

Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Кафедра «Компьютерные системы и программные технологии»

КУРСОВАЯ РАБОТА

Android-приложение для клиентов сети баров

по дисциплине «Базы данных»

Выполнил
студент гр. 3530901/70203

(подпись)

И.Д. Иванов
(инициалы, фамилия)

Руководитель

(подпись)

А.В. Мяснов
(инициалы, фамилия)

«___» _____ 2020 г.

Санкт-Петербург
2020

**ЗАДАНИЕ
НА ВЫПОЛНЕНИЕ КУРСОВОЙ РАБОТЫ**

студенту группы 3530901/70203 Иванову Илье Дмитриевичу
(номер группы) (фамилия, имя, отчество)

- 1. Срок сдачи законченной работы** 13.06.2020
- 2. Исходные данные к работе:** Задание для курсовой работы
- 3. Содержание пояснительной записки** (перечень подлежащих разработке вопросов): техническое задание, реализация, вывод.

Дата получения задания: «10» мая 2020 г.

Руководитель _____ А.В. Мяснов
(подпись) (инициалы, фамилия)

Задание принял к исполнению _____ И.Д. Иванов
(подпись студента) (инициалы, фамилия)

(дата)

Содержание

1. Техническое задание	4
1.1. Постановка задачи	4
1.2. Возможности приложения	4
1.3. План разработки	4
2. Ход работы	5
2.1. Структура базы данных	5
2.2. Веб-сервис	6
2.3. Android-приложение	10
3. Выводы	20

1. Техническое задание

1.1. Постановка задачи

Разработать мобильное приложение для операционной системы Android, позволяющее клиентам сети баров просматривать меню баров, узнавать о текущих акциях, а также просматривать и редактировать информацию о себе в личном кабинете. В качестве хранилища данных использовать базу данных, созданную в течение семестра на лабораторных работах.

1.2. Возможности приложения

- Вход для зарегистрированных клиентов;
- Регистрация новых клиентов;
- Возможность смены пользователя после входа в систему;
- Просмотр меню баров;
- Просмотр текущих акций;
- Возможность изменения хранимых личных данных;
- Возможность удаления своего аккаунта из базы.

1.3. План разработки

- Принятие решения о необходимом функционале приложения;
- Выбор архитектуры;
- Разработка сервиса для взаимодействия с базой данных;
- Разработка мобильного приложения и реализация необходимой функциональности;
- Тестирование приложения;
- Выводы о проделанной работе и полученном результате.

2. Ход работы

2.1. Структура базы данных

В качестве базы данных была взята база, разработанная в ходе лабораторных работ – “bar_db”. Схема базы данных представлена на Рис.2.1.1.

