Middleware and Encryption/Hashing Techniques in RealChat and RealMessenger

Middleware

EncryptionMiddleware:

The `EncryptionMiddleware` is a custom middleware implemented in both the `realchat_backend` and `realmessenger_backend` projects. This middleware is responsible for handling the encryption and decryption of request and response bodies to ensure secure communication between the backend services.

Location:

- `middleware.py`

Key Functions:

1. **process_request**: This function decrypts the request body if the request method is POST and the path is `/accounts/messages/`. It uses the Fernet encryption scheme to decrypt the base64 encoded request body.
2. **process_response**: This function decrypts the response content if the request method is POST and the path is `/accounts/messages/`. It uses the Fernet encryption scheme to decrypt the base64 encoded response content.
3. **ensure_padding**: This helper function ensures that the base64 encoded data has the correct padding.

Encryption/Hashing Techniques

Fernet Encryption

The Fernet encryption scheme from the cryptography library is used to encrypt and decrypt messages. Fernet is a symmetric encryption method that ensures that the message encrypted cannot be manipulated or read without the key.

Key Features:

- **Symmetric Encryption**: Uses the same key for encryption and decryption.
- **Base64 Encoding**: Encodes the encrypted data in base64 to ensure safe transmission over HTTP.

Usage in the Project:

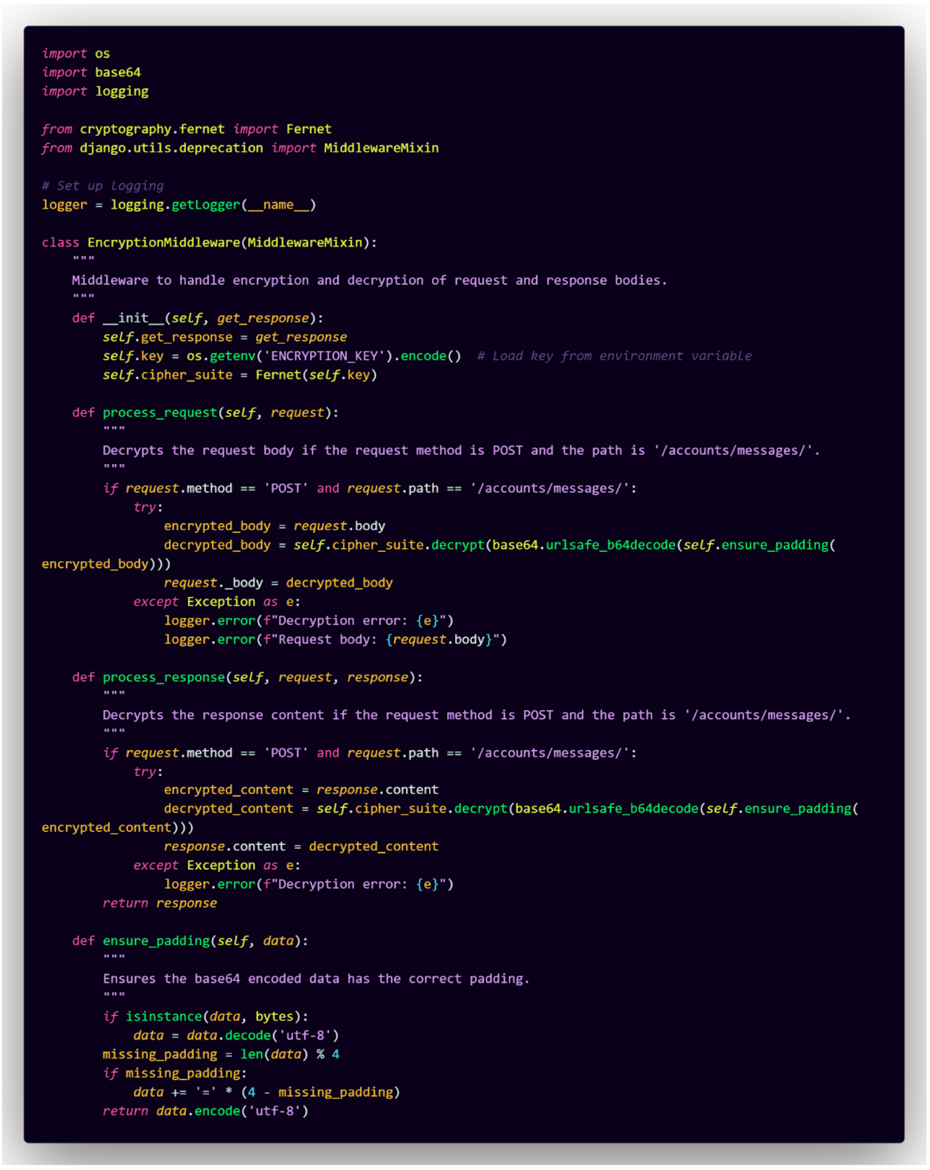
1. **Encryption**: When a message is sent, it is encrypted using the Fernet encryption scheme and then base64 encoded.

