Cryptographic Assignment

Student Name: Fateh Shahriar Sharif

Student ID: 202310667

Introduction

The University Catering Services aims to enhance student experience by implementing an innovative catering system that includes an online payment method. Recognizing PayPal's global acceptance and security (Niranjanamurthy, 2014), it has been chosen as the primary transaction mode. This report outlines the steps to integrate PayPal using Python and the Django web framework (Gore et al., 2021), using the PayPal API in a sandbox environment.

Setting up the PayPal Sandbox account

Setting up a PayPal Sandbox account is essential for testing PayPal transactions without real money. This allows mocking transactions as it should be in real-life scenarios. While developing a sandbox account, two accounts should be created to allow effective testing between the customer and the merchant. These accounts include a business account (merchant) and a Personal account (customer/student).

Here are the detailed steps to set up a PayPal sandbox account.

- Sandbox accounts are created on PayPal's developer central website https://developer.paypal.com/. Once a developer account has been created, and verified,

one can then create as many sandbox accounts as possible. Once created, and logged in, navigate to testing tools, then click on sandbox. Here, you are presented with an option to create a sandbox account for testing purposes. A screenshot is presented below.

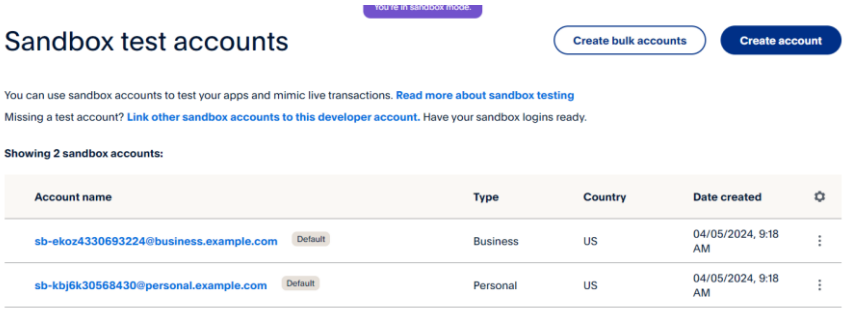


Figure 1.1 A screenshot of sandbox accounts (Business and Personal)

Once the account has been created, users have the option of generating fake credit cards which can be used for testing purposes. To create one, click on the test email address either for business or personal, scroll down to credit cards, and then click on manage. Once done, the user is given an option to create different cards. This however is optional, as users can decide to use their PayPal credits (see figure 1.10).

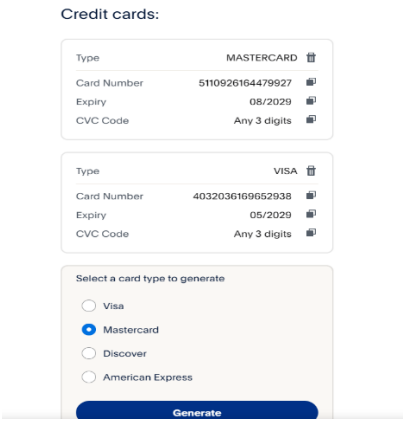


Figure 1.2 A screenshot showing the generated card and how to generate a card (Business and Personal)

- Once created, proceed to create an App that would contain credentials for PayPal API calls within the sandbox environment. Create an app, by clicking on Apps & Credential menu on the dashboard. Once clicked, you are presented with an option to create App, without any limits. Once the app has been created, click on the app to view vital information including the Client ID and Secret Key which would be used when interacting with PayPal Sandbox via API calls.

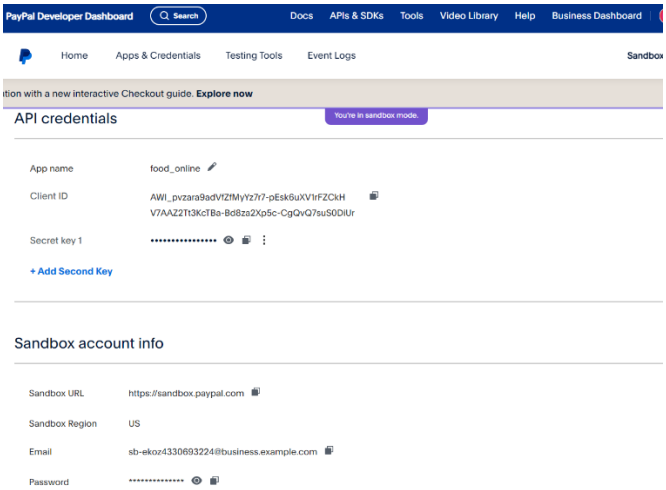


Figure 1.3 A screenshot showing the API credentials for app creation, including the sandbox information.

The credentials are stored in an .env file.

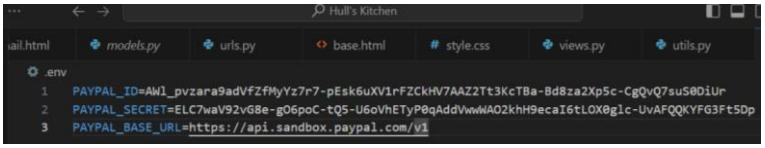


Figure 1.4 A screenshot showing the ClientID (seen as PAYPAL_ID), secret key (seen as PAYPAL_SECRET), and BASE_URL.

