# 2024-03-05 Lifeapi has to close

**WHY**

This is a project that provides daily usage, but does not really provide a value.

You kind of know if you are reading or if you are sick or if you had belly ache a few days in a row or something like this.

But you log this stuff into the app. It is something you have to do on and on, day by day…

And also fetching data from external resources.. Now each of my machine has to have rescuetime installed, we must wear certain watches.. That is too much dependency. Those things might change and look, we no longer have the data and the visualizations that I might build would break.

Additional cost for electricity for magic mirror (if I would have one)

Additional costs running the GCP instance.

I have proved myself that I can do it, I have proved that it's possible, now time to move to another project.

Kind of better to just live life rather than trying to optimize it, like that quote.

**THINGS LEARNED**

* + How to fetch from an API
  + Cronjobs
  + GCP hosting
  + Make it simple from the beginning for the user to use the app(should have added email reminders or smth, or create an app)
  + Magicmirror attitude, built having this in mind
  + How to work with backups

**STEPS**

To Do Make lifeapi repo private

To Do Stop the cronjobs

To Do Close gcp instance, cancel

To Do Remove card payment

# Lifeapi - CLOSED 2024-03-05

First I think I should learn CBV's in django,

Then learn ajax

Learn more JS

Learn jquery

Can learn all these by building an app for shelters

Only then come to lifeapi and rebuild it

More tasks in github project

Gaila negalima is garmin paimti dalyku

Ir sekimą. Kaip sekėsi šią savaitę man ar tau. Kitaip pamiršti sekti tuos metrics KRC. Ir nusivaziuoji

Nauji questions:

10k žingsnių 20min meditacija calisthenics kazka mokykis

Kiek % darai ka nori kiek ka reikia

Kaip activw/inactive. Dashboarde useris gali pasirinkti

REST framework api, front su htmx, phone app su kuo Karina sakė dirba

lifeapi kai turbinsi(20 most useful packages)

<https://cur.at/VUEAr0X?m=email&sid=G8KPNxH>

kad galetum is karto ten uzsirasyti ir nereiketu perrasineti ir galetum random quote sau susigeneruoti ir parodyti kitiems

# How to’s

## lifeapi ip change procedure

When the server is turned off and then turned on again - the IP of it changes.

might have to fix the ssh keys [[[file:~/GIT/devnotes/20240104T070918--ssh-into-gcp-server\_\_gcp\_ssh.org][like](file:///\\~\GIT\devnotes\20240104T070918--ssh-into-gcp-server__gcp_ssh.org%5d%5blike) mentioned here]]

Then will have to login to the server over putty

change the IP in nginx config file

change the allowed hosts file in =.env\_prod= file

[[[file:20240104T082011--restarting-lifeapi-services\_\_lifeapi.org][restart](file:///\\20240104T082011--restarting-lifeapi-services__lifeapi.org][restart) the services]]

login with username and pass, if can't find - check previous ip:

first - 34.88.9.236

second - 34.88.54.183

[2024-01-28 Sun] - 35.228.228.106

or go to "pasiukai" file and find the credentials here : "# lifeapi app"

show Julyte how to use it, the new ip.

## dockerizing lifeapi app

\* Intro

Dates in data\_table were messed up. Installed updates on the server. Upgrade

got stuck on gcp-cli package. Attempted to fix it unsuccessfully. Restarted the

server - lifeapi does not start any longer. It's [2024-01-09 Tue] and lifeapi

still does not work. Not cool. Thought it would be smart to dockerize the app

so it would be simplier to launch it again in the future. Otherwise now I would

have to do all the installation steps again.

\* Inspiration

<https://www.youtube.com/watch?v=_ZMGp-bSQhg&ab_channel=ScalableScripts>

\* Try to do the installation steps for a new gcp instance

First try this, then dockerize in the free time maybe.

\* Creating a separate branch for this

Version of the Master branch before the changes -

<https://github.com/arvydasg/lifeapi/tree/c4ab5020140565ced6a62b167e5cdeadc1d07030>

Dockerzing branch was created -

<https://github.com/arvydasg/lifeapi/tree/42266166e2f90a05583e796df3ed9d2798b73431>

\* Creating a test docker container and running it locally

#+begin\_src bash

FROM python:3.8-slim-buster

WORKDIR /app

COPY requirements.txt requirements.txt

RUN pip3 install -r requirements.txt

COPY . .

n

CMD ["python3", "manage.py", "runserver", "0.0.0.0:8000"]

#+end\_src

#+begin\_src bash

docker build -t django-docker-starter .

docker run -p 8000:8000 django-docker-starter

#+end\_src

\* Create a test docker container and run it in GCP over gcp shell

<https://www.youtube.com/watch?v=H0kQL_KHt3o&ab_channel=TechTrapture>

\* Run the locally created container in GCP

- Enable =Artifact Registry API= service. Sorry, not this, but =Google

Container Registry API=

- Okay krc.

#+begin\_quote

Container Registry is deprecated. After May 15, 2024. Artifact Registry

will host images for the gcr.io domain in projects without previous

Container Registry usage. If you use Container Registry, learn about the

deprecation. To get started with managing containers on Google Cloud, use

Artifact Registry.

Starting January 8, 2024(today is [2024-01-09 Tue]), if your organization

has not previously used Container Registry, new gcr.io repositories will

be hosted on Artifact Registry by default. For more information on this

change, see gcr.io hosted on Artifact Registry.

#+end\_quote

- So YEAH, sorry, Enable =Artifact Registry API= service and not =Google

Container Registry API= :)

- Download and install google cloud SDK on your windows machine from here -

<https://cloud.google.com/sdk/docs/install>

- try run it from search menu by writing =SDK= (should add sdk to path somehow not to use this workaround)

- check if it's installed with =gcloud= command in cmd/powershell

- login to your gcp account with =gcloud auth login=

- =gcloud config get-value project=

- Try to list all your services by =gcloud services list=. You should see

~containerregistry.googleapis.com Container Registry API~ in it

- You should have a docker container already in your docker desktop

- Tag your container with GCP stuff(over SDK)

- =docker tag django-docker-starter gcr.io/lifeapi-392202/django-docker-starter=

- Explanations:

- gcr.io - default

- lifeapi-392202 - this is the ID of the Lifeapi project in GCP

- /django-docker-starter - should be the same as your image

- After the container has been created, we can push it to GCP registry

- =docker push gcr.io/lifeapi-392202/django-docker-starter=

- If you get an error saying

#+begin\_quote

unauthorized: You don't have the needed permissions to perform this

operation, and you may have invalid credentials. To authenticate your

request, follow the steps in:

<https://cloud.google.com/container-registry/docs/advanced-authentication>

#+end\_quote

then run this command - =gcloud auth configure-docker= and click y.

