УДК 004:656.614.2

Олещенко Л.М., к.т.н.,
Радченко К.О.,
Ружевський М.С.,
Шроль А.Ю.

ОСОБЛИВОСТІ КЛІЄНТ-СЕРВЕРНОЇ ТЕХНОЛОГІЇ БРОНЮВАННЯ АВТОБУСНИХ МІСЦЬ У МІЖМІСЬКОМУ СПОЛУЧЕННІ

Національний технічний університет України «КПІ»

olm-86@mail.ru
radche000@mail.ru
ruzhevsky@gmail.com
Redlayn7@gmail.com

Описано особливості клієнт-серверної технології бронювання місць водієм автобуса за допомогою розробленого програмного продукту для мобільних пристроїв та планшетів на базі операційної системи Android. Додаток дозволяє водію у міжміському сполученні відправляти дані про зайняті місця у салоні під час руху автобуса на сервер АТП, має зручний інтерфейс користувача. Для клієнт-серверної взаємодії використано можливості Android Studio та Android SDK

Ключові слова: клієнт-серверна технологія, база даних, реєстрація пасажирів, транспортний засіб, *Android, REST*- запити, міжміські пасажирські перевезення, якість обслуговування пасажирів

Вступ

При дослідженні міжміських пасажирських перевезень у регіонах України було виявлено численні конфлікти між пасажирами та водіями, що було зумовлено відсутністю інформаційного забезпечення водіїв транспортних засобів (ТЗ) та диспетчерів автотранспортного підприємства (АТП) [1-4]. Згідно чинного законодавства України, на міжміському маршруті заборонене перевезення стоячих пасажирів. У регіонах України квитки на автобус можна придбати в касі, у мережі Інтернет (в автоматизованій системі «Єдиний електронний квиток») у відділеннях «Укрпошти» або «Приват Банку»; через систему «Приват-24» тощо [1]. Проте завжди існують «випадкові» пасажири, які не мають можливості придбати квиток, особливо у малонаселених селищах, розміщених між містами регіону.

Внаслідок того, що водій не реєструє таких пасажирів, на відповідні зайняті місця проводиться продаж квитків і порушується якість надання транспортних послуг.

Постановка проблеми

Для покращення якості обслуговування пасажирів в Україні АТП повинні використовувати сучасні інформаційні технології для реєстрації «випадкових» пасажирів в рухомих ТЗ у режимі *on-line*. Розроблювані рішення для водія повинні мати зручний інтерфейс для уникнення аварійних ситуацій під час передавання даних на сервер АТП.

Аналіз останніх досліджень в даній області

На даний час для міжміських автобусних пасажирських перевезень існує велика кількість ІТ, що дозволяють автоматизувати телефонію, впроваджувати IP-АТС, розроблювати сайти для *on-line* бронювання квитків, підключення *GPS* моніторингу, налаштування *GPS*– трекерів, систем відеоспостереження салону тощо [5-7]. Проте для міжміського сполучення не розроблена система реєстрації «випадкових» пасажирів, які займають

місця у салоні, їхніх пунктів посадки і висадки, що унеможливлює адекватний продаж квитків з автостанції на відповідні місця у автобусі рейсу. У попередніх дослідженнях було розглянуто бюджетну систему комунікації між водієм та диспетчером АТП з використанням додатку *APRSdroid* для УКХ-радіоканалу [2-4]. Недоліком цієї системи є незручний інтерфейс для водія при передачі даних про зайняті місця під час руху ТЗ.

Метою статті є аналіз розробленого авторами на основі операційної системи *Android* програмного продукту для водія АТП, який дозволяє реєструвати випадкових пасажирів з мінімальними витратами часу у режимі *on-line*.

Виклад основного матеріалу

Для розробки програмного продукту *BusManager* для мобільних пристроїв та планшетів було обрано операційну систему (ОС) *Android* з метою подальшого сумісного використання технології *APRS* на базі спільної ОС.

Використано програму *Android Studio* та *Android SDK* з *API* версії 16, що забезпечує роботу програмного продукту на більш як 90% всіх існуючих *Android* - пристроях. Розробка програмного продукту має відповідність із сервером, на якому зберігаються усі дані про маршрути, водіїв, ТЗ.

