Отчет по лабораторной работе №6 по диспиплине “Парадигмы и конструкции языков программирования”

base/base\_models.py

from tortoise import models, fields

class BaseModel(models.Model):

uuid = fields.UUIDField(*unique*=True, *pk*=True)

async def to\_dict(self):

d = {}

for field in self.\_meta.db\_fields:

d[field] = getattr(self, field)

for field in self.\_meta.backward\_fk\_fields:

d[field] = await getattr(self, field).all().values()

return d

class Meta:

abstract = True

bot/menu.py

from telebot.types import InlineKeyboardButton, InlineKeyboardMarkup, ReplyKeyboardMarkup, KeyboardButton, WebAppInfo

from pydantic import UUID4

from app.db.models import User, Contour, Watermark, WatermarkPicture

from app.bot import texts

def call\_start\_menu() -> InlineKeyboardMarkup:

keyboard = InlineKeyboardMarkup()

keyboard.add(InlineKeyboardButton(*text*="Обновить", *callback\_data*="/start"))

return keyboard

def main\_menu() -> ReplyKeyboardMarkup:

keyboard = ReplyKeyboardMarkup(*resize\_keyboard*=True)

*# marking\_app = WebAppInfo("https://tgweb.cherry4xo.ru", )*

*# branding\_app = WebAppInfo("https://t.me/cherry4xo\_round\_bot/rounded")*

keyboard.add(KeyboardButton(*text*=texts.upload\_video))

keyboard.add(KeyboardButton(*text*=texts.branding))

keyboard.add(KeyboardButton(*text*=texts.video\_marking)) *# , web\_app=marking\_app*

*# keyboard.add(KeyboardButton(text=texts.video\_labeling, web\_app=marking\_app))*

keyboard.add(KeyboardButton(*text*=texts.contest))

keyboard.add(KeyboardButton(*text*=texts.payment))

*# keyboard.add(KeyboardButton(text=texts.check\_payment))*

keyboard.add(KeyboardButton(*text*=texts.technical\_support))

return keyboard

def marking\_menu() -> ReplyKeyboardMarkup:

keyboard = ReplyKeyboardMarkup(*resize\_keyboard*=True)

marking\_app = WebAppInfo("https://arunov-round.ru/?mode=contour")

keyboard.add(KeyboardButton(*text*=texts.create\_sample, *web\_app*=marking\_app))

keyboard.add(KeyboardButton(*text*=texts.delete\_sample\_marking))

keyboard.add(KeyboardButton(*text*=texts.create\_marked\_video))

keyboard.add(KeyboardButton(*text*=texts.back\_to\_main\_menu))

return keyboard

def branding\_menu() -> ReplyKeyboardMarkup:

keyboard = ReplyKeyboardMarkup(*resize\_keyboard*=True)

branding\_app = WebAppInfo("https://arunov-round.ru/?mode=text")

keyboard.add(KeyboardButton(*text*=texts.create\_sample, *web\_app*=branding\_app))

keyboard.add(KeyboardButton(*text*=texts.delete\_sample\_branding))

keyboard.add(KeyboardButton(*text*=texts.create\_branded\_video))

keyboard.add(KeyboardButton(*text*=texts.back\_to\_main\_menu))

return keyboard

async def delete\_marking\_sample\_menu(to\_user: User) -> InlineKeyboardMarkup:

keyboard = InlineKeyboardMarkup()

saved\_samples = await Contour.filter(*user*=to\_user)

for sample in saved\_samples:

keyboard.add(InlineKeyboardButton(*text*=sample.title, *callback\_data*=f"del\_m\_{sample.uuid}"))

return keyboard

async def delete\_branding\_sample\_menu(to\_user: User) -> InlineKeyboardMarkup:

keyboard = InlineKeyboardMarkup()

saved\_samples = await Watermark.filter(*user*=to\_user)

for sample in saved\_samples:

keyboard.add(InlineKeyboardButton(*text*=sample.title, *callback\_data*=f"del\_b\_{sample.uuid}"))

return keyboard

async def delete\_marking\_sample\_confirm\_menu(id: UUID4) -> InlineKeyboardMarkup:

keyboard = InlineKeyboardMarkup()

sample = await Contour.get\_by\_id(*id*=id)

keyboard.add(InlineKeyboardButton(*text*=texts.delete\_sample\_menu\_confirm(*sample\_name*=sample.title), *callback\_data*=f"conf\_del\_m\_{sample.uuid}"))

keyboard.add(InlineKeyboardButton(*text*=texts.back\_to\_main\_menu, *callback\_data*="del\_marking\_menu"))

return keyboard

async def delete\_branding\_sample\_confirm\_menu(id: UUID4) -> InlineKeyboardMarkup:

keyboard = InlineKeyboardMarkup()

sample = await Watermark.get\_by\_id(*id*=id)

keyboard.add(InlineKeyboardButton(*text*=texts.delete\_sample\_menu\_confirm(*sample\_name*=sample.title), *callback\_data*=f"conf\_del\_b\_{sample.uuid}"))

keyboard.add(InlineKeyboardButton(*text*=texts.back\_to\_main\_menu, *callback\_data*="del\_branding\_menu"))

return keyboard

*# def marking\_menu() -> InlineKeyboardMarkup:*

*# keyboard = InlineKeyboardMarkup()*

*# marking\_app = WebAppInfo("https://tgweb.cherry4xo.ru")*

*# keyboard.add(InlineKeyboardButton(text=texts.create\_sample))*

*# keyboard.add(InlineKeyboardButton(text=texts.round\_samples, callback\_data=f"round\_samples"))*

*# # keyboard.add(InlineKeyboardButton(text=texts.back\_to\_samples\_menu, callback\_data=f"back\_samples"))*

*# return keyboard*

*# def branding\_menu() -> InlineKeyboardMarkup:*

*# keyboard = InlineKeyboardMarkup()*

*# marking\_app = WebAppInfo("https://tgweb.cherry4xo.ru")*

*# keyboard.add(InlineKeyboardButton(text=texts.create\_sample))*

*# keyboard.add(InlineKeyboardButton(text=texts.samples\_text, callback\_data=f"watermark\_samples"))*

*# return keyboard*

async def round\_samples\_menu(to\_user: User) -> InlineKeyboardMarkup:

keyboard = InlineKeyboardMarkup()

saved\_samples = await Contour.filter(*user*=to\_user)

for sample in saved\_samples:

keyboard.add(InlineKeyboardButton(*text*=sample.title, *callback\_data*=f"round\_sample\_{sample.uuid}"))

*# keyboard.add(InlineKeyboardButton(text=texts.back\_to\_samples\_menu, callback\_data=f"back\_round\_samples"))*

return keyboard

async def watermark\_pictures\_menu(to\_user: User) -> InlineKeyboardMarkup:

keyboard = InlineKeyboardMarkup()

saved\_pictures = await WatermarkPicture.filter(*user*=to\_user)

for picture in saved\_pictures:

keyboard.add(InlineKeyboardButton(*text*=picture.title, *callback\_data*=f"watermark\_picture\_{str(picture.id)}"))

keyboard.add(InlineKeyboardButton(*text*=texts.add\_new\_watermark, *callback\_data*=f"watermark\_new\_picture"))

keyboard.add(InlineKeyboardButton(*text*=texts.delete\_picture, *callback\_data*=f"delete\_picture"))

*# keyboard.add(InlineKeyboardButton(text=texts.back\_to\_samples\_menu, callback\_data=f"back\_watermark\_samples"))*

return keyboard

async def delete\_pictures\_menu(to\_user: User) -> InlineKeyboardMarkup:

keyboard = InlineKeyboardMarkup()

saved\_pictures = await WatermarkPicture.filter(*user*=to\_user)

for picture in saved\_pictures:

keyboard.add(InlineKeyboardButton(*text*=picture.title, *callback\_data*=f"del\_p\_{str(picture.id)}"))

keyboard.add(InlineKeyboardButton(*text*=texts.back\_to\_samples\_menu, *callback\_data*=f"pictures\_menu"))

return keyboard

async def delete\_pictures\_confirm\_menu(picture\_id: id) -> InlineKeyboardMarkup:

keyboard = InlineKeyboardMarkup()

picture = await WatermarkPicture.get\_by\_id(*id*=picture\_id)

keyboard.add(InlineKeyboardButton(*text*=texts.delete\_picture\_menu\_confirm(*picture\_name*=picture.title), *callback\_data*=f"conf\_del\_p\_{picture.id}"))

keyboard.add(InlineKeyboardButton(*text*=texts.back\_to\_main\_menu, *callback\_data*="delete\_picture"))

return keyboard

async def watermark\_samples\_menu(to\_user: User, picture\_id: str) -> InlineKeyboardMarkup:

keyboard = InlineKeyboardMarkup()

saved\_samples = await Watermark.filter(*user*=to\_user)

for sample in saved\_samples:

keyboard.add(InlineKeyboardButton(*text*=sample.title, *callback\_data*=f"w\_s\_{sample.uuid}\_{picture\_id}"))

keyboard.add(InlineKeyboardButton(*text*=texts.back\_to\_samples\_menu, *callback\_data*=f"back\_watermark\_pictures"))

