УДК 004

# Разработка мобильного приложения для отображения снимков Земли снятой орбитальной камерой

*Вихляев Дмитрий Романович*
*Приамурский государственный университет имени Шолом-Алейхема*
*Студент*

**Аннотация**
Данная статья содержит описание мобильного приложения, способного отображать фотографии Земли, полученные в результате обращения к сервису API NASA. Программа разработана с помощью языка программирования java и Android Studio. Результатом исследования станет готовая программа с подробным описанием её реализации.
**Ключевые слова:** мобильное приложение, API NASA, REST API

# Development of a mobile application for displaying images of the Earth taken by an orbital camera

*Vikhlyaev Dmitry Romanovich*
*Sholom-Aleichem Priamursky State University*
*Student*

**Abstract**
This article contains a description of a mobile application capable of displaying photos of the Earth obtained as a result of accessing the NASA API service. The program is developed using the java programming language and Android Studio. The result of the study will be a ready-made program with a detailed description of its implementation.
**Keywords**: mobile app, NASA API, REST API

## 1 Введение

### 1.1 Актуальность

DSCOVR (Deep Space Climate Observatory) была разработана NASA совместно с ВВС США для наблюдений за состоянием атмосферы Земли. Космический аппарат был запущен 11 февраля 2015 года. Спутник оснащен камерой, которая делает серию из 10 изображений Земли с разными фильтрами, что позволяет достичь высокого качества снимков с разрешением от 6,2 до 9,4 миль на пиксель. Помимо фотографий аппарат позволяет следить за опасностями космической погоды, солнечным ветром и выбросами коронарной массы и состоянием атмосферы Земли. Кроме того, каждый снимок сопровождаться детальной информацией, в том числе о позиции DSCOVR в момент съемки по отношению к Солнцу и Земле. Сервер

NASA позволяет обращаться к REST API для взаимодействия сторонних приложений с базой данных полученной DSCOVR.

### 1.2Обзор исследований

Н.Н.Гурьев, А.А.Кочетков, А.А.Удалов показали особые построения синхронных и асинхронных запросов при работе с rest api [1]. П.А.Безрук разработал системы распределенного мониторинга компьютерной сети на основе rest api [2]. Д.В.Веркошанский, А.А.Мешков, А.Б.Агаджанян продемонстрировали использование архитектурного стиля взаимодействия rest api в мобильном приложении [3]. Д.Ю.Ивакин автоматизировал процесс предоставления расписания занятий за счёт использования rest api клиент-сервера [4]. А.А.Чунихин провёл тестирование при разработке api-клиента и rest-сервиса [5].

### 1.3Цель исследования

Цель исследования – разработать программу способную подключатся к архиву фотографий Земли, сделанной орбитальной камерой, и выводить изображения на экран мобильного устройства.

## 2 Материалы и методы

Для реализации программы используются язык программирования java, среда разработки Android Studio, удалённый сервер NASA API [7].

## 3 Результаты и обсуждения

Приложения, использующие REST API, содержат в себе определённую конструкцию. Программа клиент обращается к REST серверу по протоколу HTTP и получает данные. Приложение может отображать данные в интерфейсе производить различные действия с ними и сохранять их на сервере (рис. 1).