Взаємодія з сервером відбувається за допомогою *REST*-запитів (рис.1) [8-10]. Для створення *back-end*-шару використовується реалізація *REST* у вигляді інтерфейсу *JAX-RS* та імплементації *Jersey2.2* [11], у якості шару для доступу до бази даних (БД) використовується патерн *DAO*. З'єднання з БД відбувається через *datasource* на сервері, що індексується через *JNDI*.

Реалізовано запити, які дозволяють авторизувати та реєструвати водія в системі, отримувати інформацію про маршрути, розпочинати їх, повідомляти про посадку та висадку пасажирів.

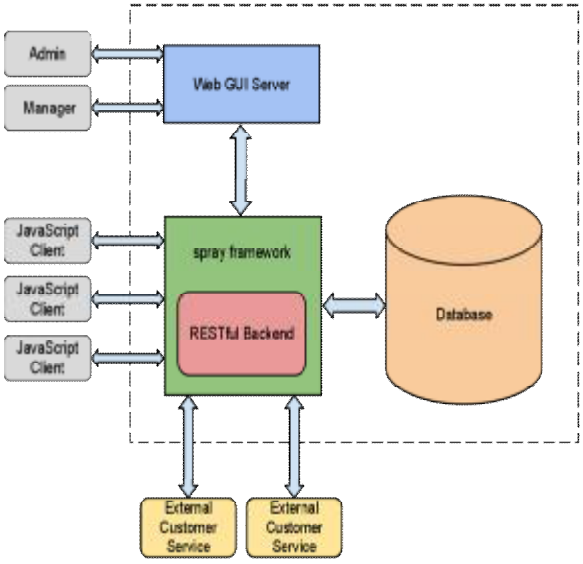


Рис.1. Схема клієнт-серверної взаємодії

Для створення *GUI* було використано стандартні методи для платформи *Android*, а саме опис інтерфейсу в файлі **.xml*. Програмний продукт *BusManager* (рис.2) має декілька основних вікон, а саме: стартове вікно, вікна авторизації та реєстрації, вікна для вибору маршруту та редагування інформації про місця у транспорті.

Стартове вікно містить кнопку авторизації та реєстрації, які відкривають вікна авторизації та реєстрації відповідно.

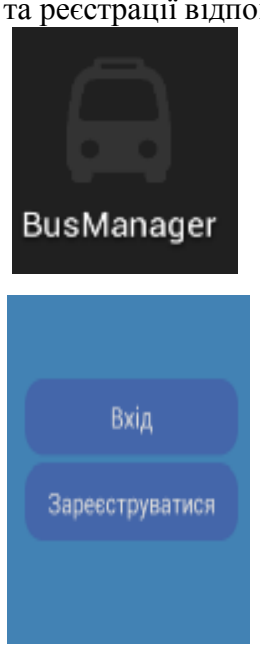


Рис. 2 – Іконка та стартове вікно додатку

На Рис.3. зображено фрагмент бази даних та дані про авторизацію в об'єктно-орієнтованій *google cloud datastore*.

Якщо сервер знаходить дані, які були введені в поля, відбувається перехід в основну частину програми.

