

# Lesson 3.1: Trip Tracker

## ACTIVITY 3.1.1

# TripTracker Startup

### INTRODUCTION

In the activities of this lesson, you will create a TripTracker app that allows users to maintain trip information from vacations and other travel using multiple devices. You will be able to track the places you go and when you travel; you'll be able to rate your trips, add friends from your contact list, and save and share data with other users.

#### 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




**TripTracker Specifications**  
Resources available online

## Procedure

### Part 1: Create a TripTracker app

All software development processes involve some up-front planning. In the traditional Waterfall approach, the design is completely developed prior to the start of the software build phase (linear model). In the Agile approach, the design, build, and testing phases evolve together in a more adaptive approach, allowing for earlier product delivery and frequent enhancements

(cyclic model). In either approach, some initial planning has to occur to ensure successful development.

- 1 Read through  **TripTracker Specifications** for the details about app requirements. Work with your partner to identify the entities that need to be stored in the database.

**NOTE**

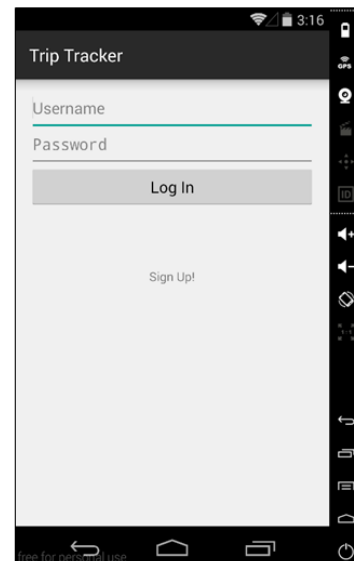
One simple method for identifying all the entities is to record all the nouns in the specifications. Then you can classify those nouns as a class or class attribute.

- 2 Read through the specifications again and make a list of the classes and their attributes.
- 3 Review the presentation. Describe how it compares with the plans you made for your app.

## LoginActivity

### Class Review

- Main Activity in AndroidManifest.xml
- Content View = activity\_login.xml
- No Fragments
- Sign Up link to sign up a new user
- Log In button to log in to the app



Computer Science A

© 2016 Project Lead The Way, Inc.

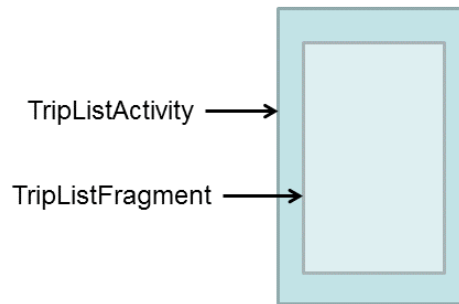
At login:

- Username and Password must not be empty
- On successful login, loads the TripListActivity

# TripListActivity

## Class Overview

- The first activity loaded after a successful login or sign up.
- Extends the AppCompatActivity.
- Contains one frame layout to hold the TripListFragment.



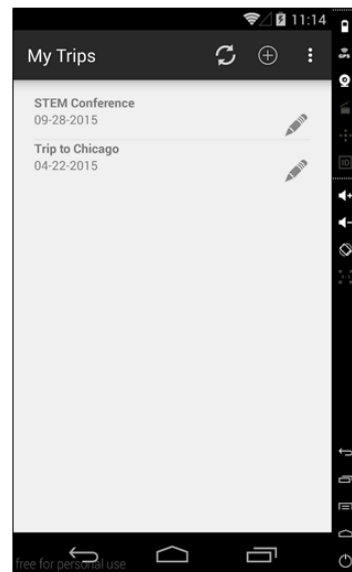
Computer Science A

© 2016 Project Lead The Way, Inc.

# TripListFragment

## Class Overview

- Displays trips list
- Contained in TripListActivity layout
- Content View = fragment\_trips\_list.xml
- Action bar menu Items:
  - Refresh
  - New
  - Shared Trips
  - My Trips
  - Logout
  - Settings



Computer Science A

© 2016 Project Lead The Way, Inc.

