Android как платформа для разработки корпоративных приложений на примере CRM

В. В. Осокин, В. А. Бендик

Рассматривается реализация CRM-системы для платформы Android. Система позволяет управлять клиентами, контактными лицами и контактами. В статье описан интерфейс данной системы и его реализация, а также реализация синхронизации данных с серверной частью.

Ключевые слова: Android, CRM, синхронизация.

Введение

В данной статье рассматривается реализация CRM-системы для платформы Android. Система должна позволять управлять клиентами, контактными лицами и контактами. Примерами контактов могут служить звонки, встречи, выставки, смс, письма. Кроме того, контакты могут быть как входящими, так и исходящими. Под контактными лицами понимаются сотрудники клиентов. Система должна обеспечивать доступ к описанным сущностям и возможность быстрого перехода между связанными сущностями при помощи механизма фильтров.

Наряду с описанием интерфейса в статье описывается механизм хранения данных и синхронизации данных с серверной частью.

Постановка задачи и полученные результаты

Необходимо реализовать CRM-систему, позволяющую добавлять и редактировать информацию по клиентам, контактным лицам клиентов и контактам с клиентами. Должна быть проработана навигация

между ключевыми разделами CRM. Должен быть обеспечен быстрый переход между сущностями системы через механизм фильтров. Система должна предоставлять надёжную синхронизацию данных с сервером.

В результате решения описанных задач реализована CRM-система для платформы Android с простой навигацией как между разделами, так и между списками и карточками. Разработан механизм фильтрации связанных сущностей, сохранение измененной информации на лету. Реализована синхронизация данных с сервером, что позволяет работать с системой нескольким пользователям с разных устройств.

Описание интерфейса

Интерфейс приложения разработан так, чтобы максимально облегчить навигацию между списками и карточками. Из карточки любой сущности можно перейти непосредственно к связанным с ней сущностям, например, из карточки контактного лица (рис. 1) сразу перейти к контактам с этим контактным лицом (рис. 2).

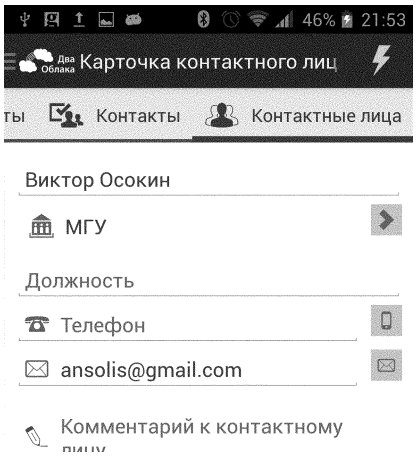

Рис. 1. Карточка.

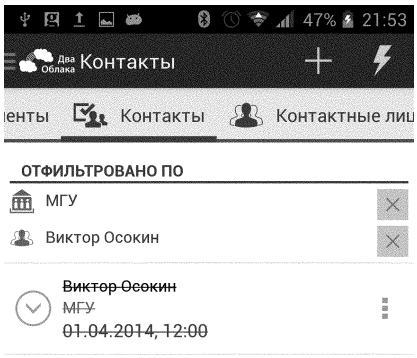

Рис. 2. Фильтры.

Навигация

Навигация по разделам приложения осуществляется при помощи табов вверху экрана. При выборе одного из табов строка с раздела-

ми центрируется так, чтобы можно было перейти к предыдущему и следующему разделу (рис. 3). Навигация внутри разделов осуществляется свайпом (swipe — проводить не отрывая, скользить) вправо или влево.

Список

Каждый раздел содержит список, который можно прокручивать вверх и вниз (рис. 4). Каждый элемент списка содержит информацию о сущности раздела (в данном случае, информацию о клиенте). При нажатии на элемент списка происходит переход к карточке.

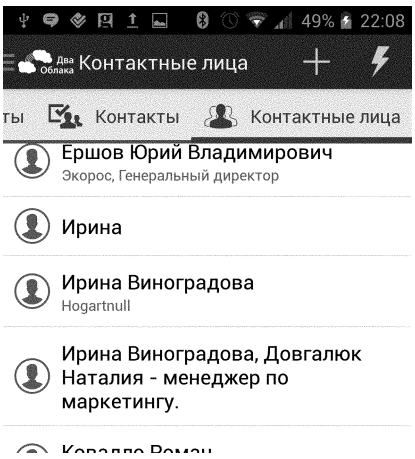

Рис. 3. Навигация.

