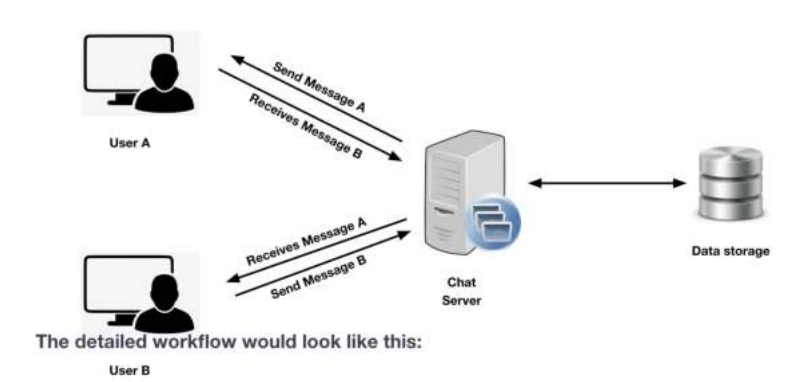
Development of a Simple Messaging App

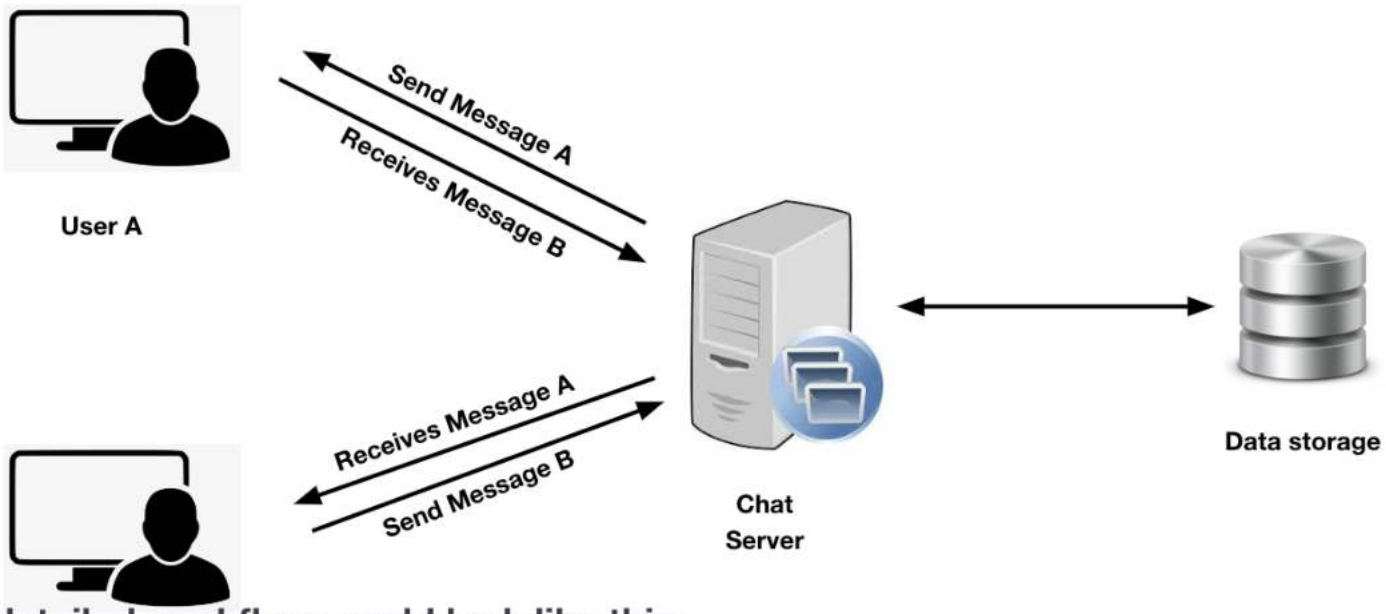


The detailed workflow would look like this:

Chandana Dayapule

Introduction

The objective of this project is to develop a simple messaging app that enables users to send and receive messages to and from other users. The app is being built using Python + Django for the backend and Vue.js for the frontend. It should be functional, intuitive, and include features like user authentication, inbox for received messages, sent box for sent messages, and a compose function for creating and sending new messages. The project will be deployed on an Ubuntu server, and the completed project should be delivered within 2-3 days.



