

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ
ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ПОВОЛЖСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ
УНИВЕРСИТЕТ»

Факультет информатики и вычислительной техники

Кафедра информационной безопасности

Отчёт к курсовому проекту

по дисциплине “Безопасность систем баз данных”

Разработка базы данных для автосервиса

Выполнили: студенты группы БИ-31

Старыгин М.А., Михайлов А. В.,

Суманеева Т.С.

Проверил: доцент кафедры

ИБ Сучков Д.С.

Йошкар-Ола

2020 г.

СОДЕРЖАНИЕ

Введение	3
1. Техническое задание	4
1.1 Требования к курсовой работе.....	4
1.2 Требования к базе данных.....	4
1.3 Требования к API (минимальное количество реализованных методов) ..	4
2. Порядок выполнения работы	5
2.1 Этапы разработки базы данных.....	5-8
2.2 Этапы разработки API	9-30
3. Приложения.....	31
3.1 ER-диаграмма	31
3.2 Ссылка на github.com.....	31
4. Вывод	32

Введение

В курсовой работе рассматривается создание базы данных, предназначенной для отслеживания заказов в автомастерской. База данных позволяет контролировать заказы, изменять данные покупателей, вывод текущих услуг. Также реализована автоматизация приобретение услуги, где можно отследить статус заказа и узнать его стоимость.

1. Техническое задание

1.1 Требования к курсовой работе:

- Получить структуру данных из файла, согласно варианту. Привести к 3й нормальной форме. Добавить недостающие таблицы.
- Составить ER-диаграмму, применяя `mySQL Workbench` или `Dbearer`.
- Разработать API для базы данных на любом языке, выполняющемся на стороне сервера (`php`, `ASP.NET`, `Java`, `python`, `node.js`, etc).
- Взаимодействие должно осуществляться по клиент-серверной архитектуре, подключение с клиентской программы недопустимо.
- Провести настройку пользователей базы данных для разграничения прав доступа, привести пример конфигурации.
- Все документы и исходные коды для курсовой работы должны храниться под контролем системы контроля версий — `git` или `mercurial` (<https://github.com/>, <https://bitbucket.org/>).
- Во время сдачи курсового проекта необходимо предоставить отчет о проделанной работе в печатном виде (отчет).

1.2 Требования к базе данных

- Наличие не менее 7 таблиц, в том числе таблицы сессий и пользователей.
- Структура таблицы должна содержать не менее 3-х полей, одно из которых ключевое.
- Правильное использование типов данных.
- Обязательно использование триггеров и/или хранимых процедур.
- Форма нормализации не менее 3NF.
- Индексирование по полям поиска.

1.3 Требование к API (минимальное количество реализованных методов)

- аутентификация пользователя (создание сессии);
- добавление/удаление/изменение данных в таблицах;
- выборка данных их ключевых таблиц по запросам;
- выборка данных из таблиц с объединением результатов.

2. Порядок выполнения работы

2.1 Этапы разработки базы данных

Разработана база данных, содержащая 7 таблиц, в каждой таблице есть ключевое поле. Владелец всех таблиц является db_creator.

List of relations			
Schema	Name	Type	Owner
public	bank_info	table	db_creator
public	curr_service	table	db_creator
public	curr_service_id_crser_seq	sequence	db_creator
public	data_login	table	db_creator
public	data_login_id_user_seq	sequence	db_creator
public	info_user	table	db_creator
public	info_work	table	db_creator
public	log	table	db_creator
public	log_id_log_seq	sequence	db_creator
public	service	table	db_creator
public	service_id_service_seq	sequence	db_creator
(11 rows)			

Таблица *data_login* отвечает за хранение данных (email и зашифрованный пароль [алгоритмом **pbkdf2**]) для успешной сессии и аутентификации.

Таблица *info_user* отвечает за подробную информацию о пользователе и содержит в себе следующие характеристики: имя, адрес проживания, дата рождения и принимаемую роль в приложении.

Таблица *info_work* отвечает за информацию о месте работы, номера телефона для определенного пользователя.

Таблица *bank_info* хранит в себе зашифрованные номера банковских карт, с помощью библиотеки **cryptography** (работает с бинарными строками в определенной кодировке), и баланс средств.

Таблица *log* выполняет роль журнала посещений пользователей и хранит в себе почту-логин и время посещения.

Таблица *service* содержит в себе информацию о предоставляемых услугах автомастерской и содержит в себе название услуги и ее цену.

Таблица *curr_service* является наследником таблицы *service* за счет триггера **cucs_tg** и хранит в себе информацию о текущем статусе заказа пользователя, и время приобретения услуги.

Структуры реализованных таблиц:

- таблица *data_login*

Column	Type	Collation	Nullable	Default
email	character varying(255)		not null	
password	character varying(80)			
id_user	integer		not null	nextval('data_login_id_user_seq'::regclass)

Indexes:
 "data_login_pkey" PRIMARY KEY, btree (email)
 "data_login_id_user_key" UNIQUE CONSTRAINT, btree (id_user)

Referenced by:
 TABLE "bank_info" CONSTRAINT "bank_info_id_user_fkey" FOREIGN KEY (id_user) REFERENCES data_login(id_user)
 TABLE "curr_service" CONSTRAINT "curr_service_id_user_fkey" FOREIGN KEY (id_user) REFERENCES data_login(id_user) ON UPDATE CASCADE ON DELETE CASCADE
 TABLE "info_user" CONSTRAINT "info_user_id_user_fkey" FOREIGN KEY (id_user) REFERENCES data_login(id_user)
 TABLE "log" CONSTRAINT "log_email_fkey" FOREIGN KEY (email) REFERENCES data_login(email) ON UPDATE CASCADE ON DELETE CASCADE

- таблица *bank_info*

Column	Type	Collation	Nullable	Default
id_user	integer			
card_number	character varying(150)			
amount	integer			

Check constraints:
 "bank_info_amount_check" CHECK (amount >= 0)

Foreign-key constraints:
 "bank_info_id_user_fkey" FOREIGN KEY (id_user) REFERENCES data_login(id_user)

Triggers:
 cucs_tg AFTER UPDATE ON bank_info FOR EACH ROW EXECUTE FUNCTION check_update_curr_service()

- таблица *log*

Column	Type	Collation	Nullable	Default
id_log	integer		not null	nextval('log_id_log_seq'::regclass)
email	character varying(255)			
date_log	character varying(25)			

Indexes:
 "log_pkey" PRIMARY KEY, btree (id_log)

Foreign-key constraints:
 "log_email_fkey" FOREIGN KEY (email) REFERENCES data_login(email) ON UPDATE CASCADE ON DELETE CASCADE

- таблица *service*

Column	Type	Collation	Nullable	Default
id_service	integer		not null	nextval('service_id_service_seq'::regclass)
serv_name	character varying(45)			
cost	integer			

Indexes:
 "service_id_service_key" UNIQUE CONSTRAINT, btree (id_service)

Check constraints:
 "service_cost_check" CHECK (cost >= 0)

- таблица *curr service*

Column	Type	Collation	Nullable	Default
id_crser	integer		not null	nextval('curr_service_id_crser_seq'::regclass)
id_user	integer			
curr_service	character varying(45)			
curr_time	character varying(30)			
status	boolean			

Indexes:
 "curr_service_pkey" PRIMARY KEY, btree (id_crser)

Foreign-key constraints:
 "curr_service_id_user_fkey" FOREIGN KEY (id_user) REFERENCES data_login(id_user) ON UPDATE CASCADE ON DELETE CASCADE

- таблица *info_work*

Column	Type	Collation	Nullable	Default
name	character varying(45)		not null	
phone	character varying(45)			
work_place	character varying(65)			

Indexes:
 "info_work_pkey" PRIMARY KEY, btree (name)
 Foreign-key constraints:
 "info_work_name_fkey" FOREIGN KEY (name) REFERENCES info_user(name) ON UPDATE CASCADE ON DELETE CASCADE

- таблица *info_user*

Column	Type	Collation	Nullable	Default
id_user	integer			
name	character varying(45)			
home_place	character varying(256)			
b_date	character varying(25)			
role	character varying(5)			'user'::character varying

Indexes:
 "info_user_name_key" UNIQUE CONSTRAINT, btree (name)
 Foreign-key constraints:
 "info_user_id_user_fkey" FOREIGN KEY (id_user) REFERENCES data_login(id_user)
 Referenced by:
 TABLE "info_work" CONSTRAINT "info_work_name_fkey" FOREIGN KEY (name) REFERENCES info_user(name) ON UPDATE CASCADE ON DELETE CASCADE

Используемый триггер и функция для него:

- cucs_tg – триггер, отвечающий за текущее состояние выполнения услуги
- CREATE FUNCTION check_update_curr_service() RETURNS TRIGGER AS \$\$
 BEGIN
 CREATE TABLE IF NOT EXISTS curr_service
 (
 id_user integer REFERENCES demo (id) ON DELETE CASCADE on
 UPDATE CASCADE,
 curr_service VARCHAR(200),
 curr_time VARCHAR(30),
 status BOOLEAN
);
 INSERT INTO curr_service (id_user, status)
 VALUES ([OLD.id](#), FALSE);
 RETURN NULL;
 END;
 \$\$ LANGUAGE plpgsql; (ред.)
- CREATE TRIGGER cucs_tg AFTER UPDATE ON demo
 FOR EACH ROW EXECUTE PROCEDURE check_update_curr_service();

Проведена настройка пользователей базы данных для разграничения прав доступа и прав на редактирование структуры базы данных:

Role name	Attributes	Member of
db_creator		{ }
head	Superuser	{ }
postgres	Superuser, Create role, Create DB, Replication, Bypass RLS	{ }

