Hannah Brookhart
Florence Lee
Jackie Wolf

Final Write-Up: Business Plan

Updated Pitch

What is it about?

Our business plan application for Android is designed to facilitate conversations between microfinance workers and their clients surrounding both the creation of a business plan and how to assess their businesses. The application was created with the intent that it would be used in the field to help the microfinance workers assess their client's current income level, business operations, and help them generate a business plan with recommendations to expand their business so to raise their level of income. To facilitate conversation and accommodate different levels of education the application is intended to be passed back and forth between the microfinance worker and the client based on the information fields which is why information is displayed both as text and picture. The application is based on a business plan prototype that Jackie Wolf worked on this summer with the NGO Fundacion Paraguaya in Asuncion, Paraguay. In short this application willl help assess the business and educate individuals about changes they can make to increase their income.

Who are the user groups?

The two user groups are microfinance assessors and their clients who are low-income entrepreneurs living below the poverty line.

Why is it important?

*I*t is estimated that over 1 billion people worldwide currently live in extreme poverty. Most of the research on poverty indicates that charity does not solve issues of poverty. This application serves to assess current levels of income among those living in poverty, and more importantly, help individuals develop the skills to overcome their poverty by expanding on their current business practices.

What are the competitors?

There are no direct competitors because of the unique setting it will function in the comprehensive services it provides. There are, however, competing strategies for the individual features of our app. Competing business planning apps, which require low-income clients to have internet access, include Enloop and, StratPad. These emphasize accounting and formal reports, and are self-directed. The Small Business Coach & Plan app aggregates articles on entrepreneurship and claims to offer live business advice. Competing tools for measuring poverty and reporting field data include the Progressing out of Poverty Index, which utilizes surveys to measure clients' progress in overcoming poverty, and TaroWorks, a mobile technology suite for managing field officers and the data they collect.

Why is your idea better?

Our goal is to facilitate the relationship between microfinance officers and entrepreneurs in impoverished areas. As it exists, microfinance officers come into the homes of these entrepreneurs, ask them a series of questions, note their responses on paper that they take away and there is no sense of ownership over the information the entreprenerus provide. Our application breaks down a basic business plan into very simple and straightforward series of questions, simplifying a complicated process. We turn the process into a collaborative one, with both groups filling in various aspects of the business plan so the entrepreneur feels more ownership over the information he/she is conveying to the officer. Finally, our app helps to facilitate ongoing conversations between the two groups and provides concrete suggestions on how entrepreneurs can improve their business.

Individual Implementations

*F*lorence

Graph: You and Your Community

The graph on the You and Your Community activity is created from two user inputs: the number of people in the household and the monthly income.

The number of people in the household comes from a spinner on ClientInformation activity. The spinner is populated with an array of the number of people available as options, num_people. We use the AdapterView.OnItemSelectedListener interface which is invoked when something in this view is selected. We use the onItemSelected method and the getSelectedItemPosition method to save the position of the selected number of people into the variable folks. The value is put into SharedPreferences using the editor.

```
spinner.setOnItemSelectedListener(
        new AdapterView.OnItemSelectedListener() {
            @Override
            public void onItemSelected(AdapterView<?> arg0, View arg1,
                                       int arg2, long arg3) {
                int position = spinner.getSelectedItemPosition();

                //create a variable of type SharedPreferences:
                SharedPreferences sharedpreferences;
                String prename="mypref";

                int folks = spinner.getSelectedItemPosition();

                SharedPreferences mySharedPreferences = getSharedPreferences
                        (prename, Activity.MODE_PRIVATE);
                SharedPreferences.Editor editor = mySharedPreferences.edit();

                editor.putInt("num_people",folks);

                editor.apply();

            }
            @Override
            public void onNothingSelected(AdapterView<?> arg0) {

            }
```