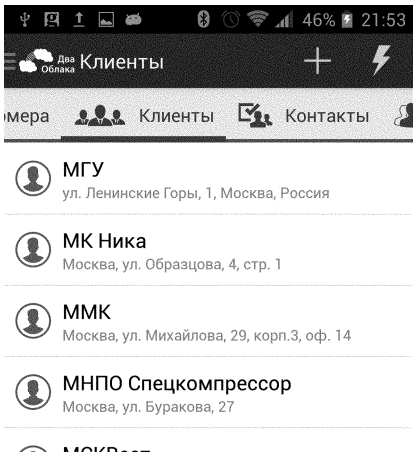

Рис. 4. Список клиентов.

Карточка

Каждый раздел содержит карточку с информацией о сущности раздела. Любое поле карточки можно редактировать на лету(рис. 5). При нажатии на кнопку «Назад» происходит возврат к списку.

Фильтры

Фильтры позволяют выбрать ту и только ту информацию, которая необходима в данный момент (рис. 2). Установить фильтр можно нажатием соответствующей кнопки в карточке, при этом приложение перейдёт к списку с уже выставленным фильтром. Удалить фильтр

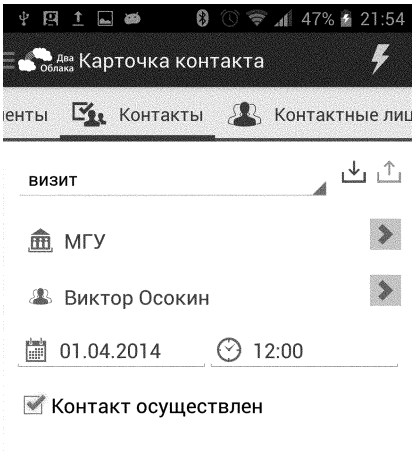

Рис. 5. Карточка контактов.

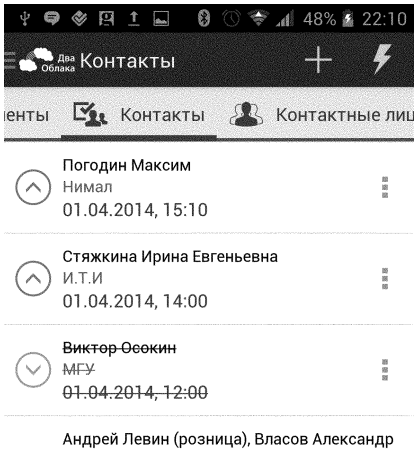

Рис. 6. Список контактов.

можно кнопкой рядом с названием фильтра или же кнопкой сброса всех фильтров в экшнбаре (англ. ActionBar) справа.

Реализация интерфейса

При реализации интерфейса задействован компонент Android ViewPager. Кроме того, использовано боковое меню (Navigation Drawer). Само приложение реализовано на одном экране (Activity) с использованием фрагментов. Загрузка данных из БД происходит асинхронно (AsyncTaskLoader), что положительно сказывается на отзывчивости интерфейса.

Навигация

Навигация между разделами реализована кнопками RadioButton, а навигация внутри раздела, между списком и карточкой, реализована компонентом ViewPager.

Для навигации по приложению используется стек, описанный в утилитарном классе по управлению параметрами приложения — Preferences. Стек содержит элементы, представляющие собой пары значений — раздел и страницу в разделе.

```
private Stack<Pair<Integer, Integer>> stack;
```

```
public Stack<Pair<Integer, Integer>> getStack() {
    return stack;
}
```

Добавление раздела и страницы в стек происходит при переходе к другому разделу, при этом в качестве идентификатора раздела используется идентификатор кнопки этого раздела, а в качестве страницы — индекс элемента в компоненте ViewPager. Кроме того, класс Preferences реализован как синглтон (англ. Singleton) для доступа к единственному его экземпляру из любой точки приложения:

```
Preferences.getInstance(getActivity()).getStack()
    .push(new Pair<Integer, Integer>(R.id.tab_clients, 0));
```

