Candidate No: 260032

Webapps Report

> During the development of this project, I encountered several issues with the IntelliJ IDE. I began the project with IntelliJ but later on in development, my IDE faced a lot of errors that could not be ultimately fixed. I sought help from coursemates and our T.A, Kannen Joshi, but the different errors it produced each time could not be completely fixed. As a result I had no choice but to complete the remainder of the project with different IDE to meet the deadline of the project.

Introduction

This project aims to create an online payment system that allows users to transfer money between their online accounts. Unlike traditional online payment services, the assumption is made that users do not have any bank account connections and start with a fixed amount of money upon registration. To register, users are required to provide their basic information such as username, first and last name, email address, and password. Users can select their account currency during registration, and the system will automatically make the appropriate conversion to assign the right initial amount of money.

Users can make direct payments to other registered users and can also request payments from other users. The system facilitates these transactions within a single Django transaction, and users can check for notifications regarding payments and requests in their accounts. Users can access all their transactions, including sent and received payments and requests for payments, as well as their current account balance.

The system administrator has access to all user accounts and transactions. In addition, the project requires a separate RESTful web service to implement currency conversion, with exchange rates being statically assigned and hard-coded in the source code.

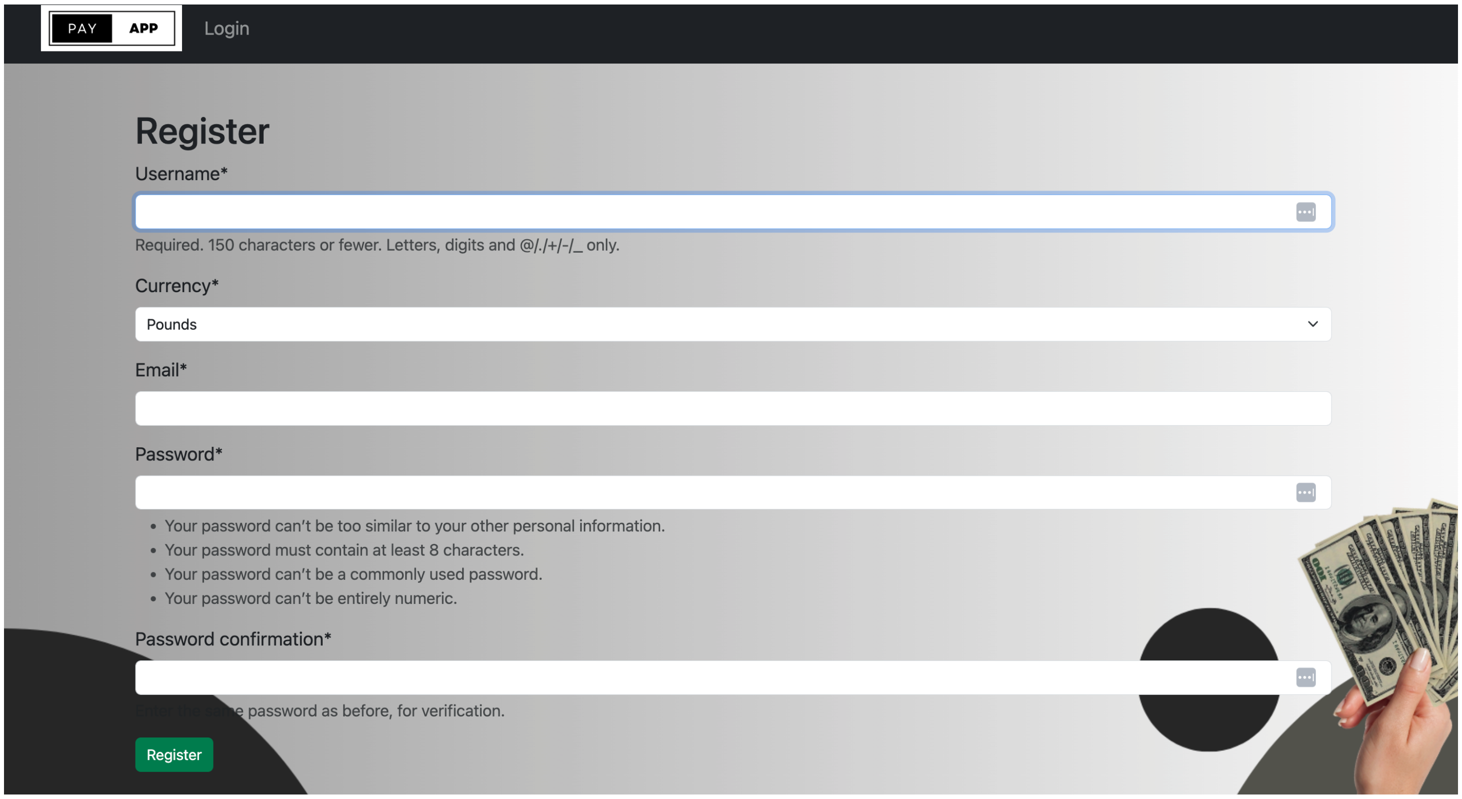
Overall, this project aims to provide a simple and efficient online payment system for users, with the ability to track transactions and account balances. The use of Django and a RESTful web service for currency conversion ensures scalability and modularity for future development.

1 - Presentation Layer

The application has five main functionalities. Firstly, it enables registered users to transfer money to other registered users. This core feature is the primary purpose of the application. Secondly, users can also request funds from other registered users. This feature allows users to initiate a transaction by requesting funds from another user. Thirdly, administrators have the privilege to see all the user accounts on the platform. This feature allows the admin to keep track of all users on the platform. Fourthly, the admin can also see all the transactions made between users. This feature provides an overview of all the transactions taking place on the platform, allowing the admin to monitor and ensure the integrity of the system. Finally, the admin can register new users and other admins. This feature allows the admin to manage user accounts and create new admins when needed. These functionalities are essential to ensure that the system is secure, reliable, and easy to use for all stakeholders involved. The website can be accessed via the link:

```
https://127.0.0.1:8000/webapps2023/home/
```

Before you can access the homepage, you will be asked to register your credentials:



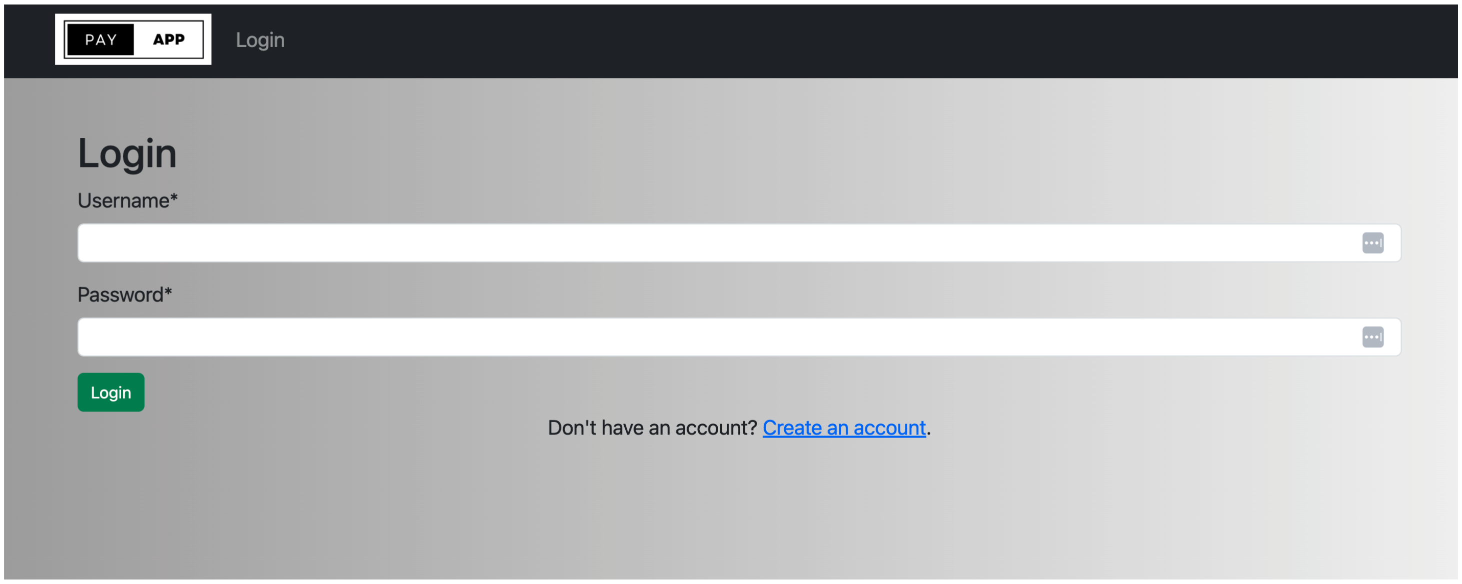
Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.

