

Online Payment Service Report

1. Presentation Layer

The presentation layer was implemented using **Django templates**, enabling efficient interaction for both users and administrators.

Implemented Features:

- **User Authentication:**
 - Register, login, and logout functionality handled by a dedicated register app.
 - Templates located at templates/register and templates/login.
- **Transaction Overview:**
 - Users can view their sent and received transaction history via their profile.
- **Send & Request Payments:**
 - Users can send payments to other registered users.
 - Users can request payments, which appear as notifications. Currency conversions are handled automatically.
- **Admin Panel:**
 - Admins can view all user accounts and transaction records.
 - Admins can promote users to admin and view data in different currencies.
- **Navigation:**
 - Seamless page transitions implemented.
- **Enhanced UI/UX:**
 - Styled using Bootstrap and Crispy Forms for responsive and elegant design.

2. Business Logic Layer

Business logic is handled in Django views (primarily in payment/views.py) to maintain separation of concerns and ensure transaction atomicity. Logic is distributed based on user role (standard user vs. admin) and action (send/request/payment). When making migrations we upload some currency conversions as well as admin user to the

Implemented Features:

- **User-Level Views:**
 - `pay_to_user`: Sends payments and displays transaction history.
 - `request_payment`: Allows users to request payments from others.
- **Admin Views:**
 - `promote_to_admin`: Promotes a user to admin.
 - `check_transactions`: Allows filtering and reviewing of all user transactions.
- **Authentication & Routing:**
 - `register`, `login_view`, and `logout_view` handle user session flow.
- **Transaction Management:**
 - Payment operations are wrapped in Django's `transaction.atomic` block to ensure ACID compliance.
- **Currency Conversion Logic:**
 - Automatically triggered during transactions via models and views.

Form Logic Integration:

- Forms like `PaymentForm`, `RequestPaymentForm`, and `CheckTransaction` are closely tied to these views to handle input and validation.
 - Admin actions and transaction requests are tied with real-time model updates to maintain business logic consistency.
-

3. Data Access Layer

Utilizes **SQLite** and Django ORM to ensure data integrity and scalability. The logic is divided into structured models that encapsulate both user data and transaction data.

Models Used:

- **Profile:**
 - Extends the default Django user.
 - Fields: `balance`, `admin flag`, `currency`
- **Pay:**

- Logs a payment transaction.
 - Fields: payer, payee, amount, timestamp, success
- RequestPayment:
 - Represents a payment request between users.
 - Fields: requester, payer, amount, timestamp, success
- Currency:
 - Stores static exchange rates for conversions.
 - Fields: code, name, value

Forms for Data Handling:

- RegisterForm: Collects user data during sign-up.
- PaymentForm: Initiates a payment.
- RequestPaymentForm: Initiates a request for payment.
- CheckTransaction: Admin form to filter transactions.

Data Flow:

- Django ORM handles all read/write operations.
- ForeignKey relationships maintain referential integrity between users, profiles, and transactions.

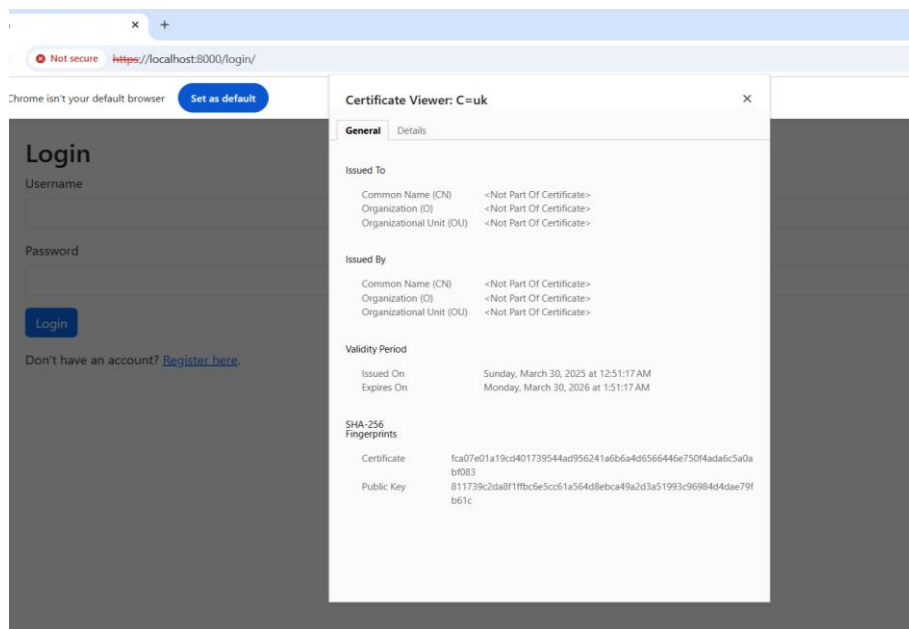
4. Security Layer

Robust security practices are applied across the application. Secure connection setup using generated public/private keys and certificates.