2. **Decryption**: When a message is received, it is base64 decoded and then decrypted using the Fernet encryption scheme.

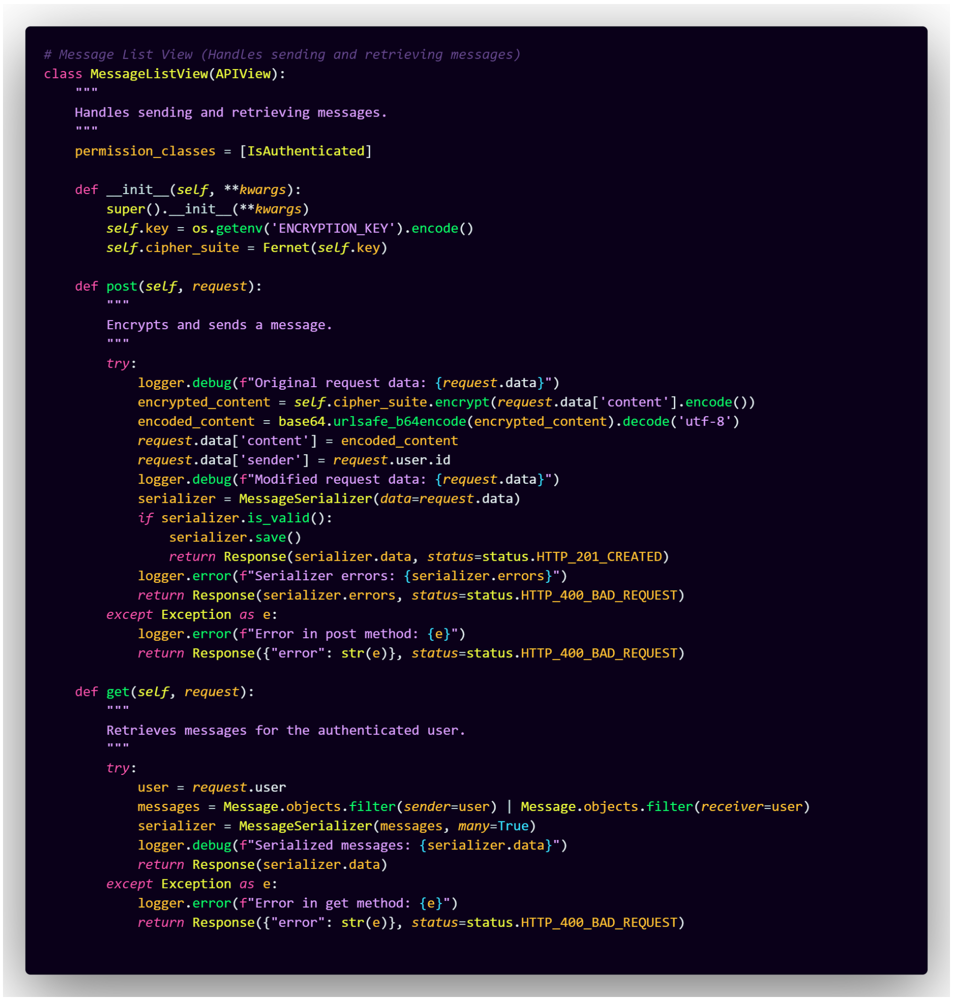
Summary

The `EncryptionMiddleware` and Fernet encryption scheme together ensure that all messages exchanged between the backend services are securely encrypted and decrypted. This approach provides a robust mechanism to protect sensitive data and maintain the integrity and confidentiality of the communication.

