

## ACTIVITY 3.3.3

# Persistent People

### INTRODUCTION

In this activity, you will make your companions list persistent by saving it in the Backendless back-end service.

#### Materials

- Computer with Android™ Studio
- Android™ tablet and USB cable, or a device emulator
- Free Backendless account per student
- A Google account and a project on the Google API Console

### RESOURCES



**Activity 3.3.3 Visual Aid**  
Resources available online

## Procedure



### Part I: Get Selected Contacts

Now that you can select some people to join you on your trip, you need to display them in the list on the Trip Details screen. The `ListView` will use an adapter that lists the `Contacts` for a trip. To manage the list, you will make changes to `TripFragment`'s `onActivityResult` method as the app returns from the Contact fragment/activity.

- 1 Open your TripTracker app in Android Studio.

**NOTE**

If you were unable to complete *Activity 3.3.2 Polymorphic People*, import *3.3.2TripTracker\_Solution* as directed by your teacher. Recall that if you import the solution, you must update keys values in *strings.xml*:

- Change `be_app_id` and `be_android_api_key` to your Backendless App ID and Android API key values, respectively. You can retrieve these from your  **Backendless Console (Manage icon)**.
- Change the value for `google_app_id` to your Google Play service API key. You can retrieve it from the  **Google API Console (Credentials)**.

- 2 Review the flowchart for `TripFragment` in *3.3.3 Visual Aid*.
- 3 Modify your `Trip` class to include an array list called `companions`.
  - a. The type should be `Contact`.
  - b. Include the getter and setter.
  - c. Don't forget to initialize the array list in the constructor.
- 4 In `TripFragment onActivityResult`, get the list of companions from the intent.
  - a. Determine if the `requestCode` indicates you have a list of companions (as opposed to a start or end date).
  - b. Get the `ArrayList` of `Contacts` from the intent data stored in `data` using the `getSerializableExtra` method.
- 5 Update the trip with the `ArrayList` of `Contacts`, indicating the trip has a new set of companions.
- 6 Update the `ListView` with the new companions:
  - a. Create a `CompanionsAdapter` for the `ListView` (`mCompanionsListView.setAdapter()`).
  - b. Before you add any contacts to the adapter, use `clear()` to clear the old companions that used to be in the list. Otherwise, you will always be adding contacts to the `ListView`.
  - c. Add the contacts you found on the intent to the `CompanionsAdapter`.
  - d. Notify the adapter that it has new data with `notifyDataSetChanged()`.

Test your `TripTracker`; whatever contacts you select in `ContactFragment` should now appear in the list view in `TripFragment`. Note: The data is still not persistent, that is, you have not saved the trip in the back-end service, but you will in a few steps.

If you have added more than one contact to your trip, the scrolling companions `ListView` does not work within the scrolling screen. This is a known limitation of Android apps. To fix this, you will set a fixed size to the companions `ListView` and disable its scrolling.

- 7 To set the size of the list,
- Create a new convenience method `setListSize`, shown below.

```
1: /**
2:  * Set the size of the listview and disable scrolling
3:  *
4:  * pre-condition: the mCompanionsListView has been created
5:  */
6: private void setListSize(int numCompanions) {
7:
8:     mCompanionsListView.measure(0, 0);
9:     LinearLayout.LayoutParams params = new
10:     LinearLayout.LayoutParams(ListView.LayoutParams.
11:     MATCH_PARENT,
12:     (int) (numCompanions * mCompanionsListView.
13:     getMeasuredHeight()));
14:     mCompanionsListView.setLayoutParams(params);
15: }
16: }
```

- Call this from `onActivityResult`, passing the size of the `CompanionsAdapter` as a parameter.
- 8 Test your app a bit more by changing the list of companions a few times.

## Part II: Persistence in Backendless

Now that you have a list of contacts, they need to be saved in Backendless. First, you must modify Backendless to accommodate your list of contacts.

- 9 In the Backendless Console, do the following:
- Create a new table called **Contact**. On the **SCHEMA** page:
    - Create a **String** column called **name**.
    - Create another **String** column called **type**.
  - In the **Trip** table, use the **SCHEMA** page to add a **New** column:
    - For the name, use **companions**.
    - The type is **DATA OBJECT RELATIONSHIP**.
    - There are no constraints.
    - The **Related Table** is your new **Contact** table.
    - The cardinality is **One-to-Many**.

Click **CREATE**.

- c. Back in the **DATA BROWSER**, check the box for **autoload**.
- d. Observe your new Contact table and explain what the column **Trip[companions]** represents.

### Check your answer

It stores the objectId of the Trip that owns this entry. Specifically, it denotes that this entry in the Contact table is a child of the Trip's companions column.

You are now ready to save the trip's companions in the database.

