Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет компьютерного проектирования

Кафедра проектирования информационно-компьютерных систем

Дисциплина «Объектно-ориентированное программирование и проектирование»

|  |  |
| --- | --- |
|  | «К ЗАЩИТЕ ДОПУСТИТЬ» |
|  | Руководитель курсового проекта  ассистент кафедры ПИКС  \_\_\_\_\_\_\_\_\_\_ В.В. Листратенко |
|  | \_\_\_.\_\_\_\_.2022 |

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовому проекту

на тему:

**«Мобильное программное средство “Погода” под OC Android»**

БГУИР КП 1-40 05 01-10

|  |  |
| --- | --- |
|  | Выполнил студент группы 114301  Корсаков Егор Андреевич  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись студента) |
|  | Курсовой проект представлен на проверку \_\_\_.\_\_\_\_.2022  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись студента) |

Минск 2022

**РЕФЕРАТ**

БГУИР КР 1-40 05 01-10 017 ПЗ

Корсаков Е.А. Мобильное программное средство «Погода» под ОС Android/Е. А. Корсаков– Минск: БГУИР, 2022. – 38 с.

Пояснительная записка 38 с., 13 рисунков, 8 источников, 3 приложения.

МОБИЛЬНОЕ ПРОГРАММНОЕ СРЕДСТВО «ПОГОДА» ПОД ОПЕРАЦИОННУЮ СИСТЕМУ ANDROID.

Предмет: средство для просмотра прогноза погоды.

Объект: получение прогноза погоды из интернета.

Цель: разработка программного средства, в котором пользователи смогут просматривать прогноз погоды нужных им городов, добавлять и удалять в список различные города.

Задачи: изучение литературных источников и существующих программных средств по теме, проектирование пользовательского интерфейса, реализация функций для выполнения задания курсового проекта, оценка результатов.

Методология проведения работы: в процессе решения поставленных задач использованы принципы получения и обработки данных из API.

Результаты работы: выполнен анализ литературных источников и существующих программных средств по теме; реализованы алгоритмы добавления и удаления городов, а также алгоритмы обработки данных полученных из API.

Программа предназначена для просмотра прогноза погоды.

Область применения результатов: личное использование.

**СОДЕРЖАНИЕ**

[1 Анализ исходных данных на курсовое проектирование 5](#_Toc121943310)

[1.1 Анализ исходных данных к курсовому проекту 5](#_Toc121943311)

[1.2 Обоснование и описание выбора языка программирования, средств разработки, используемых технологий и сторонних библиотек 5](#_Toc121943312)

[2 Проектирование и разработка программного средства 7](#_Toc121943313)

[2.1 Проектирование архитектуры и описание состояний программного средства 7](#_Toc121943314)

[2.2 Проектирование и разработка графического интерфейса 9](#_Toc121943315)

[2.3 Описание и реализация используемых в программном средстве алгоритмов 10](#_Toc121943316)

[3 Эксплуатация программного средства 11](#_Toc121943317)

[3.1 Ввод в эксплуатацию и обоснование минимальных технических требований к оборудованию 11](#_Toc121943318)

[3.2 Руководство по эксплуатации программного средства 11](#_Toc121943319)

[Приложение А (обязательное) Справка о проверке курсового проекта на заимствования 16](#_Toc121943320)

[Приложение Б (обязательное) Листинг программного кода 17](#_Toc121943321)

[Приложение В (обязательное) Ведомость курсового проекта 31](#_Toc121943322)

**ВВЕДЕНИЕ**

Сегодня практически каждый человек имеет мобильный телефон и использует его ежедневно. Он позволяет выполнять различные задачи быстро, легко и доступно.

У каждого есть свои планы на день, например: пойти на прогулку с друзьями или устроить пикник. Но будет обидно если во время данных мероприятий резко испортятся погодны условия и получить удовольствие будет уже тяжело. Поэтому было бы хорошо знать о погоде заранее, чтобы иметь возможность перенести прогулку на другой день. И потому можно посмотреть прогноз погоды в интернете.

И здесь нам может помочь наш телефон и приложение «Погода», которая обеспечит наиболее эффективный и легкий доступ к прогнозу, а также порадует глаз приятным интерфейсом.

В данной работе рассматривается разработка мобильного программного средства «Погода» под OC Android.

Цель курсового проекта – упростить, повысить эффективность к доступу прогноза погоды с помощью мобильного программного средства.

Для достижения цели курсового проекта необходимо решить следующие задачи:

* изучить предметную область;
* построить алгоритм программы;
* разработать интерфейс программы;
* разработать программное средство;
* протестировать программное средство, исправить обнаруженные ошибки.

В качестве отправной точки были рассмотрены некоторые сервисы погоды, такие как: Yandex Погода и Google Погода.

* + 1. АНАЛИЗ ИСХОДНЫХ ДАННЫХ НА КУРСОВОЕ ПРОЕКТИРОВАНИЕ
  1. Анализ исходных данных к курсовому проекту

Тема курсовой работы: разработка мобильного программного средства «Погода» под OC Android.

Язык программирования Java.

Среда разработки Android Studio.

Парадигма программирования – объектно-ориентированная.

Способ организации данных – классы.

Способ хранения данных – локальный файл.

К защите курсовой работы представляются:

1. Мобильное программное средство.
2. Схема алгоритма (формат А2/А3).
3. UML диаграмма классов (плакат, формат А2/А3).
4. Диаграмма состояний (плакат, формат А2/А3).
5. Структура графического пользовательского интерфейса (плакат, формат А2/А3) и пояснительная записка.

Текст пояснительной записки оформляется в соответствии со стандартом предприятия СТП 01–2017.