ClientInformation.java

The other value we need is the income that the user inputs on the BusinessInformation activity. To save this value, we use the addTextChangedListener method which adds a TextWatcher to the EditText field. Inside the onTextChanged method, we create a String variable, textentry, where we use the getText and toString methods to grab the text, convert it to a string, and save it to the variable. We convert this value to an integer and save it as the variable, income. The value is put into SharedPreferences using the editor.

```
    // add listener to capture the income value the user inputs after the user inputs it
    editText2.addTextChangedListener(new TextWatcher() {
        @Override
        public void beforeTextChanged(CharSequence charSequence, int i, int i2, int i3) {
        }

        @Override
        public void onTextChanged(CharSequence charSequence, int i, int i2, int i3) {

            String textentry = editText2.getText().toString();
            int income = Integer.parseInt(textentry);

            // save value into sharedpreferences
            SharedPreferences mySharedPreferences = getSharedPreferences
                    (prename, Activity.MODE_PRIVATE);
            SharedPreferences.Editor editor = mySharedPreferences.edit();

            editor.putInt("income", income);
            editor.apply();
        }

        @Override
        public void afterTextChanged(Editable editable) {
        }
    });
}
```

BusinessInformation.java

Finally, we create the graph on the YouAndYourCommunity activity by using the GraphView library for Android, available at http://android-graphview.org/. We added the library into our project by adding the compile line in the build.grade file. First, we use SharedPreferences to retrieve the num_people and income variables. We use a series of if statements that sets the value of the variable poverty_threshold, depending on the value of num_people. Income is multiplied by 12 and the value is saved in the variable annual_income. Next, we create an instance of GraphViewSeries called graphValues where we use GraphViewData array as the parameter, with annual_income and poverty_threshold as the values inside the array. We instantiate the class GraphView and create a new bar graph, adding the object graphValues to the graphView object. Finally, we use the setHorizontalLabels and setManualYAxisBounds methods to label the graph and set the Y-axis.

```
// values for the graph
GraphViewSeries graphValues = new GraphViewSeries(new GraphViewData[] {
        new GraphViewData(1, annual_income)
      , new GraphViewData(3, poverty_threshold)
});

GraphView graphView = new BarGraphView(
        this // context
        , "Annual Income" // heading
);
graphView.addSeries(graphValues); // data

LinearLayout layout = (LinearLayout) findViewById(R.id.graph2);
layout.addView(graphView);

// Set bar labels
graphView.setHorizontalLabels(new String[] {"You", "Your Community"});
graphView.setManualYAxisBounds(60000,0); // Y axis bound
```

YouAndYourCommunity.java

Hannah

Generating Suggestions (Shared Preferences and Programmatically Altering Views)

Our app is dependent on saving user generated data to programmatically alter content on other activities. In this particular case, the user selects the days and times at which their business is open. The app provides feedback for the user while they are making selections, by programmatically altering the background color of the buttons. The app also generates suggestions for improvement of the client's business plan, using the methods discussed below, and displays these suggestions using a custom Array Adapter in a separate activity.

When the user selects any of the calendar buttons on the YourHours activity, an onClick method, selectHours() is called. This method receives the Button view, including element ID and background color. Variables are assigned to each button view in the activity, which are then used to programmatically alter the existing view.

Suppose the user selects the Monday morning button. The selectHours() method receives that view and then passes on additional information, like the String day ("weekday") and the String time ("morning") to the method select(). The suggestion generating functionality in this activity tallies the number of weekdays and weekends as well as mornings, afternoons, and evenings that the client's business is open. Depending on which buttons are selected, these integers are added to and subtracted from instance variables, weekdayTotal, weekendTotal, morningTotal, afternoonTotal, and eveningTotal in the selected() method. The background color is also changed to black if the users is selecting the button and back to teal if it is unselected.

```
private Boolean select(View buttonID, boolean inSelectedDayTime, String day, String time ) {
    boolean outSelectedDayTime;

    //for each button
    //if it has already been selected
    //change the background color to black
    //change the selected variable to TRUE
    //if not, revert the color to original and change the selected variable to FALSE
    //depending on whether the variable is selected, add 1 or subtract 1 from the total
    //  number of buttons in each group (weekday/weekend and morning/afternoon/evening)
    if (inSelectedDayTime == false) {
        outSelectedDayTime = true;
        if (day == "weekday") {
            weekdayTotal += 1;
        }
        else {
            weekendTotal += 1;
        }
        if (time == "morning") {
            morningTotal += 1;
        }
        else if (time == "afternoon") {
            afternoonTotal += 1;
        }
        else {
            eveningTotal += 1;
        }
        buttonID.setBackgroundColor(Color.BLACK);
    }
```

For the Monday Morning button:
Select() receives
- buttonID – mondayMorning
- inSelectedDayTime – False
- day – "weekday"
- time – "morning"

