



E-Buy Shopping

Assignment-4



ISQA 8380

Enterprise Architecture and System Integration

Team – 2:

Shirish Srinivasa

Kavya Raviprakash

Luke Henkenius

Akhila Pillalamarri

Index

1. Team Members and Roles	4
2. Business use and description of the web application	5
3. Use Case Diagram	7
4. Data Flow Diagram	8
5. Data Model	9
6. System Requirements	10
Hardware Requirements	10
Software Requirements	10
Heroku Requirements	10
7. Screenshots	11
Admin Panel	11
Web application	22
8. API	34
Google	34
SendGrid Email	39
Currency	42
Twilio	45
Rosetta	48
Braintree	51
Custom REST API	53

9. Download code from GitHub and run it locally	56
Google API configuration to run in local	64
Choose to store media content on AWS S3 or in the local storage	68
10. Deploy the code on Heroku	69
AWS s3 configuration	76
11. Links and credentials	62
GitHub Link	83
Heroku Link	83
Test Scripts	83
Login Credentials	83

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:

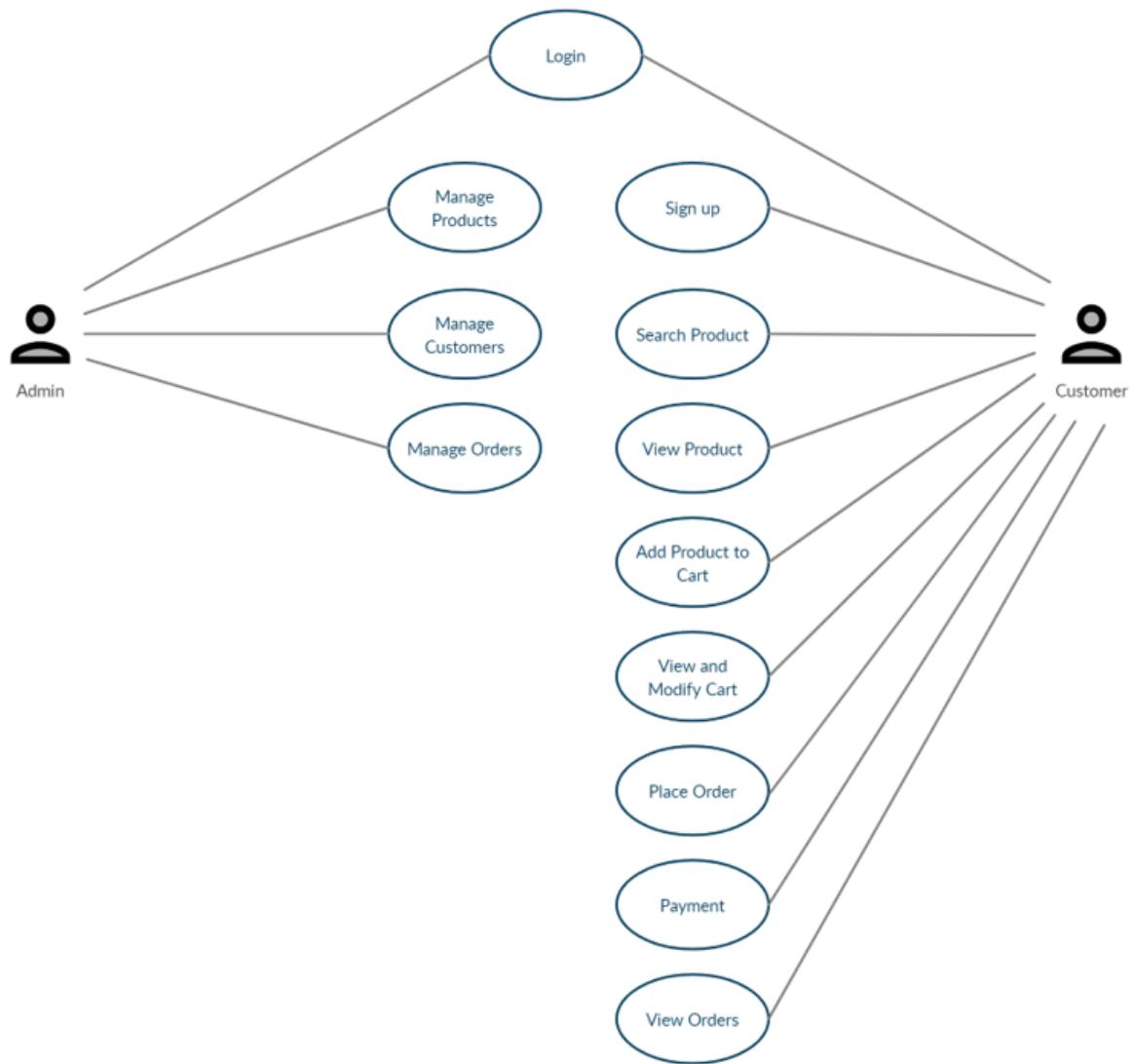
- Admin can login to the web application
- Admin can manage registered customers and also add new users to the system if needed
- Admin can View, Add, edit, and delete product categories.
- Admin can View, Add, edit, and delete product
- Admin can View, Add, edit, and delete orders and order details

Customer Features:

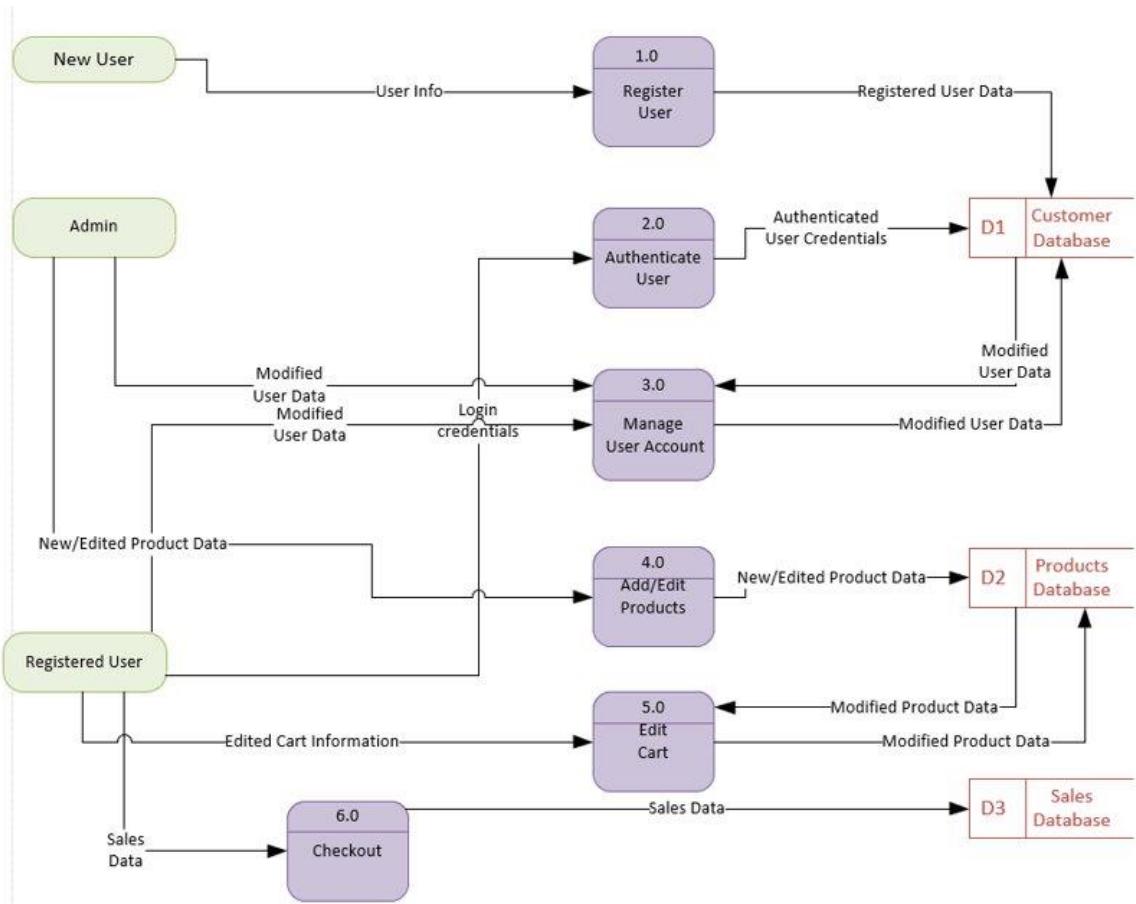
- Customer can sign up to the web application through filling a signup page or by authenticating through google sign up.
- Customer once registered can login to the web application.
- Customer can search products based on categories.

- 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.

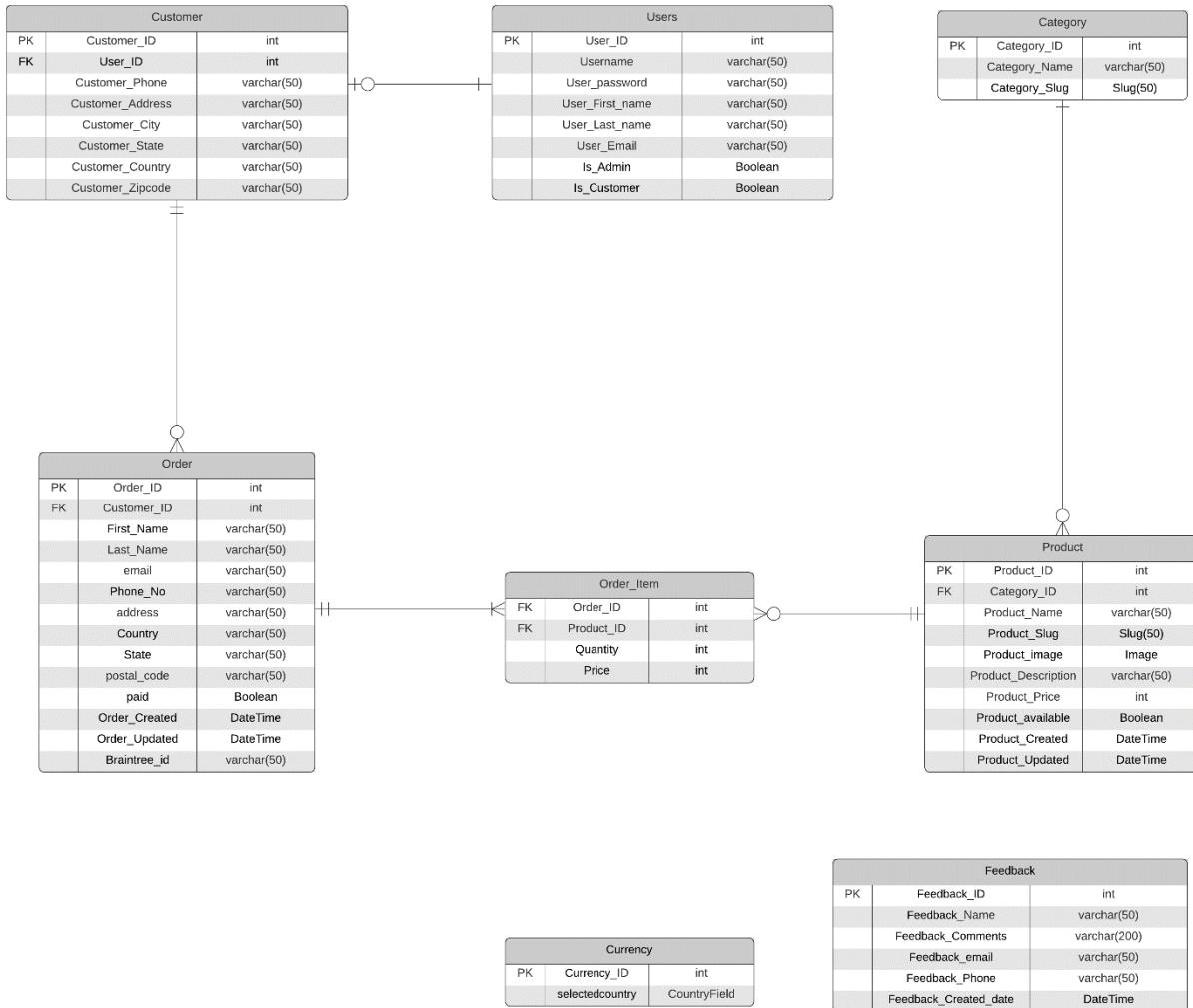
Use Case Diagram



Data Flow Diagram



Data Model



System Requirements:

Hardware Requirements:

Minimum requirements:

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

Maximum requirements

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

Software Requirements:

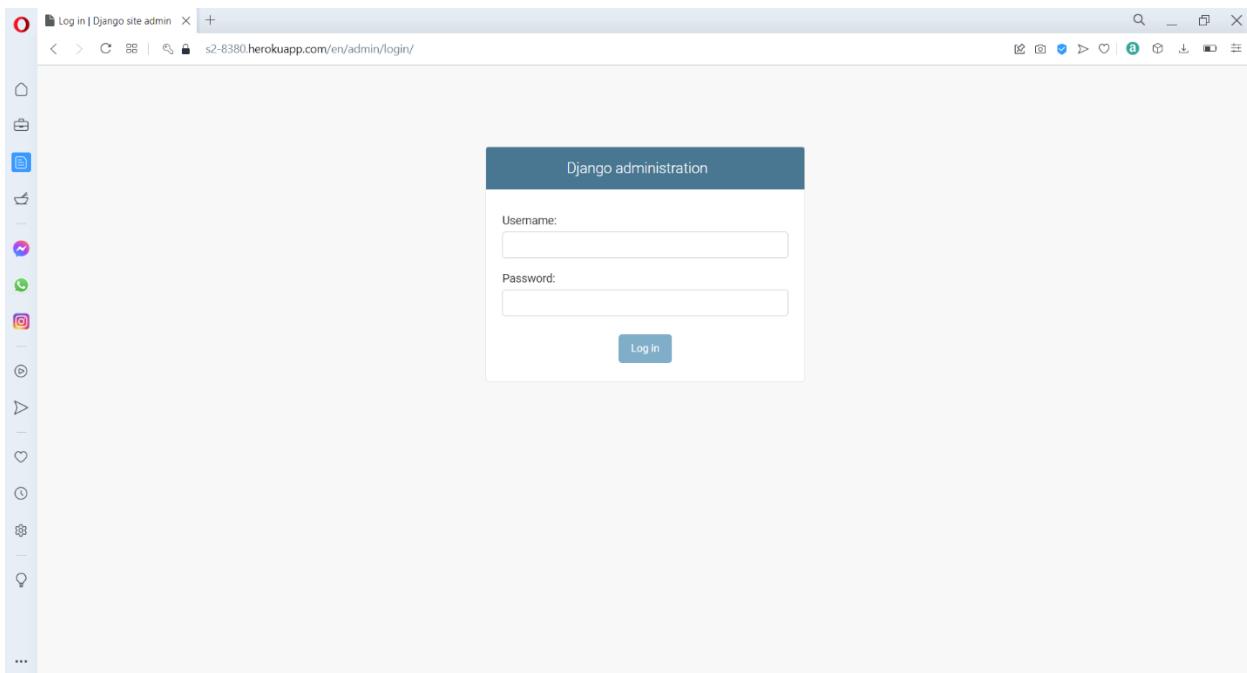
- PyCharm
- Python 3.9
- GitHub Desktop
- Windows 7 or later
- Web browser: Chrome or Firefox

Heroku Requirements:

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

Screenshots

Admin



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

A screenshot of the Django site administration panel. The title bar says "Site administration | Django". The main area shows sections for ACCOUNTS, AUTHENTICATION AND AUTHORIZATION, ORDERS, SHOP, SITES, and SOCIAL ACCOUNTS, each with "Add" and "Change" buttons. On the right, there's a "Recent actions" sidebar listing recent operations such as adding products and social applications, and viewing sites.

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

This screenshot shows the 'Select user to change' page in the Django administration interface. The left sidebar lists various models: ACCOUNTS (Email addresses), AUTHENTICATION AND AUTHORIZATION (Groups), ORDERS (Orders), SHOP (Categories, Customers, Products, Users), SITES (Sites), and SOCIAL ACCOUNTS (Social accounts, Social application tokens). The 'Users' model is selected. The main area displays a list of users: 'shirish', 'instructor', and 'shiri'. A search bar at the top right says 'WELCOME, SHIRI | VIEW SITE / CHANGE PASSWORD / LOG OUT'. A 'ADD USER +' button is located in the top right corner.

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

This screenshot shows the 'Add user' page in the Django administration interface. The left sidebar is identical to the previous screenshot. The main area has fields for 'Password' (a text input field), 'Last login' (Date: Today, Time: Now), and 'Superuser status' (a checkbox). Below these are sections for 'Groups' (a list with a '+' button) and 'User permissions' (a scrollable list of permission codes like 'account | email address | Can add email address'). At the bottom are fields for 'Username' (text input, required 150 characters or fewer), 'First name' (text input), and 'Last name' (text input).

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

This screenshot shows the Django admin 'Change user' page for a user named 'shirish'. The left sidebar navigation includes 'Home', 'Shop', 'Users', 'Email addresses', 'Groups', 'Orders', 'Categories', 'Customers', 'Products', 'Social accounts', 'Social application tokens', and 'Social applications'. The main content area displays the user's password ('!dkAcaVP6WZ1nlp0M1Y6R2KA0hc7lwToma'), last login (2021-08-12 at 18:25:38), and the option to change superuser status. It also shows the user belongs to no groups and has various user permissions related to email accounts.

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 screenshot shows a confirmation dialog in the Django admin titled 'Are you sure?'. It asks if the user wants to delete the selected user, stating that all related items will be deleted. The 'Summary' section lists one user ('shirish') and their associated objects: Email address (shirish94@gmail.com), Social account (shirish), and Social application token (ya29.a0ARdAM_dIIKHSxU6iDzbeYqGjswBrZjelzEHFIMGebRsaSrvoLcNnRRVxTusAqGUzzwLerAvM/GaTzD13rkstaLcZHQCTwMULb6Eo_qjTn16yhB5erKHQAO6TqD3vR0gjSO_XQxaOfjoLAutrnZNpE_Q). At the bottom are 'Yes, I'm sure' and 'No, take me back' buttons.

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 screenshot shows the Django administration interface for the 'Categories' model. The left sidebar lists various models: ACCOUNTS (Email addresses), AUTHENTICATION AND AUTHORIZATION (Groups), ORDERS (Orders), SHOP (Categories, Customers, Products, Users), SITES (Sites), and SOCIAL ACCOUNTS (Social accounts, Social application tokens). The 'Categories' model is currently selected and highlighted in yellow. The main content area is titled 'Select category to change' and shows a table with one entry: 'Shirts'. The table has columns for 'NAME' and 'SLUG'. The 'NAME' column contains 'Shirts' and the 'SLUG' column contains 'shirts'. A button labeled 'ADD CATEGORY +' is visible at the top right of the table.

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

This screenshot shows the 'Add category' page within the Django administration interface. The left sidebar is identical to the previous screenshot, showing the 'Categories' model as the current selection. The main content area is titled 'Add category' and contains two input fields: 'Name:' and 'Slug:'. Below these fields is a large text area for additional information. At the bottom right of the form are three buttons: 'Save and add another', 'Save and continue editing', and a prominent blue 'SAVE' button.

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

The screenshot shows the Django admin interface for a 'Shirts' category. The left sidebar lists various models: Accounts, Authentication and Authorization, Orders, Shop (Categories, Customers, Products, Users), Sites, and Social Accounts. The 'Categories' model is selected, highlighted in yellow. The main content area is titled 'Change category' and shows a form for the 'Shirts' category. The 'Name' field contains 'Shirts' and the 'Slug' field contains 'shirts'. At the bottom are buttons for 'Delete', 'Save and add another', 'Save and continue editing', and 'SAVE'. The top right corner shows a welcome message for 'SHIRI' and links for 'VIEW SITE / CHANGE PASSWORD / LOG OUT'.

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

The screenshot shows a confirmation dialog for deleting the 'Shirts' category. The title is 'Are you sure?'. It asks if the user wants to delete the category 'Shirts', noting that all related items will be deleted. A 'Summary' section lists: Categories: 1, Products: 2, Order Items: 3. An 'Objects' section shows a tree of related items: Category: Shirts (Product: Brown Shirt, Order item: 2; Product: Grey Shirt, Order item: 3). At the bottom are two buttons: 'Yes, I'm sure' (in red) and 'No, take me back'.

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.

The screenshot shows the Django administration interface for the 'Shop' section, specifically the 'Products' list. On the left is a sidebar with various administrative links like Accounts, Authentication, Orders, Shop, Sites, and Social Accounts. The 'Products' link under the Shop section is highlighted. The main area displays a table of products with columns: NAME, SLUG, PRICE, AVAILABLE, CREATED, and UPDATED. Two products are listed: 'Brown Shirt' (slug: brown-shirt, price: 13.00, available: yes) and 'Grey Shirt' (slug: grey-shirt, price: 12.00, available: yes). A 'Save' button is visible at the bottom right of the table. To the right of the table is a 'FILTER' sidebar with options for filtering by availability (All, Yes, No), creation date (Any date, Today, Past 7 days, This month, This year), and update date (Any date, Today, Past 7 days, This month, This year).