* 1. Обоснование и описание выбора языка программирования, средств разработки, используемых технологий и сторонних библиотек

Так как задача состоит в том, чтобы создать мобильное программное средство, было принято решение использовать среду разработки Android Studio. Существуют и другие среды разработки, однако у этой удобный графический интерфейс и средства отладки. Есть встроенный эмулятор, который позволяет в режиме реального времени проверить работоспособность программы. Также можно подключить и свой мобильный телефон через кабель или же сеть и проверять программное средство непосредственно на нем. Еще одна полезная вещь в Android Studio – инструмент анализа производительности, который называется MemoryMonitor. Он выдает всю информацию об использовании памяти, что дает возможность оптимизировать работу приложения. На сайте разработчиков Google отмечает, что планирует регулярно обновлять версию среды разработки.

В ходе разработки курсового проекта были выделение следующие преимущества:

* включены все «интеллектуальные» возможности по редактированию кода – авто дополнение, рефакторинг и анализ кода;
* встроенный SDK Manager;
* есть возможность визуального просмотра будущего приложения;
* среда разработки является технологиями компании Google;
* сборка проекта работает быстрее, чем в альтернативных средах разработки;
* управление локализацией и переводами;
* предпросмотр приложений на разных разрешениях, языках, и версиях API во встроенном эмуляторе.

Следующим шагом является выбор языка программирования. Исходя из средства разработки и операционной системы (Android), было решено использовать Java. Так как данный язык считается фундаментальным в разработке для Android.

Java поддерживается многими популярными средами разработки (IDE), включая Eclipse, Netbeans и JeBrains. Такие инструменты как Eclipse и Netbeans играют решающую роль в превращении Java в один из лучших языков программирования для мобильной разработки. Надёжный набор инструментов не только помогает в кодировании, но и позволяет влиять на отладку, которая необходима для исключения ошибок во время процесса разработки. Интегрированная среда сделала разработку на Java намного удобнее и быстрее. При использовании IDE легко искать и читать код, а также выполнять его рефакторинг.

Очередным преимуществом Java является кроссплатформенность. Другие языки программирования в той или иной степени привязаны к функциям программно-аппаратных платформ, но слоган Java гласит: «Напиши один раз – запускай, где угодно». Кроссплатформенность способствует распространению языка.

Хранение данных происходит посредством сохранения в файл. Так как задача стоит написать приложение «Погода», то нам необходимо хранить лишь города, которые выбрал пользователь для просмотра прогноза. Это можно сделать локально и с помощью файла, так как жизненно важных данных нам хранить не надо и соответственно защищать их тоже не нужно. Следовательно сохранение в файл – самый эффективный и рациональный способ в данном случае.

1. ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА
   1. Проектирование архитектуры и описание состояний программного средства

Процесс проектирования мобильного программного средства в Android Studio состоит из нескольких этапов:

* создание проекта в Android Studio;
* создание пользовательского интерфейса;
* написание логики активностей;
* тест приложения в эмуляторе.

Создание пользовательского интерфейса происходит посредством создания файлов расширения .xml в директории layout (рисунок 2.1).

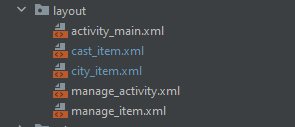


Рисунок 2.1 – Директория с файлами пользовательского интерфейса

В этой директории хранятся элементы графического интерфейса, которые создаются при помощи разметки (страницы, элементы страницы и прочее).

Еще одна директория, которая участвует в создании пользовательского интерфейса – это drawable (рисунок 2.2).

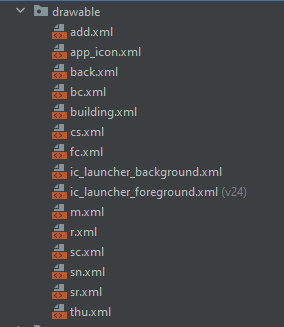


Рисунок 2.2 – Директория элементов пользовательского интерфейса

В этой директории хранятся элементы графического интерфейса, которые загружены для отображения (картинки, иконки, эффекты и прочее).

После того как графический интерфейс создан и настроен, переходим к этапу создание логики активностей. Каждая активность собирается из Java класса, в котором описывается логика и .xml файла с интерфейсом этой активности. На рисунке 2.3 представлена структура классов, в которых описывается логика.­



Рисунок 2.3 – Классы активностей

В этих классах происходит описание всего, что будет происходить на нашей активности, например:

* какой текст должен отображаться на отдельных элементах;
* что должно происходить при нажатии на кнопку;
* переход на другую активность;
* различные алгоритмы;
* какую разметку использовать для отображения пользователю.

Помимо классов активностей можно также создавать и обычные классы для работы над созданием приложения. Это могут быть различные классы объектов или же классы помогающие в работе с API или же обработкой данных. Пример этих классов представлен на рисунке 2.4.

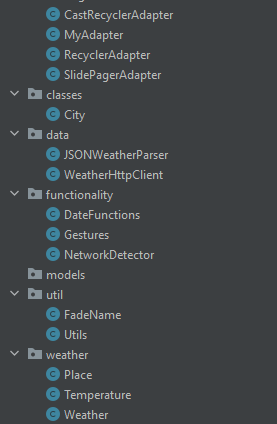


Рисунок 2.4 – Классы использующиеся в разработке приложения

После того как полностью прописана логика приложения, весь интерфейс настроен и корректно отображается мы переходим к этапу тестирования приложения в эмуляторе. На этом этапе выявляются недочеты интерфейса и ошибки.

* 1. Проектирование и разработка графического интерфейса

Пользовательский интерфейс мобильного приложения «Погода» состоит из 2-х основных экранов:

* экран отображения погоды городов;
* экран управления списком городов.