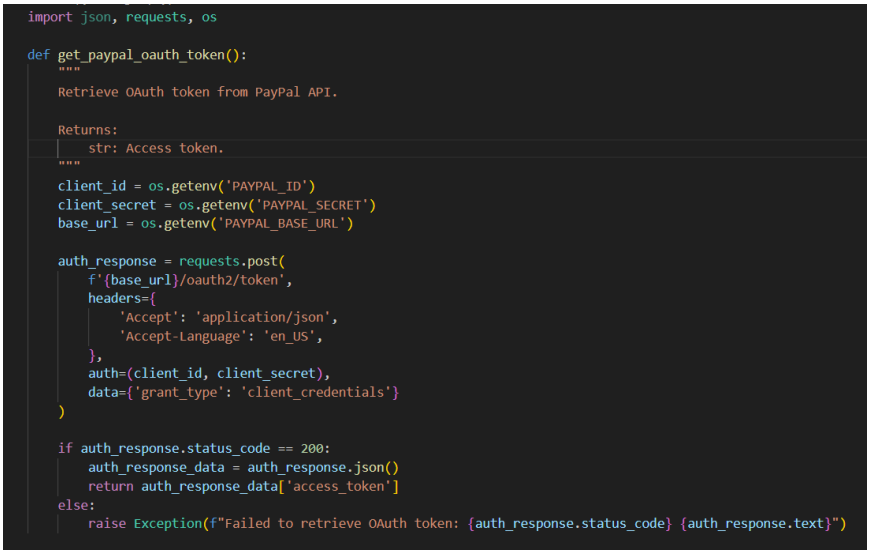
In the context of this application (the University Catering System), the merchant is the University Catering System, and the personal is the customer or student buying food or items from the University.

Credentials testing process used by PayPal

To facilitate the testing process with PayPal, the API credentials obtained after creating the app, most importantly the Client ID and secret key are used. These credentials are used for authentication, which generates a token used for subsequent API calls. For all API calls the base URL is https://api.sandbox.paypal.com/v1.

- For authentication, the URL is https://api.sandbox.paypal.com/v1/oauth2/token, This endpoint authenticates the app when calling subsequent PayPal Rest APIs. It uses the OAuth2 Protocol and accepts Client_ID and Client_SECRET as Json parameters, and grant-type as client_credentials. Once validated, it returns a JSON response that includes **scope, access_token, token_type, app_id, expires_in, and nonce,** and a status code of 200. The access_token is passed subsequently as a bearer token in

subsequent API calls. A sample code used in the application for access token generation is presented below.



```python
import json, requests, os

def get_paypal_oauth_token():
    """
    Retrieve OAuth token from PayPal API.

    Returns:
        str: Access token.
    """
    client_id = os.getenv('PAYPAL_ID')
    client_secret = os.getenv('PAYPAL_SECRET')
    base_url = os.getenv('PAYPAL_BASE_URL')

    auth_response = requests.post(
        f'{base_url}/oauth2/token',
        headers={
            'Accept': 'application/json',
            'Accept-Language': 'en_US',
        },
        auth=(client_id, client_secret),
        data={'grant_type': 'client_credentials'}
    )

    if auth_response.status_code == 200:
        auth_response_data = auth_response.json()
        return auth_response_data['access_token']
    else:
        raise Exception(f"Failed to retrieve OAuth token: {auth_response.status_code} {auth_response.text}")
```

Figure 1.5 A screenshot showing the sample code to generate an access token.

- Once you've tested that the API to generate the access token works effectively, proceed to making various API calls to simulate real transactions such as creating a payment and executing a payment as used in the context of this application. For this application, various edge cases have been attempted to check when expired tokens, fake tokens, and credentials are passed to ensure the API works as it is supposed. To create payment, the URL is https://api.paypal.com/v1/payments/payment, while to execute payment the URL is https://api.paypal.com/v1/payments/payment/{payment_id}/execute. Both URLs accept a POST request and require an Authorization header i.e. the Bearer access token, and JSON data. Create payment returns a status code of 201, including a dictionary of values containing **links, paymentid**, and other details used while creating the payment. Whereas, the execute payment URL returns a status code of 200 and a dictionary of values containing state and other metadata used when creating the payment.

