Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Кафедра «Компьютерные системы и программные технологии»

К У Р С О В А Я Р А Б О Т А

Android-приложение для поиска авиабилетов

по дисциплине «Базы данных»

Выполнил
студент гр. 3530901/70203 _____ А.С. Черникова
(подпись) *(инициалы, фамилия)*

Руководитель _____ А.В. Мяснов
(подпись) *(инициалы, фамилия)*

«___» _____ 2020 г.

Санкт-Петербург
2020

Санкт-Петербургский государственный политехнический университет

ЗАДАНИЕ
НА ВЫПОЛНЕНИЕ КУРСОВОГО ПРОЕКТА

студенту группы 3530901/70203 Черниковой Арине Сергеевне
(номер группы) *(фамилия, имя, отчество)*

1. Срок сдачи законченного проекта 13.06.2020

2. Исходные данные к проекту: Задание для курсового проекта

3. Содержание пояснительной записки (перечень подлежащих разработке вопросов): техническое задание, реализация, вывод.

Дата получения задания: «10» мая 2020 г.

Руководитель _____ А.В. Мяснов
(подпись) *(инициалы, фамилия)*

Задание принял к исполнению _____ А.С. Черникова
(подпись студента) *(инициалы, фамилия)*

_____
(дата)

Содержание

1. Техническое задание

1.1. Постановка задачи

Разработать мобильное приложение для операционной системы Android, позволяющее пользователям регистрироваться в приложении, искать и приобретать авиабилеты. В качестве хранилища данных использовать базу данных, созданную в течение семестра на лабораторных работах.

1.2. Возможности приложения

- Вход для зарегистрированных пользователей;
- Регистрация новых пользователей;
- Возможность оставить отзыв об авиакомпании;
- Поиск авиабилетов по заданным городам и дате;
- Просмотр рейтинга авиакомпаний;
- Просмотр приобретенных билетов и личной информации;
- Изменение личных данных.

1.3. План разработки

- Принятие решения о необходимом функционале приложения;
- Разработка сервиса для взаимодействия с базой данных;
- Разработка мобильного приложения и реализация необходимой функциональности;
- Тестирование приложения;
- Выводы о проделанной работе и полученном результате.

2. Ход работы

2.1. Структура базы данных

В качестве базы данных взята база, разработанная в ходе лабораторных работ – “flight_db”. Схема базы данных представлена на рис.2.1.1.

