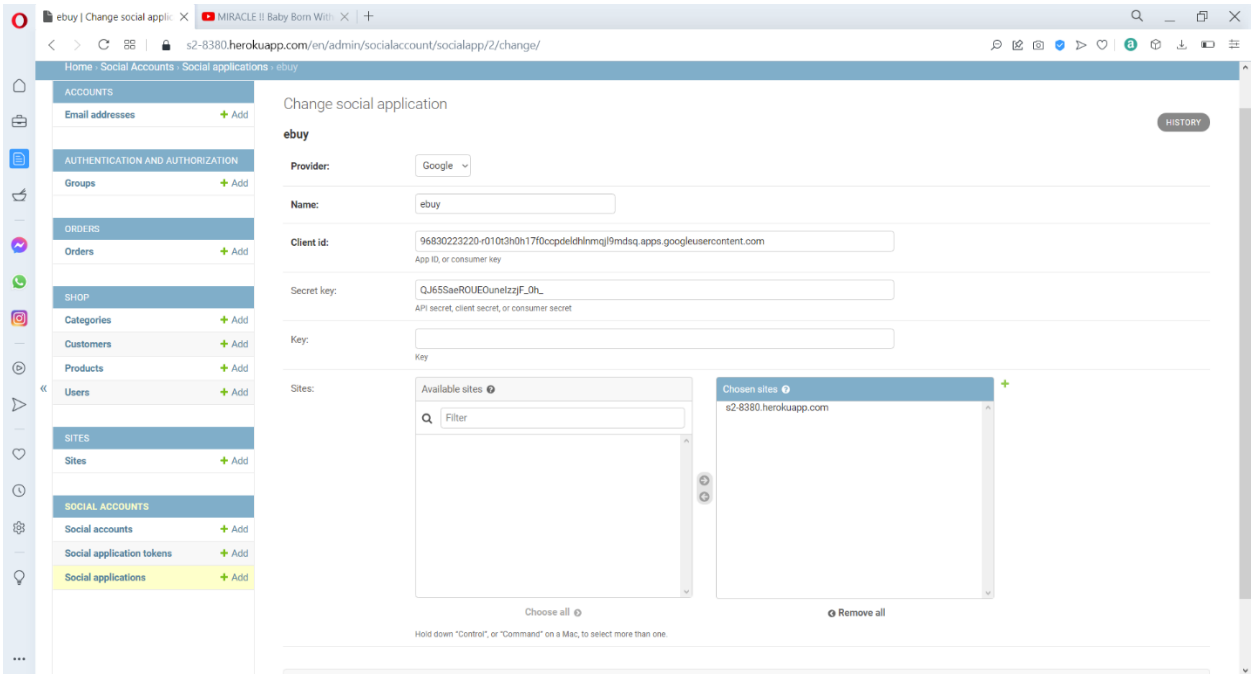


This is the social account to change page, where customers who have registered through an google account are stored.

Option

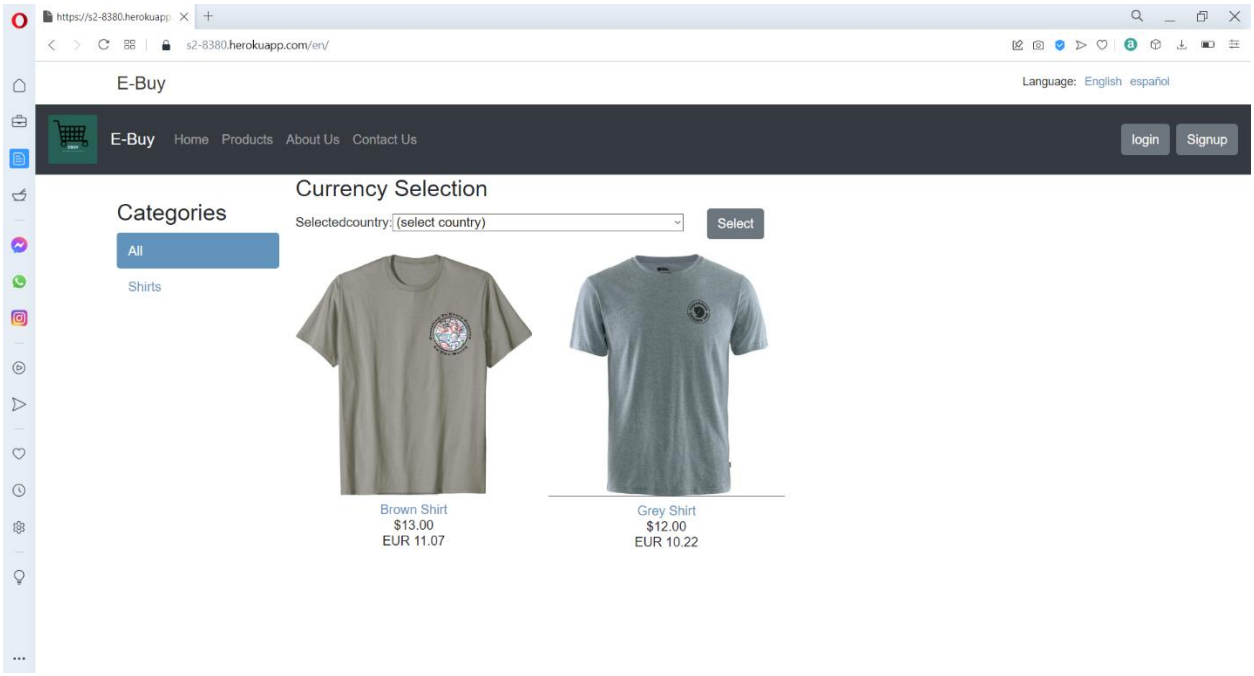


This is the social application to change page, where it will show the option for storing the client and secret key for authenticating customer using social account.

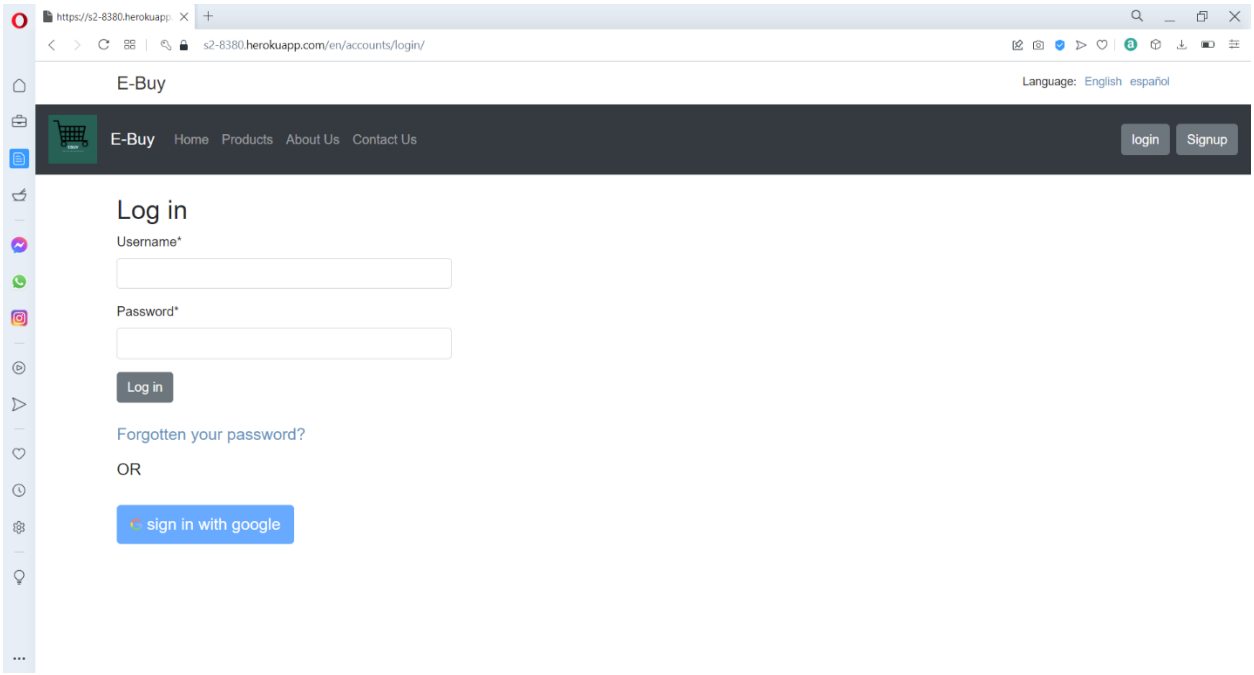


This is the social application page, where it will show the option for storing the google client and secret key for authenticating customer using social account such as google.

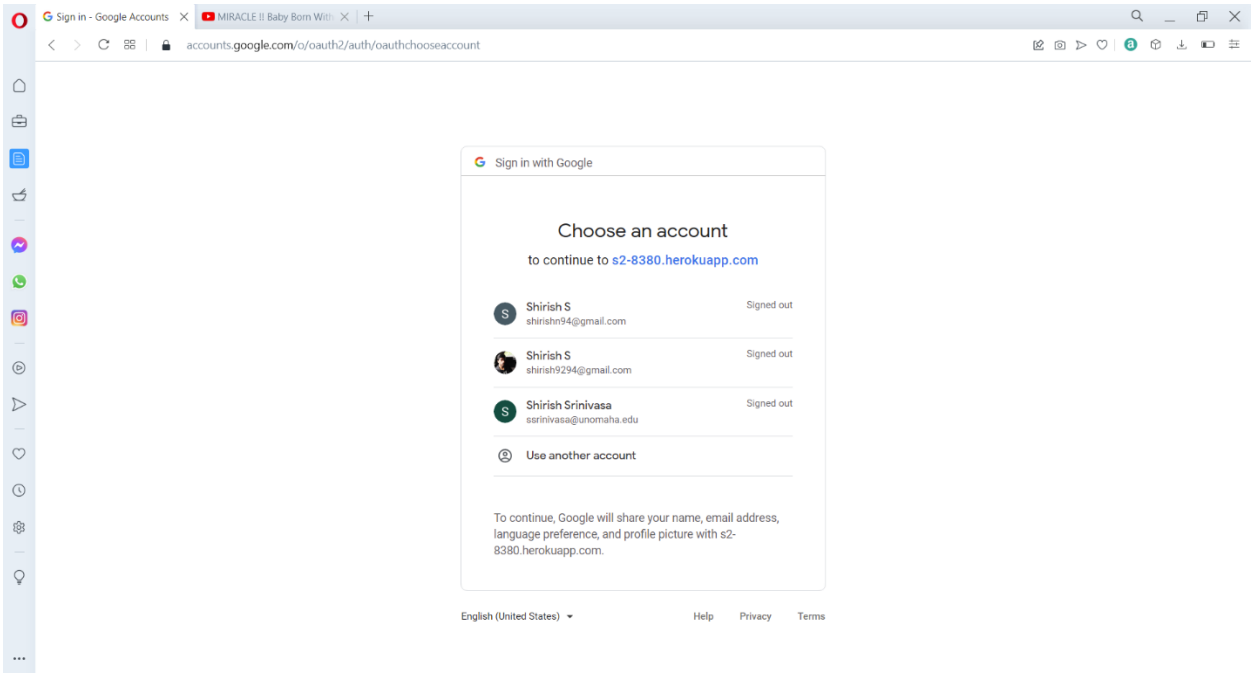
Web application



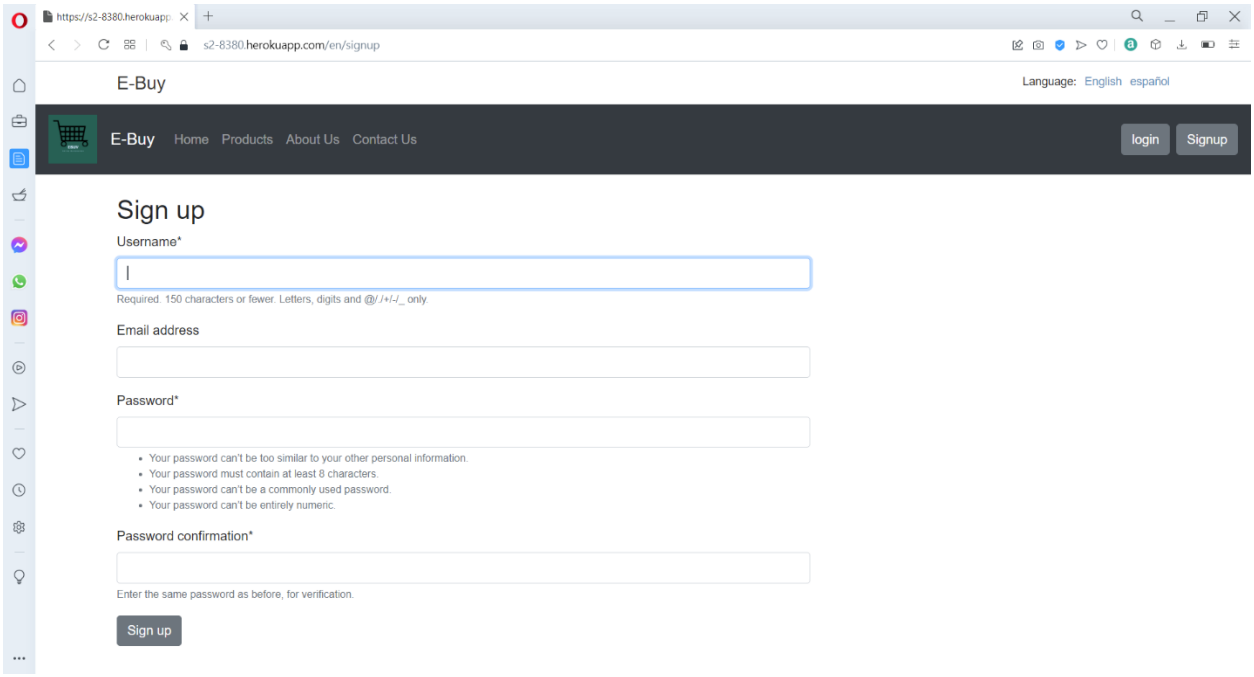
This is the home page of the ebuy application.



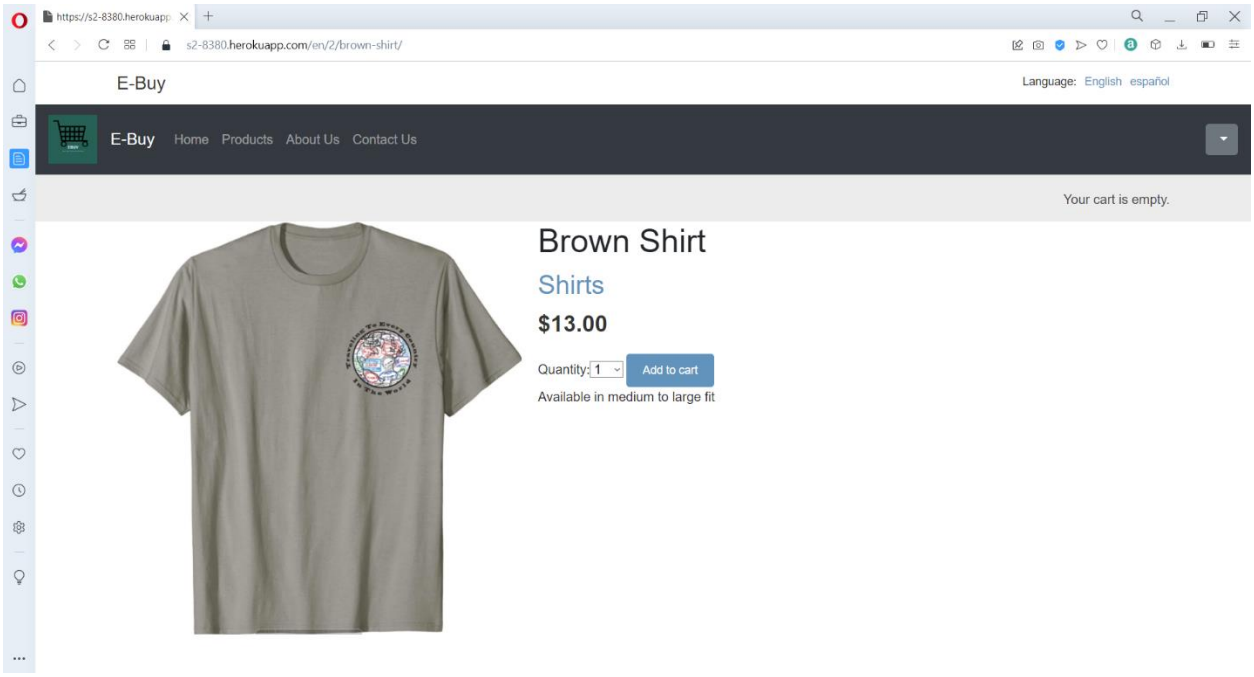
This is the login page of the e-buy application, where the customer can enter the login information or can sign in through google.



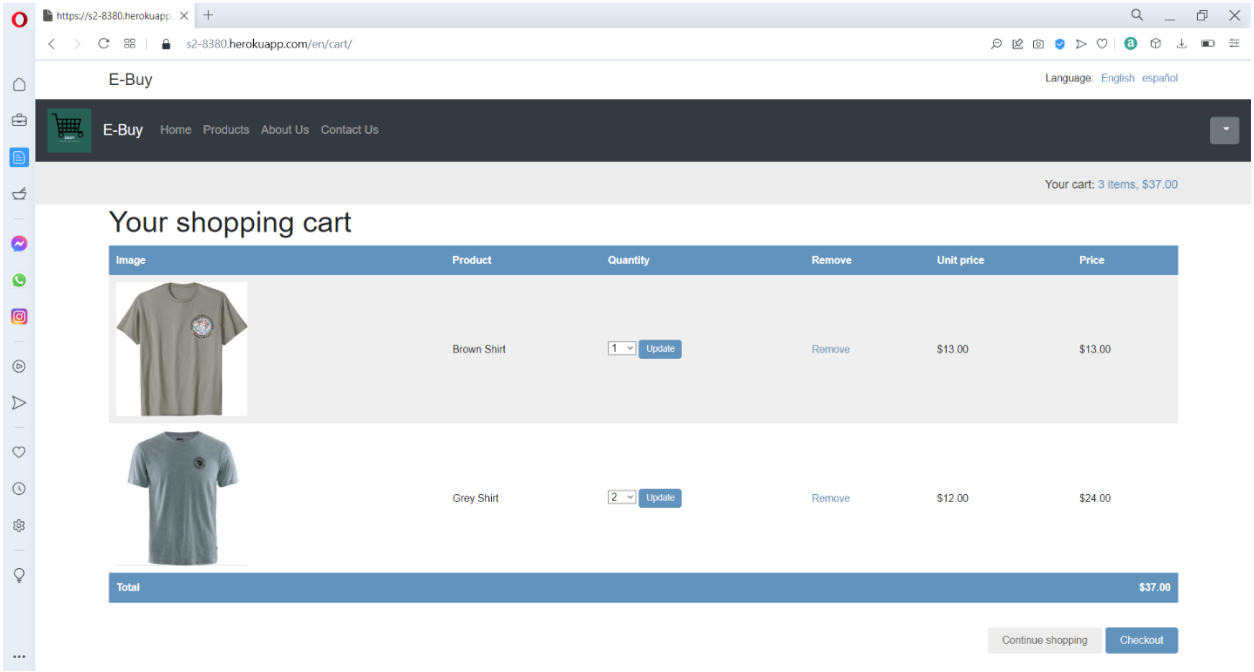
This is the google sign in screen which will be redirected for the customer, where the customer will choose an account to register and provide access to the web application.



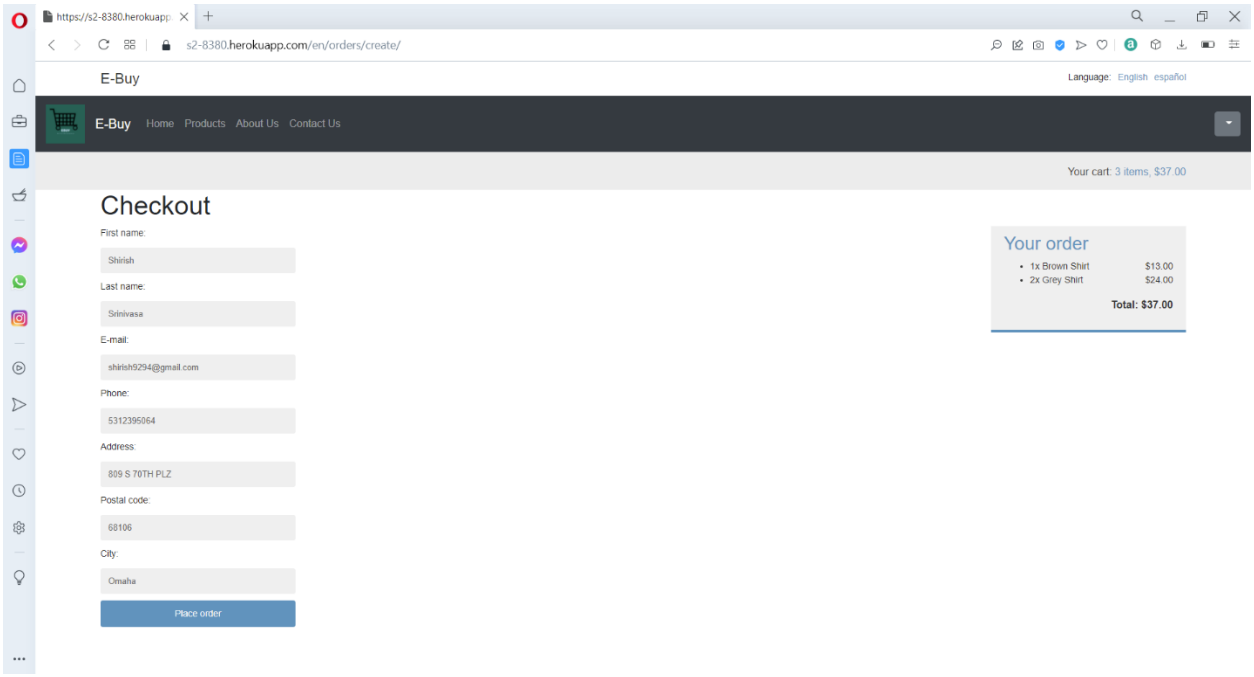
This is the sign up page where a customer can enter his details and register for the web application.



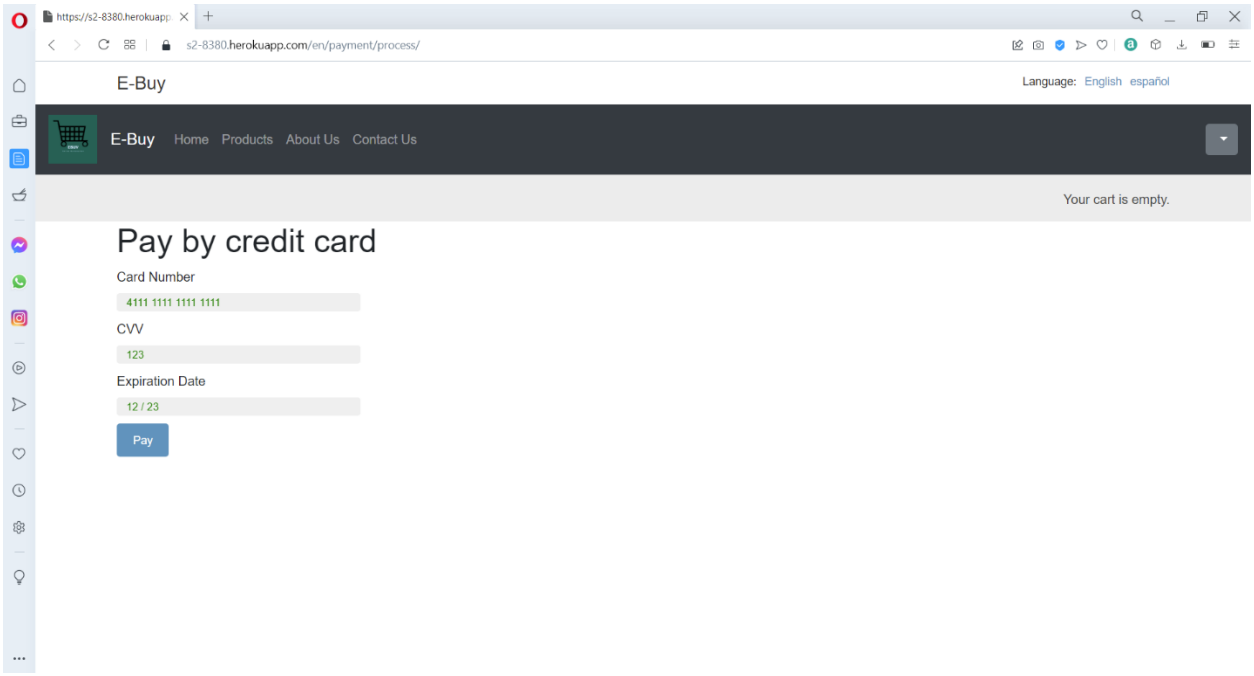
When the user selects a product, he will be redirected to the product page where he will get the information like the product name, price and options like quantity and add to cart.



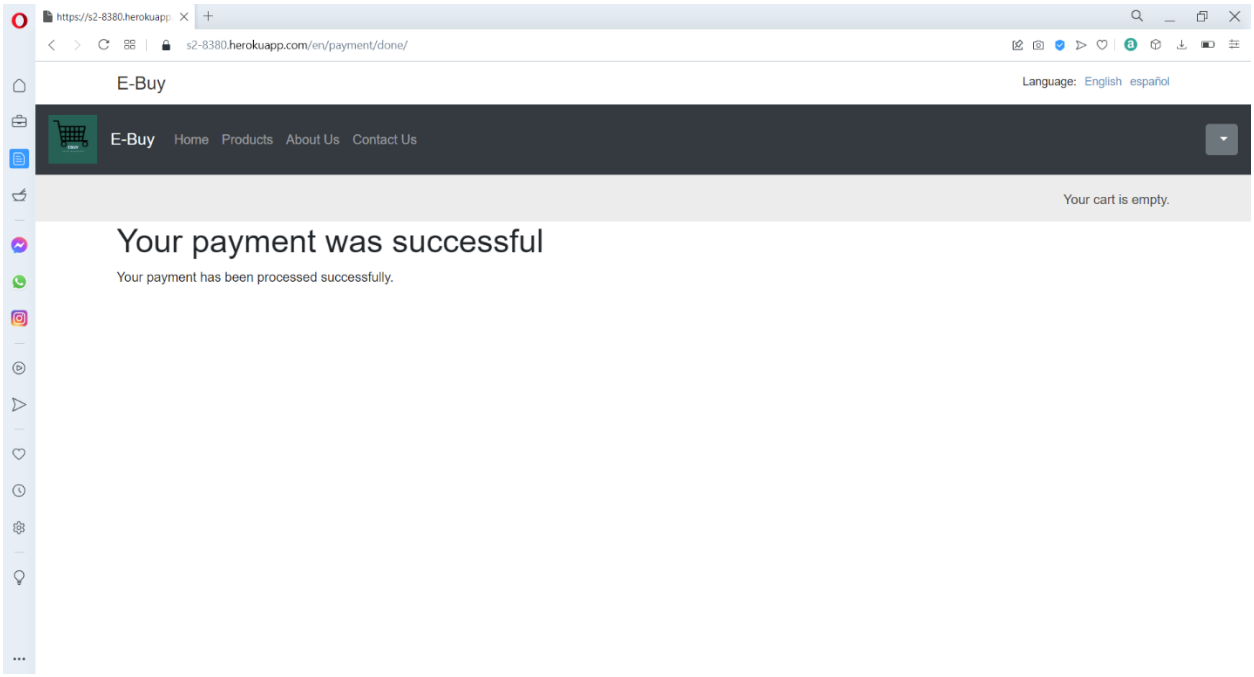
In the cart the customer can view the products that he added, he can remove the products that are not needed, and can modify the cart contents by changing the product quantity. The customer can then click on the checkout option to perform the order confirmation or can continue shopping.



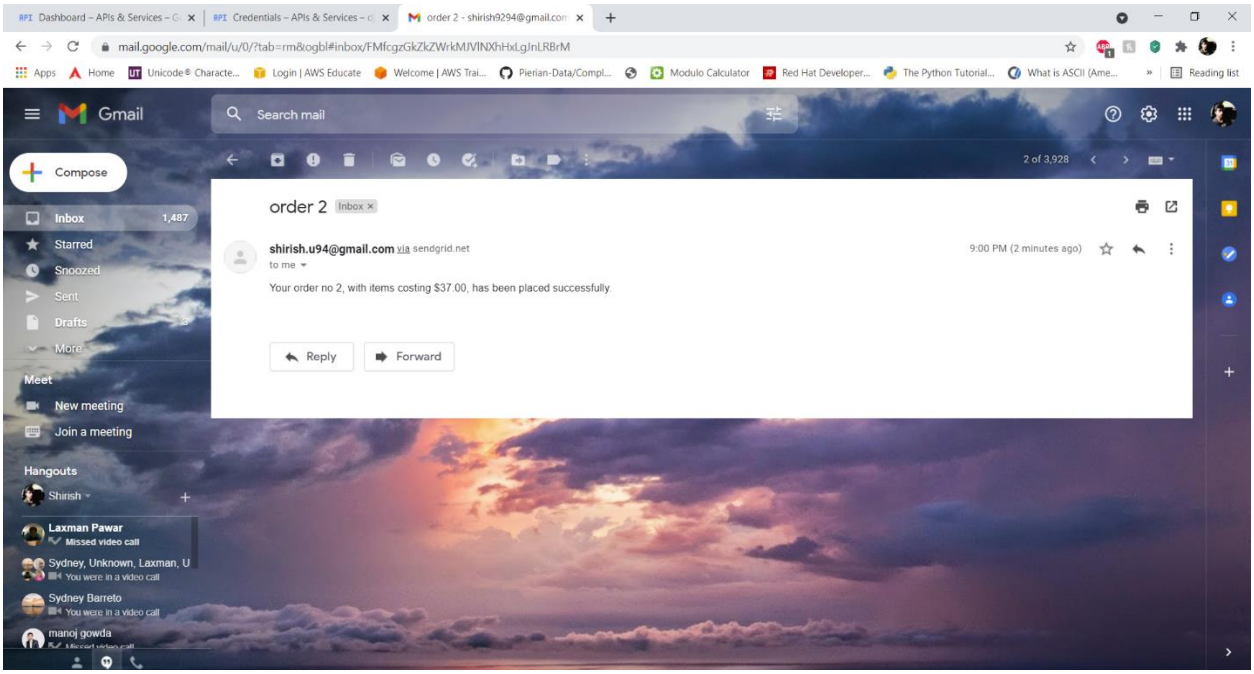
In the checkout page the customer can views the order details on the right side, which shows the product and the total amount. The customer needs to fill the shipping details to specify where to ship the product. The user needs to fill the details like the first name, last name, email, Address, City, State, Country, and postal code. Then he needs to click on the complete order option to be redirected to the payment page.



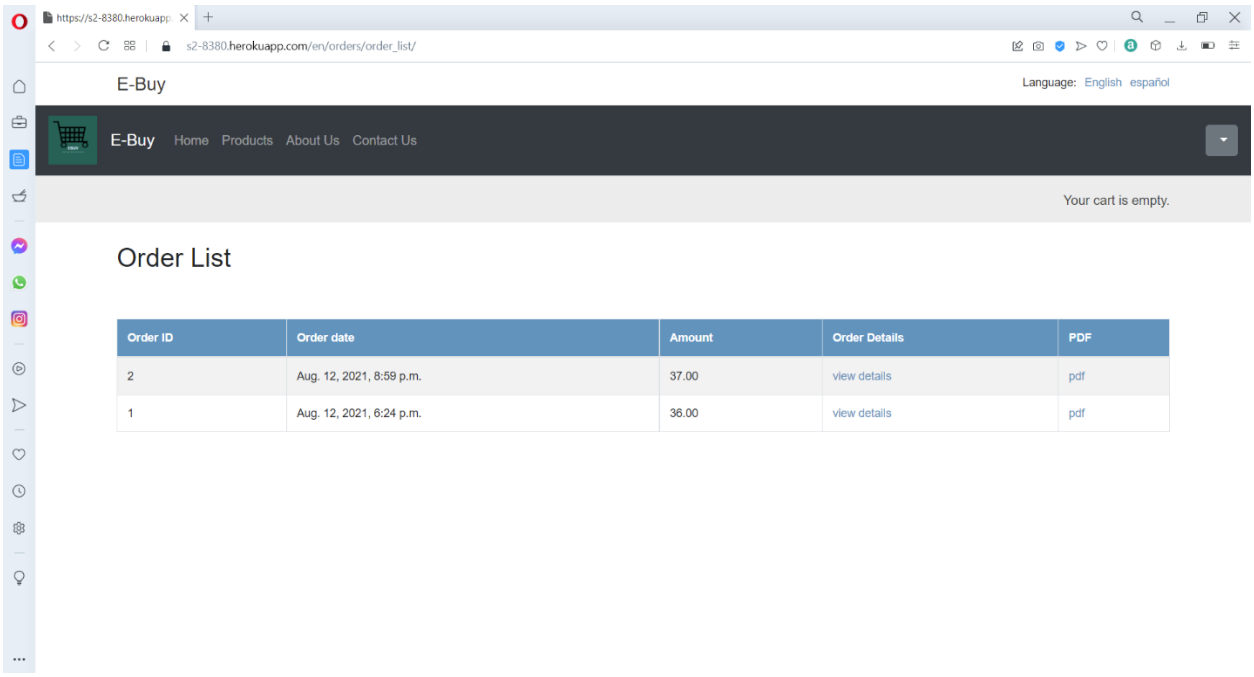
The customer needs to enter the payment details like his credit card number, CVV, and the card expiration date and click on pay. If the details are valid the user will get an order confirmation message.