return keyboard

def tariff() -> InlineKeyboardMarkup:

keyboard = InlineKeyboardMarkup()

keyboard.add(InlineKeyboardButton(*text*=texts.price(10, 250), *callback\_data*=f"tariff\_10\_250"))

keyboard.add(InlineKeyboardButton(*text*=texts.price(50, 500), *callback\_data*=f"tariff\_50\_500"))

keyboard.add(InlineKeyboardButton(*text*=texts.price(100, 1350), *callback\_data*=f"tariff\_100\_1350"))

keyboard.add(InlineKeyboardButton(*text*=texts.price(150, 1500), *callback\_data*=f"tariff\_150\_1500"))

keyboard.add(InlineKeyboardButton(*text*=texts.check\_payment, *callback\_data*="check\_payment"))

return keyboard

bot/texts.py

def price(count: int, value: int) -> str:

return f"{count} кружочков за {value} рублей"

def value(count: int) -> str:

return f"{count} кружочков"

def buy\_rounds(count: int, value: int) -> str:

return f"Вы покупаете {count} кружочков за {value} рублей"

def got\_payment(count: int, total: int) -> str:

return f"Вы успешно приобрели подписку на {count} кружочков!\nВсего на вашем счету {total} кружочков"

def payment\_answer(count: int) -> str:

if count == -1:

return f"У вас полный доступ. Вы точно хотите приобрести тариф?"

if count == -2:

return f"У вас безлимит на кружочки. Вы точно хотите приобрести тариф?"

if count > 0:

return f"У вас осталось кружочков: {count}. Укажите тариф для пополнения"

else:

return "Вы не можете создавать кружочки. Приобретите подписку для использования бота."

def user\_not\_exist\_alert(id: int) -> str:

return f"Пользователь {id} не использует этого бота!"

def user\_unlimited(id: int) -> str:

return f"Пользователю {id} выдан доступ!"

def user\_limited(id: int) -> str:

return f"У пользователя {id} больше нет доступа!"

def user\_set\_sub(id: int) -> str:

return f"Пользователю {id} выдана годовая подписка!"

def user\_cancel\_sub(id: int) -> str:

return f"Пользователю {id} отменена годовая подписка!"

def users\_count(count: int) -> str:

return f"Количество пользователей: {count}"

def rounds\_left(count: int) -> str:

return f"Принял! Сейчас закруглю!\nОсталось кружочков: {count}"

def contour\_saved(title: str) -> str:

return f"Шаблон с названием {title} сохранен"

def watermark\_sample\_saved(title: str) -> str:

return f"Шаблон с названием {title} сохранен"

def payment\_url\_and\_test(url: str, count: int, total: int) -> str:

return f"Перейдите по данной ссылке для оплаты покупки {count} кружочков за {total} рублей\n{url}"

def delete\_sample\_confirm(sample\_name: str) -> str:

return f"Вы точно хотите удалить шаблон \"{sample\_name}\"?"

def delete\_sample\_menu\_confirm(sample\_name: str) -> str:

return f"Да, удалить шаблон \"{sample\_name}\""

def deleted\_sample(sample\_name: str) -> str:

return f"Шаблон \"{sample\_name}\" удален"

def delete\_picture\_confirm(picture\_name: str) -> str:

return f"Вы точно хотите удалить картинку \"{picture\_name}\"?"

def delete\_picture\_menu\_confirm(picture\_name: str) -> str:

return f"Да, удалить картинку \"{picture\_name}\""

def deleted\_picture(picutre\_name: str) -> str:

return f"Картинка \"{picutre\_name}\" удалена"

sending\_text = """Дорогие друзья!

Я, Александр Арунов - визажист, парфюмерный стилист, основатель и генеральный директор компании [A7 community](https://a7community.ru/).

Рад сообщить, что мы с командой закончили работы по внедрению новых функций в бот [«Саня, закругляйся!»](https://t.me/arunov\_round\_bot):

- Брендирование. Функция с помощью которой можно наложить на видео логотип бренда, идеально подойдет компаниям, которые хотят сделать свой бренд более узнаваемым и запоминающимся.

- Маркировка видео. Теперь бренды и блогеры могут размещать на видео-кружочках маркировочные данные рекламного креатива. Достаточно получить токен в сервисах ОРД и разместить его по кругу на видео.

- Провести розыгрыш. Наш бот может провести до 5 разных видов розыгрышей и поднять активность на канале.

Желаю приятных впечатлений и создания яркого, красивого контента на Вашем канале!"""

upload\_video = "📹 Загрузить видео"

upload\_video\_answer = """Отправь в бот любое видео, которое хочешь получить в формате кружочка.

Для этого необходимо нажать на скрепку расположенную в левой части интерфейса, выбрать видео из галереи телефона, обрезать его по квадрату и отправить боту. Через 10-15 секунд бот пришлет кружочек. Его можно переслать в канал, предварительно выключив во время отправки имя отправителя.

Длительность видео - до 60 сек.

"""

instruction = "📝 Инструкция"

instruction\_answer = """

Инструкция по использованию бота:

https://a7community.ru/bot-sanya-zakruglyaysya

"""

null\_rounds = "Кружочки закончились! Приобретите подписку, чтобы продолжить пользоваться ботом!"

unlimited\_rounds = "Принял! Сейчас закруглю!\nУ вас неограниченное количество кружочков!"

payment = "💵 Оплата"

check\_payment = "Проверить оплату"

payment\_already\_sent = "Вы уже оформили. Произведите оплату, чтобы создать новую"

payment\_not\_sent = "Нет оплат для подтверждения"

give\_bot = "Бот [\"Халява, приди!\"](https://t.me/arunov\_give\_bot) поднимет активность среди аудитории канала и поможет провести пять разных видов розыгрыша."

technical\_support = "🙋🛠 Техподдержка"

technical\_support\_answer = """

Инструкция по использованию бота:

https://a7community.ru/bot-sanya-zakruglyaysya

В случае возникновения вопросов пишите в поддержку: @faq\_support\_bot"""

is\_not\_admin\_alert = "У вас нет доступа к данной команде!"

payment\_error\_message = "Произошла ошибка при оплате. Повторите попытку позже"

back\_to\_main\_menu = "Назад"

main\_menu = "Главное меню"

video\_marking = "✍️ Маркировка видео"

create\_marked\_video = "Создать маркированное видео"

create\_branded\_video = "Создать брендированное видео"

video\_labeling = "Брендирование видео"

contest = "🎲 Провести розыгрыш"

branding = "🖼 Брендирование"

round\_samples = "Шаблоны обводки"

watermark\_samples = "Шаблоны наложения картинки"

back\_to\_samples\_menu = "Назад"

sample\_type = "Выберите тип брендирования видео"

on\_set\_border\_round\_state = "Отлично!\nТеперь отправь мне видео, на которое хочешь наложить обводку"

on\_set\_watermark\_round\_state = "Отлично!\nТеперь отправь мне видео, на которое хочешь наложить картинку"

text\_round\_samples = "Здесь вы можете выбрать шаблон обводки"

text\_watermark\_samples = "Здесь вы можете выбрать шаблон наложения картинки"

text\_pick\_picture = "Выберете картинку, которую хотите наложить"

add\_new\_watermark = "Добавить новую картинку"

delete\_picture = "Удалить картинку"

watermark\_pictures\_menu\_text = "Выберете картинку для наложения или добавьте новую"

watermark\_add\_new\_picture = "Отлично! Теперь отправь мне картинку файлом в формате png и название картинки"

watermark\_added\_picture = "Картинка добавлена! Выберете картинку для наложения или добавьте новую"

none\_capture\_error = "В сообщении ты не отправил название картинки!\nОтправь фото еще раз, написав название"

invalid\_extention\_error = "Ты неправильно отправил фотографию!\nОтправь фото файлом"

marking\_menu = "Здесь вы можете наложить на видео обводку или картинку, а также управлять шаблонами наложения"

create\_sample = "Создать шаблон"

delete\_sample\_marking = "Удаление шаблона маркировки"

delete\_sample\_branding = "Удаление шаблона брендирования"

delete\_sample\_menu\_text = "Выберете шаблон, который хотите удалить"

delete\_picture\_menu\_text = "Выберете картинку, которую хотите удалить"

samples\_text = "Шаблоны"

watermark\_menu\_text = "Выберите шаблон брендирования или создайте новый"

round\_menu\_text = "С помощью этой функции можно разместить на кружочке маркировочные данные рекламного креатива или любой текст как показано на кружочках выше.\nВыберете шаблон маркировки видео или создайте новый"

cancelled\_payment = "Платеж был отменен"

pending\_payment = "Платеж еще не был произведен"

expired\_payment = "Время платежа истекло"

db/init\_db.py

import os

from asyncio import sleep

from tortoise import Tortoise

from yookassa import Configuration

from app.db.models import User

from app.redis.database import ping\_redis\_connection, r

from app.db\_migrator.migrate import migrate\_users

from app.settings.config import settings

def get\_app\_list():

app\_list = [f"{settings.APPLICATIONS\_MODULE}.{app}.models" for app in settings.APPLICATIONS]

return app\_list

async def init(db\_url: str | None = None):

await Tortoise.init(

*db\_url*=db\_url or settings.DB\_URL,

*modules*={"models": get\_app\_list()}

)

await Tortoise.generate\_schemas()

print(f"Connected to DB")

await ping\_redis\_connection(r)

if not os.path.exists("app/data"):

os.mkdir("app/data")