- Refresh: updates the list from the database
- New: goes to Trip screen to add a new trip
- Shared Trips: goes to the list of all users' shared trips
- My Trips: goes to the user's list of trips
- Logout: signs the user out and goes back to the Log In screen
- Settings: (not implemented)

# TripListFragment

- Uses fragment\_trips\_list\_item layout

- Each list item displays:

- Trip Name
- Trip Start Date
- Edit icon



- The Context menu has one item:



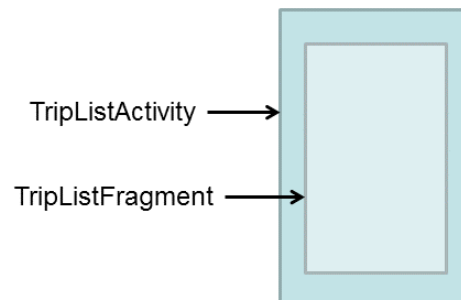
Computer Science A

© 2016 Project Lead The Way, Inc.

# TripActivity

## Class Overview

- Activity loaded when:
  - User touches New to add a new trip
  - User touches an item from the Trip List Fragment to view its details
- Extends the AppCompatActivity.
- Contains one frame layout to hold the TripFragment.



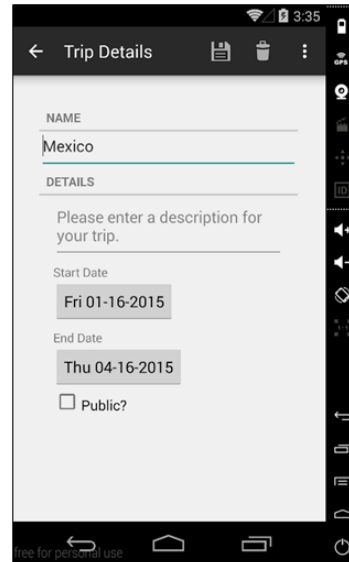
Computer Science A

© 2016 Project Lead The Way, Inc.

# TripFragment

## TripFragment class overview:

- Displays the trip details
- Contained in TripActivity layout
- Content view derived from fragment\_trip.xml
- Action bar contains:
  - Post (save)
  - Delete
  - Logout (shown on menu only)
  - Settings (shown on menu only)



Computer Science A

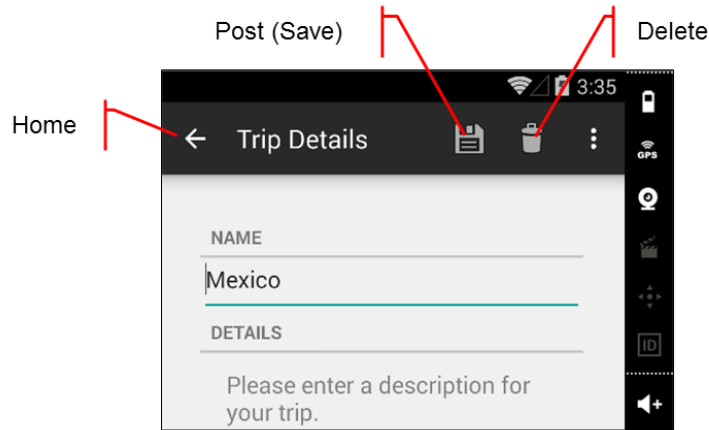
© 2016 Project Lead The Way, Inc.

## The action bar for TripFragment:

- Post saves the trip details to the database
- Delete removes the trip from the database
- Logout (on menu) signs the user out and goes back to Login Screen
- Settings (on menu) not implemented

Keep in mind that the same screen has to be flexible enough to handle a new record (insert to the database) and an existing record (update to the database).

# TripFragment



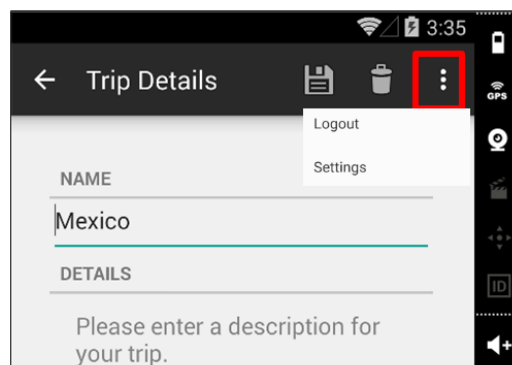
Computer Science A

© 2016 Project Lead The Way, Inc.