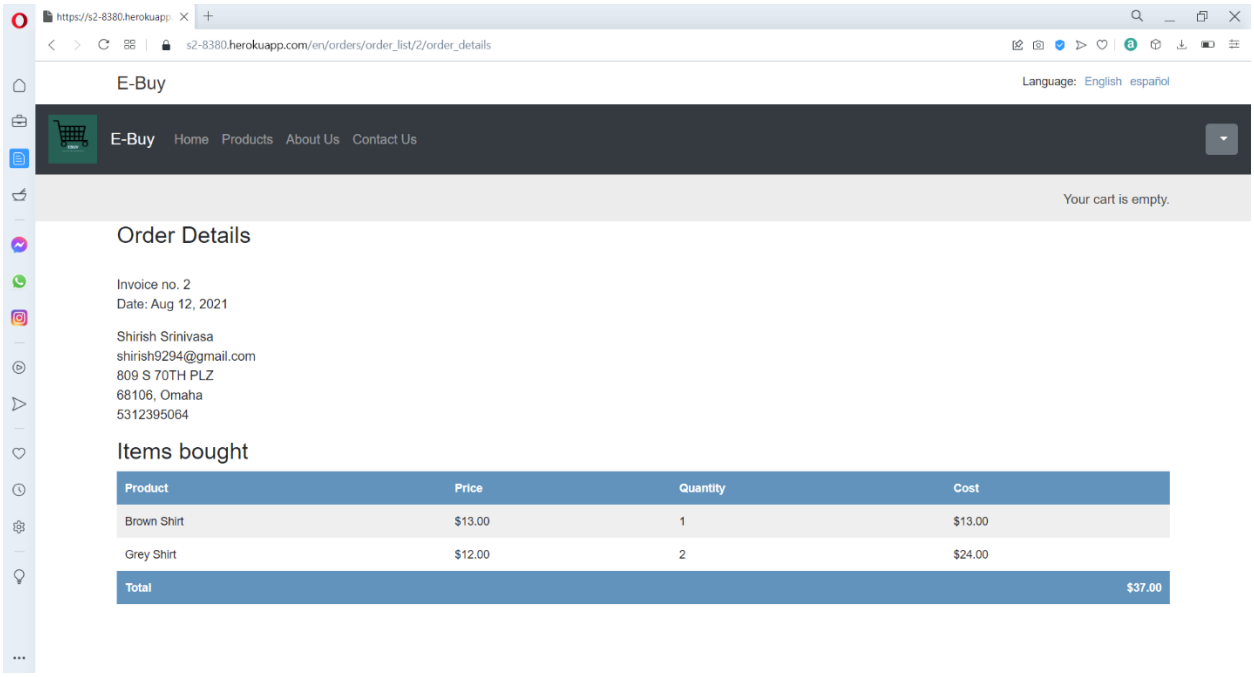
This page shows that the order has been successfully placed.



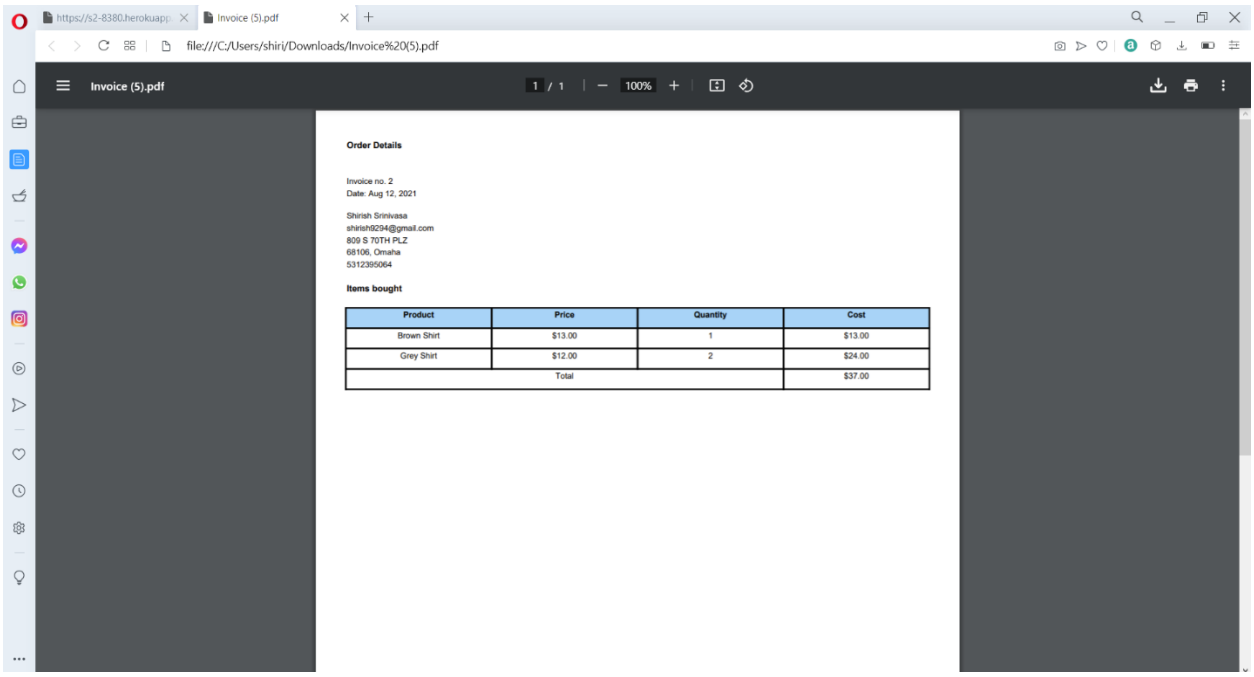
This is the email confirmation that will be received by the customer, once an order is successfully placed.



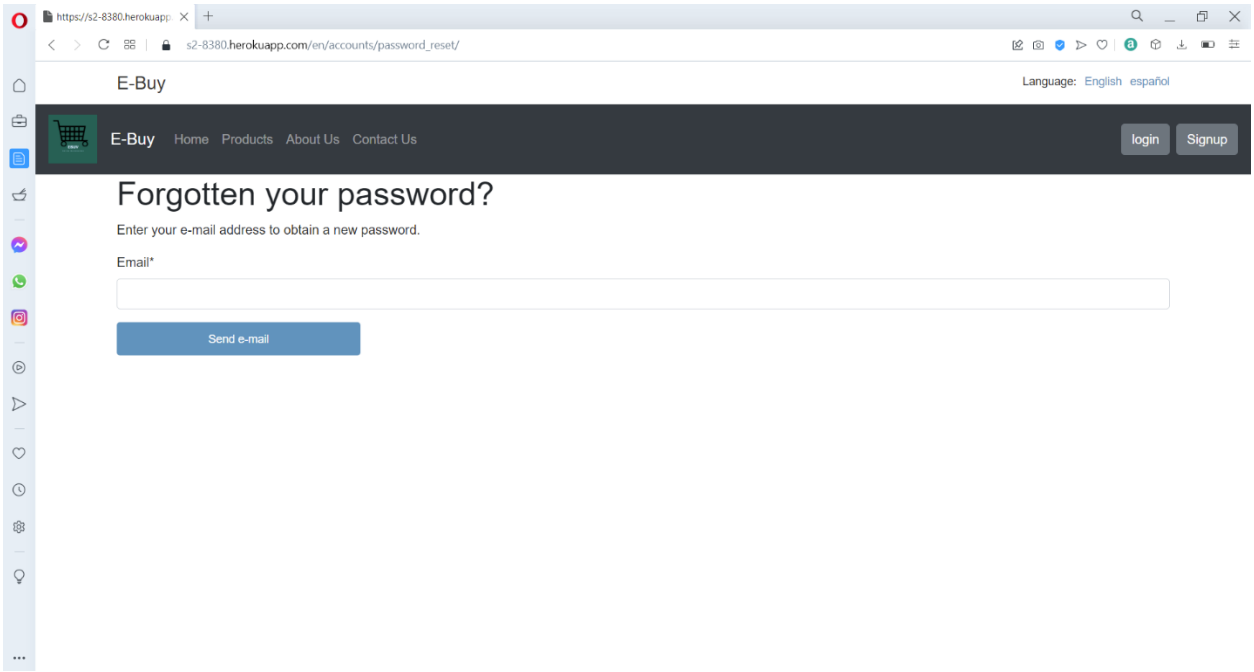
This is the order list page, where it will show the information of all the orders placed by a particular customer, the customer can also view more details on their order or can download an invoice pdf copy.



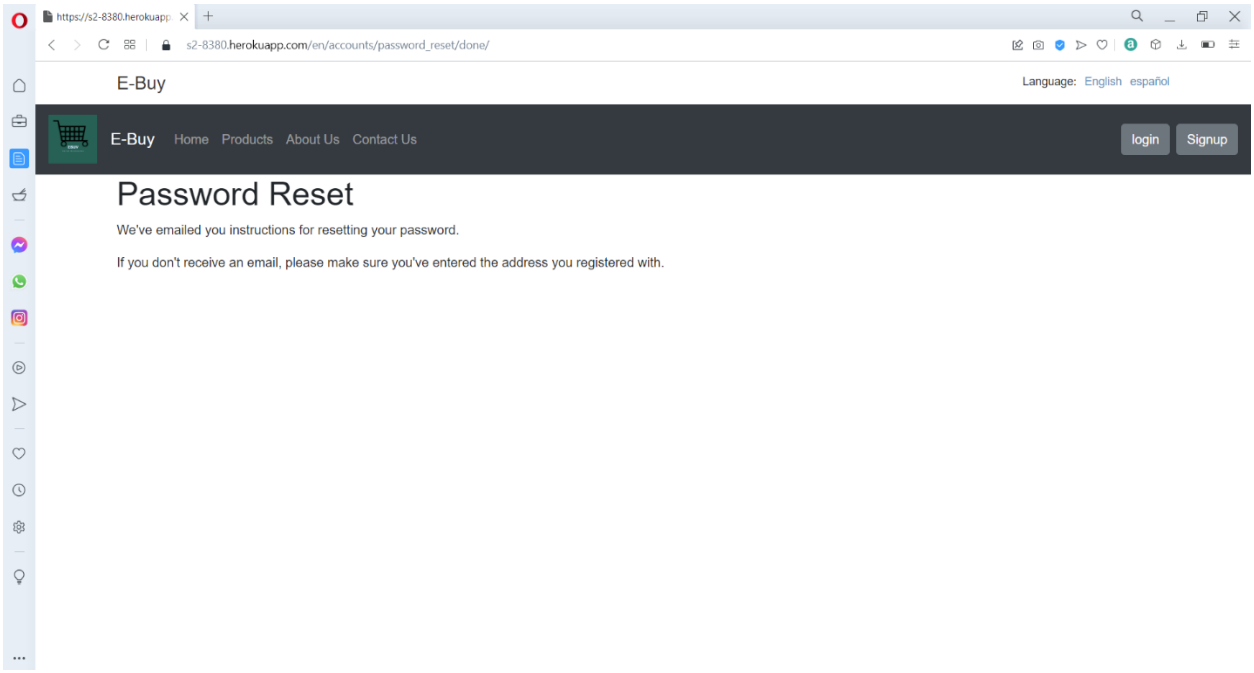
This is the order details page, where it will show the shipped information and the product details and price to the customer.



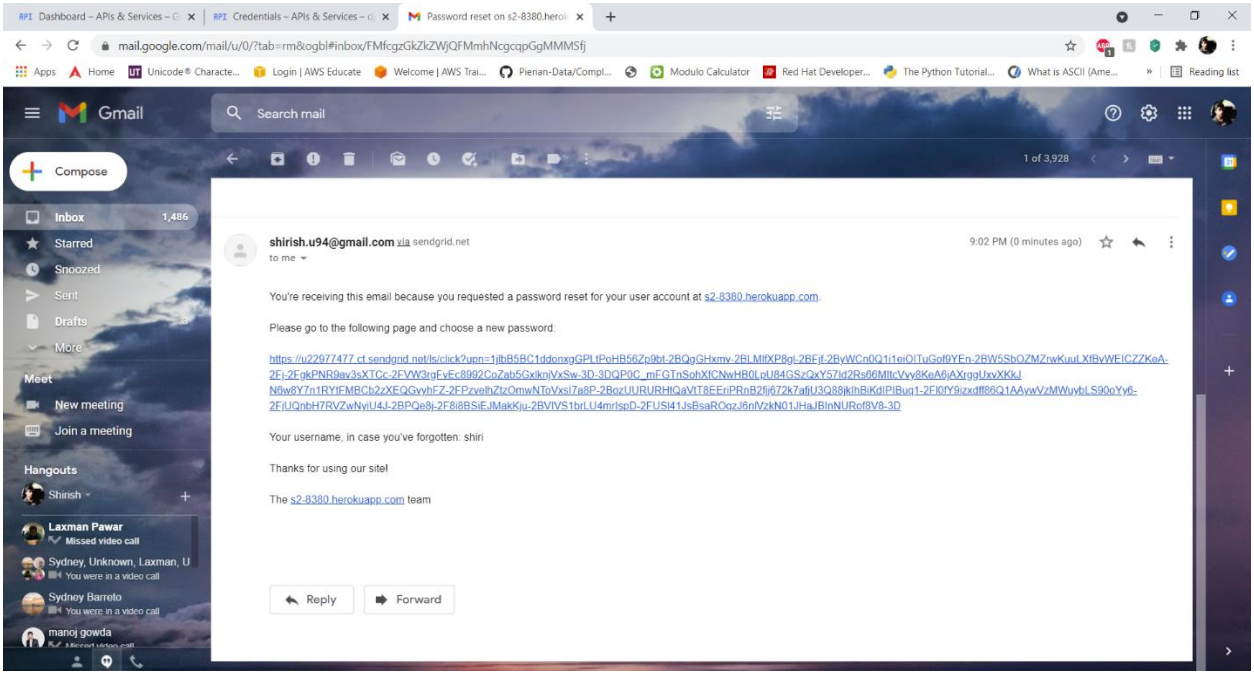
This is the pdf invoice of the order that can be generate by the customer for his orders.



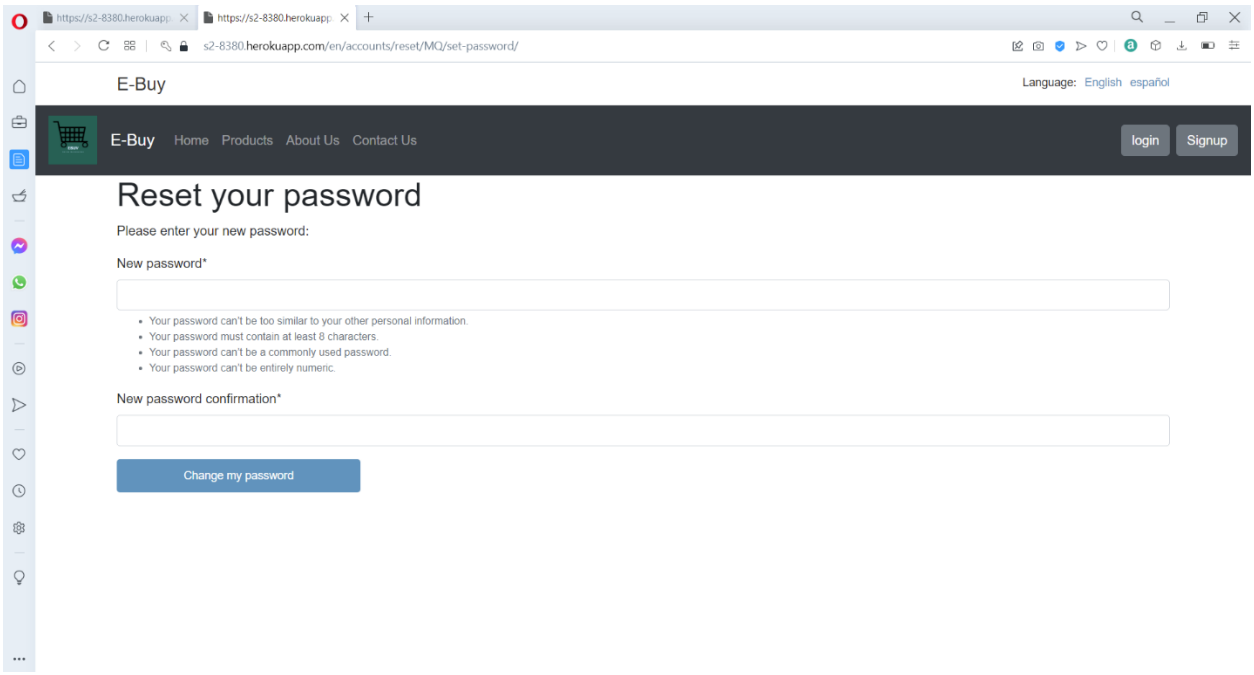
This is the forgotten your password page, where you can enter your registered email address to send the password reset link, the link can only be sent to account that have manually signed up and will not send to accounts that have signed up using social account.



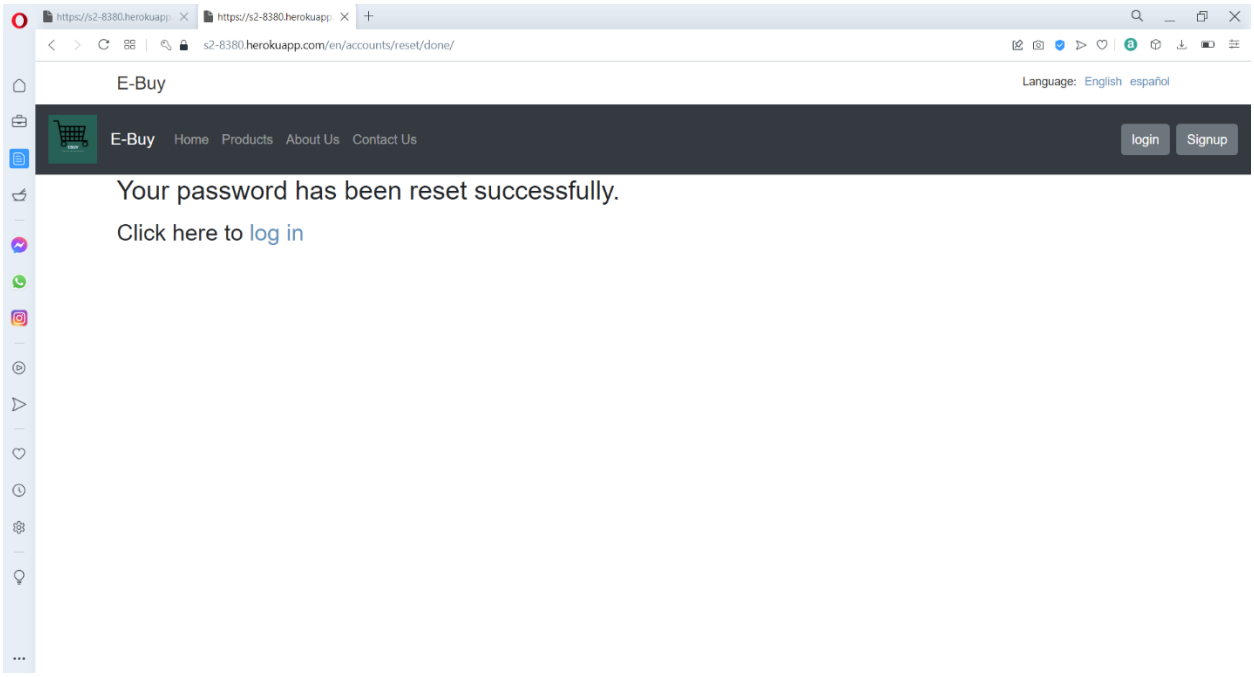
This is the password reset page, where it shows us the message that the email has been sent successfully to the registered mail address.



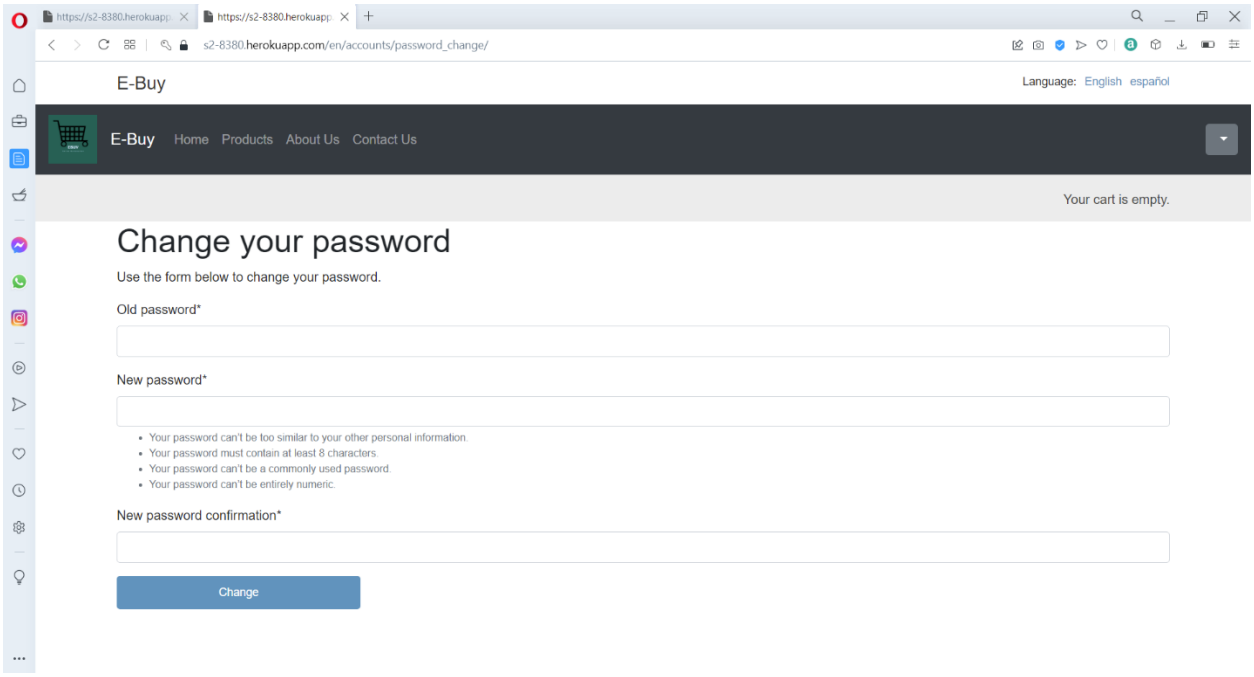
This is how the reset link will be displayed in the email, which we can use to reset our password.



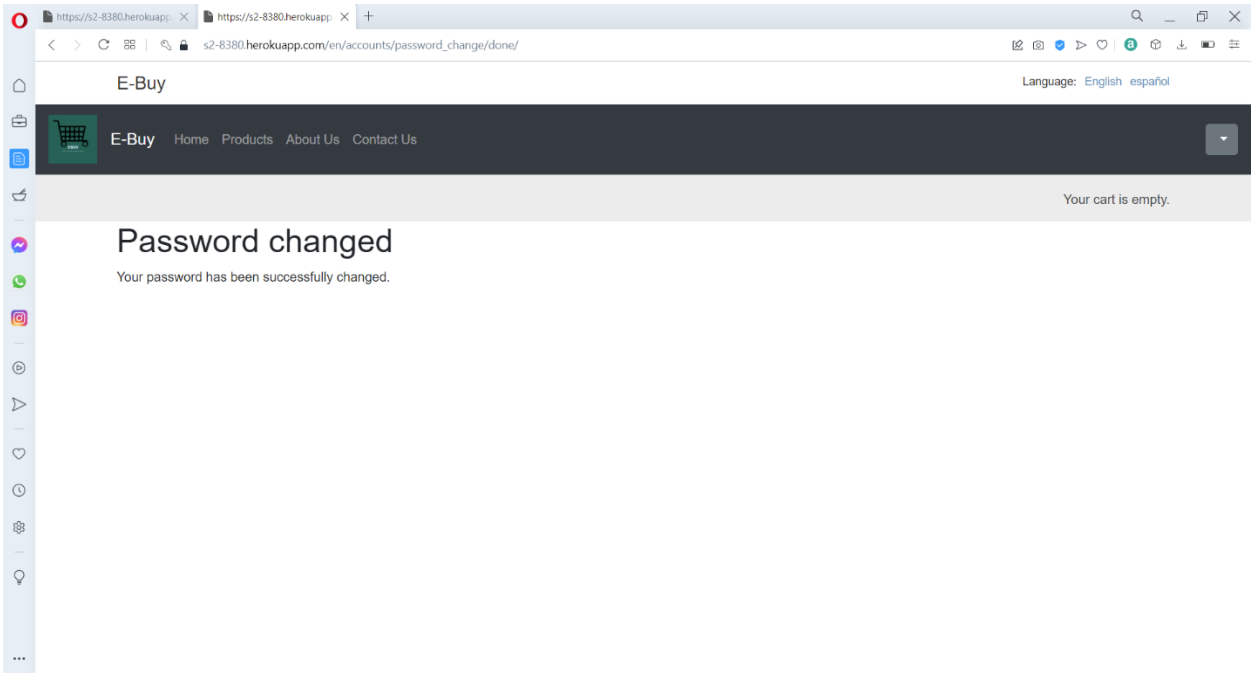
This is the reset your password page, which is accessed after using the reset link by the email sent to the customer.



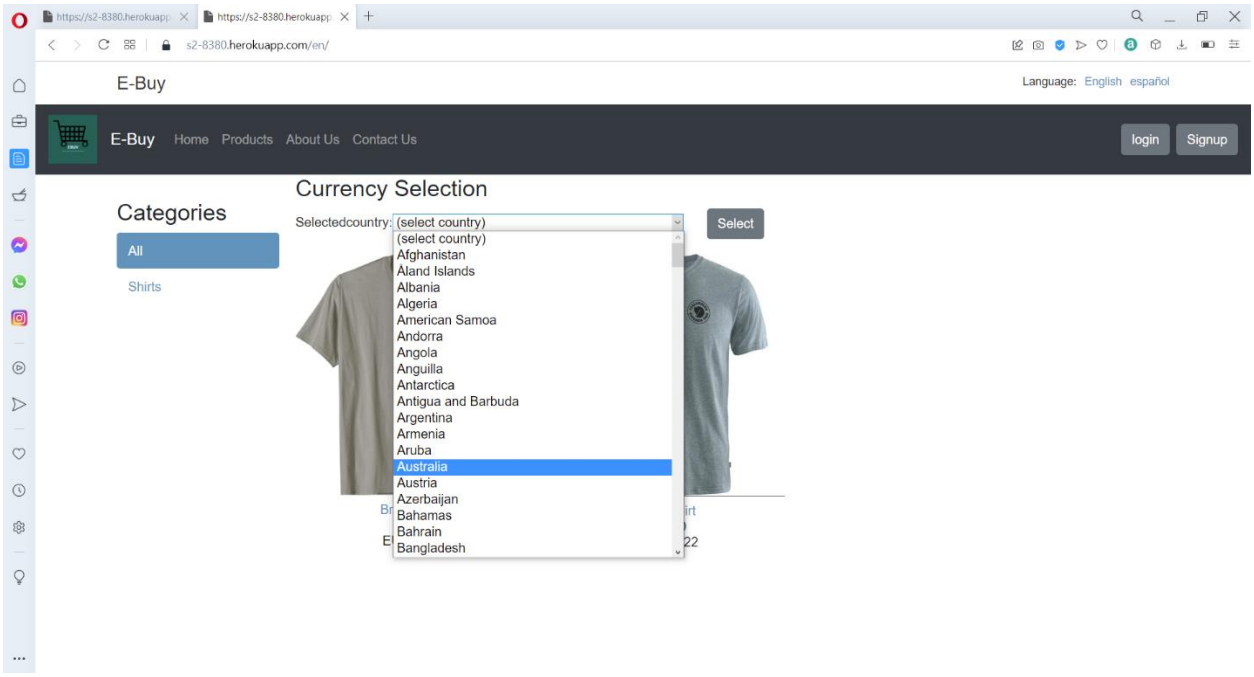
After the password has been reset, the customer is redirected to the confirmation page, where is shows us that the password has been reset successfully.



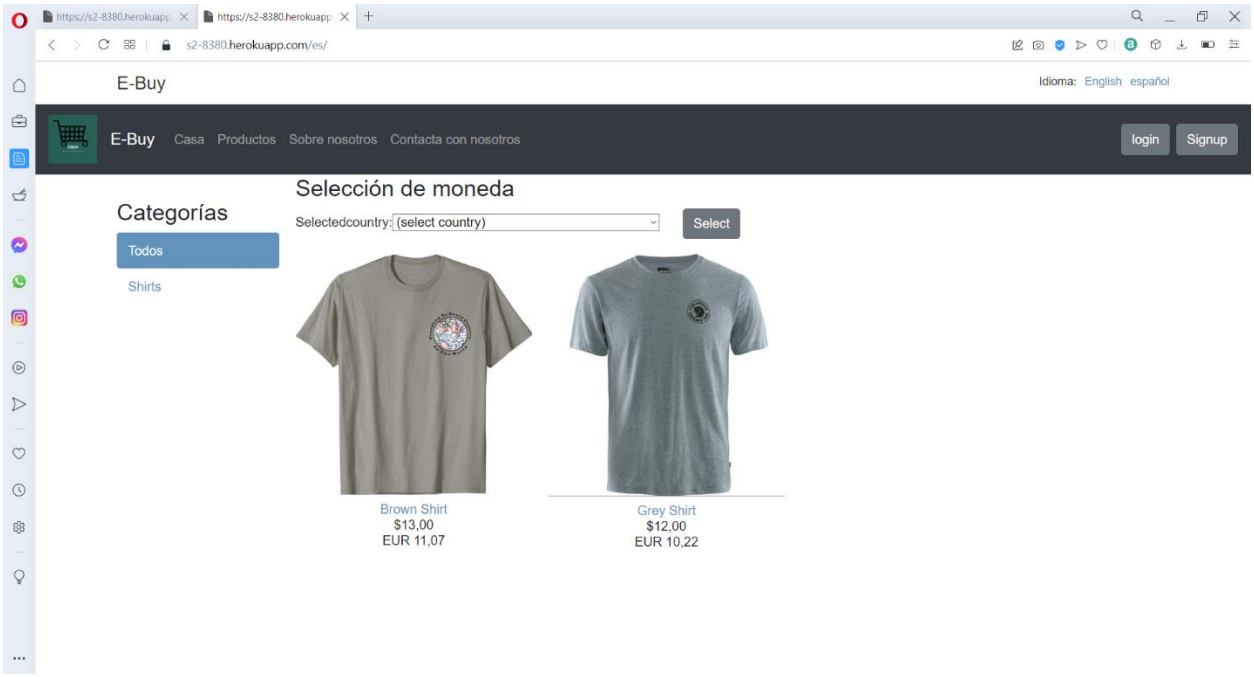
This is the change your password page, where after the customer logs in, he can reset his existing password, by using his old password for confirmation.



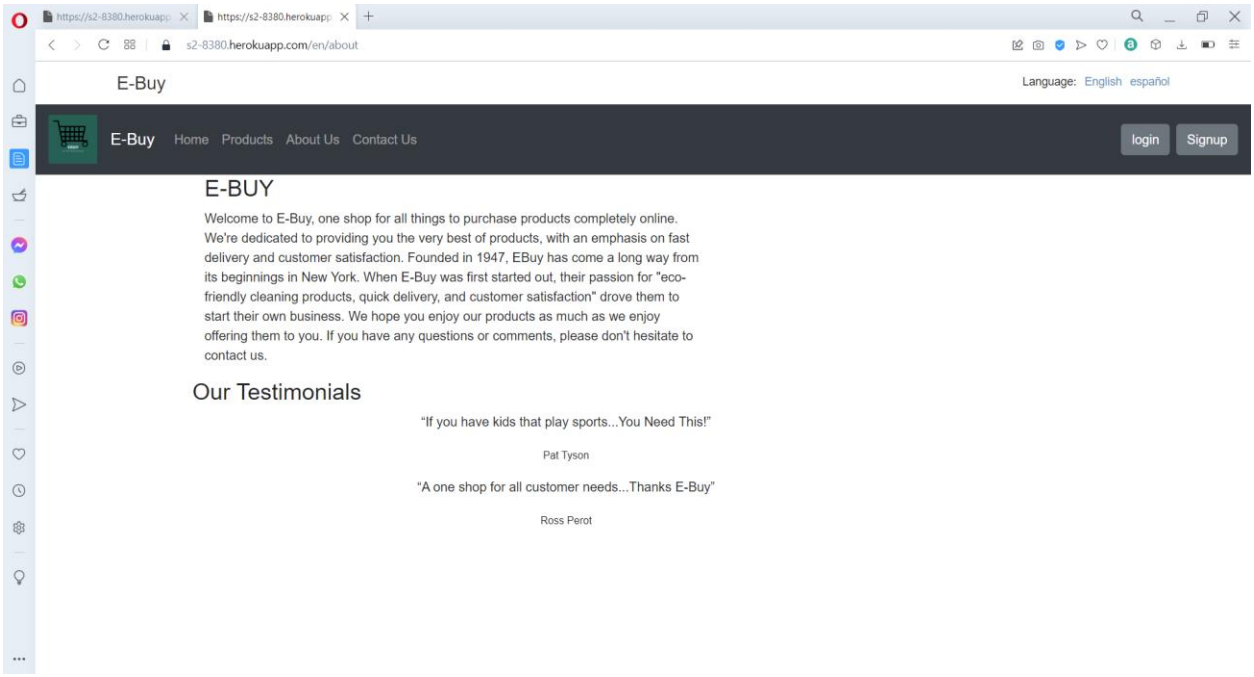
This is the password changes confirmation page, where it shows that the password has been successfully changed by the customer.



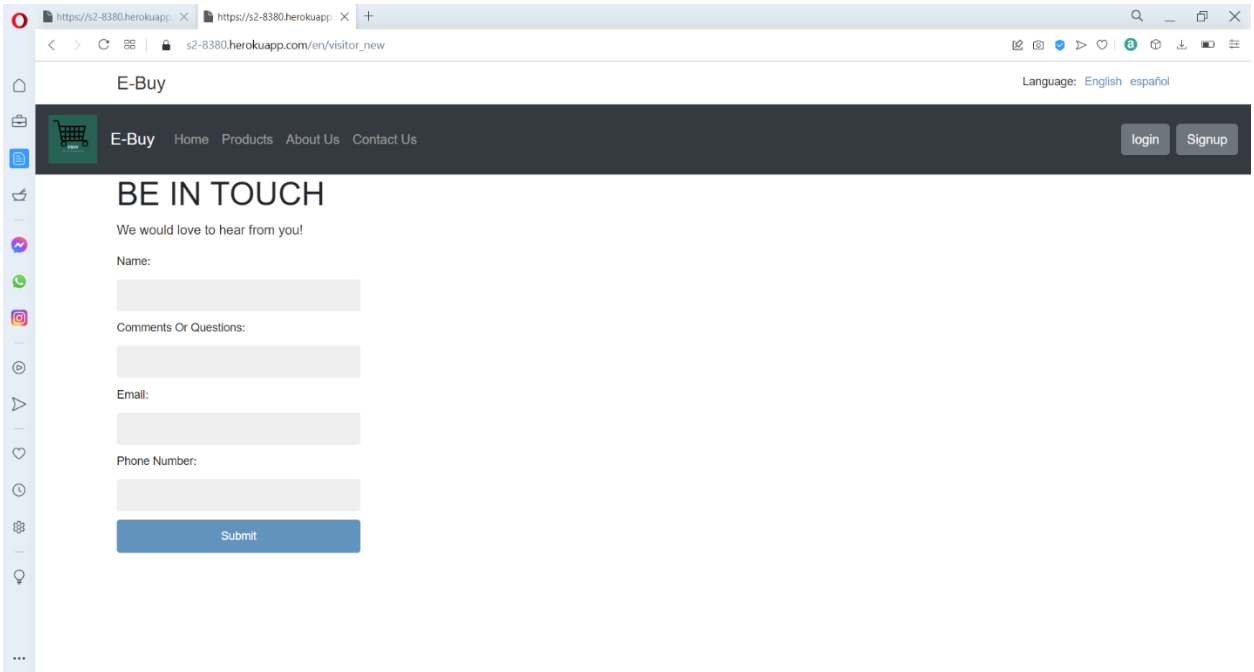
This is currency conversion option the e-buy web app, where the customer can select and country name to change US dollars to the selected country currency.