Для того, чтобы при нажатии на кнопку «Назад» не происходил выход из приложения, а осуществлялся возврат к предыдущему разделу, событие onBackPressed переопределено. В обработчике события стек проверяется на пустоту и, если там есть элементы, верхний элемент выталкивается и происходит переход к разделу, а потом к странице в данном разделе. Если же стек пуст, происходит выход из приложения.

```
@Override
public void onBackPressed()  {
    if (Preferences.getInstance(this).getStack().size()>0) {
        Pair<Integer, Integer> p = Preferences.getInstance(this).
            getStack().pop();
        onTabClick(p.first);
        pager.setCurrentItem(p.second, true);
        return;
    }
    super.onBackPressed();
}
```

Список

Списки реализуются простым классом, расширяющим BaseAdapter с ViewHolder'ом для ускорения его работы. Загрузка данных в список осуществляется через AsyncTaskLoader в котором, при необходимости, данные фильтруются перед тем, как фрагмент со списком их получит. Так как после того, как фрагменты в компоненте ViewPager уже созданы, метод onResume больше не вызывается. Для обновления данных в списке (например, если один из элементов был удалён в

карточке) переопределён метод фрагмента setUserVisibleHint. В обработчике этого события устанавливается заголовок экшнбара, а также перегружается список в том случае, если что-то было изменено в карточке:

```
@Override
public void setUserVisibleHint(boolean isVisibleToUser) {
        super.setUserVisibleHint(isVisibleToUser);
        if (isVisibleToUser == true) {
                if (getActivity()!=null) {
                        getActivity().setTitle("Контакты");
                        if (Preferences.getInstance(getActivity()).
                            getDataChanged("contact")) {
                                reload();
                        }
                }
        }
}
```

Карточка

Карточка реализована в виде редактируемых полей, которые позволяют изменять данные на лету. Обновление данных в карточке происходит аналогично тому, как обновляются данные в списке.

Фильтры

Фильтры реализуются с использованием SharedPreferences. В утилитарном классе Preferences содержатся методы для сохранения, удаления и получения фильтров. Это позволяет из любой точки приложения управлять фильтрами даже при возвращении в приложение после выхода. Кроме того, это работает быстрее, чем если бы фильтры хранились в БД.

```
public void saveFilter(String name, String value) {
        Editor editor = sharedPreferences.edit();
        editor.putString(name + FILTER_KEY, value);
        applySharedPreferences(editor);
}

public void resetFilter(String name) {
        Editor editor = sharedPreferences.edit();
        editor.remove(name + FILTER_KEY);
```

```
        applySharedPreferences(editor);
}

public String getFilter(String name) {
        return sharedPreferences.getString(name + FILTER_KEY, null
            );
}
```

В классах, расширяющих AsyncTaskLoader, помимо загрузки данных из БД реализована и их фильтрация. Если в базе данных имеются записи, то выполняется проход по ним, в ходе которого проверяется наличие определённых фильтров:

```
ArrayList<Contact> filtered_contacts = null;
if (contacts!=null && contacts.size()>0) {
        filtered_contacts = new ArrayList<Contact>();
        for (Contact c : contacts) {
                boolean toAdd = true;
                String clientId = Preferences.getInstance(mContext)
                        .getFilter("client");
                String personId = Preferences.getInstance(mContext)
                        .getFilter("person");
                ...
        }
}
```

Далее проверяется соответствие этим фильтрам записей из БД. Если запись не соответствует хотя бы одному фильтру, она отсекается. Если же запись соответствует всем фильтрам то она добавляется в список.

```
if (clientName!=null && clientId!=null
        && c.getClientId()!=Integer.parseInt(clientId)) toAdd=false;
if (personName!=null && personId!=null
        && c.getContactPersonId()!=Integer.parseInt(personId)) toAdd=
            false;
if (toAdd) {
        filtered_contacts.add(c);
}
```

Если установлены какие-то из фильтров, то после загрузки данных в список посредством AsyncTaskLoader эти данные отображаются вверху списка. В реализации метода onLoadFinished интерфейса LoaderCallbacks проверяется наличие определённых фильтров и в

зависимости от их наличия отображаются или скрываются вьюшки соответствующих фильтров.

```
Preferences prefs = Preferences.getInstance(Preferences._refActivity);
String clientName = prefs.getFilter("client_name");
filterClientView.setVisibility(View.GONE);
if (clientName!=null) {
        filterClientView.setVisibility(View.VISIBLE);
        filterClientText.setTextprefs.getFilter("client_name"));
        filterClientText.setSelected(true);
}
```

Синхронизация

Данные хранятся и изменяются в локальной БД. Поэтому довольно важным оказывается вопрос синхронизации этих данных с сервером. Синхронизация происходит по протоколу REST API. В качестве формата передачи данных используется JSON.

