Professor: Massimo Villari

Project Advisor: Mario Colosi

Student: Arya Khosravirad

Matricola: 534 170

Università degli Studi di Messina

End-to-End Encrypted Real-Time Chat Application

Table Of Contents

Introduction

The goal of this project is to create a secure and real time chat application where users can send and receive messages. The system uses end-to-end encryption to ensure that only the sender and receiver can read the messages while the server itself can only store the encrypted data. Key features are simple user interface, user authentication for secure logins, and real time message updates using AJAX. and the application is designed to run in a containerized environment by using docker which makes it easier to deploy and ensures the system runs consistently across different setups. This project prioritizes security and user privacy providing a safe platform for private communication.

System Architecture

This chat application is built with three main parts:

client: The client is the front end interface where users interact with the app they do operations like logging in, viewing and sending messages. It uses HTML, CSS, and JavaScript, with JavaScript handling real time updates using AJAX so that messages load instantly without reloading the page

server: The server is powered by PHP and is responsible for managing user logins, encrypting and decrypting messages, and communicating with the database. The server ensures that all messages are securely encrypted before they are stored and only the intended recipient can decrypt and read them.

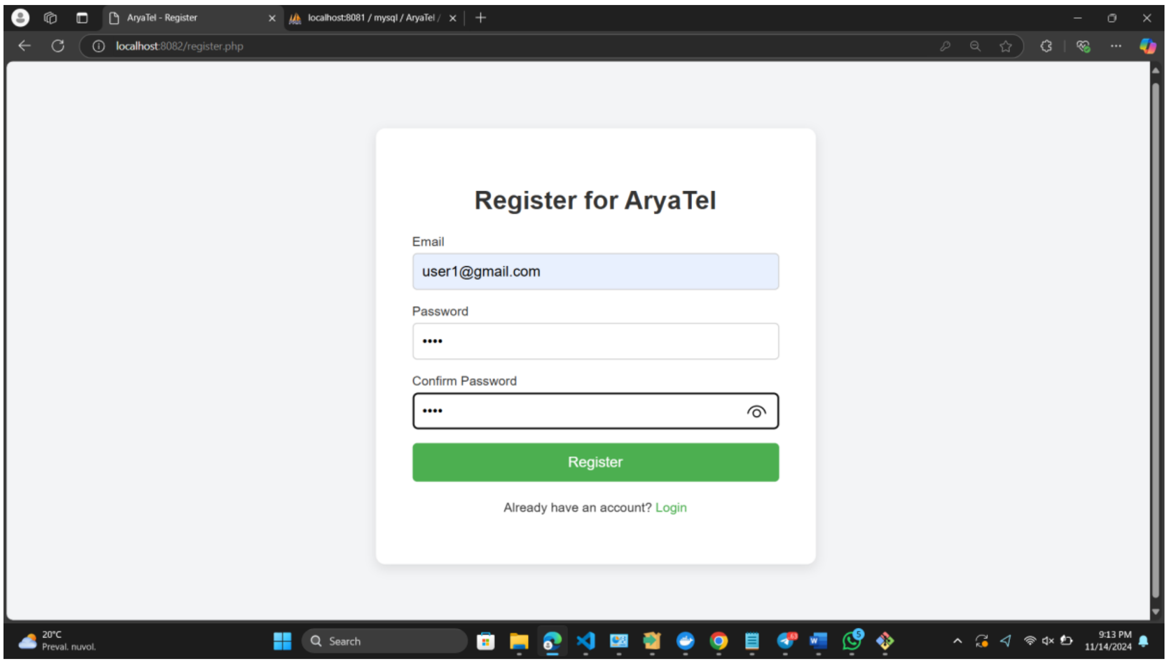
Database: The database is set up with MySQL and stores all essential data like user details, encrypted messages, and shared encryption keys that make private communication possible.

To make deployment easier and more secure, the entire application is containerized with Docker. Docker packages the client, server, and database into isolated environments, ensuring they work seamlessly together across different systems. This containerized setup also keeps each part of the application separate, reducing potential conflicts and making it easier to manage and deploy the app.

Functionality:

Registration:

The registration process allows users to create an account securely. When a user submits their email and password the system first checks if the passwords match. If yes the password is hashed using PHP's password_hash function which protects it by making it unreadable even if someone accesses the database. The hashed password and email are then safely stored in the database. This ensures that sensitive information remains secure as the actual password is never stored in plain text.


AryaTel - Register
localhost:8081 / mysql / AryaTel /
localhost:8082/register.php

