**Министерство образования и науки Российской Федерации**

Федеральное государственное автономное образовательное учреждение

высшего профессионального образования

**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ**

**ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Инженерная школа информационных технологий и робототехники

Отделение информационных технологий

Направление – Информационные системы и технологии

**Лабораторная работа №5**

по дисциплине: Программирование мобильных устройств и встраиваемых систем

|  |  |  |  |
| --- | --- | --- | --- |
| Выполнил: студент гр. 8И6А | \_\_\_\_\_\_\_\_\_\_\_ | Н. П. Шкулов | |
|  |  | |  |
| Проверил: ассистент ОИТ | \_\_\_\_\_\_\_\_\_\_\_ | | А. В. Погребной |

Томск 2020 г.

# Задание

Общее: на все фоновые запросы должен отображаться индикатор загрузки (ProgressBar), при ошибке пользователю должна предоставляться возможность переотправить запрос (кнопка «Повтор»). При повороте экрана или переход на другой экран, его состояние не должно теряться, запросы не должны переотправляться.

Реализовать поиск по репозиториям GitHub. В тулбаре расположен SearchView. При вводе строки длиной больше чем 2 символа искать на GitHub репозитории содержащие эти символы и выводить их в списке. Каждый элемент списка содержит полное название репозитория и его описание (ограничение 5 строк). При нажатии на элемент списка открывается новая Activity. На ней выводится информация по репозиторию: название репозитория в тулбаре, полное описание репозитория, список коммитов, список issue. Реализовать обновление информации по репозиторию движением pull-to-refresh (SwipeRefreshLayout).

# Ход работы

Как и в предыдущих лабораторных работах, вывод информации в виде списка (в данном случае, репозиториев GitHub) в основной активити будет осуществляться посредством RecyclerView.

Также была создана активити для вывода информации о выбранном репозитории. В этой активити все элементы размещены внутри SwipeRefreshLayout (ID: swipe\_container), позволяющем обновлять информацию посредством pull-to-refresh. Внутри свайп-контейнера располагается NestedScrollView, в котором выводится вся информация о репозитории. Чтобы иметь возможность поместить внутрь несколько дочерних элементов, внутри него находится LinearLayout. В этом лэйауте размещены:

1. Полное описание репозитория (TextView ID lab5\_repo\_description);
2. Список коммитов (NestedScrollView ID commits);
3. Список issues (NestedScrollView ID issues).

<?xml version="1.0" encoding="utf-8"?>

<androidx.swiperefreshlayout.widget.SwipeRefreshLayout

xmlns:android="http://schemas.android.com/apk/res/android"

android:id="@+id/swipe\_container"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent">

<androidx.core.widget.NestedScrollView

xmlns:android="http://schemas.android.com/apk/res/android"

android:id="@+id/lab5\_repo\_scroll"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:padding="16dp">

<LinearLayout

xmlns:android="http://schemas.android.com/apk/res/android"

android:id="@+id/lab5\_repo\_information\_list"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:orientation="vertical">

<TextView

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:id="@+id/lab5\_repo\_description">

</TextView>

<TextView

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:text="Коммиты"

android:paddingBottom="12dp"

android:paddingTop="12dp">

</TextView>

<androidx.core.widget.NestedScrollView

android:layout\_width="match\_parent"

android:layout\_height="200dp"

android:id="@+id/commits">

<LinearLayout

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:id="@+id/commits\_list"

android:orientation="vertical">

</LinearLayout>

</androidx.core.widget.NestedScrollView>

<TextView

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:text="Issues"

android:paddingBottom="12dp"

android:paddingTop="1dp">

</TextView>

<androidx.core.widget.NestedScrollView

android:layout\_width="match\_parent"

android:layout\_height="200dp"

android:id="@+id/issues">

<LinearLayout

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:id="@+id/issues\_list"

android:orientation="vertical">

</LinearLayout>

</androidx.core.widget.NestedScrollView>

</LinearLayout>

</androidx.core.widget.NestedScrollView>

</androidx.swiperefreshlayout.widget.SwipeRefreshLayout>

Для вывода в списки нужно было создать файлы с xml-разметкой для отдельных элементов: название репозитория, описание репозитория, коммит, issue. Для примера представлен commit\_item: он содержит два TextView, один из которых хранит дату коммита, а другой – сообщение.