This the language conversion option available in the e-buy web app, where on the top right-hand side, we get the option to change the language from English to Spanish and Spanish to English.



This is the About us page, of the e-buy application, where it will give a brief description about the web app.

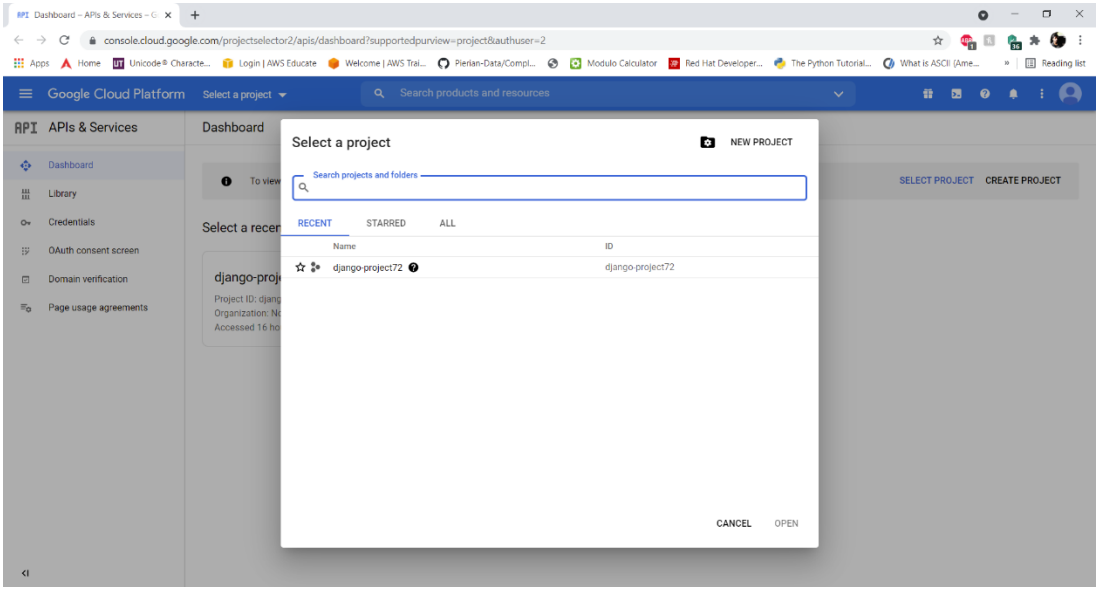


This is the contact us form where the customer can use this to contact the website admin and post a question.

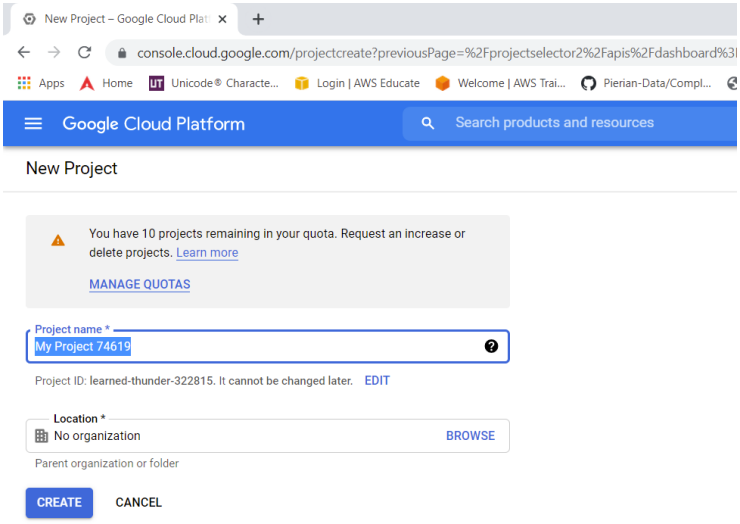
API

Google API

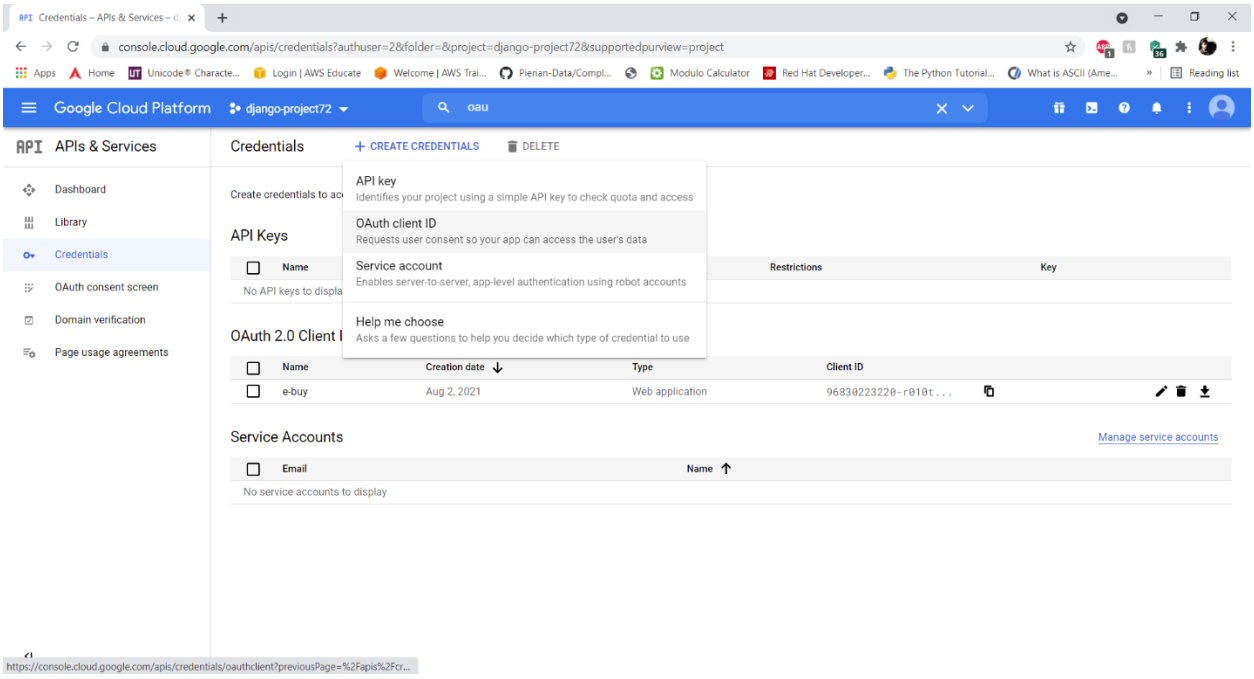
To utilize the google API, we are going to access the google develop console and accept their policy and agreement to utilize the services.



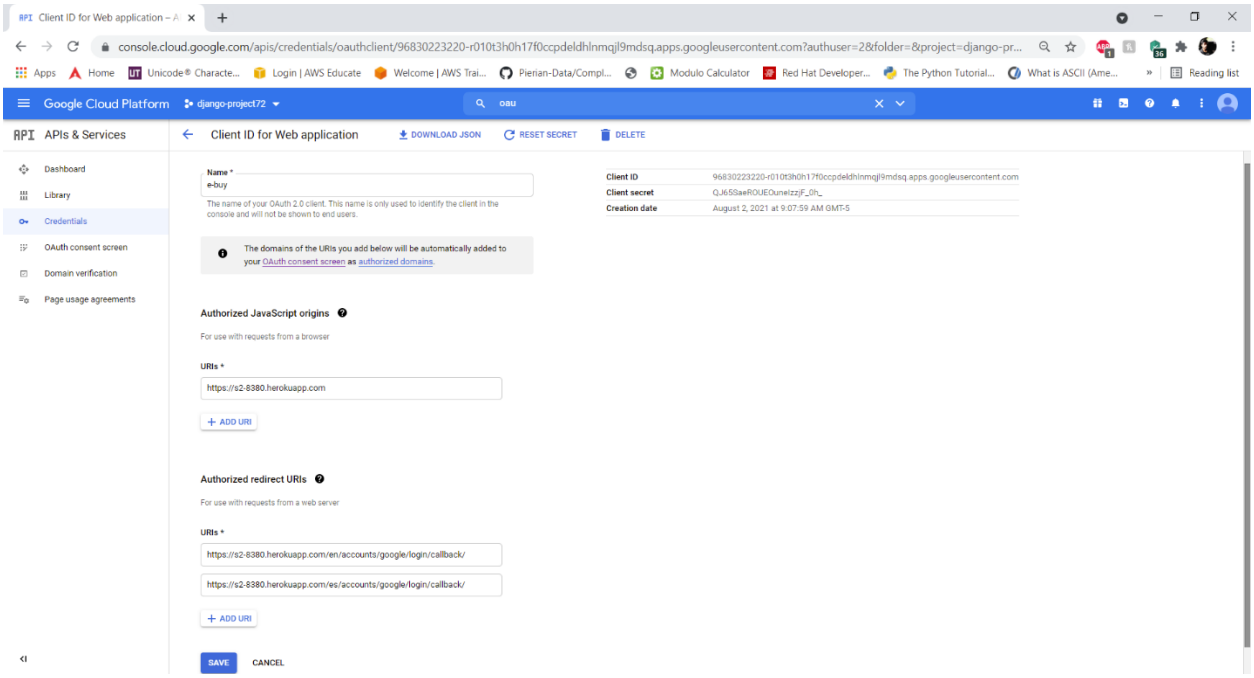
Then we need to click on select a project and then create a new project.



Then we need to enter a project name and location, and then click on create to create the project.



Now we need to go to credentials -> create credentials -> Oauth client ID, for creating an Oauth authentication for the e-buy web app.



Now we need to enter a name for the Oauth client ID and enter the URL and the URL redirection which will be utilized for the Oauth authentication, then click on save.

Now we need to install the package:

Pip install django-allauth

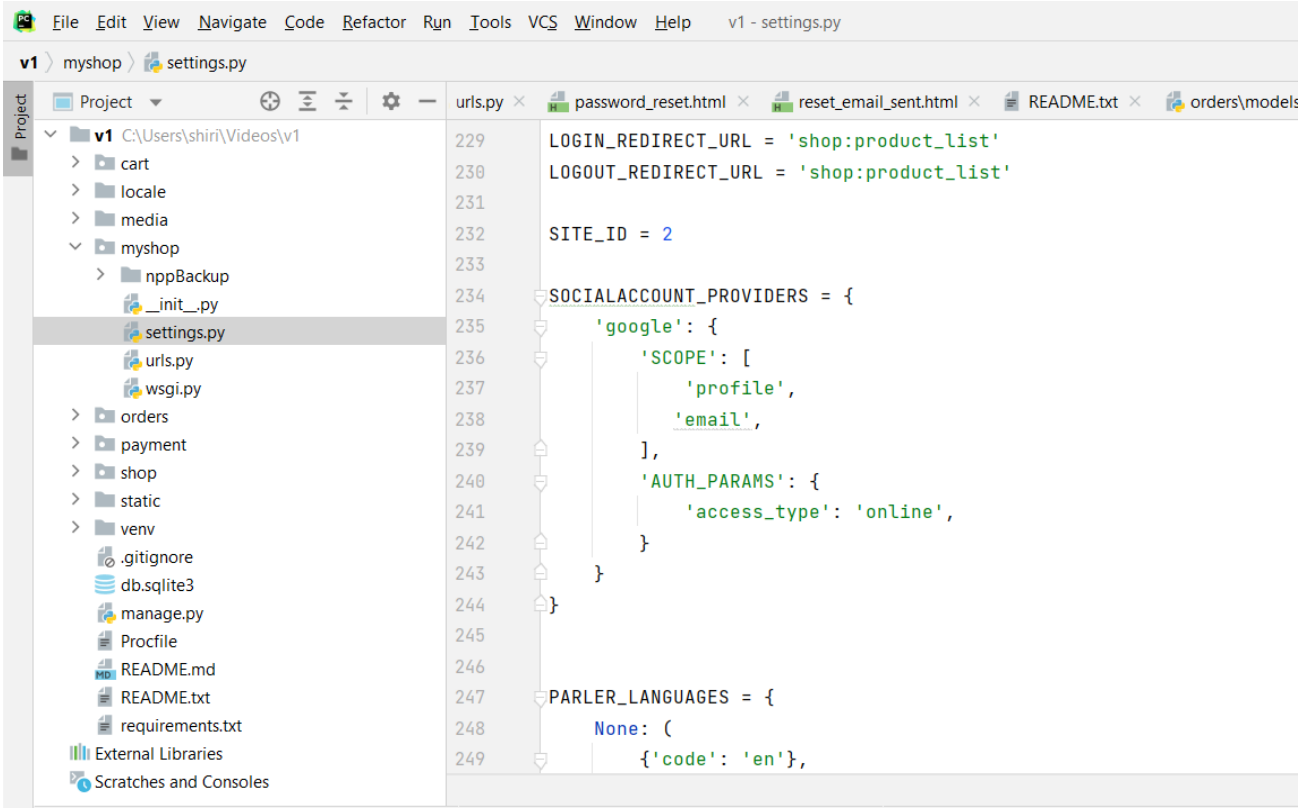
And add these lines in the installed apps

'allauth',

'allauth.account',

'allauth.socialaccount',

'allauth.socialaccount.providers.google',



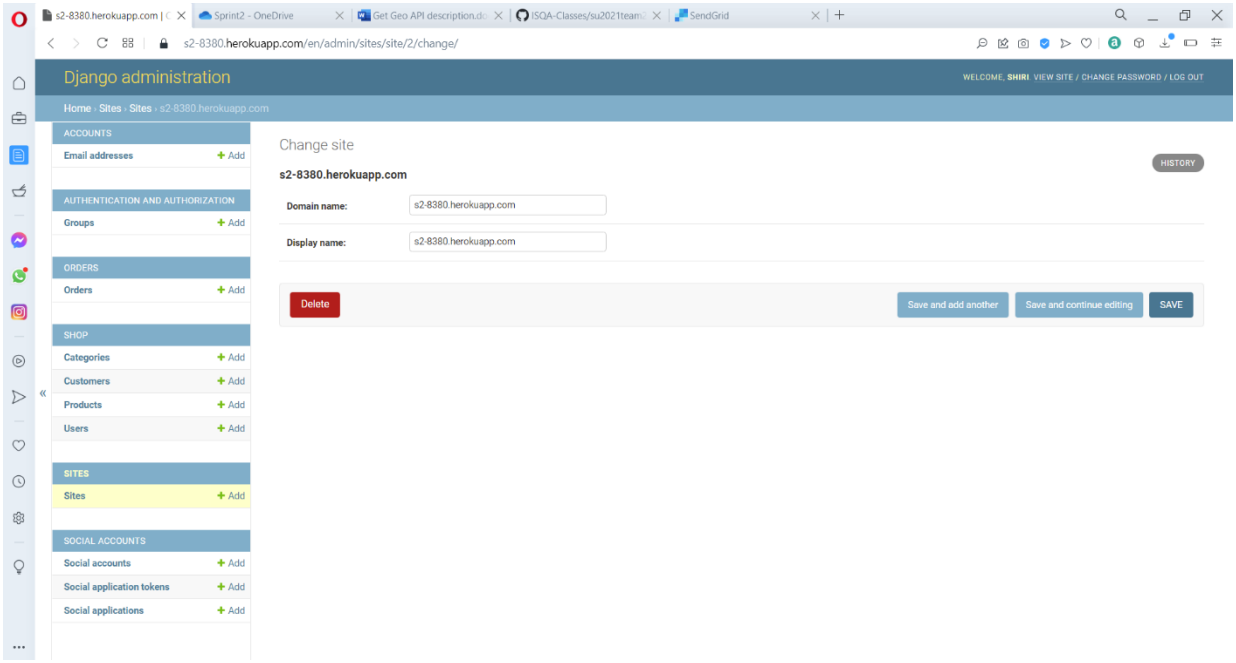
Now we need to add these lines of code to allow the e-buy web app to gain access to the profile and email information through his google account to create and store the user information on the web site.

SITE_ID = 2

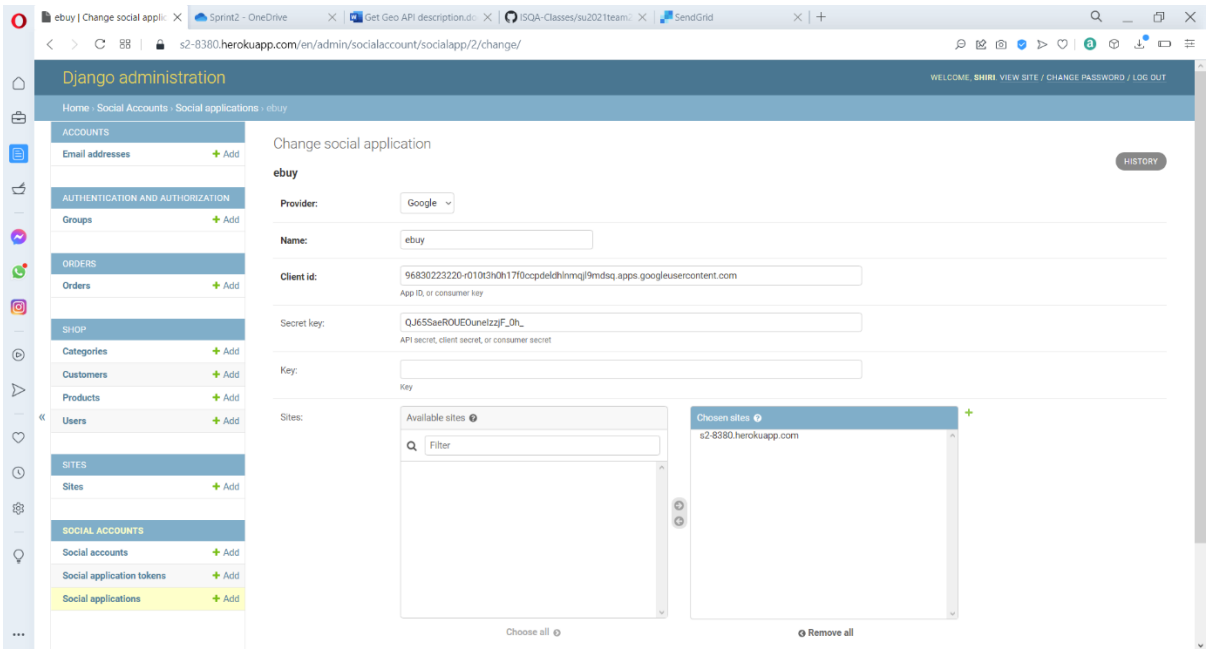
SOCIALACCOUNT_PROVIDERS = {

'google': {

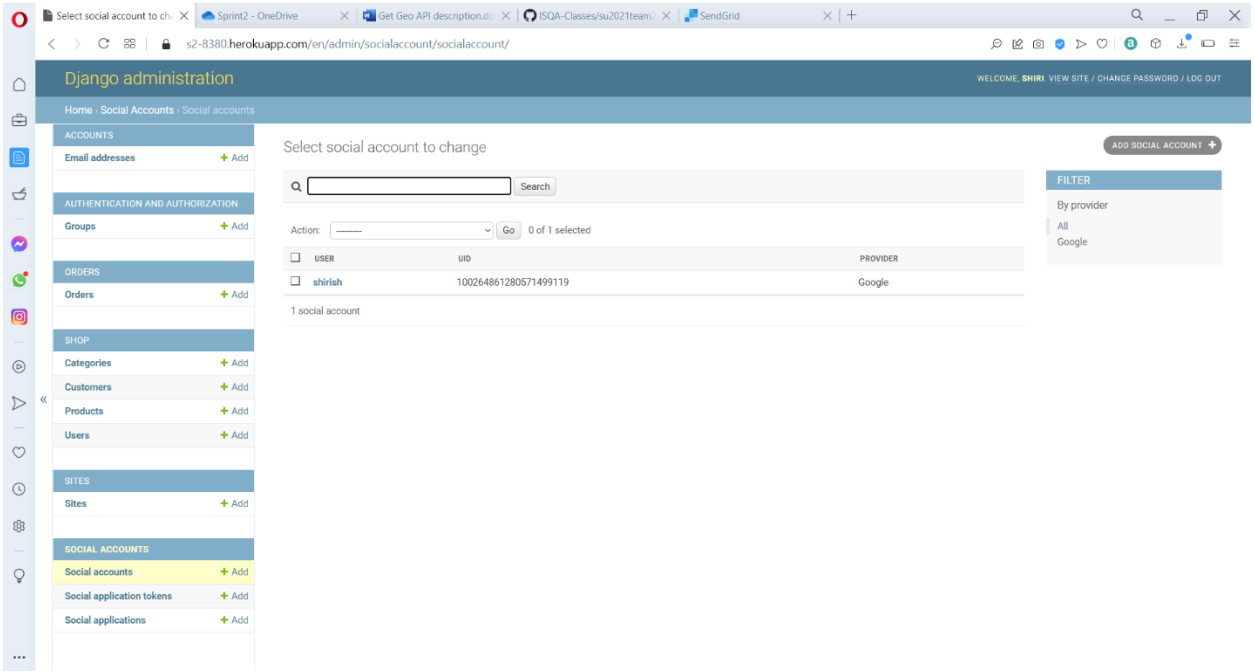
```
        'SCOPE': [
            'profile',
            'email',
        ],
        'AUTH_PARAMS': {
            'access_type': 'online',
        }
    }
}
```



Now when we deploy the code, we need to access the admin panel of the deployed code on Heroku. The under site new need to add the Heroku domain name and the display name for URLs access from the web site.



Now we need to social application and add a new social application, where we are going to enter the name, select the providers and enter the Client ID and Secret key that was generate when we created our Oauth Client ID, and in sites, select the domain name to the chose site and then click on save.



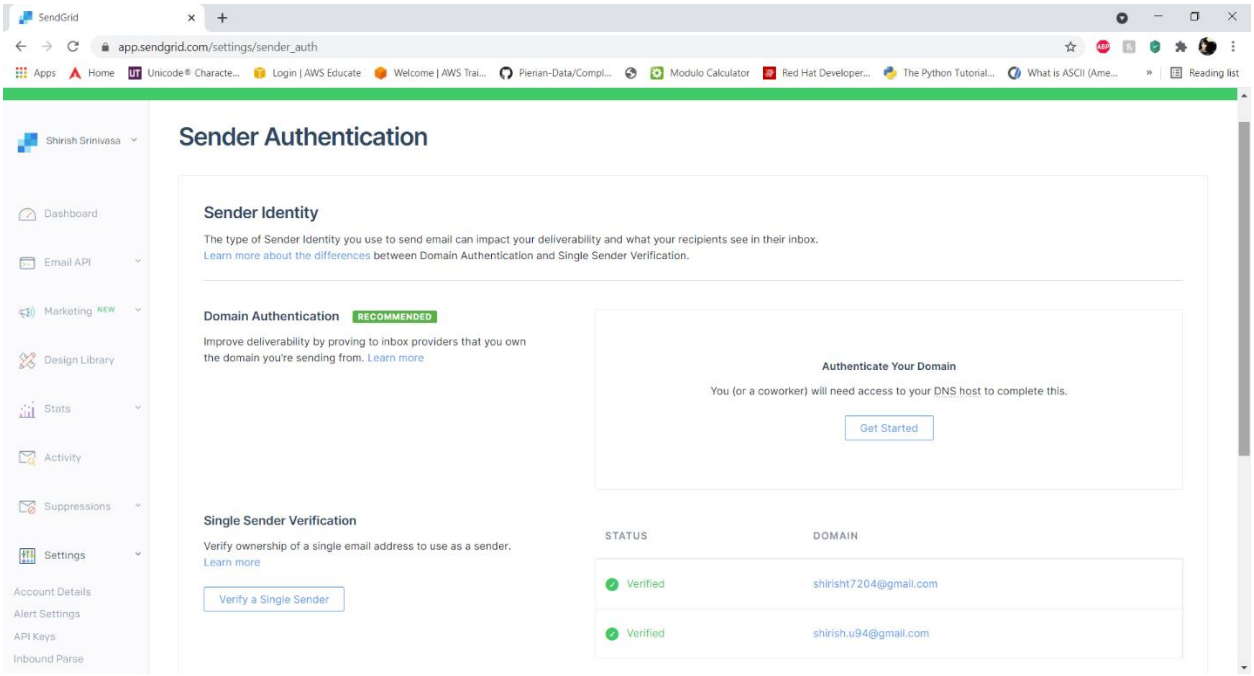
Now when ever a user signs up using a social account, his details are stored in the social accounts section in the admin panel with a UID and that information is taken and is automatically added under the registered user section.

SendGrid API

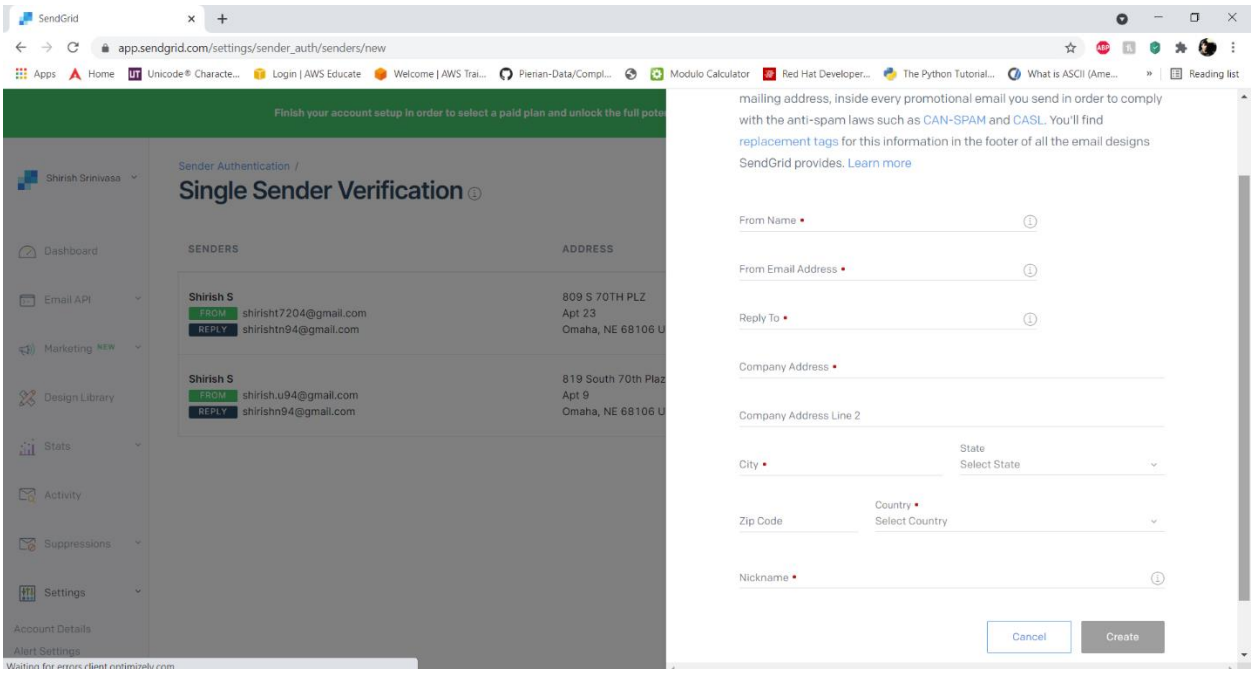
This API is used or sending email, we will start by going to:

https://app.sendgrid.com/

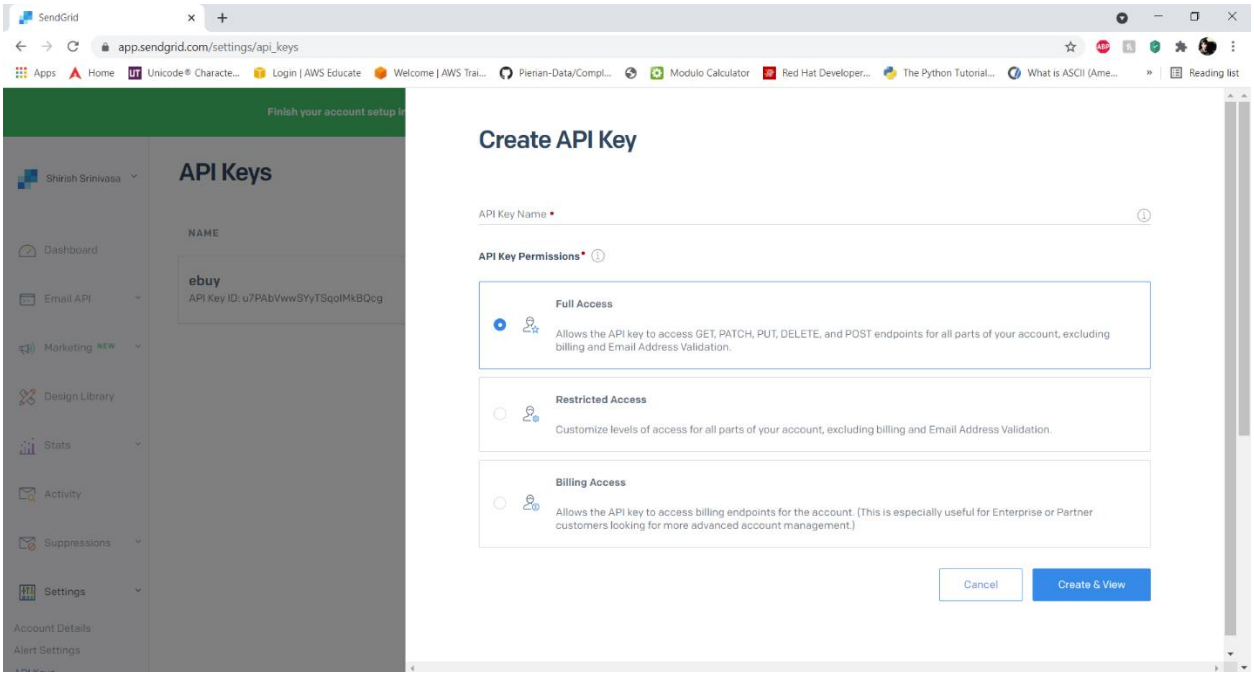
and creating an account and then login to the website.



Then we need to verify the sender's identity, usually through single sender verification or by giving a verified domain details.

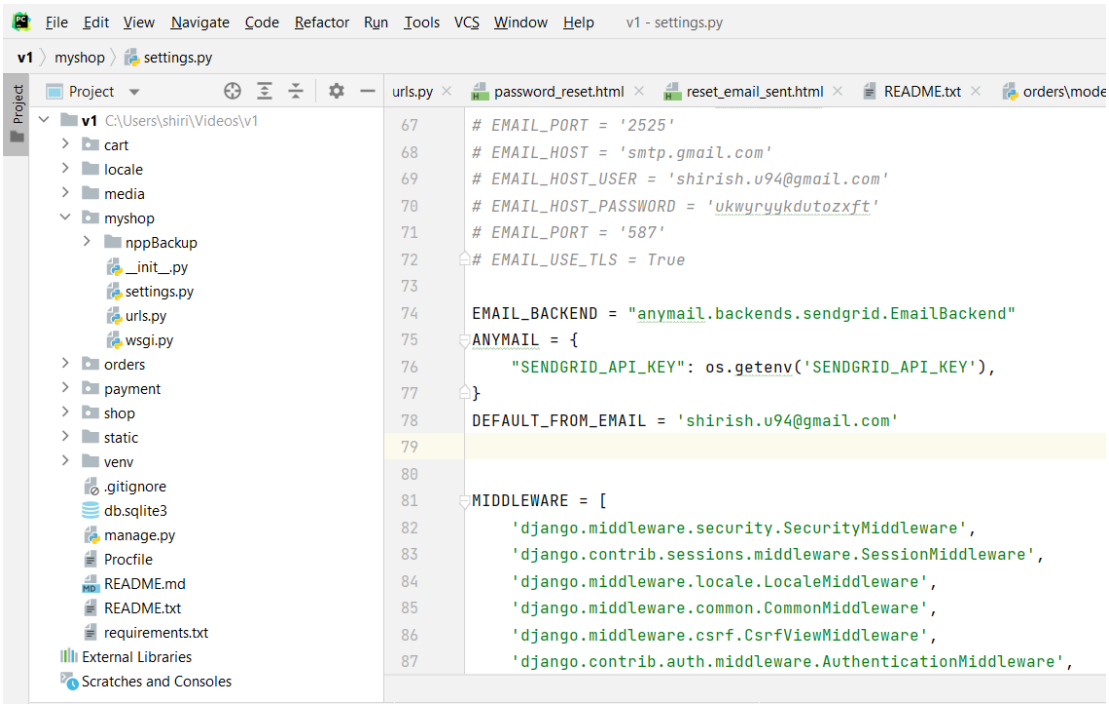


We can verify a single sender by filling out a form, where the from address will be utilized for sending the email in our e-buy web app



Then we need to create an API key with full access restriction, and give a name to the API key, and the key will be shown only once and will not be shown again for security purpose.