Register for AryaTel

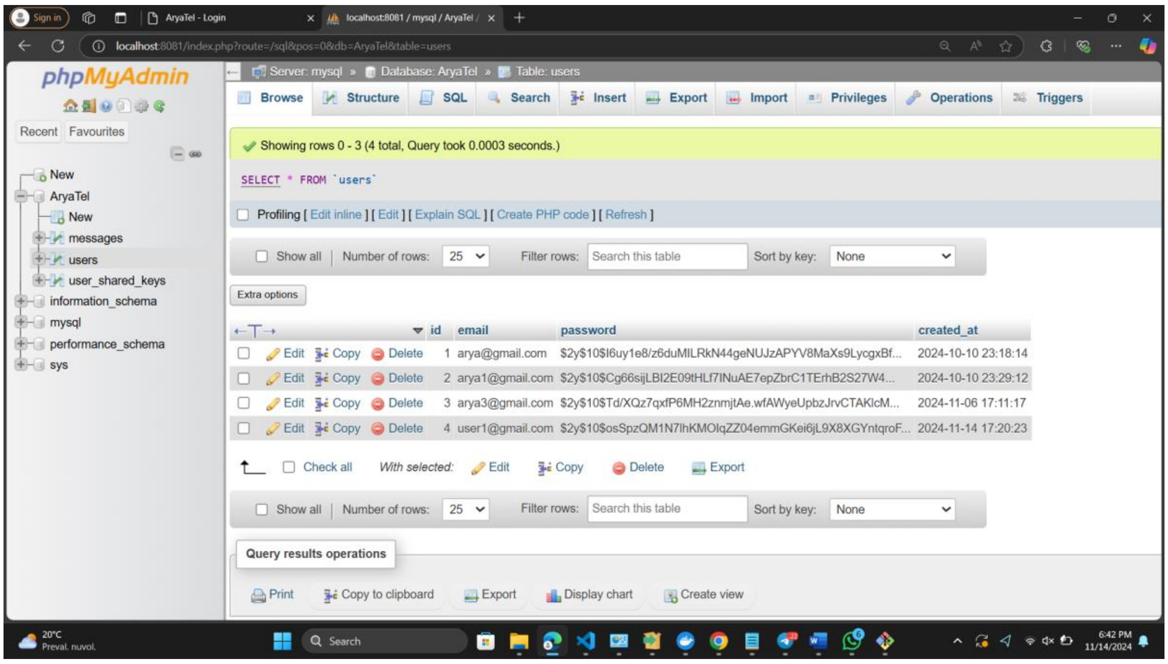
Email: user1@gmail.com

Password: ••••

Confirm Password: ••••

Register

Already have an account? Login



✓ Showing rows 0 - 3 (4 total, Query took 0.0003 seconds.)

```
SELECT * FROM `users`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None

Extra options

id	email	password	created_at
1	arya@gmail.com	$2y$10$I6uy1e8/z6duMILRkN44geNUJzAPYV8MaXs9LycgxBf...	2024-10-10 23:18:14
2	arya1@gmail.com	$2y$10$Cg66sijLBI2E09tHLf7INuAE7epZbrC1TErhB2S27W4...	2024-10-10 23:29:12
3	arya3@gmail.com	$2y$10$Td/XQz7qxfP6MH2znmjtAe.wfAWyeUpbzJrvCTAKlcM...	2024-11-06 17:11:17
4	user1@gmail.com	$2y$10$osSpzQM1N7lhKMOlqZZ04emmGKei6jL9X8XGYntqroF...	2024-11-14 17:20:23

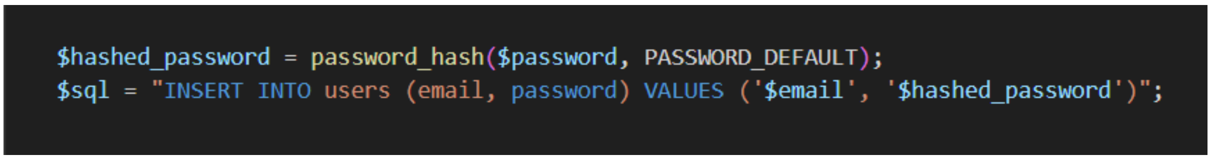
Check all With selected: Edit Copy Delete Export

Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None

Query results operations

Print Copy to clipboard Export Display chart Create view

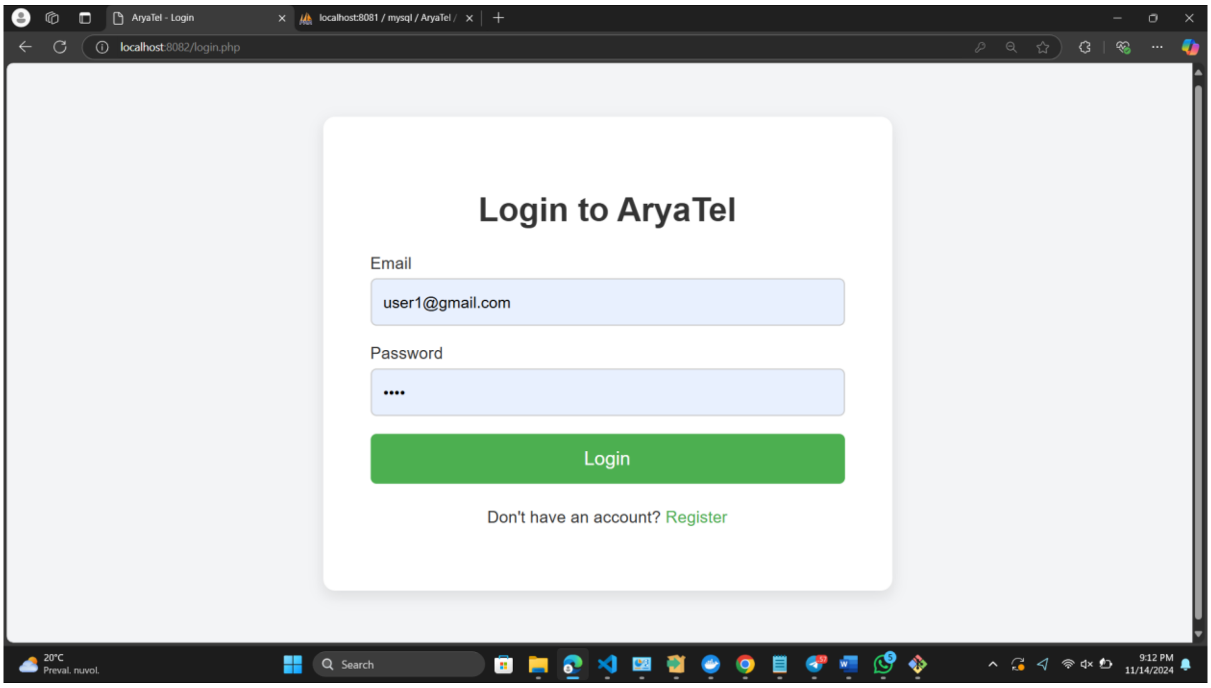
This code securely hashes the user's password before storing it in the database. The password_hash function applies a hashing algorithm to the password making it unreadable to anyone who gains access to the database.