The detailed workflow would look like this:

Stack

The objective of this project is to develop a simple messaging app that enables users to send and receive messages to and from other users. The app is being built using Python + Django for the backend and Vue.js for the frontend. It should be functional, intuitive, and include features like user authentication, inbox for received messages, sent box for sent messages, and a compose function for creating and sending new messages. The project will be deployed on an Ubuntu server, and the completed project should be delivered within 2-3 days.

Backend:

Python + Django (VERSION >= 3.0) will be used to create a REST API-based backend that will handle user authentication, message retrieval, message deletion, and message sending.

Front End

Vue.js will be used to develop the frontend of the messaging app. The UI will be designed to be user-friendly, simple, and intuitive.

Database:

Any suitable database, such as Sqlite or Postgres, will be chosen to store user data and messages.

Additional Package

Any relevant packages, libraries, or tools that aid in the development of the app can be used as necessary.

Tasks

1. Backend Development:

- Set up a **Django project** and create necessary Django apps for user authentication and messaging functionality.
- Implement user authentication mechanisms, such as registration, login, and token-based authentication to allow users to skip login after their initial login.
- Create API endpoints for fetching messages for the inbox and sent box.
- Implement API endpoints to allow users to delete individual messages.
- Create API endpoints for composing and sending new messages.

2. Frontend Development:

- Set up a **Vue.js project** for the frontend.
- Design and develop the login screen, inbox view, sent box view, and compose message view.
- Integrate the frontend with the backend using the API endpoints for message retrieval, deletion, and sending.
- Implement the functionality to save authentication tokens to avoid repeated logins.

3. Deployment

- Ensure the project is compatible with deployment on an **Ubuntu server.**
- Package the frontend and backend appropriately for deployment.
- Include detailed instructions in the README file for setting up the project on the Ubuntu server, including any external packages or dependencies required.

4. Testing and Quality Assurance:

- Thoroughly test the application to ensure all features work as expected.
- Identify and fix any bugs or issues that may arise during testing.
- Review the code to adhere to best coding practices, including commenting and proper code style.

My Solution:

Scalable Architecture Design

1. Problem statement

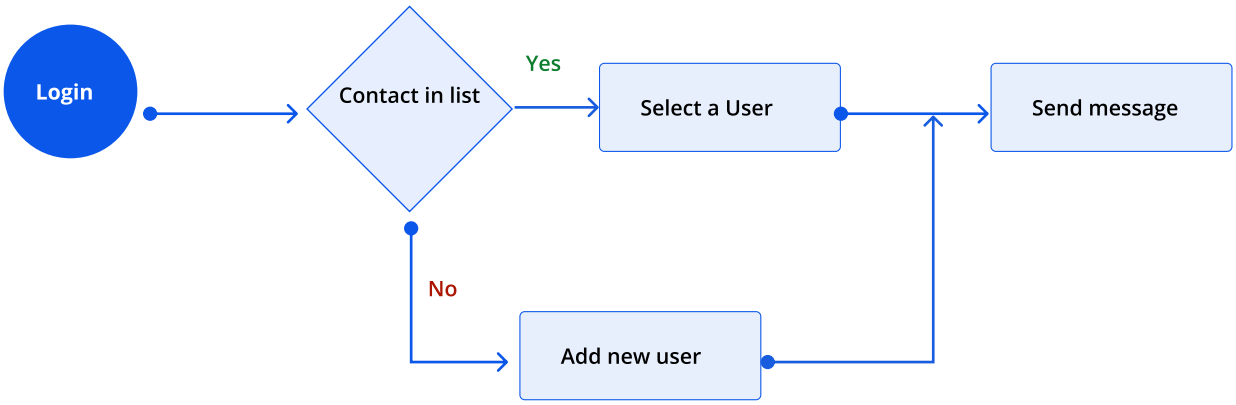
The objective is to build a simple messaging app that enables users to send and receive messages to and from other users. The app should have a login screen, save authentication tokens, list received and sent messages, allow the deletion of individual messages, and provide error messages for invalid requests.

The current architecture should be analyzed, and a scalable architecture should be designed to ensure real-time messaging with low latency, high consistency, and high availability. Future enhancements should also be considered to improve the app's functionality.