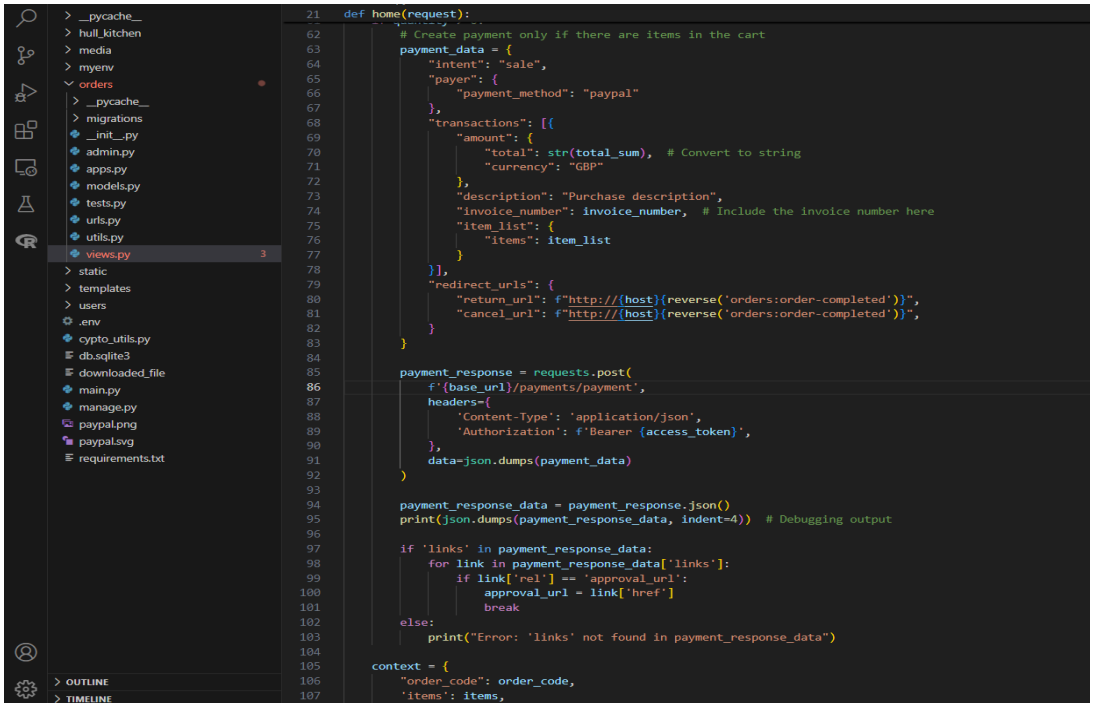


Figure 1.6 A screenshot showing the sample code to create payment.


```
{
    "id": "PAYID-M2TCVUQ399207535P808201Y",
    "intent": "sale",
    "state": "created",
    "payer": {
        "payment_method": "paypal"
    },
    "transactions": [
        {
            "amount": {
                "total": "15.35",
                "currency": "GBP"
            },
            "description": "Purchase description",
            "invoice_number": "ddae819f-96b5-4782-8bc1-59bfa1f8c9b7",
            "item_list": {
                "items": [
                    {
                        "name": "Amala & Ewedu",
                        "price": "15.35",
                        "currency": "GBP",
                        "quantity": 1
                    }
                ]
            },
            "related_resources": []
        }
    ],
    "create_time": "2024-07-28T11:26:10Z",
    "links": [
        {
            "href": "https://api.sandbox.paypal.com/v1/payments/payment/PAYID-M2TCVUQ399207535P808201Y",
            "rel": "self",
            "method": "GET"
        },
        {
            "href": "https://www.sandbox.paypal.com/cgi-bin/webscr?cmd=_express-checkout&token=EC-7F849616AD471820R",
            "rel": "approval_url",
            "method": "REDIRECT"
        },
        {
            "href": "https://api.sandbox.paypal.com/v1/payments/payment/PAYID-M2TCVUQ399207535P808201Y/execute",
            "rel": "execute",
            "method": "POST"
        }
    ]
}
```


Sample Payload / response after Payment execution

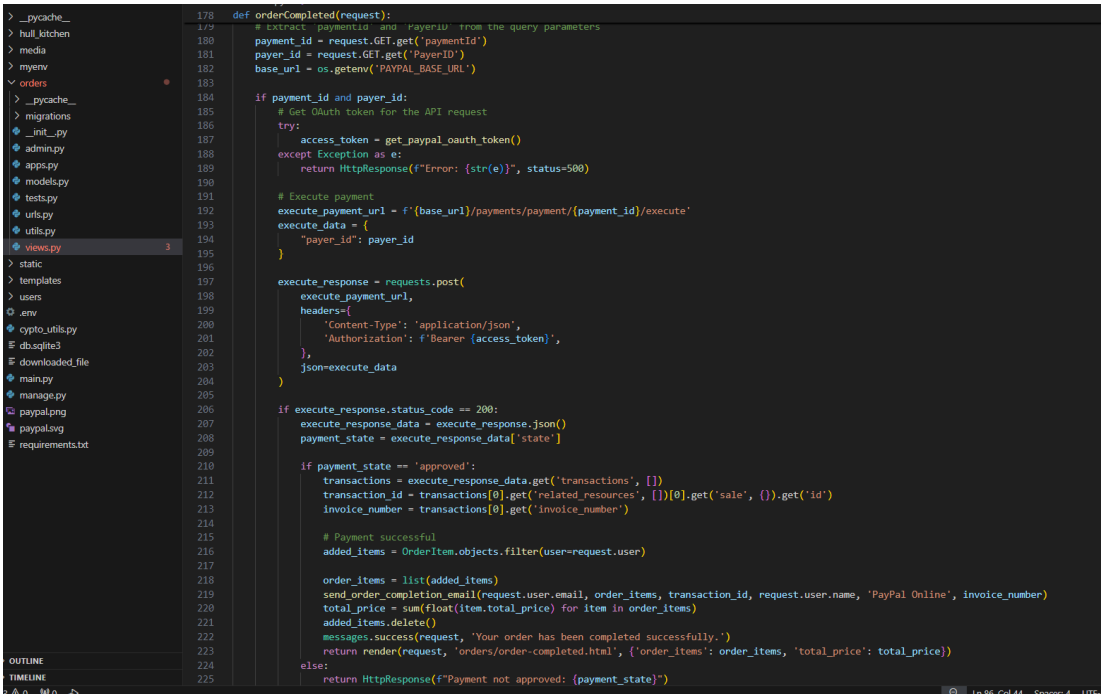


Figure 1.7 A screenshot showing the sample code to execute a payment.