We nee to go to settings -> API Keys -> Create API Key



Now we need to install the packages:

Pip install django-anymail

Pip install sendgrid

Then we need to add these lines in the installed apps:

'anymail',

Now in the settings.py file of the e-buy application, we can add the following lines of code to use the SendGrid API

EMAIL_BACKEND = "anymail.backends.sendgrid.EmailBackend"

ANYMAIL = {

"SENDGRID_API_KEY": os.getenv('SENDGRID_API_KEY'),

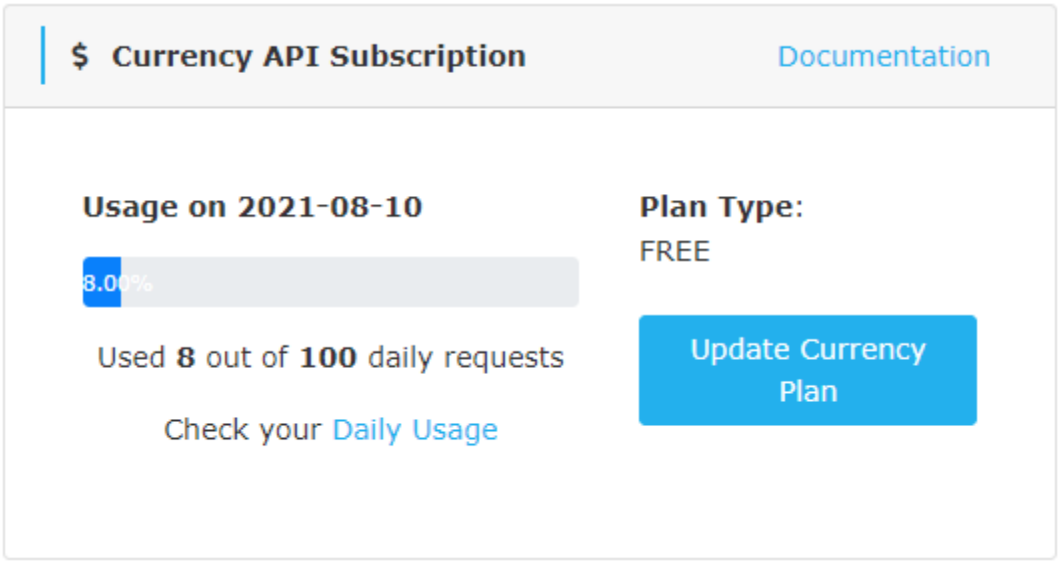
}

DEFAULT_FROM_EMAIL = 'shirish.u94@gmail.com'

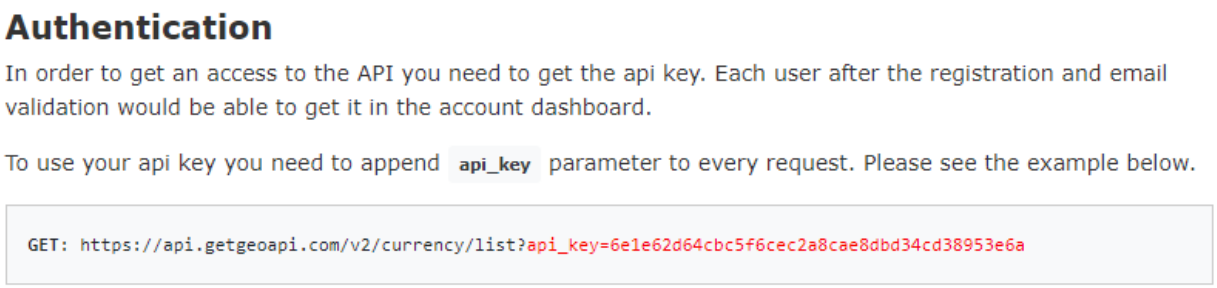
Then we are going to add the API key in the environment variable during our deployment to Heroku. The key should not be publicly exposed, as SendGrid actively monitor these keys, and if it is exposed to public, they will immediately block the account.

Get Geo Currency API:

Get Geo Currency provides exchange rates from one currency to another for a specific day.
First, I registered for a free account at https://currency.getgeoapi.com/register/.
Username: lahenkenius@unomaha.edu
Password: n3cxLVuhTFR7wDP
The free level of account allows 100 daily requests. You can monitor how many requests you have used at https://getgeoapi.com/account/.



Once I verified my email, I was able to access the documentation page at https://currency.getgeoapi.com/documentation/.
This page contained my API key along with documentation on how to formulate requests.



Get Geo Currency supports both JSON and XML requests. I chose to use a JSON request. A GET request begins with https://api.getgeoapi.com/v2/currency/convert and then your API key. You can add optional request parameters on after this.
The from parameter is optional and specifies the base currency. It is optional because the default base currency is the Euro, and this is what will be used if you do not specify a base currency. Because our store is a USD default, I used USD for the from parameter.

The to parameter specifies the currency to convert to. It is optional; if one is not provided then a list of all supported currencies is returned. However, the free plan requires you to specify a currency. This portion of the request varied based on what currency the user wanted the store converted to. I used a Django application called Django_countries which has a built-in widget to allow users to select a country. I then used a Django application called pycountry to convert from country name to the 3-character currency code associated with that country. I could then pass that code in the API request.

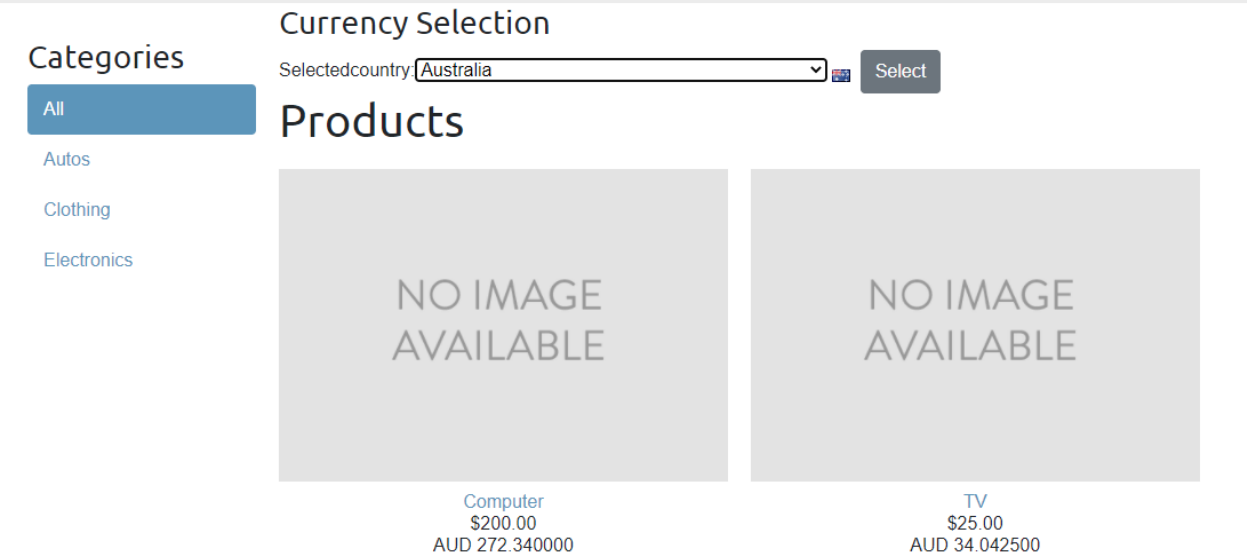
```
#pull selected country
if request.method == 'POST':
        selected_country = request.POST['selectedcountry']

# get appropriate currency
countryname = pycountry.countries.get(alpha_2=selected_country)
currency = pycountry.currencies.get(numeric=countryname.numeric)
try:
    currencycode = currency.alpha_3
except:
    currencycode = 'EUR'
else:
    currencycode = currency.alpha_3
```

```
#pull current rate
r1string = 'https://api.getgeoapi.com/v2/currency/convert?api_key=6e1e62d64cbc5f6cec2a8cae8dbd34cd38953e6a&from=USD&to='
r2string = currencycode
r3string = '&format=json'
apistring = r1string + r2string + r3string
response = requests.get(apistring)
exchangerate = response.json()
extrate = exchangerate['rates']
exttrate = extrate[currencycode]
exrate = exttrate['rate']
```

The amount parameter is optional and specifies the amount to convert. If no value is provided, 1 is used as the amount. I did not provide a value because I wanted to know what 1 dollar converted to. I could use that value as the rate so I could convert multiple prices with one API call.
The final parameter specifies the output format, and it can be JSON or XML. I chose JSON.
My requests look like this:

Base: https://api.getgeoapi.com/v2/currency/convert
+ API key: 6e1e62d64cbc5f6cec2a8cae8dbd34cd38953e6a
+ &from=USD
+ &to=GBP (note, this changes based on the request, but is always a 3 character abbreviation for the currency
+ &JSON

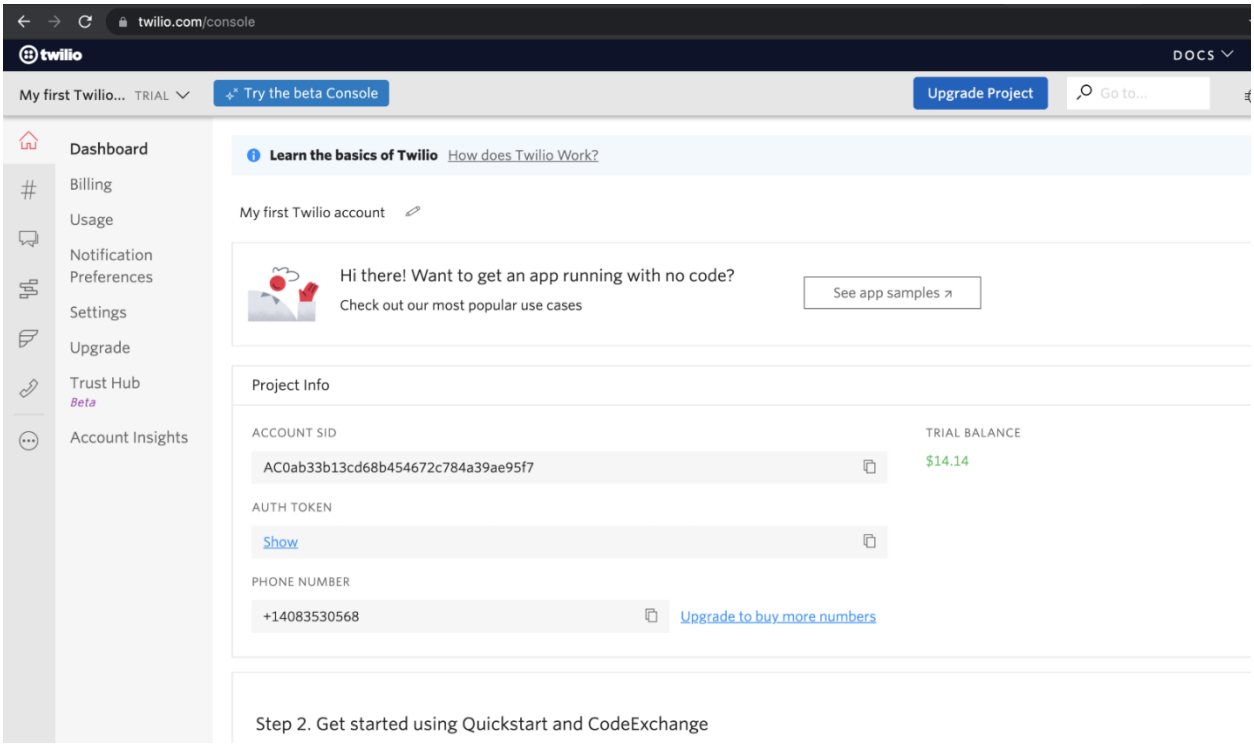
https://api.getgeoapi.com/v2/currency/convert?api_key=6e1e62d64cbc5f6cec2a8cae8dbd34c38953e6a&from=EUR&to=GBP&format=json.
This returned a rate that I could then multiply my prices by to convert from USD to the selected currency, in this case GBP. I then used Django-mathfilters to multiply the price by the exchange rate in the product list template.



Twilio API:

Twilio's Programmable SMS API helps you send and receive SMS messages. You'll need to sign up for a free Twilio account to get started. Twilio's APIs (Application Programming Interfaces) Power its platform for communications. Behind these APIs is a software layer connecting and optimizing communications networks around the world to allow your users to call and message anyone, globally.

Trial Account



With the help of an authentication token, we can implement the SMS API in our Django application. We need to register first to get a trial phone number and then activated verified caller IDS. With the help of a registered phone number SMS will be triggered to the verified CALLER ID's.

SMS API authentication:

We will use Twilio **Account SID** as the username and your **Auth Token** as the password for HTTP Basic authentication with Twilio.

Twilio SMS API is integrated into JSON Response.

JSON

Twilio also supports returning resource representations as JSON. Simply add the `.json` extension to any resource URI. Here is the above resource represented as JSON:

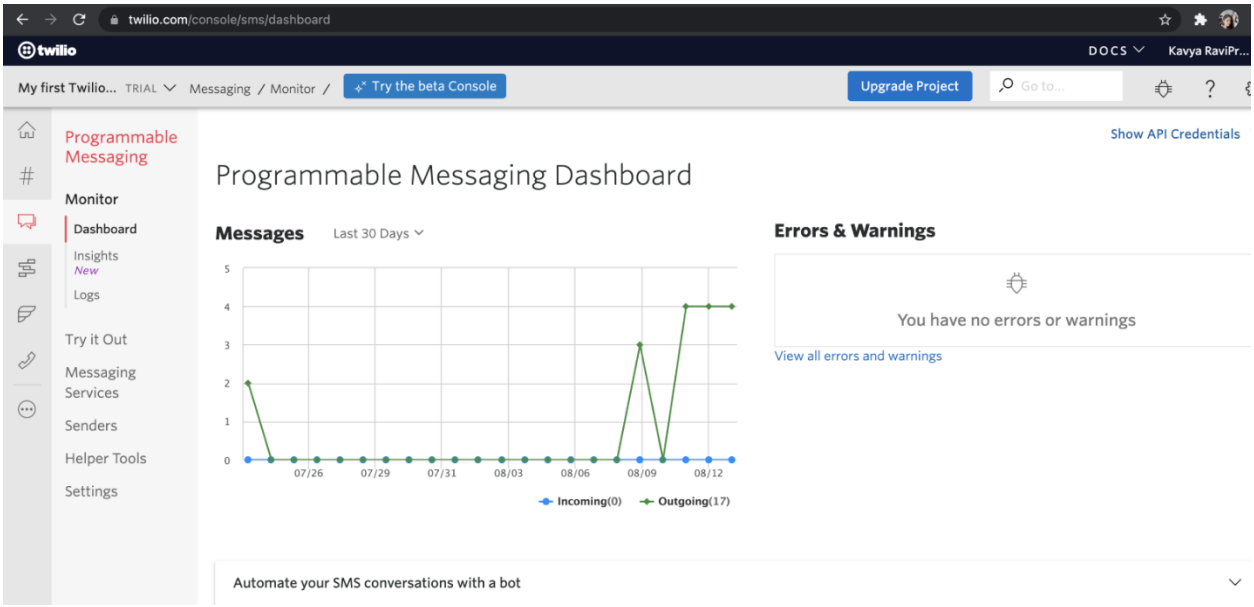
```
GET /2010-04-01/Accounts/ACXXXXX.../Messages/SM1f0
```

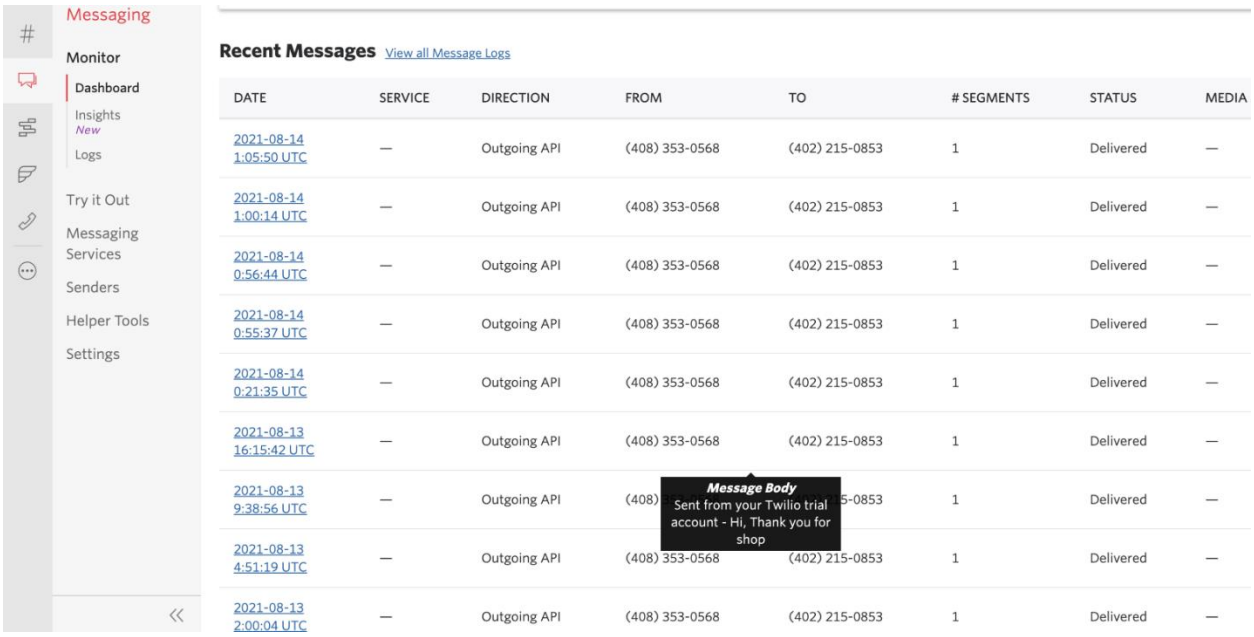

```
{
    "sid": "SM1f0e8ae6ade43cb3c0ce4525424e404f",
    "date_created": "Fri, 13 Aug 2010 01:16:24 +0000
    "date_updated": "Fri, 13 Aug 2010 01:16:24 +0000
    "date_sent": null,
    "account_sid": "ACXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
    "to": "+15305431221",
    "from": "+15104564545",
    "body": "A Test Message",
    "status": "queued",
    "flags":["outbound"],
    "api_version": "2010-04-01",
    "price": null,
    "uri": "\/2010-04-01\/Accounts\/ACXXXXXXXXXXXXXX
}
```


SMS is triggered when following code is added to the views of shop model:

```
account_sid = 'AC0ab33b13cd68b454672c784a39ae95f7'
auth_token = '6ef3b136558ad1c93b72dafb51507766'
client = Client(account_sid, auth_token)
message = client.messages.create(
    body='Hi, Thank you for shopping at e-buy,your order has been placed and we will deliver you asap.'
    from_='+14083530568',
    to='+14022150853'
)
```

Once payment is completed, with the help of Twilio authentication token SMS is triggered to the registered verified caller ID's.


twilio.com/console/sms/dashboard
twilio
DOCS
Kavya RaviPr...
My first Twilio... TRIAL
Messaging / Monitor /
Try the beta Console
Upgrade Project
Go to...
Programmable Messaging
Monitor
Dashboard
Insights New
Logs
Try it Out
Messaging Services
Senders
Helper Tools
Settings
Show API Credentials
Programmable Messaging Dashboard
Messages
Last 30 Days
5
4
3
2
1
0
07/26
07/29
07/31
08/03
08/06
08/09
08/12
Incoming(0)
Outgoing(17)
Errors & Warnings
You have no errors or warnings
View all errors and warnings
Automate your SMS conversations with a bot


Messaging
Monitor
Dashboard
Insights New
Logs
Try it Out
Messaging Services
Senders
Helper Tools
Settings
Recent Messages
View all Message Logs
DATE
SERVICE
DIRECTION
FROM
TO
SEGMENTS
STATUS
MEDIA
2021-08-14 1:05:50 UTC — Outgoing API (408) 353-0568 (402) 215-0853 1 Delivered —
2021-08-14 1:00:14 UTC — Outgoing API (408) 353-0568 (402) 215-0853 1 Delivered —
2021-08-14 0:56:44 UTC — Outgoing API (408) 353-0568 (402) 215-0853 1 Delivered —
2021-08-14 0:55:37 UTC — Outgoing API (408) 353-0568 (402) 215-0853 1 Delivered —
2021-08-14 0:21:35 UTC — Outgoing API (408) 353-0568 (402) 215-0853 1 Delivered —
2021-08-13 16:15:42 UTC — Outgoing API (408) 353-0568 (402) 215-0853 1 Delivered —
2021-08-13 9:38:56 UTC — Outgoing API (408) 353-0568 (402) 215-0853 1 Delivered —
2021-08-13 4:51:19 UTC — Outgoing API (408) 353-0568 (402) 215-0853 1 Delivered —
2021-08-13 2:00:04 UTC — Outgoing API (408) 353-0568 (402) 215-0853 1 Delivered —
Message Body
Sent from your Twilio trial account - Hi, Thank you for shop

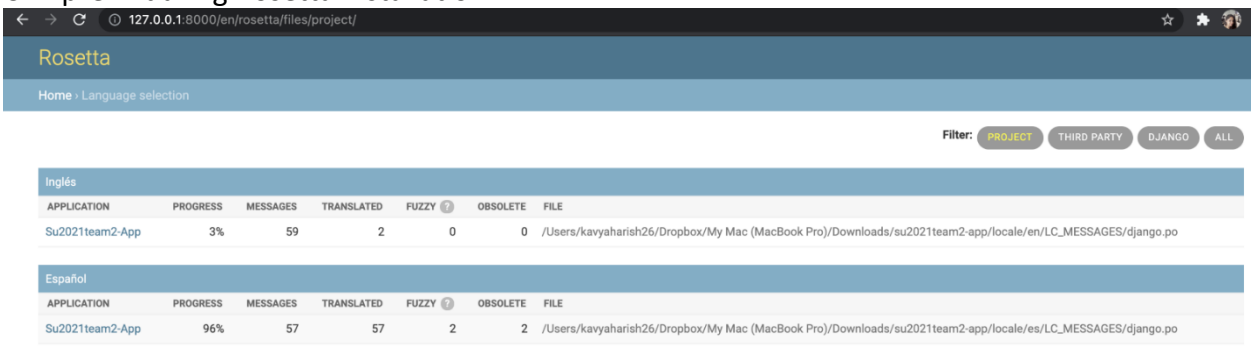
Rosetta API

Rosetta is a third-party application that eases the translation process in Django projects. Rosetta can be installed and uninstalled by simply adding and removing a single entry in your project's INSTALLED_APPS and a single line in your main urls.py file.
Install Rosetta Application to the project:

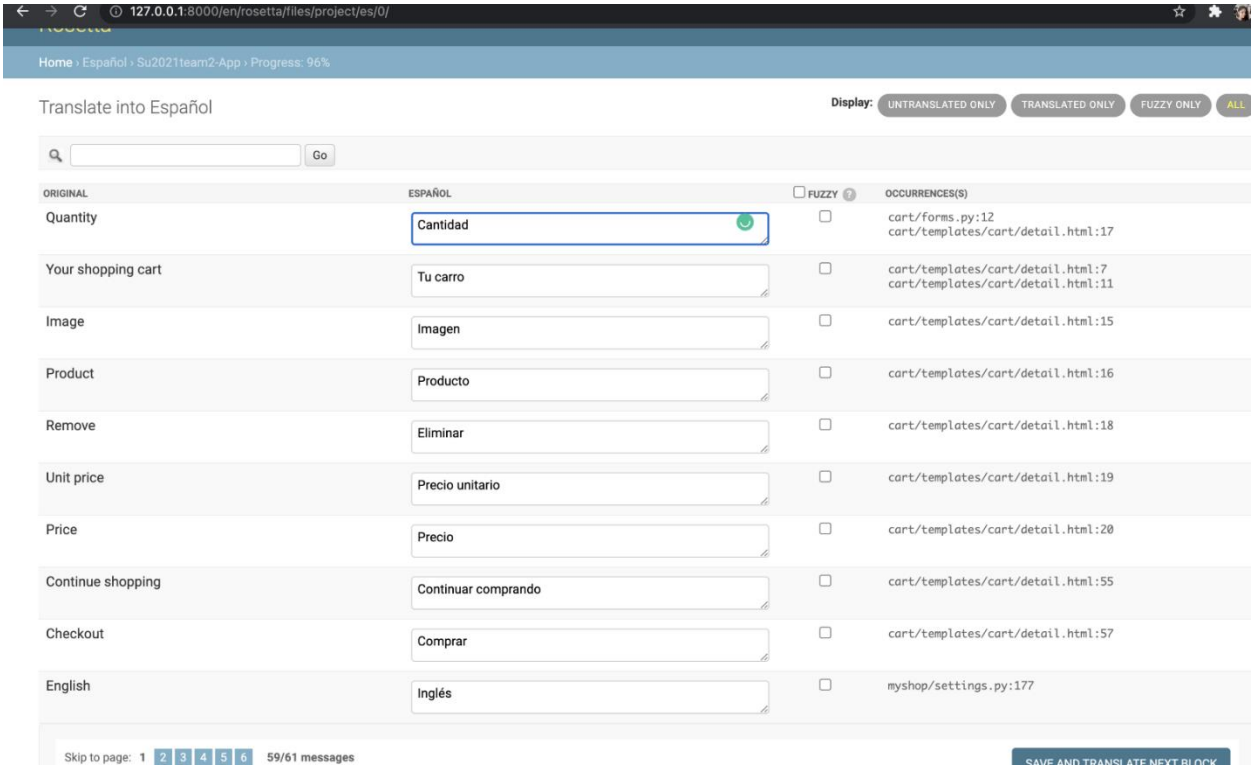
1. **pip install django-rosetta**
2. **Add 'rosetta' to the INSTALLED_APPS in your project's settings.py**
3. **Add an URL entry to your project's urls.py, for example:**

```
from django.conf import settings
from django.conf.urls import include, re_path

if 'rosetta' in settings.INSTALLED_APPS:
    urlpatterns += [
        re_path(r'^rosetta/', include('rosetta.urls'))
    ]
```

Usage of Rosetta:
URL prefix during Rosetta installation:



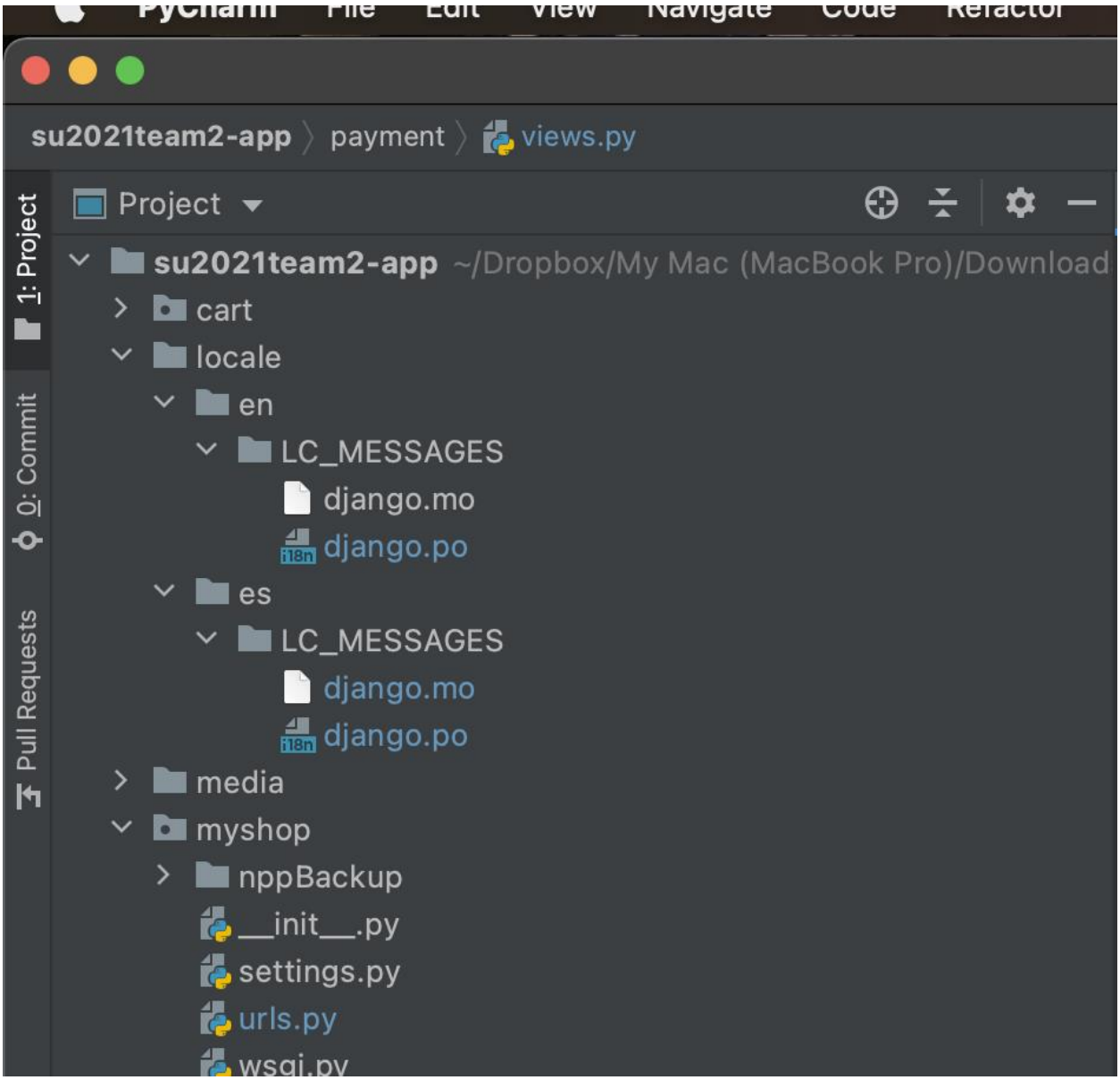
Select a file and translate each untranslated message. Whenever a new batch of messages is processed, Rosetta updates the corresponding *django.po* file and regenerates the corresponding .mo file.
We need to add Rosetta widget in WSGI Auto reload variable in conf/settings.py file.
Translate all words from untranslated to translated using Rosetta widgets:



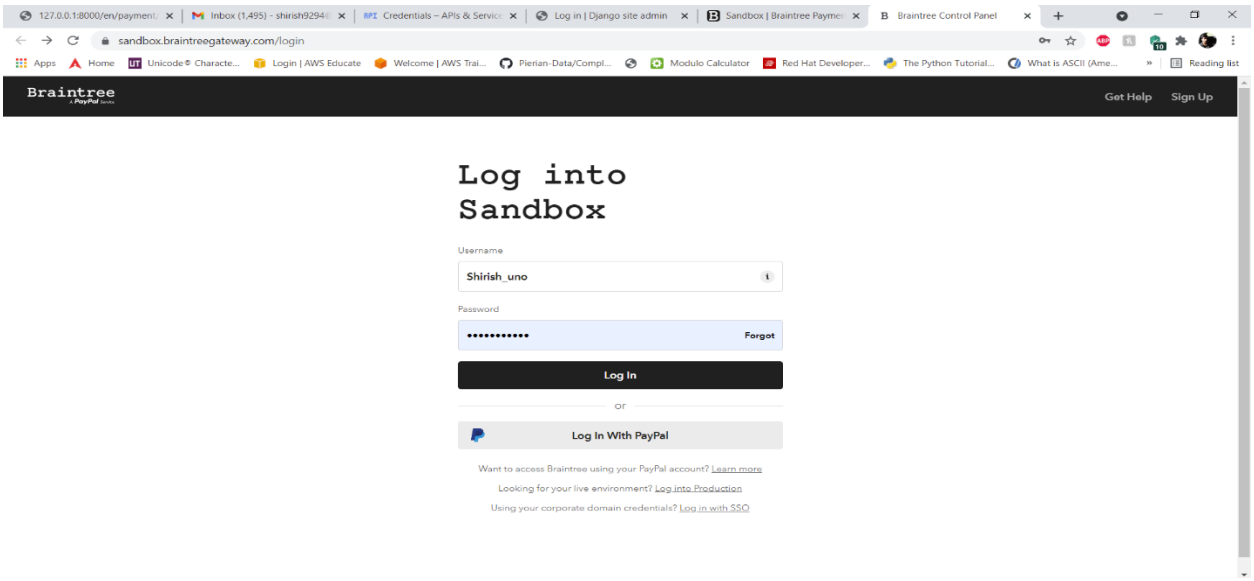
Translating Rosetta itself

By default, Rosetta hides its own catalog files in the file selection interface (shown above.) If you would like to translate Rosetta to your own language:

1. Create a subdirectory for your locale inside Rosetta's locale directory, e.g., rosetta/locale/XX/LC_MESSAGES
2. Instruct Django to create the initial catalog, by running django-admin.py makemessages -l XX inside Rosetta's directory.
3. Instruct Rosetta to look for its own catalogs, by appending *?rosetta* to the language selection page's URL, e.g. http://127.0.0.1:8000/rosetta/pick/?rosetta
4. Translate as usual
5. Send a pull request if you feel like sharing.