- try to push again, image should now be in here - <https://console.cloud.google.com/gcr/images>

- Copy image url, f.x gcr.io/lifeapi-392202/django-docker-starter

- Go to =Cloud Run= - <https://console.cloud.google.com/run>

- Click Create service

- choose the image

- change port to 8000

- tick "Allow unauthenticated invocations"

## restarting lifeapi services

#+begin\_src bash

# 1

systemctl stop nginx

# 2

systemctl stop gunicorn.service

# 3

systemctl stop gunicorn.socket

# TURN THEM ON AGAIN

# 1

systemctl start gunicorn.socket

# 2

systemctl start gunicorn.service

# 3

systemctl start nginx

#+end\_src

## local development on a new machine

- clone the repo

- create venv

- install packages from requirements.txt

- copy =.env\_example= and name it .env\_dev

- =python manage.py createsuperuser --settings=settings.development= (arvy kk)

- =python manage.py makemigrations --settings=settings.development=

- =python manage.py migrate --settings=settings.development=

- =python manage.py runserver --settings=settings.development=

- visit development server at =<http://127.0.0.1:8000/=>

- DB exist?

- Can add entries?

## step by step setup

\* Initial deployment

\*\* step by step deployment

\*\*\* ssh to the GCP server

First - [[[file:~/GIT/devnotes/20240104T070918--ssh-into-gcp-server\_\_gcp\_ssh.org][ssh-into-gcp-server](file:///\\~\GIT\devnotes\20240104T070918--ssh-into-gcp-server__gcp_ssh.org%5d%5bssh-into-gcp-server)]]

Second(optional) - [[[file:~/GIT/devnotes/20240104T075039--login-to-gcp-server-with-username-and-password\_\_gcp\_linux.org][login-to-gcp-server-with-username-and-password](file:///\\~\GIT\devnotes\20240104T075039--login-to-gcp-server-with-username-and-password__gcp_linux.org%5d%5blogin-to-gcp-server-with-username-and-password)]]

\*\*\* winscp to GCP server

add private key - connect

private key ends in .pem?

store that private key in such location, in wsl for example:

[\\wsl.localhost\Ubuntu\home\arvy\arvydas\_privatekey\_gcp.pem](file:///\\wsl.localhost\Ubuntu\home\arvy\arvydas_privatekey_gcp.pem)

the file must remain there, otherwise the connection will not work

<https://winscp.net/eng/docs/guide_google_compute_engine>

\*\*\* setup virtual env/server modifications

sudo su -

apt update

apt upgrade

apt install git

mkdir /opt/app

cd /opt/app

git clone <https://github.com/arvydasg/lifeapi.git>

apt install python3-venv python3-pip python3-dev libpq-dev postgresql

postgresql-contrib nginx curl

inspiration for deployment - -

<https://www.youtube.com/watch?v=bD75adrlkes&ab_channel=TheCodrammers>

python3 -m venv venv

source /venv/bin/activate

pradedu ties situ commit - 7904e80430b7b0f54c869c3e4366b0eb02d2e2c9

(Merge pull request #4 from arvydasg/DEV …) - <https://github.com/arvydasg/lifeapi/commit/7904e80430b7b0f54c869c3e4366b0eb02d2e2c9>

python manage.py migrate --settings=settings.production

python manage.py createsuperuser --settings=settings.production

collectstatic

# on local prod run

python manage.py collectstatic --settings=settings.production

then git ignore collectstatic dir which is "staticfiles-cdn/"

# push to dev

# pull request to master

git pull origin master # on the server

open 8000 port in server

apt install ufw

ufw allow 8000

allow 8000 on gcp

3 dots on server → view network details → default(under network) → firewalls → add firewall rule

how it looks before:

pictures how some firewall configuration was done in gcp

cd /opt/app/lifeapi

touch .env\_prod

vim .env\_prod

# add the following line

ALLOWED\_HOSTS=34.88.9.23

python manage.py runserver --settings=settings.production 0.0.0.0:8000

# refresh the page. You should see the site.

\*\*\* Okay why I am spending so much time on gunicorn and nginx

I was like dude if I can just run django in the terminal.. why bother.

same like with the db. Why bother with postgresql if you can use sqlite and why

bother with nginx/gunicorn if you can use simle process in systemd or just run

in the terminal.

Seems like there are some benefits… but still thinking about the simple

solution. Its MVP after all dude!!!!

Using Gunicorn and Nginx in combination with Django offers several benefits

over running Django in the terminal. Here are some advantages:

Improved performance and scalability: Gunicorn is a production-ready web server

that can handle multiple concurrent requests efficiently, improving the overall

performance of your Django application. It utilizes a pre-fork worker model and

supports multiple worker processes, allowing it to handle high traffic loads.

Nginx acts as a reverse proxy server, offloading some of the processing tasks

from Gunicorn and efficiently serving static files, resulting in better

scalability.

Load balancing and high availability: By using Gunicorn and Nginx together, you

can set up multiple Gunicorn worker processes and distribute incoming requests

among them. Nginx can act as a load balancer, evenly distributing the requests

across the available Gunicorn workers. This setup improves your application's

availability and can handle more traffic without overwhelming a single server

instance.

Handling concurrent connections: Gunicorn's ability to handle multiple

concurrent connections makes it more suitable for production environments

compared to running Django in the terminal. It allows your application to serve

multiple requests simultaneously, providing a better experience for your users.

Separation of concerns: Nginx and Gunicorn serve different purposes in a

typical Django deployment setup. Nginx specializes in serving static files,

handling SSL termination, and acting as a reverse proxy server. On the other

hand, Gunicorn focuses on running the Django application and managing the

dynamic content. This separation allows each component to perform its task

efficiently and enables easier troubleshooting and maintenance.

Regarding the issue with displaying the CSS of the admin panel when using Nginx

and Gunicorn, this problem is typically related to the configuration of Nginx.

You need to ensure that Nginx is configured to correctly serve static files and

that the appropriate static file directories are specified. By properly

configuring Nginx, you should be able to resolve this issue and have the admin

panel CSS displayed correctly.

Gunicorn Translates the client requests from http to our python calls that our

application can process

webserver gateway interface module - wsgi

# activate env

pip install gunicorn (ADD THIS TO REQUIREMENTS.TXT)

# if settings file would be in the project directory, then could run such

# command:

gunicorn --bind 0.0.0.0:8000 project\_name.wsgi

# but since we have moved the project files to a separate dir, do this instead:

gunicorn --env DJANGO\_SETTINGS\_MODULE=settings.production --bind 0.0.0.0:8000

lifeapi\_project.wsgi

# now the server runs over gunicorn

# IT WORKS!!!!! but it occupies our terminal.. not good can not I simply run

# tmux and launch a service in one window and do some tasks in another? or just

# do the runserver in tmux window.. hmm.. maybe only because its related to

# static files

gunicorn.socket

vim /etc/systemd/system/gunicorn.socket

[Unit] Description=gunicorn.socket

# socket location

[Socket] ListenStream=/run/gunicorn.sock

# make sure the socket is created at the right time

[Install] WantedBy=sockets.target gunicorn.service #file name should math. Can

see that both are gunicorn.