```
$hashed_password = password_hash($password, PASSWORD_DEFAULT);
$sql = "INSERT INTO users (email, password) VALUES ('$email', '$hashed_password')";
```

Login:

After registration user can try to log in. After submitting their email and password the PHP script checks if the email exists in the database. If the email is found it retrieves the user's hashed password. Then using password_verify, it compares the entered password with the hashed password in the database. If the passwords match the login is successful and the user's ID and email are stored in a session to keep them logged in. The user is then redirected to the chat page. If the email isn't found or the password is incorrect an error message is displayed. This approach securely verifies the user's identity without exposing their actual password.



Login to AryaTel

Email

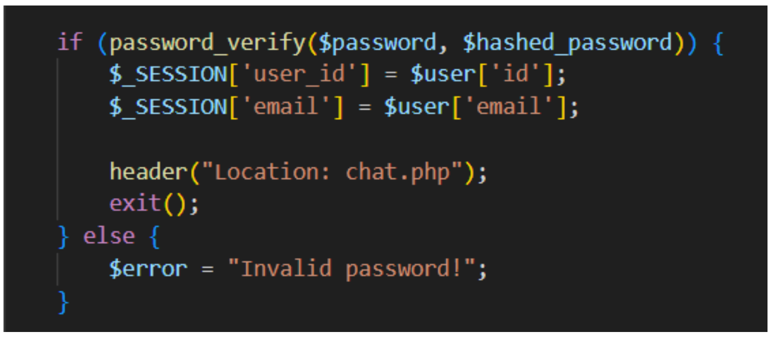
user1@gmail.com

Password

••••

Don't have an account? Register

This part verifies the user's entered password against the hashed password stored in the database ensuring only authenticated users can log in.



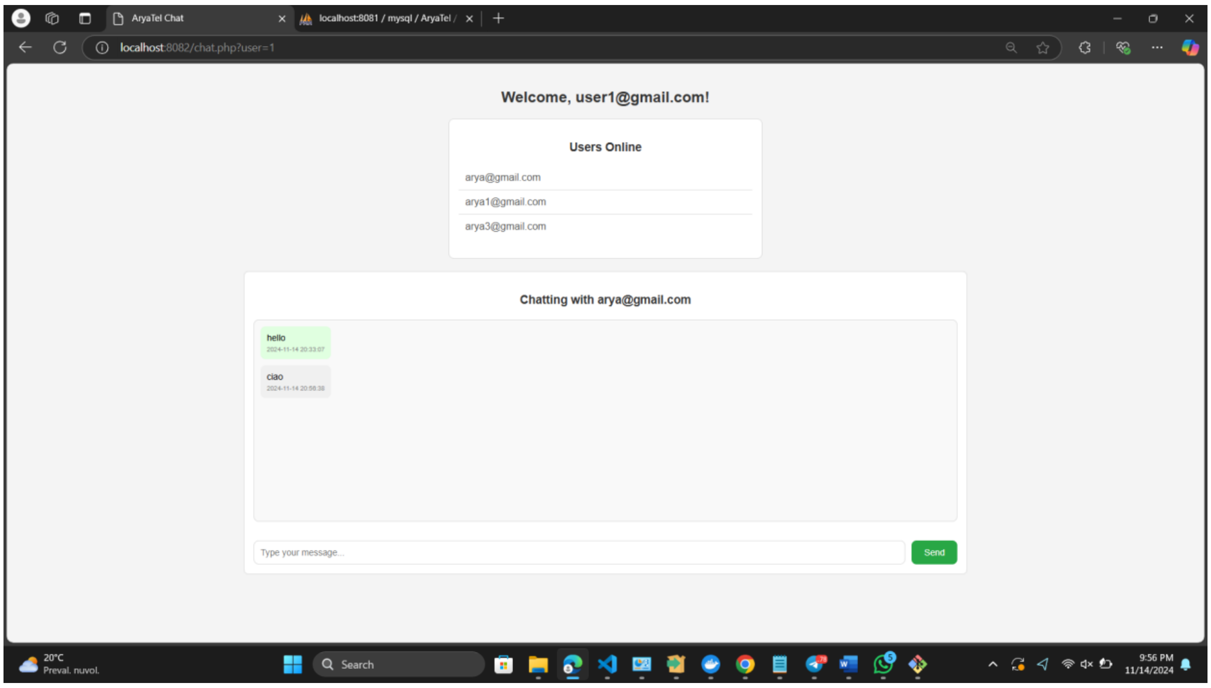
```
if (password_verify($password, $hashed_password)) {
    $_SESSION['user_id'] = $user['id'];
    $_SESSION['email'] = $user['email'];

    header("Location: chat.php");
    exit();
} else {
    $error = "Invalid password!";
}
```

End-to-End encryption:

When a user logs into the chat they see a list of other registered users. If they select a user to chat with the system checks if a unique encryption key exists for this user pair. If they are chatting for the first time a new encryption key is created and stored in the database. This shared key will be used for all future messages between these two users.

When the user types a message and hits "Send," the message is encrypted right away using AES-256 encryption with the shared key. For more security a unique initialization vector is generated for each message. The encrypted message and the IV are stored together in the database, meaning the server only has access to the scrambled unreadable version of the message.

And when the recipient opens the chat the system retrieves the encrypted message and the IV. then uses the shared key to decrypt the message so the recipient can read it. Because of E2EE only the sender and receiver can view the actual messages.



localhost:8082/chat.php?user=1

Welcome, user1@gmail.com!

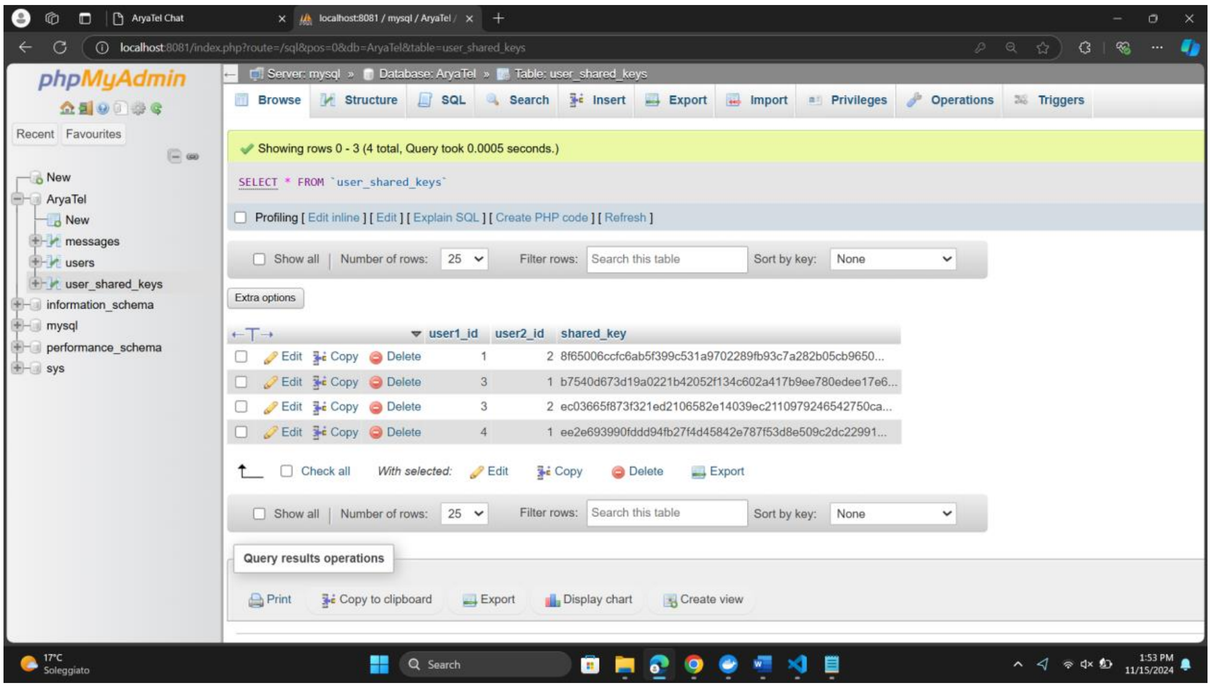
Users Online

arya@gmail.com
arya1@gmail.com
arya3@gmail.com

Chatting with arya@gmail.com

hello
2024-11-14 20:33:07

ciao
2024-11-14 20:56:38



localhost:8081/index.php?route=/sql&pos=0&db=AryaTel&table=user_shared_keys

Server: mysql » Database: AryaTel » Table: user_shared_keys

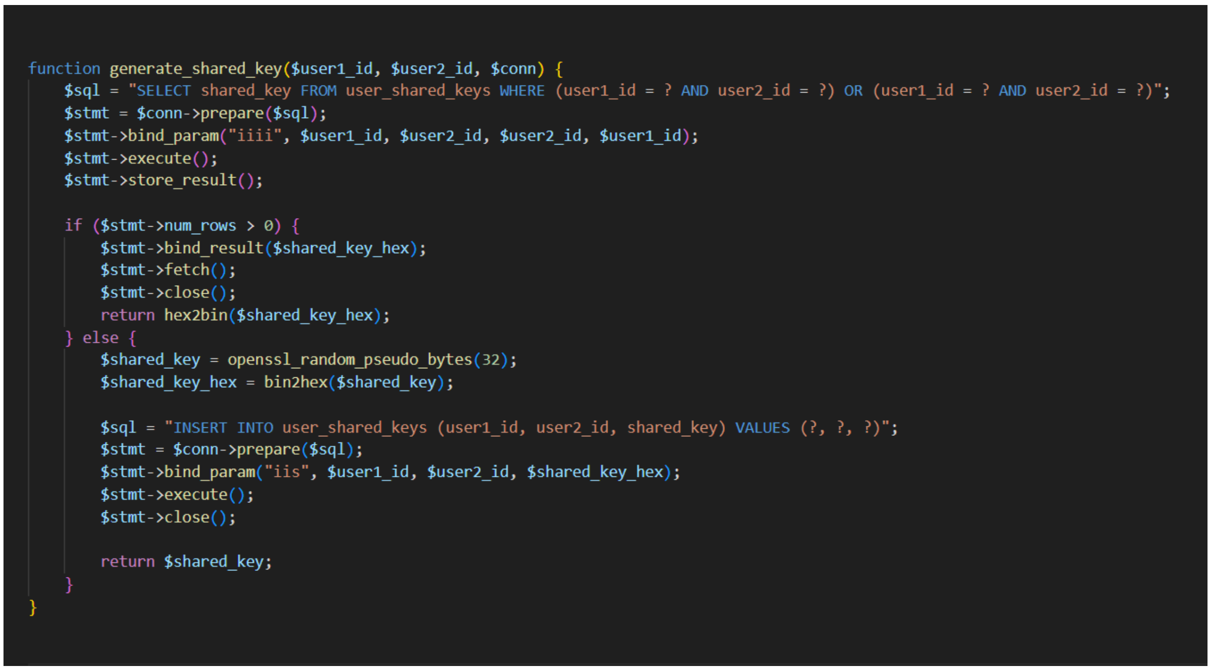
✓ Showing rows 0 - 3 (4 total, Query took 0.0005 seconds.)

