Отчет по лабораторной работе №5 по дисциплине “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

```
import json


from telebot.types import InlineKeyboardButton, InlineKeyboardMarkup
from app.db.models import User, FavoriteRoute

from app.bot import texts




def start_menu(to_user: User) -> InlineKeyboardMarkup:

    keyboard = InlineKeyboardMarkup()

    keyboard.add(InlineKeyboardButton(text=texts.favorite_routes_button,
callback_data=f"get_favorites_{to_user.tg_id}"))

    keyboard.add(InlineKeyboardButton(text=texts.get_route_schedule,
callback_data=f"new_route"))

    return keyboard




def from_favorites(to_user: User, start_id: str, finish_id: str) ->
InlineKeyboardMarkup:

    keyboard = InlineKeyboardMarkup()

    keyboard.add(InlineKeyboardButton(text=texts.refresh_schedule,
callback_data=f"updf_{start_id}_{finish_id}"))

    keyboard.add(InlineKeyboardButton(text=texts.remove_from_favorites,
callback_data=f"rm_{start_id}_{finish_id}"))

    keyboard.add(InlineKeyboardButton(text=texts.back_to_favorites,
callback_data=f"get_favorites_{to_user.tg_id}"))

    keyboard.add(InlineKeyboardButton(text=texts.back_to_welcome_menu,
callback_data=f"back_to_start_menu"))



    return keyboard
```

```python
async def favorite_routes(to_user: User) -> InlineKeyboardMarkup:
    keyboard = InlineKeyboardMarkup()

    routes = await FavoriteRoute.filter(user_id=to_user.uuid)

    for i in routes:
        added_stations_titles = {}

        with open("./app/data/data_russia_trains.json", "r") as f:
            data = json.load(f)
            for region_title, region_data in data.items():
                for station in region_data:
                    if i.start_station == station["id"]:
                        added_stations_titles.update({"start":
station["title"]})
                    if i.finish_station == station["id"]:
                        added_stations_titles.update({"finish":
station["title"]})


keyboard.add(InlineKeyboardButton(text=texts.get_route_text(added_stations
_titles["start"], added_stations_titles["finish"]),

callback_data=f"route_{i.start_station}_{i.finish_station}"))
```

```
    keyboard.add(InlineKeyboardButton(text=texts.back_to_welcome_menu,
callback_data=f"back_to_start_menu"))

    return keyboard


def searched_start_stations(stations: list) -> InlineKeyboardMarkup:
    keyboard = InlineKeyboardMarkup()

    for station in stations:

keyboard.add(InlineKeyboardButton(text=texts.get_station_text(station["tit
le"], station["region_title"]),

callback_data=f"start_\"{station['title']}\"_{station['id']}"))

    keyboard.add(InlineKeyboardButton(text=texts.cancel_search,
callback_data=f"new_route"))

    return keyboard


def searched_finish_stations(stations: list) -> InlineKeyboardMarkup:
    keyboard = InlineKeyboardMarkup()

    for station in stations:
```

```
keyboard.add(InlineKeyboardButton(text=texts.get_station_text(station["tit
le"], station["region_title"]),

callback_data=f"finish_\"{station['title']}\"_{station['id']}"))


    keyboard.add(InlineKeyboardButton(text=texts.cancel_search,
callback_data=f"new_route"))



    return keyboard




def schedule(start_id, finish_id) -> InlineKeyboardMarkup:
    keyboard = InlineKeyboardMarkup()



    # keyboard.add(InlineKeyboardButton(text=texts.add_route_to_favorites,
callback_data=f"addtofavorites_{to_user}_{start_station_id}_{finish_statio
n_id}_{start_station}_{finish_station}"))
    keyboard.add(InlineKeyboardButton(text=texts.refresh_schedule,
callback_data=f"upds_{start_id}_{finish_id}"))
    keyboard.add(InlineKeyboardButton(text=texts.add_route_to_favorites,
callback_data=f"addf_{start_id}_{finish_id}"))
    keyboard.add(InlineKeyboardButton(text=texts.back_to_welcome_menu,
callback_data=f"back_to_start_menu"))



    return keyboard



def go_to_main_menu_only():
    keyboard = InlineKeyboardMarkup()


    keyboard.add(InlineKeyboardButton(text=texts.back_to_welcome_menu, 
callback_data=f"back_to_start_menu"))



    return keyboard
```

bot/texts.py

```
import json



from telebot.types import InlineKeyboardButton, InlineKeyboardMarkup



from app.db.models import User, FavoriteRoute

from app.bot import texts




def start_menu(to_user: User) -> InlineKeyboardMarkup:

    keyboard = InlineKeyboardMarkup()

    keyboard.add(InlineKeyboardButton(text=texts.favorite_routes_button, 
callback_data=f"get_favorites_{to_user.tg_id}"))

    keyboard.add(InlineKeyboardButton(text=texts.get_route_schedule, 
callback_data=f"new_route"))

    return keyboard
```

```python
def from_favorites(to_user: User, start_id: str, finish_id: str) ->
InlineKeyboardMarkup:
    keyboard = InlineKeyboardMarkup()
    keyboard.add(InlineKeyboardButton(text=texts.refresh_schedule,
callback_data=f"updf_{start_id}_{finish_id}"))
    keyboard.add(InlineKeyboardButton(text=texts.remove_from_favorites,
callback_data=f"rm_{start_id}_{finish_id}"))
    keyboard.add(InlineKeyboardButton(text=texts.back_to_favorites,
callback_data=f"get_favorites_{to_user.tg_id}"))
    keyboard.add(InlineKeyboardButton(text=texts.back_to_welcome_menu,
callback_data=f"back_to_start_menu"))


    return keyboard



async def favorite_routes(to_user: User) -> InlineKeyboardMarkup:
    keyboard = InlineKeyboardMarkup()


    routes = await FavoriteRoute.filter(user_id=to_user.uuid)


    for i in routes:
        added_stations_titles = {}


        with open("./app/data/data_russia_trains.json", "r") as f:
            data = json.load(f)

            for region_title, region_data in data.items():

                for station in region_data:

                    if i.start_station == station["id"]:

                        added_stations_titles.update({"start":
station["title"]})

                    if i.finish_station == station["id"]:

                        added_stations_titles.update({"finish":
station["title"]})




keyboard.add(InlineKeyboardButton(text=texts.get_route_text(added_stations
_titles["start"], added_stations_titles["finish"]),


callback_data=f"route_{i.start_station}_{i.finish_station}"))


    keyboard.add(InlineKeyboardButton(text=texts.back_to_welcome_menu,
callback_data=f"back_to_start_menu"))



    return keyboard




def searched_start_stations(stations: list) -> InlineKeyboardMarkup:

    keyboard = InlineKeyboardMarkup()


    for station in stations:


keyboard.add(InlineKeyboardButton(text=texts.get_station_text(station["tit
le"], station["region_title"]),
callback_data=f"start_\"{station['title']}\"_{station['id']}"))


    keyboard.add(InlineKeyboardButton(text=texts.cancel_search,
callback_data=f"new_route"))



    return keyboard




def searched_finish_stations(stations: list) -> InlineKeyboardMarkup:

    keyboard = InlineKeyboardMarkup()



    for station in stations:

keyboard.add(InlineKeyboardButton(text=texts.get_station_text(station["tit
le"], station["region_title"]),

callback_data=f"finish_\"{station['title']}\"_{station['id']}"))


    keyboard.add(InlineKeyboardButton(text=texts.cancel_search,
callback_data=f"new_route"))



    return keyboard




def schedule(start_id, finish_id) -> InlineKeyboardMarkup:

    keyboard = InlineKeyboardMarkup()
    # keyboard.add(InlineKeyboardButton(text=texts.add_route_to_favorites, callback_data=f"addtofavorites_{to_user}_{start_station_id}_{finish_station_id}_{start_station}_{finish_station}"))

    keyboard.add(InlineKeyboardButton(text=texts.refresh_schedule, callback_data=f"upds_{start_id}_{finish_id}"))

    keyboard.add(InlineKeyboardButton(text=texts.add_route_to_favorites, callback_data=f"addf_{start_id}_{finish_id}"))

    keyboard.add(InlineKeyboardButton(text=texts.back_to_welcome_menu, callback_data=f"back_to_start_menu"))



    return keyboard



def go_to_main_menu_only():

    keyboard = InlineKeyboardMarkup()


    keyboard.add(InlineKeyboardButton(text=texts.back_to_welcome_menu, callback_data=f"back_to_start_menu"))



    return keyboard
```

bot/handler.py

```
import telebot

import os

import secrets
```

```python
import time


from app.db.init_db import init
from app.db.models import User
from app.db.schemas import BaseUserCreate
from app.bot import texts
from app.bot import menu
from app.settings.config import settings


bot = telebot.TeleBot(settings.TELEGRAM_BOT_TOKEN)


@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(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).model_dump())

        bot.send_message(user.tg_id, texts.first_join(user.first_name),