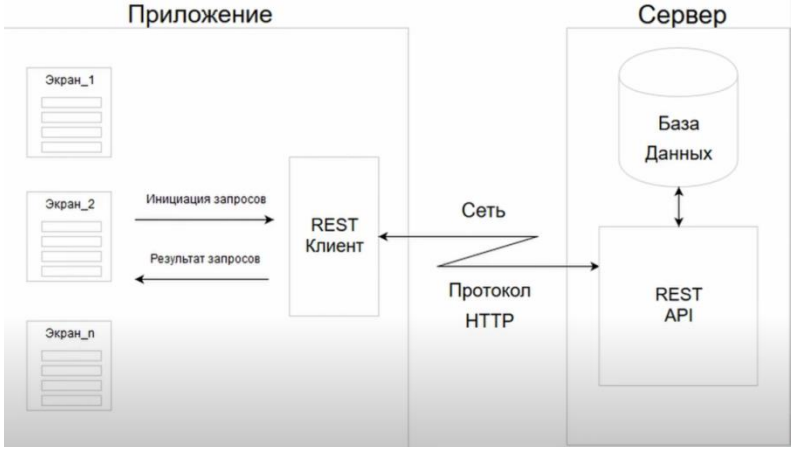


Рис. 1. Структура REST API

Программа содержит подключённые библиотеки совместимости со старыми устройствами android.

Модули «retrofit» и «okhttp» предназначены для удобного выполнения REST запросов.

Библиотека «gson» разработана компанией google, позволяет парсить «json» в объекты языка java и наоборот.

«rxjava» – библиотека для реактивного программирования.

Модули «image-view» и «image-loader» позволяют масштабировать и загружать изображения соответственно(рис.2).

```
implementation 'com.android.support:appcompat-v7:28.0.0'
implementation 'com.android.support:design:28.0.0'
implementation 'com.android.support:cardview-v7:28.0.0'
implementation 'com.android.support:recyclerview-v7:28.0.0'

implementation 'com.squareup.retrofit2:retrofit:2.4.0'
implementation 'com.squareup.okhttp3:okhttp:4.0.0'
implementation 'com.squareup.retrofit2:adapter-rxjava2:2.4.0'
implementation 'com.squareup.okhttp3:logging-interceptor:4.0.0'

implementation 'com.squareup.retrofit2:converter-gson:2.4.0'
implementation 'com.google.code.gson:gson:2.8.6'

implementation 'io.reactivex.rxjava2:rxjava:2.2.8'
implementation 'io.reactivex.rxjava2:rxandroid:2.0.1'

implementation 'com.davemorrissey.labs:subsampling-scale-image-view:3.10.0'

implementation 'com.nostra13.universalimageloader:universal-image-loader:1.9.5'
```

Рис. 2. Подключаемые модули

Главный класс приложения содержит инициализацию сетевого сервиса, для обращения к серверу и загрузчик изображений. В настройках загрузчика указывается включения кэша и задаётся размеры необходимой для него памяти (рис.3).

```java
package com.arhiser.nasa_sample;

import android.app.Application;

import com.arhiser.nasa_sample.api.NasaService;
import com.nostra13.universalimageloader.cache.memory.impl.LruMemoryCache;
import com.nostra13.universalimageloader.core.DisplayImageOptions;
import com.nostra13.universalimageloader.core.ImageLoader;
import com.nostra13.universalimageloader.core.ImageLoaderConfiguration;

public class App extends Application {

    private NasaService nasaService;

    @Override
    public void onCreate() {
        super.onCreate();

        nasaService = new NasaService();

        DisplayImageOptions defaultOptions = new DisplayImageOptions.Builder()
                .cacheInMemory(true)
                .build();
        ImageLoaderConfiguration config = new ImageLoaderConfiguration.Builder( context: this)
                .defaultDisplayImageOptions(defaultOptions)

                .memoryCache(new LruMemoryCache( maxSize: 20 * 1024 * 1024))
                .memoryCacheSize(20 * 1024 * 1024)
                .build();

        ImageLoader.getInstance().init(config);
    }

    public NasaService getNasaService() { return nasaService; }
}
```

Рис. 3. Инициализация компонентов

Интерфейс библиотеки «retrofit» позволяет описывать запросы по сети как функции. Сначала аннотации указывают, к какому типу относиться запрос, затем описан путь и тип возвращаемого значения (рис. 4).

```java
package com.arhiser.nasa_sample.api;

import com.arhiser.nasa_sample.api.model.DateDTO;
import com.arhiser.nasa_sample.api.model.PhotoDTO;

import java.util.List;

import io.reactivex.Single;
import retrofit2.http.GET;
import retrofit2.http.Path;

public interface NasaApi {

    @GET("natural/all")
    Single<List<DateDTO>> getDatesWithPhoto();

    @GET("natural/date/{date}")
    Single<List<PhotoDTO>> getPhotosForDate(@Path("date") String date);
}
```

Рис. 4. Интерфейс для создания REST запросов

Инициализация сетевого интерфейса

Для обращения к серверу необходим специальный ключ. По ключу происходит идентификация приложения, после чего сервер даёт доступ к своей базе данных. Данный сервис позволяет делать до 1000 запросов в час.