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout

xmlns:android="http://schemas.android.com/apk/res/android"

android:id="@+id/lab5\_commit\_list\_item"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:orientation="vertical">

<TextView

xmlns:android="http://schemas.android.com/apk/res/android"

android:id="@+id/commit\_date"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:paddingLeft="12dp"

android:paddingTop="12dp"

/>

<TextView

xmlns:android="http://schemas.android.com/apk/res/android"

android:id="@+id/commit\_message"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:paddingLeft="12dp"

/>

</LinearLayout>

Также для репозиториев, коммитов и issues были созданы собственные классы (представлен пример класса Commit):

public class Commit {

public String date;

public String message;

}

По аналогии с лабораторной работой №3, для хранения репозиториев был создан кэш RepoCache, в результате чего найденные репозитории будут сохраняться в памяти.

Для выполнения поиска репозиториев было внесено изменение в пример к данной лабораторной: в главную активити была добавлена статическая переменная Query, обновляющаяся при вводе поискового запроса, которая затем передается в качестве параметра задаче поиска.

@Override

@WorkerThread

protected List<Repo> executeInBackground() throws Exception {

String response = search(Lab5Activity.Query);

return parseSearch(response);

}

//запрос к API с возвращением тела ответа

private String search(String query) throws Exception {

Request request = new Request.Builder()

.url("https://api.github.com/search/repositories?q=" + query)

.build();

Response response = getHttpClient().newCall(request).execute();

if (response.code() != 200) {

throw new Exception("api returned unexpected http code: " + response.code());

}

return response.body().string();

}

При парсинге ответа от API из него выделяются полное название репозитория, описание и ссылка, которая будет использована далее для получения полной информации о репозитории, если пользователь нажмет на него в списке.

private List<Repo> parseSearch(String response) throws JSONException {

JSONObject responseJson = new JSONObject(response);

List<Repo> repos = new ArrayList<>();

JSONArray items = responseJson.getJSONArray("items");

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

JSONObject repoJson = items.getJSONObject(i);

Repo repo = new Repo();

repo.fullName = repoJson.getString("full\_name");

repo.url = repoJson.getString("html\_url");

repo.description = repoJson.getString("description");

repos.add(repo);

}

return repos;

}

В обсервере было задано появление ProgressBar на время загрузки и его исчезновение при её окончании или возникновении ошибки (в этом случае появляется также диалог с предложением попробовать повторить отправку запроса).

private Observer<List<Repo>> searchObserver = new

Observer<List<Repo>>() {

@Override

public void onLoading(@NonNull Task<List<Repo>> task) {

ProgressBar progressBar = findViewById(R.id.progress\_bar);

progressBar.setVisibility(ProgressBar.VISIBLE);

}

@Override

public void onSuccess(@NonNull Task<List<Repo>> task, @Nullable List<Repo> data) {

//загрузка в список найденных репозиториев

reposAdapter.setRepos(data);

ProgressBar progressBar = findViewById(R.id.progress\_bar);

progressBar.setVisibility(ProgressBar.INVISIBLE);

reposAdapter.notifyDataSetChanged();

}

@Override

public void onError(@NonNull Task<List<Repo>> task, @NonNull Exception e) {

ProgressBar progressBar = findViewById(R.id.progress\_bar);

progressBar.setVisibility(ProgressBar.INVISIBLE);

showDialog(DIALOG);

}

};

Сам диалог содержит заголовок «Произошла ошибка», сообщение «Повторить запрос?» и две кнопки, на которые установлен Listener.

protected Dialog onCreateDialog(int id) {

if (id == DIALOG)

{

AlertDialog.Builder adb = new AlertDialog.Builder(this);

//заголовок

adb.setTitle("Произошла ошибка");

adb.setMessage("Повторить запрос?");

adb.setIcon(android.R.drawable.ic\_dialog\_alert);

adb.setPositiveButton("Да", repeatSearchListener);

adb.setNegativeButton("Нет", repeatSearchListener);

return adb.create();

}

return super.onCreateDialog(id);

}

В самом Listener’е обрабатывается нажатая кнопка. Если была нажата кнопка «Да» - создается и выполняется такая же задача по поиску репозиториев ещё раз.

