Лабораторная работа №6

СОЗДАНИЕ ВЕБ-СЕРВЕРА И ИСПОЛЬЗОВАНИЕ ЕГО МЕТОДОВ

**Цель работы**: получить навыки создания и использования веб-серверов на мобильных устройствах Android.

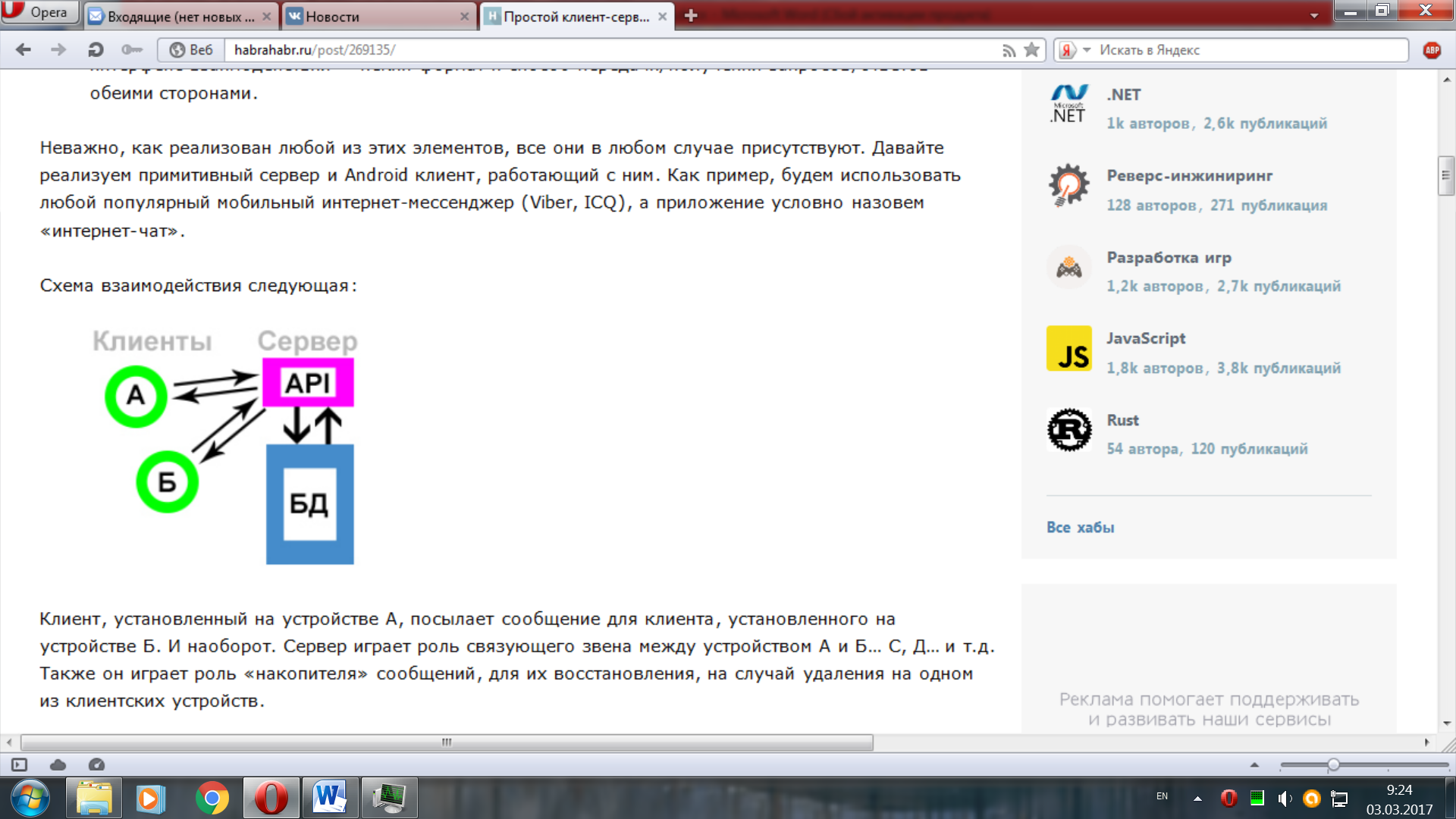
**ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

При разработке мобильных приложений часто возникает необходимость реализовать взаимодействие пользователей между собой или с базой данных, хранящейся на сервере. Сейчас существует множество готовых решений для выполнения подобных задач, например, Google Play Games Services предоставляют API для создания таблицы рекордов игроков, реализации мультиплеера, обновления приложения, внедрения системы поощрений игроков и прочих функций.

Однако иногда может возникнуть необходимость создать свой веб-сервер на Android. Рассмотрим основные теоретические моменты этой задачи и простой пример клиент-серверного приложения.

