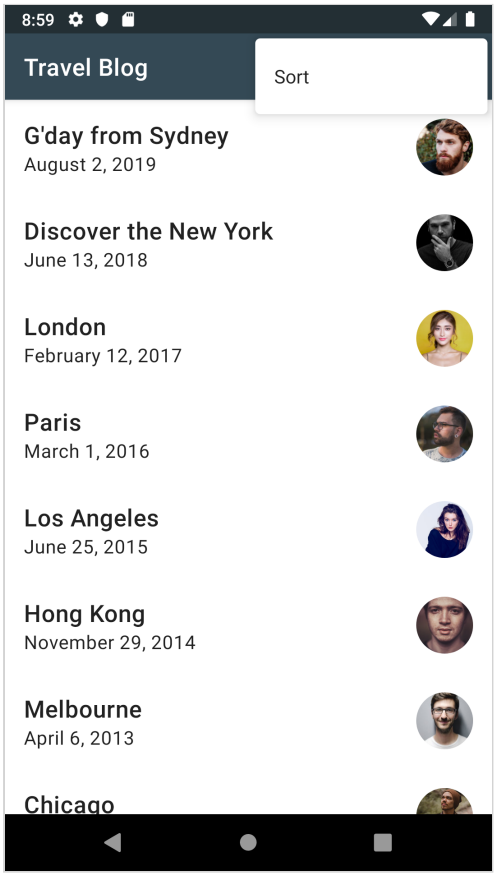
Sort

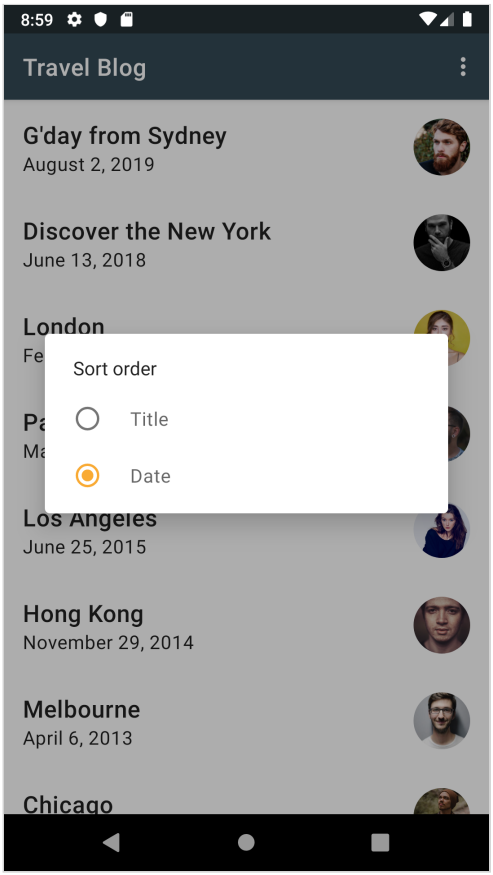
In this lesson, we will look at sort functionality for the blog list screen. The sort functionality will be available throughout the dialog which can be accessed via the toolbar menu.

WE'LL COVER THE FOLLOWING

- Final result preview
- Toolbar menu
- Sort dialog
- Sort

Final result preview





Toolbar menu

Similarly to the layout files, toolbar menu can be defined by XML tags and attributes. Let's create a `main_menu.xml` file in the `res/menu` folder.

Define a root `menu` tag along with a child `item` tag. The `item` tag describes menu item properties, via XML attributes:

- `id` attribute specifies the unique id of the menu item
- `title` attribute specificities title of the menu item
- `showAsAction` attribute specifies whether we want the menu to be displayed under the *more* menu or not

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:app="http://schemas.android.com/apk/res-auto">

    <item
            android:id="@+id/sort"
            android:title="Sort"
            app:showAsAction="never" />
</menu>
```

main_menu.xml

Next, let's open the `activity_main.xml` layout file and set the `menu` attribute to the `MaterialToolbar`. We also need to set a `style` attribute to `Widget.MaterialComponents.Toolbar.Primary`, because by default, menu icons will be rendered with black while we need white.