- Your password can't be too similar to your other personal information.
- Your password must contain at least 8 characters.
- Your password can't be a commonly used password.
- Your password can't be entirely numeric.

Enter the same password as before, for verification.

Figure 1

Following this you can login with the details you registered with:



Login

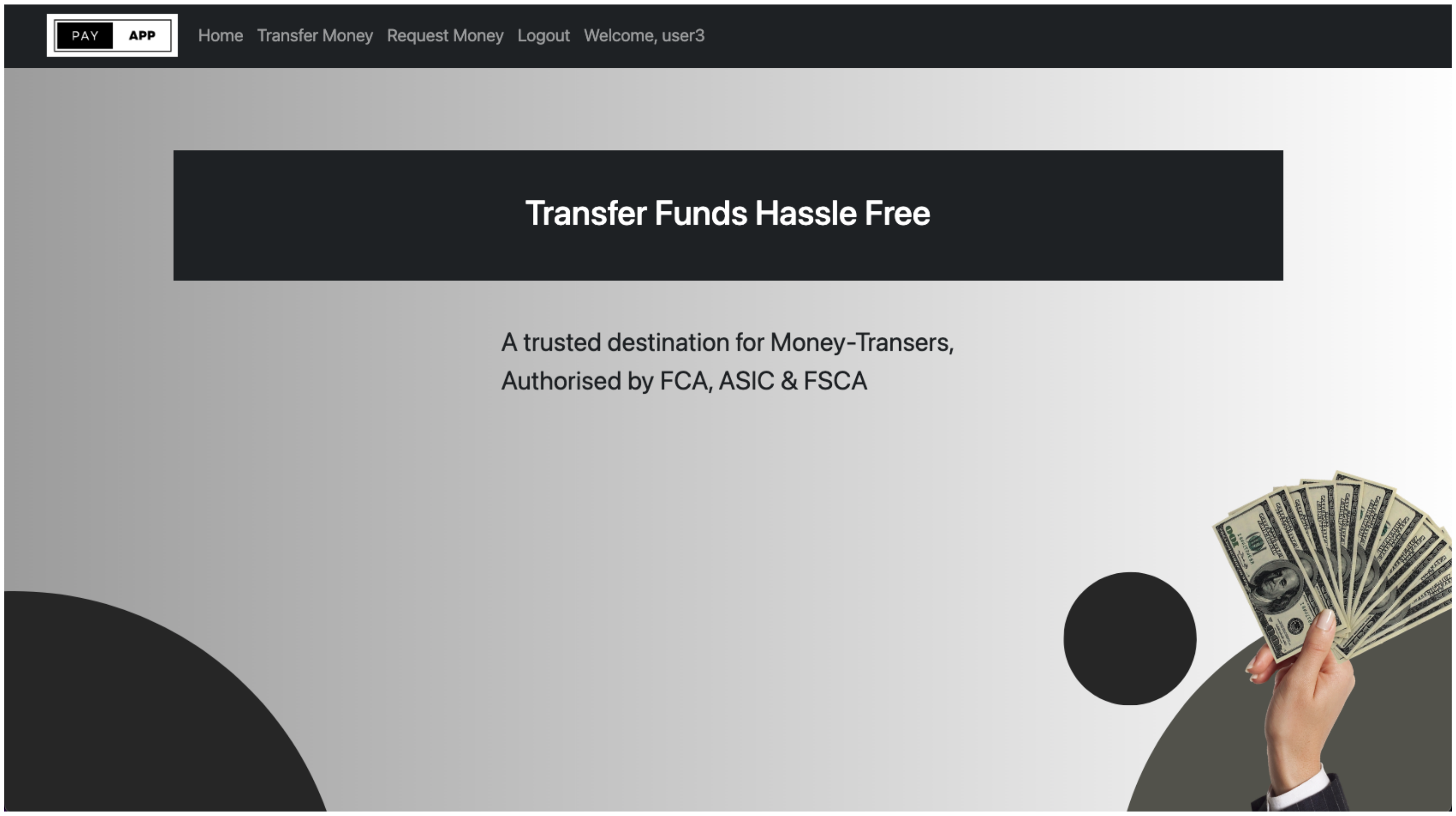
Username*

Password*

Don't have an account? Create an account.

Figure 2

When the user is registered and logged in they are directed to the landing page as seen in figure 3