vim /etc/systemd/system/gunicorn.service

# only start the service file after the gunicorn socket is created because our

# service relies on the socket from the socket file

[Unit] Description=gunicorn daemon Requires=gunicorn.socket

After=network.target

# specify user (root, since it owns all relavant files) and group under which

# we want the process to run under

[Service] User=root Group=www-data WorkingDirectory=/opt/app/lifeapi

ExecStart=/opt/app/lifeapi/venv/bin/gunicorn \ --access-logfile - \ --workers 3

\ --bind unix:/run/gunicorn.sock \ lifeapi\_project.wsgi:application \ --env

DJANGO\_SETTINGS\_MODULE=settings.production

# will tell systemd what to link this service to if we enable it to start at

# boot we want this server to start when the regular multiuser system is up and

# running

[Install] WantedBy=multi-user.target systemctl start gunicorn.socket

systemctl enable gunicorn.socket

When a connection is made to gunicorn.socket systemd will automatically will

start systemd.service to handle it.

systemctl status gunicorn.socket - should be active

systemctl status gunicorn.service- should be inactive (dead), since it did nto

receive any connections

test activation mechanism:

curl --unix-socket /run/gunicorn.sock localhost

Should see some html outputted. For the dude in the video it was full html, for

me some html and 400 errors.

systemctl status gunicorn.service - now should be active

nginx vim /etc/nginx/sites-available/lifeapi

server { listen 80; server\_name 34.88.9.236; location = /favicon.ico {

access\_log off; log\_not\_found off; } location /static/ { autoindex on;

alias /opt/app/lifeapi/staticfiles-cdn/; } location / { include

proxy\_params; proxy\_pass <http://unix:/run/gunicorn.sock>; }

} sudo ln -s /etc/nginx/sites-available/lifeapi /etc/nginx/sites-enabled/

check any syntax errors:

nginx -t

systemctl restart nginx

open firewall for normal traffic on the port 80

ufw allow ‘Nginx Full’

We no longer need access to the development server (8000), so we can remove

this firewall rule:

ufw delete allow 8000

go to the 34.88.9.236 without any ports. Should work.

\*\*\* Configuring static files

this video inspiration - 74 - Static Files in Development - Python &

Django 3.2 Tutorial Series

global styles are better if wanting to override all the app styles in

one place. better for the future

can override admin styles also and other django stylesheets

remember to turn off caching in network panel browser

debug on - takes from static folder debug off - collecstatic need

in prod then do:

#+begin\_src python

python manage.py collectstatic --settings=settings.production

# 126 static files copied to '/home/arvy/src/lifeapi/staticfiles-cdn'.

#+end\_src

Ok we have stored mine and django’s static files in one location, but

they are not used by anything now. When debug is true - django takes

the static files from usual places, where we set them.

But if we are in production and we ant them to be server, we need to

do that with whitenoise.

#+begin\_src bash

pip install whitenoise

#+end\_src

add middleware like in WhiteNoise 6.5.0 documentation . After the

security one. At the top.

#+begin\_src python

MIDDLEWARE = [

# ...

"django.middleware.security.SecurityMiddleware",

"whitenoise.middleware.WhiteNoiseMiddleware",

# ...

]

#+end\_src

#+begin\_src bash

pip freeze > requirements.txt

#+end\_src

Run the server again. Even with debug = false static files for admin

panel should be displayed, as well as your global styles.

\*\*\* Running the server

# GCP Production

# 1. run with nginx

#+begin\_src bash

python manage.py runserver --settings=settings.production 0.0.0.0:8000

python manage.py runserver --settings=settings.production

python manage.py collectstatic --settings=settings.production

#+end\_src

in dev:

- change branch to dev

- create .env\_dev file next to manage.py

- add this:

#+begin\_src bash

ALLOWED\_HOSTS=127.0.0.1,localhost

DEBUG=1

#+end\_src

- python manage.py makemigrations --settings=settings.dev

- python manage.py migrate --settings=settings.dev

- python manage.py createsuperuser --settings=settings.dev (root kk)

- python manage.py collectstatic --settings=settings.dev

- python manage.py runserver --settings=settings.dev

or simply reuse the prod DB

\*\*\* Restarting the services

Link to the instructions - [[[file:20240104T082011--restarting-lifeapi-services\_\_lifeapi.org][Restarting](file:///\\20240104T082011--restarting-lifeapi-services__lifeapi.org][Restarting) the services]]

\*\*\* Source venv on each login to root

#+begin\_src bash

sudo su -

vim ~/.bashrc

# add the following line

source /opt/app/lifeapi/venv/bin/activate

:wq

source ~/.bashrc

#+end\_src

\*\*\* Cron job to fetch data automatically

[[[file:~/GIT/devnotes/20240104T082156--set-the-correct-server-time\_\_linux.org][set](file:///\\~\GIT\devnotes\20240104T082156--set-the-correct-server-time__linux.org%5d%5bset) the correct server time]]

#+begin\_src bash

cd /opt/app/lifeapi

chmod +x weather\_job\_prod.py

crontab -e

0 17 \* \* \* /opt/app/lifeapi/venv/bin/python /opt/app/lifeapi/weather\_job\_prod.py >> /opt/app/lifeapi/weather\_job\_prod.log 2>&1

#+end\_src

This cron job now should run every day at 17:00.

\*\* deployment to gcp

Attempt2 - bare metal

Firewall config - <https://console.cloud.google.com/networking/firewalls/list?project=lifeapi-392202>

SSH key stuff - <https://www.youtube.com/watch?v=fmh94mNQHQc&ab_channel=storagefreak>

Logs - <https://console.cloud.google.com/logs/>

Attempt1 - deploy a django solution

Created a project

Linked billing account

Set limits for billing 50 dolcu

Enabling required API’s

Enable required APIs The following APIs are required to deploy a VM product from Marketplace

Compute Engine API Not enabled

Cloud Deployment Manager V2 API Not enabled

Cloud Runtime Configuration API Not enabled

Software installed

Software

Operating systemDebian(11.6)

SoftwareApache2(2.4.56)

Django(4.1.7)

Git(2.30.2)

MySQL-Client(8.0.32)

MySQL-Community-Client(8.0.32)

MySQL-Community-Server(8.0.32)

MySQL-Server(8.0.32)

Conclusion

The build failed. For some reason. Uck it. Will make it myself better.

Just because it installs django and git for me I should use a premade

component and not know exactly what’s hiding under the hood? No

thanks.

\* Second attempt

\*\* The plan

- create new instance, make it run

- reset the old instance to the beginning, make it run(to keep the same IP address)

- [[[file:20240109T072346--dockerizing-the-app\_\_docker\_lifeapi.org][Dockerize](file:///\\20240109T072346--dockerizing-the-app__docker_lifeapi.org][Dockerize) the app]]

\*\* Creating a new GCP instance

Before clicking "Create", I copied the equivalent code.