Implemented Features:

- **Authentication:** Register, login, logout.
- **Access Control:** Admin access controlled via profile flags.
- **Secure Server Login and Key Generation:**
 - Log into Unix server:
`ssh kb655@unix.sussex.ac.uk`
 - Generate private key:
`openssl genrsa -out localhost.key 2048`

- Create Certificate Signing Request (CSR)
- Generate Certificate:
`openssl x509 -req -days 365 -in localhost.csr -signkey localhost.key -out localhost.crt`
- Use FileZilla or SCP to securely download the generated .key and .crt files to your local machine.
- **HTTPS Support:**
 - Enabled with `localhost.crt` and `localhost.key`.
 - Run the server securely: `python manage.py runserver_plus --cert-file localhost.crt --key-file localhost.key`



- **Security Protections:**
 - XSS: Certificate validation, ALLOWED_HOSTS configuration.
 - CSRF: {% csrf_token %} included in all templates.
 - SQL Injection: Prevented via Django ORM.
 - Clickjacking: Prevented using security headers.
- **Default Admin Account: admin1/admin1 initialized by default.**

5. Web Services (REST API)

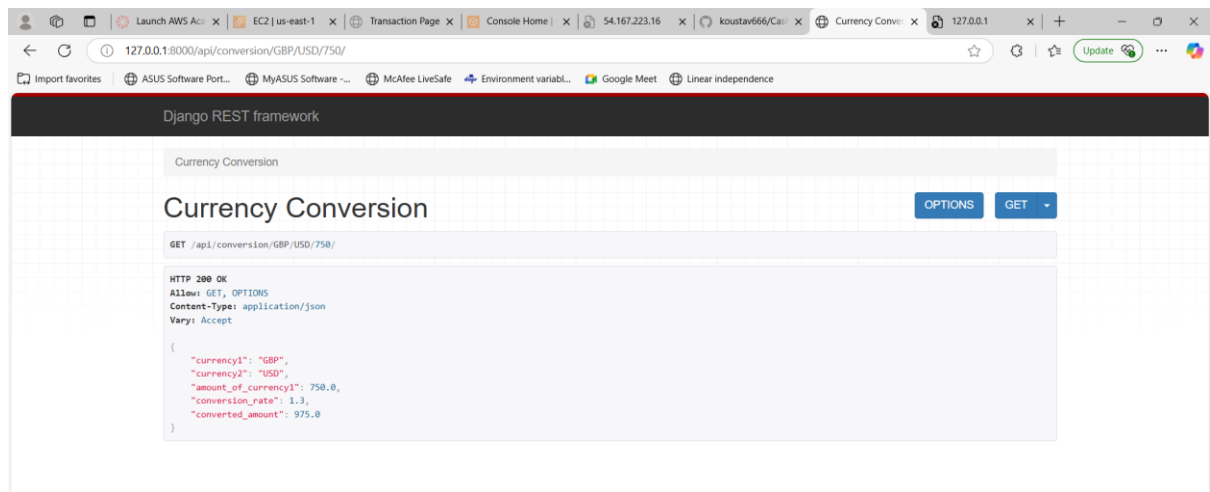
A RESTful **Currency Conversion API** is implemented in a **separate app** called `currency_conversion`.

Endpoint:

/conversion/{currency1}/{currency2}/{amount}

Features:

- Supports conversion between **GBP**, **USD**, and **EUR**.
- Exchange rates are statically loaded to the database during migration.



6. RPC with Apache Thrift

RPC service built using **Apache Thrift** to timestamp transactions.

Features:

- The interface is defined in a timestamp_service.thrift file.
- Thrift server is started using:

```
thrift --gen py timestamp_service.thrift
```



```
python timestampserver.py
```
- The server listens on **port 9090** and client timestampclient.py provides the method get_remote_timestamp().
- The client is integrated into the Django app and called during each payment or request transaction to provide accurate, real-time timestamps.
- This enhances traceability and trust for all financial operations.

7. Cloud Deployment (AWS EC2)

Deployed using **AWS EC2 instance** (created in Lab 2).