The action bar works in a similar way to the TripsListFragment.

- The  $\beta$  option (referred to as the Up button) is automatically enabled whenever a parent activity is defined for the current activity in the AndroidManifest.xml.
- If an activity on the screen is the topmost one in an app, it is considered the app's Home, and the Up button will not be shown.
- Because the TripListActivity is defined as the parent activity for this TripActivity, the Up option is enabled to the left of the screen name. (like a browser's back button).
- In TripFragment, the Up button will effectively Cancel any changes made to a trip.
- The Post (Save) option calls the updateTrip() method which is currently incomplete. You add code to it in this activity.
- The Delete option calls the deleteTrip() method which is currently empty as well. You add code to it in this activity.

# TripFragment



Computer Science A

© 2016 Project Lead The Way, Inc.

The Options for TripFragment show:

- Logout, the same as the one from TripsListFragment.
- Settings (not implemented)

- 4 Create a new project in Android Studio. Give this project the following values (when no value is specified, leave the defaults):
  - a. Application name: TripTracker
  - b. Company Domain: examples.pltw.org
  - c. Project location: Store your project in your `AndroidProjects` folder
  - d. Minimum SDK: API 22
  - e. Add an activity to Mobile: Empty Activity
  - f. Activity Name: LoginActivity
- 5 Open the `activity_login.xml` file
- 6 Replace `RelativeLayout` with `LinearLayout` as shown on line 2.

```
1: <?xml version="1.0" encoding="utf-8"?>
2: <LinearLayout xmlns:android="http://schemas.android.com/apk/
  res/android
3:               xmlns:tools="http://schemas.android.com/tools"
  ...
```

As you type, note how the closing tag at the end of the file automatically changes to `LinearLayout`.

- 7 In your `LinearLayout`, add a line to set `android:orientation` to `vertical`.
- 8 Delete the “Hello World!” `TextView`.

As you create the next few UI elements, the `@string` values will not have been defined. You will create those in a few steps.

- 9 As the first view in the layout, add the `EditText` shown below.

```
1: <EditText
2:     android:id="@+id/enter_email"
3:     android:layout_width="match_parent"
4:     android:layout_height="wrap_content"
5:     android:ems="10"
6:     android:hint="@string/email_hint" >
7:     <requestFocus />
8: </EditText>
```

What do you think the `requestFocus` tag does?

Why do you think `textPassword` is used as the `android:inputType` for the password textbox?

- 10 Create another EditText widget like the one above, except:
  - a. For the id, use `android:id="@+id/enter_password"`
  - b. Add an attribute for the type of input: `android:inputType="textPassword"`
  - c. Do not have it `<requestFocus />`
- 11 Create a button.

```
1: <Button
2:     android:id="@+id/login_button"
3:     android:layout_width="match_parent"
4:     android:layout_height="wrap_content"
5:     android:text="@string/login_button_label" />
```

- 12 Create a new TextView element.

```
1: <TextView
2:     android:id="@+id/sign_up_text"
3:     android:layout_width="match_parent"
4:     android:layout_height="wrap_content"
5:     android:layout_marginTop="@dimen/spacer"
6:     android:text="@string/sign_up_text"
7:     android:gravity="center" />
```

Notice that the `sign_up_text` has the property `android:layout_marginTop="@dimen/spacer"`. Just like string values are added to the `strings.xml` file, dimensions are added to a similar file to avoid any hard-coded dimension values in the layout file.

- 13 Open the `dimens.xml` file located in the `res/values` folder and add line 4 from below.