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

The screenshot shows the 'Add product' page within the Django administration interface. The sidebar on the left is identical to the previous screenshot. The main area is titled 'Add product' and contains several input fields: 'Category' (with a dropdown menu and a '+' icon), 'Name' (text input field), 'Slug' (text input field), 'Image' (file upload field with 'Choose File' button and placeholder 'No file chosen'), 'Description' (text area), 'Price' (text input field), and a 'Available' checkbox. At the bottom right are three buttons: 'Save and add another', 'Save and continue editing', and a larger 'SAVE' button.

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

The screenshot shows the Django admin interface for a 'Grey Shirt' product. The left sidebar has a 'Products' section with 'Orders' and 'Products' highlighted. The main area is titled 'Change product' for 'Grey Shirt'. It includes fields for Category (Shirts), Name (Grey Shirt), Slug (grey-shirt), Image (currently products/2021/08/12/fas2.png), Description (Available in small and medium fit), and Price (12.00). A checkbox for 'Available' is checked. At the bottom are buttons for 'Delete' (red), 'Save and add another', 'Save and continue editing', and 'SAVE'.

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

The screenshot shows a 'Are you sure?' confirmation page for deleting a 'Grey Shirt'. The left sidebar has a 'Products' section with 'Products' highlighted. The main area asks 'Are you sure?' and provides a summary: 'Are you sure you want to delete the product "Grey Shirt"? All of the following related items will be deleted.' It lists a 'Summary' section with 'Products: 1' and 'Order items: 2', and an 'Objects' section listing 'Product: Grey Shirt' with 'Order item: 1' and 'Order item: 3'. At the bottom are 'Yes, I'm sure' and 'No, take me back' buttons.

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.

PRODUCT	PRICE	QUANTITY	DELETE?
Q		1	
Q		1	

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

Django administration

Home > Orders > Orders > Order 3

Change order

Order 3

User: Akhila

First name: test1

Last name: sel1

E-mail: test1sel@gmail.com

Address: 123 ljh

Postal code: 54789

City: omaha

Phone: +15312567878

Paid

Braintree id: r4c02c8j

ORDER ITEMS

PRODUCT	PRICE	QUANTITY	DELETE?
Q: Brown Shirt	13.00	1	<input type="checkbox"/>

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

Django administration

Home > Orders > Orders > Order 3 > Delete

Are you sure?

Are you sure you want to delete the order "Order 3"? All of the following related items will be deleted:

Summary

- Orders: 1
- Order items: 1

Objects

- Order: Order 3
- Order item: 4

Actions

Yes, I'm sure **No, take me back**

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.

The screenshot shows the Django administration interface for managing sites. The left sidebar lists various models: ACCOUNTS (Email addresses), AUTHENTICATION AND AUTHORIZATION (Groups), ORDERS (Orders), SHOP (Categories, Customers, Products, Users), SITES (Sites), and SOCIAL ACCOUNTS (Social accounts, Social application tokens, Social applications). The 'SITES' section is currently selected. The main content area is titled 'Select site to change' and displays a table with one row:

DOMAIN NAME	DISPLAY NAME
s2-8380.herokuapp.com	s2-8380.herokuapp.com

There is a search bar at the top and a 'Go' button. A 'WELCOME, SHIRI' message is at the top right.

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

The screenshot shows the Django administration interface for managing social accounts. The left sidebar is identical to the previous screenshot. The main content area is titled 'Select social account to change' and displays a table with one row:

USER	UID	PROVIDER
shirish	100264861280571499119	Google

A 'FILTER' sidebar on the right shows 'By provider' with options 'All' and 'Google'. There is a search bar at the top and a 'Go' button. A 'WELCOME, SHIRI' message is at the top right.

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

Option

The screenshot shows the Django administration interface for 'Social accounts - Social applications'. The left sidebar has a 'SOCIAL ACCOUNTS' section with 'Social applications' highlighted in yellow. The main content area is titled 'Select social application to change' and shows a table with one entry:

NAME	PROVIDER
ebuy	Google

Below the table, it says '1 social application'.

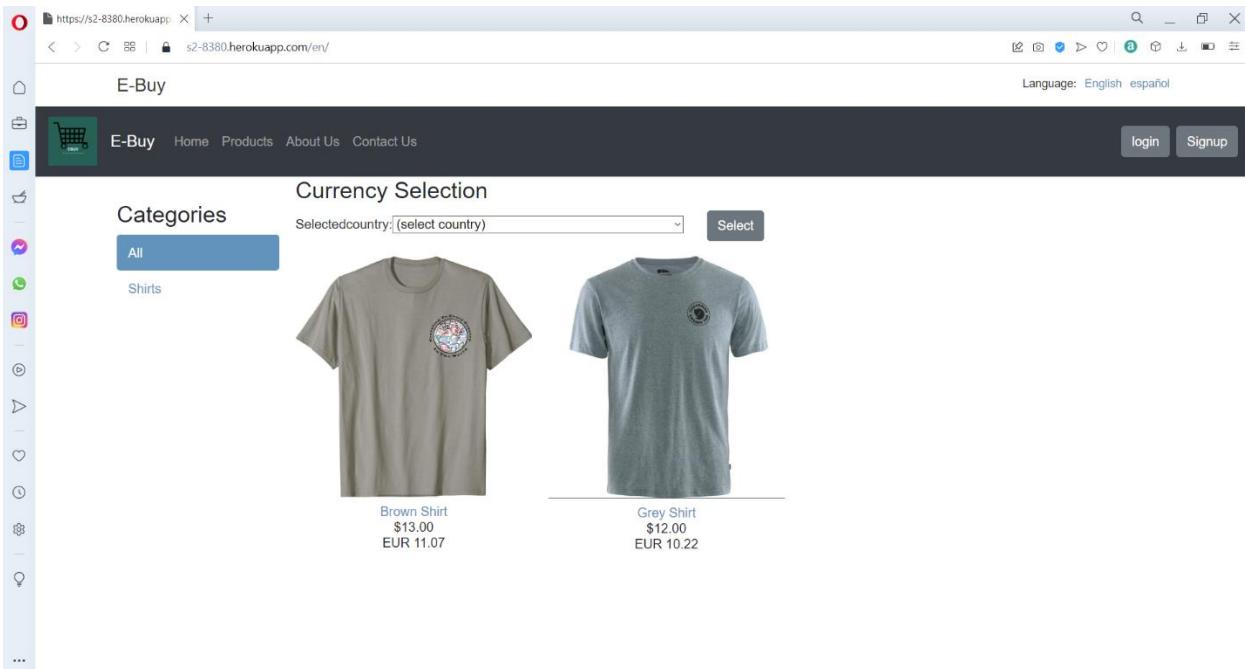
This is the social application to change page, where it will show the option for storing the client and secret key for authenticating customer using social account.

The screenshot shows the 'Change social application' page for the 'ebuy' application. The left sidebar has a 'SOCIAL ACCOUNTS' section with 'Social applications' highlighted in yellow. The main content area is titled 'Change social application' and shows the following fields:

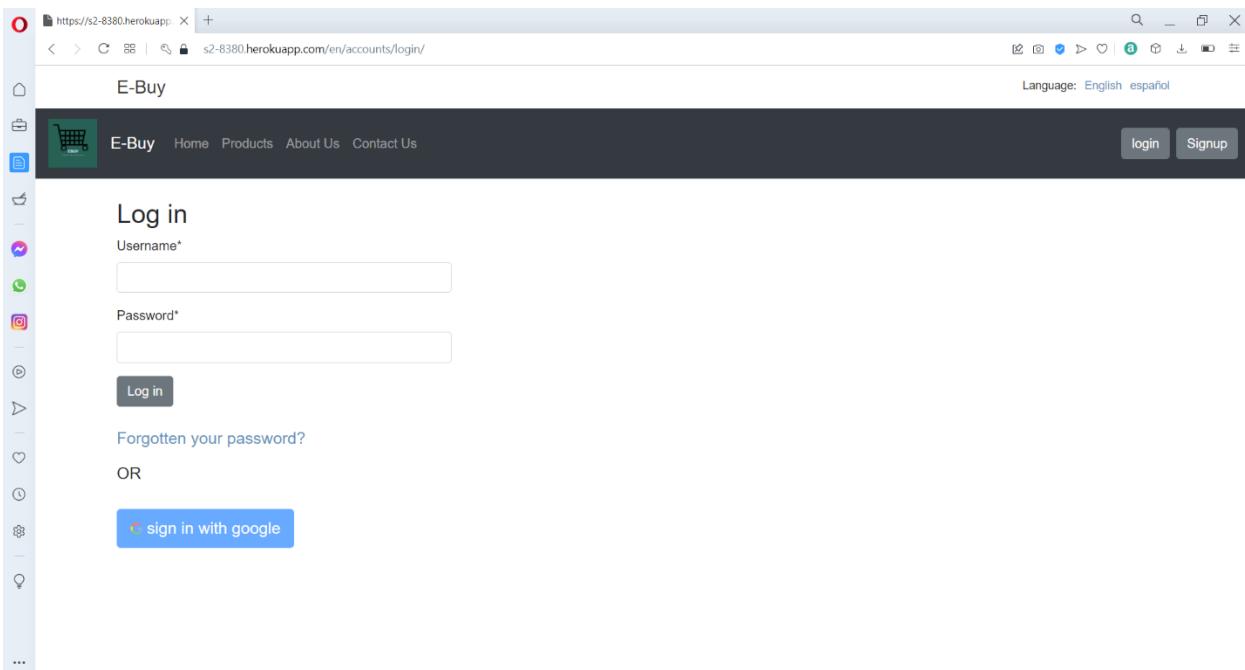
- Provider:** Google
- Name:** ebuy
- Client id:** 96830223220-r0t3h0n17f0cpdeldhnmqj9mdsq.apps.googleusercontent.com
- Secret key:** QJ65SaeROUE0unelzzF_0h...
- Key:** (empty)
- Sites:** A list of sites categorized into 'Available sites' and 'Chosen sites'. Under 'Available sites', there is a search bar and a note: 'Hold down "Control", or "Command" on a Mac, to select more than one.' Under 'Chosen sites', there is a single item: s2-8380.herokuapp.com.

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.

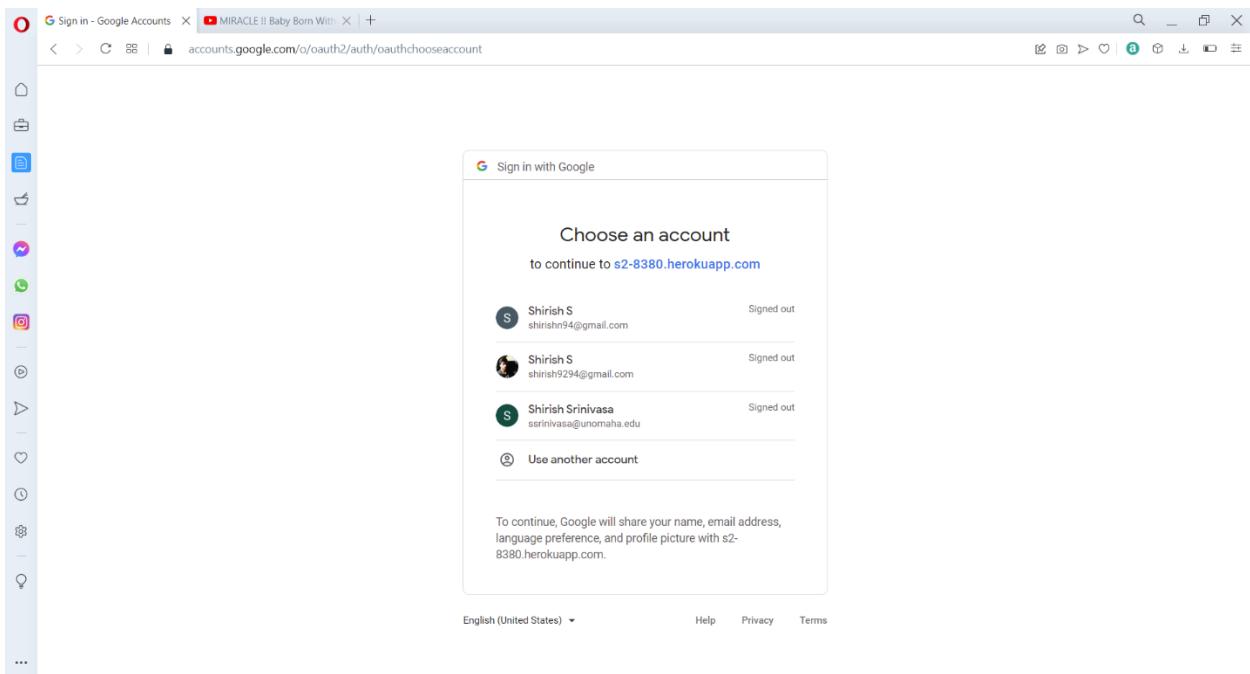
Web application



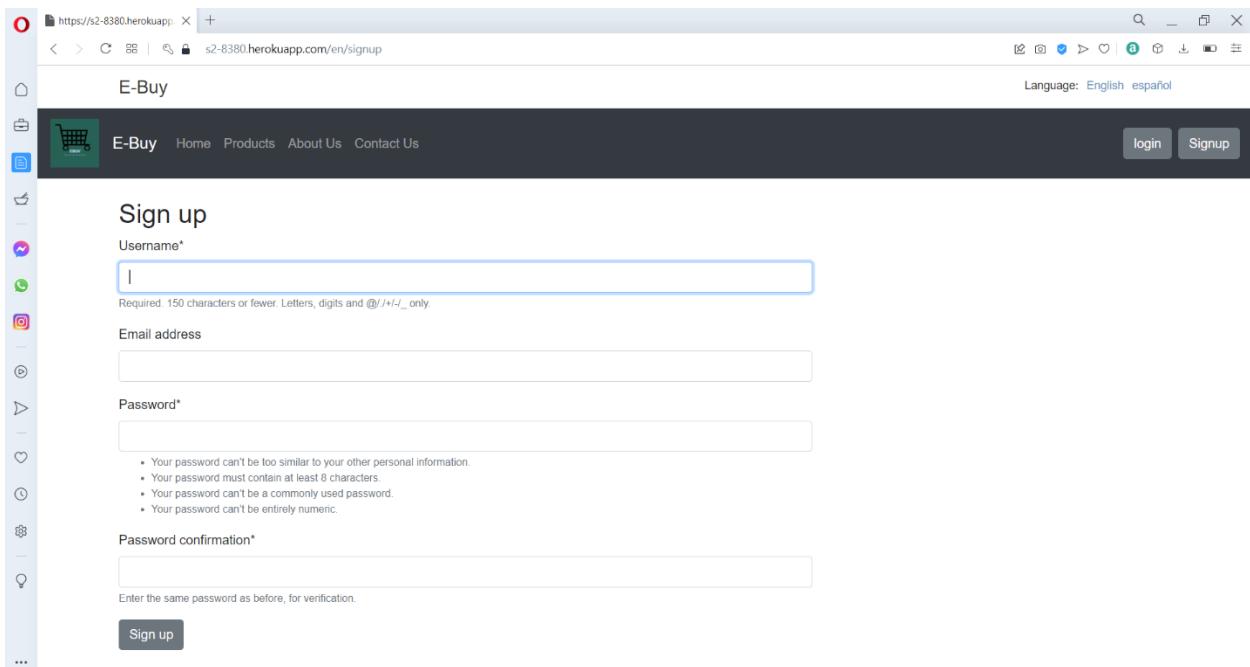
This is the home page of the ebuy application.



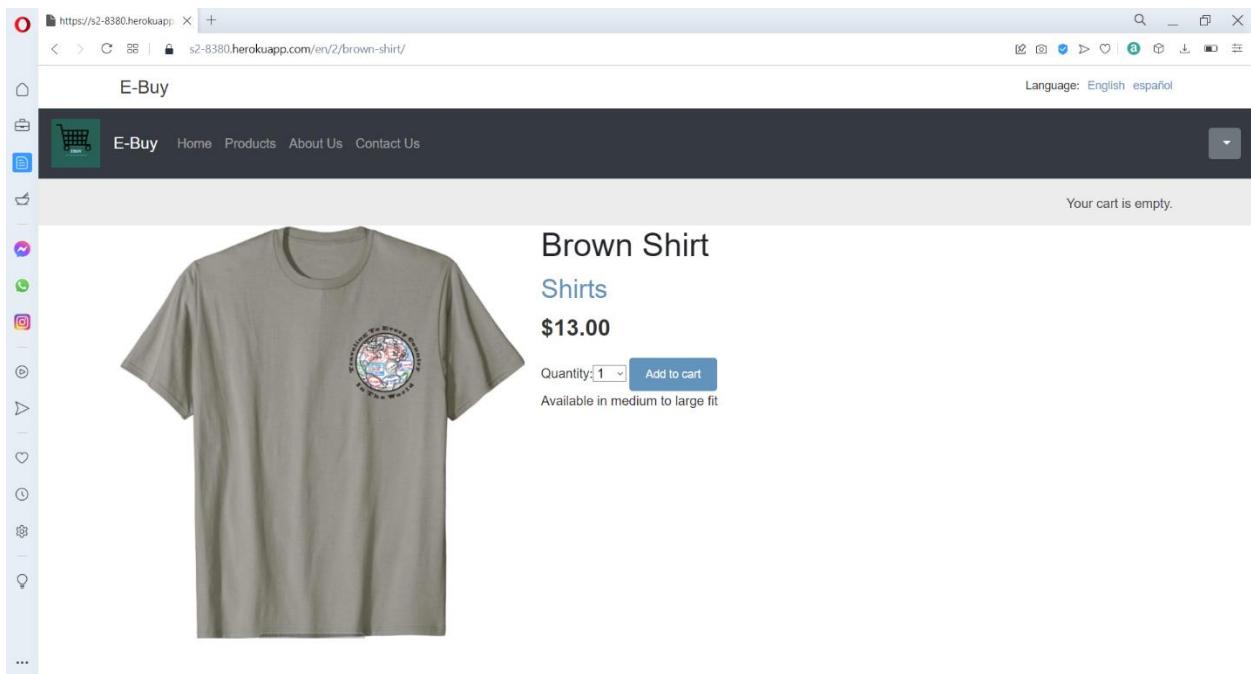
This is the login page of the e-buy application, where the customer can enter the login information or can sign in through google.



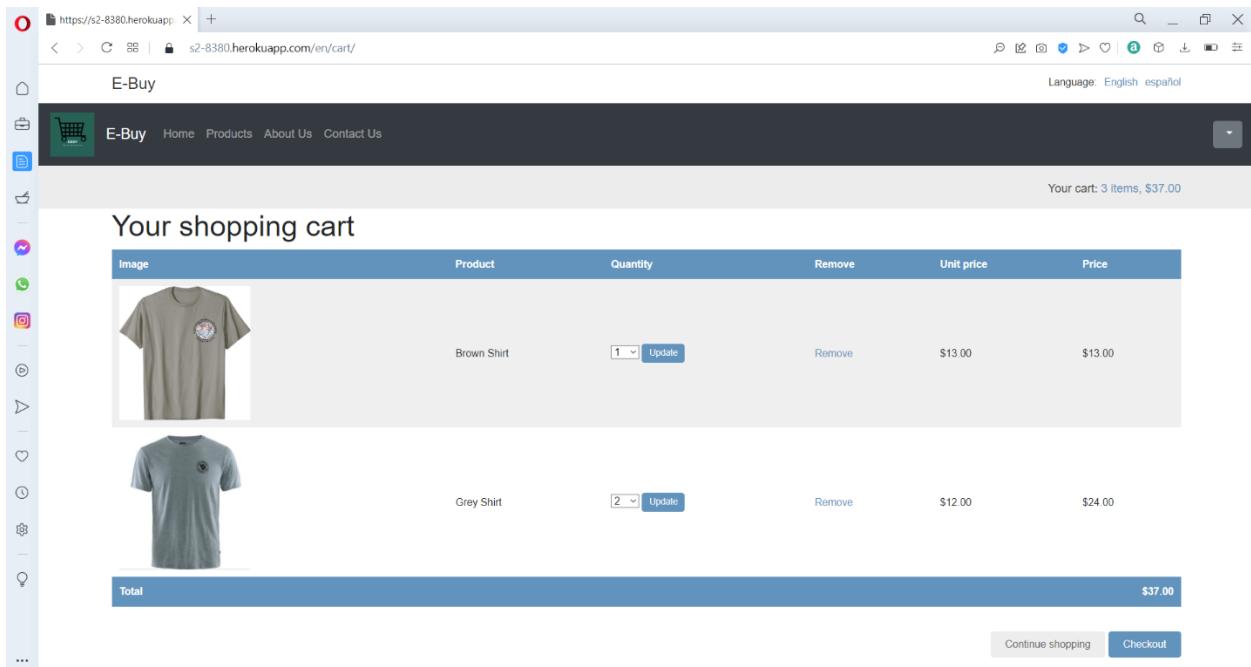
This is the google sign in screen which will be redirected for the customer, where the customer will choose an account to register and provide access to the web application.



