

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.....	5-9
2.2 Этапы разработки API	10-18
3. Приложения.....	19
3.1 ER-диаграмма	19
3.2 Ссылка на github.com.....	19
4. Вывод	20

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 приобретения 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(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, __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 *
                              FROM info user
                              WHERE id_user= %s""",
                           ([__idUser]))
            answer = cursor.fetchall()[0]

            __nameUser = answer[1]
            __b_placeUser = answer[2]
            __b_dateUser = answer[3]
            __role = answer[4]

            cursor.execute("""SELECT *
                              FROM bank info WHERE id_user= %s""",
                           ([__idUser]))
            answer = cursor.fetchall()[0]

            __total = answer[2]
            __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 = 'dsvdsdvs'.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)
```

- 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))

    user.commit()

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

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

    user.commit()
    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'))
```

- 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("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'))
```

- 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))
        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))

        user.commit()
        _total += plus
        user = psycopg2.connect(database="main", user="head", password="123456W", 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="head", password="123456W", 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('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))
        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]))
        answer = cursor.fetchall()[0]

        summ = answer[2] + minus

        cursor.execute("""UPDATE bank info
                            SET amount=%s
                            WHERE id_user=%s""",
                            (summ, 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'))
```


- 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'))
```

При помощи алгоритма SHA256 происходит хеширование 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.

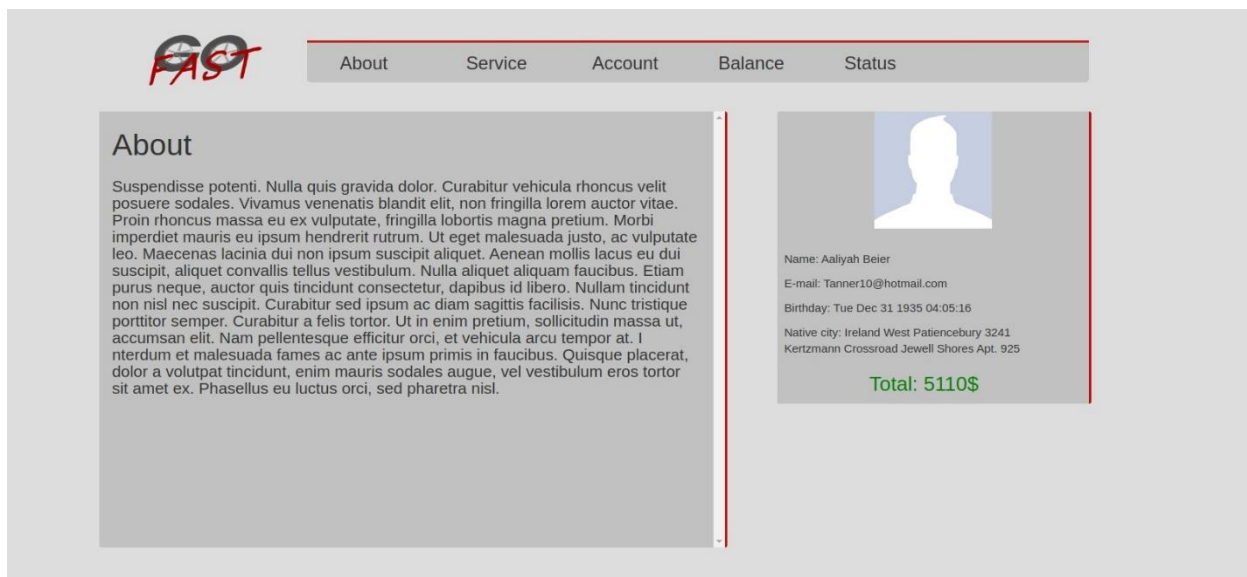
Это связывается с тем, что для добычи криптовалюты хватает производства «асиков», а именно специальной схемы особенного назначения. Асики дают возможность добывать Биткоины и прочие криптовалюты на хэш-функции SHA–256 оперативнее, результативнее и недорого.