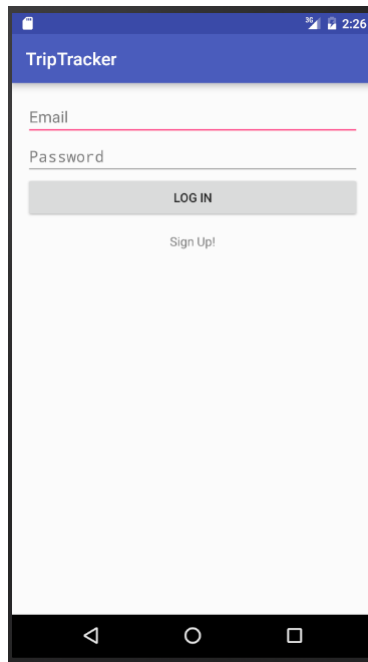
```
1: <resources>
2:     <dimen name="activity_horizontal_margin">16dp</dimen>
3:     <dimen name="activity_vertical_margin">16dp</dimen>
4:     <dimen name="spacer">16dp</dimen>
5: </resources>
```

- 14 Add the following string resources to the `strings.xml` file.
  - a. `email_hint`: Email
  - b. `password_hint`: Password
  - c. `login_button_label`: LOG IN
  - d. `sign_up_text`: Sign Up!



- 15 Save and test your code on your Android device.

Your app should look like the screen below.



**NOTE**

Note that tablet and emulator resolutions vary, so your Sign Up! text may appear too close to the LOG IN button.

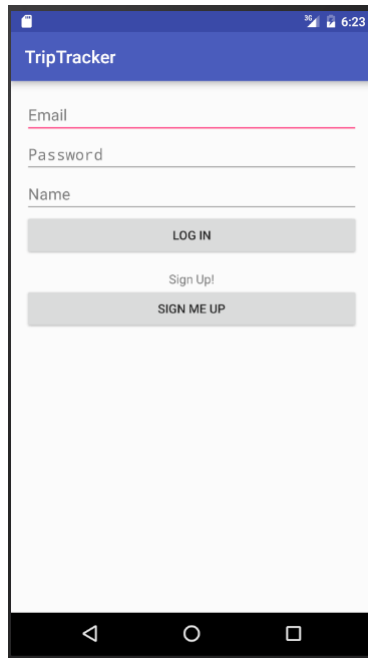
- Experiment with the spacer value in `dimens.xml` until you like the spacing.
- Type anything in both the Email and Password fields. Notice the password is masked for security (which is why `textPassword` was used as the `android:inputType`).
- Touch or click the **LOG IN** button. What happens?
- Touch or click the **Sign up!** text. What happens?

You will add code to handle the Sign Up event in just a few steps. You will create the handler for LOG IN in the next activity.

## Part II: A Multi-Use View

When a user *logs in* to the app, all that is required is their email and password. When a user *signs up*, they will *register* as a new user in Backendless. Backendless only requires an email address and password to register the new user. But an email address is not a very user-friendly way to show friends who share the app; a user should have a name. You will collect this name when a user signs up.

- 16 In `activity_login.xml` and `strings.xml`, add an additional `EditText` for name and a Sign Me Up Button with “Sign Me Up” for the label, so your Login screen should look like the image below.



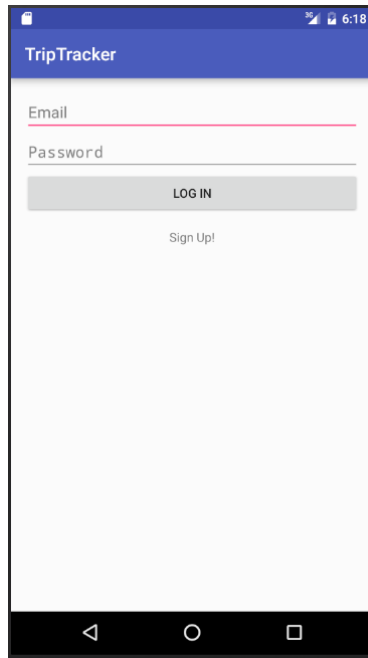
With all UI elements showing—especially the two buttons—the app is confusing, and a user may not know what to do. Hiding and showing UI elements will help eliminate this confusion and is a convenient way to use one screen for more than one function. In other apps, a separate screen is often used to register users and create accounts, gathering additional information such as addresses and phone numbers. For TripTracker, you will implement both functions on one screen.