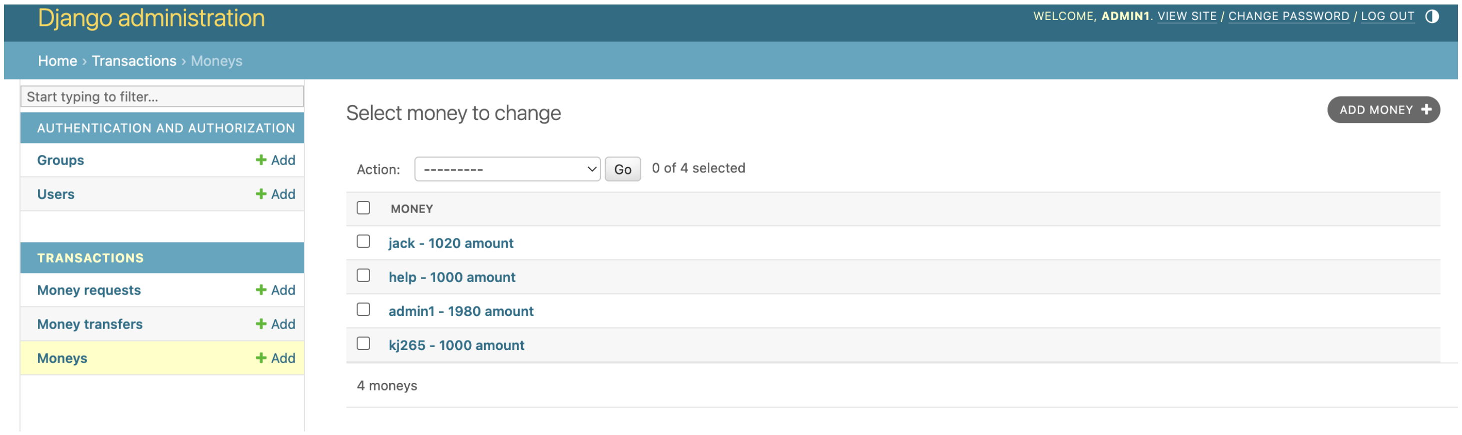


Transfer Funds Hassle Free

A trusted destination for Money-Transers,
Authorised by FCA, ASIC & FSCA

Figure 3

From figure 4 we can see each user's balance form the admin page


Django administration

WELCOME, ADMIN1. VIEW SITE / CHANGE PASSWORD / LOG OUT

Home › Transactions › Moneys

Select money to change

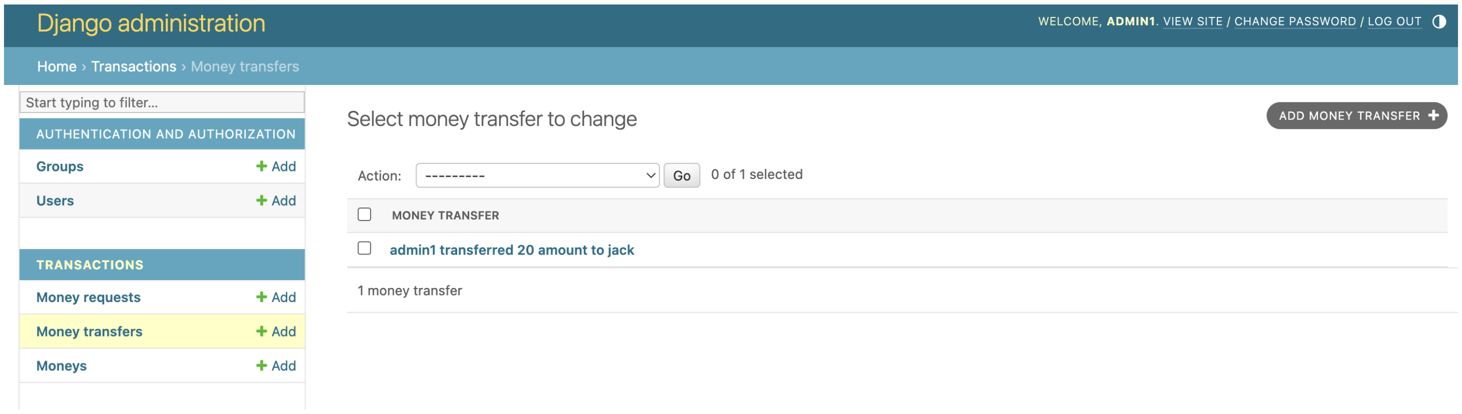
Action: --------- Go 0 of 4 selected

MONEY
jack - 1020 amount
help - 1000 amount
admin1 - 1980 amount
kj265 - 1000 amount

4 moneys

Figure 4

Admin can also see each user's transfer to another user


Django administration

WELCOME, ADMIN1. VIEW SITE / CHANGE PASSWORD / LOG OUT

Home › Transactions › Money transfers

Select money transfer to change

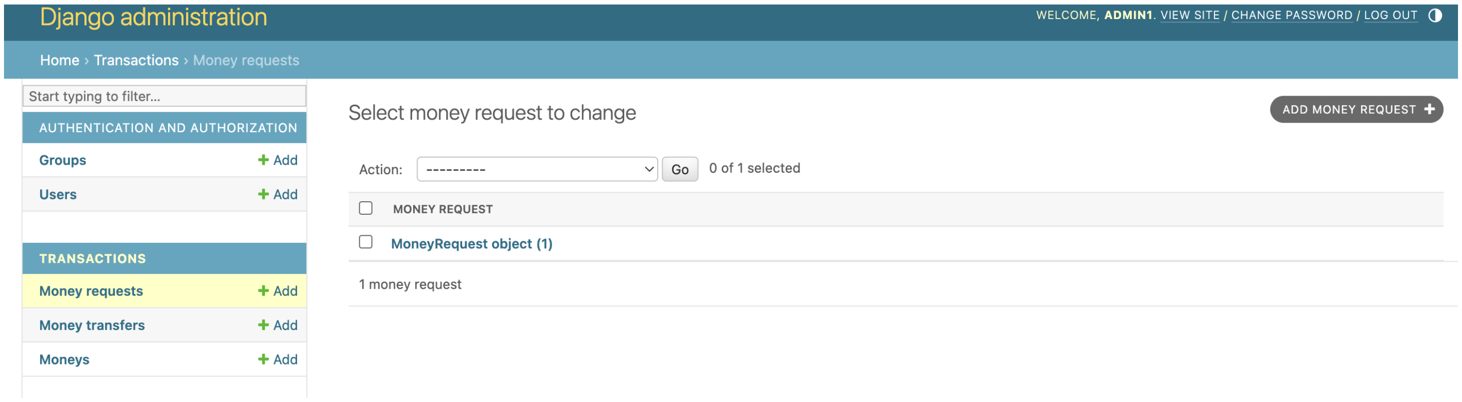
Action: --------- Go 0 of 1 selected

MONEY TRANSFER
admin1 transferred 20 amount to jack

1 money transfer

Figure 5

Admin can also see each user's request to another user


Django administration

WELCOME, ADMIN1. VIEW SITE / CHANGE PASSWORD / LOG OUT

Home › Transactions › Money requests

Select money request to change

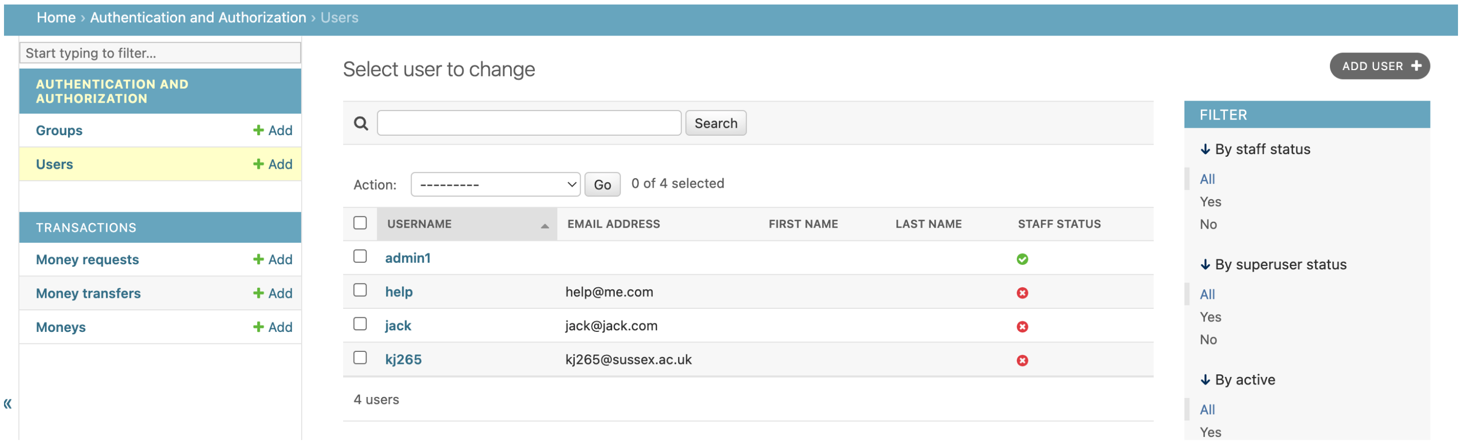
Action: --------- Go 0 of 1 selected

MONEY REQUEST
MoneyRequest object (1)

1 money request

Figure 6

Admin can also see all the users in the system


Home › Authentication and Authorization › Users

Start typing to filter...