There are several other REST PayPal APIs, however in the context of this application as seen here Rest API. However, only the one as explained above was utilized. For test purposes, an error log is documented on sandbox, detailing the number of successful and unsuccessful API calls.

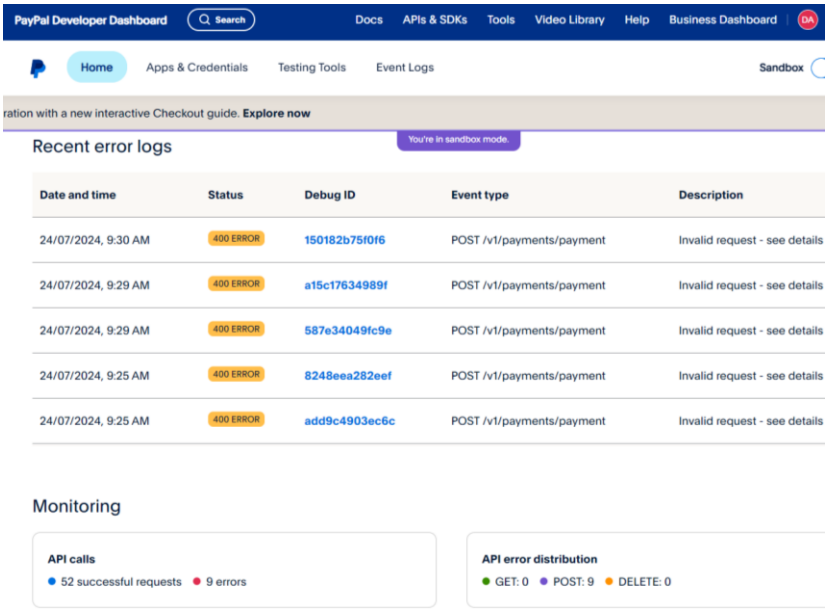


Figure 1.8 A screenshot showing all API calls, including successful and unsuccessful calls.

System Weakness

Despite its robust security measures, PayPal's system has several weaknesses that can be exploited.

- Phishing and social engineering attacks are significant threats, where users can be tricked into revealing their login credentials through fake websites or emails, leading to unauthorized transactions.
- Integration complexity and inconsistent documentation can result in improper implementation, creating security vulnerabilities such as misconfigurations or weak coding practices.
- The reliance on TLS for data encryption during transmission and AES for data at rest, while secure, can still be vulnerable if not correctly implemented or updated, potentially exposing sensitive data.
- The sandbox environment is somewhat different from the production environment, for instance using test data which may not cover all real-world scenarios and edge cases, can lead to overlooked vulnerabilities, making the live system susceptible to attacks. Moreover, the heavy reliance on PayPal's ecosystem means that any disruption or breach can have wide-reaching implications, affecting many businesses simultaneously.

Mitigation strategies

To mitigate these security weaknesses, a proactive approach is essential.

- Identifying phishing attempts and enforcing two-factor authentication (2FA) can significantly reduce the risk of account compromise. Ensuring that integration follows secure coding practices and regularly updating and auditing security protocols can prevent vulnerabilities due to implementation errors.
- Regular security audits and vulnerability assessments can help identify and address potential security gaps. Additionally, using advanced email filtering and anti-phishing technologies can protect users from phishing scams.
- Finally, diversifying payment processors and having contingency plans can reduce dependency on PayPal and minimize the impact of any potential disruptions or security breaches. By implementing these measures, businesses can strengthen their security posture and ensure more reliable and secure

Encryption and Hand-shaking protocols taking place behind the scenes

PayPal employs advanced encryption and handshaking protocols to secure data transmission between the (Merchant) university catering system and the student (buyer).

- The student selects an item from the university catering app and proceeds to checkout.
- At the payment step, the student chooses PayPal as the payment gateway.
- The app prompts the student to log into their PayPal account and enter their payment details.
- Upon submission, PayPal uses Transport Layer Security (TLS) or Secured Socket Layer (SSL) to encrypt the student's payment information.
- This encryption ensures that sensitive details, like payment information and personal data, are kept confidential and secure from unauthorized access.
- Unique session keys are generated to encode the data being transmitted.
- These session keys make the data unreadable to anyone other than PayPal and the university catering system, ensuring a secure transaction.

Handshaking Protocol

The handshaking protocol further enhances security by verifying the identities of the university catering system and PayPal before any data is exchanged. After the student checks out,

- The student (client) initiates the payment process by sending a "Client Hello" message to the university catering system, which includes a request to connect to PayPal.
- The university catering system forwards the "Client Hello" message to PayPal to handle the transaction.
- PayPal responds with a "Server Hello" message, including its digital certificate to prove its identity to the university catering system.
- The university catering system verifies the digital certificate and generates a secret key.
- The university catering system securely shares the secret key with PayPal.
- Both PayPal and the university catering system use this secret key to encrypt and decrypt the information they exchange during the session.
- This handshaking protocol ensures that the transaction is secure and trustworthy, protecting the student's sensitive information throughout the payment process.