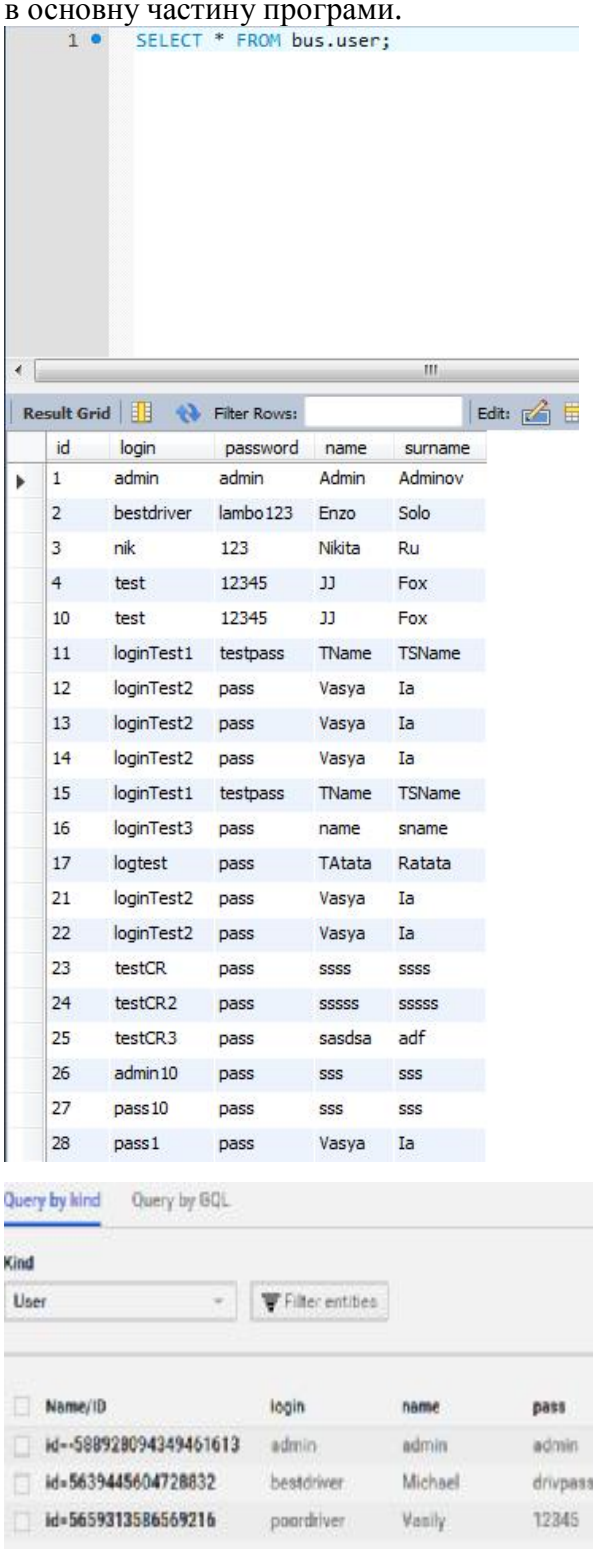
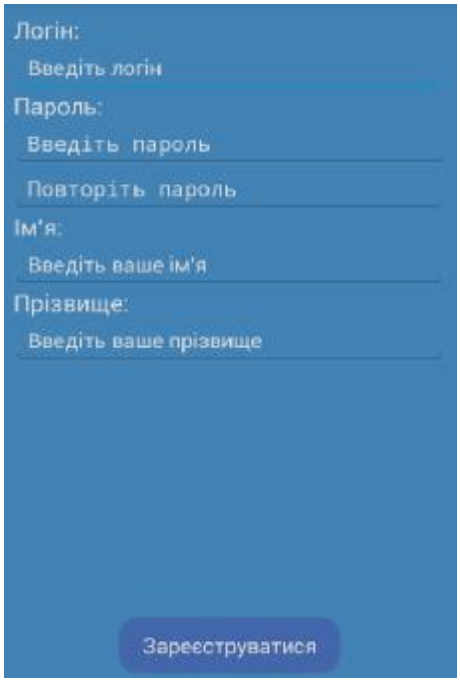


Рис.3. Фрагмент даних авторизації водія

Вікно авторизації (рис. 4) містить поля для вводу пароля та логіну, які були вказані при реєстрації водія, кнопку входу. При натисканні на кнопку входу виконуються перевірки полів логіну та паролю. Якщо в полях відсутні дані, програма повідомляє про це. В разі успішного заповнення форм, програма виконує *REST*- запит на авторизацію і очікує відповіді від серверу.