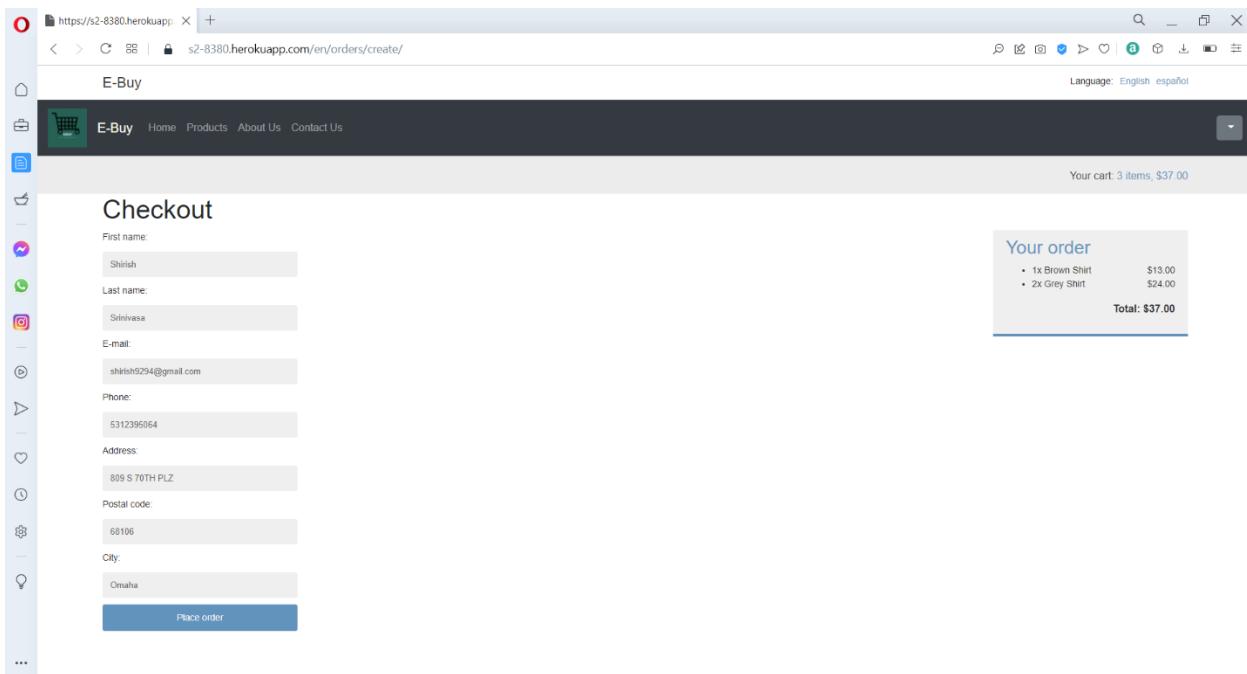
This is the sign up page where a customer can enter his details and register for the web application.



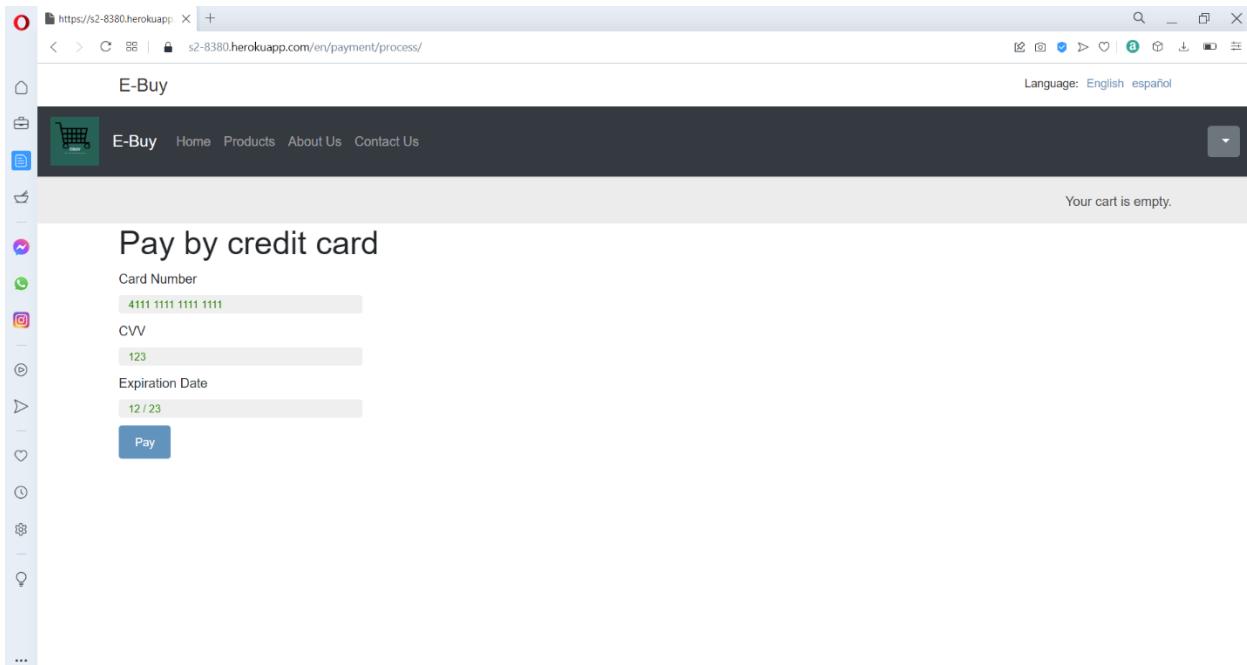
When the user selects a product, he will be redirected to the product page where he will get the information like the product name, price and options like quantity and add to cart.



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.

A screenshot of a web browser showing a successful payment confirmation. The URL is <https://s2-8380.herokuapp.com/en/payment/done/>. The page title is "E-Buy". The navigation bar includes links for "Home", "Products", "About Us", and "Contact Us". A message says "Your cart is empty." Below it, a large heading says "Your payment was successful" with the subtext "Your payment has been processed successfully." On the left side, there is a vertical sidebar with various icons for messaging, file sharing, and other applications.

This page shows that the order has been successfully placed.

A screenshot of a Gmail inbox. The subject of the top email is "order 2". The email is from "shirish.u94@gmail.com via sendgrid.net" and was sent "9:00 PM (2 minutes ago)". The body of the email reads: "Your order no 2, with items costing \$37.00, has been placed successfully." The inbox also shows other recent emails and notifications for meetings and video calls.

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

This screenshot shows the 'Order List' page of the E-Buy application. The page has a dark header with the logo 'E-Buy' and navigation links for Home, Products, About Us, and Contact Us. Below the header, a message says 'Your cart is empty.' The main content area is titled 'Order List' and contains a table with two rows of order information.

Order ID	Order date	Amount	Order Details	PDF
2	Aug. 12, 2021, 8:59 p.m.	37.00	view details	pdf
1	Aug. 12, 2021, 6:24 p.m.	36.00	view details	pdf

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 screenshot shows the 'Order Details' page for Order ID 2. The page has a dark header with the logo 'E-Buy' and navigation links for Home, Products, About Us, and Contact Us. Below the header, a message says 'Your cart is empty.' The main content area is titled 'Order Details' and displays the following information:

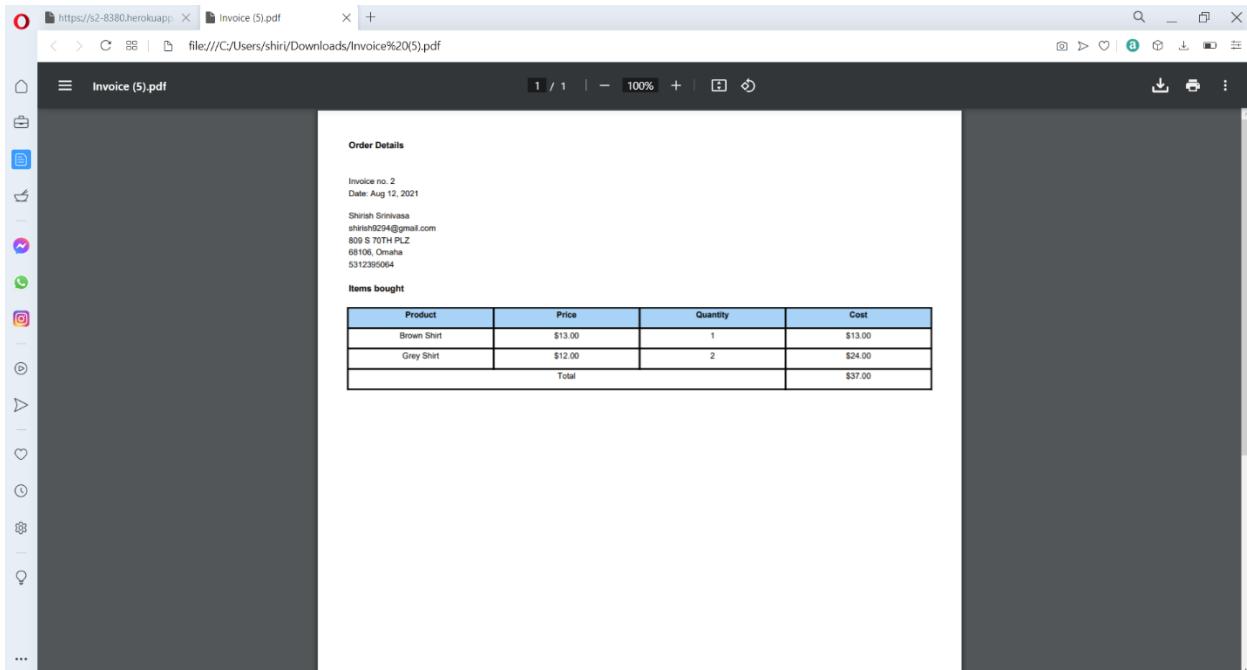
Invoice no. 2
Date: Aug 12, 2021

Shirish Srinivasa
shirish9294@gmail.com
809 S 70TH PLZ
68106, Omaha
5312395064

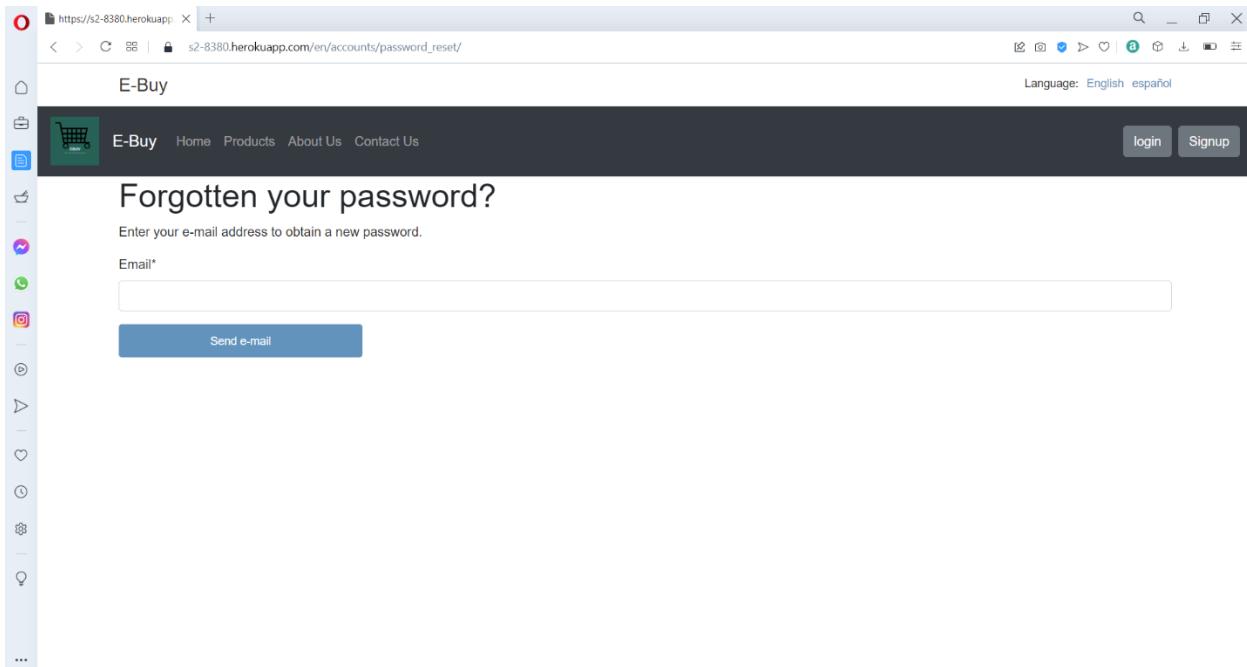
Items bought

Product	Price	Quantity	Cost
Brown Shirt	\$13.00	1	\$13.00
Grey Shirt	\$12.00	2	\$24.00
Total			\$37.00

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.

The screenshot shows a web browser window with the URL https://s2-8380.herokuapp.com/en/accounts/password_reset/done/. The page title is "E-Buy". The main content area displays the message: "We've emailed you instructions for resetting your password. If you don't receive an email, please make sure you've entered the address you registered with." Navigation icons on the left side of the browser include back, forward, search, and refresh.

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

The screenshot shows a Gmail inbox with one new email from "shirish.u94@gmail.com". The subject line is "Password reset on s2-8380.herokuapp.com". The email body contains the following text:

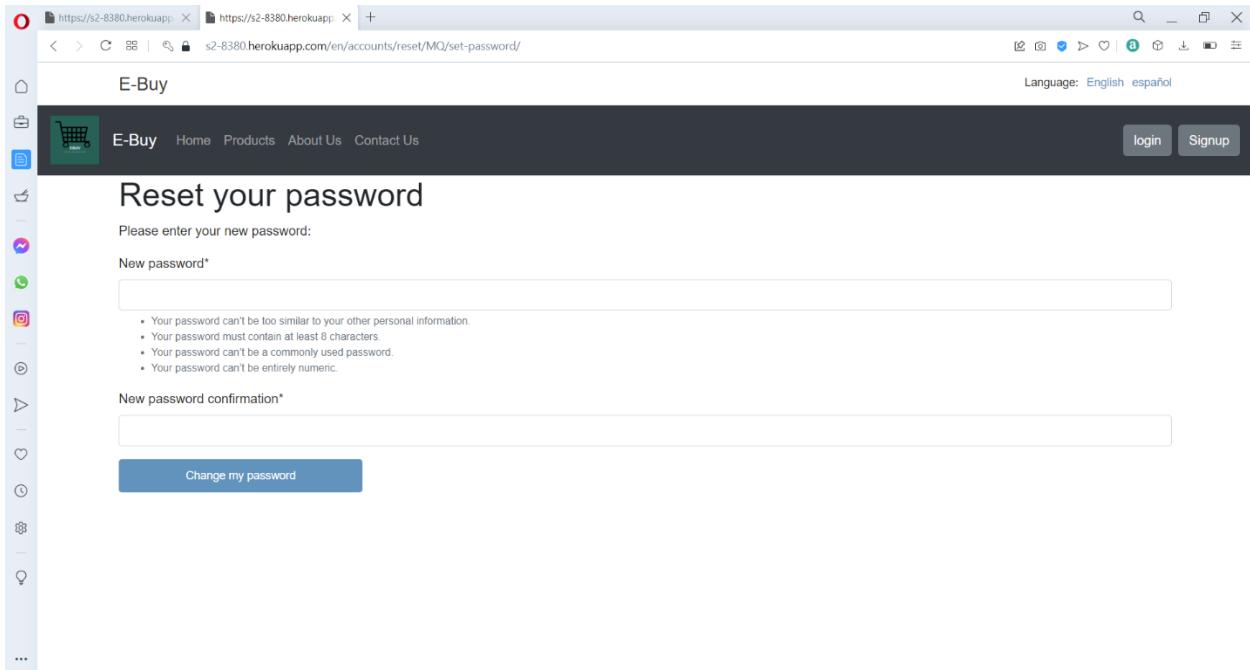
```
You're receiving this email because you requested a password reset for your user account at s2-8380.herokuapp.com.

Please go to the following page and choose a new password.

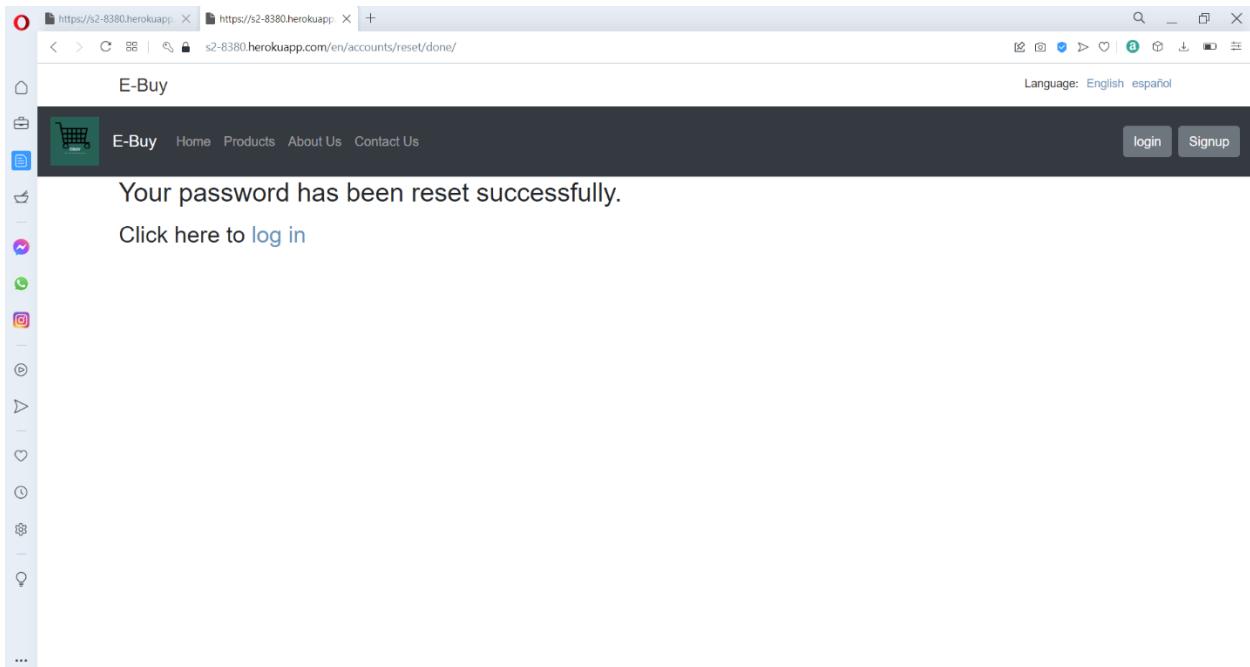


This is how the reset link will be displayed in the email, which we can use to reset our password.


```



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



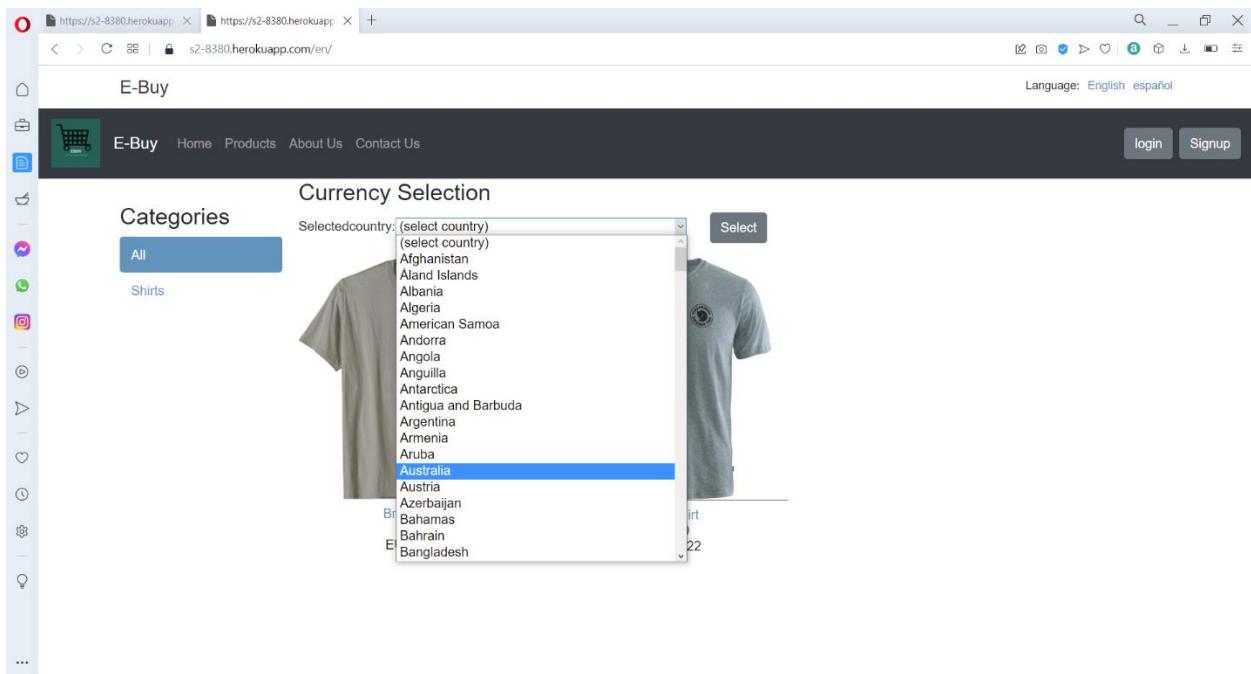
After the password has been reset, the customer is redirected to the confirmation page, where it shows us that the password has been reset successfully.

