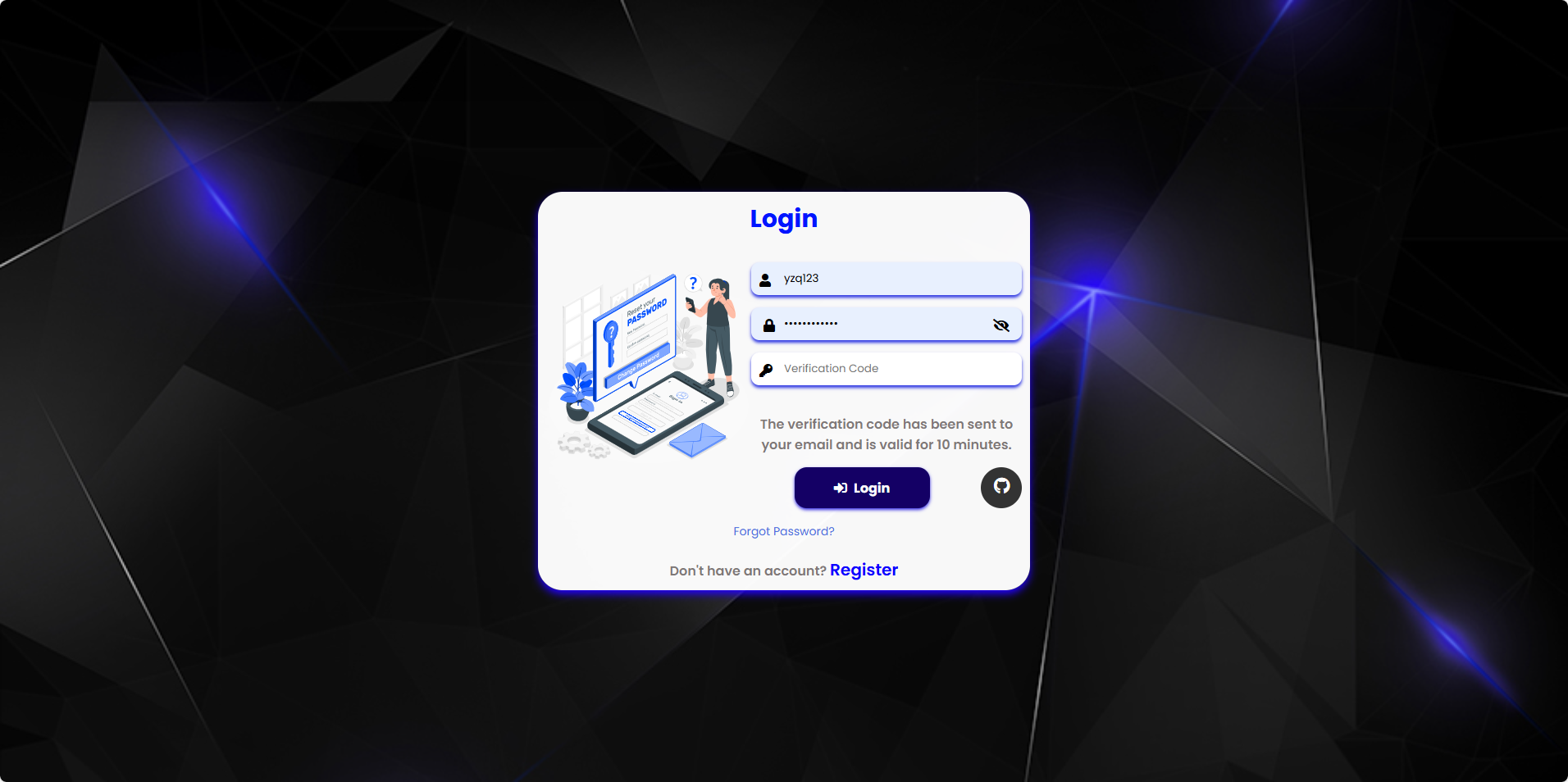