Каждый из этих экранов представляет собой .xml файл, в котором прописывается разметка экрана. Далее можно как через интерфейс, так и через код, настраивать наш экран и добавлять на него различные элементы (рисунок 2.5).

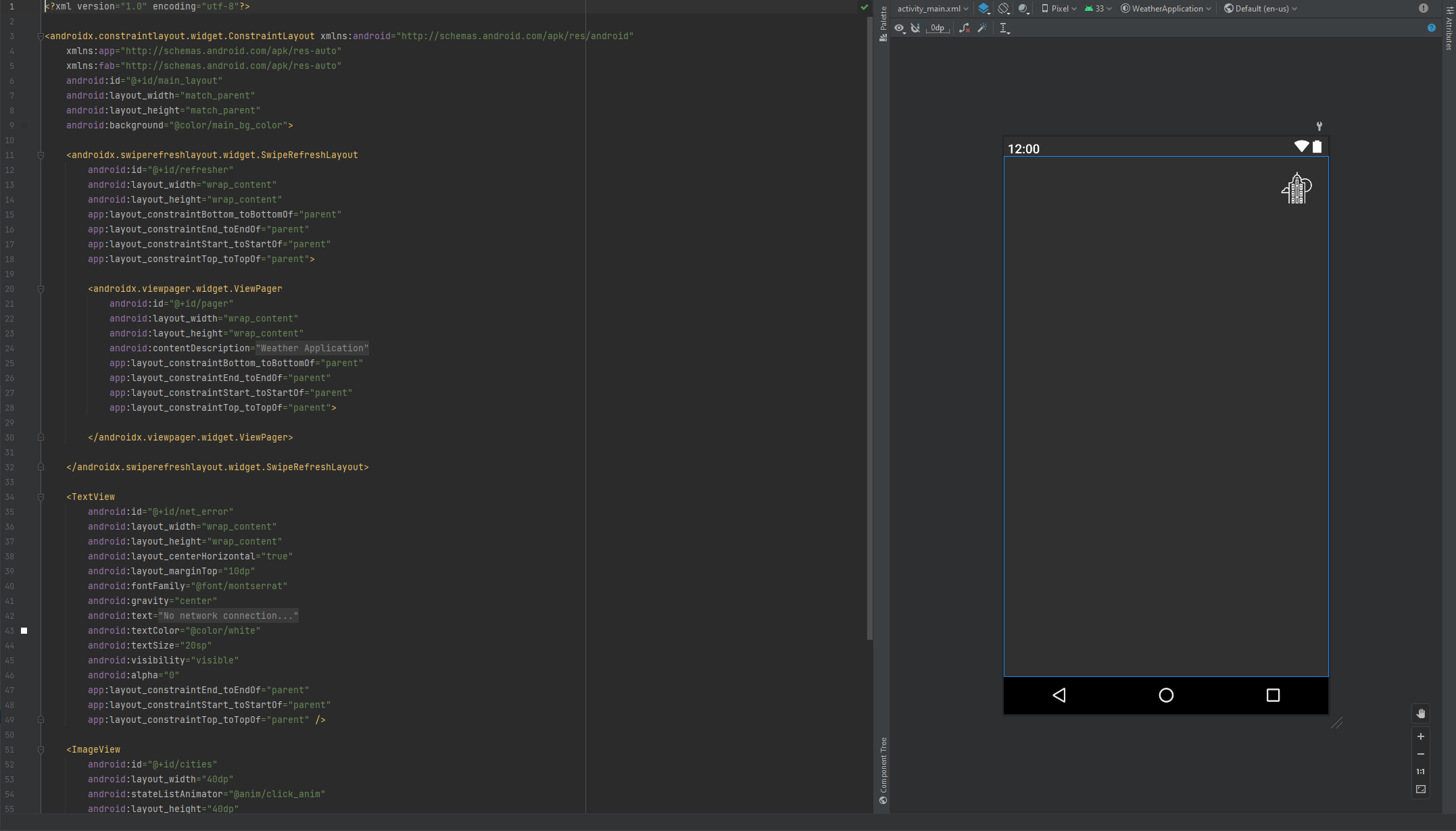


Рисунок 2.5 – Графический интерфейс главного экрана

На экране размещаются встроенные элементы (ImageView, ViewPager, TextView), а затем через код изменяются или дополняются.

* 1. Описание и реализация используемых в программном средстве алгоритмов

Основные алгоритмы, которые были использованы при разработке данного программного средства – это алгоритмы по обработке и использовании данных получаемых из API, а также по работе с редактированием списка городов.

Получение и обработка данных из API происходит при помощи классов WeatherHttpClient и JSONWeatherParser. Причем первый класс устанавливает соединение с API и получает данные, а второй обрабатывает полученные данные и преобразует их из JSON объекта в базовые типы и на основе их создает объект класса Weather.

На экране редактирования списка городов мы можем перемещать города между собой, удалять их и добавлять новые. При чем перемещение и удаление реализовано посредством жестов, а добавление по нажатию кнопки.

1. ЭКСПЛУАТАЦИЯ ПРОГРАММНОГО СРЕДСТВА
   1. Ввод в эксплуатацию и обоснование минимальных технических требований к оборудованию

Для того, чтобы начать использовать приложение на телефоне необходимо создать .apk файл. Он автоматически создается при сборке проекта. Далее его необходимо перенести на мобильное устройство и запустить. При успешном запуске можно переходить к эксплуатации программного средства.

Приложение предназначено для устройств версии SDK: API LEVEL 21. Также для получения прогноза погоды необходим доступ к интернету.

* 1. Руководство по эксплуатации программного средства

После запуска приложения у пользователя появляется экран с погодой городов (рисунок 3.1).



Рисунок 3.1 – Экран прогноза погоды

На нем отображается текущая погода, город и страна, почасовой прогноз погоды на день и кнопка перехода на другой экран.