The first screen a user sees when starting the app is the screen in login mode, one of two modes.

- In the login mode, the view will show the Email and Password `EditText`s and the LOG IN Button.
- If the user selects/touches the Sign Up! `TextView`, the screen will switch to the sign up mode. The view will hide the LOG IN button, show the three `EditText`s (Email, Password, and Name), show the Sign Me Up Button, and display some text to “Cancel Sign Up”.
- If the user clicks or touches Cancel ... the screen returns to login mode.

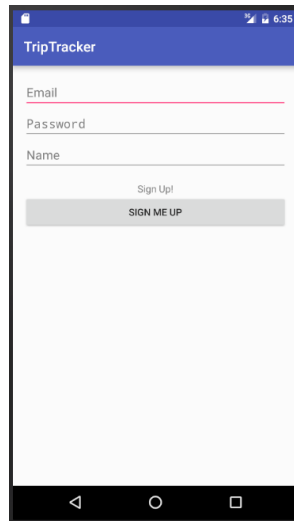
You will be guided on how to implement these hide and show features.

- 17 Use the `android:visibility` XML property with the `gone` value to hide the Name `TextEdit` and the Sign Me Up Button. Test your app to confirm they are hidden.



Now that the views for signing up are hidden, you need to show them when the user selects Sign Up!

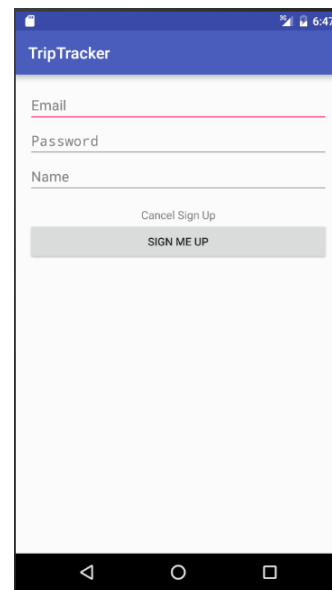
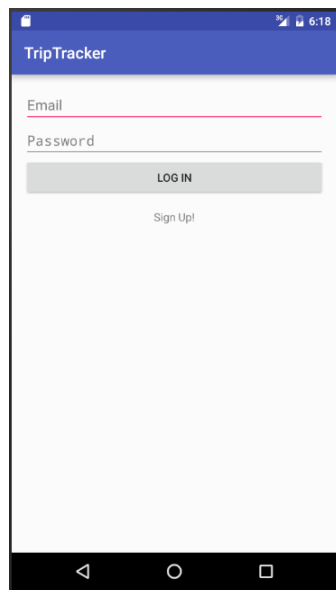
- 18 In `LoginActivity.java`
  - a. Create variable references for the `enter_name` `EditText` and the “Sign Me Up” `Button` using the `findViewById` method.
  - b. Create the variable reference for the `Login Button`. You will be hiding this.
  - c. Initialize these variable references.
- 19 Create a variable reference for the `Sign Up! TextView` and create an `onClick` listener for it. Note, later you will be creating a listener for the `Sign Me Up Button`, so it is a good idea to include “Text” or “TextView” in the name of this listener.
- 20 In the listener,
  - a. Show the hidden fields (the `Name EditText` and the `Sign Me Up Button`) with the `.setVisibility(View.VISIBLE)` method call.
  - b. Hide the `LOG IN` button.
- 21 Test your app so that when you touch or click the `Sign Up! TextView`, the `Name` view and `Sign Me Up` button show, and the `LOG IN` button is hidden.



What if the user changes his or her mind or made a mistake? You can “toggle” this hide/show feature using an `if` statement.

- 22 If the user selects “Sign Up!” the `SignUpTextView` listener is called. The listener should determine whether the app is currently hiding one of the Sign Up views. If so, the listener should show *all* Sign Up views and hide the LOG IN button. Conversely, if one of the Sign Up views is visible, the listener should hide *all* Sign Up views and show the LOG IN button. Also, when showing the views, change the Sign up! text to `Cancel Sign Up`. Remember the actual text for the label should go in `strings.xml`.

