ACTIVITY **3.1.3**

A New Trip

INTRODUCTION

Your TripTracker app will gain a lot of new functionality in this activity. Much of it will be provided for you, especially the view and presentation layers for the user interface. You will provide the functionality to create Trip objects, to retrieve trip data that a user enters, and to save trip data in the back end.

Materials

- Computer with Android™ Studio
- Android™ tablet and USB cable, or a device emulator
- Free Backendless account per student

RESOURCES

Lesson 3.1 Reference Card for Backendless
Resources available online

Activity 3.1.3 Visual Aid
Resources available online

Procedure

Part I: Create a Trip Class

With a successful registration and login, the user is ready to create a new trip.

To represent a trip in your app, you need a new class, "Trip". The Trip class requires specific variable references (see step 1), because these instance variables are used to define the trip's data in Backendless.

1. Review the design you discussed at the beginning of this unit (in *3.1.1 TripTracker Specifications*) to represent a trip. Discuss where the instance variables match what you chose and where they differ.
   - A String for `objectId`
   - A String for `name`

- A String for `description`
- A Date object for `startDate`
- A Date object for `endDate`
- A boolean for a `shared`

2. Open your TripTracker app in Android Studio.

NOTE

If you were unable to complete *Activity 3.1.1 TripTracker Startup* and *Activity 3.1.2 User Authentication*, open *`3.1.2TripTracker_Solution`* as directed by your teacher. Recall that if you import the solution, you must update keys values in *`strings.xml`* Specifically,

- Change `be_app_id` your Backendless App ID key value
- Change `be_android_api_key` to your Android API key value

You can retrieve these from your **Backendless Console** (**Manage** icon).

3. Create a new `Trip` class.
   a. Define the instance variables as stated above.
   b. Create the getters and setters for your instance fields.

Part II: Create the Date Picker

4. Get a copy of *`3.1.3TripTracker_StarterCode`* from your teacher and copy or extract the individual files to a location that is convenient, but *not* in your project folder. Your Desktop is an example of a good location.

   You should have five Java files and eight XML files in the *`res`* folder.

5. Your TripTracker is going to include information about the dates of your trip, so your app needs a way to manage dates. From your copy of the starter code:
   a. Copy *`dialog_date.xml`* into your project's *`res/layout`* folder.
   b. Copy *`DatePickerFragment.java`* into `org.pltw.examples.triptracker`.

Part III: Create the Strings

6 Add the following definitions to your *strings.xml* file.

```
<!-- 3.1.3 fragment_trip_list -->
<string name="trip_list_text">Trip List</string>

<!-- 3.1.3 fragment_trip -->
<string name="trip_details_text">Trip Details</string>
<string name="trip_name_label">Name</string>
<string name="trip_name_hint">Please enter a name for your trip.</string>
<string name="trip_details_label">Details</string>
<string name="trip_desc_hint">Please enter a description for your trip.</string>
<string name="start_date_label">Start Date</string>
<string name="end_date_label">End Date</string>
<string name="start_date_hint">Select a Start Date:</string>
<string name="end_date_hint">Select an End Date:</string>
<string name="trip_public_label">Shared/Public?</string>
<string name="trip_error_message">Please make sure to enter a trip name!</string>
<string name="trip_error_title">Error!</string>
<string name="delete_error_title">Permission Denied!</string>
<string name="delete_error_message">You cannot delete a trip that you do not own!</string>
<string name="post_error_title">Permission Denied!</string>
<string name="post_error_message">You cannot modify a trip that you do not own!</string>

 <!-- 3.1.3 menu items -->
<string name="action_settings">Settings</string>
<string name="action_post">Post</string>
<string name="action_delete">Delete</string>
<string name="action_logout">Logout</string>
<string name="action_refresh">Refresh</string>
<string name="action_new">New</string>
<string name="action_public_trips">Public Trips</string>
<string name="action_my_trips">My Trips</string>
```

As you incorporate the next few Java and XML files, you will be introducing elements not yet defined in your code. You will not be able to compile your code until everything has been defined, so don't bother to compile until you are instructed to do so.

Part IV: Create Menus

7. Your app will use two option menus and one context menu.

   What is the difference between an Options menu and a Context menu?