Хранение данных

Сохранение данных в карточке происходит при потере фокуса каким-либо из полей:

```
@Override
public void onFocusChange(View v, boolean hasFocus) {
    if(!hasFocus) {
        onClick(saveBtn);
    }
}
```

Для сохранения данных после потери фокуса, что происходит не только когда другое поле получает фокус, но и когда происходит переход к другому фрагменту, вызывается метод для сохранения в БД. В этом методе создаётся экземпляр сохраняемого класса (в данном случае класса Client) и заполняются его поля.

```
Client c = new Client();
c.setId(client!=null?client.getId():0);
 ...
c.setLastUpdate(new SimpleDateFormat("yyyy-MM-dd␣HH:mm:ss").
    format(new Date()));
```

Класс также помечается как не синхронизированный для дальнейшей его синхронизации, после чего вызывается метод execute унаследованного от AsyncTask класса для асинхронного сохранения класса в БД.

```
c.setSync(0);
updateClientTask = new UpdateClientTask();
updateClientTask.execute(c);
```

Синхронизация с сервером

Для синхронизации данных сервером используется несложный REST-клиент, реализованный на основе класса HttpURLConnection (который пришел на замену устаревшему классу DefaultHttpClient)

```
SimpleRestClient rc = new SimpleRestClient();
```

Данные передаются в form-urlencoded виде, о чём свидетельствует передаваемый на сервер заголовок "Content-Type".

```
HashMap<String, String> headers = new HashMap<String, String>();
headers.put("Content−Type", "application/x−www−form−urlencoded");
headers.put("Accept−Encoding", "");
```

Далее формируется запрос в зависимости от наличия данных в передаваемом на сервер классе.

```
StringBuilder params = new StringBuilder();
params.append("hash=" + prefs.getHash());
if (client.getId()!=0) {
        params.append("&id=" + client.getId());
}
if (client.getTitle()!=null) params.append("&name=" + URLEncoder.
    encode(client.getTitle(), "UTF−8"));
if (client.getAddress()!=null) params.append("&address=" +
    URLEncoder.encode(client.getAddress(), "UTF−8"));
...
if (client.getIsDeleted()!=0) params.append("&deleted=1");
```

Далее, в зависимости от наличия идентификатора записи, вызывается один из методов put или post. Это обусловлено тем, что если класс ещё не содержит идентификатор записи, он отправляется как новый, если же содержит, то как обновление данных.

```
if (client.getId()!=0) {
```

```
        response = rc.put(new URL(String.format(BASE_URL_API +
            CLIENT_URL, "update")), headers, params.toString().
            getBytes(), true);
} else {
        response = rc.post(new URL(String.format(BASE_URL_API +
            CLIENT_URL, "add")), headers, params.toString().
            getBytes(), true);
}
```

Серверная сторона возвращает изменённый или созданный класс в формате json, после чего десереализуется в экземпляр класса.

```
if (response.status == HttpURLConnection.HTTP_OK) {
        c = mapper.readValue(response.getBody(), Client.class);
}
```

Заключение

В результате проделанной работы реализована CRM-система для платформы Android. Реализована навигация как между разделами, так и между списками и карточками. Разработана система фильтров. Сохранение измененной информации происходит на лету, что позволяет пользователю не беспокоиться о сохранности введённых данных. Кроме того, реализована синхронизация данных с сервером, что позволяет работать с системой нескольким пользователям с разных устройств.

Список литературы

[1] http://developer.android.com/index.html

[2] Phillips B., Hardy B. Android Programming: The Big Nerd Ranch Guide. — The Big Nerd Ranch, Inc., 2013.