Deployment Steps:

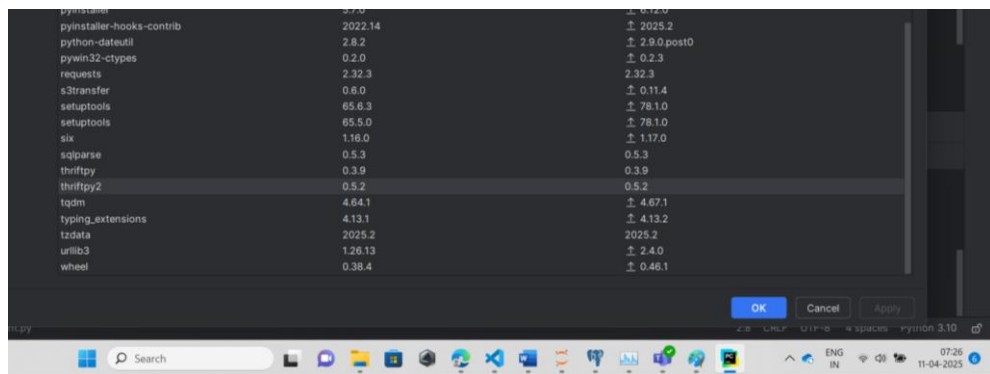
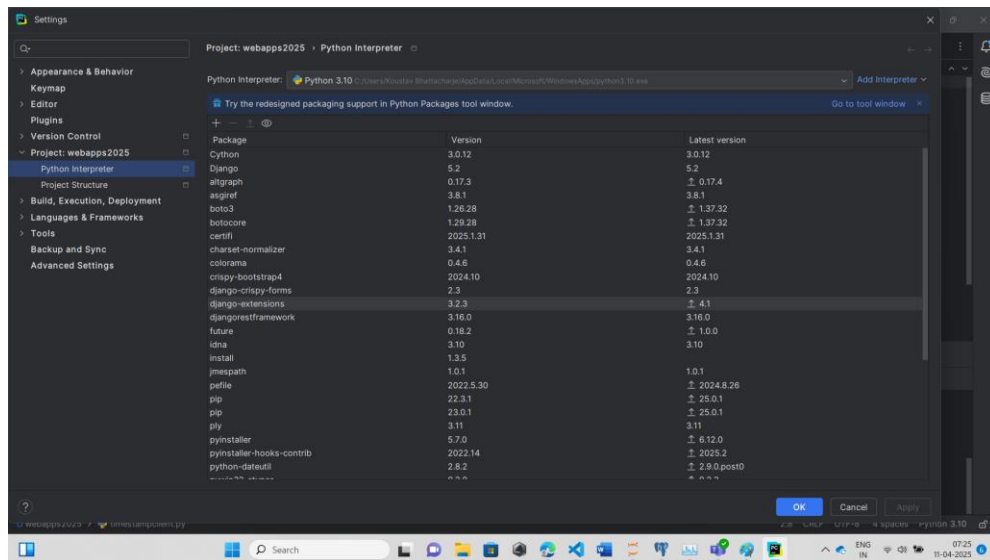
1. Clone GitHub repository:
`git clone https://github.com/koustav666/Cash-app.git`
 2. Navigate:
`cd Cash-app`
 3. Create & activate virtual environment:
`python3 -m venv env`
`source env/bin/activate`
 4. Install required packages.
 5. Initialize database:
`python3 manage.py makemigrations`
`python3 manage.py migrate`
 6. Run server:
`python3 manage.py runserver 0.0.0.0:8000`
 7. Ensure port 8000 is open in EC2 security group and inbound rule is defined.
 8. Add the EC2 Public IP 98.81.241.55 to ALLOWED_HOSTS in settings.py.
 9. Access the application from any browser at:
`http://98.81.241.55:8000/{endpoints}`
-

8. Used Packages(Install before running launching the app)

- djangorestframework
- thriftpy or thriftpy2(used here because deployment server could not install thriftpy)
- pyOpenSSL
- crispy-bootstrap4
- crispy-forms
- requests

- django-extensions (for runserver_plus)

Screenshot of packages present



9. Forms, Models, and Views Breakdown

◆ Register App

Views:

- register: Registers user and creates profile.
- login_view: Authenticates and redirects.
- logout_view: Logs user out.

Forms:

- RegisterForm (extends UserCreationForm):
 - Fields: username, email, password1, password2

◆ Payment App

Views:

1. `pay_to_user`: Send payments and display transactions.
2. `promote_to_admin`: Admin feature.
3. `check_transactions`: Admin-only filtered view.
4. `request_payment`: Users request payments.

