Gadget Habit

A bunch of technobabble.

- RSS

Search

Navigate...

- Blog
- Archives

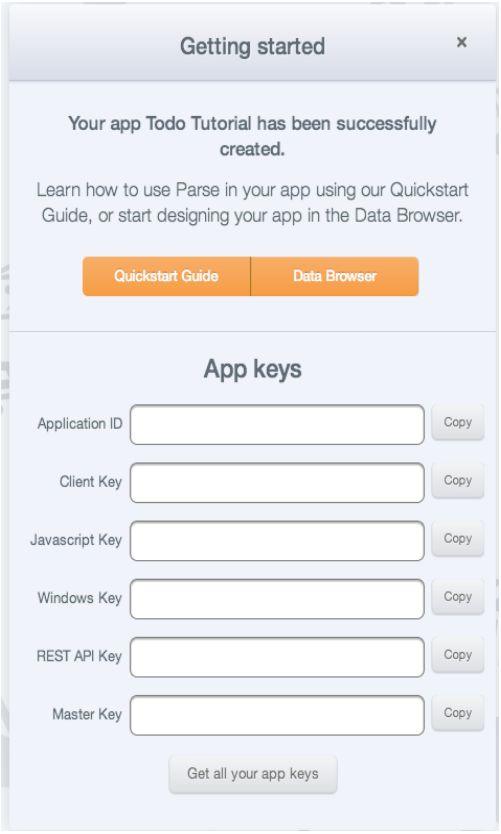
Tutorial: Building an Android To-Do List App Using Parse

Aug 14th, 2013 | Comments

For a while now I've been using Parse for small apps, it allows you to iterate quickly and create a backend for an app without too much work. It also seems like a perfect jumping off point for a series of Android tutorials, where we'll be building a "Todo List" application. (This tutorial assumes that you have Eclipse and the ADT installed already.)

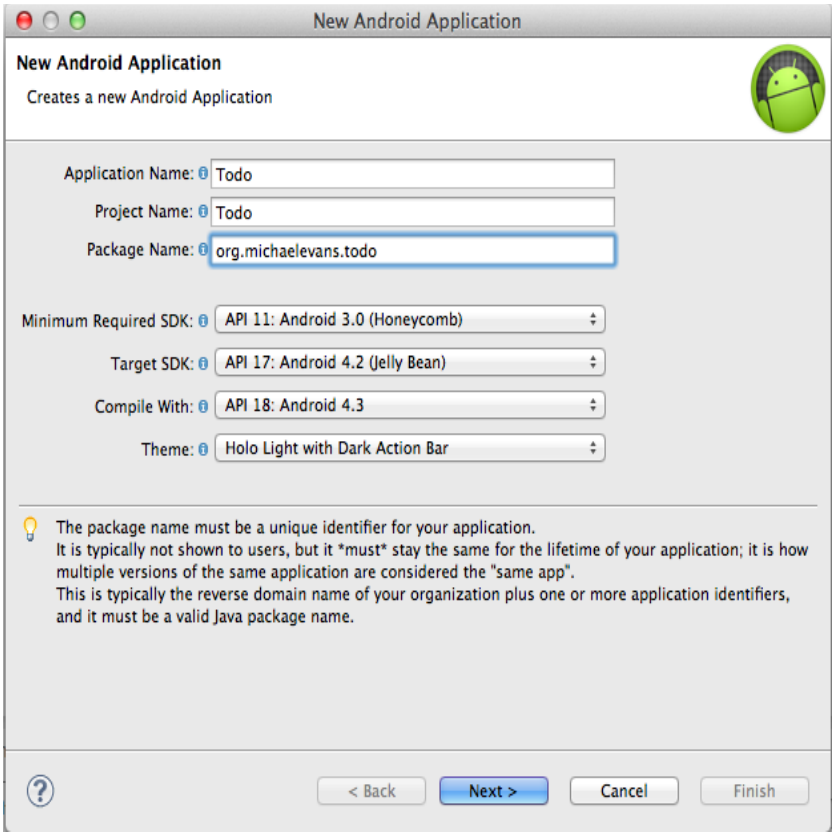
The Setup

First thing you'll want to do is sign up for Parse, and create a new Application. I called mine "Todo" here, but you can call it anything you'd like. After you pick a name, you'll be presented a screen like the following, which contains your API keys. Do not lose these, this is how your app will connect to the Parse services. (I have removed my keys in the screenshot below.)