reply_markup=menu.start_menu)



    else:

        bot.send_message(user.tg_id, texts.welcome_text(user.first_name),
reply_markup=menu.start_menu)
```

data/parser.py

```
import json



def get_russia():

    with open("./app/data/data.json", "r") as rf:

        data = (json.load(rf))["countries"]

        with open("./app/data/data_russia.json", "w", encoding="utf-8") as
wf:

            for i in data:

                if "Россия" in i["title"]:

                    json.dump(i, wf, ensure_ascii=False)



def get_only_trains():

    with open("./app/data/data_russia.json") as rf:

        data = (json.load(rf))["regions"]

        res = {}
        with open("./app/data/data_russia_trains.json", "w") as wf:

            for region in data:

                res.update({region["title"]: []})

                for settlement in region["settlements"]:

                    for stations in settlement["stations"]:

                        if stations["transport_type"] == "train":

                            res[region["title"]].append({"title":
stations["title"], "id": stations["codes"]["yandex_code"]})

            json.dump(res, wf, ensure_ascii=False)



get_russia()

get_only_trains()
```

db/init_db.py

```
from tortoise import Tortoise



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")
```

db/models.py

```
from typing import Optional



from tortoise import fields

from tortoise.exceptions import DoesNotExist



from app.base.base_models import BaseModel

from app.db.schemas import BaseUserCreate, BaseRouteCreate




class User(BaseModel):

    tg_id = fields.IntField()

    username = fields.CharField(max_length=128)

    first_name = fields.CharField(max_length=128)



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

        user_dict = user.model_dump()

        model = cls(**user_dict)

        await model.save()

        return model


    class Meta:

        table = "users"




class FavoriteRoute(BaseModel):

    start_station = fields.CharField(max_length=128)

    finish_station = fields.CharField(max_length=128)

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

        "models.User", related_name="selected_routes", to_field="uuid",
on_delete=fields.CASCADE

    )
    @classmethod
    async def create(cls, route: BaseRouteCreate, user: User) ->
"FavoriteRoute":
        route_dict = route.model_dump()
        model = cls(**route_dict)
        model.user = user
        await model.save()
        return model

    class Meta:
        table = "routes"
```

db/schemas.py

```
import uuid
from pydantic import BaseModel, validator



class BaseProperties(BaseModel):
    @validator("uuid", pre=True, always=True, check_fields=False)
    def default_hashed_id(cls, v):
        return v or uuid.uuid4()
```

```python
class BaseUserCreate(BaseProperties):
    tg_id: int
    username: str
    first_name: str


    class Config:
        from_attributes = True



class BaseRouteCreate(BaseProperties):
    start_station: str
    finish_station: str


    class Config:
        from_attributes = True
```

settings/config.py

```python
import os



from decouple import config



import string
import random
```

```python
class Settings:
    YANDEX_API_TOKEN = config("YANDEX_API_TOKEN")
    TELEGRAM_BOT_TOKEN = config("TELEGRAM_BOT_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")

    DB_URL =
f"postgres://{DB_USER}:{DB_PASS}@{DB_HOST}:{DB_PORT}/{DB_NAME}"

    APPLICATIONS = [
        "db"
    ]

    APPLICATIONS_MODULE = "app"


settings = Settings()
```

main.py

```python
import telebot

import json

import time

import asyncio

import aiohttp

import pytz

from dateutil import parser



from datetime import date, datetime, timedelta

from telebot.async_telebot import AsyncTeleBot

from telebot.asyncio_storage import StateMemoryStorage

from telebot.asyncio_filters import StateFilter

from telebot.asyncio_handler_backends import State, StatesGroup

from tortoise import run_async



from app.db.init_db import init

from app.db.models import User, FavoriteRoute

from app.db.schemas import BaseUserCreate, BaseRouteCreate

from app.bot import texts

from app.bot import menu