Test your app so your app toggles between these two configurations as the user touches Sign Up!/Cancel Sign Up.



## CONCLUSION

1. Explain how a toggle feature might have been implemented in an app that you have used recently.

# Lesson 3.1 Reference Card for Backendless

## RESOURCES



### Lesson 3.1 Reference Card for Backendless

Resources available online

Desired Functionality	Sample Java Code
Initialize Backendless with an application id and key	<pre>Backendless.initApp(getActivity()     .getApplicationContext(),     getString(R.string.app_id),     getString(R.string.secret_key),     "v1");</pre>
Register a user in the Backendless User table	<pre>Backendless.UserService.register(user,     new BackendlessCallback() ... );</pre>
Properties used for registration	<pre>newUser.setEmail(email); newUser.setPassword(password); newUser.setProperty("name", name);</pre>
Log a user into Backendless	<pre>Backendless.UserService.login(user,     new BackendlessCallback() {...} );</pre>
Properties used for login	<pre>newUser.setEmail(email); newUser.setPassword(password);</pre>
Get the current user	<pre>BackendlessUser user = Backendless.UserService.CurrentUser();</pre>
Save an object (performed in a new Thread)	<pre>Object o = Backendless.Persistence.save(Object);</pre>
Remove an object of type class (performed in a new Thread)	<pre>Backendless.Persistence.of(class)     .remove(object);</pre>
Create a query for finding objects	<pre>BackendlessDataQuery query =     new BackendlessDataQuery();</pre>
Find objects of type class	<pre>Backendless.Persistence.of(class)     .find(query,     new BackendlessCallback() {...} );</pre>
Apply a “where clause” to the query, such as a particular ownerId	<pre>whereClause = "ownerId='" +     user.getObjectId() + "'"; query.setWhereClause(whereClause);</pre>

Desired Functionality	Sample Java Code
Assign a query constraint or option, such as sorting by a name	<pre>QueryOptions opt = new QueryOptions(); opt.addSortByOption("name");</pre>
Callback method for a response	<pre>@Override public void handleResponse (BackendlessUser user) { ... }</pre>
Callback method when an error/fault occurs	<pre>@Override public void handleFault (BackendlessFault fault) { ... }</pre>
Log out	<pre>Backendless.UserService.logout( new BackendlessCallback() {...} );</pre>

# TripTracker

# Specifications

## Introduction

The TripTracker app allows users to keep a journal of trips that they take and share them with other users. The app requires users to sign in/register to create trips. Users can choose to make their trips publicly shared or keep them private. Shared trips are immediately visible to the app's online community. All TripTracker users who are connected to the same back-end cloud application will view all the shared posts. However, they should only be able to update and delete their own posts.

## Specifications

1. The first screen to launch in the app is the login screen, where users log in with a valid username and password to use the app. If a user has not signed up yet, then he/she can choose to view additional options to sign-up.
2. To sign up, users should create an account by entering their email, password, and name.
3. Once logged in, users should see the My Trips screen which shows their trips sorted from most recent to oldest (based on the Start Date attribute). On this screen, users have the following options:
  - a. Create a new trip
  - b. Select a trip to edit
  - c. Delete a trip from the list
  - d. Access shared trips
  - e. Log out of the app
4. While creating or updating a trip, users can enter the following data:
  - a. Trip Name
  - b. Trip Description
  - c. Start Date
  - d. End Date
  - e. Shared (a flag that indicates whether the trip is shared with other users)

5. On the Trip Details screen, users have the following options:
  - a. Post (save) the trip in the back-end cloud platform
  - b. Delete the trip
  - c. Cancel by selecting the Up button (on the action bar)
  - d. Log out of the app

## Assignment

1. In your notebook, identify and draw a sketch of all the screens that will be needed for this app.

This is a typical exercise during the requirements gathering phase of a software development process. Documenting the requirements ensures that the development team understands the business needs of their client.
2. Identify any action bar items needed for each screen.