**Архитектура мобильного клиент-серверного приложения**

Многие мобильные приложения используют архитектуру клиент-сервер. Модель функционирования такой системы заключается в следующем: клиент делает запрос серверу, сервер получает запрос, выполняет его и отсылает результат клиенту. Общую схему можно представить в таком виде:



Рассмотрим следующие понятия клиент-серверной архитектуры: слой, уровень, клиент, сервер.

* Слой

Функциональность кода по всему приложению не всегда равномерная. Расслоение описывает разделение работ внутри кода приложения на одной машине. Зачастую слои – это не более чем программные модули, которые размещены в разных папках или каталогах на стороне клиента или сервера. Со стороны клиента обычно имеются от нуля до трех слоев в коде приложения. Со стороны сервера – от одного до трех слоев кода приложения. Частично это важно для хорошего проектирования программного обеспечения, которое обеспечивает повторное использование кода, частично – для безопасности, а отчасти из соображений удобства. Слой, код которого наиболее тесно взаимодействует с пользователем, часто упоминается в литературе как слой представления. Второй слой обычно обрабатывает бизнес-логику кода. Третий слой называют уровнем доступа к данным (связь с базой данных или с источником данных). Вполне возможно наличие более чем трех уровней на стороне клиента или сервера, но слишком много слоев может привести к трудностям при управлении.

* Уровни

Разбиение функциональности кода приложения на слои помогает повторному использованию, но это не делает архитектуру автоматически масштабируемой. Для того чтобы это сделать, важно распределять код на нескольких машинах. Уровень описывает разделение работ прикладного кода на нескольких машинах. Многоуровневость, как правило, предполагает размещение программных модулей на разных машинах в распределенной серверной среде. Код, который наиболее тесно взаимодействует с пользователем, часто расположен на уровне представления. Второй уровень, который содержит логику бизнес-приложений и логику доступа к данным, часто упоминается как уровень приложений. Третий ярус обычно содержит базу данных или источник данных и упоминается как уровень базы данных.

* Клиент

Как правило, мобильные устройства могут работать в качестве тонких или толстых клиентов, или могут быть разработаны таким образом, что могут быть хостами веб-страниц.

Тонкие клиенты не имеют пользовательского прикладного кода, и их функциональность полностью обеспечивается сервером. Таким образом, они не зависят от операционной системы мобильного устройства или мобильного устройства типа толстого клиента. Толстые клиенты обычно имеют от одного до трех слоев прикладного кода и могут работать независимо от сервера в течение некоторого периода времени.

Тонкие клиенты имеют ряд преимуществ перед толстыми клиентами. Например, они гораздо проще в обслуживании и поддержке, поскольку не имеют прикладного кода и данных. Однако трудностью при работе с тонкими клиентами, является то, что они должны быть в постоянной связи с сервером, т.к. это их источник для обновления и получения данных. Если связь не является надежной, возможно вместо этого типа клиента потребуется рассмотреть автономные толстые клиентские приложения.

* Сервер

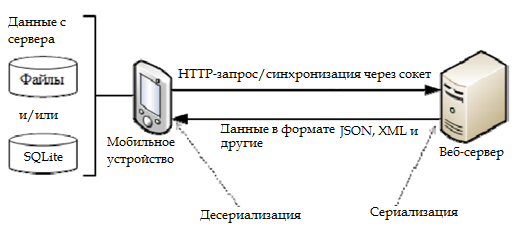
Архитектура сервера, как правило, состоит из одного-трех слоев кода, реализованного на одном-трех уровнях. Как правило, строят трехуровневую архитектуру, но у каждой есть свои плюсы и минусы.

Одноуровневая архитектура предполагает размещение всех трех слоев кода на одном сервере. Преимущества: очень удобно, быстрая разработка и развертывание. Недостатки: небольшая масштабируемость и безопасность.

При реализации двухуровневой архитектуры сервер базы данных отделяется от сервера приложений. Плюсы: удобство и возможность специализации сервера баз данных. Минусы: меньшая масштабируемость, сложности при обеспечении безопасности и дороговизна.

Трехуровневая архитектура обеспечивает разделение базы данных, сервера приложений и сервера представлений друг от друга. Преимущества: масштабируемость, защита с помощью брандмауэров и зон, возможность специализации сервера баз данных. Недостатки: избыточность, более сложно развиваться, труднее управлять, большая дороговизна.