DialogInterface.OnClickListener repeatSearchListener = new DialogInterface.OnClickListener() {

@Override

public void onClick(DialogInterface dialog, int answer) {

switch (answer)

{

case Dialog.BUTTON\_POSITIVE:

//создание задачи поиска, выполнение которой //будет происходить в обсервере

task = new SearchTask(searchObserver);

//выполнение через пул потоков

threadExecutor.execute(task);

break;

case Dialog.BUTTON\_NEGATIVE:

break;

}

}

};

В адаптере репозиториев было задано добавление репозиториев в выводимый список. Каждый элемент списка должен открывать активити с соответствующей информацией о выбранном репозитории, поэтому нужно обеспечить передачу данных о репозитории между активностями.

Для этого был использован Intent и функция putExtra. На активити репозитория выводятся полное имя и описание репозитория, поэтому они туда передаются. Также происходит передача и ссылки, которая будет преобразовываться в ссылку для получения коммитов и issues.

case TYPE\_DESCRIPTION:

DescriptionHolder descriptionHolder = (DescriptionHolder) holder;

Repo repo1 = repos.get( position / 2);

descriptionHolder.description.setText(

repo1.description

);

descriptionHolder.description.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View view) {

Intent intent = new Intent(context, RepoActivity.class);

intent.putExtra("name", repos.get(position/2).fullName);

intent.putExtra("url", repos.get(position / 2).url);

intent.putExtra("description", repos.get(position/2).description);

context.startActivity(intent);

}

});

break;

Соответственно, при создании активити репозитория получает эти данные.

Bundle arguments = getIntent().getExtras();

nameOfRepo = (String) arguments.getSerializable("name");

repoUrl = (String) arguments.getSerializable("url");

repoDescription = (String) arguments.getSerializable("description");

setTitle(nameOfRepo);

Также настраивается SwipeRefreshLayout:

swipeRefreshLayout = findViewById(R.id.swipe\_container);

swipeRefreshLayout.setOnRefreshListener(this);

swipeRefreshLayout.setColorSchemeResources(

android.R.color.holo\_blue\_bright,

android.R.color.holo\_green\_light,

android.R.color.holo\_orange\_light,

android.R.color.holo\_red\_light);

Как при создании активити, так и при выполнении действия pull-to-refresh создаются и выполняются одновременно две задачи: для коммитов и для issues.

@Override

public void onRefresh() {

new Handler().postDelayed(new Runnable() {

@Override

public void run() {

// Отменяем анимацию обновления

swipeRefreshLayout.setRefreshing(false);

taskCommits = new SearchCommitsTask(searchObserver);

taskIssues = new SearchIssuesTask(searchIssuesObserver);

//выполнение через пул потоков

threadExecutor.execute(taskCommits);

threadExecutor.execute(taskIssues);

}

}, 4000);

}

Сами эти задачи выполнены так же, как задача для поиска репозиториев, но учтены отличия в виде запросов к API и структуре получаемых данных. Так, на примере коммитов, для получения их списка используется адрес https://api.github.com/repos/{owner}/{repo}/commits.

private String search(String query) throws Exception {

query = query.substring(19, query.length());

query = "https://api.github.com/repos/" + query;

Request request = new Request.Builder()

.url(query + "/commits")

.build();

Response response = getHttpClient().newCall(request).execute();

if (response.code() != 200) {

throw new Exception("api returned unexpected http code: " + response.code());

}

return response.body().string(); }

При обработке JSON’а нужно выделить поля, соответствующие сообщению, автору коммита, а также дате.

JSONArray responseJson = new JSONArray(response);

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

{

JSONObject commit = responseJson.getJSONObject(i);

Commit newCommit = new Commit();

JSONObject commitInformation = commit.getJSONObject("commit");

newCommit.message = commitInformation.getString("message");

JSONObject committer = commitInformation.getJSONObject("committer");

newCommit.date = committer.getString("date");

commits.add(newCommit);

}

# Работа приложения

При вводе поискового запроса и нажатии на кнопку «Подтвердить» выполняется поиск репозиториев на Гитхабе и вывод их названий и описаний (с ограничением на 5 строк) в виде списка, как показано на рисунках 1 и 2.