Configuration.configure(

*account\_id*=settings.SHOP\_ID,

*secret\_key*=settings.YOO\_SECRET

)

*# await migrate\_users()*

async def create\_default\_admin\_user():

await sleep(3)

user = await User.get\_by\_tg\_id(*tg\_id*=settings.DEFAULT\_ADMIN\_TG\_ID)

if user and user.is\_admin:

return

if not user:

user = User()

user.username = settings.DEFAULT\_ADMIN\_USERNAME

user.first\_name = settings.DEFAULT\_ADMIN\_FIRST\_NAME

user.tg\_id = settings.DEFAULT\_ADMIN\_TG\_ID

user.is\_admin = True

await user.save()

return user

db/models.py

from typing import Optional

from tortoise import fields, models

from tortoise.exceptions import DoesNotExist

from pydantic import UUID4

from app.base.base\_models import BaseModel

from app.db.schemas import BaseUserCreate, BaseContourCreate, BaseWatermarkCreate, BaseWatermarkPictureCreate

class User(BaseModel):

tg\_id = fields.BigIntField()

username = fields.CharField(*max\_length*=128, *null*=True)

first\_name = fields.CharField(*max\_length*=128, *null*=True)

rounds = fields.IntField(*default*=10)

is\_admin = fields.BooleanField(*default*=False)

full\_access = fields.BooleanField(*default*=False)

unlimited\_time = fields.DateField(*null*=True)

@classmethod

async def get\_by\_tg\_id(cls, tg\_id: int) -> Optional["User"]:

try:

query = cls.get\_or\_none(*tg\_id*=tg\_id)

user = await query

return user

except DoesNotExist:

return None

@classmethod

async def get\_by\_username(cls, username: int) -> Optional["User"]:

try:

query = cls.get\_or\_none(*username*=username)

user = await query

return user

except DoesNotExist:

return None

@classmethod

async def create(cls, user: BaseUserCreate) -> "User":

user\_dict = user.model\_dump()

model = cls(\*\*user\_dict)

await model.save()

return model

class Meta:

table = "users"

class Contour(BaseModel):

user: fields.ForeignKeyRelation["User"] = fields.ForeignKeyField(

"models.User", *related\_name*="saved\_contours", *to\_field*="uuid", *on\_delete*=fields.CASCADE

)

title = fields.CharField(*max\_length*=128)

text = fields.CharField(*max\_length*=128)

font\_text = fields.CharField(*max\_length*=64)

font\_size = fields.IntField()

font\_weight = fields.IntField()

text\_color = fields.CharField(*max\_length*=20)

border = fields.IntField()

border\_color = fields.CharField(*max\_length*=20)

opacity = fields.FloatField()

angle = fields.IntField()

@classmethod

async def get\_by\_id(cls, id: UUID4) -> Optional["Contour"]:

try:

query = cls.get\_or\_none(*uuid*=id)

contour = await query

return contour

except DoesNotExist:

return None

@classmethod

async def create(cls, contour: BaseContourCreate, user: User) -> "Contour":

contour\_dict = contour.model\_dump()

model = cls(\*\*contour\_dict, *user*=user)

await model.save()

return model

class Meta:

table = "contours"

class Watermark(BaseModel):

user: fields.ForeignKeyRelation["User"] = fields.ForeignKeyField(

"models.User", *related\_name*="saved\_watermarks", *to\_field*="uuid", *on\_delete*=fields.CASCADE

)

title = fields.CharField(*max\_length*=128)

opacity = fields.FloatField()

offsetY = fields.IntField()

offsetX = fields.IntField()

@classmethod

async def get\_by\_id(cls, id: UUID4) -> Optional["Watermark"]:

try:

query = cls.get\_or\_none(*uuid*=id)

watermark = await query

return watermark

except DoesNotExist:

return None

@classmethod

async def create(cls, watermark\_in: BaseWatermarkCreate, user: User) -> "Watermark":

watermark\_db = watermark\_in.model\_dump()

model = cls(\*\*watermark\_db, *user*=user)

await model.save()

return model

class Meta:

table = "watermarks"

class WatermarkPicture(models.Model):

id = fields.IntField(*pk*=True, *unique*=True)

user: fields.ForeignKeyRelation["User"] = fields.ForeignKeyField(

"models.User", *related\_name*="saved\_watermark\_pictures", *to\_field*="uuid", *on\_delete*=fields.CASCADE

)

file\_path = fields.CharField(*max\_length*=256)

title = fields.CharField(*max\_length*=64, *null*=True)

@classmethod

async def get\_by\_id(cls, id: int) -> Optional["WatermarkPicture"]:

try:

query = cls.get\_or\_none(*id*=id)

picture = await query

return picture

except DoesNotExist:

return None

@classmethod

async def create(cls, picture\_in: BaseWatermarkPictureCreate, user: User) -> "WatermarkPicture":

picture\_db = picture\_in.model\_dump()

model = cls(\*\*picture\_db, *user*=user)

await model.save()

return model

class Meta:

table = "watermark\_pictures"

db/schemas.py

import uuid

from typing import Optional

from pydantic import BaseModel, validator, UUID4

class BaseProperties(BaseModel):

@validator("uuid", *pre*=True, *always*=True, *check\_fields*=False)

def default\_hashed\_id(cls, v):

return v or uuid.uuid4()

class BaseUserCreate(BaseProperties):

tg\_id: int

username: Optional[str] = None

first\_name: Optional[str] = None

class Config:

from\_attributes = True

class BaseContourCreate(BaseProperties):

title: str

text: str

font\_text: str

font\_size: int

font\_weight: int

text\_color: str

border: int

border\_color: str

opacity: float

angle: float

class Config:

from\_attributes = True

class BaseWatermarkCreate(BaseProperties):

title: str

opacity: float

offsetY: int

offsetX: int

class Config:

from\_attributes = True

class BaseWatermarkPictureCreate(BaseProperties):

title: Optional[str] = None

file\_path: str

class Config:

from\_attributes = True

bot/handler.py

import os

import var\_dump

import json

import telebot

import uuid

from datetime import date

from yookassa import Payment

from PIL import ImageColor

from telebot.async\_telebot import AsyncTeleBot

from telebot.asyncio\_storage import StateMemoryStorage

from telebot.asyncio\_handler\_backends import StatesGroup, State

from telebot.types import LabeledPrice

from app.db.models import User, Contour, Watermark, WatermarkPicture

from app.db.schemas import BaseUserCreate, BaseContourCreate, BaseWatermarkCreate, BaseWatermarkPictureCreate

from app.bot import texts

from app.bot import menu

from app.redis.database import r

from app.settings.config import settings

state\_storage = StateMemoryStorage()

bot = AsyncTeleBot(settings.TELEGRAM\_BOT\_TOKEN, *state\_storage*=state\_storage)

class States(StatesGroup):

default\_round = State()

border\_round = State()

watermark\_round = State()

select\_watermark\_image = State()

add\_watermark\_image = State()

*# welcome\_video = open("app/bot/answer\_data/video1.mp4", "rb")*

*# marking\_video\_1 = open("app/bot/answer\_data/marking\_sample1.mp4", "rb")*

*# marking\_video\_2 = open("app/bot/answer\_data/marking\_sample2.mp4", "rb")*

@bot.message\_handler(*commands*=["make\_sending"])

async def make\_sending(message: telebot.types.Message):

uid = message.from\_user.id

user = await User.get\_by\_tg\_id(*tg\_id*=uid)

if not user.is\_admin:

return await bot.send\_message(message.chat.id, *text*=texts.is\_not\_admin\_alert)

users = await User.all()

for user in users:

try:

await bot.send\_photo(*chat\_id*=user.tg\_id,

*photo*=open("app/bot/answer\_data/arunov.jpg", "rb"),

*caption*=texts.sending\_text,

*reply\_markup*=menu.call\_start\_menu(),

*parse\_mode*="markdown")

except:

continue

@bot.callback\_query\_handler(*func*=lambda call: call.data.startswith("/start"))

async def welcome(call: telebot.types.CallbackQuery):

await bot.set\_state(call.message.from\_user.id, States.default\_round, call.message.chat.id)

return await bot.send\_video\_note(*chat\_id*=call.message.chat.id,

*data*=open("app/bot/answer\_data/video1.mp4", "rb"),

*reply\_markup*=menu.main\_menu())