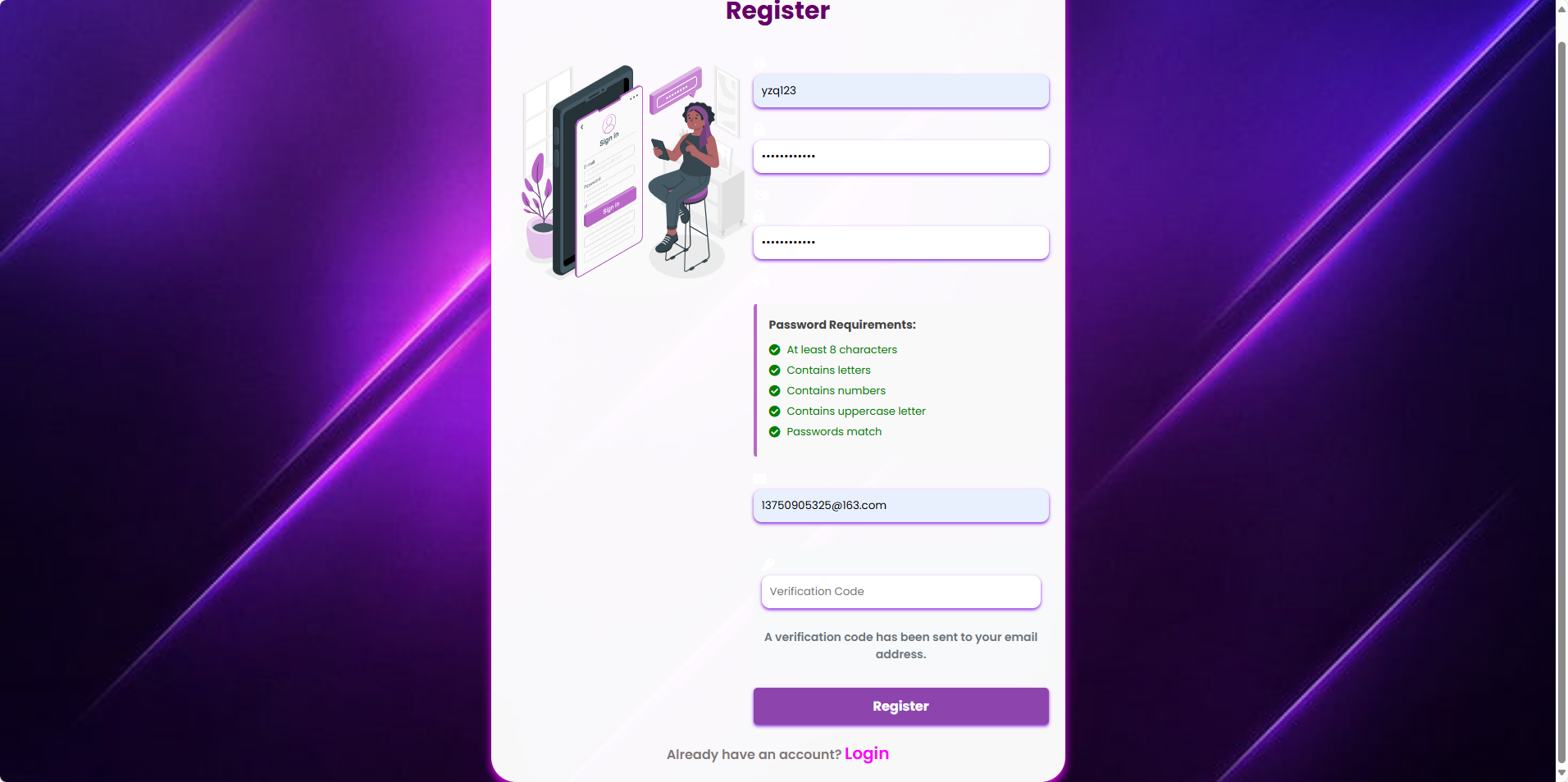
**Prototype Demonstration**