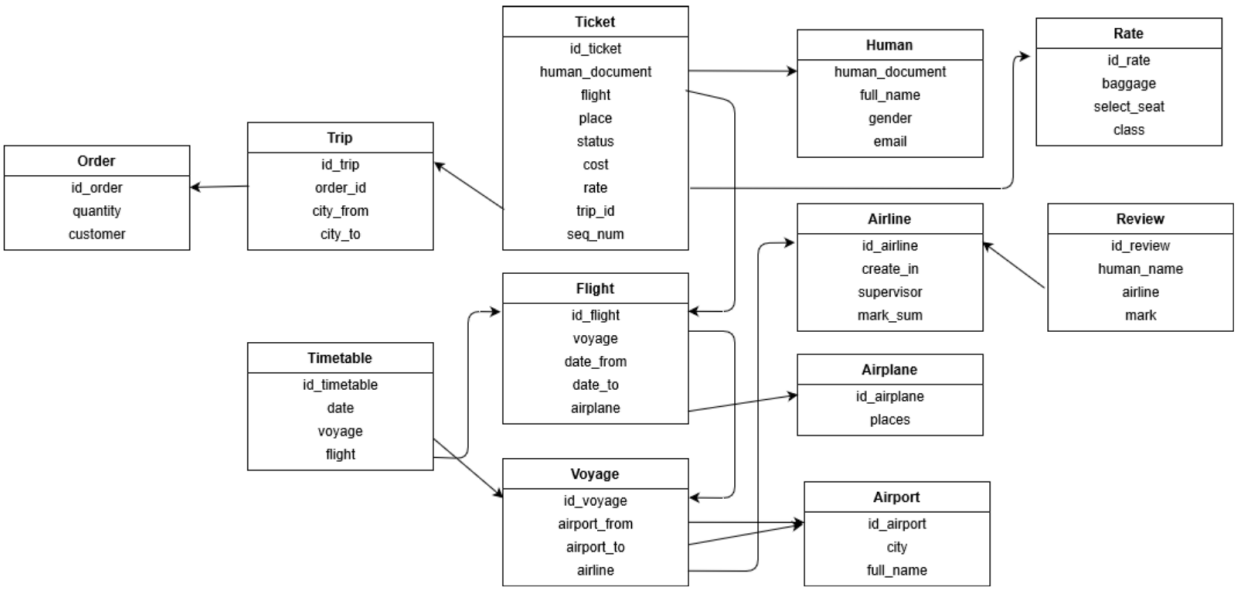


Рис.2.1.1. Схема используемой базы данных

В таблицу human добавлены поля для хранения логина и захешированного пароля (применяется десятикратное хеширование с солью алгоритмом SHA-512). Пример данных из таблицы human:

11222	name1	M	mail	nameee	36cf96fcf49090f260dcc1a33175428d85952a244b7375fead66cf20fc50bd6432d61aa4a5557d900fa192b8935a26fb9cc03cd2c6ced470d730851f5468b3c
1111 090909	full name	X	mail	logg	ed53d8818c9289a005zee31e598593b0c7beed0e9e958c7dd5e8358a7fbc362eb5a28e06d6467197931574 6c2d3216219 9bdba0e826e66f93be77a75d0df9451e
11	your name	X	mail	login	c33b5534b690ddf4871087383deb85753ae7aedd9841ee311316dd2da7ef2f67b346d99b7477abad3130ad9a6e9fe8086f6c2f3826f55baa90339ad44e1378c9
000	your name	X	email	arina	abdc0b39af7cfc3bdb0976f31b5489207cd6e3cc9a583b5086000f6af4005470b1c188f3f56098aa87931d3554e79c3782cb1d5db2b463054bc2efb079e3d5d2
1111 11111	name	M	mail	name	bae37709716cdc310b679e9ff6a93fe44f0e5c6c01da51090ad40078f502b8f789d009ea9045630b018b0afdb82463a4833420073f9b92955535dde0515a86a1
1234 565656	name	M	email@mail.ru	namelogin	1e369240c4afed127b2d9953dfdc0c61fa3b274eb5f3b219ea6eb37938556bea4655e8021d72e79c112c9d8d38da6673778a07b6a9ca444d2edcdf36663ac3a0

Рис.2.1.2. Таблица human

2.2. Веб-сервис

Для взаимодействия с базой данных разработан веб-сервис. Приложение не подключается напрямую к базе данных, а взаимодействует с ней через API сервиса.

При разработке сервиса использовался framework Django.

Сервер запускается командой python manage.py runserver 0.0.0.0:8000 из рабочей папки.

Для проекта автоматически создается manage.py. manage.py – простой интерфейс для django-admin.py, который выполняет следующие действия перед тем, как обратиться к django-admin.py:

-добавляет пакет проекта в sys.path;

-устанавливает переменную окружения DJANGO_SETTINGS_MODULE, чтобы она указывала на файл settings.py проекта;

-вызывает django.setup() для инициализации Django.

Runserver запускает простой локальный Web-сервер для разработки. По умолчанию сервер запускается на 8000 порте и 127.0.0.1 IP адресе. В данном случае явно указывается IP адрес и порт для того, чтобы была возможность подключаться к серверу с других устройств.

В файле settings.py (общие настройки проекта) также прописывается конфигурация подключения к базе данных:

Листинг 2.2.1.

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'flight_db',
        'USER': 'user_flight',
        'PASSWORD': 'user',
        'HOST': 'localhost',
        'PORT': '',
    }
}
```

В файле models.py описаны все таблицы созданной базы данных, таблица отображается в класс. В каждом классе есть class Meta, отвечающий за настройки модели: здесь указывается параметр managed=False, который означает, что таблицы не будут создаваться или удаляться, так как мы используем уже существующую базу данных.

В файле urls.py прописываются возможные пути и их связь с методом отображения.

Листинг 2.2.2.