8. Right-click the *res* folder and select **new > Android resource directory**. From the Resource type menu, select **menu** and click **OK**. Copy the three menu files from your copy of *3.1.3TripTracker_StarterCode* into the app's menu folder:

   - *`menu_trip_details.xml`*
   - *`menu_trip_list_item_context.xml`*
   - *`menu_trips.xml`*

Part V: Create the Activity and Fragment: Show a Trip

In addition to LoginActivity, TripTracker will have two other activities, and each activity requires a definition in *`AndroidManifest.xml`*. The first activity will show the list of all trips found in the back-end service.

9. Open the app's *`AndroidManifest.xml`* file and define the new activity by adding lines 2–8 inside the `application` tag.

```
1:      ...
2:      <activity
3:          android:name=".TripActivity"
4:          android:label="@string/trip_list_text" >
5:          <meta-data
6:               android:name="android.support.PARENT_ACTIVITY"
7:               android:value=".TripListActivity" />
8:      </activity>
        ...
    </application>
```

   The activity is defined by its `name` and includes `meta-data`, which indicates that this activity will be able to navigate to its parent activity, the TripList activity.

   Instead of placing all of your UI elements directly in TripActivity as you did with LoginActivity, you will use a fragment to hold the UI elements.

10. From your copy of *3.1.3TripTracker_StarterCode*:

    a. Copy *`fragment_trip.xml`* and *`activity_trip.xml`* to the app's layout folder (*`res/layout`*).

b. Copy *TripFragment.java* and *TripActivity.java* to the app's Java folder (*org.plstw.examples.triptracker*).

Part VI: Create the Activity and Fragment: List All Trips

11 The second activity is the activity that will list all of the trips currently in the back-end service. In the *AndroidManifest.xml* file, define the activity by adding lines 2–5 inside the `application` tag.

```
1:    ...
2:    <activity
3:        android:name=".TripListActivity"
4:        android:label="@string/trip_list_text" >
5:    </activity>
      ...
   </application>
```

12 Like the Trip activity, TripList activity will also use fragments.

a. Copy the layout files *fragment_trip_list.xml* and *activity_trip_list.xml* to the layout folder.

b. Copy *TripListFragment.java* and *TripListActivity.java* to the Java folder.

You now have all of the new user interface elements for your app. Review the UML diagrams in *3.1.3 Visual Aid* and become familiar with the flow of information in your TripTracker app.

With the UI elements in place, it is time for you to provide the new functionality.

Part VII: Manage the Intent Data

13 Open *TripFragment.java*.

Notice the many undefined constants such as `Trip.EXTRA_TRIP_ID`. These constants are used to represent **key-value pairs** used to share data on the intent, and therefore, shared between activities. A key-value pair is a set of data items that associate values with known keys. In the examples "color:blue" and "book:Dracula", color and book are the keys and blue and Dracula are the values.

key-value pairs
A set of two data-related items; a key that represents a known entity such asnameorsize, and a value that represents the specific data such as "*Cynthia*" or "*medium*".

Using **intent data**, your app can pass information between different activities. But if you place these constants in your `Trip` class, Backendless will save all of the constants along with the instance variables for every trip you create. This represents a large amount of wasted data and wasted time creating unnecessary information. But your trip still needs this data.

Can you think of a way to save this information for each trip in Backendless without defining it *in* the `Trip` class itself?

14. If you thought of using an interface, congratulations! Create a new Java class and specify **Interface** for Kind. Name the new interface **IntentData**.

15. In *`IntentData.java`*, create the following constants as `Strings`, as shown below.

```
String EXTRA_TRIP_ID = "org.pltw.examples.triptracker.
TRIP_ID";
String EXTRA_TRIP_NAME = "org.pltw.examples.triptracker.
TRIP_NAME";
String EXTRA_TRIP_DESC = "org.pltw.examples.triptracker.
TRIP_DESC";
String EXTRA_TRIP_START_DATE = "org.pltw.examples.
triptracker.TRIP_START_DATE";
String EXTRA_TRIP_END_DATE = "org.pltw.examples.triptracker.
TRIP_END_DATE";
String EXTRA_TRIP_PUBLIC = "org.pltw.examples.triptracker.
TRIP_PUBLIC";
String EXTRA_TRIP_PUBLIC_VIEW = "org.pltw.examples.
triptracker.TRIP_PUBLIC_VIEW";
```

16. Modify `Trip` so it implements the `IntentData` interface.

    A trip can now access all of the data it needs to share with other activities *without* writing all of the intent data constants into Backendless.