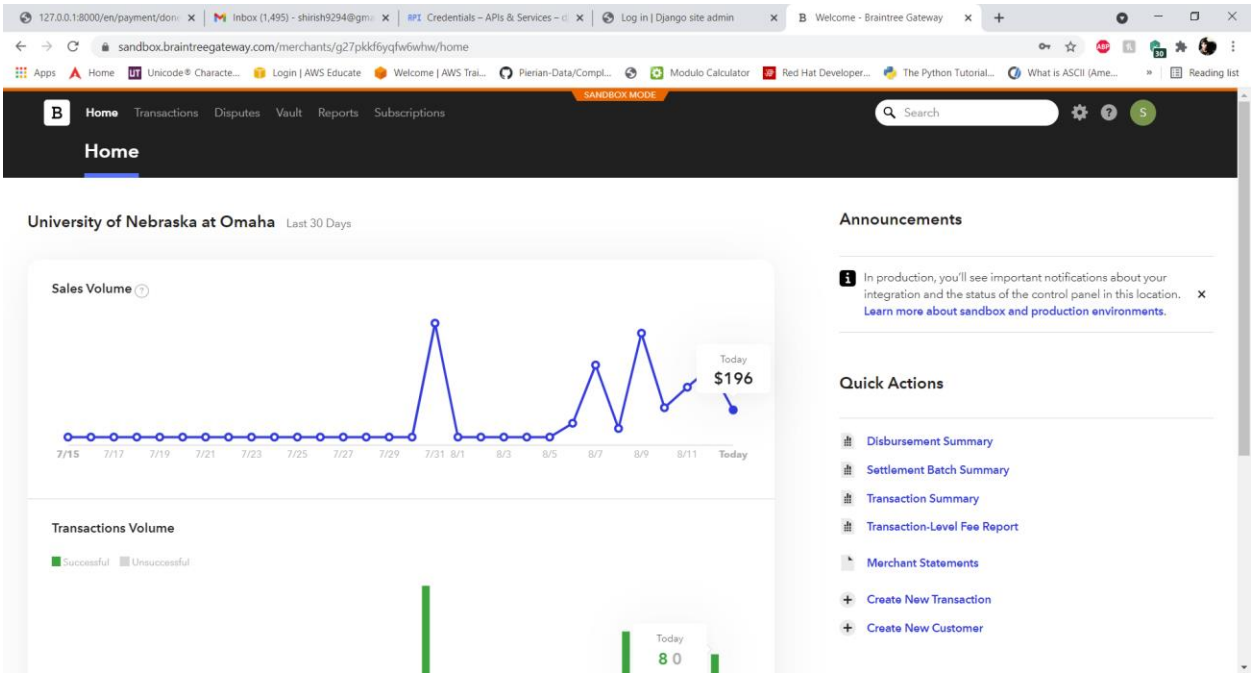

su2021team2-app
payment
views.py
Project
su2021team2-app ~/Dropbox/My Mac (MacBook Pro)/Download
cart
locale
en
LC_MESSAGES
django.mo
django.po
es
LC_MESSAGES
django.mo
django.po
media
myshop
nppBackup
__init__.py
settings.py
urls.py
wsgi.py
1: Project
0: Commit
Pull Requests

Braintree API

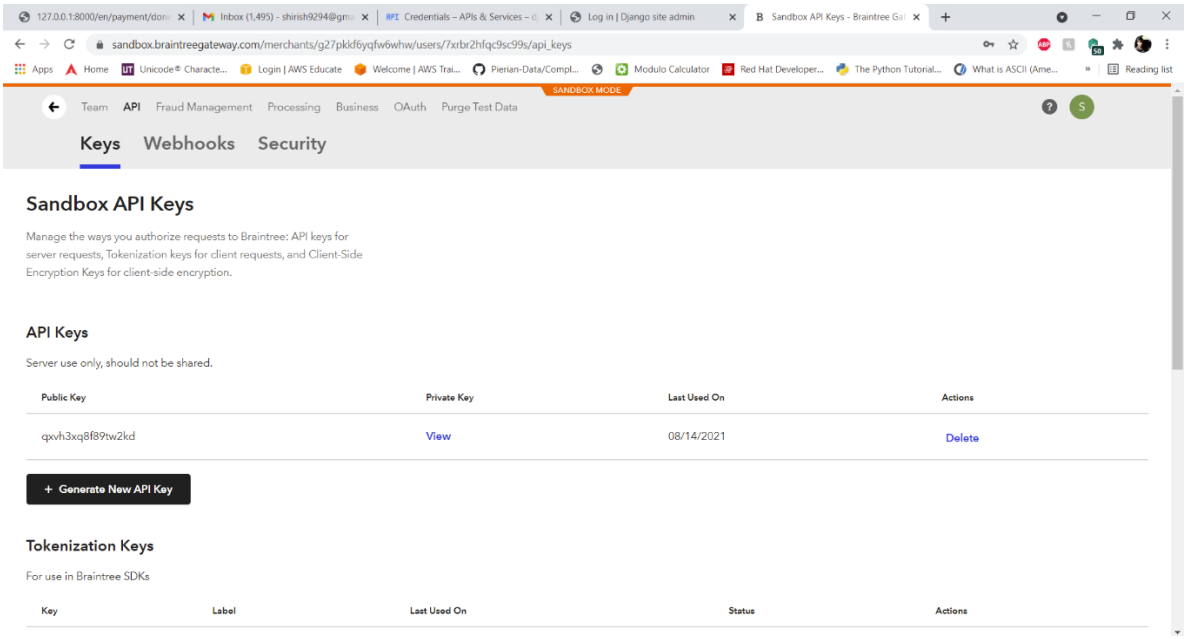


First, we are going to create an account in the Braintree sandbox, and the login.

Address: https://sandbox.braintreegateway.com/



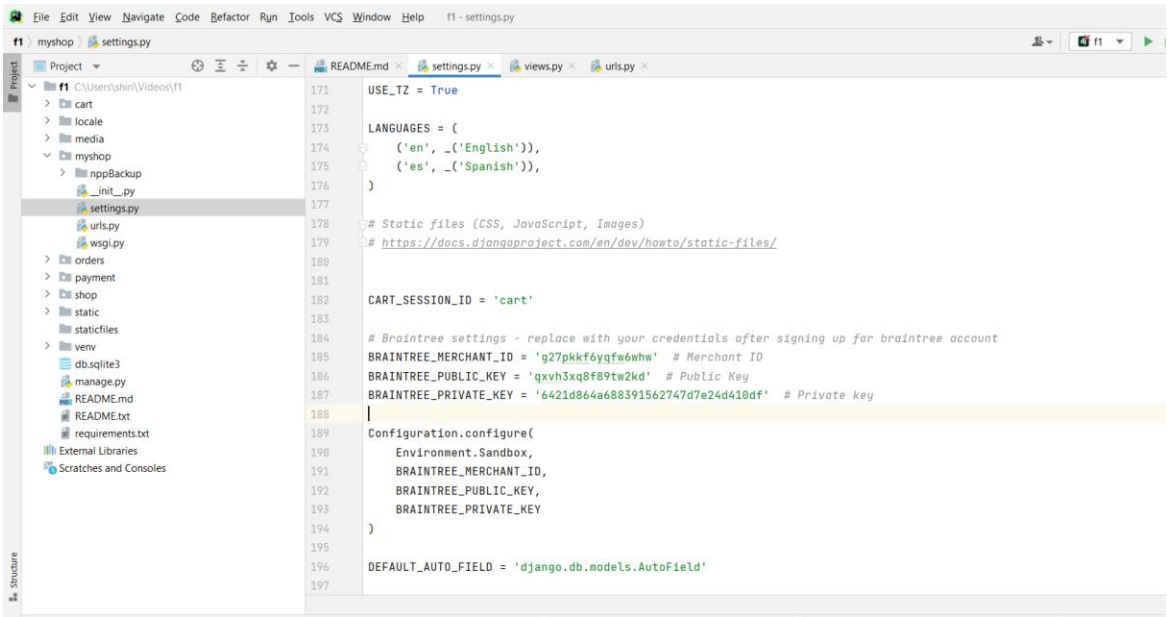
The is the Braintree dashboard where we can review any transaction that has happened recently.



To integrate our Braintree sandbox to our project we need to have our API keys namely

BRAINTREE_MERCHANT_ID, BRAINTREE_PUBLIC_KEY, BRAINTREE_PRIVATE_KEY

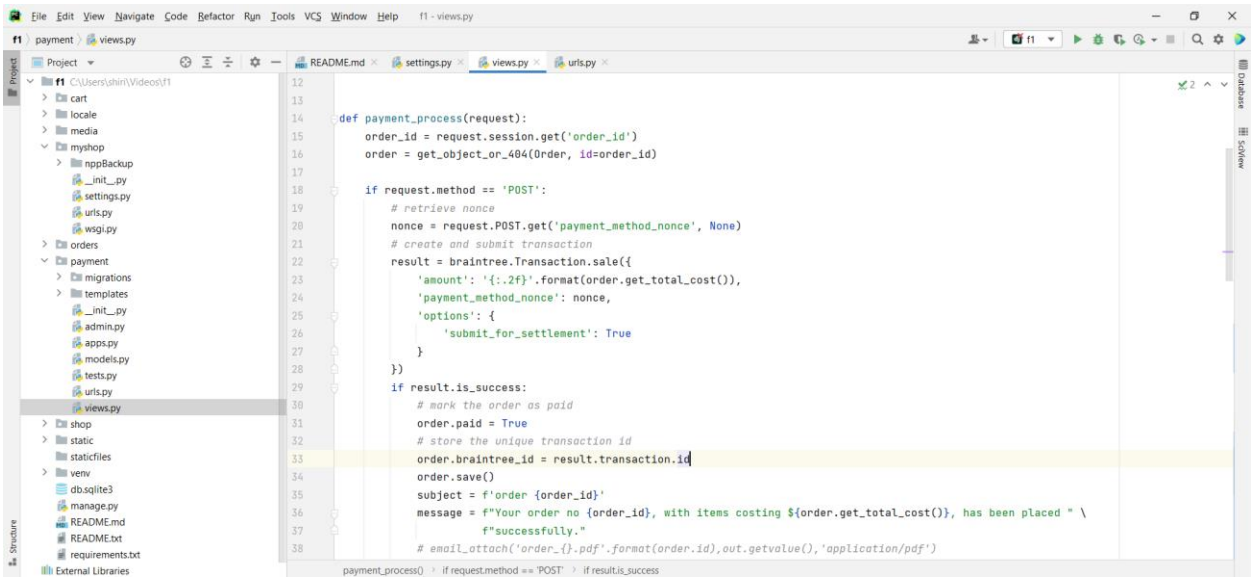
We can get this in our user account -> API -> Keys -> API keys



We will add the API keys in the settings.py file.

And run the requirement package

pip install braintree



And whenever transaction happens and if it is valid, this transaction details are stored in the Braintree account.

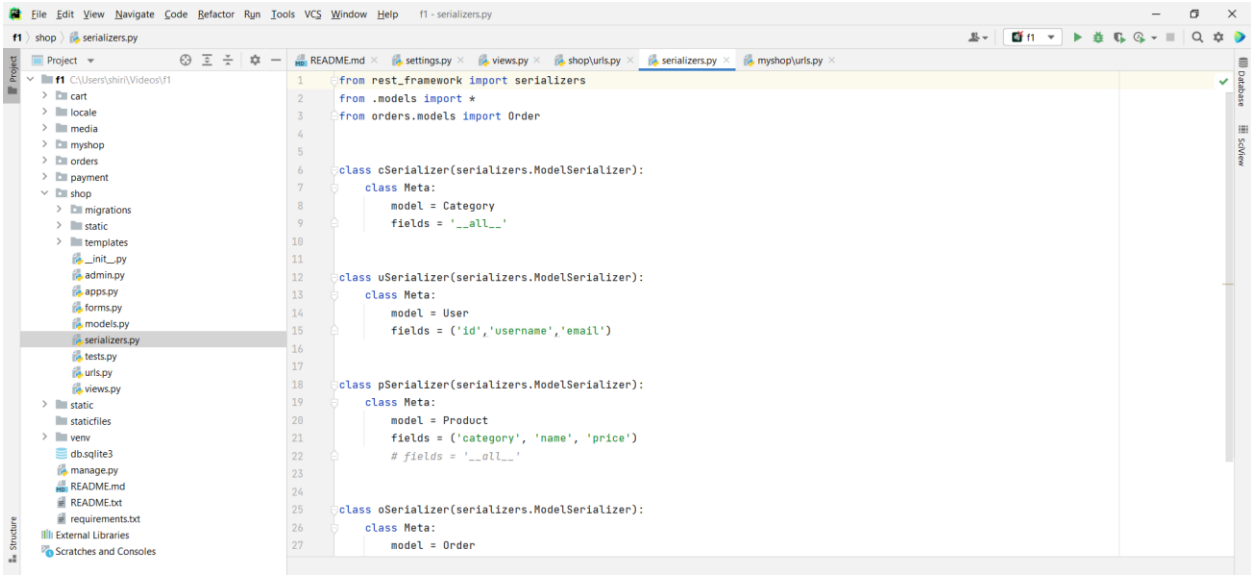
REST API

For utilizing REST API, we need to install the dependencies packages:

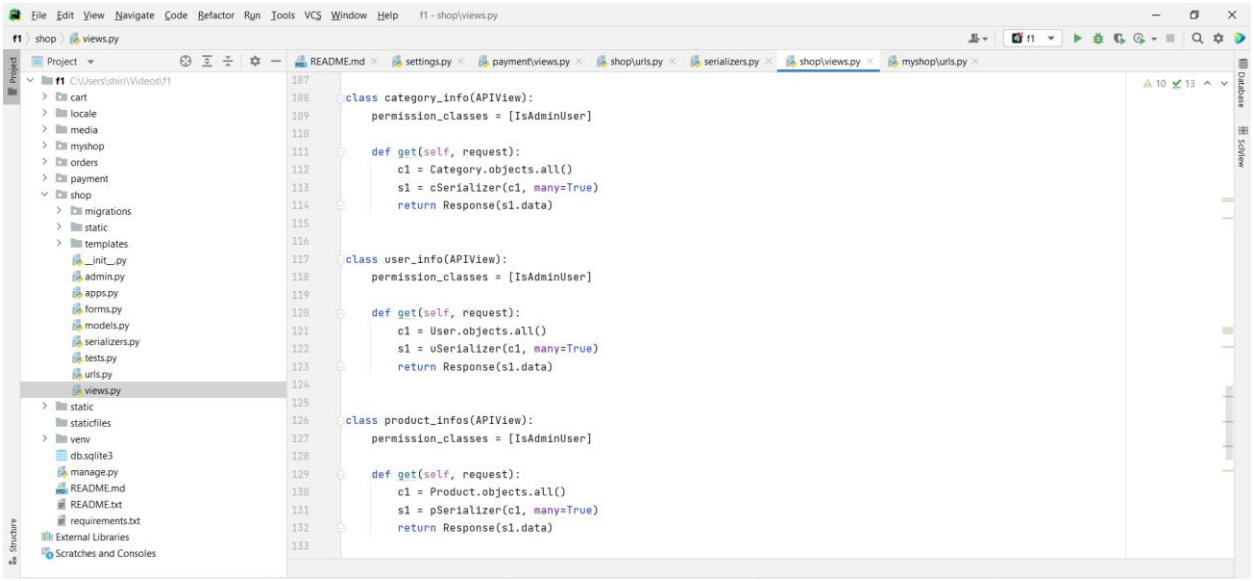
pip install djangorestframework

```
    'django_countries',
    'pycountry',
    'rosetta',
    'parler',
    'django.contrib.humanize',
    'anymail',
    'rest_framework',
    'storages',
]
```

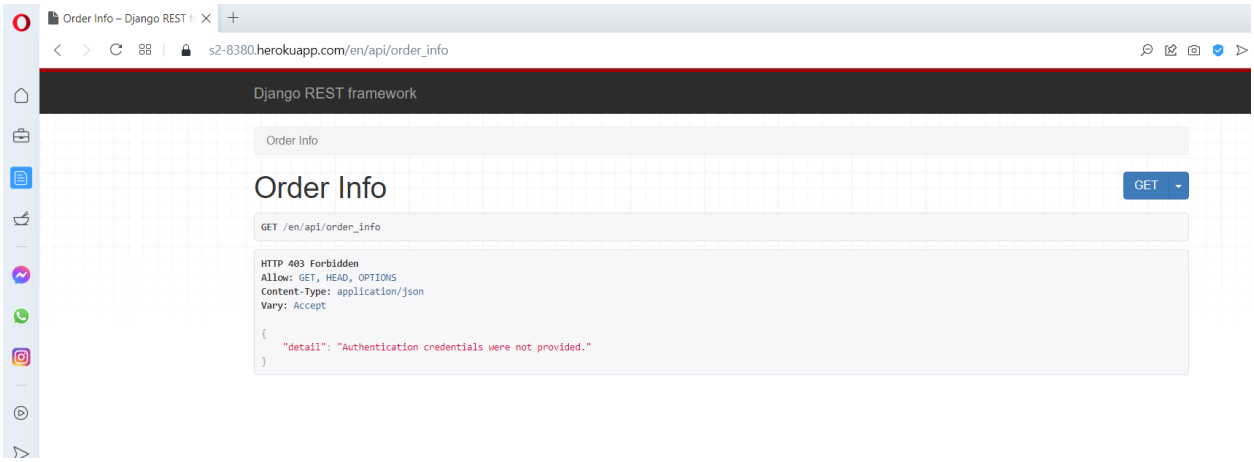
and then add rest_framework in the installed apps



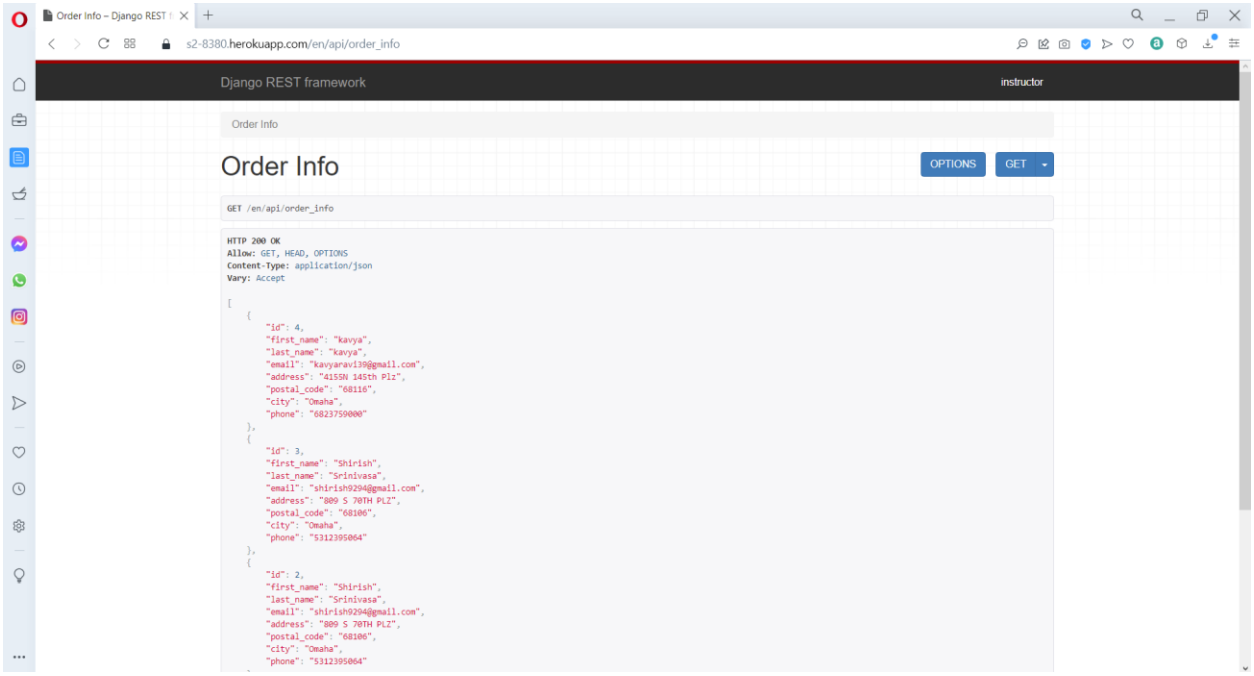
Now in the shop app we will create a file called serializers.py where we are going to create serializer to retrieve the required data from the data base.



In the views.py file under the shop app, we have added the API view and the code, where only an authorized admin user can access the REST API.



If an unauthorized user tries to access the REST API, he will be denied access.



Only when an authorized user accesses it, they can retrieve the data.

URL path for accessing different REST API data:

Order info - https://s2-8380.herokuapp.com/en/api/order_info

Product category - https://s2-8380.herokuapp.com/en/api/categorys_info

Product info - https://s2-8380.herokuapp.com/en/api/products_info

User info - https://s2-8380.herokuapp.com/en/api/users_info

Download code from GitHub and run it locally:

Important Note:

For the google sign in API to work, we need to add the URL redirection and the social authentication accounts, which is explained in these instructions.

Currently the code in the repository link:

https://github.com/ISQA-Classes/su2021team2-app.git

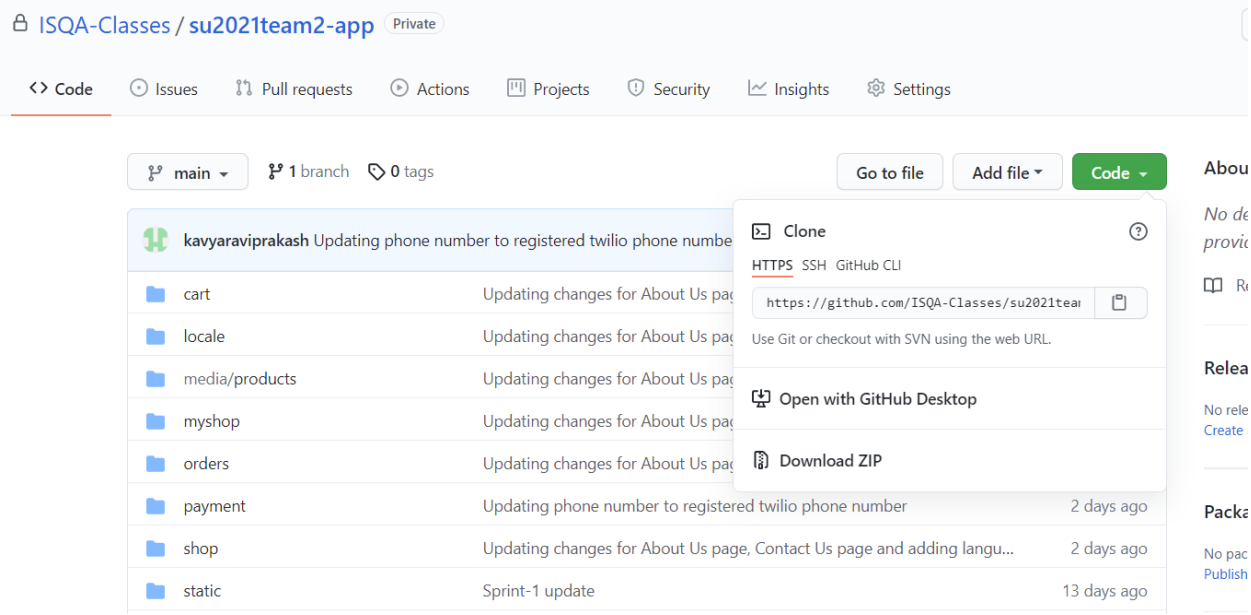
Is set to store the image and media content in the local system, the instructions will also give you an insight on how to connect to an S3 bucket and store it in an aws service.

Let's begin with the deployment process:

We can download the code from GitHub by using the following link:

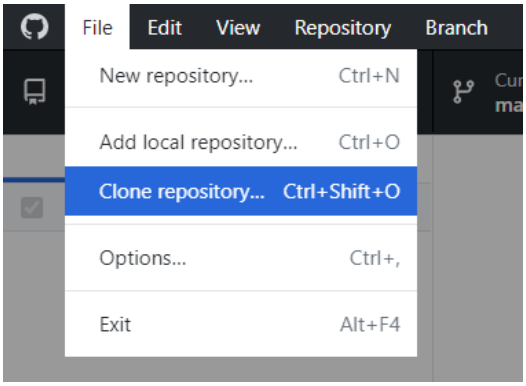
https://github.com/ISQA-Classes/su2021team2-app.git

you need to copy the URL in the repository and clone it by using GitHub desktop. So, you can run and test the code on your local computer.

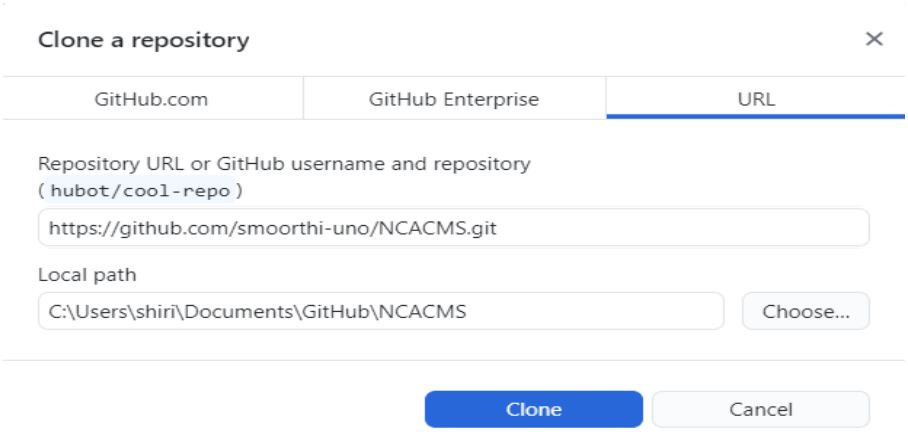


You need to copy the HTTPS link under the code option to download the files.

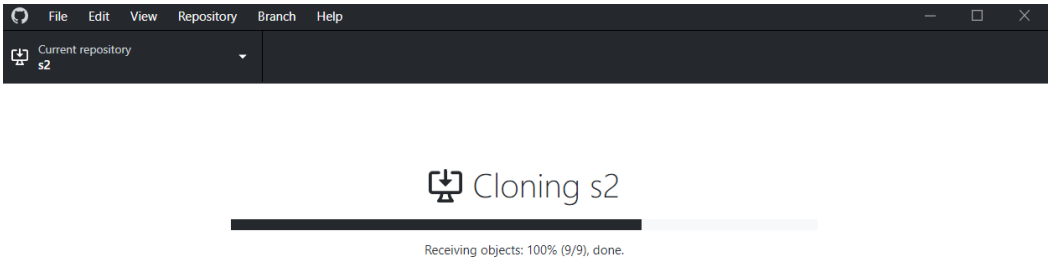
You can also download the files as a zip file and store it on your system.



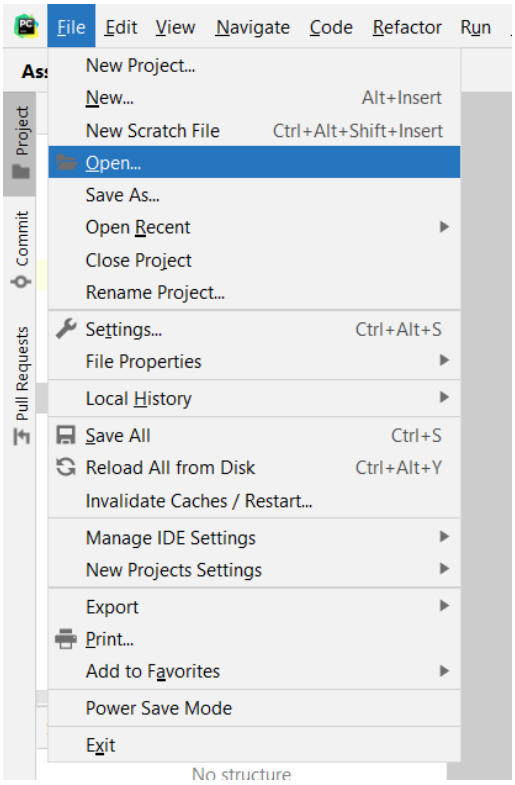
Open GitHub desktop application then click on File -> Clone Repository, then a popup window will appear.



In the Clone a repository window, click on the URL option on the top right and paste the URL link which you copied from GitHub and then click on Clone.

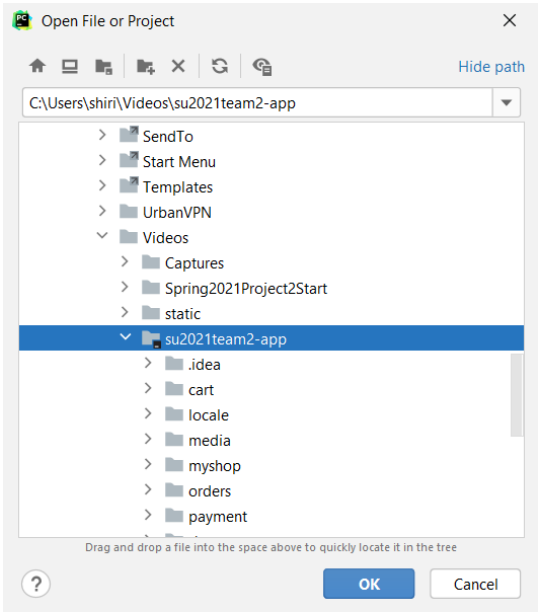


The Cloning process will start, and the files will be cloned to your local system.

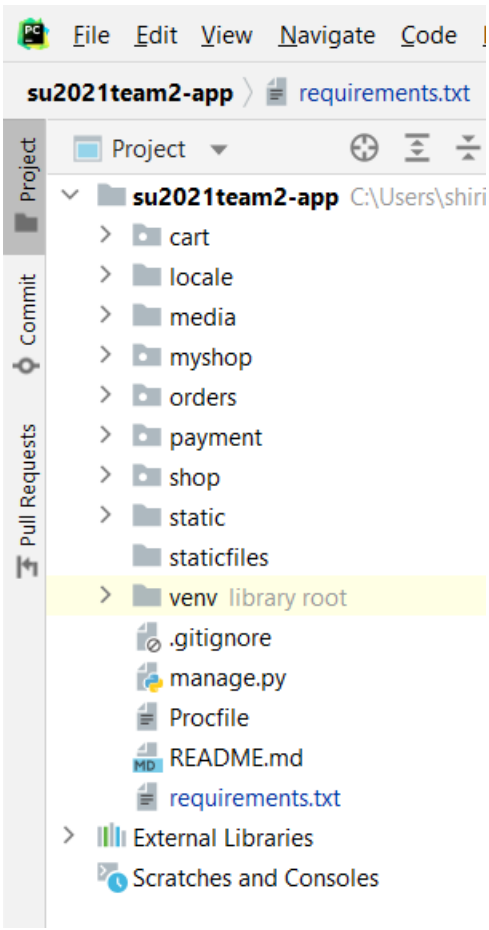


Now you need to open PyCharm and click on File -> Open.

This will open a navigation window for loading the Web application files.

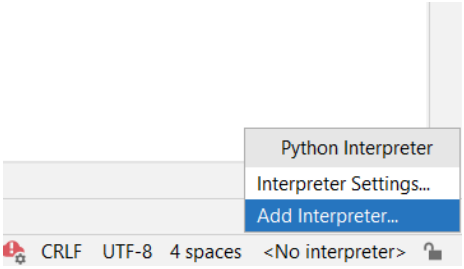


In the navigation windows locate where you have saved the web application files, select the folder, and click on OK. These files will get loaded into PyCharm.

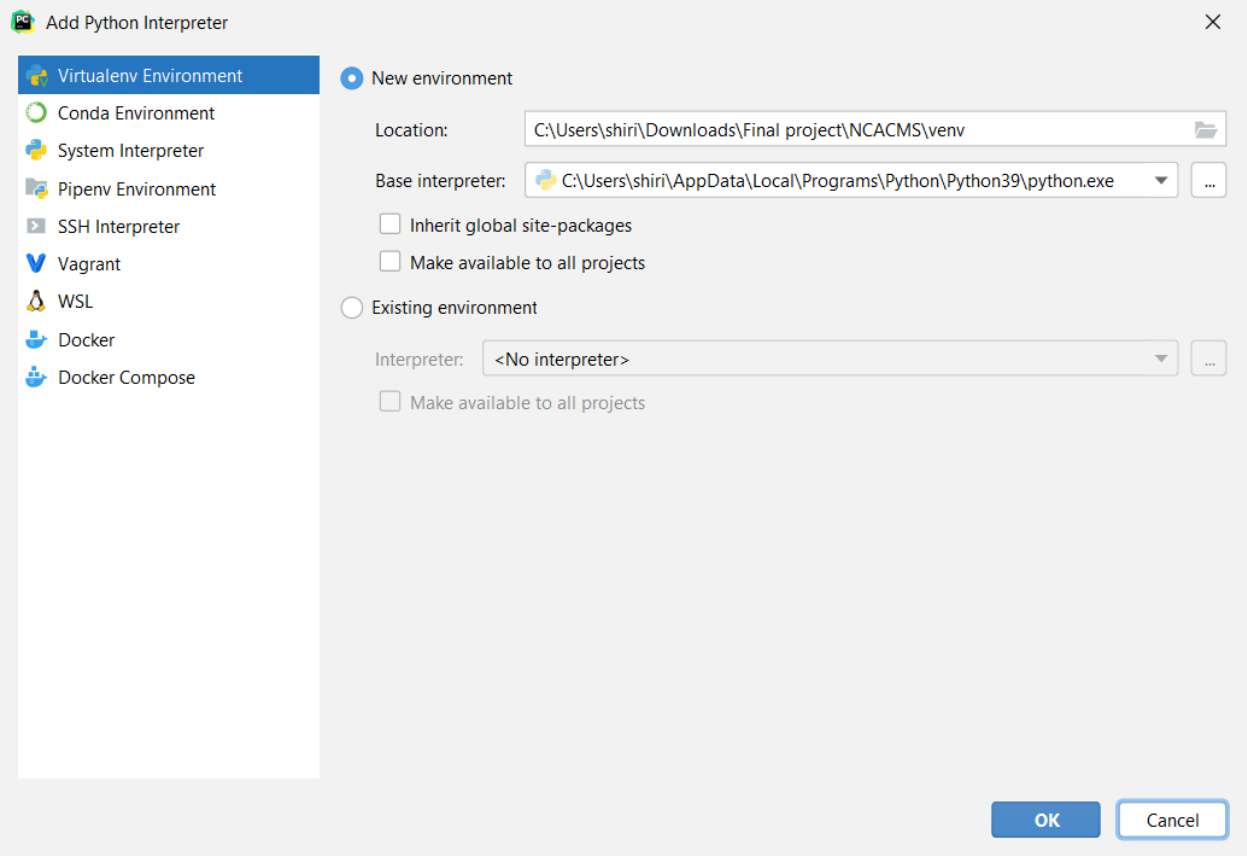


Now Click on the Project option which is located on the left side of the PyCharm application, it will show us the web application files.

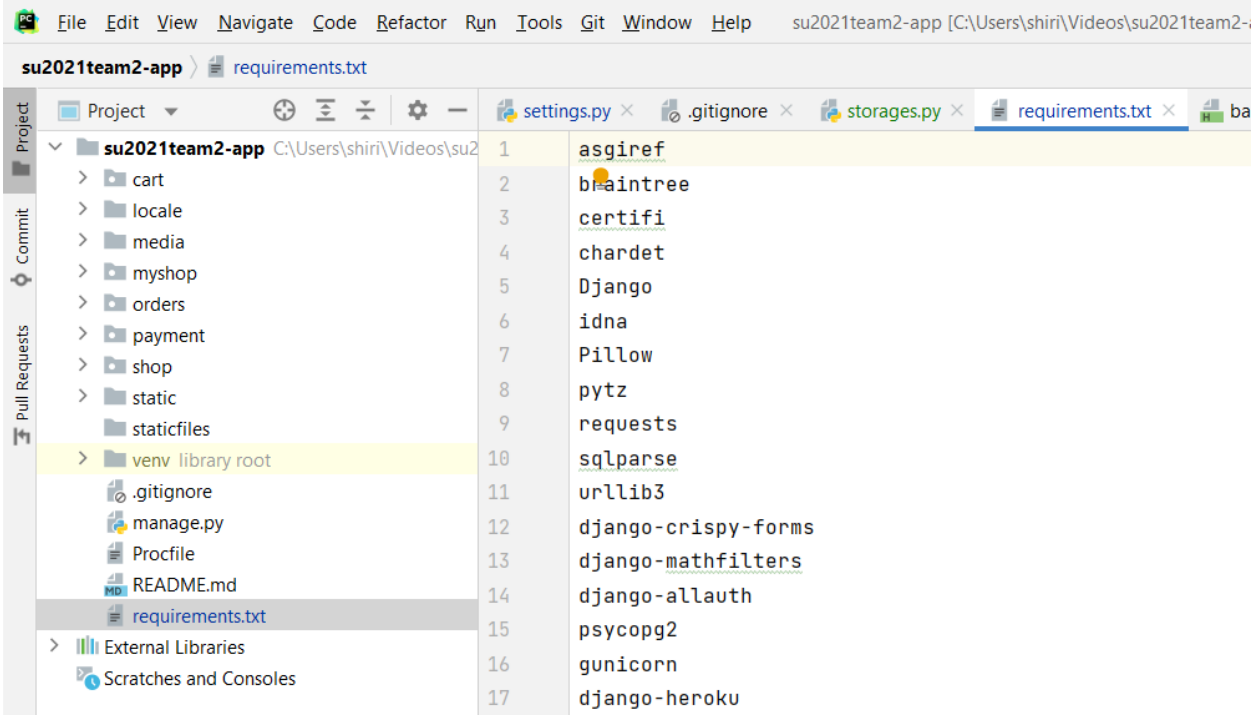
Now you need to create a virtual environment to run the web application.