```
from django.contrib import admin
from django.urls import path
from django.conf.urls import include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('basic/',include('tables.urls')),
]
```

Здесь присутствует путь по умолчанию admin/, а также добавленный путь basic/, который далее будет искать путь в файле urls.py созданного приложения.

Листинг 2.2.3.

```
from django.contrib import admin
from django.urls import path
from django.conf.urls import include
from tables import views
from datetime import date

urlpatterns = [
    path('getLogin/<str:in_login>/', views.getLogin),
    path('getPassword/<str:in_login>/', views.getPassword),
    path('getHuman/<str:in_login>/', views.getHuman),
    path('getFlights/<str:city_from>/<str:city_to>/<str:date_f>/', views.getFlights),
    path('saveHuman/<str:doc>/<str:f_name>/<str:gen>/<str:mail>/<str:log>/<str:key>/',
views.saveHuman),
    path('saveReview/<str:r_name>/<str:r_airline>/<str:r_mark>/', views.saveReview),
    path('getTickets/<str:document>/', views.getTickets),
    path('saveTicket/<str:in_login>/<str:flight>/', views.saveTicket),
    path('getTopAirlines/', views.getTopAirlines),
path('changeInfo/<str:id_docum>/<str:name>/<str:gen>/<str:email>/',views.changeInfo),
]
```

В данном файле urls.py указаны допустимые пути, которые состоят из некоторого набора параметров, передающихся в соответствующие функции представления. Таким образом, параметры для запросов к базе данных передаются в качестве url.

Соответствующие функции для отображения находятся в файле views.py:

Листинг 2.2.4.

```
from django.shortcuts import render
from django.http.response import HttpResponse
from tables import models
from django.db.models import F
import random

def getLogin(request,in_login):
        login = models.Human.objects.filter(login=in_login)
        export_list = ""
        if login.exists():
                export_list = "Such login already exists"
        return HttpResponse(export_list)

def getPassword(request,in_login):
        password = models.Human.objects.filter(login=in_login)
        export_list = ""
        if not password.exists():
                export_list = "No such login"
        else:
                export_list = password[0].password

        return HttpResponse(export_list)
```

```
def getHuman(request, in_login):
	human = models.Human.objects.filter(login=in_login)
	export_list = ""
	if not human.exists():
		export_list = "No such login"

	else: export_list = human[0].human_document + '/' + human[0].full_name + '/' +
human[0].gender + '/' + human[0].email

	return HttpResponse(export_list)

def getFlights(request,city_from,city_to,date_f):
	export_list = ""
	airport_f = models.Airport.objects.filter(city=city_from)
	airport_t = models.Airport.objects.filter(city=city_to)

	if not airport_f.exists() or not airport_t.exists():
		export_list = "No such city"
		return HttpResponse(export_list)

	voyages = models.Voyage.objects.filter(airport_from=airport_f[0].id_airport,
airport_to=airport_t[0].id_airport)

	if not voyages.exists():
		export_list = "No such flights"
	else:

		flights=models.Flight.objects.filter(voyage=voyages[0].id_voyage,
date_from__gte=date_f).order_by('date_from') [:5]

		for i in list(voyages):
			if (i != voyages[0]):
				flights = models.Flight.objects.filter(voyage=i.id_voyage,
date_from__gte=date_f).order_by('date_from') [:5]

			for j in range(len(list(flights))):
				export_list = export_list + str(flights[j].id_flight) + ' ' +
str(flights[j].voyage.id_voyage) + ' ' + str(flights[j].date_from) + ' ' + str(flights[j].date_to) + ' '
+ str(flights[j].airplane.id_airplane) + "<br>"

	return HttpResponse(export_list)

def saveHuman(request,doc,f_name,gen,mail,log,key):
	prevDoc = models.Human.objects.filter(human_document=doc)
	export_list=""
	if prevDoc.exists():
		export_list = "Such document already exists"
	else:
		newDoc =
models.Human.objects.create(human_document=doc,full_name=f_name,gender=gen,email=m
ail,login=log,password=key)
		newDoc.save()
		export_list = "Successful adding"

	return HttpResponse(export_list)

def saveReview(request,r_name,r_airline,r_mark):
```

```
        airline_exists = models.Airline.objects.filter(id_airline=r_airline)
        export_list = ""
        if not airline_exists.exists():
                export_list = "No such airline"
        else:
                airline_exists = models.Airline.objects.get(id_airline=r_airline)
                if (int(r_mark)>10): export_list = "Max mark is 10"
                else:
                        newReview = models.Review.objects.create(human_name=r_name,
airline=airline_exists, mark=int(r_mark))
                        newReview.save()