For the Monday Morning button:
Select()
- adds 1 to the weekdayTotal
- adds 1 to the morningTotal
- set mondayMorning background to black
- returns outSelectedDayTime - False

YourHours.java

In the calculateRecommendations() method, which is called when the user selects either the next or previous button to indicate they are done with this activity. If the user has selected too few times of day or days of the week, a string is added to an empty list, called suggestionsList. For example, if fewer than three "weekend" buttons are selected, the string "Try seling more during the weekend" is added to the suggestionsList.

```java
public void calculateRecommendations() {
    //get the current list of recommendations from shared preferences
    List<String> suggestionList = new ArrayList<~>(); //empty list
    //suggestionList = SharedPreferencesUtility.getStringList(this, "recommendation");

    //if the condition is met, add the suggestion to the suggestionList
    if (weekdayTotal<= 7) {
        String suggestion = "Try selling more during the week";
        suggestionList.add(suggestion);
    }
    if (weekendTotal <= 3) {
        String suggestion = "Try selling on the weekend";
        suggestionList.add(suggestion);
    }
    if (morningTotal <= 3) {
        String suggestion = "Try selling more during the morning";
        suggestionList.add(suggestion);
    }
    if (afternoonTotal <= 3) {
        String suggestion = "Try selling more during the afternoon";
        suggestionList.add(suggestion);
    }
    if (eveningTotal <= 3) {
        String suggestion = "Try selling more during the evening";
        suggestionList.add(suggestion);
    }

    bpSuggestions = suggestionList;
    Log.i("YourHours", "getting suggestionList: " + bpSuggestions);
}
```

If the user selects only the Monday Morning button:
weekdayTotal = 1
morningTotal = 1
calculateRecommendations() will add all 5 strings to the suggestionList

YourHours.java

Next, this list of suggestions is added to the Shared Preferences. The addSharedPreferences() method, which is also called when the user clicks next or back, takes items from suggestionList and joins them together into a string. Each item is delimited by a ";". (See the stringListUtility() method below, which is called inside of the addSharedPreferences() method).

A SharedPreferences object, mySharedPreferences, is created and using the SharedPreferences.Editor interface, the suggestions are saved as a key, value pair - ("suggestion", String suggestionsListString). *To account for suggestions generated in previous activities, the existing "suggestions" Shared Preferences is concatenated with the newly generated string. The result, combinedSuggestionStringList is saved to the SharedPreferences object using the same key,"suggestions".

```
public void addSharedPreferences(List suggestionList) {

    String suggestionStringList = stringListUtility(suggestionList);

    //concatenate the string of suggestions already saved in SP (saved in YourCustomers)
    //with the string of suggestions generated during this activity

    String combinedSuggestionStringList = currentSuggestionString + suggestionStringList;

    SharedPreferences mySharedPreferences = getSharedPreferences(prename, Activity.MODE_PRIVATE);
    SharedPreferences.Editor editor = mySharedPreferences.edit();

    editor.putString("suggestions", combinedSuggestionStringList);
    editor.apply();

    String debugSuggestions = mySharedPreferences.getString("suggestions", "");
    Log.i("YourHours", "SP suggestions: " + debugSuggestions);

    //check complete SharedPreferences
    Log.i("YourHours", "SP ALL: " + mySharedPreferences.getAll());

}
```

Create SharedPreferences object called mySharedPreferences

Add the suggestionsList to SharedPreferences using the key "suggestions"

```
public String stringListUtility(List list) {
    //takes in a List and returns a string with list items delimited by a ";"
    String listString = TextUtils.join(";", list);
    listString = ";" + listString;

    return listString;
}
```

//List looks like:
["Try selling more during the week", "Try selling more in the morning."]