Don't choose e2-micro instance, since it will not be able to install needed

packages during sudo apt upgrade -

<https://serverfault.com/questions/1134676/apt-upgrade-y-command-stuck-on-preparing-to-unpack-google-cloud-cli-436-0>

\*\*\* Code for creating instance

Can look at this code and know what settings I have chosen.

#+begin\_src bash

gcloud compute instances create instance-2 \

--project=lifeapi-392202 \

--zone=us-central1-a \

--machine-type=e2-micro \

--network-interface=network-tier=PREMIUM,stack-type=IPV4\_ONLY,subnet=default \

--maintenance-policy=MIGRATE \

--provisioning-model=STANDARD \

--service-account=46844210845-compute@developer.gserviceaccount.com \

--scopes=https://www.googleapis.com/auth/devstorage.read\_only,https://www.googleapis.com/auth/logging.write,https://www.googleapis.com/auth/monitoring.write,https://www.googleapis.com/auth/servicecontrol,https://www.googleapis.com/auth/service.management.readonly,https://www.googleapis.com/auth/trace.append \

--tags=http-server,https-server,lb-health-check \

--create-disk=auto-delete=yes,boot=yes,device-name=instance-2,image=projects/debian-cloud/global/images/debian-11-bullseye-v20231212,mode=rw,size=10,type=projects/lifeapi-392202/zones/us-central1-a/diskTypes/pd-standard \

--no-shielded-secure-boot \

--shielded-vtpm \

--shielded-integrity-monitoring \

--labels=goog-ec-src=vm\_add-gcloud \

--reservation-affinity=any

#+end\_src

\*\*\* Pricing

Basically it's a e2-micro instance

Monthly estimate

US$6.51

That's about US$0.01 hourly

Pay for what you use: No upfront costs and per-second billing

Item

Monthly estimate

2 vCPU + 1 GB memory

US$6.11

10 GB standard persistent disk

US$0.40

Total

US$6.51

\*\* Connect to the machine over gcp console

connect over the provided default console first. should log you in with

arvydas\_gaspa username.

\*\* Connect over ssh

First - [[[file:~/GIT/devnotes/20240104T070918--ssh-into-gcp-server\_\_gcp\_ssh.org][ssh-into-gcp-server](file:///\\~\GIT\devnotes\20240104T070918--ssh-into-gcp-server__gcp_ssh.org%5d%5bssh-into-gcp-server)]]

Second(optional) - [[[file:~/GIT/devnotes/20240104T075039--login-to-gcp-server-with-username-and-password\_\_gcp\_linux.org][login-to-gcp-server-with-username-and-password](file:///\\~\GIT\devnotes\20240104T075039--login-to-gcp-server-with-username-and-password__gcp_linux.org%5d%5blogin-to-gcp-server-with-username-and-password)]]

\*\*\* ssh key info for instace2

fingerprint

#+begin\_quote

ssh-rsa 2048 SHA256:9gdAeVoX567uowDrkLCSOFVGlziSlKnSbpZDVLLdHdU

#+end\_quote

key comment(username)

#+begin\_quote

arvydas

#+end\_quote

public\_key

#+begin\_quote

ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQCufZnGwCFbJyYGJtRULgEBf1qDzb8jsCFoArSd8AkCUr9mAsEbHwvuxPKQ84w3BiIeizS/Y7w168ZHcaUheWAG/pSPYxhgGU6QknW5PAVVCHPIxiXb6p6LhUbrluahcE5Cq5+HYhFZXQQ+pftI7fG/OGvBAG6f9FoxuxFaUJMhvKrQ1DG1j65I6U1UMfWVdOtyqZTw0qMpXTMjgd3rKy5bTyxEqIXH+9PL6pmSCckQZOPXXeouAqQMBRQuoxKpsG28ctTgKbjDOMoyytW9YmAAHFzxoP9tLViQMHkScWBtPKuud95IyTEdgNbwjnjUD8SCfNbjBGFITFOdURgVQwgx arvydas

#+end\_quote

private\_key

#+begin\_quote

PuTTY-User-Key-File-3: ssh-rsa

Encryption: none

Comment: arvydas

Public-Lines: 6

AAAAB3NzaC1yc2EAAAADAQABAAABAQCufZnGwCFbJyYGJtRULgEBf1qDzb8jsCFo

ArSd8AkCUr9mAsEbHwvuxPKQ84w3BiIeizS/Y7w168ZHcaUheWAG/pSPYxhgGU6Q

knW5PAVVCHPIxiXb6p6LhUbrluahcE5Cq5+HYhFZXQQ+pftI7fG/OGvBAG6f9Fox

uxFaUJMhvKrQ1DG1j65I6U1UMfWVdOtyqZTw0qMpXTMjgd3rKy5bTyxEqIXH+9PL

6pmSCckQZOPXXeouAqQMBRQuoxKpsG28ctTgKbjDOMoyytW9YmAAHFzxoP9tLViQ

MHkScWBtPKuud95IyTEdgNbwjnjUD8SCfNbjBGFITFOdURgVQwgx

Private-Lines: 14

AAABAFUBPUP034sflEeU7QWhb74CA9+IASDqsiuQfdsfT9RA6ZtRpi+HPXHxolX5

QAqiQ0br/CNs/AistuihNZgMIDroFQmRdhOC4KJPp2g5FEPrnTRnS5RKRTilEfq9

hdeJ9aZHI615mggV53Z5t+Q8fvPwEZZxlnL4QGRPxNFhxXu+NQWItmZ2Omkg3st7

8oJhTgi1WuYvVnZcIzeNJhITf1pYdrOat4tFAL7J4QjMVIFa+KpKuK3Fa4DKY0XD

r+KAZuigU3vFdQjdMRvi27oJqb1qnyL1G1Z2iR3cjYWFM3KFT7mewQ7X+FERDhBp

APjQ9ac4NieojYoQNtrvL00uHX0AAACBAPIz8KqgrUW1OEbr1XH9PrTdmxPCgZEp

mAUfETiqYfDvhBKGzJNxekS9nHSUxjVQXWWDiXJ6KePCPu0Y1H/oKcX4ntxin5Wk

8CljbfbPHR+dvFC3jV0iGv8Gdctq8bp+eEp4ZCTU3YDWgO5tjeup3TarpooPFv2Y

LwMZprbNZoyvAAAAgQC4bjVa1+BPCSpNZAiYOyUbXCv5mGWTbstyB+7dTqEOjCHy

QEHdrtK8YoxmYY3ItJdjuoqKB0Souo9pSn7Vi/xN75DfapnSkF/JA1qIV5BHyuQG

P6OyPLAY08PGqapxKqI0LEyFQ9WxcJBAoxUyCdQDvysO979Snr2oLVcdvV2xHwAA

AIEA1tfFjfPQCHPzW40hhB/SA6l96lVRKykAtNQoDwNz7pv6Arg662yY6A8+RHRJ

FuWeBwlmg/BHgAW40RSOYIkw95CFXqOF7YxH7XgnR1RDcMwyJKoJ71lJLKFOOJze

RVLnm1zeO/BSSS2lKXEGEo6weH5ur4sNLYLi1aBKPTYAJMw=

Private-MAC: 89ee5eb4d7113171bf86a39f453364f288e910bac34081acf1b4c117bf01a8f3

#+end\_quote

## cronjobs

\* weather fetch cron

#+begin\_src bash

0 18 \* \* \* /opt/app/lifeapi/venv/bin/python /opt/app/lifeapi/weather\_job\_prod.py >> /opt/app/lifeapi/weather\_job\_prod.log 2>&1

#+end\_src

\* rescuetime fetch

making the cronjob a bit later, like at 5am, because linux "date"

command shows one time and python's datetime.now() shows a different

time. Can make it equal with pytz, but no matter for now, lets see if

it works with 5am.