Program Development and Testing

Python was used as the programming language for building the application, and the Django web framework was utilized for both the client and server sides. SQLite was chosen as the database to store user orders and personal identifiable information, such as names and email addresses. The application also includes an email service that sends receipts to students (customers) upon successful orders via PayPal or Cash. These receipts include transaction IDs, invoice numbers, and details of purchased items with their prices, enhancing transparency and aiding in dispute resolution if needed. Below is a screenshot of various pages of the application, showcasing the integration of the PayPal API as discussed. The **python-dotenv** needs to be installed to load environment variables.

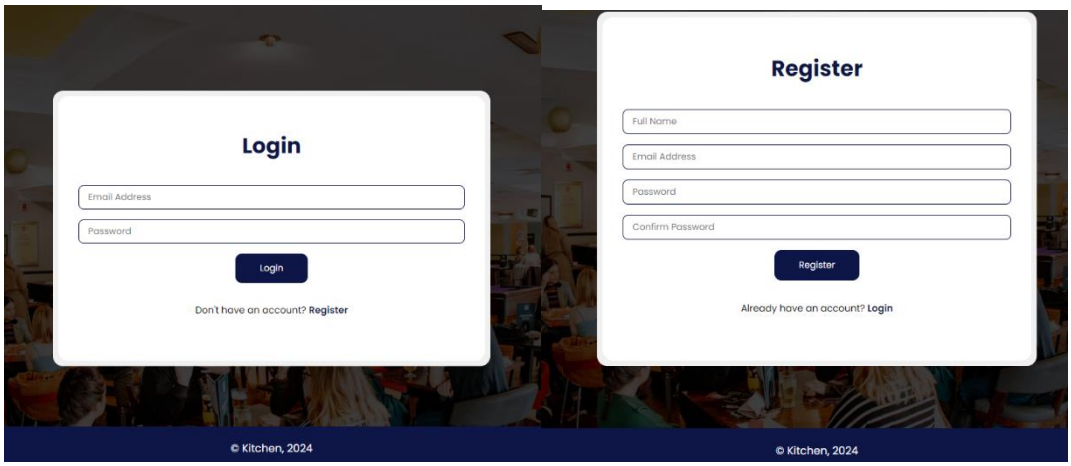


Figure 1.8 A screenshot showing the login and registration page.

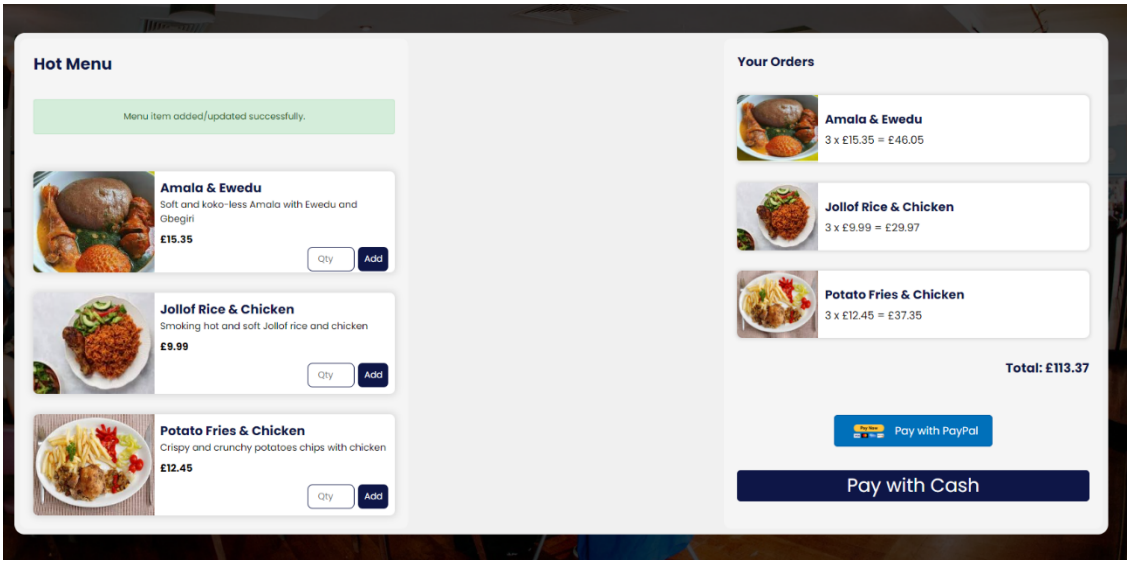


Figure 1.9 A screenshot showing cart page.

