

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

<b>Введение .....</b>	<b>3</b>
<b>1. Техническое задание .....</b>	<b>4</b>
1.1 Требования к курсовой работе.....	4
1.2 Требования к базе данных.....	4
1.3 Требования к API (минимальное количество реализованных методов) ..	4
<b>2. Порядок выполнения работы .....</b>	<b>5</b>
2.1 Этапы разработки базы данных.....	5-9
2.2 Этапы разработки API .....	10-18
<b>3. Приложения.....</b>	<b>19</b>
3.1 ER-диаграмма .....	19
3.2 Ссылка на github.com.....	19
<b>4. Вывод .....</b>	<b>20</b>

## **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.
- Правомерное 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 **SHA-256**]) 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(25)			
work_place	character varying(45)			

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('/')
@app.route('/login')
def login():
    return render_template('reg.html')

@app.route('/validate', methods=["POST"])
def validate():
    if request.method == "POST":
        global nameUser, mailUser, _b_dateUser, _b_placeUser, __mailUser, __passwordUser, __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()
        answer = user.printCurrEl("data_login WHERE email={}".format(addKav(__mailUser), addKav(__passwordUser)))
        if (len(answer) != 0):
            idUser = answer[0][2]
            answer = user.printCurrEl("info_user WHERE id_user={}".format(__idUser))[0]
            nameUser = answer[1]
            __b_placeUser = answer[2]
            __b_dateUser = answer[3]
            __role = answer[4]
            answer = user.printCurrEl("bank_info WHERE id_user={}".format(__idUser))[0]
            total = answer[2]
            now = getTime()
            mt = dictMounth[str(now.month)]
            day = dictDay[str(now.weekday())]

            user.createElTable("log", (__mailUser, now.strftime("{} {} %d %Y %H:%M:%S").format(day, mt)))
            if __role == "owner":
                user = Control("main", "head", "123456W", "localhost", 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.updateElTable("data_login", "id user="+str(id_user),
            email=addKav(mailUser), password=addKav(passwordUser))
        user.updateElTable("info_user", "id user="+str(id_user),
            name=addKav(nameUser))
        user.updateElTable("info_work", "name={}".format(addKav(nameUser)),
            phone=addKav(phone))
        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" + " " + dictMounth[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.updateElTable("data_login", "id user="+str(__idUser),
        email=addKav(mailUser), password=addKav(__passwordUser))
    user.updateElTable("info_user", "id user="+str(__idUser),
        name=addKav(nameUser), home_place=addKav(__b_placeUser),
        b_date=addKav(_b_dateUser))
    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 = []
```

```
for i in (i for i in user.printEl("service")):
    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 = []
    for i in (i for i in user.printCurrEl("curr_service WHERE id_user={}".format(__idUser), "*", "curr_time DESC")):
        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

```

app.route('/edit/<id>user/', methods=['POST'])
def edit(id user):
    answer = user.print(CurRf("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={}".format(id user),
        "data_login.id_user, info_user.name, info_work.phone, data_login.email, data_login.password", "id user")[0])
    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("adcount.html", dict=helpDict,
        name= nameUser, login= mailUser,
        b date= b dateUser, b place= b placeUser)

```

- Вывод текущих заявок в меню админа

```

app.route('/service', methods=['POST', 'GET'])
def service():
    dictServ = {}
    role = "Owner"
    for i in (1 for i in user.printCurrEl("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 AND curr_service.id_user = info_user.name, curr_service.curr_service, curr_service.curr_time, service.cost, curr_service.status",
    "curr_service.curr_time DESC", 5, curr_pg.st)):
        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.update(helpDict)
    return render_template("adstatus.html", name= nameUser, login= mailUser,
        b date= b dateUser, b place= b placeUser,
        dictDictServ= dictServ, total= curr_pg.printCurrPgSt(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)[1:]), values)) #for pull
        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, ofset=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, ofset))
        array = self.current.fetchall()
        return array

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

```

Данный файл **work\_withBD.py** помогает нам выполнять запросы в приложении, а также автоматизацию заполнения таблиц из файлов формата .CSV:



## pullinfo.py:

```
from werkzeug import *
from cryptography.fernet import Fernet
import hashlib

admin = Control("main", "db creator", "123450", "localhost", 5432)
cipher = Fernet(b'NYrglWwX0HXsabMDuxApVII00X8NXRLSZBbdmNI9nus=')
salt = 'dsdsvs'.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():
    answer = (user.printCurrEl("bank_info WHERE id_user={}".format(_idUser), 'card_number'))[0]
    answer = cipher.decrypt(answer[0].encode('utf-8')).decode('utf-8')
    cardNum = "***** * " + str(answer)[4:]
    answer = user.printCurrEl("info_user RIGHT JOIN info_work ON (info_user.name=info_work.name) WHERE info_user.id_user={}".format(_idUser),
        "info_user.id_user, info_user.name, info_work.phone")
    phone = answer[0][2]
    return render_template("addbalance.html", name= nameUser,
        login= mailUser, b_date= b_dateUser,
        b_place= b_placeUser, money= total,
        card=cardNum, phone=phone)
```

- Изменения данных пользователя

```
@app.route('/edit/<id_user>', methods=["POST"])
def edit(id_user):
    answer = user.printCurrEl("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 info_user.id_user={}".format(id_user), "data_login.id_user, info_user.name, info_work.phone, data_login.email, data_login.password", "id_user")[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.updateElTable("bank info", "id user={}".format(_idUser),
                      amount=str(_total))
    now = getTime()
    mt = dictMounth[str(now.month)]
    day = dictDay[str(now.weekday())]
    user.updateElTable("curr service", "id user={} AND curr service IS NULL AND curr time IS NULL".format(_idUser),
                      "curr service=addKav(ordName), curr_time=addKav(now.strftime("{} {} %d %Y %H:%M:%S").format(day, mt)))
    return redirect(url_for('indexer'))
```

- Статусы заказов пользователя

```
@app.route('/status', methods=["POST", "GET"])
def status():
    dictServ = []
    for i in (i for i in user.printCurrEl("curr_service WHERE id_user={}".format(_idUser), "**", "curr_time DESC")):
        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
    answer = user.printCurrEl("curr_service RIGHT JOIN service ON (curr_service.curr_service = service.serv_name) WHERE id_user = {} AND curr_time = {} AND curr_service = {}".format(_idUser, addKav(ordDate), addKav(ordName)),
                             "curr_service, curr_time, service.cost")[0]
    total += answer[2]
    user.deleteElTable("curr_service", "id user={} AND curr time={} AND curr service={}".format(_idUser, addKav(ordDate), addKav(ordName)))
    user.updateElTable("bank info", "id user={}".format(_idUser), amount=str(_total))
    user.deleteElTable("curr_service", "curr_service IS NULL AND curr_time IS NULL")
    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" + " " + dictMounth[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.updateElTable("data login", "id user="+str(_idUser),
                      email=addKav(_mailUser), password=addKav(_passwordUser))
    user.updateElTable("info user", "id user="+str(_idUser),
                      name=addKav(_nameUser), home_place=addKav(_b_placeUser),
                      b_date=addKav(_b_dateUser))
    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.updateElTable("bank info", "id user={}".format(_idUser),
                      amount=str(_total))
    user.deleteElTable("curr_service", "curr_service IS NULL AND curr_time IS NULL")
    return redirect(url_for('indexer'))
```