2.2 Этапы разработки API

Было разработано API для логинации и аутентификации пользователей, написанное на языке Python 3.8.5 + Flask.

```
@app.route('/validate', methods=["POST"])
def validate():
    if request.method == "POST":
        global __nameUser, __mailUser, __b_dateUser, __b_placeUser, __mailUser, __passwordUser
        global __idUser, __total, __role, now, user, i_counter

        __mailUser = request.form.get("email")
        __passwordUser = hashlib.pbkdf2_hmac('sha256', request.form.get("pass").encode(), salt, 100000).hex()

        cursor.execute("""SELECT *
                            FROM data_login
                            WHERE email=%s AND password=%s """, (__mailUser, __passwordUser))
        answer = cursor.fetchall()

        if (len(answer) != 0):
            __idUser = answer[0][2]

            cursor.execute("""SELECT info_user.name, info_user.home_place, info_user.b_date, info_user.role, bank_info.amount
                                FROM info_user
                                RIGHT JOIN bank_info ON (info_user.id_user = bank_info.id_user)
                                WHERE info_user.id_user = %s
                                ORDER BY info_user.id_user;""",
                            (__idUser))

            user.commit()
            answer = cursor.fetchall()[0]

            __nameUser = answer[0]
            __b_placeUser = answer[1]
            __b_dateUser = answer[2]
            __role = answer[3]
            __total = answer[4]
            now = getTime()
            mt = dictMonth[str(now.month)]
            day = dictDay[str(now.weekday())]

            cursor.execute("""INSERT INTO log (email, date_log)
                                VALUES (%s, %s)""",
                            (__mailUser, now.strftime("{} {} %d %Y %H:%M:%S").format(day, mt)))

            user.commit()
            if __role == "owner":
                user = psycopg2.connect(database="main", user="head", password="123456W", host="localhost", port=5432)
                return redirect(url_for('indexer'))
            else: return redirect(url_for('login'))
```

Изменения данных в таблицах

- Со стороны админа

```
@app.route('/change/<id_user>', methods=["POST"])
def changeByAdmin(id_user):
    if __role == "owner":
        nameUser = request.form.get("name")
        mailUser = request.form.get("login")
        phone = request.form.get("phone")
        passwordUser = hashlib.pbkdf2_hmac('sha256', request.form.get("pass").encode(), salt, 100000).hex()

        user = psycopg2.connect(database="main", user="head", password="123456W", host="localhost", port=5432)
        cursor = user.cursor()
        cursor.execute("""UPDATE data_login
                            SET email=%s, password=%s
                            WHERE id_user=%s""",
                        (mailUser, passwordUser, id_user))

        user.commit()

        user = psycopg2.connect(database="main", user="head", password="123456W", host="localhost", port=5432)
        cursor = user.cursor()
        cursor.execute("""UPDATE info_user
                            SET name=%s
                            WHERE id_user=%s""",
                        (nameUser, id_user))

        user.commit()

        user = psycopg2.connect(database="main", user="head", password="123456W", host="localhost", port=5432)
        cursor = user.cursor()
        cursor.execute("""UPDATE info_work
                            SET phone=%s
                            WHERE name=%s""",
                        (nameUser, id_user))

        user.commit()
        return redirect(url_for('indexer'))
```

- Со стороны пользователя

```
@app.route('/change', methods=["POST"])
def change():
    global nameUser, mailUser, __b_dateUser, __b_placeUser, __mailUser, __passwordUser, __idUser
    nameUser = request.form.get("name")
    mailUser = request.form.get("login")
    __b_dateUser = "Mon" + " " + dictMonth[request.form.get("mounth")] + " " + request.form.get("day") + " " + request.form.get("year") + " " + "21:04:37"
    __b_placeUser = request.form.get("ncity")
    __passwordUser = hashlib.pbkdf2_hmac('sha256', request.form.get("pass").encode(), salt, 100000).hex()

    user = psycopg2.connect(database="main", user="db_creator", password="12345Q", host="localhost", port= 5432)
    cursor = user.cursor()
    cursor.execute("""UPDATE data_login
                      SET email=%s, password=%s
                      WHERE id user = %s""",
                  (__mailUser, __passwordUser, __idUser))

    user.commit()

    user = psycopg2.connect(database="main", user="db_creator", password="12345Q", host="localhost", port= 5432)
    cursor = user.cursor()
    cursor.execute("""UPDATE info_user
                      SET name=%s, home_place=%s, b_date=%s
                      WHERE id user = %s""",
                  (__nameUser, __b_placeUser, __b_dateUser, __idUser))

    user.commit()

    return redirect(url_for('indexer'))
```

Выборка данных из ключевых таблиц по запросам и из таблиц с объединением результата

- Вывод услуг для пользователя

```
@app.route('/service', methods=["POST", "GET"])
def service():
    dictServ = []
```

```
user = psycopg2.connect(database="main", user="db_creator", password="12345Q", host="localhost", port= 5432)
cursor = user.cursor()
cursor.execute("""SELECT *
                  FROM service""")
answer = cursor.fetchall()
for i in (i for i in answer):
    if total >= int(i[2]):
        helpDict = {}
        helpDict["id"] = int(i[0])
        helpDict["name"] = i[1]
        helpDict["cost"] = int(i[2])
        dictServ.append(helpDict)

return render_template("service.html", name= nameUser, login= mailUser,
                      b_date= __b_dateUser, b_place= __b_placeUser,
                      dict=dictServ, money= __total)
```

- Вывод статуса заказа

```
@app.route('/status', methods=["POST", "GET"])
def status():
    dictServ = []

    user = psycopg2.connect(database="main", user="db_creator", password="12345Q", host="localhost", port= 5432)
    cursor = user.cursor()
    cursor.execute("""SELECT *
                      FROM curr_service
                      WHERE id user=%s
                      ORDER BY curr_time DESC""",
                  (__idUser))

    answer = cursor.fetchall()
    for i in (i for i in answer):
        helpDict = {}
        helpDict["id user"] = int(i[1])
        helpDict["curr_service"] = i[2]
        helpDict["curr_time"] = i[3]
        helpDict["status"] = i[4]
        if helpDict["status"] == False: helpDict["status"] = "In process"
        else: helpDict["status"] = "Ready"
        dictServ.append(helpDict)

    return render_template("status.html", name= nameUser, login= mailUser,
                          b_date= __b_dateUser, b_place= __b_placeUser,
                          dict=dictServ, money= __total)
```

- Вывод пользователей в меню изменений пользователя со стороны админа с пагинацией

```
@app.route('/last', methods=["POST"])
def last():
    global curr_pg_user
    curr_pg_user -= 5
    return redirect(url_for('account'))

@app.route('/next', methods=["POST"])
def next():
    global curr_pg_user
    curr_pg_user += 5
    return redirect(url_for('account'))

@app.route('/account', methods=["POST", "GET"])
def account():
    if __role == "owner":
        dictServ = []
        user = psycopg2.connect(database="main", user="head", password="123456W", host="localhost", port=5432)
        cursor = user.cursor()

        cursor.execute("""SELECT data_login.id_user, info_user.name, info_work.phone, data_login.email, data_login.password
        FROM data_login
        RIGHT JOIN info_user ON (data_login.id_user=info_user.id_user)
        RIGHT JOIN info_work ON (info_work.name = info_user.name)
        ORDER BY id_user
        LIMIT 5
        OFFSET %s
        """, ([curr_pg_user]))

        answer = cursor.fetchall()
        for i in answer:
            helpDict = {}
            helpDict["id user"] = int(i[0])
            helpDict["name"] = i[1]
            helpDict["phone"] = i[2]
            helpDict["email"] = i[3]
            helpDict["password"] = i[4]
            dictServ.append(helpDict)

        return render_template('users.html', name= nameUser,
                               login= mailUser, b_date= b.dateUser,
                               b_place= b.placeUser, dict=dictServ,
                               money= __total, cpg=int(curr_pg_user/5)+1)
```

- Вывод текущих заявок в меню админа с пагинацией

```
@app.route('/lastPg', methods=["POST"])
def lastPg():
    global curr_pg_st
    curr_pg_st -= 5
    return redirect(url_for('service'))

@app.route('/nextPg', methods=["POST"])
def nextPg():
    global curr_pg_st
    curr_pg_st += 5
    return redirect(url_for('service'))

@app.route('/service', methods=["POST", "GET"])
def service():
    dictServ = []
    if __role == "owner":
        user = psycopg2.connect(database="main", user="head", password="123456W", host="localhost", port=5432)
        cursor = user.cursor()
        cursor.execute("""SELECT curr_service.id_user, info_user.name, curr_service.curr_service,
        curr_service.curr_time, service.cost, curr_service.status
        FROM curr_service
        RIGHT JOIN info_user ON (curr_service.id_user = info_user.id_user)
        RIGHT JOIN service ON (curr_service.curr_service = service.serv_name)
        WHERE curr_service.curr_service IS NOT NULL
        ORDER BY curr_service.curr_time DESC
        LIMIT 5
        OFFSET %s""", ([curr_pg_st]))

        answer = cursor.fetchall()
        for i in (i for i in answer):
            helpDict = {}
            helpDict["id user"] = int(i[0])
            helpDict["name"] = i[1]
            helpDict["curr service"] = i[2]
            helpDict["curr time"] = i[3]
            helpDict["cost"] = i[4]
            helpDict["status"] = i[5]
            if helpDict["status"] == False: helpDict["status"] = "In process"
            else: helpDict["status"] = "Ready"
            dictServ.append(helpDict)

        return render_template("adstatus.html", name= nameUser, login= mailUser,
                               b_date= b.dateUser, b_place= b.placeUser,
                               dict=dictServ, money= __total, cpg=int(curr_pg_st/5)+1)
```


