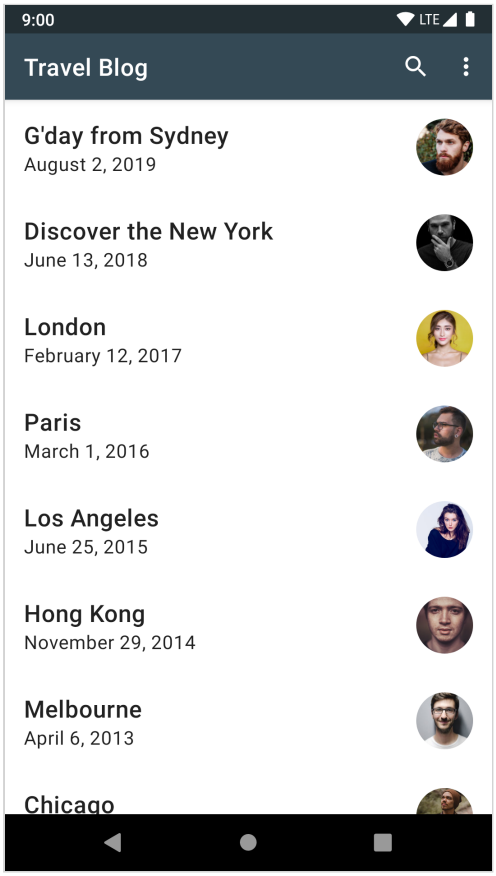
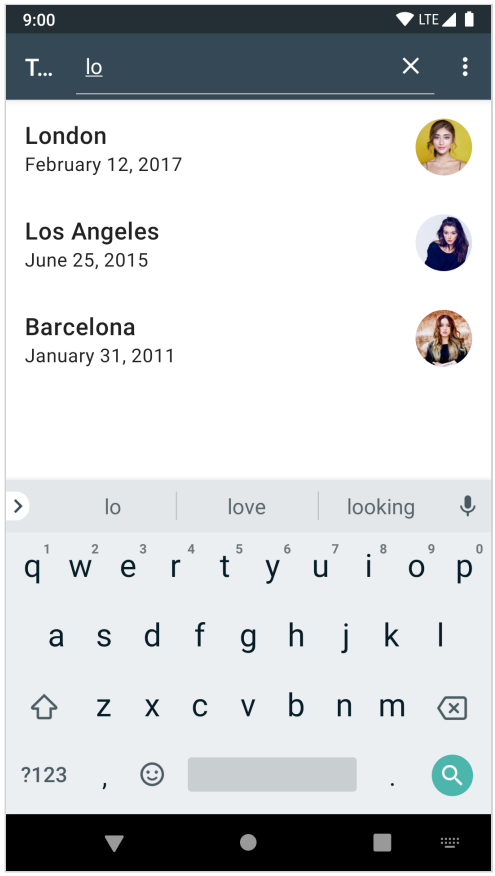
Search

This lesson will teach you how to add a search functionality to the toolbar and filter items in the blog list screen.

WE'LL COVER THE FOLLOWING

Final result preview





Toolbar menu

In a similar manner to how we added sort item to the toolbar menu, we can add search item with few adjustments:

- To always show the search icon, we will use the `showAsAction="always"`

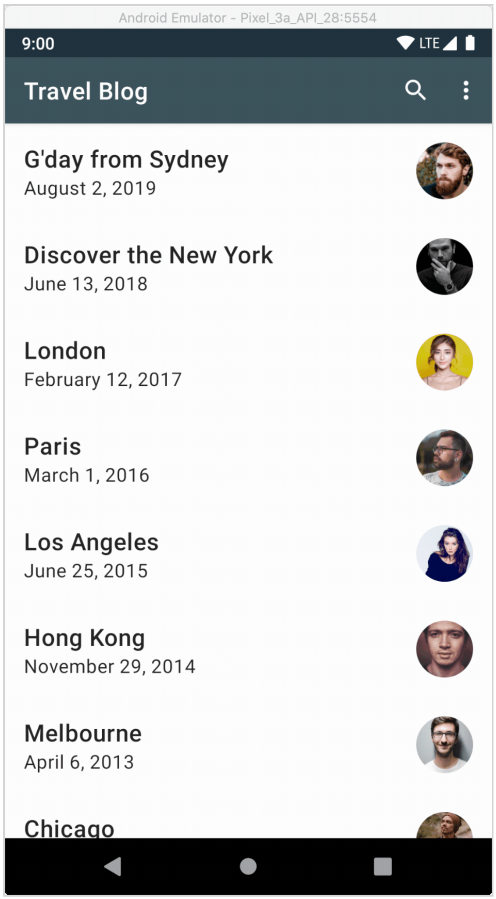
attribute and specify icon via `icon` attribute.