Forms:

1. `PaymentForm`: `ModelForm` for `Pay` (fields: `payee`, `amount`).
2. `CheckTransaction`: Regular form (fields: `payee`, `currency`).
3. `RequestPaymentForm`: `ModelForm` for `RequestPayment` (fields: `payer`, `amount`).

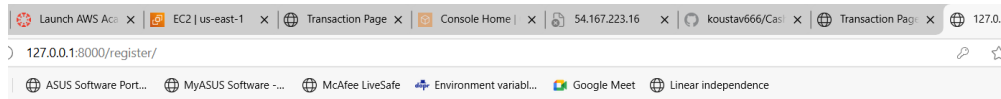
Models:

1. `Currency`: Stores exchange rates.
2. `Profile`: Extends user with `balance`, `admin flag`, `currency`.
3. `Pay`: Represents a payment.
4. `RequestPayment`: Stores payment requests.

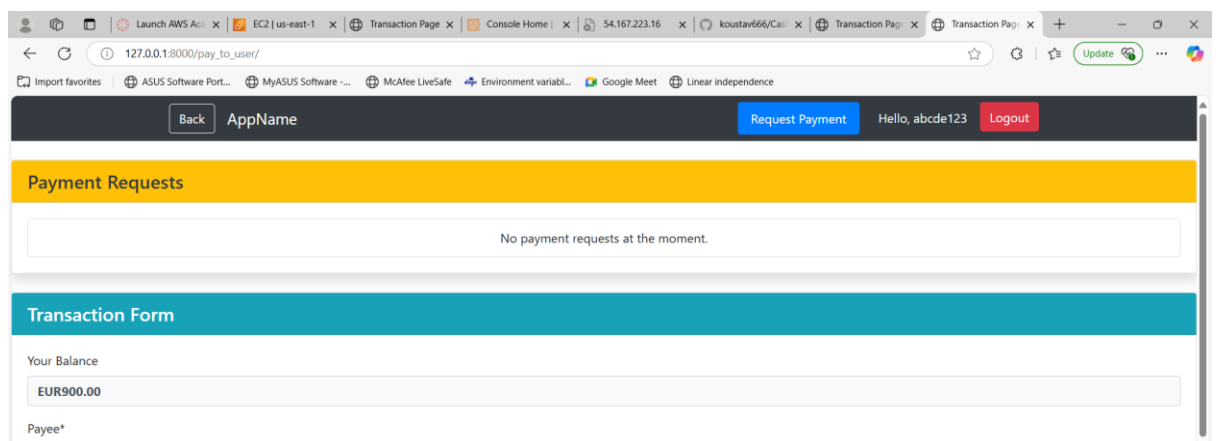
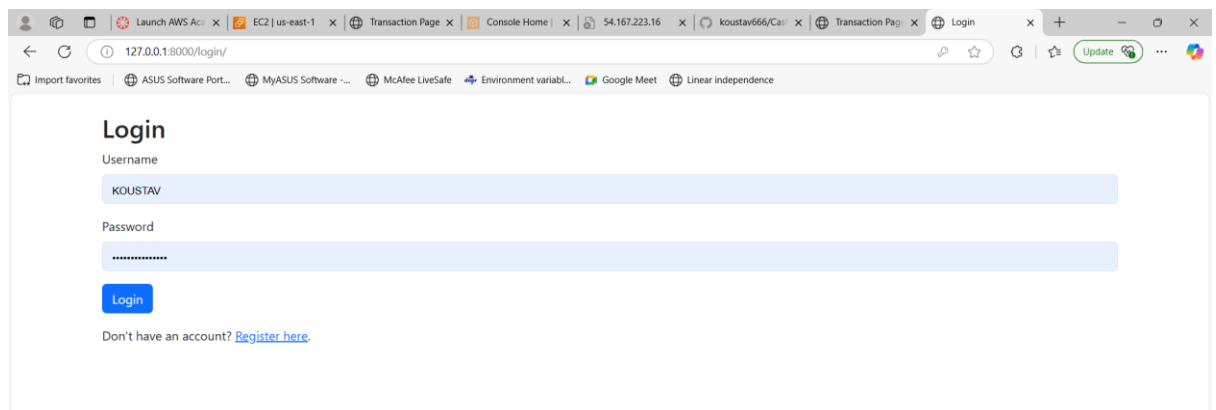
10. User Manual (Endpoints are mentioned in the screen shots)

For Users:

1. Register with details.



2. Log in to dashboard.



3. Request payments.

Back

AppName
Hello, admin1
Logout

Request Payment in GBP

Payer*
KOUSTAV

Amount*
24

Request Payment

Back

AppName
Hello, KOUSTAV
Logout

Request Payment in EUR

Payer*

Amount*

Request Payment

4. Respond to payment requests.

Back AppName Request Payment Admin Console Hello, KOUSTAV Logout

Payment Requests

admin1 requested EUR24.00 Accept Reject

Transaction Form

Your Balance
EUR952.00

Payee*

Amount*

5. Transaction History

The screenshot shows a web browser window with the URL `127.0.0.1:8000/pay_to_user/`. The page has a form at the top with fields for "Payee*" and "Amount*", a "Submit Transaction" button, and a "Reset" button. A red notification bar indicates a successful payment: "Successfully paid 24.00 to admin1." Below the form, there are two sections: "Sent Transactions" and "Received Transactions".

Sent Transactions

Sent to: admin1 Amount: EUR20.00 Time: None Success: True
Sent to: admin1 Amount: EUR24.00 Time: April 11, 2025, 6:44 a.m. Success: True

Received Transactions

Received from: admin1 Amount: EUR24.0 Success: True

6. Send payment

The screenshot shows a "Transaction Form" with a teal header. It includes a "Your Balance" field showing "EUR900.00", a "Payee*" field with "KOUSTAV" selected, and an "Amount*" field with "20" entered. There are "Submit Transaction" and "Reset" buttons at the bottom.

Launch AWS Ac... x EC2 | us-east-1 x Page not found x Console Home x 54.167.223.16 x TS Module units: V... x Currency Conve... x Transaction Page x + - x

127.0.0.1:8000/pay_to_user/

Import favorites ASUS Software Port... MyASUS Software ~... McAfee LiveSafe Environment variabl... Google Meet Linear independence

Back AppName Request Payment Hello, abcde123 Logout

Payment Requests

No payment requests at the moment.

Transaction Form

Your Balance

EUR880.00

Payee*

Amount*

• Successfully paid 20 to KOUSTAV.

Submit Transaction Reset

7. Appropriate conversions are made using the currency conversion REST API

Launch AWS Ac... x EC2 | us-east-1 x Transaction Page x Console Home x 54.167.223.16 x koustav666/Ca... x Transaction Page x Transaction Page x + - x

127.0.0.1:8000/pay_to_user/

Import favorites ASUS Software Port... MyASUS Software ~... McAfee LiveSafe Environment variabl... Google Meet Linear independence

Amount*

Submit Transaction Reset

Sent Transactions

Sent to: KOUSTAV Amount: GBP20.00 Time: April 9, 2025, 7:59 a.m. Success: True
Sent to: KOUSTAV Amount: GBP20.00 Time: April 10, 2025, 11:04 p.m. Success: True
Sent to: KOUSTAV Amount: GBP20.00 Time: April 10, 2025, 11:08 p.m. Success: True