17. With two modifications to `LoginActivity`, you will be able to compile and test TripTracker with the new activities.

    a. After a successful login, start the TripList activity:

```
Intent intent = new Intent(LoginActivity.this,
                           TripListActivity.class);
startActivity(intent);
```

    This will automatically dismiss the progress dialog you wrote in the previous activity and start `TripListActivity`.

b. After a user registers, they must still log in with their new account. Restart the Login activity with:

```
1: Intent intent = new Intent(LoginActivity.this,
   LoginActivity.class);
2: startActivity(intent);
```

18 Compile and test your app by logging in.

a. With a successful login, the app will display a “My Trips” screen that was inflated from the `TripListFragment`. (The terms “My Trips screen” and “TripList screen” are used interchangeably throughout this lesson.) The screen displays “no data” because you have no trips defined.

b. In the action bar, you will see:

- The title of the screen
- A REFRESH option
- An Add icon
- The option menu

19 Identify the resources in this project that belong to the Model, View, and Presenter layers. Include the Java and xml files.

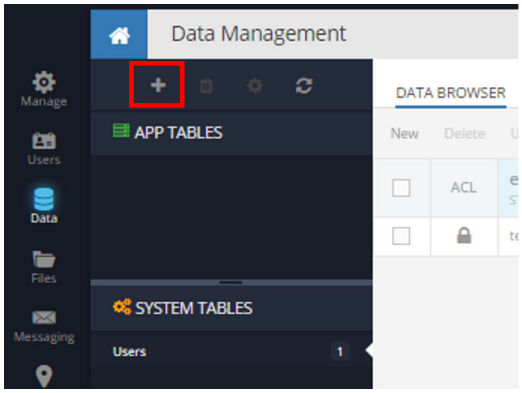
You will become familiar with all of these features in the next few activities. For now, you will implement the functionality for the Add icon.

Part VIII: Create a Table in Backendless

Before you add your first trip in your app, you will create a table manually in Backendless that will store all of your Trips. This will help you become more familiar with how Backendless provides database functionality in your app.

20 Log in to the Backendless Console and navigate to the Data view for your TripTracker app.

21 To create a new table, click the + icon above the APP TABLES panel.



22. In the popup that appears, enter **Trip** for the Table name and click **CREATE**.

23. In the message box that asks, “Would you like to switch to Schema Editing to configure table schema?”, click **NO**.

    The DATA BROWSER view is shown for your new table. Notice the column names in your Trip table. Backendless automatically created four columns, when you created a table.

    What data type is `objectId`?

    What do you think is the purpose of the `objectId` column?

    What do you think is the purpose of the `ownerId` column?

In the SCHEMA view of a table, you can create the columns for the table. Manually creating columns will give you insight into what Backendless does behind the scenes when it performs a save.

24. In the menu bar above the table data, click **SCHEMA**. Because you have not created any custom fields for your Trip table, Backendless automatically displays the **New Column** popup.

25. Use the **New Column** popup to create your first column:

    a. For the new column Name, enter **name**, representing the name of a trip.

    b. For Type, match the data type you used for your `name` instance field in your Trip class.

    a. Do not enter anything for Default Value, Constraints, or Validator.

    b. Click **CREATE**.

    Your Trip table should now show a new COLUMN NAME, “name”.

26. Above the table data and below the menu bar, you should see links for the Schema functions **New** and **Delete**. Except for `objectId` which is already defined, use the **New** link to create a column for the other instance variables in your `Trip` class, matching the name and the data type in Backendless to your `Trip` class. (Use a `DATETIME` data type in Backendless for a `Date` type in Java. Make the column names exactly match the instance variable names, including capitalization.)

27. When you are finished adding columns to the `Trip` table, click **DATA BROWSER** in the menu bar to return to the Data View. You should see your columns listed across the page, followed by the columns that were automatically created by Backendless.

    In what order are your new, custom columns listed?

Part IX: Save a Trip

28. Open *`TripFragment.java`* and become familiar with some of the methods by answering the following questions.
    a. Can you tell what the Intent Extra `EXTRA_TRIP_ID` is used for in the `onCreate()` method of `TripFragment`?
    b. Also in the `onCreate()` method, what are the `mEnabled` and the `mPublicView` attributes used for?
    c. Refer to the code in `TripFragment` and review the `MyDateOnClickListener` and `onActivityResult()`. Can you explain how the date values for a trip's start date and end date get updated and how they get displayed? How does the code tell if it is the start or end date being updated?
29. Starting at the beginning of `TripFragment`, search for **mTrip** and review all occurrences.

    What does `mTrip` represent?

