Pull to Refresh

This lesson will introduce how we can add a pull-to-refresh functionality to the blog list screen to indicate that data loading is in progress and to give a user the ability to manually refresh the data.

WE'LL COVER THE FOLLOWING

Swipe-refresh-layout

The pull-to-refresh layout *(also known as swipe-refresh-layout)* is available as a separate library, so in order to use it we need to update our dependencies in *build.gradle* file.

```
dependencies {
    // ui
    implementation 'androidx.appcompat:appcompat:1.1.0'
    implementation 'androidx.constraintlayout:constraintlayout:1.1.3'
    implementation "androidx.swiperefreshlayout:swiperefreshlayout:1.0.0"
    implementation 'com.google.android.material:material:1.1.0-alpha10'
    implementation 'com.github.bumptech.glide:glide:4.10.0'
    implementation 'com.squareup.okhttp3:okhttp:4.2.1'
    implementation 'com.google.code.gson:gson:2.8.6'
}
```

build.gradle

Next, open the *main_activity.xml* layout file and wrap the `RecyclerView` with `SwipeRefreshLayout`.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/
        xmlns:app="http://schemas.android.com/apk/res-auto"
        xmlns:tools="http://schemas.android.com/tools"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">

    ...
```

```
    <androidx.swiperefreshlayout.widget.SwipeRefreshLayout
            android:id="@+id/refresh"
            android:layout_width="match_parent"

            android:layout_height="0dp"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintLeft_toLeftOf="parent"
            app:layout_constraintLeft_toRightOf="parent"
            app:layout_constraintTop_toBottomOf="@+id/toolbar">

        <androidx.recyclerview.widget.RecyclerView
                android:id="@+id/recyclerView"
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                tools:listitem="@layout/item_main" />

    </androidx.swiperefreshlayout.widget.SwipeRefreshLayout>

</androidx.constraintlayout.widget.ConstraintLayout>
```

main_activity.xml

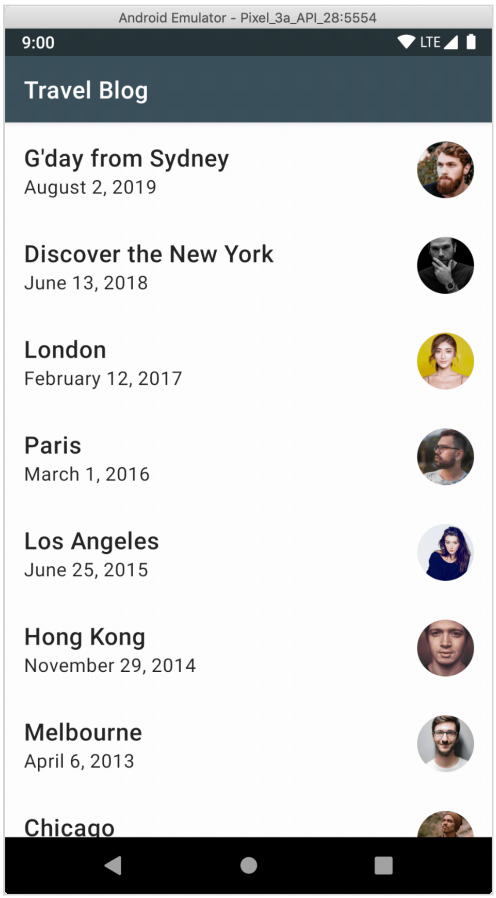
Finally, let's bind `SwipeRefreshLayout` in the `MainActivity` and set the pull-to-refresh listener via the `setOnRefreshListener` method (1). When this listener is triggered, we can simply execute the `loadData` method to reload the data.

To make the loader indicator stay on the screen while data is loading, we can use the `setRefreshing(true)` method (2) and when the data loading has been completed we can use the `setRefreshing(false)` method (3) (4).

```
public class MainActivity extends AppCompatActivity {

    private MainAdapter adapter;
    private SwipeRefreshLayout refreshLayout;

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

        adapter = new MainAdapter();

        RecyclerView recyclerView = findViewById(R.id.recyclerView);
        recyclerView.setLayoutManager(new LinearLayoutManager(this));
        recyclerView.setAdapter(adapter);

        refreshLayout = findViewById(R.id.refresh);
        refreshLayout.setOnRefreshListener(this::loadData); // 1

        loadData();
    }

    private void loadData() {
        refreshLayout.setRefreshing(true); // 2
        BlogHttpClient.INSTANCE.loadBlogArticles(new BlogArticlesCallback() {
            @Override
```

```
            public void onSuccess(List<Blog> blogList) {
                runOnUiThread(() -> {
                    refreshLayout.setRefreshing(false); // 3

                    adapter.submitList(blogList);
                });
            }

            @Override
            public void onError() {
                runOnUiThread(() -> {
                    refreshLayout.setRefreshing(false); // 4
                    showErrorSnackbar();
                });
            }
        });
    }
}
```

That's all we need to implement a pull-to-refresh functionality.



Hit the *run* button to try it yourself.

```
package com.travelblog.adapter;

import android.view.*;
import android.widget.*;

import androidx.annotation.*;
import androidx.recyclerview.widget.ListAdapter;
import androidx.recyclerview.widget.*;
```

```
import com.bumptech.glide.*;
import com.bumptech.glide.load.resource.bitmap.*;
import com.bumptech.glide.load.resource.drawable.*;

import com.travelblog.R;
import com.travelblog.http.*;

public class MainAdapter extends ListAdapter<Blog, MainAdapter.MainViewHolder> {

    public MainAdapter() {
        super(DIFF_CALLBACK);
    }

    @NonNull
    @Override
    public MainViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
        LayoutInflater inflater = LayoutInflater.from(parent.getContext());
        View view = inflater.inflate(R.layout.item_main, parent, false);
        return new MainViewHolder(view);
    }

    @Override
    public void onBindViewHolder(MainViewHolder holder, int position) {
        holder.bindTo(getItem(position));
    }

    static class MainViewHolder extends RecyclerView.ViewHolder {

        private TextView textTitle;
        private TextView textDate;
        private ImageView imageAvatar;

        MainViewHolder(@NonNull View itemView) {
            super(itemView);
            textTitle = itemView.findViewById(R.id.textTitle);
            textDate = itemView.findViewById(R.id.textDate);
            imageAvatar = itemView.findViewById(R.id.imageAvatar);
        }

        void bindTo(Blog blog) {
            textTitle.setText(blog.getTitle());
            textDate.setText(blog.getDate());

            Glide.with(itemView)
                    .load(blog.getAuthor().getAvatarURL())
                    .transform(new CircleCrop())
                    .transition(DrawableTransitionOptions.withCrossFade())
                    .into(imageAvatar);
        }

    }

    private static final DiffUtil.ItemCallback<Blog> DIFF_CALLBACK =
            new DiffUtil.ItemCallback<Blog>() {
                @Override
                public boolean areItemsTheSame(@NonNull Blog oldData,
                                               @NonNull Blog newData) {
                    return oldData.getId().equals(newData.getId());
                }

                @Override
                public boolean areContentsTheSame(@NonNull Blog oldData,
                                                  @NonNull Blog newData) {
```

```
                return oldData.equals(newData);
            }


        };
}
```

The next lesson will explain how to add a click listener for the list item.