\* make backup push to git cron

#+begin\_src bash

0 0 \* \* \* /opt/app/lifeapi\_db\_backups/backup\_prd.sh >> /opt/app/lifeapi\_db\_backups/db\_backups.log 2>&1

#+end\_src

## rescuetime api implementation

<https://github.com/arvydasg/lifeapi/pull/65>

go to the site, get api key for each person

AG api key is - B63fYDJzgZhrzaNAbRXRgMoQ1Qxowu3iHU3Ukrpw

you can then make such queries - <https://www.rescuetime.com/anapi/daily_summary_feed?key=B63fYDJzgZhrzaNAbRXRgMoQ1Qxowu3iHU3Ukrpw>

Then go to insomnia, add the key with API key auth, then you can make

such queries then -

<https://www.rescuetime.com/anapi/daily_summary_feed>

here are some more examples -

<https://www.rescuetime.com/anapi/data?&perspective=interval&restrict_kind=productivity&interval=hour&restrict_begin=2023-10-10&restrict_end=2023-10-13&format=csv>

<https://www.rescuetime.com/anapi/data?&perspective=rank&restrict_kind=overview&restrict_begin=2023-10-12&restrict_end=2023-10-13&format=csv>

sadly you can not extract day values from daily summary, so we will

have to do it like such:

in this example I have my api keys in env variables.

#+begin\_src python

import requests

import django

import os

# Set the DJANGO\_SETTINGS\_MODULE environment variable

os.environ.setdefault("DJANGO\_SETTINGS\_MODULE", "settings.development")

# Configure Django settings

django.setup()

# Your RescueTime API key

api\_key\_ag = os.getenv("RESCUETIME\_API\_KEY\_AG")

api\_key\_js = os.getenv("RESCUETIME\_API\_KEY\_JS")

# Define the API endpoint URL

api\_url = f"https://www.rescuetime.com/anapi/daily\_summary\_feed?key={api\_key\_ag}"

# Make the API request

response = requests.get(api\_url)

if response.status\_code == 200:

data = response.json()

# Define the date you want to filter for

target\_date = "2023-10-14"

# Filter the data for the specific date

# here we get a big json file, so this list comperhension takes what we need

filtered\_data = [entry for entry in data if entry['date'] == target\_date] # list comprehension

if filtered\_data:

# Process the filtered data as needed

for entry in filtered\_data:

print(f"Date: {entry['date']}") # we're accessing the 'date' key in the dictionary called 'entry'

print(f"Total hours: {entry['total\_hours']}")

print(f"All productive hours: {entry['all\_productive\_hours']}")

print(f"All distracting hours: {entry['all\_distracting\_hours']}")

else:

print(f"No data found for {target\_date}")

else:

print(f"Failed to retrieve data. Status code: {response.status\_code}")

#+end\_src

but can also do it like such in view:

#+begin\_src python

def rescuetime\_app\_dsf(request):

# take the api key from environment variables

rescuetime\_api\_key\_ag = os.getenv("RESCUETIME\_API\_KEY\_AG")

# Define the URL

url = f'https://www.rescuetime.com/anapi/daily\_summary\_feed?key={rescuetime\_api\_key\_ag}'

url2 = f'https://www.rescuetime.com/anapi/data?key={rescuetime\_api\_key\_ag}&perspective=rank&restrict\_kind=overview&restrict\_begin=2023-10-10&restrict\_end=2023-10-10&format=json'

try:

# Make an HTTP GET request to the URL

response = requests.get(url)

response2 = requests.get(url2)

# Check if the request was successful (status code 200)

if response.status\_code == 200:

# Parse the JSON response

data = response.json()

data2 = response2.json()

# Pass the data to the template for rendering

context = {

'data': data,

'data2': data2

}

return render(request, 'rescuetime\_app\_dsf.html', context)

else:

# Handle any other status code (e.g., display an error)

return render(request, 'rescuetime\_app\_error.html', {'error\_message': 'Failed to fetch data'})

except Exception as e:

# Handle any exceptions (e.g., network issues, JSON parsing errors)

return render(request, 'rescuetime\_app\_error.html', {'error\_message': str(e)})

#+end\_src

## daylt api implementation

Vardadieniai bei dienos informacija Tavo svetainėje!-

<https://day.lt/dienos_info_paaiskinimai.html>

Nothing to explain, its just too easy to implement. If it dies - it

dies. If it runs - it runs.

## implementing strava api

\* Strava api

\*\* links

- strava developers - <https://developers.strava.com/>

- strava account - <https://www.strava.com/dashboard?num_entries=60>

- strava everyone activity - <https://www.strava.com/activities/9074302414>

- strava api section in account page - <https://www.strava.com/settings/api>

- strava api docs - <https://developers.strava.com/docs/reference/#api-Activities-getActivityById>

- Getting Started with the Strava API -

<https://developers.strava.com/docs/getting-started/>

## lifeapi git work

\* applying the change from dev to PRD

\*\* Check services status

#+begin\_src bash

systemctl status gunicorn.socket

systemctl status gunicorn.service

systemctl status nginx

#+end\_src

\*\* Fetch changes

check if there are any changes upstream(on github repo on the web)

git fetch

Push changes to DEV

Github site make pull request to master

login to gcp server

login as root user

git pull origin master on gcp

if some small change, maybe systemctl restart nginx is enough

if change in css - need to do python manage.py collectstatic --settings=settings.production

if change in some templates - need to restart ir gunicorn systemctl

restart gunicorn.service (gal ir socket?)

\* update master

To login to MASTER branch on GCP - username "arvydas", ssh setup, so

no need for password

1. Login as root with - sudo su - . Should already be venv since

bashrc is made

2. git status

3. cd /opt/app/lifeapi/

4. gipt fetch

5. git pull origin master

6. python manage.py collectstatic --settings=settings.production

7. systemctl restart nginx

8. systemctl restart gunicorn.service

\* git repo change remote

krc buvo taip. as parsiputes repo buvau su https. paskui pasetupinau

ssh keys serveryje del lifeapi\_db\_backup. Su ssh keys viskas veikia

toje repo, bet nustojo veikti lifeapi. Tai teko daryti situos

zingsniu. Aciu dievui, kad pavyko sitaip, nereikejo reclone ir rebuild

stuff...