@bot.message\_handler(*commands*=["start"])

async def welcome(message: telebot.types.Message):

uid = int(message.from\_user.id)

user = await User.get\_by\_tg\_id(*tg\_id*=uid)

if not user:

username = message.from\_user.username

first\_name = message.from\_user.first\_name

user = await User.create(BaseUserCreate(*tg\_id*=uid, *first\_name*=first\_name, *username*=username))

if user.username != message.from\_user.username:

user.username = message.from\_user.username

if user.first\_name != message.from\_user.first\_name:

user.first\_name = message.from\_user.first\_name

await bot.set\_state(message.from\_user.id, States.default\_round, message.chat.id)

return await bot.send\_video\_note(*chat\_id*=message.chat.id,

*data*=open("app/bot/answer\_data/video1.mp4", "rb"),

*reply\_markup*=menu.main\_menu())

@bot.message\_handler(*content\_types*=["video"])

async def upload\_video(message: telebot.types.Message):

uid = message.from\_user.id

user = await User.get\_by\_tg\_id(*tg\_id*=uid)

if not user:

username = message.from\_user.username

first\_name = message.from\_user.first\_name

user = await User.create(BaseUserCreate(*tg\_id*=uid, *first\_name*=first\_name, *username*=username))

if user.full\_access:

await bot.send\_message(*chat\_id*=message.chat.id, *text*=texts.unlimited\_rounds)

elif user.unlimited\_time is not None:

if user.unlimited\_time > date.today():

await bot.send\_message(*chat\_id*=message.chat.id, *text*=texts.unlimited\_rounds)

elif user.rounds > 0:

user.rounds -= 1

await user.save()

await bot.send\_message(*chat\_id*=message.chat.id, *text*=texts.rounds\_left(user.rounds))

else:

return await bot.send\_message(*chat\_id*=message.chat.id, *text*=texts.null\_rounds)

file\_id = message.video.file\_id

file\_info = await bot.get\_file(message.video.file\_id)

downloaded\_file = await bot.download\_file(file\_info.file\_path)

new\_file\_path = f"app/data/{file\_id}.mp4"

with open(new\_file\_path, "wb") as file:

file.write(downloaded\_file)

request\_id = str(uuid.uuid4())

if await bot.current\_states.get\_state(message.chat.id, message.from\_user.id) == "States:default\_round":

state\_r = "default"

elif await bot.current\_states.get\_state(message.chat.id, message.from\_user.id) == "States:border\_round":

state\_r = "border"

elif await bot.current\_states.get\_state(message.chat.id, message.from\_user.id) == "States:watermark\_round":

state\_r = "watermark"

else:

state\_r = "default"

data = {

"id": request\_id,

"type": state\_r,

"user\_id": user.tg\_id,

"file\_path": f"{new\_file\_path}"

}

data\_json = json.dumps(data)

await r.lpush("request", data\_json)

@bot.message\_handler(*content\_types*=["web\_app\_data"])

async def answer(webAppMes: telebot.types.Message):

data = json.loads(webAppMes.web\_app\_data.data)

if data["type"] == "contour":

user = await User.get\_by\_tg\_id(*tg\_id*=webAppMes.from\_user.id)

data["border\_color"] = data["border\_color"][1:]

border\_color\_rgb = tuple(int(data["border\_color"][i:i+2], 16) for i in (0, 2, 4))

data["border\_color"] = f"rgb({border\_color\_rgb[0]}, {border\_color\_rgb[1]}, {border\_color\_rgb[2]})"

data["text\_color"] = data["text\_color"][1:]

text\_color\_rgb = tuple(int(data["text\_color"][i:i+2], 16) for i in (0, 2, 4))

data["text\_color"] = f"rgb({text\_color\_rgb[0]}, {text\_color\_rgb[1]}, {text\_color\_rgb[2]})"

data["angle"] = int(data["angle"]) + 90

contour\_db = BaseContourCreate(\*\*data)

contour = await Contour.create(*contour*=contour\_db, *user*=user)

await bot.send\_message(*chat\_id*=webAppMes.chat.id, *text*=texts.contour\_saved(*title*=contour\_db.title))

if data["type"] == "text":

user = await User.get\_by\_tg\_id(*tg\_id*=webAppMes.from\_user.id)

watermark\_db = BaseWatermarkCreate(\*\*data)

watermark = await Watermark.create(*watermark\_in*=watermark\_db, *user*=user)

await bot.send\_message(*chat\_id*=webAppMes.chat.id, *text*=texts.contour\_saved(*title*=watermark\_db.title))

@bot.message\_handler(*func*=lambda x: x.text == texts.branding)

async def saved\_samples(message: telebot.types.Message):

uid = message.from\_user.id

user = await User.get\_by\_tg\_id(*tg\_id*=uid)

await bot.send\_video\_note(*chat\_id*=message.chat.id,

*data*=open("app/bot/answer\_data/arunov\_branding.mp4", "rb"))

await bot.send\_message(*chat\_id*=message.chat.id,

*text*=texts.watermark\_menu\_text,

*reply\_markup*=menu.branding\_menu())

@bot.message\_handler(*func*=lambda x: x.text == texts.video\_marking)

async def get\_marking\_menu(message: telebot.types.Message):

await bot.send\_video\_note(*chat\_id*=message.chat.id,

*data*=open("app/bot/answer\_data/arunov\_marking1.mp4", "rb"))

await bot.send\_video\_note(*chat\_id*=message.chat.id,

*data*=open("app/bot/answer\_data/arunov\_marking2.mp4", "rb"))

await bot.send\_message(*chat\_id*=message.chat.id,

*text*=texts.round\_menu\_text,

*reply\_markup*=menu.marking\_menu())

*# reply\_markup=await menu.round\_samples\_menu(to\_user=user))*

@bot.message\_handler(*func*=lambda x: x.text == texts.back\_to\_main\_menu)

async def get\_main\_menu(message: telebot.types.Message):

await bot.send\_message(*chat\_id*=message.chat.id,

*text*=texts.main\_menu,

*reply\_markup*=menu.main\_menu())

@bot.message\_handler(*func*=lambda x: x.text == texts.create\_marked\_video)

async def get\_saved\_samples(message: telebot.types.Message):

uid = message.from\_user.id

user = await User.get\_by\_tg\_id(*tg\_id*=uid)

await bot.send\_message(*text*=texts.text\_round\_samples,

*chat\_id*=message.chat.id,

*reply\_markup*=await menu.round\_samples\_menu(user))

@bot.message\_handler(*func*=lambda x: x.text == texts.delete\_sample\_marking)

async def delete\_marking\_sample(message: telebot.types.Message):

uid = message.from\_user.id

user = await User.get\_by\_tg\_id(*tg\_id*=uid)

await bot.send\_message(*text*=texts.delete\_sample\_menu\_text,

*chat\_id*=message.chat.id,

*reply\_markup*=await menu.delete\_marking\_sample\_menu(*to\_user*=user))

@bot.callback\_query\_handler(*func*=lambda call: call.data.startswith("del\_marking\_menu"))

async def delete\_marking\_sample(call: telebot.types.CallbackQuery):

uid = call.from\_user.id

user = await User.get\_by\_tg\_id(*tg\_id*=uid)

await bot.edit\_message\_text(*text*=texts.delete\_sample\_menu\_text,

*message\_id*=call.message.id,

*chat\_id*=call.message.chat.id,

*reply\_markup*=await menu.delete\_marking\_sample\_menu(*to\_user*=user))

@bot.callback\_query\_handler(*func*=lambda call: call.data.startswith("del\_m\_"))

async def delete\_marking\_sample\_confirm(call: telebot.types.CallbackQuery):

sample\_id = call.data.split("\_")[-1]

sample = await Contour.get\_by\_id(*id*=sample\_id)

await bot.edit\_message\_text(*text*=texts.delete\_sample\_confirm(*sample\_name*=sample.title),

*message\_id*=call.message.id,

*chat\_id*=call.message.chat.id,

*reply\_markup*=await menu.delete\_marking\_sample\_confirm\_menu(*id*=sample\_id))

@bot.callback\_query\_handler(*func*=lambda call: call.data.startswith("conf\_del\_m\_"))

async def deleted\_marking\_sample(call: telebot.types.CallbackQuery):

uid = call.from\_user.id

user = await User.get\_by\_tg\_id(*tg\_id*=uid)

sample\_id = call.data.split("\_")[-1]

sample = await Contour.get\_by\_id(*id*=sample\_id)

await sample.delete()

await bot.edit\_message\_text(*text*=texts.deleted\_sample(*sample\_name*=sample.title),

*message\_id*=call.message.id,

*chat\_id*=call.message.chat.id)

await bot.send\_message(*text*=texts.delete\_sample\_menu\_text,

*chat\_id*=call.message.chat.id,

*reply\_markup*=await menu.delete\_marking\_sample\_menu(*to\_user*=user))

@bot.message\_handler(*func*=lambda x: x.text == texts.delete\_sample\_branding)

async def delete\_marking\_branding(message: telebot.types.Message):

uid = message.from\_user.id

user = await User.get\_by\_tg\_id(*tg\_id*=uid)

await bot.send\_message(*text*=texts.delete\_sample\_menu\_text,

*chat\_id*=message.chat.id,

*reply\_markup*=await menu.delete\_branding\_sample\_menu(*to\_user*=user))