                        allReview = models.Review.objects.filter(airline=r_airline)
                        sums=0
                        for i in range(len(list(allReview))):
                                sums = sums + int(allReview[i].mark)
                        sums = sums/len(list(allReview))

                        airline_exists.mark_sum = sums
                        airline_exists.save()

                        export_list = "Successful adding"

        return HttpResponse(export_list)


def getTickets(request,document):
        tickets = models.Ticket.objects.filter(human_document=document)
        export_list = ""
        if not tickets.exists():
                export_list = "You have no tickets"
        else:
                for i in range(len(list(tickets))):
                        export_list = export_list +
tickets[i].human_document.human_document + '/' + str(tickets[i].flight.id_flight) + '/' +
tickets[i].place + '/' +  tickets[i].status + '/' + str(tickets[i].cost) + '/' + str(tickets[i].seq_num) +
"<br>"

        return HttpResponse(export_list)

def saveTicket(request,in_login,flight):
        seat = ('ABCDEF')

        human = models.Human.objects.get(login=in_login)
        current_flight = models.Flight.objects.get(id_flight=flight)
        current_voyage =
models.Voyage.objects.get(id_voyage=current_flight.voyage.id_voyage)
        city_f = models.Airport.objects.get(id_airport=current_voyage.airport_from.id_airport)
        city_t = models.Airport.objects.get(id_airport=current_voyage.airport_to.id_airport)

        newOrder = models.Order.objects.create(quantity=1,customer=human.full_name)
        newOrder.save()

        newTrip =
models.Trip.objects.create(order_id=newOrder.id_order,city_from=city_f.full_name,city_to=ci
ty_t.full_name)
        newTrip.save()
```

```
            newPlace = str(random.randint(1,30)) + random.choice(seat)
            someRate = models.Rate.objects.get(id_rate=random.randrange(1,24))
            newTicket = models.Ticket.objects.create(human_document=human,
    flight=current_flight,
                        place=newPlace,status='Оплачено',cost=random.randrange(1,100000,50),
                        rate=someRate,trip_id=newTrip.id_trip,seq_num=str(1))
            newTicket.save()

            return HttpResponse("Successful buy")


    def getTopAirlines(request):
            airlines = models.Airline.objects.all().order_by('-mark_sum')[:10]

            export_list=""
            for i in range(len(list(airlines))):
                        export_list = export_list + airlines[i].id_airline + '/' + str(airlines[i].mark_sum)
    + "<br>"

    return HttpResponse(export_list)