from app.settings.config import settings



state_storage = StateMemoryStorage()
```

```
bot = AsyncTeleBot(settings.TELEGRAM_BOT_TOKEN,
state_storage=state_storage)


class States(StatesGroup):
    search_start_station = State()
    search_finish_station = State()


async def get_schedule_today_request(session, to_user: User,
start_station_id: str, finish_station_id: str):
    url =
f"https://api.rasp.yandex.net/v3.0/search/?apikey={settings.YANDEX_API_TOK
EN}&format=json&from={start_station_id}&to={finish_station_id}&lang=ru_RU&
page=1&date={date.isoformat(date.today())}&limit=10000"
    try:
        async with session.get(url=url) as response:
            return await response.json()
    except BaseException:
        bot.send_message(to_user.tg_id, text=texts.cannot_get_schedule,
reply_markup=menu.go_to_main_menu_only())


async def get_schedule_today_task(to_user: User, start_station_id: str,
finish_station_id: str):
    async with aiohttp.ClientSession(headers={"Accept":
"application/json"}) as session:
        task = get_schedule_today_request(session, to_user,
start_station_id, finish_station_id)
```

```python
        return await asyncio.gather(task)




async def get_schedule_tomorrow_request(session, to_user: User,
start_station_id: str, finish_station_id: str):

    url =
f"https://api.rasp.yandex.net/v3.0/search/?apikey={settings.YANDEX_API_TOK
EN}&format=json&from={start_station_id}&to={finish_station_id}&lang=ru_RU&
page=1&date={date.isoformat(date.today()+timedelta(hours=24))}&limit=10000
"

    try:

        async with session.get(url=url) as response:

            return await response.json()

    except BaseException:

        bot.send_message(to_user.tg_id, text=texts.cannot_get_schedule,
reply_markup=menu.go_to_main_menu_only())




async def get_schedule_tomorrow_task(to_user: User, start_station_id: str,
finish_station_id: str):

    async with aiohttp.ClientSession(headers={"Accept":
"application/json"}) as session:

        task = get_schedule_today_request(session, to_user,
start_station_id, finish_station_id)

        return await asyncio.gather(task)




def schedule_filter(trip: dict):
    return parser.parse(trip["departure"]) >
pytz.utc.localize(datetime.utcnow()) and
parser.parse(trip["departure"]).date() == date.today()




def get_normalized_schedule_response(schedule: dict, count: int, today:
bool) -> list:

    norm_schedule = []

    for train in list(filter(schedule_filter,
schedule["segments"]))[:count] if today is True else
schedule["segments"][:count]:

        trip = {"number": train["thread"]["number"],

                "title": train["thread"]["title"],

                "train_subtype":
train["thread"]["transport_subtype"]["title"],

                "stops": train["stops"],

                "from": train["from"]["title"],

                "to": train["to"]["title"],

                "departure_platform": train["departure_platform"],

                "arrival_platform": train["arrival_platform"],

                "departure_time": train["departure"],

                "arrival_time": train["arrival"],

                "duration": train["duration"]}


        norm_schedule.append(trip)


    return norm_schedule
```

```python
async def user_does_not_exist_message(id: int):
    await bot.send_message(id,
                           texts.user_does_not_exist(id),
                           reply_markup=menu.start_menu(id))


temp_station_search = dict()


@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(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))
        message = await bot.send_message(user.tg_id,
texts.first_join(user.first_name), reply_markup=menu.start_menu(user))
    else:
        message = await bot.send_message(user.tg_id,
texts.welcome_text(user.first_name), reply_markup=menu.start_menu(user))
```

```
@bot.message_handler(state=States.search_start_station)
async def search_start_station(msg: telebot.types.Message):
    search_list = []
    with open("./app/data/data_russia_trains.json", "r") as f:
        data = json.load(f)
        for region_title, region_data in data.items():
            for station in region_data:
                if msg.text.lower() in station["title"].lower():