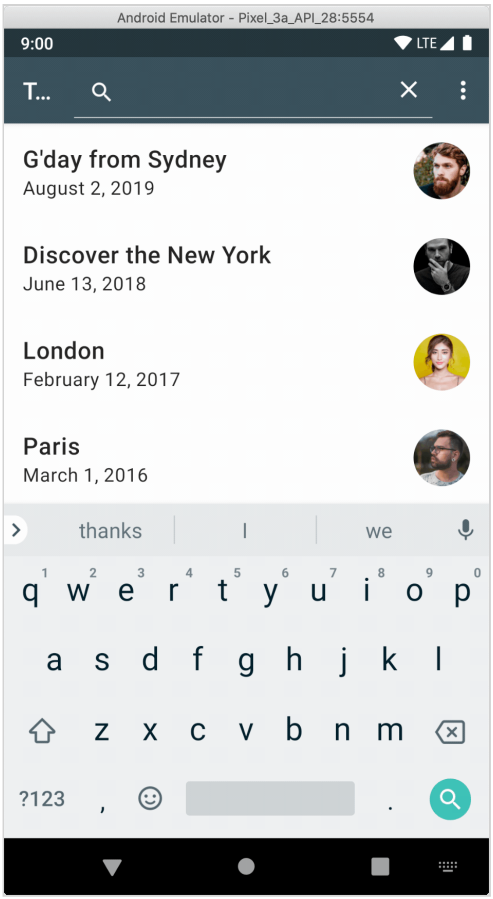
- The logic to expand the search icon to the search field is already available in the *Android* framework. We can use it by setting `actionViewClass` attribute value to the `android.widget.SearchView`.

```
<?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/search"
        android:icon="@drawable/ic_search_white_24px"
        android:title="Search"
        app:actionViewClass="android.widget.SearchView"
        app:showAsAction="always" />
<item
        android:id="@+id/sort"
        android:title="Sort"
        app:showAsAction="never" />
</menu>
```

main_menu.xml

With just a few lines of code, we have a search icon that expands search input fields along with a clear/close button.





The only issue is that the input field text color is too dark. Since we don't have

direct access to the input field, we can change the input field text color via a custom theme.

Create a new `MainTheme` in the `res/values/styles.xml` with parent set to `AppTheme`. Next, add a new item with `name="android:editTextColor"` and a value `@android:color/white`. Doing so tells *Android* to overwrite input field text color for all input fields, which are defined in the layout of the activity with this theme.

```
<resources>
    <style name="AppTheme" parent="Theme.MaterialComponents.DayNight.NoActionBar">
        <item name="colorPrimary">@color/blue700</item>
        <item name="colorSecondary">@color/orange500</item>
        <item name="colorPrimaryDark">@color/blue800</item>
    </style>

    <style name="MainTheme" parent="AppTheme">
        <item name="android:editTextColor">@android:color/white</item>
    </style>
</resources>
```