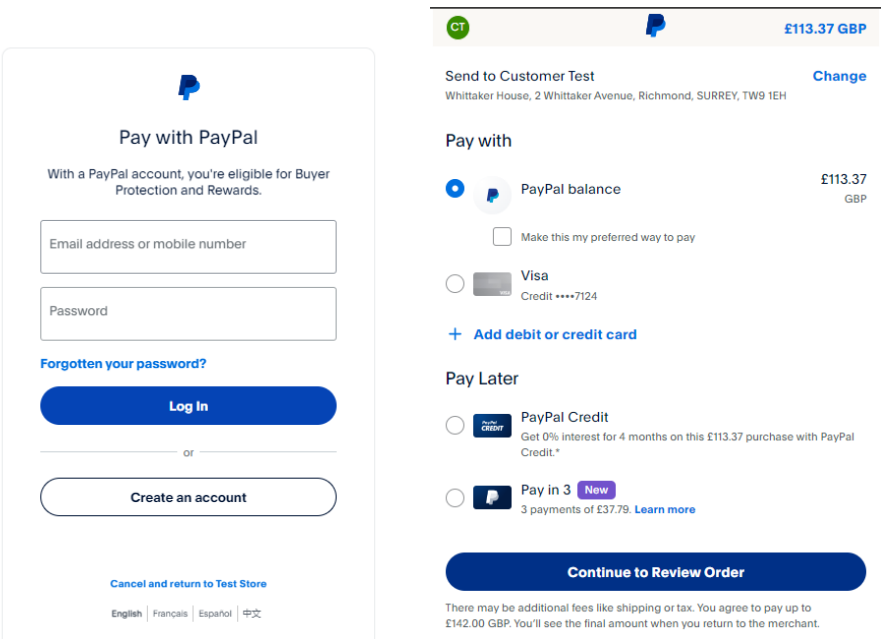


Figure 1.10 A screenshot showing PayPal verification page for customers before making payment, and also a page that allows the user to select card for payment.

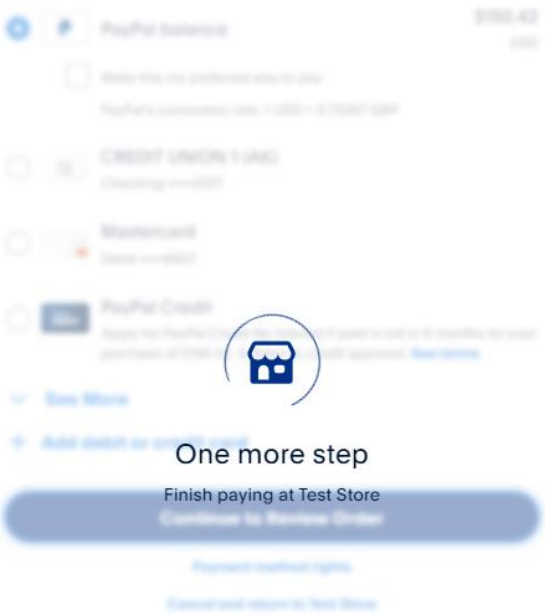


Figure 1.11 A screenshot showing the PayPal Payment processing page.

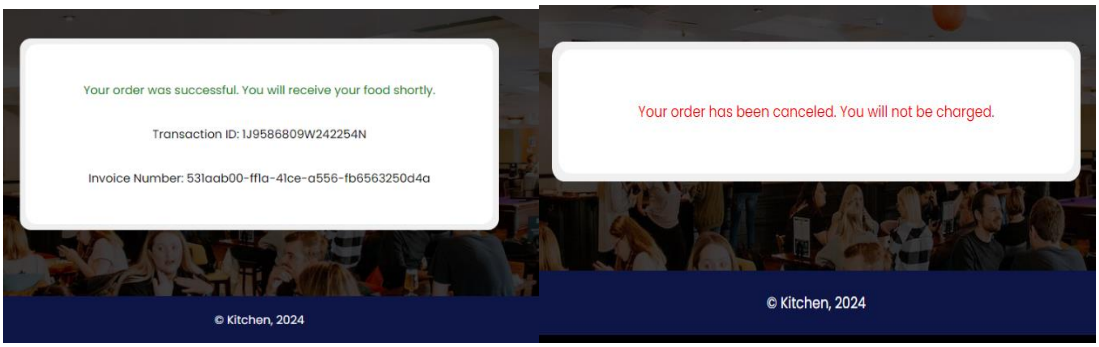


Figure 1.12 A screenshot showing Application success and error/cancel page after processing or rejection via PayPal.