                    search_list.append({"title": station["title"],
                                        "id": station["id"],
                                        "region_title": region_title})


    await bot.delete_state(msg.from_user.id, msg.chat.id)
    await bot.send_message(msg.chat.id,
                           text=texts.choose_start_station,

reply_markup=menu.searched_start_stations(search_list))



@bot.message_handler(state=States.search_finish_station)
async def search_finish_station(msg: telebot.types.Message):
    search_list = []
    with open("./app/data/data_russia_trains.json", "r") as f:
        data = json.load(f)
        for region_title, region_data in data.items():
            for station in region_data:
                if msg.text.lower() in station["title"].lower():
                    search_list.append({"title": station["title"],
                                        "id": station["id"],
                                        "region_title": region_title})


    await bot.send_message(msg.chat.id,
                           text=texts.choose_finish_station,
reply_markup=menu.searched_finish_stations(search_list))



@bot.callback_query_handler(func=lambda call:
call.data.startswith("get_favorites_"))
async def get_favorites_callback_handler(call:
telebot.types.CallbackQuery):
    user = await User.get_by_tg_id(call.from_user.id)


    if user is None:
        await user_does_not_exist_message(call.from_user.id)
    else:
        await bot.edit_message_text(text=texts.favorite_routes_response,
                                    chat_id=call.message.chat.id,
                                    message_id=call.message.message_id,
                                    reply_markup=await
menu.favorite_routes(user))
```

```
@bot.callback_query_handler(func=lambda call:
call.data.startswith("back_to_start_menu"))
async def start_menu_callback_handler(call: telebot.types.CallbackQuery):
    uid = int(call.from_user.id)
    user = await User.get_by_tg_id(uid)


    if not user:
        username = call.from_user.username
        first_name = call.from_user.first_name
        user = await User.create(BaseUserCreate(tg_id=uid,
first_name=first_name, username=username))
        await
bot.edit_message_text(text=texts.first_join(user.first_name),
                                    chat_id=call.message.chat.id,
                                    message_id=call.message.message_id,
                                    reply_markup=menu.start_menu(user))
    else:
        await
bot.edit_message_text(text=texts.welcome_text(user.first_name),
                                    chat_id=call.message.chat.id,
                                    message_id=call.message.message_id,
                                    reply_markup=menu.start_menu(user))



@bot.callback_query_handler(func=lambda call:
call.data.startswith("new_route"))
```

```python
async def new_route_callback_handler(call: telebot.types.CallbackQuery):
    user = await User.get_by_tg_id(call.from_user.id)


    if not user:
        await user_does_not_exist_message(call.from_user.id)



    await bot.set_state(user.tg_id, States.search_start_station,
call.message.chat.id)
    await bot.edit_message_text(text=texts.search_start_station,
                                chat_id=call.message.chat.id,
                                message_id=call.message.message_id,
                                reply_markup=menu.go_to_main_menu_only())




@bot.callback_query_handler(func=lambda call:
call.data.startswith("start_"))
async def pick_finish_station_callback_handler(call:
telebot.types.CallbackQuery):
    user = await User.get_by_tg_id(call.from_user.id)


    if not user:
        await user_does_not_exist_message(call.from_user.id)



    data = call.data.split('_')[1:]
    start_station_title = data[0][1:-1]
    start_station_id = data[1]
    temp_station_search.update({user.tg_id: {"title": start_station_title,
                                             "id": start_station_id}})


    await bot.set_state(user.tg_id, States.search_finish_station,
call.message.chat.id)
    await bot.edit_message_text(text=texts.search_finish_station,
                                chat_id=call.message.chat.id,