Предварительно перед выполнением вышеуказанных и последующих запросов был написан файл на языке python для работы с postgresql: название файла **work_withBD.py**

```
class Control:
    def __init__(self, db_name, user_name, password, host, port):
        self.databaseName = db_name
        self.userName = user_name
        self.userPassword = password
        self.host = host
        self.port = port

        self.connection = psycopg2.connect(database=self.databaseName, user=self.userName,
                                           password=self.userPassword, host=self.host,
                                           port=self.port)
        self.current = self.connection.cursor()

    def createTable(self, nameTable: str, arrayLines: dict):
        keys = [i for i in arrayLines]
        helpString = "CREATE TABLE " + nameTable + " ("
        for i in keys:
            helpString += i + " " + arrayLines[i] + ", "
        helpString = helpString[:len(helpString) - 2]
        helpString += ")"
        self.current.execute(helpString)
        self.connection.commit()

    def updateTable(self, nameTable: str, condition: str):
        self.current.execute("ALTER TABLE {} {}".format(nameTable, condition))
        self.connection.commit()

    def getTableColumns(self, nameTable: str):
        self.current.execute("SELECT * FROM " + nameTable + " LIMIT 0")
        self.connection.commit()
        return ([desc[0] for desc in self.current.description])

    def createElTable(self, nameTable: str, values: tuple):
        self.current.execute(
            "INSERT INTO {} ({} ) VALUES {}".format(nameTable, ", ".join(self.getTableColumns(nameTable)[0:]), values)) #for pullinfo change [1:] -> [0:]
        self.connection.commit()

    def updateElTable(self, nameTable: str, condition: str, **kwargs):
        self.current.execute("UPDATE {} SET {} WHERE {}".format(
            nameTable,
            ", ".join(key+'='+value for key, value in kwargs.items()),
            condition))
        #updateElTable("apps", "city = 'San Francisco' AND date = '2003-07-03'", temp_lo='temp_lo+1', temp_hi='temp_lo+15')
        self.connection.commit()

    def deleteElTable(self, nameTable: str, usl: str):
        self.current.execute("DELETE FROM {} WHERE {}".format(nameTable, usl))
        self.connection.commit()

    def printEl(self, nameTable: str, orderBy='', limit=10000, offset=0):
        helpString = ""
        for i in self.getTableColumns(nameTable): helpString += i + ", "
        helpString = helpString[:len(helpString) - 2]
        if (orderBy == ''): orderBy = self.getTableColumns(nameTable)[0]
        self.current.execute("SELECT {} FROM {} ORDER BY {} LIMIT {} OFFSET {}".format(
            helpString, nameTable, orderBy, limit, offset))
        array = self.current.fetchall()
        return array

    def printCurrEl(self, nameTable: str, wtfselect='', orderBy='', limit=10000, offset=0):
        if (orderBy == ''): orderBy = self.getTableColumns(nameTable)[0]
        self.current.execute("SELECT {} FROM {} ORDER BY {} LIMIT {} OFFSET {}".format(
            wtfselect, nameTable, orderBy, limit, offset))
        array = self.current.fetchall()
        return array
```

Данный файл **work_withBD.py** помогает нам автоматизировать заполнение таблиц из файлов формата.csv:

pullinfo.py:

```
from work withBD import *
from cryptography.fernet import Fernet
import hashlib

admin = Control("main", "db creator", "12345Q", "localhost", 5432)
cipher = Fernet(b'NYrglWwX0HXsabMDuxApVII00X8NXRLSZBbdmNI9nus=')
salt = 'dsvsdsvs'.encode()

with open('data_login.csv', 'r') as csvfile:
    spamreader = csv.reader(csvfile)
    i = 1
    for row in spamreader:
        arr = row[0].split(";")
        admin.createElTable("data_login", (arr[0], hashlib.pbkdf2_hmac('sha256', arr[1].encode(), salt, 100000).hex(), i)) #data_login.csv
        i += 1

with open('bank_info.csv', 'r') as csvfile:
    spamreader = csv.reader(csvfile)
    i = 1
    for row in spamreader:
        arr = row[0].split(";")
        text = cipher.encrypt(bytes(arr[0], 'utf-8'))
        admin.createElTable("bank_info", (i, text.decode("utf-8"), int(arr[1]))) #bank_info.csv
        i += 1

with open('info_user.csv', 'r') as csvfile:
    spamreader = csv.reader(csvfile)
    i = 1
    for row in spamreader:
        arr = row[0].split(";")
        admin.createElTable("info_user", (i, arr[0], arr[1], arr[2], 'user')) #info_work.csv
        i += 1

with open('info_work.csv', 'r') as csvfile:
    spamreader = csv.reader(csvfile)
    for row in spamreader:
        arr = row[0].split(";")
        admin.createElTable("info_work", (arr[0], arr[1], arr[2])) #info_work.csv

admin.updateElTable("info_user", "name='Sergey Davidov'", "role='''+owner+'''")

with open('service.csv', 'r') as csvfile:
    spamreader = csv.reader(csvfile)
    i = 1
    for row in spamreader:
        arr = row[0].split(";")
        admin.createElTable("service", (i, arr[0][0:25], arr[1])) #service.csv
        i += 1
```

(Данный скрипт выполняется один раз, перед запуском приложения, и в дальнейшем часть кода комментируется)

- Пополнение баланса пользователя

```
@app.route('/add', methods=["POST"])
def add():
    user = psycopg2.connect(database="main", user="db_creator", password="12345Q", host="localhost", port= 5432)
    cursor = user.cursor()
    cursor.execute("""SELECT card number
                      FROM bank_info
                      WHERE id_user= %s""",
                  ([ idUser]))
    answer = cursor.fetchall()[0]

    answer = cipher.decrypt(answer[0].encode('utf-8')).decode('utf-8')
    cardNum = "**** * " + str(answer)[-4:]

    cursor.execute("""SELECT info user.id user, info user.name, info work.phone
                      FROM info user RIGHT JOIN info work ON (info_user.name=info_work.name)
                      WHERE info user.id_user= %s""",
                  ([ idUser]))
    answer = cursor.fetchall()[0]

    phone = answer[2]
    return render_template("addbalance.html", name= nameUser,
                           login= mailUser, b_date= b_dateUser,
                           b_place= b_placeUser, money= __total,
                           card=cardNum, phone=phone)
```

```

@app.route('/up', methods=["POST"])
def up():
    global __total
    addbalance = request.form.get("add")
    __total = __total + int(addbalance)

    user = psycopg2.connect(database="main", user="db_creator", password="12345Q", host="localhost", port= 5432)
    cursor = user.cursor()

    cursor.execute("""UPDATE bank info
                      SET amount=%s
                      WHERE id_user= %s""",
                      (__total, __idUser))

    user.commit()
    cursor.execute("""DELETE FROM curr_service WHERE (curr_service IS NULL) AND (curr_time IS NULL)""")
    user.commit()
    return redirect(url_for('indexer'))

```

- Изменения данных пользователя

```

@app.route('/edit/<id_user>', methods=["POST"])
def edit(id_user):
    user = psycopg2.connect(database="main", user="db_creator", password="12345Q", host="localhost", port= 5432)
    cursor = user.cursor()

    cursor.execute("""SELECT data_login.id_user, info_user.name, info_work.phone, data_login.email, data_login.password
                      FROM data_login
                      RIGHT JOIN info_user ON (data_login.id_user=info_user.id_user)
                      RIGHT JOIN info_work ON (info_work.name = info_user.name)
                      WHERE data_login.id_user=%s""",
                      ([id_user]))
    answer = cursor.fetchall()[0]
    helpDict = {}
    helpDict["id_user"] = int(answer[0])
    helpDict["name"] = answer[1]
    helpDict["phone"] = answer[2]
    helpDict["email"] = answer[3]
    helpDict["password"] = answer[4]
    return render_template("adccount.html", dict=helpDict,
                           name= nameUser, login= mailUser,
                           b_date= __b_dateUser, b_place= __b_placeUser)

```

- Оплата заказа

```

@app.route('/pay/<ordName>/<int:ordCost>', methods=["POST"])
def pay(ordCost, ordName):
    global __total, now
    __total = __total - ordCost

    user = psycopg2.connect(database="main", user="db_creator", password="12345Q", host="localhost", port= 5432)
    cursor = user.cursor()
    cursor.execute("""UPDATE bank info
                      SET amount=%s
                      WHERE id_user=%s""",
                      (__total, __idUser))

    now = getTime()
    mt = dictMonth[str(now.month)]
    day = dictDay[str(now.weekday())]

    cursor.execute("""UPDATE curr_service
                      SET curr_service=%s, curr_time=%s
                      WHERE id_user=%s AND curr_service IS NULL AND curr_time IS NULL""",
                      (ordName, now.strftime("{ } {} %d %Y %H:%M:%S").format(day, mt), (__idUser)))
    user.commit()
    return redirect(url_for('indexer'))

```


- Статусы заказов пользователя

```
@app.route('/status', methods=["POST", "GET"])
def status():
    dictServ = []

    user = psycopg2.connect(database="main", user="db_creator", password="12345Q", host="localhost", port= 5432)
    cursor = user.cursor()
    cursor.execute("""SELECT *
                        FROM curr_service
                        WHERE id_user=%s
                        ORDER BY curr_time DESC""",
                    ([_idUser]))

    answer = cursor.fetchall()
    for i in (i for i in answer):
        helpDict = {}
        helpDict["id_user"] = int(i[1])
        helpDict["curr_service"] = i[2]
        helpDict["curr_time"] = i[3]
        helpDict["status"] = i[4]
        if helpDict["status"] == False: helpDict["status"] = "In process"
        else: helpDict["status"] = "Ready"
        dictServ.append(helpDict)
    return render_template("status.html", name= nameUser, login= mailUser,
                           b_date= b_dateUser, b_place= b_placeUser,
                           dict=dictServ, money= total)
```

- Отмена заказа пользователем

```
@app.route('/cancel/<ordName>/<ordDate>', methods=["POST"])
def cancel(ordName, ordDate):
    global _total
    user = psycopg2.connect(database="main", user="db_creator", password="12345Q", host="localhost", port= 5432)
    cursor = user.cursor()
    cursor.execute("""SELECT curr_service, curr_time, service.cost
                        FROM curr_service
                        RIGHT JOIN service ON (curr_service.curr_service = service.serv_name)
                        WHERE id_user = %s AND curr_time LIKE %s AND curr_service LIKE %s""",
                    (_idUser, ordDate, ordName))

    answer = cursor.fetchall()[0]
    _total += answer[2]
    cursor.execute("""DELETE FROM curr_service
                        WHERE id_user=%s AND curr_time LIKE %s AND curr_service LIKE %s""",
                    (_idUser, ordDate, ordName))

    cursor.execute("""UPDATE bank_info
                        SET amount=%s
                        WHERE id_user=%s""",
                    (_total, _idUser))

    cursor.execute("""DELETE FROM curr_service
                        WHERE curr_service IS NULL AND curr_time IS NULL""")
    user.commit()

    return redirect(url_for('service'))
```