При нажатии на эту кнопку у пользователя появляется возможность редактировать список его городов:

* + - добавлять другой город;
    - перемещать их между собой;
    - удалять город.

Интерфейс этого экрана представлен на рисунке 3.2.

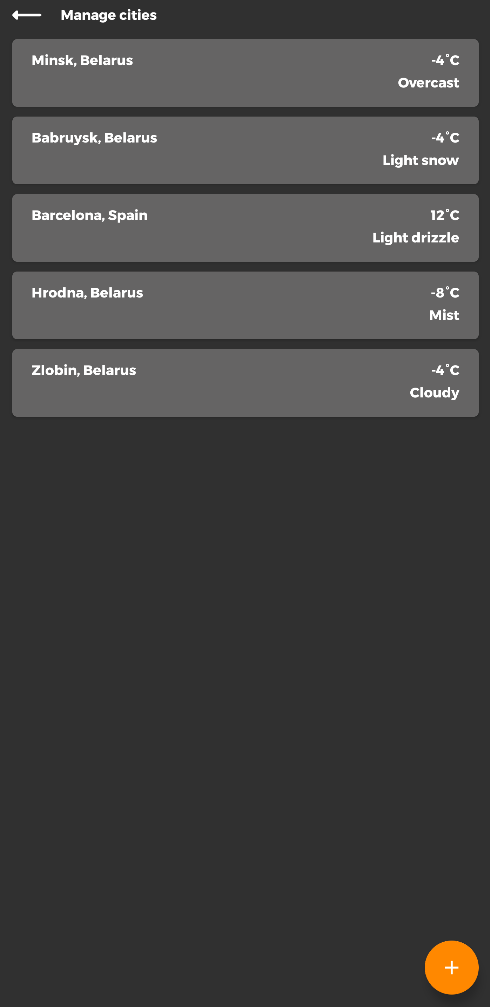


Рисунок 3.2 – Экран редактирования списка городов

Для перемещения городов между собой, нужно удерживать несколько секунд и затем перемещать по вертикали город на нужную вам позицию.

Для добавления города, необходимо нажать на кнопку в правом нижнем углу экрана и в появившемся окне написать название города, который вы хотите добавить (рисунок 3.3).

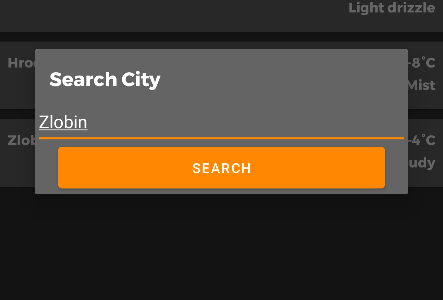


Рисунок 3.3 – Окно добавление города

Для удаления города достаточно сделать свайп влево или вправо, при этом вас уведомят о том, что вы удалили город и в течении нескольких секунд вы сможете вернуть город на ту же позицию.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы было создано мобильное программное средство «Погода» под ОС Android.

Для создания данного приложения необходимо было ознакомиться со средой разработки Android Stuido и её компонентами, а также освоить работу с API. Были выполнены такие задачи как: проектирование пользовательского интерфейса, реализация функционала поставленного курсовой работой, тестирование приложения, а также его оценка. Был реализован такой функционал как:

* получение и обработка данных из API;
* обновление прогноза погоды;
* просмотр прогноза погоды на текущий момент;
* просмотр прогноза погоды по часам;
* отмена удаления города и возвращение его на ту же позицию;
* перемещение городов между собой;
* добавление городов;
* удаление городов.

В результате выполнения курсовой работы было получено работающее мобильное приложение. Все требования к приложению были реализованы.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Разработка на Android Studio / Урок #2 – Дизайн основного окна [Электронный ресурс].  –  Режим доступа: https://www.youtube.com/watch?v=VZDGQ88M5So/. Дата доступа: 01.10.2022.

[2] Разработка на Android Studio / Урок #3 – Категории товаров [Электронный ресурс].  –  Режим доступа: https://www.youtube.com/watch?v=PG2xh1sOrT8/. Дата доступа: 01.10.2022.

[3] Разработка на Android Studio / Урок #6 – Переход между страницами с анимацией [Электронный ресурс].  –  Режим доступа: https://www.youtube.com/watch?v=mJT8ePVIoss&t/. Дата доступа: 01.10.2022.

[4] How to connect android to restful api. [Электронный ресурс] – Режим доступа: https://stackoverflow.com/questions/31023352/how-to-connect-android-to-restful-api/. Дата доступа: 06.11.2022.

[5] RecyclerView and SwipeRefreshLayout [Электронный ресурс]. – Режим доступа: https://stackoverflow.com/questions/25178329/recyclerview-and-swiperefreshlayout/. Дата доступа: 07.11.2022.

[6] How to change background color of the snackbar [Электронный ресурс]? – Режим доступа: https://stackoverflow.com/questions/34020891/how-to-change-background-color-of-the-snackbar/. Дата доступа: 07.11.2022.