@bot.callback\_query\_handler(*func*=lambda call: call.data.startswith("del\_branding\_menu"))

async def delete\_marking\_branding(call: telebot.types.CallbackQuery):

uid = call.from\_user.id

user = await User.get\_by\_tg\_id(*tg\_id*=uid)

await bot.edit\_message\_text(*text*=texts.delete\_sample\_menu\_text,

*message\_id*=call.message.id,

*chat\_id*=call.message.chat.id,

*reply\_markup*=await menu.delete\_branding\_sample\_menu(*to\_user*=user))

@bot.callback\_query\_handler(*func*=lambda call: call.data.startswith("del\_b\_"))

async def delete\_marking\_sample\_confirm(call: telebot.types.CallbackQuery):

sample\_id = call.data.split("\_")[-1]

sample = await Watermark.get\_by\_id(*id*=sample\_id)

await bot.edit\_message\_text(*text*=texts.delete\_sample\_confirm(*sample\_name*=sample.title),

*message\_id*=call.message.id,

*chat\_id*=call.message.chat.id,

*reply\_markup*=await menu.delete\_branding\_sample\_confirm\_menu(*id*=sample\_id))

@bot.callback\_query\_handler(*func*=lambda call: call.data.startswith("conf\_del\_b\_"))

async def deleted\_marking\_sample(call: telebot.types.CallbackQuery):

uid = call.from\_user.id

user = await User.get\_by\_tg\_id(*tg\_id*=uid)

sample\_id = call.data.split("\_")[-1]

sample = await Watermark.get\_by\_id(*id*=sample\_id)

await sample.delete()

await bot.edit\_message\_text(*text*=texts.deleted\_sample(*sample\_name*=sample.title),

*message\_id*=call.message.id,

*chat\_id*=call.message.chat.id)

await bot.send\_message(*text*=texts.delete\_sample\_menu\_text,

*chat\_id*=call.message.chat.id,

*reply\_markup*=await menu.delete\_branding\_sample\_menu(*to\_user*=user))

@bot.callback\_query\_handler(*func*=lambda call: call.data.startswith("pictures\_menu"))

async def pictures\_menu(call: telebot.types.CallbackQuery):

uid = call.from\_user.id

user = await User.get\_by\_tg\_id(*tg\_id*=uid)

await bot.edit\_message\_text(*text*=texts.watermark\_pictures\_menu\_text,

*chat\_id*=call.message.chat.id,

*message\_id*=call.message.id,

*reply\_markup*=await menu.watermark\_pictures\_menu(user))

@bot.callback\_query\_handler(*func*=lambda call: call.data.startswith("delete\_picture"))

async def delete\_picture(call: telebot.types.CallbackQuery):

uid = call.from\_user.id

user = await User.get\_by\_tg\_id(*tg\_id*=uid)

await bot.edit\_message\_text(*text*=texts.delete\_picture\_menu\_text,

*message\_id*=call.message.id,

*chat\_id*=call.message.chat.id,

*reply\_markup*=await menu.delete\_pictures\_menu(*to\_user*=user))

@bot.callback\_query\_handler(*func*=lambda call: call.data.startswith("del\_p\_"))

async def delete\_picture\_confirm(call: telebot.types.CallbackQuery):

picture\_id = call.data.split("\_")[-1]

picture = await WatermarkPicture.get\_by\_id(*id*=picture\_id)

await bot.edit\_message\_text(*text*=texts.delete\_picture\_confirm(*picture\_name*=picture.title),

*message\_id*=call.message.id,

*chat\_id*=call.message.chat.id,

*reply\_markup*=await menu.delete\_pictures\_confirm\_menu(*picture\_id*=picture.id))

@bot.callback\_query\_handler(*func*=lambda call: call.data.startswith("conf\_del\_p\_"))

async def deleted\_picture(call: telebot.types.CallbackQuery):

uid = call.from\_user.id

user = await User.get\_by\_tg\_id(*tg\_id*=uid)

picture\_id = call.data.split("\_")[-1]

picture = await WatermarkPicture.get\_by\_id(*id*=picture\_id)

await picture.delete()

if os.path.exists(picture.file\_path):

os.remove(picture.file\_path)

await bot.edit\_message\_text(*text*=texts.deleted\_picture(*picutre\_name*=picture.title),

*message\_id*=call.message.id,

*chat\_id*=call.message.chat.id)

await bot.send\_message(*text*=texts.delete\_picture\_menu\_text,

*chat\_id*=call.message.chat.id,

*reply\_markup*=await menu.delete\_pictures\_menu(*to\_user*=user))

@bot.message\_handler(*func*=lambda x: x.text == texts.create\_branded\_video)

async def get\_saved\_branding\_samples(message: telebot.types.Message):

uid = message.from\_user.id

user = await User.get\_by\_tg\_id(*tg\_id*=uid)

await bot.send\_message(*text*=texts.text\_pick\_picture,

*chat\_id*=message.chat.id,

*reply\_markup*=await menu.watermark\_pictures\_menu(user))

@bot.callback\_query\_handler(*func*=lambda call: call.data.startswith("back\_watermark\_pictures"))

async def get\_saved\_watermark\_samples(call: telebot.types.CallbackQuery):

uid = call.from\_user.id

user = await User.get\_by\_tg\_id(*tg\_id*=uid)

await bot.edit\_message\_text(*text*=texts.watermark\_pictures\_menu\_text,

*chat\_id*=call.message.chat.id,

*message\_id*=call.message.id,

*reply\_markup*=await menu.watermark\_pictures\_menu(user))

@bot.callback\_query\_handler(*func*=lambda call: call.data.startswith("watermark\_new\_picture"))

async def add\_new\_watermark\_picture(call: telebot.types.CallbackQuery):

uid = call.from\_user.id

await bot.set\_state(*user\_id*=uid, *state*=States.add\_watermark\_image, *chat\_id*=call.message.chat.id)

await bot.edit\_message\_text(*text*=texts.watermark\_add\_new\_picture,

*chat\_id*=call.message.chat.id,

*message\_id*=call.message.id)

@bot.message\_handler(*content\_types*=["document"], *state*=States.add\_watermark\_image)

async def new\_watermark\_picute\_handler(message: telebot.types.Message):

if message.caption is None:

return await bot.send\_message(*chat\_id*=message.chat.id,

*text*=texts.none\_capture\_error)

uid = message.from\_user.id

user = await User.get\_by\_tg\_id(*tg\_id*=uid)

file\_id = message.document.file\_id

file\_info = await bot.get\_file(file\_id)

if message.document.file\_name.split('.')[-1].lower() != "png":

return await bot.send\_message(*chat\_id*=message.chat.id,

*text*=texts.invalid\_extention\_error,

*reply\_markup*=await menu.watermark\_pictures\_menu(*to\_user*=user))

downloaded\_file = await bot.download\_file(file\_info.file\_path)

new\_file\_path = f"app/data/{file\_id}.png"

with open(new\_file\_path, "wb") as file:

file.write(downloaded\_file)

watermark\_picture\_db = BaseWatermarkPictureCreate(*file\_path*=new\_file\_path, *title*=message.caption)

await WatermarkPicture.create(*picture\_in*=watermark\_picture\_db, *user*=user)

await bot.send\_message(*chat\_id*=message.chat.id,

*text*=texts.watermark\_added\_picture,

*reply\_markup*=await menu.watermark\_pictures\_menu(*to\_user*=user))

await bot.set\_state(*user\_id*=uid, *state*=States.default\_round, *chat\_id*=message.chat.id)

@bot.message\_handler(*content\_types*=["photo"], *state*=States.add\_watermark\_image)

async def photo\_recieve\_notification(message: telebot.types.Message):

await bot.send\_message(*chat\_id*=message.chat.id,

*text*=texts.invalid\_extention\_error)

@bot.callback\_query\_handler(*func*=lambda call: call.data.startswith("watermark\_picture\_"))

async def choose\_watermark\_sample(call: telebot.types.CallbackQuery):

uid = call.from\_user.id

user = await User.get\_by\_tg\_id(*tg\_id*=uid)

picture\_id = "".join(call.data.split("\_")[2:])

await bot.edit\_message\_text(*text*=texts.text\_watermark\_samples,

*chat\_id*=call.message.chat.id,

*message\_id*=call.message.id,

*reply\_markup*=await menu.watermark\_samples\_menu(*to\_user*=user, *picture\_id*=picture\_id))

*# @bot.callback\_query\_handler(func=lambda call: call.data.startswith("back\_watermark\_pictures"))*

*# async def back\_watermark\_pictures(call: telebot.types.CallbackQuery):*

*# uid = call.message.from\_user.id*

*# user = await User.get\_by\_tg\_id(tg\_id=uid)*

*# await bot.edit\_message\_text(chat\_id=call.message.chat.id,*

*# text=texts.watermark\_menu\_text,*

*# message\_id=call.message.id,*

*# reply\_markup=await menu.watermark\_pictures\_menu(to\_user=user))*

*# @bot.callback\_query\_handler(func=lambda call: call.data.startswith("back\_watermark\_samples"))*