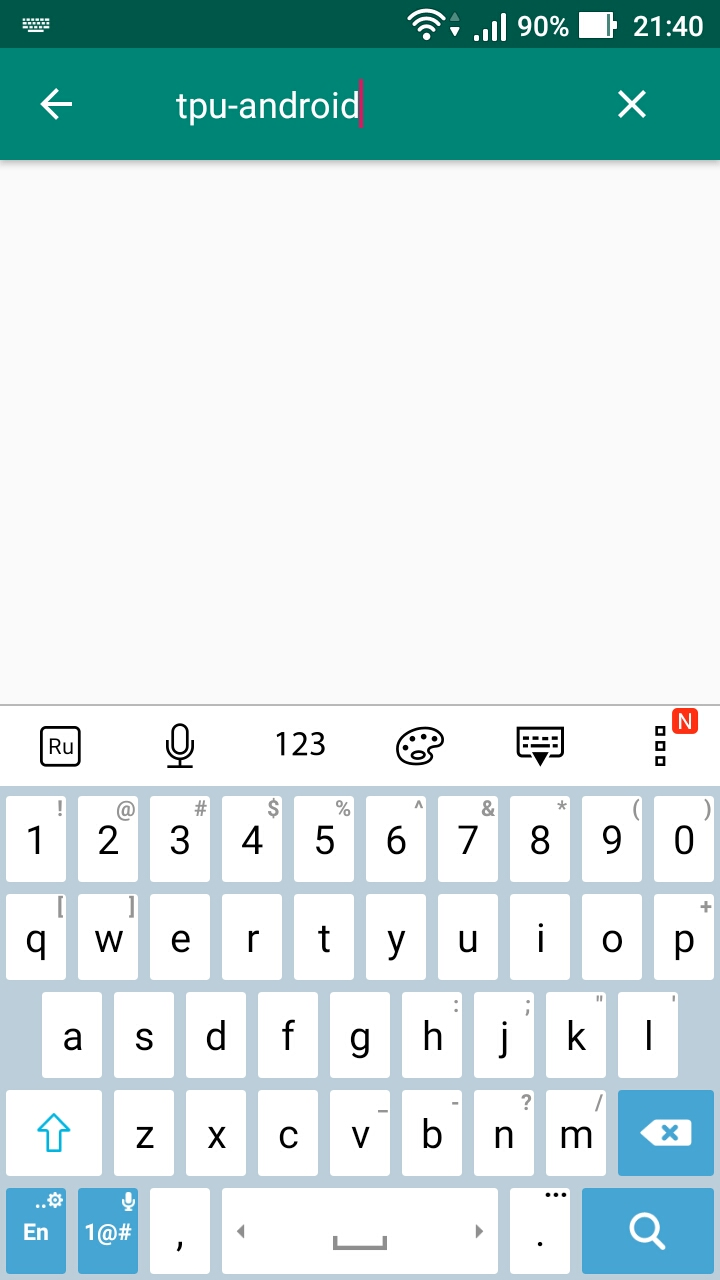


Рисунок 1 – Ввод поискового запроса

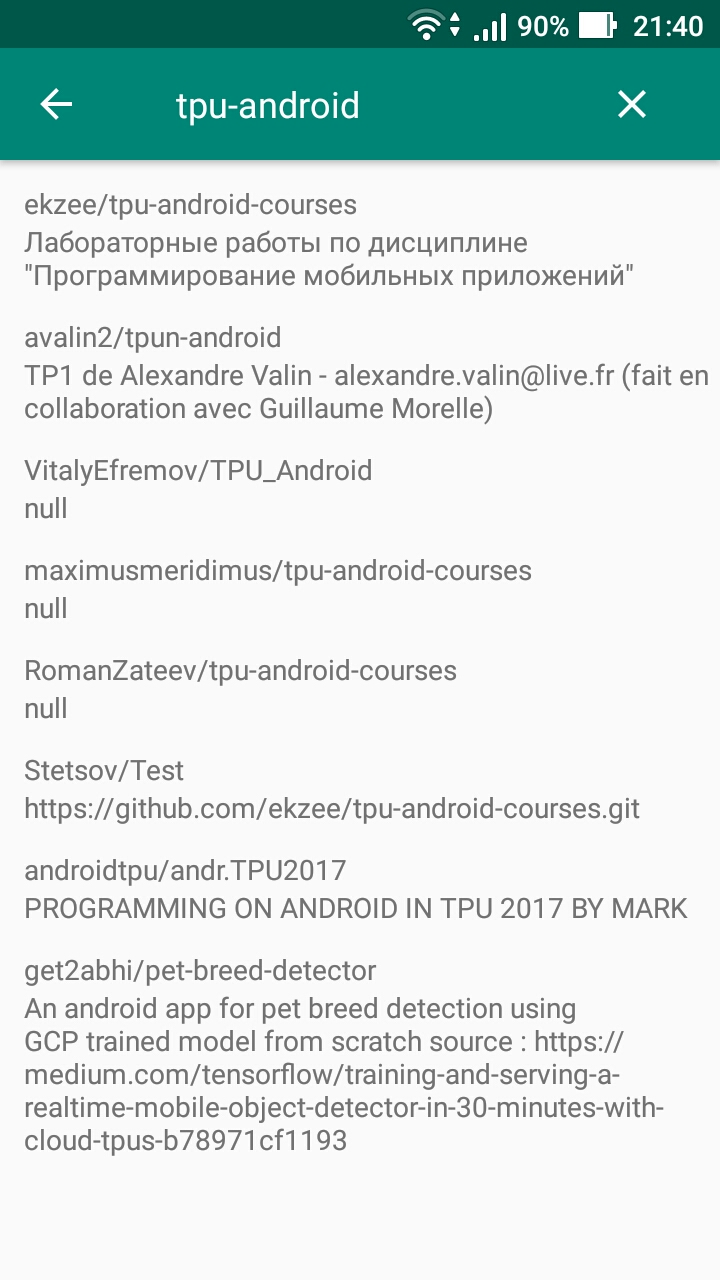


Рисунок 2 – Вывод списка репозиториев по запросу

При этом после отправки запроса, пока происходит загрузка данных, на экране отображается вращающийся ProgressBar, как можно видеть на рисунке 3.



Рисунок 3 – Загрузка данных

Если во время загрузки данных произойдет ошибка (например, набрав поисковой запрос, отключить на смартфоне интернет), то отображается соответствующее сообщение, предлагающее повторить запрос, как показано на рисунке 4.