- 10 In `TripFragment.updateTrip`, these sub-steps will add the companions from the `ListView` to the trip.
- a. Find the thread that saves the trip. After the save, get the companions `ListView` adapter.
  - b. Use a call to `getCount ( )` to get the number of companions stored in the adapter, in other words, the number of contacts the user has selected from the Trip Companions screen.
  - c. If there are companions, iterate over the companions adapter to get all items in the list, using a standard for loop. Then:
    - i. Create a new contact, setting the name and type with values from the adapter, and then saving to the database:

```
1: Contact newContact = new Contact();
2: newContact.setName(ca.getItem(i).getName());
3: newContact.setType(ca.getItem(i).getType());
4: Backendless.Data.save(newContact);
```

- ii. You will have an error in this code; fix it by creating a default constructor for `Contact`. Backendless will also use this default constructor to manage the `Contact` table in the database.
    - iii. Add each new contact to a new `ArrayList` list of `Contacts`.
  - d. After saving all of the new contacts, use the trip's `setCompanions` and your new `ArrayList` to set the companions for `mTrip`.
  - e. Similar to how you saved geolocations for a trip, use `setRelation` to save the list of companions for the trip:
    - i. In place of `locations:GeoPoints:n` use `companions:Contact:n` to indicate the relationship you want.
    - ii. Use your `ArrayList` of companions in place of `tripLocations`.
- 11 To test your app, add companions to your trip and save. (The companions will *not* show in your list yet; that is the next part.)

- 12 In Backendless, confirm the list of companions. Observe the changes to your Trip table and its data. How are companions represented in your Trip table?

**Check your answer**

They are Contact objects with their objectIds.

## Part III: From Backendless to Your App

To retrieve trip companions from Backendless and to populate the companions list, review the tasks you need to complete. Each task is described in detail starting in the next step (Step 13).

- Put the trip companions on the intent.
  - In `TripFragment`, retrieve the companions from the intent.
  - Use the adapter to load the companions array into the companion's list.
  - Display the companions in the list.
- 13 In Backendless, confirm that your **Trip's companions** column has "autoload" checked.
- 14 In `TripListFragment`, use the `putExtra` method to put the trip's companions on the intent when an item in the list is clicked.
- 15 In `TripFragment onCreate`, use `getSerializableExtra` to retrieve the `ArrayList` companions from the intent, then add them to `mTrip` with a call to `setCompanions` method.
- 16 Like you did in Part I, add companions to your `ListView`. In `TripFragment onCreateView`:
- a. Load the companions into the `CompanionsAdapter`. Because you are creating and not modifying the adapter, you do not need to add the call to `notifyDataSetChanged`.
  - b. Set the size of the `Companions ListView` with a call to `setListSize`.
- 17 Test your app and use the Backendless Console to confirm your trip companions are properly loaded in a Trip.

When your Trip List is populated, all companions are also loaded for each trip. Can you think of a situation where this autoload feature might not be desirable or efficient?

**Check your answer**

What if a cruise ship company were to use this app? The database would have a large number of trips and each trip would have a large number of companions. If all companions were loaded for all trips, the data set would be huge and would likely slow down the app to a crawl and consume all available storage capacity.

## Part IV: Replace / Delete Companions

---

Observe your Contact table in Backendless. As you change companions for a trip, you are always adding to the Contact table, without first removing the old contacts. This results in “orphaned data”, that is, data no longer used or needed in the database. It is a common problem with database management, and you need to fix this problem.

- 18 First, to delete contacts, Backendless needs to know the `objectIds` for all companions. Create a `String objectId` variable in your `Contact` class, along with its getter and setter. Through polymorphism, all subclasses now have an `objectId`, too.
- 19 In `TripFragment`, create a new, global variable `mPreviousCompanions`.
- 20 When you get the companions off of the intent, set `mPreviousCompanions`. This now contains the list of contacts to delete before adding new ones.
- 21 Before you save the trip in `updateTrip`, delete old companions from Backendless:

```
1: if (mPreviousCompanions != null) {
2:     for (Contact c : mPreviousCompanions) {
3:         Backendless.Data.of(Contact.class).remove(c);
4:     }
5:     Log.i(TAG, "Deleted old companions");
6: }
```

- 22 Test your app to confirm old contacts are deleted when you save a trip with new/different companions.
- 23 When you delete an entire trip, you are leaving behind orphaned data, the companions for the trip. Before you delete an entire trip, delete the companions from the trip, using the same algorithm as in `updateTrip`. Since you are reusing this code, this is a good place to create a helper method.

## Part VI: Regression Test and Cleanup

---

- 24 Test thoroughly and confirm that all of your companions are saving properly.
  - a. You are displaying trip companions when the trip is a public trip, but the names of your family and friends are considered private information. Show them only if you are the owner of the trip (recall the `mEnabled` variable).
  - b. Test to be sure other features are not affected by your changes, such as locations, cost, and rating.

- c. Do some “cleanup” tasks. In *TripFragment.java* and *ContactFragment.java*, there are a few instance variables whose scope is too broad; they do not need to be class-wide instance variables. Convert them to local variables, removing the `m` reference in their names.

25 Re-test your app to verify its functionality.

## CONCLUSION

1. When you create a new trip in Backendless, the Array of companions is undefined. If you delete all companions, the Array is empty. Explain how these two conditions are handled in your code.
2. If you wanted to check if your companions list exists, you would use the following if statement.

```
if ((mTrip.getCompanions() != null) &&  
    !mTrip.getCompanions().isEmpty())
```

This performs a short-circuit test. Explain what the conditional statement accomplishes and then rewrite the statement using DeMorgan’s Law. Is there a scenario where the rewritten version may cause a runtime error?

# Activity 3.3.3 Visual Aid

## TripFragment

This flowchart represents new functionality to add to TripFragment