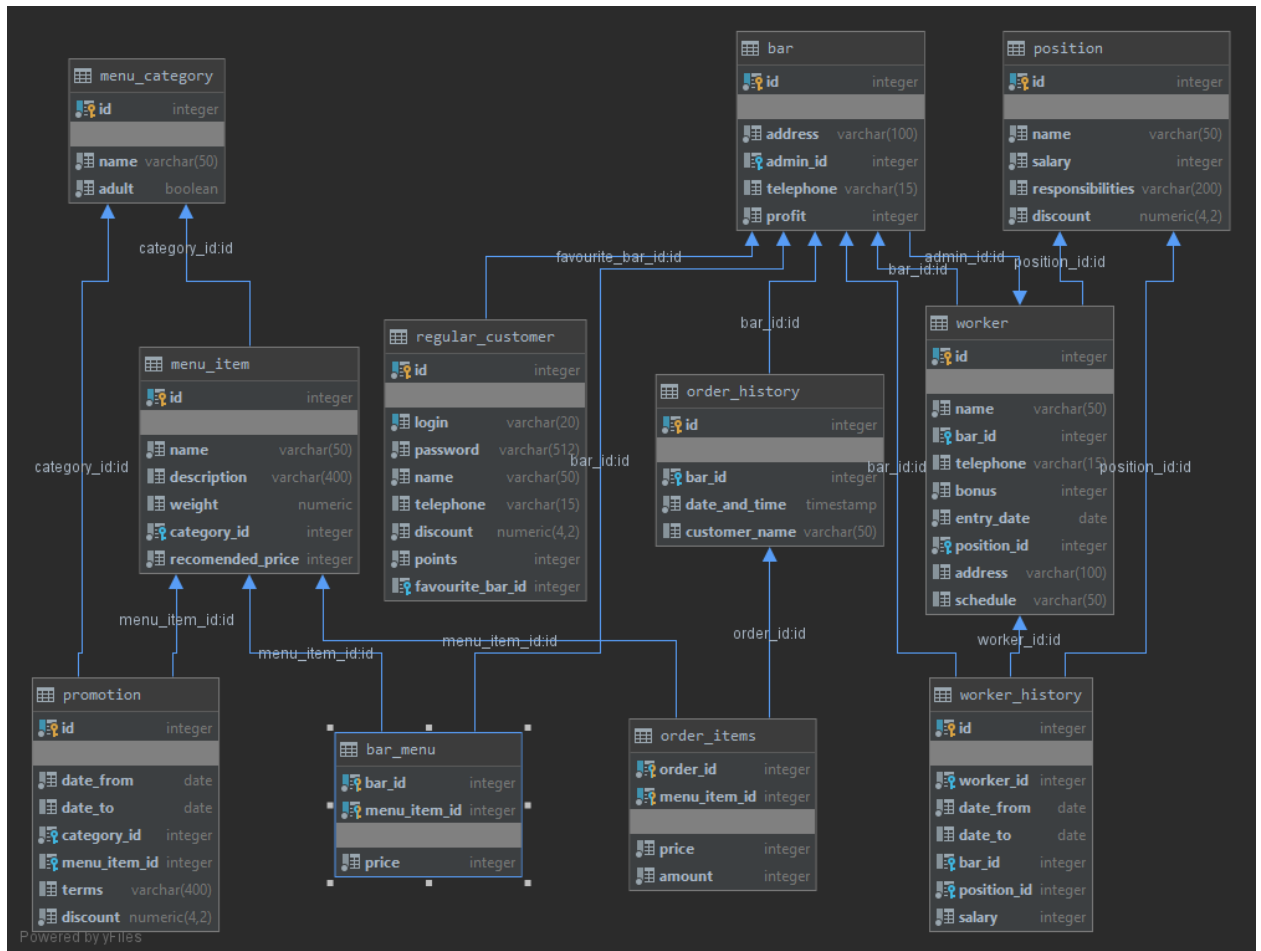


Рис.2.1.1. Схема используемой базы данных

В таблицу regular_customer были добавлены поля для хранения логина и зашифрованного пароля (применяется пятикратное хеширование алгоритмом SHA-512). Эта таблица также хранит и все остальные личные данные клиентов. Пример данных, хранимых в таблице regular_customer:

	id	login	password	name	telephone	discount
1	1 0		c785674868f9bb555c26d09e02509056f28a8bfd1b8dc296978d...	Ломоносов Василий Андреевич	79163705033	0.1
2	2 1		c785674868f9bb555c26d09e02509056f28a8bfd1b8dc296978d...	Манторова Елена Андрияновна	79137791778	0.0
3	3 2		c785674868f9bb555c26d09e02509056f28a8bfd1b8dc296978d...	Невянцев Лаврентий Родионович	79107711504	0.1
4	4 3		c785674868f9bb555c26d09e02509056f28a8bfd1b8dc296978d...	Можяев Борислав Андроникович	79293625519	0.1
5	5 4		c785674868f9bb555c26d09e02509056f28a8bfd1b8dc296978d...	Дресвянина Агния Федоровна	79138695001	0.0

Рис.2.1.2. Пример данных, хранимых в regular_customer

2.2. Веб-сервис

Для взаимодействия с базой данных был разработан веб-сервис. Приложение не подключается напрямую к базе данных, а взаимодействует с ней через API сервиса.

При разработке сервиса использовался framework Express для окружения node.js.

Входной точкой для сервиса является файл index.js. В нем, кроме прочего, определяется используемый порт. В данном случае, 3222:

Листинг 2.2.1.

```
const port = 3222
```

В файле queries.js хранится конфигурация подключения к базе данных:

Листинг 2.2.2.

```
const Pool = require('pg').Pool
const pool = new Pool({
  user: 'bar_user',
  host: '192.168.0.101',
  database: 'bar_db',
  password: 'barbar',
  port: 5432,
})
```

Для различных типов запросов (Select, Insert, Update, Delete) были созданы различные endpoint-ы:

Листинг 2.2.3.

```
app.post('/get', [
  body('table').trim().escape(),
  body('colons').trim().escape()
], db.db_get)
app.post('/insert', [
  body('table').trim().escape(),
  body('fields').trim().escape()
], db.db_insert)
app.post('/delete', [
  body('table').trim().escape(),
  body('field').trim().escape()
], db.db_delete)
app.post('/update', [
  body('table').trim().escape()
], db.db_update)

app.listen(port, () => {
  console.log(`App running on port ${port}.`)
})
```

Параметры для всех запросов передаются через тело POST-запросов. Передаваемые параметры обрабатываются с помощью trim() и escape() для борьбы с SQL-инъекциями.