```
SELECT * FROM `user_shared_keys`
```

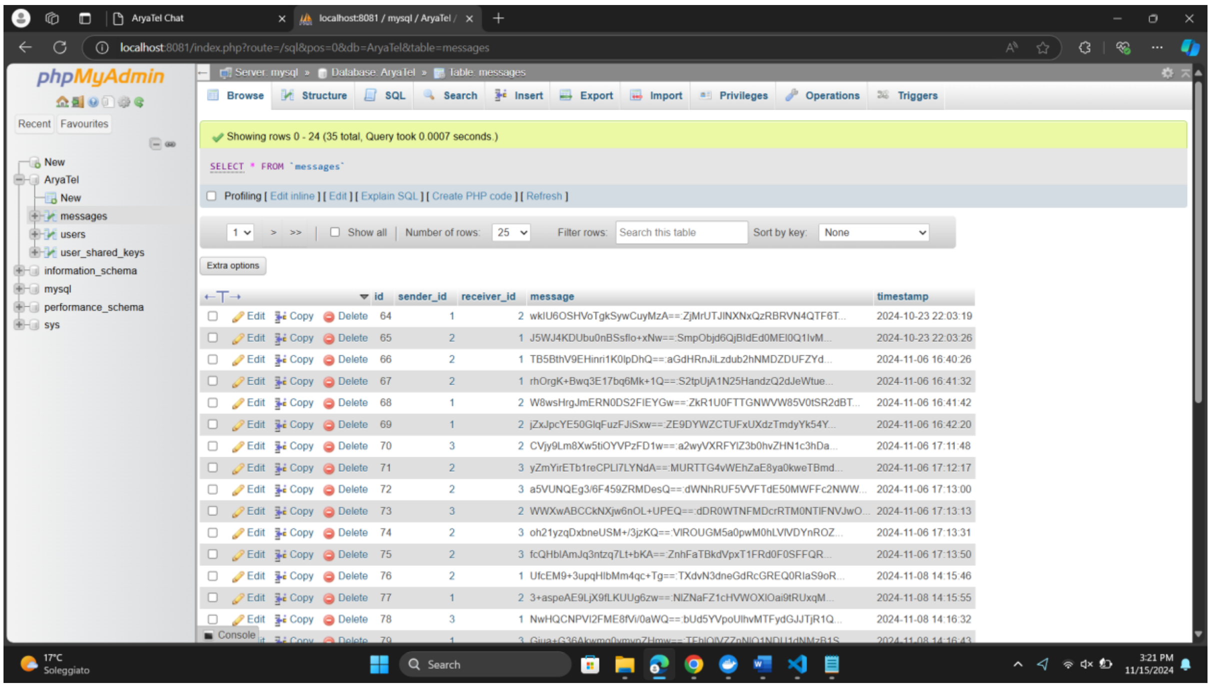
user1_id	user2_id	shared_key
1	2	8f65006ccfc6ab5f399c531a9702289fb93c7a282b05cb9650...
3	1	b7540d673d19a0221b42052f134c602a417b9ee780edee17e6...
3	2	ec03665f873f321ed2106582e14039ec2110979246542750ca...
4	1	ee2e693990fddd94fb27f4d45842e787f53d8e509c2dc22991...

Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None

This function retrieves or creates a shared encryption key for two users. If they have chatted before it fetches the existing key from the database. If not it generates a new secure key and stores it and returns it. This key is then used to encrypt and decrypt messages between the two users ensuring secure communication.(chat.php)