//listString looks like:
"Try selling more during the week;Try selling more in the morning"

YourHours.java

Ultimately, these suggestions, retrieved from the SharedPreferences object with the key "suggestions" are used to create of objects of the class Suggestion, which are displayed as a list of checkboxes in SuggestionsForImprovement.java. This is done using a custom adapter, which inflates checkbox views with instances of the class Suggestion. Each instantiation of Suggestion has two attributes, a String name (which comes from the suggestions stored in SharedPreferences) and a Boolean selected (which determines whether the checkbox view is checked). The selected attribute for all Suggestion objects is false by default. It will change when the user selects that item in the SuggestionsForImprovement activity. I used a tutorial for the custom adapter and checkbox view inflater. [1]

[1] betterThanZero, "Android ListView Checkbox Example - OnItemClickListener() and OnClickListener()," *MySampleCode*, 2013 http://www.mysamplecode.com/2012/07/android-listview-checkbox-example.html

```
//This class is used in SuggestionsForImprovement.java
public class Suggestion {

    //String code = null;
    String name = null; //create a variable, name, and set it to null
    boolean selected = false; //create a variable, selected, and set it to false

    //create class Suggestion with two attributes
    public Suggestion(String name, boolean selected) {
        super();
        this.name = name;
        this.selected = selected;
    }

    //method to get the string
    public String getName() { return name; }

    //method to set the string
    public void setName(String name) { this.name = name; }

    //method to return the status of the checkbox (either selected/True or unselected/False)
    public boolean isSelected() { return selected; }

    //method to set the checkbox as either selected/True or unselected/False
    public void setSelected(boolean selected) { this.selected = selected; }

}
```

Suggestion.java

In the SuggestionsForImprovement activity, the method constructSuggestions() iterates through the list of suggestions (stored in SharedPreferences as a string delimited with ";" and split into a list once retrieved) to create a List of suggestions.

```
public List constructSuggestions(String stringList) {
    //takes a string of suggestions delimited by ";" and returns a List of suggestions
    List<String> list = new ArrayList<String>(); //default list

    if(stringList.length() != 0) {
    // string.split will create an array returning everything in between the provided "delimiter"
    // parameter
    // example: if the string is hello;world;!, calling split(";") on it would return an array
    // with 3 items: "hello", "world", and "!"
        String[] items = stringList.split(";");
// loop through the array and add it to a list so we can give it back to the method caller
        for (String i : items) {
            list.add(i);
        }
    }

    return list;
}
```

SuggestionsForImprovement.java

The displayListView() method iterates through the list of suggestions and creates a Suggestion object (with a name and boolean) for each list item.

The dataAdapter, myCustomAdapter, programmatically adds checkbox views to a list defined in suggesitons_for_improvement.xml. For each Suggestion object, in the ArrayList, suggestionList, the adapter displays the Suggestion attribute String name and the checkbox is either checked or unchecked based on the value of the Suggestion attribute Boolean selected.

```
private void displayListView(List list) {
    //create an array list with class Suggestion (String, boolean) called suggestionList
    ArrayList<Suggestion> suggestionClassList = new ArrayList<Suggestion>();
    List<String> suggestionList = list;
    //create an instance of the class Suggestion (String, boolean)

      List<String> sharedList = new ArrayList<String>();
      sharedList = SharedPreferencesUtility.getStringList(this, "recommendation");


    for (String t: suggestionList) {
        //create an instance of the class Suggestion (String, boolean)
        Suggestion suggestion = new Suggestion(t,false);

        //add the instance to the array list, suggestionList
        suggestionClassList.add(suggestion);

    }
    dataAdapter = new MyCustomAdapter(this,
            R.layout.suggestion_info, suggestionClassList);
    ListView listView = (ListView) findViewById(R.id.listView1);
    // Assign adapter to ListView
    listView.setAdapter(dataAdapter);

    listView.setOnItemClickListener((parent, view, position, id) -> {
            // When clicked, show a toast with the TextView text
            Suggestion suggestion = (Suggestion) parent.getItemAtPosition(position);

            //DEBUGGING
            //Toast.makeText(getApplicationContext(),
            //        "Clicked on Row: " + suggestion.getName(),
            //        Toast.LENGTH_LONG).show();
    });


}
```

New suggestion object is created with a String, t (the list item string) and a Boolean, false (to indicate the Suggestion is not checked)