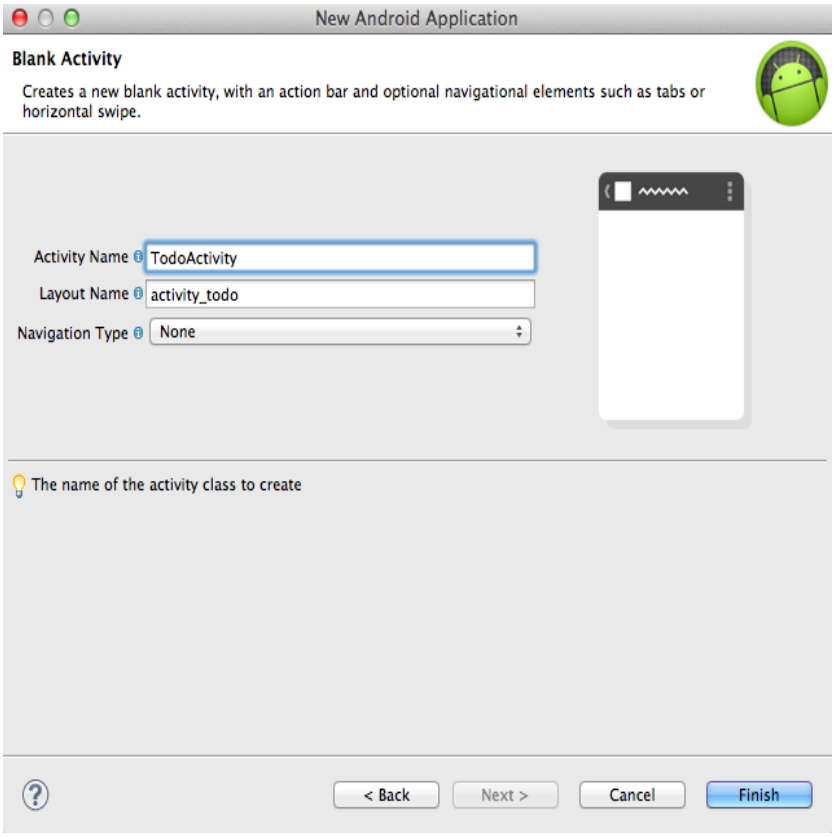


Next, you'll want to download the Parse libraries from here.

Back in Eclipse, open the New Project Wizard and create a new Android application. I set the minimum SDK version to Honeycomb in the screenshot below, but feel free to pick something newer if you'd like.



You'll then be taken to a screen to create your starting activity. Choose Blank Activity, and feel free to name it whatever you like.



After you complete the wizard, you'll have a new project in your workspace. Copy the Parse jar file that you downloaded before to the `libs/` directory, and you'll be all set to begin coding.

Let's Code

Parse Setup

First thing we'll add is the code to set up Parse. In the onCreate method of your Activity, add the following (remember to replace APP_ID and CLIENT_ID with the keys you got from Parse earlier):

```
1 Parse.initialize(this, "APP_ID", "CLIENT_ID");
2 ParseAnalytics.trackAppOpened(getIntent());
```