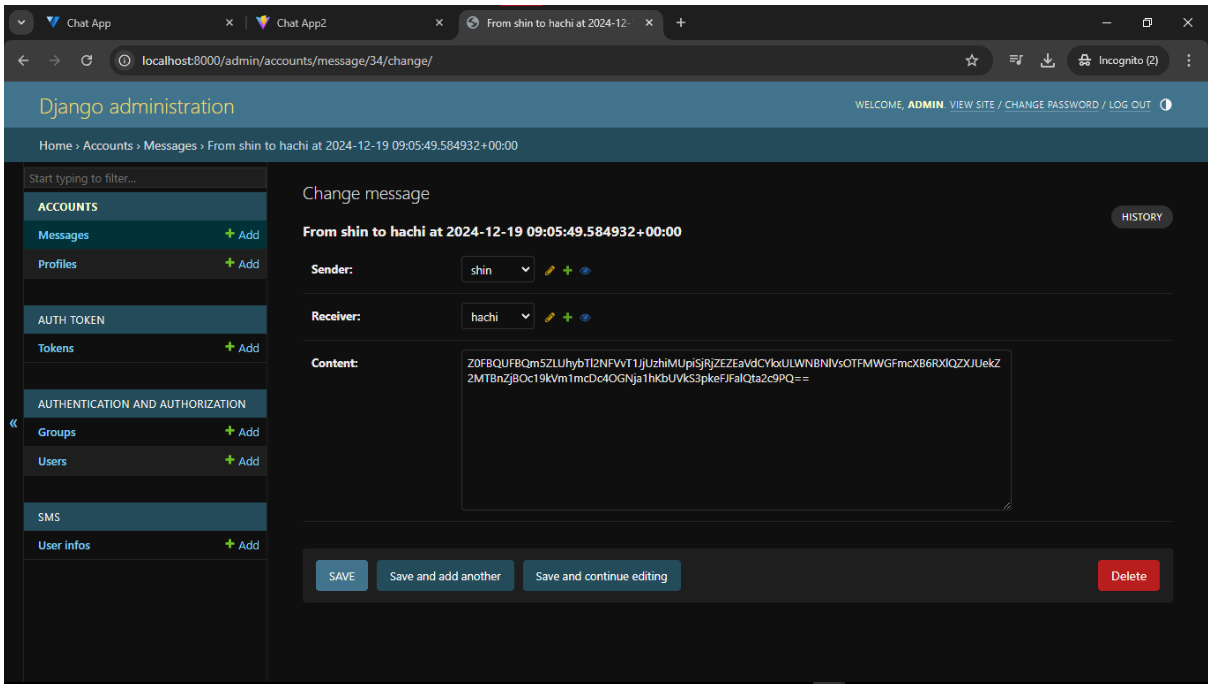
Images



```python
import os
import base64
import logging

from cryptography.fernet import Fernet
from django.utils.deprecation import MiddlewareMixin

# Set up logging
logger = logging.getLogger(__name__)

class EncryptionMiddleware(MiddlewareMixin):
    """
    Middleware to handle encryption and decryption of request and response bodies.
    """
    def __init__(self, get_response):
        self.get_response = get_response
        self.key = os.getenv('ENCRYPTION_KEY').encode()  # Load key from environment variable
        self.cipher_suite = Fernet(self.key)

    def process_request(self, request):
        """
        Decrypts the request body if the request method is POST and the path is '/accounts/messages/'.
        """
        if request.method == 'POST' and request.path == '/accounts/messages/':
            try:
                encrypted_body = request.body
                decrypted_body = self.cipher_suite.decrypt(base64.urlsafe_b64decode(self.ensure_padding(
encrypted_body)))
                request._body = decrypted_body
            except Exception as e:
                logger.error(f"Decryption error: {e}")
                logger.error(f"Request body: {request.body}")

    def process_response(self, request, response):
        """
        Decrypts the response content if the request method is POST and the path is '/accounts/messages/'.
        """
        if request.method == 'POST' and request.path == '/accounts/messages/':
            try:
                encrypted_content = response.content
                decrypted_content = self.cipher_suite.decrypt(base64.urlsafe_b64decode(self.ensure_padding(
encrypted_content)))
                response.content = decrypted_content
            except Exception as e:
                logger.error(f"Decryption error: {e}")
        return response

    def ensure_padding(self, data):
        """
        Ensures the base64 encoded data has the correct padding.
        """
        if isinstance(data, bytes):
            data = data.decode('utf-8')
        missing_padding = len(data) % 4