AUTHENTICATION AND AUTHORIZATION

Groups	+ Add
Users	+ Add

TRANSACTIONS

Money requests	+ Add
Money transfers	+ Add
Moneys	+ Add

Select user to change

ADD USER +

Search

Action: --------- Go 0 of 4 selected

USERNAME	EMAIL ADDRESS	FIRST NAME	LAST NAME	STAFF STATUS
admin1				✓
help	help@me.com			✗
jack	jack@jack.com			✗
kj265	kj265@sussex.ac.uk			✗

4 users

FILTER

↓ By staff status
All
Yes
No

↓ By superuser status
All
Yes
No

↓ By active
All
Yes

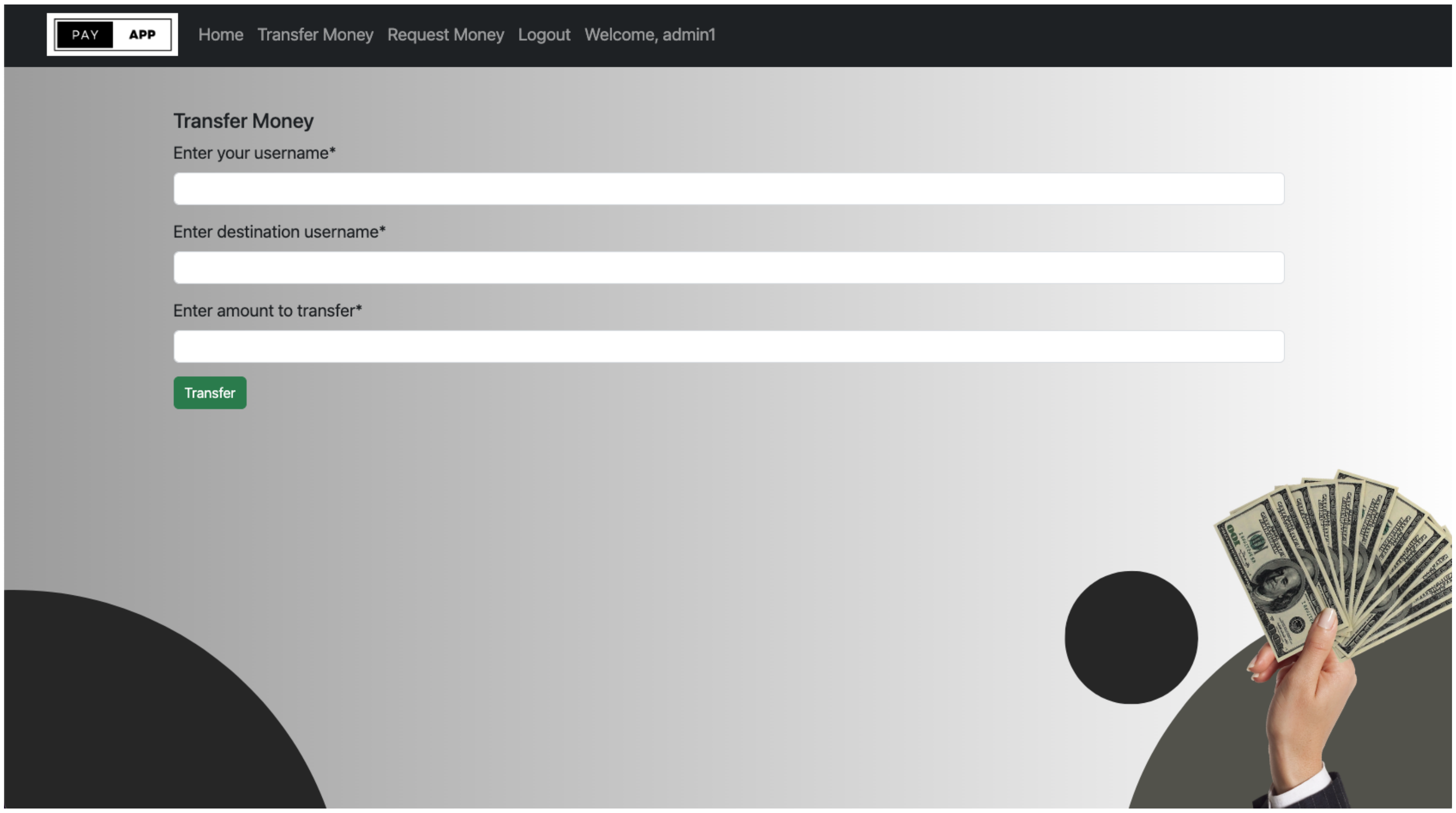
Figure 7

2 - Business Logic Layer

The views layer is a critical component of this project, as it contains the code responsible for accessing the models. The views layer employs the use of the Django framework's built-in `transaction.atomic()` function to guarantee transaction atomicity. By wrapping a block of code with this function, the views layer ensures that all database operations inside the block either succeed or fail entirely as a single unit of work. This approach is vital to maintain data consistency and integrity, especially in the face of concurrent database access.

Furthermore, the views layer guarantees the ACID properties of the database. ACID properties ensure that transactions are reliable and consistent, as they provide guarantees that the database will remain in a consistent state before and after each transaction. The views layer adheres to the four ACID properties, which are: the entire transaction takes place at once, the database is consistent before and after the transaction, multiple transactions occur independently, and the changes of the successful transaction occur even when a system failure occurs.

From the figure 8 we can see the page where users can transfer money to other registered users