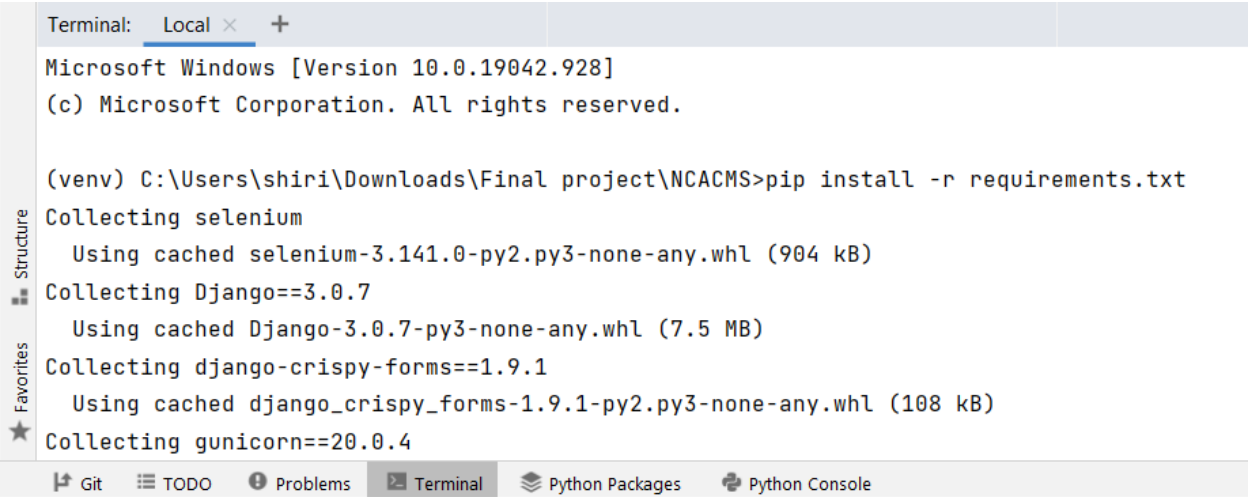
On the bottom right click on the -> Add Interpreter, this will open a new window where you can create a virtual environment.



When the Add Python Interpreter window opens, click on the Virtualenv Environment -> New environment, and click on Ok. This will create a new Virtual environment for the web application.



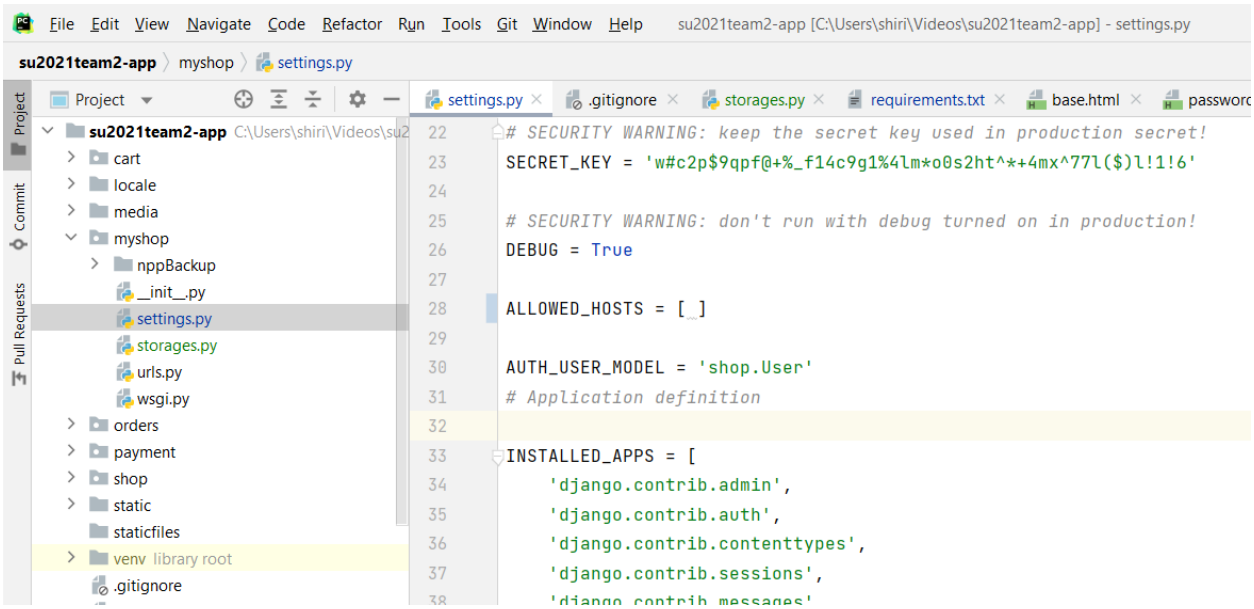
Now you need to install a list of packages that are in the requirements.txt file.



Open the Terminal, which is located at the bottom of the PyCharm window and type the command:

pip install -r requirements.txt

to install the packages specified in the requirements.txt file.



Make sure in the setting.py file that the debug is set to true, and the allowed hosts is empty.

DEBUG = True

ALLOWED_HOSTS = [ ]

To make migration run the commands in the terminal:

python manage.py makemigrations

python manage.py migrate

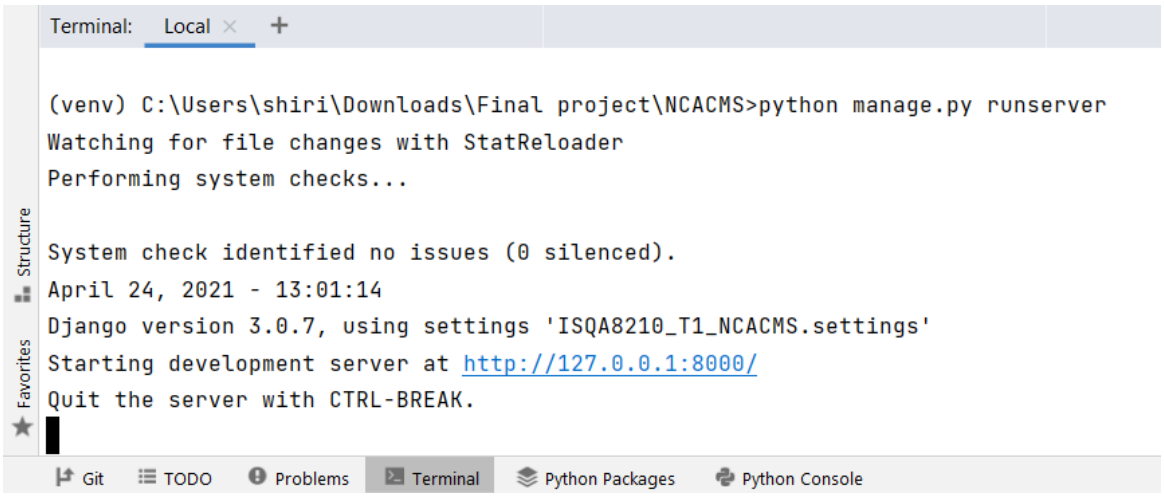
Then create a super user to gain access to the administrator panel, use the command:

python manage.py createsuperuser

and then enter the username, password, and the email address for creating the super user.

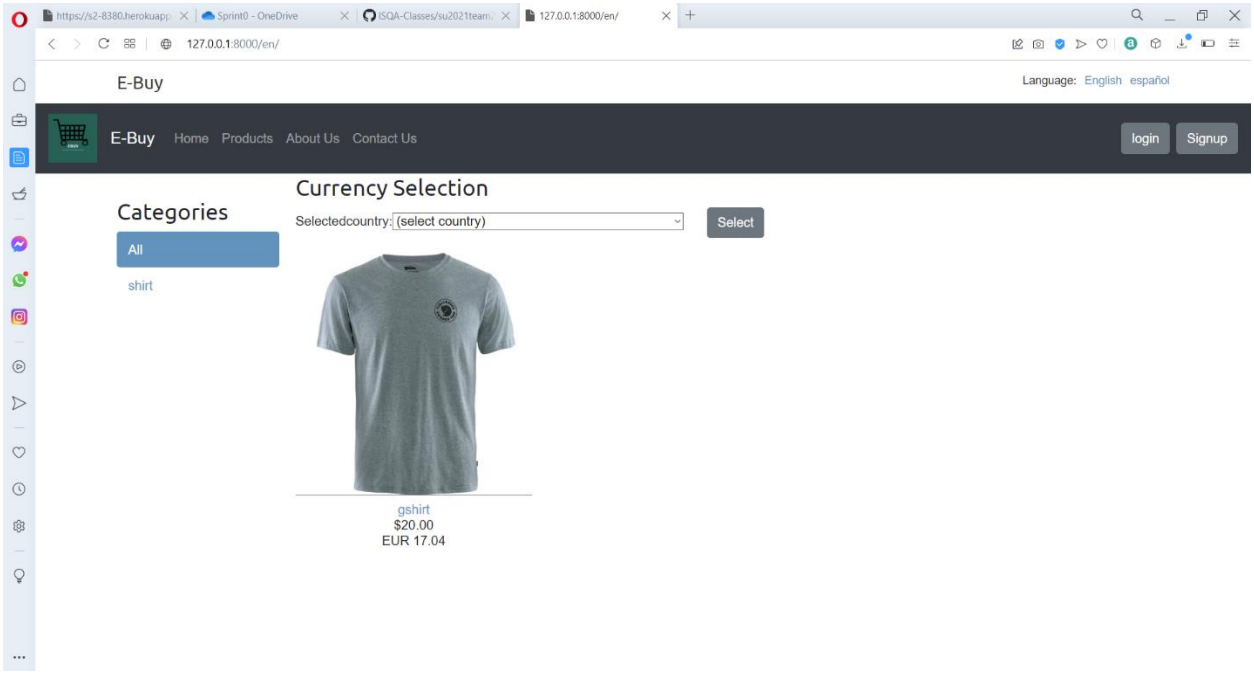
Now you can run the application by using the command:

python manage.py runserver



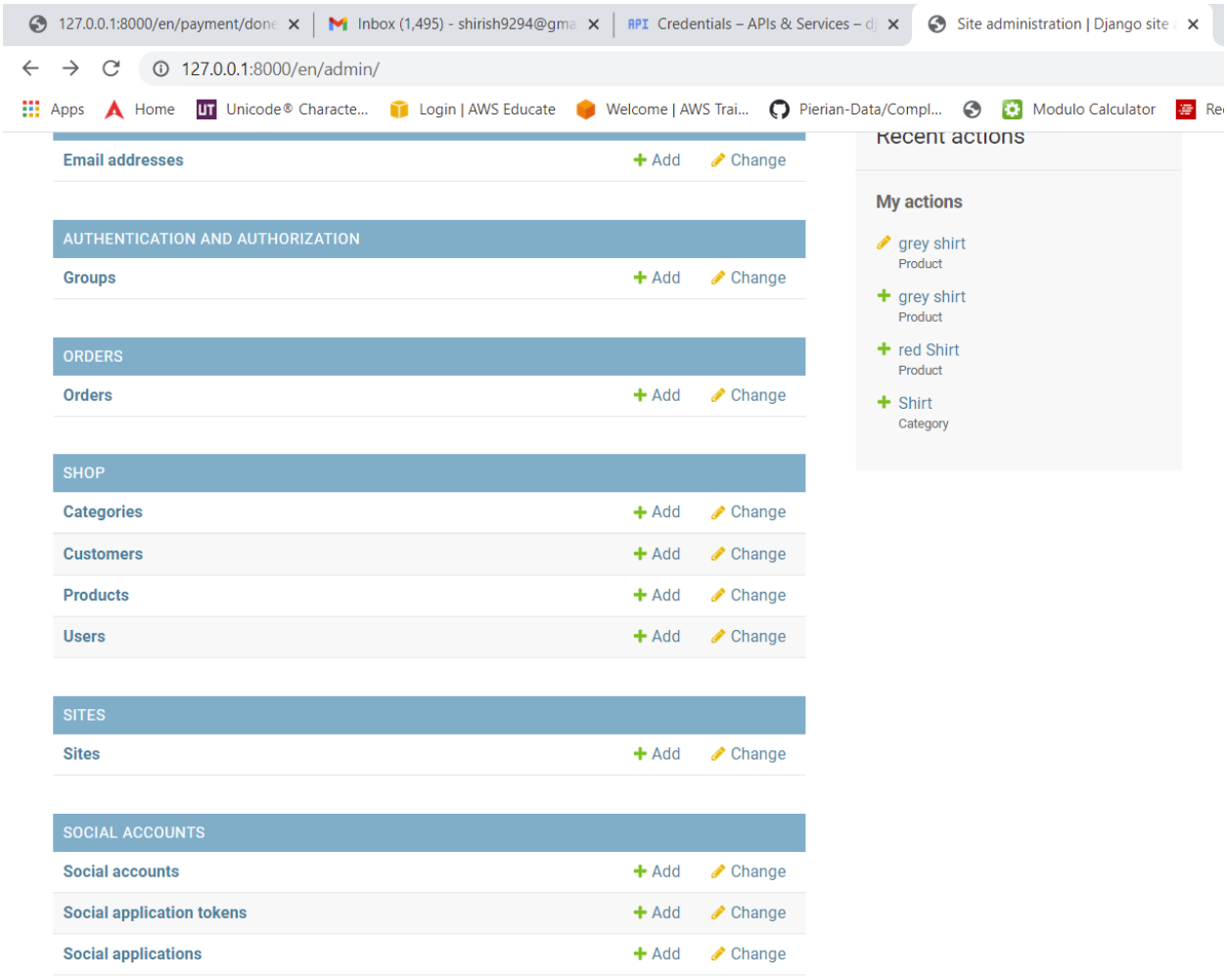
It will start the development server and will generate the http link for accessing the web application.

Link: http://127.0.0.1:8000/

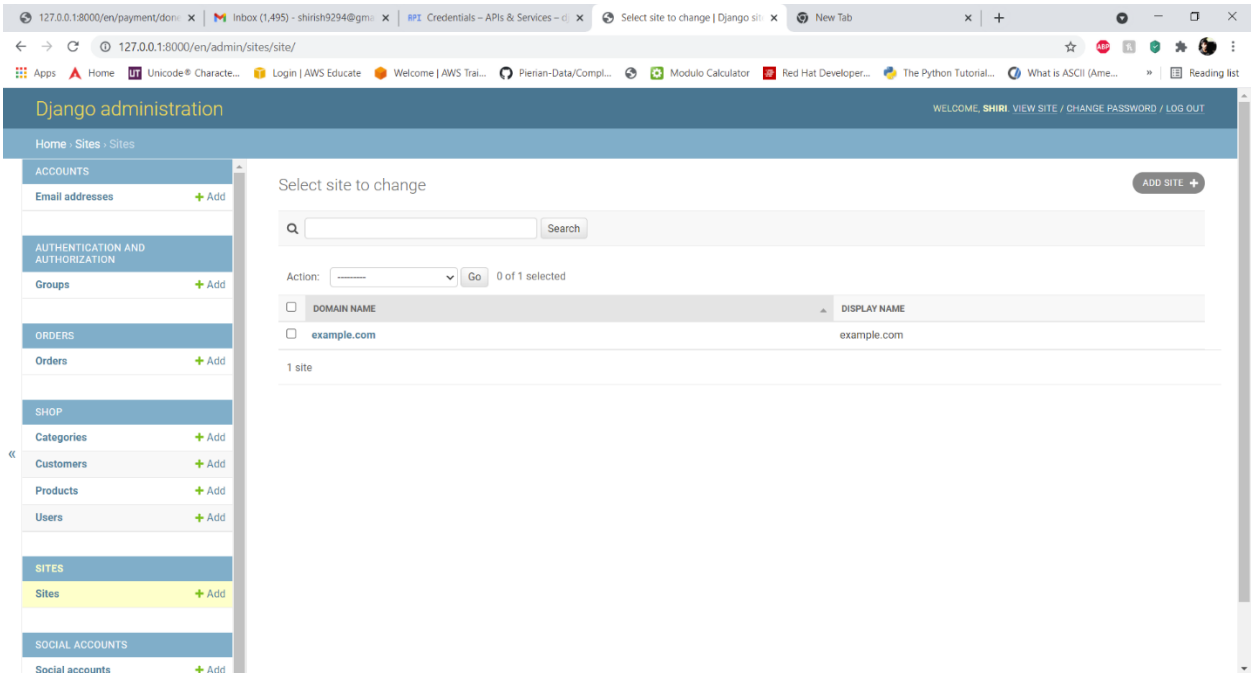


Google API configuration to run in local

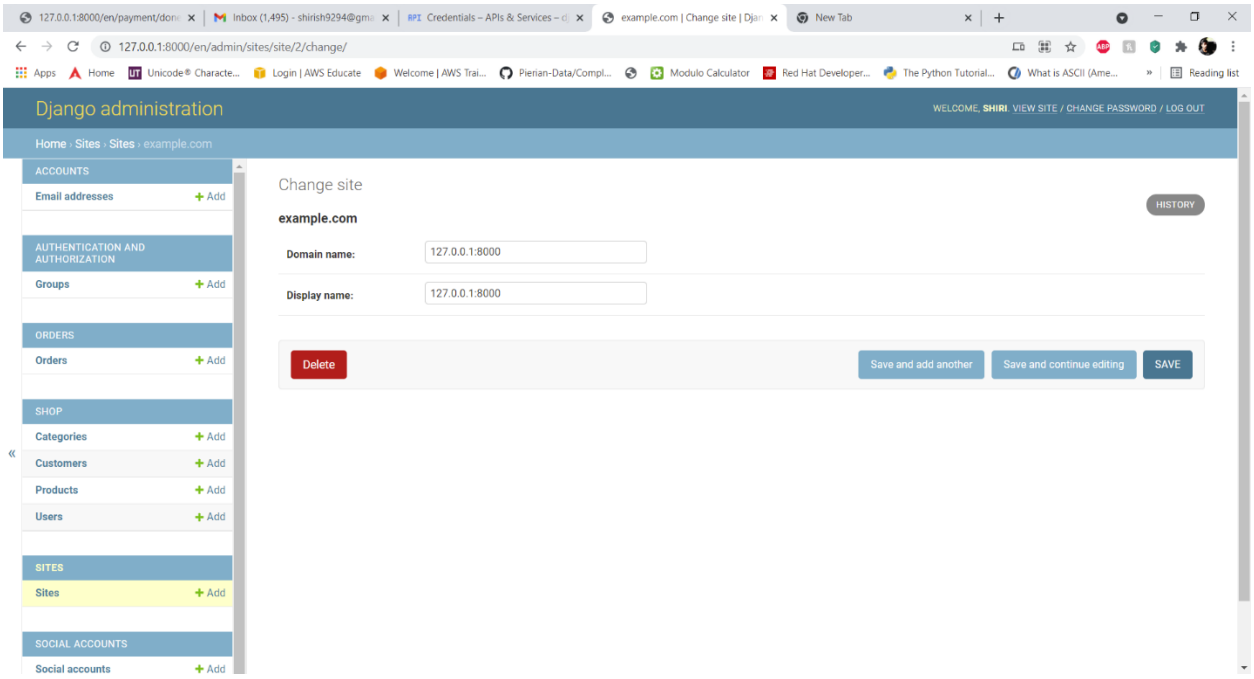
Now let’s get the Google API working in our local system



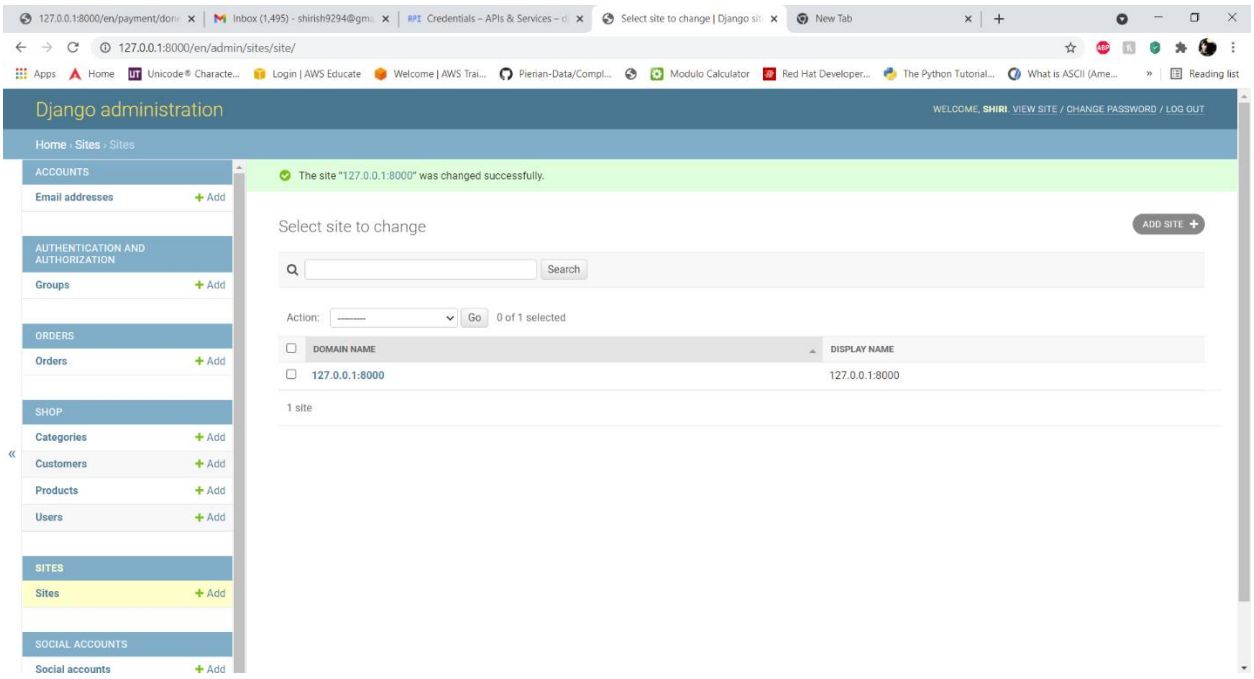
Now login to the admin panel and select sites.



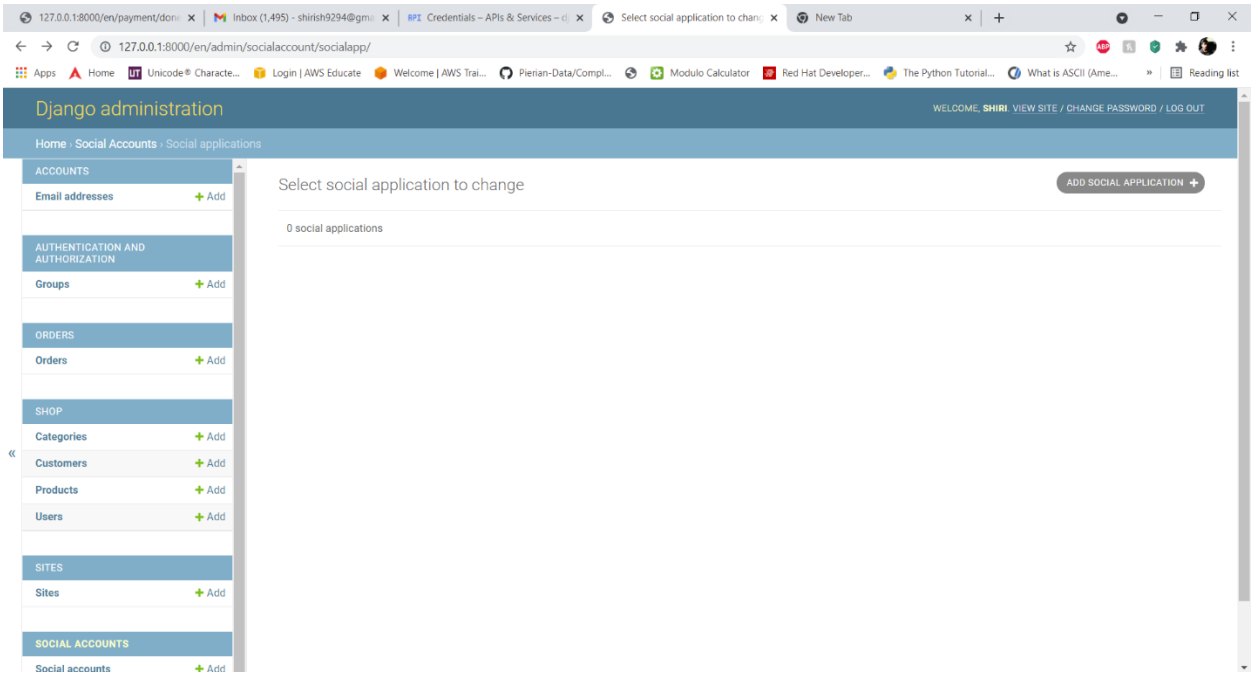
Inside site you will see something called example.com now we are going to edit its content and will change the domain name.



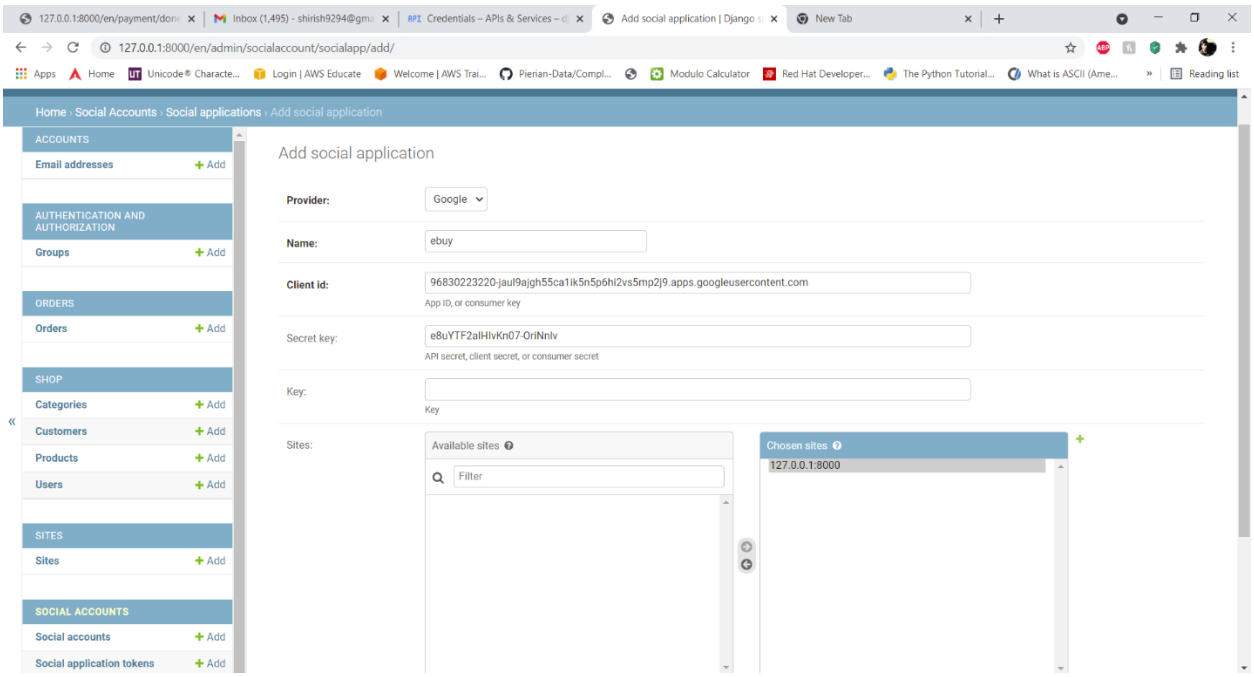
Inside we will put our local domain address 127.0.0.1:8000 in the domain name and in the display name also we are going to give the same value 127.0.0.1:8000 and we will save the configuration.



After saving it will look something like the image above with the domain name and the display name as 127.0.0.1:8000.



Now we will go back to the admin panel and under social accounts -> social applications -> add social applications.



Here we will add the social application details as follows:

Provider: Google

Name: ebuy

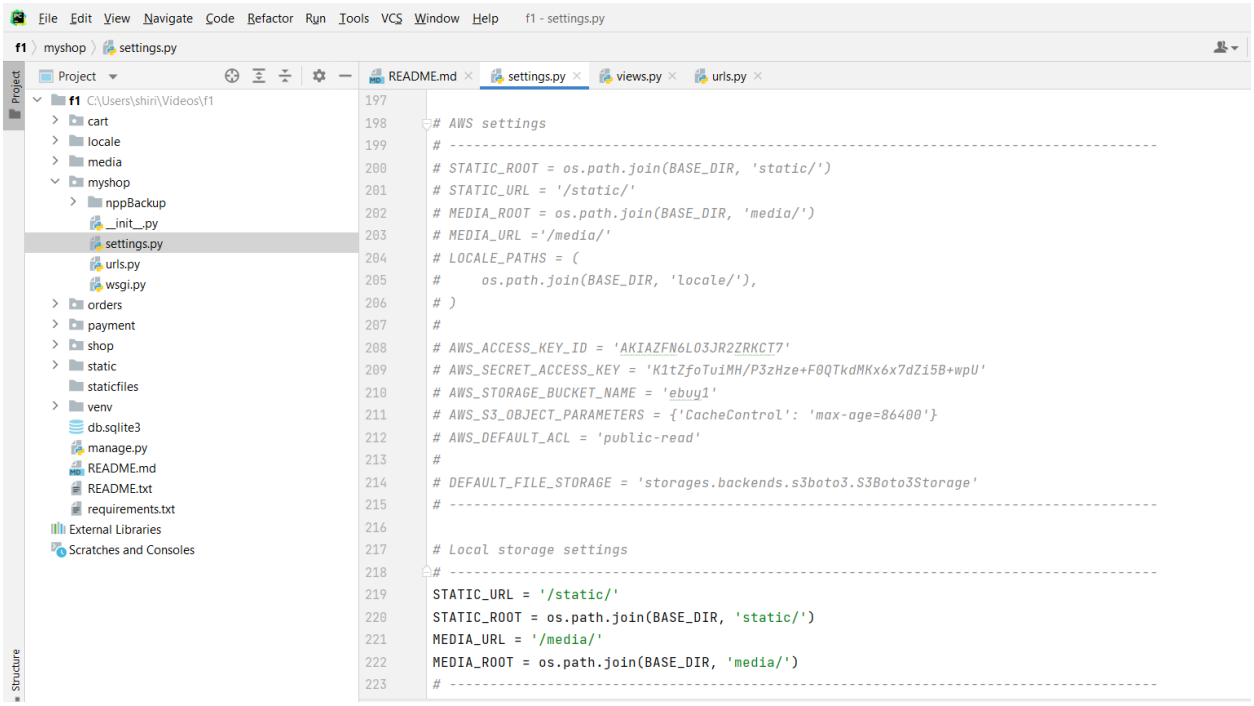
Client id: 96830223220-jaul9ajgh55ca1ik5n5p6hi2vs5mp2j9.apps.googleusercontent.com

Secret key: e8uYTF2alHIvKn07-OriNnlv

Sites: move 127.0.0.1:8000 from available sites to chosen sites

Now save your configuration and the google sign in API will work.

Choose to store media content on AWS S3 or in the local storage:



In the settings.py file we have created 2 set of configurations for storing the media files.

In the local storage settings

This is the default configuration for storing the media files in the local drive.

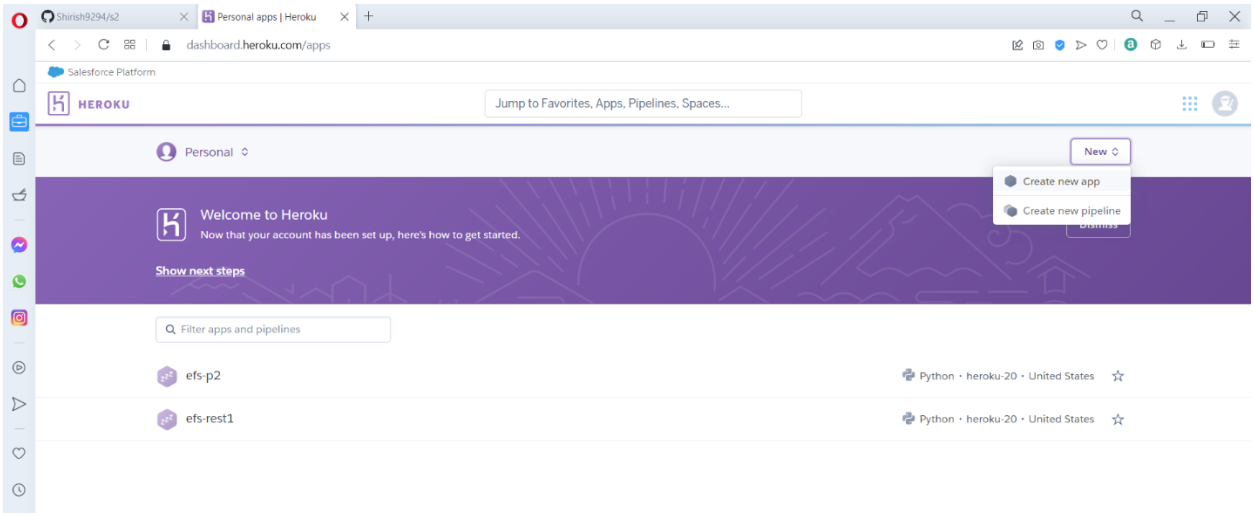
In the AWS settings

These settings are commented, but you can use these settings by uncommenting these settings and commenting the local settings.