Ниже представлены скриншоты работы приложения:

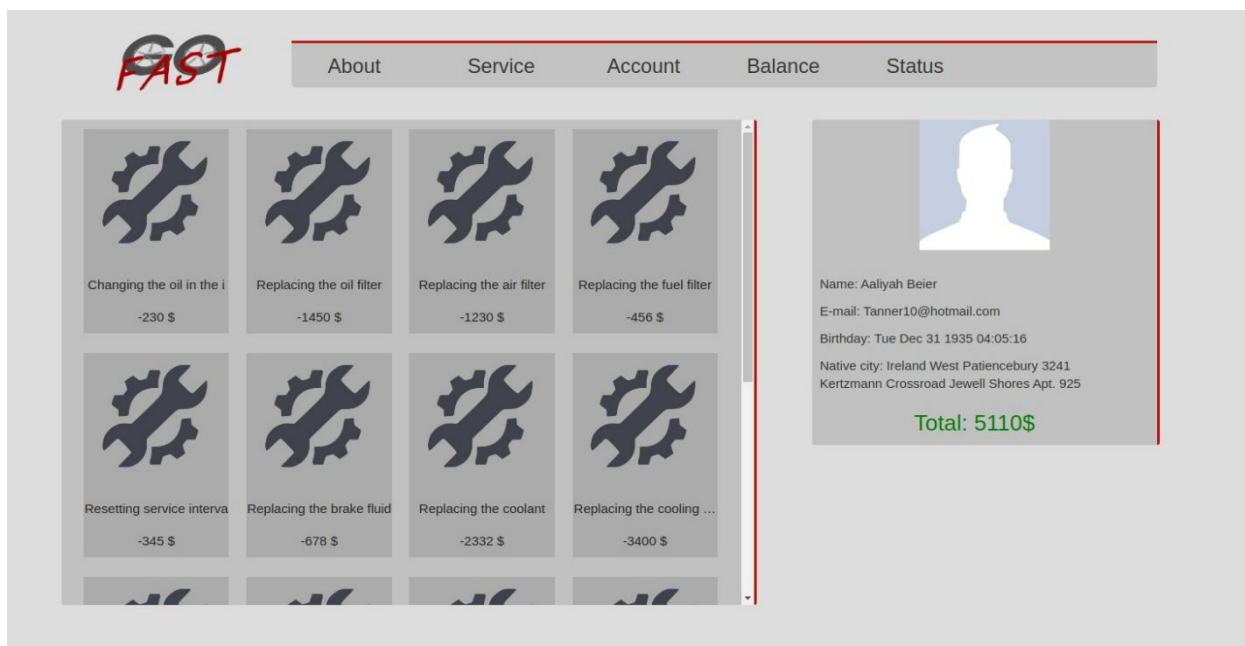
Логинация:



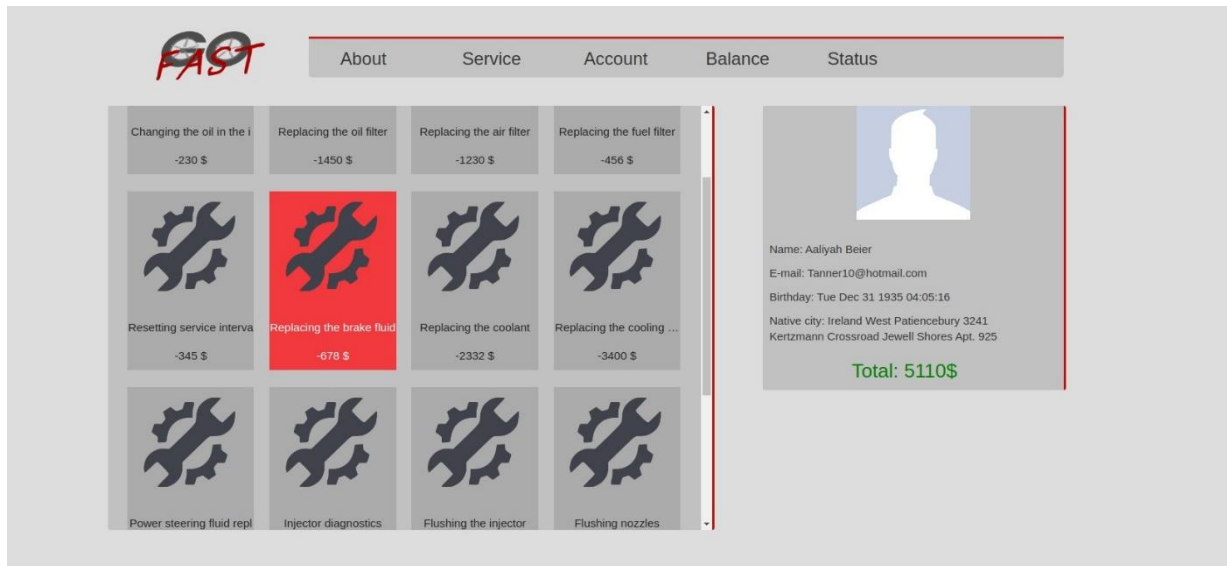
Меню пользователя – после нажатия кнопки «About» (рыба-текст)



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, under the Deutsche Bank logo, there is a form for entering payment details. On the right, there is a user profile section with a silhouette icon and personal information.

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 divided into two sections. On the left, there is a large empty gray box. On the right, the user profile section is updated with the new total balance.