Received Transactions

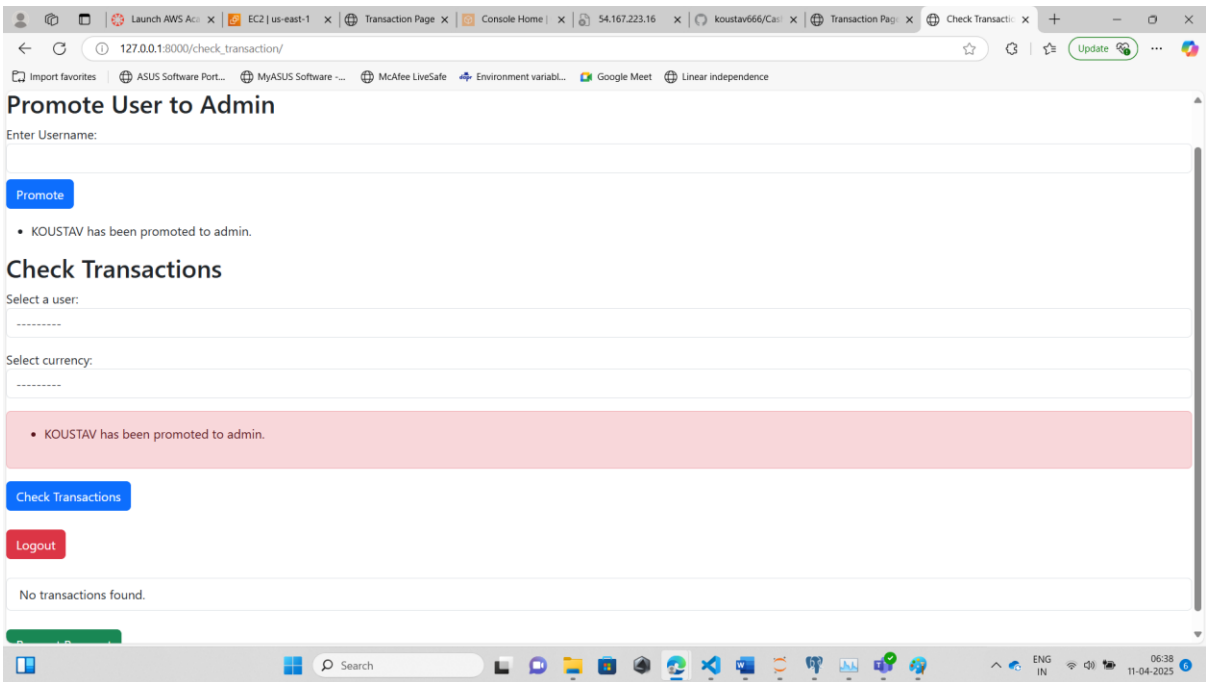
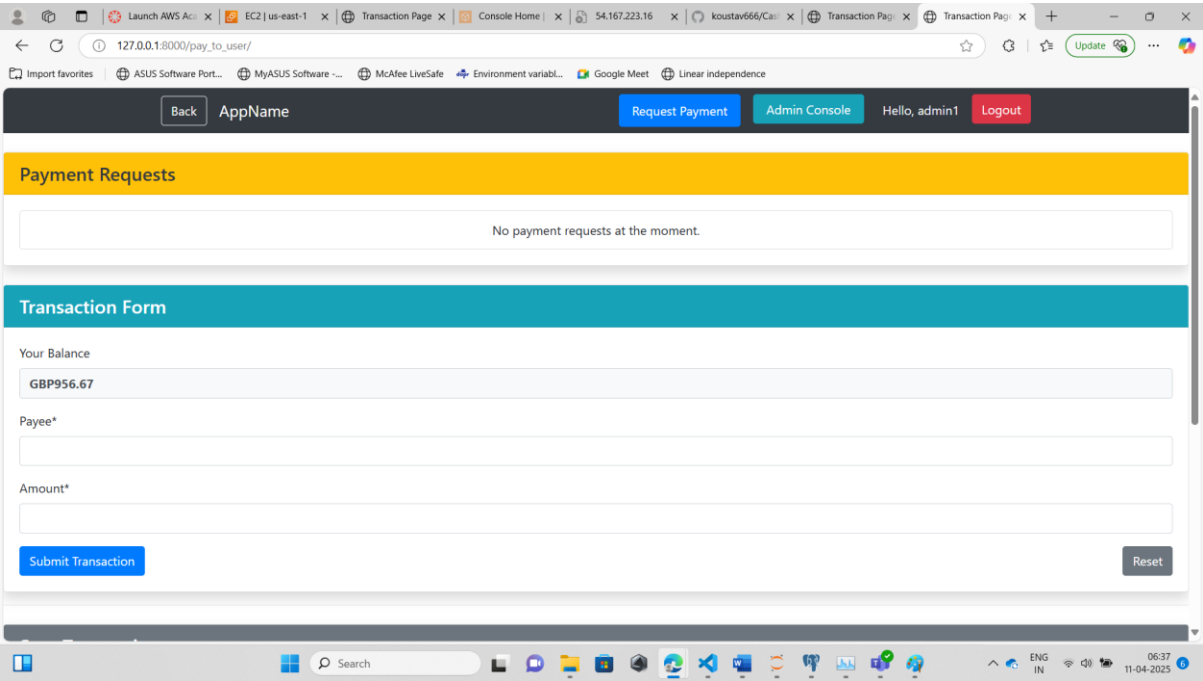
Received from: KOUSTAV Amount: GBP16.6667 Time: None Success: True
Received from: KOUSTAV Amount: GBP20.0 Time: April 11, 2025, 6:44 a.m. Success: True
Received from: KOUSTAV Amount: GBP16.6667 Time: April 11, 2025, 6:53 a.m. Success: True

Search

ENG IN 06:57 11-04-2025

For Admins:

- 1. Log in with admin1/admin1. It has an additional button Admin console



2. View user data and promote users.

The screenshot shows a web application interface with the following elements:

- Promote User to Admin:** A section with an "Enter Username:" input field, a "Promote" button, and a "Check Transactions" button.
- Select a user:** A dropdown menu showing "admin1".
- Select currency:** A dropdown menu showing "USD".
- Check Transactions:** A button to view transaction history.
- Logout:** A red button to log out.
- Balance:** A display showing "USD1243.67".
- Transaction History:** A table with the following data:

Sent from	Received by	Amount	Time	Success
admin1	KOUSTAV	USD26.00	April 10, 2025, 11:08 p.m.	True
admin1	KOUSTAV	USD26.00	April 10, 2025, 11:04 p.m.	True
admin1	KOUSTAV	USD26.00	April 9, 2025, 7:59 a.m.	True
KOUSTAV	admin1	USD21.67	None	True
- Request Payment:** A green button to request a payment.