The screenshot shows a web browser window with two tabs open. The active tab is titled 'https://s2-8380.herokuapp.com/en/accounts/password_change/'. The page header 'E-Buy' is visible, along with a shopping cart icon and navigation links for Home, Products, About Us, and Contact Us. A language selector at the top right offers English and español. Below the header, a message states 'Your cart is empty.' The main content area is titled 'Change your password' and contains instructions: 'Use the form below to change your password.' It features four input fields: 'Old password*', 'New password*', 'New password confirmation*', and a 'Change' button. To the left of the form is a vertical sidebar with various icons for social media sharing and other site functions.

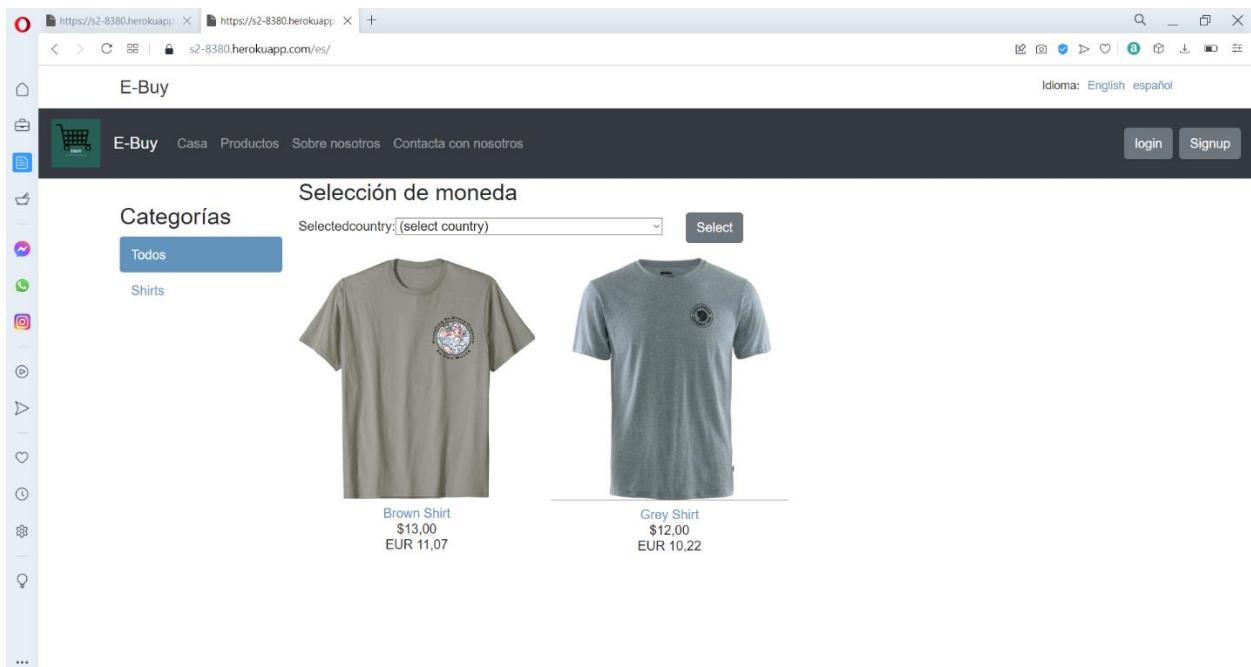
This is the change your password page, where after the customer logs in, he can reset his existing password, by using his old password for confirmation.

The screenshot shows a web browser window with two tabs open. The active tab is titled 'https://s2-8380.herokuapp.com/en/accounts/password_change/done/'. The page header 'E-Buy' is visible, along with a shopping cart icon and navigation links for Home, Products, About Us, and Contact Us. A language selector at the top right offers English and español. Below the header, a message states 'Your cart is empty.' The main content area is titled 'Password changed' and displays the message 'Your password has been successfully changed.' To the left of the content is a vertical sidebar with various icons for social media sharing and other site functions.

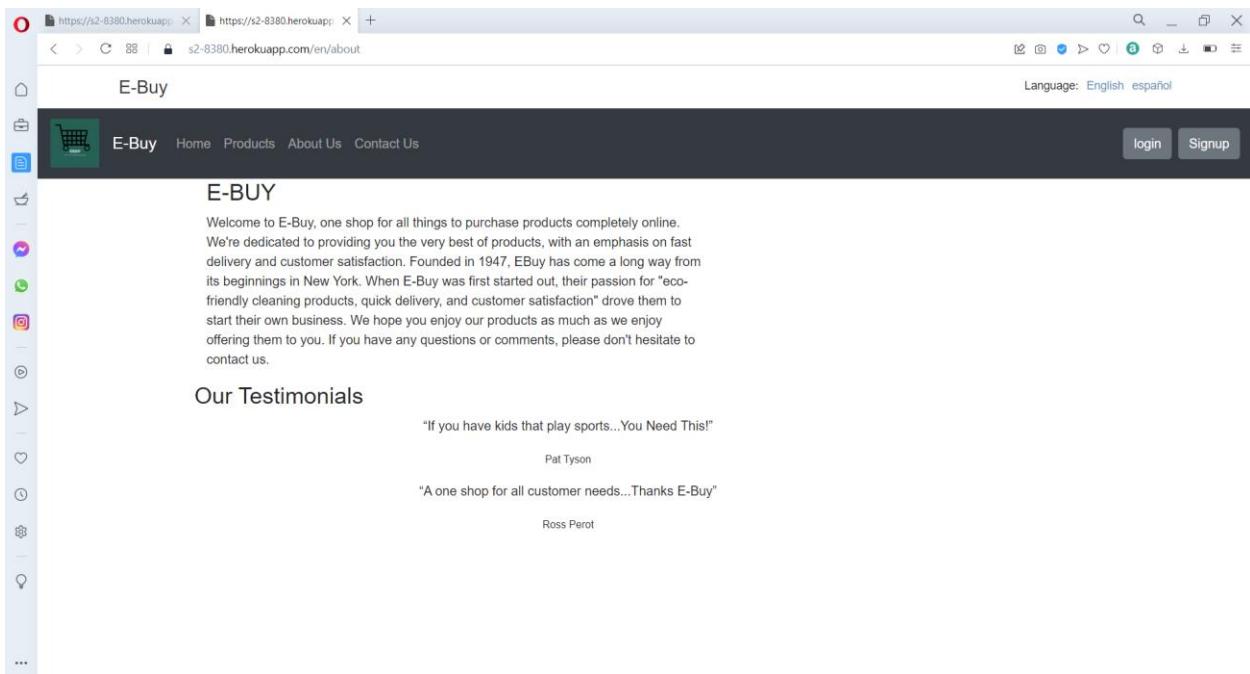
This is the password changes confirmation page, where it shows that the password has been successfully changed by the customer.



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.

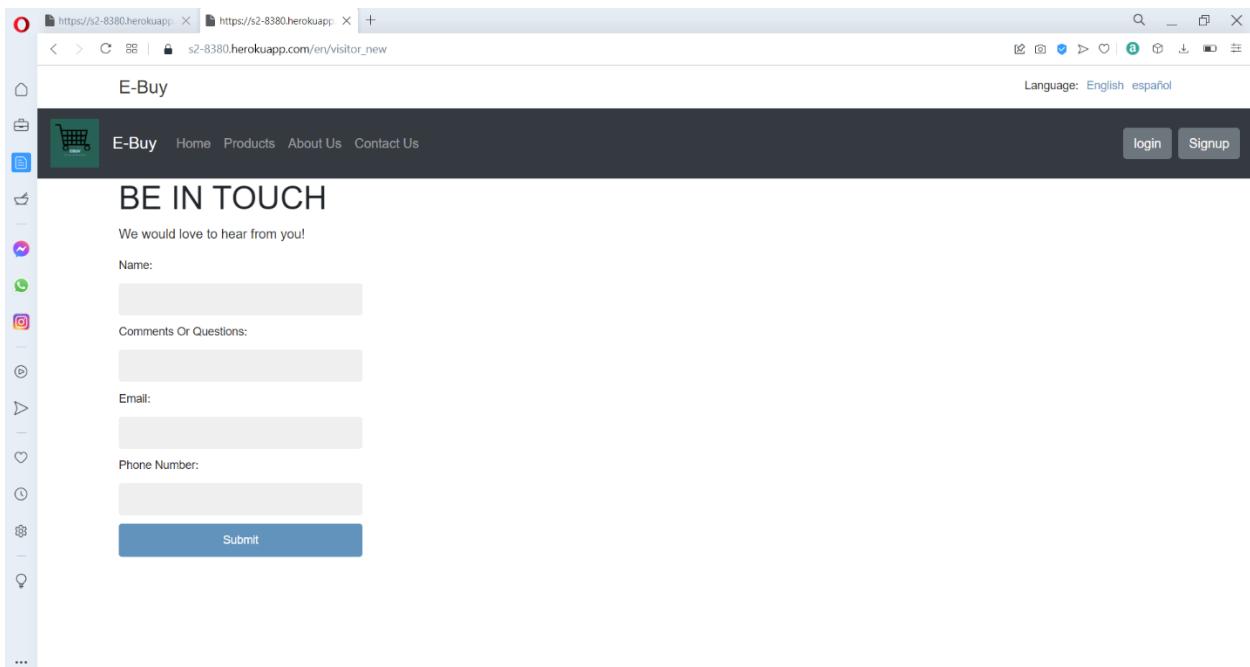


This is the language conversion option available in the e-buy web app, where on the top right-hand side, we get the option to change the language from English to Spanish and Spanish to English.



The screenshot shows a web browser window with two tabs open. The active tab is titled "About Us" and displays the content of the "About Us" page for the "E-Buy" application. The page has a dark header with the "E-Buy" logo and navigation links for Home, Products, About Us, and Contact Us. On the left, there is a vertical sidebar with icons for Home, Products, About Us, Contact Us, and other social media links. The main content area features a heading "E-BUY" and a paragraph describing the company's history and mission. Below this, there is a section titled "Our Testimonials" with two entries. The first testimonial is from Pat Tyson, who says, "If you have kids that play sports...You Need This!". The second testimonial is from Ross Perot, who says, "A one shop for all customer needs...Thanks E-Buy". At the bottom right of the page are "login" and "Signup" buttons.

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



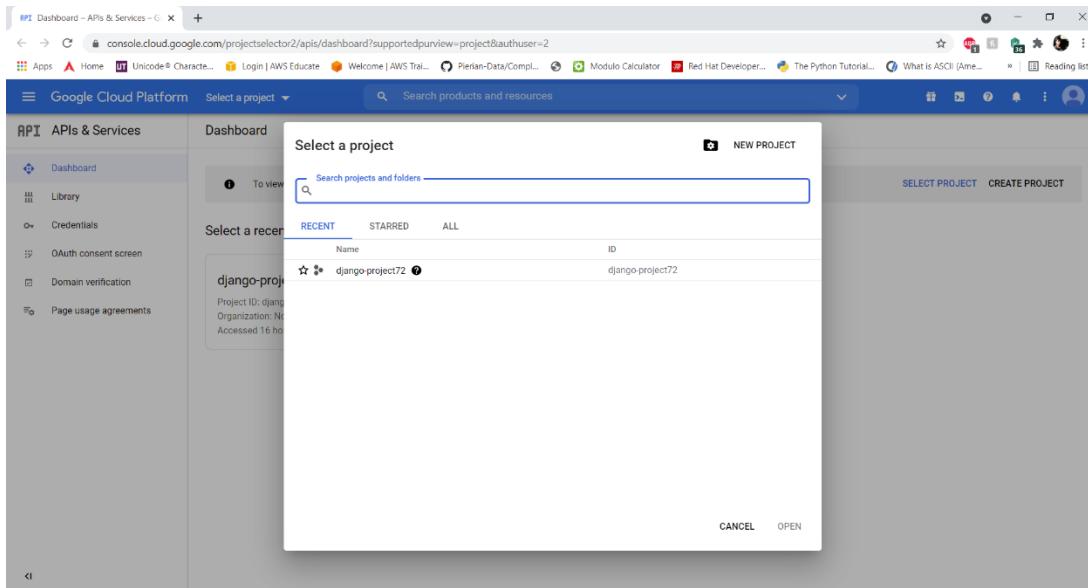
The screenshot shows a web browser window with two tabs open. The active tab is titled "visitor_new" and displays the content of the "Contact Us" page for the "E-Buy" application. The page has a dark header with the "E-Buy" logo and navigation links for Home, Products, About Us, and Contact Us. On the left, there is a vertical sidebar with icons for Home, Products, About Us, Contact Us, and other social media links. The main content area features a heading "BE IN TOUCH" and a message "We would love to hear from you!". Below this, there are four input fields labeled "Name:", "Comments Or Questions:", "Email:", and "Phone Number:". At the bottom right of the form is a blue "Submit" button.

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

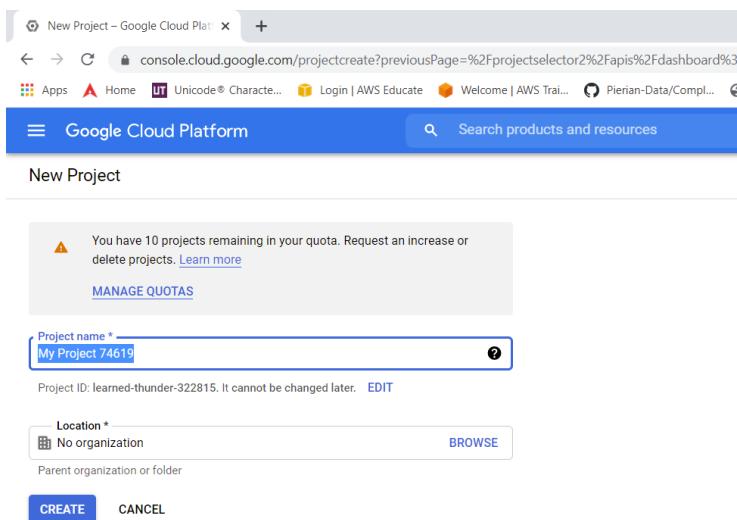
API

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.

The screenshot shows the Google Cloud Platform API Credentials page for a project named 'django-project72'. The left sidebar has 'APIs & Services' selected under 'Credentials'. The main area shows 'OAuth 2.0 Client I' credentials. A table lists one client: 'e-buy' (Client ID: 96830223220-r010t3h0h17f0ccpdelhlnmqj9mdsq.apps.googleusercontent.com, Type: Web application). Below the table is a section for 'Service Accounts' which is currently empty.

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

The screenshot shows the Google Cloud Platform OAuth Client ID creation page for a web application. The 'Name' field is set to 'e-buy'. The 'Authorized JavaScript origins' field contains 'https://s2-6380.herokuapp.com'. The 'Authorized redirect URIs' field contains 'https://s2-6380.herokuapp.com/en/accounts/google/login/callback/' and 'https://s2-6380.herokuapp.com/es/accounts/google/login/callback/'. Both fields have '+ ADD URI' buttons. At the bottom are 'SAVE' and 'CANCEL' buttons.

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',
```

```
File Edit View Navigate Code Refactor Run Tools VCS Window Help v1 - settings.py  
v1 myshop settings.py  
Project C:\Users\shiri\Videos\v1  
  > cart  
  > locale  
  > media  
  > myshop  
    > nppBackup  
      <__init__.py  
      <settings.py  
      <urls.py  
      <wsgi.py  
  > orders  
  > payment  
  > shop  
  > static  
  > venv  
    <.gitignore  
    <db.sqlite3  
    <manage.py  
    <Procfile  
    <README.md  
    <README.txt  
    <requirements.txt  
  External Libraries  
  Scratches and Consoles  
  urls.py  password_reset.html  reset_email_sent.html  README.txt  orders\models  
229 LOGIN_REDIRECT_URL = 'shop:product_list'  
230 LOGOUT_REDIRECT_URL = 'shop:product_list'  
231  
232 SITE_ID = 2  
233  
234 SOCIALACCOUNT_PROVIDERS = {  
235     'google': {  
236         'SCOPE': [  
237             'profile',  
238             'email',  
239         ],  
240         'AUTH_PARAMS': {  
241             'access_type': 'online',  
242         }  
243     }  
244 }  
245  
246 PARLER_LANGUAGES = {  
247     'None': [  
248         {'code': 'en'},  
249     ]  
250 }
```

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',
}
}
}

```

The screenshot shows the Django admin interface for a 'Sites' model. The left sidebar has categories like ACCOUNTS, AUTHENTICATION AND AUTHORIZATION, ORDERS, SHOP, SITES, and SOCIAL ACCOUNTS. Under SITES, 'Sites' is selected. The main content area shows a single site entry: 's2-8380.herokuapp.com'. The 'Domain name' field contains 's2-8380.herokuapp.com' and the 'Display name' field also contains 's2-8380.herokuapp.com'. At the bottom are 'Delete', 'Save and add another', 'Save and continue editing', and a large blue 'SAVE' button.

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.

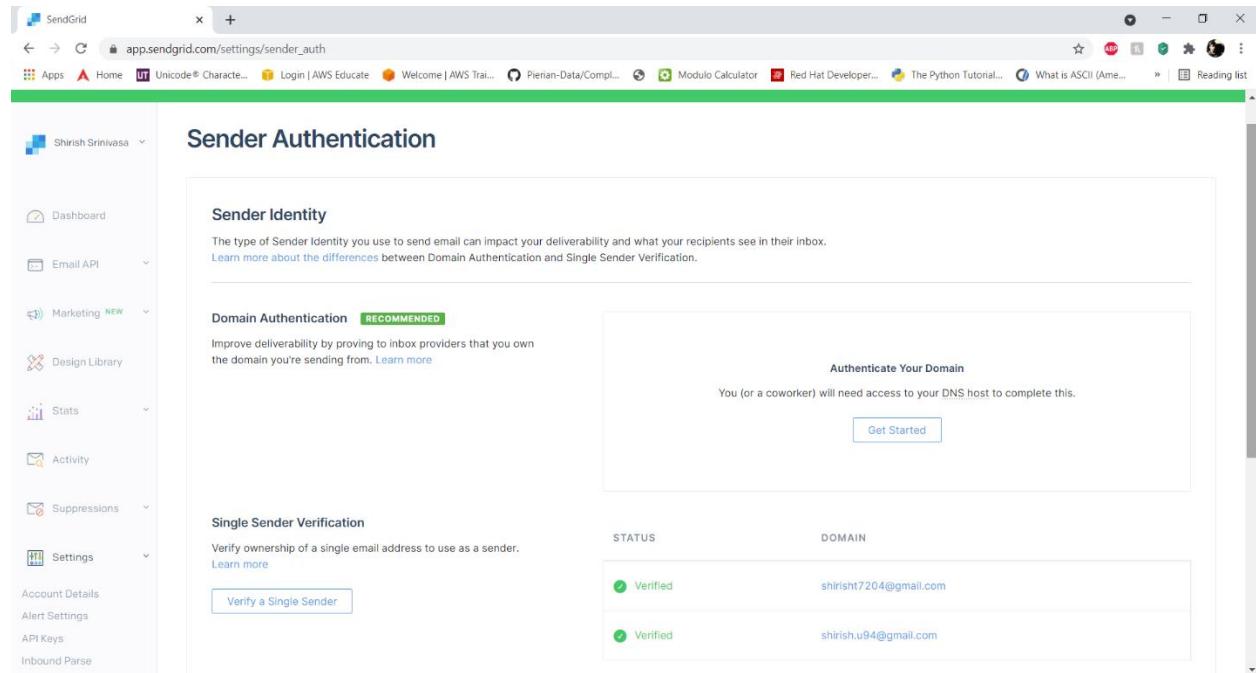
Now when ever a user signs up using a social account, his details are stored in the social accounts section in the admin panel with a UID and that information is taken and is automatically added under the registered user section.