ПРИЛОЖЕНИЕ А

(обязательное)

Справка о проверке курсового проекта на плагиат

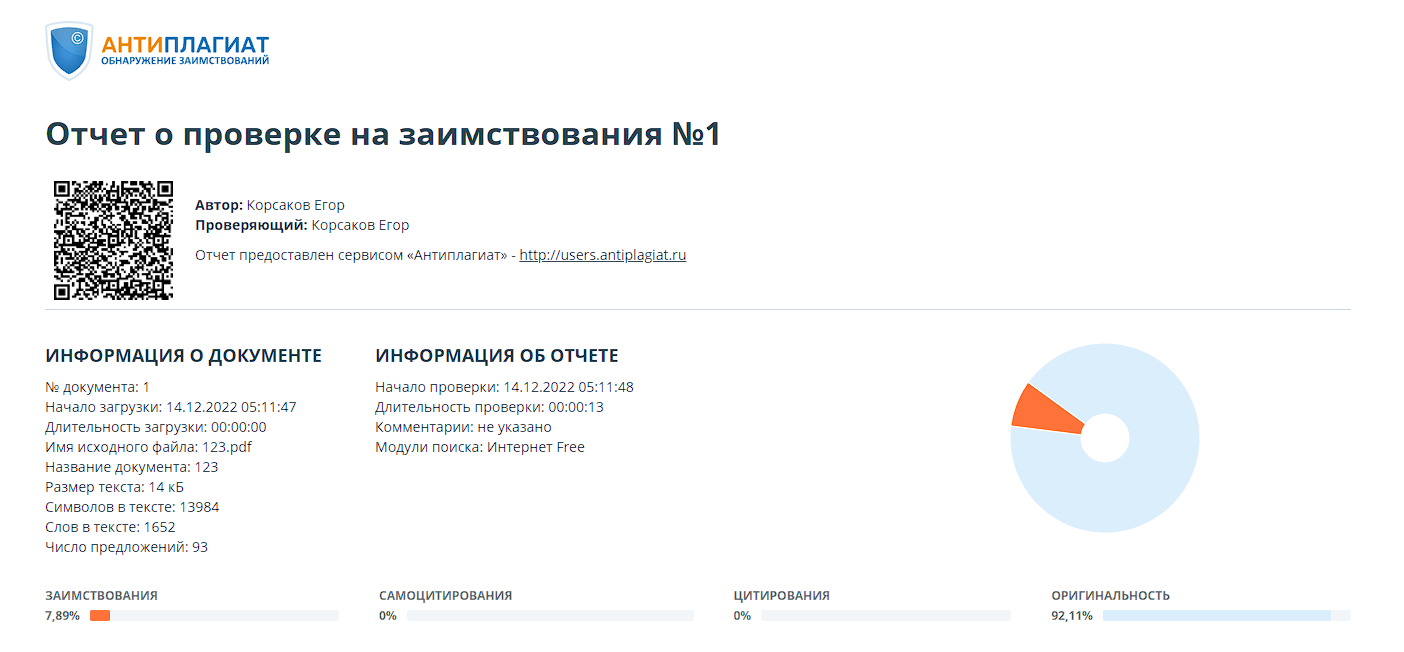


Рисунок А.1 – Отчёт работы системы проверки на заимствования

ПРИЛОЖЕНИЕ Б

(обязательное)

Листинг программного кода

Класс MainActivity.java

package com.example.weatherapplication;

public class MainActivity extends AppCompatActivity{

private ArrayList<City> offlineCities;

private ViewPager viewPager;

private TextView netError;

private SwipeRefreshLayout refresher;

private ImageView city;

@SuppressLint("MissingInflatedId")

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

viewPager = findViewById(R.id.pager);

viewPager.setPageTransformer(false, new FadeName());

offlineCities = new ArrayList<>();

refresher = findViewById(R.id.refresher);

netError = findViewById(R.id.net\_error);

city = findViewById(R.id.cities);

try{

loadData();

}catch (Exception e) {}

updateData();

city.setOnClickListener(view -> {

view.startAnimation(AnimationUtils.loadAnimation(MainActivity.this, R.anim.click\_anim));

Intent intent = new Intent(MainActivity.this, ManageActivity.class);

intent.putExtra("cities", (Serializable) offlineCities);

startActivityForResult(intent, 100);

});

setViewPager();

refresher.setOnRefreshListener(this::updateData);

}

private void setViewPager() {

viewPager.setAdapter(new MyAdapter(this, offlineCities, refresher));

viewPager.addOnPageChangeListener(new ViewPager.OnPageChangeListener() {

@Override

public void onPageScrolled(int position, float positionOffset, int positionOffsetPixels) {

}

@Override

public void onPageSelected(int position) {

}

@Override

public void onPageScrollStateChanged(int state) {

if(refresher != null && !refresher.isRefreshing())

{

refresher.setEnabled(state == ViewPager.SCROLL\_STATE\_IDLE);

}

}

});

}

@Override

protected void onActivityResult(int requestCode, int resultCode, @Nullable Intent data) {

super.onActivityResult(requestCode, resultCode, data);

if(resultCode == RESULT\_OK) {

ArrayList<City> cities = (ArrayList<City>) data.getSerializableExtra("cities");

offlineCities.clear();

offlineCities.addAll(cities);

viewPager.getAdapter().notifyDataSetChanged();

try {

saveData();

} catch (IOException e) {

e.printStackTrace();

}

}

}

public void updateData()

{

refresher.setRefreshing(true);

if (NetworkDetector.isConnected(this)) {

if (!offlineCities.isEmpty()) {

for (City city : offlineCities) {

renderWeatherData(city.getName());

}

}

else

{

refresher.setRefreshing(false);

}

}

else{

refresher.setRefreshing(false);

netError.animate().setDuration(500).alpha(1);

}

}

private void saveData() throws IOException {

FileOutputStream file = openFileOutput("data", MODE\_PRIVATE);

ObjectOutputStream object = new ObjectOutputStream(file);

object.writeObject(offlineCities);

object.close();

file.close();

}

private void loadData() throws IOException, ClassNotFoundException {

FileInputStream file = openFileInput("data");

ObjectInputStream object = new ObjectInputStream(file);

offlineCities = (ArrayList<City>) object.readObject();

object.close();

file.close();

viewPager.getAdapter().notifyDataSetChanged();

}

public void renderWeatherData(String city)

{

WeatherTask weatherTask = new WeatherTask();

weatherTask.execute(city);

