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

base/base_models.py

```python
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

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

from pydantic import UUID4
```

```python
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(sa
mple_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(sa
mple_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
```

```python
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
```

```python
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(p
icture_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
```

```python
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

```python
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!"
```

```python
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

```python
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)
```

```python
    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

```python
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
```

```python
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())
```

```python
@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())
```

```python
@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))
```

```python
@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))
```

```python
@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=pictu
re.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.titl
e),
                                  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)
```

```python
@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"))
```

```python
# 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
```

```python
@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
```

```python
@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))
```

```python
@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
```

```python
@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
```

```python
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(opac
ity)

    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=Tru
e"



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

```python
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)
```