```php
function generate_shared_key($user1_id, $user2_id, $conn) {
    $sql = "SELECT shared_key FROM user_shared_keys WHERE (user1_id = ? AND user2_id = ?) OR (user1_id = ? AND user2_id = ?)";
    $stmt = $conn->prepare($sql);
    $stmt->bind_param("iiii", $user1_id, $user2_id, $user2_id, $user1_id);
    $stmt->execute();
    $stmt->store_result();

    if ($stmt->num_rows > 0) {
        $stmt->bind_result($shared_key_hex);
        $stmt->fetch();
        $stmt->close();
        return hex2bin($shared_key_hex);
    } else {
        $shared_key = openssl_random_pseudo_bytes(32);
        $shared_key_hex = bin2hex($shared_key);

        $sql = "INSERT INTO user_shared_keys (user1_id, user2_id, shared_key) VALUES (?, ?, ?)";
        $stmt = $conn->prepare($sql);
        $stmt->bind_param("iis", $user1_id, $user2_id, $shared_key_hex);
        $stmt->execute();
        $stmt->close();

        return $shared_key;
    }
}
```



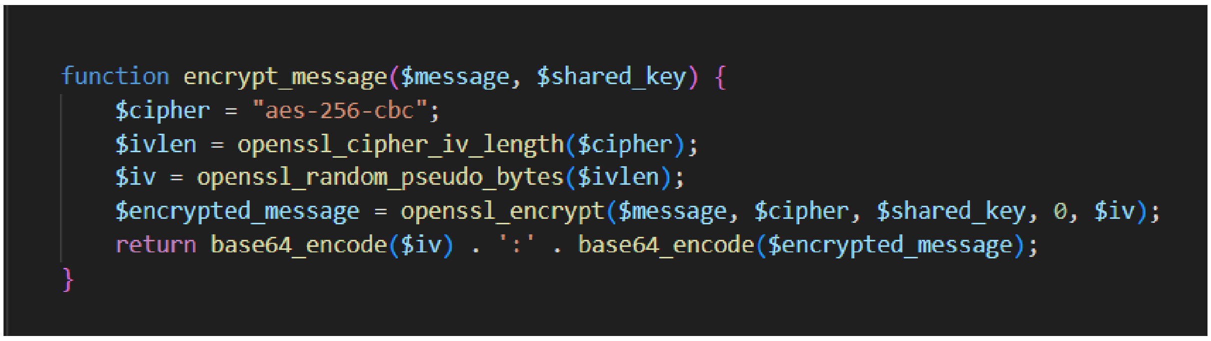
Showing rows 0 - 24 (35 total, Query took 0.0007 seconds.)

```sql
SELECT * FROM `messages`
```

id	sender_id	receiver_id	message	timestamp
64	1	2	wkIU6OSHVoTgkSywCuyMzA==:ZjMrUTJINXNxQzRBRVN4QTF6T...	2024-10-23 22:03:19
65	2	1	J5WJ4KDUbu0nBSsflo+xNw==:SmpObjd6QjBldEd0MEI0Q1IvM...	2024-10-23 22:03:26
66	2	1	TB5BthV9EHinri1K0lpDhQ==:aGdHRnJiLzdub2hNMDZDUFZYd...	2024-11-06 16:40:26
67	2	1	rhOrgK+Bwq3E17bq6Mk+1Q==:S2tpUjA1N25HandzQ2dJeWtue...	2024-11-06 16:41:32
68	1	2	W8wsHrgJmERN0DS2FIEYGw==:ZkR1U0FTTGNWVW85V0tSR2dBT...	2024-11-06 16:41:42
69	1	2	jZxJpcYE50GlqFuzFJiSxw==:ZE9DYWZCTUFxUXdzTmdyYk54Y...	2024-11-06 16:42:20
70	3	2	CVjy9Lm8Xw5tiOYVPzFD1w==:a2wyVXRFYlZ3b0hvZHN1c3hDa...	2024-11-06 17:11:48
71	2	3	yZmYirETb1reCPLI7LYNdA==:MURTTG4vWEhZaE8ya0kweTBmd...	2024-11-06 17:12:17
72	2	3	a5VUNQEg3/6F459ZRMDesQ==:dWNhRUF5VVFTdE50MWFFc2NWW...	2024-11-06 17:13:00
73	3	2	WWXwABCCkNXjw6nOL+UPEQ==:dDR0WTNFMDcrRTM0NTlFNVJwO...	2024-11-06 17:13:13
74	2	3	oh21yzqDxbneUSM+/3jzKQ==:VlROUGM5a0pwM0hLVlVDYnROZ...	2024-11-06 17:13:31
75	2	3	fcQHblAmJq3ntzq7Lt+bKA==:ZnhFaTBkdVpxT1FRd0F0SFFQR...	2024-11-06 17:13:50
76	2	1	UfcEM9+3upqHlbMm4qc+Tg==:TXdvN3dneGdRcGREQ0RlaS9oR...	2024-11-08 14:15:46
77	1	2	3+aspeAE9LjX9fLKUUg6zw==:NlZNaFZ1cHVWOXlOai9tRUxqM...	2024-11-08 14:15:55
78	3	1	NwHQCNPVl2FME8fVi/0aWQ==:bUd5YVpoUlhvMTFydGJJTjR1Q...	2024-11-08 14:16:32
79	1	3	Giua+G36Akwmq0vmvnZHmw==:TEhlQlVZZnNlQ1NDU1dNMzB1S	2024-11-08 14:16:43

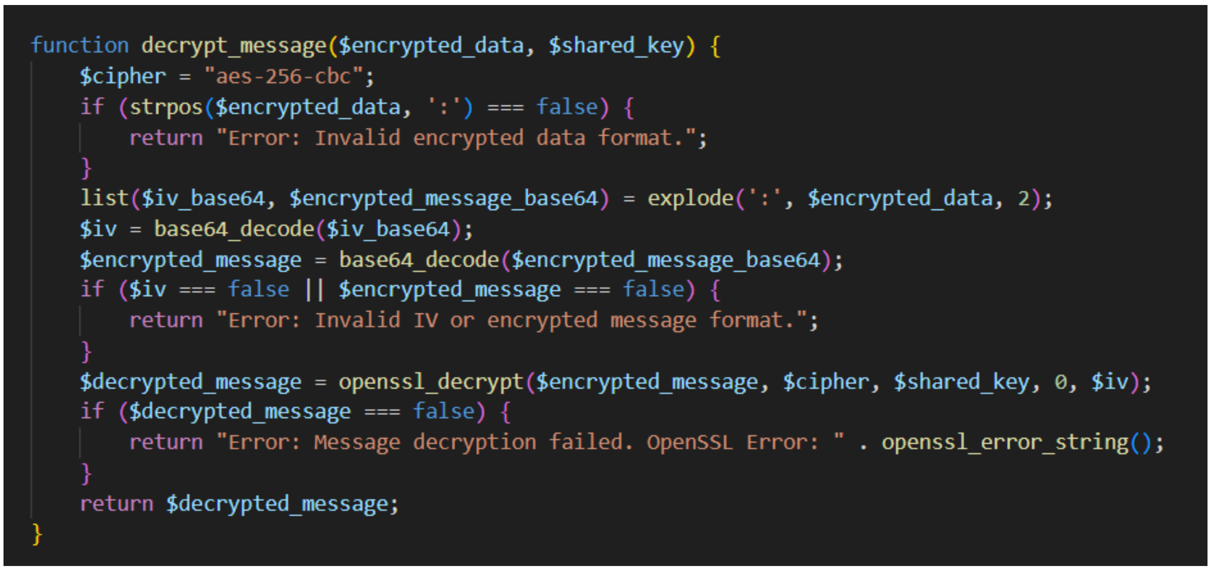
This function encrypts a message using AES-256 encryption with a unique IV for extra security.

The encrypted message and IV are combined and returned for storage ensuring that messages are unreadable to anyone except the intended recipient.(send_message.php)