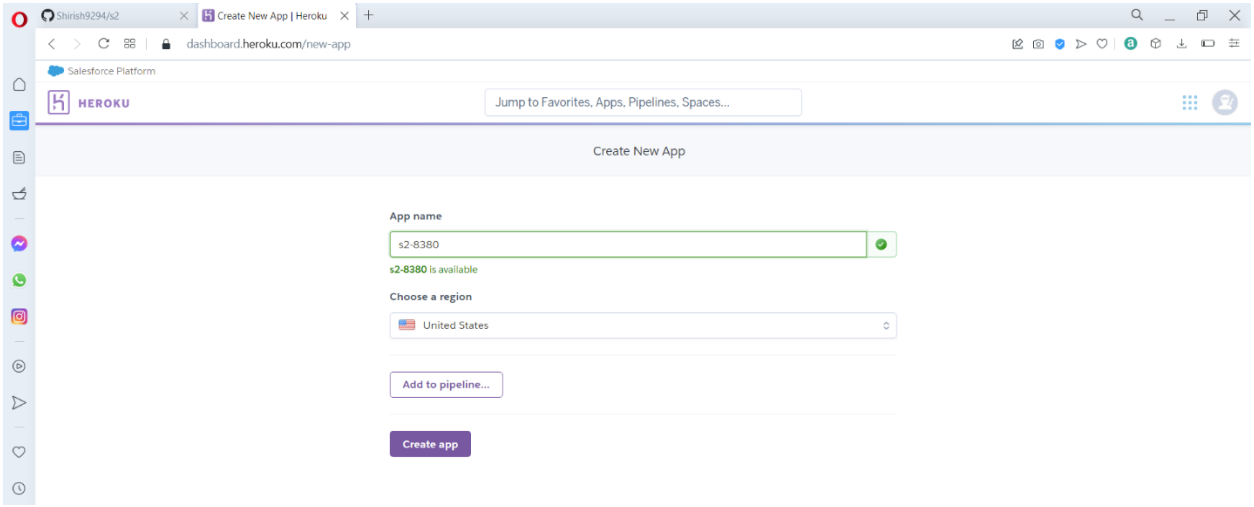
The AWS access keys and credentials in the settings are working and valid.

Deploy the code on Heroku:

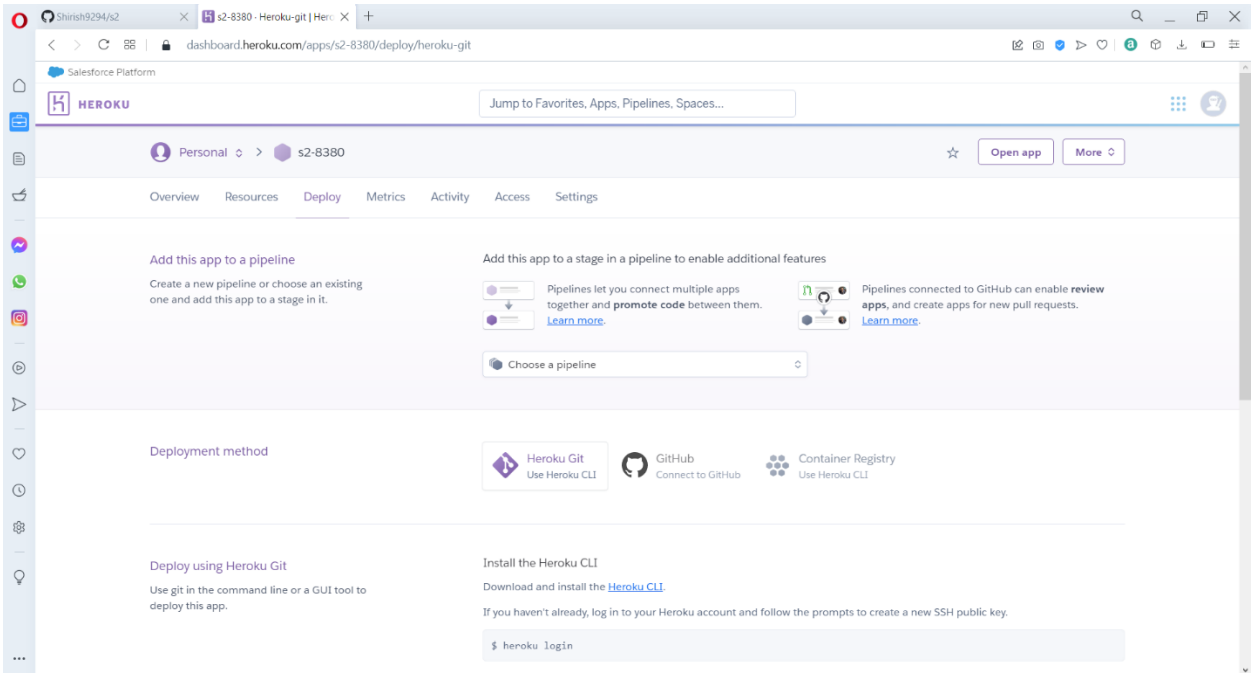
Launch the Heroku: Cloud Application Platform https://www.heroku.com if you don't already have an account; sign up here.



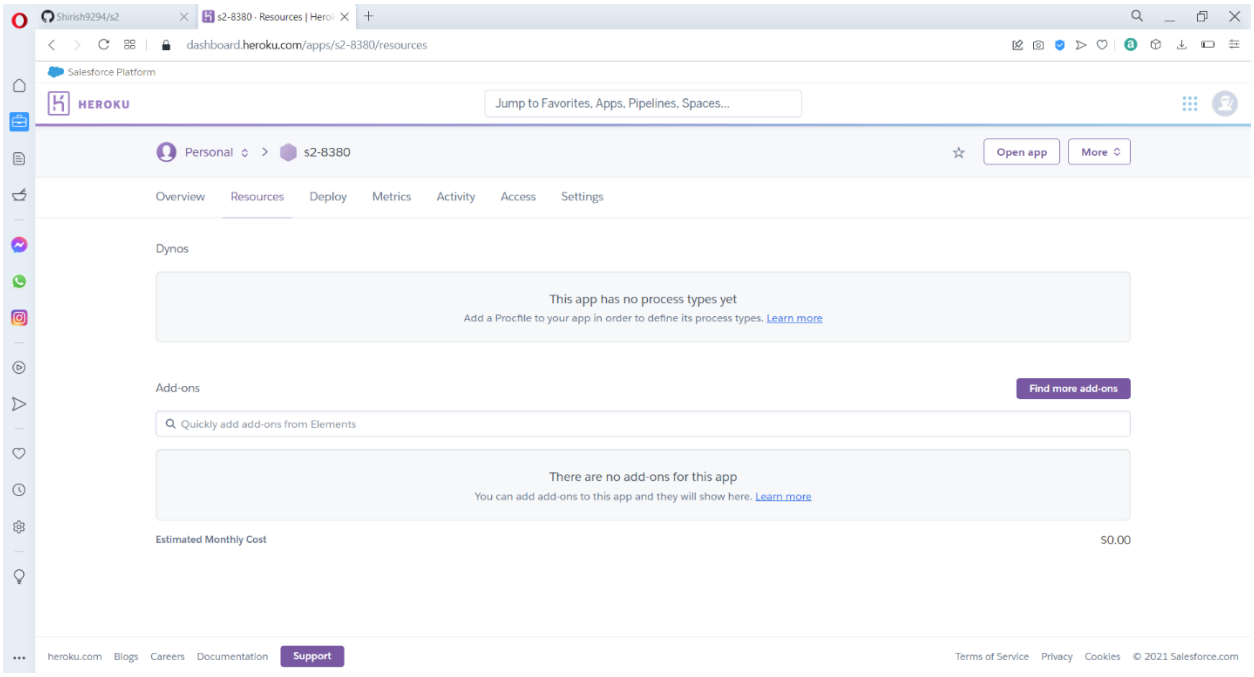
Once you login, it will take you to the https://dashboard.heroku.com/apps page where it lists all your apps. Click on the New button on the top right corner and select "Create new app" from the options.



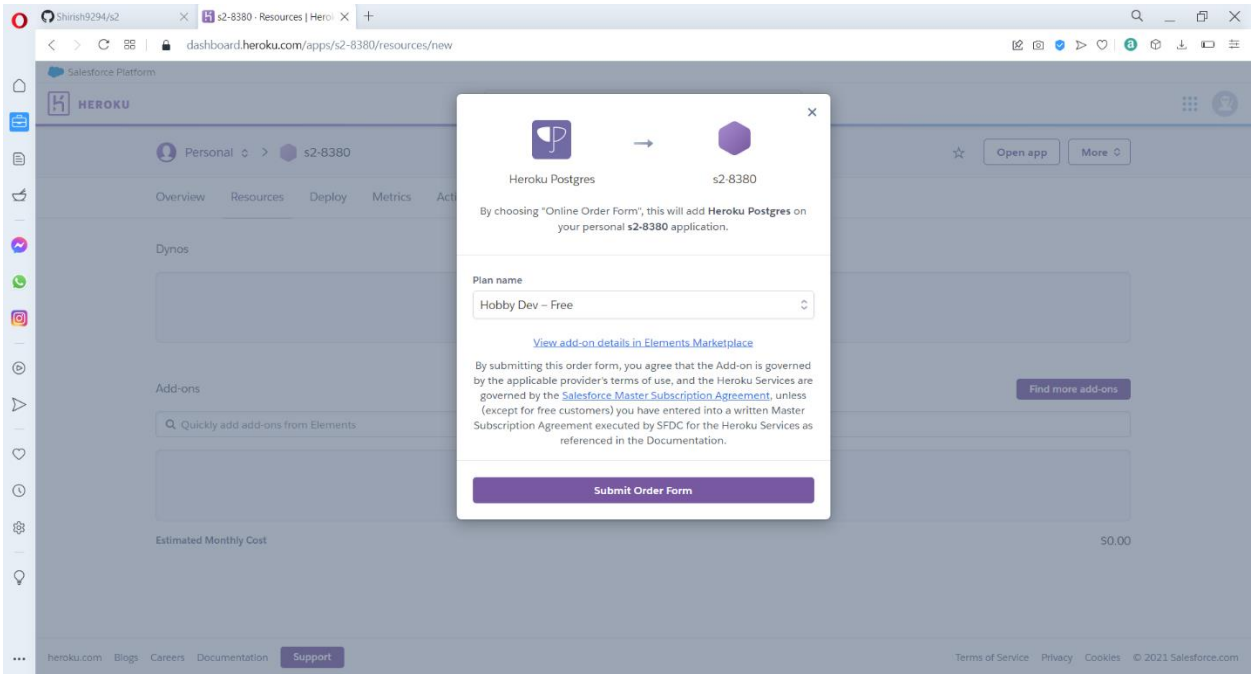
Enter your "App name", Heroku will let you know if the name is available, otherwise you might have to enter a different name, then click on create app.



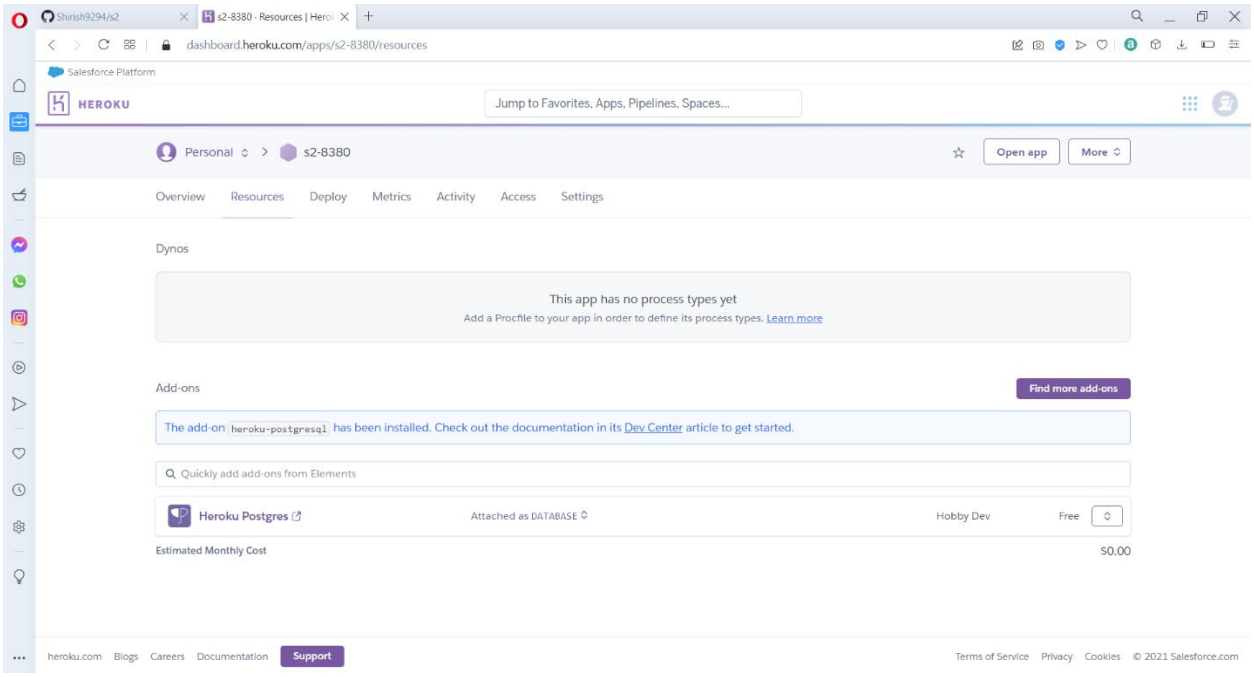
This how the application dashboard will look, now we need to create a Postgres database under the resources option.



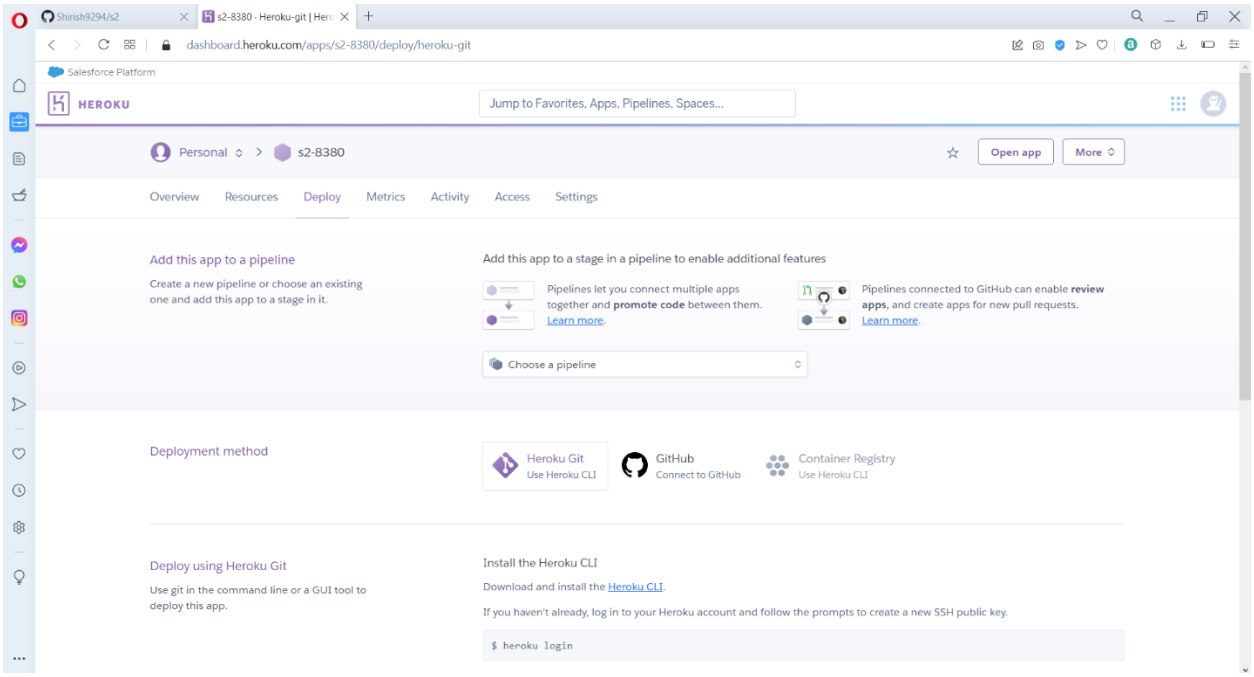
Now in the resources, under addon, type Postgres and search for that particular addon.



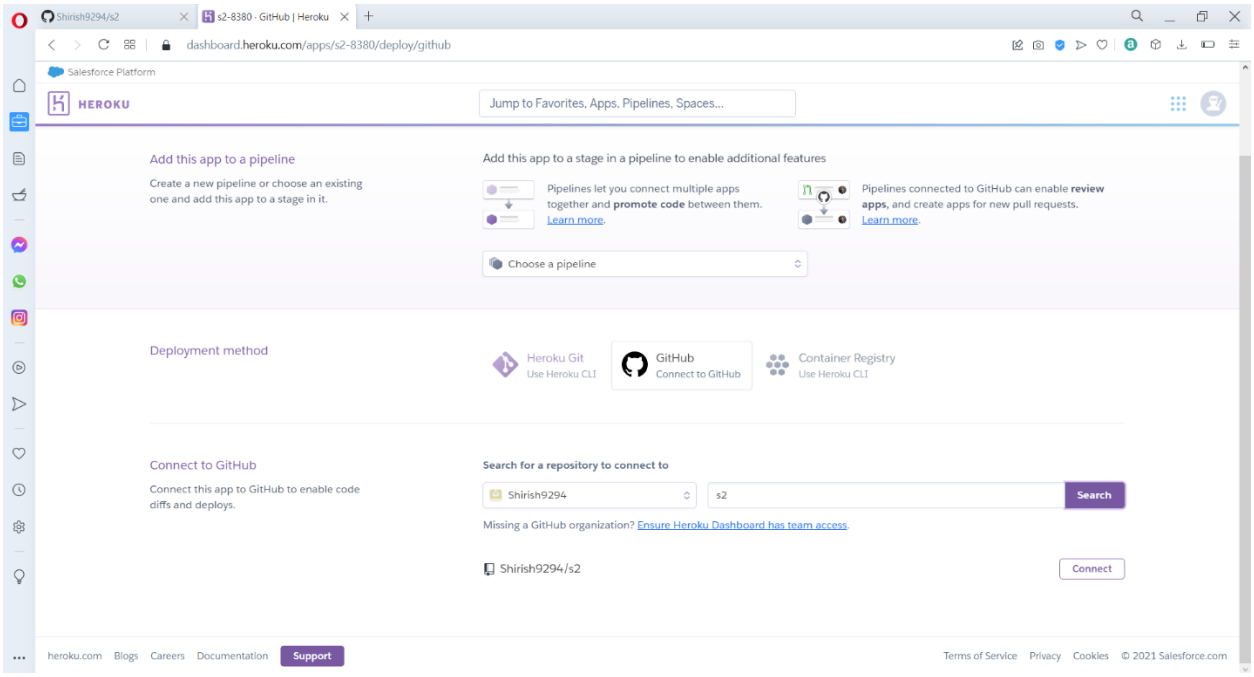
After the required addon is found, click on the submit order form, to place the order and to add it to the app.



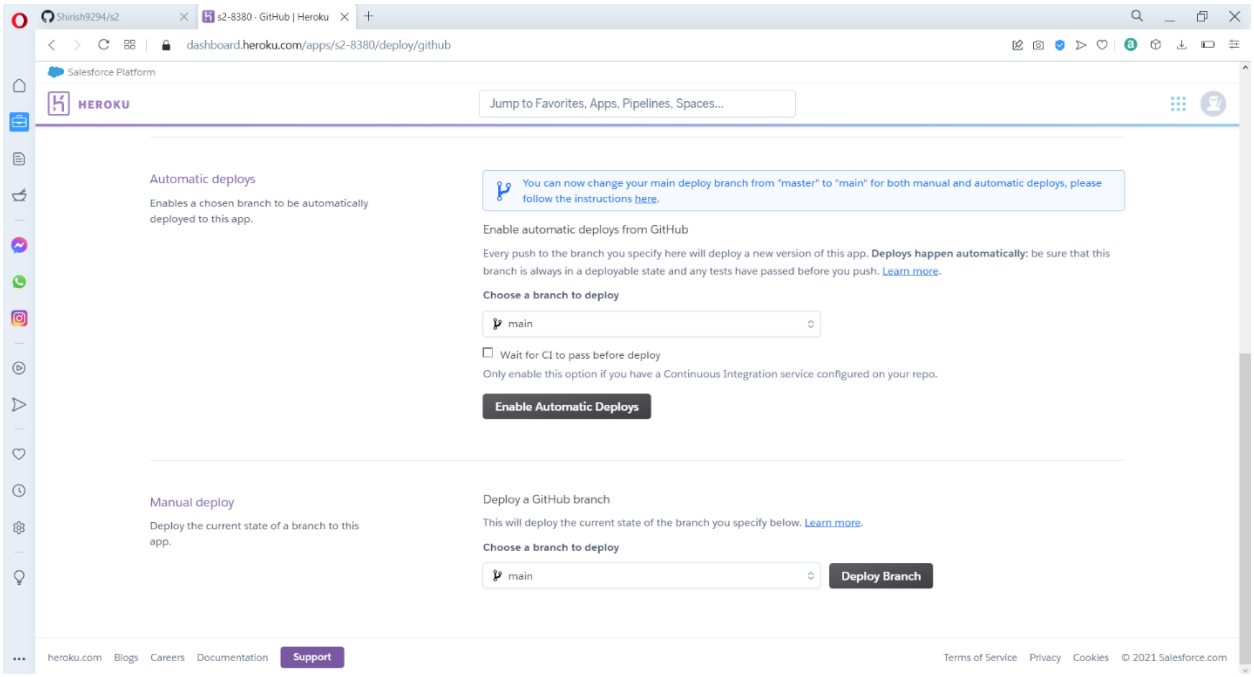
After the order is placed successfully, we can see that Postgres is attached as database under addons.



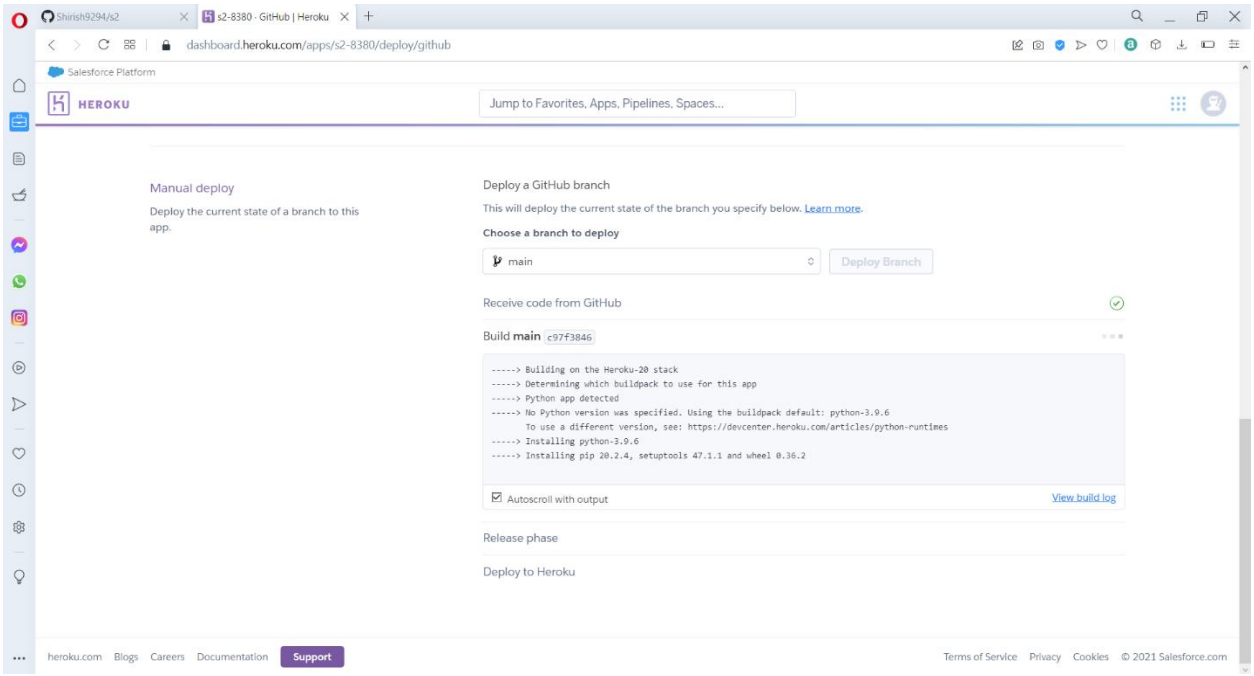
After adding Postgres, now we need to the deploy option, where Heroku will provide us with various deployment method, and we are going to choose the GitHub method.



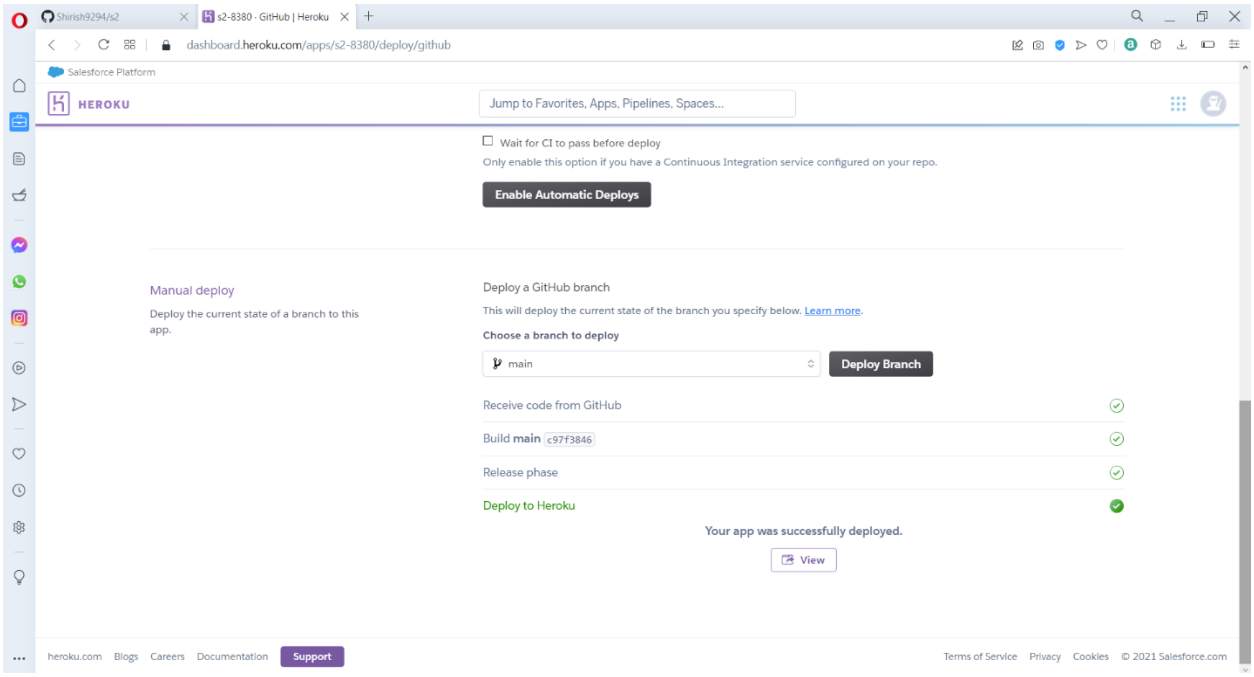
Now select the GitHub option and under that type the repository that you want to deploy and click on connect.



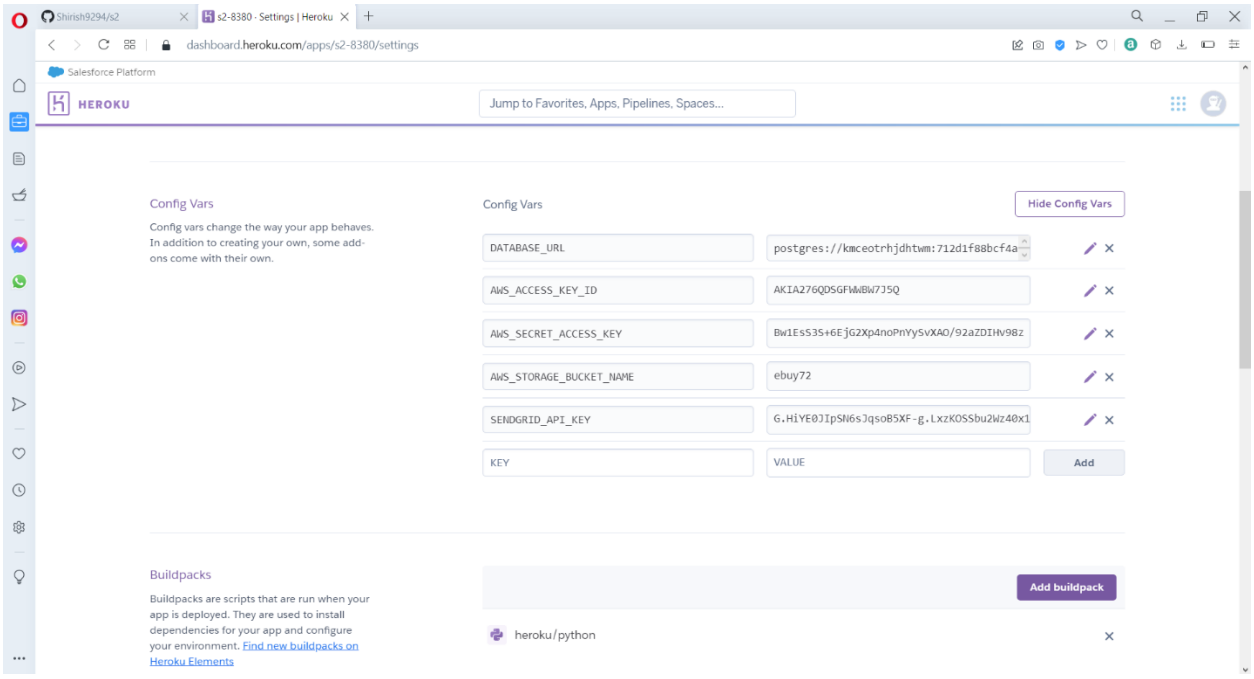
Now in the manual deploy option, select the main branch, then click on deploy branch.



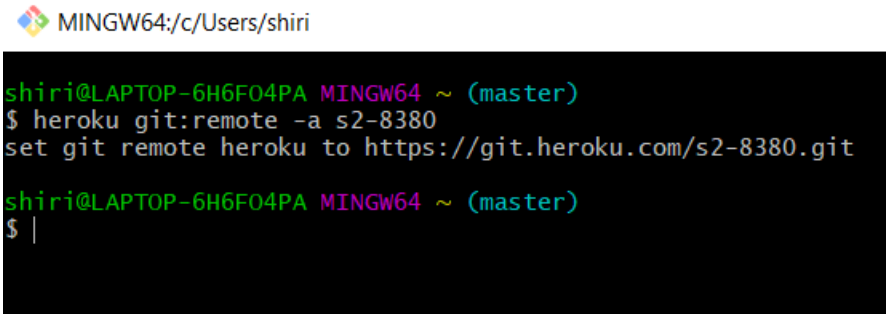
The deployment process will start and Heroku will install all the requirement packages specified in the requirement.txt file.



After the deployment is finished the deploy to Heroku will highlight to green to indicate that the app was deployed successfully. Now we need to go to the app settings option in Heroku and add config variables, which are keys that are going to be utilized by the API's in the web app.



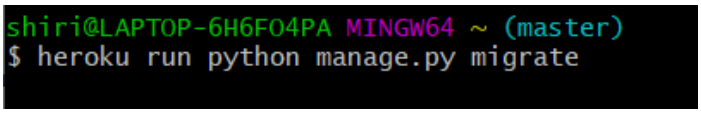
Under config vars we will add the necessary keys like the aws access key, aws secret access key, aws storage bucket name and the SendGrid API key.



Now need to establish connection from our local bash to the Heroku app by running the command **Heroku login**

Then we need to run the command **Heroku git: remote -a s2-8380**

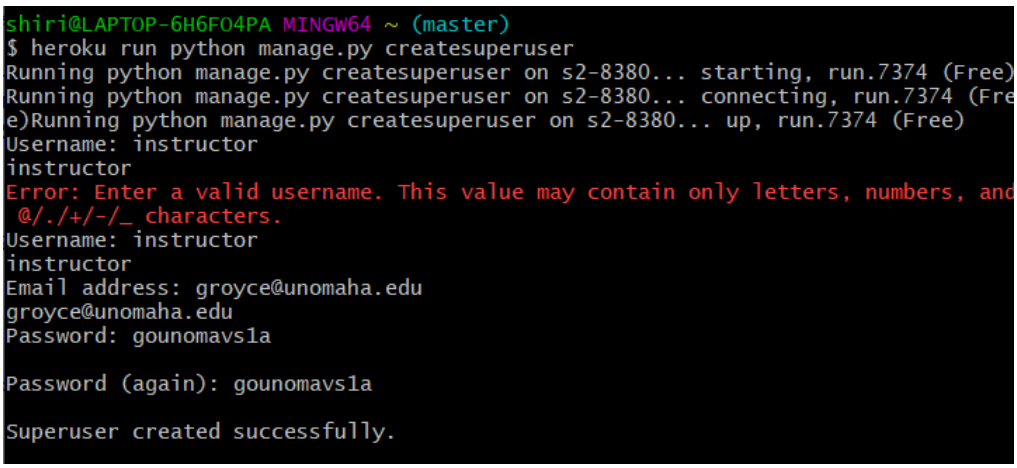
To access the app.



Then we need to run the command

Heroku run python manage.py migrate

To migrate the data base.