SendGrid API

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

<https://app.sendgrid.com/>

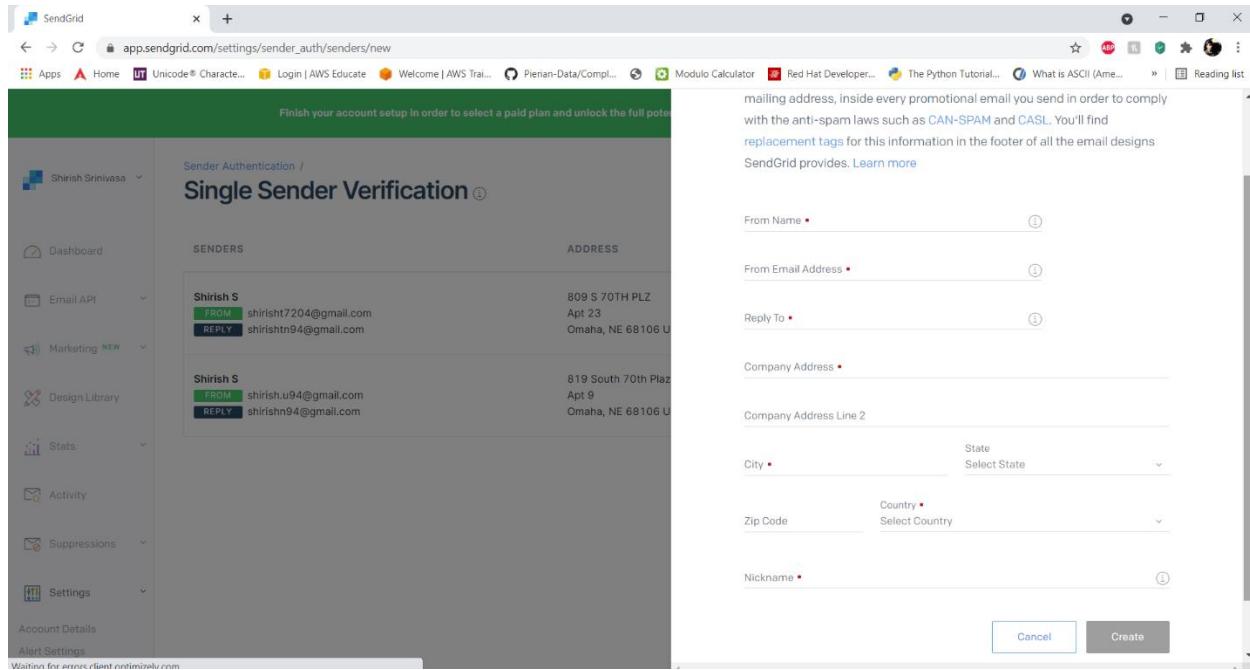
and creating an account and then login to the website.



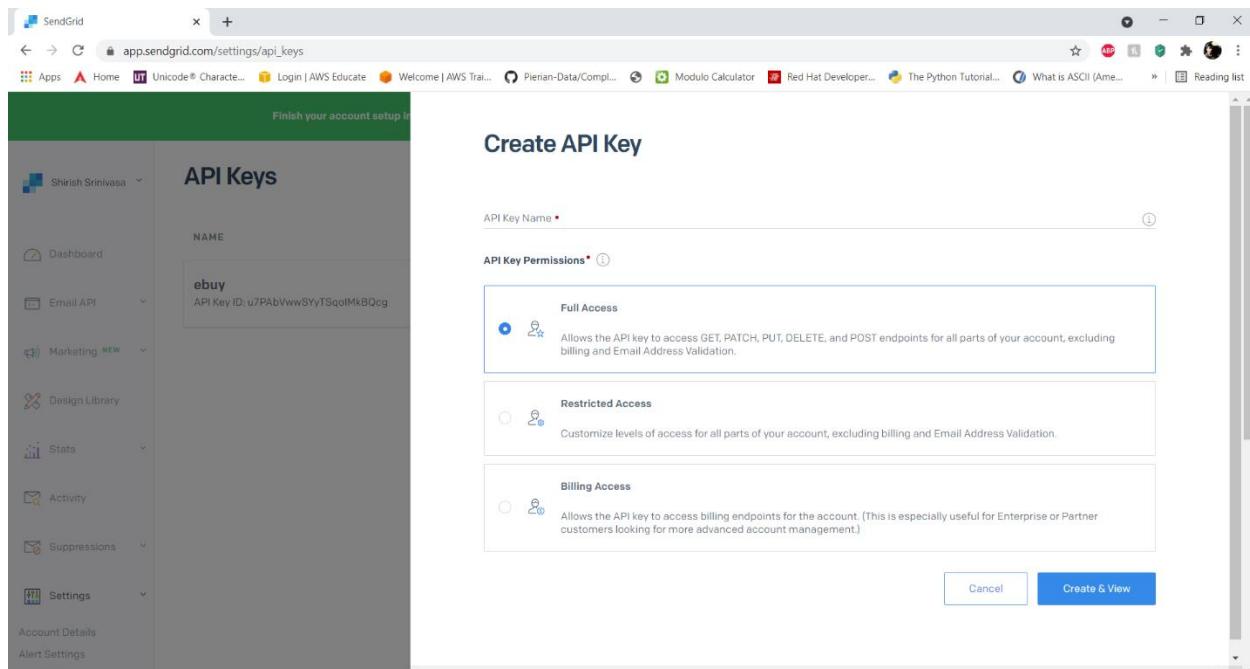
The screenshot shows the SendGrid web interface. The left sidebar contains navigation links: Dashboard, Email API, Marketing (NEW), Design Library, Stats, Activity, Suppressions, Settings, Account Details, Alert Settings, API Keys, and Inbound Parse. The main content area is titled "Sender Authentication". It features a "Sender Identity" section with a note about deliverability and a link to learn more about Domain Authentication vs Single Sender Verification. Below this is a "Domain Authentication" section with a green "RECOMMENDED" badge. It includes a sub-section "Authenticate Your Domain" with a "Get Started" button. A note states that access to a DNS host is required. The "Single Sender Verification" section shows two entries in a table:

STATUS	DOMAIN
Verified	shirish7204@gmail.com
Verified	shirish.u94@gmail.com

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

```

File Edit View Navigate Code Refactor Run Tools VCS Window Help v1 - settings.py
v1 myshop > settings.py
Project C:\Users\shiri\Videos\v1
  > cart
  > locale
  > media
  > myshop
    > __init__.py
    > settings.py
    > urls.py
    > wsgi.py
  > orders
  > payment
  > shop
  > static
  > venv
    > gitignore
    db.sqlite3
    manage.py
    Procfile
    README.md
    README.txt
    requirements.txt
  External Libraries
  Scratches and Consoles
urls.py < password_reset.html < reset_email_sent.html < README.txt < orders\mode
67 # EMAIL_PORT = '2525'
68 # EMAIL_HOST = 'smtp.gmail.com'
69 # EMAIL_HOST_USER = 'shirish.u94@gmail.com'
70 # EMAIL_HOST_PASSWORD = 'ukwyryykdutozxft'
71 # EMAIL_PORT = '587'
72 # EMAIL_USE_TLS = True
73
74 EMAIL_BACKEND = "anymail.backends.sendgrid.EmailBackend"
75 ANYMAIL = {
76     "SENDGRID_API_KEY": os.getenv('SENDGRID_API_KEY'),
77 }
78 DEFAULT_FROM_EMAIL = 'shirish.u94@gmail.com'
79
80 MIDDLEWARE = [
81     'django.middleware.security.SecurityMiddleware',
82     'django.contrib.sessions.middleware.SessionMiddleware',
83     'django.middleware.locale.LocaleMiddleware',
84     'django.middleware.common.CommonMiddleware',
85     'django.middleware.csrf.CsrfViewMiddleware',
86     'django.contrib.auth.middleware.AuthenticationMiddleware',
87 ]

```

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:

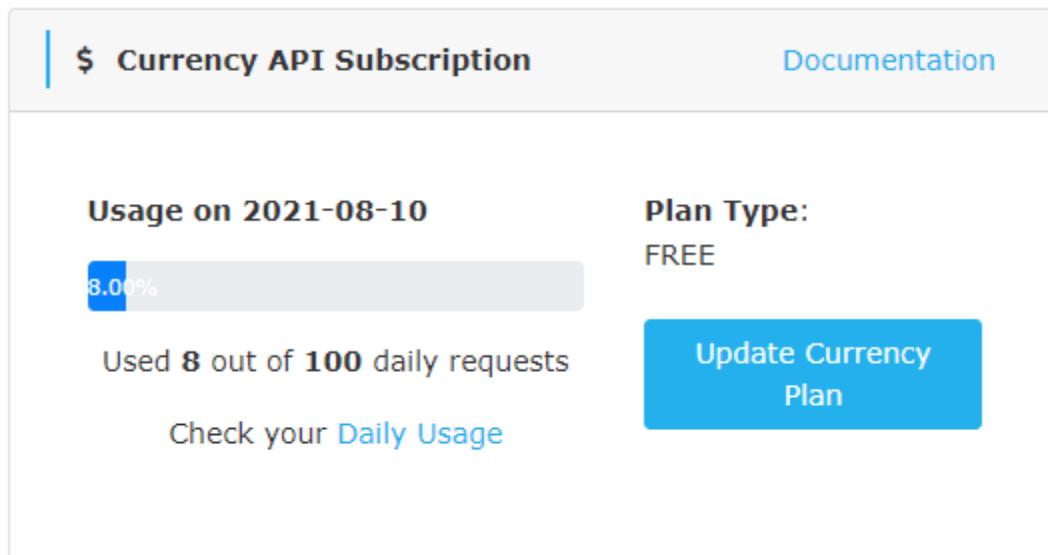
Get Geo Currency provides exchange rates from one currency to another for a specific day.

First, I registered for a free account at <https://currency.getgeoapi.com/register/>.

Username: lahrenkenius@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.

Authentication

In order to get an access to the API you need to get the api key. Each user after the registration and email validation would be able to get it in the account dashboard.

To use your api key you need to append `api_key` parameter to every request. Please see the example below.

```
GET: https://api.getgeoapi.com/v2/currency/list?api_key=6e1e62d64cbc5f6cec2a8cae8dbd34cd38953e6a
```

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']
extrate = extrate[currencycode]
extrate = extrate['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.

Currency Selection

Selected country 

Categories

All

Autos

Clothing

Electronics

Products

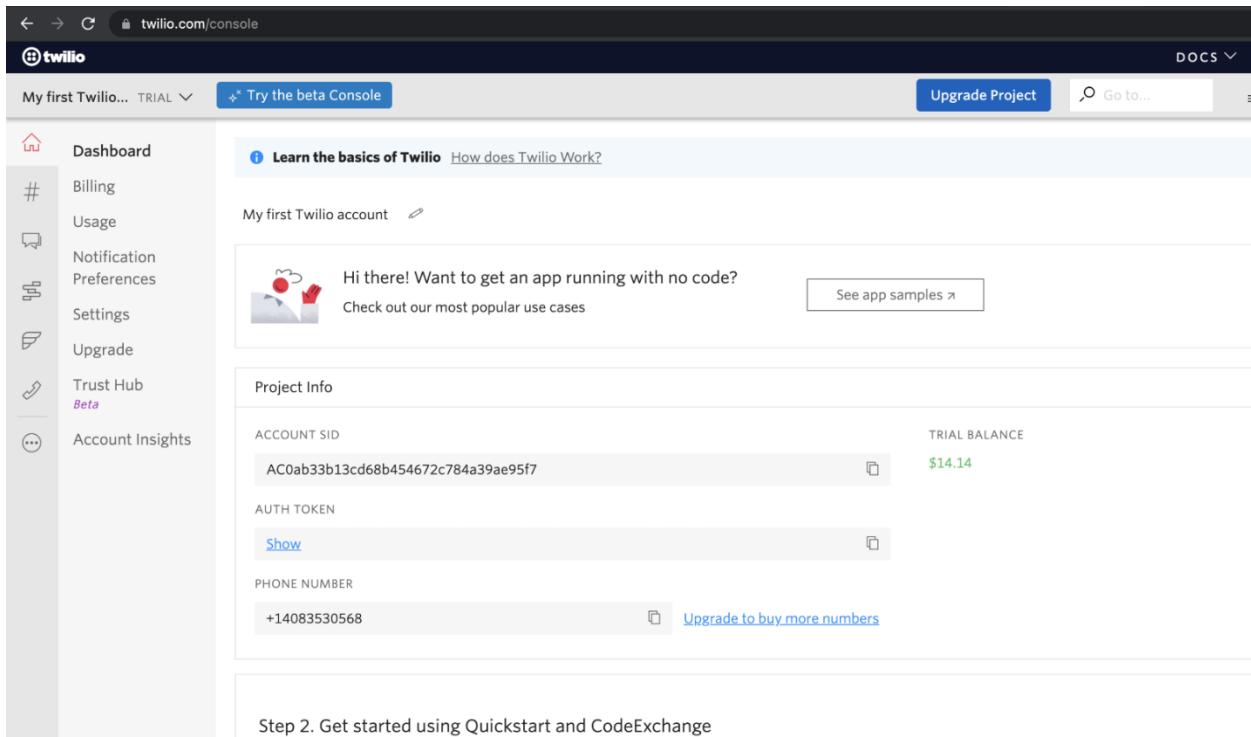


Product	Description	Price	Converted Price
Computer	\$200.00	AUD 272.340000	£160.00
TV	\$25.00	AUD 34.042500	£20.00

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



The screenshot shows the Twilio console dashboard. On the left, there is a sidebar with icons and links: Dashboard (selected), Billing, Usage, Notification Preferences, Settings, Upgrade, Trust Hub (Beta), and Account Insights. The main area has a header with 'My first Twilio... TRIAL' and a 'Try the beta Console' button. Below the header, there is a 'Learn the basics of Twilio' section with a 'How does Twilio Work?' link. A 'My first Twilio account' section includes a 'Hi there! Want to get an app running with no code?' message, a 'See app samples' button, and a 'Project Info' section. The 'Project Info' section displays the ACCOUNT SID (AC0ab33b13cd68b454672c784a39ae95f7), AUTH TOKEN (Show), and PHONE NUMBER (+14083530568). It also includes a 'TRIAL BALANCE' of '\$14.14' and a link to 'Upgrade to buy more numbers'. At the bottom, there is a 'Step 2. Get started using Quickstart and CodeExchange' section.

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/ACXXXXXX.../Messages/SM1f0e8ae6ade43cb3c0ce4525424e404f.json
```

```
1 | {
2 |     "sid": "SM1f0e8ae6ade43cb3c0ce4525424e404f",
3 |     "date_created": "Fri, 13 Aug 2010 01:16:24 +0000",
4 |     "date_updated": "Fri, 13 Aug 2010 01:16:24 +0000",
5 |     "date_sent": null,
6 |     "account_sid": "ACXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
7 |     "to": "+15305431221",
8 |     "from": "+15104564545",
9 |     "body": "A Test Message",
10 |    "status": "queued",
11 |    "flags": ["outbound"],
12 |    "api_version": "2010-04-01",
13 |    "price": null,
14 |    "uri": "\/2010-04-01\/Accounts\/ACXXXXXXXXXXXXXX"
15 | }
```

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

My first Twilio... TRIAL ▾ Messaging / Monitor / Try the beta Console Upgrade Project Go to... Docs Kavya RaviPr...

Programmable Messaging

Monitor

Dashboard

Insights New Logs

Try it Out Messaging Services Senders Helper Tools Settings

Programmable Messaging Dashboard

Messages Last 30 Days

Automate your SMS conversations with a bot

Errors & Warnings

You have no errors or warnings

Show API Credentials

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	—

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:

The screenshot shows the Rosetta web interface at 127.0.0.1:8000/en/rosetta/files/project/. The interface has a header with a back arrow, forward arrow, and refresh icon. Below the header is a blue navigation bar with the word "Rosetta". Underneath the navigation bar is a breadcrumb trail: "Home > Language selection". On the right side of the header is a "Filter" button with options: PROJECT (highlighted in yellow), THIRD PARTY, DJANGO, and ALL. The main content area is divided into two sections: "Inglés" and "Español". Each section has a table with columns: APPLICATION, PROGRESS, MESSAGES, TRANSLATED, FUZZY, OBSOLETE, and FILE. For the "Inglés" section, the data is: Su2021team2-App, 3%, 59, 2, 0, 0, /Users/kavyaharish26/Dropbox/My Mac (MacBook Pro)/Downloads/su2021team2-app/locale/en/LC_MESSAGES/django.po. For the "Español" section, the data is: Su2021team2-App, 96%, 57, 57, 2, 2, /Users/kavyaharish26/Dropbox/My Mac (MacBook Pro)/Downloads/su2021team2-app/locale/es/LC_MESSAGES/django.po.

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:

The screenshot shows the Rosetta translation interface at 127.0.0.1:8000/en/rosetta/files/project/es/0/. The page title is "Rosetta". The URL bar shows "Home · Español · Su2021team2-App · Progress: 96%". The top navigation bar has tabs for "Display: UNTRANSLATED ONLY", "TRANSLATED ONLY", "FUZZY ONLY", and "ALL". A search bar and a "Go" button are at the top left. Below is a table with two columns: "ORIGINAL" and "ESPAÑOL". The "ESPAÑOL" column contains translated words like "Cantidad", "Tu carro", "Imagen", etc. To the right of the table are "FUZZY" and "OCCURRENCES(S)" sections with checkboxes and lists of file paths where each word appears. At the bottom, there are links to "Skip to page: 1 2 3 4 5 6" and "59/61 messages", and a "SAVE AND TRANSLATE NEXT BLOCK" button.

ORIGINAL	ESPAÑOL	FUZZY	OCCURRENCES(S)
Quantity	Cantidad	<input type="checkbox"/>	cart/forms.py:12 cart/templates/cart/detail.html:17
Your shopping cart	Tu carro	<input type="checkbox"/>	cart/templates/cart/detail.html:7 cart/templates/cart/detail.html:11
Image	Imagen	<input type="checkbox"/>	cart/templates/cart/detail.html:15
Product	Producto	<input type="checkbox"/>	cart/templates/cart/detail.html:16
Remove	Eliminar	<input type="checkbox"/>	cart/templates/cart/detail.html:18
Unit price	Precio unitario	<input type="checkbox"/>	cart/templates/cart/detail.html:19
Price	Precio	<input type="checkbox"/>	cart/templates/cart/detail.html:20
Continue shopping	Continuar comprando	<input type="checkbox"/>	cart/templates/cart/detail.html:55
Checkout	Comprar	<input type="checkbox"/>	cart/templates/cart/detail.html:57
English	Inglés	<input type="checkbox"/>	myshop/settings.py:177

Translating Rosetta itself

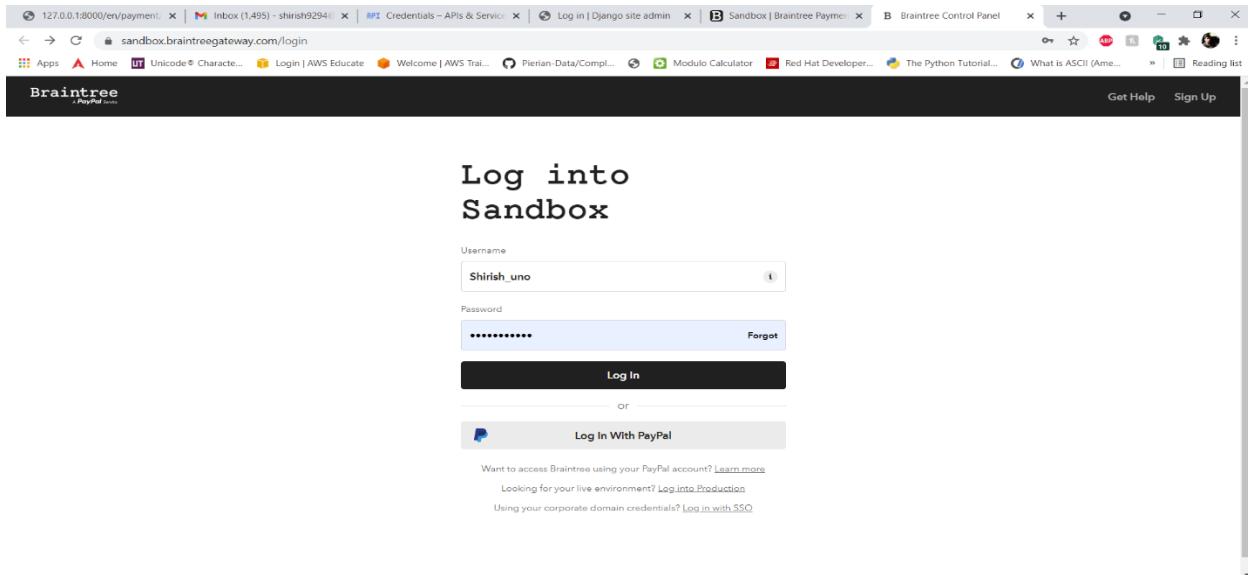
By default, Rosetta hides its own catalog files in the file selection interface (shown above.) If you would like to translate Rosetta to your own language:

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.

The screenshot shows the PyCharm IDE interface with the following details:

- Top Bar:** PyCharm, File, Edit, View, Navigate, Code, Refactor.
- Navigation Bar:** su2021team2-app > payment > views.py
- Left Sidebar:**
 - Project: su2021team2-app (~/Dropbox/My Mac (MacBook Pro)/Download)
 - 1: Project
 - 0: Commit
 - Pull Requests
- File Structure:**
 - 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
 - wsai.py
 - nppBackup
 - __init__.py
 - settings.py
 - urls.py
 - wsai.py

Braintree API



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

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

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

The screenshot shows a web browser window with multiple tabs open. The active tab is titled "Sandbox API Keys - Braintree Gateway". The page displays a table of API keys under the heading "Sandbox API Keys". There is one row in the table:

Public Key	Private Key	Last Used On	Actions
qxvh3xq8f89tw2kd	View	08/14/2021	Delete

Below the table is a button labeled "+ Generate New API Key".

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

The screenshot shows a code editor with several files open in tabs: README.md, settings.py, views.py, and urls.py. The settings.py file is the active tab and contains the following code:

```

USE_TZ = True

LANGUAGES = [
    ('en', _('English')),
    ('es', _('Spanish')),
]

# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/dev/howto/static-files/

CART_SESSION_ID = 'cart'

# Braintree settings - replace with your credentials after signing up for braintree account
BRAINTREE_MERCHANT_ID = 'g27pkkf6yqfw6whw' # Merchant ID
BRAINTREE_PUBLIC_KEY = 'qxvh3xq8f89tw2kd' # Public Key
BRAINTREE_PRIVATE_KEY = '6421d864a688391562747d7e24d410df' # Private key

Configuration.configure(
    Environment.Sandbox,
    BRAINTRIE_MERCHANT_ID,
    BRAINTRIE_PUBLIC_KEY,
    BRAINTRIE_PRIVATE_KEY
)

DEFAULT_AUTO_FIELD = 'django.db.models.AutoField'

```

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

And run the requirement package

pip install braintree

```

12
13
14     def payment_process(request):
15         order_id = request.session.get('order_id')
16         order = get_object_or_404(Order, id=order_id)
17
18         if request.method == 'POST':
19             # retrieve nonce
20             nonce = request.POST.get('payment_method_nonce', None)
21             # create and submit transaction
22             result = braintree.Transaction.sale(
23                 {
24                     'amount': '{:.2f}'.format(order.get_total_cost()),
25                     'payment_method_nonce': nonce,
26                     'options': {
27                         'submit_for_settlement': True
28                     }
29                 }
30             )
31             if result.is_success:
32                 # mark the order as paid
33                 order.paid = True
34                 # store the unique transaction id
35                 order.braintree_id = result.transaction.id
36                 order.save()
37                 subject = f'Order {order_id}'
38                 message = f'Your order no {order_id}, with items costing ${order.get_total_cost()}, has been placed \
            successfully.'
            # email_attach('order_{}.pdf'.format(order.id), out.getvalue(), 'application/pdf')

```

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',
'storage',
]

```

and then add rest_framework in the installed apps

```

from rest_framework import serializers
from .models import *
from orders.models import Order

class cSerializer(serializers.ModelSerializer):
    class Meta:
        model = Category
        fields = '__all__'

class uSerializer(serializers.ModelSerializer):
    class Meta:
        model = User
        fields = ('id', 'username', 'email')

class pSerializer(serializers.ModelSerializer):
    class Meta:
        model = Product
        fields = ('category', 'name', 'price')
        # fields = '__all__'

class oSerializer(serializers.ModelSerializer):
    class Meta:
        model = Order

```

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.

```

class category_info(APIView):
    permission_classes = [IsAdminUser]

    def get(self, request):
        c1 = Category.objects.all()
        s1 = cSerializer(c1, many=True)
        return Response(s1.data)

class user_info(APIView):
    permission_classes = [IsAdminUser]

    def get(self, request):
        c1 = User.objects.all()
        s1 = uSerializer(c1, many=True)
        return Response(s1.data)

class product_infos(APIView):
    permission_classes = [IsAdminUser]

    def get(self, request):
        c1 = Product.objects.all()
        s1 = pSerializer(c1, many=True)
        return Response(s1.data)

```

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.

The screenshot shows a browser window titled "Order Info - Django REST". The address bar contains the URL "s2-8380.herokuapp.com/en/api/order_info". The main content area is titled "Django REST framework" and "Order Info". A "GET /en/api/order_info" button is visible. Below it, an error message is displayed:
HTTP 403 Forbidden
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept
{
 "detail": "Authentication credentials were not provided."
}

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

The screenshot shows a browser window titled "Order Info - Django REST". The address bar contains the URL "s2-8380.herokuapp.com/en/api/order_info". The main content area is titled "Django REST framework" and "Order Info". A "GET /en/api/order_info" button is visible. Below it, a success message is displayed:
HTTP 200 OK
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept
[
 {
 "id": 4,
 "first_name": "Kavya",
 "last_name": "Kavya",
 "email": "kavyaerdin@gmail.com",
 "address": "415H 145TH PLZ",
 "postal_code": "68116",
 "city": "Omaha",
 "phone": "6823759000"
 },
 {
 "id": 3,
 "first_name": "Shirish",
 "last_name": "Srinivasa",
 "email": "shirish924@gmail.com",
 "address": "880 S 70TH PLZ",
 "postal_code": "68106",
 "city": "Omaha",
 "phone": "5312395064"
 },
 {
 "id": 2,
 "first_name": "Shirish",
 "last_name": "Srinivasa",
 "email": "shirish924@gmail.com",
 "address": "880 S 70TH PLZ",
 "postal_code": "68106",
 "city": "Omaha",
 "phone": "5312395064"
 }
]

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/categories_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:

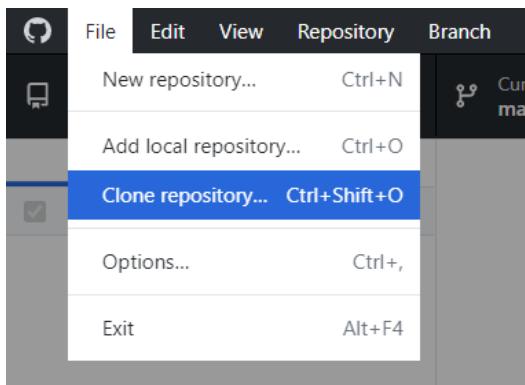
We can download the code from GitHub by using the following link:

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

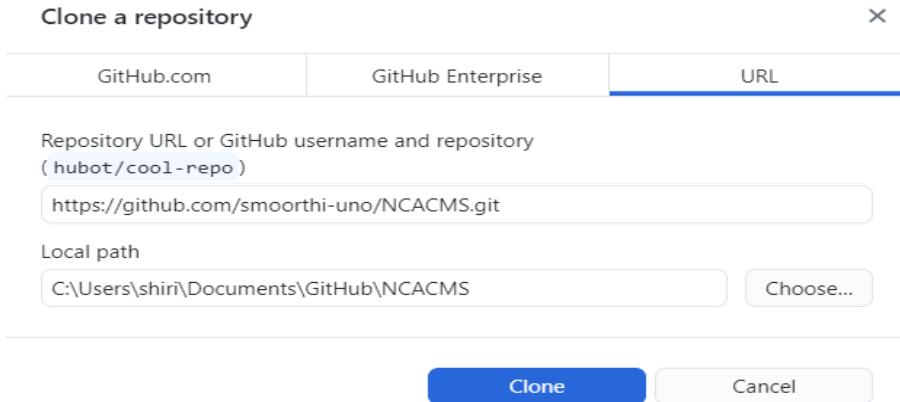
you need to copy the URL in the repository and clone it by using GitHub desktop. So, you can run and test the code on your local computer.

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

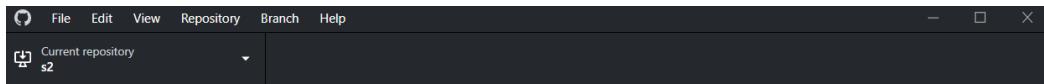
You can also download the files as a zip file and store it on your system.



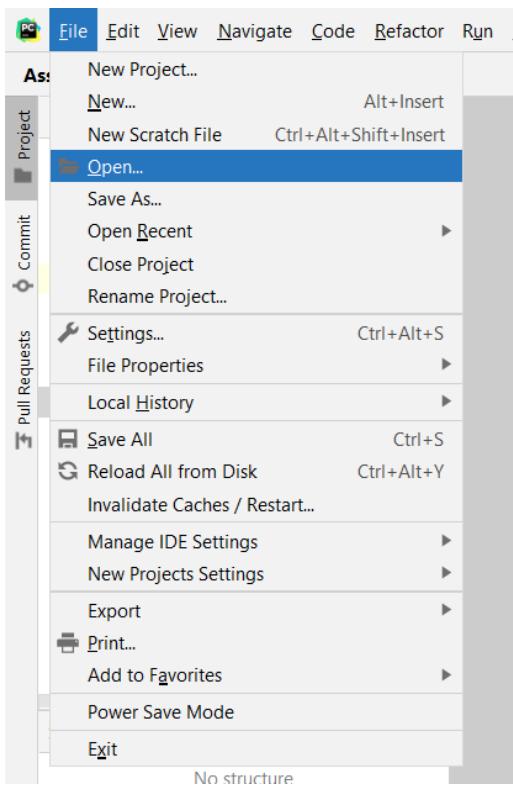
Open GitHub desktop application then click on File -> Clone Repository, then a popup window will appear.



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.

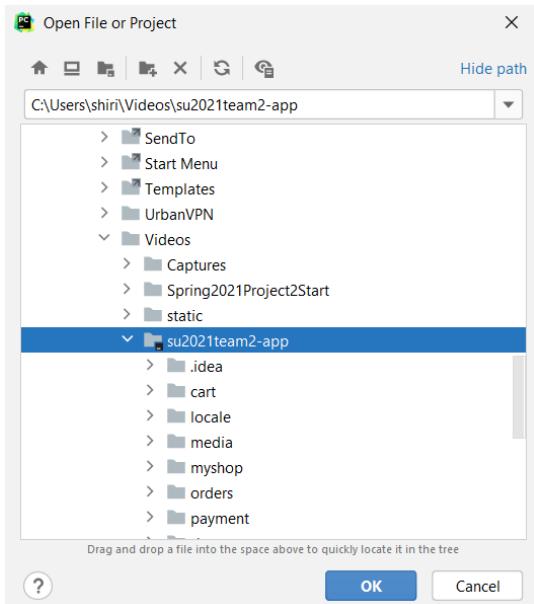


The Cloning process will start, and the files will be cloned to your local system.

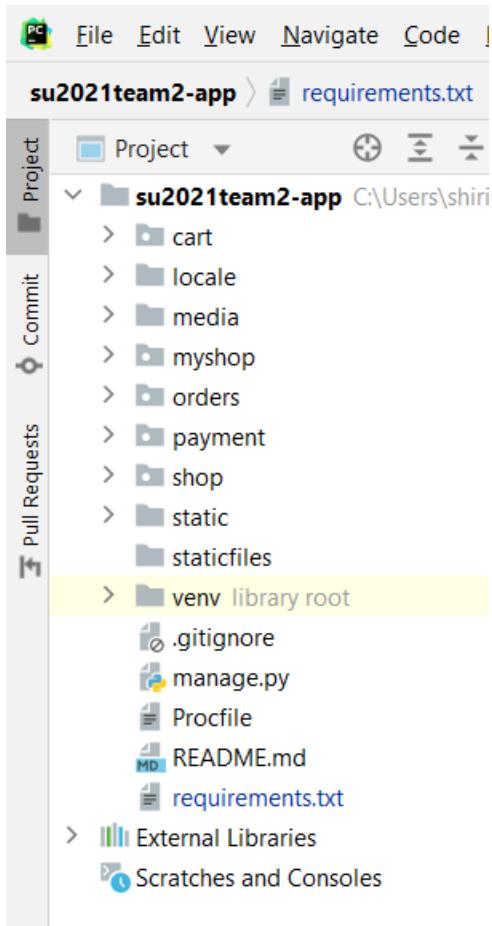


Now you need to open PyCharm and click on File -> Open.

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

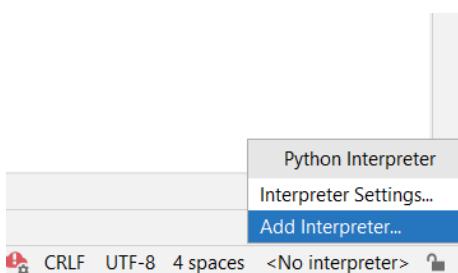


In the navigation windows locate where you have saved the web application files, select the folder, and click on OK. These files will get loaded into PyCharm.

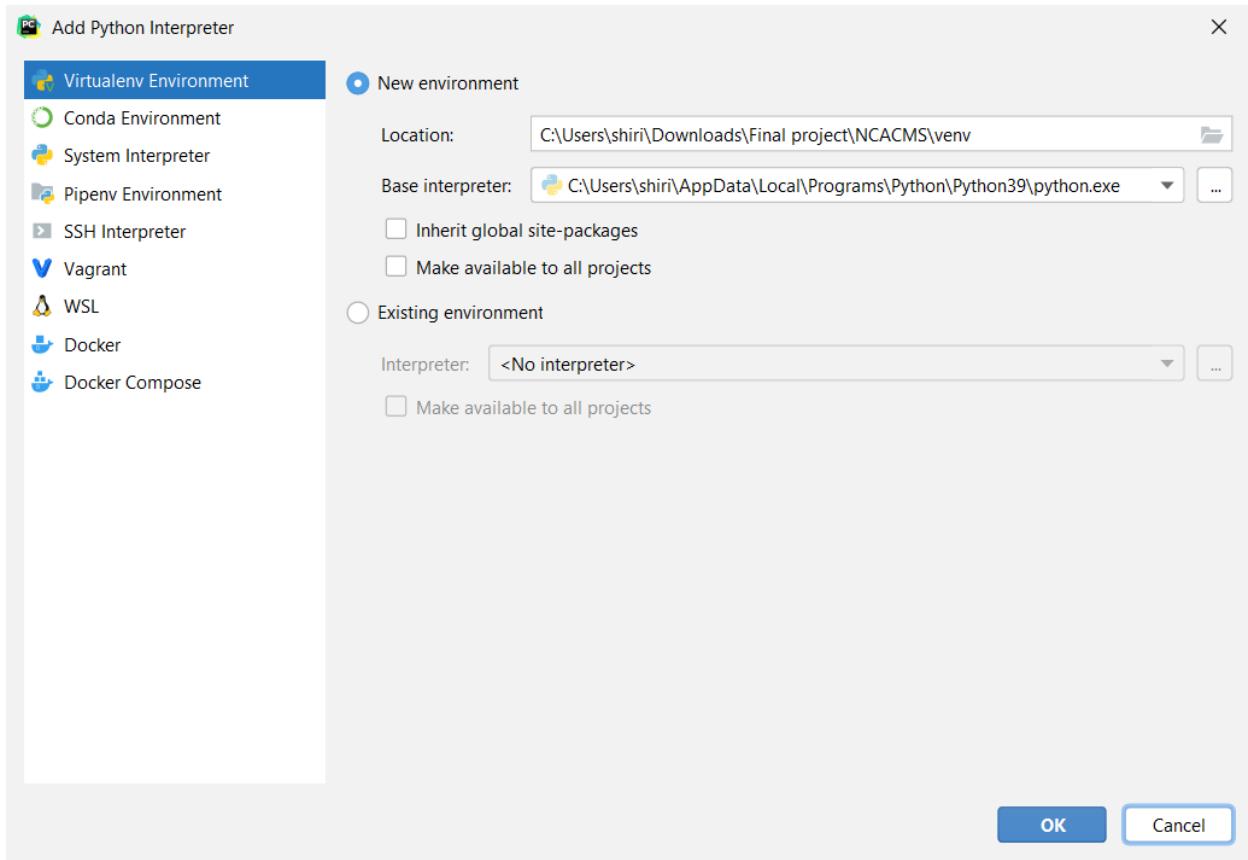


Now Click on the Project option which is located on the left side of the PyCharm application, it will show us the web application files.

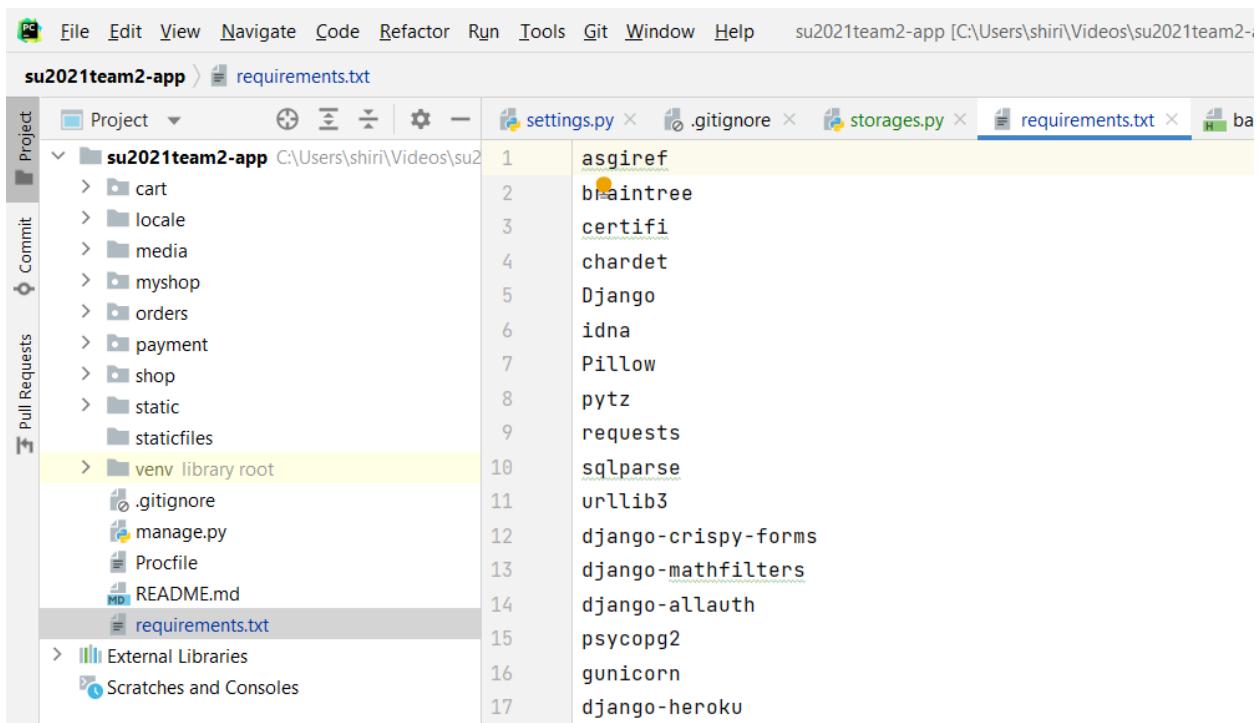
Now you need to create a virtual environment to run the web application.



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.

The screenshot shows the PyCharm terminal window titled 'Terminal: Local'. The terminal output shows the command 'pip install -r requirements.txt' being run in a virtual environment ('(venv) C:\Users\shiri\Downloads\Final project\NCACMS>'). The process of installing packages is shown:

```
Microsoft Windows [Version 10.0.19042.928]
(c) Microsoft Corporation. All rights reserved.

(virtualenv) C:\Users\shiri\Downloads\Final project\NCACMS>pip install -r requirements.txt
Collecting selenium
  Using cached selenium-3.141.0-py2.py3-none-any.whl (904 kB)
Collecting Django==3.0.7
  Using cached Django-3.0.7-py3-none-any.whl (7.5 MB)
Collecting django-crispy-forms==1.9.1
  Using cached django_crispy_forms-1.9.1-py2.py3-none-any.whl (108 kB)
Collecting gunicorn==20.0.4
```

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.