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 Kert 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

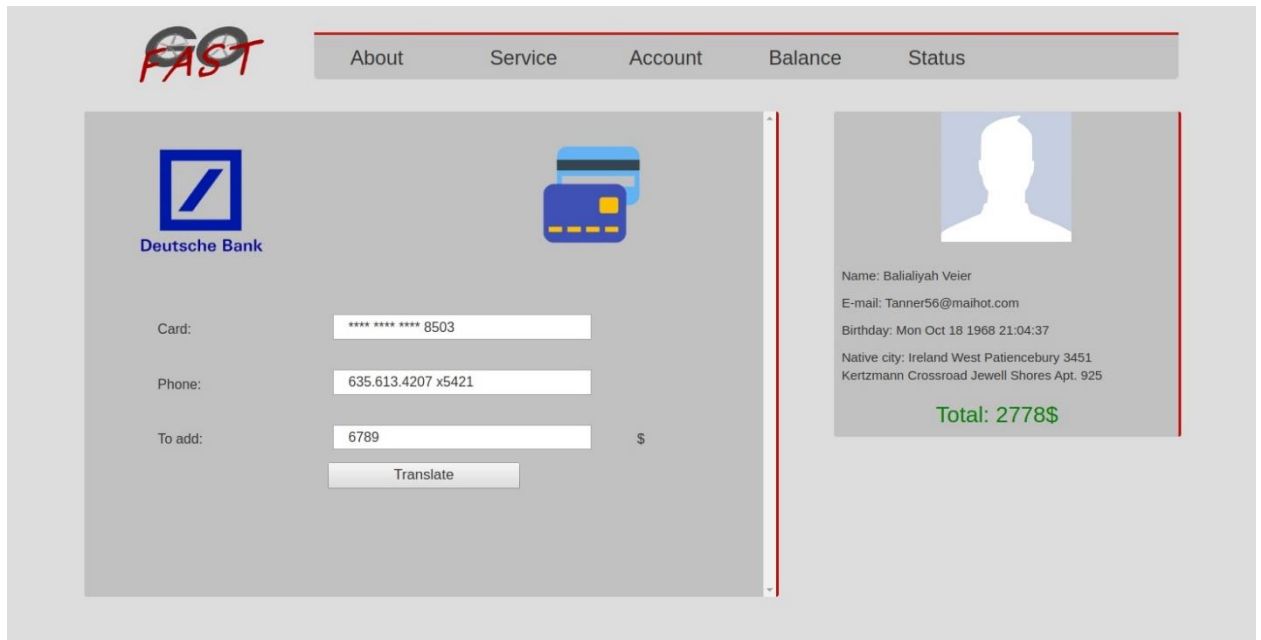
E-mail: Tanner56@maihot.com

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 'GO 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 form for account top-up. The form includes fields for Card (**** * 8503), Phone (635.613.4207 x5421), and To add (6789 \$). A 'Translate' button is located below the 'To add' field. The right section displays a user profile with a placeholder image and the following information: Name: Balialiyah 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. The total balance is shown as 'Total: 2778\$'.

GO FAST

About Service Account Balance Status

Deutsche Bank

Card: **** * 8503

Phone: 635.613.4207 x5421

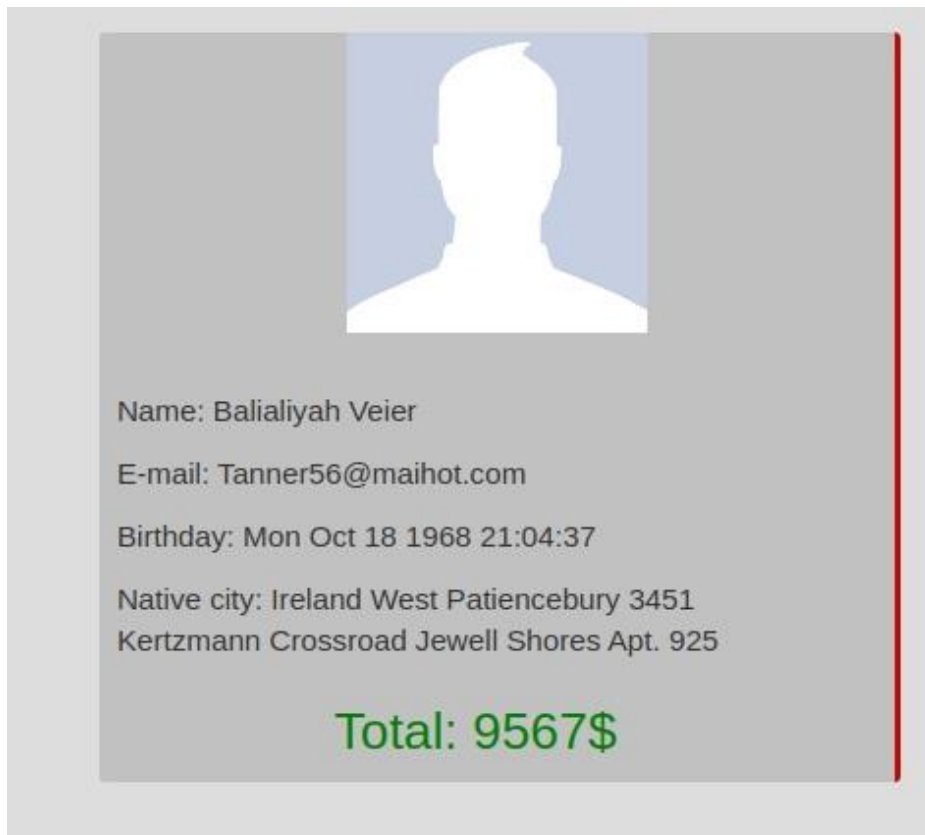
To add: 6789 \$

Translate

Name: Balialiyah 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: 2778\$

Результат пополнения счета:



The screenshot shows the result of the account top-up. It displays the same user profile information as the previous screenshot, but the total balance is now 'Total: 9567\$'.