В качестве клиента для передачи данных используется модуль «okhttp». Инициализация осуществляется с помощью метода «Builder». После чего

добавляются объекты, встраиваемые в цепочку обработки запросов, для возможности добавления, удаления и модификации. В данном случае используются для добавления ко всем запросам серверного ключа.

После инициализации REST клиента можно использовать его при создании «retrofit». Инициализация происходит с помощью метода «Builder», указываются базовый адрес сервера, метод конвертации «json», REST клиент и добавляется адаптер, использующий реактивное расширение при работе с запросами.

У Экземпляра класса «retrofit» вызывается метод «create», в который передаётся интерфейс с методами обращения к серверу(рис. 5,6,7).

```
public class NasaService {
    public static String KEY = serverKey;

    NasaApi api;

    public NasaService() {
        Retrofit retrofit = createRetrofit();
        api = retrofit.create(NasaApi.class);
    }

    public NasaApi getApi() { return api; }
```

Рис. 5. Указание серверного ключа и обращение к экземпляру «retrofit»

```
private OkHttpClient createOkHttpClient() {
    final OkHttpClient.Builder httpClient = new OkHttpClient.Builder();
    httpClient.addInterceptor(new Interceptor() {
        @NotNull
        @Override
        public Response intercept(@NotNull Chain chain) throws IOException {
            final Request original = chain.request();
            final HttpUrl originalHttpUrl = original.url();
            final HttpUrl url = originalHttpUrl.newBuilder()
                    .addQueryParameter( name: "api_key", KEY)
                    .build();
            final Request.Builder requestBuilder = original.newBuilder()
                    .url(url);
            final Request request = requestBuilder.build();
            return chain.proceed(request);
        }
    });

    HttpLoggingInterceptor logging = new HttpLoggingInterceptor();
    logging.level(HttpLoggingInterceptor.Level.BODY);
    httpClient.addInterceptor(logging);
    return httpClient.build();
}
```

Рис. 6. Инициализация REST клиента

```
private Retrofit createRetrofit() {
    return new Retrofit.Builder()
            .baseUrl("https://api.nasa.gov/EPIC/api/")
            .addConverterFactory(GsonConverterFactory.create())
            .client(createOkHttpClient())
            .addCallAdapterFactory(RxJava2CallAdapterFactory.create())
            .build();
```

Рис. 7. Преобразование серверных методов в java интерфейс

Запрос к серверу осуществляется с помощью раннее созданного класса «app», который содержит интерфейс для работы с REST сервисом.

Композит «disposable» является контейнером объектов.

Метод «subscribeOn» определяет поток для выполнения запроса по сети. Принимаемый параметр означает, что поток ориентирован на операции ввода вывода.

С помощью функции «observeOn» можно определить, в каком потоке будут обрабатываться данные полученные от сервера, в данном случае используется главный поток.

Функция «subscribe» возвращает интерфейс типа «disposable», он необходим, чтобы прекратить работу запроса, при смене текущего «activity».