It seems like you've set up SSH keys for your GitHub account but are

still trying to access a repository using HTTPS, and you're

encountering authentication issues. You don't need to re-clone the

repository with SSH; you can update the remote URL for your existing

repository to use SSH. Here's how you can do it:

Open a terminal and navigate to your local repository.

Check the current remote URL using the following command:

#+begin\_src bash

git remote -v

#+end\_src

You'll see the remote URL for your repository, which should currently be in HTTPS format.

To update the remote URL to use SSH, use the following command

(replace username/repo with your actual repository path):

#+begin\_src bash

git remote set-url origin git@github.com:username/repo.git

#+end\_src

After updating the remote URL, you can verify that it's using SSH by

running git remote -v again. It should now show the SSH URL.

Try performing any Git operation (e.g., git pull, git push, etc.) to

confirm that you're no longer prompted for a username and password.

## data table creation

commit - ba650b7

Created data table · arvydasg/lifeapi@ba650b7 - <https://github.com/arvydasg/lifeapi/commit/ba650b7b80596689d196be3a648904d9d63bb41f>

This html template seems a bit complicated:

#+begin\_src html

{% extends 'base.html' %}

{% block content %}

<h2>Data Table</h2>

<div class="table-responsive">

<table class="table table-striped">

<thead>

<tr>

<th>Date</th>

<th>Temperature</th>

{% for question in questions %}

<th>{{question.description}}</th>

{% endfor %}

</tr>

</thead>

<tbody>

{% for weather\_entry in weather\_entries|dictsortreversed:'date' %}

<tr>

<td>{{ weather\_entry.date|date:"Y-m-d" }}</td>

<td>{{ weather\_entry.temperature }}</td>

{% for question in questions %}

{% for answer in question.answer\_set.all %}

{% if answer.date\_added.date == weather\_entry.date %}

<td>{{ answer.answer }}</td>

{% endif %}

{% endfor %}

{% endfor %}

</tr>

{% endfor %}

</tbody>

</table>

</div>

{% endblock %}

#+end\_src

Here is an explanation:

The nested loop you mentioned is used to populate the table rows (<tr>) with data from the weather\_entries and questions. Let's break down the loop step by step:

{% for weather\_entry in weather\_entries %}: This loop iterates over each weather\_entry object in the weather\_entries queryset. It represents each row in the table.

<tr>: Starts a new table row for each weather\_entry.

<td>{{ weather\_entry.date }}</td>: Displays the date value of the current weather\_entry object in a table cell. This corresponds to the "Date" column of the table.

<td>{{ weather\_entry.temperature }}</td>: Displays the temperature value of the current weather\_entry object in a table cell. This corresponds to the "Temperature" column of the table.

{% for question in questions %}: This loop iterates over each question object in the questions queryset. It represents the columns for the questions in the table.

{% for answer in question.answer\_set.all %}: This loop iterates over each answer object related to the current question object. It retrieves all answers associated with the current question.

{% if answer.date\_added.date == weather\_entry.date %}: This conditional statement checks if the date\_added value of the current answer object is equal to the date value of the current weather\_entry object. It ensures that the answer is associated with the correct date and should be displayed in the table.

<td>{{ answer.answer }}</td>: If the condition is met, it displays the answer value of the current answer object in a table cell. This corresponds to the answer for the specific question and date.

By nesting the loops, the code checks each question and its associated answer objects for each weather\_entry, allowing the table to display the relevant answers for each question and date.

## user question answer implementation

\* First created the models:

#+begin\_src python

from django.db import models

from django.utils import timezone

# Create your models here.

class Question(models.Model):

description = models.TextField()

class Answer(models.Model):

question = models.ForeignKey(Question, on\_delete=models.CASCADE)

answer = models.CharField(max\_length=255, blank=False)

date\_added = models.DateTimeField(default=timezone.now)

#+end\_src

\* Then the views:

#+begin\_src python

from django.shortcuts import render, redirect

from .models import Question, Answer

from django.http import HttpResponse

from django.utils import timezone

from datetime import date

from django.contrib import messages

from .forms import QuestionForm

def quiz\_app\_home(request):

return render(request, 'quiz\_app\_home.html')

def quiz\_questions(request):

'''View to display all the questions'''

questions = Question.objects.all()

context = {'questions': questions}

return render(request, 'quiz\_app\_questions.html', context)

def quiz\_add\_question(request):

'''View to add a posibility to add new questions'''

if request.method == 'POST':

form = QuestionForm(request.POST)

if form.is\_valid():

form.save()

return redirect('quiz\_questions')

else:

form = QuestionForm()

context = {'form': form}

return render(request, 'quiz\_app\_add\_question.html', context)

def quiz\_start(request):

'''View to add a posibility to add new questions'''

if request.method == 'POST':

# Check if the user clicked the "Start Quiz" button

if 'start\_quiz' in request.POST:

# Check if entries for today already exist

today = date.today()

if Answer.objects.filter(date\_added\_\_date=today).exists():

messages.warning(request, "You have already answered the quiz for today.")

else:

# Retrieve the first question from the database

first\_question = Question.objects.first()

context = {'question': first\_question}

return render(request, 'quiz\_app\_question.html', context)

# Check if the user clicked the "Delete Answers" button

elif 'delete\_answers' in request.POST:

today = date.today()

Answer.objects.filter(date\_added\_\_date=today).delete()

messages.success(request, "Your answers for today have been deleted.")

# Retrieve any flash messages and pass them to the template context

messages\_to\_display = messages.get\_messages(request)

context = {'messages': messages\_to\_display}

return render(request, 'quiz\_app\_ready.html', context)

def quiz\_question(request, question\_id):

if request.method == 'POST':

question\_id = int(request.POST.get('question\_id'))

answer\_text = request.POST.get('answer')

# Save the answer to the database

answer = Answer(question\_id=question\_id, answer=answer\_text)

answer.save()

# Redirect to the next question or finish the quiz if all questions are answered

next\_question\_id = question\_id + 1

if next\_question\_id > Question.objects.count():

return redirect('quiz\_summary') # Redirect to the quiz summary page

else:

question = Question.objects.get(id=next\_question\_id)

context = {'question': question}

return render(request, 'quiz\_app\_question.html', context)

# Retrieve the question based on the question\_id

question = Question.objects.get(id=question\_id)

context = {'question': question}

return render(request, 'quiz\_app\_question.html', context)

def quiz\_summary(request):

'''View to display all entries of answers table'''

answers = Answer.objects.all()

context = {'answers': answers}

return render(request, 'quiz\_app\_summary.html', context)

#+end\_src

\* Then the urls:

#+begin\_src python

from django.urls import path

from . import views

urlpatterns = [

path("", views.quiz\_app\_home, name="quiz\_app\_home"),

path('start\_quiz/', views.quiz\_start, name='quiz\_start'),

path("quiz/questions/", views.quiz\_questions, name="quiz\_questions"),

path('quiz/<int:question\_id>/', views.quiz\_question, name='quiz\_question'),

path('quiz/summary/', views.quiz\_summary, name='quiz\_summary'),

path('add-question/', views.quiz\_add\_question, name='quiz\_add\_question'),

]

#+end\_src

\* And forms:

#+begin\_src python

from django import forms

from .models import Question

class QuestionForm(forms.ModelForm):

class Meta:

model = Question

fields = ['description']

#+end\_src

\* And templates:

\*\* quiz\_app\_add\_question.html

#+begin\_src html

{% extends 'base.html' %}

{% block content %}

<h2>Add Question</h2>

<form method="POST">

{% csrf\_token %}

{{ form.as\_p }}

<button type="submit">Add</button>

</form>

{% endblock %}

#+end\_src

\*\* quiz\_app\_home.html

#+begin\_src html

{% extends 'base.html' %}

{% block content %}

<h1>Welcome to My QUIZ APP</h1>

<p>Here you can answer some questions bla.</p>

<p>Quiz is here - <a href="{% url 'quiz\_start' %}">Here</a></p>

<p>all questions can see - <a href="{% url 'quiz\_questions' %}">Here</a></p>

{% endblock %}

#+end\_src

\*\* quiz\_app\_question.html

#+begin\_src html

{% extends 'base.html' %}

{% block content %}

<h2>Question:</h2>

<p>{{ question.description }}</p>

<form action="{% url 'quiz\_question' question\_id=question.id %}" method="POST">

{% csrf\_token %}

<input type="hidden" name="question\_id" value="{{ question.id }}">

<input type="text" name="answer" placeholder="Enter your answer" required>

<button type="submit">Submit</button>

</form>

{% endblock %}

#+end\_src

\*\* quiz\_app\_questions.html

#+begin\_src html

{% extends 'base.html' %}

{% block content %}

<h1>Questions</h1>

<table class="table">

<thead>

<tr>

<th>ID</th>

<th>Description</th>

</tr>

</thead>

<tbody>

{% for question in questions %}

<tr>

<td>{{ question.id }}</td>

<td>{{ question.description }}</td>

</tr>

{% endfor %}

</tbody>

</table>

<a href="{% url 'quiz\_add\_question' %}" class="btn btn-primary">Add Question</a>

{% endblock %}

#+end\_src

\*\* quiz\_app\_ready.html

#+begin\_src html

{% extends 'base.html' %}

{% block content %}

<h2>Ready to start the quiz?</h2>

<form action="{% url 'quiz\_start' %}" method="POST">

{% csrf\_token %}

<button type="submit" name="start\_quiz">Start Quiz</button>

</form>

<form action="{% url 'quiz\_start' %}" method="POST">

{% csrf\_token %}

<button type="submit" name="delete\_answers">Delete My Answers for Today</button>

</form>

{% for message in messages %}

<div class="flash-message">{{ message }}</div>

{% endfor %}

{% endblock %}

#+end\_src

\*\* quiz\_app\_summary.html

#+begin\_src html

{% extends 'base.html' %}

{% block content %}

<h2>Quiz Summary</h2>

<table>

<thead>

<tr>

<th>Question</th>

<th>Answer</th>

</tr>

</thead>

<tbody>

{% for answer in answers %}

<tr>

<td>{{ answer.question.description }}</td>

<td>{{ answer.answer }}</td>

</tr>

{% endfor %}

</tbody>

</table>

{% endblock %}

#+end\_src

\*\* Quiz looks like this:

If there are answers to the quiz already for today - you won’t be able

to do the quiz. You can delete the entries before that.

## Weather api implementation

\* code

Using this as an api - Meteo.lt API

Display data from an api in html.

\*\* Step 1 Create a separate django app

add it to the project. Inside the app’s views.py write such code:

#+begin\_src python

import requests

from django.shortcuts import render

from datetime import datetime, timedelta

# Create your views here.

def weather\_view(request):

# Get yesterday's date

yesterday = datetime.now() - timedelta(days=5)

date\_str = yesterday.strftime("%Y-%m-%d")

# Make the API request

response = requests.get(

f"https://api.meteo.lt/v1/stations/vilniaus-ams/observations/{date\_str}"

)

data = response.json()

print(data)

# Find the desired observation

desired\_observation = None

for x in data["observations"]:

if x["observationTimeUtc"] == f"{date\_str} 12:00:00":

desired\_observation = x

break

# assign the desired\_observation to context

context = {

"observation": desired\_observation,

}

# pass that context to the template and render it

return render(request, "weather/weather\_template.html", context)

#+end\_src

Then /templates/weather/weather\_template.html can be like such:

#+begin\_src html

<!DOCTYPE html>

<html>

<head>

<title>Weather Information</title>

</head>

<body>

<h1>Weather Information</h1>

{% if observation %}

<p>Observation Time: {{ observation.observationTimeUtc }}</p>

<p>Air Temperature: {{ observation.airTemperature }}</p>

<p>Feels Like Temperature: {{ observation.feelsLikeTemperature }}</p>

<p>Wind Speed: {{ observation.windSpeed }}</p>

<p>Wind Gust: {{ observation.windGust }}</p>

<p>Wind Direction: {{ observation.windDirection }}</p>

<p>Cloud Cover: {{ observation.cloudCover }}</p>

<p>Sea Level Pressure: {{ observation.seaLevelPressure }}</p>

<p>Relative Humidity: {{ observation.relativeHumidity }}</p>

<p>Precipitation: {{ observation.precipitation }}</p>

<p>Condition Code: {{ observation.conditionCode }}</p>

{% else %}

<p>No observation found for the specified time.</p>

{% endif %}

</body>

</html>

#+end\_src

\*\* Step 2 Storing entries from an api in db

Let’s create a DB first. Inside app/model.py file:

#+begin\_src python

from django.db import models

# Create your models here.

class Weather(models.Model):

date = models.DateField()

temperature = models.DecimalField(max\_digits=5, decimal\_places=2)

#+end\_src

then in terminal:

#+begin\_src bash

python manage.py makemigrations

python manage.py migrate

python manage.py runserver

#+end\_src

Now update the view function to include:

# Store the values in the database

#+begin\_src python

from . models import Weather

weather = Weather(date=date\_str, temperature=desired\_observation.get("airTemperature", ""))

weather.save()

#+end\_src

Now whenever you visit the <http://127.0.0.1:8000/weather/> page, the

data will be fetched and stored in the DB.