The app should include the following features:

- A login screen for user authentication.
- The ability to save authentication tokens to avoid repeated logins.
- An inbox that displays a list of received messages.
- The ability to delete individual messages from the inbox.
- A sent box that displays a list of sent messages.
- The ability to delete individual messages from the sent box.
- A compose function that allows users to create and send new messages.
- Error handling to display appropriate messages for invalid requests.

2. User workflow



Scalable architecture

To achieve real-time chatting with low latency, high consistency, and high availability, the following scalable architecture design is proposed:

1. Load Balancer:

- Introduce a load balancer to distribute incoming requests evenly across multiple chat servers.
- This ensures that no single server is overloaded, optimizing performance and availability.

2. Chat Servers:

- Deploy multiple chat servers to handle incoming messages and real-time communication.
- Each chat server should maintain a WebSocket connection with users to enable real-time message delivery.
- WebSocket protocol allows for bi-directional communication, making it suitable for instant messaging.

3. Database Sharding

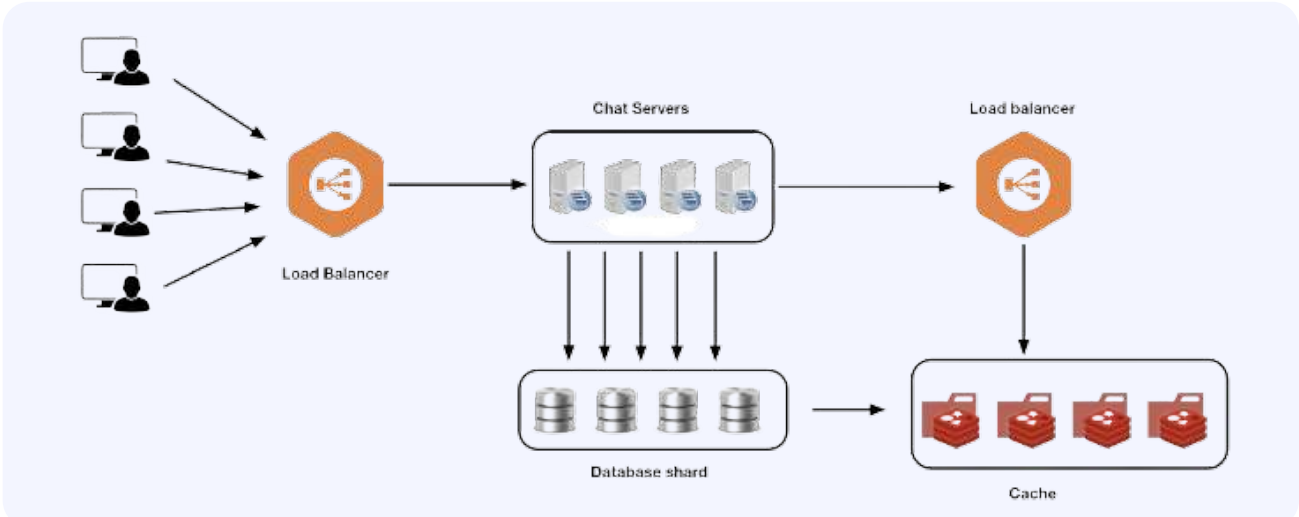
- Implement database sharding to distribute data across multiple database nodes.
- Sharding will prevent any single database server from becoming a bottleneck.
- Use a consistent hashing algorithm to determine which shard should handle each user's messages.
- Ensure that related messages between users are stored on the same shard to maintain high consistency.

4. Load Balancer (Again):

- Introduce a load balancer for database requests to distribute read and write operations across database shards.
- This load balancer ensures that database requests are evenly distributed, optimizing database performance.

5. Cache:

- Implement caching mechanisms to store frequently accessed data in a fast, in-memory cache.
- Cache user information, message history, and other relevant data to reduce database load and minimize latency.
- Consider using caching solutions like Redis or Memcached for efficient data retrieval.