- Изменение данных пользователем

```
@app.route('/change', methods=["POST"])
def change():
    global nameUser, mailUser, _b_dateUser, _b_placeUser, mailUser, _passwordUser, _idUser
    _nameUser = request.form.get("name")
    _mailUser = request.form.get("login")
    _b_dateUser = "Mon" + " " + dictMonth[request.form.get("month")] + " " + request.form.get("day") + " " + request.form.get("year") + " " + "21:04:37"
    _b_placeUser = request.form.get("ncity")
    _passwordUser = hashlib.pbkdf2_hmac('sha256', request.form.get("pass").encode(), salt, 100000).hex()

    user = psycopg2.connect(database="main", user="db_creator", password="12345Q", host="localhost", port= 5432)
    cursor = user.cursor()
    cursor.execute("""UPDATE data_login
                        SET email=%s, password=%s
                        WHERE id_user = %s""",
                    (_mailUser, _passwordUser, _idUser))
    cursor.execute("""UPDATE info_user
                        SET name=%s, home_place=%s, b_date=%s
                        WHERE id_user = %s""",
                    (_nameUser, _b_placeUser, _b_dateUser, _idUser))
    user.commit()

    return redirect(url_for('indexer'))
```

- Пополнение баланса пользователем

```
@app.route('/up', methods=["POST"])
def up():
    global __total
    addbalance = request.form.get("add")
    __total = __total + int(addbalance)

    user = psycopg2.connect(database="main", user="db_creator", password="12345Q", host="localhost", port= 5432)
    cursor = user.cursor()

    cursor.execute("""UPDATE bank_info
                      SET amount=%s
                      WHERE id_user= %s""",
                  (__total, __idUser))
    user.commit()

    user = psycopg2.connect(database="main", user="db_creator", password="12345Q", host="localhost", port=5432)
    cursor = user.cursor()

    cursor.execute("""DELETE FROM curr_service WHERE (curr_service IS NULL) AND (curr_time IS NULL)""")
    user.commit()
    return redirect(url_for('indexer'))
```

- Статус заказа отмена/выполнено на стороне админа

- Выполнено

```
@app.route('/done/<idUser>/<ordName>/<ordDate>', methods=["POST"])
def done(idUser, ordName, ordDate):
    global __total
    if __role == "owner":
        user = psycopg2.connect(database="main", user="head", password="123456W", host="localhost", port=5432)
        cursor = user.cursor()
        cursor.execute("""SELECT curr_service, curr_time, service.cost
                          FROM curr_service
                          RIGHT JOIN service ON (curr_service.curr_service = service.serv_name)
                          WHERE id_user = %s AND curr_time LIKE %s AND curr_service LIKE %s""",
                      (idUser, ordDate, ordName))

        user.commit()
        answer = cursor.fetchall()[0]
        plus = answer[2]

        cursor.execute("""UPDATE curr_service
                          SET status=%s
                          WHERE id_user=%s AND curr time LIKE %s AND curr_service LIKE %s""",
                      (True, idUser, ordDate, ordName))

        __total += plus
        cursor.execute("""UPDATE bank_info
                          SET amount=%s
                          WHERE id_user=%s""",
                      (__total, __idUser))

        user.commit()
        cursor.execute("""DELETE FROM curr_service
                          WHERE curr_service IS NULL AND curr_time IS NULL""")
        user.commit()

        return redirect(url_for('service'))
```

-Отменено


```

@app.route('/cancel/<idUser>/<ordName>/<ordDate>', methods=["POST"])
def canceler(idUser, ordName, ordDate):
    global total
    if __role == "owner":
        user = psycopg2.connect(database="main", user="head", password="123456W", host="localhost", port=5432)
        cursor = user.cursor()
        cursor.execute("""SELECT curr_service, curr_time, service.cost
                           FROM curr_service
                           RIGHT JOIN service ON (curr_service.curr_service = service.serv_name)
                           WHERE id_user = %s AND curr_time LIKE %s AND curr_service LIKE %s""",
                           (idUser, ordDate, ordName))

        user.commit()
        answer = cursor.fetchall()[0]
        minus = answer[2]

        user = psycopg2.connect(database="main", user="head", password="123456W", host="localhost", port=5432)
        cursor = user.cursor()
        cursor.execute("""DELETE FROM curr_service
                           WHERE id_user=%s AND curr_time LIKE %s AND curr_service LIKE %s""",
                           (idUser, ordDate, ordName))

        user.commit()

        user = psycopg2.connect(database="main", user="head", password="123456W", host="localhost", port=5432)
        cursor = user.cursor()
        cursor.execute("""SELECT *
                           FROM bank_info
                           WHERE id user=%s""",
                           ([idUser]))

        user.commit()
        answer = cursor.fetchall()[0]

        summ = answer[2] + minus

        cursor.execute("""UPDATE bank_info
                           SET amount=%s
                           WHERE id user=%s""",
                           (summ, idUser))

        cursor.execute("""DELETE FROM curr_service
                           WHERE curr_service IS NULL AND curr_time IS NULL""")
        user.commit()

        return redirect(url_for('service'))

```

- Изменение данных пользователя админом

```

@app.route('/change/<id_user>', methods=["POST"])
def changeByAdmin(id_user):
    if __role == "owner":
        nameUser = request.form.get("name")
        mailUser = request.form.get("login")
        phone = request.form.get("phone")
        passwordUser = hashlib.pbkdf2_hmac('sha256', request.form.get("pass").encode(), salt, 100000).hex()

        user = psycopg2.connect(database="main", user="head", password="123456W", host="localhost", port=5432)
        cursor = user.cursor()
        cursor.execute("""UPDATE data_login
                           SET email=%s, password=%s
                           WHERE id user=%s""",
                           (mailUser, passwordUser, id_user))

        user.commit()

        user = psycopg2.connect(database="main", user="head", password="123456W", host="localhost", port=5432)
        cursor = user.cursor()
        cursor.execute("""UPDATE info_user
                           SET name=%s
                           WHERE id user=%s""",
                           (nameUser, id_user))

        user.commit()

        user = psycopg2.connect(database="main", user="head", password="123456W", host="localhost", port=5432)
        cursor = user.cursor()
        cursor.execute("""UPDATE info_work
                           SET phone=%s
                           WHERE name=%s""",
                           (nameUser, id_user))

        user.commit()
        return redirect(url_for('indexer'))

```

PBKDF2 (англ. *Password-Based Key Derivation Function*) — стандарт формирования ключа на основе пароля. Является частью PKCS #5 v2.0 (RFC 2898). Заменяет PBKDF1, который ограничивал длину порождаемого ключа 160 битами.

PBKDF2 использует псевдослучайную функцию для получения ключей. Длина генерируемого ключа не ограничивается (хотя эффективная мощность пространства ключей может быть ограничена особенностями применяемой псевдослучайной функции). Использование PBKDF2 рекомендовано для новых программ и продуктов. В качестве псевдослучайной может быть выбрана криптографическая хеш-функция, шифр, HMAC.

В Российской Федерации использование функции PBKDF2 регламентируется рекомендациями по стандартизации Р 50.1.111-2016 "Парольная защита ключевой информации"^[1], при этом в качестве псевдослучайной функции рекомендуется использовать функцию выработки имитовставки HMAC на основе бесключевой функции хеширования Стрибог (ГОСТ Р 34.11-2012).

Алгоритм

Общий вид вызова PBKDF2:

$$DK = PBKDF2(PRF, P, S, c, dkLen)$$

Опции алгоритма:

- *PRF* — псевдослучайная функция, с выходом длины *hLen*
- *P* — мастер-пароль
- *S* — соль (salt)
- *c* — количество итераций, положительное целое число
- *dkLen* — желаемая длина ключа (не более $(2^{32} - 1) * hLen$)
- Выходной параметр: *DK* — сгенерированный ключ длины *dkLen*

Ход вычислений:

1. *l* — количество блоков длины *hLen* в ключе (округление вверх), *r* — количество байт в последнем блоке:
2. Для каждого блока применить функцию *F* с параметрами *P* — мастер пароль, *S* — соль, *c* — количество итераций, и номером блока *F* определена как операция *xor* () над первыми *c* итерациями функции PRF, примененной к

паролю P и объединению соли S и номеру блока, записанному как 4-байтовое целое с первым msb байтом.

3. Объединение полученных блоков составляет ключ DK . От последнего блока берется r байт.

