# Gerenciador de senhas

acesse diretamente pelo Notion:

```
https://grizzly-amaranthus-f6a.notion.site/Gerenciador-de-senhas-12e6cf8ea89f80ccac80ebb6e3efdf07?pvs=4
```

Primeiro devemos criar o ambiente virtual:

```
# Criar
    # Linux
        python3 -m venv venv
    # Windows
        python -m venv venv
```

Após a criação do venv vamos ativa-lo:

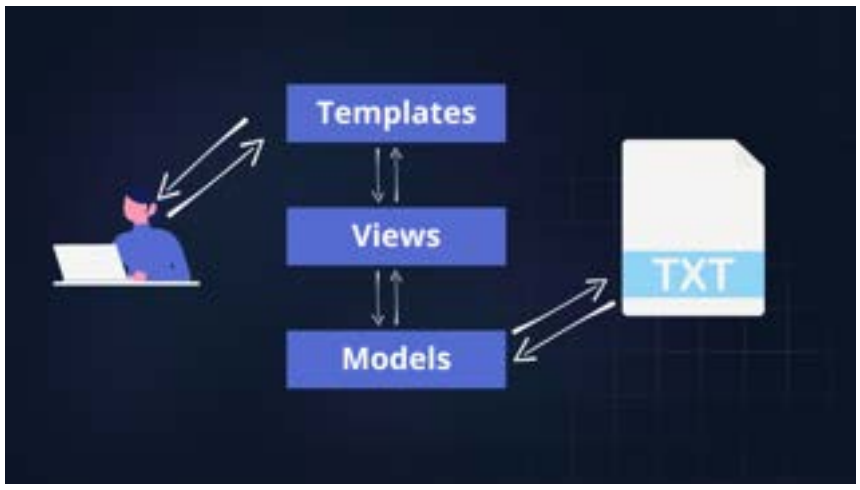
```
#Ativar
    # Linux
        source venv/bin/activate
    # Windows
        venv\Scripts\Activate

# Caso algum comando retorne um erro de permissão execute o código e tente novamente:

Set-ExecutionPolicy -Scope CurrentUser -ExecutionPolicy RemoteSigned
```

Agora vamos fazer a instalação do Django e as demais bibliotecas:

```
pip install cryptography
```




## ▼ Criptografia

Em views/password_views.py crie a classe FernetHasher e o método get random string para gerar uma chave aleatóri

```
import string, secrets

class FernetHasher:
    RANDOM_STRING_CHARS = string.ascii_lowercase + string.ascii_lowercase


    @classmethod
    def _get_random_string(cls, lenght=25):
        string = ''
        for i in range(lenght):
            string += secrets.choice(cls.RANDOM_STRING_CHARS)

        return string
```

Agora, crie a hash da chave e converta para b64:

```
    @classmethod
    def create_key(cls, archive=False):
        value = cls._get_random_string()
        hasher = hashlib.sha256(value.encode('utf-8')).digest()
        key = base64.b64encode(hasher)
        return key, None
```

Vamos oferecer a opção do usuário em salvar a chave em um arquivo.

Para isso crie as constantes:

```
BASE_DIR = Path(__file__).resolve().parent.parent
KEY_DIR = BASE_DIR / 'keys'
```

Crie o método para salvar a chave

```
@classmethod
def archive_key(cls, key):
    file = 'key.key'
    while Path(cls.KEY_DIR / file).exists():
        file = f'key_{cls._get_random_string(5)}.key'

    with open(cls.KEY_DIR / file, 'wb') as arq:
        arq.write(key)


    return cls.KEY_DIR / file
```

Na criação da chave salve-a:

```
@classmethod
def create_key(cls, archive=False):
    value = cls._get_random_string()
    hasher = hashlib.sha256(value.encode('utf-8')).digest()
    key = base64.b64encode(hasher)
    if archive:
        return key, cls.archive_key(key)
    return key, None
```

Agora vamos trabalhar com métodos de instancia para criptografar e descriptografar as senhas

Primeiro crie o init para instanciar o Fernet

```
from cryptography.fernet import Fernet, InvalidToken
from typing import Union

def __init__(self, key: Union[Path, str]):
    if not isinstance(key, bytes):
        key = key.encode()
```

```
    self.fernet = Fernet(key)
```

Agora crie o método para criptografar a senha:

```
def encrypt(self, value):
    if not isinstance(value, bytes):
        value = value.encode('utf-8')
    return self.fernet.encrypt(value)
```

E por fim, crie o método para descriptografar:

```
def decrypt(self, value):
    if not isinstance(value, bytes):
        value = value.encode('utf-8')

    try:
        return self.fernet.decrypt(value).decode()
    except InvalidToken as e:
        return 'Token inválido'
```

## ▼ Banco de dados

Em model/passwords.py crie uma model para representar uma “tabela no banco de dados”

```
from datetime import datetime

class Password:
    def __init__(self, domain=None, password=None, expire=False):
        self.domain = domain
        self.password = password
        self.create_at = datetime.now().isoformat()
        self.expire = 1 if expire else 0
```

Agora vamos criar uma classe para servir de base a todas as classes que representam tabelas em banco de dados para criar os métodos de save e get.

```
class BaseModel:
    BASE_DIR = Path(__file__).resolve().parent.parent
    DB_DIR = BASE_DIR / 'db'

    def save(self):
        table_path = Path(self.DB_DIR / f'{self.__class__.__name__}.txt')
        if not table_path.exists():
            table_path.touch()

        with open(table_path, 'a') as arq:
            arq.write("|".join(list(map(str, self.__dict__.values()))))
            arq.write('\n')
```

Herde BaseModel em Password.

Crie agora o método get para buscar os dados do banco:

```
@classmethod
def get(cls):
    table_path = Path(cls.DB_DIR / f'{cls.__name__}.txt')
    if not table_path.exists():
        table_path.touch()

    with open(table_path, 'r') as arq:
        x = arq.readlines()

    results = []

    atributos = vars(cls()).keys()

    for i in x:
        split_v = i.split('|')
        tmp_dict = dict(zip(atributos, split_v))
        results.append(tmp_dict)

    return results
```

## ▼ Interagindo com usuário

Permita a importação de todas as pastas:

```
import sys
import os
sys.path.append(os.path.abspath(os.curdir))
```

Crie toda a comunicação com o usuário:

```
import __init__
from model.passwords import Password
from view.passwords_view import FernetHasher
```

```
if __name__ == '__main__':
    action = input('Digite 1 salvar uma nova senha ou 2 para ver uma determinada senha: ')
    match action:
        case '1':
            if len(Password.get()) == 0:
                key, path = FernetHasher.create_key(archive=True)
                print('Sua chave foi criada, salve-a com cuidado caso a perca nao poderá recuperar suas senhas.')
                print(f'Chave: {key.decode('utf-8')}')
                if path:
                    print('Chave salva no arquivo, lembre-se de remover o arquivo após o transferir de local')
                    print(f'Caminho: {path}')
            else:
                key = input('Digite sua chave usada para criptografia, use sempre a mesma chave: ')

            domain = input('Domínio: ')
            password = input('Digite a senha: ')
            fernet = FernetHasher(key)
            p1 = Password(domain=domain, password=fernet.encrypt(password).decode('utf-8'))
            p1.save()

        case '2':
            domain = input('Domínio: ')
            key = input('Key: ')
            fernet = FernetHasher(key)
            data = Password.get()
            password = ''
            for i in data:
                if domain in i['domain']:
                    password = fernet.decrypt(i['password'])

            if password:
                print(f'Sua senha: {password}')
            else:
                print('Nenhuma senha encontrada para esse domínio.')
```