В файле queries.js находятся функции, обрабатывающие запросы к различным endpoint-ам:

Листинг 2.2.4.

```
const db_get = (request, response) => {

  const table = request.body.table
  var colons = request.body.colons
  var where = request.body.where
  var parse_where=""

  if (typeof colons == 'undefined' )
  {
    parse_cols="*"
  }
  else{
    colons=colons.substring(1, colons.length - 1)
    colons=colons.split(",")
    var parse_cols = ""

    colons.forEach(elem => {
      parse_cols+=elem
      parse_cols+=","
    });
    parse_cols=parse_cols.substring(0, parse_cols.length - 1);

  }

  if (typeof where != 'undefined' )
  {
    where=where.substring(1, where.length - 1)
    where=where.split(",")
    if(where.length % 3 == 0){

      for (i=0;i<where.length;i=i+3){
        if(i!=0){
          parse_where=parse_where+ " AND "
        }
        parse_where=parse_where + where[i] + where[i+2] +
        where[i+1]
      }

    }

  }

  if ( parse_where != ""){
    pool.query(`SELECT ${parse_cols} FROM ${table} WHERE
    ${parse_where} ORDER BY id ASC`,(error, results) => {
      if (error) {
        response.status(400).json([{'details':'Bad request'}])
      }
      else{
        response.status(200).json(results.rows)
      }
    })
  }
  else{
```

```

        pool.query(`SELECT ${parse_cols} FROM ${table} ORDER BY id
        ASC`,(error, results) => {
            if (error) {
                response.status(400).json([{'details':'Bad request'}])
            }
            else{
                response.status(200).json(results.rows)
            }
        })
    }
}

```

```

const db_insert = (request, response) => {

    const table = request.body.table
    var fields = request.body.fields
    var values = request.body.values
    var parse_fields="("
    var parse_values="("

    fields=fields.substring(1, fields.length - 1)
    parse_fields=parse_fields+fields+")"

    values=values.substring(1, values.length - 1)
    parse_values=parse_values+values+")"

    pool.query(`INSERT INTO ${table} ${parse_fields} VALUES
    ${parse_values}`,(error, results) => {
        if (error) {
            response.status(400).json([{'details':'Bad request'}])
        }else{
            response.status(200).json(results.rows)
        }
    })
}

```

```

const db_update = (request, response) => {

    const table = request.body.table
    const field = request.body.field
    const value = request.body.value
    var where = request.body.where
    var parse_where=""
    if (typeof where !== 'undefined' )
    {
        where=where.substring(1, where.length - 1)
        where=where.split(",")
        if(where.length % 3 == 0){

            for (i=0;i<where.length;i=i+3){
                if(i!=0){
                    parse_where=parse_where+ " AND "
                }
                parse_where=parse_where + where[i] + where[i+2] +
                where[i+1]
            }
        }
    }
}

```



```

    }
  }
  if ( parse_where !== ""){
    pool.query(`UPDATE ${table} SET ${field}=${value} WHERE
    ${parse_where}`,(error, results) => {
      if (error) {
        response.status(400).json([{'details':'Bad request'}])
      }
      else{
        response.status(200).json(results.rows)
      }
    })
  }
}

const db_delete = (request, response) => {

  const table = request.body.table
  var where = request.body.where
  var parse_where=""
  if (typeof where !== 'undefined' )
  {
    where=where.substring(1, where.length - 1)
    where=where.split(",")
    if(where.length % 3 == 0){

      for (i=0;i<where.length;i=i+3){
        if(i!=0){
          parse_where=parse_where+ " AND "
        }
        parse_where=parse_where + where[i] + where[i+2] +
        where[i+1]
      }
    }
  }
  if ( parse_where !== ""){
    pool.query(`DELETE FROM ${table} WHERE ${parse_where}`,(error, results) => {
      if (error) {
        response.status(400).json([{'details':'Bad request'}])
      }
      else{
        response.status(200).json(results.rows)
      }
    })
  }
}
}

```

При успешном выполнении запроса, возвращается ответ в формате JSON, содержащий ответ базы данных, с кодом 200. При возникновении ошибки при запросе к БД возвращается ответ с кодом 400 и телом ответа “[{‘details’:‘Bad request’}]”.