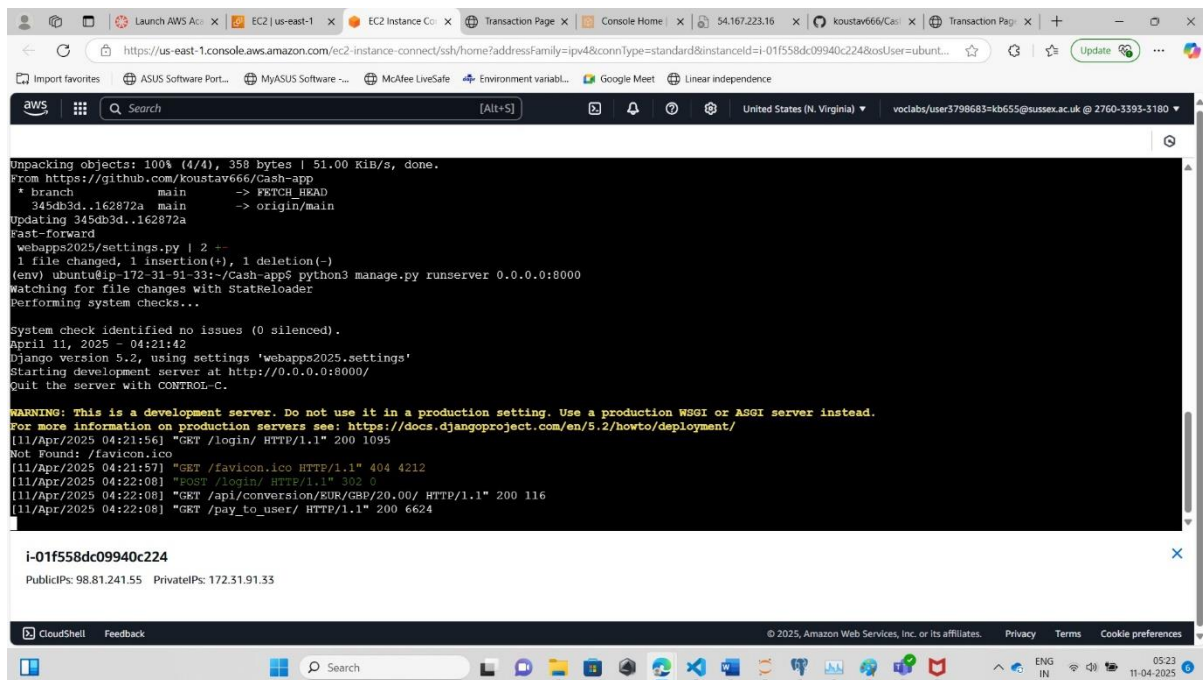
AWS screenshots:

1. Running on 98.81.241.55 IP

The screenshot shows a web application interface with the following elements:

- Navigation Bar:** Includes a "Back" button, "AppName", "Request Payment", "Admin Console", "Hello, admin1", and a "Logout" button.
- Payment Requests:** A section with a yellow header and a message: "No payment requests at the moment."
- Transaction Form:** A section with a teal header and the following fields:
 - Your Balance:** A display showing "GBP956.67".
 - Payee*:** An input field for the payee's name.