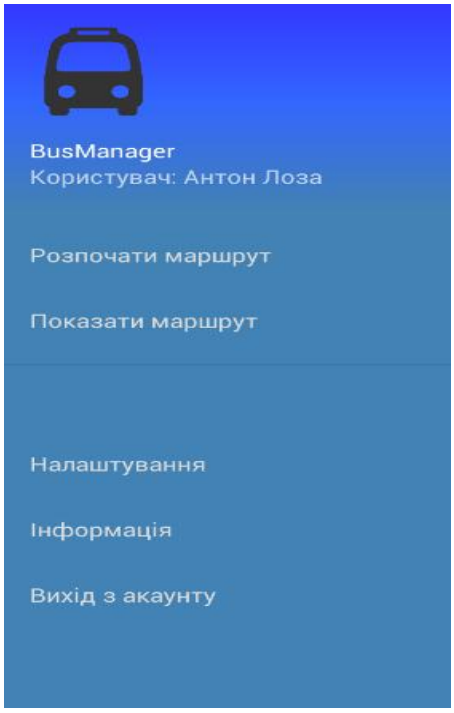


Рис. 4. Вікно авторизації

Вікно реєстрації містить поля для вводу пароля та логіну, які будуть потрібні для подальшого входу в програму, поля імені та прізвища, кнопку підтвердження реєстрації.

При натисканні на кнопку реєстрації виконуються перевірки всіх полів та перевірка збігу паролю та його підтвердження, щоб не допустити помилки в паролі.

Якщо в полях вказані різні паролі, програма повідомляє про це. В разі успішного заповнення форм, програма виконує *REST*-запит на реєстрацію і очікує відповіді від серверу.

Якщо сервер успішно виконує операцію, він повідомляє про це і відбувається перехід в основну частину програми.

Під час відкриття основної частини *BusManager* відбувається перевірка, чи існує у водія вже активний маршрут і, відповідно, відкриває вікно створення або відображення маршруту.

В основному вікні присутнє бокове меню, в якому у користувача є можливість відкрити вікна для відображення або створення маршруту, редагування акаунту, інформації про додаток та вихід з акаунту.

Вікно створення маршруту (рис.5) містить випадаючі списки для вибору пунктів відправлення, призначення, вибору маршруту, кнопку початку маршруту.

Після вибору пунктів маршруту програма посилає запит на сервер для пошуку існуючих маршрутів з такими ж параметрами початку і кінця та виводить ці дані у випадаючий список з маршрутами (рис.6).

Після вибору всіх параметрів і натиснення кнопки «Розпочати маршрут» на сервер відправлюються дані, і якщо відправлення успішне, відкривається вікно з демонстрацією маршруту.

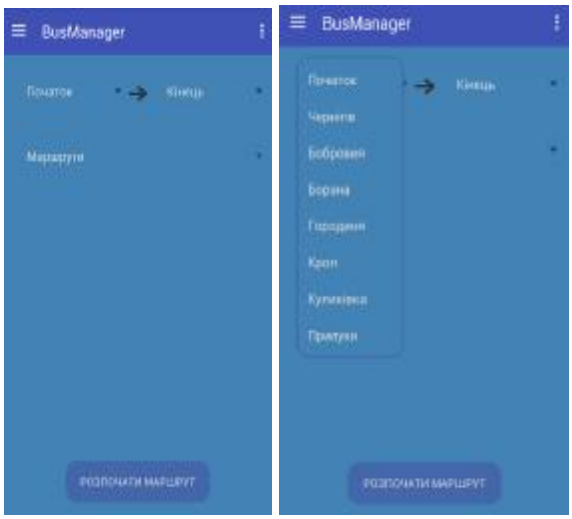


Рис. 5. Вікно вибору початку і кінця маршруту

Рис. 6. Вибір маршруту

У вікні «Розпочати маршрут» відображаються початковий та кінцевий пункт маршруту та схема салону ТЗ (рис.7). Під час відкривання даного вікна програма *BusManager* відсилає запит, результатом якого отримує інформацію про маршрут, а саме пункти маршруту та інформацію про зайняті місця.