Name: Balialiyah 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: 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	Cancel
Oil change	Tue Jan 12 2021 02:04:12	In process	Cancel
Replacing the cooling sys	Tue Jan 12 2021 02:04:06	In process	Cancel
Replacing the coolant	Tue Jan 12 2021 01:58:23	In process	Cancel

User Profile:

Name: Balialiyah 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 service that was not cancelled.

Service	Date	Status	Action
Replacing the coolant	Tue Jan 12 2021 01:58:23	In process	Cancel

User Profile:

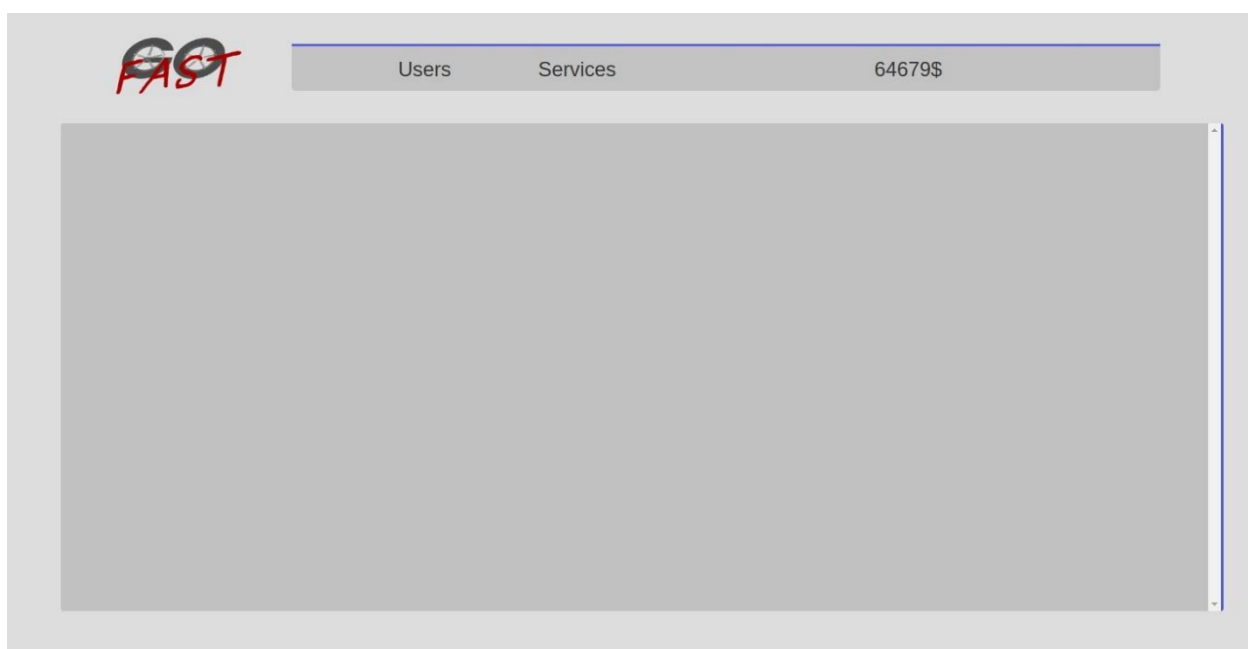
Name: Balialiyah Veier
E-mail: Tanner56@mailhot.com
Birthday: Mon Oct 18 1968 21:04:37
Native city: Ireland West Patiencebury 3451
Kertzmann Crossroad Jewell Shores Apt. 925

Total: 3907\$

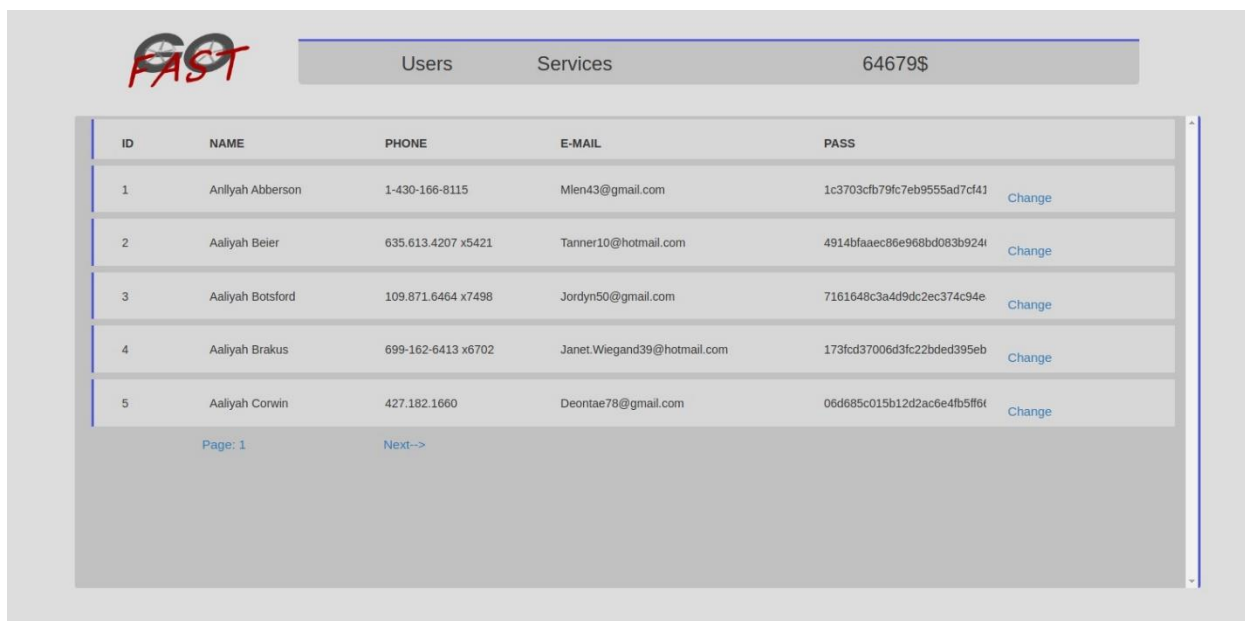
Логинация админа:



Начальный экран:



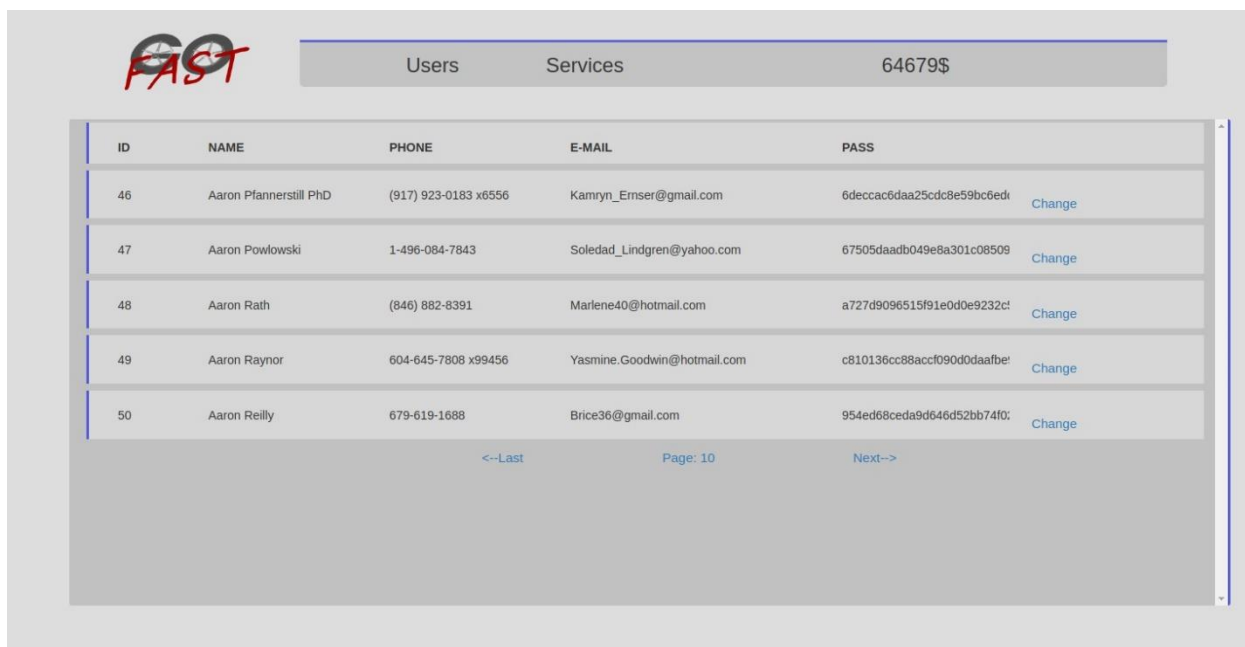
После нажатия кнопки «Users»:



ID	NAME	PHONE	E-MAIL	PASS	
