Offline Support

This lesson will provide instructions on how the application should function when not connected to a network.

Now that we have all the necessary pieces to implement offline support, we can finally use them in the `MainActivity`.

Start by creating a `BlogRepository` object in the `onCreate` method *(1)*.

In the `OnRefreshListener`, we are going to call `loadDataFromNetwork` method *(2)*.

At the very end of the `onCreate` method, when everything is initialized, we can load data from database *(3)* and then launch loading data from the internet *(4)*. Because loading data from the database is very fast and loading data from the internet is slow, we will see cached data until data from the internet is loaded.

```
public class MainActivity extends AppCompatActivity {

    ...

    private MainAdapter adapter;
    private SwipeRefreshLayout refreshLayout;
    private BlogRepository repository;

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

        repository = new BlogRepository(getApplicationContext()); // 1

        ...

        refreshLayout = findViewById(R.id.refresh);
        refreshLayout.setOnRefreshListener(this::loadDataFromNetwork); // 2

        loadDataFromDatabase(); // 3
        loadDataFromNetwork(); // 4
    }
...
}
```

Let's implement the `loadDataFromDatabase` first. We can use our repository method to load data from the database, switch to UI thread and display the data.

```
private void loadDataFromDatabase() {
    repository.loadDataFromDatabase(blogList -> runOnUiThread(() -> {
        adapter.setData(blogList);
        sortData();
    }));
}
```

The implementation of the `loadDataFromNetwork` is a bit more complicated, but it's very similar to what we have before. Basically, we show the pull-to-refresh loading indicator *(1)*, use a repository pattern to load data from the network *(2)*, and switch to UI thread and display the data (3).

```
private void loadDataFromNetwork() {
    refreshLayout.setRefreshing(true); // 1

    repository.loadDataFromNetwork(new DataFromNetworkCallback() { // 2
        @Override
        public void onSuccess(List<Blog> blogList) {
            runOnUiThread(() -> { // 3
                adapter.setData(blogList);
                sortData();
                refreshLayout.setRefreshing(false);
            });
        }

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

That's all we need to implement offline support for our application. If we launch the application without an internet connection, it is going to show the data stored in the database.

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.*;

import java.util.*;

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

    public interface OnItemClickListener {
        void onItemClicked(Blog blog);
    }

    private OnItemClickListener clickListener;

    private List<Blog> originalList = new ArrayList<>();

    public MainAdapter(OnItemClickListener clickListener) {
        super(DIFF_CALLBACK);
        this.clickListener = clickListener;
    }

    @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, clickListener);
    }

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

    public void setData(@Nullable List<Blog> list) {
        originalList = list;
        super.submitList(list);
    }

    public void filter(String query) {
        List<Blog> filteredList = new ArrayList<>();
        for (Blog blog : originalList) {
            if (blog.getTitle().toLowerCase().contains(query.toLowerCase())) {
                filteredList.add(blog);
            }
        }
        submitList(filteredList);
    }

    public void sortByTitle() {
        List<Blog> currentList = new ArrayList<>(originalList);
        Collections.sort(currentList, (o1, o2) -> o1.getTitle().compareTo(o2.getTitle()));
```

```
        submitList(currentList);
    }

    public void sortByDate() {
        List<Blog> currentList = new ArrayList<>(originalList);
        Collections.sort(currentList, (o1, o2) -> o2.getDateMillis().compareTo(o1.getDateMill
        submitList(currentList);
    }

    static class MainViewHolder extends RecyclerView.ViewHolder {

        private TextView textTitle;
        private TextView textDate;
        private ImageView imageAvatar;
        private Blog blog;

        MainViewHolder(@NonNull View itemView, OnItemClickListener listener) {
            super(itemView);
            itemView.setOnClickListener(v -> listener.onItemClicked(blog));
            textTitle = itemView.findViewById(R.id.textTitle);
            textDate = itemView.findViewById(R.id.textDate);
            imageAvatar = itemView.findViewById(R.id.imageAvatar);
        }

        void bindTo(Blog blog) {
            this.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() == newData.getId();
                }

                @Override
                public boolean areContentsTheSame(@NonNull Blog oldData,
                                                  @NonNull Blog newData) {
                    return oldData.equals(newData);
                }
            };
}
```

This is the last lesson of the “Practical Guide to Android Development” course. There is still a lot to learn and the next lesson will help you to understand where to go next.