        if missing_padding:
            data += '=' * (4 - missing_padding)
        return data.encode('utf-8')
```



```python
# Message List View (Handles sending and retrieving messages)
class MessageListView(APIView):
    """
    Handles sending and retrieving messages.
    """
    permission_classes = [IsAuthenticated]

    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.key = os.getenv('ENCRYPTION_KEY').encode()
        self.cipher_suite = Fernet(self.key)

    def post(self, request):
        """
        Encrypts and sends a message.
        """
        try:
            logger.debug(f"Original request data: {request.data}")
            encrypted_content = self.cipher_suite.encrypt(request.data['content'].encode())
            encoded_content = base64.urlsafe_b64encode(encrypted_content).decode('utf-8')
            request.data['content'] = encoded_content
            request.data['sender'] = request.user.id
            logger.debug(f"Modified request data: {request.data}")
            serializer = MessageSerializer(data=request.data)
            if serializer.is_valid():
                serializer.save()
                return Response(serializer.data, status=status.HTTP_201_CREATED)
            logger.error(f"Serializer errors: {serializer.errors}")
            return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
        except Exception as e:
            logger.error(f"Error in post method: {e}")
            return Response({"error": str(e)}, status=status.HTTP_400_BAD_REQUEST)

    def get(self, request):
        """
        Retrieves messages for the authenticated user.
        """
        try:
            user = request.user
            messages = Message.objects.filter(sender=user) | Message.objects.filter(receiver=user)
            serializer = MessageSerializer(messages, many=True)
            logger.debug(f"Serialized messages: {serializer.data}")
            return Response(serializer.data)
        except Exception as e:
            logger.error(f"Error in get method: {e}")
            return Response({"error": str(e)}, status=status.HTTP_400_BAD_REQUEST)
```