1	Anilyah Abberson	1-430-166-8115	Mlen43@gmail.com	1c3703cfb79fc7eb9555ad7cf41	Change
2	Aaliyah Beier	635.613.4207 x5421	Tanner10@hotmail.com	4914bfaaec86e968bd083b924f	Change
3	Aaliyah Botsford	109.871.6464 x7498	Jordyn50@gmail.com	7161648c3a4d9dc2ec374c94e	Change
4	Aaliyah Brakus	699-162-6413 x6702	Janet.Wiegand39@hotmail.com	173fcd37006d3fc22bded395eb	Change
5	Aaliyah Corwin	427.182.1660	Deontae78@gmail.com	06d685c015b12d2ac6e4fb5ff6f	Change

Page: 1 Next-->

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



ID	NAME	PHONE	E-MAIL	PASS	
46	Aaron Pfannerstill PhD	(917) 923-0183 x6556	Kamryn_Ernser@gmail.com	6deccac6daa25cdc8e59bc6edf	Change
47	Aaron Powlowski	1-496-084-7843	Soledad_Lindgren@yahoo.com	67505daadb049e8a301c08509	Change
48	Aaron Rath	(846) 882-8391	Marlene40@hotmail.com	a727d9096515f91e0d0e9232c	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 = 6, и изменение информации пользователя:

GO FAST

Users Services \$

Name: Daliyah Bietrich OK!