SuggestionsForImprovement.java

MyCustomeAdapter then inflates the checkbox view laid out in suggestion_info.xml to display the suggestions that were generated based on user input in the YourHours activity.

```
private class MyCustomAdapter extends ArrayAdapter<Suggestion> {

    private ArrayList<Suggestion> suggestionList;

    public MyCustomAdapter(Context context, int textViewResourceId,
                           ArrayList<Suggestion> suggestionList) {
        super(context, textViewResourceId, suggestionList);
        this.suggestionList = new ArrayList<Suggestion>();
        this.suggestionList.addAll(suggestionList);
    }

    private class ViewHolder {
        CheckBox name;
    }
    @Override
    public View getView(int position, View convertView, ViewGroup parent) {

        ViewHolder holder = null;
        Log.v("ConvertView", String.valueOf(position));

        if (convertView == null) {
            LayoutInflater vi = (LayoutInflater)getSystemService(
                    Context.LAYOUT_INFLATER_SERVICE);
            convertView = vi.inflate(R.layout.suggestion_info, null);

            holder = new ViewHolder();
            holder.name = (CheckBox) convertView.findViewById(R.id.checkBox1);
            convertView.setTag(holder);
```

Suggestion_info.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:orientation="vertical"
android:padding="6dip" >

<CheckBox
    android:id="@+id/checkBox1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:focusable="false"
    android:focusableInTouchMode="false"
    android:text="CheckBox"
    style="@style/suggestion"/>

</RelativeLayout>
```

SuggestionsForImprovement.java

Jackie Wolf

Summary Page: SharedPreferences & TextWatcher

The creation of the business plan summary page is created from several user inputs throughout the application including information they inputted from Client Business Information and action items they have selected from Suggestions for Improvement. SharedPreferences is used to collect the information from different activities and then display it on the summary plan page.

The client name and city come from edittext fields on ClientInformation activity. Through the class TextWatcher we were able to create our own object generalTextWatcher and used addTextChangedListener method to add the generalTextWatcher to those particular EditText fields. In the onTextChanged methods that capture the information the user inputs after they input it into editText1 (name) and editText2 (city) using .setOnFocusChangeListener which grabs the input after the focus has changed from the field to convert the information inputted by the user into a string and save it as the variable (bpClientName) and add that to SharedPreferences using the editor .

```
// add listener to capture the name the user inputs after the user inputs it
TextWatcher generalTextWatcher = new TextWatcher() {
    @Override
    public void beforeTextChanged(CharSequence charSequence, int i, int i2, int i3) {
    }

    @Override
    public void onTextChanged(CharSequence charSequence, int i, int i2, int i3) {


        //grab input from editText2 after focus has changed - convert to String
        //add the city to shared preferences
        editText2.setOnFocusChangeListener(new View.OnFocusChangeListener() {

            public void onFocusChange(View v, boolean hasFocus) {
                if (!hasFocus) { //SAVE the DATA
                    final String city = editText2.getText().toString();
                    bpCity = city;
                    Log.i("Client Information", "Getting city " + bpCity);
                }
            }
        });
        //grab input from editText1 after focus has changed - convert to String
        //add the client name to shared preferences
        editText1.setOnFocusChangeListener((v, hasFocus) -> {
                if (!hasFocus) { //SAVE the DATA
                    final String name = editText1.getText().toString();
                    bpClientName = name;
                    Log.i("Client Information", "Getting name " + bpClientName);
                }
        });
```

ClientInformation.java

Then to create the summary on the ActionPlan Activity we used our SharedPreferences object mySharedPreferences to pull those variables that were saved on ClientInformation Activity for name and city using the method .getString. Using the method .setText the variable name is set as the value that is called from bpClientName our SharedPreferences variable. The information that is then displayed in the TextView name is set based on the information pulled from SharedPreferences. The same is done with the city information.

```
    //create Shared Preferences object
    SharedPreferences sharedpreferences;
    final String prename = "mypref";

    // a list class type must be used when using a list view
/ list items are added to a list view programatically and not through xml
    List<Map<String, String>> actionList = new ArrayList<~>();
    final Context context = this;
    List<Map<String, String>> summaryList=new ArrayList<~>();
    SimpleAdapter simpleAdapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_action_plan);
        TextView name= (TextView)findViewById(R.id.name);
        TextView city= (TextView)findViewById(R.id.city);
/

        SharedPreferences mySharedPreferences = getSharedPreferences(prename, Activity.MODE_PRIVATE);
        String name1 = mySharedPreferences.getString("name", " ");
        String city1 = mySharedPreferences.getString("city", " ");
        bpSave = mySharedPreferences.getBoolean("new", true);

        name.setText(name1);
        bpClientName = name1;
        city.setText(city1);
        bpCity = city1;
```

ActionPlan.java

Business Plan UML Diagram