```
...
<com.google.android.material.appbar.MaterialToolbar
        android:id="@+id/toolbar"
        style="@style/Widget.MaterialComponents.Toolbar.Primary"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        app:menu="@menu/main_menu"
        app:title="Travel Blog"
        app:titleTextColor="@android:color/white" />
...
```

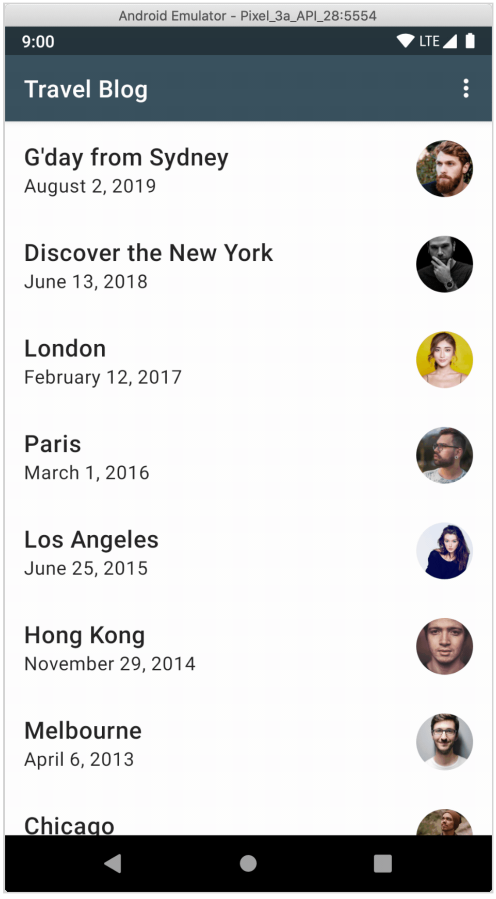
activity_main.xml

Similar to any other view, we can set a click listener to the menu item via `setOnMenuItemClickListener` and, because we may have more than one menu item, the id check is necessary.

```
public class MainActivity extends AppCompatActivity {
```

```
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        MaterialToolbar toolbar = findViewById(R.id.toolbar);
        toolbar.setOnMenuItemClickListener(item -> {
            if (item.getItemId() == R.id.sort) {
                onSortClicked(); // implemented later in this lesson
            }
            return false;
        });
        ...
    }
}
```

That's all we need to have a fully functional toolbar menu.



Sort dialog

It's time to handle the menu item click listener and show a sort dialog with two options: sort by date or title.

Start with adding fields that will indicate indexes of items in the sort dialog *(1)* and currently selected index *(2)*.

```
public class MainActivity extends AppCompatActivity {
```

```
    private static final int SORT_TITLE = 0; // 1
    private static final int SORT_DATE = 1; // 1

    private int currentSort = SORT_DATE; // 2
}
```

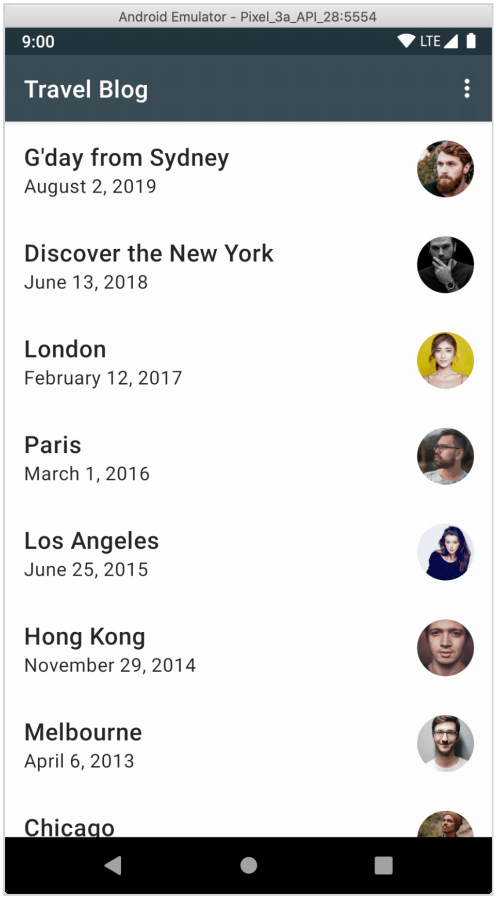
Next, let's implement the `onSortClicked` method. To create a single choice dialog, we can use the `MaterialAlertDialogBuilder` class along with the `setSingleChoiceItems` method with following parameters:

- `items` array of items to display
- `currentSort` pre-selected index of element from `items` array
- `OnClickListener` item click listener

Inside the `OnClickListener`, we need to close the dialog, save selected item index and update the list adapter.