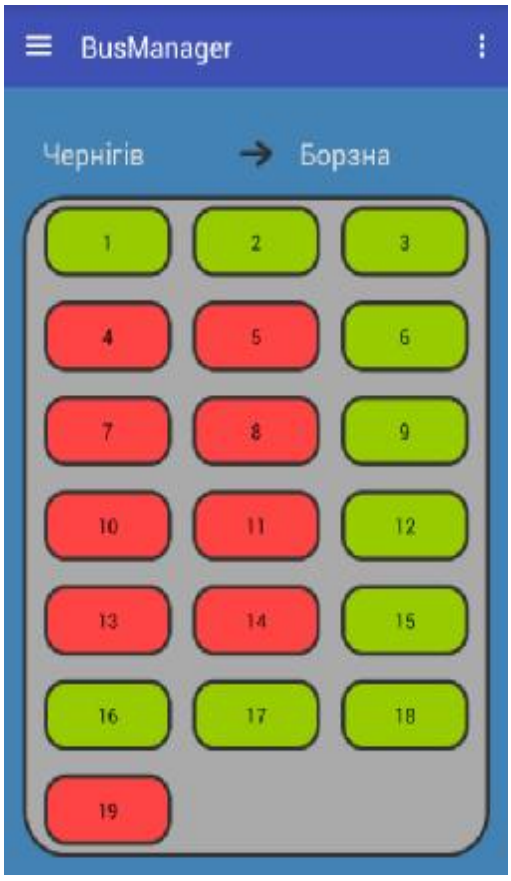


Рис. 7. Вибір рейсу та вікно салону

Під час руху ТЗ водій, підбираючи пасажира, повинен відмітити те місце, яке зайняв пасажир, для цього достатньо натиснути відповідну кнопку на екрані девайсу (рис.7). У цей момент програма відсилає запит типу *«http://bus-nikichxp.rhcloud.com/api/route/addclient?token=9b68140b-ef2a-4711-94e6-3da5dded5793&seatnumber=2»*, сервер додає в базу інформацію про те, що місце №2 зайняте, і відсилає назад відповідь про успішність процесу. В результаті кнопка із зображенням місця змінює колір з зеленого на червоний, це свідчить про те, що місце зайняте. Якщо водій висадив пасажира, він натискає на відповідну кнопку, віправляються дані на сервер, і в разу успіху кнопка стає зеленою (місце вільне).

За створення схеми відповідає клас *Generator* та адаптер *NumberedAdapter*. У першому класі проходить аналіз кількості місць у ТЗ. Ці дані передаються в *NumberedAdapter*, який і створює відповідну кількість кнопок, які відповідають кількості місць.

В *Generator* за це відповідає фрамент коду:

```
public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.activity_recycler_view,
                container, false);
        SharedPreferences tok = getActivity().getSharedPreferences("token", 0);
        String token = tok.getString("token", null);
        SharedPreferences settings = getActivity().getSharedPreferences("route", 0);
        int seats = settings.getInt("seats", 0);
        if (seats>21){
           column = 4;
        } else column = 3;
        RecyclerView recyclerView = (RecyclerView) v.findViewById(R.id.recycler_view);

recyclerView.addItemDecoration(new MarginDecoration(getActivity()));
        recyclerView.setHasFixedSize(true);
```

```
        recyclerView.setLayoutManager(new
GridLayoutManager(getActivity(), 3));
        recyclerView.setAdapter(new
NumberedAdapter(getActivity(), 19, token));
        return v;    }
```

За створення кнопок в *NumberedAdapter* відповідає фрамент коду:

```
    @Override
      public ToggleButtonHolder
onCreateViewHolder(ViewGroup parent, int
viewType) {
        View view =
LayoutInflater.from(parent.getContext()).inflate(
R.layout.bus_scheme_item, parent, false);
        return new
ToggleButtonHolder(view);    }
      @Override
      public void onBindViewHolder(final
ToggleButtonHolder holder, final int position) {
        final String label =
labels.get(position);
        holder.toggleButton.setText(label);

holder.toggleButton.setTextOn(label);

holder.toggleButton.setTextOff(label);
        if (iSeats.contains(position+1)){

holder.toggleButton.setChecked(true);}
    holder.toggleButton.setOnClickListener(n
ew View.OnClickListener() {
            @Override
            public void onClick(View v) {
              if
(holder.toggleButton.isChecked()){
Toast.makeText(holder.toggleButton.getContext(
), "activate "+(position+1),
Toast.LENGTH_SHORT).show();
      mApi.addClient(sToken, position);
}else{
Toast.makeText(holder.toggleButton.getContext(
), "deactivate "+(position+1),
Toast.LENGTH_SHORT).show();
      mApi.removeClient(sToken, position);
}      }      });    }
```

За відправленням запиту та отриманням результату від серверу відповідає бібліотека *Volley*. Оскільки відповідь від серверу приходить в текстовому форматі *JSON*, то потрібно привести його вміст до даних, які відповідають структурі програми. Для цього було використано бібліотеку *GSON*.