That second line is optional, but it adds analytics tracking to your app, which is a nice feature to get for free. In addition to that initializer code, we'll need to add get some permissions for our app. Add the following two permissions to your AndroidManifest.xml file, above the `<application>` tag:

```
1 <uses-permission android:name="android.permission.INTERNET" />
2 <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

Data Model

Now we're able to interact with Parse using the included library. The next thing we'll want to do is to declare a Task model. This object will represent an individual Task Object in Parse's datastore. Create a new class named Task.java, and fill it out like the following:

Task.java

```
1  import com.parse.ParseClassName;
2  import com.parse.ParseObject;
3
4  @ParseClassName("Task")
5  public class Task extends ParseObject{
6    public Task(){
7
8    }
9
10   public boolean isCompleted(){
11       return getBoolean("completed");
12   }
13
14   public void setCompleted(boolean complete){
15       put("completed", complete);
16   }
17
18   public String getDescription(){
19       return getString("description");
20   }
21
22   public void setDescription(String description){
23       put("description", description);
24   }
25 }
```

The annotation tells Parse what "table" that our object corresponds with, and then we are providing methods to get and set two properties: a description, and a completed status. In addition to those, Parse will give us created_at and updated_at fields for free.

Now that we've declared our model, we need to register this class with our activity. Below the initialization code, add a line like this:

```
1 ParseObject.registerSubclass(Task.class)
```

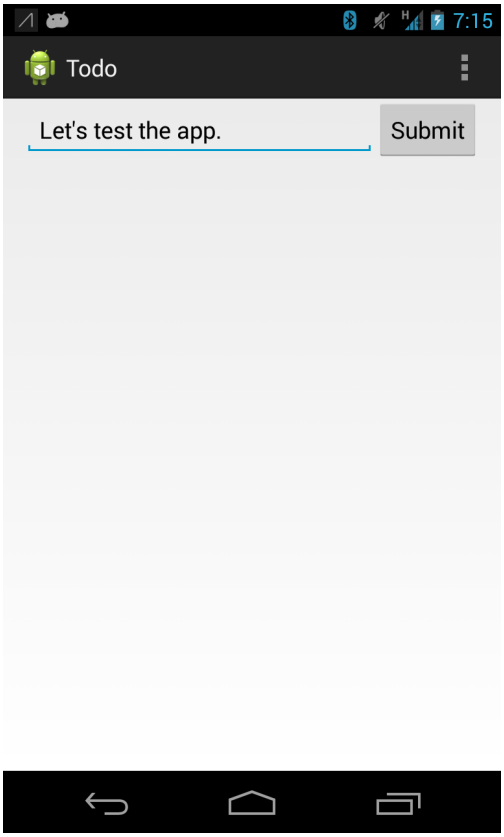
The Layout

This tells Parse to use the annotation that we declared at the top of the model. The next thing we want to do is set up the layout for our activity. This XML file will represent the UI of our application. Open up the todo_activity.xml file, and replace the contents with this:

```
1  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2      xmlns:tools="http://schemas.android.com/tools"
3      android:layout_width="match_parent"
4      android:layout_height="match_parent"
```