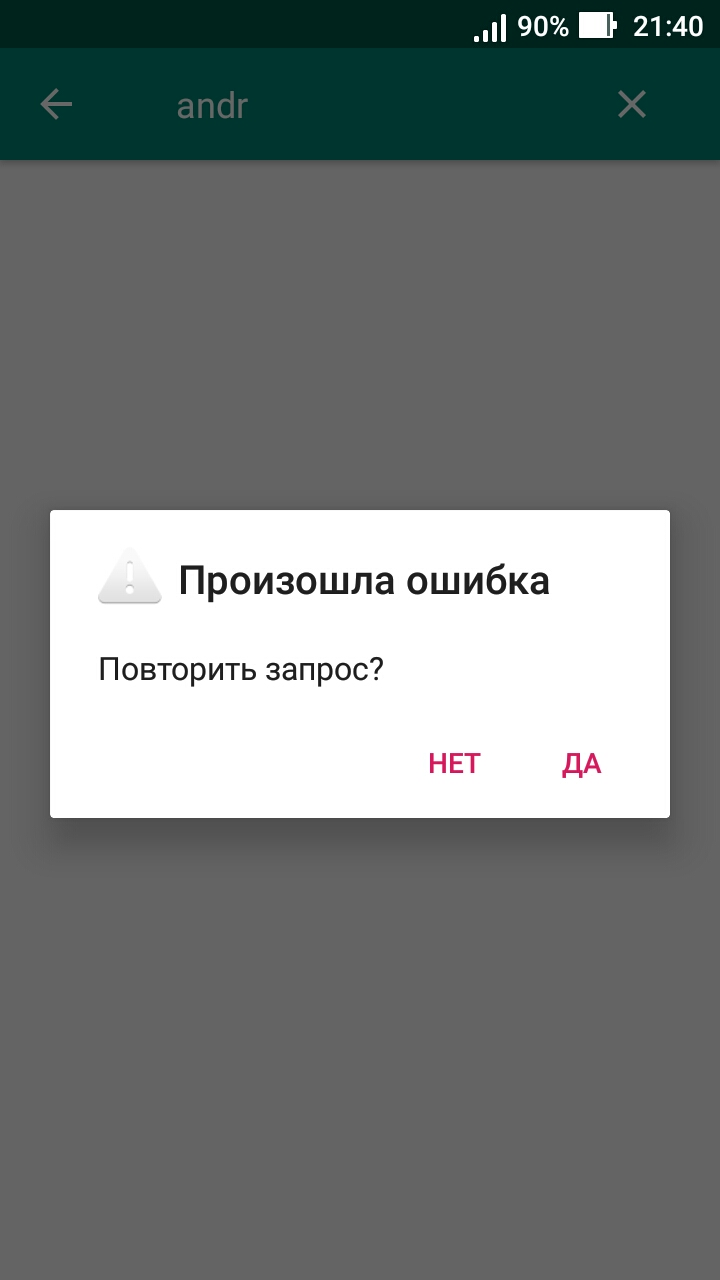


Рисунок 4 – Сообщение об ошибке

Если нажать в списке на какой-нибудь репозиторий, то будет открыт экран с названием этого репозитория, его полным описанием и списком коммитов и issues, как показано на рисунке 5. Причем, если issues не найдены, вместо них выводится соответствующее сообщение.



Рисунок 5 – Просмотр информации о репозитории ekzee/tpu-android-courses

На рисунке 6 показан пример репозитория с имеющимся списком issues.