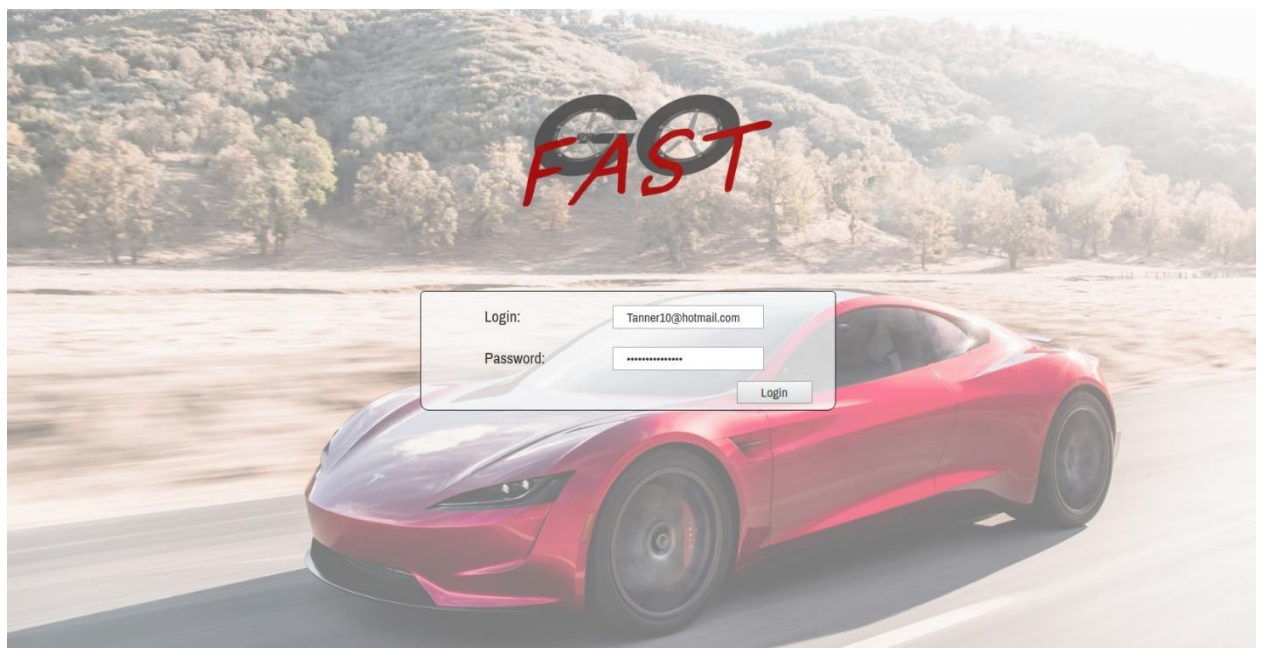
Одной из задач при создании PBKDF2 было усложнить перебор паролей. Благодаря множеству зацепленных вычислений PRF скорость генерации ключа является небольшой. Например, для WPA-PSK с параметрами

$$DK = PBKDF2(HMAC - SHA1, passphrase, ssid, 4096, 256)$$

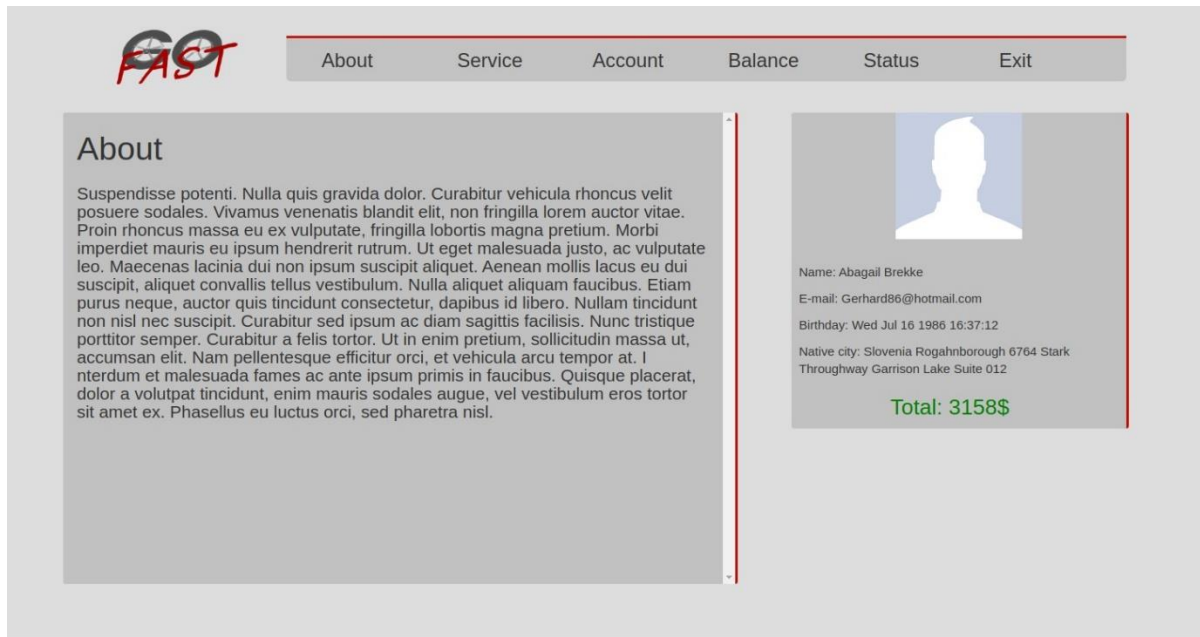
были достигнуты скорости перебора ключей 70 штук в секунду для Intel Core2 и около 1 тысячи на ПЛИС Virtex-4 FX60. Для сравнения классические функции хеширования пароля LANMAN имеют скорость перебора около сотен миллионов вариантов в секунду.

Ниже представлены скриншоты работы приложения:

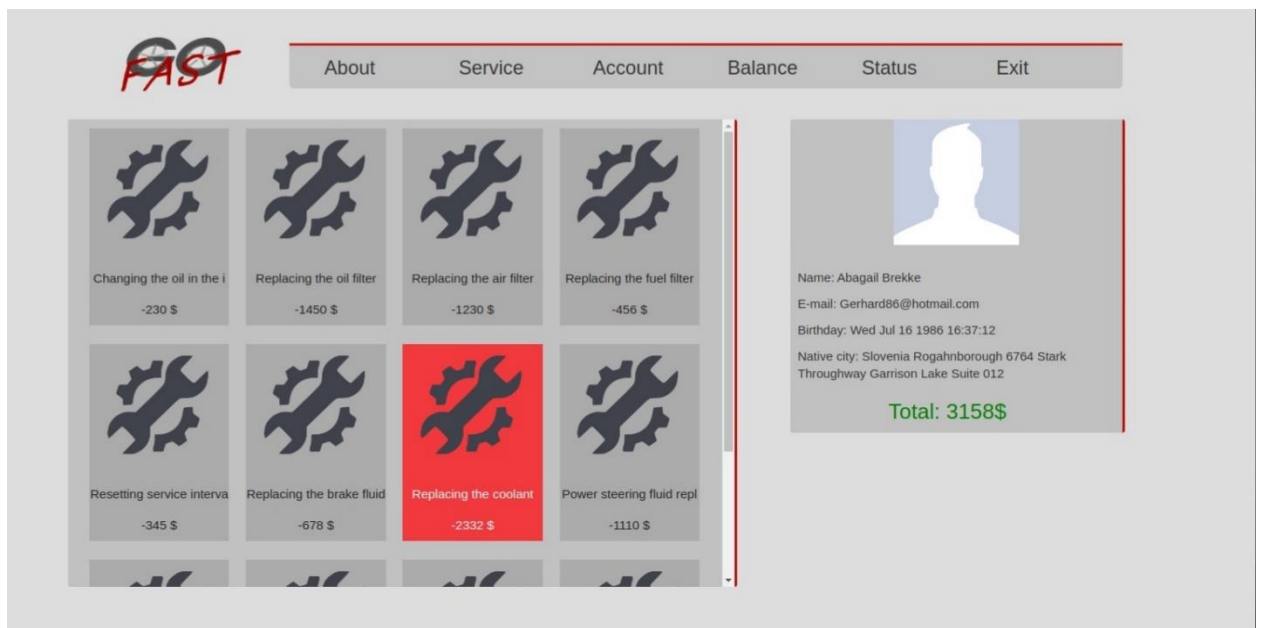
Логинация:



Меню пользователя – после нажатия кнопки «About» (рыба-текст)



Меню пользователя - после нажатия кнопки «Service» (выбор услуг):



Выбор услуги – каждая услуга (отдельная кнопка):

The screenshot shows the 'FAST' service selection interface. At the top, there is a navigation bar with links: About, Service, Account, Balance, Status, and Exit. The main area displays a grid of service options, each with a wrench and gear icon, a description, and a price. The 'Replacing the coolant' option is highlighted in red. To the right, a user profile section shows the name 'Abagail Brekke', email 'Gerhard86@hotmail.com', birthday 'Wed Jul 16 1986 16:37:12', and native city 'Slovenia Rogahnborough 6764 Stark Throughway Garrison Lake Suite 012'. The total amount is displayed as 'Total: 3158\$'.

Service	Price
Changing the oil in the i	-230 \$
Replacing the oil filter	-1450 \$
Replacing the air filter	-1230 \$
Replacing the fuel filter	-456 \$
Resetting service interva	-345 \$
Replacing the brake fluid	-678 \$
Replacing the coolant	-2332 \$
Power steering fluid repl	-1110 \$

User Profile:

- Name: Abagail Brekke
- E-mail: Gerhard86@hotmail.com
- Birthday: Wed Jul 16 1986 16:37:12
- Native city: Slovenia Rogahnborough 6764 Stark Throughway Garrison Lake Suite 012
- Total: 3158\$

Форма заполнения заказа:

The screenshot shows the 'FAST' payment form. At the top, there is a navigation bar with links: About, Service, Account, Balance, Status, and Exit. The main area displays a form for entering payment details. The 'Deutsche Bank' logo is visible. The form includes fields for Card number (**** * 5705), Phone (255.498.2375 x94169), Name order (Replacing the coolant), and To pay (2332 \$). A 'Pay' button is located at the bottom. To the right, a user profile section shows the name 'Abagail Brekke', email 'Gerhard86@hotmail.com', birthday 'Wed Jul 16 1986 16:37:12', and native city 'Slovenia Rogahnborough 6764 Stark Throughway Garrison Lake Suite 012'. The total amount is displayed as 'Total: 3158\$'.

Deutsche Bank

Card: **** * 5705

Phone: 255.498.2375 x94169

Name order: Replacing the coolant

To pay: 2332 \$

Pay

User Profile:

- Name: Abagail Brekke
- E-mail: Gerhard86@hotmail.com
- Birthday: Wed Jul 16 1986 16:37:12
- Native city: Slovenia Rogahnborough 6764 Stark Throughway Garrison Lake Suite 012
- Total: 3158\$


Состояние аккаунта после оплаты услуги:

The screenshot shows the FAST website's account status page. At the top left is the FAST logo. A navigation bar contains links: About, Service, Account, Balance, Status, and Exit. The main content area is split into two panels. The left panel is a large, empty gray rectangle. The right panel features a user profile with a placeholder silhouette, the name 'Abigail Brekke', email 'Gerhard86@hotmail.com', birthday 'Wed Jul 16 1986 16:37:12', and address 'Slovenia Rogahnborough 6764 Stark Throughway Garrison Lake Suite 012'. At the bottom of this panel, the total balance is displayed as 'Total: 826\$' in green text.