```
5       android:orientation="vertical"
6       android:paddingLeft="@dimen/activity_horizontal_margin"
7       android:paddingRight="@dimen/activity_horizontal_margin"
8       tools:context=".TodoActivity" >
9
10      <LinearLayout
11          android:layout_width="match_parent"
12          android:layout_height="wrap_content"
13          android:orientation="horizontal" >
14
15          <EditText
16              android:id="@+id/task_input"
17              android:layout_width="0dp"
18              android:layout_height="wrap_content"
19              android:layout_weight="1"
20              android:ems="10"
21              android:inputType="text"
22              android:hint="Enter a Task">
23              <requestFocus />
24          </EditText>
25
26          <Button
27              android:id="@+id/submit_button"
28              android:layout_width="wrap_content"
29              android:layout_height="wrap_content"
30              android:onClick="createTask"
31              android:text="Submit" />
32      </LinearLayout>
33
34      <ListView
35          android:id="@+id/task_list"
36          android:layout_width="match_parent"
37          android:layout_height="wrap_content" />
38
39  </LinearLayout>
```

Here we are declaring a layout that contains an input field and a button (for creating new tasks), as well as a ListView for showing the existing tasks. If you run what we've done so far, you should end up with an application on your device that looks like this:



You can type in the input box, but if you hit the submit button, you'll notice the app will crash. Why's that? Because we said in our XML that the button will perform the `createTask` method when you click on it, but we never declared that method in our activity. Let's do that now.

In your onCreate method, let's get a reference to the EditText and ListView, since we'll be using these later. You can do that by declaring using the `findViewById()` method. We'll save the results of these calls as private variables. For example:

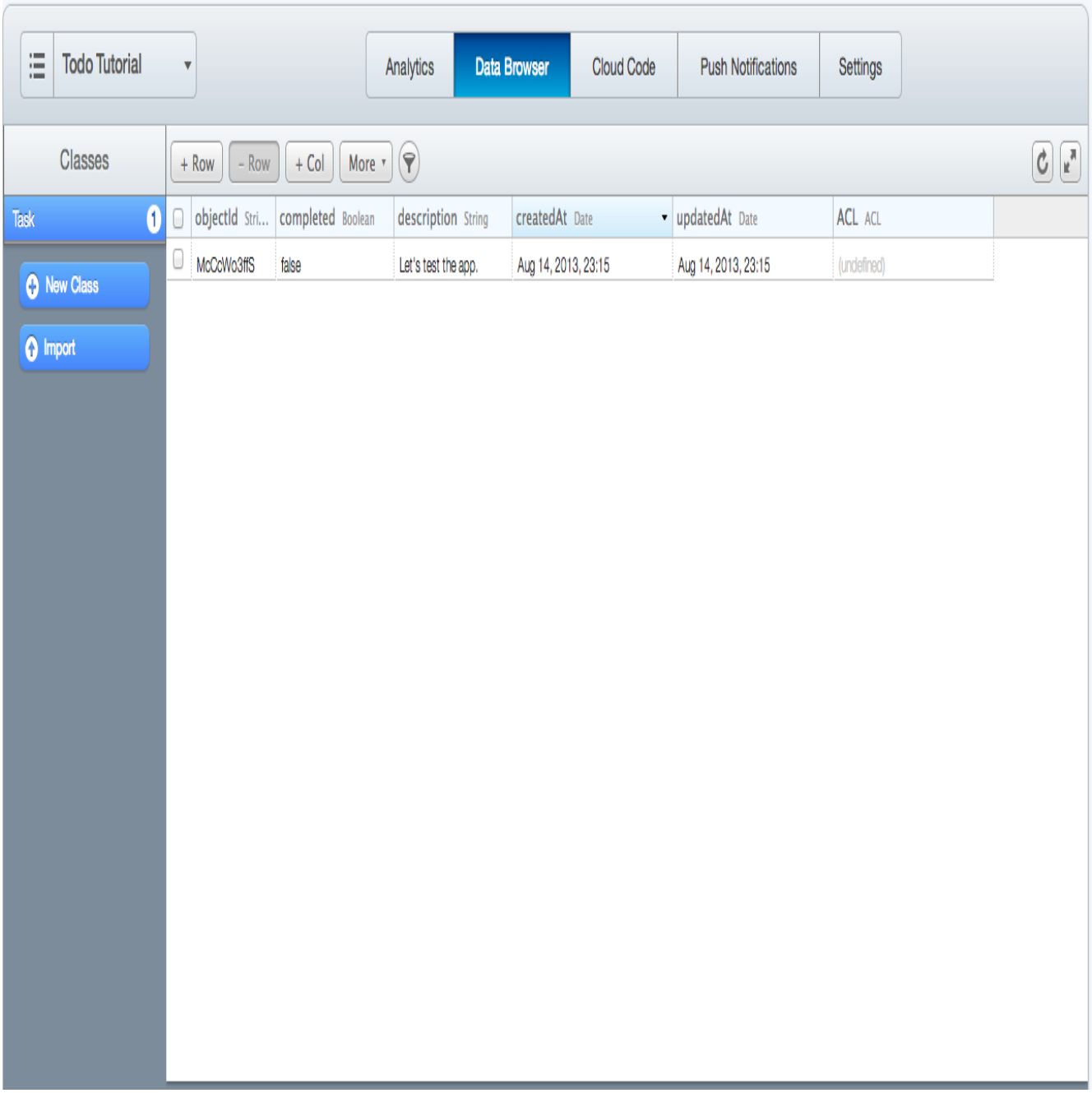
```
1 mTaskInput = (EditText) findViewById(R.id.task_input);
2 mListView = (ListView) findViewById(R.id.task_list);
```

Then we can create the createTask method:

```
1 public void createTask(View v) {
2       if (mTaskInput.getText().length() > 0){
3           Task t = new Task();
4           t.setDescription(mTaskInput.getText().toString());
5           t.setCompleted(false);
6           t.saveEventually();
7           mTaskInput.setText("");
8       }
9   }
```

What we are doing here is checking to see if the input has anything in it (don't want to create a task without a description), creating a new Task object, setting its fields, and then calling `saveEventually()`. This is a convenience method from Parse, that will queue this object to be saved. That way, if the user doesn't have a network connection, the task will be uploaded later when they are back online. Finally, we empty out the input field so that it's ready for another task.

Now if you type in a task and hit save, the input field will be blank, but if you go to your Parse Console, you'll see the data in the data browser:



Now let's set up the app to fetch the Tasks from Parse.

Querying

First let's set up a TaskAdapter. An adapter is what you add to a ListView to decide what kind of behavior the list will have (layout of each row, etc.) Create a new class called TaskAdapter that extends from ArrayAdapter like the following:

```
1  import java.util.List;
2
3  import android.content.Context;
4  import android.graphics.Paint;
5  import android.view.LayoutInflater;
6  import android.view.View;
7  import android.view.ViewGroup;
8  import android.widget.ArrayAdapter;
9  import android.widget.TextView;
10
11 public class TaskAdapter extends ArrayAdapter<Task> {
12   private Context mContext;
13   private List<Task> mTasks;
14
```

```
15   public TaskAdapter(Context context, List<Task> objects) {
16       super(context, R.layout.task_row_item, objects);
17       this.mContext = context;
18       this.mTasks = objects;
19   }
20
21   public View getView(int position, View convertView, ViewGroup parent){
22       if(convertView == null){
23           LayoutInflater mLayoutInflater = LayoutInflater.from(mContext);
24           convertView = mLayoutInflater.inflate(R.layout.task_row_item, null);
25       }
26
27       Task task = mTasks.get(position);
28
29       TextView descriptionView = (TextView) convertView.findViewById(R.id.task_description);
30
31       descriptionView.setText(task.getDescription());
32
33       if(task.isCompleted()){
34           descriptionView.setPaintFlags(descriptionView.getPaintFlags() | Paint.STRIKE_THRU_TEXT_FLAG);
35       }else{
36           descriptionView.setPaintFlags(descriptionView.getPaintFlags() & (~Paint.STRIKE_THRU_TEXT_FLAG));
37       }
38
39       return convertView;
40   }
41
42 }
```

This class will take an array of Tasks, and for each row in the ListView, set a TextView with the id `task_description` to the description value, and set a paint flag for Strikethrough if the task is completed. You'll also notice that this class inflates a layout called `task_row_item`, which we can create now.

In the `res/layouts/` folder, create a file called 'task_row_item.xml', and fill it with the following content:

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      android:layout_width="match_parent"
4      android:layout_height="wrap_content"
5      android:paddingTop="@dimen/activity_vertical_margin"
6      android:paddingBottom="@dimen/activity_vertical_margin"
7      android:orientation="horizontal" >
8
9      <TextView
10         android:id="@+id/task_description"
11         android:layout_width="0dp"
12         android:layout_height="wrap_content"
13         android:layout_gravity="center_vertical"
14         android:layout_weight="1"
15         android:textAppearance="?android:attr/textAppearanceMedium" />
16
17 </LinearLayout>
```