```
public class MainActivity extends AppCompatActivity {
    ...
    private void onSortClicked() {
        String[] items = {"Title", "Date"};
        new MaterialAlertDialogBuilder(this)
                .setTitle("Sort order")
                .setSingleChoiceItems(items, currentSort, (dialog, which) -> {
                    dialog.dismiss();
                    currentSort = which;
                    sortData();
                }).show();
    }

    private void sortData() {
        if (currentSort == SORT_TITLE) {
            adapter.sortByTitle();
        } else if (currentSort == SORT_DATE) {
            adapter.sortByDate();
        }
    }
}
```

Now, when we launch the application and click on the sort menu item, we should see a selection dialog.



Sort

It's time to implement a sort logic inside the `MainAdapter`.

Start by implementing the `sortByTitle` method. Since we can't modify the current list, first we need to make a copy of it; to access the current list we can use the `getCurrentList` method *(1)*. Next, we can sort the list by title via the `Collections.sort` method *(2)*. Finally, to update the list we can use the `submitList` method *(3)*.

```
public class MainAdapter extends ListAdapter<Blog, MainAdapter.MainViewHolder> {
    ...
    public void sortByTitle() {
        List<Blog> currentList = new ArrayList<>(getCurrentList()); // 1
        Collections.sort(currentList,
                         (o1, o2) -> o1.getTitle().compareTo(o2.getTitle())); // 2
        submitList(currentList); // 3
    }
}
```

Implementing `sortByDate` will be a bit more challenging since our `date` filed in the `Blog` class has a `String` type. Before implementing `sortByDate` first, we need to add `getDateMillis` to the `Blog` class to convert `String` date value to

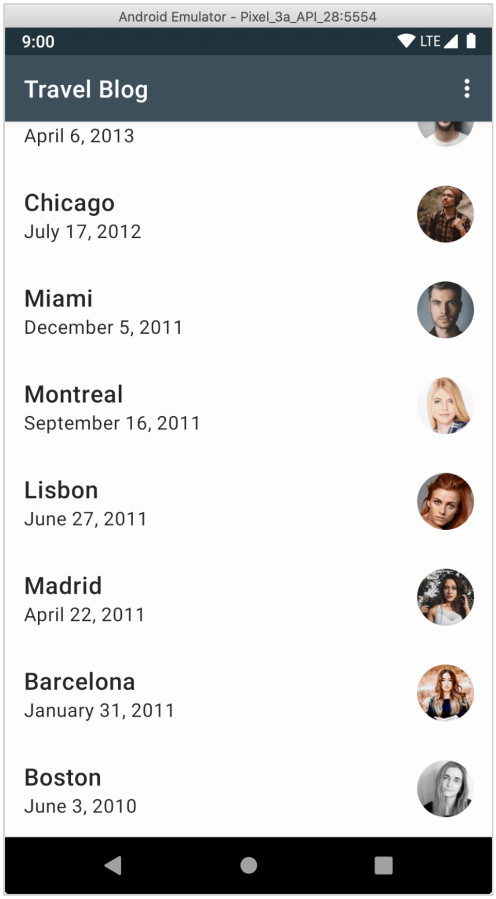
`Long`. We can easily do this using a `SimpleDateFormat` *(1)* to parse the `String` date value *(2)*.

```
public class Blog implements Parcelable {

    private static final SimpleDateFormat dateFormat =
                                new SimpleDateFormat("MMMM dd, yyyy"); // 1
    ...
    private String date;
    ...
    public String getDate() {
        return date;
    }

    public Long getDateMillis() {
        try {
            Date date = dateFormat.parse(getDate()); // 2
            return date != null ? date.getTime() : null;
        } catch (ParseException e) {
            e.printStackTrace();
        }

        return null;
    }
}
```

It's time to switch back to `MainAdapter` and implement `sortByDate` in a similar way we implemented `sortByTitle`.

```
public class MainAdapter extends ListAdapter<Blog, MainAdapter.MainViewHolder> {
    ...
    public void sortByDate() {
        List<Blog> currentList = new ArrayList<>(getCurrentList());
        Collections.sort(currentList,
                         (o1, o2) -> o2.getDateMillis().compareTo(o1.getDateMillis()));
        submitList(currentList);
    }
}
```

Now, when we launch the application and select a specific sort order, the list should be sorted and displayed.



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;

    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 sortByTitle() {
        List<Blog> currentList = new ArrayList<>(getCurrentList());
        Collections.sort(currentList, (o1, o2) -> o1.getTitle().compareTo(o2.getTitle()));
        submitList(currentList);
    }

    public void sortByDate() {
        List<Blog> currentList = new ArrayList<>(getCurrentList());
        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().equals(newData.getId());
                }
```

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

In the next lesson, we will take a look at how to add the search functionality to our blog list activity.