PIECY PROJECT LOGBOOK

This document traces steps we made to set up our project Piecy.

Piecy is a full stack web application for managing an automotive spare parts business or any similar entity.

# Rocky Linux installation and setup

1. Install Rocky Linux 10.
2. Set up root password: azrou@2Uorza
3. Create a regular user piecer and set its password: fershi@2Ihsref
4. Configure remote access over SSH using key pairs.
5. Disable SSH access with passwords.
6. Update system and check that python version 3.12.9 is installed.
7. Add an alias python = “python3” to get rid of the ugly 3 once for all.

# Django configuration

1. Create a folder /var/opt/django and change its ownership to piecer.
2. Optional: Create a repository on Github to keep track of your code.
3. Create a python virtual environment and source from it.
4. Install Django. The current Django version is 5.2.4.
5. Create a new django project piecy.
6. Open port 8000/tcp for testing in development.
7. Test if the rocket is ready to go to space.

# Postgresql installation and config

1. Installed postgresql-server using:

*→ dnf install postgresql-server*

1. Init a database:

*→ postgresql-setup --initdb*

*\* Initializing database in '/var/lib/pgsql/data'*

*\* Initialized, logs are in /var/lib/pgsql/initdb\_postgresql.log*

1. Set a strong password for user postgres:

*→ passwd postgres*

*New password: arich@2Hcira*

*Retype new password: arich@2Hcira*

*passwd: password updated successfully*

1. Create a regular role to use for db connections:

*→ createuser --pwprompt dbaser*

*Enter password for new role: dbaser@2Resabd*

*Enter it again: dbaser@2Resabd*

1. Create a database for the user:

*→ createdb -O dbaser dbieces*

1. Manage users and accesses:

*→ nano /var/lib/pgsql/data/pg\_hba.conf*

*# TYPE DATABASE USER ADDRESS METHOD*

…

*local dbieces dbaser md5*

*local all postgres trust*

*host dbieces dbaser 127.0.0.1/32 md5*

*host dbieces dbaser ::1/128 md5*

*local replication all peer*

*host replication all 127.0.0.1/32 ident*

*host replication all ::1/128 ident*

…

1. Add some security hardenings:

*→ nano /var/lib/pgsql/data/postgresql.conf*

*…*

*listen\_addresses = 'localhost'*

*port = 49161*

*…*

1. Restart postgresql service and run some tests:

*→ systemctl restart postgresql*

On a local terminal:

*→ psql -p 49161 -U postgres -d dbieces*

This should ask for dbaser password and if valid enter a psql promp.

To test that things are set up correctly, allow port 49161 on firewall:

*→ firewall-cmd --add-port=49161/tcp*

*success*

From a remote host on the network (192.168.10.1, for example), test using:

*→ psql -p 49161 -h 192.168.10.216 -U postgres -d dbieces*

*psql: error: connection to server at "192.168.10.216", port 49161 failed: Connection refused*

*Is the server running on that host and accepting TCP/IP connections?*

Do the same with our standard user dbaser:

*psql -p 49161 -h 192.168.10.216 -U dbaser -d dbieces*

*psql: error: connection to server at "192.168.10.216", port 49161 failed: Connection refused*

*Is the server running on that host and accepting TCP/IP connections?*

Error “Connection refused” shows that our configuration worked fine.

Do not forget to remove the firewall rule for postgresql port, unless otherwise needed.

1. Configure django database backend:

Update .env file in our project directory (/var/opt/django/piecy) with the new database data:

# Database server configuration.

DB\_ENGINE = "django.db.backends.postgresql"

DB\_NAME = "dbieces"

DB\_USER = "dbaser"

DB\_PASSWORD = "dbaser@2Resabd"

DB\_HOST = "localhost"

DB\_PORT = "49161"

Update the settings.py file to use postgresql instead of default sqlite:

DATABASES = {

"default": {

"ENGINE": DB\_ENGINE,

"HOST": DB\_HOST,

"PORT": DB\_PORT,

"NAME": DB\_NAME,

"USER": DB\_USER,

"PASSWORD": DB\_PASSWORD,

}

}

Do not forget to install psycopg2-binary to the project virtual environment for Postgresql to work correctly.