В случае успешного запроса, программа получает список объектов типа «DateDTO», который отвечает за хранение дат в строковом типе. Если запрос закончится неудачно, то вернётся исключение (рис.7,8,9,10).

```
public class MainActivity extends AppCompatActivity {

    CompositeDisposable disposable = new CompositeDisposable();

    RecyclerView recyclerView;
    Adapter adapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        recyclerView = findViewById(R.id.list);

        adapter = new Adapter();

        GridLayoutManager layoutManager = new GridLayoutManager( context: this,  spanCount: 2);
        recyclerView.setLayoutManager(layoutManager);
        recyclerView.setAdapter(adapter);

        getSupportActionBar().setTitle("Выберите день");

        App app = (App) getApplication();

        disposable.add(app.getNasaService().getApi().getDatesWithPhoto()
                .subscribeOn(Schedulers.io())
                .observeOn(AndroidSchedulers.mainThread())
                .subscribe(new BiConsumer<List<DateDTO>, Throwable>() {
                    @Override
                    public void accept(List<DateDTO> dates, Throwable throwable) throws Exception {
                        if (throwable != null) {
                            Toast.makeText( context: MainActivity.this,  text: "Data loading error", Toast.LENGTH_SHORT).show();
                        } else {
                            adapter.setDates(dates);
```

Рис. 7. Программный код «MainActivity»

```
public class DateDTO {
    private String date;

    public String getDate() { return date; }
}
```

Рис. 8. Структура класса «DateDTO»

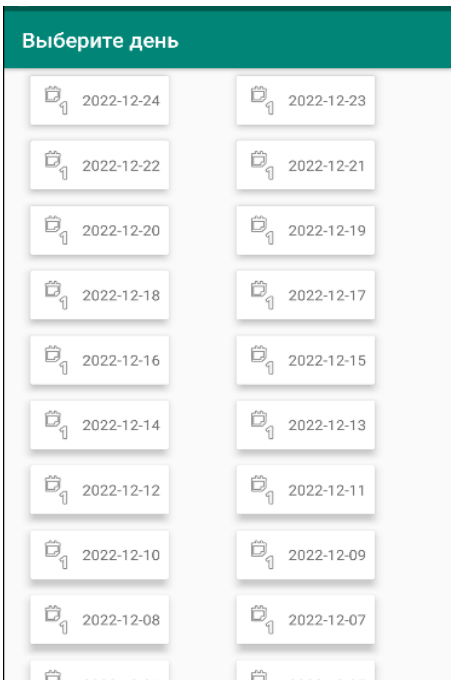


Рис. 9. Внешний вид «MainActivity»