- Просмотр админом пользователей с помощью пагинации

```
@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 = []
        answer = user.printCurrEl("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)",
                                "data_login.id_user, info_user.name, info_work.phone, data_login.email, data_login.password", "id_user", 5, curr_pg_user)
        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)
```

- Изменения пользователя админом

```
@app.route('/edit/<id_user>', methods=["POST"])
def edit(id_user):
    answer = user.printCurrEl("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={}".format(id_user),
                            "data_login.id_user, info_user.name, info_work.phone, data_login.email, data_login.password", "id_user")[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("adcount.html", dict=helpDict,
                            name=__nameUser__, login=__mailUser__,
                            b_date=__b_dateUser__, b_place=__b_placeUser__)
```

- Статус отмена/выполнено на стороне админа

```
@app.route('/done/<idUser>/<ordName>/<ordDate>', methods=["POST"])
def done(idUser, ordName, ordDate):
    global __total__
    if __role__ == "owner":
        answer = user.printCurrEl("curr_service RIGHT JOIN service ON (curr_service.curr_service = service.serv_name) WHERE id_user = {} AND curr_time = {} AND curr_service = {}".format(idUser, addKav(ordDate), addKav(ordName)),
                                "curr_service, curr_time, service.cost")[0]
        plus = answer[2]
        user.updateElTable("curr_service", "id_user={}".format(idUser) AND curr_time={}".format(idUser, addKav(ordDate), addKav(ordName)), status=addKav('TRUE'))
        __total__ += plus
        user.updateElTable("bank info", "id_user={}".format(idUser), amount=+str(__total__))
        user.deleteElTable("curr_service", "curr_service IS NULL AND curr_time IS NULL")
        return redirect(url_for('service'))

@app.route('/cancel/<idUser>/<ordName>/<ordDate>', methods=["POST"])
def cancel(idUser, ordName, ordDate):
    global __total__
    if __role__ == "owner":
        answer = user.printCurrEl("curr_service RIGHT JOIN service ON (curr_service.curr_service = service.serv_name) WHERE id_user = {} AND curr_time = {} AND curr_service = {}".format(idUser, addKav(ordDate), addKav(ordName)),
                                "curr_service, curr_time, service.cost")[0]
        minus = answer[2]
        user.deleteElTable("curr_service", "id_user={}".format(idUser) AND curr_time={}".format(idUser, addKav(ordDate), addKav(ordName)))
        answer = user.printCurrEl("bank info WHERE id_user={}".format(idUser))[0]
        sum = answer[2] + minus
        user.updateElTable("bank info", "id_user={}".format(idUser), amount=+str(sum))
        user.deleteElTable("curr_service", "curr_service IS NULL AND curr_time IS NULL")
        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.updateElTable("data_login", "id_user="+str(id_user),
                            email=addKav(mailUser), password=addKav(passwordUser))
        user.updateElTable("info_user", "id_user="+str(id_user),
                            name=addKav(nameUser))
        user.updateElTable("info_work", "name={}".format(addKav(nameUser)),
                            phone=addKav(phone))
        return redirect(url_for('indexer'))
```

При помощи алгоритма SHA256 происходит шифрация паролей. Расшифровка SHA256 – сокращение от Secure Hashing Algorithm – это актуальный на сегодня алгоритм хеширования, созданный National Security Agency – Агентством национальной безопасности США. Задача данного алгоритма заключается в том, чтобы выполнить из случайного набора данных определённые значения с длиной, которая зафиксирована. Эта длина является идентификатором. Значение, которое получится, сравнивается с дубликатами изначальных данных, получить которые нет возможности.

### Что такое алгоритм SHA-256?

Аббревиатуру SHA расшифровывают как «**безопасный расчет хеша**». С помощью данного метода вычислений обеспечивается защита криптографических наборов данных. Ведь без специального кода, который есть только у владельца, невозможно получить доступ к зашифрованной информации.

Алгоритм SHA-2, подвидом которого является SHA-256, был разработан в начале третьего тысячелетия Агентством Национальной Безопасности США. Число 256 обозначает количество фрагментов, из которых состоит данный криптографический код

Через несколько лет после выхода Агентство запатентовало второй выпуск алгоритма SHA-2 под лицензией **Royalty-free**, благодаря чему технологию можно было направить в мирное русло.

### Технические параметры SHA-256:

- Объем блока информации: 64 байт;
- Допустимая длина одного сообщения: 33 байт;
- Размер хеш-подписи блока: 32 байт;
- Число смешиваний в раунде: 64;
- Скорость передачи данных по сети: около 140 MiB/s.
- Плюсы и минусы алгоритма

SHA256 имеет некие преимущества перед другими алгоритмами. Это наиболее востребованный алгоритм майнинга среди всех существующих. Он показал себя как надежный к взламыванию (случается не часто) и результативный алгоритм как для задач майнинга, так и для прочих целей.

Имеются и недостатки:

- Главным минусом SHA256 валюты является контролирование майнерами.
- Те, у кого имеются огромные вычислительные мощности, получают основную часть крипто, что исключает один из основных принципов виртуальных денег – децентрализованность.
- Как только пошли инвестиции в вычислительные мощности для промышленного майнинга Биткоина, сложность добычи значительно возросла и стала требовать исключительных вычислительных мощностей. Этот минус исправлен в прочих протоколах, наиболее инновационных и «заточенных» под применение в добыче цифровых валют, таких как Script (для криптовалюты Litecoin).

Криптовалюты SHA256, как и SHA512 наиболее защищены от данного отрицательного момента, но вероятность развития риска все-таки есть. Miner на SHA256, как и на любом ином хешировании – это процесс разрешения какой-то сложнейшей криптографической задачи, которую генерирует программа для майнинга на основе информации, полученной с блоков.

Майнинг при помощи хэш-функции SHA256 можно осуществлять 3 методами:

- CPU.
- GPU.
- ASIC.

В майнинге хеш–сумма применяется как идентификатор уже присутствующих блоков, и создания новых на основе тех, что имеются. Процесс майнинга отражен в интерфейсе в виде «accepted f33ae3bc9...». Где f33ae3bc9 – это хешированная сумма, часть данных, которая требуется для дешифрования. Главный блок включает в себя огромное число такого рода хеш-сумм.