E-mail: Lynn.Lauch@hotmail.com

Phone: 347-798-4916 x2735

Password: *****

Re-password: ***** OK!

Change Info

После нажатия кнопки «Change info»:

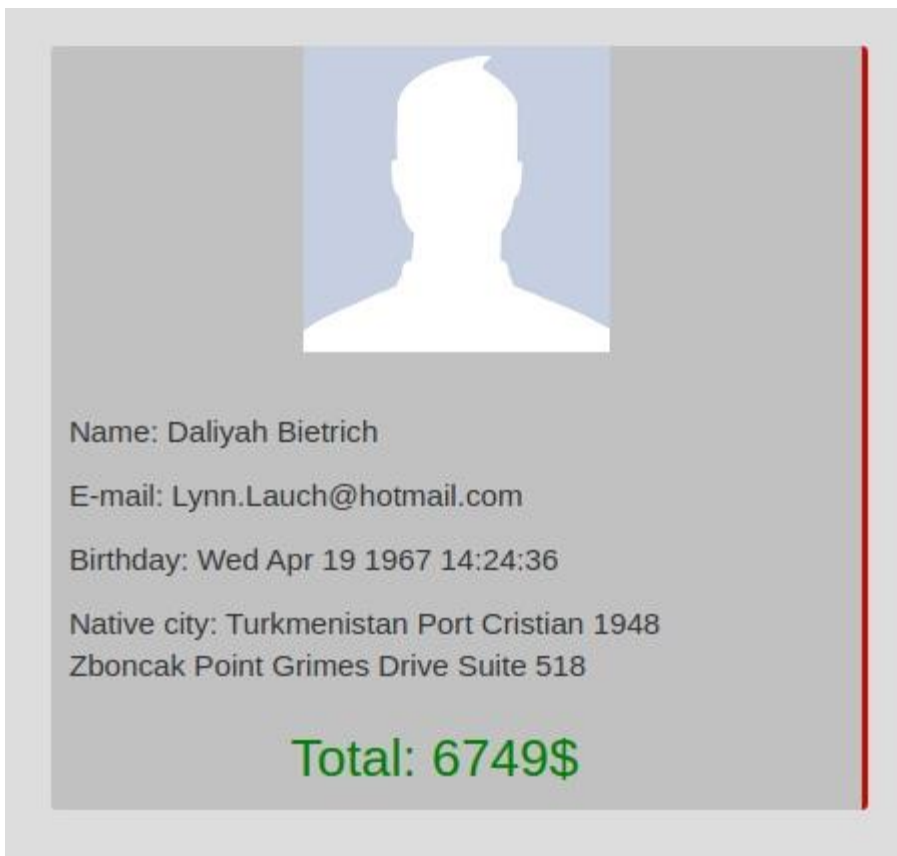
GO FAST

Users Services 64679\$

ID	NAME	PHONE	E-MAIL	PASS
6	Daliyah Bietrich	317-698-4916 x2735	Lynn.Lauch@hotmail.com	099699435a321ab673e673484 Change
7	Aaliyah Doyle	1-812-092-7813 x7775	Rylan.Franecki89@yahoo.com	3de68c65c92016f9fdf3089faf5e Change
8	Aaliyah Gibson	836-117-6756	Edmund30@gmail.com	ecc38d6c73971b0aa46e90d34 Change
9	Aaliyah Gislason	254-716-1755 x5687	Justen81@yahoo.com	f012089c5a6394bb8d84e1bd8f Change
10	Aaliyah Glover	(547) 511-9002	Raven75@hotmail.com	de2075af332a34fdb8312c34db Change

<--Last Page: 2 Next-->

Демонстрация изменения данных из меню пользователя:

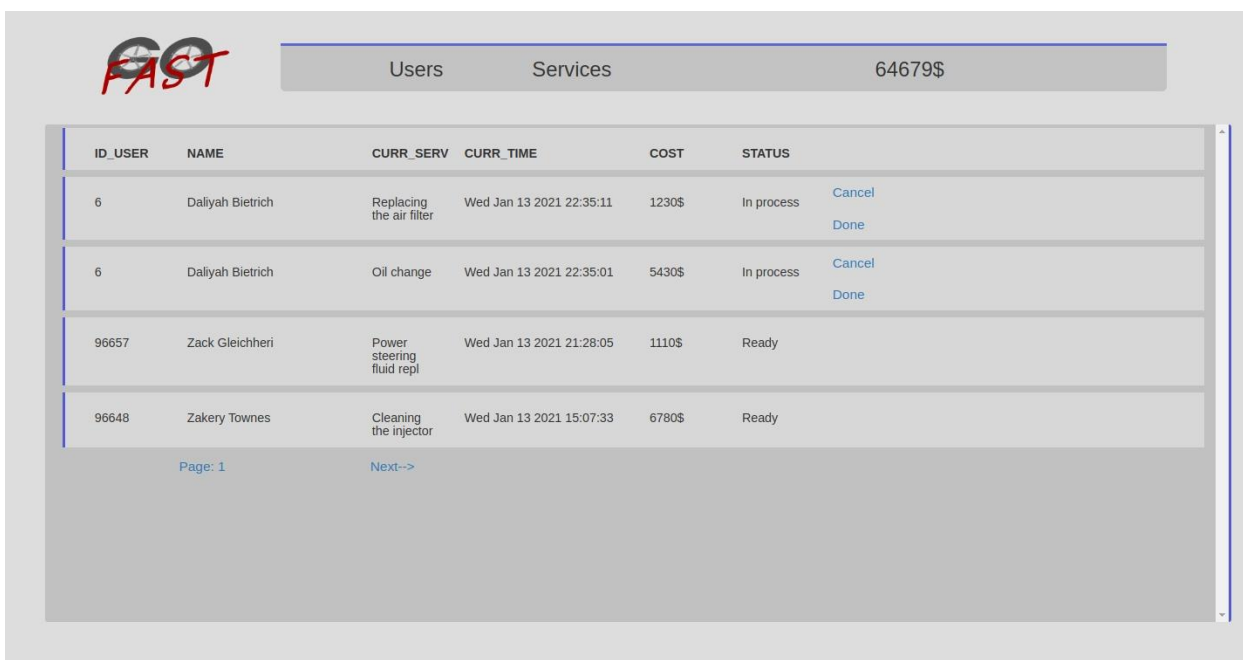


A user profile card with a grey background and a red vertical line on the right. At the top is a blue square placeholder for a profile picture. Below it, the following text is displayed:

Name: 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

At the bottom, the total amount is shown in green text: **Total: 6749\$**