```php
function encrypt_message($message, $shared_key) {
    $cipher = "aes-256-cbc";
    $ivlen = openssl_cipher_iv_length($cipher);
    $iv = openssl_random_pseudo_bytes($ivlen);
    $encrypted_message = openssl_encrypt($message, $cipher, $shared_key, 0, $iv);
    return base64_encode($iv) . ':' . base64_encode($encrypted_message);
}
```

This function decrypts an encrypted message so the recipient can read it. It separates the IV from the encrypted message then uses the shared key and IV to return the original message. This process ensures that only the intended user can access the message content.(fetch_message.php)

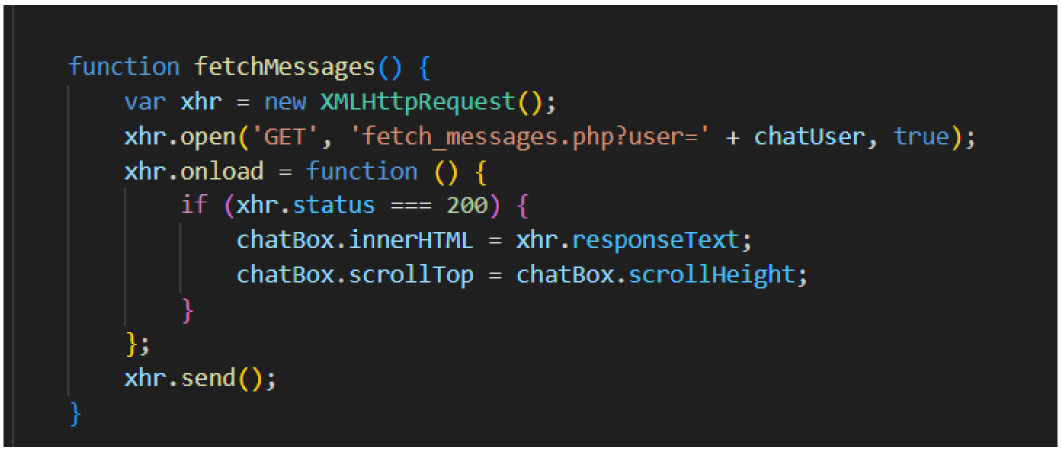