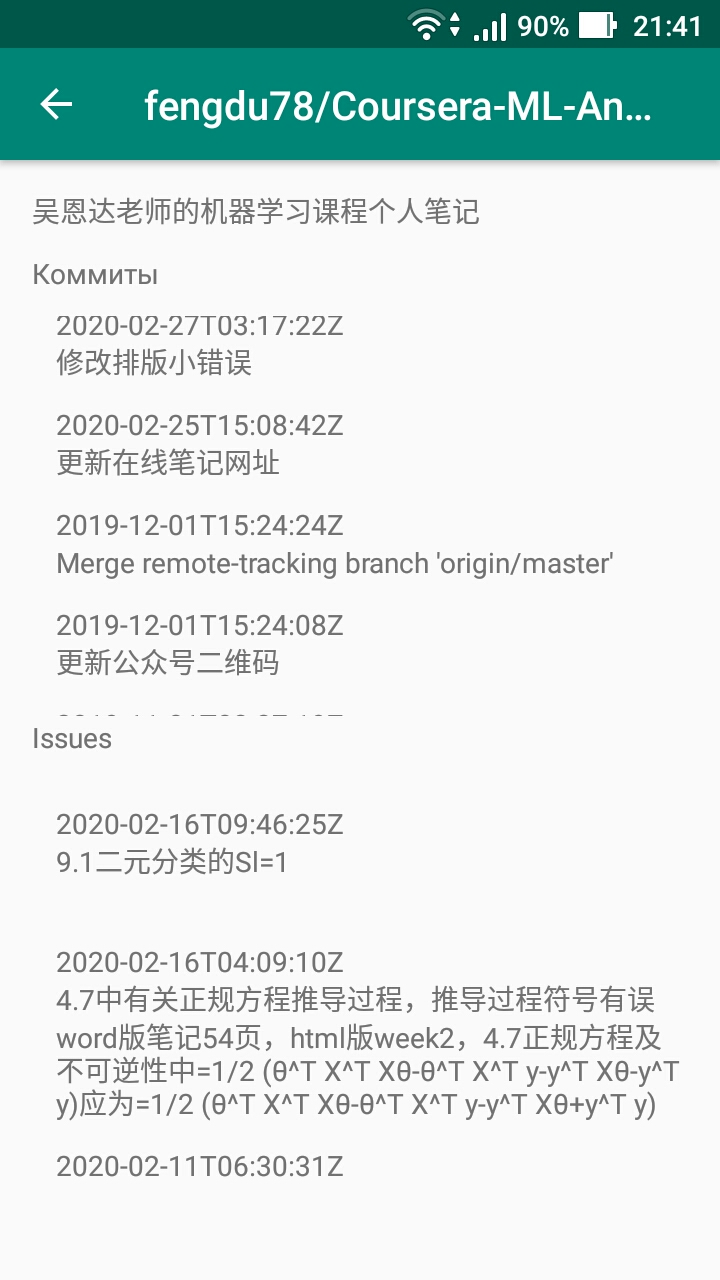


Рисунок 6 – Репозиторий с обоими заполненными списками коммитов и issues

При вертикальном свайпе экрана с информацией о репозитории данные загружаются заново, причем в процессе отображается соответствующий вращающийся символ загрузки, как можно видеть на рисунке 7. Причем, за счет использования NestedScrollView, стало возможным прокручивать списки коммитов и Issues вверх и вниз, не вызывая ненужного срабатывания свайпа.