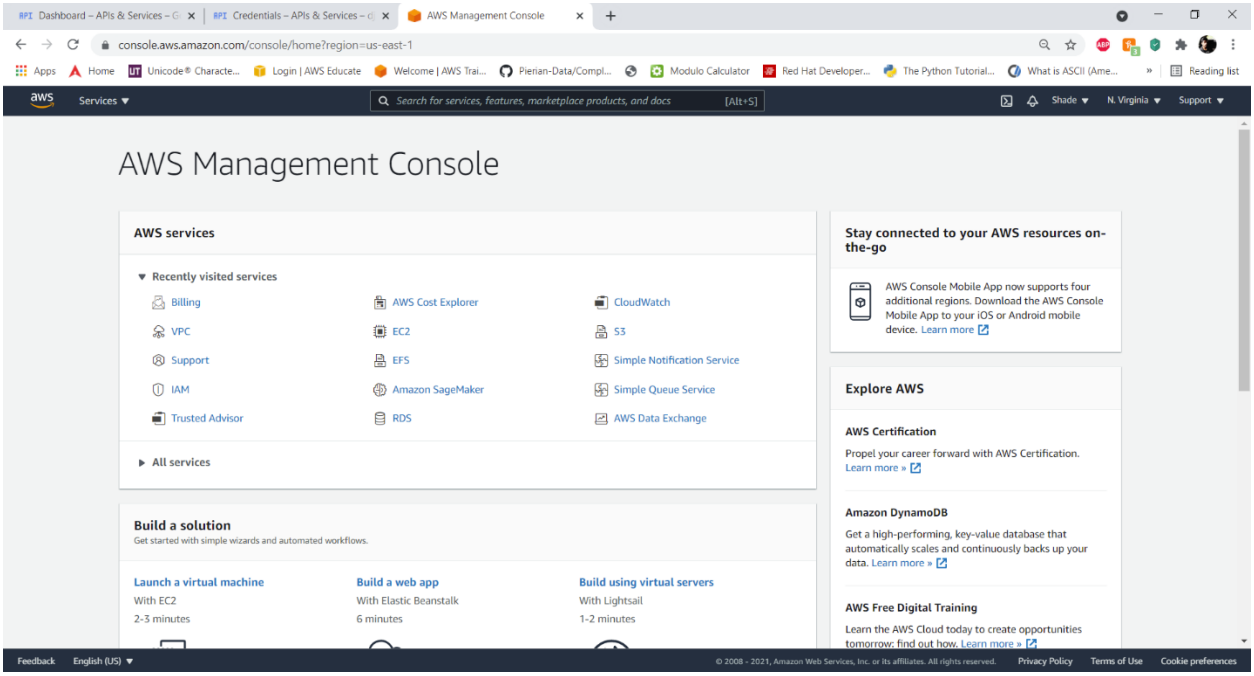


Then we need to create a superuser by running the command

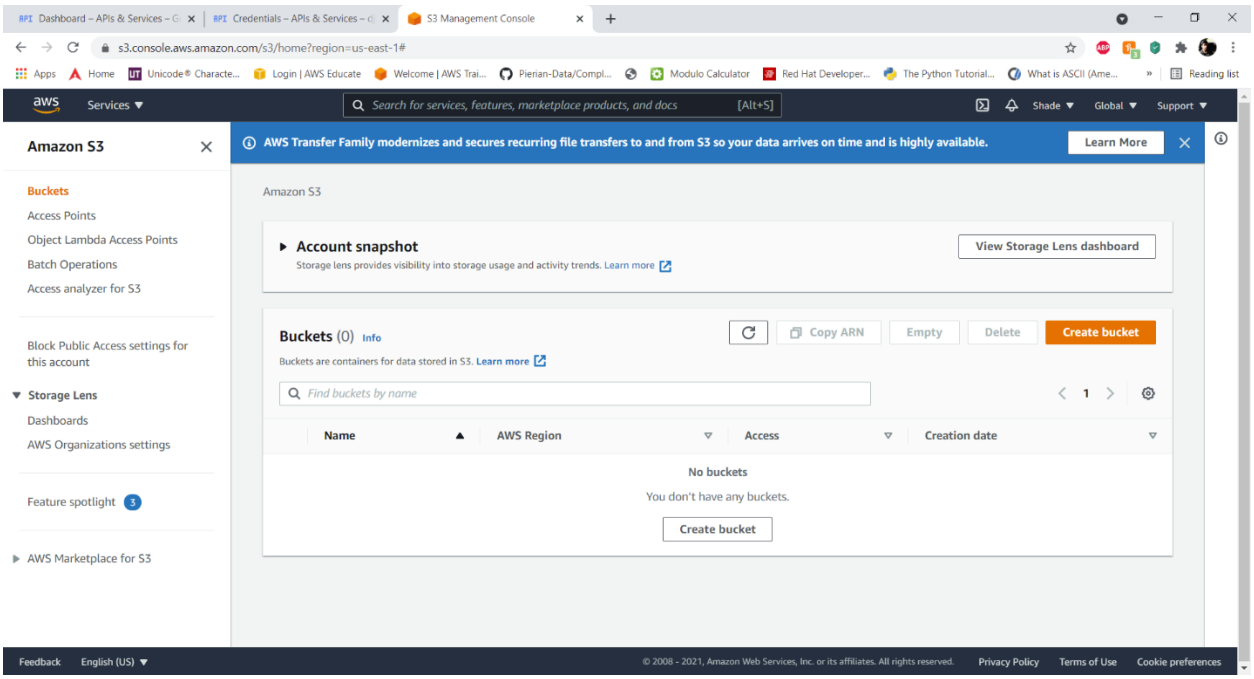
Heroku run python manage.py createsuperuser.

Then we need to enter the username, email and the password to create the super user.

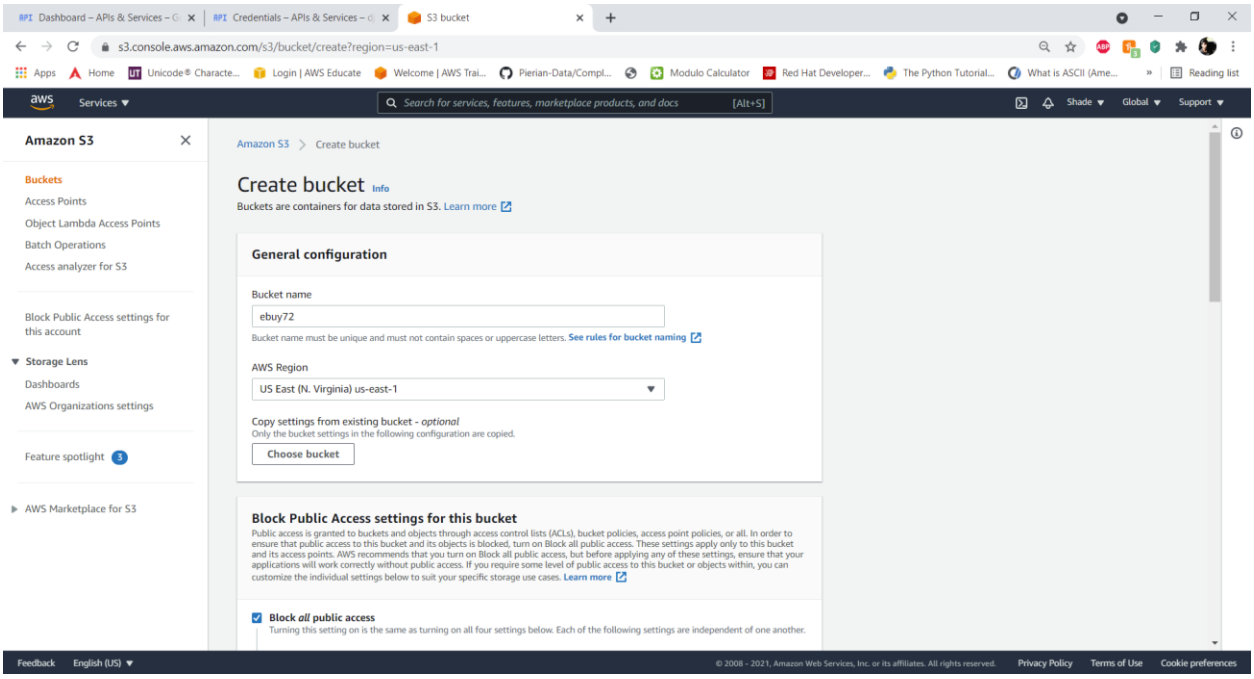
AWS S3 configuration:



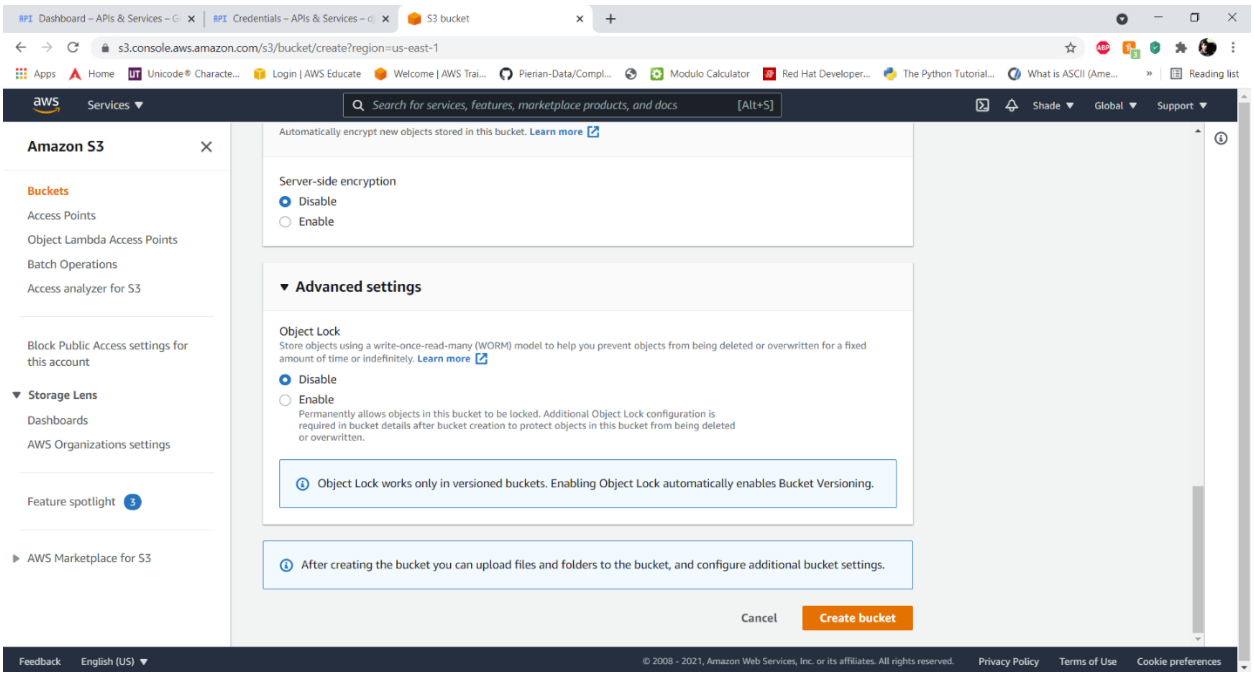
Login to the AWS console.



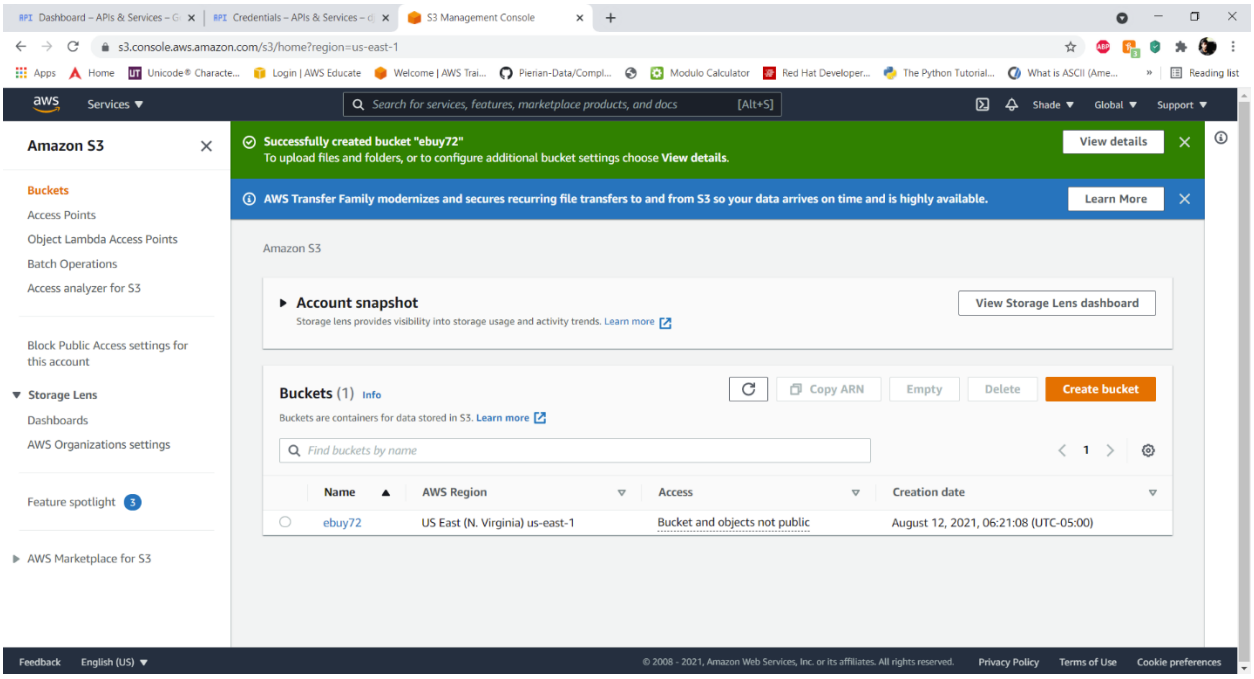
Go to the S3 bucket service and click on create bucket.



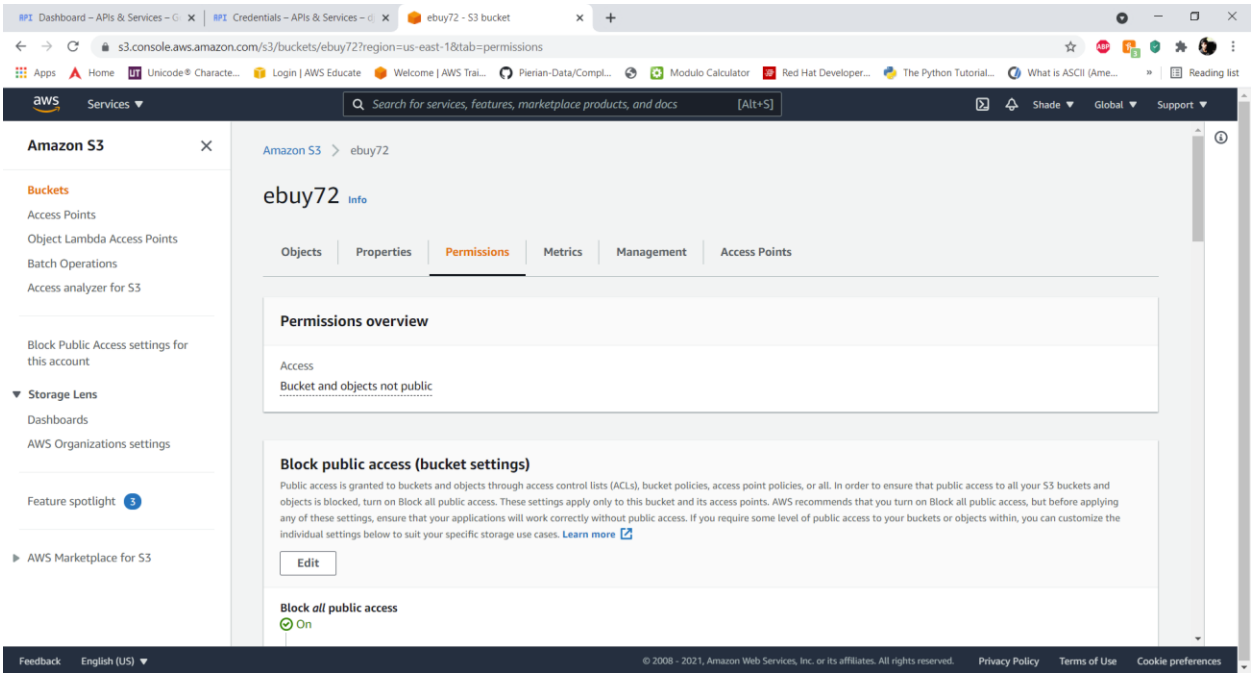
In the create bucket option, enter the bucket name and select the aws region.



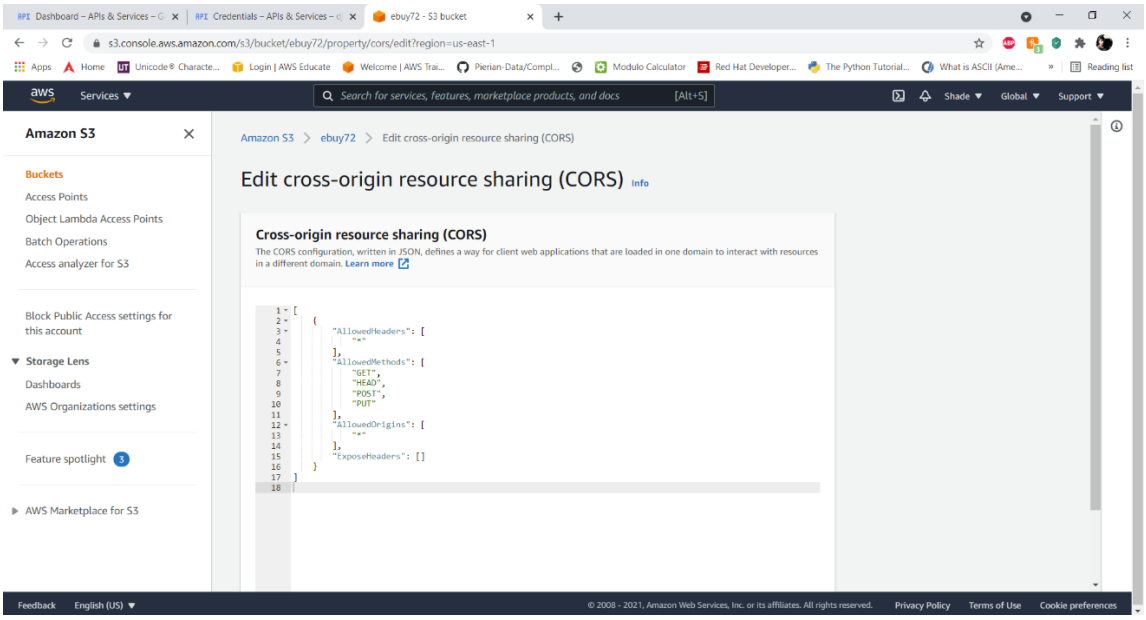
Disable the server-side encryption and click on create bucket.



Now that the bucket is created, we need to edit the access permissions.

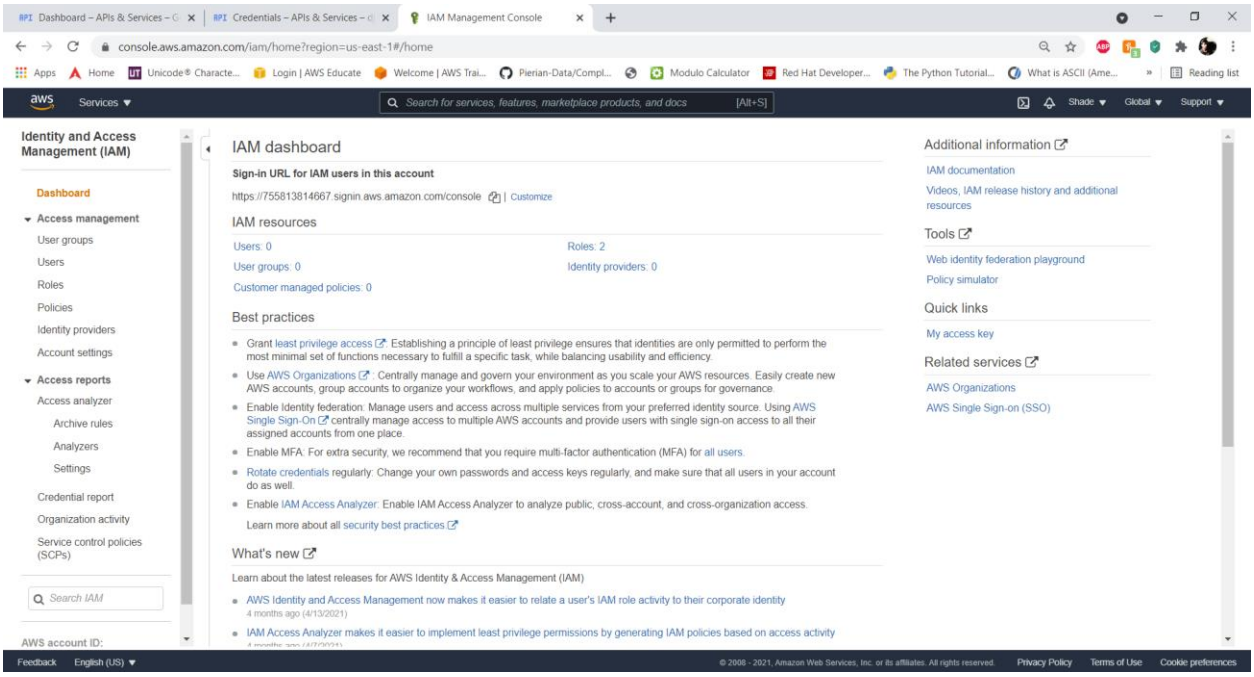


We are going to go to the permissions section in the bucket and will untick block all public access, we need to allow public access for this bucket to establish the connection.

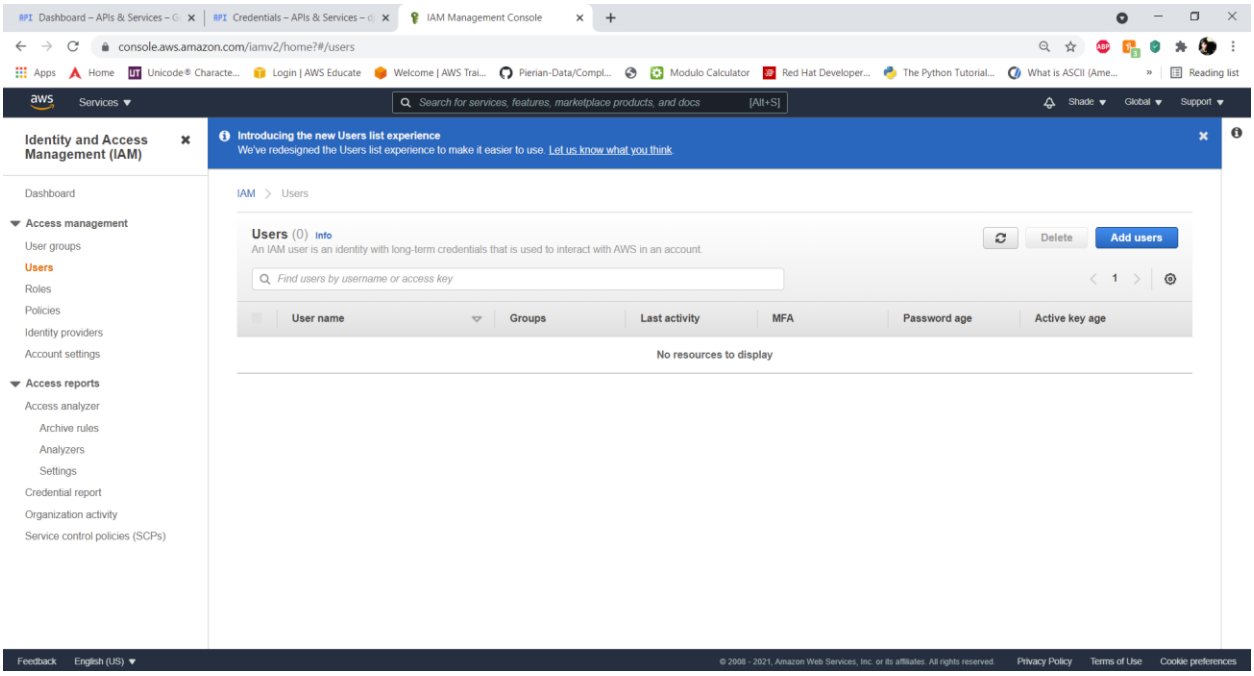


Now in the cross-origin resource sharing add:

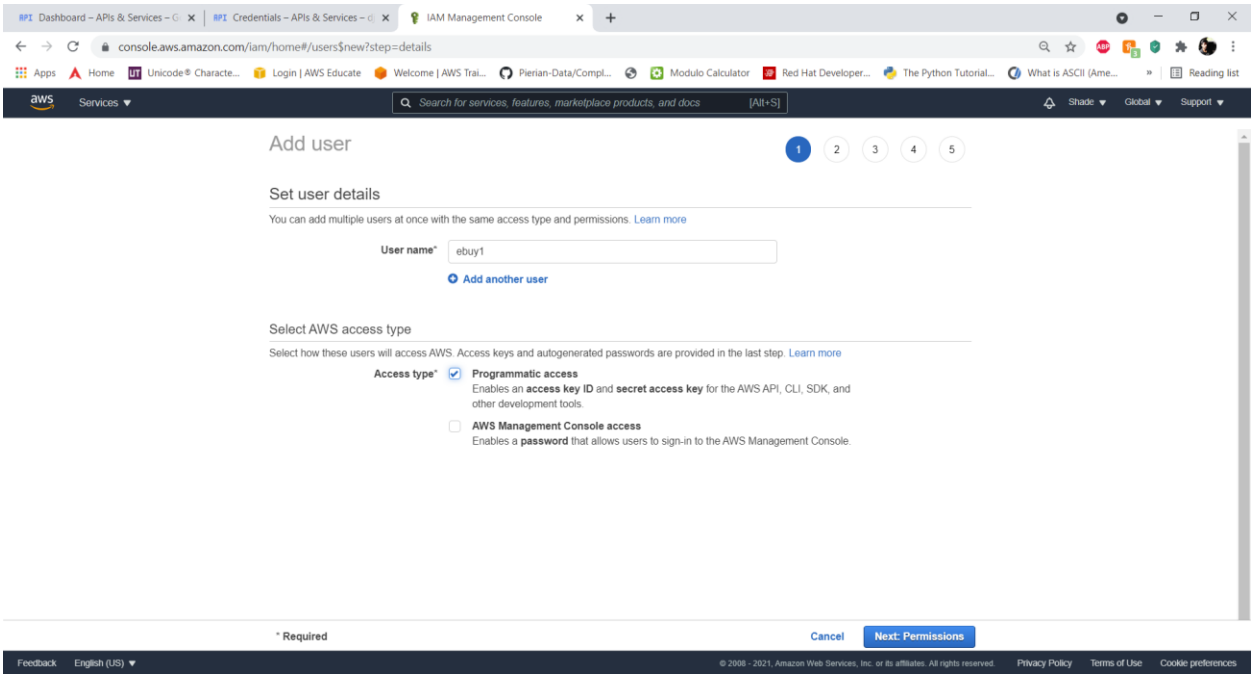
```
[
  {
    "AllowedHeaders": [
      "*"
    ],
    "AllowedMethods": [
      "GET",
      "PUT",
      "POST",
      "HEAD",
      "DELETE"
    ],
    "AllowedOrigins": [
      "*"
    ],
    "ExposeHeaders": []
  }
]
```



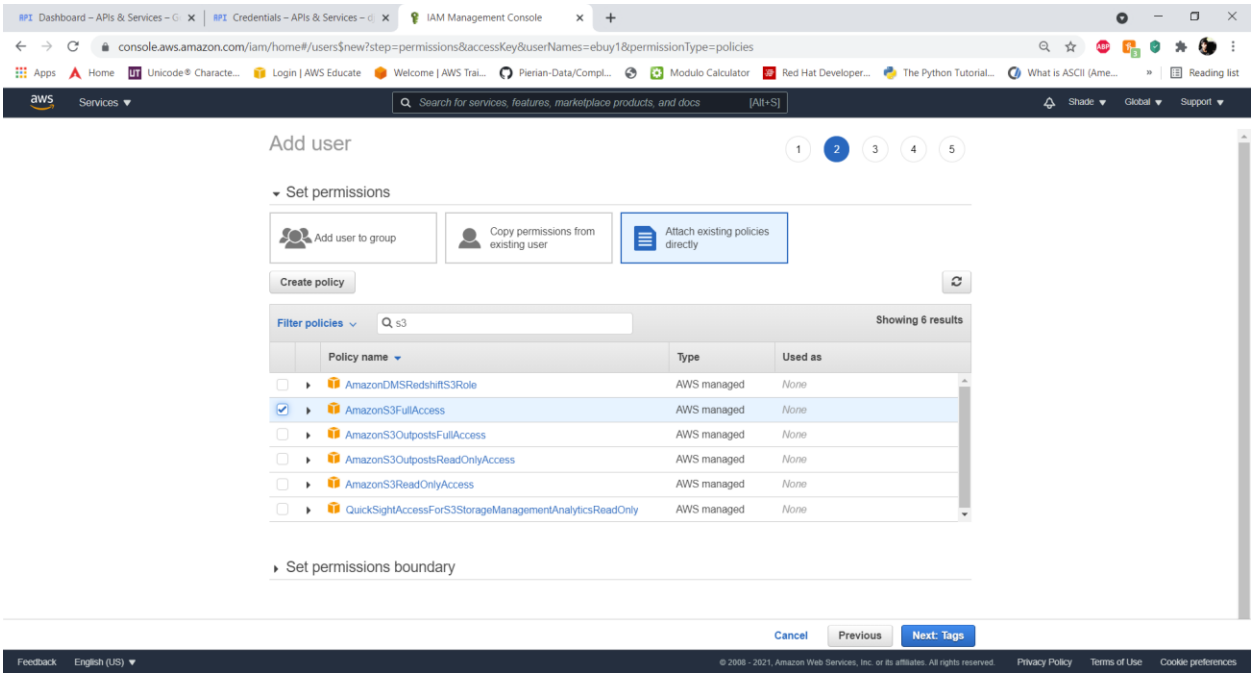
Now we need to go to the IAM dashboard and create a new user.



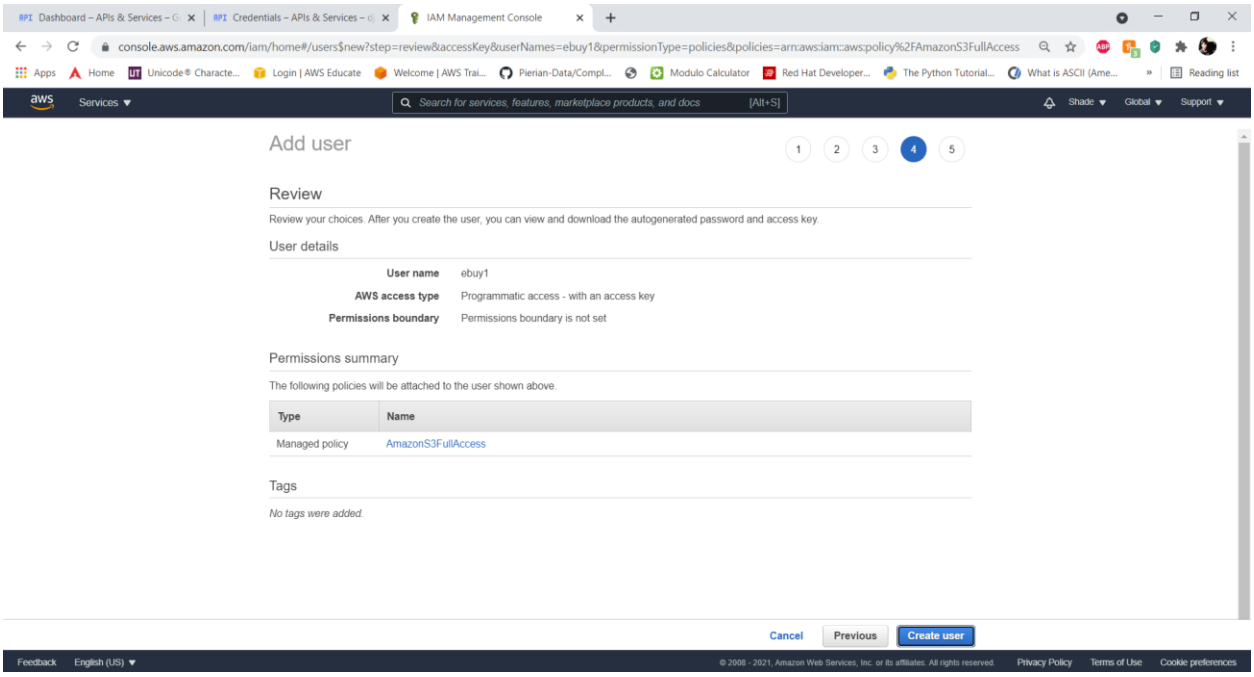
Go to Users -> Add users.



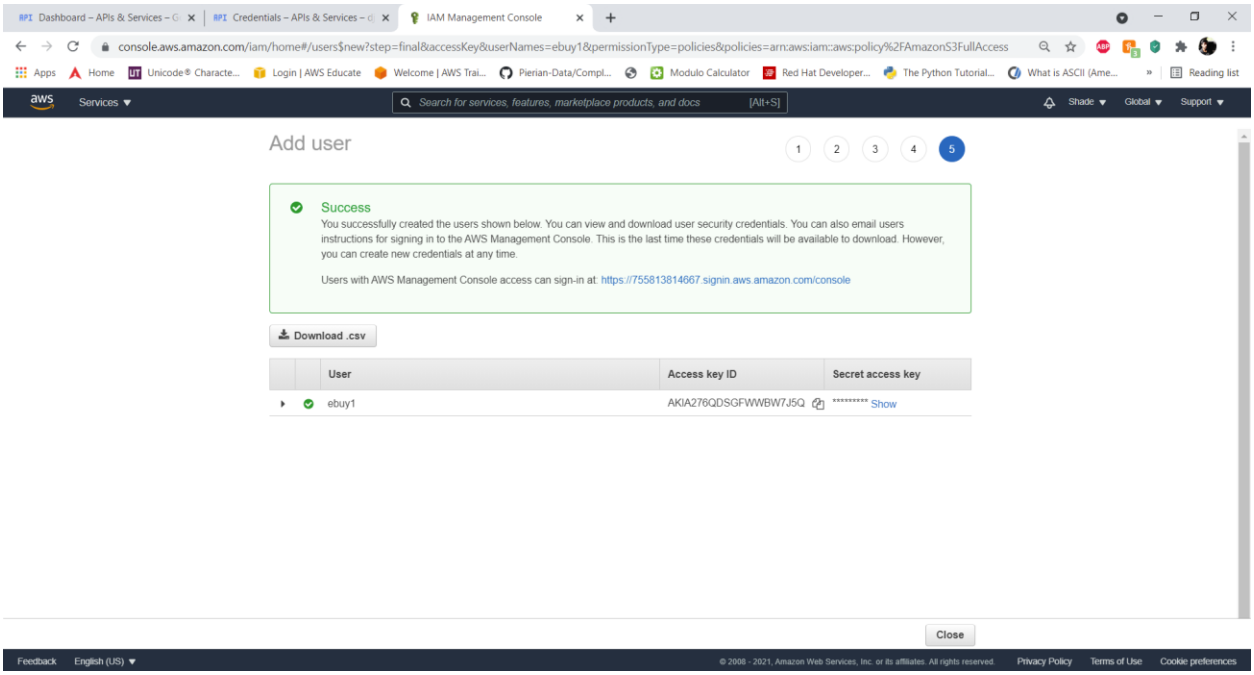
In the Add user page, enter a name and click on next: permissions.



In permission we are going to add the AmazonS3FullAccess permission to the user, to by using this user AWS key we can perform read and write operations on the bucket.



After we review the user details, we will create the user.



Now when the user is created, we will copy the Access Key ID and the Secret access key ID and will net this value in the Heroku environment variables.

Links and credentials

GitHub Link:

https://github.com/ISQA-Classes/su2021team2-app.git

Test Scripts:

https://github.com/ISQA-Classes/su2021team2-test.git

Heroku Link:

https://s2-8380.herokuapp.com

Login Credentials:

ID: instructor

Email: groyce@unomaha.edu

Password: gounomavs1a