This is just a basic TextView which will hold the description of the item. Now that we have the adapter set up, let's create one and apply it to our ListView. Back in the onCreate method of our Activity, create an instance of our TaskAdapter, and set its initial contents to a new ArrayList of Tasks, and then set the adapter of our ListView to this adapter.

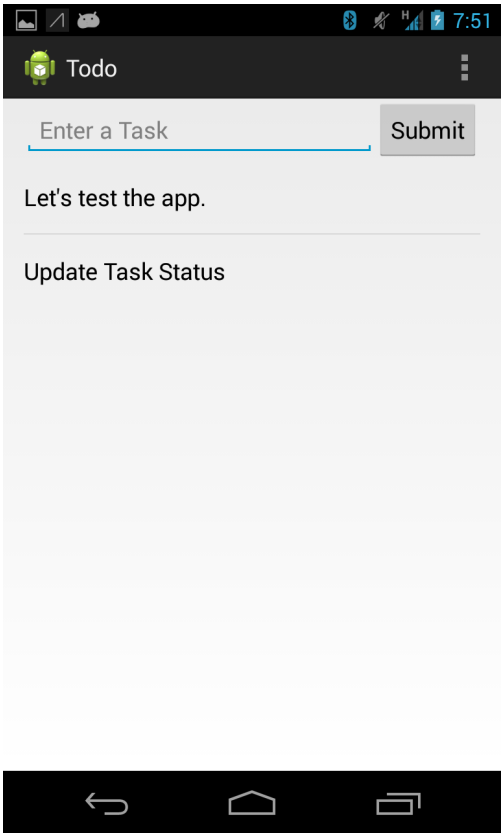
```
1 mAdapter = new TaskAdapter(this, new ArrayList<Task>());
2 mListView.setAdapter(mAdapter);
```

You'll notice that nothing happens if you run the application again, since we didn't yet fetch any data from Parse. Create a new method called `updateData()` and put in the following code: (be sure to make a call to this method at the end of `onCreate()`)

```
1  public void updateData(){
2    ParseQuery<Task> query = ParseQuery.getQuery(Task.class);
3    query.findInBackground(new FindCallback<Task>() {
4
5        @Override
6        public void done(List<Task> tasks, ParseException error) {
```

```
7            if(tasks != null){
8                mAdapter.clear();
9                mAdapter.addAll(tasks);
10           }
11       }
12   });
13 }
```

This will query Parse for all the Task objects, and return a list of them. Then we will clear the existing set of Tasks from our adapter, and replace the existing list with those. Now if you run the app, you'll be greeted with something like the following:



Let's also add this to our `createTask()` method, to insert the Task we just created at the top of the list:

```
1 mAdapter.insert(t, 0);
```

This will provide the user with an immediate hint of what happened, rather than uploading it, and waiting to resync the data or something.

Now we're starting to have a functional app! Let's add one more feature though. When you tap on a ListView row, the "completed" status of the task should toggle.

Let's Toggle!