The screenshot shows the PyCharm IDE interface. The left sidebar displays the project structure under 'su2021team2-app'. The 'myshop' directory contains several files: __init__.py, settings.py (which is selected), storage.py, urls.py, and wsgi.py. Other directories include cart, locale, media, orders, payment, shop, static, and staticfiles. A 'venv library root' folder is also present. The right pane shows the code editor for 'settings.py'. The code is as follows:

```
# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'w#c2p$9qpf@%_f14c9g1%4lm*o8s2ht^*+4mx^77l($)l!1!6'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = []

AUTH_USER_MODEL = 'shop.User'

# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages'
]
```

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`

Terminal: Local +

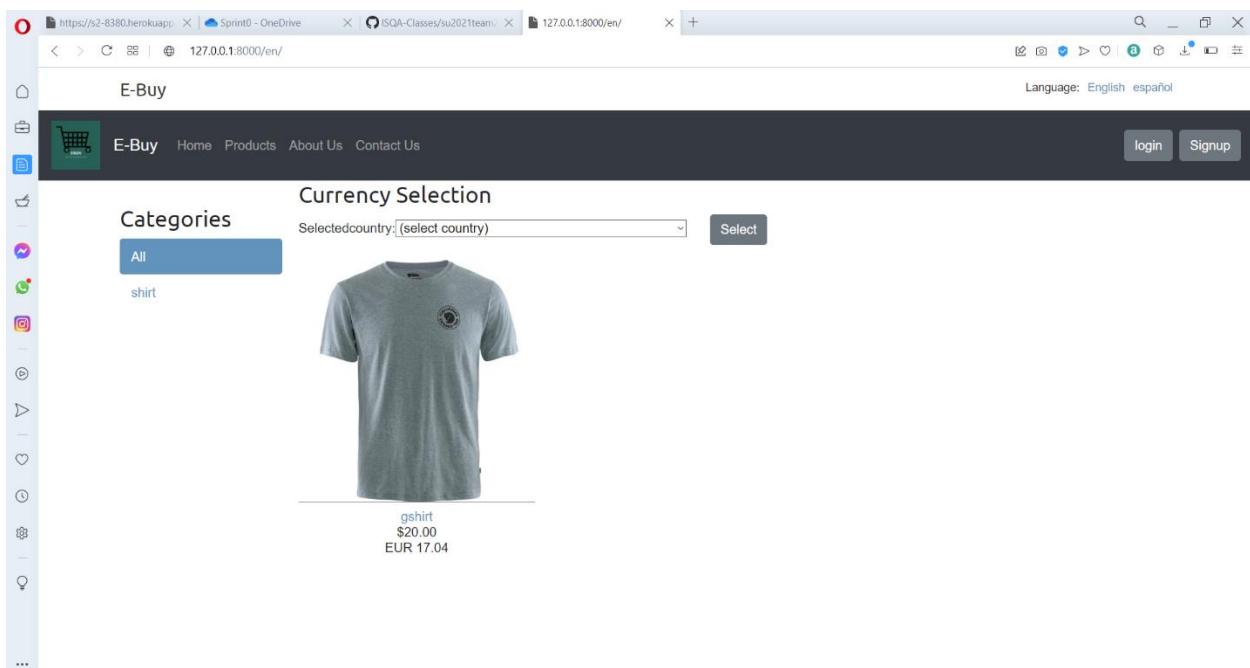
```
(venv) C:\Users\shiri\Downloads\Final project\NCACMS>python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
April 24, 2021 - 13:01:14
Django version 3.0.7, using settings 'ISQA8210_T1_NCACMS.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

Git TODO Problems Terminal Python Packages Python Console

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

The screenshot shows the Django admin interface at 127.0.0.1:8000/en/admin/. The top navigation bar includes links for payment/done, inbox, credentials, and site administration. The main content area is organized into sections: AUTHENTICATION AND AUTHORIZATION (Groups), ORDERS (Orders), SHOP (Categories, Customers, Products, Users), SITES (Sites), and SOCIAL ACCOUNTS (Social accounts, Social application tokens, Social applications). Each section has 'Add' and 'Change' buttons. A sidebar on the right lists recent actions: grey shirt (Product), red Shirt (Product), and Shirt (Category).

Now login to the admin panel and select sites.

The screenshot shows the Django administration interface. On the left, there is a sidebar with categories: ACCOUNTS, AUTHENTICATION AND AUTHORIZATION, ORDERS, SHOP, SITES, and SOCIAL ACCOUNTS. Under SITES, 'Sites' is highlighted. The main content area is titled 'Select site to change'. It contains a search bar and a table with one row. The table has columns for 'DOMAIN NAME' and 'DISPLAY NAME'. A checkbox next to 'example.com' is checked. The 'DISPLAY NAME' column shows 'example.com'. At the top right of the main area, there is a button labeled 'ADD SITE +'. The top navigation bar shows the URL '127.0.0.1:8000/en/admin/sites/site/'.

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

The screenshot shows the 'Change site' page for 'example.com'. The URL in the address bar is '127.0.0.1:8000/en/admin/sites/site/2/change/'. The main content area is titled 'Change site' and shows the current values for 'Domain name:' (127.0.0.1:8000) and 'Display name:' (127.0.0.1:8000). Below these fields are three buttons: 'Delete', 'Save and add another', 'Save and continue editing', and 'SAVE'. The top navigation bar shows the URL 'example.com | Change site | Django'.

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.

The site "127.0.0.1:8000" was changed successfully.

Action:	DOMAIN NAME	DISPLAY NAME
<input type="checkbox"/>	127.0.0.1:8000	127.0.0.1:8000

1 site

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

Home : Social Accounts : Social applications

Select social application to change

ADD SOCIAL APPLICATION +

0 social applications

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

The screenshot shows the Django Admin interface for adding a social application. The left sidebar has a 'SOCIAL ACCOUNTS' section with 'Social accounts' selected. The main form is titled 'Add social application' and includes fields for Provider (set to Google), Name (ebuy), Client id (96830223220-jau19ajgh55ca1k5n5p6hi2vs5mp2j9.apps.googleusercontent.com), Secret key (e8uYTF2alHlVKn07-OriNnlv), and Key (empty). The Sites section shows 'Available sites' with '127.0.0.1:8000' moved to 'Chosen sites'.

Here we will add the social application details as follows:

Provider: Google

Name: ebuy

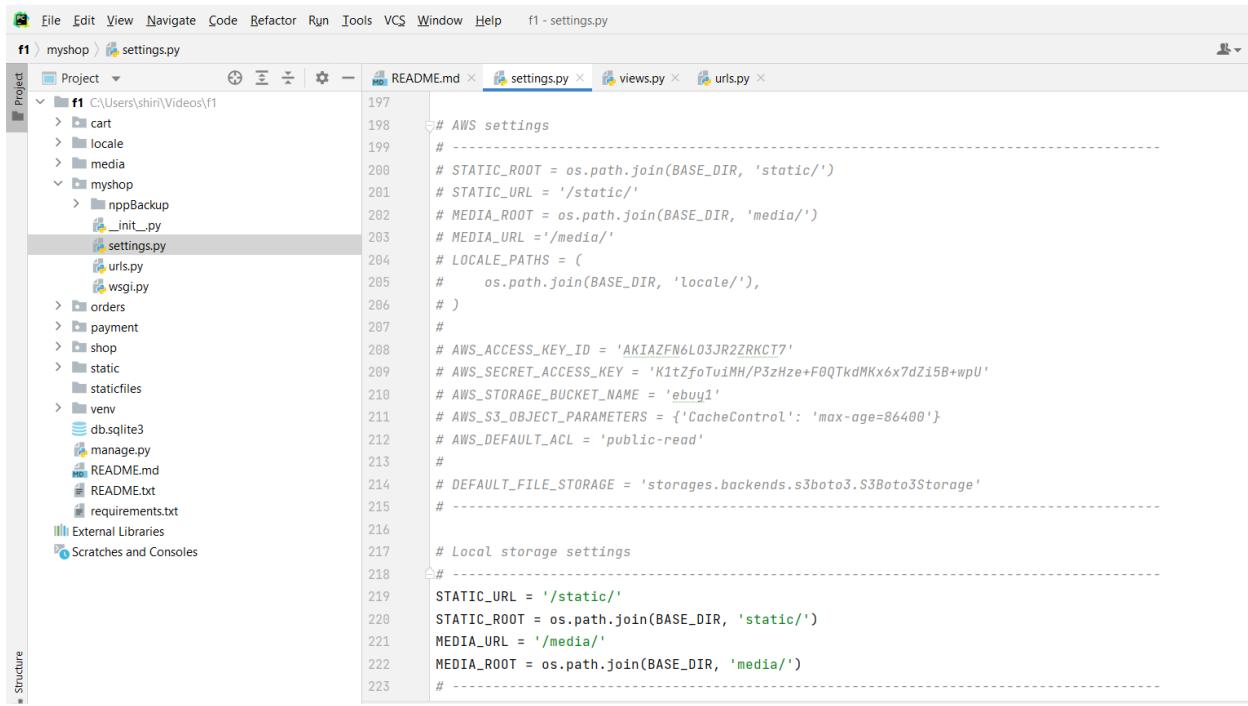
Client id: 96830223220-jau19ajgh55ca1k5n5p6hi2vs5mp2j9.apps.googleusercontent.com

Secret key: e8uYTF2alHlVKn07-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:



The screenshot shows the PyCharm IDE interface with the 'settings.py' file open. The project structure on the left includes 'cart', 'locale', 'media', 'myshop' (which contains 'nppBackup', '_init_.py', 'settings.py', 'urls.py', and 'wsgi.py'), 'orders', 'payment', 'shop', 'static', 'staticfiles', 'venv', 'db.sqlite3', 'manage.py', 'README.md', 'README.txt', and 'requirements.txt'. The 'External Libraries' and 'Scratches and Consoles' sections are also visible. The code in 'settings.py' is as follows:

```
197 # AWS settings
198 # -----
199 # STATIC_ROOT = os.path.join(BASE_DIR, 'static/')
200 # STATIC_URL = '/static/'
201 # MEDIA_ROOT = os.path.join(BASE_DIR, 'media/')
202 # MEDIA_URL = '/media/'
203 # LOCALE_PATHS =
204 #     os.path.join(BASE_DIR, 'locale/'),
205 # )
206 #
207 # AWS_ACCESS_KEY_ID = 'AKIAZFN6L03JR2ZRKCT7'
208 # AWS_SECRET_ACCESS_KEY = 'K1t2foTuiMH/P3zHze+F0QTkdMKx6x7dZ15B+wpuU'
209 # AWS_STORAGE_BUCKET_NAME = 'ebuy1'
210 # AWS_S3_OBJECT_PARAMETERS = {'CacheControl': 'max-age=86400'}
211 # AWS_DEFAULT_ACL = 'public-read'
212 #
213 # DEFAULT_FILE_STORAGE = 'storages.backends.s3boto3.S3Boto3Storage'
214 #
215 # Local storage settings
216 #
217 STATIC_URL = '/static/'
218 STATIC_ROOT = os.path.join(BASE_DIR, 'static/')
219 MEDIA_URL = '/media/'
220 MEDIA_ROOT = os.path.join(BASE_DIR, 'media/')
```

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.

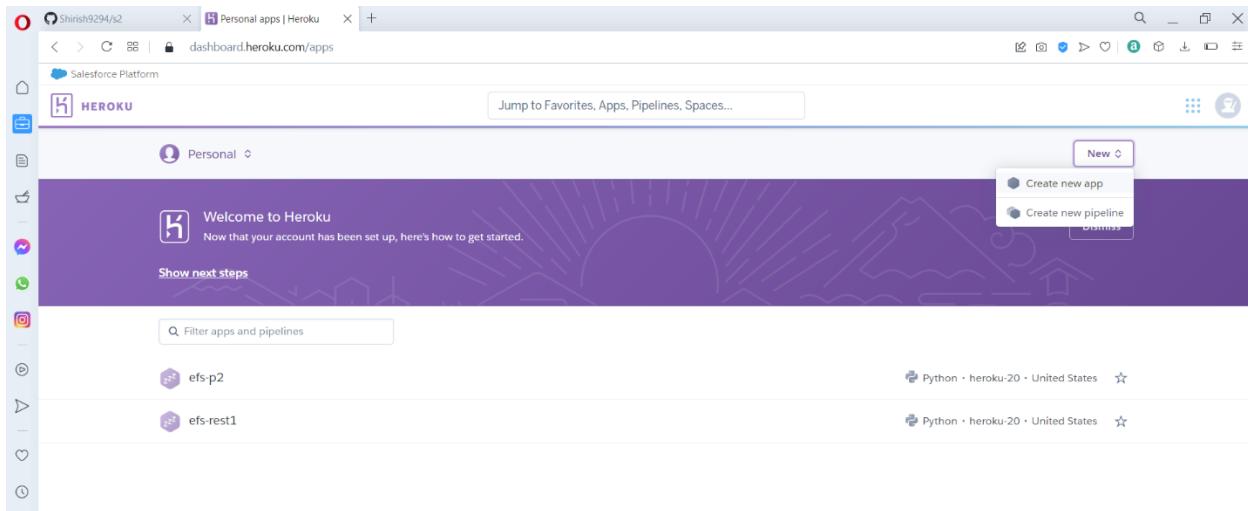
In the AWS settings

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

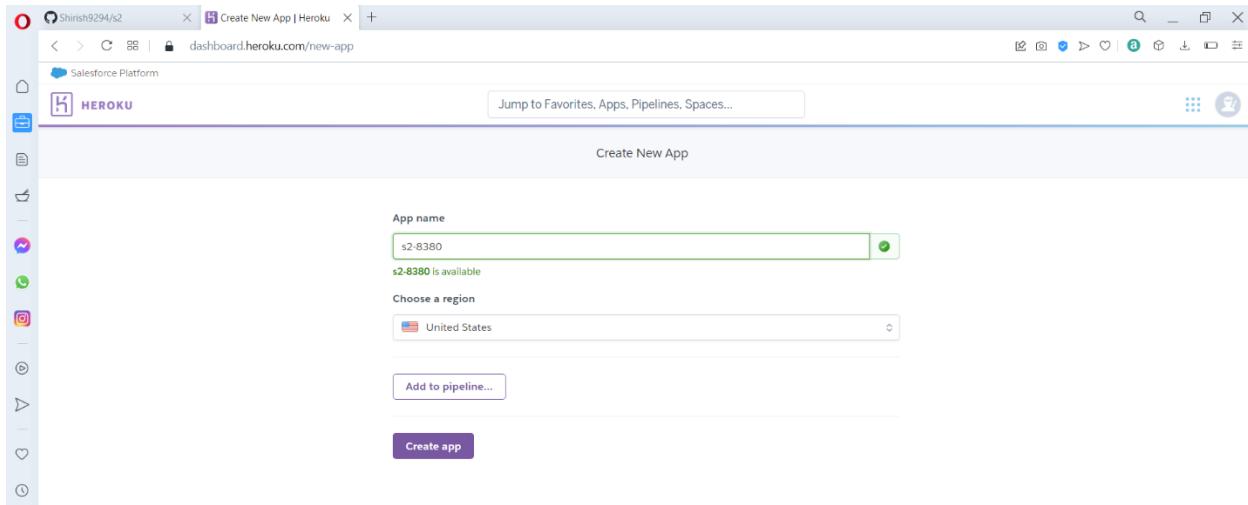
The AWS access keys and credentials in the settings are working and valid.

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.



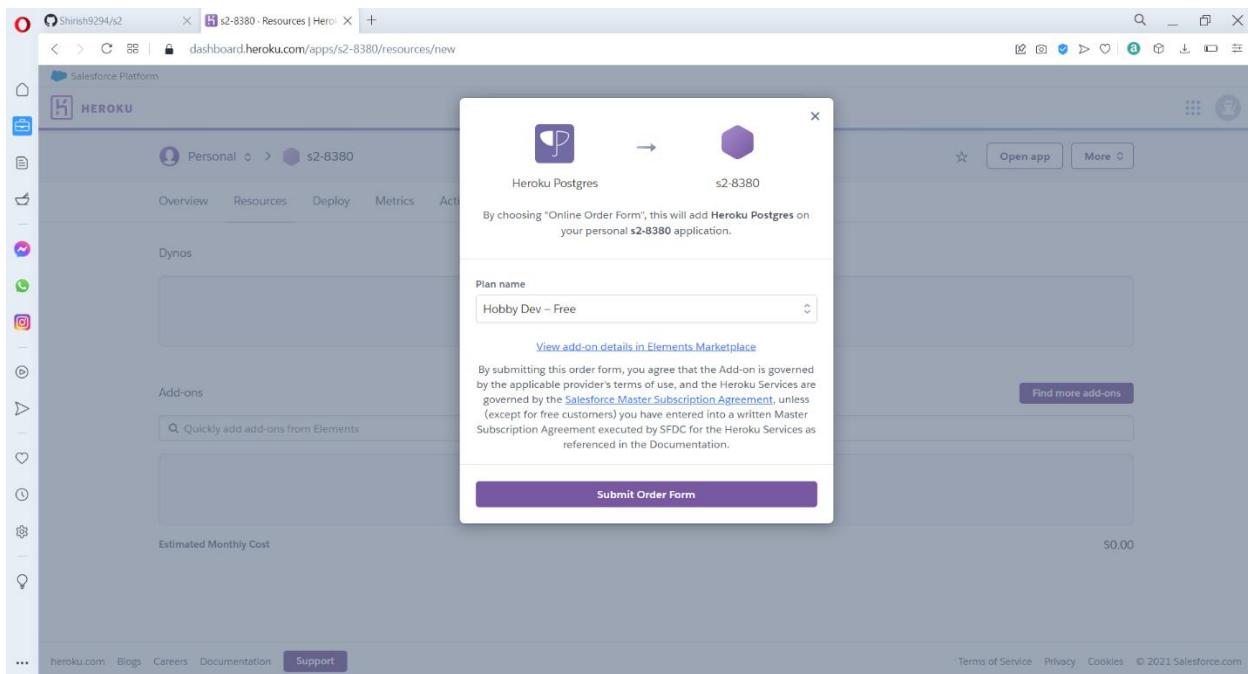
Enter your “App name”, Heroku will let you know if the name is available, otherwise you might have to enter a different name, then click on create app.

The screenshot shows the Heroku application dashboard for an app named 's2-8380'. The main navigation bar includes 'Overview', 'Resources', 'Deploy', 'Metrics', 'Activity', 'Access', and 'Settings'. On the left, there's a sidebar with various icons. The 'Deploy' section contains two main sections: 'Add this app to a pipeline' and 'Add this app to a stage in a pipeline to enable additional features'. It also includes a 'Choose a pipeline' dropdown. Below this, the 'Deployment method' section offers 'Heroku Git' (using Heroku CLI), 'GitHub' (Connect to GitHub), and 'Container Registry' (using Heroku CLI). Further down, there are sections for 'Deploy using Heroku Git' (using git in the command line or a GUI tool) and 'Install the Heroku CLI' (with download and login instructions).

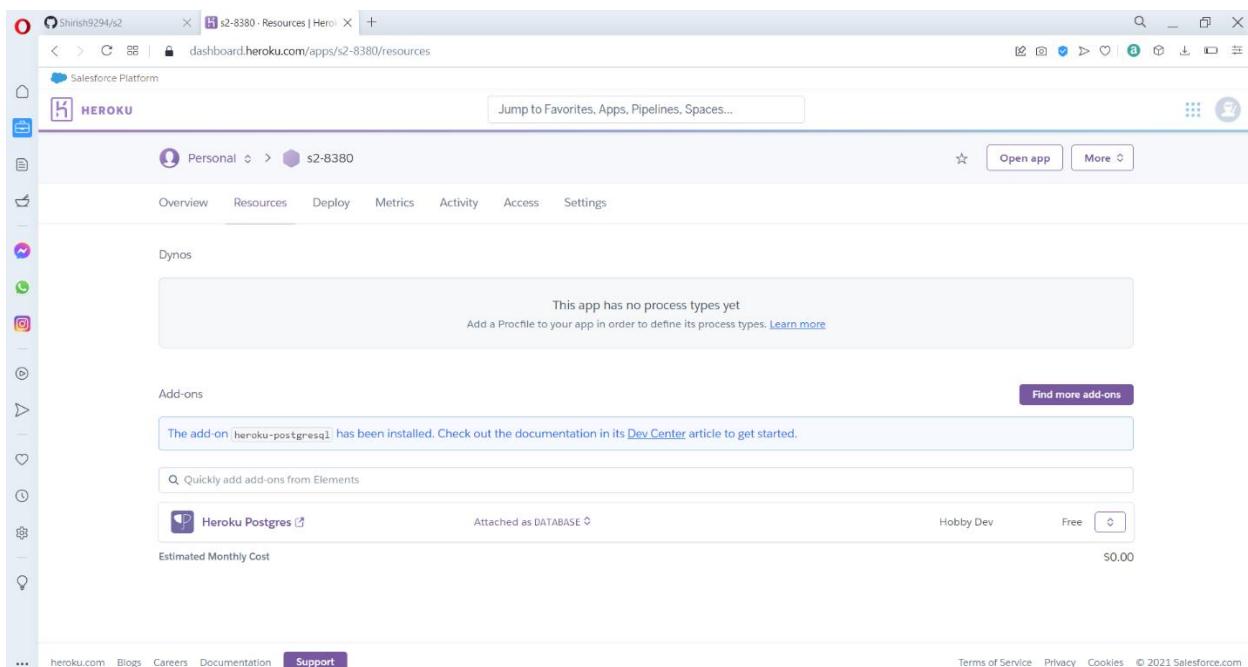
This how the application dashboard will look, now we need to create a Postgres database under the resources option.

The screenshot shows the 'Resources' tab of the Heroku application dashboard for 's2-8380'. The top navigation bar remains the same. The 'Resources' tab has several sections: 'Dynos' (which states 'This app has no process types yet'), 'Add-ons' (with a search bar and a 'Find more add-ons' button), and 'Estimated Monthly Cost' (showing \$0.00). At the bottom, there are links for 'heroku.com', 'Blogs', 'Careers', 'Documentation', 'Support', 'Terms of Service', 'Privacy', 'Cookies', and a copyright notice for 2021 Salesforce.com.

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.

The screenshot shows the Heroku dashboard for app `s2-8380`. In the center, there's a section titled "Deployment method" with three options: "Heroku Git" (selected), "GitHub" (disabled), and "Container Registry". Below this, under "Deploy using Heroku Git", it says "Use git in the command line or a GUI tool to deploy this app." To the right, there's a "Install the Heroku CLI" section with a link to download and install the CLI. At the bottom, there's a terminal-like input field containing the command `$ heroku login`.

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.

The screenshot shows the Heroku dashboard for app `s2-8380` with the GitHub deployment method selected. It displays a "Connect to GitHub" section where users can search for a repository to connect. A search bar shows "Shirish9294" and "s2". A "Search" button is visible. Below the search bar, a note says "Missing a GitHub organization? [Ensure Heroku Dashboard has team access](#)". At the bottom, there's a "Connect" button.

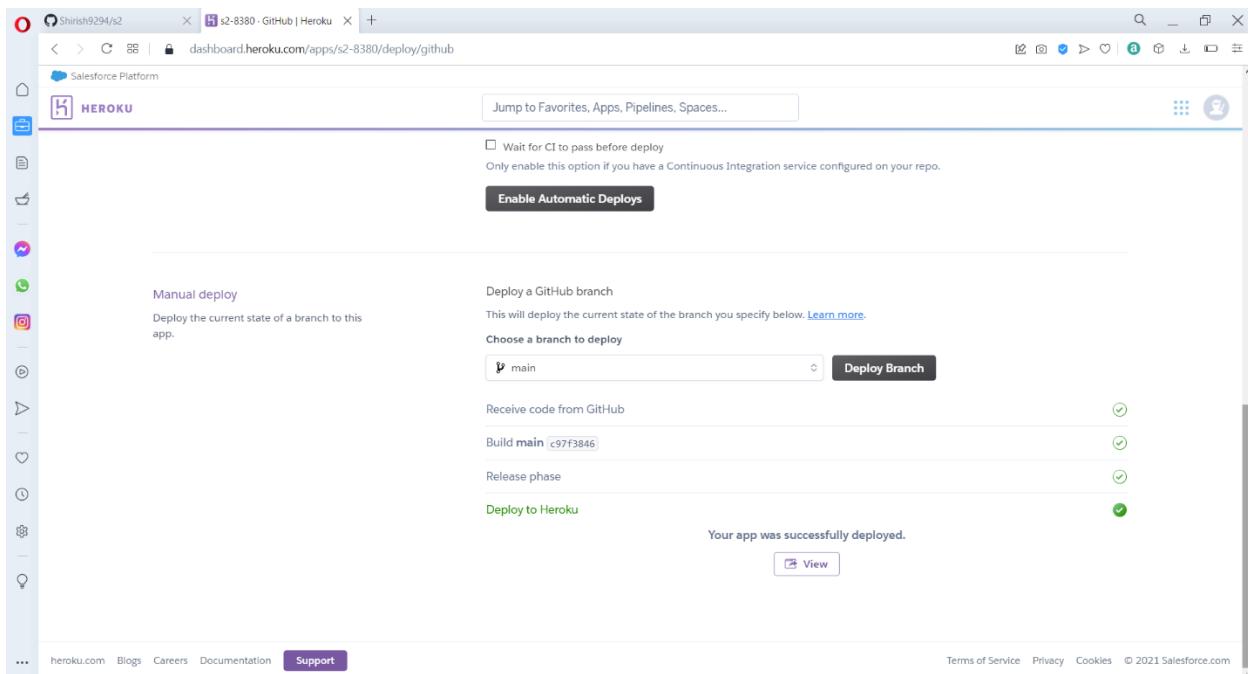
Now select the GitHub option and under that type the repository that you want to deploy and click on connect.

The screenshot shows the Heroku dashboard for the app 's2-8380'. In the 'Automatic deploys' section, there is a message: 'You can now change your main deploy branch from "master" to "main" for both manual and automatic deploys, please follow the instructions here.' Below this, there is a dropdown menu set to 'main' and a checkbox for 'Wait for CI to pass before deploy'. A button labeled 'Enable Automatic Deploys' is visible. In the 'Manual deploy' section, there is a dropdown menu also set to 'main' and a button labeled 'Deploy Branch'.

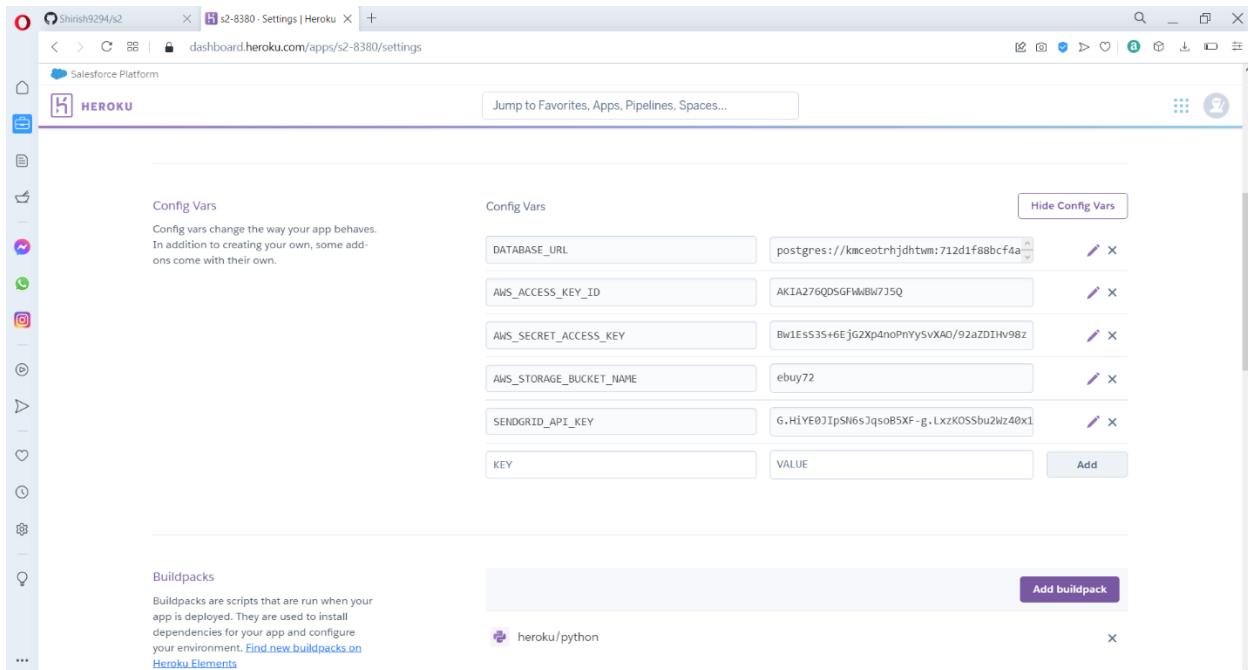
Now in the manual deploy option, select the main branch, then click on deploy branch.

The screenshot shows the Heroku dashboard for the app 's2-8380'. In the 'Manual deploy' section, there is a dropdown menu set to 'main' and a button labeled 'Deploy Branch'. Below this, a build log window is open, showing the deployment process. The log output includes: 'Building on the Heroku-20 stack', 'Determining which buildpack to use for this app', 'Python app detected', 'No Python version was specified. Using the buildpack default: python-3.9.6', 'Installing python-3.9.6', and 'Installing pip 20.2.4, setuptools 47.1.1 and wheel 0.36.2'. There is also a link to 'View build log'.

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.

