***Great job on completing the MySQL setup! Now let’s proceed with configuring and running the Django chat application from the provided GitHub repository. Here’s a step-by-step guide to get your chat application up and running on \*\*VM3\*\*.***

**### Step 1: SSH into Your Private VM (VM3)**

1. \*\*SSH into VM3\*\*:

```bash

ssh <username>@<private\_vm\_private\_ip>

```

### Step 2: Install Required Dependencies

2. \*\*Update the Package List\*\*:

```bash

sudo apt update

```

3. \*\*Install Python and pip\*\*:

- Ensure you have Python and pip installed.

```bash

sudo apt install python3 python3-pip -y

```

4. \*\*Install Git\*\*:

- If Git is not installed, install it.

```bash

sudo apt install git -y

```

5. \*\*Install Django\*\*:

- Install Django and other dependencies from the requirements file later.

```bash

pip3 install Django

```

**### Step 3: Clone the Chat Application Repository**

6. \*\*Clone the Repository\*\*:

```bash

git clone https://github.com/peyyala7hills/new\_chatapp.git

```

7. \*\*Navigate to the Project Directory\*\*:

```bash

cd new\_chatapp

```

**### Step 4: Install Project Dependencies**

8. \*\*Install Required Python Packages\*\*:

- If there's a `requirements.txt` file in the repository, install the dependencies listed there.

```bash

pip3 install -r requirements.txt

```

**### Step 5: Configure Django Settings**

9. \*\*Edit the `settings.py` file\*\*:

- Open the `settings.py` file in the project (usually located in the `new\_chatapp` directory):

```bash

Sudo vim /new\_chatapp/fundoo/fundoo/settings.py

```

- Update the \*\*database settings\*\* to connect to the MySQL database you created earlier:

```python

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.mysql',

'NAME': 'my\_django\_db', # Your database name

'USER': 'django\_user', # Your database username

'PASSWORD': 'strong\_password', # Your database password

'HOST': '<private\_vm\_private\_ip>', # IP of VM3

'PORT': '3306',

}

}

```

10. \*\*Add Allowed Hosts\*\*:

- In the same `settings.py` file, add your \*\*allowed hosts\*\*:

```python

ALLOWED\_HOSTS = ['\*'] # Use '\*' for testing, change to specific IPs or domains in production

```

**### Step 6: Apply Database Migrations**

**Install:**

**pip install djangorestframework**

**pip install django-rest-auth**

11. \*\*Run Migrations\*\*:

- Apply the migrations to set up the database schema:

```bash

python3 manage.py migrate

```

**### Step 7: Run the Django Application**

13. \*\*Run the Development Server\*\*:

- Start the Django development server:

```bash

python3 manage.py runserver 0.0.0.0:8000

```

**0.0.0.0 means the server will listen on all available network interfaces. This allows your Django application to be accessible by any IP address assigned to the server (both public and private).**

**### Step 8: Access the Application**

14. \*\*Access the Chat Application\*\*:

- Open a web browser and navigate to:

```

http://<private\_vm\_private\_ip>:8000

```

- If you created a superuser, you can access the admin panel at:

```

http://<private\_vm\_private\_ip>:8000/admin

```

***:-*** Additional Notes:

- Ensure that your \*\*Network Security Group (NSG)\*\* allows traffic on port \*\*8000\*\* if you want to access it externally.

- Since you are in a private network, if you want to access it directly, you may need to SSH into a public VM and then use SSH tunneling or a similar method to access it.

**Avoiding hard coding for security purpose:**

Using environment variables for your database configuration in Django (or any application) is a best practice for several reasons:

### 1. \*\*Security\*\*

- \*\*Sensitive Information\*\*: Database credentials (like username, password, and database name) are sensitive information. By using environment variables, you can avoid hardcoding them into your source code, which reduces the risk of accidentally exposing them in version control systems like Git.

- \*\*Access Control\*\*: Environment variables can be restricted by the operating system, ensuring that only authorized users or processes can access sensitive information.

### 2. \*\*Flexibility\*\*

- \*\*Different Environments\*\*: Environment variables allow you to easily switch configurations between different environments (development, testing, production) without changing the code. For example, you can have different database settings in your local environment compared to your production environment without modifying your Django settings file.

- \*\*Deployment\*\*: When deploying applications to cloud environments or containers (like Docker), you often have to set up different configurations for different environments. Environment variables make it easier to manage these configurations.