*# async def saved\_watermark\_samples(call: telebot.types.CallbackQuery):*

*# uid = call.message.from\_user.id*

*# user = await User.get\_by\_tg\_id(tg\_id=uid)*

*# await bot.edit\_message\_text(chat\_id=call.message.chat.id,*

*# text=texts.watermark\_menu\_text,*

*# message\_id=call.message.id,*

*# reply\_markup=await menu.watermark\_samples\_menu(to\_user=user))*

@bot.callback\_query\_handler(*func*=lambda call: call.data.startswith("back\_round\_samples"))

async def saved\_round\_samples(call: telebot.types.CallbackQuery):

await bot.edit\_message\_text(*chat\_id*=call.message.chat.id,

*text*=texts.round\_menu\_text,

*message\_id*=call.message.id,

*reply\_markup*=menu.round\_samples\_menu())

@bot.callback\_query\_handler(*func*=lambda call: call.data.startswith("round\_sample\_"))

async def set\_state\_round\_sample\_video(call: telebot.types.CallbackQuery):

uid = call.from\_user.id

user = await User.get\_by\_tg\_id(*tg\_id*=uid)

await bot.set\_state(user.tg\_id, States.border\_round, call.message.chat.id)

await bot.edit\_message\_text(*chat\_id*=call.message.chat.id, *text*=texts.on\_set\_border\_round\_state, *message\_id*=call.message.id)

async with r.pipeline(*transaction*=True) as pipe:

await (pipe.hset(

*name*=f"{user.tg\_id}:round\_sample",

*mapping*={

"sample\_id": "".join(call.data.split('\_')[2:]),

}

).execute())

return

@bot.callback\_query\_handler(*func*=lambda call: call.data.startswith("w\_s\_"))

async def set\_state\_watermark\_sample\_video(call: telebot.types.CallbackQuery):

uid = call.from\_user.id

user = await User.get\_by\_tg\_id(*tg\_id*=uid)

await bot.set\_state(user.tg\_id, States.watermark\_round, call.message.chat.id)

await bot.edit\_message\_text(*chat\_id*=call.message.chat.id, *text*=texts.on\_set\_watermark\_round\_state, *message\_id*=call.message.id)

print(await bot.current\_states.get\_state(call.message.chat.id, call.from\_user.id))

async with r.pipeline(*transaction*=True) as pipe:

await (pipe.hset(

*name*=f"{user.tg\_id}:watermark\_sample",

*mapping*={

"sample\_id": "".join(call.data.split('\_')[2:3]),

"picture\_id": "".join(call.data.split('\_')[3:4])

}

).execute())

return

@bot.message\_handler(*commands*=["set\_unlimited\_sub"])

async def set\_full\_access(message: telebot.types.Message):

uid = message.from\_user.id

user = await User.get\_by\_tg\_id(*tg\_id*=uid)

if not user.is\_admin:

return await bot.send\_message(message.chat.id, *text*=texts.is\_not\_admin\_alert)

to\_user\_id = message.text.split(" ")[-1]

to\_user = await User.get\_by\_tg\_id(*tg\_id*=to\_user\_id)

if not to\_user:

return await bot.send\_message(message.chat.id, *text*=texts.user\_not\_exist\_alert(*id*=to\_user\_id))

to\_user.full\_access = True

await to\_user.save()

return await bot.send\_message(message.chat.id, *text*=texts.user\_unlimited(*id*=to\_user\_id))

@bot.message\_handler(*commands*=["cancel\_unlimited\_sub"])

async def cancel\_full\_access(message: telebot.types.Message):

uid = message.from\_user.id

user = await User.get\_by\_tg\_id(*tg\_id*=uid)

if not user.is\_admin:

return await bot.send\_message(message.chat.id, *text*=texts.is\_not\_admin\_alert)

to\_user\_id = message.text.split(" ")[-1]

to\_user = await User.get\_by\_tg\_id(*tg\_id*=to\_user\_id)

if not to\_user:

return await bot.send\_message(message.chat.id, *text*=texts.user\_not\_exist\_alert(*id*=to\_user\_id))

to\_user.full\_access = False

await to\_user.save()

return await bot.send\_message(message.chat.id, *text*=texts.user\_limited(*id*=to\_user\_id))

@bot.message\_handler(*commands*=["set\_year\_sub"])

async def set\_year\_sub(message: telebot.types.Message):

uid = message.from\_user.id

user = await User.get\_by\_tg\_id(*tg\_id*=uid)

if not user.is\_admin:

return await bot.send\_message(message.chat.id, *text*=texts.is\_not\_admin\_alert)

to\_user\_id = message.text.split(" ")[-1]

to\_user = await User.get\_by\_tg\_id(*tg\_id*=to\_user\_id)

if not to\_user:

return await bot.send\_message(message.chat.id, *text*=texts.user\_not\_exist\_alert(*id*=to\_user\_id))

to\_user.unlimited\_time = date(*year*=date.today().year + 1, *month*=date.today().month, *day*=date.today().day)

await to\_user.save()

return await bot.send\_message(message.chat.id, *text*=texts.user\_set\_sub(*id*=to\_user\_id))

@bot.message\_handler(*commands*=["cancel\_year\_sub"])

async def cancel\_year\_sub(message: telebot.types.Message):

uid = message.from\_user.id

user = await User.get\_by\_tg\_id(*tg\_id*=uid)

if not user.is\_admin:

return await bot.send\_message(message.chat.id, *text*=texts.is\_not\_admin\_alert)

to\_user\_id = message.text.split(" ")[-1]

to\_user = await User.get\_by\_tg\_id(*tg\_id*=to\_user\_id)

if not to\_user:

return await bot.send\_message(message.chat.id, *text*=texts.user\_not\_exist\_alert(*id*=to\_user\_id))

to\_user.unlimited\_time = None

await to\_user.save()

return await bot.send\_message(message.chat.id, *text*=texts.user\_cancel\_sub(*id*=to\_user\_id))

@bot.message\_handler(*commands*=["users"])

async def users\_count(message: telebot.types.Message):

uid = message.from\_user.id

user = await User.get\_by\_tg\_id(*tg\_id*=uid)

if not user.is\_admin:

return await bot.send\_message(message.chat.id, *text*=texts.is\_not\_admin\_alert)

count = await User.all().count()

return await bot.send\_message(message.chat.id, *text*=texts.users\_count(count))

@bot.message\_handler(*func*=lambda x: x.text == texts.upload\_video)

async def upload\_video\_message(message: telebot.types.Message):

await bot.set\_state(message.from\_user.id, States.default\_round, message.chat.id)

return await bot.send\_message(message.chat.id, *text*=texts.upload\_video\_answer)

*# @bot.message\_handler(func=lambda x: x.text == texts.instruction)*

*# async def instruction(message: telebot.types.Message):*

*# return await bot.send\_message(message.chat.id, text=texts.instruction\_answer, disable\_web\_page\_preview=True)*

@bot.message\_handler(*func*=lambda x: x.text == texts.payment)

async def payment\_message(message: telebot.types.Message):

uid = message.from\_user.id

user = await User.get\_by\_tg\_id(uid)

if user.full\_access:

rounds = -1

elif user.unlimited\_time is not None:

if user.unlimited\_time > date.today():

rounds = -2

else:

rounds = user.rounds

return await bot.send\_message(message.chat.id, *text*=texts.payment\_answer(rounds), *reply\_markup*=menu.tariff())

@bot.callback\_query\_handler(*func*=lambda call: call.data.startswith("tariff"))

async def tariff(call: telebot.types.CallbackQuery):

uid = call.from\_user.id

user = await User.get\_by\_tg\_id(*tg\_id*=uid)

count, total = call.data.split("\_")[1:3]

*# prices = [LabeledPrice(label=texts.value(count), amount=f"{total}00")]*

*# await bot.send\_invoice(call.message.chat.id,*

*# title=texts.price(count, total),*

*# description=texts.buy\_rounds(count, total),*

*# invoice\_payload=f"{count}",*

*# provider\_token=settings.PROVIDER\_TOKEN,*

*# currency="rub",*

*# prices=prices)*

async with r.pipeline(*transaction*=True) as pipe:

payment = (await (pipe.hgetall(

f"{user.uuid}:payment\_id"

).execute()))[0]

if payment != {}:

await bot.send\_message(

*chat\_id*=call.message.chat.id,

*text*=texts.payment\_already\_sent

)

return

idempotence\_key = str(uuid.uuid4())

res = Payment.create(

{

"amount": {

"value": total,

"currency": "RUB"

},

"confirmation": {

"type": "redirect",

"return\_url": "https://yoomoney.ru/"

},

"capture": True,

"metadata": {

"count": f"{count}"

},

"description": f"{count} кружочков"

},

idempotence\_key

)

payment\_url = res.confirmation.confirmation\_url

await bot.edit\_message\_text(

*text*=texts.payment\_url\_and\_test(*url*=payment\_url, *count*=count, *total*=total),

*chat\_id*=call.message.chat.id,

*message\_id*=call.message.id

)

async with r.pipeline(*transaction*=True) as pipe:

await (pipe.hset(

f"{user.uuid}:payment\_id",

*mapping*={

"id": res.id

}

).execute())