Форма изменения данных пользователя с валидацией:

The screenshot shows the FAST website's user information change form. The layout is identical to the previous screenshot, but the left panel now contains a form with the following fields: Name (Abaglai Breke), E-mail (Gfrhard87@mailhot.ru), Birthday (M: 8, D: 17, Y: 1962), Native city (Slovenia Rogahnborough 64 Stark Th), Password (masked with asterisks), and Re-password (masked with asterisks). Each field has a green 'OK!' status indicator to its right. A 'Change Info' button is located at the bottom right of the form panel. The right panel remains unchanged, showing the user profile and the total balance of 'Total: 826\$'.

После применения изменений:

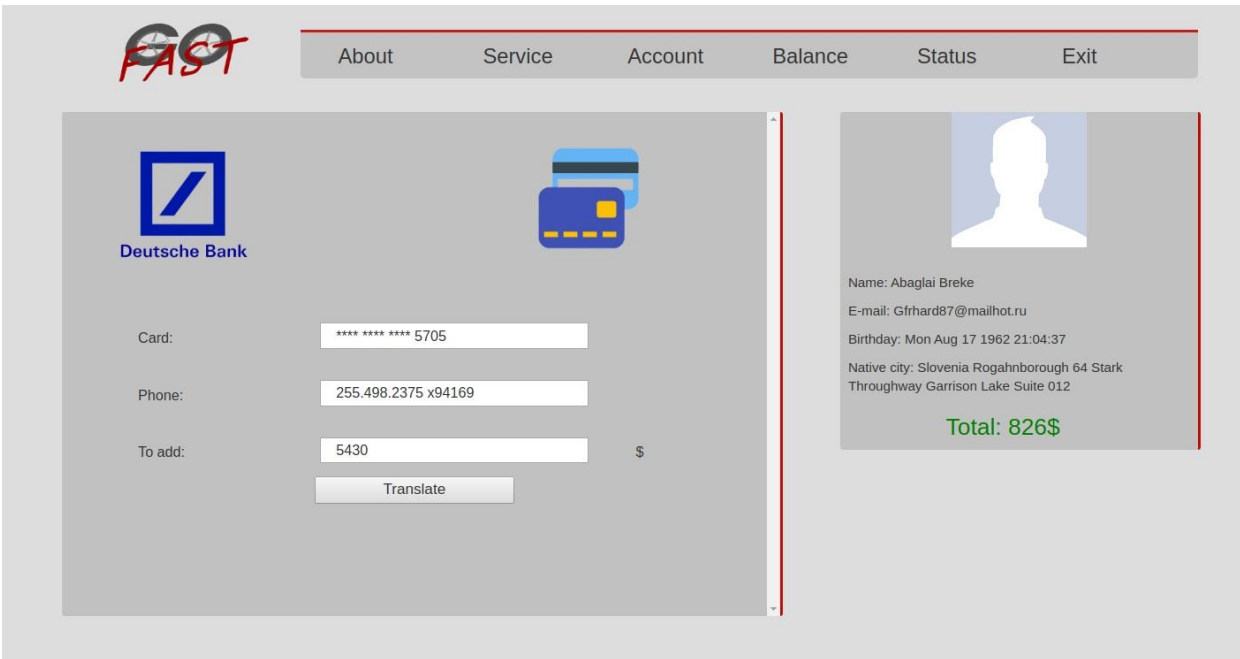


A user profile card with a grey background and a red vertical line on the right. At the top is a blue square placeholder for a profile picture. Below it, the following text is displayed:

Name: Abaglai Breke
E-mail: Gfrhard87@mailhot.ru
Birthday: Mon Aug 17 1962 21:04:37
Native city: Slovenia Rogahnborough 64 Stark
Throughway Garrison Lake Suite 012

At the bottom, the total amount is shown in green text: **Total: 826\$**

Фейковая форма пополнения счета пользователя:




A screenshot of a web application interface for a fake account top-up. The interface has a grey background with a red header bar. The header bar contains the logo "GO FAST" on the left and a navigation menu with the following items: About, Service, Account, Balance, Status, and Exit. The main content area is divided into two sections. The left section is a form for top-up, featuring the Deutsche Bank logo and a credit card icon. The form fields are:

Card:
Phone:
To add: \$

Below the fields is a "Translate" button. The right section is a user profile card, identical to the one in the first image, showing the user's name, email, birthday, native city, and total amount of 826\$.

Результат пополнения счета:

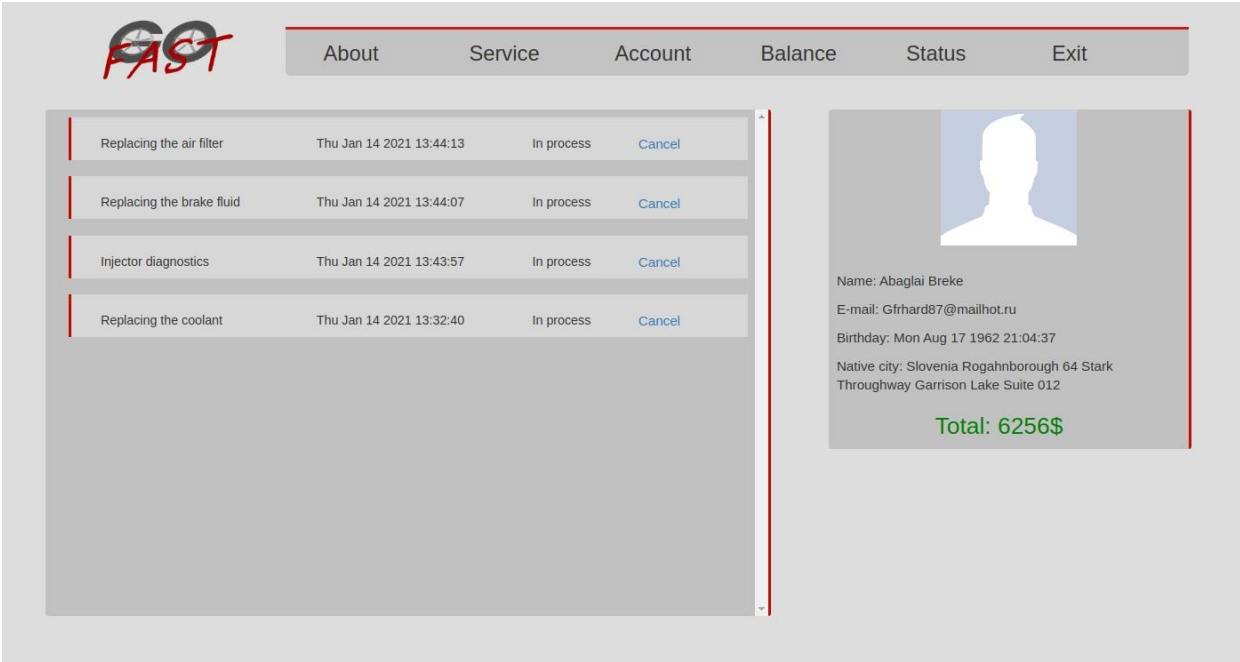


A user profile card with a grey background and a red vertical line on the right. At the top is a blue square placeholder for a profile picture. Below it, the following text is displayed:

Name: Abaglai Breke
E-mail: Gfrhard87@mailhot.ru
Birthday: Mon Aug 17 1962 21:04:37
Native city: Slovenia Rogahnborough 64 Stark
Throughway Garrison Lake Suite 012

Total: 6256\$

После нажатия кнопки «Status» (статус заказа) и приобретение нескольких услуг:



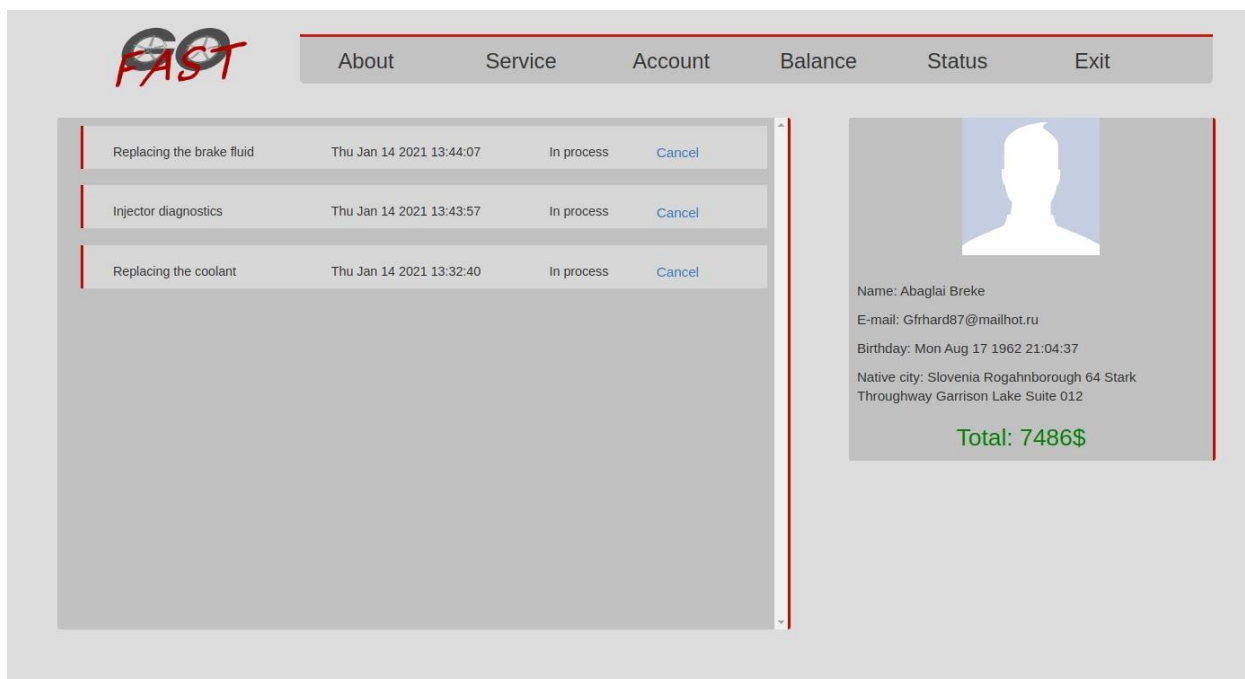
A screenshot of a web application interface. At the top left is the logo "GO FAST" in red. To its right is a navigation bar with links: "About", "Service", "Account", "Balance", "Status", and "Exit". The "Status" link is highlighted. Below the navigation bar is a table with four rows of service records. To the right of the table is a user profile card, identical to the one in the previous image.