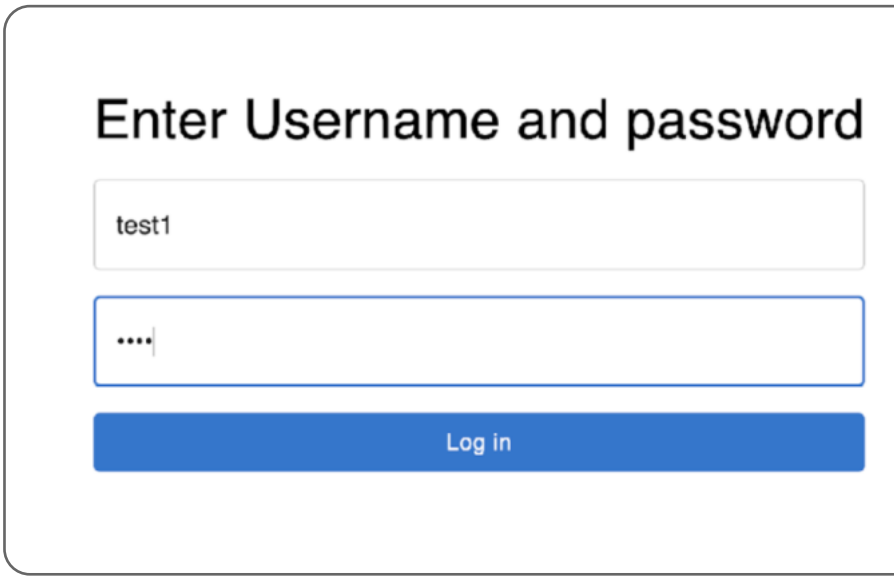


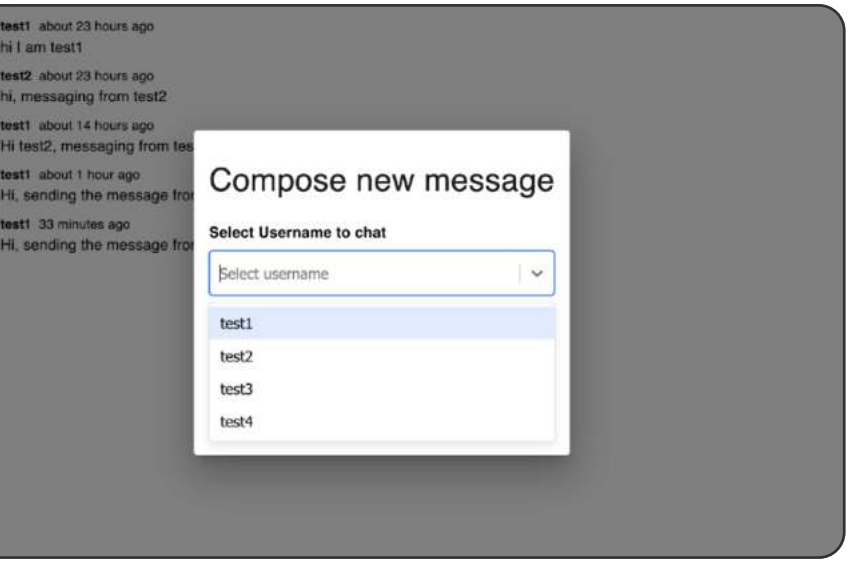
Future Enhancements

To achieve real-time chatting with low latency, high consistency, and high availability, the following scalable architecture design is proposed:

User Login

Allow users to create and participate in group conversations with multiple participants.