                                message_id=call.message.message_id,
                                reply_markup=menu.go_to_main_menu_only())



@bot.callback_query_handler(func=lambda call:
call.data.startswith("finish_"))
async def make_route_callback_handler(call: telebot.types.CallbackQuery):
    user = await User.get_by_tg_id(call.from_user.id)


    if not user:
        await user_does_not_exist_message(call.from_user.id)


    data = call.data.split('_')[1:]
    finish_station_id = data[1]


    start_station_data = temp_station_search.pop(user.tg_id)
    schedule = await get_schedule_today_task(user,
start_station_data["id"], finish_station_id)



    normalized_schedule = get_normalized_schedule_response(schedule[0], 3,
True)



    if len(normalized_schedule) < 3:
        schedule = await get_schedule_tomorrow_task(user,
start_station_data["id"], finish_station_id)
        normalized_tomorrow_schedule =
get_normalized_schedule_response(schedule[0], 3 -
len(normalized_schedule), False)
        normalized_schedule.extend(normalized_tomorrow_schedule)



    await bot.send_message(call.message.chat.id,

text=texts.get_schedule(json.dumps(normalized_schedule)),

reply_markup=menu.schedule(start_station_data["id"], finish_station_id),
                           parse_mode="Markdown")




@bot.callback_query_handler(func=lambda call:
call.data.startswith("addf_"))
async def add_route_to_favorites(call: telebot.types.CallbackQuery):
    user = await User.get_by_tg_id(call.from_user.id)
```

```python
    if not user:

        await user_does_not_exist_message(call.from_user.id)



    data = call.data.split('_')[1:]

    start_station_id = data[0]

    finish_station_id = data[1]



    route = await
FavoriteRoute.get_or_none(start_station=start_station_id,
finish_station=finish_station_id, user=user)

    if not route:

        await
FavoriteRoute.create(route=BaseRouteCreate(start_station=start_station_id,
finish_station=finish_station_id), user=user)



    added_stations_titles = {}



    with open("./app/data/data_russia_trains.json", "r") as f:

        data = json.load(f)

        for region_title, region_data in data.items():

            for station in region_data:

                if start_station_id == station["id"]:

                    added_stations_titles.update({"start":
station["title"]})

                if finish_station_id == station["id"]:
                    added_stations_titles.update({"finish":
station["title"]})



    await
bot.edit_message_text(text=call.message.text+texts.add_route(added_station
s_titles["start"], added_stations_titles["finish"]),

                                chat_id=call.message.chat.id,

                                message_id=call.message.message_id,

                                reply_markup=menu.go_to_main_menu_only(),

                                parse_mode="Markdown")





@bot.callback_query_handler(func=lambda call:
call.data.startswith("route_"))

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

    user = await User.get_by_tg_id(call.from_user.id)



    if not user:

        await user_does_not_exist_message(call.from_user.id)



    data = call.data.split('_')[1:]

    start_station_id = data[0]

    finish_station_id = data[1]



    schedule = await get_schedule_today_task(user, start_station_id,
finish_station_id)
```

```python
    normalized_schedule = get_normalized_schedule_response(schedule[0], 3,
True)



    if len(normalized_schedule) < 3:
        schedule = await get_schedule_tomorrow_task(user,
start_station_id, finish_station_id)
        normalized_tomorrow_schedule =
get_normalized_schedule_response(schedule[0], 3 -
len(normalized_schedule), False)
        normalized_schedule.extend(normalized_tomorrow_schedule)



    await
bot.edit_message_text(text=texts.get_schedule(json.dumps(normalized_schedu
le)),
                                chat_id=call.message.chat.id,
                                message_id=call.message.message_id,
                                reply_markup=menu.from_favorites(user,
start_station_id, finish_station_id),
                                parse_mode="Markdown")




@bot.callback_query_handler(func=lambda call: call.data.startswith("rm_"))