Более детально один из вариантов взаимодействия компонентов архитектуры клиент-серверного мобильного приложения можно отобразить следующим образом:

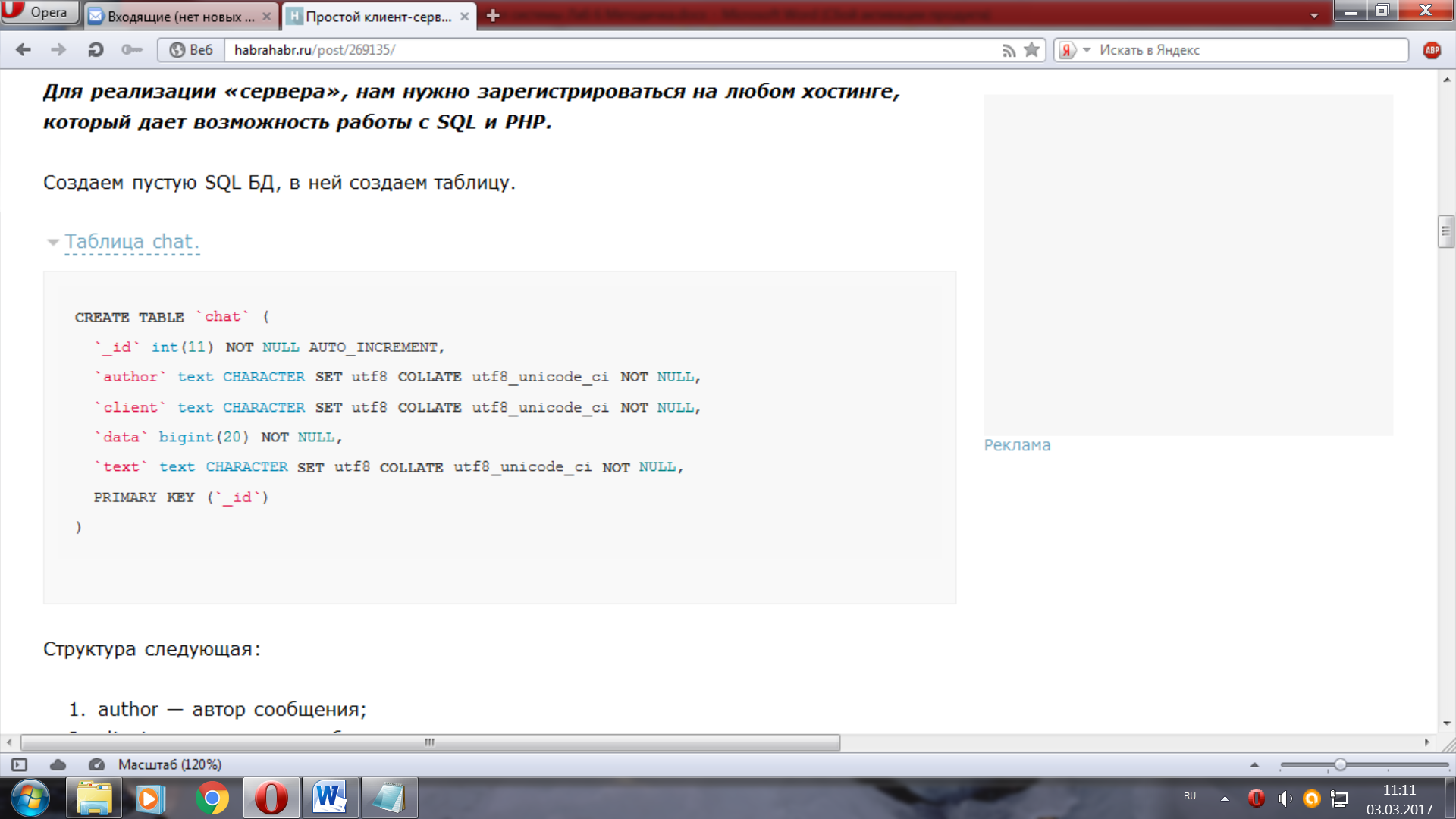


Для реализации обмена данными между клиентом и сервером, хранения информации, организации доступа к данным и других функций приложения можно использовать различные технологии, библиотеки и средства программирования на усмотрение разработчика в зависимости от поставленных задач.

Рассмотрим реализацию простого клиент-серверного приложения – интернет-мессенджера. Для хранения сообщений используем SQL БД. Работа клиентской части интернет-мессенджеров сводится к постоянной синхронизации локальной и удаленной БД с сообщениями. Такой чат должен стартовать вместе с запуском устройства и работать в фоновом режиме. Взаимодействие будет происходить путем HTTP запросов и JSON ответов (можно было бы реализовать синхронизацию через порт/сокет, это, с одной стороны, упрощает задачу (не нужно циклично слать HTTP запросы на проверку новых сообщений, достаточно проверять состояние прослушиваемого сокета), но, с другой стороны, усложняет серверную часть приложения). Клиентскую часть напишем на языке Java, серверную – на PHP.