Let's register an OnItemClickListener for our class, so that we'll get a callback whenever a list item is clicked. We can do that with a call to

```
1 mListView.setOnItemClickListener(this);
```

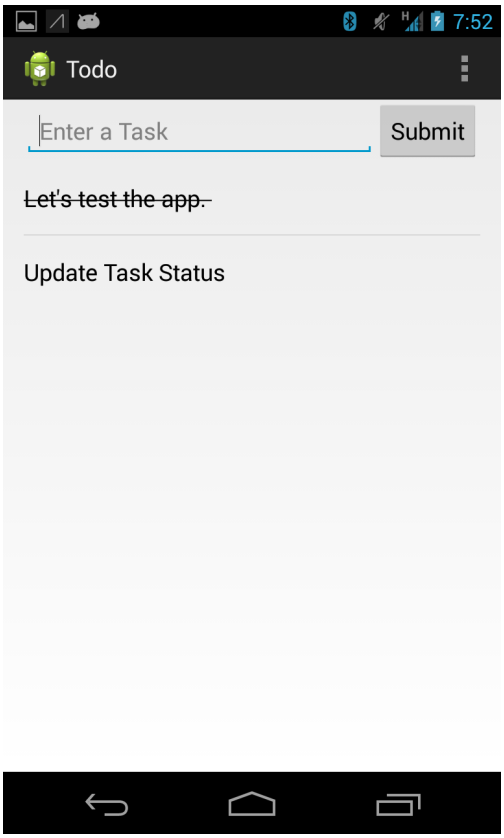
Then we just make our Activity implement the `OnItemClickListener` interface, and override the `onItemClick` method. We can use some code like this:

```
1  @Override
2  public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
3    Task task = mAdapter.getItem(position);
4    TextView taskDescription = (TextView) view.findViewById(R.id.task_description);
```

```
5
6    task.setCompleted(!task.isCompleted());
7
8    if(task.isCompleted()){
9        taskDescription.setPaintFlags(taskDescription.getPaintFlags() | Paint.STRIKE_THRU_TEXT_FLAG);
10   }else{
11       taskDescription.setPaintFlags(taskDescription.getPaintFlags() & (~Paint.STRIKE_THRU_TEXT_FLAG));
12   }
13
14   task.saveEventually();
15 }
```

This will toggle the completion status of the clicked item, and then update the strikethrough, just as we did in the ArrayAdapter before. Then we'll make another call to `saveEventually()`, to update that task on Parse's servers, when a network connection is available.

That's it! Now you should be able to launch the application, and create, complete and un-complete tasks as you wish!



Optimizations/Improvements

Let's make some quick improvements to the app. First, let's extract the hardcoded strings in our layout to a strings.xml file, so that we can support other languages. Change the text in the `android:text=` attributes to something like `@string/submit_text`, and create a `strings.xml` file in `res/values` where these strings will live. This allows you to create other values folders for quick localization.

The other optimization that we'll make is regarding the caching. You'll notice that if you leave the application and come back, you're left with a blank white screen while the Tasks are loaded from Parse. This is a bit ugly, and leaves a bad user experience. We can mitigate that by adding caching, where the results of our query will be saved locally so that we have results instantly, and then we can request the updated tasks from the network. With Parse, this is dead simple. Add the following snippet to your `query` object before you do a findInBackground:

```
1 query.setCachePolicy(CachePolicy.CACHE_THEN_NETWORK);
```

Now when you load up the app, you'll see the tasks from last time in the list, while the network is being queried.

Possibilities for Next Time

Next time we can take a look at adding multi-user support (so that not everyone shares a task list), and possibly some fancier UI, like a swipe-to-remove feature. Feel free to make other suggestions of things you'd like to see!

You can download the APK for this tutorial here, and find the source on Github here.

Like this post? Questions, concerns or mistakes? Any other Android tutorials you'd like to hear about? Let me know on Twitter or Google Plus, or leave a comment below.

Posted by Michael Evans Aug 14th, 2013 Android, Development, Parse, Tutorial

« Getting Mavericks' Compressed Memory Feature in Ubuntu Parse Android To-do App Tutorial - Part Two - Users »

Comments

Recent Posts

- Embedding Google+ Posts in Octopress
- ADB over WiFi
- 2013 in Review
- ColorArt: a library to do iTunes 11-style color matching for Android
- Hands On with Google Helpouts

GitHub Repos

- Status updating...

@MichaelEvans on GitHub

Google+