async def remove_route_from_favorites(call: telebot.types.CallbackQuery):
    user = await User.get_by_tg_id(call.from_user.id)



    if not user:
        await user_does_not_exist_message(call.from_user.id)



    data = call.data.split("_")[1:]

    start_station_id = data[0]

    finish_station_id = data[1]



    route = await
FavoriteRoute.get_or_none(start_station=start_station_id,
finish_station=finish_station_id, user=user)



    if not route:

        await bot.edit_message_text(text=texts.route_does_not_exist,

                                    chat_id=call.message.chat.id,

                                    message_id=call.message.message_id,

                                    reply_markup=await
menu.favorite_routes(user))

    else:

        await route.delete()

        await bot.edit_message_text(text=texts.route_delete_success,

                                    chat_id=call.message.chat.id,

                                    message_id=call.message.message_id,

                                    reply_markup=await
menu.favorite_routes(user))




@bot.callback_query_handler(func=lambda call:
call.data.startswith("updf_"))
```

```python
async def update_schedule_from_favorites(call:
telebot.types.CallbackQuery):
    user = await User.get_by_tg_id(call.from_user.id)


    if not user:
        await user_does_not_exist_message(call.from_user.id)


    data = call.data.split('_')[1:]
    start_station_id = data[0]
    finish_station_id = data[1]


    schedule = await get_schedule_today_task(user, start_station_id,
finish_station_id)


    normalized_schedule = get_normalized_schedule_response(schedule[0], 3,
True)


    if len(normalized_schedule) < 3:
        schedule = await get_schedule_tomorrow_task(user,
start_station_id, finish_station_id)
        normalized_tomorrow_schedule =
get_normalized_schedule_response(schedule[0], 3 -
len(normalized_schedule), False)
        normalized_schedule.extend(normalized_tomorrow_schedule)


    try:
        await
bot.edit_message_text(text=texts.get_schedule(json.dumps(normalized_schedu
le)),
                                    chat_id=call.message.chat.id,
                                    message_id=call.message.message_id,
                                    reply_markup=menu.from_favorites(user,
start_station_id, finish_station_id),
                                    parse_mode="Markdown")
    except Exception:
        return


@bot.callback_query_handler(func=lambda call:
call.data.startswith("upds_"))
async def update_schedule_from_new_route(call:
telebot.types.CallbackQuery):
    user = await User.get_by_tg_id(call.from_user.id)

    if not user:
        await user_does_not_exist_message(call.from_user.id)

    data = call.data.split('_')[1:]
    start_station_id = data[0]
    finish_station_id = data[1]

    schedule = await get_schedule_today_task(user, start_station_id,
finish_station_id)
```

```python
    normalized_schedule = get_normalized_schedule_response(schedule[0], 3, True)

    if len(normalized_schedule) < 3:
        schedule = await get_schedule_tomorrow_task(user, start_station_id, finish_station_id)
        normalized_tomorrow_schedule = get_normalized_schedule_response(schedule[0], 3 - len(normalized_schedule), False)
        normalized_schedule.extend(normalized_tomorrow_schedule)

    try:
        await bot.edit_message_text(text=texts.get_schedule(json.dumps(normalized_schedule)),
                                    chat_id=call.message.chat.id,
                                    message_id=call.message.message_id,
                                    reply_markup=menu.from_favorites(user, start_station_id, finish_station_id),
                                    parse_mode="Markdown")
    except Exception:
        return


if __name__ == "__main__":
    bot.add_custom_filter(StateFilter(bot))

    run_async(init())

    while True:

        try:

            run_async(bot.polling(none_stop=True))

        except Exception as e:

            delay = 3

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

            print(text)

            time.sleep(delay)

            text = f'Restarted'

            print(text)
```