**Реализация сервера и работы с БД**

Для реализации серверной части необходимо зарегистрироваться на любом хостинге, который дает возможность работы с SQL и PHP. Создадим базу данных следующего вида:



Структура таблицы достаточно простая:

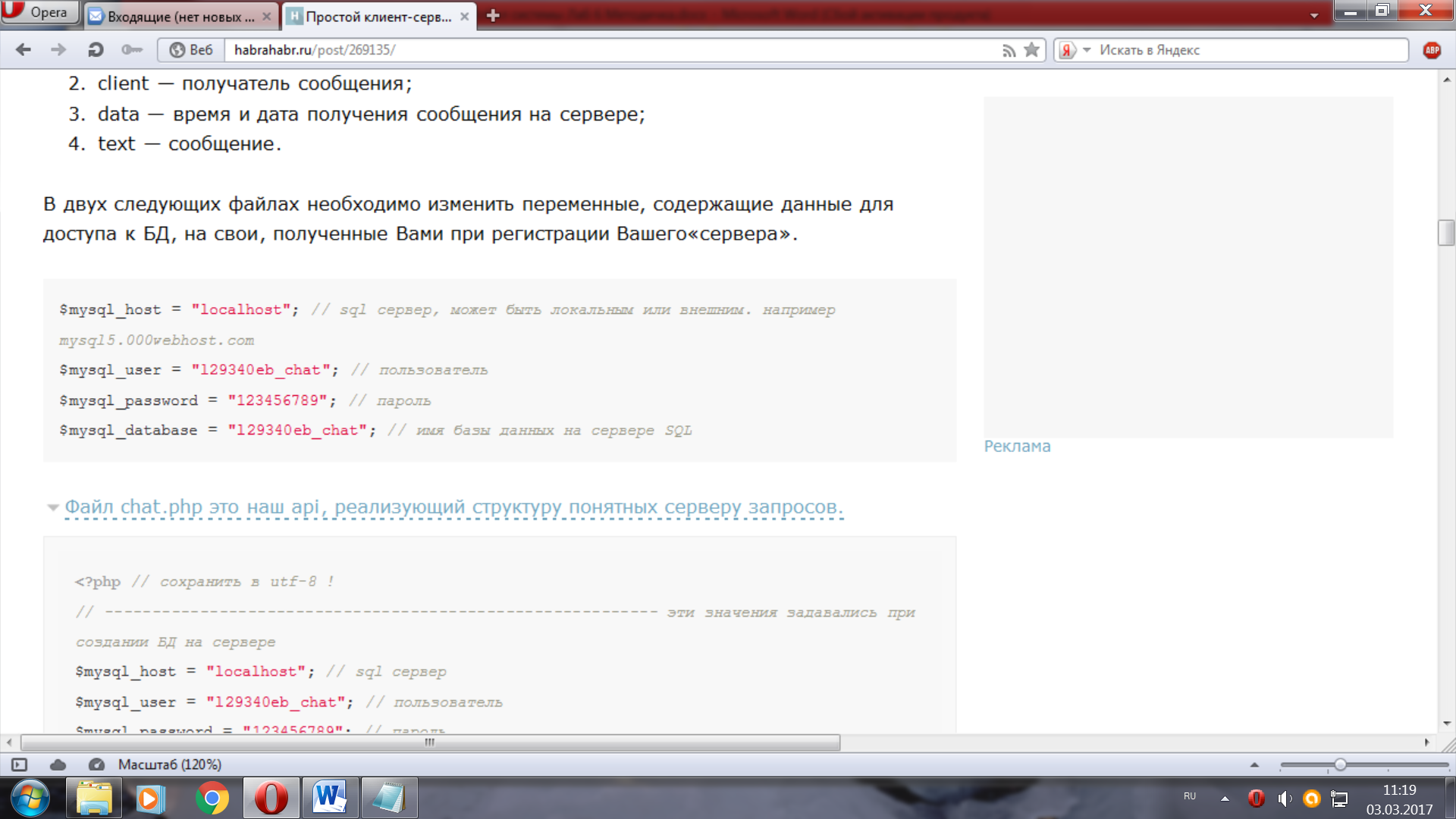
- author – автор сообщения;

- client – получатель сообщения;

- data – время и дата получения сообщения на сервере;

- text – сообщение.