    def changeInfo(request,id_docum,name,gen,email):
        human = models.Human.objects.get(human_document=id_docum)
        human.full_name = name
        human.gender = gen
        human.email = email
        human.save()
    return HttpResponse("Successful change")
```

В каждой функции происходит обращение к базе данных, формируется строка, состоящая из необходимых данных или сообщения об ошибке, если данные не найдены.

Таким образом, база данных и веб-сервис полностью реализованы. Их совокупная функциональность протестирована.

Описанный сервис на GitLab: http://gitlab.icc.spbstu.ru/chernrina/db-task/tree/master/coursework/service

2.3. Android-приложение

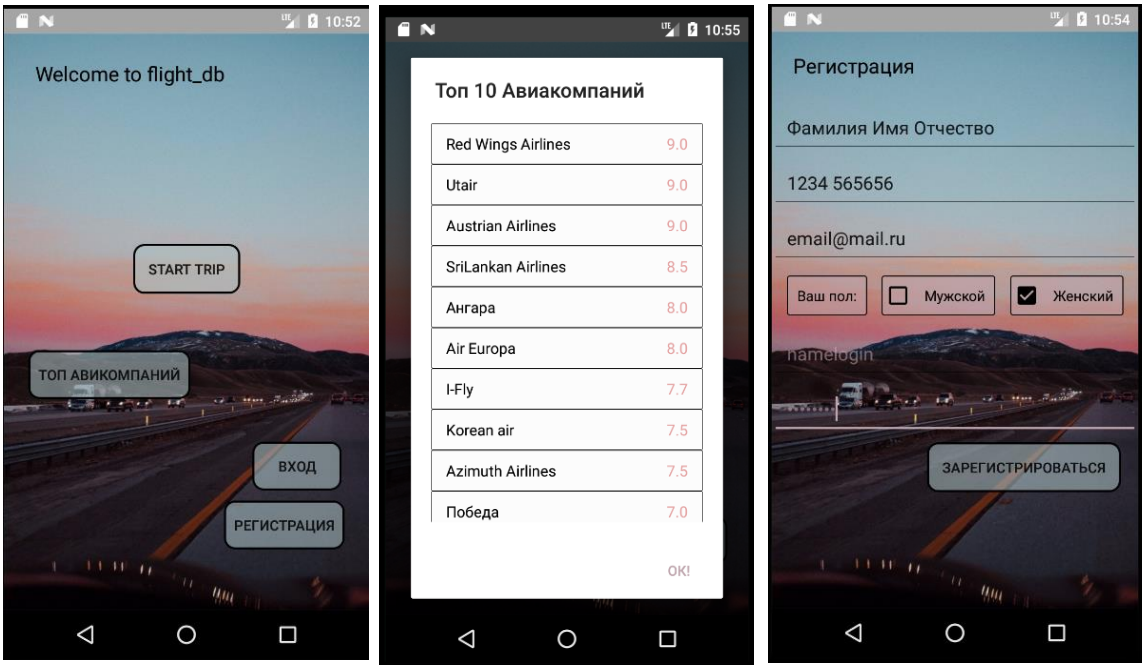
Android-приложение не контактирует напрямую с базой данных, а взаимодействует с ней через API веб-сервиса.

Каждый раз, когда необходимо обратиться к базе, формируется url-адрес, состоящий из необходимых для запроса параметров. В отдельном потоке происходит обращение к данному адресу, считывается информация и передается в основной поток приложения. В данном приложении за обращение к серверу отвечает класс downloadAsyncTask.

Листинг 2.3.1.