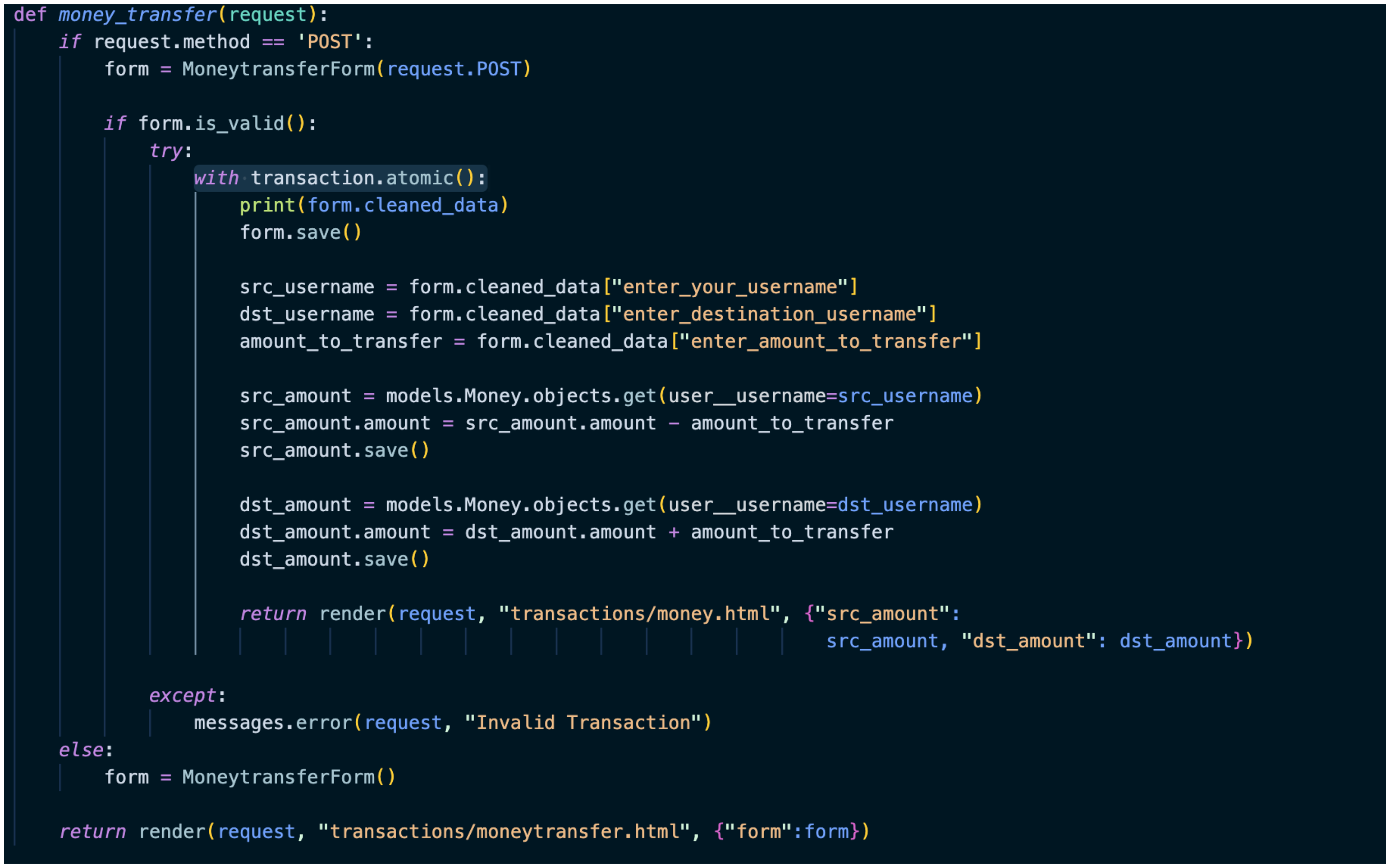
Transfer Money

Enter your username*

Enter destination username*

Enter amount to transfer*

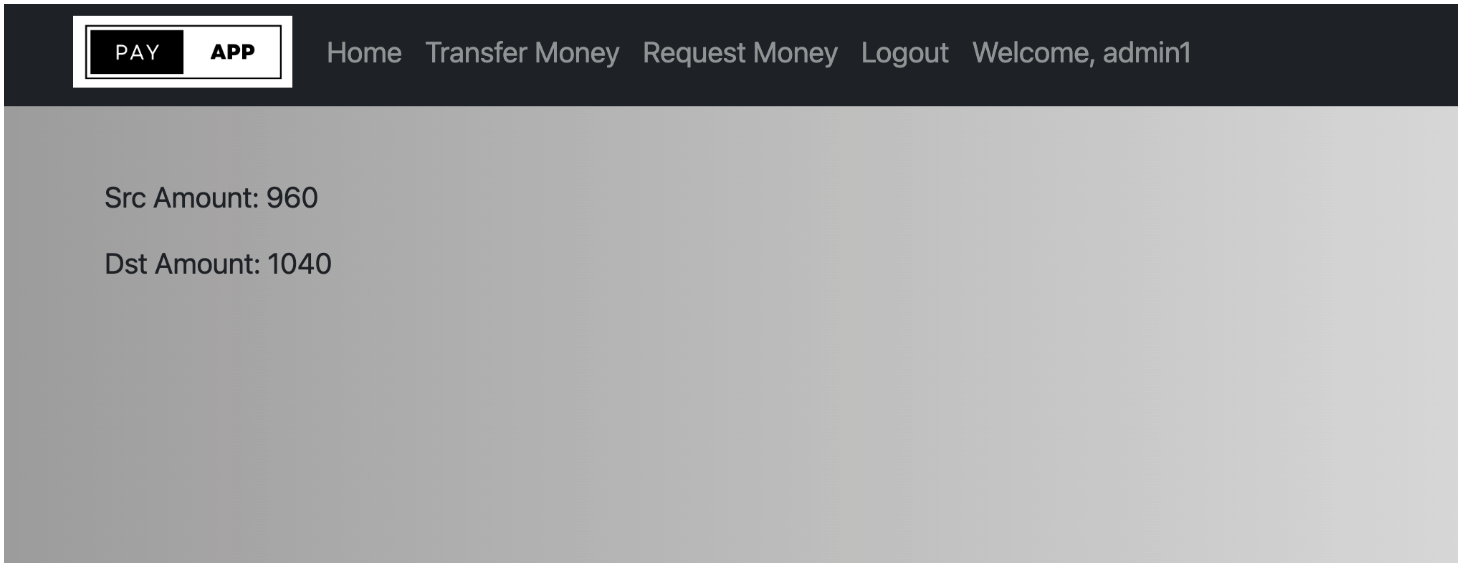
Figure 8



```
def money_transfer(request):
    if request.method == 'POST':
        form = MoneytransferForm(request.POST)

        if form.is_valid():
            try:
                with transaction.atomic():
                    print(form.cleaned_data)
                    form.save()

                    src_username = form.cleaned_data["enter_your_username"]
                    dst_username = form.cleaned_data["enter_destination_username"]
                    amount_to_transfer = form.cleaned_data["enter_amount_to_transfer"]

                    src_amount = models.Money.objects.get(user__username=src_username)
                    src_amount.amount = src_amount.amount - amount_to_transfer
                    src_amount.save()

                    dst_amount = models.Money.objects.get(user__username=dst_username)
                    dst_amount.amount = dst_amount.amount + amount_to_transfer
                    dst_amount.save()

                    return render(request, "transactions/money.html", {"src_amount":
                                                                          src_amount, "dst_amount": dst_amount})

            except:
                messages.error(request, "Invalid Transaction")
    else:
        form = MoneytransferForm()

    return render(request, "transactions/moneytransfer.html", {"form":form})
```

Figure 9

After the transaction has taken place, we can see each balance for the sender and the receiever in figure 10