Создадим файл chat.php, который будет определять API, структуру множества понятных серверу запросов. Зададим в нем параметры (содержащие данные для доступа к БД), полученные при регистрации на хостинге:



Возможная реализация серверной части (файл chat.php) приведено ниже:

<?php // сохранить в utf-8

// ----------------- эти значения задавались при создании БД на сервере

$mysql\_host = "localhost"; // sql сервер

$mysql\_user = "l29340eb\_chat"; // пользователь

$mysql\_password = "123456789"; // пароль

$mysql\_database = "l29340eb\_chat"; // имя базы данных chat

// ---------------- проверяем переданные в строке запроса параметры

// например ...chat.php?action=select

// переменная action может быть:

// select - формируем содержимое таблицы chat в JSON и отправляем назад

// insert - встваляем новую строку в таблицу chat, так же нужны 4 параметра :

//автор/получатель/время создания/сообщение

// важно: время создания мы не передаем в параметрах, берем текущее на сервере

// delete - удаляет ВСЕ записи из таблицы chat - пусть будет для быстрой очистки

// ------------------- получим переданный action

if (isset($\_GET["action"])) {

$action = $\_GET['action'];

}

// ------------------- если action=insert тогда получим еще author|client|text

if (isset($\_GET["author"])) {

$author = $\_GET['author'];

}

if (isset($\_GET["client"])) {

$client = $\_GET['client'];

}

if (isset($\_GET["text"])) {

$text = $\_GET['text'];

}

// --------------------- если action=select тогда получим еще data - от после какого времени передавать ответ

if (isset($\_GET["data"])) {

$data = $\_GET['data'];

}

mysql\_connect($mysql\_host, $mysql\_user, $mysql\_password); // коннект к серверу SQL

mysql\_select\_db($mysql\_database); // коннект к БД на сервере

mysql\_set\_charset('utf8'); // кодировка

// ---------------------------- обрабатываем запрос если он был

if($action == select){ // если действие SELECT

if($data == null){

// выберем из таблицы chat ВСЕ данные что есть и вернем их в JSON

$q=mysql\_query("SELECT \* FROM chat");

}else{

// выберем из таблицы chat ВСЕ данные ПОЗНЕЕ ОПРЕДЕЛЕННОГО ВРЕМЕНИ и вернем их в JSON

$q=mysql\_query("SELECT \* FROM chat WHERE data > $data");

}

while($e=mysql\_fetch\_assoc($q))

$output[]=$e;

print(json\_encode($output));

} // если действие INSERT и есть все что нужно

if($action == insert && $author != null && $client != null && $text != null){

// время = время сервера а не клиента !

$current\_time = round(microtime(1) \* 1000);

// пример передачи скрипту данных:

// chat.php?action=insert&author=author&client=client&text=text

// вставим строку с переданными параметрами

mysql\_query("INSERT INTO `chat`(`author`,`client`,`data`,`text`) VALUES ('$author','$client','$current\_time','$text')");

}

if($action == delete){ // если действие DELETE

// полностью обнулим таблицу записей

mysql\_query("TRUNCATE TABLE `chat`");

}

mysql\_close();

?>

Структура описанных запросов к API:

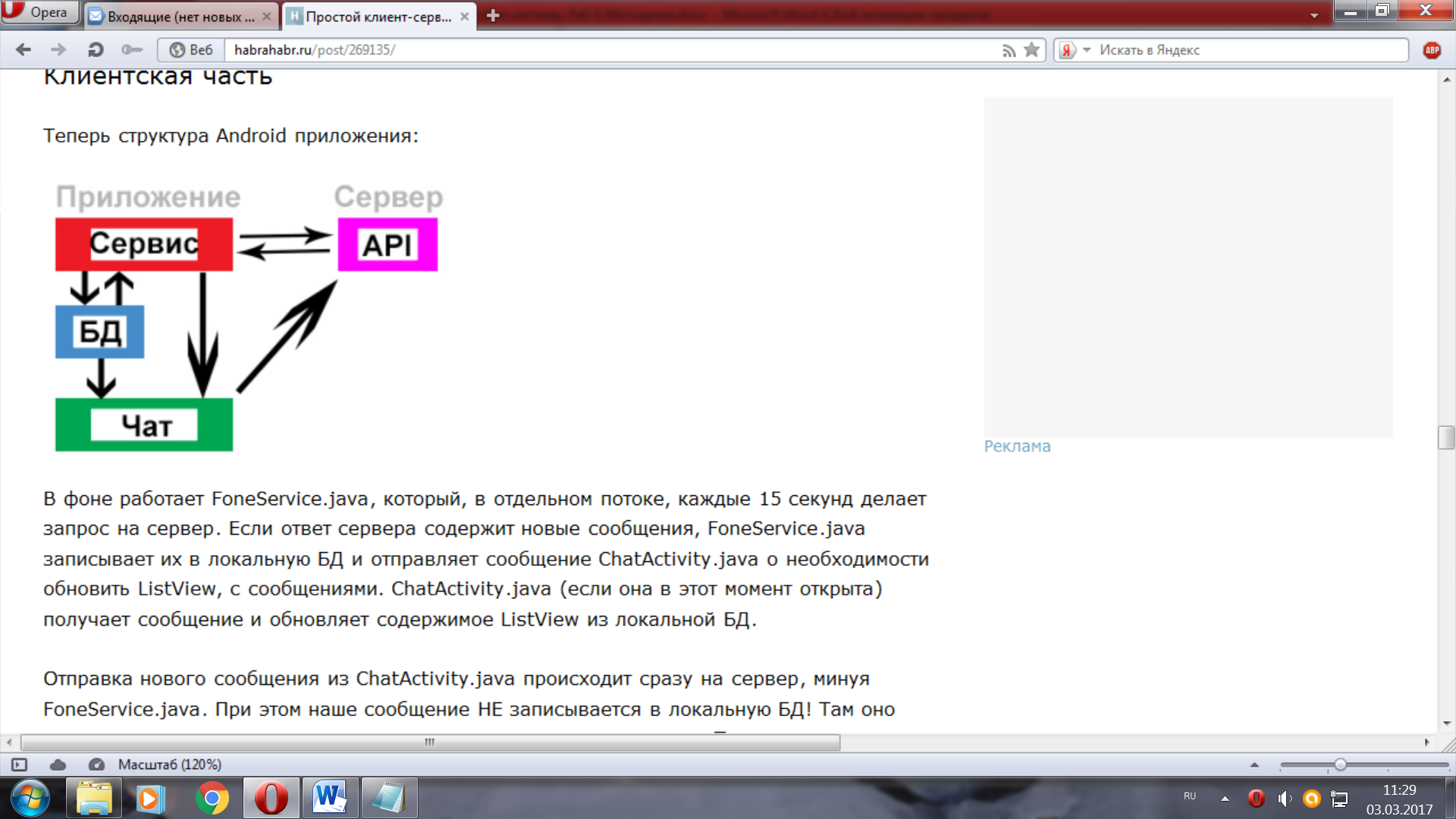
* обязательный атрибут action – может быть равен select (сервер ответит списком записей из своей БД), insert (сервер добавит новую запись в свою БД), delete (сервер очистит свою БД);
* если action=insert, нам надо будет передать дополнительные параметры: author (кто написал сообщение), client (кому адресовано сообщение), text (сообщение);
* action=select может содержать дополнительный параметр data, в этом случае ответ сервера содержит не все сообщения из БД, а только те, у которых время создания позднее переданного.

Примеры запросов:

* chat.php?action=delete – удалит все записи на сервере.
* chat.php?action=insert&author=Jon&client=Smith&text=Hello – добавит на сервере новую запись: автор Jon, получатель Smith, содержание Hello.
* chat.php?action=select&data=151351333 – вернет все записи, полученные после переданного времени в long формате.

**Клиентская часть**

Структура клиентской части (Android-приложения) и его взаимодействие с другими модулями системы показаны ниже:



Приложение на java может состоять из таких модулей:

* AutoRun.java – автозапуск приложения при включении устройства;
* FoneService.java – синхронизация с сервером в фоновом режиме;
* ChatActivity.java – обработка результатов синхронизации с сервером;
* SendActivity.java – обработка отправки сообщения на сервер.
* Файлы разметки приложения .xml, включающие компонент ListView для отображения сообщений пользователей, компоненты для редактирования и отправки сообщения и, возможно, вспомогательные элементы интерфейса.

В фоне будет работать FoneService.java, который в отдельном потоке каждые 15 секунд должен делать запрос на сервер. Если ответ сервера содержит новые сообщения, FoneService.java записывает их в локальную БД и отправляет сообщение ChatActivity.java о необходимости обновить ListView с отображаемыми пользователю сообщениями. ChatActivity.java (если она в этот момент открыта) получает сообщение и обновляет содержимое ListView из локальной БД.

Отправка нового сообщения из ChatActivity.java происходит сразу на сервер, минуя FoneService.java. При этом важно, что новое сообщение пользователя не записывается сразу в локальную БД. Там оно появится только после получения его назад в виде ответа сервера. Такую реализацию стоит использовать в связи с особенностями работы любого интернет-чата – обязательной группировкой сообщений по времени. Если не использовать группировку по времени, будет нарушена последовательность сообщений. Учитывая, что клиентские приложения просто физически не могут быть синхронизированы с точностью до миллисекунд, а возможно будут работать даже в разных часовых поясах, логичнее всего будет использовать время сервера, как мы и реализовали.

Создавая новое сообщение, мы передаем запросом на сервер: имя автора сообщения, имя получателя сообщения, текст сообщения. Получая эту запись назад в виде ответа сервера, мы получаем то, что отправляли плюс четвертый параметр: время получения сообщения сервером.

