Отчет по лабораторной работе №5 по диспиплине “Парадигмы и конструкции языков программирования”

base/base\_models.py

from tortoise import models, fields

class BaseModel(models.Model):

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

async def to\_dict(self):

d = {}

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

d[field] = getattr(self, field)

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

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

return d

class Meta:

abstract = True

bot/menu.py

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

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["title"], 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["title"], 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/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

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["title"], 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["title"], 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

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()

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

import os

from decouple import config

import string

import random

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

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\_TOKEN}&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)

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\_TOKEN}&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

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"))

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)

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\_stations\_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)

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\_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")

@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\_"))

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\_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

@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)

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)