```php
function decrypt_message($encrypted_data, $shared_key) {
    $cipher = "aes-256-cbc";
    if (strpos($encrypted_data, ':') === false) {
        return "Error: Invalid encrypted data format.";
    }
    list($iv_base64, $encrypted_message_base64) = explode(':', $encrypted_data, 2);
    $iv = base64_decode($iv_base64);
    $encrypted_message = base64_decode($encrypted_message_base64);
    if ($iv === false || $encrypted_message === false) {
        return "Error: Invalid IV or encrypted message format.";
    }
    $decrypted_message = openssl_decrypt($encrypted_message, $cipher, $shared_key, 0, $iv);
    if ($decrypted_message === false) {
        return "Error: Message decryption failed. OpenSSL Error: " . openssl_error_string();
    }
    return $decrypted_message;
}
```

Real-Time Chat Functionality with AJAX

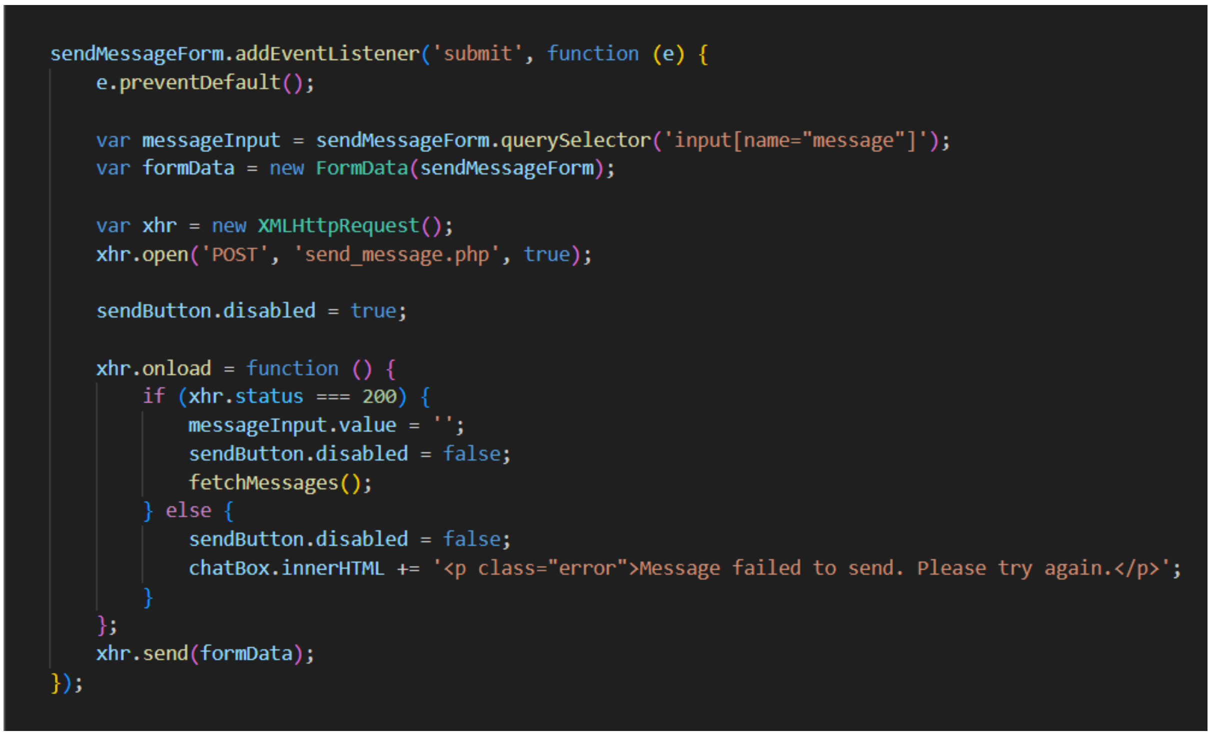
The chat.js file makes the chat work in real time by using AJAX. Instead of reloading the whole page it fetches and sends new messages in the background. Every 5 seconds it checks for new messages and updates the chat box so users see new messages fast. When a user sends a message it is sent without refreshing the page making the chat feel smooth and fast.

This function uses AJAX to request messages from fetch_messages.php for the selected chat user without reloading the page. It sends an asynchronous GET request and upon receiving a successful response (status 200) it updates the chat box with the new messages and scrolls to the bottom.



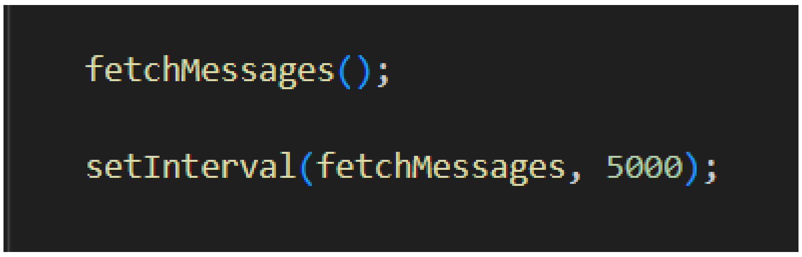
```
function fetchMessages() {
    var xhr = new XMLHttpRequest();
    xhr.open('GET', 'fetch_messages.php?user=' + chatUser, true);
    xhr.onload = function () {
        if (xhr.status === 200) {
            chatBox.innerHTML = xhr.responseText;
            chatBox.scrollTop = chatBox.scrollHeight;
        }
    };
    xhr.send();
}
```

This part handles the message submission process. When the user submits a message e.preventDefault() stops the page from reloading. It retrieves the message input and sends it to send_message.php using AJAX. The submit button is temporarily disabled to prevent duplicate messages. If the message is sent successfully the message box is cleared and messages are reloaded.



```
sendMessageForm.addEventListener('submit', function (e) {
    e.preventDefault();

    var messageInput = sendMessageForm.querySelector('input[name="message"]');
    var formData = new FormData(sendMessageForm);

    var xhr = new XMLHttpRequest();
    xhr.open('POST', 'send_message.php', true);

    sendButton.disabled = true;

    xhr.onload = function () {
        if (xhr.status === 200) {
            messageInput.value = '';
            sendButton.disabled = false;
            fetchMessages();
        } else {
            sendButton.disabled = false;
            chatBox.innerHTML += '<p class="error">Message failed to send. Please try again.</p>';
        }
    };
    xhr.send(formData);
});
```

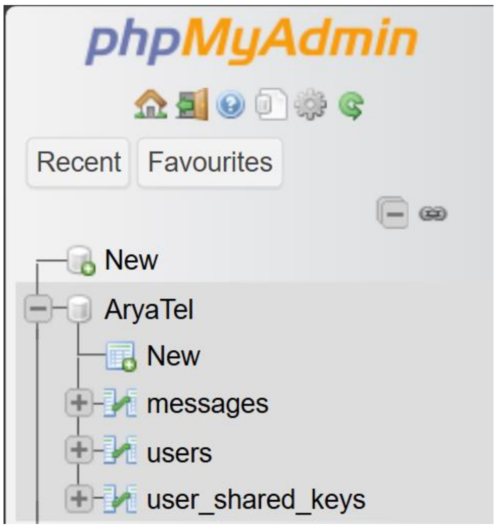
fetchMessages() immediately loads the chat messages when the page first opens. It makes sure users see any previous messages as soon as they open the chat. And set Interval sets an interval to call fetchMessages() every 5 seconds. This is because we want that the chat stays updated in real time without refreshing the page.



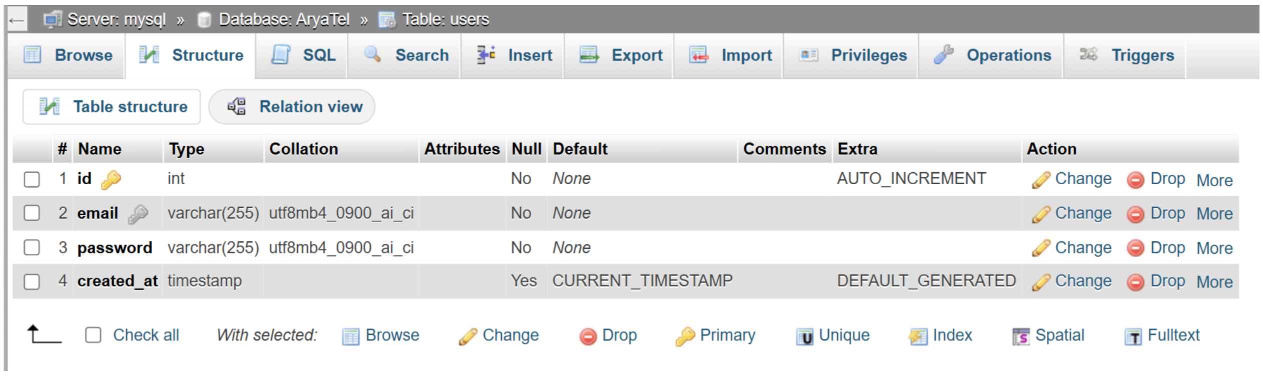
```
fetchMessages();