Replacing the air filter	Thu Jan 14 2021 13:44:13	In process	Cancel
Replacing the brake fluid	Thu Jan 14 2021 13:44:07	In process	Cancel
Injector diagnostics	Thu Jan 14 2021 13:43:57	In process	Cancel
Replacing the coolant	Thu Jan 14 2021 13:32:40	In process	Cancel

User profile card details:

Name: Abaglai Breke
E-mail: Gfrhard87@mailhot.ru
Birthday: Mon Aug 17 1962 21:04:37
Native city: Slovenia Rogahnborough 64 Stark
Throughway Garrison Lake Suite 012
Total: 6256\$

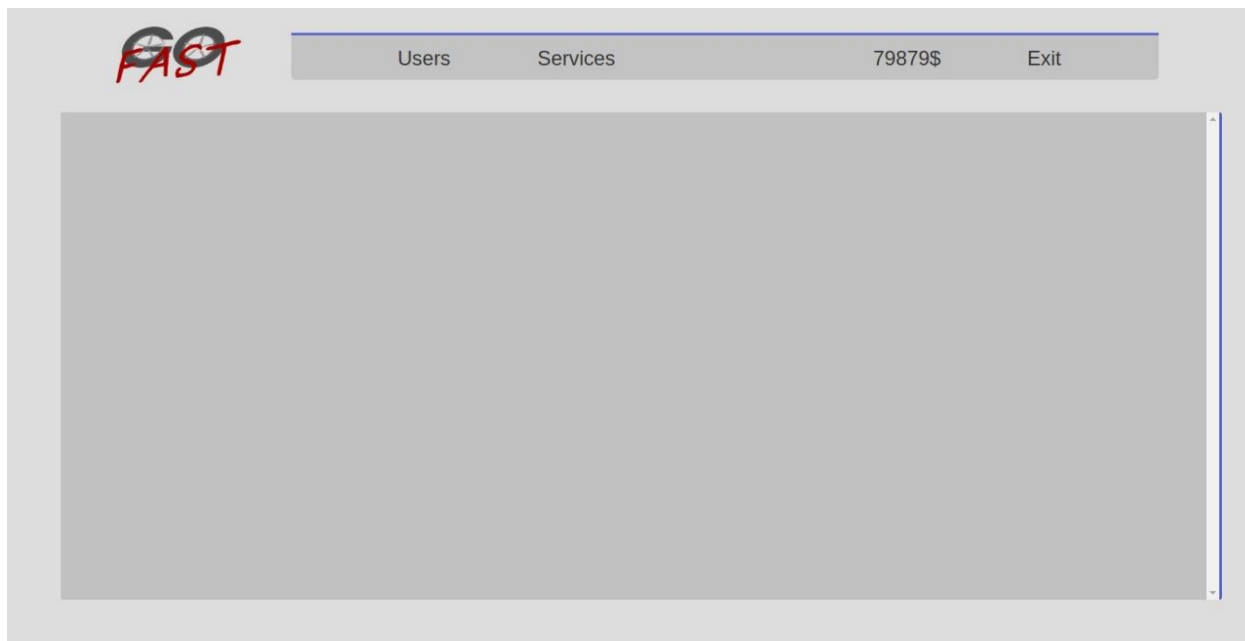
После нажатия «Cancel» (отмена заказа):



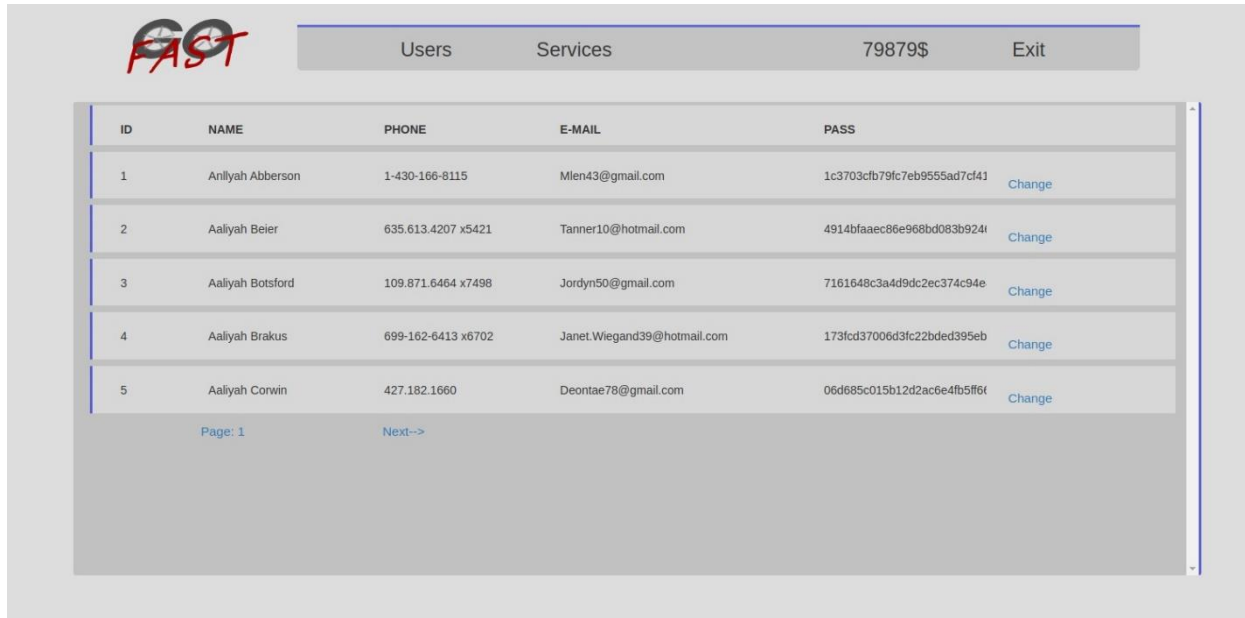
Логинация админа:



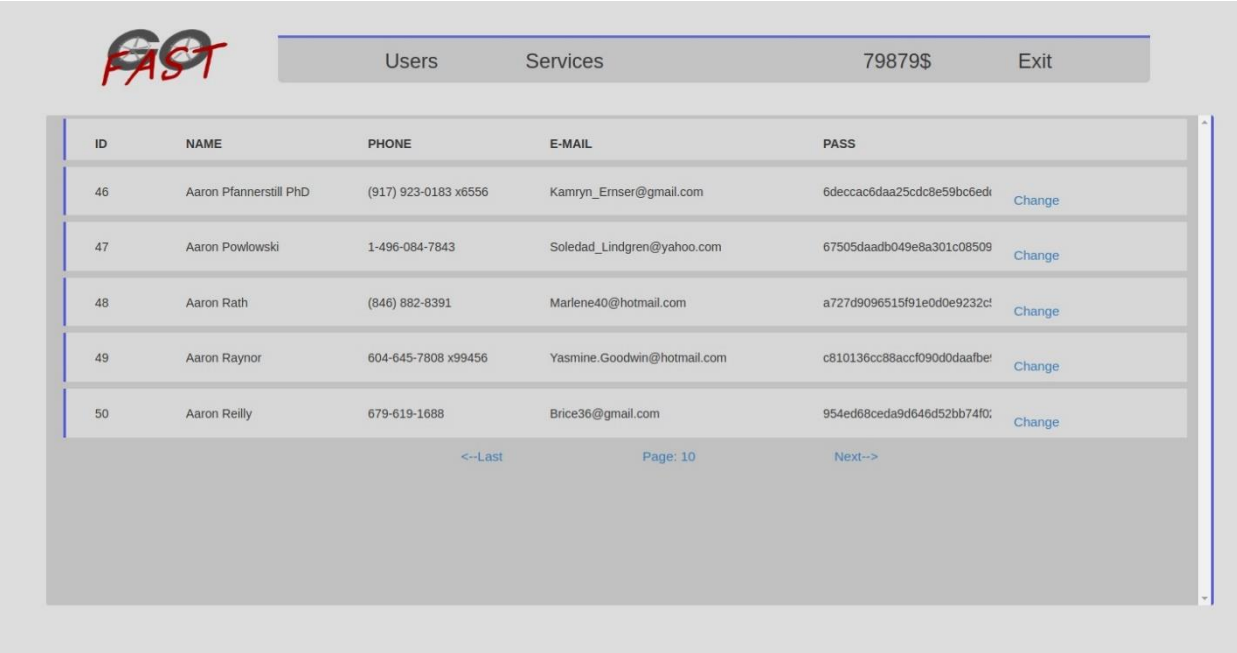
Начальный экран:



После нажатия кнопки «Users»:



Демонстрация пагинации (управление за счет кнопок «Last»/«Next» + отображение текущей страницы):