Selecting user for composing a new message

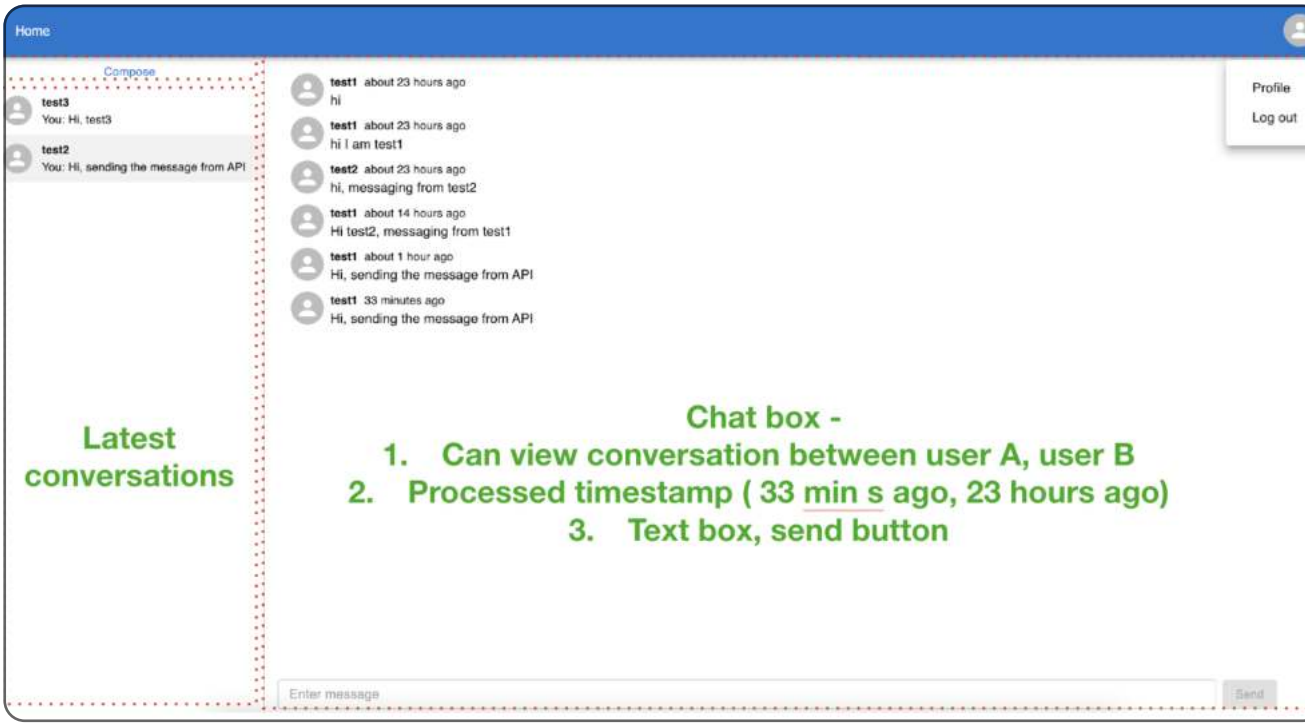
Sending new message to the sert

Future Enhancements

To achieve real-time chatting with low latency, high consistency, and high availability, the following scalable architecture design is proposed:

Message app

Allow users to create and participate in group conversations with multiple participants.



Backend service documentation

1. Service to register an user to app

http://127.0.0.1:8000/api/account/register/

Method - POST

json input

```
{
    "email":"test3@gmail.com",
    "password":"1234",
    "username":"test3"
}
```

Result

```
{
    "username": "test3",
    "email": "test3@gmail.com"
}
```

2. Service to get auth token

http://127.0.0.1:8000/api/auth/token/

Method - POST

json input

```
{
    "username":"test1",
    "password":"1234"
}
```

Result

```
{
    "token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.
eyJ1c2VyX2lkIjoyLCJ1c2VybmFtZSI6InRlc3QxIiwiZXhwIjoxNjYwMjc4Mjg4LCJlbWFp
bCI6InRlc3QxQGdtYWlsLmNvbSJ9.
35XKrOaxMjGCstnBQWLftXF_jgormRQOpn_2BX6CvRs"
}
```

3. Service to verify if a token is expired or not

http://127.0.0.1:8000/api/token-verify/

Method - POST

json input

```
{
    "token":"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.
eyJ1c2VyX2lkIjoyLCJ1c2VybmFtZSI6InRlc3QxIiwiZXhwIjoxNjYwMjc4Mjg4LCJlbWFp
bCI6InRlc3QxQGdtYWlsLmNvbSJ9.
35XKrOaxMjGCstnBQWLftXF_jgormRQOpn_2BX6CvRs"
}
```

Result

```
{
    "token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.
eyJ1c2VyX2lkIjoyLCJ1c2VybmFtZSI6InRlc3QxIiwiZXhwIjoxNjYwMjc4Mjg4LCJlbWFp
bCI6InRlc3QxQGdtYWlsLmNvbSJ9.
35XKrOaxMjGCstnBQWLftXF_jgormRQOpn_2BX6CvRs"
}
```

4. Service to get all accounts associated with app (for starting a conversation)

http://127.0.0.1:8000/api/account/

-H Authorization : jwt <token>

Method - GET

Result-

```
[
    {
        "id": 1,
        "username": "test",
        "email": "test@gmail.com",
        "first_name": "",
        "last_name": ""
    },
    {
        "id": 2,
        "username": "test1",
        "email": "test1@gmail.com",
        "first_name": "",
        "last_name": ""
    }
]
```