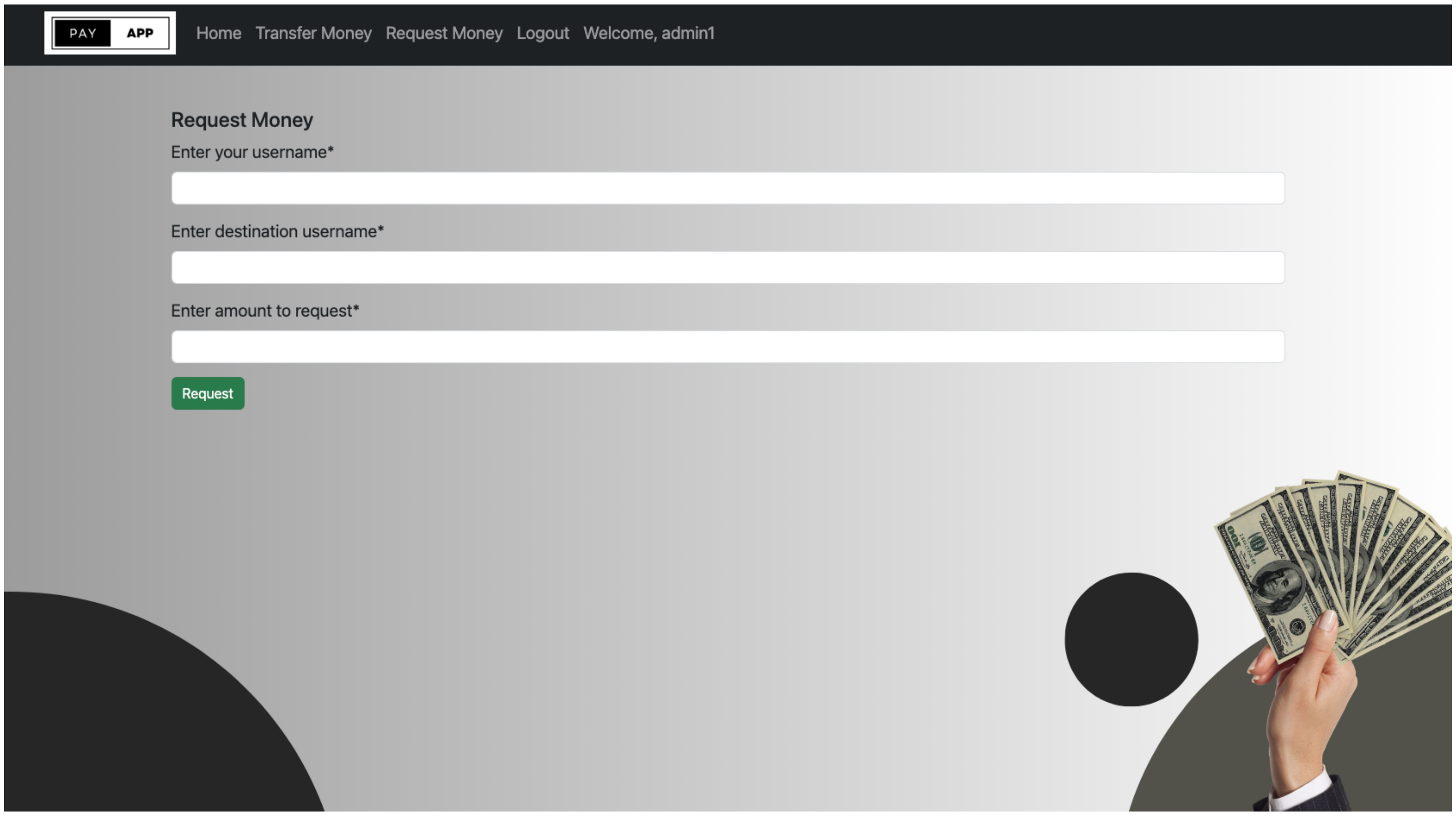


Src Amount: 960

Dst Amount: 1040

Figure 10

Likewise the users can also make a request to another registered user for funds and get a confirmation message, as seen in figure 11