Приведем возможную реализацию для некоторых описанных модулей.

* AutoRun.java – автоматический запуск приложения (синхронизация данных в фоновом режиме) при включении устройства.

public class AutoRun extends BroadcastReceiver {

@Override

public void onReceive(Context context, Intent intent) {

if (intent.getAction().equals("android.intent.action.BOOT\_COMPLETED")) {

// получили boot\_completed - запустили FoneService

context.startService(new Intent(context, FoneService.class));

Log.i("chat", "+ AutoRun - отработал");

}

}

}

* FoneService.java – реализует запросы на сервер с целью обнаружения новых сообщений.

public class FoneService extends Service {

// ИМЯ СЕРВЕРА (url зарегистрированного вами сайта)

String server\_name = "http://l29340eb.bget.ru";

SQLiteDatabase chatDBlocal;

HttpURLConnection conn;

Cursor cursor;

Thread thr;

ContentValues new\_mess;

Long last\_time; // время последней записи в БД, отсекаем по нему, что нам тянуть с сервера, а что уже есть

@Override

public IBinder onBind(Intent intent) {

return null;

}

public void onStart(Intent intent, int startId) {

Log.i("chat", "+ FoneService - запуск сервиса");

chatDBlocal = openOrCreateDatabase("chatDBlocal.db", Context.MODE\_PRIVATE, null);

chatDBlocal.execSQL("CREATE TABLE IF NOT EXISTS chat (\_id integer primary key autoincrement, author, client, data, text)");

// создадим и покажем notification - это позволит стать сервису постоянным

// и будет визуально видно в трее

Intent iN = new Intent(getApplicationContext(), MainActivity.class);

intent.setFlags(Intent.FLAG\_ACTIVITY\_CLEAR\_TOP | Intent.FLAG\_ACTIVITY\_SINGLE\_TOP);

PendingIntent pI = PendingIntent.getActivity(getApplicationContext(), 0, iN, PendingIntent.FLAG\_CANCEL\_CURRENT);

Notification.Builder bI = new Notification.Builder( getApplicationContext());

bI.setContentIntent(pI).setSmallIcon(R.drawable.ic\_launcher) .setLargeIcon(

BitmapFactory.decodeResource(getApplicationContext().getResources(), R.drawable.ic\_launcher))

.setAutoCancel(true) .setContentTitle(getResources().getString(R.string.app\_name)) .setContentText("работаю...");

Notification notification = bI.build();

startForeground(101, notification);

startLoop();

}

// запуск потока, внутри которого будет происходить регулярное соединение с сервером для чтения новых сообщений.

// если сообщения найдены - отправим broadcast для обновления ListView в ChatActivity

private void startLoop() {

thr = new Thread(new Runnable() {

// ansver = ответ на запрос

// lnk = линк с параметрами

String ansver, lnk;

public void run() {

while (true) { // стартуем бесконечный цикл

// глянем локальную БД на наличие сообщщений чата

cursor = chatDBlocal.rawQuery( "SELECT \* FROM chat ORDER BY data", null);

// если какие-либо сообщения есть - формируем запрос

// по которому получим только новые сообщения

if (cursor.moveToLast()) {

last\_time = cursor.getLong(cursor.getColumnIndex("data"));

lnk = server\_name + "/chat.php?action=select&data=" + last\_time.toString();

// если сообщений в БД нет - формируем запрос

// по которому получим всё

} else {

lnk = server\_name + "/chat.php?action=select";

}

cursor.close();

// создаем соединение ---------------------------------->

try {

Log.i("chat","+ FoneService --------------- ОТКРОЕМ СОЕДИНЕНИЕ");

conn = (HttpURLConnection) new URL(lnk).openConnection();

conn.setReadTimeout(10000);

conn.setConnectTimeout(15000);

conn.setRequestMethod("POST");

conn.setRequestProperty("User-Agent", "Mozilla/5.0");

conn.setDoInput(true);

conn.connect();

} catch (Exception e) {

Log.i("chat", "+ FoneService ошибка: " + e.getMessage());

}

// получаем ответ ---------------------------------->

try {

InputStream is = conn.getInputStream();

BufferedReader br = new BufferedReader( new InputStreamReader(is, "UTF-8"));

StringBuilder sb = new StringBuilder();

String bfr\_st = null;

while ((bfr\_st = br.readLine()) != null) {

sb.append(bfr\_st);

}

Log.i("chat", "+ FoneService - полный ответ сервера:\n"+ sb.toString());