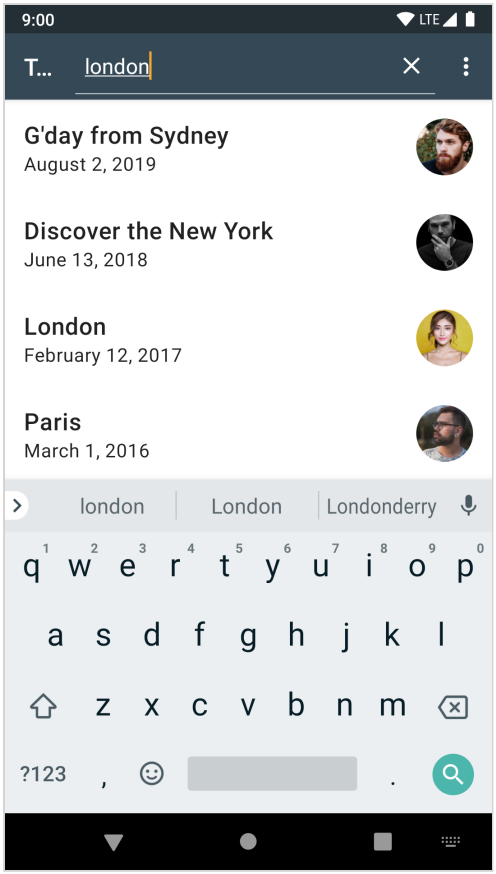
styles.xml

Finally, let's set the `theme` attribute of the `MainActivity` to `@style/MainTheme` in the `AndroidManifest.xml` file.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
        package="com.travelblog">
    <uses-permission android:name="android.permission.INTERNET" />
    <application
            android:label="Travel Blog"
            android:theme="@style/AppTheme">
        ...
        <activity
                android:name=".MainActivity"
                android:theme="@style/MainTheme" />
        ...
    </application>
</manifest>
```

AndroidManifest.xml

Now the text color of the input field is white.



In the next section, we will implement the search filter that will parse the list of blogs for the user-provided input.

Filter

Let's implement a filter logic in the `MainAdapter`. In order to filter the list and revert it back, first, we need to save the original list. Create a `setData` method, where we first save the list and then submit it to the adapter.

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

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

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

Now we can implement the `filter` method, where we just iterate through the original list *(1)*, compare blog title to the `query` *(2)* and submit matched items to the adapter *(3)*.

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

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

    public void filter(String query) {
        List<Blog> filteredList = new ArrayList<>();
        for (Blog blog : originalList) { // 1
            if (blog.getTitle().toLowerCase().contains(query.toLowerCase())) { // 2
                filteredList.add(blog);
            }
        }
        submitList(filteredList); // 3
    }
}
```

Finally, we can glue the adapter and the toolbar search in the `MainActivity`.

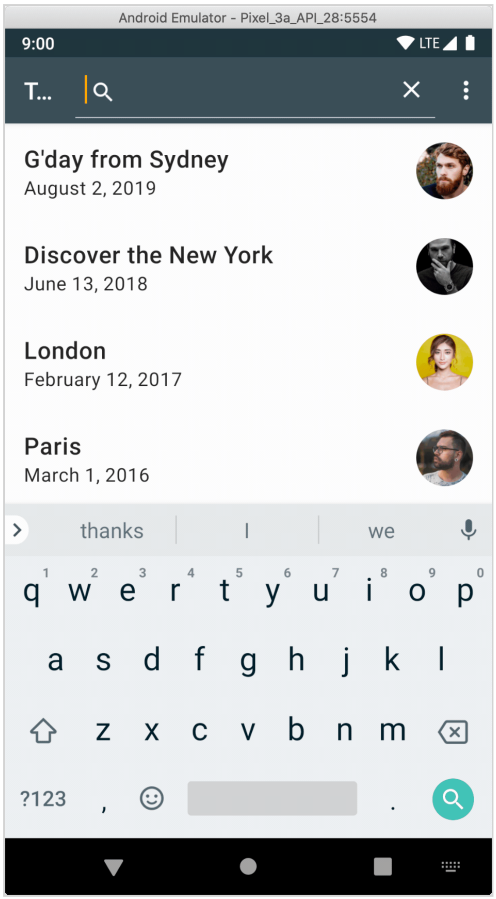
- *(1)* use the `findItem` method to bind search menu item to the *Java* `MenuItem` object
- *(2)* call the `getActionView` method to get a reference to the `SearchView`
- *(3)* set the `OnQueryTextListener` via `setOnQueryTextListener` method to have a trigger when search text has changed
- *(4)* use the previously created adapter `filter` method to filter the list items with matched blog article titles

```
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);
        ...

        MenuItem searchItem = toolbar.getMenu().findItem(R.id.search); // 1
        SearchView searchView = (SearchView) searchItem.getActionView(); // 2
        searchView.setOnQueryTextListener(new SearchView.OnQueryTextListener() { // 3
            @Override
            public boolean onQueryTextSubmit(String query) {
                return false;
            }

            @Override
            public boolean onQueryTextChange(String newText) {
                adapter.filter(newText); // 4
                return true;
            }
        });
}
```

One last step would be to use the adapter `setData` method *(1)* instead of `submitList` when the data is loaded, otherwise our list will not be saved.

```
public class MainActivity extends AppCompatActivity {
    ...
    private void loadData() {
        refreshLayout.setRefreshing(true);
        BlogHttpClient.INSTANCE.loadBlogArticles(new BlogArticlesCallback() {
            @Override
            public void onSuccess(List<Blog> blogList) {
                runOnUiThread(() -> {
                    refreshLayout.setRefreshing(false);
                    adapter.setData(blogList); // 1
                    sortData();
                });
            }

            @Override
            public void onError() {
                ...
            }
        });
    }
}
```

Now, when we launch the application and try to search, the blog list should be filtered.



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<>(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 chapter, we will begin by implementing offline support for our

travel blog application.