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



The screenshot shows the FAST application interface. At the top left is the FAST logo. To its right is a navigation bar with three items: "Users", "Services", and "64679\$". The "Services" item is selected. Below the navigation bar is a table of current orders. The table has six columns: ID_USER, NAME, CURR_SERV, CURR_TIME, COST, and STATUS. There are four rows of data. Below the table, there is a pagination bar with "Page: 1" and "Next-->" links.


ID_USER	NAME	CURR_SERV	CURR_TIME	COST	STATUS
6	Daliyah Bietrich	Replacing the air filter	Wed Jan 13 2021 22:35:11	1230\$	In process Cancel Done
6	Daliyah Bietrich	Oil change	Wed Jan 13 2021 22:35:01	5430\$	In process Cancel Done
96657	Zack Gleichheri	Power steering fluid repl	Wed Jan 13 2021 21:28:05	1110\$	Ready
96648	Zakery Townes	Cleaning the injector	Wed Jan 13 2021 15:07:33	6780\$	Ready

Page: 1 Next-->

Удаление текущего заказа:

6	Daliyah Bietrich	Replacing the air filter	Wed Jan 13 2021 22:35:11	1230\$	In process	Cancel
						Done


После нажатия «Cancel»:

		Users	Services	64679\$		
ID_USER	NAME	CURR_SERV	CURR_TIME	COST	STATUS	
6	Daliyah Bietrich	Oil change	Wed Jan 13 2021 22:35:01	5430\$	In process	Cancel Done
96657	Zack Gleichheri	Power steering fluid repl	Wed Jan 13 2021 21:28:05	1110\$	Ready	
96648	Zakery Townes	Cleaning the injector	Wed Jan 13 2021 15:07:33	6780\$	Ready	
Page: 1		Next-->				

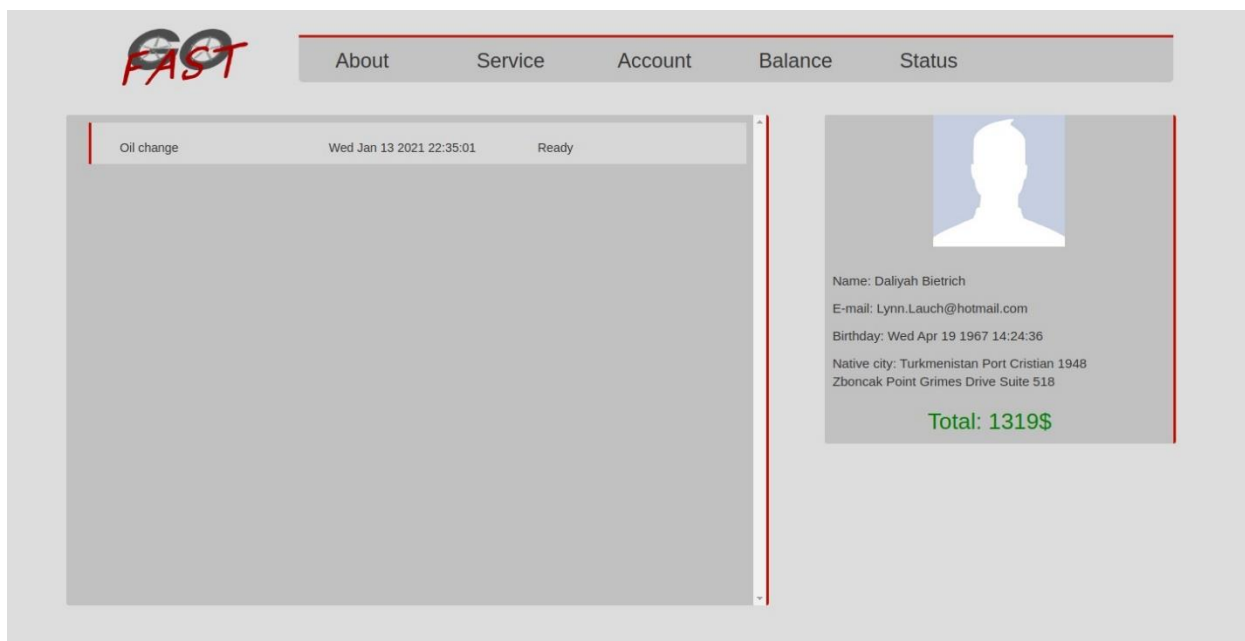
Изменим статус текущего заказа с помощью кнопки «Done»:

6	Daliyah Bietrich	Oil change	Wed Jan 13 2021 22:35:01	5430\$	In process	Cancel Done
---	------------------	------------	--------------------------	--------	------------	--

Результат:

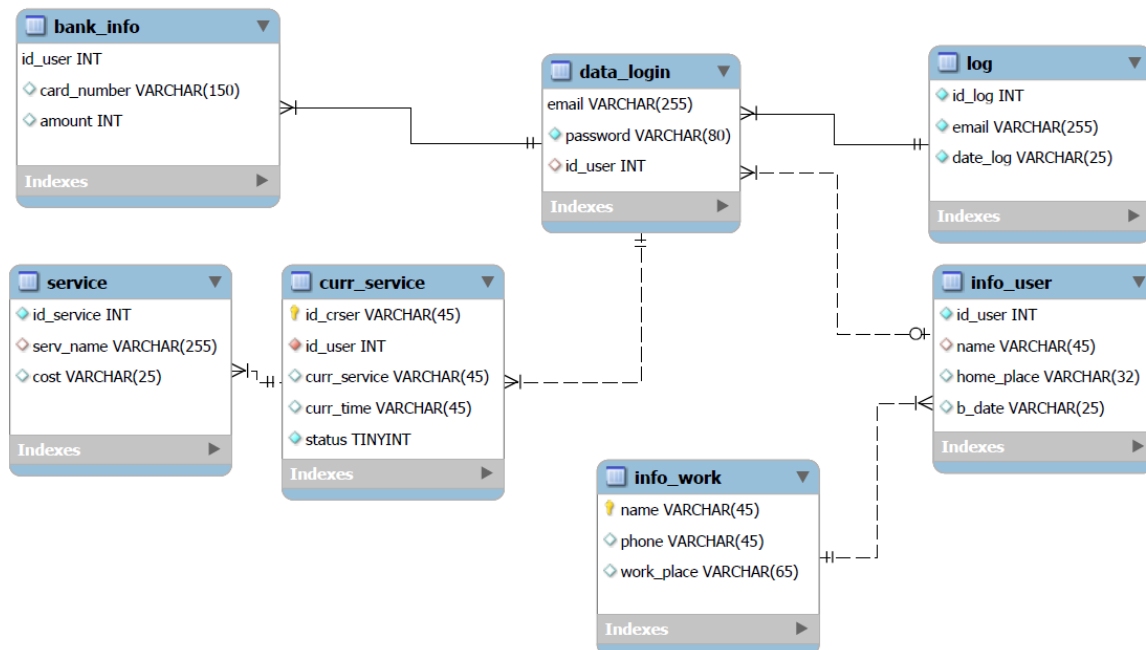
		Users	Services	70109\$		
ID_USER	NAME	CURR_SERV	CURR_TIME	COST	STATUS	
6	Daliyah Bietrich	Oil change	Wed Jan 13 2021 22:35:01	5430\$	Ready	
96657	Zack Gleichheri	Power steering fluid repl	Wed Jan 13 2021 21:28:05	1110\$	Ready	
96648	Zakery Townes	Cleaning the injector	Wed Jan 13 2021 15:07:33	6780\$	Ready	
Page: 1		Next-->				

Демонстрация выполненного заказа для пользователя:



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.