    In `onCreate`, what does a `tripId` of `0` indicate?
30. To find "to do" items, select **View > Tool Windows > TODO**. Go to the code for the TODO that instructs you to save the trip in Backendless and review the existing code in the `updateTrip` method.
31. Use your mutator (setter) methods for `mTrip` to set the values for the trip's data items:
    a. name
    b. description
    c. startDate
    d. endDate
    e. shared

    Do not set the trip's `objectId`, because Backendless automatically sets this when the record is created in the table.

As you did in CollegeApp, you will save and retrieve your trips in the remote database. In TripTracker, you will often need a save/update process before the user can continue in the app. To wait for a save to finish in Backendless, you will need to use a thread. A **thread** is a small, independent set of instructions you can have executing at the same time as your main app. This is often referred to as a "thread of execution".

thread
A small set of instructions that are executing separately from and simultaneously to the main thread, or app.

32 Start a new thread and use a `try/catch` with an `AlertDialog`, if there is an `InterruptedException`. If there is no error, log a helpful message saying the trip was saved.

```
// save on a new thread and wait for the save to finish
Thread thread = new Thread(new Runnable() {
    @Override
    public void run() {
        Backendless.Data.of(Trip.class).save(mTrip);
    }
});
thread.start();
try {
    thread.join();
} catch (InterruptedException e) {
    Log.e(TAG, "Saving trip failed: " + e.getMessage());
    AlertDialog.Builder builder = new AlertDialog.
    Builder(getActivity());
    builder.setMessage(e.getMessage());
    builder.setTitle(R.string.trip_error_title);
    builder.setPositiveButton(android.R.string.ok, null);
    AlertDialog dialog = builder.create();
    dialog.show();
}
```

This code begins a new `Thread`, defines its run method, `starts` the thread and then waits for the thread to finish with `thread.join()`. You must have a `try/catch` for the `join`, in case the thread has problems and does not finish executing properly.

33 One last step: In `onOptionsItemSelected`, invoke the `updateTrip` method to save the trip when the user selects the Post/Save option.

34 Test your app. Create trips and confirm that they were created in Backendless using the Backendless Console.

NOTE

- The progress bar (a rotating circle) will continue to rotate until you go back to your TripList/My Trips screen
- Your TripList/My Trips screen is still empty, because you have not populated it with your new trip(s).

You will fix both of these problems in the next activity.

CONCLUSION

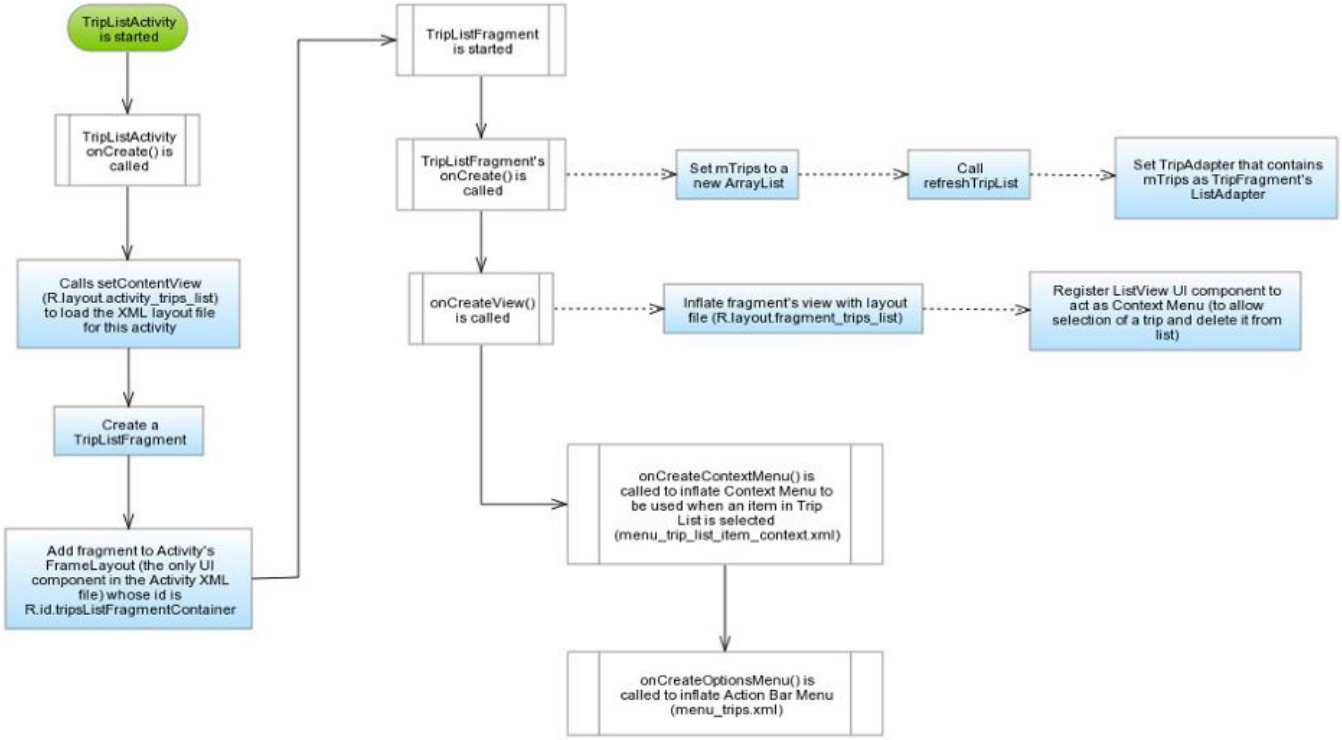
1. Create comments for the methods in `TripFragment`, reviewing the code to determine their functionality. Copy and paste the comments here. The first one has been done for you.