localhost:8000/admin/accounts/message/34/change/

Django administration

WELCOME, **ADMIN**. VIEW SITE / CHANGE PASSWORD / LOG OUT

Home › Accounts › Messages › From shin to hachi at 2024-12-19 09:05:49.584932+00:00

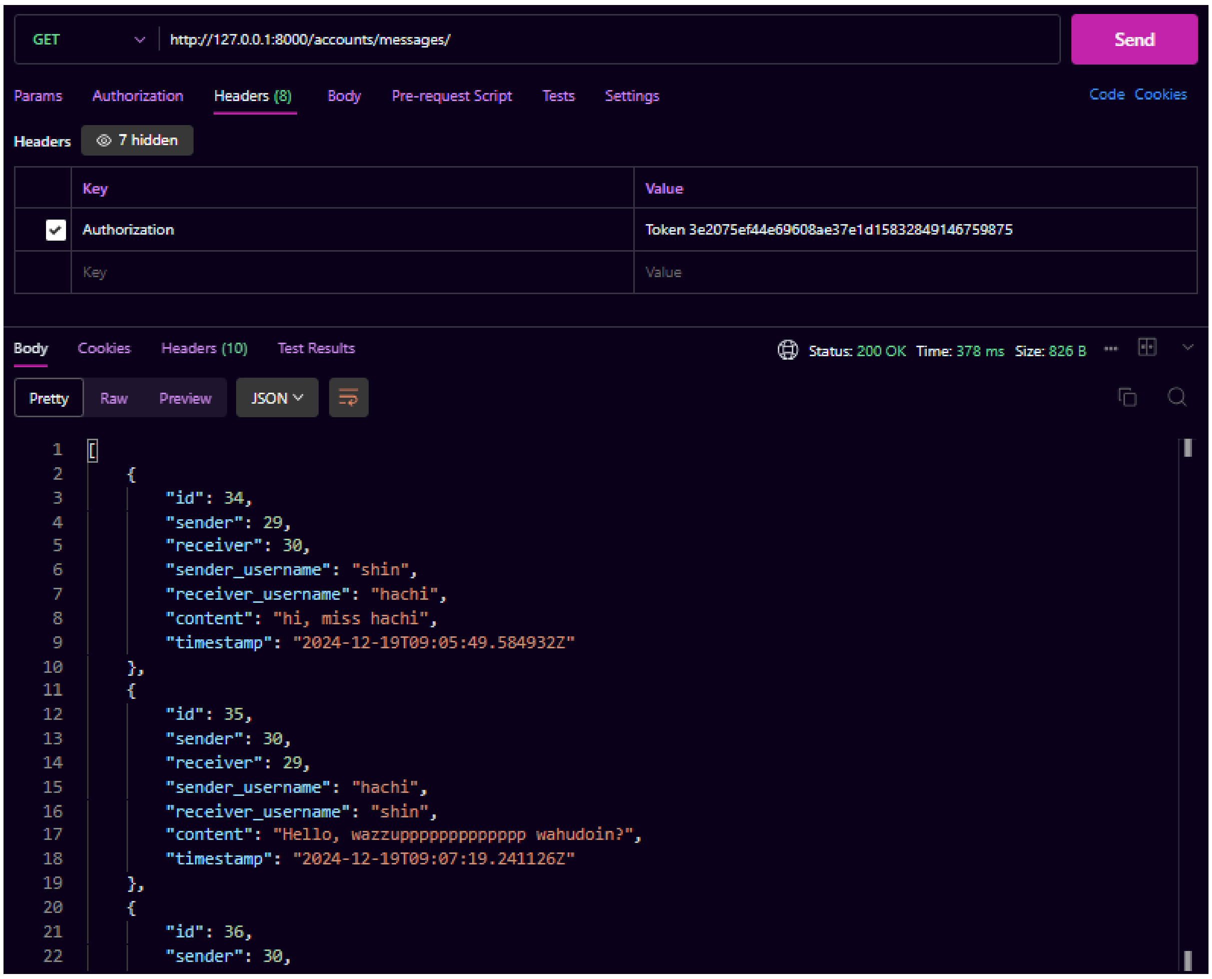
Change message

From shin to hachi at 2024-12-19 09:05:49.584932+00:00

Sender: shin

Receiver: hachi

Content: Z0FBQUFBQm5ZLUhybTl2NFVvT1JjUzhiMUpiSjRjZEZEaVdCYkxULWNBNlVsOTFMWGFmcXB6RXlQZXJUekZ2MTBnZjBOc19kVm1mcDc4OGNja1hKbUVkS3pkeFJFalQta2c9PQ==



```
http://127.0.0.1:8000/accounts/messages/
```

	Key	Value
✓	Authorization	Token 3e2075ef44e69608ae37e1d15832849146759875
	Key	Value

Status: 200 OK Time: 378 ms Size: 826 B

```
[
    {
        "id": 34,
        "sender": 29,
        "receiver": 30,
        "sender_username": "shin",
        "receiver_username": "hachi",
        "content": "hi, miss hachi",
        "timestamp": "2024-12-19T09:05:49.584932Z"
    },
    {
        "id": 35,
        "sender": 30,
        "receiver": 29,
        "sender_username": "hachi",
        "receiver_username": "shin",
        "content": "Hello, wazzuppppppppppppp wahudoin?",
        "timestamp": "2024-12-19T09:07:19.241126Z"
    },
    {
        "id": 36,
        "sender": 30,
```