```
MINGW64:/c/Users/shiri
```

```
shiri@LAPTOP-6H6F04PA MINGW64 ~ (master)
$ heroku git:remote -a s2-8380
set git remote heroku to https://git.heroku.com/s2-8380.git

shiri@LAPTOP-6H6F04PA MINGW64 ~ (master)
$ |
```

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.

```
shiri@LAPTOP-6H6F04PA MINGW64 ~ (master)
$ heroku run python manage.py migrate
```

Then we need to run the command

Heroku run python manage.py migrate

To migrate the data base.

```
shiri@LAPTOP-6H6F04PA MINGW64 ~ (master)
$ heroku run python manage.py createsuperuser
Running python manage.py createsuperuser on s2-8380... starting, run.7374 (Free)
Running python manage.py createsuperuser on s2-8380... connecting, run.7374 (Free)
Running python manage.py createsuperuser on s2-8380... up, run.7374 (Free)
Username: instructor
instructor
Error: Enter a valid username. This value may contain only letters, numbers, and
@./+/-/_ characters.
Username: instructor
instructor
Email address: groyce@unomaha.edu
groyce@unomaha.edu
Password: gounomavsl1
Password (again): gounomavsl1
Superuser created successfully.
```

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:

The screenshot shows the AWS Management Console homepage. At the top, there are three tabs: 'API Dashboard – APIs & Services', 'API Credentials – APIs & Services', and 'AWS Management Console'. The main content area is titled 'AWS Management Console'. On the left, there's a sidebar titled 'AWS services' with sections for 'Recently visited services' (Billing, VPC, Support, IAM, Trusted Advisor) and 'All services'. Below that is a 'Build a solution' section with three options: 'Launch a virtual machine' (With EC2, 2-3 minutes), 'Build a web app' (With Elastic Beanstalk, 6 minutes), and 'Build using virtual servers' (With Lightsail, 1-2 minutes). To the right, there are two boxes: 'Stay connected to your AWS resources on-the-go' (noting support for four additional regions) and 'Explore AWS' (sections for 'AWS Certification', 'Amazon DynamoDB', and 'AWS Free Digital Training'). At the bottom, there are links for 'Feedback', 'English (US)', 'Privacy Policy', 'Terms of Use', and 'Cookie preferences'.

Login to the AWS console.

The screenshot shows the S3 Management Console homepage. At the top, there are three tabs: 'API Dashboard – APIs & Services', 'API Credentials – APIs & Services', and 'S3 Management Console'. The main content area is titled 'Amazon S3'. On the left, there's a sidebar with sections for 'Buckets', 'Access Points', 'Object Lambda Access Points', 'Batch Operations', 'Access analyzer for S3', 'Storage Lens' (Dashboards, AWS Organizations settings), 'Feature spotlight', and 'AWS Marketplace for S3'. The main area has a banner about AWS Transfer Family modernizing file transfers to and from S3. It includes an 'Account snapshot' section with a 'View Storage Lens dashboard' button, a 'Buckets (0)' table with columns for Name, AWS Region, Access, and Creation date, and a 'Create bucket' button. At the bottom, there are links for 'Feedback', 'English (US)', 'Privacy Policy', 'Terms of Use', and 'Cookie preferences'.

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

Bucket name: ebuy72

AWS Region: US East (N. Virginia) us-east-1

Block all public access

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

Server-side encryption: Disable

Disable

Enable

Object Lock: Disable

Object Lock works only in versioned buckets. Enabling Object Lock automatically enables Bucket Versioning.

After creating the bucket you can upload files and folders to the bucket, and configure additional bucket settings.

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

The screenshot shows the AWS S3 Management Console. A green success message at the top states: "Successfully created bucket 'ebuy72'. To upload files and folders, or to configure additional bucket settings choose View details." Below this, a blue banner from AWS Transfer Family promotes modernizing recurring file transfers. The main area displays an "Account snapshot" with a "Storage lens" link and a "View Storage Lens dashboard" button. A table titled "Buckets (1) Info" lists the single bucket "ebuy72" with details: Name (ebuy72), AWS Region (US East (N. Virginia) us-east-1), Access (Bucket and objects not public), and Creation date (August 12, 2021, 06:21:08 (UTC-05:00)).

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

The screenshot shows the AWS S3 Management Console for the "ebuy72" bucket. The "Permissions" tab is selected in the navigation bar. Under "Permissions overview", it shows "Access" and "Bucket and objects not public". In the "Block public access (bucket settings)" section, there is a note about public access being granted through ACLs, policies, and access point policies. It also notes that turning off all public access applies only to this bucket and its access points. A "Edit" button is present. Below it, a "Block all public access" setting is shown as "On".

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.

The screenshot shows the AWS S3 service dashboard. On the left, there's a sidebar with 'Buckets' selected, showing options like Access Points, Object Lambda Access Points, Batch Operations, and Access analyzer for S3. Below that is 'Storage Lens' with Dashboards and AWS Organizations settings. A 'Feature spotlight' section is also present. At the bottom of the sidebar, it says 'AWS Marketplace for S3'. The main content area is titled 'Edit cross-origin resource sharing (CORS)'. It includes a 'Cross-origin resource sharing (CORS)' section with a note about the JSON configuration. Below that is a code editor containing the following JSON:

```

1  [
2   {
3     "AllowedHeaders": [
4       "*"
5     ],
6     "AllowedMethods": [
7       "GET",
8       "HEAD",
9       "POST",
10      "PUT"
11    ],
12    "AllowedOrigins": [
13      "*"
14    ],
15    "ExposeHeaders": []
16  }
17 ]
18

```

Now in the cross-origin resource sharing add:

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

IAM dashboard

Sign-In URL for IAM users in this account
https://755813814667.sigin.aws.amazon.com/console | Customize

IAM resources

- Users: 0
- User groups: 0
- Roles: 2
- Identity providers: 0

Best practices

- Grant least privilege access
- Use AWS Organizations
- Enable Identity federation
- Enable MFA
- Rotate credentials regularly
- Enable IAM Access Analyzer

What's new

- AWS Identity and Access Management now makes it easier to relate a user's IAM role activity to their corporate identity
- IAM Access Analyzer makes it easier to implement least privilege permissions by generating IAM policies based on access activity

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

Introducing the new Users list experience

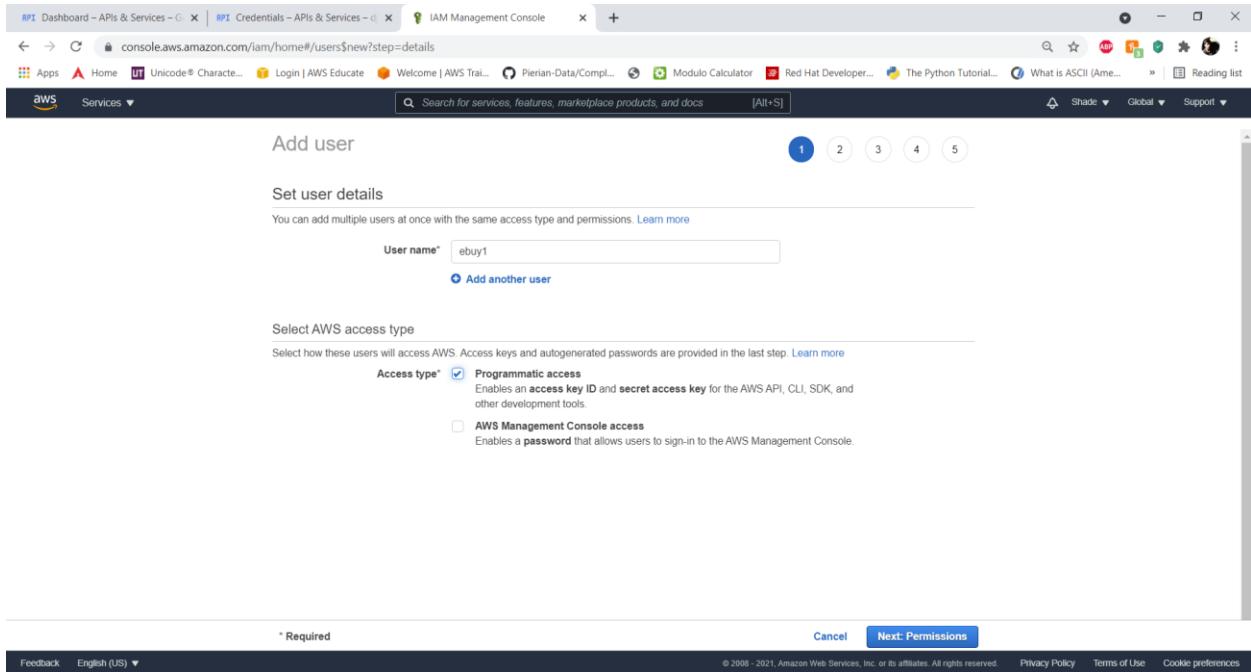
We've redesigned the Users list experience to make it easier to use. [Let us know what you think.](#)

Users (0) Info

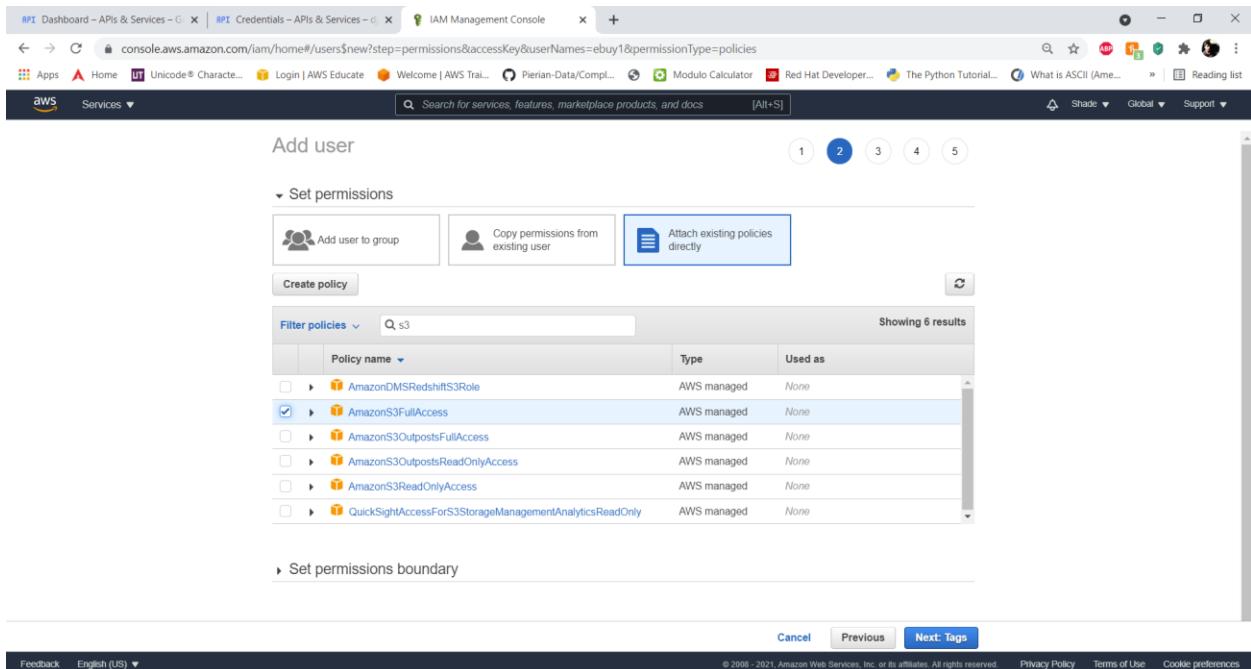
An IAM user is an identity with long-term credentials that is used to interact with AWS in an account.

User name	Groups	Last activity	MFA	Password age	Active key age
No resources to display					

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.

User name: ebuy1

AWS access type: Programmatic access - with an access key

Permissions boundary: Permissions boundary is not set

Type	Name
Managed policy	AmazonS3FullAccess

Tags: No tags were added.

Cancel Previous Create user

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

Success: You successfully created the users shown below. You can view and download user security credentials. You can also email users instructions for signing in to the AWS Management Console. This is the last time these credentials will be available to download. However, you can create new credentials at any time.

Users with AWS Management Console access can sign-in at: <https://755813814667.signin.aws.amazon.com/console>

Download .csv

User	Access key ID	Secret access key
ebuy1	AKIA276QDSGFWWB7J5Q	***** Show

Close

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