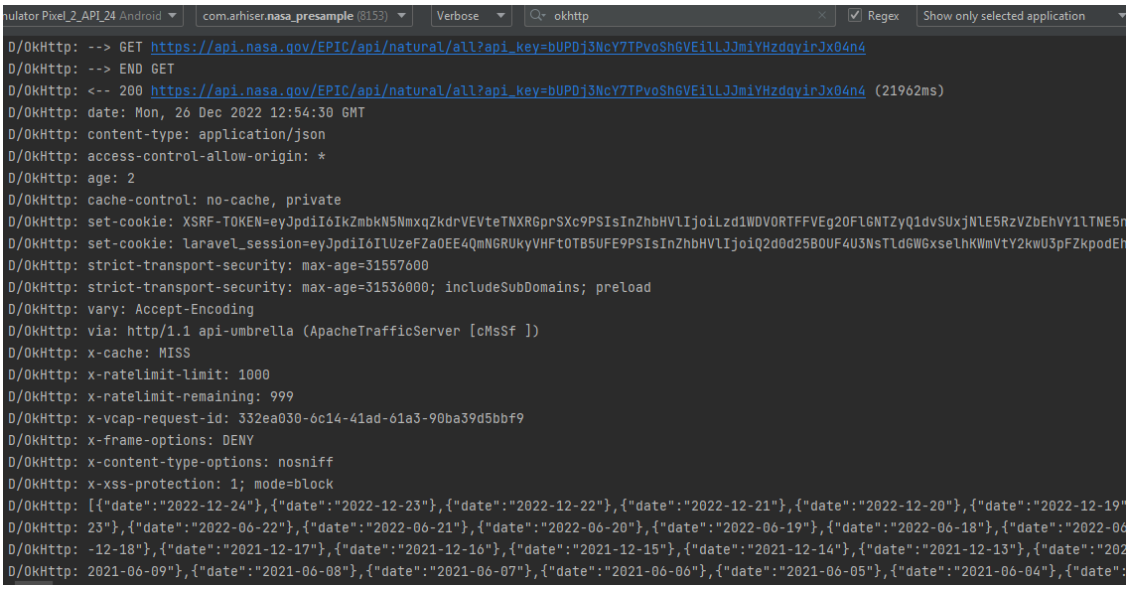


Рис. 10. Передача данных в «MainActivity»

Аналогичным образом происходит запрос по фотографиям за определённую дату. В качестве обработчика выступает класс «PhotoDTO», который получает адрес изображения из «json» ответа сервера (рис.11,12,13).

```java
public class PhotoListActivity extends AppCompatActivity {

    private static final String EXTRA_DATE = "PhotoListActivity.EXTRA_DATE";

    CompositeDisposable disposable = new CompositeDisposable();

    RecyclerView recyclerView;
    Adapter adapter;

    public static void start(Context caller, String date) {
        Intent intent = new Intent(caller, PhotoListActivity.class);
        intent.putExtra(EXTRA_DATE, date);
        caller.startActivity(intent);
    }
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        recyclerView = findViewById(R.id.list);

        adapter = new Adapter();

        getSupportActionBar().setTitle("Выберите время");

        recyclerView.setLayoutManager(new LinearLayoutManager( context: this, LinearLayoutManager.VERTICAL,  reverseLayout: false));
        recyclerView.setAdapter(adapter);

        App app = (App) getApplication();

        disposable.add(app.getNasaService().getApi().getPhotosForDate(getIntent().getStringExtra(EXTRA_DATE))
                .observeOn(AndroidSchedulers.mainThread())
                .subscribeOn(Schedulers.io())
                .subscribe(new BiConsumer<List<PhotoDTO>, Throwable>() {
                    @Override
                    public void accept(List<PhotoDTO> photos, Throwable throwable) throws Exception {
                        if (throwable != null) {
                            Toast.makeText( context: PhotoListActivity.this,  text: "Data loading error", Toast.LENGTH_SHORT).show();
                        } else {
                            adapter.setPhotos(photos);
```

Рис. 11. Программный код «PhotoListActivity»

```java
public class PhotoDTO {
    private String identifier;
    private String caption;
    private String image;
    private String date;

    public String getIdentifier() { return identifier; }

    public void setIdentifier(String identifier) { this.identifier = identifier; }

    public String getCaption() { return caption; }

    public void setCaption(String caption) { this.caption = caption; }

    public String getImage() { return image; }

    public void setImage(String image) { this.image = image; }

    public String getDate() { return date; }

    public void setDate(String date) { this.date = date; }

    public String getImageUrl() {
        //https://api.nasa.gov/EPIC/archive/enhanced/2016/12/04/png/epic_RBG_20161204003633.png?api_key=DEMO_KEY
        StringBuilder sb = new StringBuilder();
        sb.append("https://api.nasa.gov/EPIC/archive/natural/");
        String[] dateComponents = date.split( regex: " ")[0].split( regex: "-");
        sb
                .append(dateComponents[0]).append('/')
                .append(dateComponents[1]).append('/')
                .append(dateComponents[2]).append("/png/")
                .append(image).append(".png?api_key=").append(NasaService.KEY);
        return sb.toString();
    }
```

Рис. 12. Структура класса «PhotoDTO»