First, because Wireshark may try to capture sensitive information (e.g. usernames, passwords, chats, etc.) in clear text, we implemented TLS encrypted communication for our web application, upgrading it from the original HTTP protocol to HTTPS, and this section documents the whole process from certificate generation to the Flask project's access to TLS. First, we generate a self-signed certificate and a private key locally using OpenSSL: cert.pem: server-side public key certificate and key.pem: private key. Then, in the Flask project, we enable TLS using the ssl\_context parameter, simply by specifying the path to cert.pem and key.pem at runtime. With this encryption, even if an attacker uses Wireshark to capture packets on the same network, he will not be able to obtain valid sensitive information. The data transmission process is changed from plaintext to encrypted stream, which significantly improves the security of user authentication, chat and other operations.

Next, we add the authentication scheme in the main interface, that is, we add the email verification code as a double guarantee, to prevent hackers from breaking in violently. To cooperate with the email authentication code function, we modify the specific requirements during registration, besides the simple username and password, we add the registration of additional email information. The specific information is shown in the figure below.

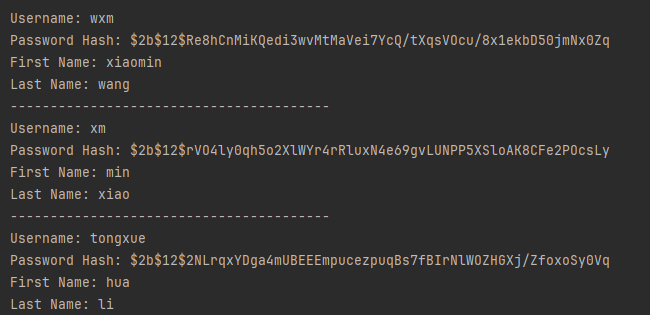


New login screen



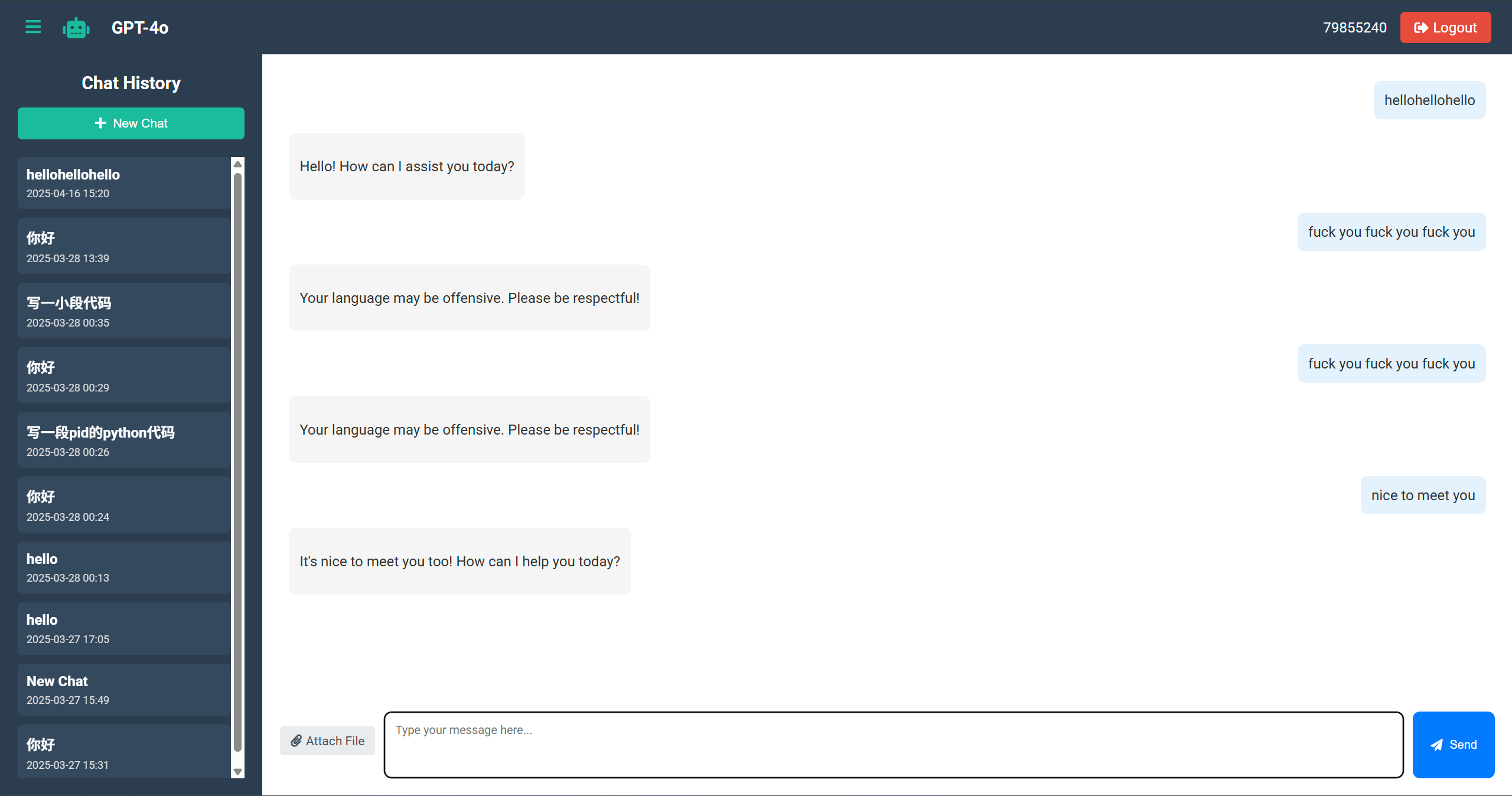
New registration screen

At the same time we are registering information into the database, considering the use of the MongoDB database itself has a very strong privacy protection mechanism, so we do not consider for the time being the processing of this database. But in this particular case, after grabbing the .env file, we also have the opportunity to enter the database through the URI. Therefore, when we store the relevant information, as shown in the figure below, we encrypt the password with a hash algorithm, which can effectively protect the user's information even if it is illegally invaded.

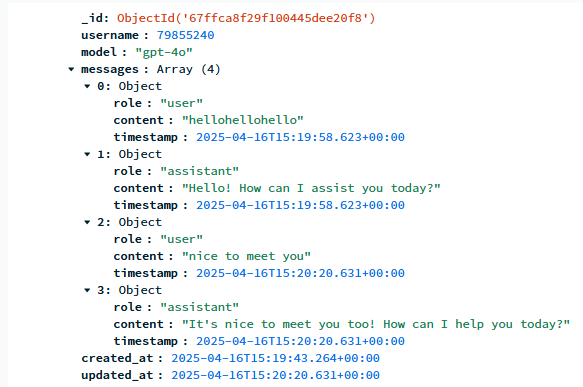


Partially encrypted information

Meanwhile, we added a machine learning mechanism to block sensitive words in the user chat interface to prevent them from entering the database and polluting the content. As shown in the figure, although we present four dialogues in the dialogue box, the AI system detects that some of the dialogues contain sensitive words and therefore initiates a warning. And we can also clearly see in the database that the content stored in the database does not contain the sensitive content that was warned.



Chat interface(after anti-pollution)



Database information (after anti-pollution)