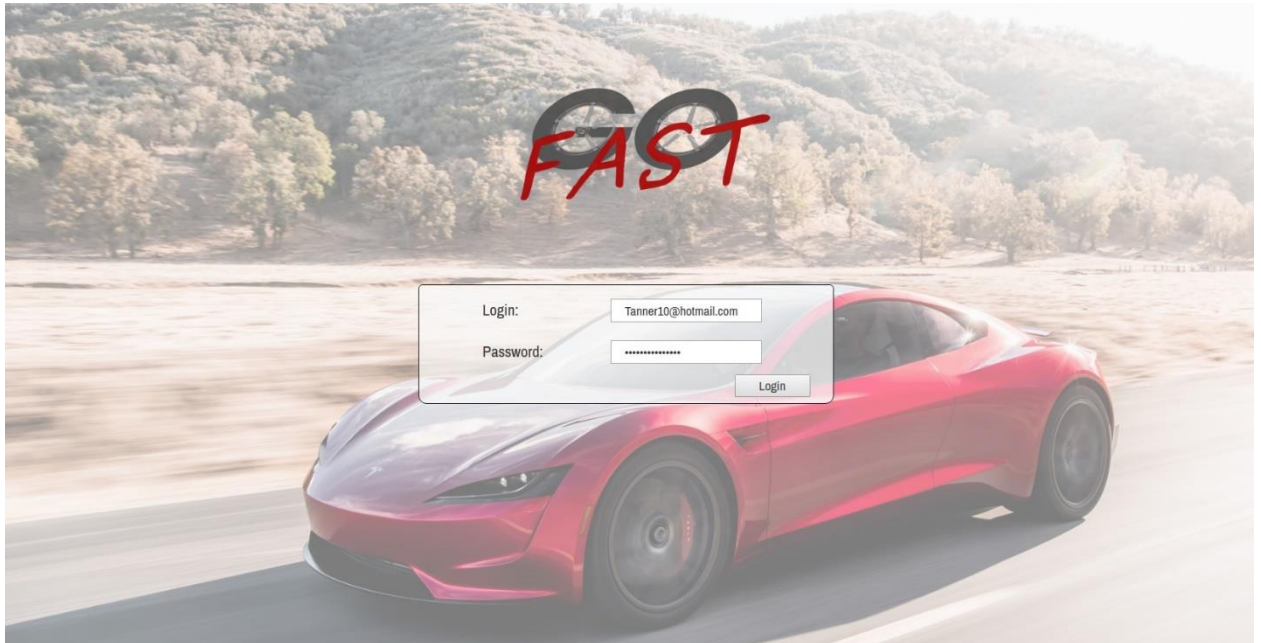
То есть, добыча с алгоритмом SHA256 – это подбор правильного значения хешированной суммы без остановки, перебор чисел для того, чтобы создать очередной блок. Чем мощнее оборудование, тем больше шансов стать владельцем того самого правильного блока: скорость перебирания разного рода сумм зависит от мощностей. Потому как Биткоин построен на алгоритме SHA256, для конкурентоспособного майнинга на нём требуются крайне большие вычислительные мощности.

Это связывается с тем, что для добычи криптовалюты хватает производства «асиков», а именно специальной схемы особенного назначения. Асики дают возможность добывать Биткоины и прочие криптовалюты на хэш-функции SHA–256 оперативнее, результативнее и недорого.

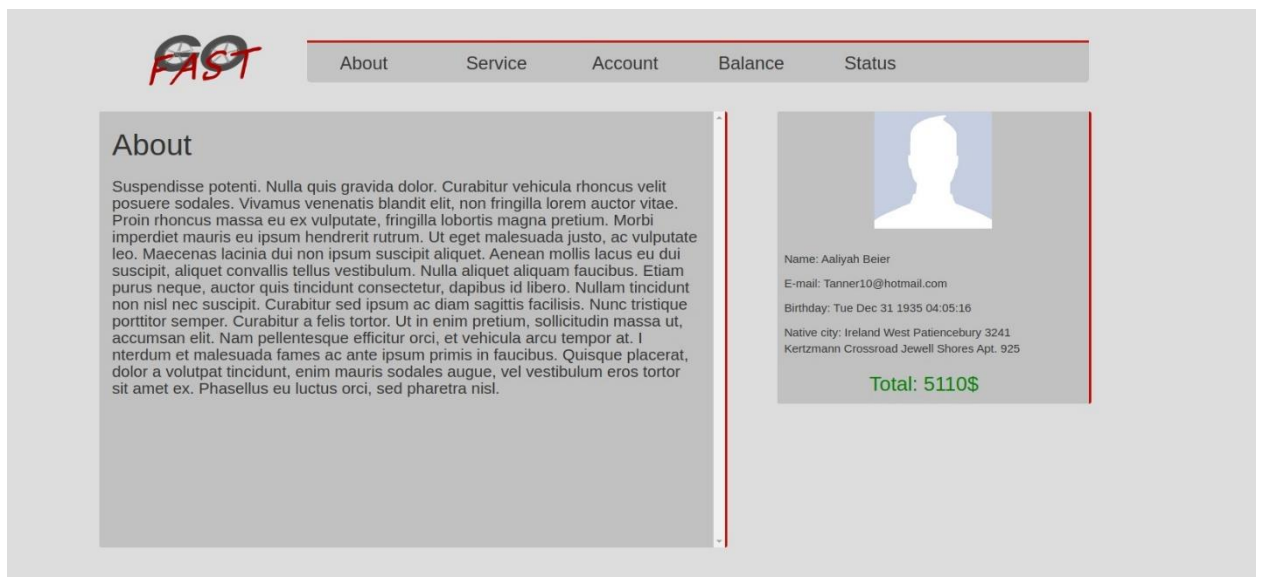


Ниже представлены скриншоты работы приложения:

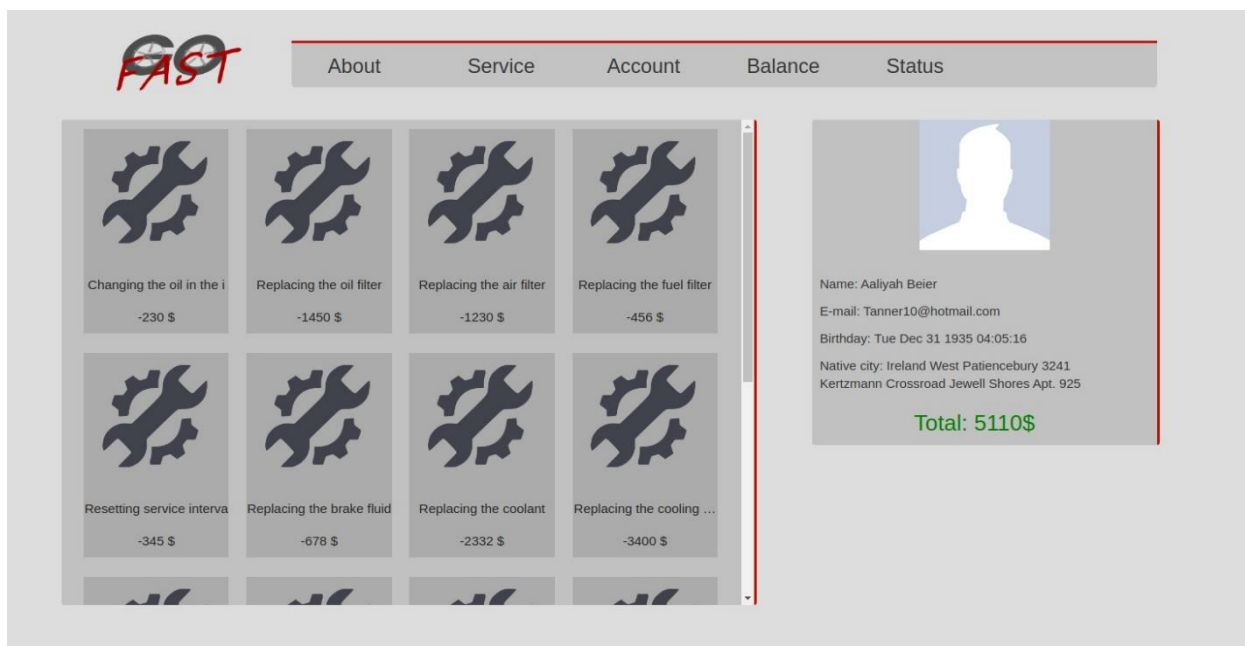
Логинация:



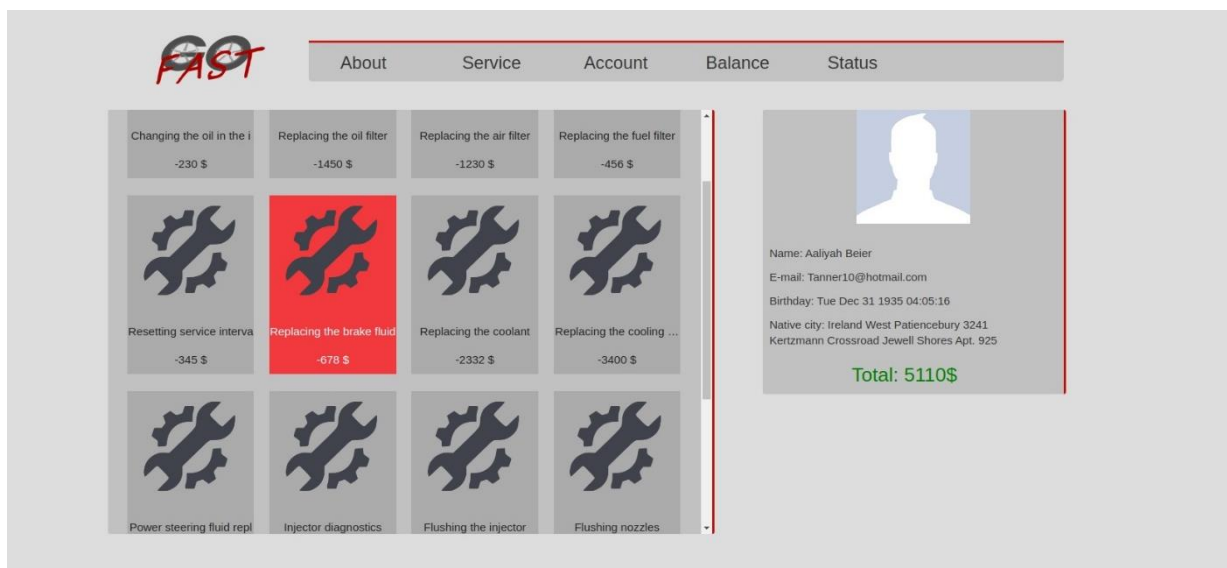
Меню пользователя – после нажатия кнопки «About» (рыба-текст)



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



Выбор услуги – каждая услуга (отдельная кнопка):



## Форма заполнения заказа:

The screenshot shows the FAST website interface. At the top, there is a navigation bar with links: About, Service, Account, Balance, and Status. The main content area is divided into two sections. On the left, there is a Deutsche Bank logo and a credit card icon. Below this, there are input fields for Card (\*\*\*\* \* 8503), Phone (635.613.4207 x5421), Name order (Replacing the coolant), and To pay (2332). A 'Pay' button is located below the 'To pay' field. On the right, there is a user profile section with a silhouette icon and the following information: Name: Aaliyah Beier, E-mail: Tanner10@hotmail.com, Birthday: Tue Dec 31 1935 04:05:16, Native city: Ireland West Patiencebury 3241, Kertzmann Crossroad Jewell Shores Apt. 925. The total amount is displayed as Total: 5110\$.

FAST

About Service Account Balance Status

Deutsche Bank

Card: \*\*\*\* \* 8503

Phone: 635.613.4207 x5421

Name order: Replacing the coolant

To pay: 2332 \$

Pay

Name: Aaliyah Beier  
E-mail: Tanner10@hotmail.com  
Birthday: Tue Dec 31 1935 04:05:16  
Native city: Ireland West Patiencebury 3241  
Kertzmann Crossroad Jewell Shores Apt. 925

Total: 5110\$

## Состояние аккаунта после оплаты услуги:

The screenshot shows the FAST website interface after payment. The navigation bar remains the same. The main content area is now mostly empty, with a large gray rectangle where the order form was previously located. The user profile section on the right remains the same, but the total amount is now displayed as Total: 2778\$.

FAST

About Service Account Balance Status

Name: Aaliyah Beier  
E-mail: Tanner10@hotmail.com  
Birthday: Tue Dec 31 1935 04:05:16  
Native city: Ireland West Patiencebury 3241  
Kertzmann Crossroad Jewell Shores Apt. 925

Total: 2778\$

Форма изменения данных пользователя с валидацией:

**GO FAST**

About Service Account Balance Status

Name: Balialiyah Veier OK!

E-mail: Tanner56@maihot.com OK!

Birthday: M: 10 D: 18 Y: 1968

Native city: Ireland West Patiencebury 3451 Kertz OK!

Password: \*\*\*\*\*

Re-password: \*\*\*\*\* OK!

Change Info

Name: Aaliyah Beier

E-mail: Tanner10@hotmail.com

Birthday: Tue Dec 31 1935 04:05:16

Native city: Ireland West Patiencebury 3241 Kertzmann Crossroad Jewell Shores Apt. 925

Total: 2778\$

После применения изменений:

Name: Balialiyah Veier

E-mail: Tanner56@maihot.com

Birthday: Mon Oct 18 1968 21:04:37

Native city: Ireland West Patiencebury 3451 Kertzmann Crossroad Jewell Shores Apt. 925

Total: 2778\$

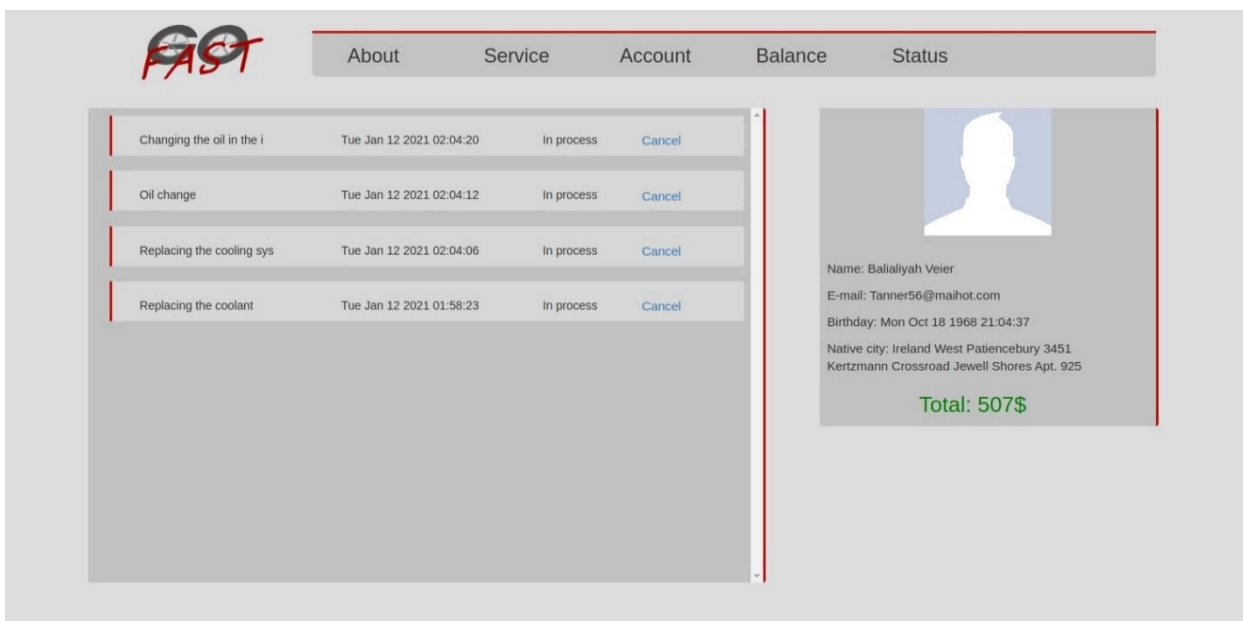
Фейковая форма пополнения счета пользователя:

The screenshot shows a web interface for a service called "FAST". At the top, there is a navigation bar with links: "About", "Service", "Account", "Balance", and "Status". The main content area is divided into two sections. The left section features the "Deutsche Bank" logo and a credit card icon. Below these, there are input fields for "Card:" (containing "\*\*\*\* \* 8503"), "Phone:" (containing "635.613.4207 x5421"), and "To add:" (containing "6789"). A "Translate" button is located below the "To add:" field. The right section displays a user profile with a placeholder image, the name "Name: Balialiyah Veier", email "E-mail: Tanner56@mailhot.com", birthday "Birthday: Mon Oct 18 1968 21:04:37", and address "Native city: Ireland West Patiencebury 3451 Kertzmann Crossroad Jewell Shores Apt. 925". At the bottom of this section, the total balance is shown as "Total: 2778\$".

Результат пополнения счета:

The screenshot shows the result of a top-up transaction. It features a large placeholder image of a person's head and shoulders. Below the image, the user's details are listed: "Name: Balialiyah Veier", "E-mail: Tanner56@mailhot.com", "Birthday: Mon Oct 18 1968 21:04:37", and "Native city: Ireland West Patiencebury 3451 Kertzmann Crossroad Jewell Shores Apt. 925". At the bottom, the updated total balance is displayed in green text as "Total: 9567\$".

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



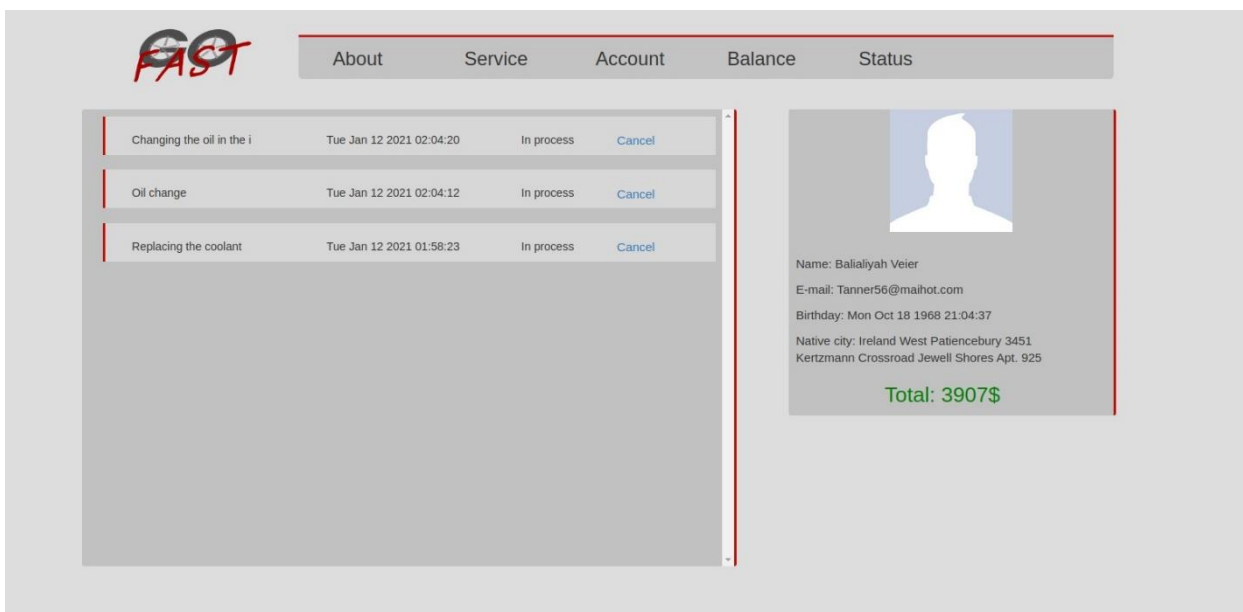
The screenshot shows the FAST website interface. At the top, there is a navigation bar with the FAST logo and links for About, Service, Account, Balance, and Status. The Status link is highlighted. Below the navigation bar, there is a table listing services and their status. To the right of the table, there is a user profile section with a placeholder for a profile picture and personal information.

Service	Date	Status	Action
Changing the oil in the i	Tue Jan 12 2021 02:04:20	In process	<a href="#">Cancel</a>
Oil change	Tue Jan 12 2021 02:04:12	In process	<a href="#">Cancel</a>
Replacing the cooling sys	Tue Jan 12 2021 02:04:06	In process	<a href="#">Cancel</a>
Replacing the coolant	Tue Jan 12 2021 01:58:23	In process	<a href="#">Cancel</a>

User Profile:

Name: Balaliyah Veier  
E-mail: Tanner56@mailhot.com  
Birthday: Mon Oct 18 1968 21:04:37  
Native city: Ireland West Patiencebury 3451  
Kertzmann Crossroad Jewell Shores Apt. 925  
**Total: 507\$**

После нажатия «Cancel» (отмена заказа):



The screenshot shows the FAST website interface after the cancellation of services. The navigation bar and user profile section are the same as in the previous screenshot. However, the table listing services now only shows the 'Replacing the coolant' service, as the others have been cancelled.

Service	Date	Status	Action
Replacing the coolant	Tue Jan 12 2021 01:58:23	In process	<a href="#">Cancel</a>

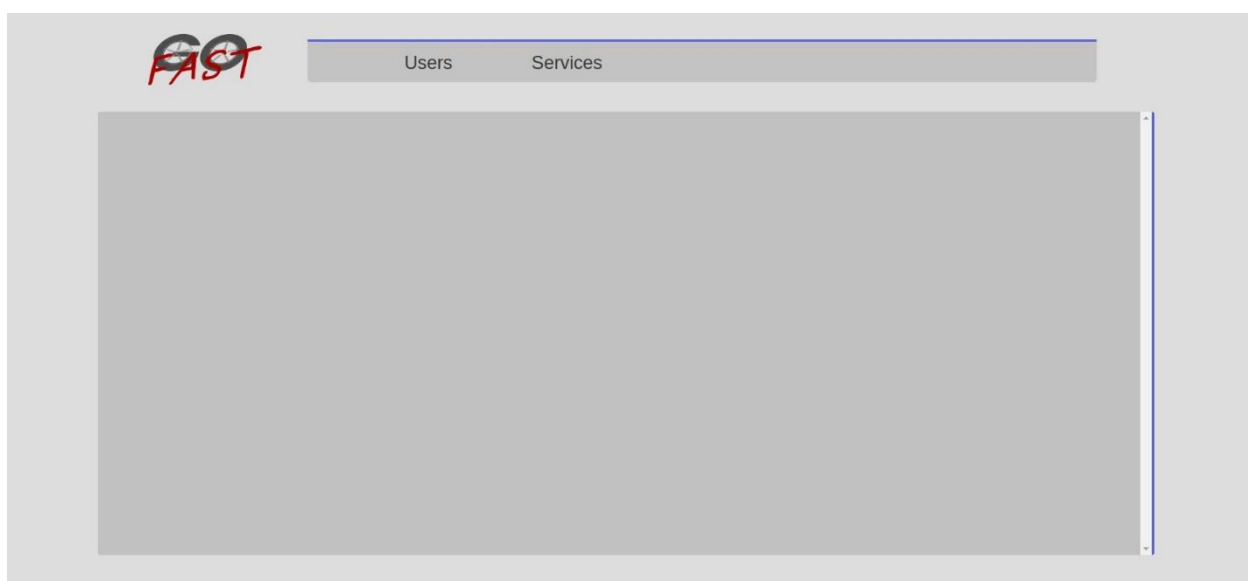
User Profile:

Name: Balaliyah Veier  
E-mail: Tanner56@mailhot.com  
Birthday: Mon Oct 18 1968 21:04:37  
Native city: Ireland West Patiencebury 3451  
Kertzmann Crossroad Jewell Shores Apt. 925  
**Total: 3907\$**

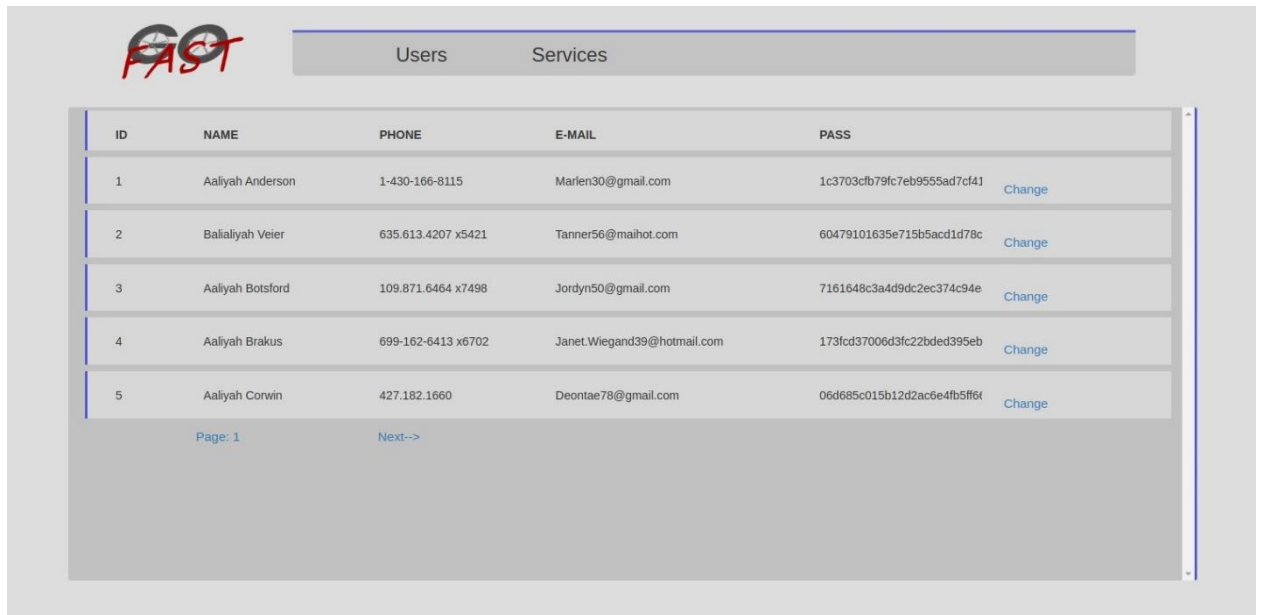
Логинация админа:



Начальный экран:



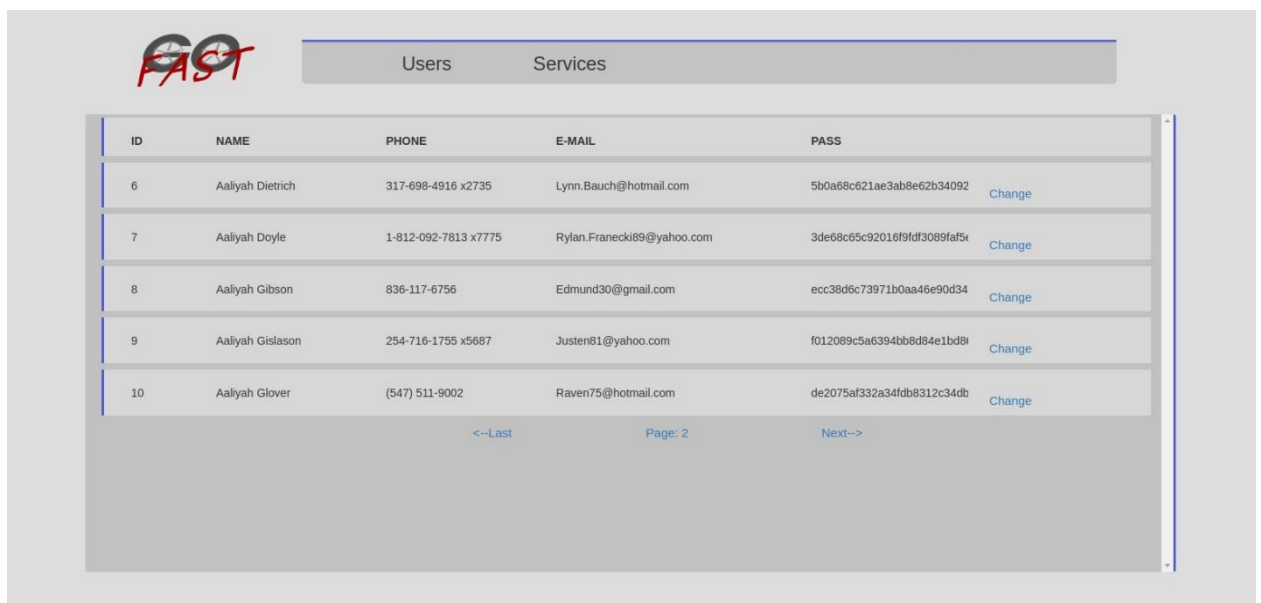
После нажатия кнопки «Users»:



ID	NAME	PHONE	E-MAIL	PASS	
1	Aaliyah Anderson	1-430-166-8115	Marlen30@gmail.com	1c3703cfb79fc7eb9555ad7cf41	<a href="#">Change</a>
2	Baliyah Veier	635.613.4207 x5421	Tanner56@mailhot.com	60479101635e715b5acd1d78c	<a href="#">Change</a>
3	Aaliyah Botsford	109.871.6464 x7498	Jordyn50@gmail.com	7161648c3a4d9dc2ec374c94e	<a href="#">Change</a>
4	Aaliyah Brakus	699-162-6413 x6702	Janet.Wiegand39@hotmail.com	173fcd37006d3fc22bdc395eb	<a href="#">Change</a>
5	Aaliyah Corwin	427.182.1660	Deontae78@gmail.com	06d685c015b12d2ac6e4fb5ff6	<a href="#">Change</a>

Page: 1      Next-->

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




ID	NAME	PHONE	E-MAIL	PASS	
6	Aaliyah Dietrich	317-698-4916 x2735	Lynn.Bauch@hotmail.com	5b0a68c621ae3ab8e62b34092	<a href="#">Change</a>
7	Aaliyah Doyle	1-812-092-7813 x7775	Rylan.Franecki89@yahoo.com	3de68c65c92016f9fdf3089faf5e	<a href="#">Change</a>
8	Aaliyah Gibson	836-117-6756	Edmund30@gmail.com	ecc38d6c73971b0aa46e90d34	<a href="#">Change</a>
9	Aaliyah Gislason	254-716-1755 x5687	Justen81@yahoo.com	f012089c5a6394bb8d84e1bd8f	<a href="#">Change</a>
10	Aaliyah Glover	(547) 511-9002	Raven75@hotmail.com	de2075af332a34fdb8312c34db	<a href="#">Change</a>

<--Last      Page: 2      Next-->



После нажатия кнопки «Change» для пользователя, у которого id\_user = 6, и изменение информации пользователя:



Users

Services

Name:

OK!