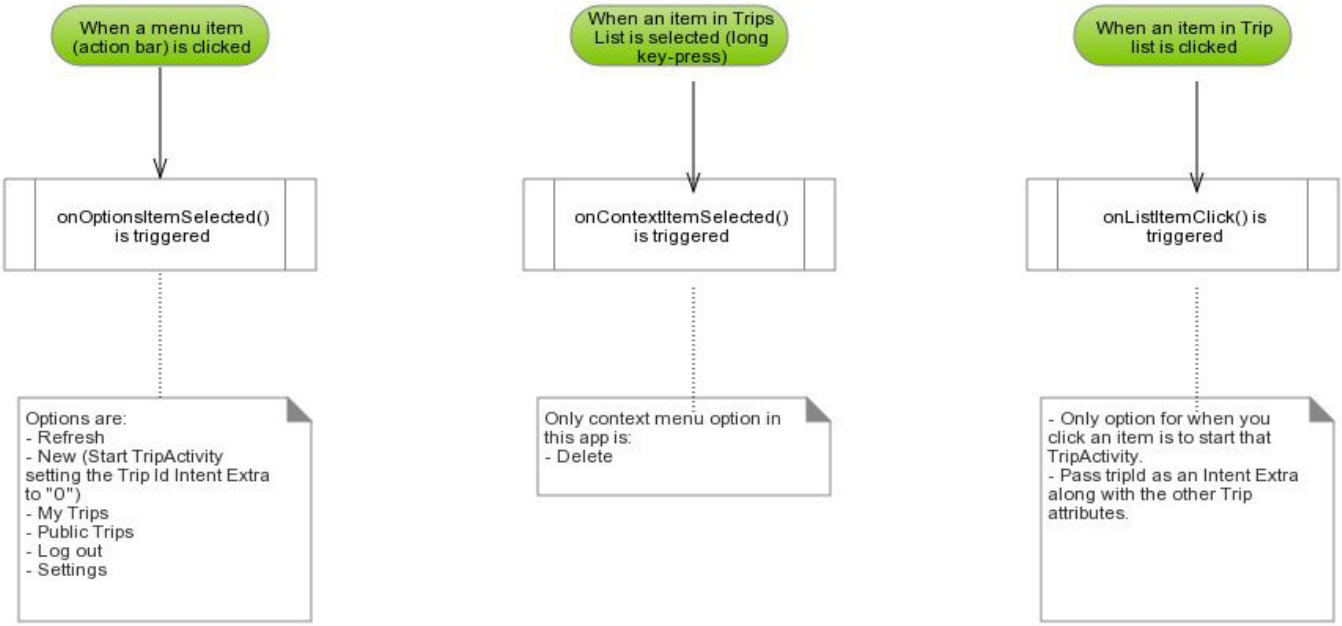
Method	Comments
`onCreate`	*/* *Create the options menu, initialize backendless, get the current trip from the intent (if any) and define mTrip* **/*
`onCreateView`	
`myDateOnClickListener` (especially, how both start and end date use this one listener)	
`onCreateOptionsMenu`	
`onOptionsItemSelected`	

Method	Comments
`onActivityResult`	
`updateDateView`	
`getDataFromView`	
`updateTrip`	

Activity 3.1.3 Visual Aid

TripListActivity and TripListFragment





TripActivity and TripFragment