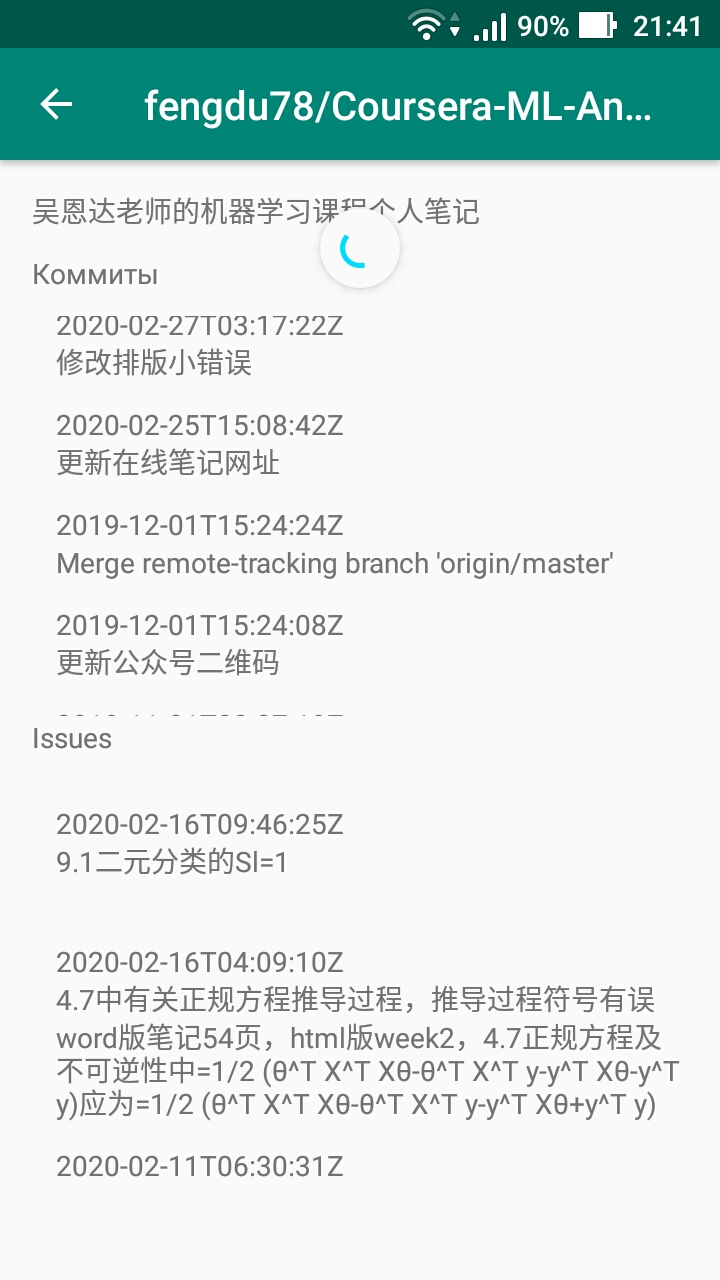


Рисунок 7 – Повторная загрузка данных

# Итоги работы

В результате выполнения лабораторной работы было реализовано приложение, получающее данные о репозиториях посредством API Гитхаба. В процессе работы, с одной стороны, нужно было использовать знания, полученные в предыдущих лабораторных работах (так, здесь для списка репозиториев снова использовался RecyclerView, а для работы с данными были реализованы адаптер и холдеры); с другой стороны, нужно было научиться реализовывать также получение и обработку данных, полученных с сервера в формате JSON, а также использовать задачи, выполняющиеся в отдельных потоках, что может быть крайне важным для приложений, использующихся для решения реальных задач.