Приклад запиту отримання маршрутів, та його обробки:

```
    public static final String URL
="http://bus-nikichxp.rhcloud.com/api/";
      public void getRoutes(int from, int to){
        gson = new Gson();
        String URL_Request =
URL+"route/get?from="+from+"&to="+to;
        Log.d("q getRoutes", URL_Request);
jsonObjectRequest =
            new
JsonObjectRequest(Request.Method.GET,
URL_Request,                new
Response.Listener<JSONObject>() {
                @Override
                public                void
onResponse(JSONObject response) {
                  try {
                    JSONObject    js    =
response;    String s = js.getString("error");
                    Toast.makeText(ctx,
"Error: "+s, Toast.LENGTH_SHORT).show();
                  } catch (JSONException e)
{
                    if
(e.getMessage().equals("No value for error")){
                      RoutePack routePack
=          gson.fromJson(response.toString(),
RoutePack.class);

mGetRoutesListener.onGetRoutes(routePack);
                    } else
Toast.makeText(ctx, "Error: "+e.getMessage(),
Toast.LENGTH_SHORT).show();    }    }
                }, new Response.ErrorListener()
{@Override
                public void
onErrorResponse(VolleyError error) {
                  try {
                    Log.d("error",
error.getMessage());          }catch
(NullPointerException e){
                    Toast.makeText(ctx,
"Server error",
Toast.LENGTH_SHORT).show();    }
        }
      }
    );
        mQueue.add(jsonObjectRequest);
}
```

Для додавання і видалення зайнятого місця у БД використовуються методи:

```
    @GET
@Path("addclient")
    /**
     * Adding client to seat
     */
public String addClient
(@QueryParam("token")String token,
@QueryParam("seatnumber")String
seatNumber) {
try {
seatNumber.trim();
DataContainer.getRoute(token).addClient(Intege
r.parseInt(seatNumber));
return gson.toJson(new Error("Success"));
  } catch (Exception e) {
return gson.toJson(new
Error(e.toString()+seatNumber+"|"));
  } }
@GET
@Path("deleteclient")
    /**
     * When client left seat
     */
public String deleteClient
(@QueryParam("token")String token,
@QueryParam("seatnumber")String
seatNumber) {

DataContainer.getRoute(token).removeClient(Int
eger.parseInt(seatNumber));
return gson.toJson(new Error ("Success"));
```

Запит утентифікації клієнта відбувається таким чином. На шлях *API user/auth* приходить *GET*-запит аутентифікації з вказівкою логіну і пароля:

```
    @GET
@Path("auth")
public String auth(@QueryParam("login")
String login @QueryParam("pass") String
pass){UserEntity u = USERDAO.getUser(login,
pass);
if (u == null) {return gson.toJson(new
Error("No such user existing"));
  } else {return gson.toJson(u);
  }
```

Викликається метод *getUser* в класі *USERDAO*:

```
    public static UserEntitygetUser(String
login, String pass){ResultSetrs = null;
try {rs = Query.query("Select * from user where
login = \"" + login + "\" and password = \"" +
pass + "\";");
if (!rs.first()) {return null;
    }  } catch (Exception e) {return null;
  }  UUID uuid = UUID.randomUUID();
try {UserEntity u = new
UserEntity(rs.getInt("id"),
rs.getString("login"),
rs.getString("password"),
rs.getString("name"),
rs.getString("surname"),
uuid.toString()      );
DataContainer.addToken(uuid.toString(), u);
return u;
  } catch (SQLException e) {return null;
  }
```