setInterval(fetchMessages, 5000);
```

Database Design

I used Docker and phpMyAdmin to set up the database for my chat application for security and efficiency. Here's how each table contributes to the security and functionality of the app:

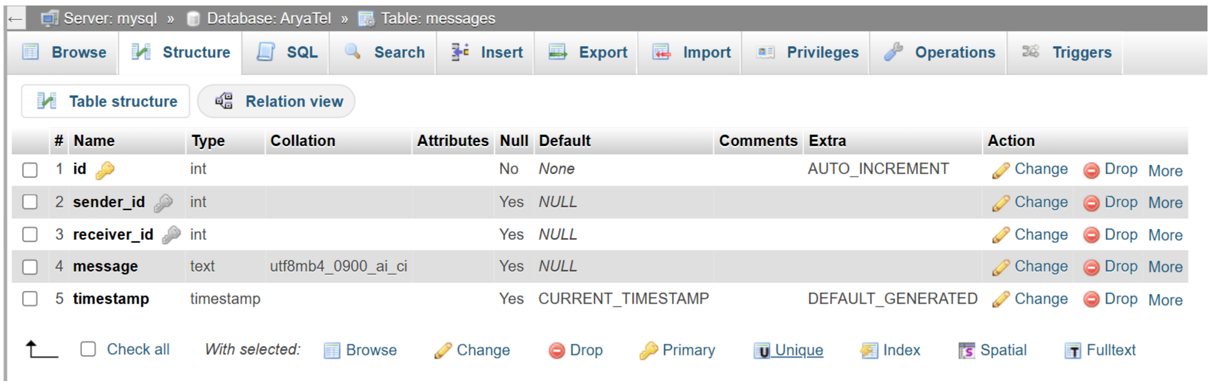


The users table is designed to securely store users data. It contains each user's ID, email, hashed password, and the registration timestamp. By hashing the passwords I ensure that plain text passwords are not stored so even if the database is compromised passwords remain protected.



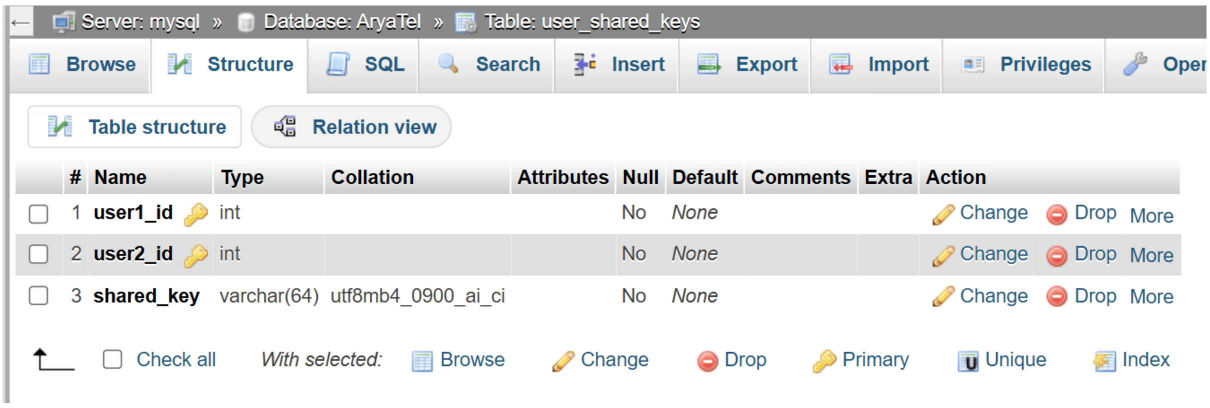
#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	id	int			No	None		AUTO_INCREMENT	Change Drop More
2	email	varchar(255)	utf8mb4_0900_ai_ci		No	None			Change Drop More
3	password	varchar(255)	utf8mb4_0900_ai_ci		No	None			Change Drop More
4	created_at	timestamp			Yes	CURRENT_TIMESTAMP		DEFAULT_GENERATED	Change Drop More

The messages table is where encrypted messages between users are stored. Each message has sender and receiver's IDs, the encrypted message text, and a timestamp for when the message was sent. This table ensures that messages are unreadable without decryption keeping conversations private and secure.


Server: mysql » Database: AryaTel » Table: messages

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	id	int			No	*None*		AUTO_INCREMENT	Change Drop More
2	sender_id	int			Yes	*NULL*			Change Drop More
3	receiver_id	int			Yes	*NULL*			Change Drop More
4	message	text	utf8mb4_0900_ai_ci		Yes	*NULL*			Change Drop More
5	timestamp	timestamp			Yes	CURRENT_TIMESTAMP		DEFAULT_GENERATED	Change Drop More

The user_shared_keys table manages encryption keys for secure message exchanges between users. For each user pair it stores both users' IDs and a unique shared key which is used to encrypt and decrypt messages. This setup makes sure that only the intended participants of a conversation can access the messages maintaining privacy within the chat system.


Server: mysql » Database: AryaTel » Table: user_shared_keys

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	user1_id	int			No	*None*			Change Drop More
2	user2_id	int			No	*None*			Change Drop More
3	shared_key	varchar(64)	utf8mb4_0900_ai_ci		No	*None*			Change Drop More

Together, these tables allow me to securely manage user data, messages, and encryption keys, creating a reliable and private chat platform.

Overview of Security Measures

End-to-End Encryption

Messages are protected so that only the sender and receiver can read them. The server cannot see the actual messages because they are encrypted before being sent and only decrypted on the client side.

Key Generation and Management

Unique encryption keys are created for each pair of users to keep their messages secure. These keys are stored safely in the database. The server handles only the shared key but the actual encryption and decryption process happens on the client side keeping everything private.

Input Sanitization

To protect the database any input from users is cleaned using mysqli_real_escape_string so no harmful SQL code can be used. On the front end, htmlspecialchars is used to make sure displayed messages can't run harmful scripts keeping the chat safe from attacks like XSS.

Password Hashing

Passwords are not stored as plain text. Instead, they are turned into secure, hashed versions using PHP's password_hash function. This makes it very hard for anyone to figure out the actual password, even if the database gets hacked.

These steps work together to make sure the app is safe for users and keeps their data protected.

Conclusion

This project successfully showed a real time chat app that keeps messages private with end to end encryption. Only the sender and receiver can read the messages keeping user privacy safe. Docker made it easy to set up and run the app on different systems. The app includes secure login, encrypted messages, and real time updates using AJAX, making it a safe and simple to use chat platform.

References:

OpenSSL documentation

PHP documentation

MySQL documentation

Java script documentation

phpMyAdmin documentation

Docker documentation