### 3. \*\*Ease of Use\*\*

- \*\*Configuration Management\*\*: Using environment variables centralizes configuration management. It simplifies updates and changes to configurations without needing to edit the source code.

- \*\*Readability\*\*: The code remains cleaner and more readable without clutter from configuration settings. You can easily identify which values are configurable and where they come from.

### 4. \*\*Best Practices\*\*

- \*\*Common Practice\*\*: Using environment variables for sensitive settings is a widely accepted best practice in software development. It promotes better security and maintainability.

### Example

In your Django settings:

```python

import os

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.mysql',

'NAME': os.getenv('DB\_NAME', ''),

'USER': os.getenv('DB\_USER', ''),

'PASSWORD': os.getenv('DB\_PASSWORD', ''),

'HOST': os.getenv('DB\_HOST', ''),

'PORT': os.getenv('DB\_PORT', '3306'), # Default port for MySQL

}

}

```

- \*\*os.getenv()\*\*: This function retrieves the value of the environment variable. If the variable doesn't exist, it returns an empty string (or a default value you specify, like `'3306'` for the port).

- \*\*Environment Variables\*\*: You would set these variables in your environment or in a `.env` file (if using a library like `python-decouple` or `django-environ`).

### Setting Environment Variables

You can set environment variables in your shell, or if you're using a web server, you can set them in your server configuration. For example, in a terminal, you can run:

```bash

export DB\_NAME='my\_django\_db'

export DB\_USER='django\_user'

export DB\_PASSWORD='my\_secure\_password'

export DB\_HOST='10.0.3.4' # IP of VM3

export DB\_PORT='3306'

```

With this approach, your database configuration is more secure and adaptable across different environments.

***To set environment variables for your Django application, you can follow these steps to ensure they are properly exported and available to your application. Here’s how you can do it:***

### 1. Exporting Environment Variables

You can export the environment variables either in your terminal session or in a shell configuration file for persistent use.

#### Option 1: Temporary Export in Terminal Session

You can directly run the export commands in your terminal session. However, these will only last for the duration of the terminal session.

1. \*\*Open your terminal\*\*.

2. \*\*Run the export commands\*\*:

```bash

export DB\_NAME='my\_django\_db'

export DB\_USER='django\_user'

export DB\_PASSWORD='Himanshu@2001' # Use a more secure password in production

export DB\_HOST='10.0.3.4' # IP of your MySQL server

export DB\_PORT='3306' # Default MySQL port

```

3. \*\*Verify that the variables are set\*\*:

You can check if the variables are set correctly by running:

```bash

echo $DB\_NAME

echo $DB\_USER

echo $DB\_PASSWORD

echo $DB\_HOST

echo $DB\_PORT

```

#### Option 2: Permanent Export in a Shell Configuration File

To make these environment variables persistent (i.e., they will remain set even after closing the terminal), you can add them to your shell's configuration file. This is usually `~/.bashrc` or `~/.bash\_profile` for Bash users or `~/.zshrc` for Zsh users.

1. \*\*Open the configuration file\*\* in your preferred text editor (e.g., `nano`, `vi`, etc.):

For Bash, use:

```bash

nano ~/.bashrc

```

or

```bash

vi ~/.bashrc

```

For Zsh, use:

```bash

nano ~/.zshrc

```

2. \*\*Add the export commands\*\* to the end of the file:

```bash

export DB\_NAME='my\_django\_db'

export DB\_USER='django\_user'

export DB\_PASSWORD='Himanshu@2001' # Use a more secure password in production

export DB\_HOST='10.0.3.4' # IP of your MySQL server

export DB\_PORT='3306' # Default MySQL port

```

3. \*\*Save and close the editor\*\*.