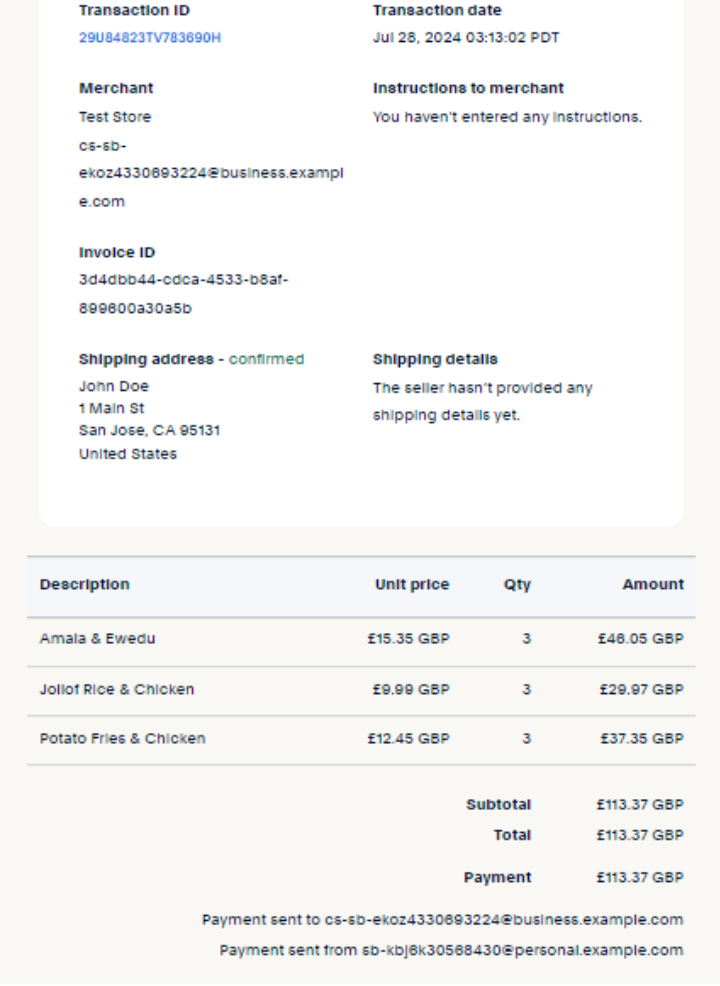


Figure 1.13 A screenshot showing the sandbox notification – This is received by both buyer and merchant.

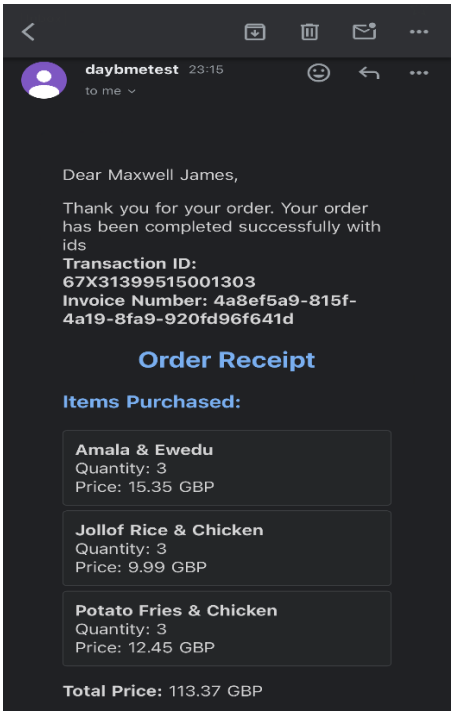


Figure 1.14 A screenshot showing the email receipt received after successful payment (triggered from the application, not sandbox).

Pseudocode:

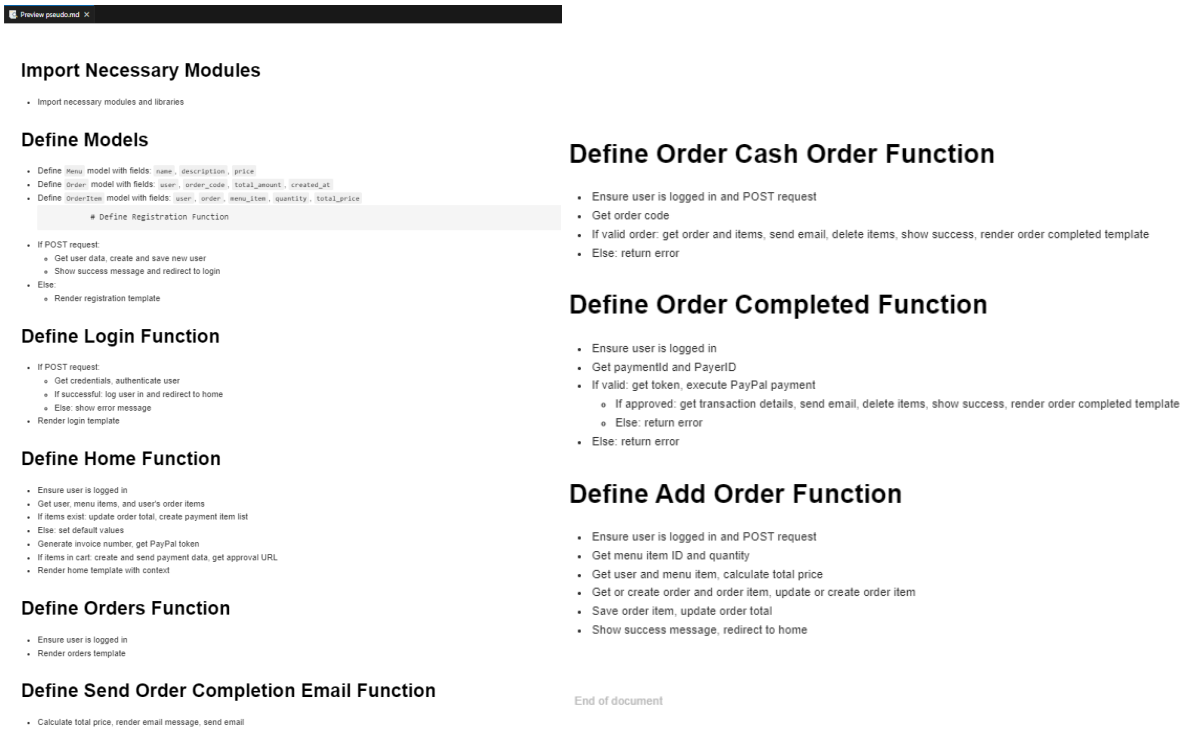


Figure 1.13 A Pseudocode detailing reasoning behind the application using PayPal

Authentication Encryption Methods

Authenticated Encryption (AE)