E-mail:

Phone:


Password:

Re-password:

OK!

Change Info

После нажатия кнопки «Change info»:



Users

Services

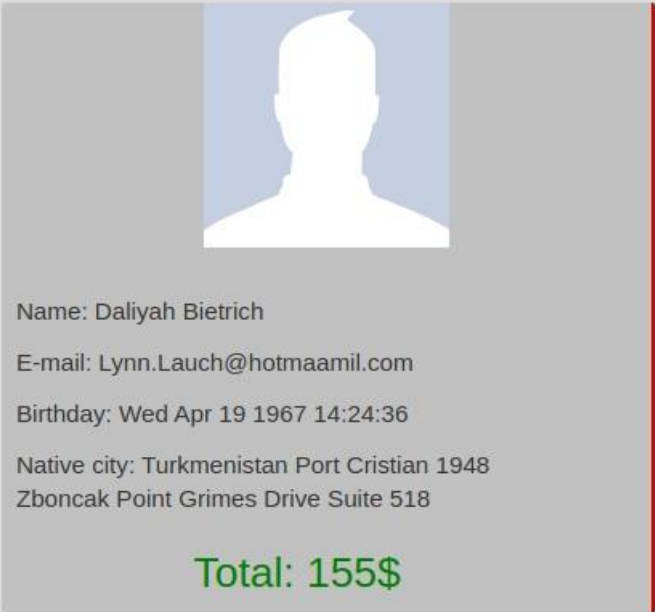
ID	NAME	PHONE	E-MAIL	PASS	
6	Daliyah Bietrich	347-6128-4916 x2745	Lynn.Lauch@hotmail.com	2ace2fdcec8ae1768fa374799f	<a href="#">Change</a>
7	Aaliyah Doyle	1-812-092-7813 x7775	Rylan.Franecki89@yahoo.com	3de68c65c92016f9fd3089faf5	<a href="#">Change</a>
8	Aaliyah Gibson	836-117-6756	Edmund30@gmail.com	ecc38d6c73971b0aa46e90d34	<a href="#">Change</a>
9	Aaliyah Gislason	254-716-1755 x5687	Justen81@yahoo.com	f012089c5a6394bb8d84e1bd8	<a href="#">Change</a>
10	Aaliyah Glover	(547) 511-9002	Raven75@hotmail.com	de2075af332a34fdb8312c34db	<a href="#">Change</a>

<--Last

Page: 2

Next-->

Демонстрация изменения данных из меню пользователя:




A user profile card with a placeholder image of a person. Below the image, the following information is displayed:

Name: Daliyah Bietrich  
E-mail: Lynn.Lauch@hotmail.com  
Birthday: Wed Apr 19 1967 14:24:36  
Native city: Turkmenistan Port Cristian 1948  
Zboncak Point Grimes Drive Suite 518

Total: 155\$

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

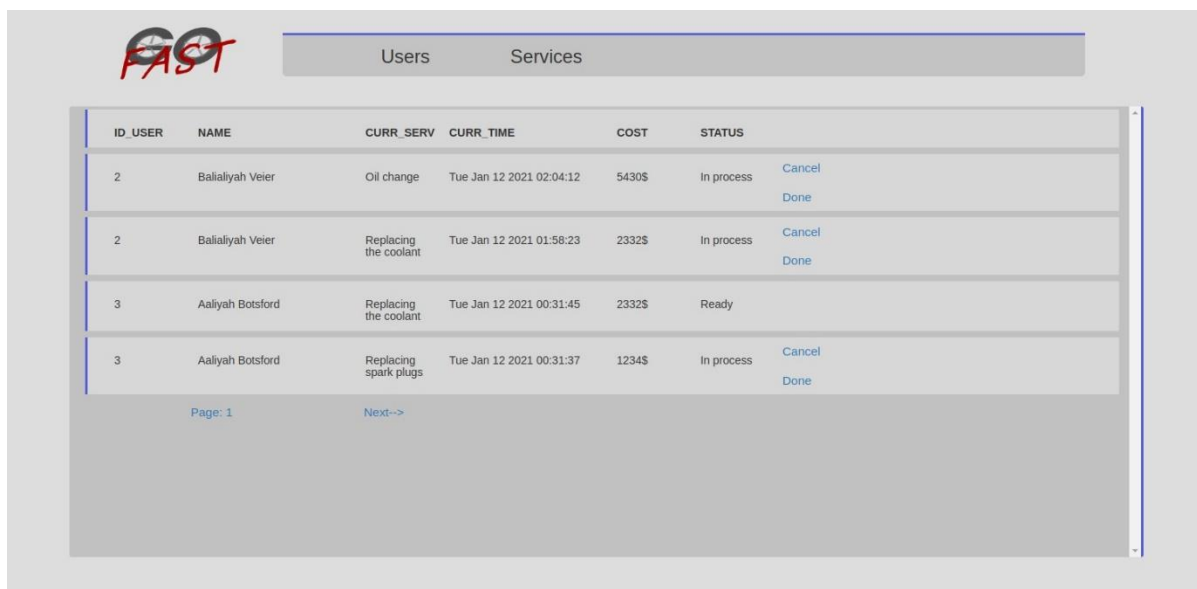


Users Services						
ID_USER	NAME	CURR_SERV	CURR_TIME	COST	STATUS	
2	Balialiyah Veier	Changing the oil in the i	Tue Jan 12 2021 02:04:20	230\$	In process	<a href="#">Cancel</a> <a href="#">Done</a>
2	Balialiyah Veier	Oil change	Tue Jan 12 2021 02:04:12	5430\$	In process	<a href="#">Cancel</a> <a href="#">Done</a>
2	Balialiyah Veier	Replacing the coolant	Tue Jan 12 2021 01:58:23	2332\$	In process	<a href="#">Cancel</a> <a href="#">Done</a>
3	Aaliyah Botsford	Replacing the coolant	Tue Jan 12 2021 00:31:45	2332\$	Ready	
3	Aaliyah Botsford	Replacing spark plugs	Tue Jan 12 2021 00:31:37	1234\$	In process	<a href="#">Cancel</a> <a href="#">Done</a>
Page: 1 Next-->						

Удаление текущего заказа:

2	Balialiyah Veier	Changing the oil in the i	Tue Jan 12 2021 02:04:20	230\$	In process	<a href="#">Cancel</a> <a href="#">Done</a>
---	------------------	---------------------------	--------------------------	-------	------------	--

После нажатия «Cancel»:



FAST

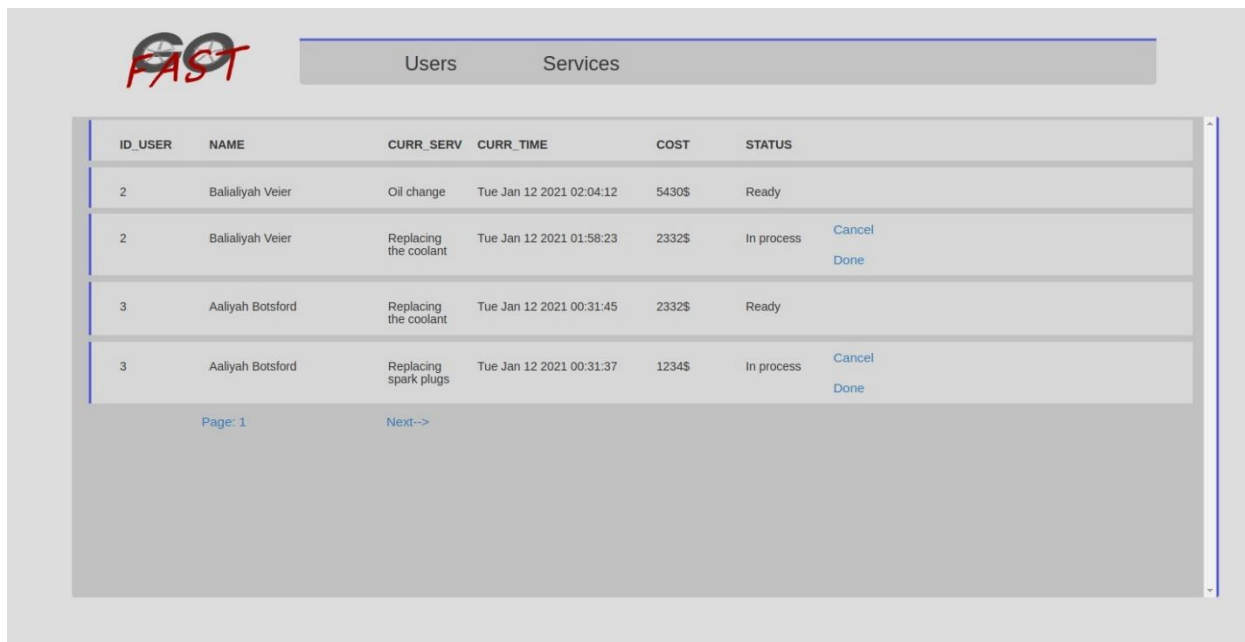
ID_USER	NAME	CURR_SERV	CURR_TIME	COST	STATUS
2	Balialiyah Veier	Oil change	Tue Jan 12 2021 02:04:12	5430\$	In process
2	Balialiyah Veier	Replacing the coolant	Tue Jan 12 2021 01:58:23	2332\$	In process
3	Aaliyah Botsford	Replacing the coolant	Tue Jan 12 2021 00:31:45	2332\$	Ready
3	Aaliyah Botsford	Replacing spark plugs	Tue Jan 12 2021 00:31:37	1234\$	In process

Page: 1 Next-->

Изменим статус текущего заказа с помощью кнопки «Done»:

2	Balialiyah Veier	Oil change	Tue Jan 12 2021 02:04:12	5430\$	In process	Cancel
						Done

Результат:




FAST

ID_USER	NAME	CURR_SERV	CURR_TIME	COST	STATUS
2	Balialiyah Veier	Oil change	Tue Jan 12 2021 02:04:12	5430\$	Ready
2	Balialiyah Veier	Replacing the coolant	Tue Jan 12 2021 01:58:23	2332\$	In process
3	Aaliyah Botsford	Replacing the coolant	Tue Jan 12 2021 00:31:45	2332\$	Ready
3	Aaliyah Botsford	Replacing spark plugs	Tue Jan 12 2021 00:31:37	1234\$	In process


Page: 1 Next-->

## Демонстрация выполненного заказа для пользователя:



[About](#) [Service](#) [Account](#) [Balance](#) [Status](#)

Replacing the coolant	Tue Jan 12 2021 00:31:45	Ready
Replacing spark plugs	Tue Jan 12 2021 00:31:37	In process <a href="#">Cancel</a>

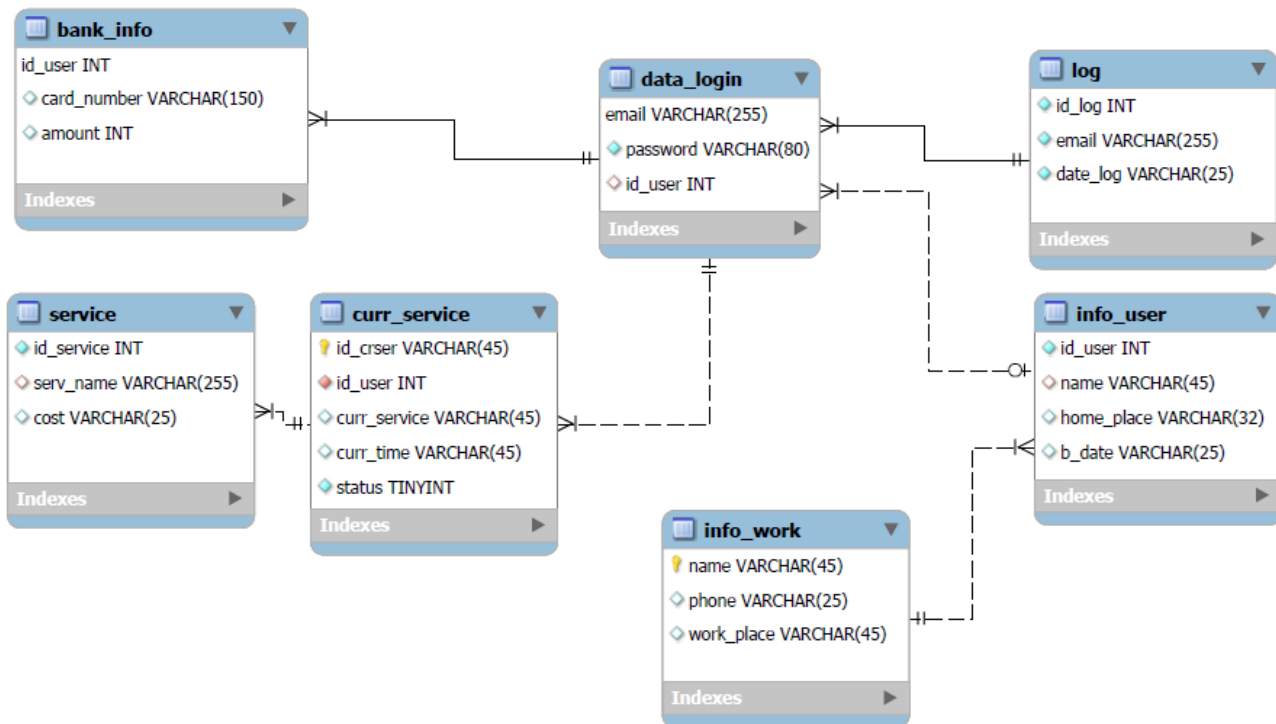


Name: Aaliyah Botsford  
E-mail: Jordyn50@gmail.com  
Birthday: Mon Mar 09 1970 11:41:44  
Native city: Libyan Arab Jamahiriya New  
Waylontown 59315 Bartoletti Islands Gaylord  
Cliff Apt. 201

Total: 1742\$

# 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.