5. Get conversation between two users (sender, receiver)

http://127.0.0.1:8000/api/conversations/

-H Authorization : jwt <token>

params - `msg_sender`=<username> & `msg_receiver`= <username>

Method - GET

Result -


```
[
        {
                "id": 5,
                "msg_sender_data": {
                        "id": 1,
                        "username": "test1",
                        "first_name": "",
                        "last_name": "",
                        "email": "test3@gmail.com"
                },
                "msg_receiver_data": {
                        "id": 2,
                        "username": "test2",
                        "first_name": "",
                        "last_name": "",
                        "email": "test2@gmail.com"
                },
                "message": "Hi test2, messaging from test1",
                "image": "",
                "file": "",
                "conversation_id": "1660535258988",
                "conversation_subject": "",
                "archived": false,
                "publish": "2022-08-15",
                "timestamp": "2022-08-15T03:47:38.999542",
                "updated": "2022-08-15T03:47:38.999586"
        },
        {
                "id": 4,
                "msg_sender_data": {
                        "id": 2,
                        "username": "test2",
                        "first_name": "",
                        "last_name": "",
                        "email": "test2@gmail.com"
                },
                "msg_receiver_data": {
                        "id": 1,
                        "username": "test1",
                        "first_name": "",
```

```
                "last_name": "",
                "email": "test3@gmail.com"
        },
        "message": "hi, messaging from test2",
        "image": "",
        "file": "",
        "conversation_id": "1660501948405",
        "conversation_subject": "",
        "archived": false,
        "publish": "2022-08-14",
        "timestamp": "2022-08-14T18:32:28.408969",
        "updated": "2022-08-14T18:32:28.408978"
},
{
        "id": 2,
        "msg_sender_data": {
                "id": 1,
                "username": "test1",
                "first_name": "",
                "last_name": "",
                "email": "test3@gmail.com"
        },
        "msg_receiver_data": {
                "id": 2,
                "username": "test2",
                "first_name": "",
                "last_name": "",
                "email": "test2@gmail.com"
        },
        "message": "hi I am test1",
        "image": "",
        "file": "",
        "conversation_id": "1660501900544",
        "conversation_subject": "",
        "archived": false,
        "publish": "2022-08-14",
        "timestamp": "2022-08-14T18:31:40.548393",
        "updated": "2022-08-14T18:31:40.548404"
},
{
        "id": 1,
        "msg_sender_data": {
                "id": 1,
                "username": "test1",
                "first_name": "",
                "last_name": "",
                "email": "test3@gmail.com"
        },
        "msg_receiver_data": {
                "id": 2,
```

```
                        "username": "test2",
                        "first_name": "",
                        "last_name": "",
                        "email": "test2@gmail.com"
                },
                "message": "hi",
                "image": "",
                "file": "",
                "conversation_id": "1660501893268",
                "conversation_subject": "",
                "archived": false,
                "publish": "2022-08-14",
                "timestamp": "2022-08-14T18:31:33.273469",
                "updated": "2022-08-14T18:31:33.273477"
        }
]
```


6. Creating a conversation between User A, User B

http://127.0.0.1:8000/api/conversation/create

-H Authorization Jwt <token>

Method - POST

json input -

```
{
    "msg_sender":"test1",
    "msg_receiver":"test2",
    "message":"Hi, sending the message from API",
    "conversation_subject":""
}
```

Result -


```
{
        "msg_sender": 1,
        "msg_receiver": 2,
        "message": "Hi, sending the message from API",
        "image": null,
        "conversation_id": "1660581449378"
}
```


7. Service to delete a conversation

127.0.0.1:8000/api/conversation/delete/<conversation_id>/

-H Authorization Jwt <token>

Param username <username>

Method - DELETE

Result -

```
{
    "success": "successfully deleted message"
}
```

8. Service to get latest conversations (to display in left panel of dashboard)

127.0.0.1:8000/api/last_conversation

-H Authorization Jwt <token>

Param username <username>

Method - GET

Result -

```
{
        "status": "success",
        "result": [
                {
                        "id": 2,
                        "username": "test2",
                        "email": "test2@gmail.com",
                        "first_name": "",
                        "last_name": "",
                        "last_message": "Hi, sending the message from
API",
                        "timestamp": "2022-08-15T16:37:29.384853",
                        "read": false,
                        "delivered": false,
                        "msg_sender": "test1",
                        "msg_receiver": "test2"
                }
        ]
}
```