2. EC2 logs



```
Unpacking objects: 100% (4/4), 358 bytes | 51.00 KiB/s, done.
From https://github.com:koustav666/Cash-app
* branch      main      -> FETCH_HEAD
   345db3d..162872a  main    -> origin/main
Updating 345db3d..162872a
Fast-forward
 webapps2025/settings.py | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
(env) ubuntu@ip-172-31-91-33:~/Cash-app$ python3 manage.py runserver 0.0.0.0:8000
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
April 11, 2025 - 04:21:42
Django version 5.2, using settings 'webapps2025.settings'
Starting development server at http://0.0.0.0:8000/
Quit the server with CONTROL-C.

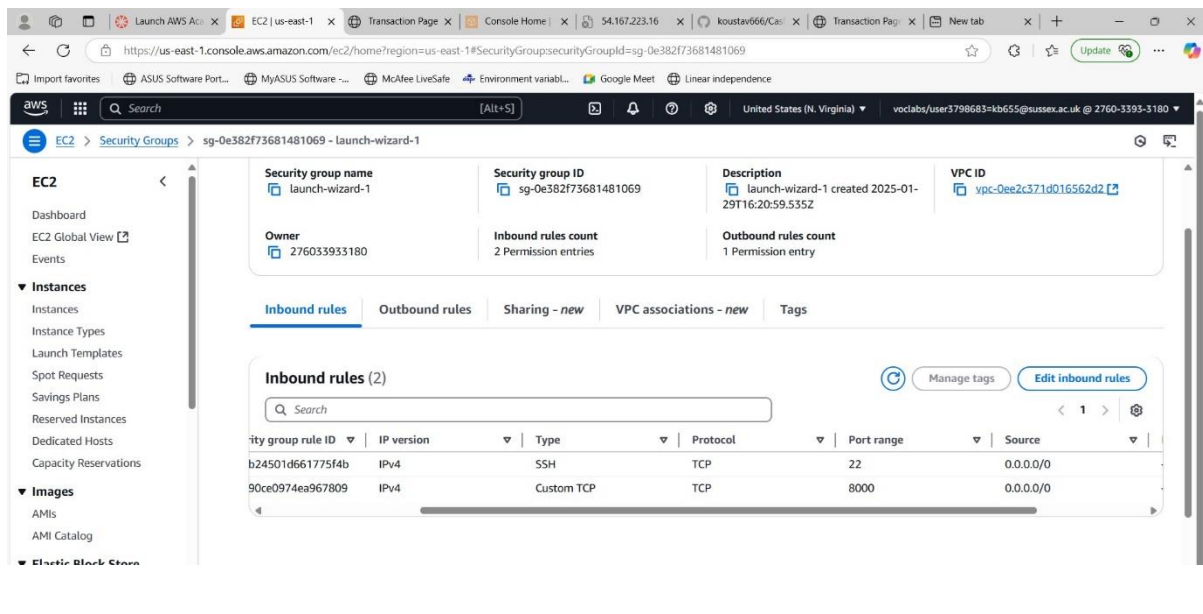
WARNING: This is a development server. Do not use it in a production setting. Use a production WSGI or ASGI server instead.
For more information on production servers see: https://docs.djangoproject.com/en/5.2/howto/deployment/

[11/Apr/2025 04:21:56] "GET /login/ HTTP/1.1" 200 1095
Not Found: /favicon.ico
[11/Apr/2025 04:21:57] "GET /favicon.ico HTTP/1.1" 404 4212
[11/Apr/2025 04:22:08] "POST /login/ HTTP/1.1" 302 0
[11/Apr/2025 04:22:08] "GET /api/conversion/EUR/GBP/20.00/ HTTP/1.1" 200 116
[11/Apr/2025 04:22:08] "GET /pay_to_user/ HTTP/1.1" 200 6624
```

i-01f558dc09940c224

Public IPs: 54.167.223.16 Private IPs: 172.31.91.33

3. Security group inbound rules



Security group name	Security group ID	Description	VPC ID
launch-wizard-1	sg-0e382f73681481069	launch-wizard-1 created 2025-01-29T16:20:59.535Z	vpc-0ee2c371d016562d2

Owner	Inbound rules count	Outbound rules count
276033933180	2 Permission entries	1 Permission entry

Security group rule ID	IP version	Type	Protocol	Port range	Source
b24501d661775f4b	IPv4	SSH	TCP	22	0.0.0.0/0
90ce0974ea967809	IPv4	Custom TCP	TCP	8000	0.0.0.0/0

11. Conclusion

This project implements a secure, modular, and user-friendly **online payment system** using Django.

Key Achievements:

- Secure and atomic transaction handling
- Admin and access control
- REST API (separate app for currency conversion)
- RPC (timestamping via Apache Thrift)
- Deployment on AWS EC2 (IP: 98.81.241.55)

The application demonstrates **layered architecture**, **modular design**, and **full-stack development** principles, making it scalable for real-world use.

Potential Fixes:

- The messages appearing are a bit buggy and needs to be improved.
- The Pay model needs to be more detailed as we need to call the conversion API lots of times to convert amounts into the user's currency. This increases the response time.
- The request payment layout could be improved
- Overall all functionalities are ok but better message labelling like '20EUR has been paid to user' needs to be done.
- Certificates don't have an **issued to** and **issued by** section as it was left blank.