```
class DownloadAsyncTask :  AsyncTask<String, Void, String>() {
  override fun doInBackground(vararg urls: String): String {
      var out = ""
      val url = URL(urls[0])
      val conn: URLConnection = url.openConnection()
      val rd = InputStreamReader(conn.getInputStream())
      val allpage = StringBuilder()
      val buffer = BufferedReader(rd)
      var line = buffer.readLine()
      while (line != null) {
         allpage.append(line)
         line = buffer.readLine()
      }
      out = allpage.toString()
      return out
   }
}
```

Внешний вид приложения показан на рисунках ниже:



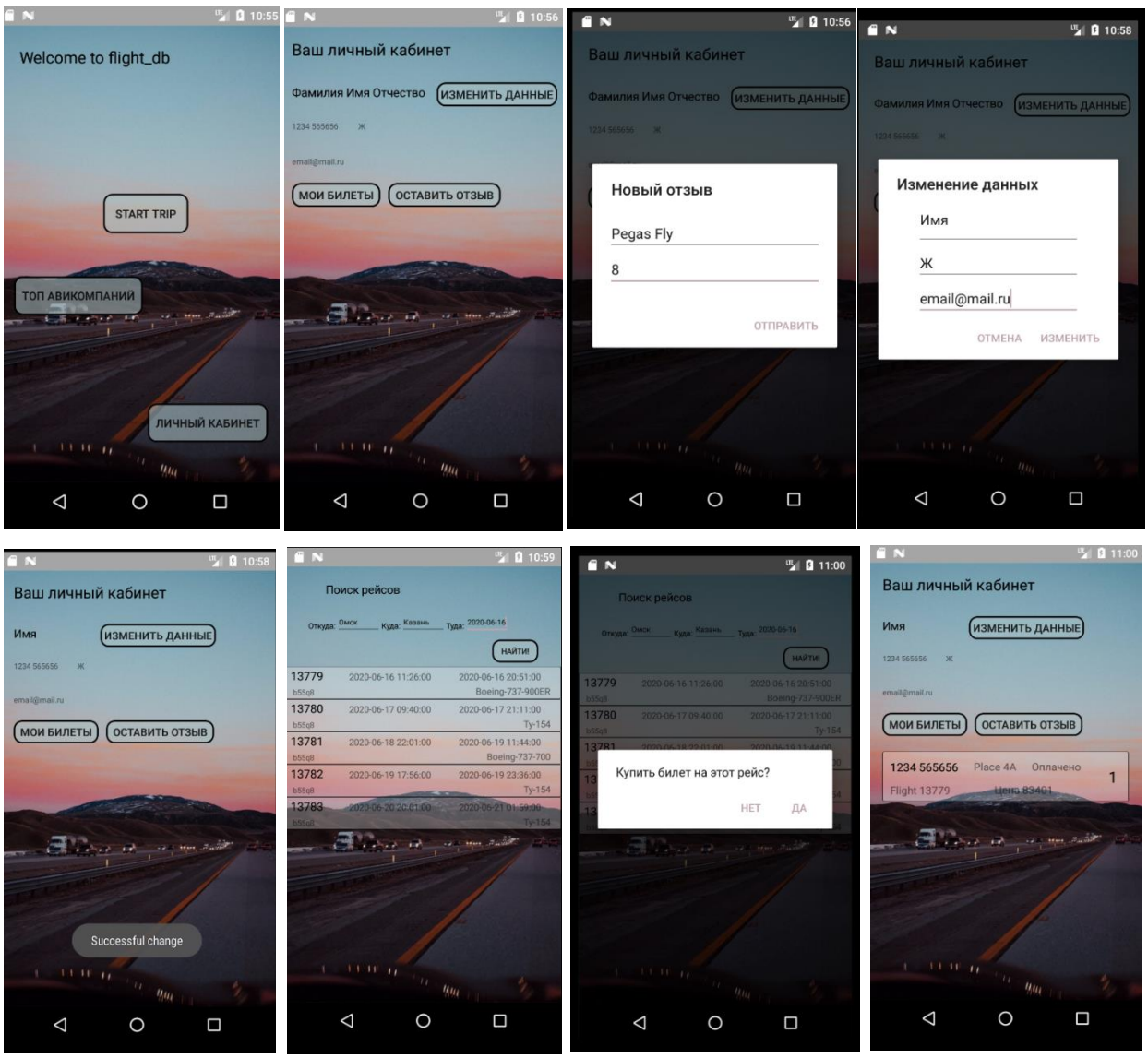


Рис.2.3.1. Внешний вид приложения

Главный экран приложения позволит перейти к регистрации пользователя или позволит войти пользователям, у которых уже есть login и пароль. Также можно открыть топ лучших авиакомпаний или перейти к поиску авиабилетов. При попытке купить билет без авторизации приложение попросит пользователя авторизоваться.

После успешной авторизации открыт доступ к личному кабинету, из которого можно оставить отзыв об авиакомпании, вывести список приобритенных пользователем билетов или изменить имя, пол или email.

Также при успешной авторизации можно купить билет на найденный полет. Полеты можно найти, указав два города (начальный и конечный пункт назначения) и дату полета. При некорректном указании имени города или указании несуществующего города, приложении сообщит о том, что такой город не найден.

Код приложения на GitLab: http://gitlab.icc.spbstu.ru/chernrina/db-task/tree/master/coursework/flightdb

3. Выводы

В ходе выполнения курсовой работы получены навыки по созданию веб-сервиса для взаимодействия с базой данных, по написанию API для него, а также по использованию созданного сервиса в Android-приложении. Данные навыки будут в дальнейшем полезны как в мобильной разработке, так и при работе с приложениями для других платформ.