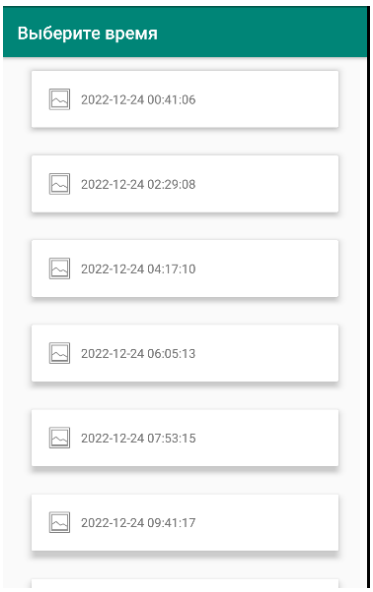


Рис. 13. Внешний вид «PhotoListActivity»

При обращении пользователя к фотографии запрашивается полноэкранный режим. С помощью «ImageLoader» загружается фотография с указанного адреса. Полученное изображение устанавливается внутри подключенной библиотеки «SubsamplingScaleImageView». Данный модуль самостоятельно займётся корректным отображением картинки на экране (рис.14,15).

```
public class PhotoActivity extends AppCompatActivity {

    private static final int REQUEST_PERMISSION_WRITE_STORAGE = 1111;

    private static final String EXTRA_URL = "PhotoActivity.EXTRA_URL";

    private SubsamplingScaleImageView imageView;
    private Toolbar toolbar;

    private boolean isToolbarVisible;

    private Bitmap photo;

    public static void start(Context caller, String url) {
        Intent intent = new Intent(caller, PhotoActivity.class);
        intent.putExtra(EXTRA_URL, url);
        caller.startActivity(intent);
    }

    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        getWindow().addFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN);

        setContentView(R.layout.activity_photo);

        imageView = findViewById(R.id.image);
        toolbar = findViewById(R.id.toolbar);
        toolbar.setTitle("");

        setSupportActionBar(toolbar);

        ImageLoader.getInstance().loadImage(getIntent().getStringExtra(EXTRA_URL), new SimpleImageLoadingListener() {
            @Override
            public void onLoadingComplete(String imageUri, View view, Bitmap loadedImage) {
                if (!isFinishing()) {
                    photo = loadedImage;
                    imageView.setImage(ImageSource.cachedBitmap(loadedImage));
                    findViewById(R.id.progress).setVisibility(View.GONE);
```

Рис. 14. Загрузка и отображение фотографии

Рис. 15. Загруженная фотография

В результате исследования было разработано мобильное приложение, способное взаимодействовать с REST API сервисом NASA. Описано обращение к удалённому серверу, получение и загрузка данных. Продемонстрированы исходный код и результаты работы приложения.

**Библиографический список**

1. Гурьев Н.Н., Кочетков А.А., Удалов А.А. Особенности построения синхронных и асинхронных запросов при работе с rest api // Молодежный научно-технический вестник. 2017. № 2. С. 14.
2. Безрук П.А. Разработка системы распределенного мониторинга компьютерной сети на основе rest api // Актуальные проблемы авиации и космонавтики. 2017. Т. 2. № 13. С. 94-95.
3. Веркошанский Д.В., Мешков А.А., Агаджанян А.Б. Использование архитектурного стиля взаимодействия rest api в мобильном приложении // Интернаука. 2022. № 4-1 (227). С. 15-16.
4. Ивакин Д.Ю. Автоматизация процесса предоставления расписания занятий за счёт использования rest api клиент-сервера // В сборнике: Студенческая молодёжь XXI века: наука, творчество, карьера, цифровизация. Сборник материалов Межвузовской научно-практической конференции. Под общей редакцией Е.А. Руднева, под научной редакцией

Л.Н. Горбуновой. Москва, 2022. С. 235-244.
5. Чунихин А.А. Разработка api-клиента и тестирование rest-сервиса // В книге: ЧЕЛОВЕК И ПРИРОДА. сборник материалов студенческой научно-практической конференции. Омск, 2021. С. 179-181.
6. Зотова Ю.А., Котилевец И.Д. Разработка архитектуры rest api для взаимодействия с сервисами приложения // В сборнике: Информационные технологии и математическое моделирование систем 2018. труды международной научно-технической конференции. 2018. С. 61-65.
7. https://api.nasa.gov