}

private class WeatherTask extends AsyncTask<String, Void, Weather>

{

@Override

protected Weather doInBackground(String... params) {

try {

String data = new WeatherHttpClient().getWeatherData(params[0]);

return JSONWeatherParser.getWeather(data);

} catch (JSONException e) {} catch (IOException e) {

e.printStackTrace();

}

return null;

}

@RequiresApi(api = Build.VERSION\_CODES.N)

@Override

protected void onPostExecute(Weather weather) {

super.onPostExecute(weather);

try {

getPostData(weather);

netError.animate().setDuration(500).alpha(0);

}catch (Exception e)

{

netError.animate().setDuration(500).alpha(1);

}finally {

refresher.setRefreshing(false);

}

}

}

@RequiresApi(api = Build.VERSION\_CODES.N)

public void getPostData(Weather weather)

{

offlineCities.stream().filter(x -> x.getName().equals(weather.getPlace().getCity())).forEach(x -> x.setData(weather));

viewPager.getAdapter().notifyDataSetChanged();

try {

saveData();

} catch (Exception e){}

}

public static String formatStatus(String status)

{

return status.substring(0, 1).toUpperCase(Locale.ROOT) + status.substring(1);

}

}

Класс ManageActivity.java

package com.example.weatherapplication;

public class ManageActivity extends AppCompatActivity {

private RecyclerView recycler;

private ImageView back;

private FloatingActionButton add;

private ArrayList<City> cities;

private ItemTouchHelper helper;

private AlertDialog dialog;

@Override

protected void onCreate(@Nullable Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.manage\_activity);

cities = (ArrayList<City>) getIntent().getSerializableExtra("cities");

back = findViewById(R.id.back);

back.setOnClickListener(view -> {

view.startAnimation(AnimationUtils.loadAnimation(ManageActivity.this, R.anim.click\_anim));

onBackPressed();

});

recycler = findViewById(R.id.recycler);

setRecycler();

setHelper();

setDialog();

add = findViewById(R.id.add);

add.setOnClickListener(view -> {

dialog.show();

});

}

@Override

public void onBackPressed() {

Intent intent = new Intent();

intent.putExtra("cities", (Serializable) cities);

setResult(RESULT\_OK, intent);

super.onBackPressed();

}

private void setDialog()

{

LayoutInflater inflater = this.getLayoutInflater();

View view = inflater.inflate(R.layout.dialog\_item,null);

dialog = (new AlertDialog.Builder(this)).create();

dialog.setView(view);

EditText searchText = view.findViewById(R.id.search\_text);

Button searchButton = view.findViewById(R.id.search\_button);

searchButton.setOnClickListener(v -> {

renderWeatherData(searchText.getText().toString());

});

}

public void setHelper()

{

helper = new ItemTouchHelper(new ItemTouchHelper.SimpleCallback(ItemTouchHelper.UP | ItemTouchHelper.DOWN, ItemTouchHelper.START | ItemTouchHelper.END) {

@Override

public boolean onMove(@NonNull RecyclerView recyclerView, @NonNull RecyclerView.ViewHolder drag, @NonNull RecyclerView.ViewHolder drop) {

int dragPosition = drag.getAdapterPosition();

int dropPosition = drop.getAdapterPosition();

Collections.swap(cities, dragPosition, dropPosition);

recycler.getAdapter().notifyItemMoved(dragPosition, dropPosition);

return false;

}

@Override

public void onSwiped(@NonNull RecyclerView.ViewHolder viewHolder, int direction) {

int position = viewHolder.getAdapterPosition();

City city = cities.get(position);

cities.remove(position);

recycler.getAdapter().notifyItemRemoved(position);

Snackbar snackbar = Snackbar.make(recycler, "City Removed", Snackbar.LENGTH\_LONG).

setAction("UNDO", view -> {

setCity(city, position);

recycler.getAdapter().notifyItemInserted(position);

});

ViewGroup contentLay = (ViewGroup) snackbar.getView().findViewById(com.google.android.material.R.id.snackbar\_text).getParent();

View view = ManageActivity.this.getLayoutInflater().inflate(R.layout.progressbar\_item, null);

TextView tv = view.findViewById(R.id.timer);

new CountDownTimer(5100, 1000) {

public void onTick(long mills) {

tv.setText(mills/1000 + "");

}

public void onFinish() {

}

}.start();

contentLay.addView(view, 0);

snackbar.setTextColor(getResources().getColor(R.color.white));

snackbar.setActionTextColor(Color.parseColor("#FFFF8800"));

snackbar.setBackgroundTint(getResources().getColor(R.color.card\_bg));

snackbar.setAnimationMode(BaseTransientBottomBar.ANIMATION\_MODE\_SLIDE);

snackbar.setDuration(4600);

snackbar.show();

}

});

helper.attachToRecyclerView(recycler);

}

private void setCity(City city, int position)

{

ArrayList<City> newCities = new ArrayList<>();

int i = 0;

if(position == cities.size())

{

cities.add(city);

}

else {

for (City c : cities) {

if (i != position) {

newCities.add(c);

} else {

newCities.add(city);

newCities.add(c);

}

i++;

}

cities.clear();

cities.addAll(newCities);

}

}

public void setRecycler()

{

RecyclerView.LayoutManager layoutManager = new LinearLayoutManager(this, LinearLayoutManager.VERTICAL, false);

recycler.setLayoutManager(layoutManager);

RecyclerAdapter adapter = new RecyclerAdapter(this, cities);

recycler.setAdapter(adapter);

}

public void renderWeatherData(String city)

{

WeatherTask weatherTask = new WeatherTask();

weatherTask.execute(city);