Authenticated Encryption (AE) is a cryptographic technique introduced in 2000 that ensures data secrecy and integrity simultaneously (Jimale et al., 2022). AE is designed to achieve the following key objectives, otherwise known as the AE modes:

I. **Confidentiality**: Data is encrypted, preventing unauthorized access and ensuring that only intended recipients can read the information.
II. **Integrity**: Ensures that the data has not been altered or tampered with during transmission or storage. This is verified using integrity checks that detect any changes to the original data.
III. **Authentication**: Confirms that the data originates from a legitimate source, preventing forgery or impersonation. This is achieved through techniques such as digital signatures or Message Authentication Codes (MACs).

Generic Composition Methods of Authenticated Encryption (AE)

There are three conventional approaches to constructing an AE scheme, known as generic composition (Namprempre et al., 2014):

1. **Encrypt-and-MAC**: Encrypt the data, then apply a MAC to the plaintext. Send both ciphertext and MAC to the receiver.
2. **MAC-then-Encrypt**: Apply a MAC to the plaintext, then encrypt both data and MAC together. Send the combined encrypted data and MAC.
3. **Encrypt-then-MAC**: Encrypt the data first, then apply a MAC to the ciphertext. Send both ciphertext and MAC to the receiver.

Encrypt-then-MAC is considered the most secure method, as it ensures the ciphertext has not been tampered with before decryption (Namprempre et al., 2014).

Single-Pass AE Modes

Single-pass Authenticated Encryption (AE) modes provide both encryption and authentication in one operation, ensuring data confidentiality and integrity during transmission or storage (Eichlseder, 2024). Their efficiency and robust security make them ideal for fast-processing systems. The main types include:

1. **Galois/Counter Mode (GCM)**: Combines Counter (CTR) mode encryption with Galois mode authentication in a single pass (Durad et al., 2015). In one pass, data is encrypted using the CTR mode while simultaneously generating an authentication tag using Galois field operations.
2. **Offset Codebook (OCB) Mode**: Uses block cipher encryption with offset values for both encryption and authentication in a single pass. The block cipher encrypts the data blocks while also generating authentication tags using the offset values.

Application to University Catering system incorporating Paypal

Authenticated Encryption (AE) is essential for securing the PayPal University Catering System, which handles sensitive financial transactions and personal data. By implementing AE, PayPal ensures data confidentiality by encrypting all transaction information. For example, when a student uses PayPal to pay for their meal order through the university catering system, AE encrypts the payment details. This encryption prevents unauthorized access, ensuring that only the intended recipient (the catering service) can access the sensitive payment information, protecting against data breaches and cyber threats.

Additionally, AE guarantees data integrity, which is crucial for maintaining accurate transaction records. For instance, when a student's payment is processed via PayPal, AE ensures that the transaction details remain unchanged during transmission. Integrity checks detect any alterations to the original data, ensuring the accuracy of the transaction. This means that the order details and payment amounts are verified, confirming that what the student ordered and paid for is exactly what the catering service receives, with no tampering during transmission.

Lastly, when a student makes a payment, AE uses techniques like digital signatures or Message Authentication Codes (MACs) to ensure the data originates from the verified student and not from a malicious actor. By utilizing single-pass AE modes such as Galois/Counter Mode (GCM) or Offset Codebook (OCB) Mode, PayPal efficiently combines encryption and authentication in one operation. This ensures robust security and swift processing.

References

Durad, M. H., Khan, M. N., & Ahmad, Z. (2015, 13-17 Jan. 2015). Analysis and optimization of Galois/Counter Mode (GCM) using MPI. 2015 12th International Bhurban Conference on Applied Sciences and Technology (IBCAST)

Eichlseder, M. (2024). Authenticated Encryption Schemes. *Symmetric Cryptography, Volume 1: Design and Security Proofs*, 87.

Gore, H., Singh, R. K., Singh, A., Singh, A. P., Shabaz, M., Singh, B. K., & Jagota, V. (2021). Django: Web development simple & fast. *Annals of the Romanian Society for Cell Biology*, *25*(6), 4576-4585.

Jimale, M. A., Z'aba, M. R., Kiah, M. L. B. M., Idris, M. Y. I., Jamil, N., Mohamad, M. S., & Rohmad, M. S. (2022). Authenticated encryption schemes: A systematic review. *IEEE Access*, *10*, 14739-14766.

Namprempre, C., Rogaway, P., & Shrimpton, T. (2014). Reconsidering generic composition. Advances in Cryptology–EUROCRYPT 2014: 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings 33,

Niranjanamurthy, M. (2014). E-commerce: Recommended online payment method-Paypal. *International journal of computer science and mobile computing*, *3*(7), 669-679.