// сформируем ответ сервера в string обрежем в полученном ответе все,

// что находится за "]". Это необходимо, т.к. json ответ приходит с мусором

// и если этот мусор не убрать - будет невалидным

ansver = sb.toString();

ansver = ansver.substring(0, ansver.indexOf("]") + 1);

is.close(); // закроем поток

br.close(); // закроем буфер

} catch (Exception e) {

Log.i("chat", "+ FoneService ошибка: " + e.getMessage());

} finally {

conn.disconnect();

Log.i("chat","+ FoneService --------------- ЗАКРОЕМ СОЕДИНЕНИЕ");

}

// запишем ответ в БД ---------------------------------->

if (ansver != null && !ansver.trim().equals("")) {

Log.i("chat","+ FoneService ---------- ответ содержит JSON:");

try {

// ответ превратим в JSON массив

JSONArray ja = new JSONArray(ansver);

JSONObject jo;

Integer i = 0;

while (i < ja.length()) {

// разберем JSON массив построчно

jo = ja.getJSONObject(i);

Log.i("chat","=================>>> " + jo.getString("author")

+ " | "+ jo.getString("client")+ " | " + jo.getLong("data")

+ " | " + jo.getString("text"));

// создадим новое сообщение

new\_mess = new ContentValues();

new\_mess.put("author", jo.getString("author"));

new\_mess.put("client", jo.getString("client"));

new\_mess.put("data", jo.getLong("data"));

new\_mess.put("text", jo.getString("text"));

// запишем новое сообщение в БД

chatDBlocal.insert("chat", null, new\_mess);

new\_mess.clear();

i++;

// отправим броадкаст для ChatActivity

// если она открыта - она обновить ListView

sendBroadcast(new Intent("by.andreidanilevich.action

.UPDATE\_ListView"));

}

} catch (Exception e) {

// если ответ сервера не содержит валидный JSON

Log.i("chat","+ FoneService ---------- ошибка ответа сервера:\n"

+ e.getMessage());

}

} else {

// если ответ сервера пустой

Log.i("chat","+ FoneService ---------- ответ не содержит JSON!");

}

try {

Thread.sleep(15000);

} catch (Exception e) {

Log.i("chat","+ FoneService - ошибка процесса: "+ e.getMessage());

}

}

}

});

thr.setDaemon(true);

thr.start();

}

}

**Вопросы для самопроверки**

1. Что представляет собой архитектура мобильного клиент-серверного приложения? Из каких компонентов она состоит? Какими параметрами характеризуется?
2. Что такое слой и уровень в архитектуре приложения? Какие слои и уровни вам известны?
3. Какие типы клиентов бывают? В чем их отличие?
4. Какие типы клиент-серверных архитектур вы знаете? В чем их преимущества и недостатки?
5. Каким образом можно реализовать синхронизацию клиента и сервера на мобильных устройствах (с помощью каких технологий, протоколов, методов)?
6. С помощью каких технологий и средств можно реализовать серверную часть мобильного приложения? Какие основные задачи (этапы) такой реализации?
7. В чем состоит задача клиента (клиентской части) приложения интернет-мессенджера? Какие методы (классы) могут понадобиться для его реализации?

**Дополнительные материалы**

1. Устимов А. И. Архитектура клиент-серверных мобильных приложений – [Электронный ресурс] – <http://sntbul.bmstu.ru/file/out/773937> (дата обращения 04.03.2017).
2. Клиент-серверное взаимодействие под Android в деталях – [Электронный ресурс] – <https://www.slideshare.net/KZotin/android-10828914> (дата обращения 04.03.2017).
3. Архитектура мобильного клиент-серверного приложения – [Электронный ресурс] – <https://habrahabr.ru/post/246877/> (дата обращения 04.03.2017).
4. Архитектурный дизайн мобильных приложений – [Электронный ресурс] – <https://habrahabr.ru/company/redmadrobot/blog/246551/> (дата обращения 04.03.2017).
5. Android Client-Server Using Sockets – Client Implementation – [Электронный ресурс] – <http://androidsrc.net/android-client-server-using-sockets-client-implementation/> (дата обращения 04.03.2017).
6. Android Client-Server Using Sockets – Server Implementation – [Электронный ресурс] – <http://androidsrc.net/android-client-server-using-sockets-server-implementation/> (дата обращения 04.03.2017).

**Задание**

Создать Web-сервер и реализовать его метод, который реализует одну из функций лабораторной работы №1. Изменить программу из лабораторной работы №1 таким образом, чтобы она вызывала метод Web-сервера, либо WiFi.

Примеры возможных реализуемых функций:

- составление и просмотр таблицы рекордов всех игроков.

- добавление чата или форума для игроков.

- игра против другого игрока через веб-сервер или Wi-Fi.

- игра в паре с другим игроком через веб-сервер или Wi-Fi.