}

private class WeatherTask extends AsyncTask<String, Void, Weather>

{

@Override

protected Weather doInBackground(String... params) {

try {

String data = new WeatherHttpClient().getWeatherData(params[0]);

return JSONWeatherParser.getWeather(data);

} catch (JSONException e) {} catch (IOException e) {

e.printStackTrace();

}

return null;

}

@Override

protected void onPostExecute(Weather weather) {

super.onPostExecute(weather);

try {

City city = new City(weather);

if(!containsCity(city)) {

cities.add(city);

recycler.getAdapter().notifyItemInserted(cities.size() - 1);

}

else

{

Toast.makeText(ManageActivity.this, "This city already exists", Toast.LENGTH\_SHORT).show();

}

}catch (NullPointerException e){

Toast.makeText(ManageActivity.this, "Something went wrong. Check your internet connection or city name", Toast.LENGTH\_SHORT).show();

}catch (Exception e)

{

Toast.makeText(ManageActivity.this, "There is no such city", Toast.LENGTH\_SHORT).show();

}

}

}

private boolean containsCity(City city)

{

for (City c : cities)

{

if(c.getName().equals(city.getName()))

return true;

}

return false;

}

}

Класс JSONWeatherParser.java

public class JSONWeatherParser {

public static Weather getWeather(String data) throws JSONException {

try {

JSONObject object = new JSONObject(data);

JSONObject locObj = object.getJSONObject("location");

Place place = new Place(locObj.getString("country"), locObj.getString("name"));

JSONObject curObj = object.getJSONObject("current");

JSONObject condObj = curObj.getJSONObject("condition");

JSONObject fcastObj = object.getJSONObject("forecast");

JSONObject dayObj = fcastObj.getJSONArray("forecastday").getJSONObject(0).getJSONObject("day");

Temperature temperature = new Temperature((float) dayObj.getDouble("mintemp\_c"),

(float) dayObj.getDouble("maxtemp\_c"), (float) curObj.getDouble("temp\_c"));

JSONArray castArr = fcastObj.getJSONArray("forecastday").getJSONObject(0).getJSONArray("hour");

ArrayList<Integer> castTemp = new ArrayList<>();

ArrayList<String> castTime = new ArrayList<>();

ArrayList<Integer> castIcon = new ArrayList<>();

for (int i=0; i<castArr.length(); i++)

{

castTemp.add((int) Math.ceil(castArr.getJSONObject(i).getDouble("temp\_c")));

castTime.add(castArr.getJSONObject(i).getString("time").substring(10));

castIcon.add(castArr.getJSONObject(i).getJSONObject("condition").getInt("code"));

}

Weather weather = new Weather(place, condObj.getString("text"), temperature, condObj.getInt("code"),

castTemp, castTime, castIcon);

return weather;

} catch (Exception e){}

return null;

}

}

Класс Weather.java

package com.example.weatherapplication.weather;

public class Weather {

private Place place;

private String status;

private Temperature temperature;

private int icon;

private ArrayList<Integer> castTemp;

private ArrayList<String> castTime;

private ArrayList<Integer> castIcon;

public Weather(Place place, String status, Temperature temperature, int icon, ArrayList<Integer> castTemp,

ArrayList<String> castTime, ArrayList<Integer> castIcon) {

this.place = place;

this.status = status;

this.temperature = temperature;

this.icon = icon;

this.castIcon = castIcon;

this.castTemp = castTemp;

this.castTime = castTime;

}

public Place getPlace() {

return place;

}

public String getStatus() {

return status;

}

public Temperature getTemperature() {

return temperature;

}

public int getIcon() {

return icon;

}

public ArrayList<Integer> getCastTemp() {

return castTemp;

}

public ArrayList<String> getCastTime() {

return castTime;

}

public ArrayList<Integer> getCastIcon() {

return castIcon;

}

}

Класс WeatherHttpClient.java

package com.example.weatherapplication.data;

public class WeatherHttpClient {

public String readAll(Reader rd) throws IOException

{

StringBuilder sb = new StringBuilder();

int cp;

while ((cp = rd.read()) != -1){

sb.append((char) cp);

}

return sb.toString();

}

public String getWeatherData(String place) throws IOException, JSONException {

InputStream inputStream = new URL(Utils.BASE\_URL + place + Utils.API\_KEY).openStream();

try{

BufferedReader rd = new BufferedReader(new InputStreamReader(inputStream, Charset.forName("UTF-8")));

String jsonText = readAll(rd);

JSONObject json = new JSONObject(jsonText);

return json.toString();

}finally {

inputStream.close();

}

}

}

Класс DateFunctions.java

package com.example.weatherapplication.functionality;

public class DateFunctions {

public static String getDate()

{

Date date = new Date();

Calendar cal = Calendar.getInstance();

return getMonth(date) + " " + cal.get(Calendar.DAY\_OF\_MONTH) + " " + getDay(date);

}

private static String getDay(Date date)

{

Map<Integer, String> day = new HashMap<Integer, String>();

day.put(1, "Mon");

day.put(2, "Tue");

day.put(3, "Wed");

day.put(4, "Thu");

day.put(5, "Fri");

day.put(6, "Sat");

day.put(0, "Sun");

return day.get(date.getDay());

}

private static String getMonth(Date date)

{

Map<Integer, String> month = new HashMap<Integer, String>();

month.put(0, "Jan");

month.put(1, "Feb");

month.put(2, "Mar");

month.put(3, "Apr");

month.put(4, "May");

month.put(5, "Jun");

month.put(6, "Jul");

month.put(7, "Aug");

month.put(8, "Sep");

month.put(9, "Oct");

month.put(10, "Nov");

month.put(11, "Dec");

return month.get(date.getMonth());

}

}

Класс NetworkDetector.java

package com.example.weatherapplication.functionality;

public class NetworkDetector {

public static boolean isConnected(Context context)