4. \*\*Apply the changes\*\* by running the following command:

```bash

source ~/.bashrc

```

or

```bash

source ~/.zshrc

```

***Creating a virtual environment (venv) for your Django project inside the `new\_chatapp` directory and configuring it with the necessary installations is a straightforward process. Here are the detailed steps to do this:***

### \*\*Steps to Create and Configure a Virtual Environment in `new\_chatapp`\*\*

#### \*\*1. Install Python and pip\*\*

Make sure Python and pip are installed on your system. You can check their installation with the following commands:

```bash

python3 --version

pip3 --version

```

If they are not installed, you can install Python and pip using:

```bash

sudo apt update

sudo apt install python3 python3-pip python3-venv

```

#### \*\*2. Navigate to Your Project Directory\*\*

Change your working directory to where your Django project (`new\_chatapp`) is located:

```bash

cd /new\_chatapp

```

#### \*\*3. Create a Virtual Environment\*\*

Use the following command to create a virtual environment named `venv` within the `new\_chatapp` directory:

```bash

python3 -m venv venv

```

This command creates a new directory called `venv` inside `new\_chatapp`, containing the virtual environment.

#### \*\*4. Activate the Virtual Environment\*\*

Before installing any packages, you need to activate the virtual environment:

```bash

source venv/bin/activate

```

After activation, your shell prompt will change to indicate that you are now working inside the virtual environment.

#### \*\*5. Upgrade pip (Optional)\*\*

It’s a good practice to ensure that `pip` is up to date. You can upgrade it using:

```bash

pip install --upgrade pip

```

#### \*\*6. Install Django and Other Required Packages\*\*

Now you can install Django and any other necessary packages, such as MySQL client, using pip. For example:

```bash

pip install django mysqlclient

```

If you have a `requirements.txt` file, you can install all dependencies listed there using:

```bash

pip install -r requirements.txt

```

### \*\*7. Install Gunicorn\*\*

Gunicorn is a Python WSGI HTTP server for UNIX that will serve your Django application. You can install it within your virtual environment using pip:

```bash

pip install gunicorn

```

### \*\*8. Create a systemd Service File for Your Django Application\*\*

You’ll need to create a systemd service file to manage your Django application as a service. Follow these steps:

1. \*\*Open a Terminal\*\* and create a new service file:

```bash

sudo nano /etc/systemd/system/django\_app.service

```

2. \*\*Add the following configuration to the service file\*\* (adjust paths and settings as necessary):

```ini

[Unit]

Description=Django Application Service

After=network.target

[Service]

User=sysadmin

Group=sysadmin

WorkingDirectory=/new\_chatapp/fundoo

ExecStart=/new\_chatapp/venv/bin/gunicorn --bind 0.0.0.0:8000 fundoo.wsgi:application

Restart=always

# Specify environment variables directly

Environment="DB\_NAME=my\_django\_db"

Environment="DB\_USER=django\_user"

Environment="DB\_PASSWORD=Himanshu@2001"

Environment="DB\_HOST=10.0.3.4"

Environment="DB\_PORT=3306"

[Install]

WantedBy=multi-user.target

```

- \*\*`User` and `Group`\*\*: Replace `sysadmin` with the actual username under which you want the service to run.

- \*\*`WorkingDirectory`\*\*: Path to your Django project.

- \*\*`ExecStart`\*\*: The command to start Gunicorn with your Django application. Adjust `fundoo.wsgi:application` to match your project structure if necessary.

- \*\*Environment variables\*\*: Include your database credentials as needed.

3. \*\*Save and exit\*\* the editor (in nano, you can do this by pressing `CTRL + X`, then `Y`, and `ENTER`).

### \*\*9. Reload systemd and Start the Service\*\*

After creating the service file, you need to reload the systemd daemon to recognize the new service and then start the service.

1. \*\*Reload systemd\*\*:

```bash

sudo systemctl daemon-reload

```

2. \*\*Start the Django application service\*\*:

```bash

sudo systemctl start django\_app.service

```

3. \*\*Enable the service to start on boot\*\*:

```bash

sudo systemctl enable django\_app.service

```

### \*\*10. Check the Status of Your Service\*\*

To verify that your service is running correctly, you can check its status:

```bash

sudo systemctl status django\_app.service

```

You should see output indicating that the service is active and running.

### \*\*11. Allow the Port Through the Firewall\*\*

If you have a firewall enabled (like UFW), make sure to allow traffic on port 8000:

```bash

sudo ufw allow 8000

```

### \*\*12. Access Your Application\*\*

Now that your Django application is running as a service, you can access it by navigating to `http://<your\_server\_ip>:8000` in your web browser.

### \*\*Conclusion\*\*

You have successfully set up Gunicorn to serve your Django application and configured a systemd service to run it as a daemon. Your application will now start automatically on system boot, and you can manage it with systemd commands.

If you have any further questions or need more assistance, feel free to ask!