\*\* Step 3 Displaying from an api AND storing to db

Its possible to do this in one single views.py file. We must have

models and urls set up like so:

models

#+begin\_src python

from django.db import models

# Create your models here.

class Weather(models.Model):

date = models.DateField()

temperature = models.DecimalField(max\_digits=5, decimal\_places=2)

#+end\_src

urls

#+begin\_src python

from django.urls import path

from . import views

urlpatterns = [

path("", views.weather\_view, name="weather\_view"),

]

#+end\_src

views

#+begin\_src python

import requests

from django.shortcuts import render

from datetime import datetime, timedelta

from .models import Weather

def weather\_view(request):

# Get yesterday's date

yesterday = datetime.now() - timedelta(days=1)

date\_str = yesterday.strftime("%Y-%m-%d")

# Make the API request to fetch weather data

api\_url = f"https://api.meteo.lt/v1/stations/vilniaus-ams/observations/{date\_str}"

response = requests.get(api\_url)

data\_fetched\_from\_api = response.json()

# Retrieve desired observation from the fetched data

desired\_observation = None

for observation in data\_fetched\_from\_api["observations"]:

if observation["observationTimeUtc"] == f"{date\_str} 12:00:00":

desired\_observation = observation

break

# Save the weather data to the database

save\_weather\_to\_db(date\_str, desired\_observation)

# Retrieve the latest weather data from the database

weather\_from\_db = retrieve\_latest\_weather()

# Prepare the context to pass to the template

context = {

"observation": desired\_observation,

"weather\_from\_db": weather\_from\_db,

}

# Render the template with the context

return render(request, "weather/weather\_template.html", context)

def save\_weather\_to\_db(date\_str, desired\_observation):

# Save the weather data to the database

weather = Weather(

date=date\_str,

temperature=desired\_observation.get("airTemperature", "")

)

weather.save()

def retrieve\_latest\_weather():

# Retrieve the latest weather data from the database

weather\_from\_db = Weather.objects.last()

return weather\_from\_db

#+end\_src

html template

#+begin\_src html

<!DOCTYPE html>

<html>

<head>

<title>Weather Information</title>

</head>

<body>

<!-- data that is fetched from an api -->

<h1>Weather Information</h1>

{% if observation %}

<p>Observation Time: {{ observation.observationTimeUtc }}</p>

<p>Air Temperature: {{ observation.airTemperature }}</p>

<p>Feels Like Temperature: {{ observation.feelsLikeTemperature }}</p>

<p>Wind Speed: {{ observation.windSpeed }}</p>

<p>Wind Gust: {{ observation.windGust }}</p>

<p>Wind Direction: {{ observation.windDirection }}</p>

<p>Cloud Cover: {{ observation.cloudCover }}</p>

<p>Sea Level Pressure: {{ observation.seaLevelPressure }}</p>

<p>Relative Humidity: {{ observation.relativeHumidity }}</p>

<p>Precipitation: {{ observation.precipitation }}</p>

<p>Condition Code: {{ observation.conditionCode }}</p>

{% else %}

<p>No observation found for the specified time.</p>

{% endif %}

<!-- data fetched from the db an api -->

{% if weather\_from\_db %}

<p>Observation Time: {{ weather\_from\_db.date|date:"Y-m-d" }}</p>

<p>Temperature: {{ weather\_from\_db.temperature }}</p>

{% else %}

<p>No observation found.</p>

{% endif %}

</body>

</html>

#+end\_src

Now upon each refresh we will get data displayed in html template from

an api and also from the db.

DB will populate like so:

Using this DB viewer - SQLite Viewer Web App

And will be displayed like so:

Now have to separate those processes and make it so that ONLY the data

from the DB is displayed and the fetch from api and store to db action

happens not on each page refresh but at specific times, without my

intervention.

\*\* Step 4 Automating fetching and storing into DB with cron

So since we don’t want fetch and storing to DB happen each time we

open /website tab, lets clean up the views.py to contain only the

information needed for information display.

#+begin\_src python

from django.shortcuts import render

from .models import Weather

def weather\_view(request):

# Retrieve the latest weather data from the database

weather\_from\_db = retrieve\_latest\_weather()

# Prepare the context to pass to the template

context = {

"weather\_from\_db": weather\_from\_db,

}

# Render the template with the context

return render(request, "weather/weather\_template.html", context)

def retrieve\_latest\_weather():

# Retrieve the latest weather data from the database

weather\_from\_db = Weather.objects.last()

return weather\_from\_db

#+end\_src

Let’s add the content from from views.py file to another file called

job\_weather.py that should be located in the same directory as

manage.py

#+begin\_src python

import requests

from datetime import datetime, timedelta

from weather.models import Weather

def save\_weather\_to\_db(date\_str, desired\_observation):

# Save the weather data to the database

weather = Weather(

date=date\_str,

temperature=desired\_observation.get("airTemperature", "")

)

weather.save()

print(f"{date\_str} and {desired\_observation.get('airTemperature', '')} are saved to the database")

# Get yesterday's date

yesterday = datetime.now() - timedelta(days=1)

date\_str = yesterday.strftime("%Y-%m-%d")

# Make the API request to fetch weather data

api\_url = f"https://api.meteo.lt/v1/stations/vilniaus-ams/observations/{date\_str}"

response = requests.get(api\_url)

data\_fetched\_from\_api = response.json()

# Retrieve desired observation from the fetched data

desired\_observation = None

for observation in data\_fetched\_from\_api["observations"]:

if observation["observationTimeUtc"] == f"{date\_str} 12:00:00":

desired\_observation = observation

break

# Save the weather data to the database

save\_weather\_to\_db(date\_str, desired\_observation)

#+end\_src

If we try to run this file now, we get an error saying:

ImproperlyConfigured: Requested setting INSTALLED\_APPS, but settings

are not configured. You must either define the environment variable

DJANGO\_SETTINGS\_MODULE or call settings.configure() before accessing

settings.

This means that this newly created file can not make calls and operate

PUT operations to our DJANGO database. We need to configure Django

settings for script execution outside of a Django project. This

enables the proper functioning of Django-related features, including

database access via models. Okay, so the solution is to add this a the

top of the file:

#+begin\_src python

import os

import django

# Set the DJANGO\_SETTINGS\_MODULE environment variable

os.environ.setdefault("DJANGO\_SETTINGS\_MODULE", "lifeapi.settings")

# Configure Django settings

django.setup()

#+end\_src

Now when we run this python file, the outcome should be:

#+begin\_src bash

(venv) arvy@DESKTOP-AUDMJ7D:~/src/lifeapi/lifeapi$ python job\_weather.py

2023-06-20 and 25.9 are saved to the database

#+end\_src

Check the DB to confirm that additional line was added.

\*\* Step 5 creating cronjob

We don’t want to run this command manually daily, so let’s create a

cron job to do this for us once every one minute(make once every

24hours later).

#+begin\_src bash

# open cron editor

crontab -e

# add this line at the end of it:

,\*/1 \* \* \* \* /home/arvy/src/lifeapi/venv/bin/python /home/arvy/src/lifeapi/lifeapi/job\_weather.py

# close the file

# make it executable

chmod +x job\_weather.py

#+end\_src

Now our cron job is created, it will use python with all the

dependencies from the environment that we have provided and run the

job\_weather.py file for us each minute.

Great!!