Для того, чтобы иметь возможность получить из `index.js` доступ к функциям, реализованным в `queries.js`, функции были экспортированы, используя метод `module.exports`:

Листинг 2.2.5.

```
module.exports = {
  db_get,
  db_insert,
  db_delete,
  db_update,
}
```

Таким образом, база данных и веб-сервис были полностью реализованы. Их совокупная функциональность была протестирована.

Описанный сервис на GitLab: http://gitlab.icc.spbstu.ru/BigAwesomeTurtle/bar-db/tree/master/node_api_postgres

2.3. Android-приложение

Android-приложение, как и планировалось, не контактирует напрямую с базой данных, а взаимодействует с ней через API веб-сервиса. Используются POST запросы с внесением в тела запросов необходимых параметров.

Был реализован класс `ServerImpl.kt`, содержащий методы для взаимодействия с веб-сервисом. Функциям `db_get`, `db_insert`, `db_remove` и `db_update` передаются необходимые параметры в виде кортежа, содержащего данные в специальном формате. (Например, для указания таблицы, с которой требуется что-то сделать, необходимо передать `"table":"имя_таблицы"`). Каждый метод отправляет запросы на соответствующий endpoint, возвращает ответ сервиса в случае успеха и выводит сообщение об ошибке при неудачном обращении к сервису.

Листинг 2.3.1.

```
private fun db_request(
    curr_url: String,
    params: Map<String, Any>,
    currCallback: (resp: String) -> Unit
) {
    url = curr_url
    json = JSONObject()
    callback = currCallback
    for (elem in params) {
        json.put(elem.key, elem.value)
    }

    backgroundThread = Thread(getInfoRunnable)
    backgroundThread.start()
}
```

```

fun db_get(params: Map<String, Any>, currCallback: (resp: String) -> Unit) {
    db_request("HTTP://192.168.0.101:3222/get", params, currCallback)
}

fun db_insert(params: Map<String, Any>, currCallback: (resp: String) -> Unit) {
    db_request("HTTP://192.168.0.101:3222/insert", params, currCallback)
}

fun db_remove(params: Map<String, Any>, currCallback: (resp: String) -> Unit) {
    db_request("HTTP://192.168.0.101:3222/delete", params, currCallback)
}

fun db_update(params: Map<String, Any>, currCallback: (resp: String) -> Unit) {
    db_request("HTTP://192.168.0.101:3222/update", params, currCallback)
}

```

При открытии приложения пользователь видит следующий экран:

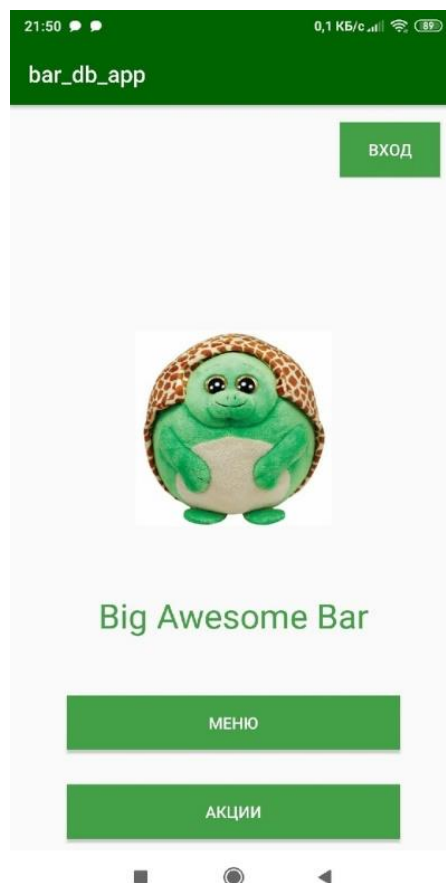


Рис.2.3.1. Главное меню приложения

При нажатии на кнопку “Меню”, список категорий запрашивается из базы данных и выводится на экран:

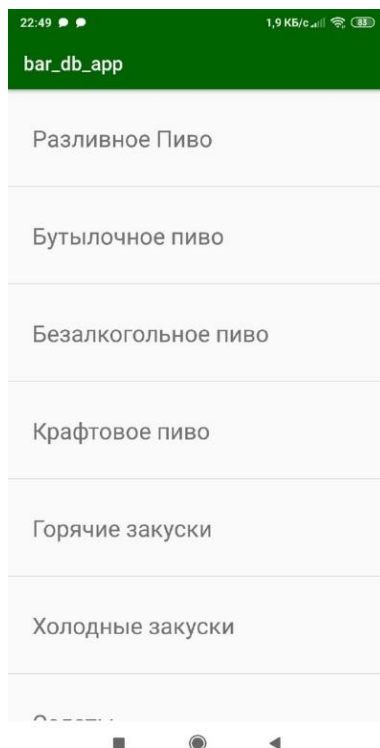


Рис.2.3.2. Выбор категории

При выборе какой-либо категории, запрашивается и выводится информация о позициях этой категории:

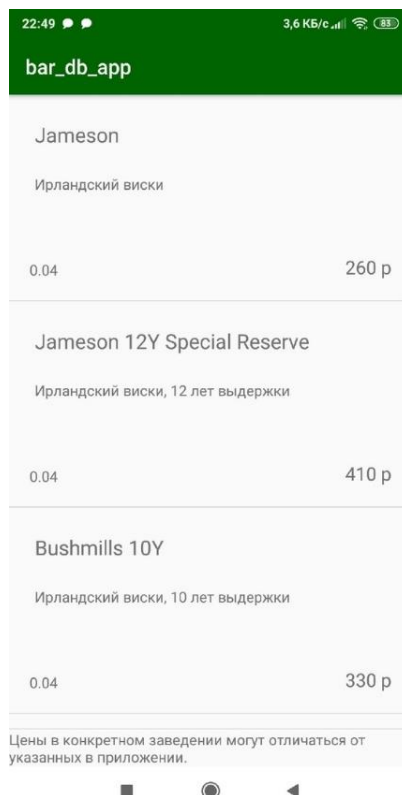


Рис.2.3.3. Позиции выбранной категории

При нажатии на кнопку “Акции”, выводится информация об актуальных акциях:



Рис.2.3.4. Актуальные акции

При нажатии на кнопку “Вход”, показывается диалоговое окно авторизации, где пользователь может ввести логин и пароль и, при успешной проверке, зайти в личный кабинет:

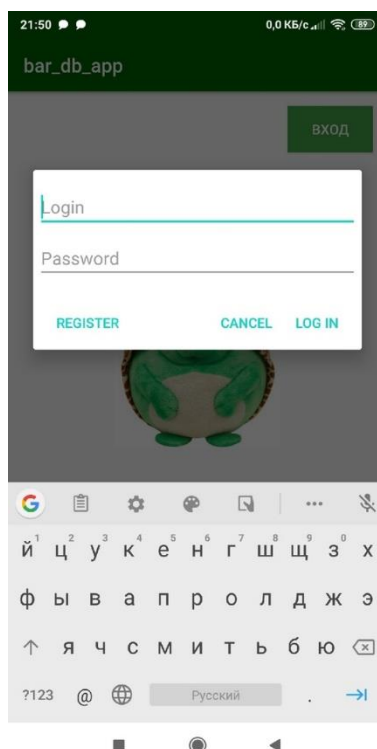


Рис.2.3.5. Диалоговое окно авторизации

В случае, если введён неправильный пароль, приложение сообщит об этом:

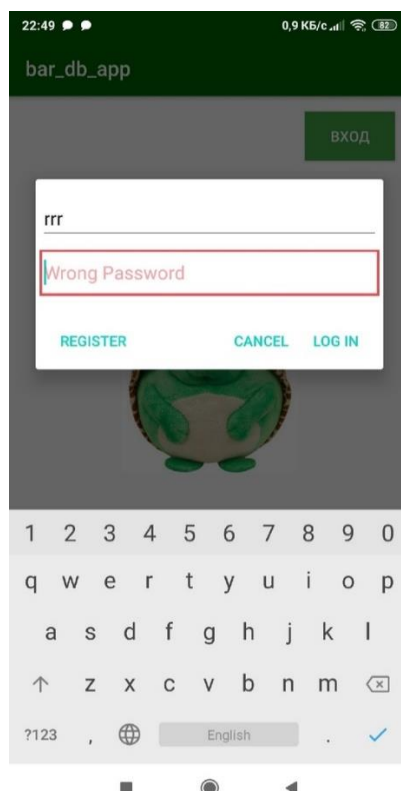


Рис.2.3.6. Ввод неверного пароля

При нажатии на кнопку “Register”, произойдёт переход к экрану регистрации:

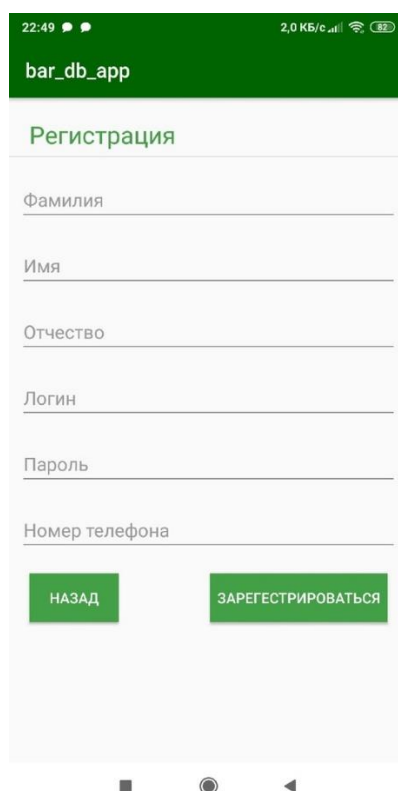


Рис.2.3.7. Экран регистрации

Если пользователь при регистрации заполнил не все поля, приложение напомним ему об этом:

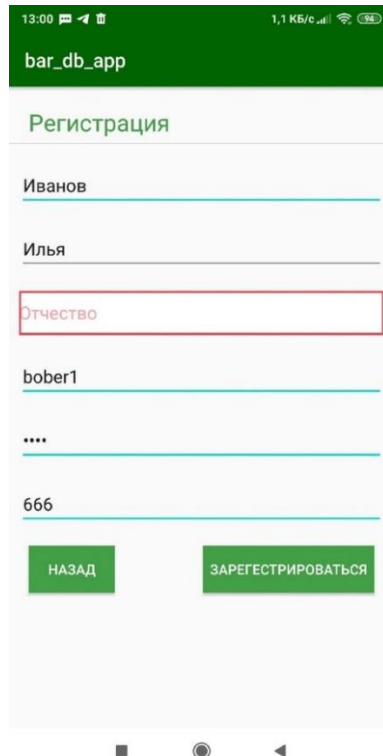


Рис.2.3.8. Заполнение не всех полей при регистрации

После успешной регистрации появится сообщение: «Регистрация прошла успешно» и пользователь будет перенаправлен на главную страницу:

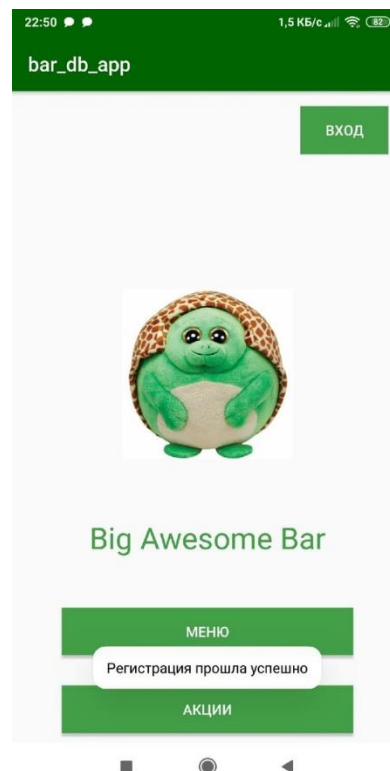


Рис.2.3.9. Сообщение об успешной регистрации

После входа в систему, пользователь может посмотреть свои данные в личном кабинете:

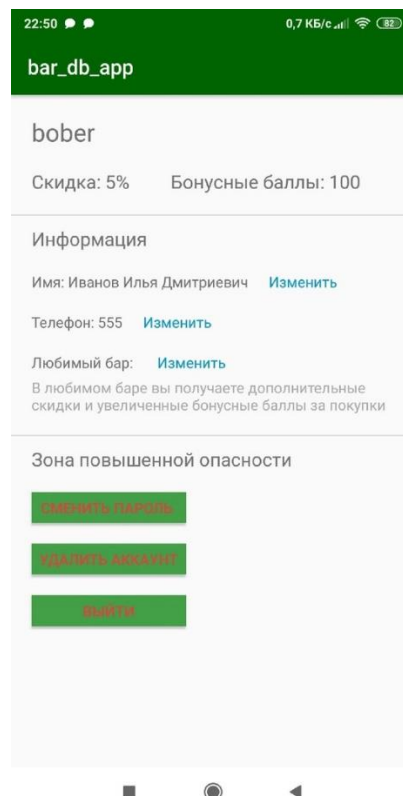


Рис.2.3.10. Личный кабинет

При желании, пользователь может изменить их. Например, поменять номер телефона:

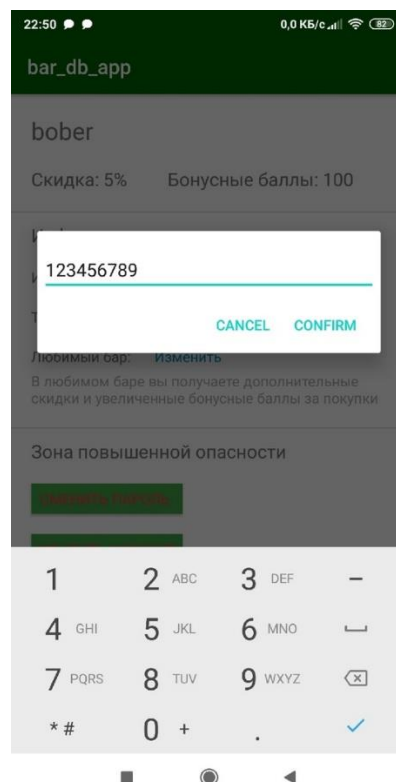


Рис.2.3.11. Изменение номера телефона

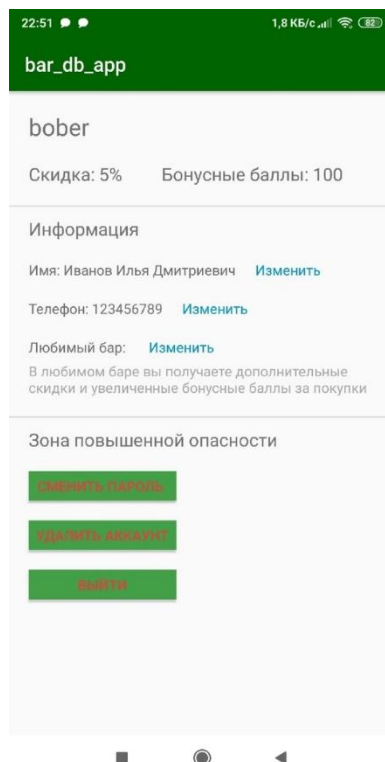


Рис.2.3.12. Изменение номера телефона

Или выбрать любимый бар:

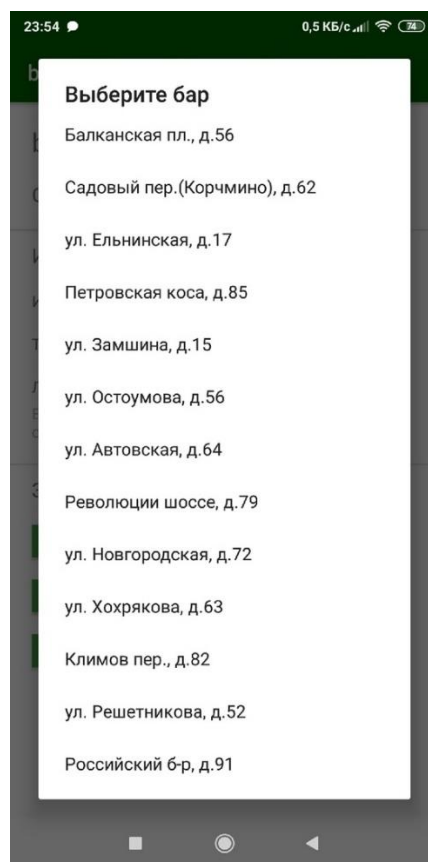


Рис.2.3.13. Выбор любимого бара

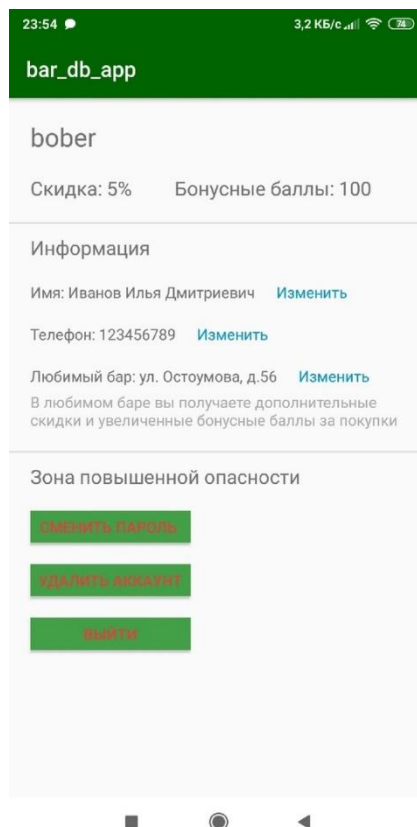


Рис.2.3.14. Выбор любимого бара

Изменить пароль для входа в систему:

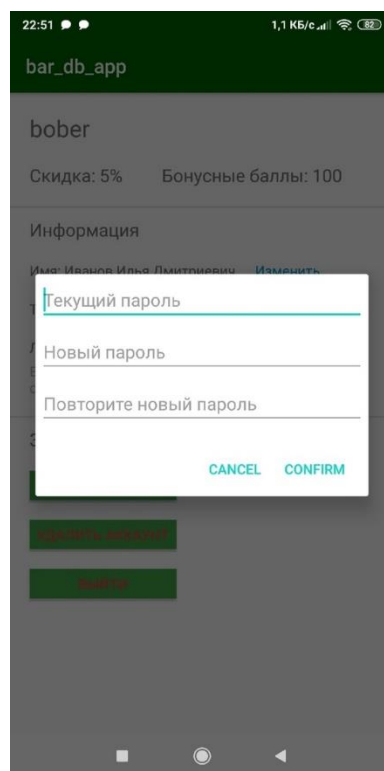


Рис.2.3.15. Изменение пароля

А также удалить аккаунт:

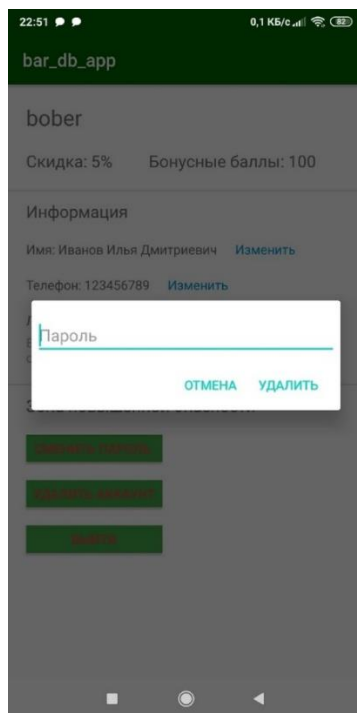


Рис.2.3.16. Удаление аккаунта

Кроме того, находясь в личном кабинете, можно выйти из системы, нажав на кнопку «Выйти», что делает возможным смену пользователя.

При любых ошибках при обращении к сервису пользователь будет видеть всплывающее сообщение “Что-то пошло не так при обращении к серверу”:

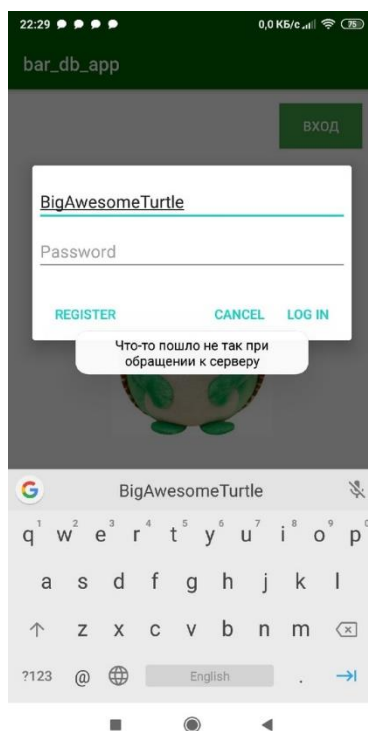


Рис.2.3.17. Ошибка при обращении к сервису

Код приложения на GitLab: http://gitlab.icc.spbstu.ru/BigAwesomeTurtle/bar-db/tree/master/bar_db_app

3. Выводы

В ходе выполнения данной курсовой работы были получены навыки по созданию веб-сервиса для взаимодействия с базой данных, по написанию API для него, а также по использованию созданного сервиса в Android-приложении. Данные навыки будут в дальнейшем полезны как в мобильной разработке, так и при работе с приложениями для других платформ.