@bot.callback\_query\_handler(*func*=lambda call: call.data.startswith("check\_payment"))

async def check\_payment(call: telebot.types.CallbackQuery):

uid = call.from\_user.id

user = await User.get\_by\_tg\_id(*tg\_id*=uid)

async with r.pipeline(*transaction*=True) as pipe:

payment = (await (pipe.hgetall(

f"{user.uuid}:payment\_id"

).execute()))[0]

if payment == {}:

return await bot.send\_message(

*chat\_id*=call.message.chat.id,

*text*=texts.payment\_not\_sent,

)

confirmed = Payment.find\_one(*payment\_id*=payment["id"])

if confirmed.status == "succeeded":

user.rounds += int(confirmed.metadata["count"])

await user.save()

await (pipe.delete(f"{user.uuid}:payment\_id").execute())

return await bot.send\_message(

*chat\_id*=call.message.chat.id,

*text*=texts.got\_payment(*count*=confirmed.metadata["count"], *total*=user.rounds)

)

elif confirmed.status == "canceled":

await (pipe.delete(f"{user.uuid}:payment\_id").execute())

return await bot.send\_message(

*chat\_id*=call.message.chat.id,

*text*=texts.cancelled\_payment

)

elif confirmed.status == "pending":

return await bot.send\_message(

*chat\_id*=call.message.chat.id,

*text*=texts.pending\_payment

)

else:

await (pipe.delete(f"{user.uuid}:payment\_id").execute())

await bot.send\_message(

*chat\_id*=call.message.chat.id,

*text*=texts.expired\_payment

)

return

@bot.message\_handler(*func*=lambda x: x.text == texts.contest)

async def give\_bot(message: telebot.types.Message):

await bot.send\_message(

*chat\_id*=message.from\_user.id,

*text*=texts.give\_bot,

*parse\_mode*="markdown"

)

@bot.pre\_checkout\_query\_handler(*func*=lambda query: True)

async def checkout(pre\_checkout\_query: telebot.types.PreCheckoutQuery):

await bot.answer\_pre\_checkout\_query(pre\_checkout\_query.id, *ok*=True,

*error\_message*=texts.payment\_error\_message)

@bot.message\_handler(*content\_types*=["successful\_payment"])

async def got\_payment(message: telebot.types.Message):

uid = message.from\_user.id

user = await User.get\_by\_tg\_id(*tg\_id*=uid)

count = message.successful\_payment.invoice\_payload

user.rounds += int(count)

await user.save()

total = user.rounds

return await bot.send\_message(message.chat.id,

*text*=texts.got\_payment(count, total))

@bot.message\_handler(*func*=lambda x: x.text == texts.technical\_support)

async def technical\_support(message: telebot.types.Message):

return await bot.send\_message(message.chat.id, *text*=texts.technical\_support\_answer)

editing/handler.py

import json

import asyncio

import os

import time

from app.redis.database import r

from app.settings.config import settings

from app.bot.handler import bot, States

from app.editing.utils import make\_watermark, make\_rounded, make\_default\_rounded

from app.db.models import Contour, Watermark, WatermarkPicture

async def send\_rounded() -> None:

while True:

try:

await asyncio.sleep(0.25)

request = await r.rpop("request")

if request is None:

continue

data = json.loads(request)

if data["type"] == "default":

make\_default\_rounded(*filename*=data["file\_path"])

filename\_out = "".join(data["file\_path"].split("."))[:-1] + "temp.mp4"

await bot.send\_video\_note(*chat\_id*=data["user\_id"],

*data*=open(filename\_out, "rb"))

await bot.set\_state(*user\_id*=data["user\_id"], *chat\_id*=data["user\_id"], *state*=States.default\_round)

if os.path.exists(data["file\_path"]):

os.remove(data["file\_path"])

if os.path.exists(filename\_out):

os.remove(filename\_out)

elif data["type"] == "border":

async with r.pipeline(*transaction*=True) as pipe:

round\_sample\_data = (await pipe.hgetall(*name*=f"{data['user\_id']}:round\_sample").execute())[0]

await (pipe.delete(f"{data['user\_id']}:round\_sample").execute())

round\_sample\_id = round\_sample\_data["sample\_id"]

sample = await Contour.get\_by\_id(*id*=round\_sample\_id)

make\_rounded(

*filename*=data["file\_path"],

*size*=576,

*text*=sample.text,

*font\_text*=sample.font\_text,

*font\_size*=sample.font\_size,

*font\_weight*=sample.font\_weight,

*text\_color*=sample.text\_color,

*border*=sample.border,

*border\_color*=sample.border\_color,

*border\_opacity*=sample.opacity,

*angle*=sample.angle

)

filename\_out = "".join(data["file\_path"].split("."))[:-1] + "temp.mp4"

await bot.send\_video\_note(*chat\_id*=data["user\_id"],

*data*=open(filename\_out, "rb"))

if os.path.exists(data['file\_path']):

os.remove(data['file\_path'])

if os.path.exists(filename\_out):

os.remove(filename\_out)

await bot.set\_state(*user\_id*=data["user\_id"], *chat\_id*=data["user\_id"], *state*=States.default\_round)

elif data["type"] == "watermark":

async with r.pipeline(*transaction*=True) as pipe:

watermark\_sample\_data = (await pipe.hgetall(*name*=f"{data['user\_id']}:watermark\_sample").execute())[0]

await (pipe.delete(f"{data['user\_id']}:watermark\_sample").execute())

watermark\_sample\_id = watermark\_sample\_data["sample\_id"]

picture\_id = watermark\_sample\_data["picture\_id"]

sample = await Watermark.get\_by\_id(*id*=watermark\_sample\_id)

picture = await WatermarkPicture.get\_by\_id(*id*=picture\_id)

make\_watermark(

*filename*=data["file\_path"],

*opacity*=sample.opacity,

*offsetX*=sample.offsetX,

*offsetY*=sample.offsetY,

*picture\_file\_path*=picture.file\_path

)

filename\_out = "".join(data["file\_path"].split("."))[:-1] + "temp.mp4"

await bot.send\_video\_note(*chat\_id*=data["user\_id"],

*data*=open(filename\_out, "rb"))

if os.path.exists(data['file\_path']):

os.remove(data['file\_path'])

if os.path.exists(filename\_out):

os.remove(filename\_out)

await bot.set\_state(*user\_id*=data["user\_id"], *chat\_id*=data["user\_id"], *state*=States.default\_round)

except Exception as e:

delay = 3

text = f'Error: {e}, restarting after {delay} seconds'

print(text)

time.sleep(delay)

text = f'Restarted'

print(text)

editing/svg.py

import os

from svgwrite import Drawing

from svgwrite.shapes import Ellipse

from svgwrite.path import Path

from svgwrite.text import TextPath, Text

from math import pi, cos, sin, radians, degrees

*# def make\_circle(size: int,*

*# border: int,*

*# text: str,*

*# font\_size:int,*

*# round\_c,*

*# text\_c,*

*# round\_opacity: float,*

*# angle: int,*

*# id: str):*

*# dwg = Drawing(f"{id}.svg", (size, size))*

*# text = text + "‎"*

*# c = size // 2*

*# r = (size - border) // 2*

*# angle = radians(angle) - pi / 2*

*# ellipse = Ellipse(center=(c, c),*

*# r=(r, r),*

*# stroke=f"rgb({round\_c[0]}, {round\_c[1]}, {round\_c[2]})",*

*# fill\_opacity=0,*

*# stroke\_width=f"{border}",*

*# opacity=round\_opacity)*

*# dwg.add(ellipse)*

*# border \*= 0.75*

*# r\_path = size // 2 - border*

*# center = size // 2*

*# text\_length = 2 \* pi \* r\_path*

*# path\_form = f"""M {center \* (1 - cos(angle)) + border \* cos(angle)} {center \* (1 - sin(angle)) + border \* sin(angle)}*

*# A {r\_path} {r\_path} 0 1 1 {center \* (1 - cos(angle + pi)) + border \* cos(angle + pi)} {center \* (1 - sin(angle + pi)) + border \* sin(angle + pi)}*

*# M {center \* (1 - cos(angle + pi)) + border \* cos(angle + pi)} {center \* (1 - sin(angle + pi)) + border \* sin(angle + pi)}*

*# A {r\_path} {r\_path} 0 1 1 {center \* (1 - cos(angle)) + border \* cos(angle)} {center \* (1 - sin(angle)) + border \* sin(angle)}*