{

ConnectivityManager connectivityManager =

(ConnectivityManager) context.getSystemService(Service.CONNECTIVITY\_SERVICE);

if (connectivityManager != null)

{

NetworkInfo info = connectivityManager.getActiveNetworkInfo();

if (info != null && info.isConnected())

{

return true;

}

}

return false;

}

}

Класс ViewPagerAdapter.java

public class ViewPagerAdapter extends PagerAdapter {

private Context context;

private ArrayList<City> cities;

private SwipeRefreshLayout refresher;

public ViewPagerAdapter(Context context, ArrayList<City> cities, SwipeRefreshLayout refresher) {

this.context = context;

this.cities = cities;

this.refresher = refresher;

}

@Override

public int getCount() {

return cities.size();

}

@Override

public boolean isViewFromObject(@NonNull View view, @NonNull Object object) {

return view.equals(object);

}

@NonNull

@Override

public Object instantiateItem(@NonNull ViewGroup container, int position) {

View view = LayoutInflater.from(context).inflate(R.layout.city\_item, container, false);

TextView name, temperature, tempRange, status, date;

ImageView icon;

RecyclerView castRecycler;

name = view.findViewById(R.id.city\_name);

temperature = view.findViewById(R.id.temperature);

tempRange = view.findViewById(R.id.temp\_range);

status = view.findViewById(R.id.status);

date = view.findViewById(R.id.date);

icon = view.findViewById(R.id.weather\_icon);

castRecycler = view.findViewById(R.id.cast);

name.setText(cities.get(position).getName() + ", " + cities.get(position).getCountry());

temperature.setText(String.valueOf(cities.get(position).getDegrees()));

tempRange.setText(cities.get(position).getMaxDegrees() + "°C / " + cities.get(position).getMinDegrees() + "°C");

status.setText(cities.get(position).getStatus());

date.setText(DateFunctions.getDate());

RecyclerView.LayoutManager layoutManager = new LinearLayoutManager(view.getContext(), LinearLayoutManager.HORIZONTAL, false);

castRecycler.setLayoutManager(layoutManager);

CastRecyclerAdapter adapter = new CastRecyclerAdapter(view.getContext(), cities.get(position).getCastTime(),

cities.get(position).getCastTemp(), cities.get(position).getCastIcon());

castRecycler.setAdapter(adapter);

castRecycler.setOnScrollListener(new RecyclerView.OnScrollListener() {

@Override

public void onScrollStateChanged(@NonNull RecyclerView recyclerView, int newState) {

super.onScrollStateChanged(recyclerView, newState);

if(refresher != null && !refresher.isRefreshing())

{

refresher.setEnabled(newState == RecyclerView.SCROLL\_STATE\_IDLE);

}

}

});

int cityIcon = cities.get(position).getIcon();

icon.setImageResource(getIcon(cityIcon));

container.addView(view, 0);

return view;

}

private static boolean isHere(int cityIcon, int... params)

{

for (int param : params)

{

if(cityIcon == param)

return true;

}

return false;

}

public static int getIcon(int cityIcon)

{

int iconValue = 0;

if(isHere(cityIcon, 1000))

{

iconValue = R.drawable.cs;

}

else if(isHere(cityIcon, 1003))

{

iconValue = R.drawable.fc;

}

else if(isHere(cityIcon, 1006))

{

iconValue = R.drawable.sc;

}

else if(isHere(cityIcon, 1009))

{

iconValue = R.drawable.bc;

}

else if(isHere(cityIcon, 1030, 1135, 1147))

{

iconValue = R.drawable.m;

}

else if(isHere(cityIcon, 1063, 1069, 1072, 1150, 1153, 1168, 1171, 1180, 1183, 1198, 1204))

{

iconValue = R.drawable.r;

}

else if(isHere(cityIcon, 1186, 1189, 1192, 1195, 1201, 1207, 1240, 1243, 1246, 1249, 1252))

{

iconValue = R.drawable.sr;

}

else if(isHere(cityIcon, 1087, 1273, 1276, 1279, 1282))

{

iconValue = R.drawable.thu;

}

else

{

iconValue = R.drawable.sn;

}

return iconValue;

}

@Override

public void destroyItem(@NonNull ViewGroup container, int position, @NonNull Object object) {

container.removeView((View) object);

}

@Override

public int getItemPosition(@NonNull Object object) {

return POSITION\_NONE;

}

}

Класс RecyclerAdapter.java

package com.example.weatherapplication.adapter;

public class RecyclerAdapter extends RecyclerView.Adapter<RecyclerAdapter.CityViewHolder> {

private Context context;

private ArrayList<City> cities;

public RecyclerAdapter(Context context, ArrayList<City> cities) {

this.context = context;

this.cities = cities;

}

@NonNull

@Override

public CityViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {

View cityItems = LayoutInflater.from(context).inflate(R.layout.manage\_item, parent, false);

return new CityViewHolder(cityItems);

}

@Override

public void onBindViewHolder(@NonNull CityViewHolder holder, int position) {

holder.name.setText(cities.get(position).getName() + ", " + cities.get(position).getCountry());

holder.temperature.setText(cities.get(position).getDegrees() + "°C");

holder.status.setText(cities.get(position).getStatus());

}

@Override

public int getItemCount() {

return cities.size();

}

public static final class CityViewHolder extends RecyclerView.ViewHolder {

TextView name, temperature, status;

public CityViewHolder(@NonNull View itemView) {

super(itemView);

name = itemView.findViewById(R.id.city\_name);

temperature = itemView.findViewById(R.id.temperature);

status = itemView.findViewById(R.id.status);

}

}

}

ПРИЛОЖЕНИЕ В

(обязательное)

Ведомость курсового проекта