1. Handcraft an automatic backup system:
   1. Update replication privilege in pg\_hba.conf:

*→ nano /var/lib/pgsql/data/pg\_hba.conf*

…

# replication privilege.

#local replication all md5

local replication postgres trust

#host replication all 127.0.0.1/32 md5 #ident

#host replication all ::1/128 md5 #ident

…

* 1. Create a script to be executed regularly:

*→ nano /var/opt/django/dback.sh*

#!/bin/bash

# Config

BACKUP\_DIR="/var/backups/pgsql"

DATE=$(date +%F\_%H-%M-%S)

BACKUP\_NAME="PG\_BKUP\_$DATE"

PG\_USER="postgres"

# Ensure backup directory exists

mkdir -p "$BACKUP\_DIR"

# Create backup

pg\_basebackup -U $PG\_USER -p 49161 -D "$BACKUP\_DIR/$BACKUP\_NAME" -F tar -z -P

# Check success

if [ $? -eq 0 ]; then

echo "---------- Backup created successfully ----------"

echo "================== Backup file =================="

echo " $BACKUP\_DIR/$BACKUP\_NAME "

echo "================================================="

else

echo "XXXXXXXXXX pg\_basebackup failed! XXXXXXXXXX"

fi

* 1. Setup a cron job to run every day at 07:07:

*→ export EDITOR=”/usr/bin/nano”*

*→ crontab -e*

# Database backup cron job:

CRON\_TZ = "Africa/Casablanca"

7 7 \* \* \* bash /var/opt/django/dback.sh

* 1. Check configuration

Make sure crond.service is enabled:

*→ systemctl status crond*

crond.service - Command Scheduler

Loaded: loaded (/usr/lib/systemd/system/crond.service; enabled; preset: enabled)

Active: active (running) since Mon 2025-07-14 11:44:22 CDT; 3h 52min ago

Invocation: 63925c24344942dc9f088ac0e22c3683

…

1. Test **restoration** procedure: **Coming next**.
2. Rotate backups

Backups will certainly pile up over time and fill up the storage. To avoid this, update the update script to remove backups older than a given age like 30 or 90 days.

*→ nano /var/opt/django/dback.sh*

# Safety check: Directory must exist

if [ ! -d "$BACKUP\_DIR" ]; then

echo "Error: Directory '$BACKUP\_DIR' does not exist." >&2

exit 1

fi

# Find and delete matching folders older than BAKUP\_DAYS

find "$BACKUP\_DIR" -maxdepth 1 -type d -name "$BKUP\_PREFIX" -mtime +"$BAKUP\_DAYS" | while read >

folder\_name=$(basename "$folder")

if rm -rf "$folder"; then

echo "----- Deleted old folder $folder\_name"

else

echo "xxxxx Failed to delete folder $folder\_name" >&2

fi

done

# Multi-tenancy ?

Multi-tenancy is one of the key aspects of any modern SaaS service.

Django out of the box does not provide any support for multi-tenancy.

There is a package called django-tenants that gives this possibility, but it seems too long to follow and somehow complicated and not guaranteed to work with no flaws.

App level multitenancy is the only remaining option for us.

One database, one schema.

Each record should have an owner field …

# Database models

We drafted our first data models and relationships on paper. It was so complicated and growing fast with new ideas and new needs.

We thought about an easy way to put our ideas into django models. It is tedious to type everything by hand and a lot of things are repeated. We found pgModeler. I installed it on my Arch Linux desktop and struggled to figure out how things work. We finally found it was easy and very helpful in visualizing the whole database and tables’ relationships…

It threw some warnings and errors from time to time, but nothing serious. I thought it was serious until I saved my work and found that a great part of it was lost. Never again, I told myself. I added the file I was working on to our django directory and committed changes to github.

## Bridge to Django

The database is visually good in pgModeler and is easy to understand and edit. Although pgModeler offers an easy way to connect to PostgreSQL server and write the database to it, we opted to go through local shell and Django.

* Export database model from pgModeler to a *models.sql* file.
* Edit the file to remove database creation (already done) and change owner role to dbaser.
* Write the fileto the database from a command shell as *piecer* user:

*→ psql -p 49161 -U dbaser -d dbieces -f base/models.sql*

This showed some errors in our sql file, especially default values like *0* for boolean instead of *false* or some strings … Correct any error and retry until it passes.

* From our virtual environment, read the database tables and write them to the django models file (pay attention as this will overwrite anything in base/models.py):

*→ source .venv/bin/activate*

*→ python manage.py inspectdb > base/models.py*

If the operation succeeds, no output is printed.

* Refer to the beginning of the created *base/models.py* file to make Django manage the database (including migrations).
* Remove tables starting with Auth…(models.Model) and Django…(models.Model) as they are already handled elsewhere by Django.
* To make sure Django builds database from scratch and ensure max compliance, remove the database *dbieces* and use django to recreate it (makemigrations and migrate).