У даному методі робиться *SQL*-запит в БД. Якщо такої комбінації логіну і пароля не існує, (наприклад, при введенні пароля була здійснена помилка), то виникне помилка і транзакція не буде завершена, об'єкт буде повернений в *null* (порожній об'єкт). Перевірка результату на null поверне клієнтському додатку відповідь «користувача не існує», або дані про користувача. Важливим є зберігання аутентифікаційного ключа. Під час авторизації користувача у спеціальному класі оперативної пам'яті зберігається спеціальний *UUID* (унікальний ключ). Особливість *UUID* полягає у тому, що загальна кількість унікальних ключів *UUID* складає 2^{128}. Це означає, що генеруючи 1 трильйон ключів кожну наносекунду, перебрати всі можливі значення вдасться лише за 10 мільярдів років. Формат зберігання *UUID* в системі максимально оптимальний в плані витрат ресурсів, він дозволяє реалізувати захист від махінацій. Всі запити відносно управління маршрутами виконуються виключно за даномим ключем. Якщо ключ буде скомпрометований, то проблему можна буде вирішити звичайним перезапуском серверу. Приклад реалізації:

```
    @GET
@Path("get")
public String getRoute
(@QueryParam("token")String token) {if
(!DataContainer.checkToken(token))
    {return gson.toJson(new Error ("Auth
failed"));
  }RouteEntityent = dc.getRoute(token);
if (ent == null) {return gson.toJson(new Error
("No such route with this token"));
  }return gson.toJson(ent);
```

При неспівпаданні ключів користувачеві буде видано: помилка аутентифікації, якщо такого ключа не існує; помилка пошуку шляху, якщо ключ вірний, але користувач ще не почав маршрут; інформація про маршрут, якщо до токену вже прикріплений маршрут.

Висновки

Розроблений програмний продукт для пристроїв з ОС *Android* має зручний інтерфейс для водія, аутентифікацію та авторизацію користувачів системи з метою уникнення несанкціонованого доступу до даного додатку. Проте програмний продукт *BusManager* передбачає можливість його використання лише у мережі стільникового покриття. Це може зумовлювати незручності при передаванні даних на сервер водієм під час руху ТЗ місцевістю, яка не має стільниковового покриття, оскільки необхідне додаткове переключення на УКХ-радіоканал. Метою подальших досліджень є створення сумісної автоматизованої системи передавання даних з на основі протоколів передачі пакетів *TCP* та *APRS*.

Список літератури

1.Олещенко Л.М., Мошенський А.О. Комп'ютерні мережі комунікації учасників пасажирсько-транспортного процесу // Наукові записки УНДІЗ. – 2014. – №1(29). – С. 82-86.

2.Олещенко Л.М., Мошенський А.О. Експериментальне дослідження зони покриття УКХ радіоканалу для зв'язку диспетчера автотранспортного підприємства з водіями рухомого складу // Наукові записки УНДІЗ. – 2014. – №3(31). – С. 47-52.

3.Олещенко Л.М. Використання безпроводових автоматизованих технологій передавання даних при побудові ІТ для АТП // Телекомунікаційні та інформаційні технології. – 2014. – №2. – С. 67-72.

4. Олещенко Л.М. // Про можливість впровадження технології *APRS* на міжміських автобусних маршрутах регіону // Наукові записки УНДІЗ. Науково-виробничий збірник. – №1(35). – 2015. – С.93-96.

5. Кравченко Е.А., Голоскоков М.А. Информационные технологии транспортной логистики в междугородных пассажирских перевозках // Фундаментальные исследования. – 2007. – № 12. – С. 509-510.

6. Бочков А.А., Екшикеев Т.К., Филенко С.А. Новые информационные технологии на автомобильном транспорте // Технико-технологические проблемы сервиса. – 2009. – №9. – С.74-76.

7. Власов В. М. Информационные технологии на автомобильном транспорте / В. М. Власов [и др.]; под общ. ред. В. М. Приходько. – М.: Наука, 2006. – 283 с.

8. Машнин Т. С. Технология Web-сервисов платформы *Java*. – БХВ-Петербург, 2012. – 560 с.

9. *Thomas Erl, Benjamin Carlyle, Cesare Pautasso, Raj Balasubramanian. SOA with REST: Principles, Patterns & Constraints for Building Enterprise Solutions with REST. – Prentice Hall,* 2013. – 624 *p.*

10. *Create RESTful Web services with Java technology* // [Електронний ресурс] // – Режим доступу: *http://www.ibm.com/developerworks/webservices/library/wa-jaxrs/*

11. *Jersey 2.2 User Guide* // [Електронний ресурс] // – Режим доступу: *https://jersey.java.net*/

Статтю подано до редакції 13.12.2015