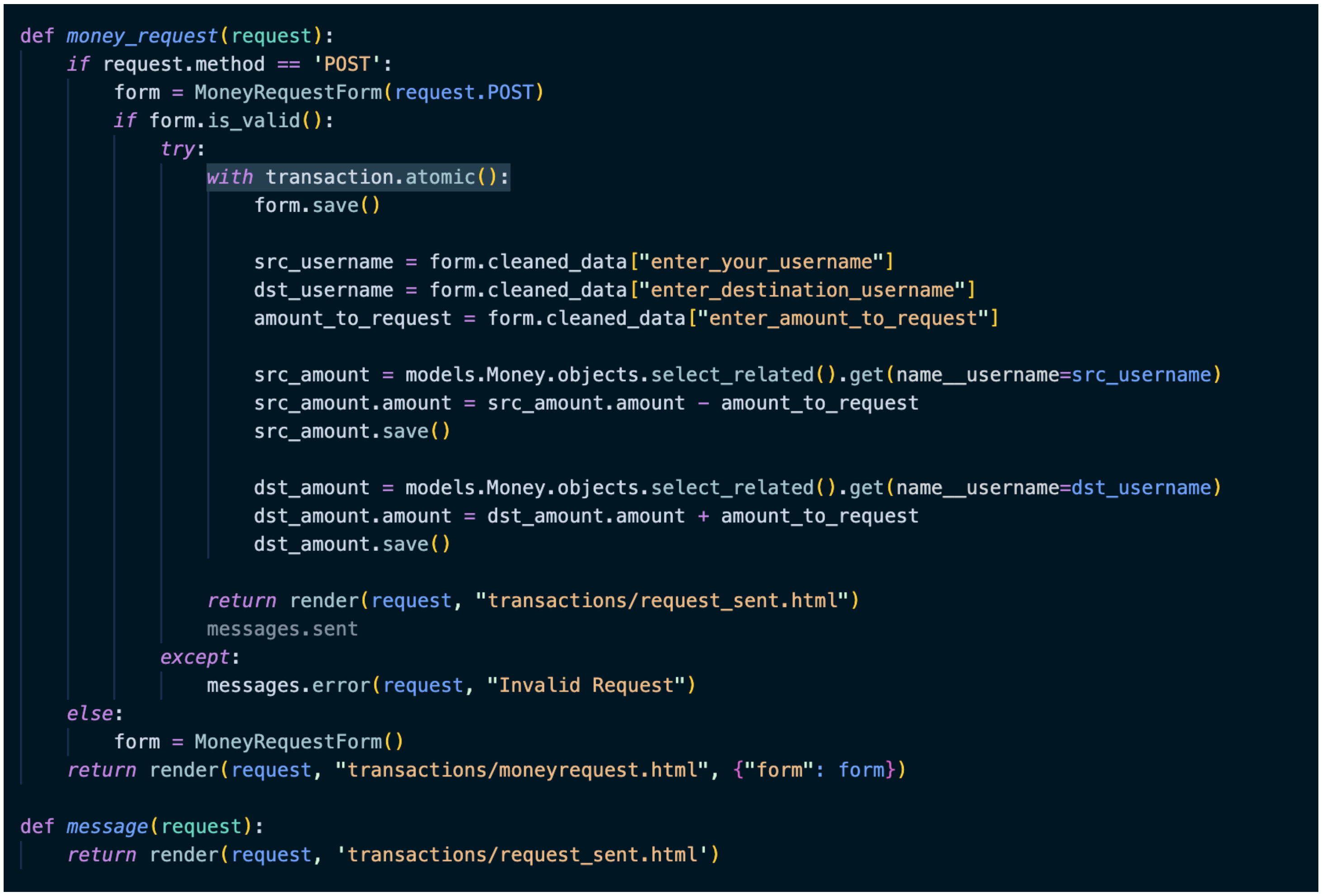
Request Money

Enter your username*

Enter destination username*

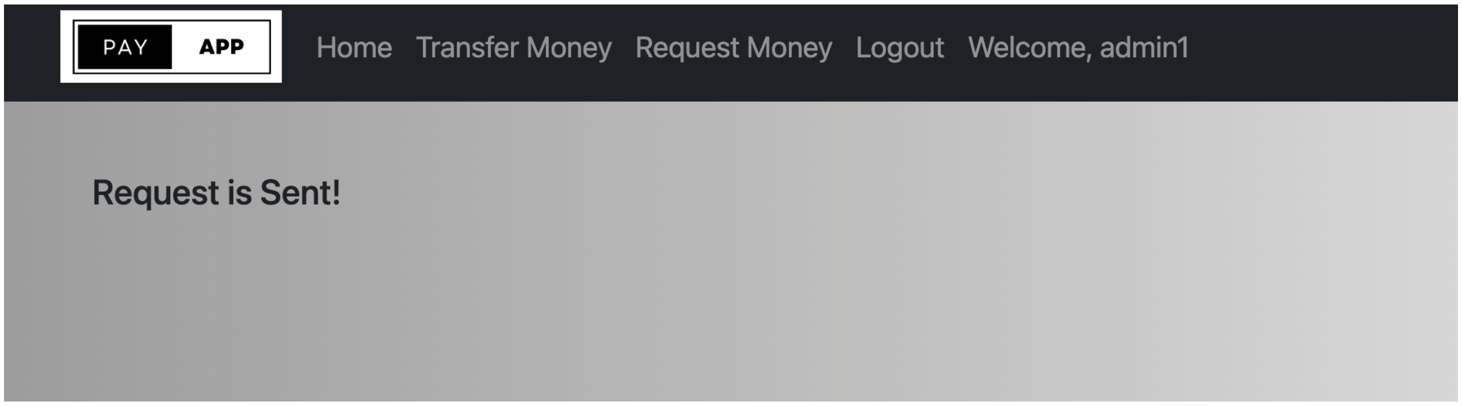
Enter amount to request*

Figure 11



```python
def money_request(request):
    if request.method == 'POST':
        form = MoneyRequestForm(request.POST)
        if form.is_valid():
            try:
                with transaction.atomic():
                    form.save()

                    src_username = form.cleaned_data["enter_your_username"]
                    dst_username = form.cleaned_data["enter_destination_username"]
                    amount_to_request = form.cleaned_data["enter_amount_to_request"]

                    src_amount = models.Money.objects.select_related().get(name__username=src_username)
                    src_amount.amount = src_amount.amount - amount_to_request
                    src_amount.save()

                    dst_amount = models.Money.objects.select_related().get(name__username=dst_username)
                    dst_amount.amount = dst_amount.amount + amount_to_request
                    dst_amount.save()

                return render(request, "transactions/request_sent.html")
                messages.sent
            except:
                messages.error(request, "Invalid Request")
    else:
        form = MoneyRequestForm()
    return render(request, "transactions/moneyrequest.html", {"form": form})

def message(request):
    return render(request, 'transactions/request_sent.html')
```

Figure 12



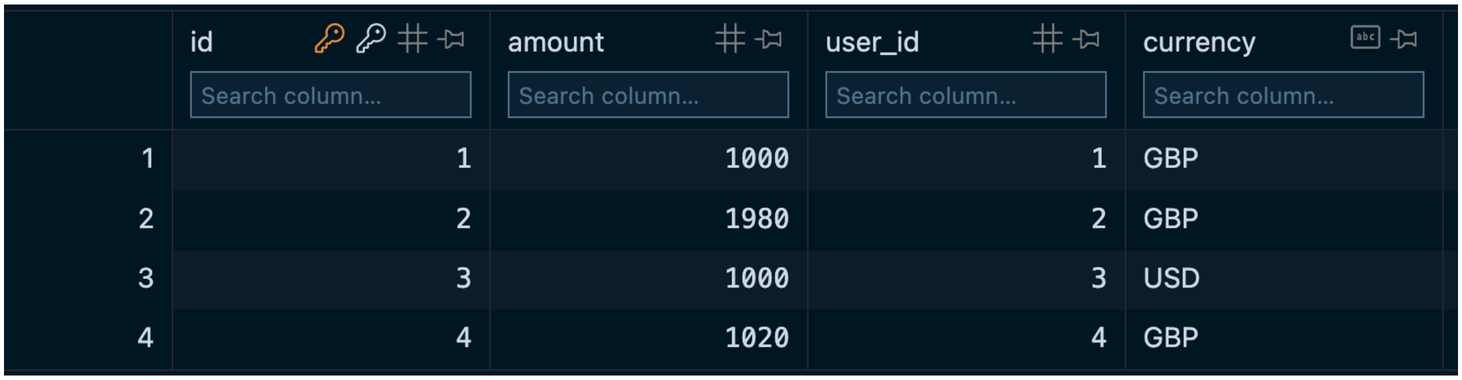
Request is Sent!

Figure 13

3 - Data Access Layer

The data storage of the application is handled by the Models layer, which is responsible for managing the data and interactions with the database. In this layer, all the data models that the application uses are defined. For this project, SQLite was selected as the Relational Database Management System (RDBMS) to store the application data. SQLite is a lightweight, file-based RDBMS that offers a good balance between ease of use and functionality. The database was named `db.webapps`. This layer ensures that the data is correctly stored and retrieved from the database, and that it is presented in a format that is easy for the other layers to use.

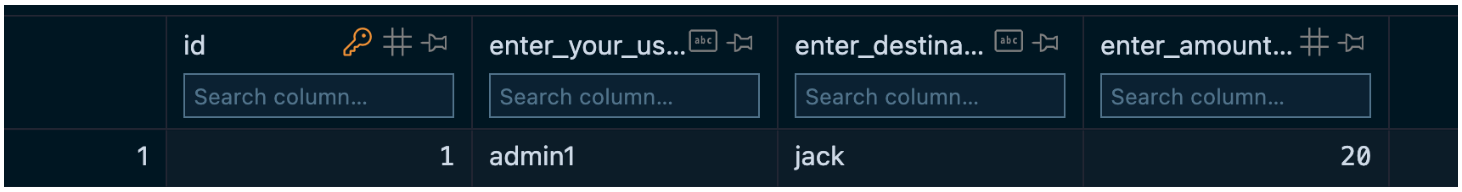
Figure 14 is the database table for the user balances:



	id	amount	user_id	currency
	Search column...	Search column...	Search column...	Search column...
1	1	1000	1	GBP
2	2	1980	2	GBP
3	3	1000	3	USD
4	4	1020	4	GBP

Figure 14

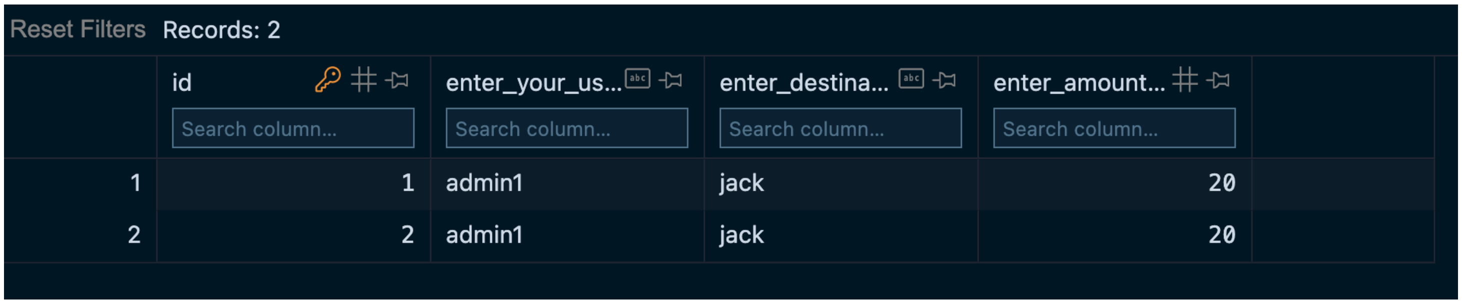
The database table for user transfers:



	id	enter_your_us...	enter_destina...	enter_amount...
	Search column...	Search column...	Search column...	Search column...
1	1	admin1	jack	20

Figure 15

The database table for user requests:



Reset Filters Records: 2

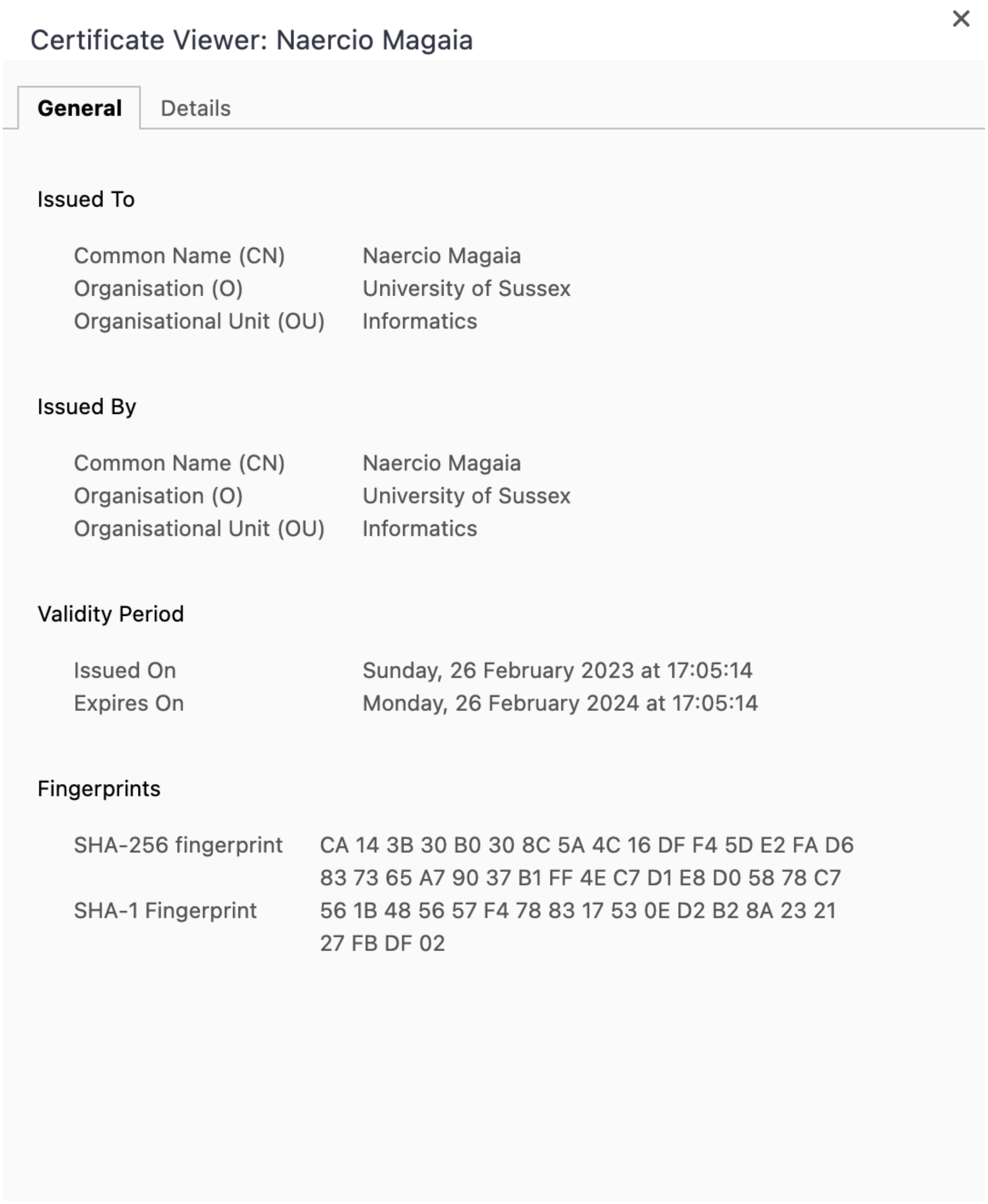
	id	enter_your_us...	enter_destina...	enter_amount...
	Search column...	Search column...	Search column...	Search column...
1	1	admin1	jack	20
2	2	admin1	jack	20

Figure 15

4 - Security Layer

User registration, login, and logout are essential functionalities in the system, and access is restricted to authorized users only, i.e., registered users and administrators.

To ensure secure data transmission between the client and server, the web application is accessed via HTTPS, which encrypts all data. Protection against cross-site request forgery (CSRF) is also implemented to prevent malicious users from executing unauthorized actions on behalf of a legitimate user. The certificate can be seen in figure 16:


Certificate Viewer: Naercio Magaia

General Details

Issued To

Common Name (CN)	Naercio Magaia
Organisation (O)	University of Sussex
Organisational Unit (OU)	Informatics

Issued By

Common Name (CN)	Naercio Magaia
Organisation (O)	University of Sussex
Organisational Unit (OU)	Informatics

Validity Period

Issued On	Sunday, 26 February 2023 at 17:05:14
Expires On	Monday, 26 February 2024 at 17:05:14

Fingerprints

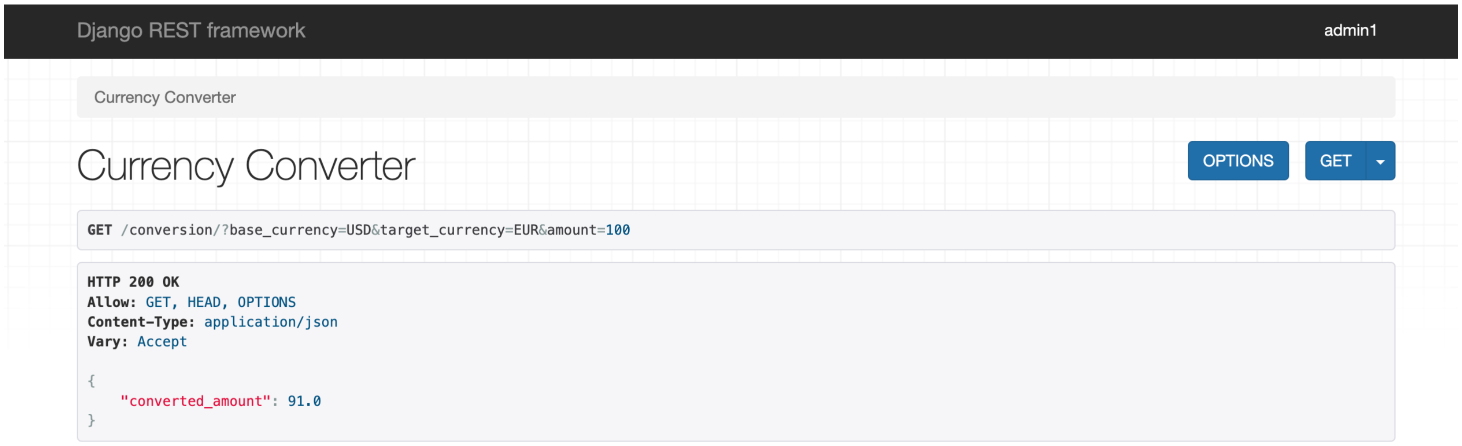
SHA-256 fingerprint	CA 14 3B 30 B0 30 8C 5A 4C 16 DF F4 5D E2 FA D6 83 73 65 A7 90 37 B1 FF 4E C7 D1 E8 D0 58 78 C7
SHA-1 Fingerprint	56 1B 48 56 57 F4 78 83 17 53 0E D2 B2 8A 23 21 27 FB DF 02

Figure 16

The system has a registered admin account with the credentials *username: admin1, password:admin1.* This account is used by the administrator to perform tasks such as managing user accounts, monitoring transactions, and registering new administrators.

5 - Web Services

The project implemented a REST service that can be accessed by the business logic layer. The RESTful web service is designed to respond only to GET requests, with the resource having the name `conversion` in the path.



```
GET /conversion/?base_currency=USD&target_currency=EUR&amount=100
```

```
HTTP 200 OK
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "converted_amount": 91.0
}
```

Figure 17

The RESTful service returns `USD` to `EUR` conversion rate as an HTTP response with status code `200`. The URL can be accessed via: `https://127.0.0.1:8000/conversion/?base_currency=USD&target_currency=EUR&amount=100`.