*# """*

*# path = Path(path\_form, fill\_opacity=0)*

*# dwg.add(path)*

*# text = Text(text="",*

*# fill=f"rgb({text\_c[0]}, {text\_c[1]}, {text\_c[2]})",*

*# style=f"font-size:{font\_size}px; font-family:Arial",*

*# textLength=text\_length,*

*# lengthAdjust="spacing")*

*# text.add(TextPath(path=path,*

*# text=text,*

*# method='align',*

*# spacing='exact'))*

*# dwg.add(text)*

*# dwg.save()*

*# os.system(f"inkscape {id}.svg -o {id}.png")*

*# os.remove(f"{id}.svg")*

def make\_circle(size: int,

text: str,

font\_text: str,

font\_size: int,

font\_weight: int,

text\_color: str,

border: int,

border\_color: str,

border\_opacity: float,

angle: int,

id: str):

image\_name = "".join(id.split("."))[:-1]

dwg = Drawing(f"{image\_name}.svg", (size, size))

text = text + "‎"

c = size // 2

r = (size - border) // 2

angle = radians(angle) - pi / 2

ellipse = Ellipse(*center*=(c, c),

*r*=(r, r),

*stroke*=border\_color,

*fill\_opacity*=0,

*stroke\_width*=f"{border}",

*opacity*=border\_opacity)

dwg.add(ellipse)

border \*= 0.75

r\_path = size // 2 - border

center = size // 2

text\_length = 2 \* pi \* r\_path

path\_form = f"""M {center \* (1 - cos(angle)) + border \* cos(angle)} {center \* (1 - sin(angle)) + border \* sin(angle)}

A {r\_path} {r\_path} 0 1 1 {center \* (1 - cos(angle + pi)) + border \* cos(angle + pi)} {center \* (1 - sin(angle + pi)) + border \* sin(angle + pi)}

M {center \* (1 - cos(angle + pi)) + border \* cos(angle + pi)} {center \* (1 - sin(angle + pi)) + border \* sin(angle + pi)}

A {r\_path} {r\_path} 0 1 1 {center \* (1 - cos(angle)) + border \* cos(angle)} {center \* (1 - sin(angle)) + border \* sin(angle)}

"""

path = Path(path\_form, *fill\_opacity*=0)

dwg.add(path)

text\_obj = Text(*text*="",

*fill*=text\_color,

*style*=f"font-size:{font\_size}px; font-family:{font\_text}; font-weight:{font\_weight}",)

*# textLength=text\_length,)*

*# lengthAdjust="spacing")*

text\_obj.add(TextPath(*path*=path,

*text*=text,

*method*='align', ))

*# spacing='exact'))*

dwg.add(text\_obj)

dwg.save()

os.system(f"inkscape {image\_name}.svg -o {image\_name}.png")

os.remove(f"{image\_name}.svg")

*# make\_circle(800, 50, "sample sample", 50, (99, 57, 116), (214, 137, 16), 0.7, 90, id="another\_try")*

*# make\_circle(600, "SAMPLE SAMPLE SAMPLE", "Arial", 40, 100, "rgb(0, 0, 0)", 50, "rgb(255, 255, 255)", 0.5, 90, "another try")*

def make\_text(size: int, offset: int, text: str, text\_weight: int, font\_family: str, font\_size: int, text\_c: tuple, image\_name: str):

dwg = Drawing(f"{id}.svg", *size*=(size, size), *debug*=True)

dwg.add(Text(*text*=text,

*fill*=f"rgb({text\_c[0]}, {text\_c[1]}, {text\_c[2]})",

*x*=[size//2],

*y*=[size//2 + offset],

*style*=f"font-size:{font\_size}px; font-family:{font\_family}; text-weight:{text\_weight}",

*text\_anchor*="middle"))

dwg.save()

os.system(f"inkscape {image\_name}.svg -o {image\_name}.png")

os.remove(f"{image\_name}.svg")

*# make\_text(576, 50, "sample sample sample", 32, (255, 255, 255), "1337")*

editing/utils.py

from moviepy.editor import \*

import numpy as np

from app.editing.svg import make\_circle, make\_text

def make\_rounded(filename, size, text, font\_text, font\_size, font\_weight, text\_color, border, border\_color, border\_opacity, angle):

video = VideoFileClip(filename)

video = video.resize((576, 576))

make\_circle(*size*=size,

*text*=text,

*font\_text*=font\_text,

*font\_size*=font\_size,

*font\_weight*=font\_weight,

*text\_color*=text\_color,

*border*=border,

*border\_color*=border\_color,

*border\_opacity*=border\_opacity,

*angle*=angle,

*id*=filename)

*# bg\_image = ColorClip(size=(size, size), color=(255, 255, 255), duration=video.duration)*

image\_name = "".join(filename.split("."))[:-1]

image = ImageClip(f"{image\_name}.png").set\_duration(video.duration)

os.remove(f"{image\_name}.png")

filename\_out = "".join(filename.split("."))[:-1] + "temp.mp4"

final = CompositeVideoClip([video.set\_position("center", "center"), image.set\_position("center", "center")])

final.write\_videofile(

filename\_out,

*codec*="libx264",

*fps*=60,

*logger*=None)

*# make\_rounded("app/data/another\_sample.mp4", 576, 80, "samplesamplesample", 54, (99, 57, 116), (214, 137, 16), 0.7, 135, "228")*

def overlay\_text(filename, size, offset, text, text\_weight, font\_family, font\_size, text\_c, image\_name):

video = VideoFileClip(filename)

video = video.resize((600, 600))

make\_text(*size*=size, *offset*=offset, *text*=text, *text\_weight*=text\_weight, *font\_family*=font\_family, *font\_size*=font\_size, *text\_c*=text\_c, *image\_name*=image\_name)

text = ImageClip(f"{id}.png").set\_duration(video.duration)

filename\_out = "".join(filename.split("."))[:-1] + "temp.mp4"

final = CompositeVideoClip([video.set\_position("center", "center"), text.set\_position("center", "center")])

final.write\_videofile(

filename\_out,

*codec*="libx264",

*fps*=60

)

def make\_watermark(filename: str, opacity: float, offsetX: int, offsetY: int, picture\_file\_path: str) -> None:

video = VideoFileClip(filename)

video = video.resize((600, 600))

image = ImageClip(picture\_file\_path).set\_duration(video.duration).set\_opacity(opacity)

image\_y = image.size[1]

scale = 200 / image\_y

image = image.resize(scale)

filename\_out = "".join(filename.split("."))[:-1] + "temp.mp4"

position = (video.size[0] // 2 + offsetX - image.size[0] // 2, video.size[1] // 2 + 150 - image.size[1] // 2)

final = CompositeVideoClip([video, image.set\_position(position)], *size*=(600, 600))

final.write\_videofile(

filename\_out,

*codec*="libx264",

*fps*=60,

*logger*=None

)

def make\_default\_rounded(filename: str) -> None:

video = VideoFileClip(filename)

video = video.resize((600, 600))

filename\_out = "".join(filename.split("."))[:-1] + "temp.mp4"

video.write\_videofile(

filename\_out,

*codec*="libx264",

*fps*=60,

*logger*=None

)

redis/database.py

from typing import Optional

from redis import Redis

from redis.asyncio import from\_url

from redis.exceptions import ConnectionError

from app.settings.config import settings

connection\_url = f"redis://{settings.REDIS\_HOST}:{settings.REDIS\_PORT}?decode\_responses=True"

r = from\_url(connection\_url)

async def ping\_redis\_connection(r: Redis):

try:

await r.ping()

print("Redis pinged. Successfully connected")

except ConnectionError:

raise Exception(

f"Redis error: failed to connect to redis database with url {connection\_url}"

)

settings/config.py

import os

from decouple import config

import string

import random

class Settings:

TELEGRAM\_BOT\_TOKEN = config("TELEGRAM\_BOT\_TOKEN")

PROVIDER\_TOKEN = config("PROVIDER\_TOKEN")

DB\_NAME = config("DB\_NAME")

DB\_USER = config("DB\_USER")

DB\_PASS = config("DB\_PASS")

DB\_HOST = config("DB\_HOST")

DB\_PORT = config("DB\_PORT")

REDIS\_HOST = config("REDIS\_HOST")

REDIS\_PORT = config("REDIS\_PORT")

DB\_URL = f"postgres://{DB\_USER}:{DB\_PASS}@{DB\_HOST}:{DB\_PORT}/{DB\_NAME}"

APPLICATIONS = [

"db"

]

APPLICATIONS\_MODULE = "app"

DEFAULT\_ADMIN\_TG\_ID = config("DEFAULT\_ADMIN\_TG\_ID")

DEFAULT\_ADMIN\_USERNAME = config("DEFAULT\_ADMIN\_USERNAME")

DEFAULT\_ADMIN\_FIRST\_NAME = config("DEFAULT\_ADMIN\_FIRST\_NAME")

SHOP\_ID = config("SHOP\_ID")

YOO\_SECRET = config("YOO\_SECRET")

settings = Settings()

main.py

import time

import asyncio

from tortoise import run\_async

from telebot.asyncio\_filters import StateFilter

from app.db.init\_db import init, create\_default\_admin\_user

*# from app.editing.handlers import send\_rounded*

from app.editing.handler import send\_rounded

from app.bot.handler import bot

if \_\_name\_\_ == "\_\_main\_\_":

bot.add\_custom\_filter(StateFilter(bot))

run\_async(init())

run\_async(create\_default\_admin\_user())

while True:

try:

loop = asyncio.get\_event\_loop()

f1 = loop.create\_task(bot.polling(*none\_stop*=True))

f2 = loop.create\_task(send\_rounded())

loop.run\_forever()

except Exception as e:

delay = 3

text = f'Error: {e.with\_traceback()}, restarting after {delay} seconds'

print(text)

time.sleep(delay)

text = f'Restarted'

print(text)