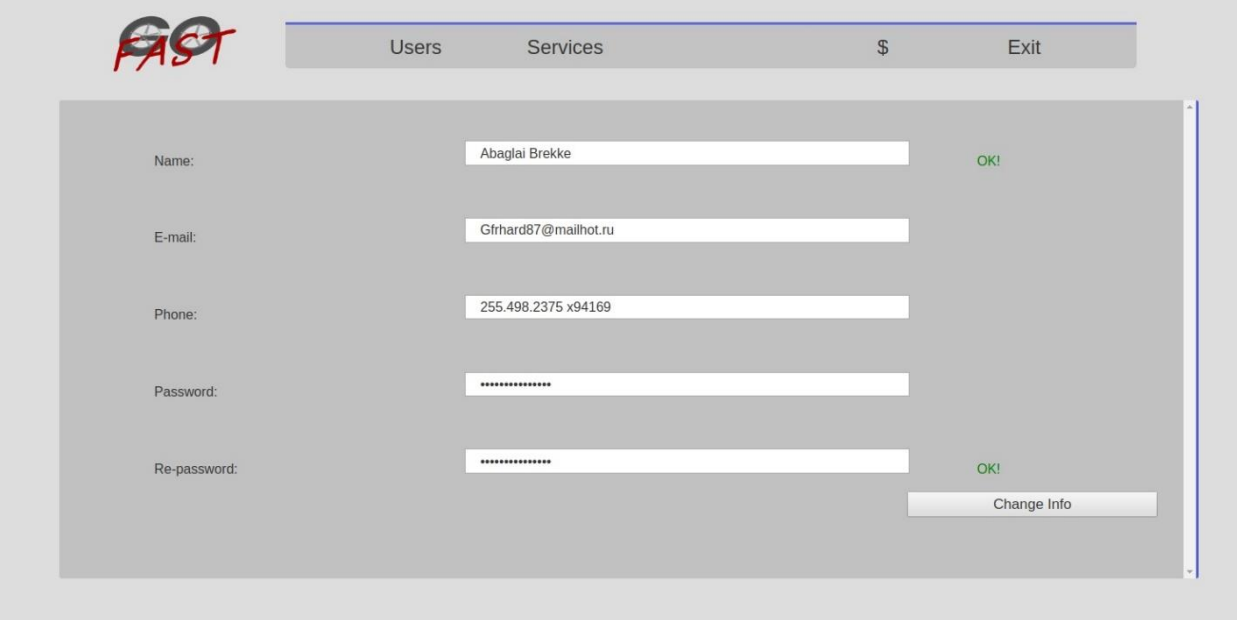


The screenshot shows the FAST application interface. At the top left is the FAST logo. A navigation bar contains 'Users', 'Services', '79879\$', and 'Exit'. Below this is a table with 5 columns: ID, NAME, PHONE, E-MAIL, and PASS. The table contains 5 rows of user data. At the bottom of the table are pagination controls: '<--Last', 'Page: 10', and 'Next-->'. Each row in the table has a 'Change' link to its right.

ID	NAME	PHONE	E-MAIL	PASS	
46	Aaron Pfannerstill PhD	(917) 923-0183 x6556	Kamryn_Erner@gmail.com	6deccac6daa25cdc8e59bc6edi	Change
47	Aaron Powlowski	1-496-084-7843	Soledad_Lindgren@yahoo.com	67505daadb049e8a301c08509	Change
48	Aaron Rath	(846) 882-8391	Marlene40@hotmail.com	a727d9096515f91e0d0e9232ct	Change
49	Aaron Raynor	604-645-7808 x99456	Yasmine.Goodwin@hotmail.com	c810136cc88accf090d0daafbe	Change
50	Aaron Reilly	679-619-1688	Brice36@gmail.com	954ed68ceda9d646d52bb74f0	Change

<--Last Page: 10 Next-->

После нажатия кнопки «Change» для пользователя, у которого id_user = 60, и изменение информации пользователя:



The screenshot shows the FAST application interface with the user information update form. The navigation bar is the same as in the previous screenshot, but the balance is now '\$'. The form has five input fields: Name, E-mail, Phone, Password, and Re-password. The Name field contains 'Abaglai Brekke', the E-mail field contains 'Gfrhard87@mailhot.ru', and the Phone field contains '255.498.2375 x94169'. The Password and Re-password fields are masked with asterisks. There are 'OK!' labels next to the Name and Re-password fields. A 'Change Info' button is at the bottom right.

Name: OK!


E-mail:

Phone:

Password:


Re-password: OK!

После нажатия кнопки «Change info»:



Users		Services		79879\$	Exit
ID	NAME	PHONE	E-MAIL	PASS	
56	Aaron Thompson	355-160-5699	Manley.Brekke97@yahoo.com	7d8c69840018c85aa14539e65	Change
57	Aaron Tromp	010-946-6472 x3500	Rita94@hotmail.com	207adad1fa2179ff534e38117at	Change
58	Abigail Bashirian	557-664-1262 x022	Eldridge_Crist71@yahoo.com	94d2d9e170f40616860c21b56r	Change
59	Abigail Bode	1-996-311-8581	Elvie_Sporer@hotmail.com	aaddf96fad5c44e08bb9eff836c	Change
60	Abaglai Breke	255.498.2375 x94169	Gfrhard87@mailhot.ru	c7d2a7e373301aa1bd4e0cda0	Change
<--Last Page: 12 Next-->					

Демонстрация изменения данных из меню пользователя:



Name: Abaglai Breke


E-mail: Gfrhard87@mailhot.ru

Birthday: Mon Aug 17 1962 21:04:37

Native city: Slovenia Rogahnborough 64 Stark
Throughway Garrison Lake Suite 012

Total: 7486\$


Вывод текущих заказов после нажатия кнопки «Services» (пагинация для заказов):

		Users	Services	79879\$	Exit
ID_USER	NAME	CURR_SERV	CURR_TIME	COST	STATUS
60	Abaglai Breke	Injector diagnostics	Thu Jan 14 2021 13:43:57	432\$	In process Cancel Done
60	Abaglai Breke	Replacing the brake fluid	Thu Jan 14 2021 13:44:07	678\$	In process Cancel Done
2515	Alessandro Hodkiewicz	Replacing the coolant	Thu Jan 14 2021 06:30:13	2332\$	Ready
2515	Alessandro Hodkiewicz	Flushing the injector	Thu Jan 14 2021 06:30:38	980\$	Ready
10	Aaliyah Glover	Power steering fluid repl	Thu Jan 14 2021 08:33:25	1110\$	Ready
<--Last		Page: 2	Next-->		

Удаление текущего заказа:

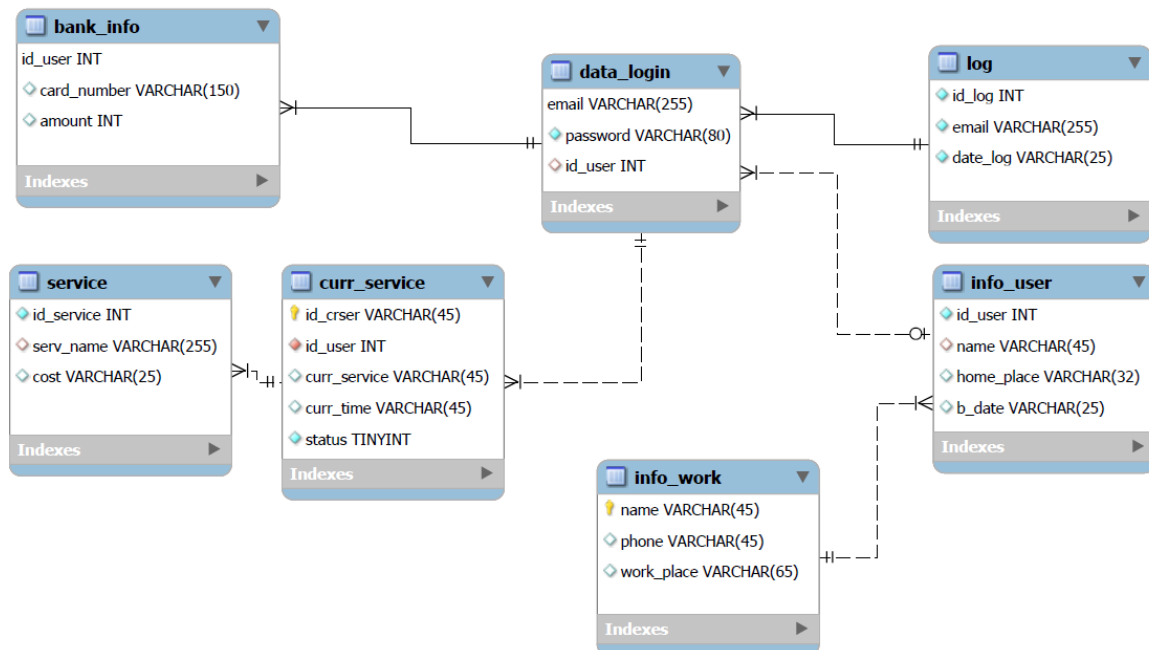
60	Abaglai Breke	Injector diagnostics	Thu Jan 14 2021 13:43:57	432\$	In process	Cancel Done
----	---------------	----------------------	--------------------------	-------	------------	--

После нажатия «Cancel»:

		Users	Services	79879\$	Exit
ID_USER	NAME	CURR_SERV	CURR_TIME	COST	STATUS
60	Abaglai Breke	Replacing the brake fluid	Thu Jan 14 2021 13:44:07	678\$	In process Cancel Done
2515	Alessandro Hodkiewicz	Replacing the coolant	Thu Jan 14 2021 06:30:13	2332\$	Ready
2515	Alessandro Hodkiewicz	Flushing the injector	Thu Jan 14 2021 06:30:38	980\$	Ready
10	Aaliyah Glover	Power steering fluid repl	Thu Jan 14 2021 08:33:25	1110\$	Ready
11	Aaliybh Herman	Replacing the oil filter	Thu Jan 14 2021 11:06:57	1450\$	Ready
<--Last		Page: 2	Next-->		

Приложения

1. ER-диаграмма



2. Исходные коды и документы:

<https://github.com/jkj89507/kursachBd2021>

Вывод

Во время выполнения курсового проекта были изучены методы работы с базами данных, способы управления базы данных с помощью PostgreSQL, познакомились с основами системы контроля версий Git, использование фреймворка Bootstrap3, шаблонизатора Jinga2, работу с виртуальной машиной (VirtualBox + Ubuntu 20.04.1), построение ER-диаграмм с помощью MySQL Workbench.